Comparação e validação de vários métodos de mapeamento cortical para o planeamento pré-cirúrgico por Imagem Funcional de Ressonância Magnética

José Gabriel Lira Gomes

15 de Novembro de 2007

Conteúdo

Capítulo 1

Introdução

Na análise funcional de imagens do cérebro podem utilizar-se diferentes métodos na identificação de zonas de activação. Tem havido uma evolução desde o método de correlação [19], para outros métodos [9] [14] até o método baseado no modelo linear generalizado que é mais comum ser utilizado hoje e que levou ao pacote de software SPM [15]. Deve-se principalmente à versatilidade que o método tem em realizar testes com diferentes objectivos.

Têm sido publicados alguns estudos comparativos. Poucos têm sido quantitativos [20] e quando o são, o número de métodos testados é reduzido[22]. Há muitos estudos comparativos do ponto de vista da estatística envolvida (da matemática) mas que têm em geral apenas fins académicos. Um objectivo deste estudo é comparar os resultados obtidos por diferentes métodos. É de particular interesse averiguar o comportamento de cada método na fronteira do local de activação. As diferenças serão avaliadas numericamente para os seguintes métodos clássicos: t de Student, coeficiente de correlação e o modelo linear generalizado. Três novos métodos são também propostos - o método de picos de Fourier, o método de sobreposição e o método de amplitude. O segundo pode ser aplicado para o melhoramento dos métodos de t de Student, coeficiente de correlação e modelo linear generalizado. Ele pode no entanto, também manter-se como um método de análise independente. A influência exercida em cada método pelos parâmetros pertinentes é também medida. É adoptado um conjunto de dados clínicos que está amplamente estudado e documentado. Desta forma elimina-se a possibilidade dos resultados obtidos serem interpretados como sendo específicos do caso em estudo.

Há situações em que a influência do método utilizado na identificação das áreas de activação de imagens funcionais do cérebro é crucial. Tal acontece, por exemplo, quando um tumor desenvolve-se perto de uma zona de activação responsável por uma função importante . Para o cirurgião torna-se indispensável avaliar se existe alguma sobreposição. A escolha de um dos métodos disponíveis poderá ter influência sobre a decisão final. Se o método escolhido for mais conservador, pode verificar-se sobreposição e eliminar-se a possibilidade de cirurgia. Porém, se o método for mais restritivo a decisão final pode ser favorável à cirurgia. Artigos recentes têm suportado a ideia de que a ressonância magnética funcional é de facto muito útil no processo de decisão pré-operatório [12].

O segundo objectivo do estudo é então avaliar a sobreposição entre um volume de activação e o volume do tumor.

Os programas informáticos de análise funcional disponíveis são variados em vários aspectos: na plataforma em que funcionam (macintosh, linux, windows ou outras), na linguagem em que foram desenvolvidos (e.g. c+motif, c+matlab, matlab, etc.) no tratamento inicial dos dados (antes da aplicação do método de análise), no formato das imagens e no(s) método(s) de análise escolhido(s). Este facto dificulta qualquer tentativa de comparação. À partida esta poderá apenas ser qualitativa. Uma comparação quantitativa implicaria a necessidade de ocorrerem três factos: o utilizador tem acesso ao código do programa, sabe programar nas diferentes linguagens e tem licença de utilização de software comercial (e.g. matlab).

Sendo assim foi decidido adoptar uma estratégia unificadora. Ou seja, criar um novo programa

desenvolvido numa linguagem independente da plataforma, que não utilize software comercial e que permita aplicar (e comparar quantitativamente) diferentes métodos de análise funcional. A linguagem escolhida foi o JAVA. O programa desenvolvido no âmbito desta tese chama-se *Cérebro*.

Capítulo 2

Os dados

2.1 A organização dos dados

Em ressonância magnética funcional dispomos de uma sequência de T volumes tridimensionais $V_0, V_1, ..., V_T$ de imagens do cérebro. Cada volume está dividido em unidades de imagem v. Esta unidade chama-se o *voxel*. A cada voxel está associado um valor e quatro coordenadas. As três primeiras identificam a posição do voxel (informação espacial) no volume e a quarta identifica o instante a que se refere o valor (informação temporal) $v = v(x, y, z, t)$.

2.2 O design do estudo

Para além da informação contida no ficheiro de dados funcionais é necessário ter em conta a sequência temporal de estímulos a que o paciente foi sujeito e a forma como os dados foram adquiridos.

Para o processamento dos dados é necessário saber quantos estados de estímulo existem no estudo. Por exemplo, num estudo em blocos de duas tarefas típico existem dois estados: repouso e activação.

Em seguida é também preciso saber quando cada estímulo ocorreu e qual a sua duração. Esta informação fica armazenada numa matriz chamada de *matriz de design*.

Cada coluna da matriz de design refere-se a um estado. Cada linha contém a informação referente a um instante para todos os estados. Assim, a matriz terá um número de linhas igual ao número de volumes do estudo e um número de colunas igual ao número de estados mais um. Por exemplo, se no décimo volume, o paciente está sob a influência do estado de repouso, a linha 10 da coluna referente ao estado de repouso terá o valor de 1 e a linha 10 da coluna referente ao estado de activação terá o valor de 0.

A última coluna da matriz de design é apenas uma coluna de 1's e representa todos os estímultos constantes ao longo do estudo.

A matriz de design é portanto uma matriz de 0's e 1's a partir da qual será retirada toda a informação de design do estudo.

Finalmente, o tipo de estudo é determinante na forma como os dados serão tratados, assim como o tempo de aquisição de cada volume (TR).

Para a filtragem dos dados de estudos do tipo bloco é necessário saber também o número de amostras por ciclo.

Capítulo 3

O *Cérebro* e o pré-processamento dos dados

3.1 O carregamento dos dados

O *Cérebro* lê os ficheiros de dados no formato Mayo Clinic Analyze 4D . Neste formato os dados são distribuidos por dois ficheiros: o *"header"* (*.hdr) e a *"image"* (*.img). No primeiro encontra-se muita informação sobre a aquisição dos dados. Esta informação está contida em 348 bytes e está organizada segundo o formato apresentado no anexo A. Na *"image"* está a informação das imagens sob a forma de um array com quatro dimensões e que representa valores da função v. Só depois de saber as dimensões do array e o formato dos números que o constituem é que é possível realizar a leitura do mesmo. É portanto necessário que a leitura dos dados seja precedida de uma leitura do ficheiro *"header"*.

Após a leitura, o *Cérebro* armazena esta função v sob a forma de um array de dupla precisão quadri-dimensional: imageData[x][y][z][t] .

Para extrair informação temporal fixam-se as variáveis x, y, e z na posição desejada. Fazendo t variar de 0 a T-1 , o array uni-dimensional obtido é a *sequência temporal* do voxel na posição (x, y, z). T é o número de volumes do estudo.

Os dados imageData contêm dois tipos de informação: estrutural e funcional.

Os estruturais são constantes ao longo do tempo e permitem-nos identificar as estruturas anatómicas (e.g. matéria cinzenta).

Os funcionais são variáveis no tempo e mostram-nos a resposta fisiológica do cérebro aos estímulos externos do estudo.

No programa esta separação de dados é feita criando dois arrays: usedImagingData e usedAnalysisData. Ambos tomam inicialmente os valores de imageData. Assim, podemos modificar os dados funcionais sem que haja alteração dos dados estruturais e vice-versa.

3.2 A máscara

As imagens originais contêm uma grande percentagem de dados que representam o fundo. Este subgrupo de dados é inútil para um estudo funcional. Como tal temos todo o interesse em identificá-los. Uma vez disponível, tal informação permite-nos poupar tempo de execução do programa. A análise pode ser feita apenas para os dados que não são de fundo.

Além disso, o nosso objecto de estudo é o cérebro e não o fundo. Não podemos ter em conta os dados de fundo na análise estatística.

A máscara é armazenada sob a forma de um exemplar da classe Mask (dataMask) que contém a sua informação sob a forma de um array booleano tridimensional. Esse array depende de x, y e z e tem o valor de "verdadeiro" nos voxels de dados e de "falso" nos voxels de fundo.

O programa permite a classificação dos voxels por dois métodos diferentes: o método de Mahalanobis e o de clusters.

3.2.1 O método de Mahalanobis

Este pode ser separado em três etapas principais: amostragem, avaliação e decisão. A amostragem é automática (não assistida) e consiste em identificar e recolher os valores relativos a voxels que sabemos ser de fundo. Assumindo que um cérebro tem simetria elipsoidal, podemos aceitar a premissa de que todos os voxels que se encontram nas arestas do paralelipípedo que é o volume, são de fundo. São estes os voxels de amostragem neste método. Em cada volume é feita uma amostragem independente. Obtemos assim uma matriz coluna μ de$N$ médias e uma matriz covariância C, $N \times N$, em que N é o número de volumes.

Em seguida cria-se uma matriz coluna com os valores do voxel $(0, 0, 0)$ e mede-se a distância de Mahalanobis deste voxel até à matriz média. A distância é dada por:

$$d(\mu, \bar{\mu}) = \sqrt{(\mu - \bar{\mu})^T C^{-1} (\mu - \bar{\mu})} \tag{3.1}$$

Este processo é repetido para todas as posições possíveis de voxel no volume.

Finalmente estabelecemos um critério. Por exemplo, se a distância de Mahalanobis for inferior a 10 o voxel é de fundo. Caso contrário, não é. Este método é executado automaticamente logo após a leitura dos dados do ficheiro Analyze.

3.2.2 O método de clusters

Este método executa uma segmentação assistida das imagens. Ver o capítulo de segmentação (cap.5).

3.3 O design

A quantificação da forma como foi executada a aquisição dos dados é executada pelo *Cérebro* em três etapas:

- definição dos estados - todos os estados da aquisição têm que ser identificados de duas formas: uma *Sigla* de três letras e um *Nome* (frase ou palavra mais descritiva até um máximo de 25 caracteres).

- sequência de estados - Uma janela com uma barra (inicialmente toda cinzenta) representa a sequência dos estados do estudo. É possível adicionar estados definindo a sigla o início e a duração. Se existe uma sequência de estados repetitiva basta desenhar essa sequência e depois mandar repetir. Se houver engano é também possível apagar. Uma vez construída, a sequência de estados pode ser guardada num ficheiro com extensão ".tml". Sempre que necessário o *Cérebro* poderá carregá-lo.

- parâmetros - Nesta janela define-se o tempo de aquisição (TR). O programa reconhece automaticamente se a sequência de estados é do tipo bloco e, neste caso pede ao utilizador que insira também o número de aquisições por ciclo.

Toda esta informação fica concentrada num exemplar da classe DesignMatrix. Este objecto contém a matriz de design e tem uma série de métodos pertinentes associados (ver apêndice E). Entre eles consta o getDesignVector(). Este método fornece uma sequência temporal de estímulos referente a dois estados que pretendamos comparar. Ela é designada de *sequência de design*. Podemos também olhar para esta sequência como sendo um vector em que cada componente é dada pelo seu valor em cada instante. Neste caso chamá-la-emos de *vector de design*.

Este array unidimensional pode tomar três valores: 0, +1 e −1 e tem uma dimensão igual ao número de volumes do estudo. É +1 quando num instante temos o primeiro estado, −1 quando temos o segundo estado e 0 quando não estamos em qualquer dos dois estados.

O vector de design será utilizado por vários métodos de análise funcional como uma primeira aproximação do sinal fisiológico de activação que esperamos medir. É apenas uma aproximação por duas razões: o sinal fisiológico tem sempre um atraso em relação ao vector de design e o sinal resulta de um aumento de oxigenação por via do fluxo sanguíneo. Este sinal é conhecido por BOLD (Blood Oxigenation Level Dependent).

É portanto uma maneira indirecta de medir um aumento de actividade cerebral. O pressuposto é que se uma zona do cérebro necessita de mais oxigenação e nutrientes então ela está em actividade.

Percebe-se então porque é que há o atraso. O aumento de fluxo sanguíneo só aumenta quando há necessidade (depois dos neurónios consumirem algumas reservas) e depois dos vasos sanguíneos terem tido tempo de dilatar.

Mais, a resposta do vaso sanguíneo apresenta um pico inicial de compensação que depois reduz gradualmente para um estado de equilíbrio. Esta variação do fluxo sanguíneo é prevista pelo modelo do balão [4].

A função que descreve esta variação temporal é conhecida por HRF (Hemodynamic Response Function) e neste modelo tem a forma da densidade de probabilidade de uma distribuição gama:

$$\gamma_\ell(t) = \frac{1}{\Gamma(\ell)} t^{\ell-1} e^{-t} \tag{3.2}$$

Outros modelos [13][16] propõem que após o pico de fluxo sanguíneo a função HRF chega a ir abaixo do estado de equilíbrio. Este modelo é conhecido pelo modelo SPM canónico por ser o modelo adoptado pelo pacote de software SPM e é descrito pela equação:

$$h_C(t) = \gamma_6(t) - \frac{1}{6}\gamma_{16}(t) \tag{3.3}$$

Estudos mais recentes [10] de estimativa da resposta hemodinâmica sugerem que o sinal HRF varia de região para região, de tarefa para tarefa e de paciente para paciente.

Se o sinal de design tem a pretensão de reproduzir o sinal fisiológico, ele tem que ser modificado. O programa *Cérebro* oferece ao utilizador a possibilidade de compensar estas diferenças. Ele pode inserir o atraso que quiser no sinal. Pode também realizar a convolução do sinal de design com a função HRF (eq.3.3). Uma posterior versão do *Cérebro* irá incluir a funcionalidade de estimativa da função HRF a partir dos dados com o método apresentado em [11].

Ao sinal escolhido (sequência de design, sequência de design convolvida com o HRF ou outro) chamamos *sequência de modelo* ou *vector de modelo* (função $u = u(t)$).

3.4 Os outliers ou pontos espúrios

Por vezes, numa sequência temporal aparece um ponto (ou vários!) estranho. Por exemplo, a sequência da figura tem sempre valores nulos excepto no instante 112. Este valor é chamado um "*outlier*" ou "*ponto espúrio*". Estes pontos podem ter uma influência pesada sobre as medições estatísticas efectuadas sobre uma sequência temporal e alterar drasticamente a inferência estatística resultante.

O *Cérebro* permite ao utilizador retirar estes pontos. Isto é conseguido em varios passos:

- ordena por ordem crescente a sequência temporal.
- calcula a mediana e o desvio absoluto da mediana (DAM) dos primeiros 95% dos dados ordenados.
- mede a quantos DAMs de distância está cada um dos restantes 5% da mediana.
- se a distância é muito grande então modifica este valor para um valor interpolando entre os valores anterior e posterior da sequência temporal inicial (não ordenada).

Os dados sem outliers são automaticamente exportados no formato Analyze para um ficheiro com o nome do ficheiro original concatenado com a letra c. Se os dados foram previamente filtrados são exportados com o tipo complexo, caso contrário são exportados com o tipo complexo. Neste caso, os valores reais são os estruturais e os imaginários são os funcionais.

3.5 A filtragem dos dados

Os campos magnéticos ditos estáticos aplicados durante uma sequência de ressonância magnética nem sempre são homogéneos. Ou seja, nem sempre são constantes ao longo do espaço e do tempo. Este facto causa o aparecimento de ruídos de baixa frequência nas sequências temporais. O *Cérebro* oferece a possibilidade ao utilizador de retirar este ruído aplicando um filtro passa-alto. O filtro é baseado na ideia de realizar uma convolução da sequência temporal com uma função $g(t)$.

$$v_f(t) = v(t) \otimes g(t) \tag{3.4}$$

em que $g(t)$ é tal que a sua transformada de Fourier é a função de transferência do filtro passa-alto ideal $\tilde{g}(f)$:

$$\tilde{g}(f) = \mathcal{F}(g(t)) \tag{3.5}$$

A função $g(t)$ será então uma função do tipo *sinc*. Porém, neste caso não temos uma função *sinc* contínua de valores mas sim uma sequência de valores de tamanho finito. O cálculo de $g(t)$ pela Transformada Rápida de Fourier (TRF) implica o aparecimento do fenómeno de "aliasing". Ou seja, a sequência $\tilde{g}(f)$ já não é exactamente a de um filtro ideal.

Este problema é torneado fugindo da função *sinc*. Primeiro é feito um cálculo separado da TRF da sequência temporal:

$$\tilde{v}(f) = \mathcal{F}(v(t)) \tag{3.6}$$

Em segundo lugar constroi-se a sequência de valores complexos da função de transferência do filtro passa-alto ideal $\tilde{g}(f)$. Em terceiro lugar realiza-se o produto das duas sequências no espaço das frequências. Finalmente, calcula-se a Transformada Rápida de Fourier Inversa (TRFI) do resultado do produto:

$$v_f(t) = \mathcal{F}^{-1}(\tilde{g}(f) \cdot \tilde{v}(f)) \tag{3.7}$$

Nas figuras 3.1 e 3.2 podemos observar a filtragem de uma sequência temporal com ruído de baixa frequência.

foram retiradas as componentes de baixa frequência.

Os dados filtrados são automaticamente exportados no formato Analyze para um ficheiro com o nome do ficheiro original concatenado com a letra f. Os seus valores são atribuídos ao array usedAnalysisData. É de notar que a partir deste momento estes dados já não contêm qualquer informação estrutural.

Os dados exportados são do tipo complexo. Os valores reais são os estruturais e os imaginários são os funcionais.

3.6 A visualização dos dados

O *Cérebro* possibilita a vizualização dos dados sob quatro formas diferentes.

Imagens - uma janela (figura 3.3) mostra imagens dos dados segundo a orientação axial. Tem as seguintes funcionalidades: selecção da fatia apresentada; selecção do instante a que se refere a fatia; exportação das imagens no formato jpeg; exportação das sequências temporais para qualquer voxel no formato txt; selecção do voxel clicando na imagem; ampliação com CTRL-clic; redução com ALT-clic.

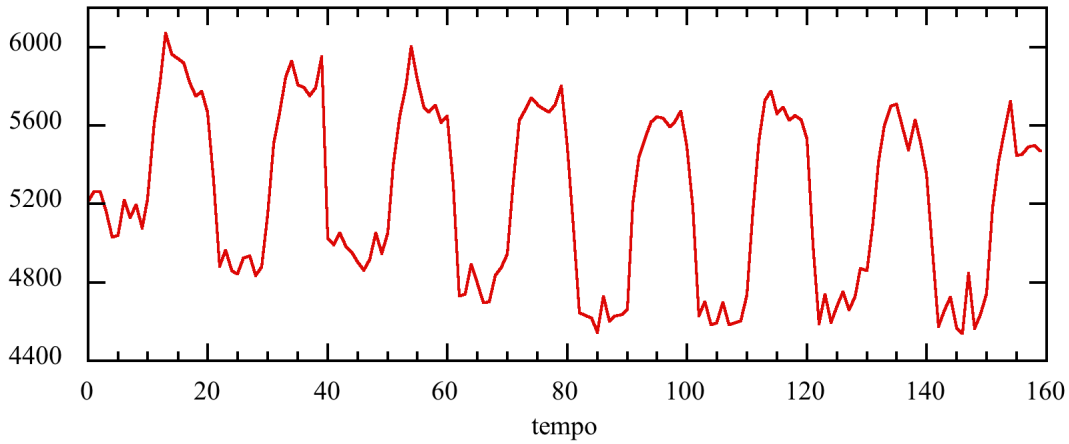


Figura 3.1: Sequência temporal não filtrada.

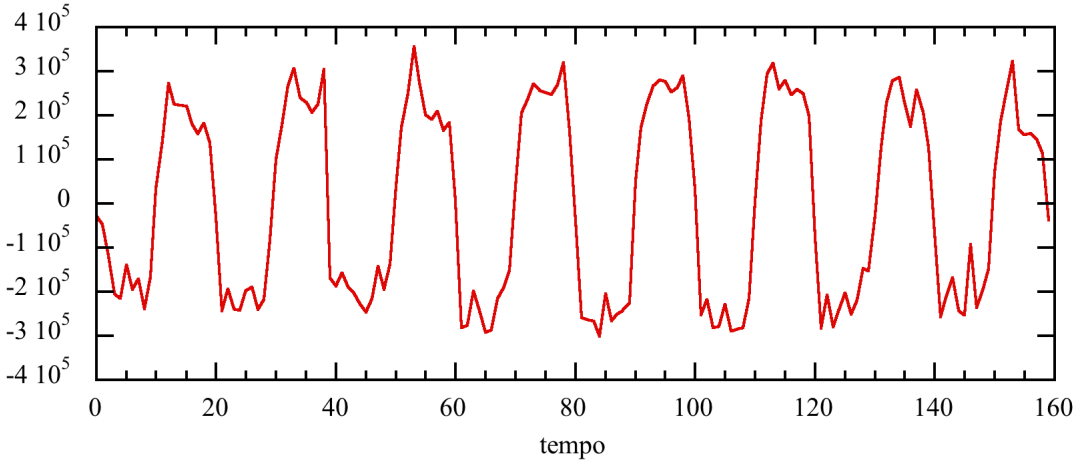


Figura 3.2: Sequência temporal filtrada.

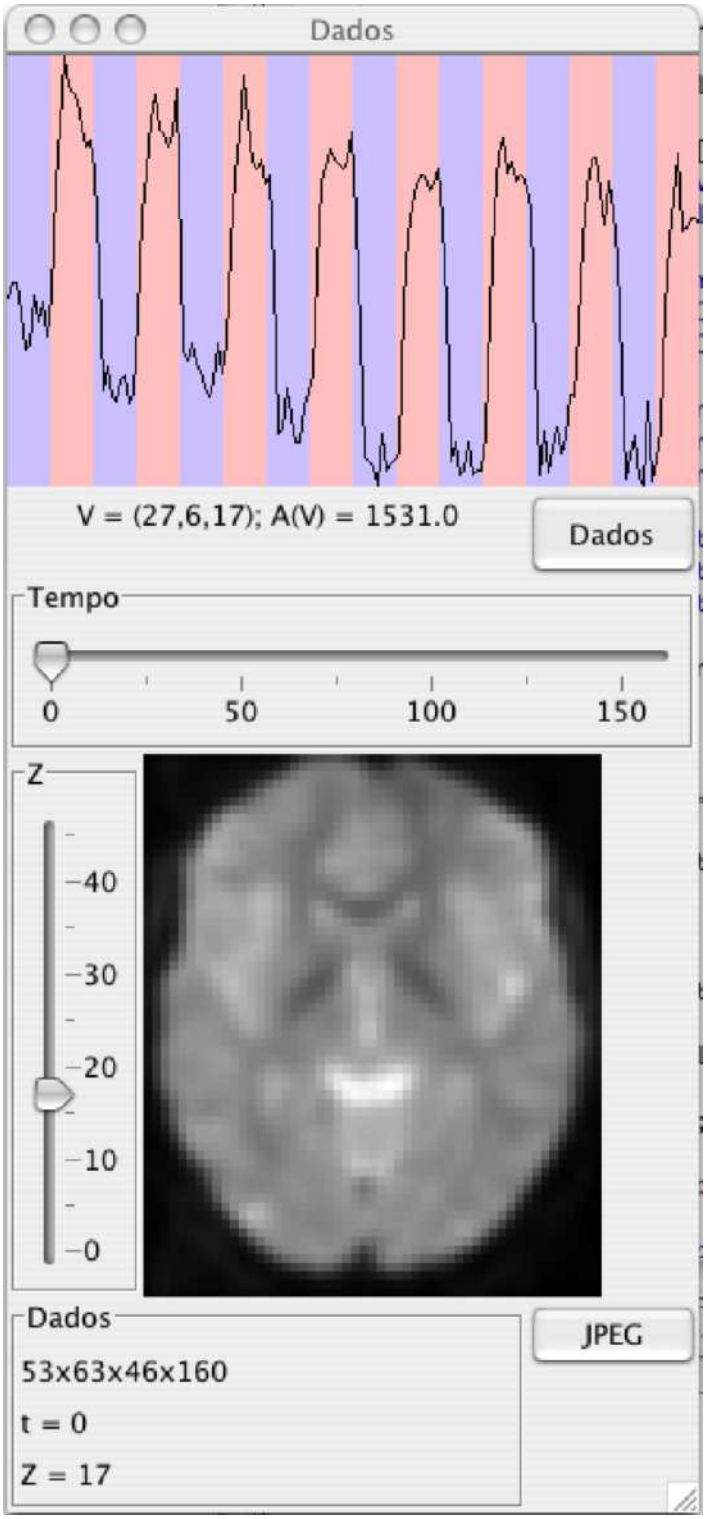


Figura 3.3: Janela de vizualização de imagens e sequências temporais dos dados.

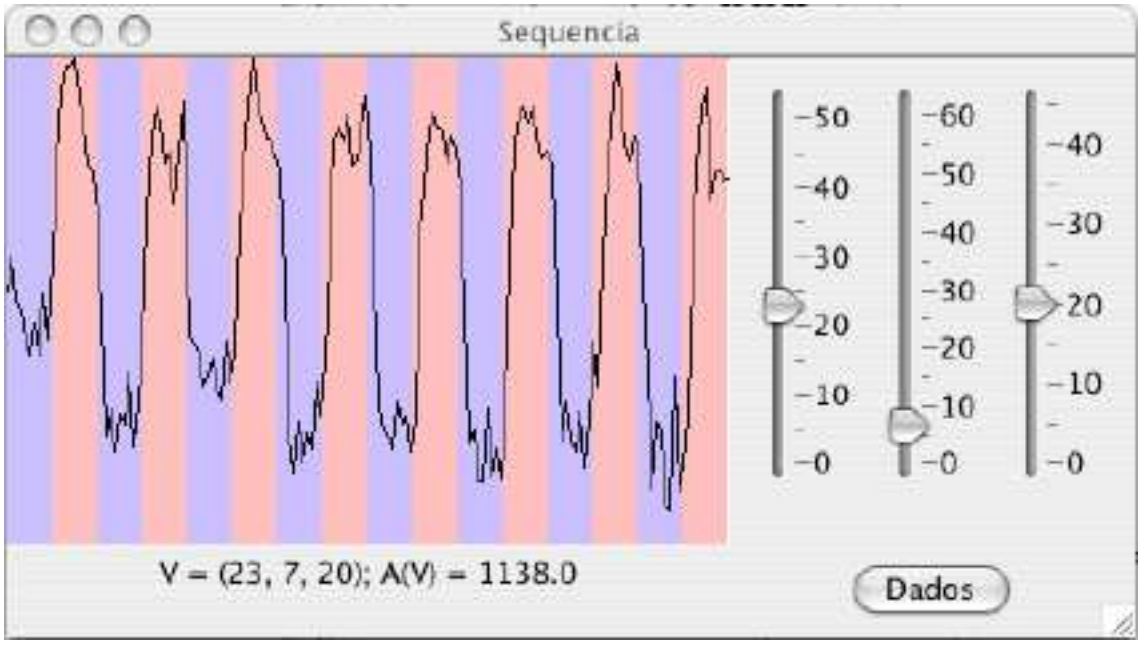


Figura 3.4: Janela de sequência temporal.

Sequência temporal - uma janela (figura 3.4) mostra um gráfico da sequência temporal de um voxel. Tem as seguintes funcionalidades: selecção do voxel pretendido; exportação da sequência temporal no formato txt.

Espectro - uma janela (figura 3.5) mostra o espectro de frequências de cada sequência temporal. Tem as seguintes funcionalidades: escolha do voxel pretendido; exportação do espectro no formato txt. O espectro pode ser apresentado com ou sem janelas (Hann, Hamming).

Resultados - uma janela (figura 3.6) mostra os resultados sob a aforma de imagens e perfis de probabilidade segundo a três direcções ortogonais. Tem as seguintes funcionalidades: selecção da fatia apresentada; selecção do limiar de probabilidade na fatia; exportação das imagens no formato jpeg; exportação dos perfis de probabilidade para qualquer voxel no formato txt; selecção do voxel clicando na imagem; ampliação com CTRL-clic; redução com ALT-clic.

Perfis - uma janela (figura 3.7) mostra os perfis de probabilidade separadamente para permitir a comparação entre perfis de voxels diferentes sem ter que fazer novo cálculo. Tem as seguintes funcionalidades: selecção do voxel por posição; exportação dos três perfis de probabilidade para qualquer voxel no formato txt.

Pode-se também visualizar a máscara dos dados actualmente a ser utilizada para os cálculos, assim como a matriz de design do estudo.

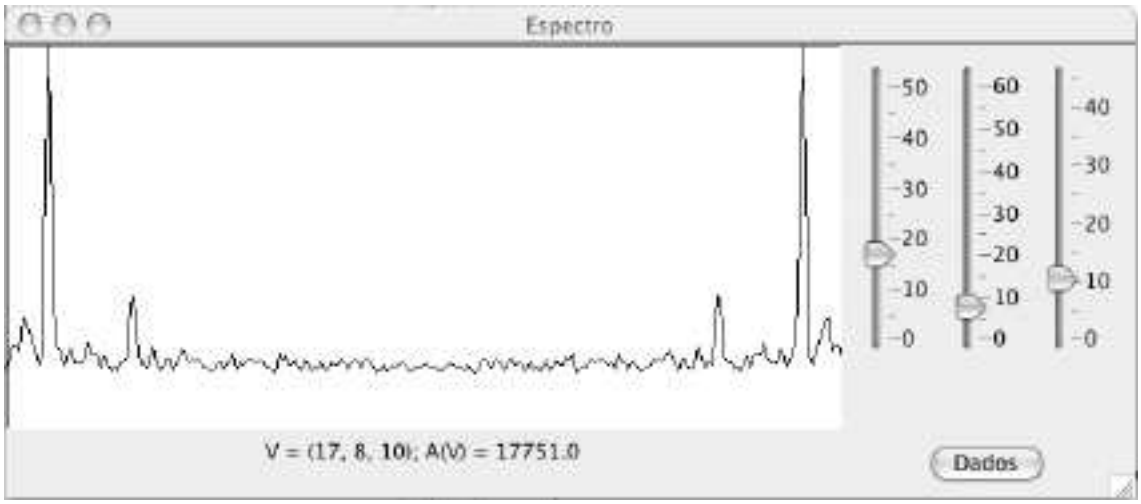


Figura 3.5: Janela de espectro de uma sequência temporal.

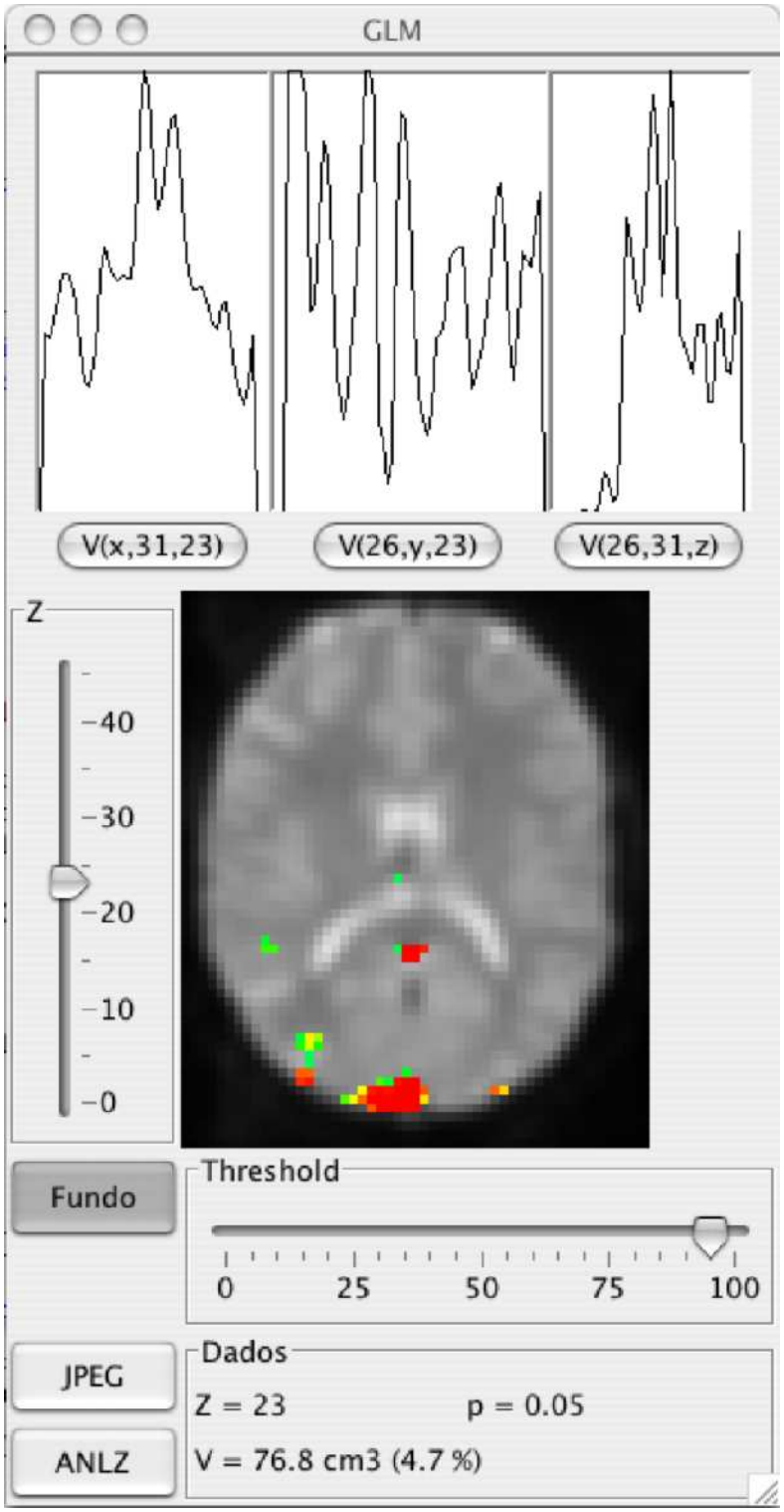


Figura 3.6: Janela de resultados.

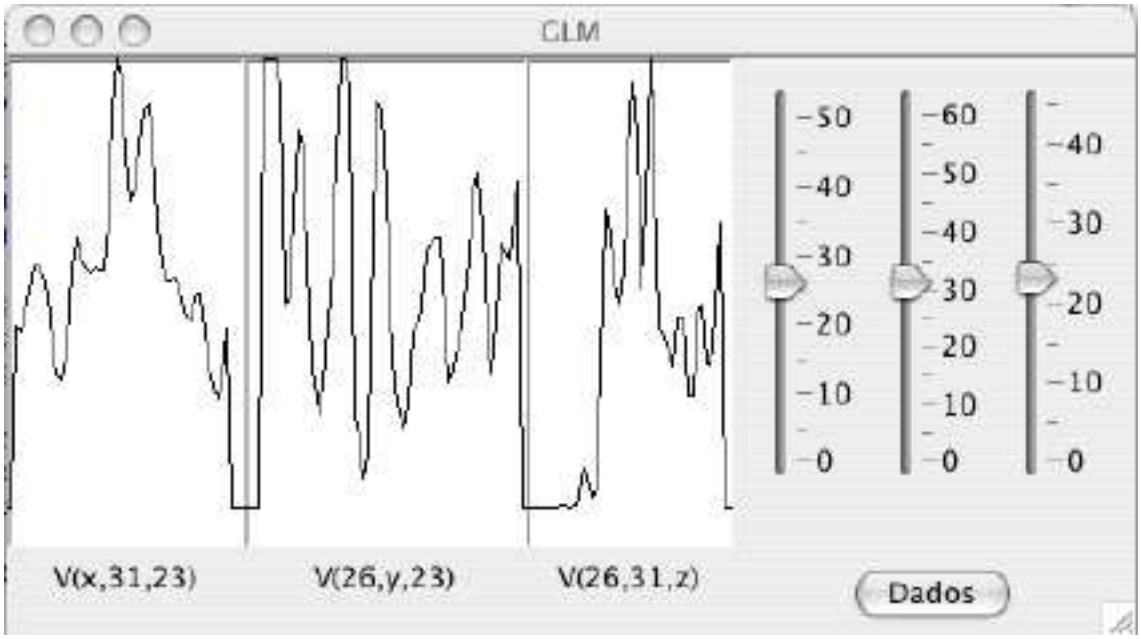


Figura 3.7: Janela de perfis

Capítulo 4

O *Cérebro* e a análise funcional

4.1 Introdução

O *Cérebro* permite a avaliação de activação funcional por seis métodos distintos: t de Student, coeficiente de correlação, linear generalizado, picos de Fourier, amplitude e sobreposição.

Os resultados da aplicação de um método são apresentados sob a forma de um array tridimensional (dependente de x, y e z) que será designado de *volume de probabilidade*. Esta forma única de apresentação dos resultados permite uma comparação quantitativa dos métodos.

4.2 O método t de Student

4.2.1 Teoria

O método t de Student é um método estatístico desenvolvido para avaliar se as médias de duas populações são diferentes. Ele baseia-se no seguinte pressuposto: Se uma variável Z é obtida pela razão entre duas variaveis $X \sim N(0,1)$ e $Y \sim \chi^2$ então ela tem uma distribuição de Student.

Se a este pressuposto aliarmos um critério de decisão temos o método t de Student.

A variável com distribuição normal centrada X será dada pela diferenca entre as médias $\overline{v_A}$ e $\overline{v_R}$. A variável com distribuição do tipo χ^2 será dada pela quantidade:

$$\sqrt{\frac{\sigma_A^2}{N_A} + \frac{\sigma_R^2}{N_R}} \tag{4.1}$$

com N_A é o número de estados de activação e N_R é o número de estados de repouso.

A variável obtida a partir da razão entre estas duas:

$$\vartheta = \frac{\overline{v_A} - \overline{v_R}}{\sqrt{\frac{\sigma_A^2}{N_A} + \frac{\sigma_R^2}{N_R}}} \tag{4.2}$$

tem uma distribuição t de Student.

$$\phi(\vartheta, n) = \frac{1}{\sqrt{n} B\left(\frac{1}{2}, \frac{n}{2}\right)} \left(1 + \frac{\vartheta^2}{n}\right)^{-\frac{n+1}{2}} \tag{4.3}$$

em que n é o número de graus de liberdade e B é a função beta (ver apêndice B).

No gráfico da figura 4.1apresenta-se a variação da função distribuição com t para vários graus de liberdade $(1, 2, 5, 100)$:

Observa-se que a função:

- tem caudas mais elevadas que a distribuição normal.
- coincide com a distribuição normal quando $n \to \infty$.

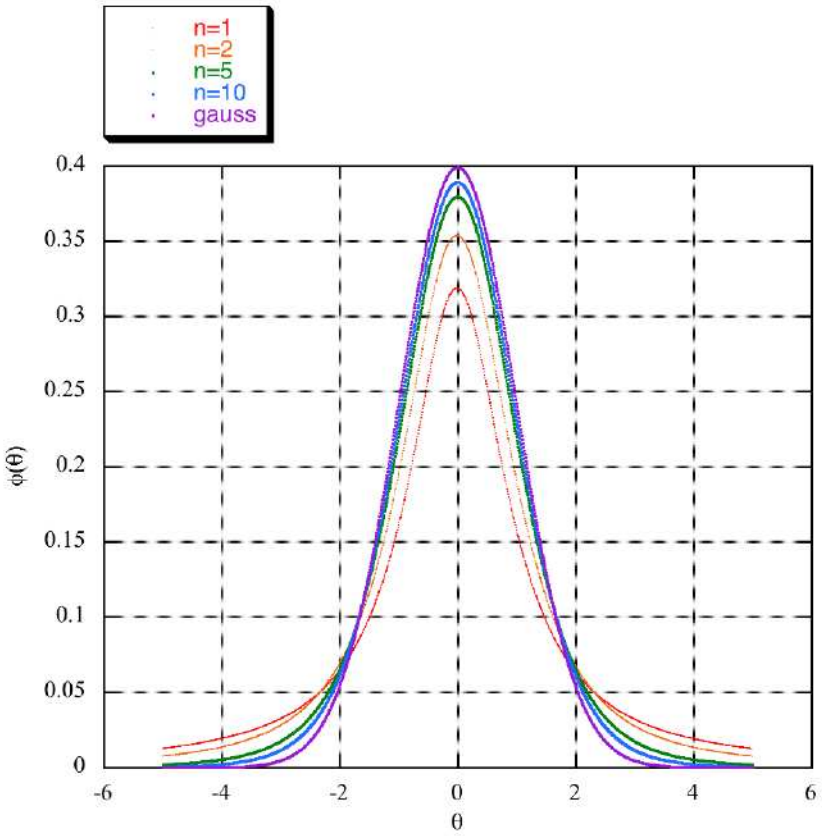


Figura 4.1: Densidades de probabilidade segundo t de Student.

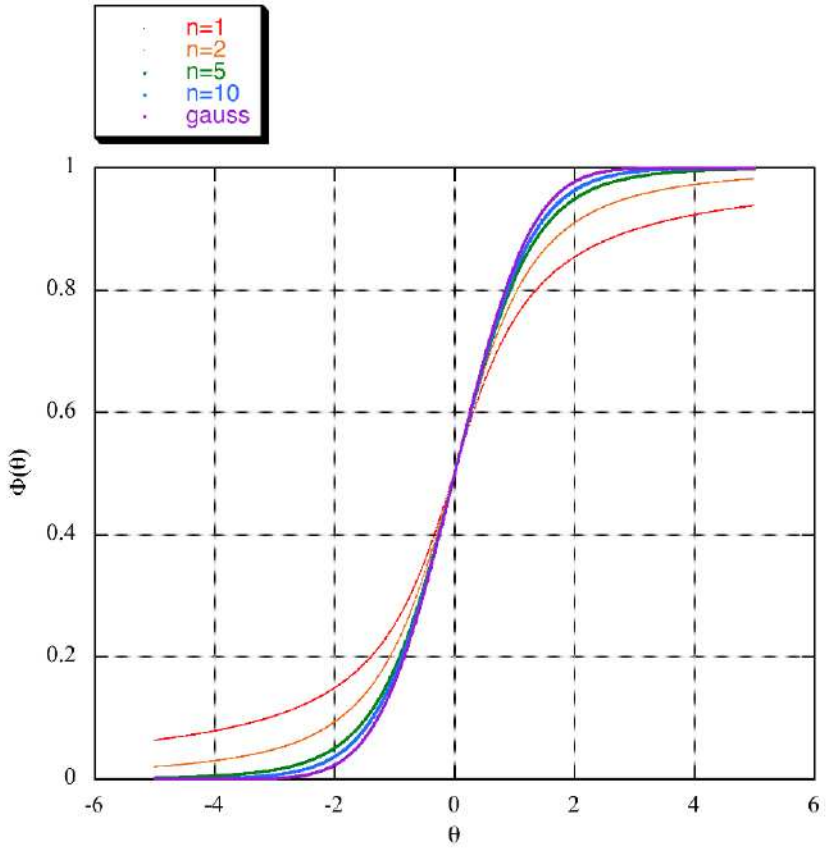


Figura 4.2: Densidades cumulativas de probabilidade segundo t de Student.

A distribuição cumulativa t de Student (ver figura 4.2) é dada por:

$$\Phi(\vartheta, n) = \int_{-\infty}^{\vartheta} \phi(x, n)\, dx = \frac{1}{2} + sgn(\vartheta)\, \frac{1}{2} I_{\frac{\vartheta^2}{n+\vartheta^2}}\left(\frac{1}{2}, \frac{n}{2}\right) \tag{4.4}$$

em que I é a função beta incompleta (ver apêndice B).

A demonstração da integração é apresentada no apêndice C.

4.2.2 Aplicação

Admitindo que o estímulo do cérebro é periódico e do tipo da figura4.3

podemos identificar dois estados: activação e repouso. Para cada sequência temporal tomamos então todos os valores de v obtidos durante o estímulo como pertencentes ao primeiro estado e todos os obtidos durante o repouso como pertencentes ao segundo estado. Calculamos a média $\overline{v_A}$ e o desvio padrão σ_A para os dados do primeiro estado e repetimos para os dados do segundo estado $(\overline{v_R}, \sigma_R)$.

A diferença entre as médias dos valores nos dois estados terá uma distribuição normal?

Pode-se argumentar que não porque o sinal não é uma função de heavyside. É antes se quisermos, uma convolução do sinal HRF com uma função de heavyside. Por isso a distribuição para os dados originais terá dois picos (ver [21]). Ou que se o atraso for apreciável, os dados escolhidos para cada estado ainda incluem os do estado anterior.

Uma análise a este aspecto demonstrou que de facto os dados não estão normalmente distribuídos quando há atraso. No entanto a validade do primeiro argumento é questionável principalmente

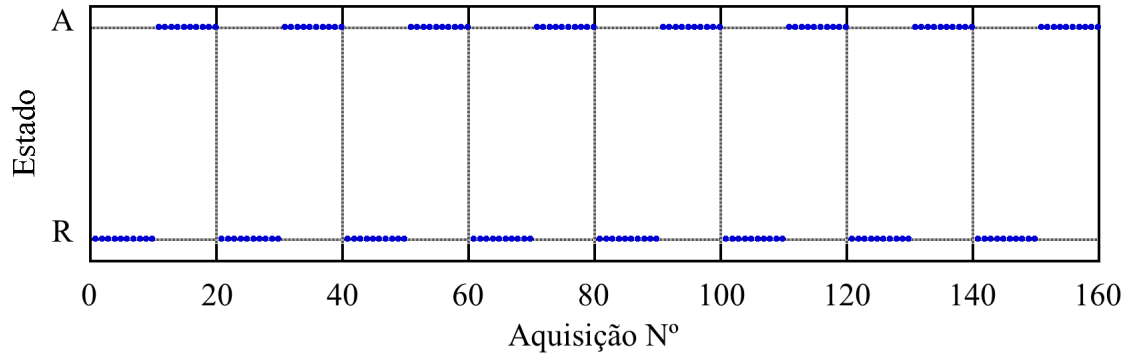


Figura 4.3: Sequência de estímulos.

se tivermos em conta que os níveis de ruído são muitas vezes superiores aos da modelação introduzida pela função HRF.

Sendo assim, o *Cérebro* permite ao utilizador compensar para atrasos mas não dá presentemente hipótese de retirar (e.g. por deconvolução de HRF) a modelação HRF.

O programa calcula a probabilidade da hipótese nula ser válida (p) para cada voxel. O resultado é apresentado sob a forma de um volume de probabilidade com o valor de $1-p$ para cada voxel e gravável num ficheiro Analyze a partir da janela de resultado.

4.3 O método de correlação

4.3.1 Teoria

O método de correlação permite-nos quantificar se existe uma relação linear entre duas variáveis. A medida utilizada para tal é o *coeficiente de correlação linear* (r). Consideremos um conjunto de n pares de dados (u_i, v_i). O coeficiente de correlação linear é dado por*:*

$$r = \frac{\sum_{i=0}^{n-1} (u_i - \bar{u})(v_i - \bar{v})}{\sqrt{\sum_{i=0}^{n-1} (u_i - \bar{u})^2} \sqrt{\sum_{i=0}^{n-1} (v_i - \bar{v})^2}} \tag{4.5}$$

Assumindo que a relação entre v e u é do tipo $v = au + b$, podemos estimar pelo método dos mínimos quadrados quais são os valores de a e b que melhor adaptam o modelo aos dados (minoram a quantidade χ^2).

Em seguida façamos uma inversão do modelo. Ou seja, vamos assumir agora que a relação entre u e v é do tipo $u = a'v + b'$. Se repetirmos o método dos mínimos quadrados obtemos a' e b'. Podemos reformular o segundo modelo para $v = \frac{u}{a'} - \frac{b'}{a'}$.

Se a relação entre os dados fosse perfeitamente linear a e $\frac{1}{a'}$ teriam o mesmo valor. Ou seja, $a \times a'$ seria igual a 1. O coeficiente de correlação linear é dado por $r = \sqrt{aa'}$.

Podemos concluír que quando a relação entre duas grandezas é perfeitamente linear o coeficiente de correlação é unitário. Quando x e y são independentes, r é nulo.

É possível demonstrar que a quantidade:

$$r\sqrt{\frac{n-2}{1-r^2}} \tag{4.6}$$

está distribuída segundo t de Student (eq.4.3). Esta quantidade é utilizada para avaliar a validade da hipótese nula.

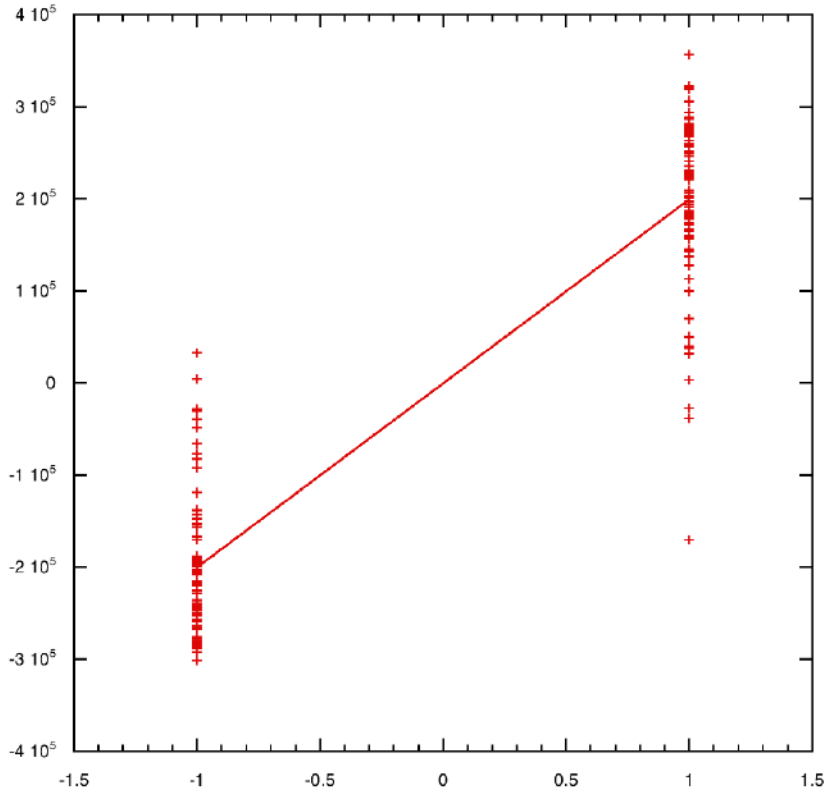


Figura 4.4: Representação gráfica de uma sequência temporal em função da sequência de modelo.

4.3.2 Aplicação

Na aplicação deste método mede-se qual é o coeficiente de correlação linear entre a sequência de modelo de dois estados e a sequência temporal de cada voxel (figura 4.4). A sequência de modelo pode ser com atraso e pode ter três formas distintas: a sequência de design, a sequência de design convolvida com HRF ou uma sequência temporal de outro voxel à escolha).

Determina-se a probabilidade da hipótese nula ser válida (p) para cada voxel. O resultado é apresentado sob a forma de um volume de probabilidade com o valor de $1 - p$ para cada voxel e gravável num ficheiro Analyze a partir da janela de resultado.

4.4 O método linear generalizado

4.4.1 Teoria

O método do modelo linear generalizado (MLG) é um método estatístico que surge como uma generalização do conceito de interpolação linear. Dado um conjunto de n pares de dados (t_i, v_i), na interpolação linear quantificam-se os coeficientes a_0 e a_1 da equação:

$$v(t) = a_0 \cdot 1 + a_1 \cdot t \tag{4.7}$$

Dito de outra forma assumimos que a variável v é uma combinação linear das funções 1 e t.

No método linear generalizado expandimos este conceito para uma combinação linear de quaisquer m funções:

$$v(t) = \sum_{j=0}^{m-1} a_j u_j(t) \tag{4.8}$$

As funções $u_j(t)$ são designadas de *funções de base* e podem ser não lineares.
Tal como na interpolação linear procuramos os coeficientes que minimizam a quantidade χ^2:

$$\chi^2 = \sum_{i=0}^{n-1} \left[\frac{v_i - \sum_{j=0}^{m-1} a_j u_j(t_i)}{\sigma_i} \right] \tag{4.9}$$

É possível demonstrar que o mínimo de χ^2 acontece quando os coeficentes são:

$$a_j = \sum_{k=0}^{m-1} C_{jk} \left[\sum_{i=0}^{n-1} \frac{v_i u_k(t_i)}{\sigma_i^2} \right] \tag{4.10}$$

em que C_{jk} é a matriz covariância dos parâmetros (ver apêndice D).

4.4.2 Aplicação

O *Cérebro* aplica este método calculando para cada sequência temporal quais os coeficientes a_j que satisfazem a seguinte condição:

$$v_i = \sum_{i=0}^{T} a_j D_{ij} \tag{4.11}$$

em que D_{ij} é o elemento da linha i e coluna j da matriz de design.

O sistema de equações obtido na eq. 4.11 é resolvido em ordem aos coeficientes a_j numericamente pelo método de eliminação de Gauss-Jordan. Este cálculo é feito pela classe GaussJordanElimination. Esta classe tem também o método getCovarianceArray() que gera o array bidimensional da matriz covariância.

Uma vez calculados os coeficientes para cada estado, o utilizador escolhe um para de estados (*e.g.* R e A). Os coeficientes são comparados pelo teste t de Student:

$$\theta = \frac{a_A - a_R}{\sqrt{C_{AA} + C_{RR}}} \tag{4.12}$$

Obtemos a probabilidade da hipótese nula ser válida (p) para cada voxel. O resultado é apresentado sob a forma de um volume de probabilidade com o valor de $1 - p$ para cada voxel e gravável num ficheiro Analyze a partir da janela de resultado.

4.5 O método de Fourier

4.5.1 Teoria

O método de Fourier baseia-se na transformada de Fourier de um sinal periódico. Pretende-se comparar o módulo da transformada de Fourier da sequência de modelo:

$$\tilde{u} = |\mathcal{F}(u)| \tag{4.13}$$

com o módulo da transformada de Fourier de cada sequência temporal:

$$\tilde{v} = |\mathcal{F}(v)| \tag{4.14}$$

Ou seja, pretende-se comparar espectros e gerar uma grandeza única que quantifique o grau de activação. O valor de tal grandeza deve ser ponderado por um factor que se aproxime de um quando a sequência temporal aproxima-se da sequência de modelo.

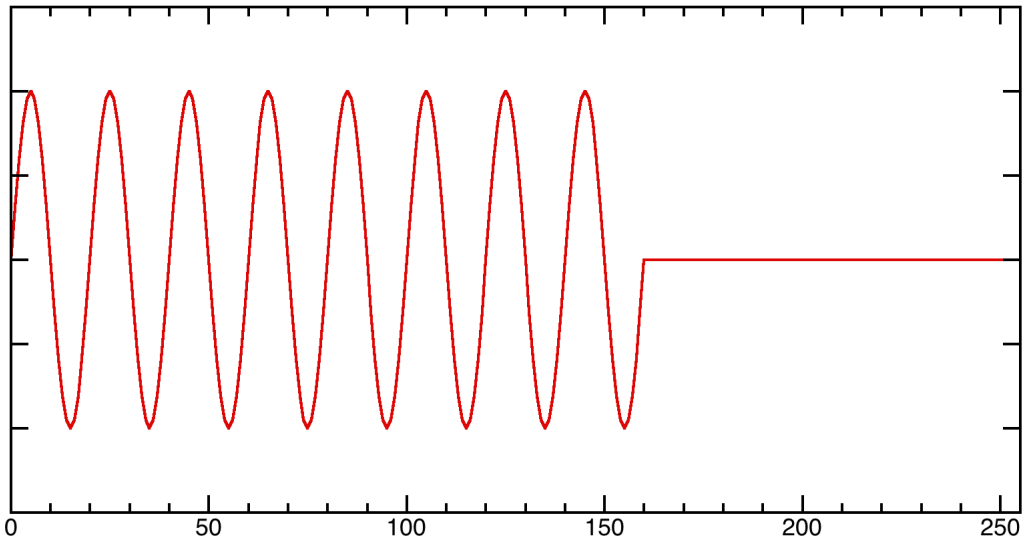


Figura 4.5: Sequência sinusoidal com extensão nula.

Este método é imune ao atraso do sinal temporal porque excluímos a informação de fase ao concentrarmo-nos somente no espectro.

4.5.2 Aplicação

A aplicação prática deste método é feita através da TRF ([21, 6, 7, 1, 2, 8]. Esta transformada funciona apenas com sequências de dados com um número de pontos que seja uma potência de dois. Como as sequências temporais têm um número arbitrário de pontos, é necessário acrescentar pontos artificiais aos dados originais. A literatura recomenda que se acrescente o número necessário de pontos com valor nulo [6, 21]. Esta técnica é designada neste trabalho de *extensão nula*.

Se a função tende para zero de forma que no último ponto da sequência original já está muito próxima de zero, a extensão nula é adequada porque não está a introduzir uma frequência superior à frequência de Nyquist. Porém, os dados funcionais de design em bloco têm uma frequência tipicamente entre $\frac{1}{5}$ e $\frac{1}{10}$ da frequência de Nyquist e não tendem suavemente para zero no final da sequência. Uma extensão nula provoca a introdução de uma transição abrupta para zero no final da sequência original. Esta transição traz um sinal com frequências superiores à frequência de Nyquist e o espectro passa a apresentar *derrame*.

Este fenómeno é apresentado nas figuras 4.5 e 4.6

Na primeira temos uma sequência de pontos de um sinal sinusoidal com extensão nula. Na segunda temos a respectiva TRF.

O fenómeno de derrame caracteriza-se pelo aparecimento de oscilações artificiais nas zonas em que o espectro apresenta picos.

Como a transformada de Fourier assume que o sinal irá repetir-se indefinidamente, podemos diminuir este problema com *auto-extensão* (ver fig. 4.7) em vez de extensão nula. Isto quer dizer que para acrescentar pontos à sequência temporal vamos buscar os pontos necessários ao início da mesma sequência. Como se ele se repetisse *ad eternum*. Agora, não há introdução de uma transição abrupta e torneia-se parcialmente o problema do derrame (ver fig. 4.8). Por esta razão o programa *Cérebro* realiza as TRF com auto-extensão.

Mesmo assim, o derrame surge quando um sinal tem ruído com frequências superiores à frequência de Nyquist. Neste caso recorre-se ao uso de janelas. Uma janela é uma função que modela a sequência de dados obrigando-a a tender para zero no final. O efeito da janela sobre o espectro é de eliminar o efeito de derrame. O preço de utilizá-las é uma redução da altura e um alargamento

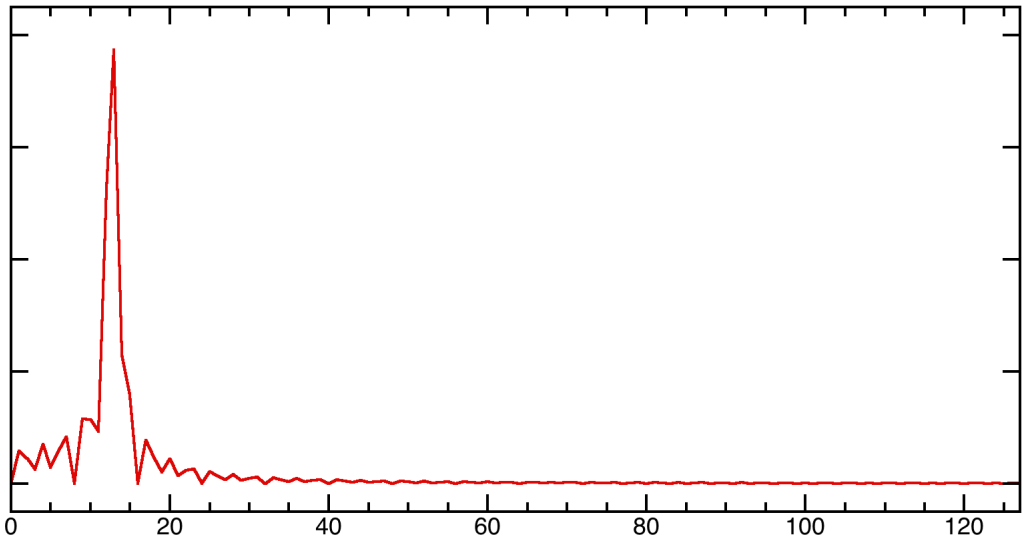


Figura 4.6: Espectro da sinusoide com extensão nula.

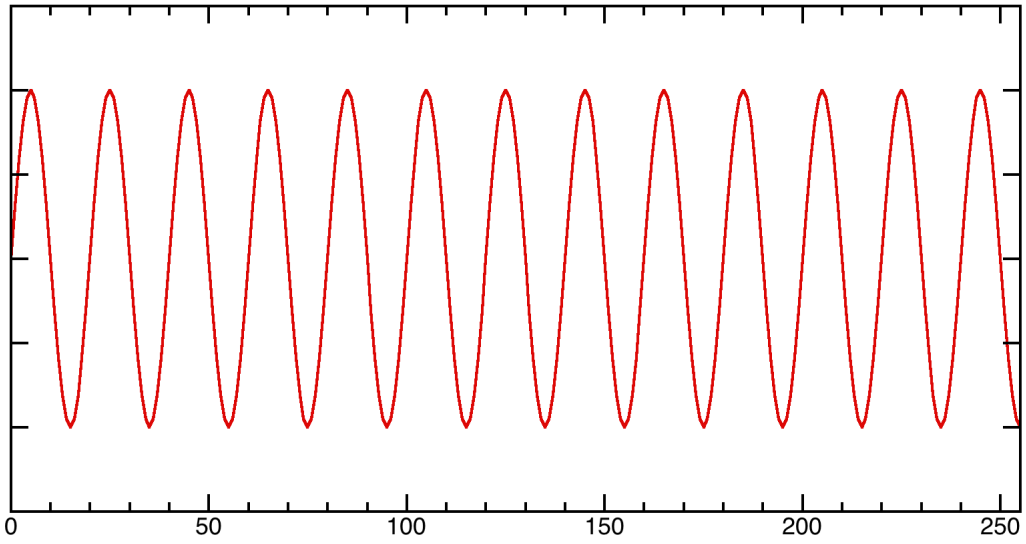


Figura 4.7: Sequência sinusoidal com auto-extensão.

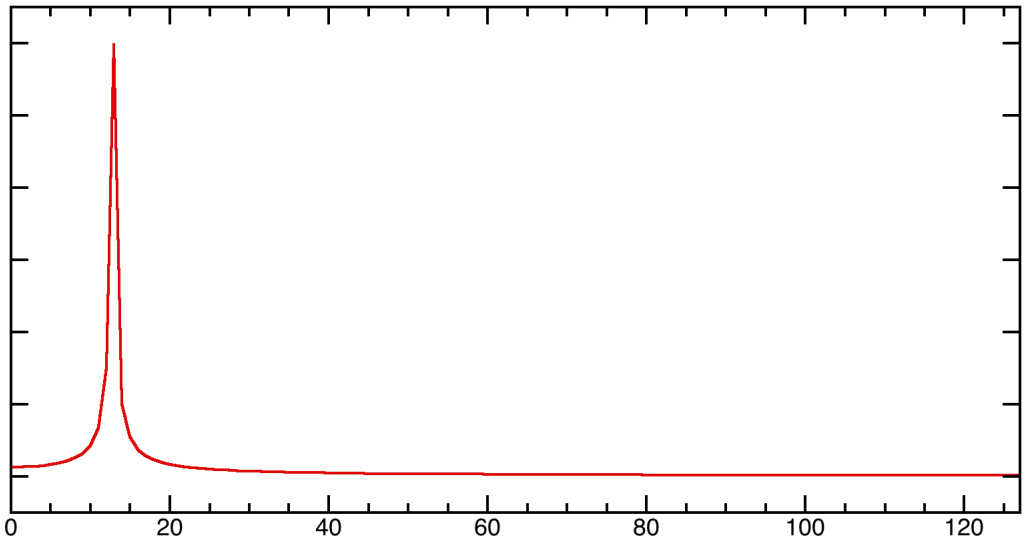


Figura 4.8: Espectro da sinusoide com auto-extensão.

dos picos. No programa *Cérebro* o utilizador pode escolher se quer que a TRF seja avaliada sem janela, com janela de Hann ou com janela de Hamming.

A auto-extensão revelou-se útil também na convolução (aplicada para a filtragem e para a modelação do vector de design com a HRF). É necessário realizar uma auto-extensão duas vezes. No final aplica-se a TRFI e retira-se apenas a sequência central para que o sinal mantenha o mínimo de integridade.

No método de avaliação de activação de picos de Fourier pretende-se estabelecer um parâmetro que inclua a altura do pico da primeira harmónica do espectro de cada sequência temporal mas que também seja influenciado pela semelhança deste espectro como o espectro da sequência de modelo. A altura do pico é proporcional à amplitude do sinal de activação. Para evitar que um sinal fortuito de igual frequência seja tomado como um sinal de activação (erro tipo I), o parâmetro deve ser pesado por um factor que avalie a semelhança dos dois espectros.

Uma primeira tentativa foi realizar uma interpolação linear entre o espectro da sequência temporal de cada voxel e o espectro da sequência de modelo.

$$\tilde{v}_i(f) = \alpha \cdot \tilde{u}_i(f) + \beta \tag{4.15}$$

O declive da linha interpolada (α) seria directamente proporcional à activação. Se as sequências fossem muito diferentes, o desvio padrão do declive seria elevado. Foi então decidido utilizar como parâmetro de activação (Π) a razão entre o declive e o seu desvio padrão:

$$\Pi = \frac{\alpha}{\sigma_\alpha} \tag{4.16}$$

Verificou-se na prática que os pontos do gráfico podem dividir-se em dois grupos. O primeiro é formado pelos valores baixos dos espectros e é o mais abundante. O segundo é formado pelos pontos dos picos que são em muito menor número.

Numa interpolação linear todos os pontos têm igual peso no cálculo do declive da recta da fig.4.9. Como o primeiro grupo de pontos é muito mais numeroso do que o segundo, é muito mais determinante no resultado do declive. De facto a linha em geral nem passa perto dos pontos dos picos.

O método é então aplicado apenas ao segundo grupo de pontos (ver 4.10).

Foi criada uma classe chamada SpectrumAnalyzer que toma o espectro da sequência de modelo e identifica as frequências dos vários picos. A identificação dos picos é feita procurando os pontos

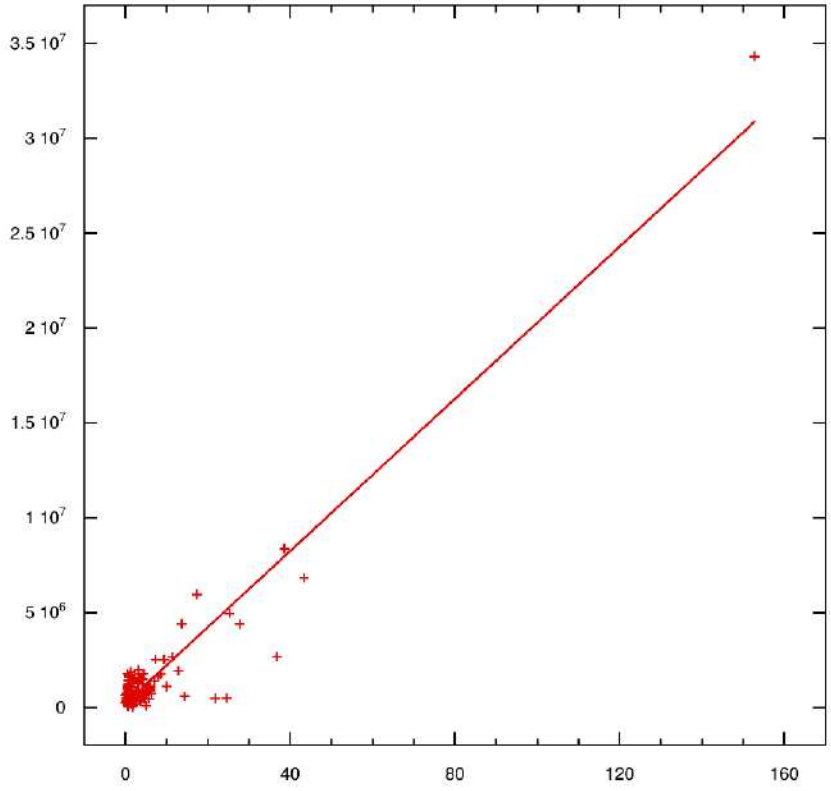


Figura 4.9: Representação gráfica do espectro de uma sequência em função do espectro da sequência de modelo.

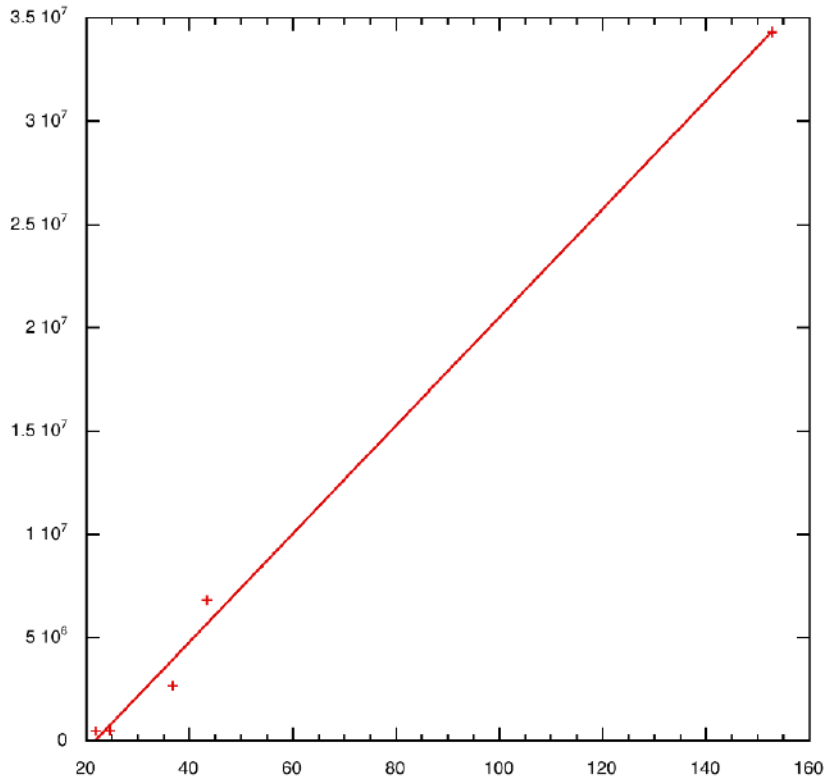


Figura 4.10: Representação gráfica dos picos de um espectro de uma sequência em função dos picos do espectro da sequência de modelo.

de derivada nula e de segunda derivada negativa. O espectro da sequência de modelo é calculado pela TRF com uma janela de Hann para evitar os picos do efeito de derrame.

Poder-se-ia ter determinado as frequências dos picos teoricamente assumindo que a sequência de modelo era sempre um sinal quadrado. Esta técnica foi rejeitada por limitar a utilização deste método a designs de bloco.

Uma vez identificadas as frequências dos picos faz-se uma interpolação linear apenas com os pontos (k) dos picos:

$$\tilde{v}_k(f) = \alpha \cdot \tilde{u}_k(f) + \beta \tag{4.17}$$

Se a sequência de modelo está modelada pela função HRF a posição dos picos do espectro não se altera. Há apenas um aumento do primeiro pico e uma diminuição muito pequena e progressiva dos restantes picos em relação a uma sequência de modelo quadrada. Isto quer dizer que o método é válido também para este caso.

Assume-se que o parâmetro Π tem uma distribuição normal. A hipótese nula é de que o parâmetro tem um valor inferior ao medido. Obtemos a probabilidade da hipótese nula ser válida (p) para cada voxel. O resultado é apresentado sob a forma de um volume de probabilidade com o valor de $1-p$ para cada voxel e gravável num ficheiro Analyze a partir da janela de resultado.

4.6 O método de amplitude

4.6.1 Teoria

O método de amplitude baseia-se na ideia de modificar a sequência de modelo de forma a adaptar-se a cada sequência temporal. A activação é avaliada pela razão entre a amplitude (Amp) do sinal obtido e o desvio padrão dos resíduos:

$$\Pi = \frac{Amp}{\sqrt{Var(res)}}$$

4.6.2 Aplicação

Para redimensionarmos a sequência de modelo (de u_i para $\underline{u}_i$) tomamos para cada sequência temporal todos os valores de v obtidos durante o estímulo como pertencentes ao primeiro estado e todos os obtidos durante o repouso como pertencentes ao segundo estado. Calculamos a média para os dados do primeiro estado ($\overline{v_A}$) e repetimos para os dados do segundo estado ($\overline{v_R}$).

Como a sequência de modelo tem amplitude unitária, os valores da mesma que pertencem ao estado A são multiplicados por $\overline{v_A}$ e os relativos ao estado R são multiplicados por ($\overline{v_R}$).

Os resíduos são calculados subtraindo a sequência de modelo ampliada de cada sequência temporal:

$$r_i = v_i - \underline{u}_i$$

A amplitude é dada por:

$$\overline{v_A} - \overline{v_R}$$

Assume-se que o parâmetro Π tem uma distribuição normal. A hipótese nula é de que o parâmetro tem um valor inferior ao medido. Obtemos a probabilidade da hipótese nula ser válida (p) para cada voxel. O resultado é apresentado sob a forma de um volume de probabilidade com o valor de $1-p$ para cada voxel e gravável num ficheiro Analyze a partir da janela de resultado.

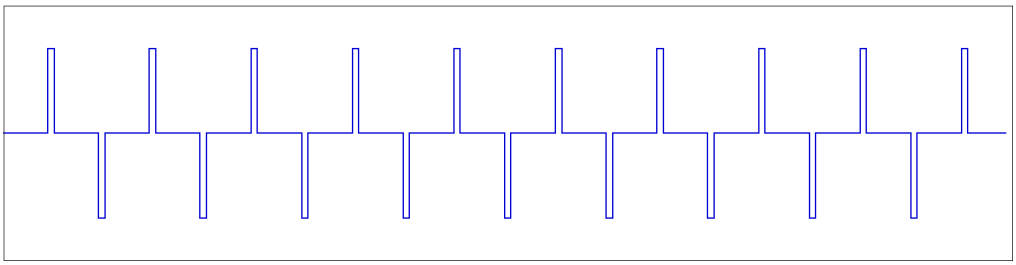


Figura 4.11: Derivada da sequência de modelo.

4.7 O método de sobreposição

4.7.1 Teoria

Nesta tese, o tempo que decorre entre a ocorrência de um estímulo externo e a resposta fisiológica hemodinâmica é designado de *atraso* δt. Ele inclui o tempo de aquisição dos dados por parte do cérebro, o tempo que os neurónios funcionam sem haver um acréscimo da necessidade de oxigénio e o tempo que o sistema vascular demora a reagir quando finalmente surge uma necessidade de aumento dos níveis de oxigenação. O atraso é em média de 5 a 6 s.

Assumindo que o atraso é variável ao longo do volume inteiro (ou seja, zonas diferentes podem ser recrutadas em instantes diferentes):

$$\delta t = \delta t\,(x, y, z) \tag{4.18}$$

uma compensação temporal do sinal extraído dos dados deve ser calculada para cada voxel. Isto porque as análises estatísticas são sempre realizadas tendo por base a sequência temporal dos estímulos.

Inicialmente, o método da sobreposição foi desenhado com o intuito único de detectar transições rápidas e periódicas do sinal temporal.

Calcula-se a primeira derivada temporal da sequência de design:

$$u_k^{'} = u_{k+1} - u_k \tag{4.19}$$

É nula em todos os pontos excepto nos instantes de transição entre estados diferentes (ver fig. 4.11).

A derivação de todas as sequências temporais:

$$v_k^{'} = v_{k+1} - v_k \tag{4.20}$$

evidenciam picos periódicos nos voxels de maior activação (ver 4.12).

Para quantificar até que ponto há um padrão semelhante ao da primeira derivada temporal da sequência de design, é calculado o produto interno entre os dois vectores. Ou seja, para cada instante é calculado o produto das derivadas e depois soma-se todas estas parcelas:

$$w_0^{'} = \sum_{k=0}^{n-2} u_k^{'} v_k^{'} \tag{4.21}$$

O produto interno só é elevado quando acontece grande sobreposição dos sinais (os picos coincidem). Verifica-se no entanto, que mesmo nos voxels com grande activação muitas vezes o produto interno é muito pequeno. A sobreposição é fraca quando há atraso.

Compara-se o produto interno para sucessivas translações temporais da derivada da sequência de modelo:

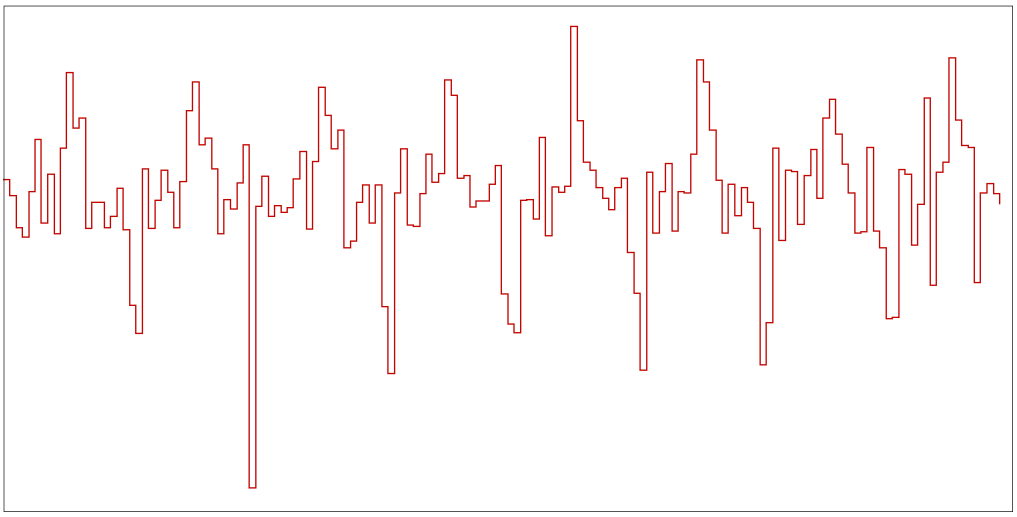


Figura 4.12: Derivada de uma sequência temporal.

$$w_j = \sum_{k=0}^{n-2} u'_{k+j} v'_k \tag{4.22}$$

e toma-se o maior valor:

$$w = max\,(w_j) \tag{4.23}$$

Como os picos são maiores nas zonas de maior activação (maior amplitude) o produto interno pode ser tomado como uma medida de activação.

Para além de permitir uma avaliação da activação, este método oferece algo novo. Sabendo o número de instantes de translação com que se obteve o valor máximo temos agora uma estimativa do atraso de cada voxel em relação à sequência de estímulos:

$$\delta t = j \cdot \Delta t \tag{4.24}$$

em que Δt é o tempo de aquisição (TR).

Pode-se aplicar este método tanto para a primeira derivada como para outra derivada de qualquer ordem (incluindo a própria função).

$$w_k^{(j)} = \sum_{i=0}^{n-1-j} u_{i+k}^{(j)} \cdot v_i^{(j)} \tag{4.25}$$

Os dados do atraso podem ser mapeados para todo o volume e podem ser utilizados para melhorar a aplicação dos métodos mais sensíveis ao atraso (e.g. Student, coeficiente de correlação).

$$\omega^{(j)} = max_k \left(w_k^{(j)} \right) \tag{4.26}$$

4.7.2 Aplicação

Após os testes verificou-se que os volumes de maior activação diminuem para ordens iguais ou superiores a 3. Por isso o programa *Cérebro* dá como opções possíveis apenas as ordens 0, 1 e 2.

O parâmetro $w_k^{(j)}$ tem uma distribuição segundo qui-quadrado. O programa *Cérebro* constroi a densidade de probabilidade a partir do histograma. O valor de p é calculado obtendo a probabilidade cumulativa por integração numérica da densidade de probabilidade.

Método	maxVox	minVox	V (%)
Student	(29,7,16)	(15,19,21)	39.2
Coef Corr	(29,7,16)	(15,19,21)	41.8
picos de Fourier	(27,6,17)	(26,32,15)	5.1
MLG	(27,6,17)	(31,11,23)	4.7
Amplitude	(27,6,17)	(28,30,32)	1.9
Sobrep(0)	(27,6,17)	(29,29,20)	4.7
Sobrep(1)	(36,56,14)	(0,0,0)	4.7
Sobrep(2)	(28,30,24)	(0,0,0)	4.5

Tabela 4.1: Voxels de activação máxima e mínima e volume de activação em função dos métodos para dados filtrados.

O resultado é apresentado sob a forma de um volume de probabilidade com o valor de $1-p$ para cada voxel e gravável num ficheiro Analyze a partir da janela de resultado.

4.8 Análise comparativa

Para realizar uma análise comparativa dos seis métodos de avaliação de activação escolheu-se um estudo de atenção visual com alta resolução e já largamente estudado em vários artigos ([3], [5]) e livros ([15], [17]).

Esta análise é feita de acordo com os seguintes parâmetros de activação: extremos, volumes, sequências, espectros e perfis.

Os dados utilizados foram os originais e os filtrados (passa-alto com frequência de corte 0.012 Hz). Os volumes de activação são apresentados em valores relativos ao volume da máscara $\left(V = 1632cm^3\right)$ porque o volume da máscara é uma estimativa por excesso do volume real.

Os volumes são calculados multiplicando o número de voxels do volume de probabilidade com valor igual ou maior a $1-p$ (neste caso $p = 0.05$) pelo volume de um voxel.

4.8.1 Extremos e volumes

Nesta secção apresentamos os voxels de activação máxima e mínima para cada método assim como a percentagem do volume total.

Na tabela 4.1, o voxel de máxima activação mais frequente é o (27,6,17). Para os métodos de Student e de coeficiente de correlação o voxel de activação máxima não é o mesmo dos outros mas encontra-se na mesma vizinhança (dentro de um volume de $2 \times 1 \times 1$). Isto sugere que estes voxels não são resultados com erro do tipo I. Para o método de sobreposição (ordens 1 e 2) os voxels de activação máxima estão muito longe do voxel (27,6,17). Estes sugerem à primeira vista erros do tipo I.

Por outro lado os voxels de activação mínima variam muito entre os diferentes métodos. São iguais apenas para os métodos de Student e de coeficiente de correlação.

Quanto aos volumes de activação verificamos logo que os métodos de Student e de coeficiente de correlação fazem uma estimativa de volume muito maior do que para os outros métodos (p foi igual para todos). Dos restantes métodos o que apresenta maior volume de activação (menos selectivo) é o de picos de Fourier. O que apresenta um menor volume é o de amplitude (cerca de 40% dos outros).

O métodos de modelo linear generalizado e de sobreposição apresentam resultados muito semelhantes. Isto sugere que apesar dos máximos de sobreposição de ordens 1 e 2, é possível que os volumes de activação sejam válidos.

Na tabela 4.2, desta vez quase podemos distinguir os mesmos grupos de máximos. Os métodos de Student e de coeficiente de correlação alteraram a sua posição por um voxel segundo x mas mantiveram-se na mesma área. Do grupo dos métodos com máximo em (27,6,17) apenas o método

Método	maxVox	minVox	V (%)
Student	(30,7,16)	(22,33,42)	29.4
Coef Corr	(30,7,16)	(22,33,42)	32.0
picos de Fourier	(27,6,17)	(26,32,15)	5.1
MLG	(27,6,17)	(27,27,11)	4.7
Amplitude	(28,6,17)	(26,23,10)	4.8
Sobrep(0)	(27,6,17)	(26,23,9)	4.6
Sobrep(1)	(36,56,14)	(0,0,0)	4.5
Sobrep(2)	(28,30,24)	(0,0,0)	4.5

Tabela 4.2: Voxels de activação máxima e mínima e volume de activação em função dos métodos para dados não filtrados e com um atraso nulo.

Método	maxVox	minVox	V (%)
Student	(29,7,16)	(18,22,24)	30.0
Coef Corr	(29,7,16)	(18,22,24)	33.2
picos de Fourier	(27,6,17)	(25,35,16)	5.2
MLG	(27,6,17)	(48,22,12)	4.7
Amplitude	(28,6,17)	(26,23,10)	4.8
Sobrep(0)	(27,6,17)	(26,23,10)	4.5
Sobrep(1)	(36,56,14)	(0,0,0)	4.5
Sobrep(2)	(28,30,24)	(0,0,0)	4.4

Tabela 4.3: Voxels de activação máxima e mínima e volume de activação em função dos métodos para dados não filtrados e com um atraso unitário.

de amplitude afastou-se um voxel segundo x. Finalmente os voxels de ordem superior do método de sobreposição não sofreram qualquer alteração.

Nos mínimos a mudança foi grande. Os métodos de Student e de coeficiente de correlação continuaram a ser iguais entre si mas alteraram a sua posição muito (da fatia 21 para a 42). Os únicos métodos que não mudaram de voxel mínimo foram o de picos de Fourier e o de sobreposição para as ordens 1 e 2. Os restantes métodos concentraram os mínimos na mesma zona (fatias 9, 10 e 11).

Quanto aos volumes, verificou-se uma grande redução em relação aos dados filtrados para os métodos de Student e coeficiente de correlação (da ordem dos 10%). A variação dos restantes não foi assinalável com excepção do método de amplitude que apresentou um aumento de 2.9%.

Na tabela 4.3 os máximos mantiveram-se nas mesmas posições com excepção dos métodos de Student e de coeficiente de correlação. Estes tomaram a posição obtida para os dados filtrados.

Os métodos de amplitude e sobreposição de ordens 1 e 2 mantiveram os voxels mínimos. O método de sobreposição de ordem nula teve uma variação mínima. O método de picos de Fourier sofreu uma variação maior mas manteve-se nas imediações das anteriores.

No que toca aos volumes as mudanças foram pouco significativas face aos dados não filtrados anteriores.

Na tabela 4.4 os máximos para os métodos de Student e coeficiente de correlação sofrem uma grande mudança de posição para o voxel (17,8,10). A verificar-se a validade deste máximo, isto sugere que é possível detectarem-se diferentes zonas activadas ao longo do tempo. Ou seja, uma sequência de activações.

A permanência nas posições dos máximos para os outros métodos sugere também que estes são insensíveis (ou pouco sensíveis) à evolução temporal das activações.

Os voxels mínimos não se alteram para os métodos de Student, coeficiente de correlação e sobreposição (ordens 1 e 2) em relação aos de atraso unitário. O método de picos de Fourier altera o voxel mínimo mas pouco. O método linear generalizado sofre um grande mudança para a periferia da fatia 22 e os métodos de amplitude e sobreposição de ordem zero têm activação

Método	maxVox	minVox	V (%)
Student	(17,8,10)	(18,22,24)	32.5
Coef Corr	(17,8,10)	(18,22,24)	36.2
picos de Fourier	(27,6,17)	(25,32,15)	5.1
MLG	(27,6,17)	(7,36,22)	4.8
Amplitude	(28,6,17)	(28,30,32)	4.8
Sobrep(0)	(27,6,17)	(28,30,32)	4.7
Sobrep(1)	(36,56,14)	(0,0,0)	4.4
Sobrep(2)	(28,30,24)	(0,0,0)	4.7

Tabela 4.4: Voxels de activação máxima e mínima e volume de activação em função dos métodos para dados não filtrados e com um atraso igual a 2.

Método	maxVox	minVox	V (%)
Student	(23,6,26)	(18,22,24)	34.3
Coef Corr	(23,6,26)	(18,22,24)	38.3
picos de Fourier	(27,6,17)	(28,30,22)	4.9
MLG	(27,6,17)	(23,42,19)	4.8
Amplitude	(27,6,17)	(28,30,32)	4.8
Sobrep(0)	(27,6,17)	(28,30,25)	4.7
Sobrep(1)	(36,56,14)	(0,0,0)	4.8
Sobrep(2)	(28,30,24)	(0,0,0)	4.7

Tabela 4.5: Voxels de activação máxima e mínima e volume de activação em função dos métodos para dados não filtrados e com um atraso igual a 3.

mínima no mesmo voxel tal como acontecia para um atraso unitário.

Os volumes aumentaram para os métodos de Student e coeficiente de correlação (2.5% e 3.0% respectivamente). Para os restantes a variação foi pouco significativa ($\leq 0.2\%$).

O último grupo de resultados não filtrados é para um atraso de 3 TR (ver tabela 4.5).

Novamente verifica-se uma mudança das posições dos máximos para os métodos de Student e de coeficiente de correlação. Ou seja, mantém-se a hipótese de que em instantes sucessivos as zonas de activação estão em mudança (neste caso de (17,8,10) para (23,6,26)). Quando testarmos as sequência temporais e espectros teremos oportunidade de rever esta hipótese.

Os mínimos dos métodos de Student, coeficiente de correlação e sobreposição de ordem zero, mais uma vez mantiveram-se constantes. A amplitude também não variou.

Os volumes continuaram o seu aumento para os métodos de Student e coeficiente de correlação. Para os restantes a maior variação foi positiva e foi de 0.4%.

Na tabela 4.6 não há variação das posições dos máximos relativamente ao caso dos dados

Método	maxVox	minVox	V (%)
Student	-	-	-
Coef Corr	(29,7,16)	(10,16,23)	40.9
picos de Fourier	(27,6,17)	(26,32,15)	5.1
MLG	-	-	-
Amplitude	(27,6,17)	(28,30,32)	1.8
Sobrep(0)	(27,6,17)	(29,29,20)	4.7
Sobrep(1)	(36,56,14)	(0,0,0)	4.7
Sobrep(2)	(28,30,24)	(0,0,0)	4.4

Tabela 4.6: Voxels de activação máxima e mínima e volume de activação em função dos métodos para dados filtrados e com HRF.

Método	maxVox	minVox	V (%)
Student	-	-	-
Coef Corr	(30,7,16)	(22,33,42)	31.1
picos de Fourier	(27,6,17)	(26,32,15)	5.1
MLG	-	-	-
Amplitude	(27,6,17)	(26,23,10)	4.8
Sobrep(0)	(27,6,17)	(26,23,10)	4.9
Sobrep(1)	(36,56,14)	(0,0,0)	4.4
Sobrep(2)	(28,30,24)	(0,0,0)	4.5

Tabela 4.7: Voxels de activação máxima e mínima e volume de activação em função dos métodos para dados não filtrados, com HRF e com um atraso nulo.

Método	maxVox	minVox	V (%)
Student	-	-	-
Coef Corr	(29,7,16)	(18,22,24)	32.3
picos de Fourier	(27,6,17)	(25,35,16)	5.2
MLG	-	-	-
Amplitude	(28,6,17)	(26,23,10)	4.7
Sobrep(0)	(27,6,17)	(26,23,10)	4.8
Sobrep(1)	(36,56,14)	(0,0,0)	4.5
Sobrep(2)	(28,30,24)	(0,0,0)	4.8

Tabela 4.8: Voxels de activação máxima e mínima e volume de activação em função dos métodos para dados não filtrados, com HRF e com um atraso unitário.

filtrados iniciais.

Nos mínimos há apenas alteração da posição para o método do coeficiente de correlação.

Nos volumes observa-se que o método de coeficiente de correlação fica mais selectivo do que era para os dados filtrados sem HRF. Essa tendência confirma-se também para os outros métodos, embora de forma menos intensa. Confirma-se que o método de amplitude é muito mais selectivo que os restantes para dados filtrados.

Para os dados não filtrados, com HRF e atraso igual a 0 (ver tabela 4.7).

Nos máximos apenas o voxel do método de amplitude modifica-se relativamente aos mesmos dados sem HRF. A variação é de apenas 1 voxel segundo x.

Nos mínimos só o método de sobreposição de ordem zero vê uma alteração de voxel mas com uma variação unitária segundo z.

O método do coeficiente de correlação revela-se novamente mais selectivo com uma sequência de modelo com HRF. O volume diminuiu 0.9% relativamente aos mesmos dados sem HRF.

Na tabela 4.8 verificamos que não há alteração das posições dos máximos relativamente aos mesmos dados sem HRF. O mesmo acontece com os mínimos.

Observa-se novamente uma diminuição do volume em 0.9% para o método do coeficiente de correlação em relação aos mesmos dados sem HRF.

Na tabela 4.9 os máximos mantêm-se em relação aos mesmos dados sem HRF com excepção de dois métodos. O método do coeficiente de correlação que antes tinha identificado máximos no voxel (17,8,10) muda para o voxel (24,6,20). O método de amplitude sofre um ajuste de um voxel segundo x.

As posições dos mínimos não se alteram.

O volume de activação para o método do coeficiente de correlação tem uma variação ainda negativa mas de menor módulo (0.8%). Para o método de sobreposição a variação é positiva para as ordens 0 e 1. A segunda tem uma variação maior (0.4%).

Na tabela 4.10 as posições de activação máxima e mínima não mudam relativamente aos mesmos dados sem HRF.

Método	maxVox	minVox	V (%)
Student	-	-	-
Coef Corr	(24,6,20)	(18,22,24)	35.4
picos de Fourier	(27,6,17)	(25,32,15)	5.1
MLG	-	-	-
Amplitude	(27,6,17)	(28,30,32)	4.8
Sobrep(0)	(27,6,17)	(28,30,32)	4.8
Sobrep(1)	(36,56,14)	(0,0,0)	4.8
Sobrep(2)	(28,30,24)	(0,0,0)	4.6

Tabela 4.9: Voxels de activação máxima e mínima e volume de activação em função dos métodos para dados não filtrados, com HRF e com um atraso igual a 2.

Método	maxVox	minVox	V (%)
Student	-	-	-
Coef Corr	(23,6,26)	(18,22,24)	37.9
picos de Fourier	(27,6,17)	(28,30,22)	4.9
MLG	-	-	-
Amplitude	(27,6,17)	(28,30,32)	4.8
Sobrep(0)	(27,6,17)	(28,30,25)	4.3
Sobrep(1)	(36,56,14)	(0,0,0)	4.8
Sobrep(2)	(28,30,24)	(0,0,0)	4.7

Tabela 4.10: Voxels de activação máxima e mínima e volume de activação em função dos métodos para dados não filtrados, com HRF e com um atraso igual a 3.

Nos volumes vemos uma diminuição de 0.4% para o método do coeficiente de correlação e para o método de sobreposição de ordem 1.

Finalmente temos um grupo de duas tabelas para testar a conectividade funcional com o voxel (27,6,17) (tabela 4.11).

Naturalmente que os máximos estão todos no voxel (27,6,17). O mínimo para o método do coeficiente de correlação é na mesma posição que o obtido com os dados filtrados sem HRF. O mínimo para o método de picos de Fourier é no mesmo voxel que foi obtido para um atraso unitário dos dados não filtrados. Dos restantes apenas se mantiveram as posições mínimas para o método de sobreposição de ordens 1 e 2.

O método de amplitude apresenta o volume mais pequeno de todos os dados. O método do coeficiente de correlação apresenta um volume muito maior face aos outros dados filtrados (sem e com HRF).

Na tabela 4.12 há um máximo surpreendente para o método de sobreposição de ordem 0.

Método	maxVox	minVox	V (%)
Student	-	-	-
Coef Corr	(27,6,17)	(15,19,21)	49.2
picos de Fourier	(27,6,17)	(25,35,16)	5.1
MLG	-	-	-
Amplitude	(27,6,17)	(14,55,13)	1.6
Sobrep(0)	(27,6,17)	(25,24,22)	4.7
Sobrep(1)	(27,6,17)	(0,0,0)	4.6
Sobrep(2)	(27,6,17)	(0,0,0)	4.7

Tabela 4.11: Voxels de activação máxima e mínima e volume de activação em função dos métodos para dados filtrados, e com sequência de modelo igual à sequência do voxel (27,6,17).

Método	maxVox	minVox	V (%)
Student	-	-	-
Coef Corr	(27,6,17)	(17,21,24)	66.2
picos de Fourier	(27,6,17)	(25,35,16)	5.1
MLG	-	-	-
Amplitude	(27,6,17)	(26,23,10)	4.7
Sobrep(0)	(28,24,18)	(0,0,0)	3.8
Sobrep(1)	(27,6,17)	(0,0,0)	4.6
Sobrep(2)	(27,6,17)	(0,0,0)	4.5

Tabela 4.12: Voxels de activação máxima e mínima e volume de activação em função dos métodos para dados não filtrados, e com sequência de modelo igual à sequência do voxel (27,6,17).

Veremos nas sequências qual a razão para este facto.

O método de picos de Fourier mostra uma variação nula na posição do mínimo em relação aos dados filtrados.

4.8.2 Sequências e espectros

Vejamos os resultados da primeira tabela a partir das sequências temporais. O voxel (27,6,17) tem nitidamente maior amplitude que o voxel (29,7,16) (fig. 4.13).

No entanto os métodos de Student e de coeficiente de correlação têm um máximo de probabilidade para o segundo. Isto é porque a variância dos valores para cada um dos dois estados é maior. Ou seja, há mais flutuação nos extremos da sequência (27,6,17).

No método de Student, a medida estatística é inversamente proporcional a esta variância, logo o parâmetro obtido é menor (ver eq.4.2).

No método do coeficiente de correlação, quanto maior é a dispersão em cada um dos estados menor é o valor de r (ver eq.4.6).

Uma possível correcção para os métodos de Student e de coeficiente de correlação poderia envolver uma filtragem de altas frequência reduzindo assim, a variância nos extremos dos sinais.

Se compararmos o sinal com maior probabilidade pelo método de sobreposição de ordem 1 com a sequência de (27,6,17) (fig. 4.14) vemos que as amplitudes são comparáveis. O valor elevado do produto interno dos vectores (eq.4.22) deve-se principalmente aos picos muito pronunciados na zona de transição. A derivada nestes picos é elevada. Note-se que apesar de haver um atraso de cerca de 10 TR o método detectou o sinal.

Uma consulta das imagens anatómicas mostra que esta é uma área ocular. Na realidade estamos a observar actividade vascular ocular.

O espectro obtido pela TRF indica que apesar do sinal ter uma forma semelhante ao do voxel (27,6,17) a altura do pico da primeira harmónica é muito mais pequena. Ou seja, a sua amplitude é menor.

O voxel (28,30,24) tem uma sequência com grande frequência (fig. 4.16) e uma amplitude comparável à sequência (27,6,17). A sua forma não é semelhante à de (27,6,17).

Mesmo que exista uma parcela do sinal com frequência semelhante à da primeira harmónica da sequência de (27,6,17) é de difícil visualização pela variação temporal. Para verificar se existe recorremos à TRF (fig. 4.17).

Mesmo que exista um sinal com a mesma forma da sequência de (27,6,17), o pico da primeira harmónica indica que é de pequena amplitude. Concluímos que se trata de um erro do tipo I. De facto se tivermos em conta a sua localização anatómica e a sua frequência podemos especular que se trata de um voxel onde passa uma artéria principal. A alta frequência virá então das variações de fluxo devidas ao batimento cardíaco.

A segunda derivada da sequência de modelo terá uma oscilação semelhante em termos de frequência e é possivel que o método de sobreposição para ordem 2 tenha tendência para erros do

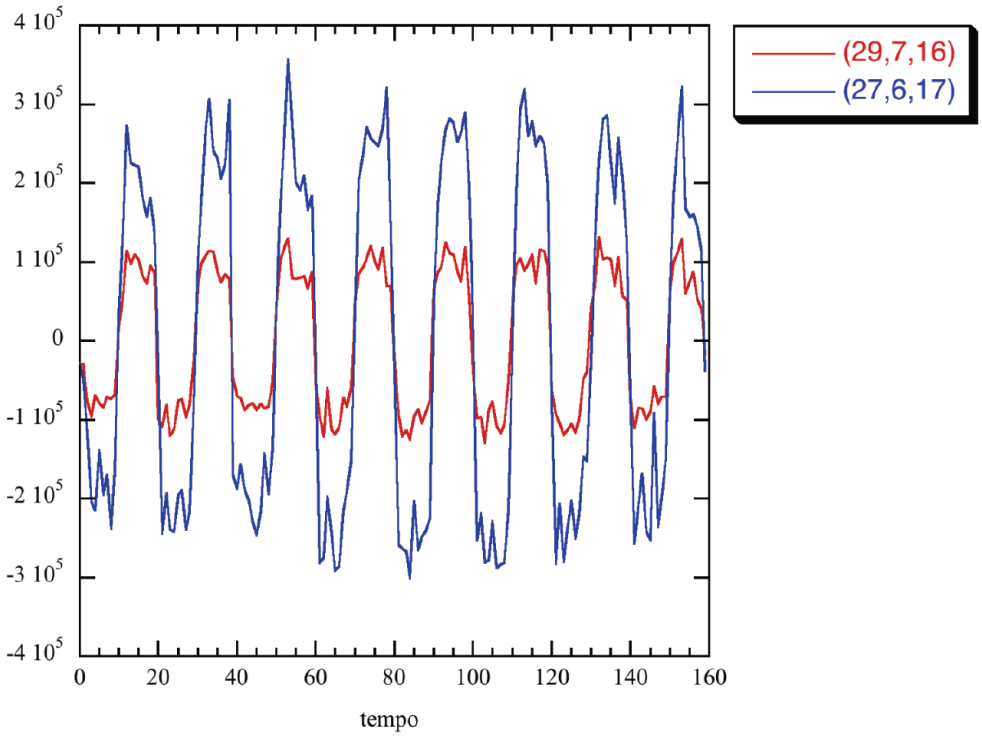


Figura 4.13: Comparação das sequências dos voxels (29,7,16) e (27,6,17).

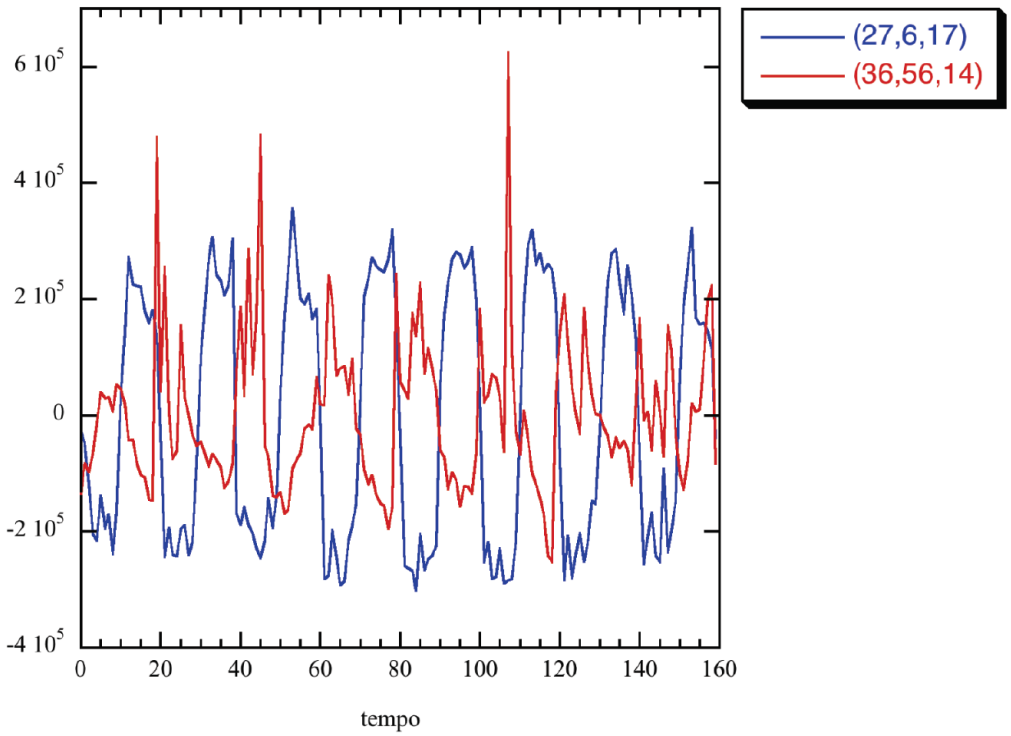


Figura 4.14: Comparação das sequências dos voxels (27,6,17) e (36,56,14).

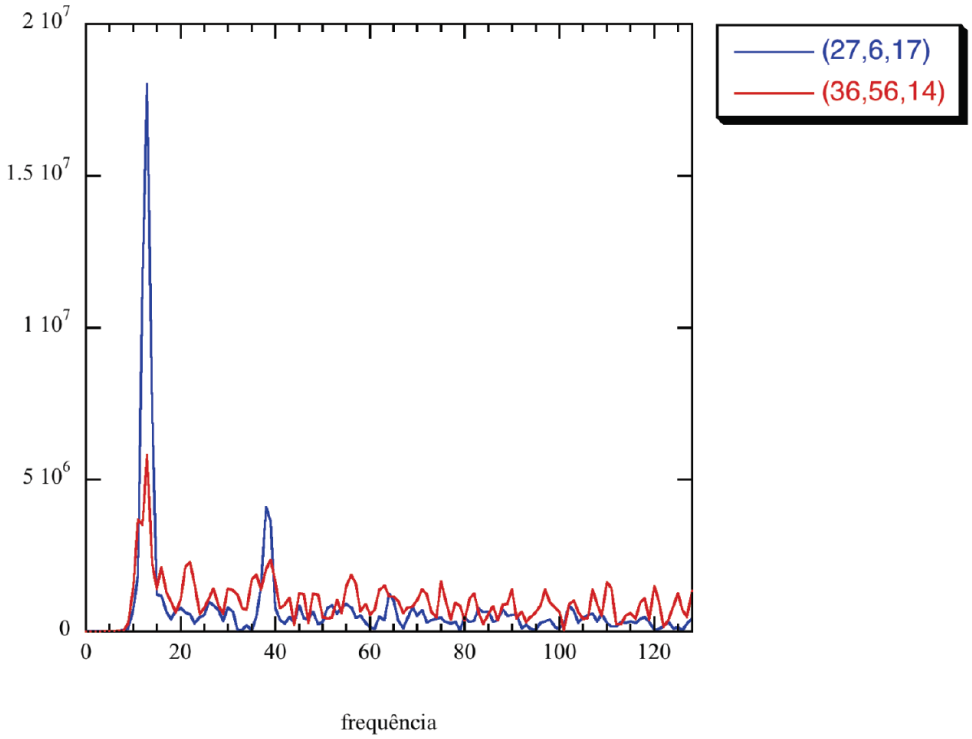


Figura 4.15: Comparação dos espectros das sequências dos voxels (27,6,17) e (36,56,14).

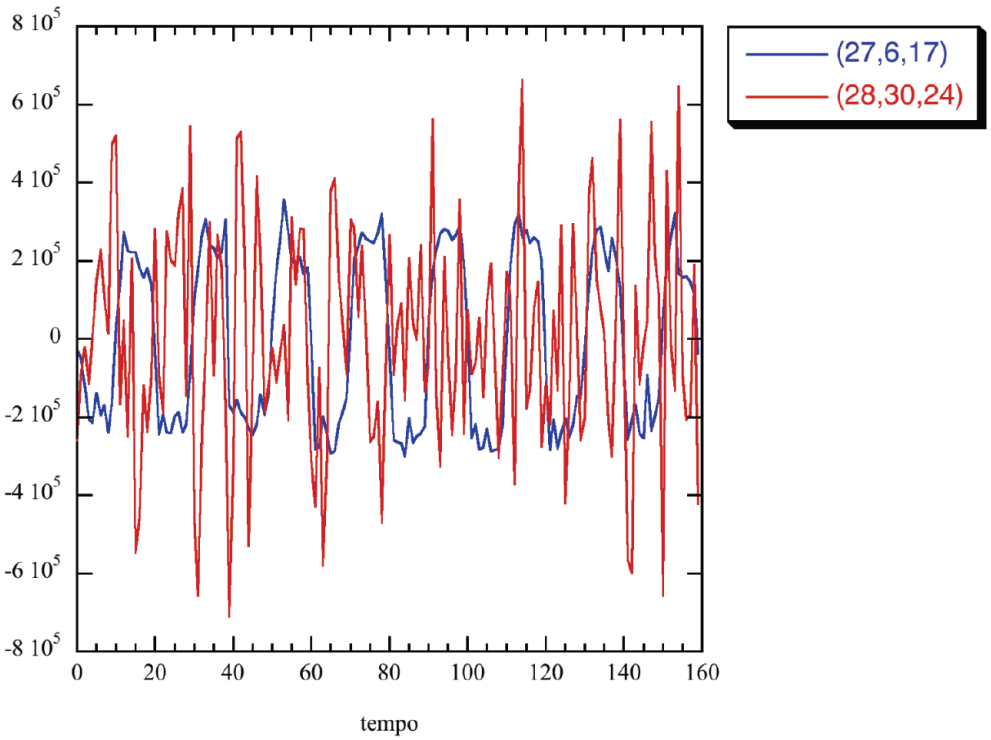


Figura 4.16: Comparação das sequências dos voxels (27,6,17) e (28,39,24).

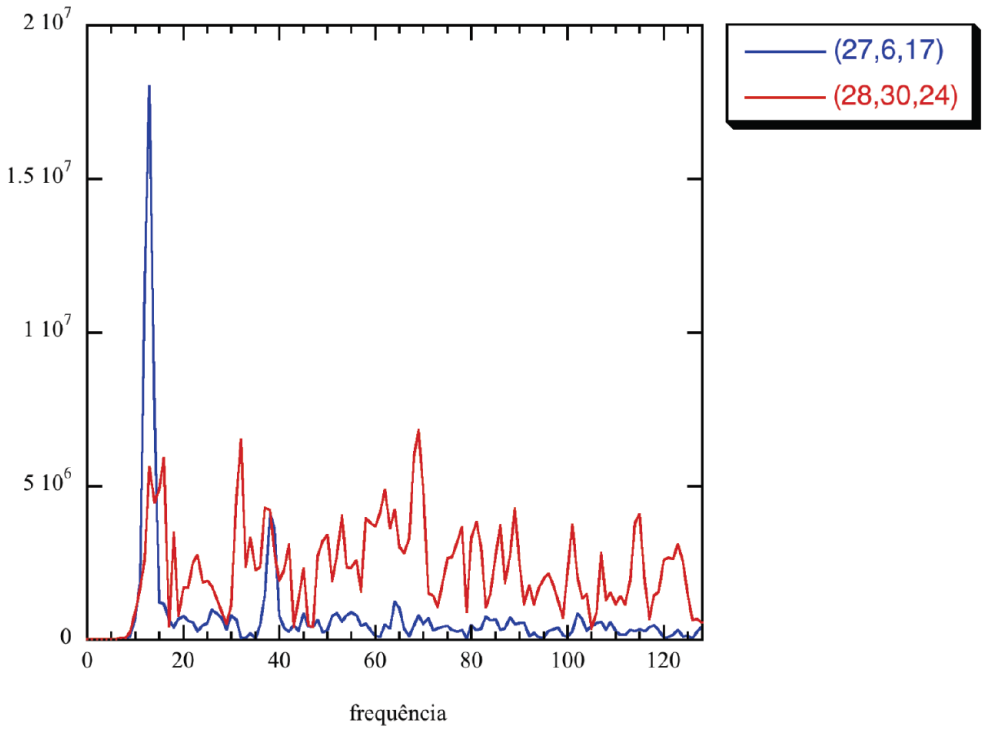


Figura 4.17: Comparação dos espectros das sequências dos voxels (27,6,17) e (28,30,24).

tipo I com sinais como o do voxel (28,30,24). Uma correcção possível para este método é filtrar as altas frequências.

O voxel (30,7,16) é apresentado como o que tem maior activação para os dados não filtrados quando o atraso é nulo para os métodos de Student de coeficiente de correlação (com ou sem HRF). A sequência não apresenta ruído de baixa frequência (fig. 4.18). A média dos pontos para cada estado de activação fica bem definida (com variância reduzida). Isto faz com que o parâmetro estatístico obtido seja elevado mesmo com uma amplitude relativamente pequena.

Outra razão para esta sequência dar um parâmetro elevado é o seu atraso. Ele é apresentado como voxel de activação máxima para uma extracção dos dados com atraso igual a 1. É então possível extrair informação temporal da activação.

A sequência de (27,6,17) apresenta um ruído de baixa frequência muito pronunciado. Os dados não são filtrados e há valores da sequência pertencentes a um estado que ficam muito próximos de valores pertencentes a outro estado. Por exemplo, o primeiro valor da sequência é praticamente nulo e está muito próximo do último valor. Dito de outra forma, a variância dos estados de repouso e activação é enorme.

A compensação a fazer aqui é filtrar os dados com um filtro passa-alto. Os dados filtrados retiram todos os pequenos atraso. Por isso a desvantagem é a perda de informação sobre a sequência de activações.

O voxel (28,6,17) que aparece por vezes como local de activação máxima para o método de amplitude tem apenas um pouco menos ruído de baixa frequência (fig. 4.19). Este facto é suficiente para compensar o facto de ter um pouco de menos amplitude.

O sinal do voxel (17,8,10) apresenta pouco ruído (fig. 4.20) de baixa frequência tal como o voxel (30,7,16). Tem um pouco de mais amplitude que este mas mesmo assim está longe da amplitude para o voxel (27,6,17).

Aquilo que o permite um resultado estatístico maior é estar atrasado de 2 TR. Quando os dados são retirados com este atraso é ele que melhor se adapta à sequência de modelo.

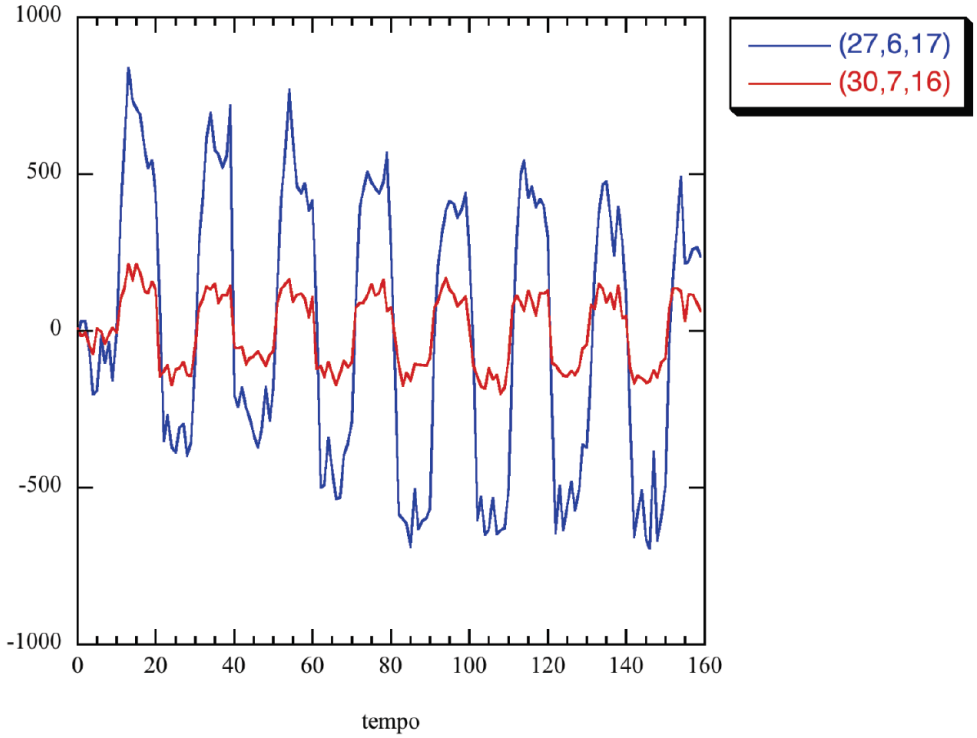


Figura 4.18: Comparação das sequências dos voxels (27,6,17) e (30,7,16).

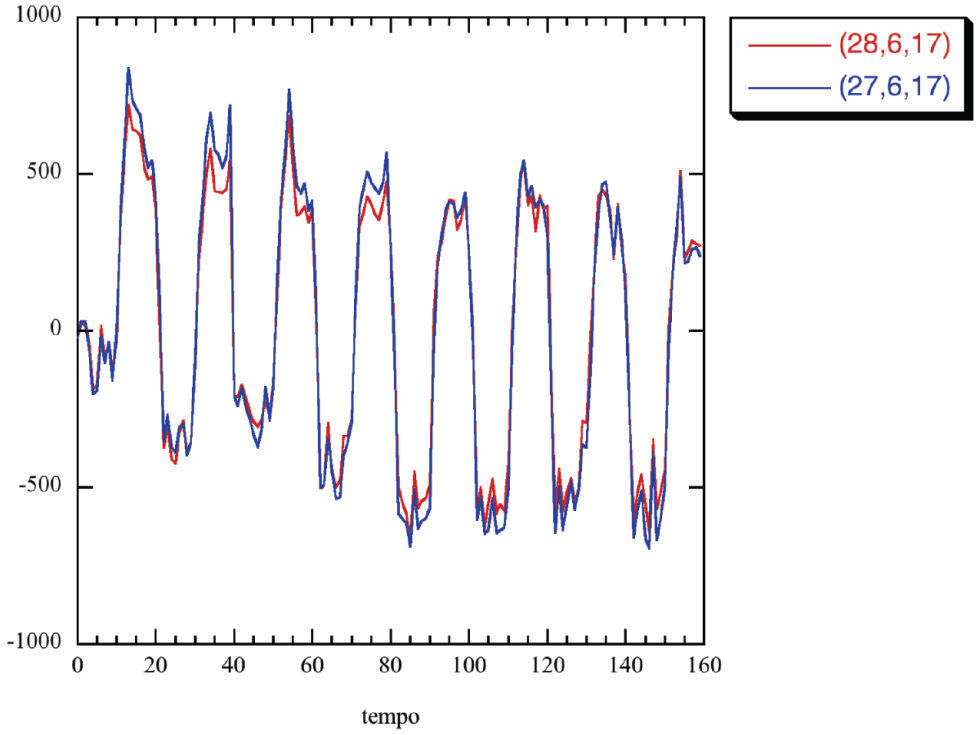


Figura 4.19: Comparação das sequências dos voxels (28,6,17) e (27,6,17).

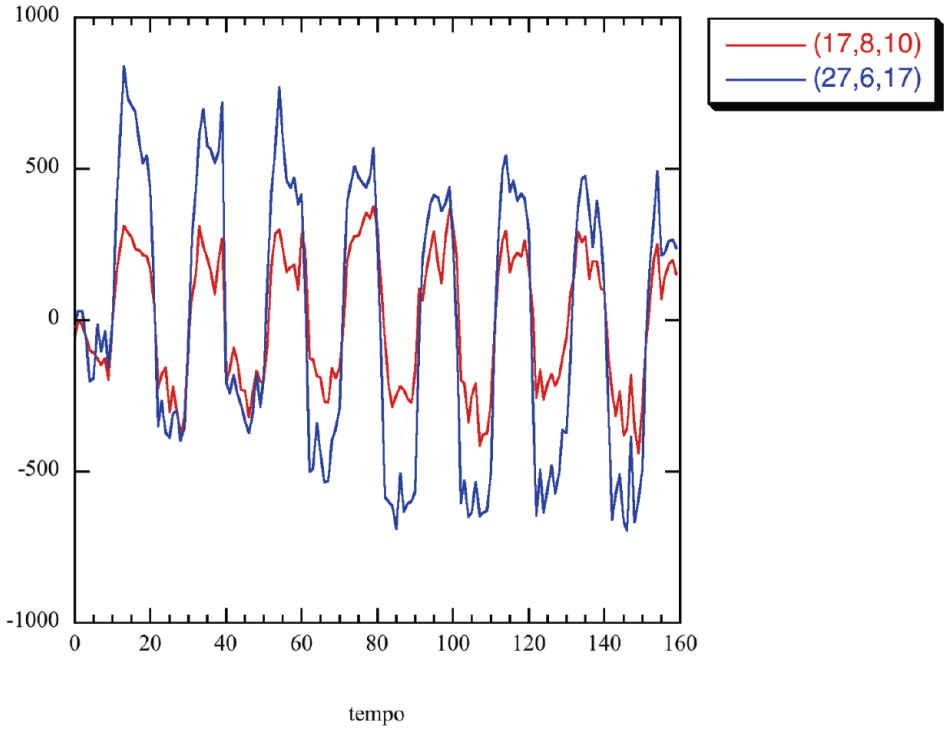


Figura 4.20: Comparação das sequências dos voxels (17,8,10) e (27,6,17).

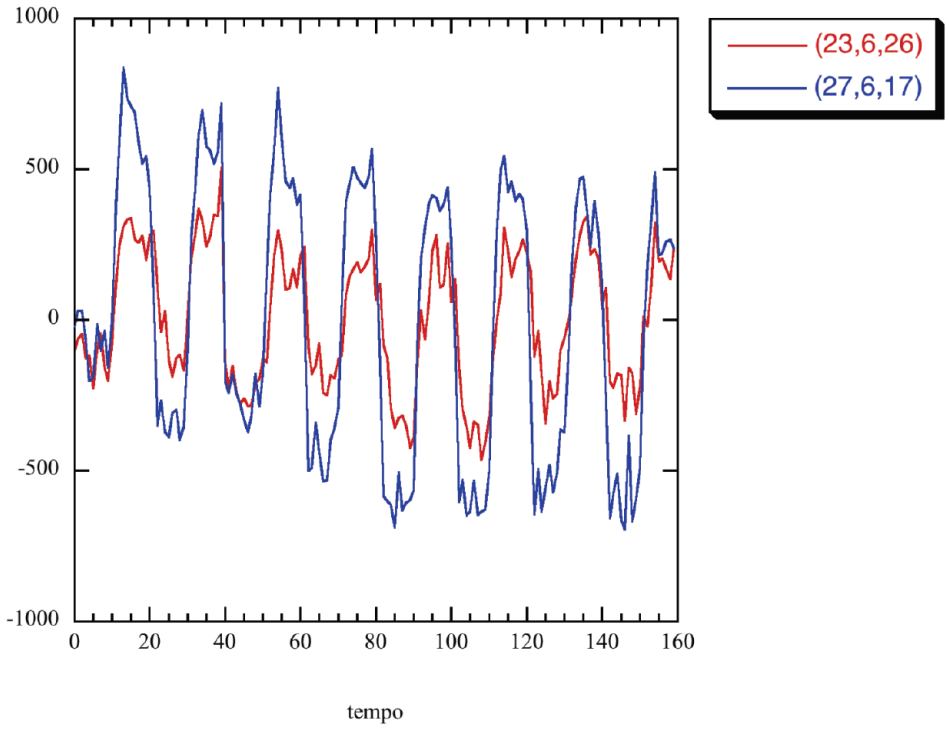


Figura 4.21: Comparação das sequências dos voxels (23,6,26) e (27,6,17).

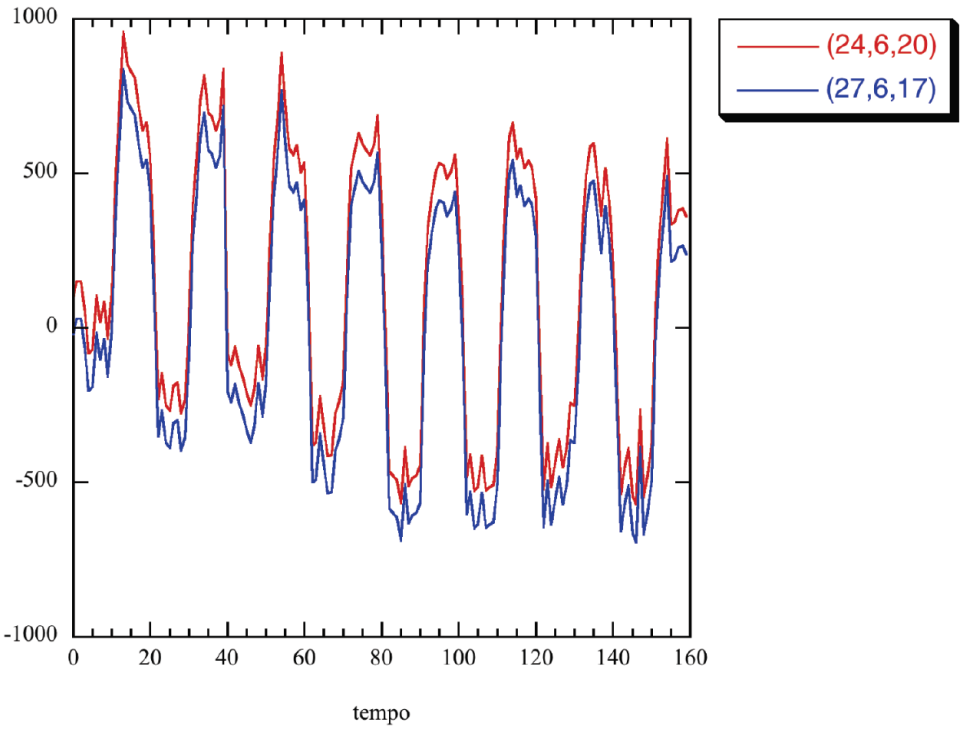


Figura 4.22: Comparação das sequências dos voxels (24,6,20) e (27,6,17).

Finalmente se considerarmos que há um atraso de 2 TR e uma sequência de modelo com HRF o voxel de activação máxima é o (24,6,20). De facto até a sua amplitude é comparável à do voxel (27,6,17) (ver fig. 4.22).

Em resumo, os métodos de Student e de coeficiente de correlação são sensíveis ao atraso dos sinais. Neste estudo dizem-nos que há uma sequência temporal de activações que começa na fatia 16, mantém-se em 16 durante 1 TR, passa para a fatia 10 (ou 20 se considerarmos o HRF) passados 2 TR e fica na fatia 26 3 TR's depois. As activações passam-se no lobo occipital tal como esperado ([17]).

Os restantes métodos revelam-se pouco sensíveis à filtragem dos dados, à introdução de uma sequência de modelo com HRF e aos atrasos de extracção das sequências.

O método que parece ser o mais consistente é o de picos de Fourier. Todos os parâmetros de avaliação (máximo, mínimo e volume têm muito pouca oscilação. Este é seguido pelos métodos de modelo linear generalizado, amplitude e de sobreposição de ordem zero.

No entanto, com dados clínicos de baixa resolução e muito ruído muitas vezes os métodos de Student e de coeficiente de correlação dão resultados surpreendentes em termos de extremos.

4.8.3 Perfis

Para comparar os métodos de activação, o *Cérebro* permite exportar quaisquer perfis ortogonais de probabilidade. apresentam-se em seguida os perfis do voxel (27,6,17) para todas as situações apresentadas anteriormente (4.8.1). Cada conjunto de perfis segundo uma direcção está dividido em dois gráficos para que sejam legíveis.

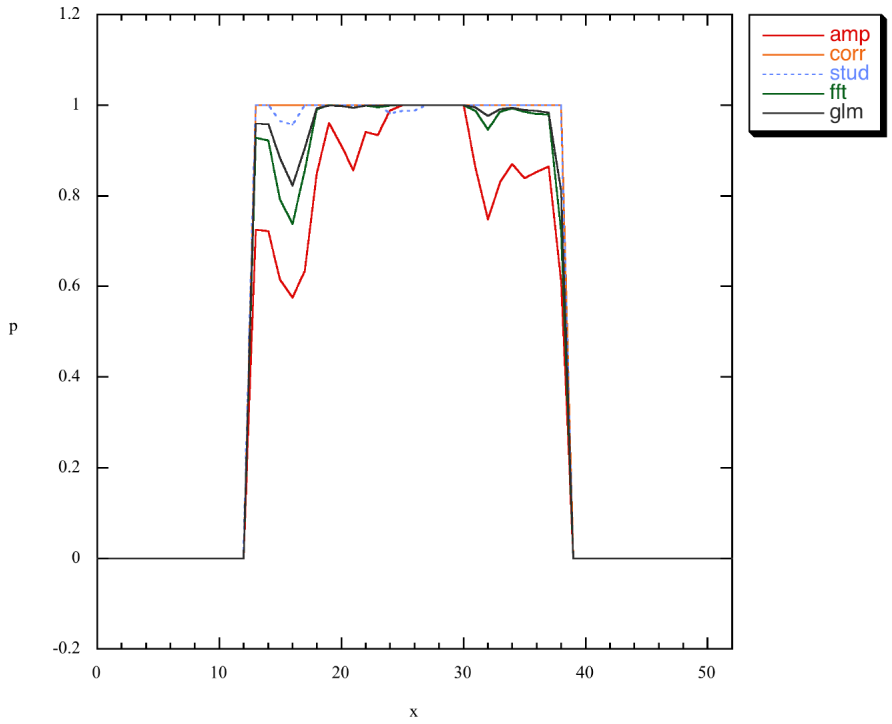


Figura 4.23: Perfil segundo x para os métodos de amplitude, correlação, TRF e MLG (dados filtrados sem HRF).

4.8.3.1 Filtrados sem HRF

Apresentam-se aqui os perfis de probabilidade para dados filtrados, analisados com uma sequência de modelo em bloco segundo as três direcções ortogonais (figuras 4.23, 4.24, 4.25, 4.26, 4.27 e 4.28). Para que as linhas dos vários métodos sejam discerníveis, os métodos são separados em dois gráficos por direcção.

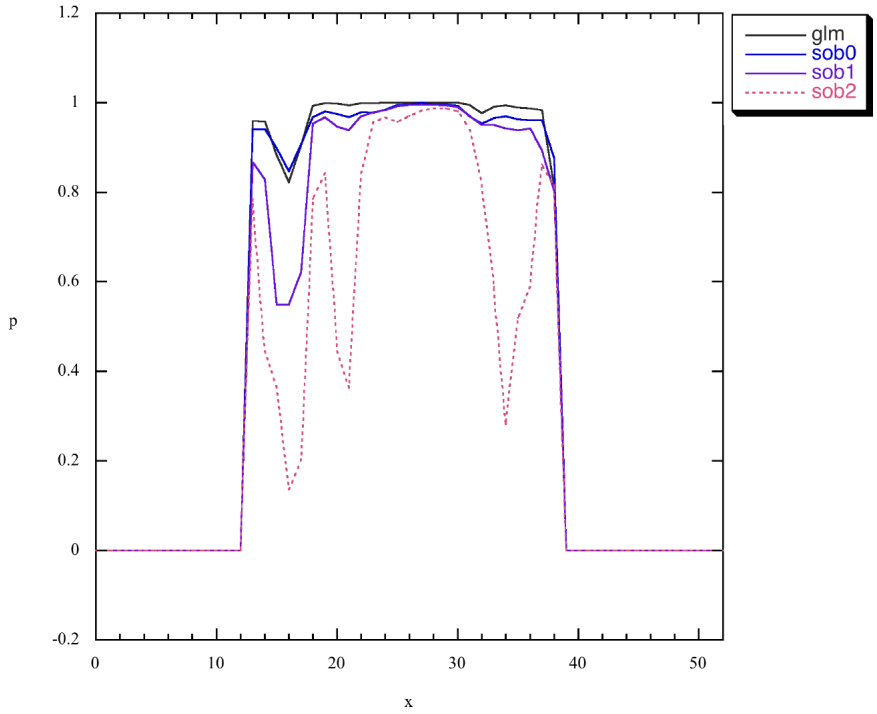


Figura 4.24: Perfil segundo x para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados filtrados sem HRF).

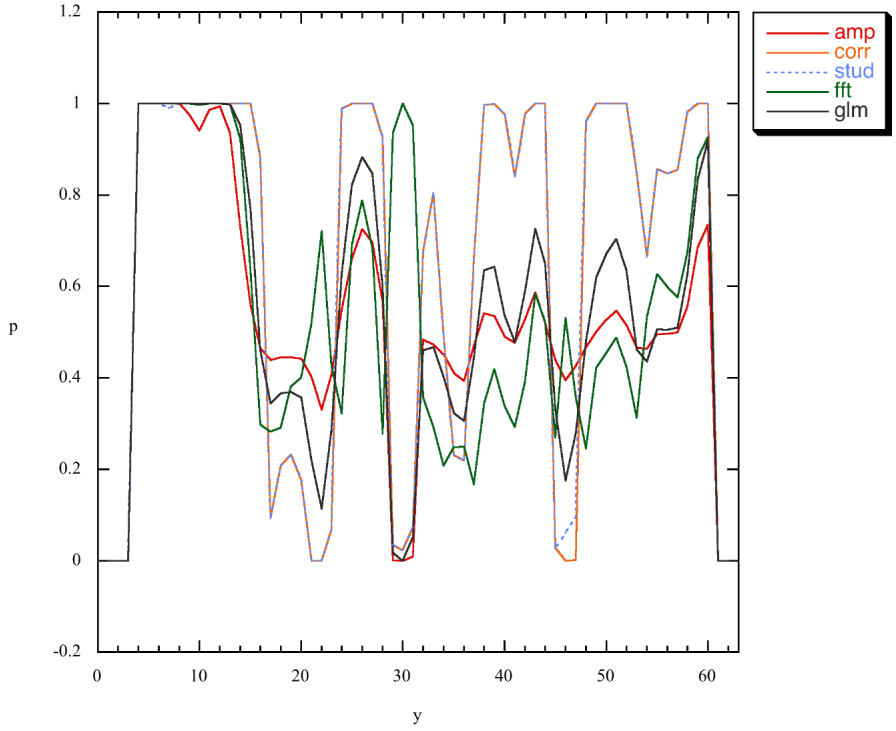


Figura 4.25: Perfil segundo y para os métodos de amplitude, correlação, TRF e MLG (dados filtrados sem HRF).

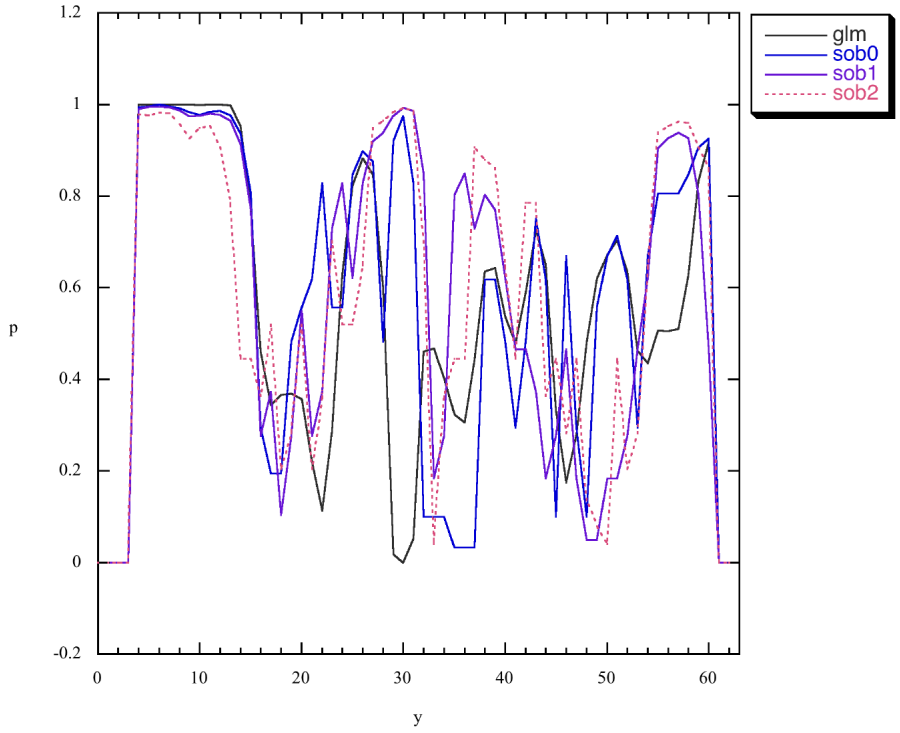


Figura 4.26: Perfil segundo y para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados filtrados sem HRF).

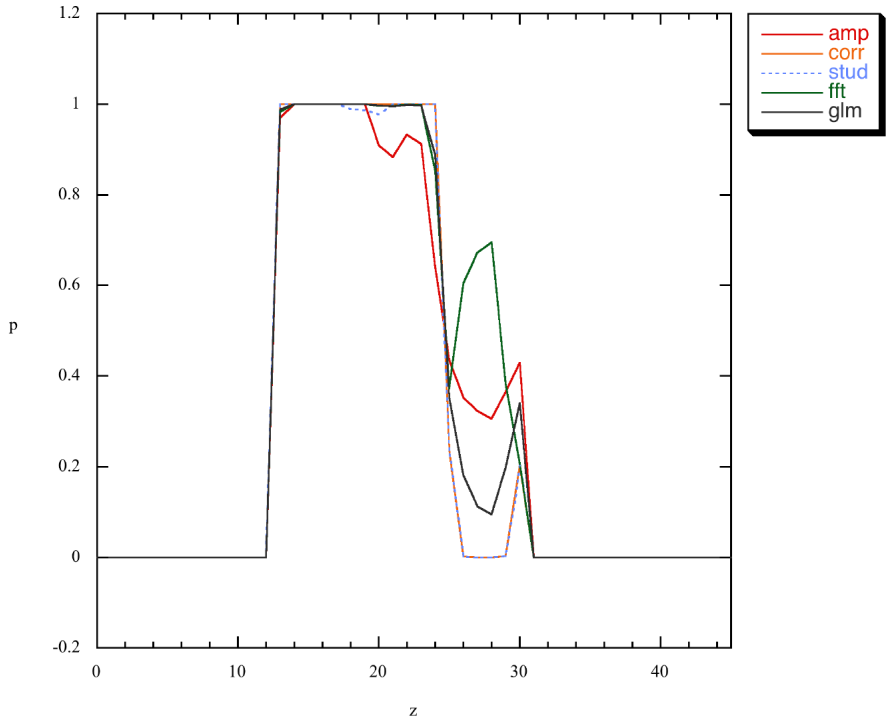


Figura 4.27: Perfil segundo z para os métodos de amplitude, correlação, TRF e MLG (dados filtrados sem HRF).

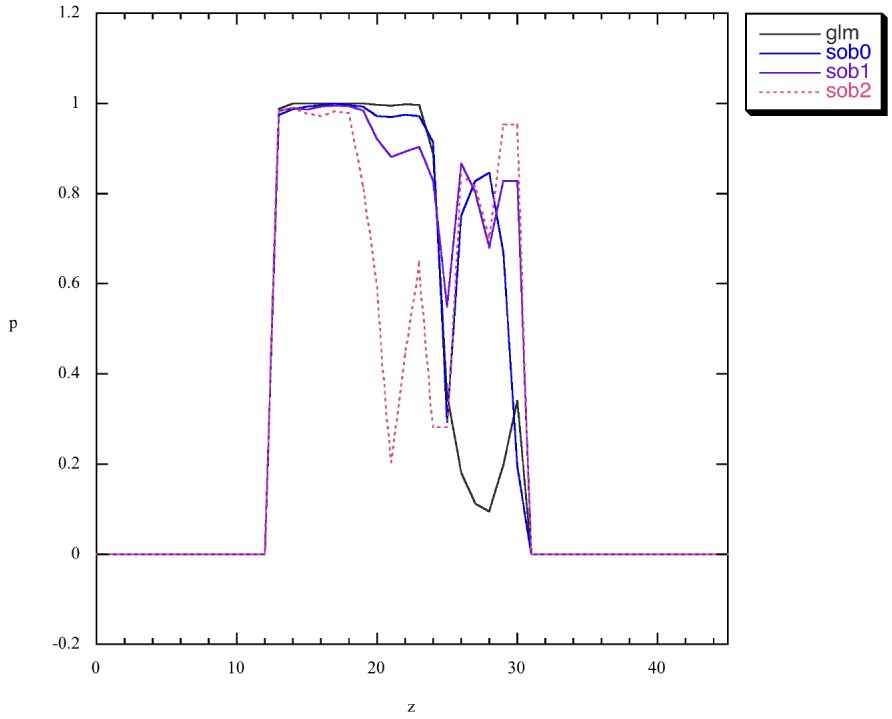


Figura 4.28: Perfil segundo z para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados filtrados sem HRF).

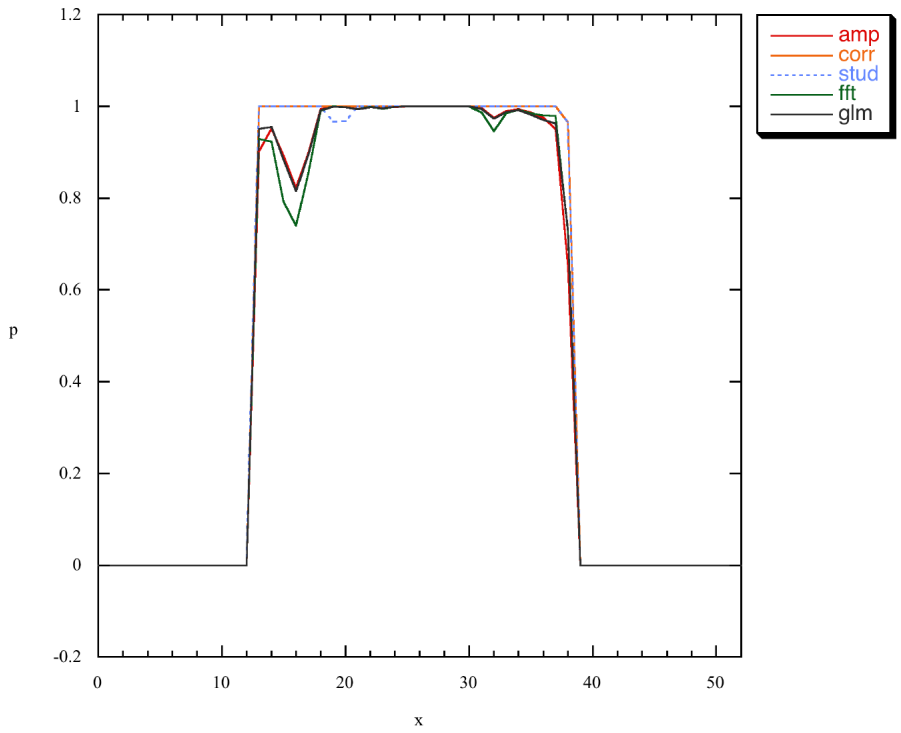


Figura 4.29: Perfil segundo x para os métodos de amplitude, correlação, Student, TRF e MLG (dados não filtrados, sem HRF e com atraso igual a 0).

4.8.3.2 Não filtrados, sem HRF e com atraso igual a 0

Apresentam-se aqui os perfis de probabilidade para dados não filtrados, com atraso nulo e analisados com uma sequência de modelo em bloco segundo as três direcções ortogonais (figuras 4.29, 4.30, 4.31, 4.32, 4.33 e 4.34). Para que as linhas dos vários métodos sejam discerníveis, os métodos são separados em dois gráficos por direcção.

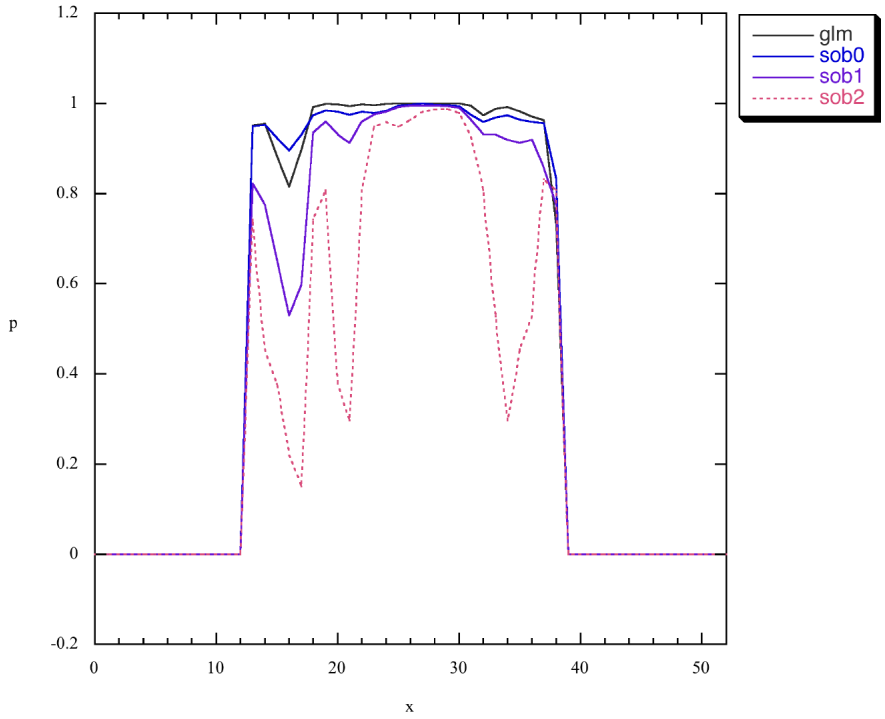


Figura 4.30: Perfil segundo x para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados não filtrados, sem HRF e com atraso igual a 0).

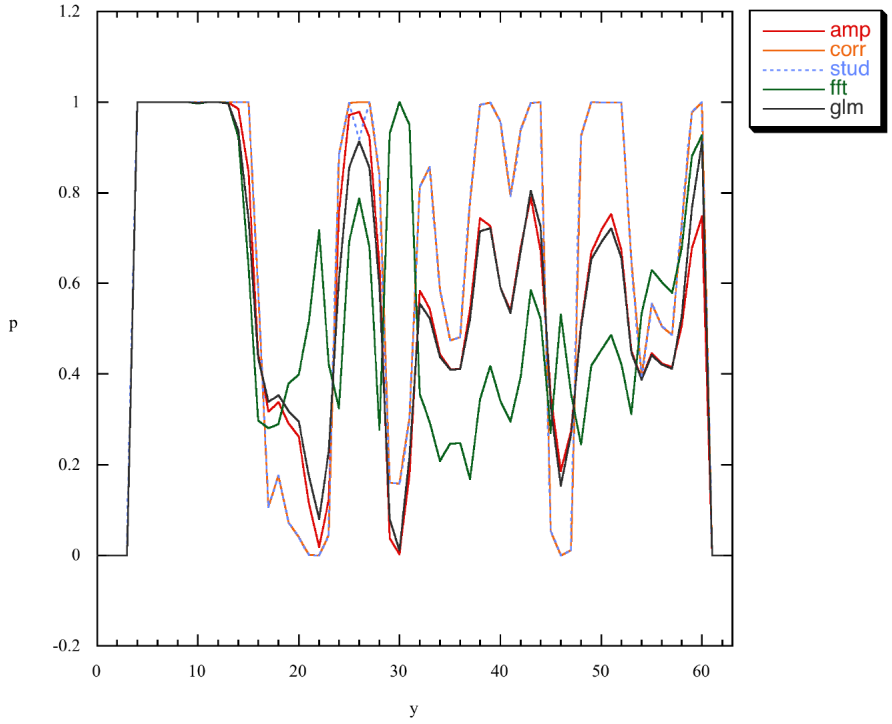


Figura 4.31: Perfil segundo y para os métodos de amplitude, correlação, Student, TRF e MLG (dados não filtrados, sem HRF e com atraso igual a 0).

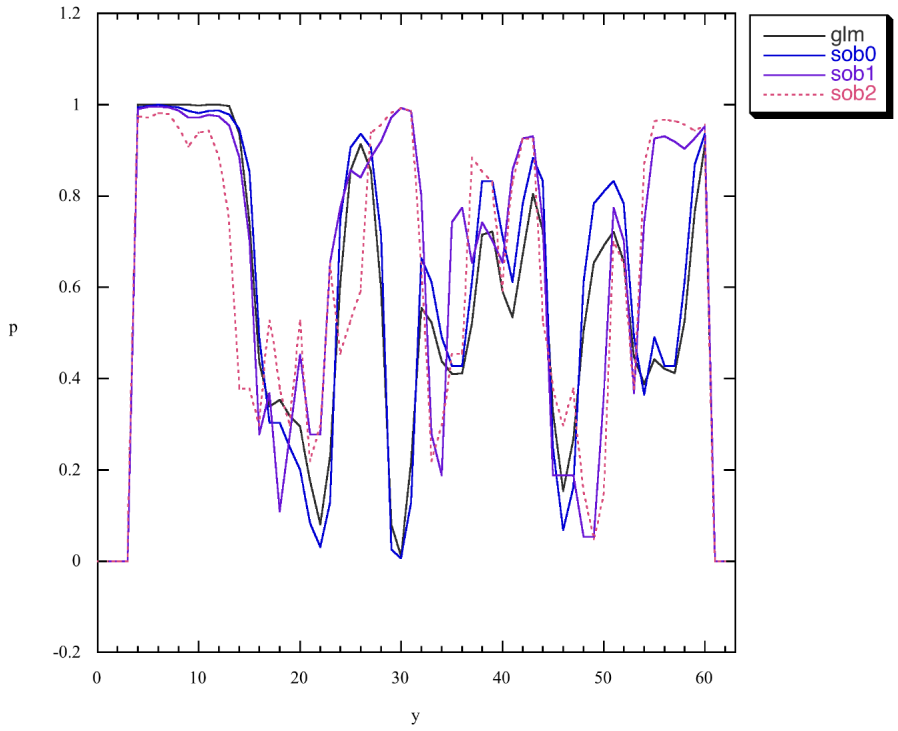


Figura 4.32: Perfil segundo y para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados não filtrados, sem HRF e com atraso igual a 0).

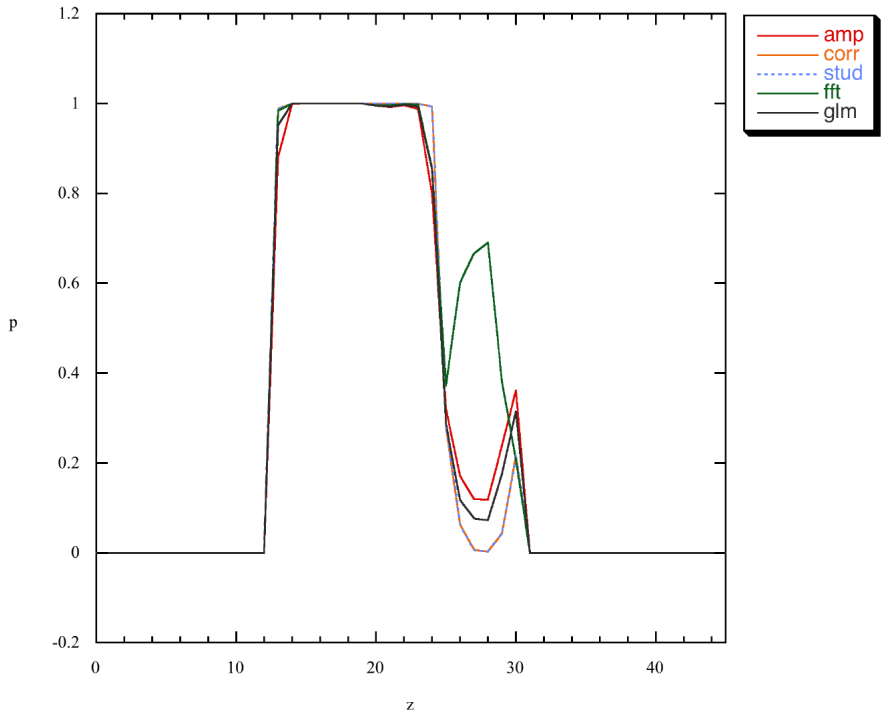


Figura 4.33: Perfil segundo z para os métodos de amplitude, correlação, Student, TRF e MLG (dados não filtrados, sem HRF e com atraso igual a 0).

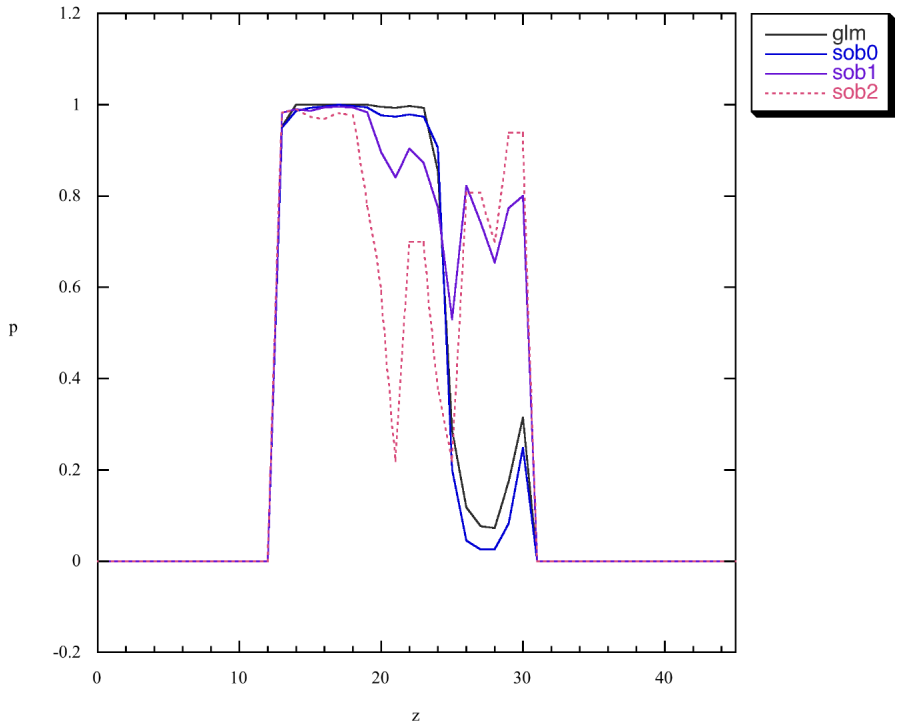


Figura 4.34: Perfil segundo z para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados não filtrados, sem HRF e com atraso igual a 0).

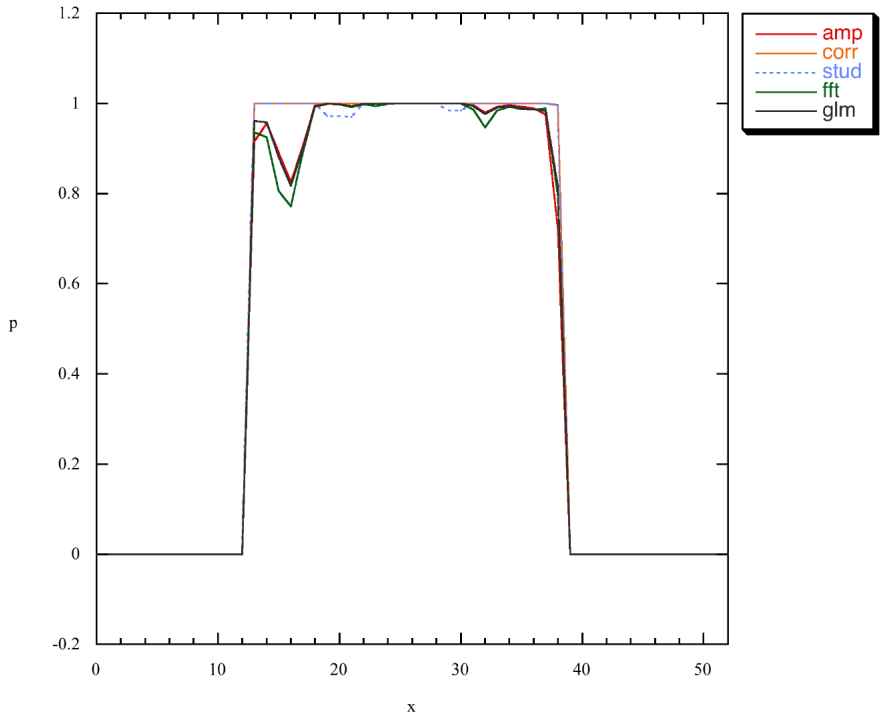


Figura 4.35: Perfil segundo x para os métodos de amplitude, correlação, Student, TRF e MLG (dados não filtrados, sem HRF e com atraso igual a 1).

4.8.3.3 Não filtrados, sem HRF e com atraso igual a 1

Apresentam-se aqui os perfis de probabilidade para dados não filtrados, com atraso unitário e analisados com uma sequência de modelo em bloco segundo as três direcções ortogonais (figuras 4.35, 4.36, 4.37, 4.38, 4.39 e 4.40). Para que as linhas dos vários métodos sejam discerníveis, os métodos são separados em dois gráficos por direcção.

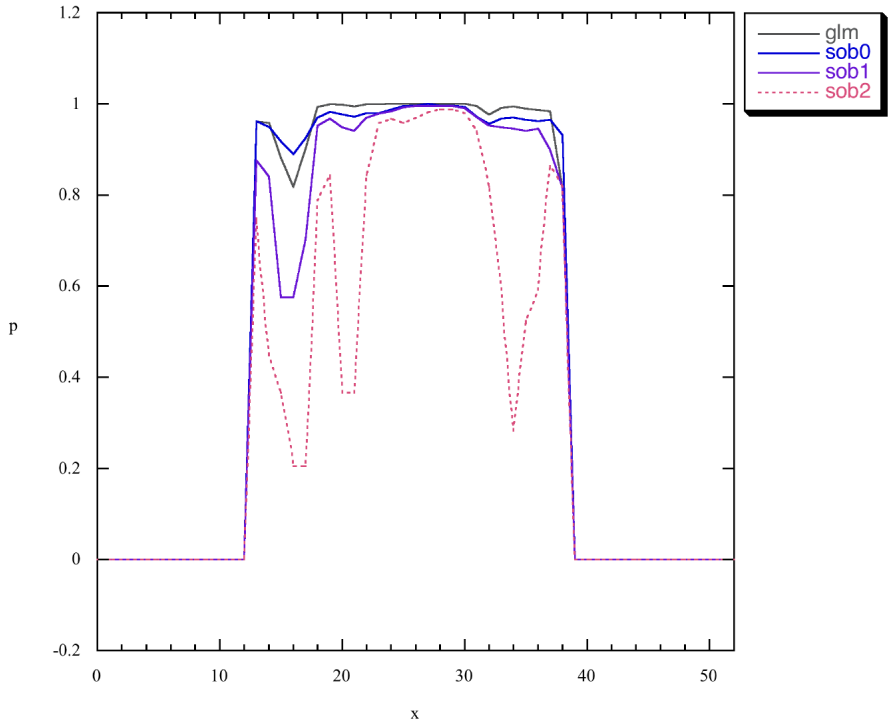


Figura 4.36: Perfil segundo x para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados não filtrados, sem HRF e com atraso igual a 1).

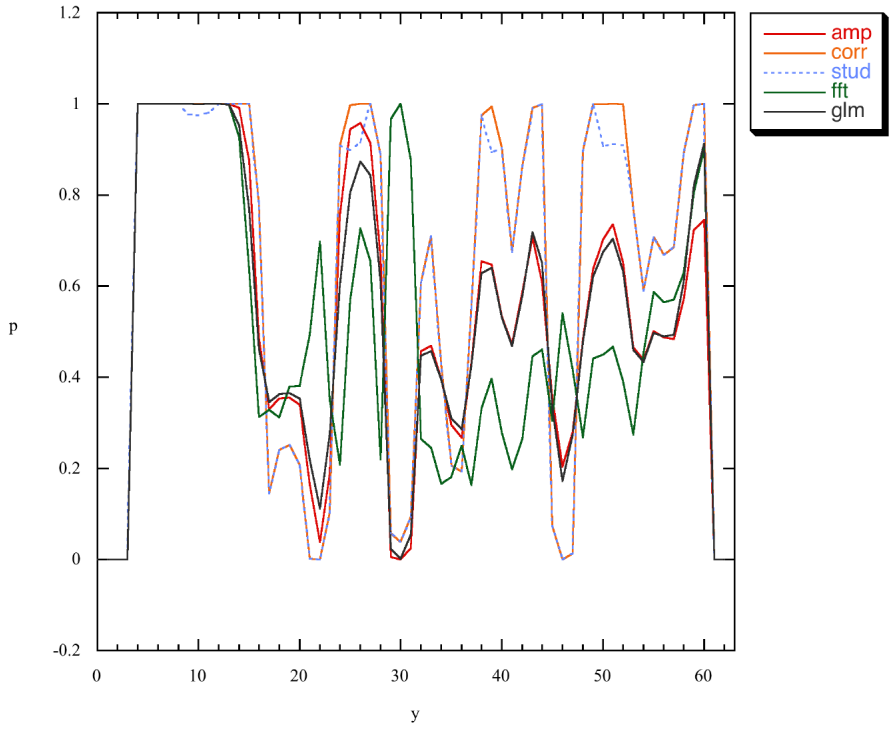


Figura 4.37: Perfil segundo y para os métodos de amplitude, correlação, Student, TRF e MLG (dados não filtrados, sem HRF e com atraso igual a 1).

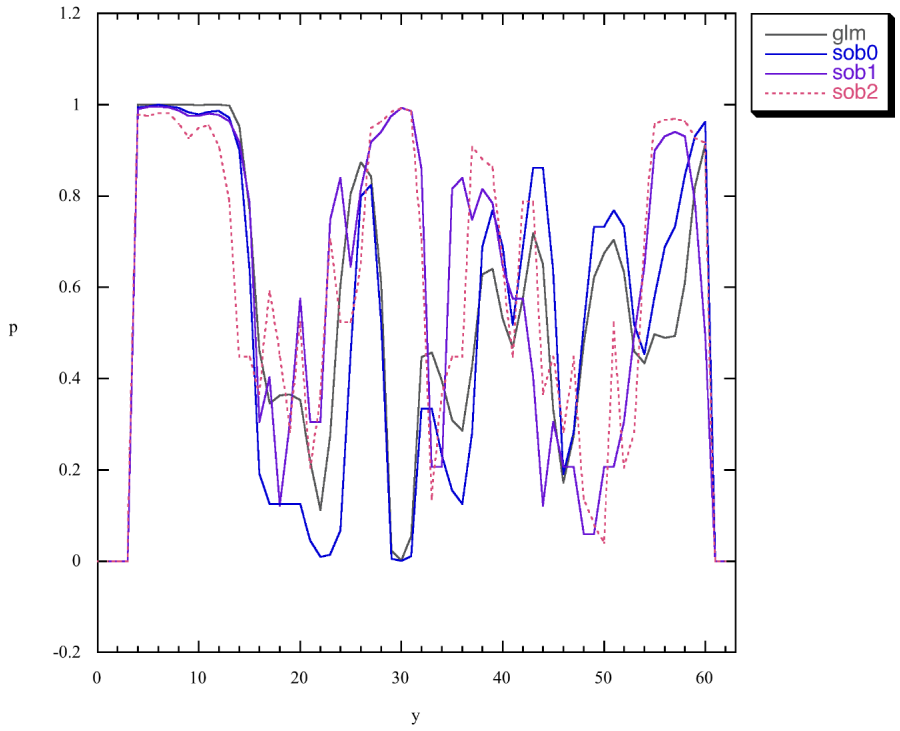


Figura 4.38: Perfil segundo y para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados não filtrados, sem HRF e com atraso igual a 1).

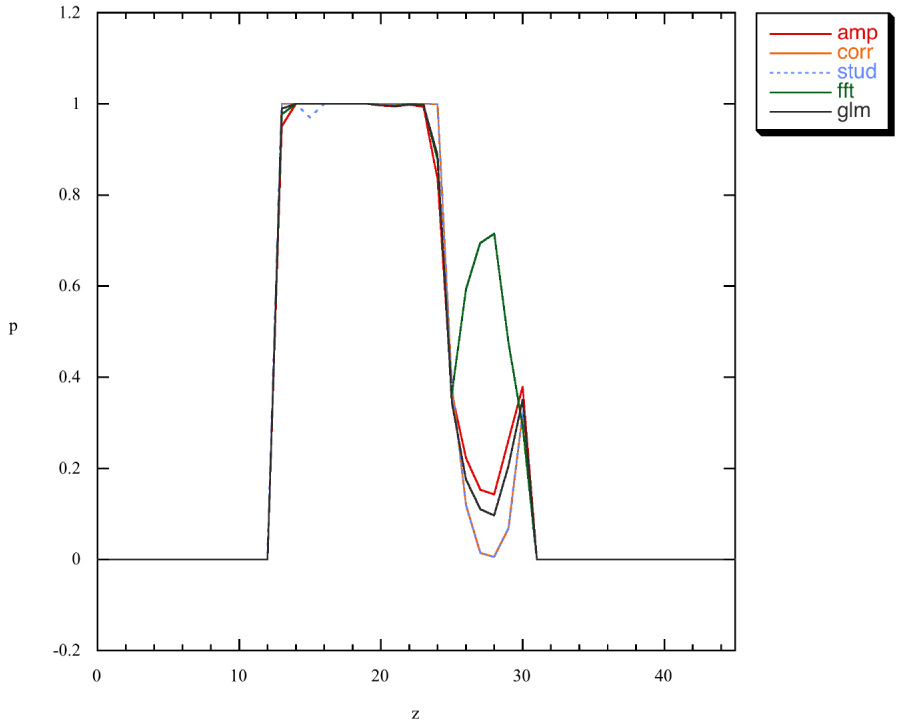


Figura 4.39: Perfil segundo z para os métodos de amplitude, correlação, Student, TRF e MLG (dados não filtrados, sem HRF e com atraso igual a 1).

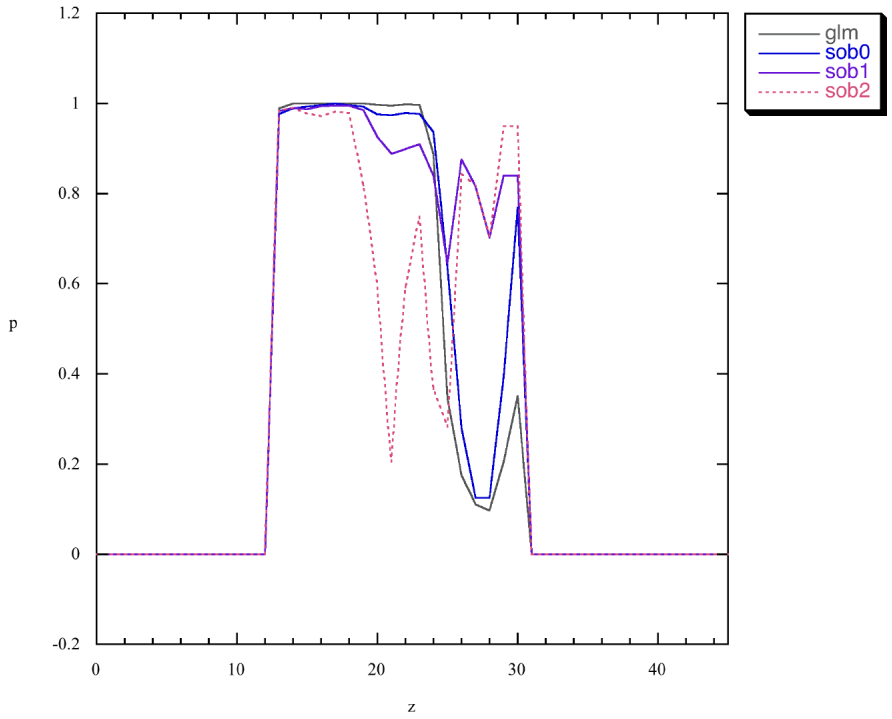


Figura 4.40: Perfil segundo z para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados não filtrados, sem HRF e com atraso igual a 1).

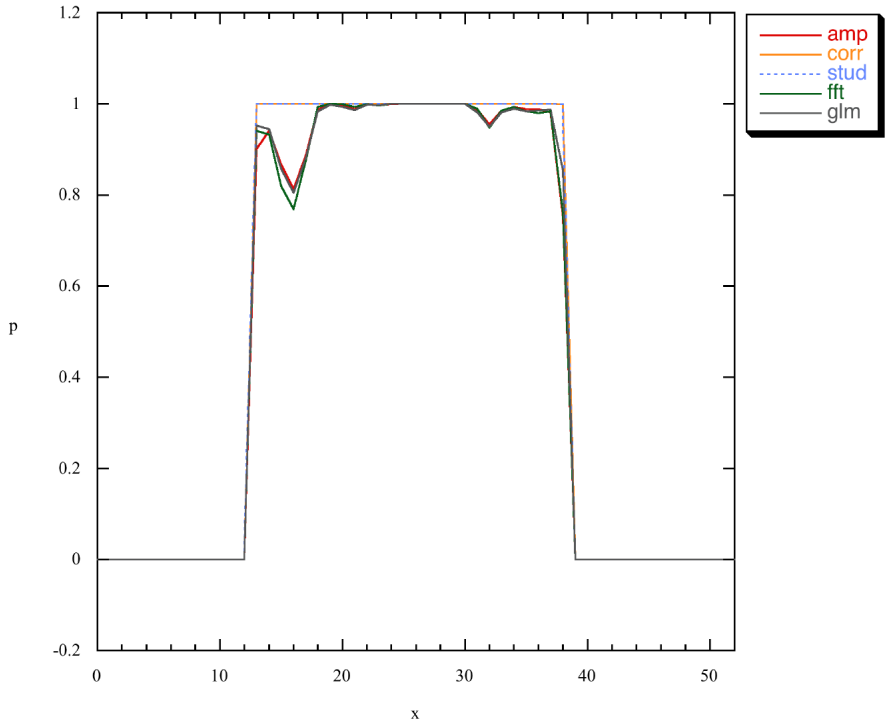


Figura 4.41: Perfil segundo x para os métodos de amplitude, correlação, Student, TRF e MLG (dados não filtrados, sem HRF e com atraso igual a 2).

4.8.3.4 Não filtrados, sem HRF e com atraso igual a 2

Apresentam-se aqui os perfis de probabilidade para dados não filtrados, com atraso de dois e analisados com uma sequência de modelo em bloco segundo as três direcções ortogonais (figuras 4.41, 4.42, 4.43, 4.44, 4.45 e 4.46). Para que as linhas dos vários métodos sejam discerníveis, os métodos são separados em dois gráficos por direcção.

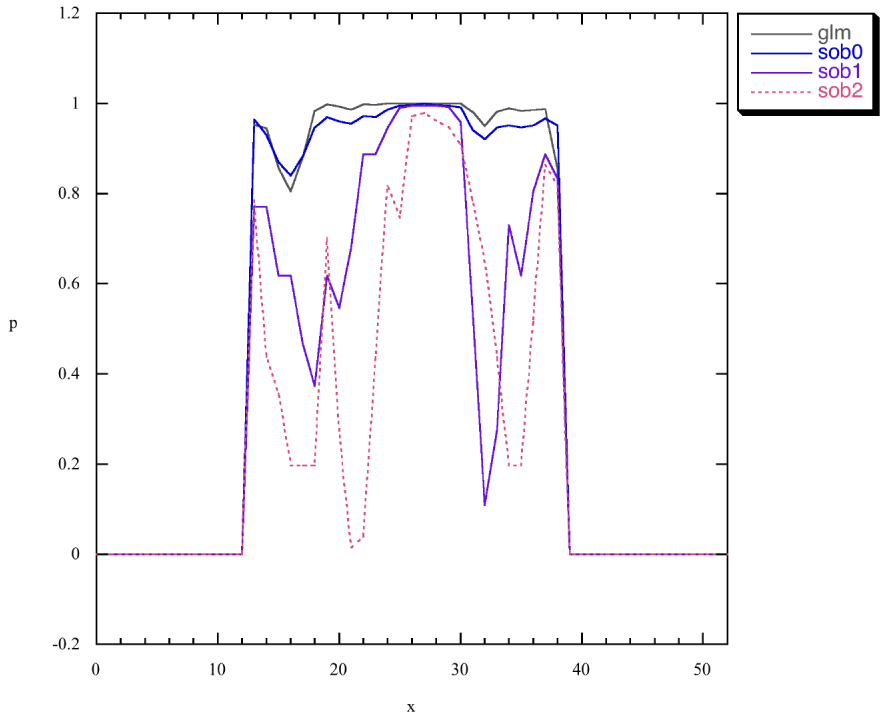


Figura 4.42: Perfil segundo x para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados não filtrados, sem HRF e com atraso igual a 2).

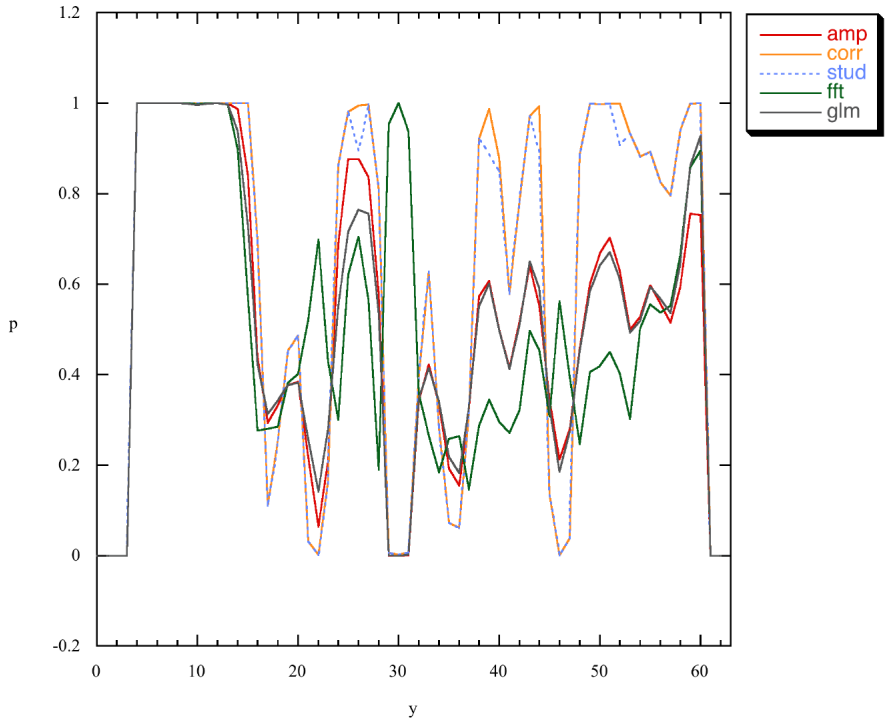


Figura 4.43: Perfil segundo y para os métodos de amplitude, correlação, Student, TRF e MLG (dados não filtrados, sem HRF e com atraso igual a 2).

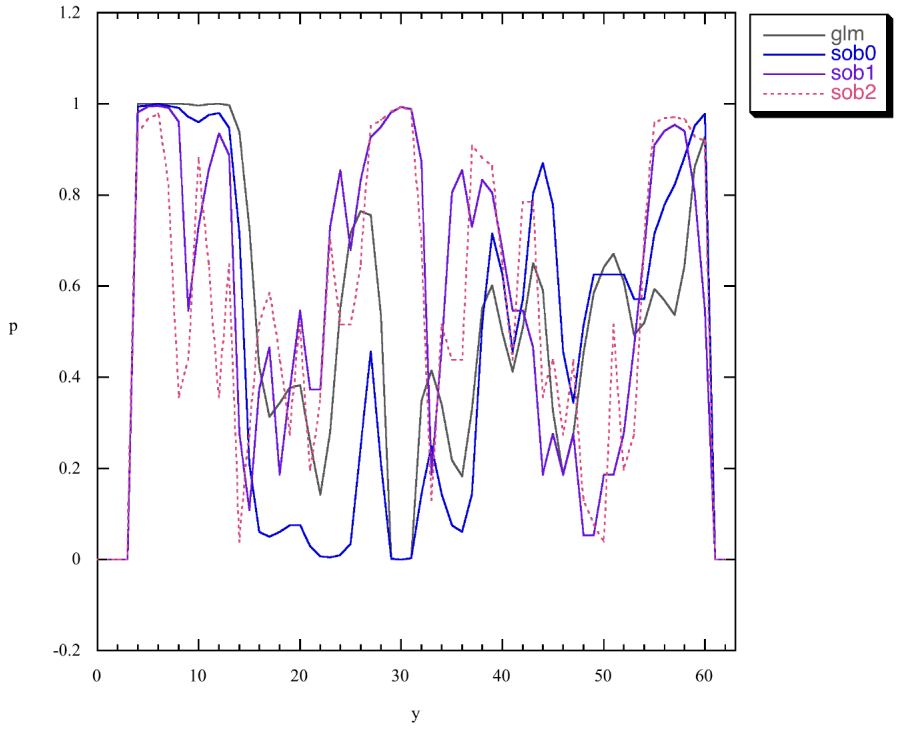


Figura 4.44: Perfil segundo y para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados não filtrados, sem HRF e com atraso igual a 2).

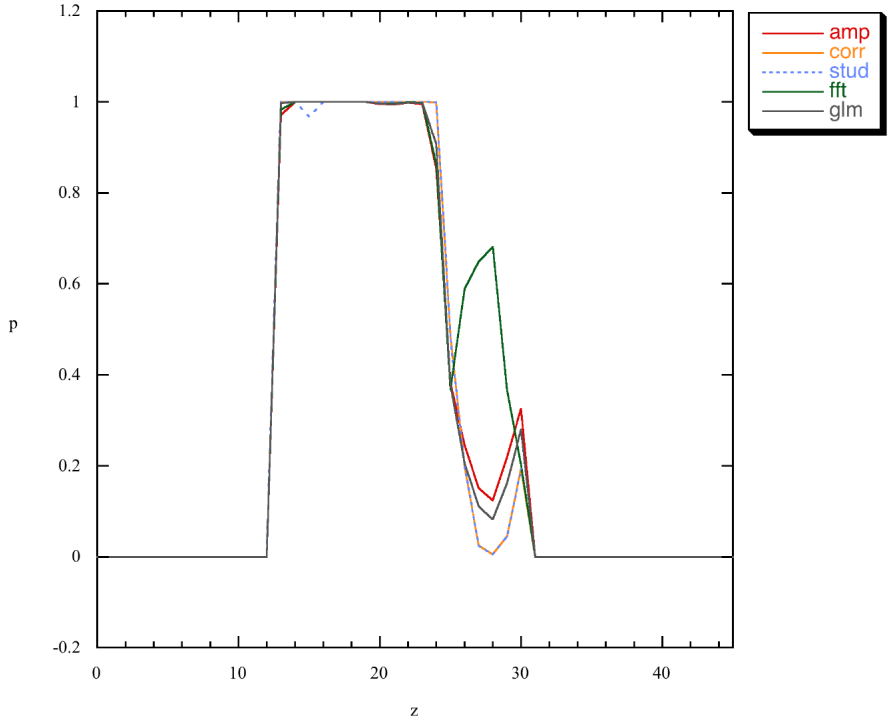


Figura 4.45: Perfil segundo z para os métodos de amplitude, correlação, Student, TRF e MLG (dados não filtrados, sem HRF e com atraso igual a 2).

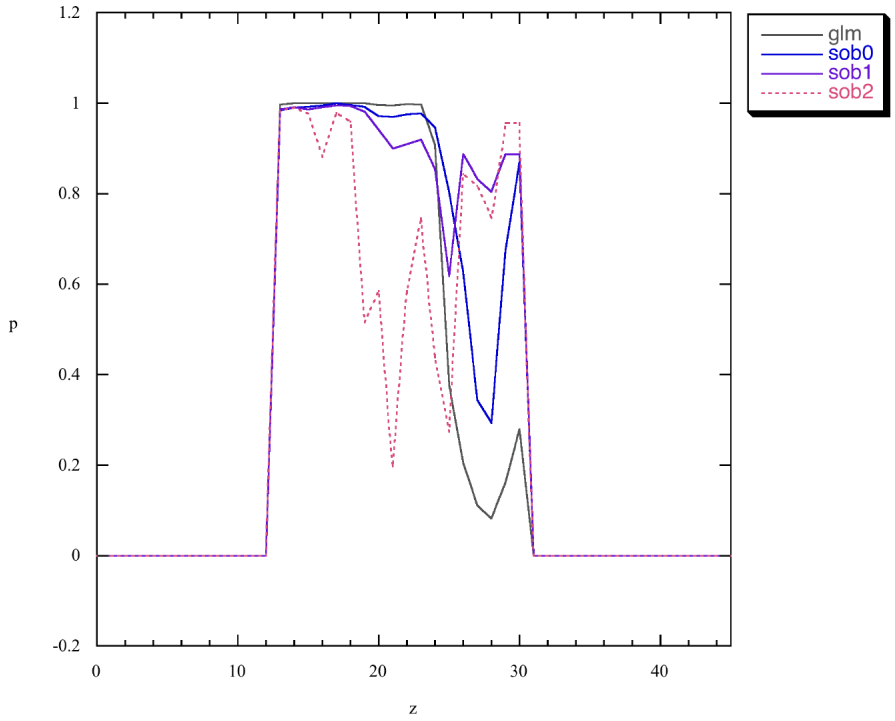


Figura 4.46: Perfil segundo z para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados não filtrados, sem HRF e com atraso igual a 2).

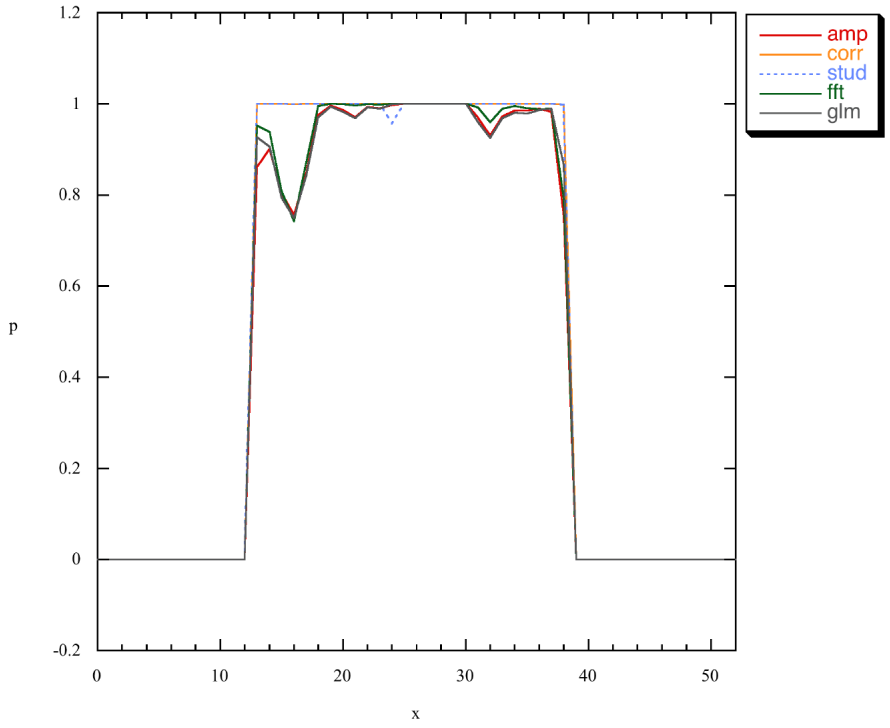


Figura 4.47: Perfil segundo x para os métodos de amplitude, correlação, Student, TRF e MLG (dados não filtrados, sem HRF e com atraso igual a 3).

4.8.3.5 Não filtrados, sem HRF e com atraso igual a 3

Apresentam-se aqui os perfis de probabilidade para dados não filtrados, com atraso de três e analisados com uma sequência de modelo em bloco segundo as três direcções ortogonais (figuras 4.47, 4.48, 4.49, 4.50, 4.51 e 4.52). Para que as linhas dos vários métodos sejam discerníveis, os métodos são separados em dois gráficos por direcção.

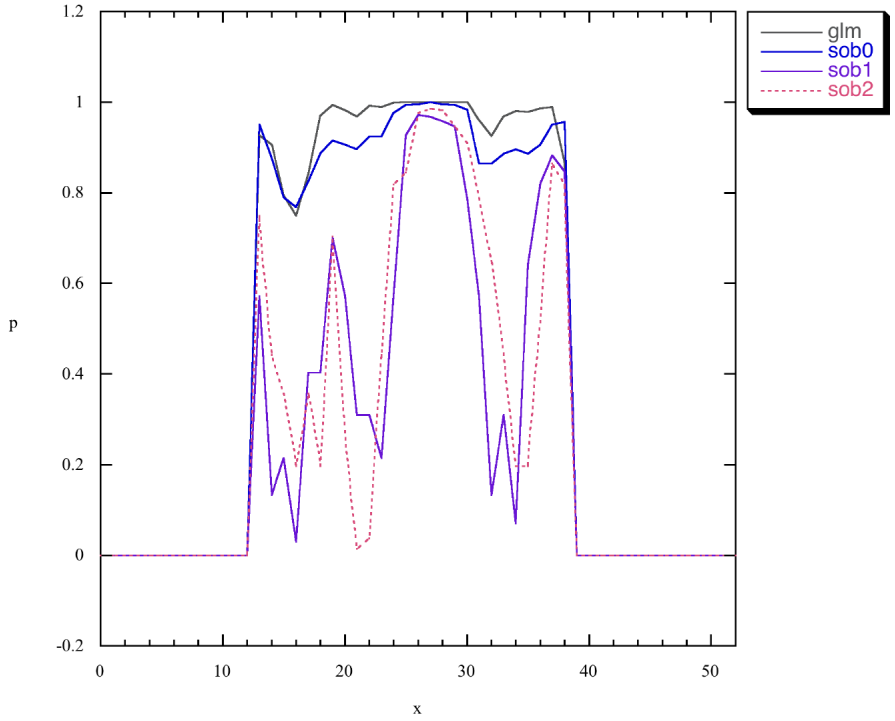


Figura 4.48: Perfil segundo x para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados não filtrados, sem HRF e com atraso igual a 3).

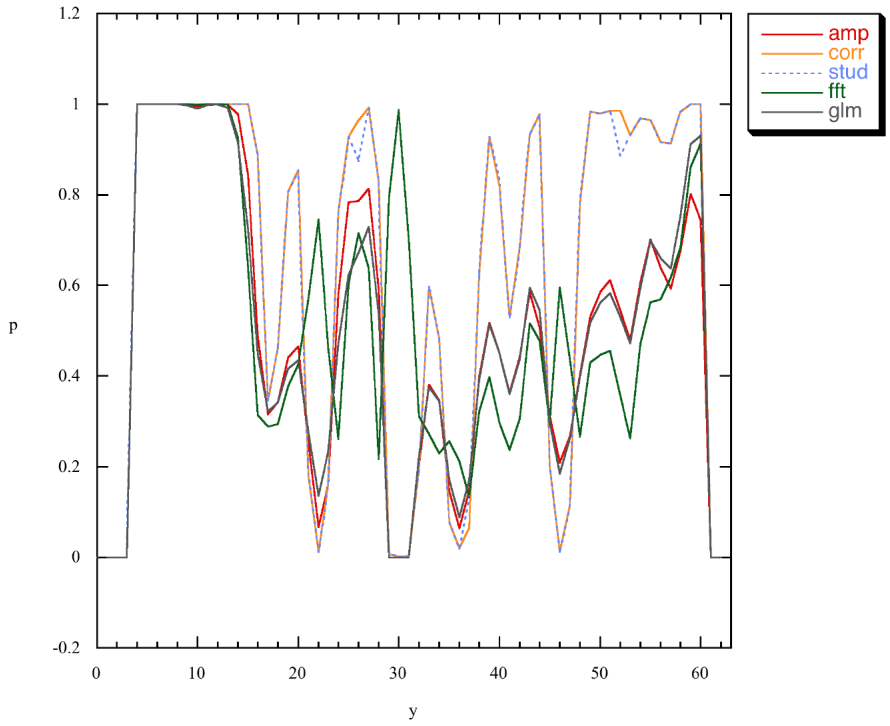


Figura 4.49: Perfil segundo y para os métodos de amplitude, correlação, Student, TRF e MLG (dados não filtrados, sem HRF e com atraso igual a 3).

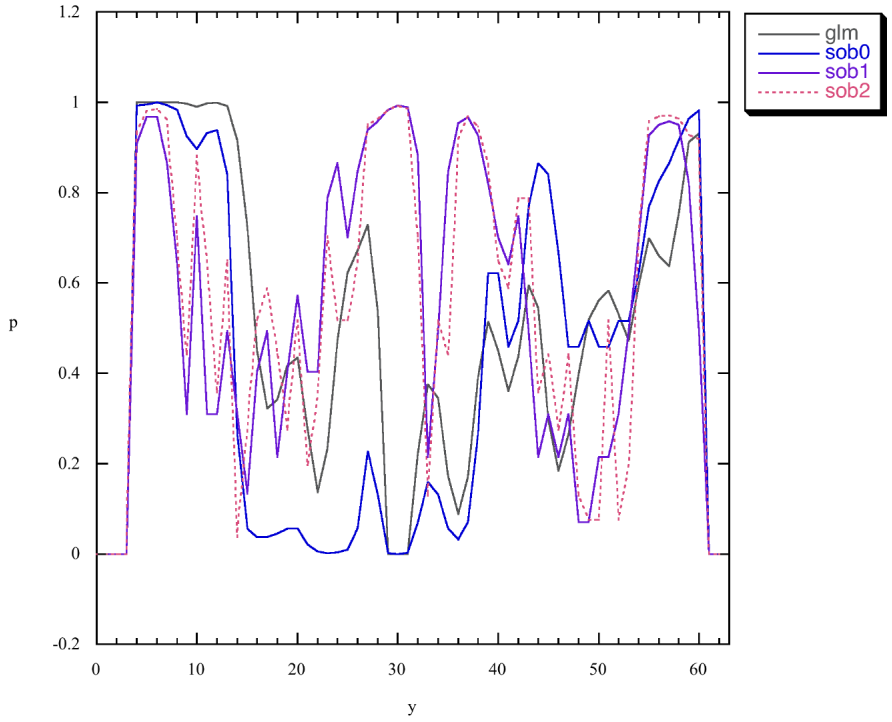


Figura 4.50: Perfil segundo y para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados não filtrados, sem HRF e com atraso igual a 3).

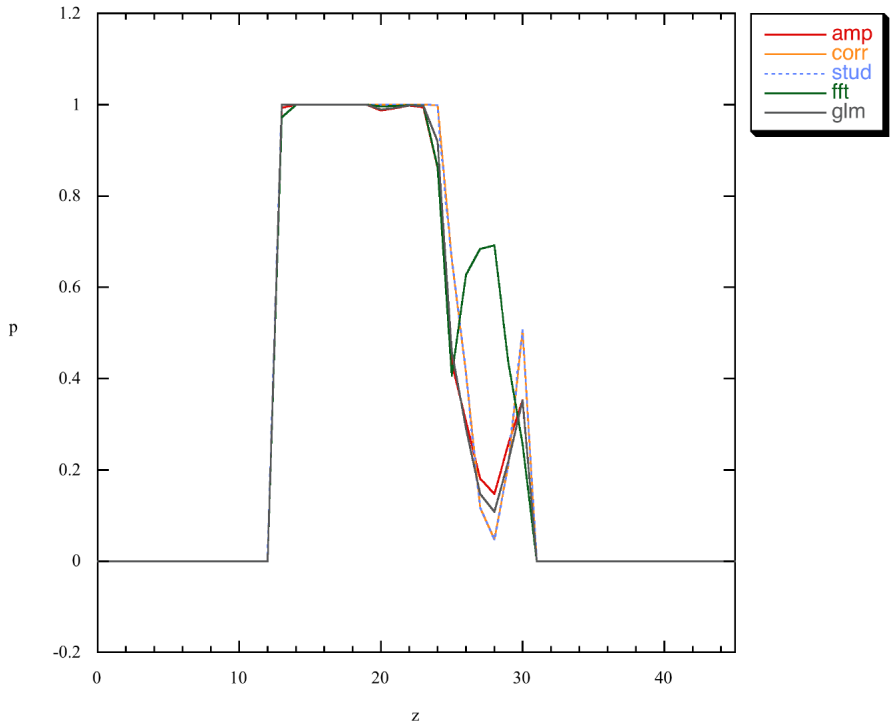


Figura 4.51: Perfil segundo z para os métodos de amplitude, correlação, Student, TRF e MLG (dados não filtrados, sem HRF e com atraso igual a 3).

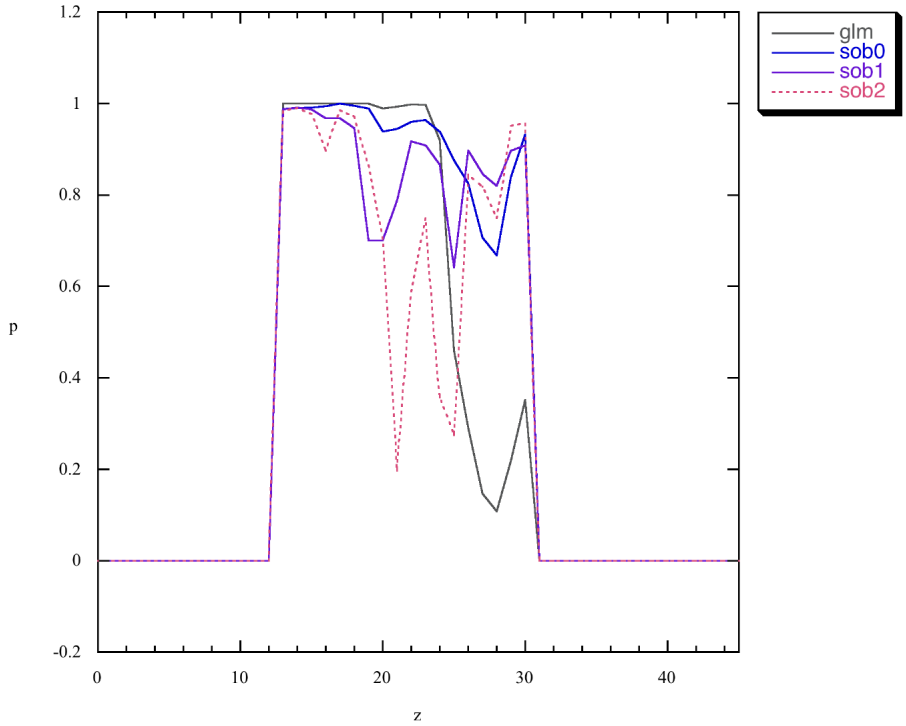


Figura 4.52: Perfil segundo z para os métodos de MLG e sobreposição de ordens 0, 1 e 2 (dados não filtrados, sem HRF e com atraso igual a 3).

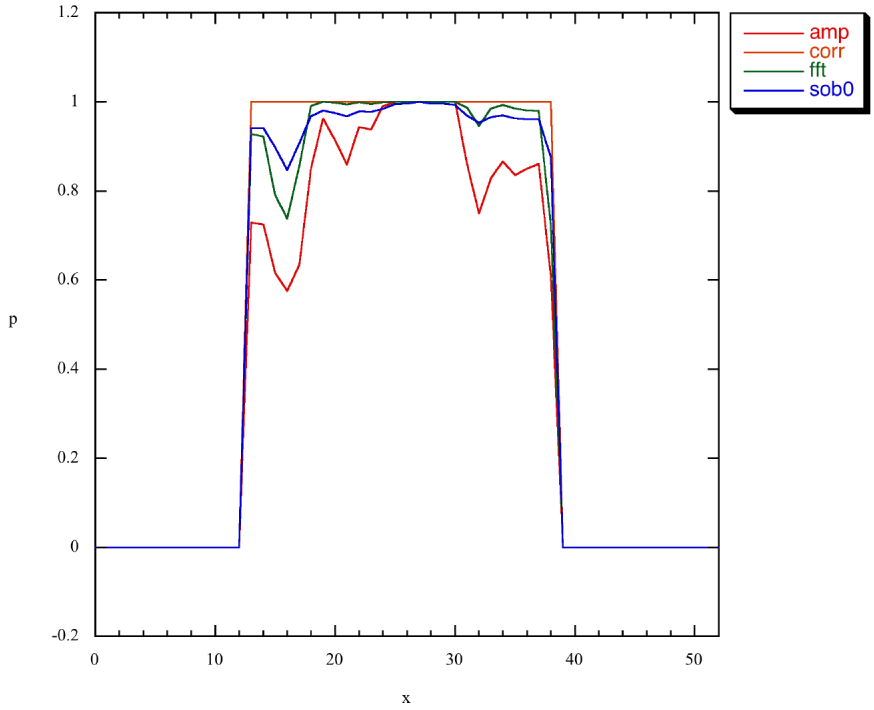


Figura 4.53: Perfil segundo x para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados filtrados com HRF).

4.8.3.6 Filtrados, com HRF

Apresentam-se aqui os perfis de probabilidade para dados filtrados e analisados com uma sequência de modelo com HRF segundo as três direcções ortogonais (figuras 4.53, 4.54, 4.55, 4.56, 4.57 e 4.58). Para que as linhas dos vários métodos sejam discerníveis, os métodos são separados em dois gráficos por direcção.

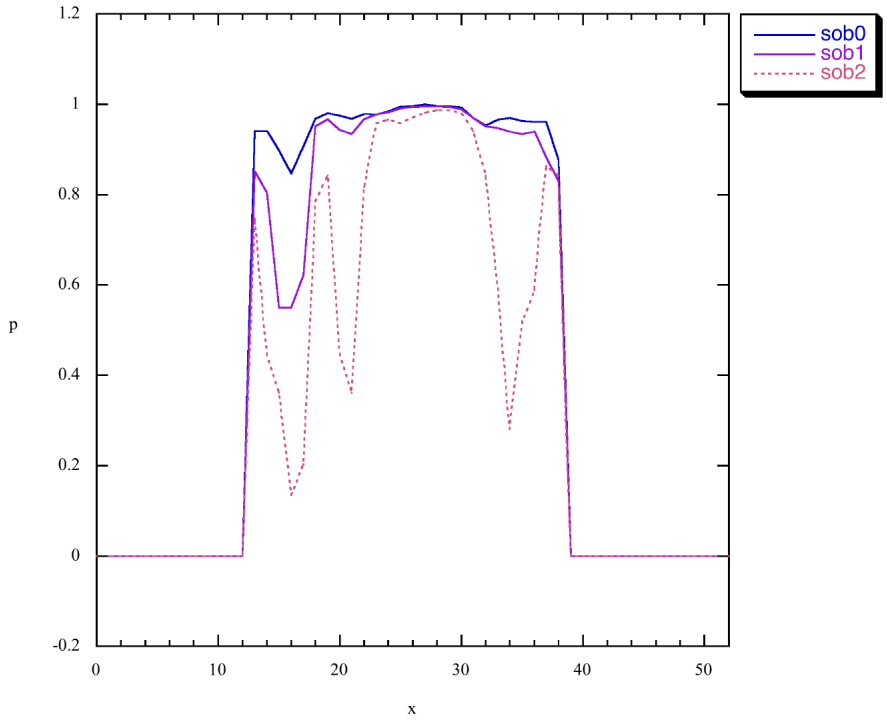


Figura 4.54: Perfil segundo x para os métodos de sobreposição de ordens 0, 1 e 2 (dados filtrados com HRF).

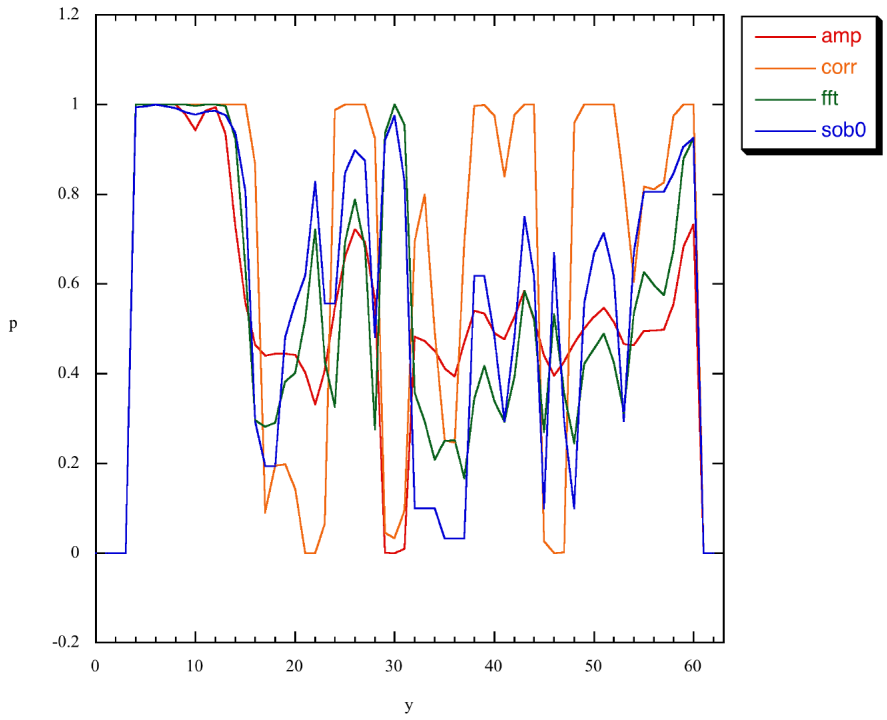


Figura 4.55: Perfil segundo y para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados filtrados com HRF).

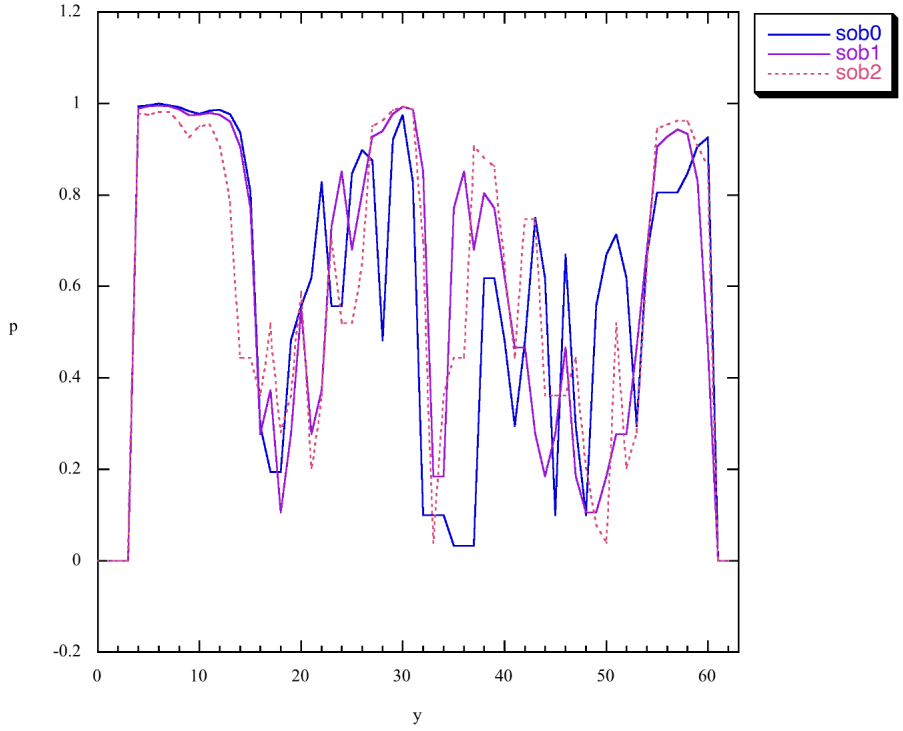


Figura 4.56: Perfil segundo y para os métodos de sobreposição de ordens 0, 1 e 2 (dados filtrados com HRF).

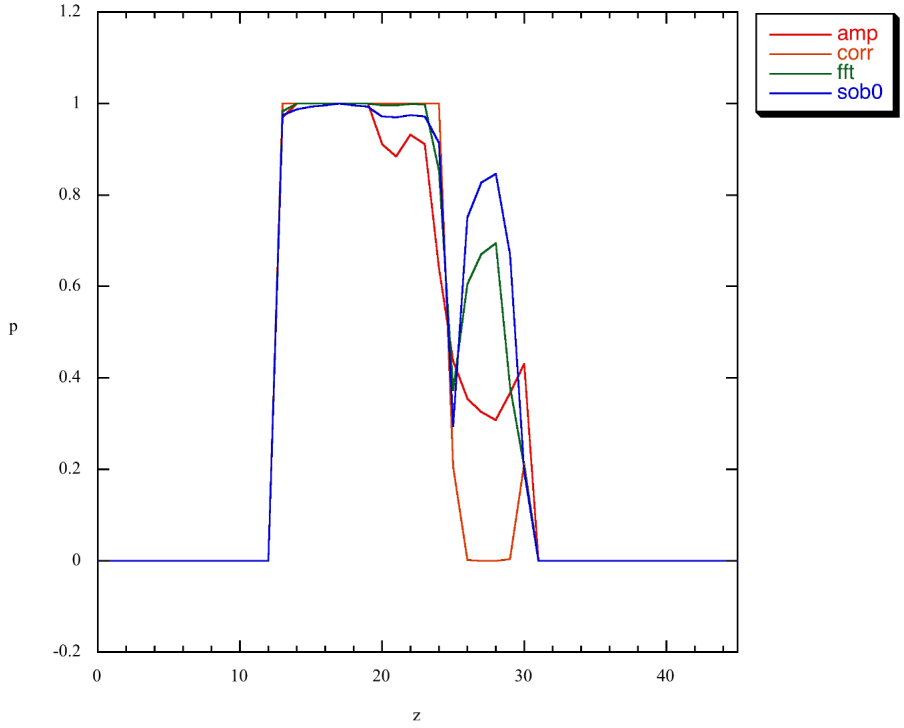


Figura 4.57: Perfil segundo z para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados filtrados com HRF).

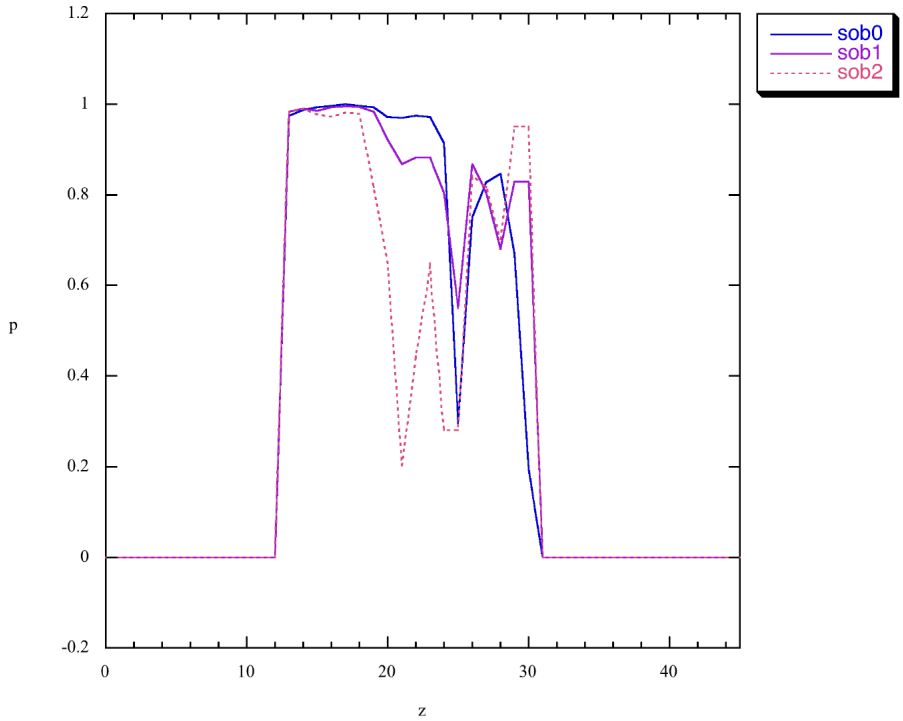


Figura 4.58: Perfil segundo z para os métodos de sobreposição de ordens 0, 1 e 2 (dados filtrados com HRF).

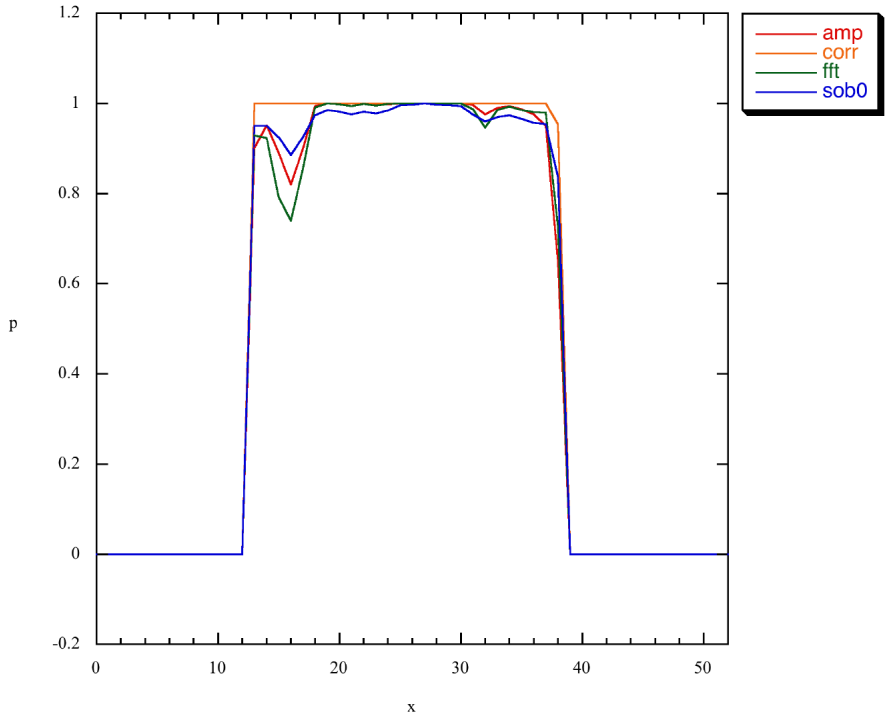


Figura 4.59: Perfil segundo x para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 0).

4.8.3.7 Não filtrados, com HRF e com atraso igual a 0

Apresentam-se aqui os perfis de probabilidade para dados não filtrados, com atraso nulo e analisados com uma sequência de modelo com HRF segundo as três direcções ortogonais (figuras 4.59, 4.60, 4.61, 4.62, 4.63 e 4.64). Para que as linhas dos vários métodos sejam discerníveis, os métodos são separados em dois gráficos por direcção.

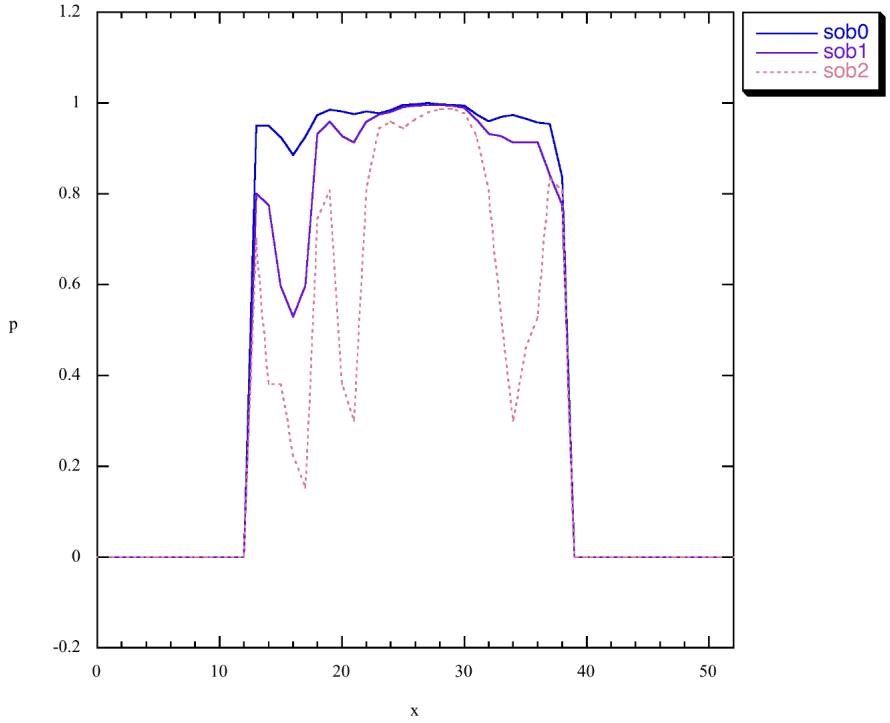


Figura 4.60: Perfil segundo x para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com HRF e com atraso igual a 0).

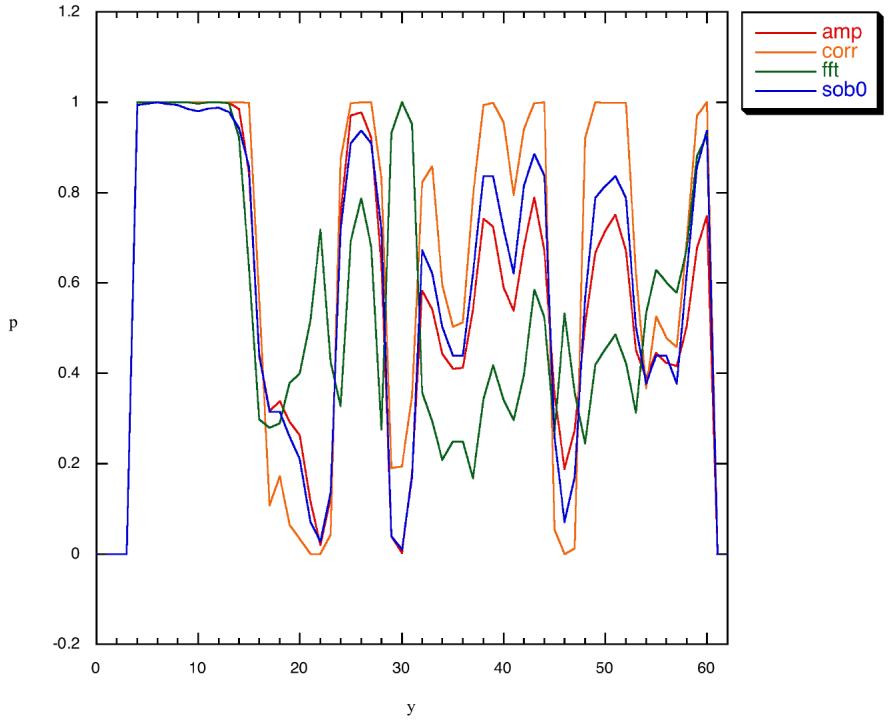


Figura 4.61: Perfil segundo y para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 0).

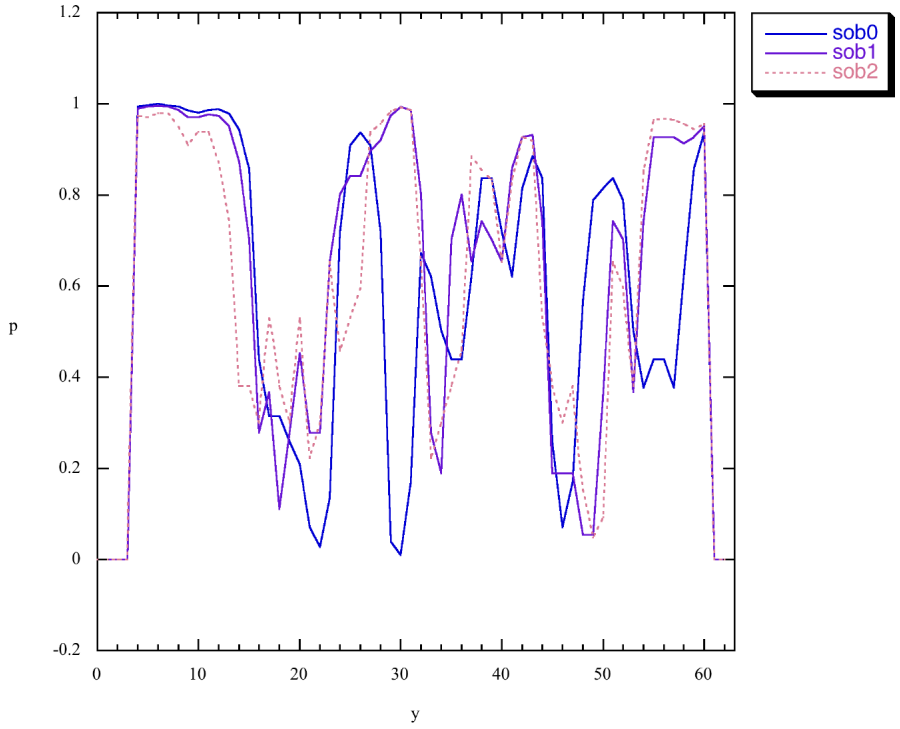


Figura 4.62: Perfil segundo y para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com HRF e com atraso igual a 0).

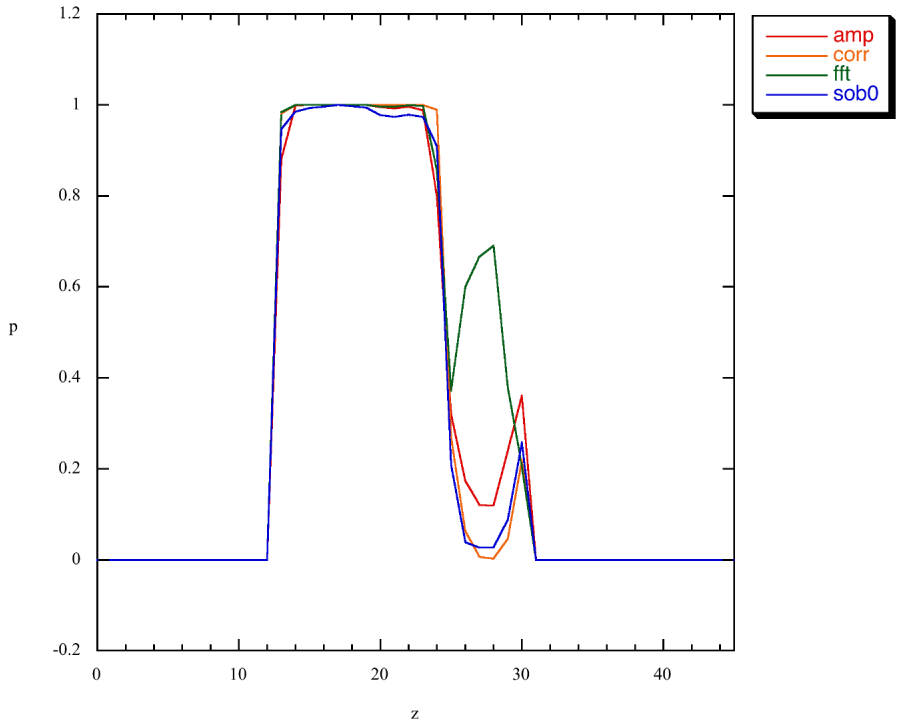


Figura 4.63: Perfil segundo z para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 0).

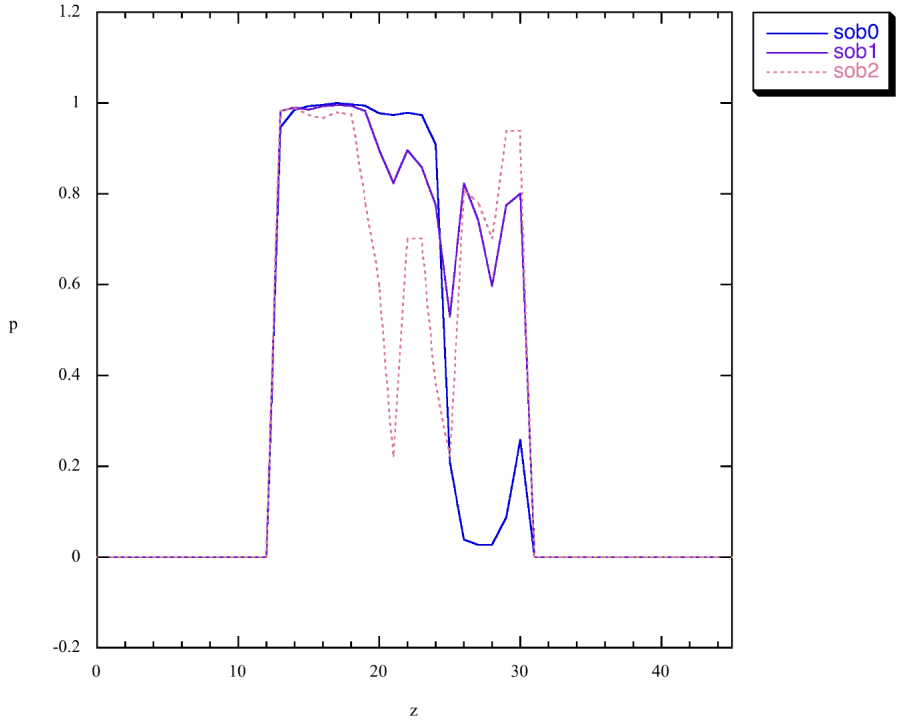


Figura 4.64: Perfil segundo z para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com HRF e com atraso igual a 0).

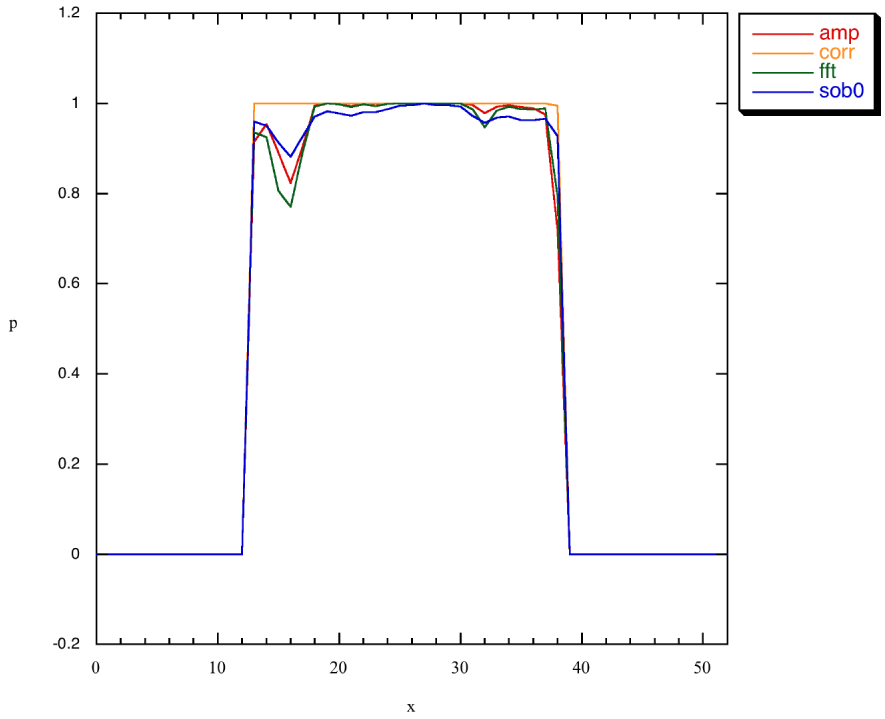


Figura 4.65: Perfil segundo x para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 1).

4.8.3.8 Não filtrados, com HRF e com atraso igual a 1

Apresentam-se aqui os perfis de probabilidade para dados não filtrados, com atraso unitário e analisados com uma sequência de modelo com HRF segundo as três direcções ortogonais (figuras 4.65, 4.66, 4.67, 4.68, 4.69 e 4.70). Para que as linhas dos vários métodos sejam discerníveis, os métodos são separados em dois gráficos por direcção.

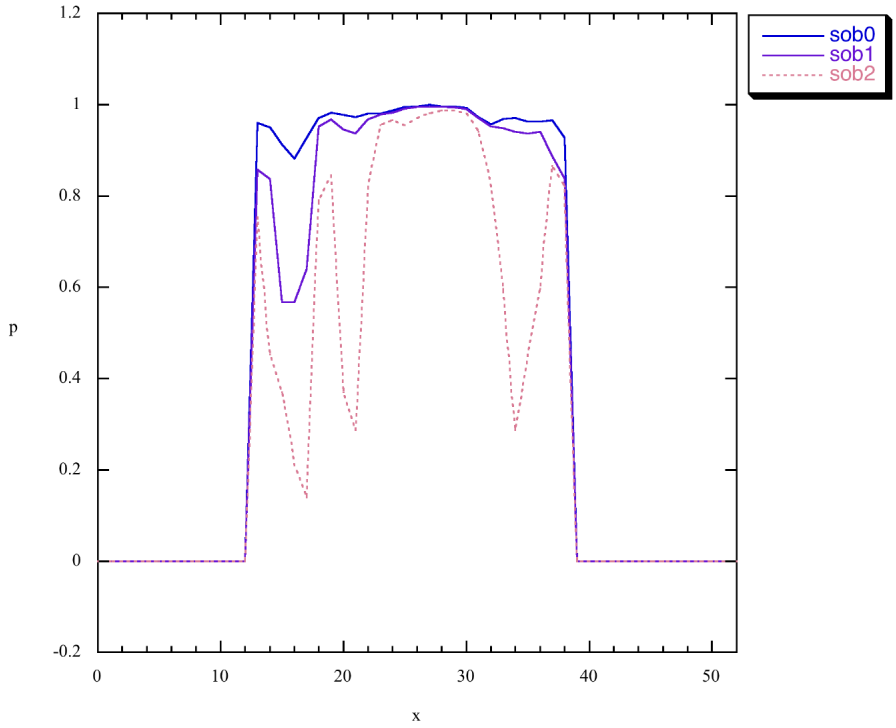


Figura 4.66: Perfil segundo x para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com HRF e com atraso igual a 1).

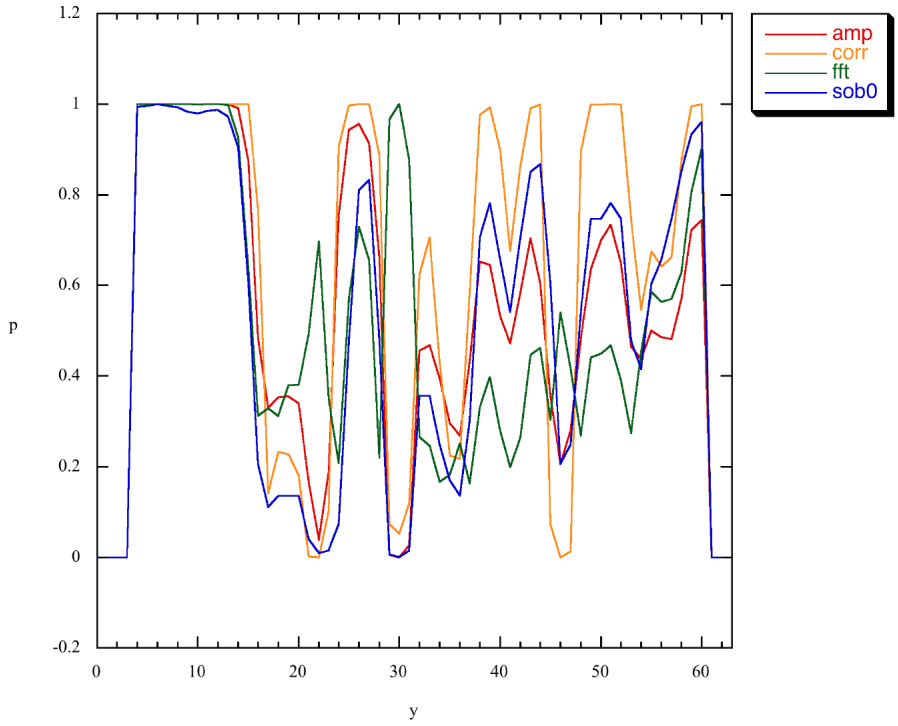


Figura 4.67: Perfil segundo y para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 1).

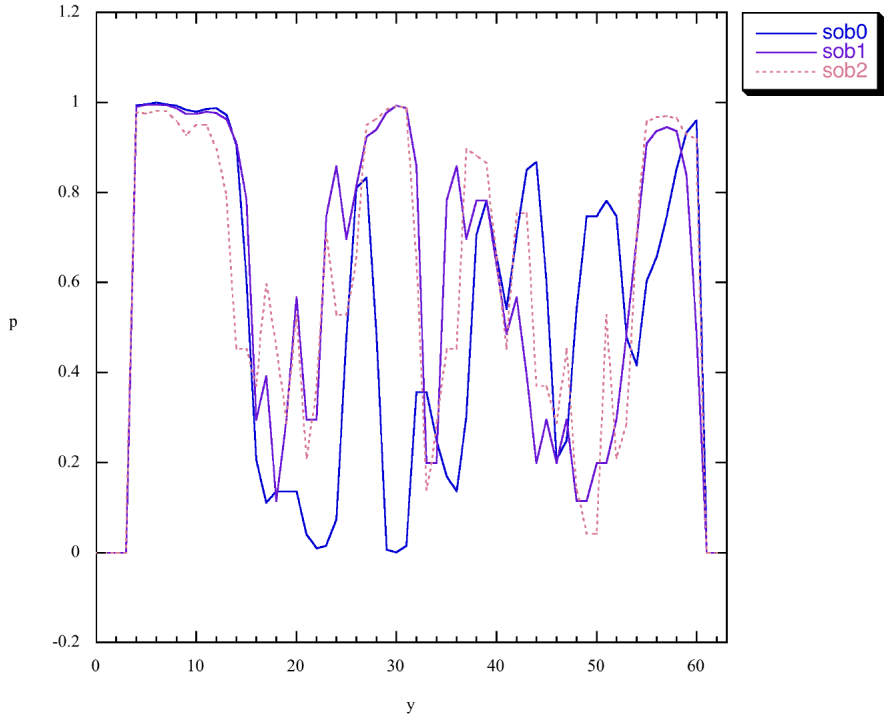


Figura 4.68: Perfil segundo y para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com HRF e com atraso igual a 1).

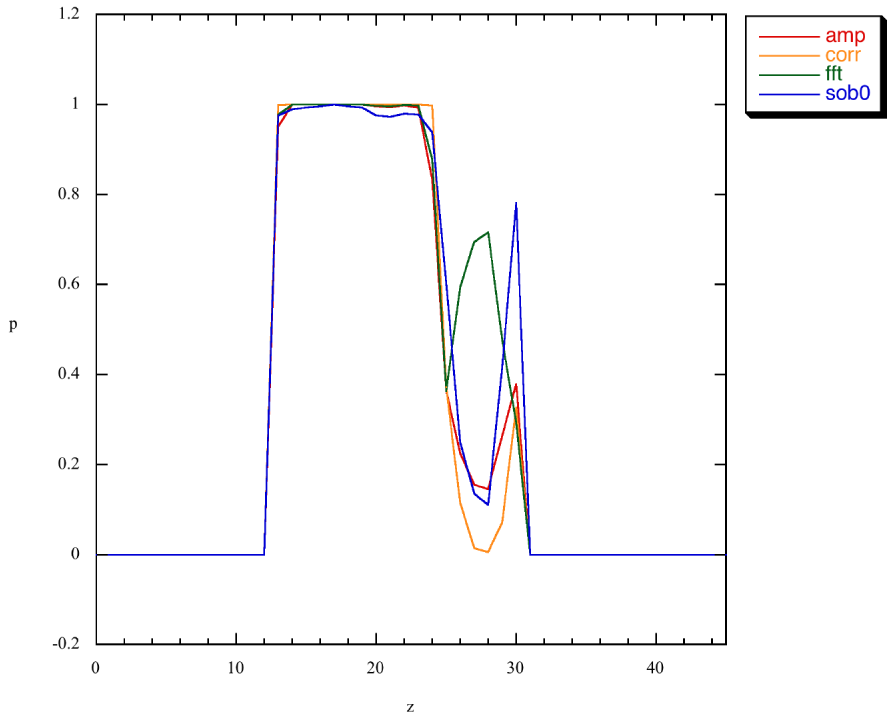


Figura 4.69: Perfil segundo z para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 1).

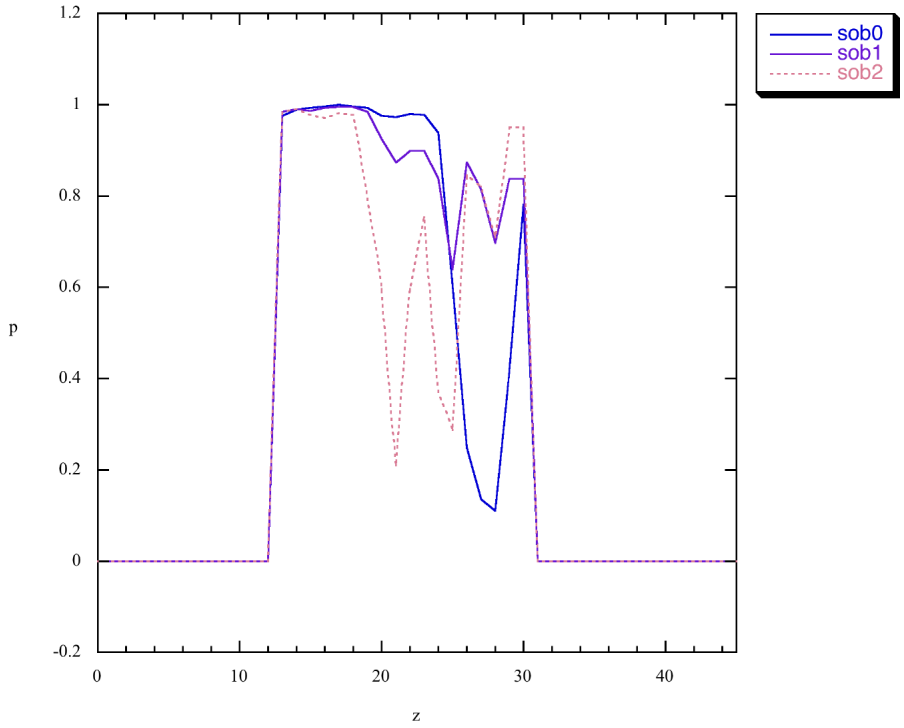


Figura 4.70: Perfil segundo z para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com HRF e com atraso igual a 1).

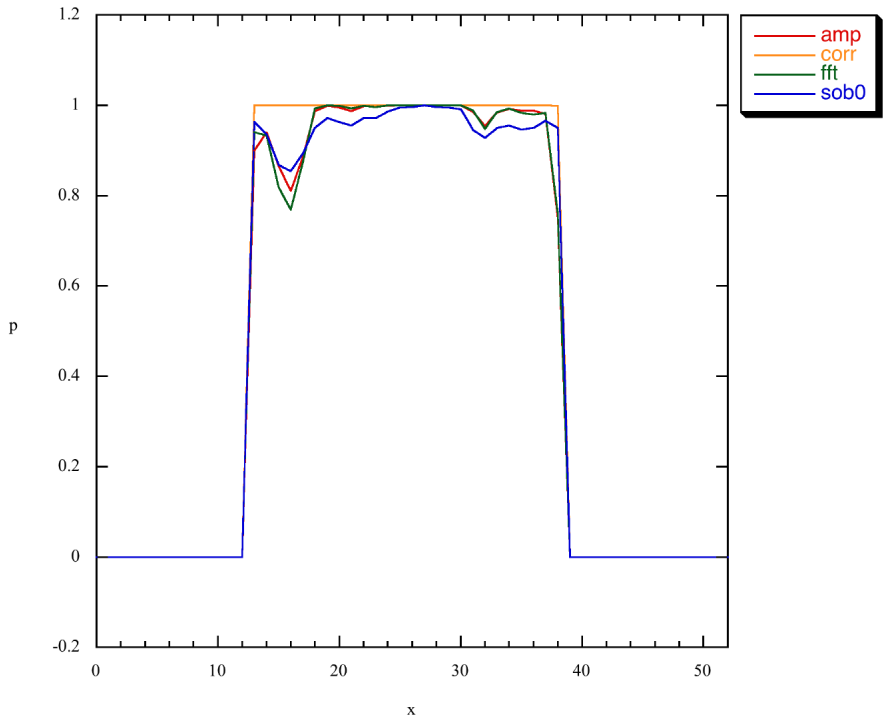


Figura 4.71: Perfil segundo x para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 2).

4.8.3.9 Não filtrados, com HRF e com atraso igual a 2

Apresentam-se aqui os perfis de probabilidade para dados não filtrados, com atraso de dois e analisados com uma sequência de modelo com HRF segundo as três direcções ortogonais (figuras 4.71, 4.72, 4.73, 4.74, 4.75 e 4.76). Para que as linhas dos vários métodos sejam discerníveis, os métodos são separados em dois gráficos por direcção.

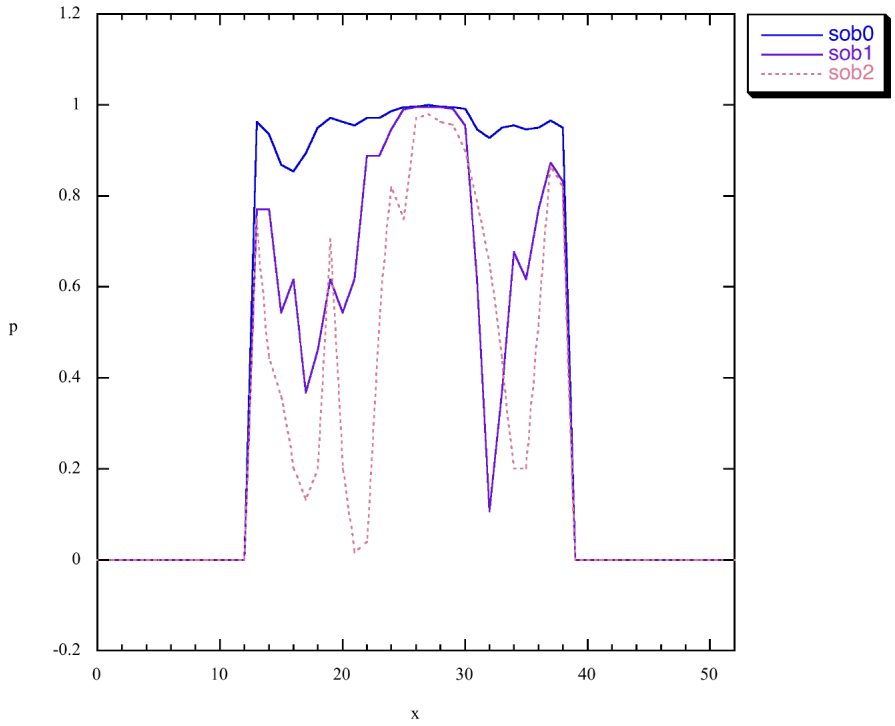


Figura 4.72: Perfil segundo x para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com HRF e com atraso igual a 2).

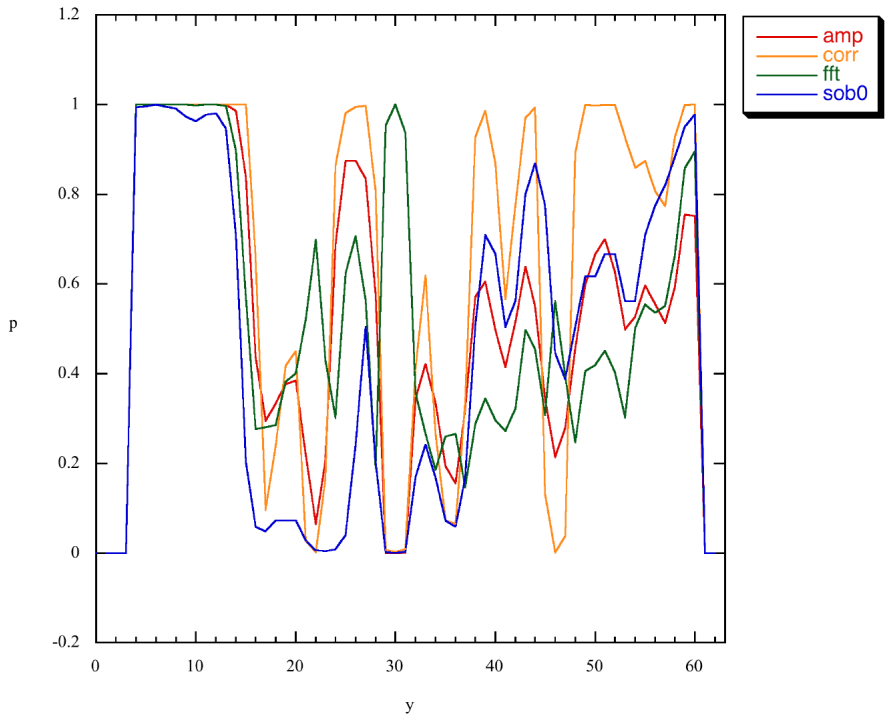


Figura 4.73: Perfil segundo y para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 2).

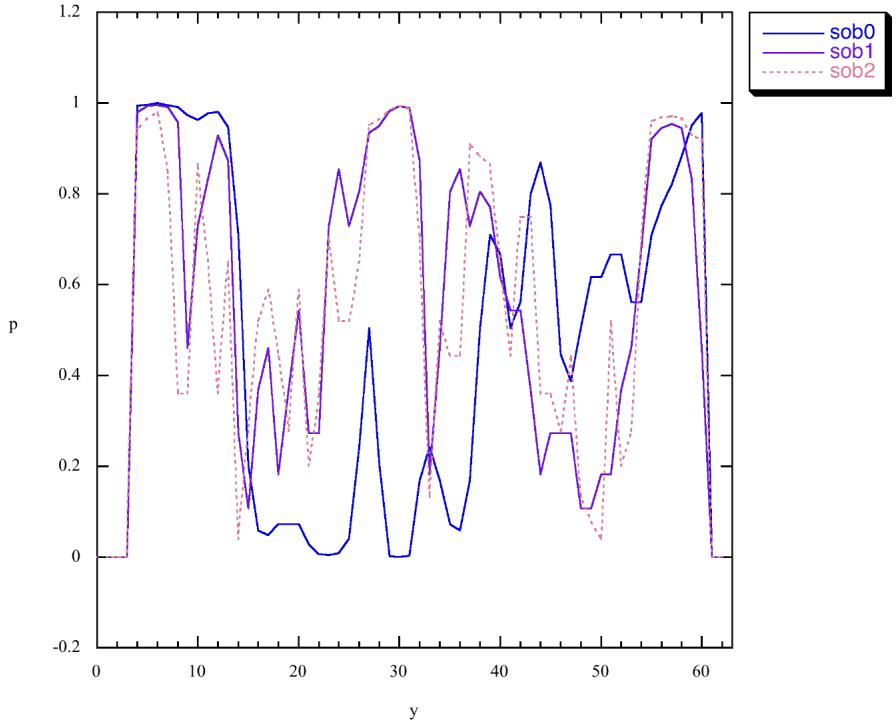


Figura 4.74: Perfil segundo y para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com HRF e com atraso igual a 2).

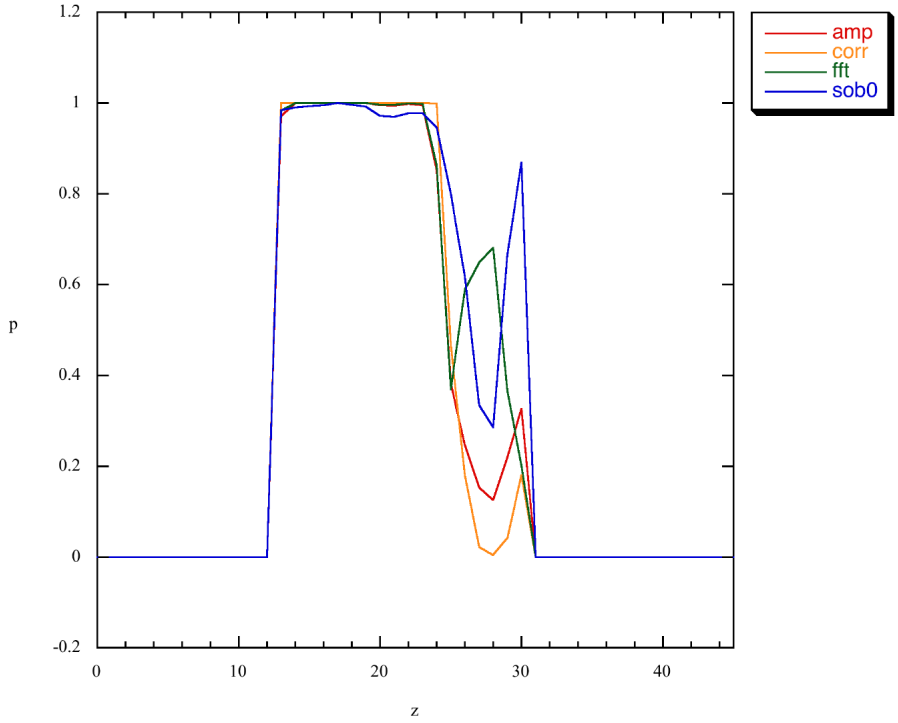


Figura 4.75: Perfil segundo z para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 2).

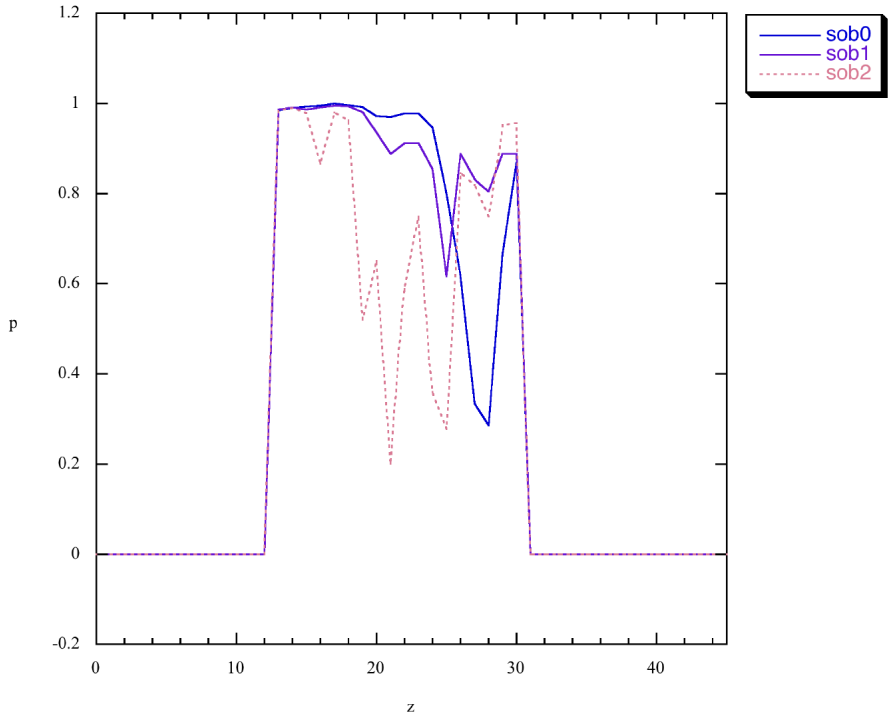


Figura 4.76: Perfil segundo z para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com HRF e com atraso igual a 2).

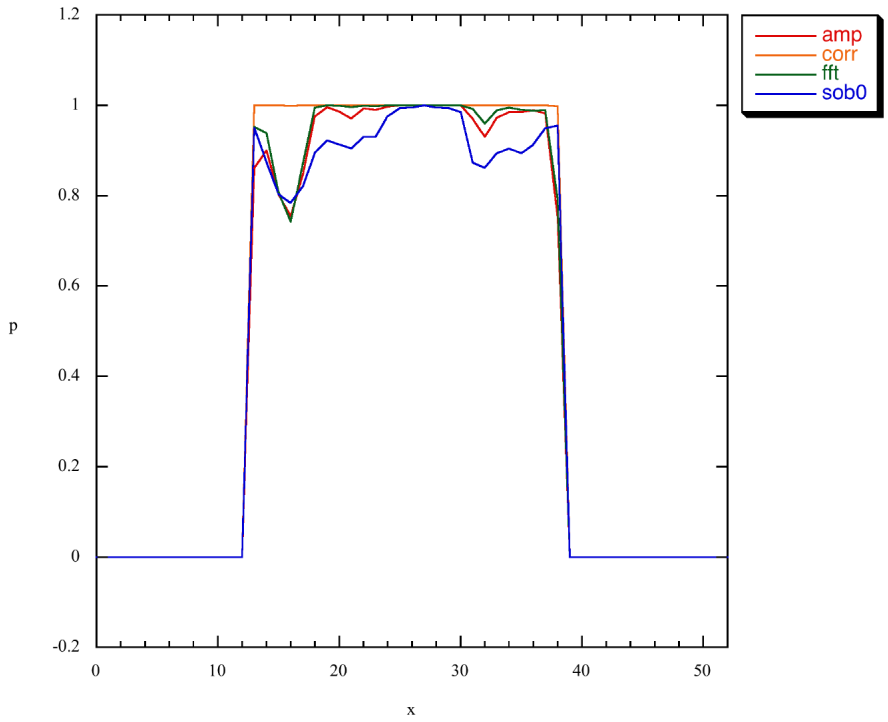


Figura 4.77: Perfil segundo x para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 3).

4.8.3.10 Não filtrados, com HRF e com atraso igual a 3

Apresentam-se aqui os perfis de probabilidade para dados não filtrados, com atraso de três e analisados com uma sequência de modelo com HRF segundo as três direcções ortogonais (figuras 4.77, 4.78, 4.79, 4.80, 4.81 e 4.82). Para que as linhas dos vários métodos sejam discerníveis, os métodos são separados em dois gráficos por direcção.

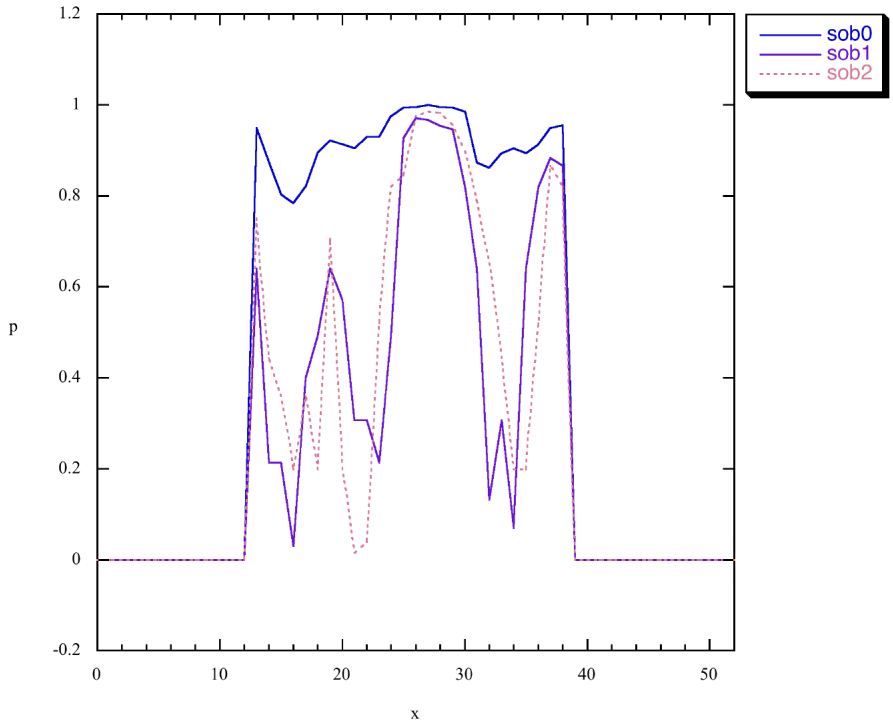


Figura 4.78: Perfil segundo x para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com HRF e com atraso igual a 3).

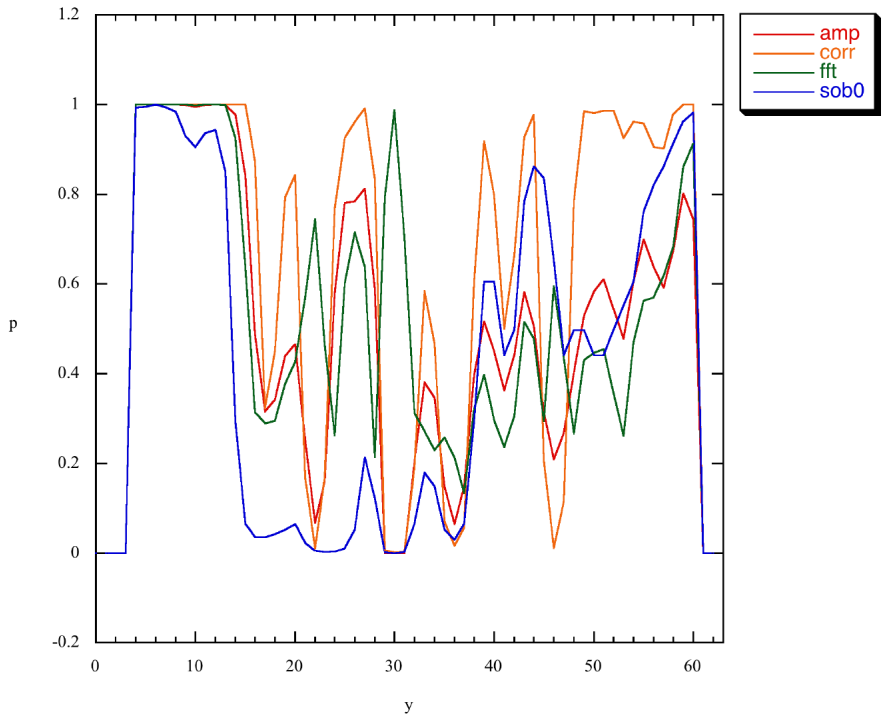


Figura 4.79: Perfil segundo y para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 3).

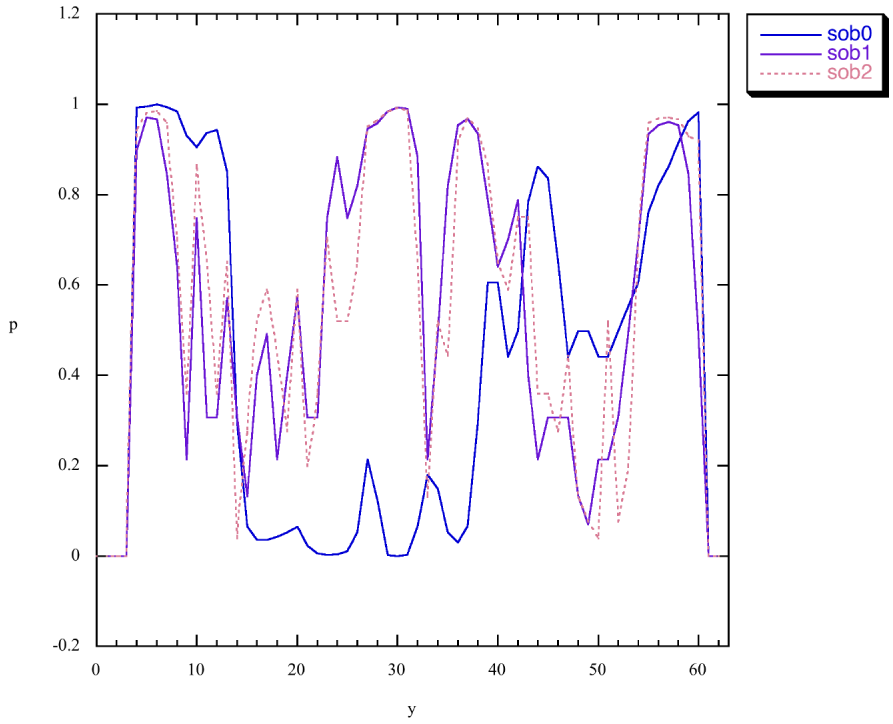


Figura 4.80: Perfil segundo y para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com HRF e com atraso igual a 3).

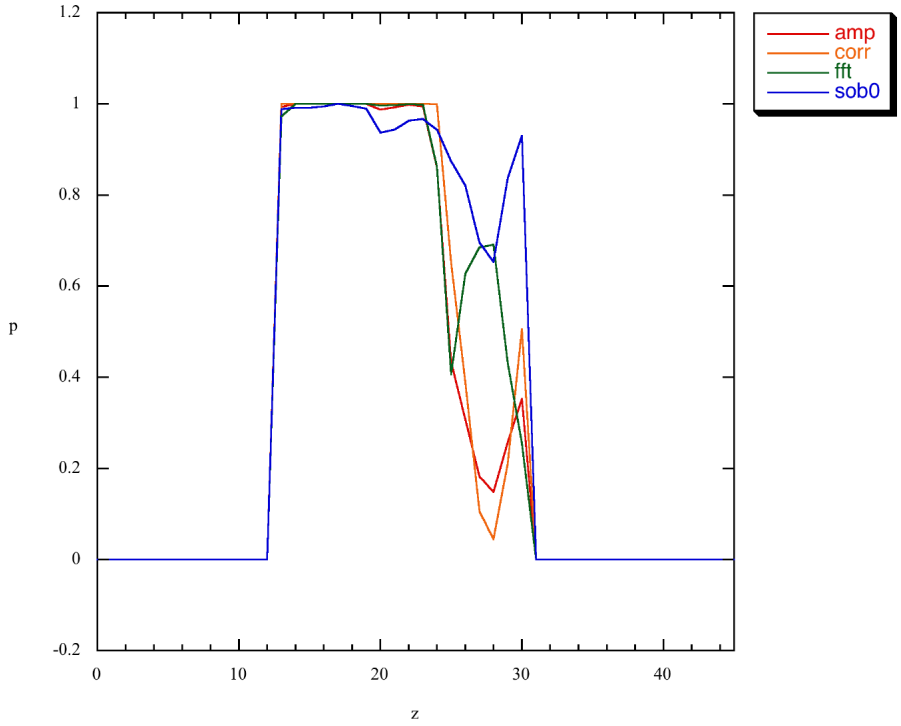


Figura 4.81: Perfil segundo z para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 3).

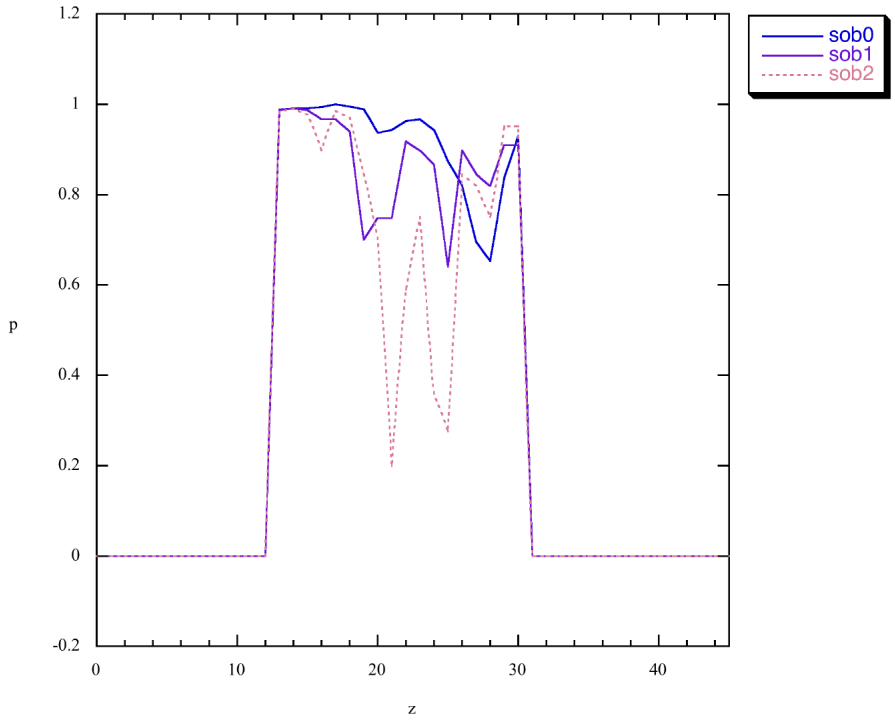


Figura 4.82: Perfil segundo z para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com HRF e com atraso igual a 3).

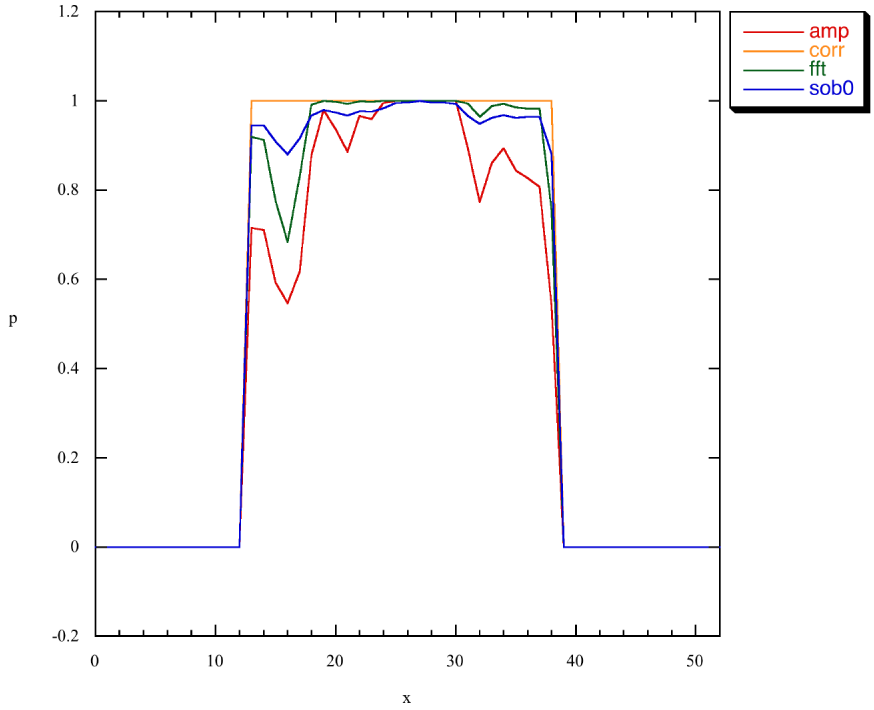


Figura 4.83: Perfil segundo x para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados filtrados, com sequência de modelo igual à sequência temporal do voxel (27,6,17)).

4.8.3.11 Filtrados e com conectividade funcional com (27,6,17)

Apresentam-se aqui os perfis de probabilidade para dados filtrados e analisados com uma sequência de modelo igual à sequência do voxel (27,6,17) (figuras 4.83, 4.84, 4.85, 4.86, 4.87 e 4.88). Para que as linhas dos vários métodos sejam discerníveis, os métodos são separados em dois gráficos por direcção.

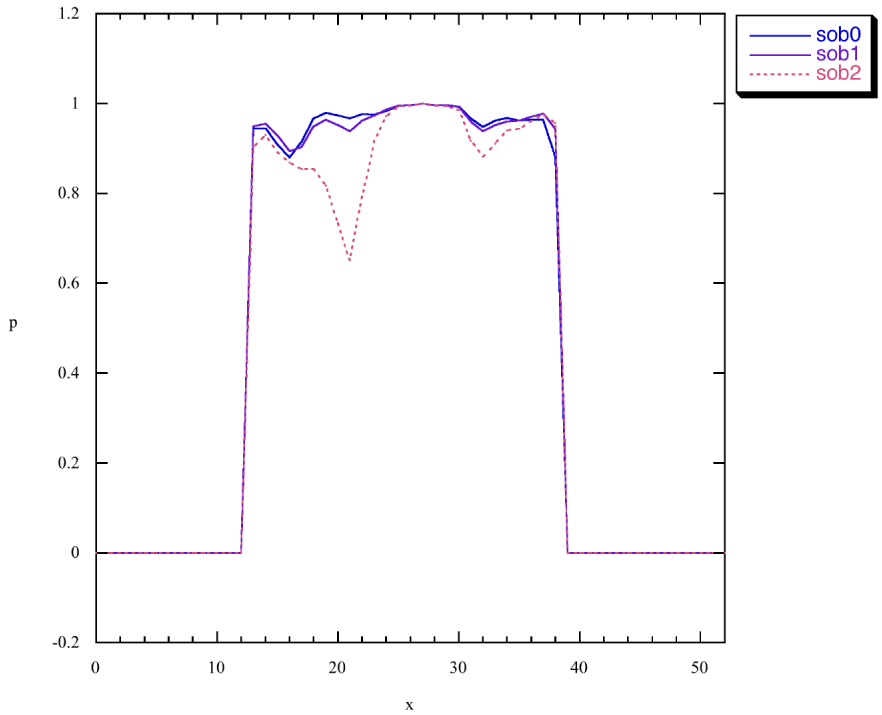


Figura 4.84: Perfil segundo x para os métodos de sobreposição de ordens 0, 1 e 2 (dados filtrados, com sequência de modelo igual à sequência temporal do voxel (27,6,17)).

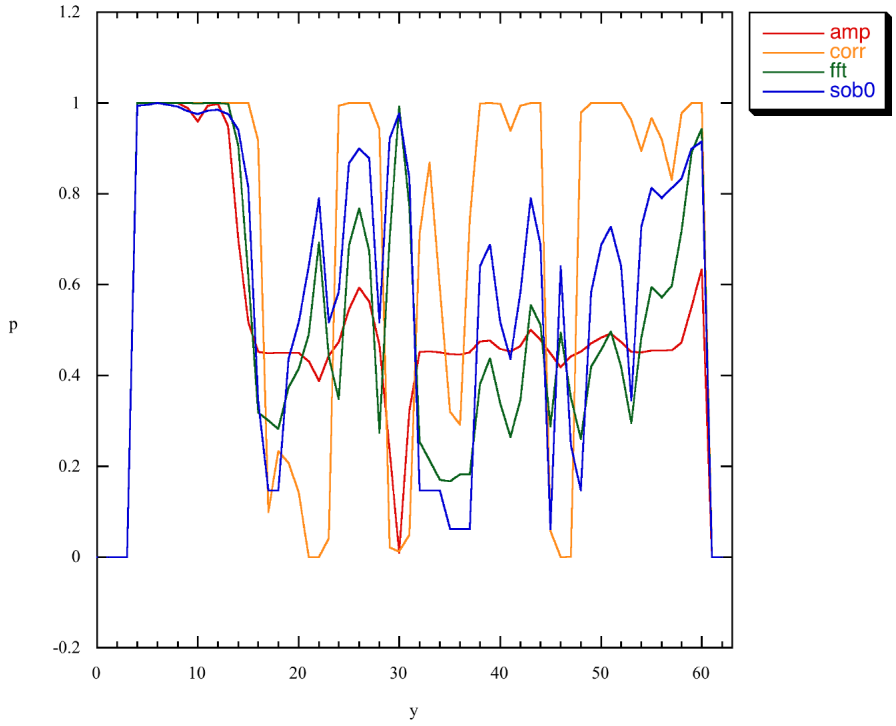


Figura 4.85: Perfil segundo y para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados filtrados, com sequência de modelo igual à sequência temporal do voxel (27,6,17)).

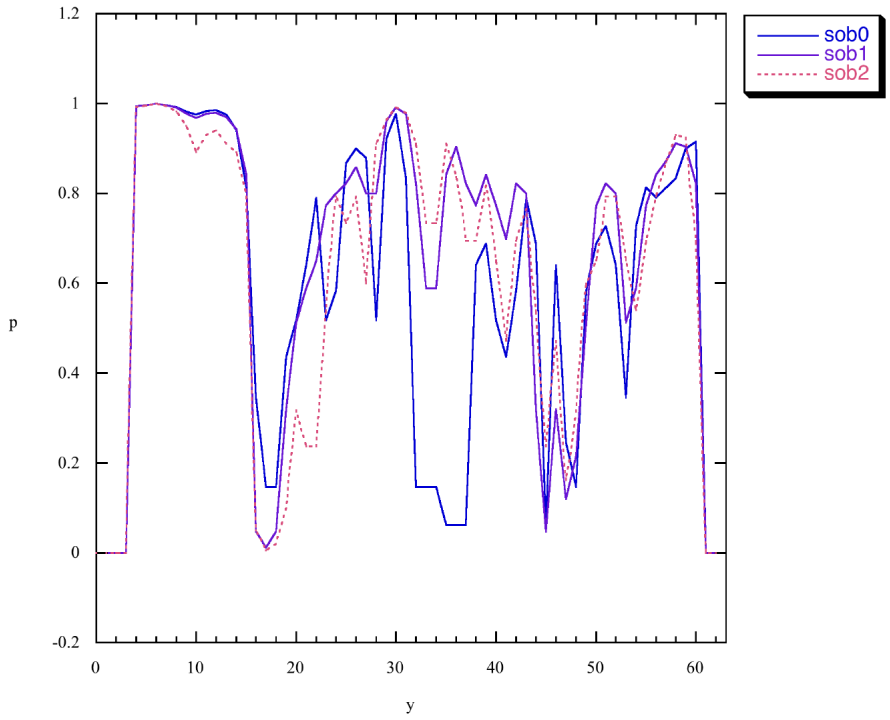


Figura 4.86: Perfil segundo y para os métodos de sobreposição de ordens 0, 1 e 2 (dados filtrados, com sequência de modelo igual à sequência temporal do voxel (27,6,17)).

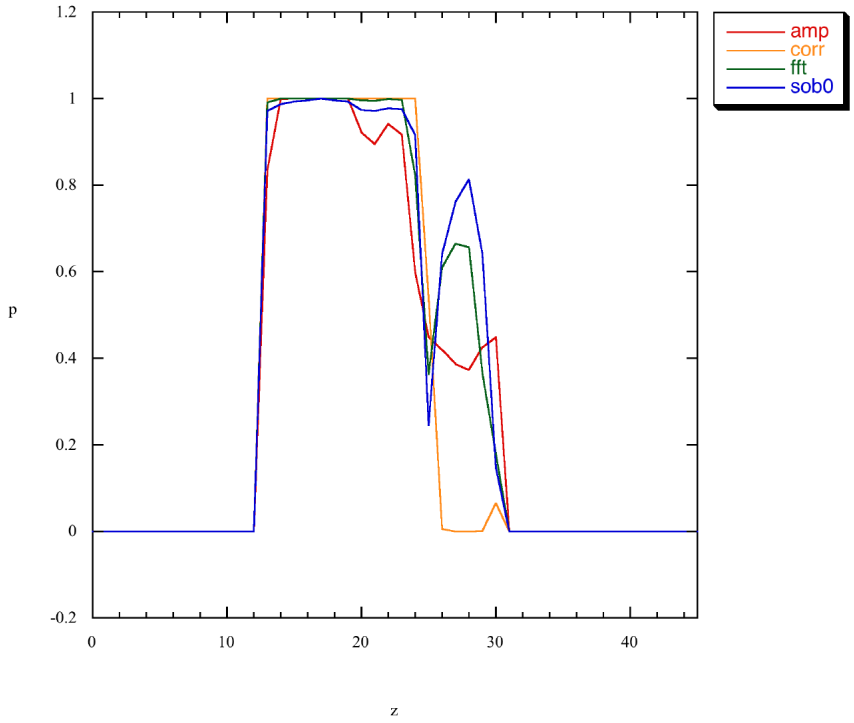


Figura 4.87: Perfil segundo z para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados filtrados, com sequência de modelo igual à sequência temporal do voxel (27,6,17)).

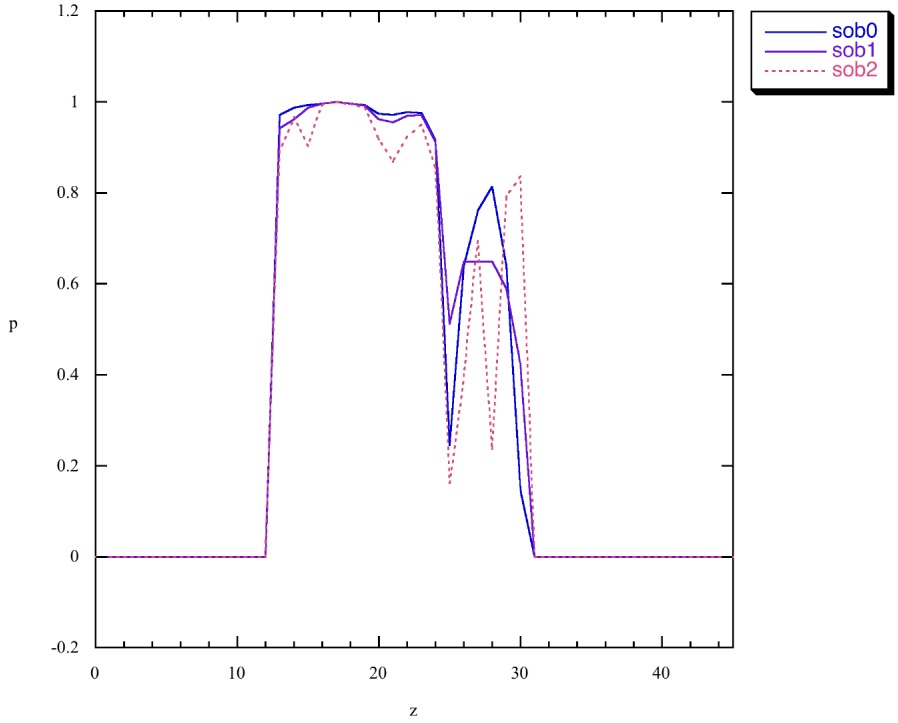


Figura 4.88: Perfil segundo z para os métodos de sobreposição de ordens 0, 1 e 2 (dados filtrados, com sequência de modelo igual à sequência temporal do voxel (27,6,17)).

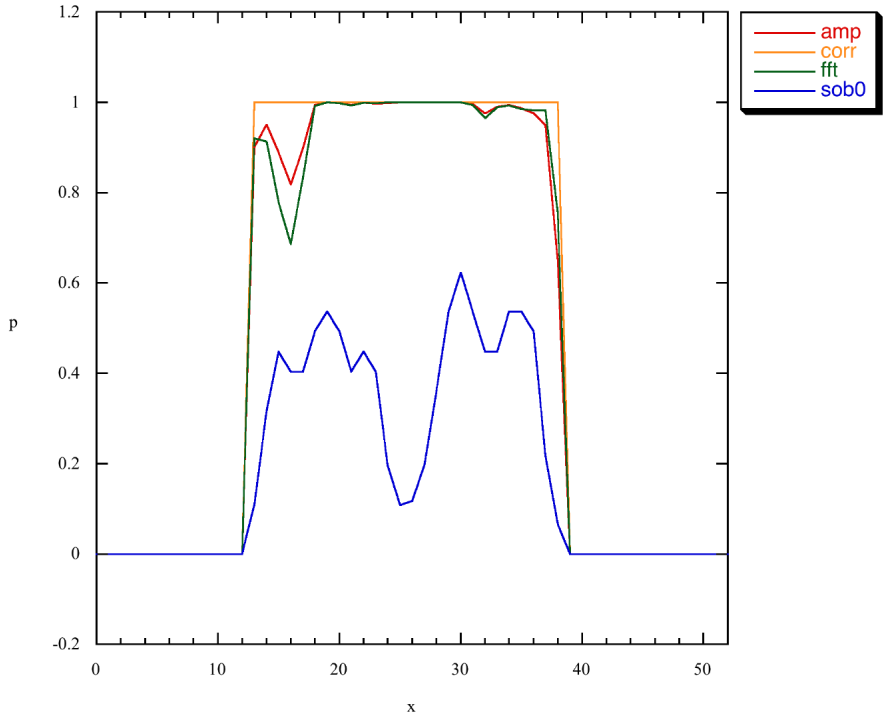


Figura 4.89: Perfil segundo x para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com sequência de modelo igual à sequência temporal do voxel (27,6,17)).

4.8.3.12 Não filtrados e com conectividade funcional com (27,6,17)

Apresentam-se aqui os perfis de probabilidade para dados não filtrados e analisados com uma sequência de modelo igual à sequência do voxel (27,6,17) (figuras 4.89, 4.90, 4.91, 4.92, 4.93 e 4.94). Para que as linhas dos vários métodos sejam discerníveis, os métodos são separados em dois gráficos por direcção.

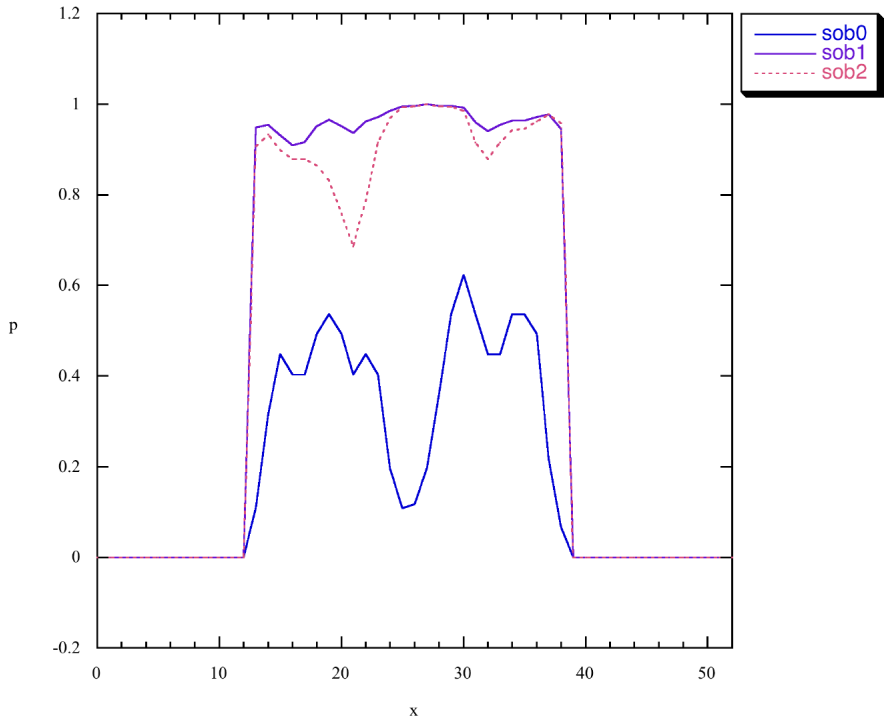


Figura 4.90: Perfil segundo x para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com sequência de modelo igual à sequência temporal do voxel (27,6,17)).

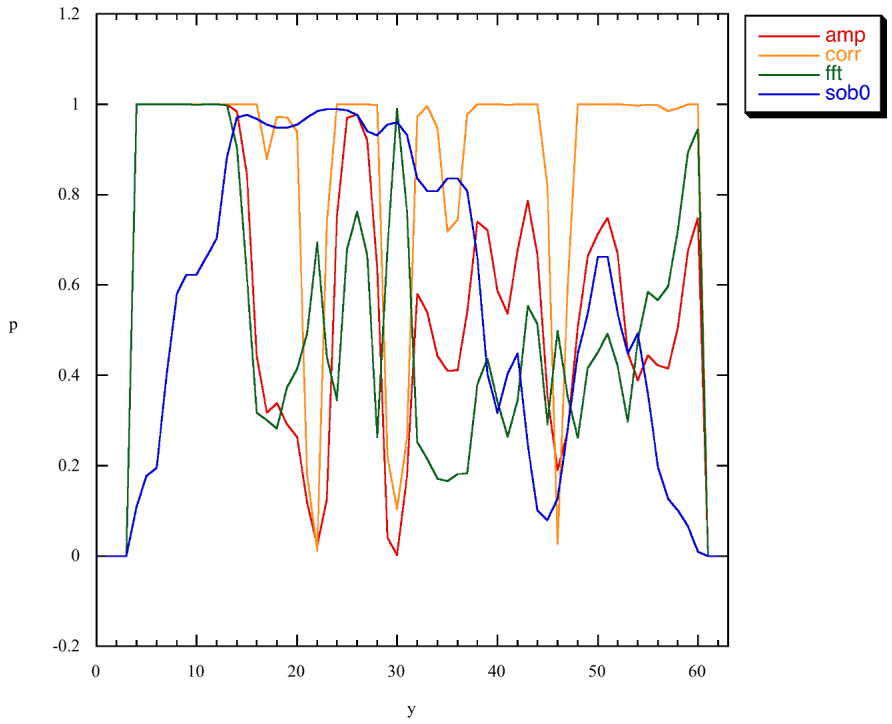


Figura 4.91: Perfil segundo y para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com sequência de modelo igual à sequência temporal do voxel (27,6,17)).

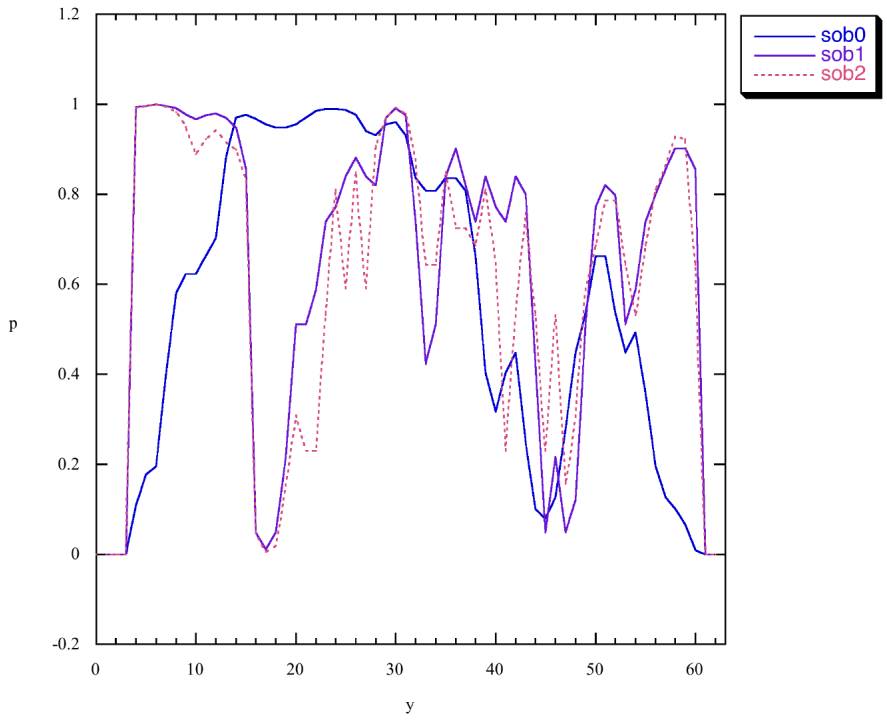


Figura 4.92: Perfil segundo y para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com sequência de modelo igual à sequência temporal do voxel (27,6,17)).

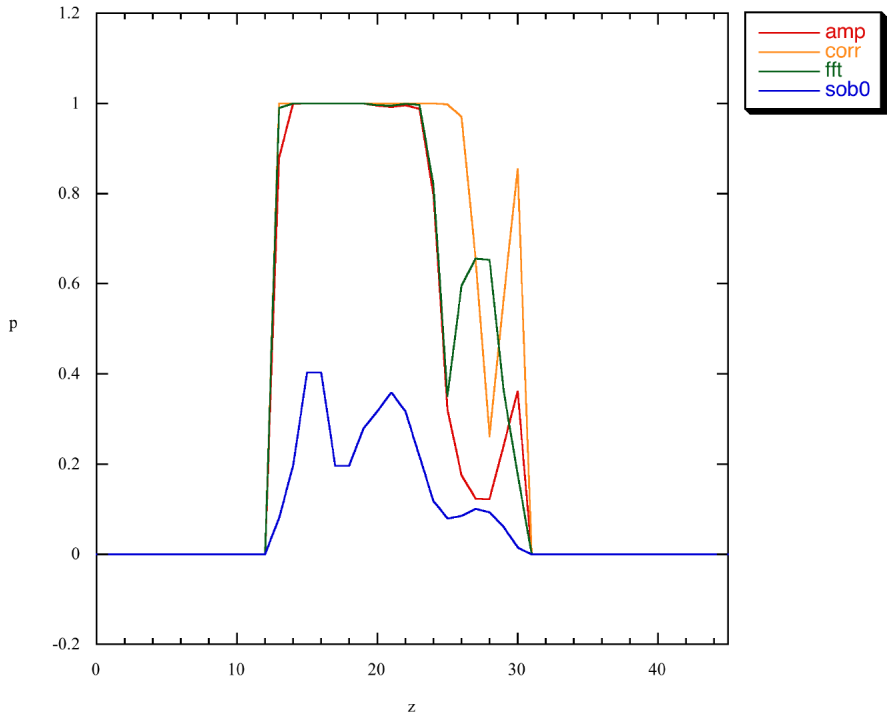


Figura 4.93: Perfil segundo z para os métodos de amplitude, correlação, TRF e sobreposição de ordem 0 (dados não filtrados, com sequência de modelo igual à sequência temporal do voxel (27,6,17)).

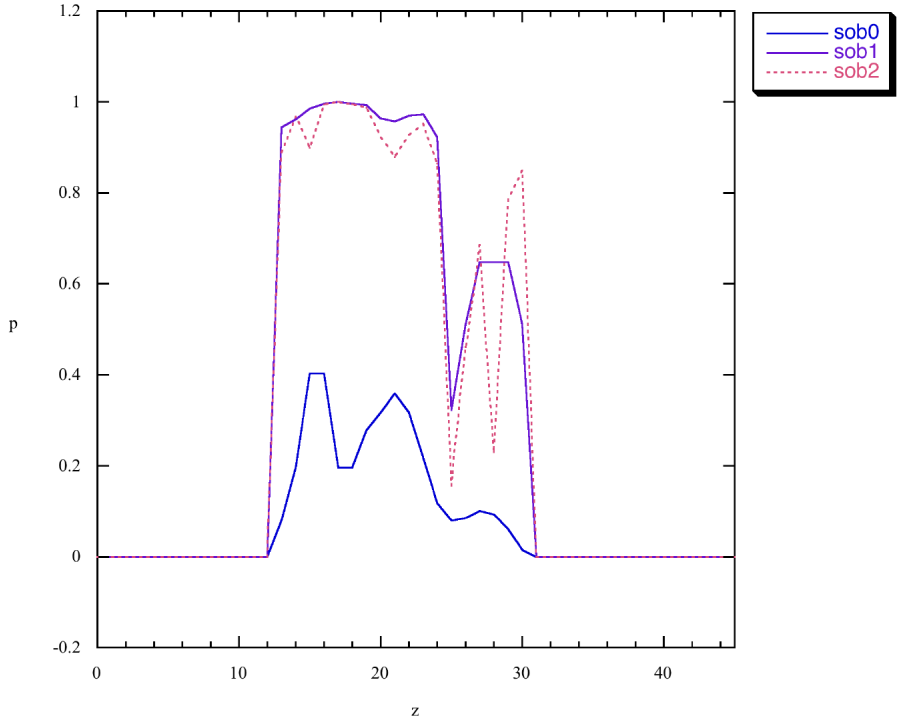


Figura 4.94: Perfil segundo z para os métodos de sobreposição de ordens 0, 1 e 2 (dados não filtrados, com sequência de modelo igual à sequência temporal do voxel (27,6,17)).

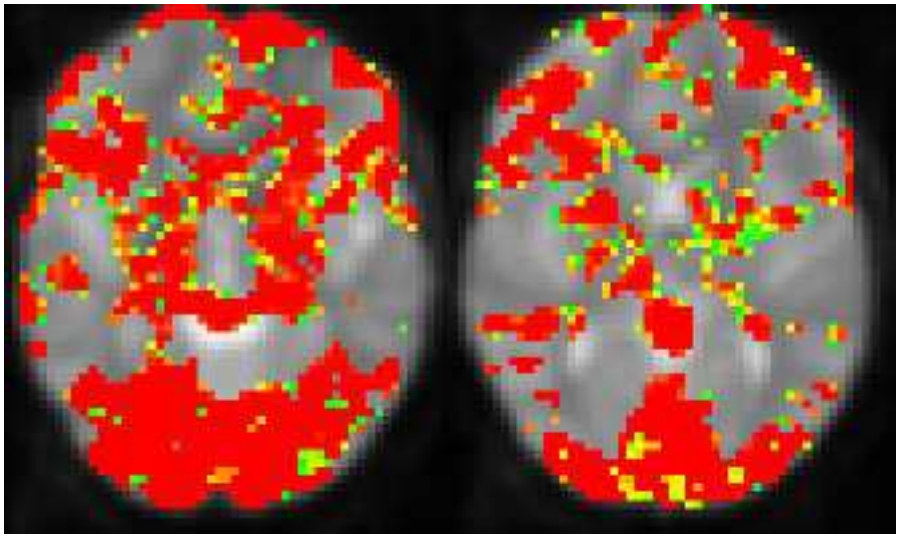


Figura 4.95: Método t de Student (dados filtrados sem HRF).

4.8.3.13 Conclusões sobre os perfis

Os perfis dos dados filtrados não apresentam diferenças significativas quando se passa de uma sequência de modelo de bloco para HRF. O mesmo acontece para os dados não filtrados.

Quanto aos atrasos, apenas o método de sobreposição tem variações significativas com o aumento do atraso: torna-se mais selectivo. Esse efeito aumenta com a ordem e parece ser independente da adopção de modelo com ou sem HRF.

No geral podemos dizer que o método mais conservador (menos selectivo) é o de coeficiente de correlação. O seguinte nesta escala é o método de t de Student.

O método de sobreposição de ordem 2 é o mais selectivo para os perfis.

Quanto aos restantes métodos a ordem é variável.

A forma geral dos perfis é semelhante na zona de activação para todos os métodos. Apenas os métodos de picos de Fourier e de sobreposição têm por vezes comportamentos distintos fora das zonas de activação.

Nos perfis de conectividade funcional com o voxel (27,6,17) observa-se que o método de sobreposição de ordem 0 tem um comportamento totalmente distinto (como anteviam os resultados dos extremos) para os dados não filtrados. De facto a zona de activação máxima está muito longe da obtida para os restantes métodos. Esse comportamento não é seguido pelos seus congéneres de ordem superior.

O método de amplitude torna-se significativamente mais selectivo para os dados filtrados.

4.8.4 Imagens

Aqui são apresentadas as imagens obtidas das fatias 17 e 20 para todos os métodos.

4.8.4.1 Filtrados sem HRF

Apresentamos nesta secção as imagens de activação produzidas pelo programa *Cérebro* para cada um dos métodos com dados filtrados analisados com uma sequência de modelo em bloco (figuras 4.95, 4.96, 4.97, 4.98, 4.99, 4.100, 4.101 e 4.102).

A área de activação é muito grande e é praticamente idêntica à obtida para o método de coeficiente de correlação.

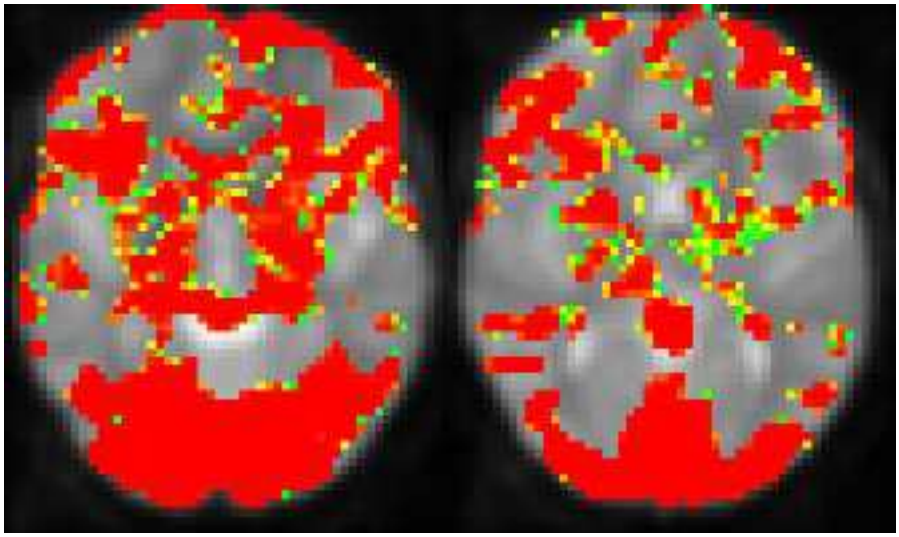


Figura 4.96: Método de coeficiente de correlação (dados filtrados sem HRF).

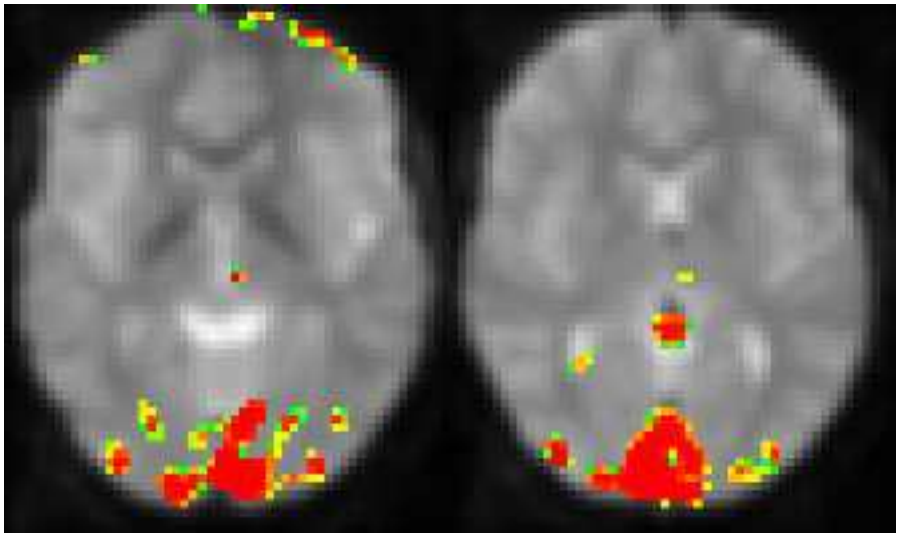


Figura 4.97: Método dos picos de Fourier (dados filtrados sem HRF).

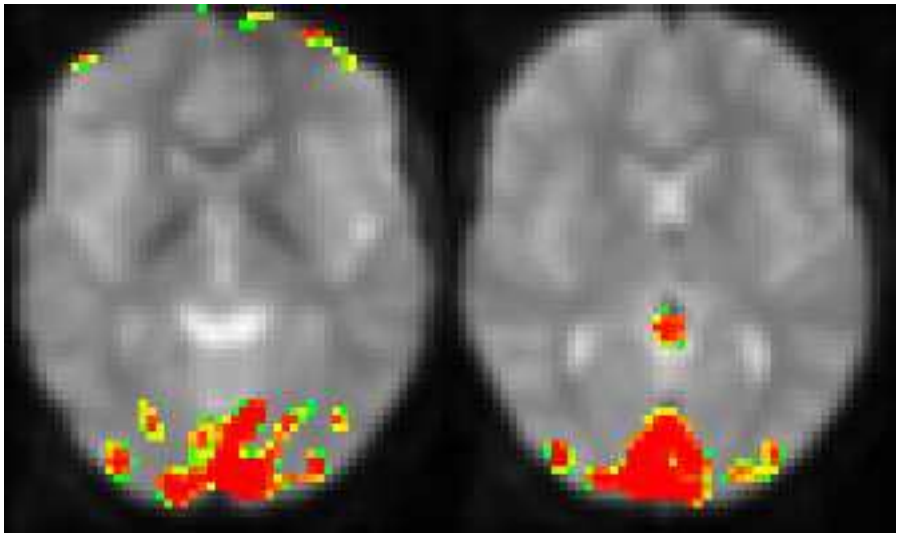


Figura 4.98: Método do modelo linear generalizado (dados filtrados sem HRF)

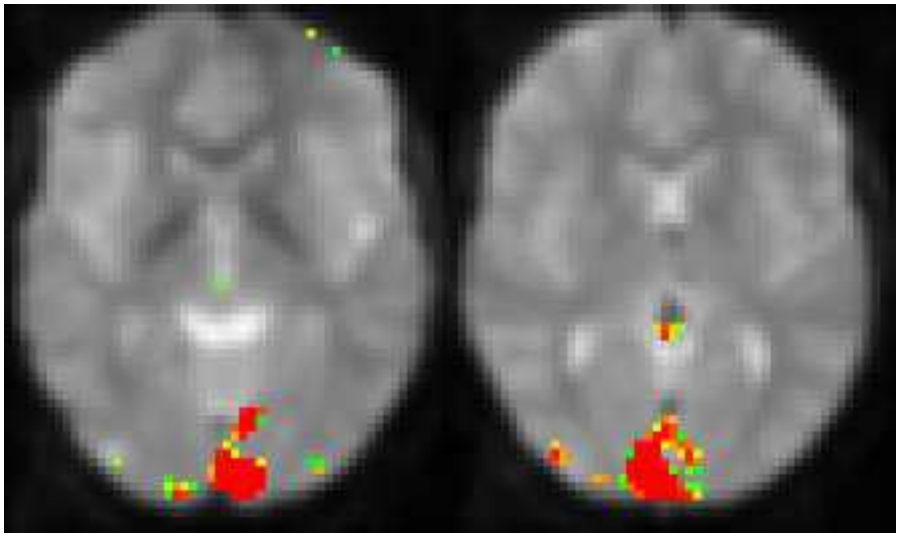


Figura 4.99: Método de amplitude (dados filtrados sem HRF)

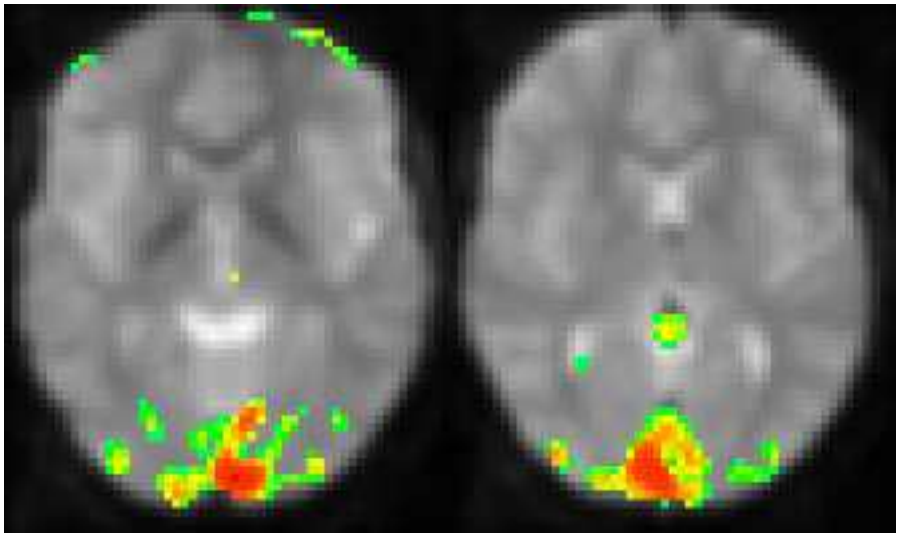


Figura 4.100: Método de sobreposição de ordem 0 (dados filtrados sem HRF).

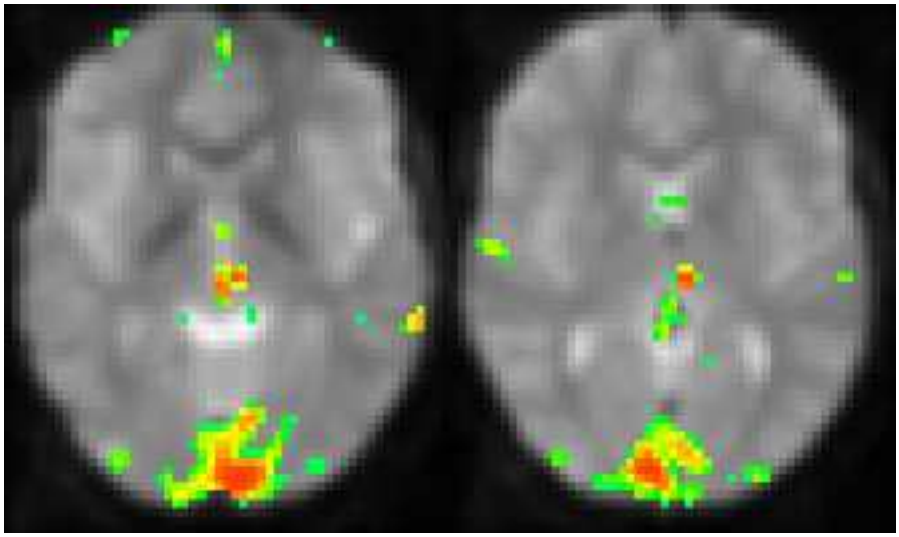


Figura 4.101: Método de sobreposição dde ordem 1 (dados filtrados sem HRF).

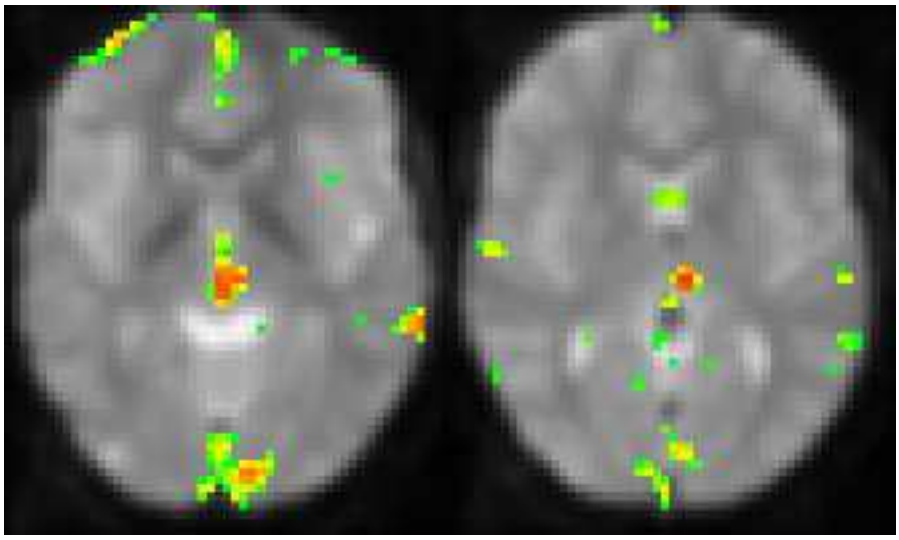


Figura 4.102: Método de sobreposição de ordem 2 (dados filtrados sem HRF).

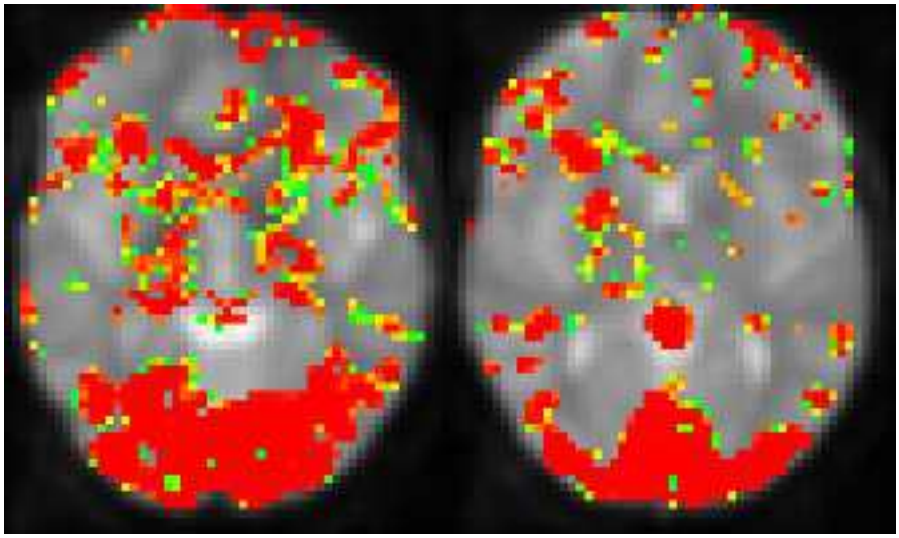


Figura 4.103: Método de t de Student (dados não filtrados, sem HRF e com atraso igual a 0).

4.8.4.2 Não filtrados, sem HRF e com atraso igual a 0

Apresentamos nesta secção as imagens de activação produzidas pelo programa *Cérebro* para cada um dos métodos com dados não filtrados e analisados com uma sequência de modelo em bloco com atraso nulo (figuras 4.103, 4.104, 4.105, 4.106, 4.107, 4.108, 4.109 e 4.110).

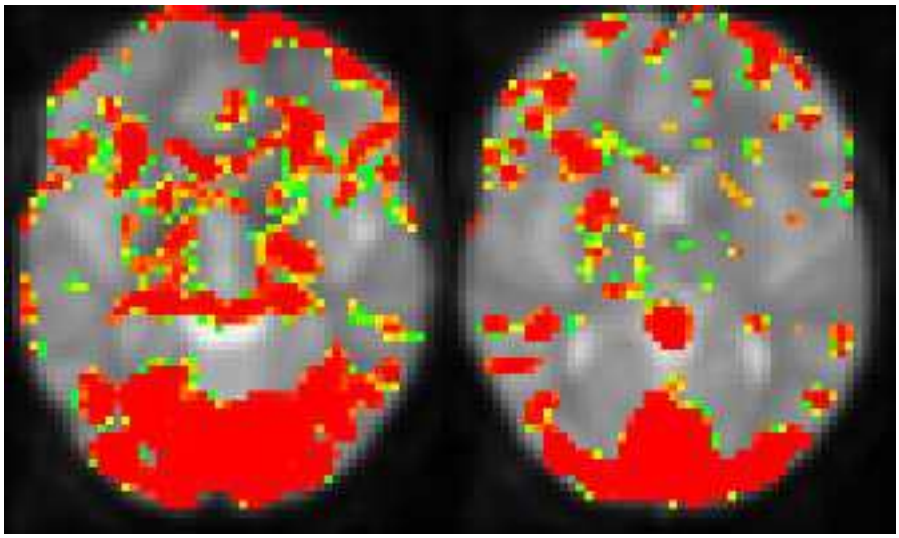


Figura 4.104: Método de coeficiente de correlação (dados não filtrados, sem HRF e com atraso igual a 0).

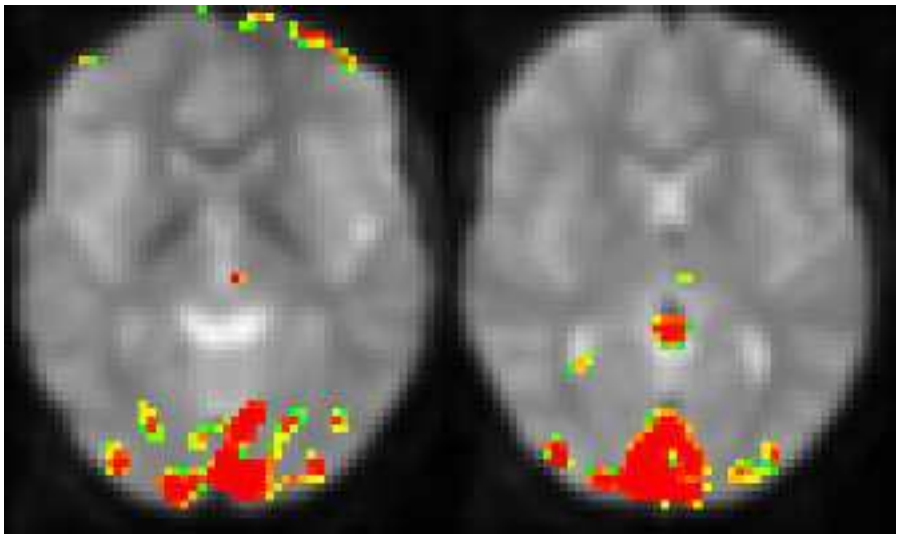


Figura 4.105: Método dos picos de Fourier (dados não filtrados, sem HRF e com atraso igual a 0).

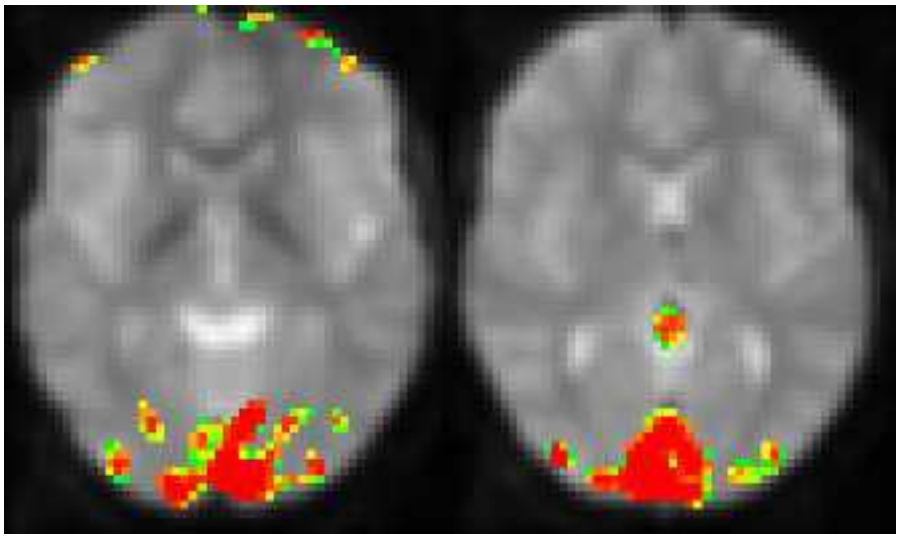


Figura 4.106: Método do modelo linear generalizado (dados não filtrados, sem HRF e com atraso igual a 0).

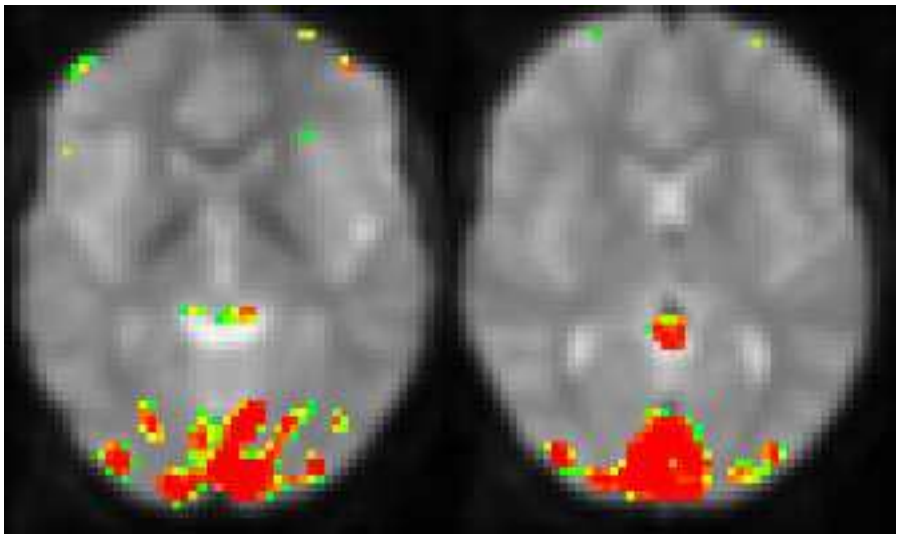


Figura 4.107: Método de amplitude (dados não filtrados, sem HRF e com atraso igual a 0).

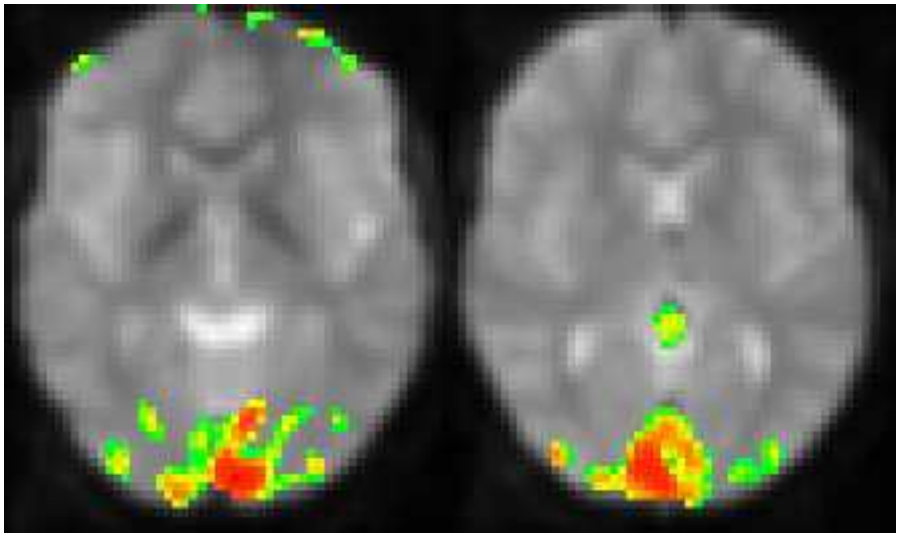


Figura 4.108: Método de sobreposição de ordem 0 (dados não filtrados, sem HRF e com atraso igual a 0).

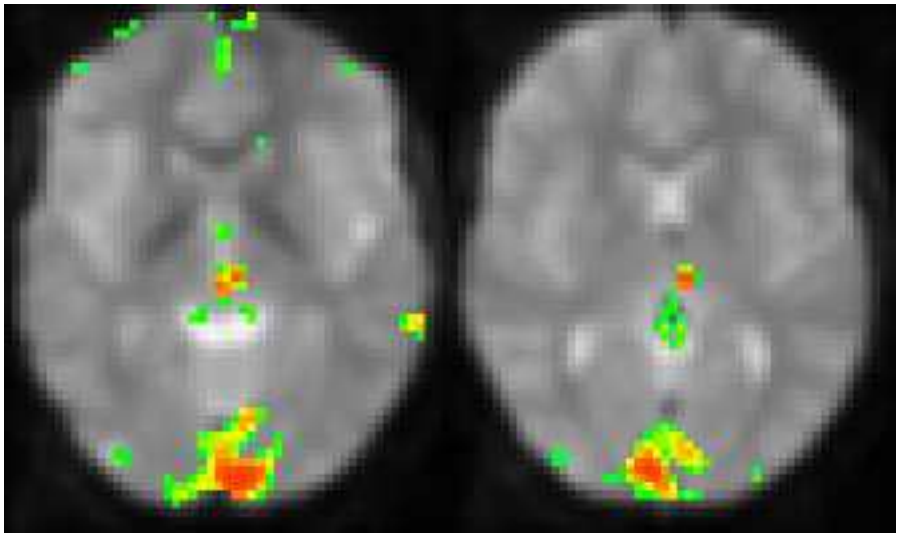


Figura 4.109: Método de sobreposição de ordem 1 (dados não filtrados, sem HRF e com atraso igual a 0).

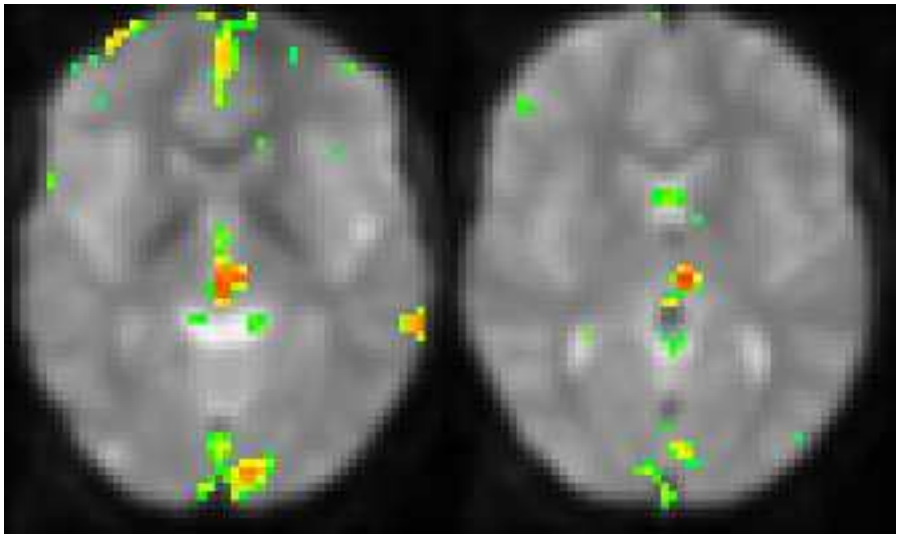


Figura 4.110: Método de sobreposição de ordem 2 (dados não filtrados, sem HRF e com atraso igual a 0).

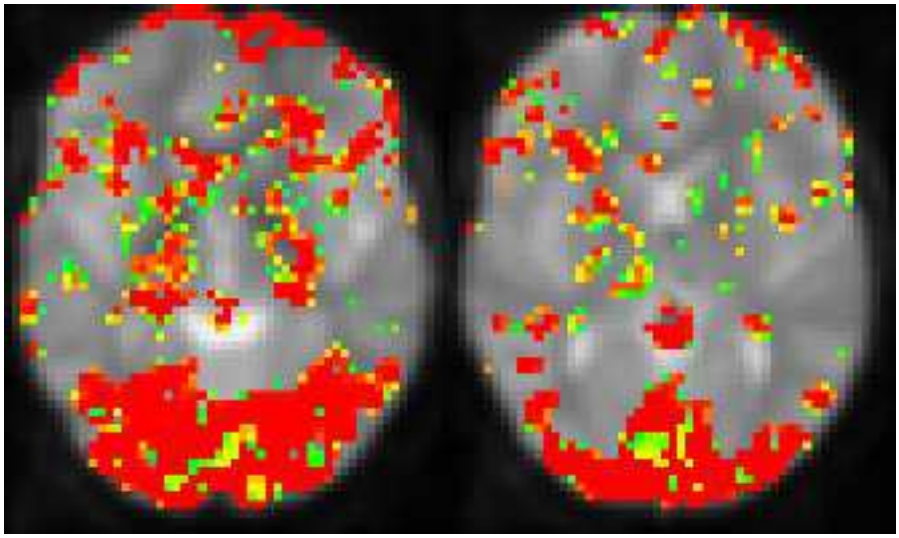


Figura 4.111: Método de t de Student (dados não filtrados, sem HRF e com atraso igual a 1).

4.8.4.3 Não filtrados, sem HRF e com atraso igual a 1

Apresentamos nesta secção as imagens de activação produzidas pelo programa *Cérebro* para cada um dos métodos com dados não filtrados e analisados com uma sequência de modelo em bloco com atraso unitário (figuras 4.111, 4.112, 4.113, 4.114, 4.115, 4.116, 4.117 e 4.118).

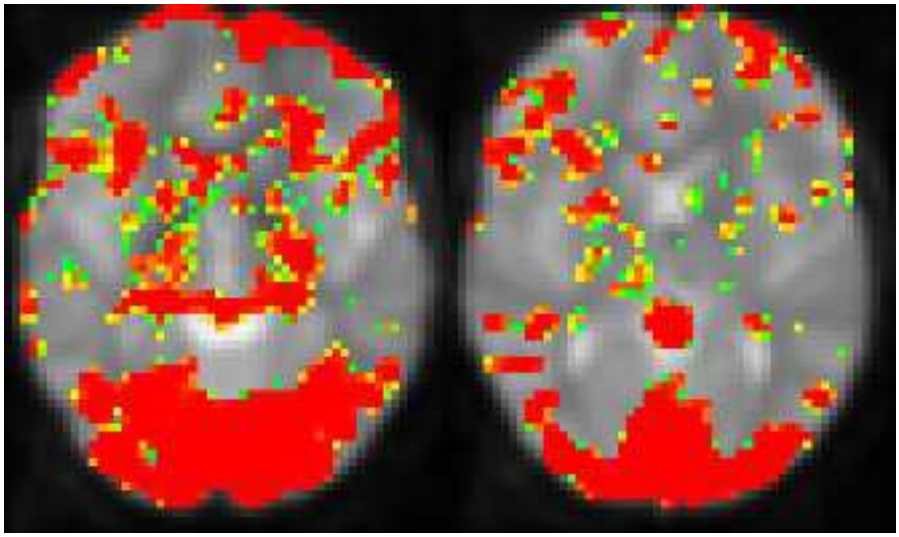


Figura 4.112: Método de coeficiente de correlação (dados não filtrados, sem HRF e com atraso igual a 1).

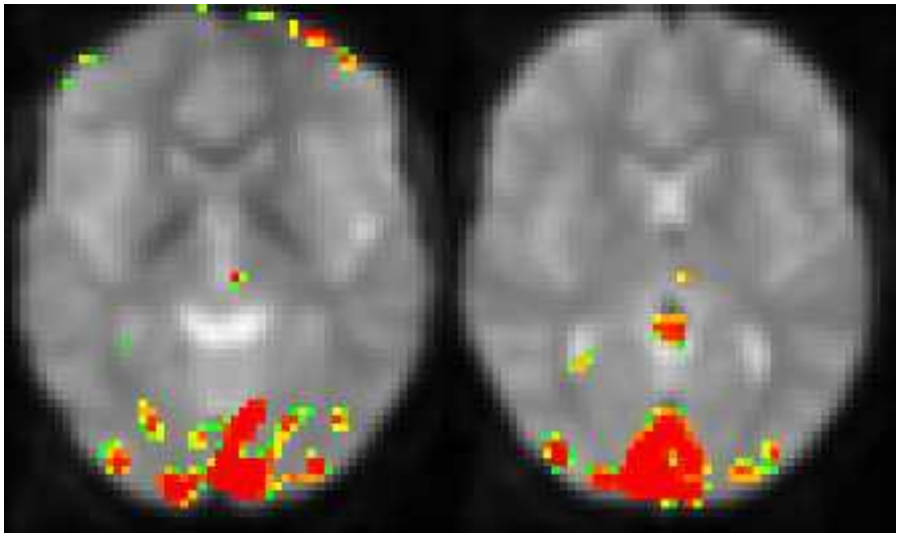


Figura 4.113: Método dos picos de Fourier (dados não filtrados, sem HRF e com atraso igual a 1).

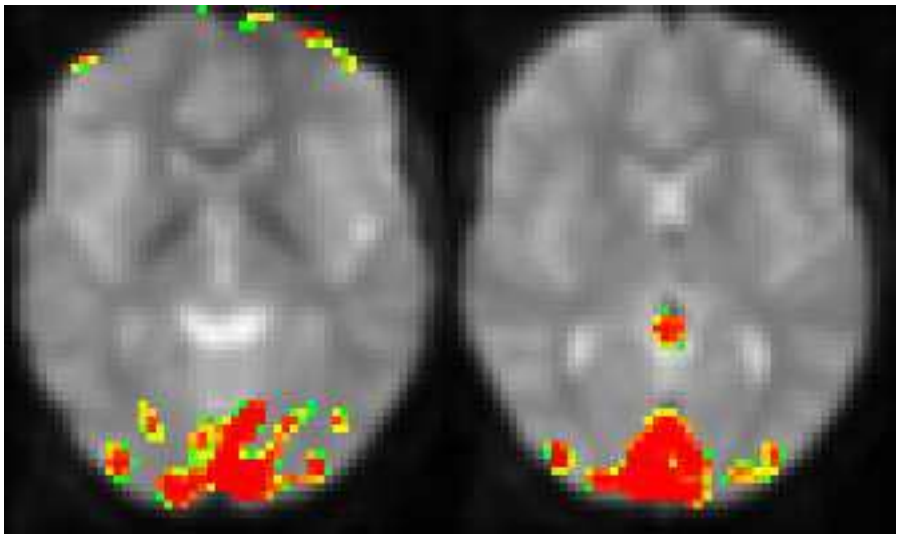


Figura 4.114: Método do modelo linear generalizado (dados não filtrados, sem HRF e com atraso igual a 1).

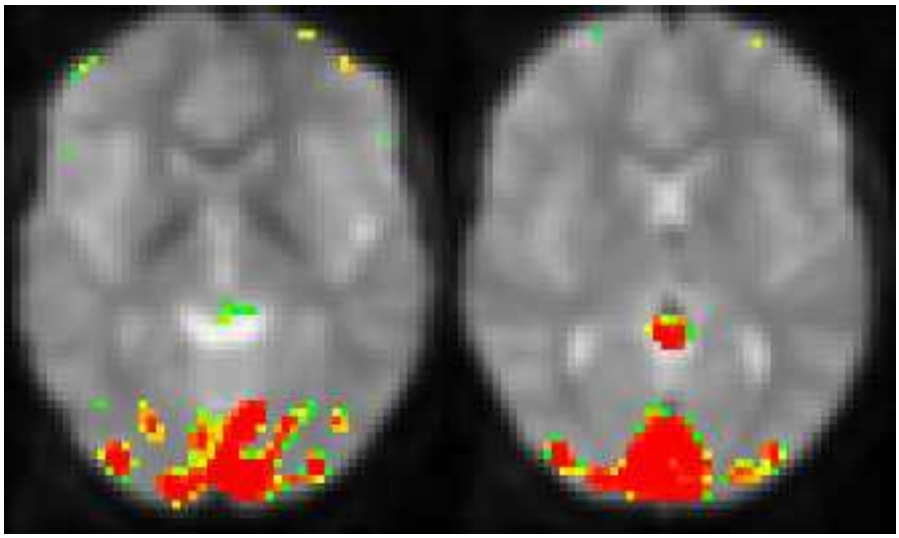


Figura 4.115: Método de amplitude (dados não filtrados, sem HRF e com atraso igual a 1).

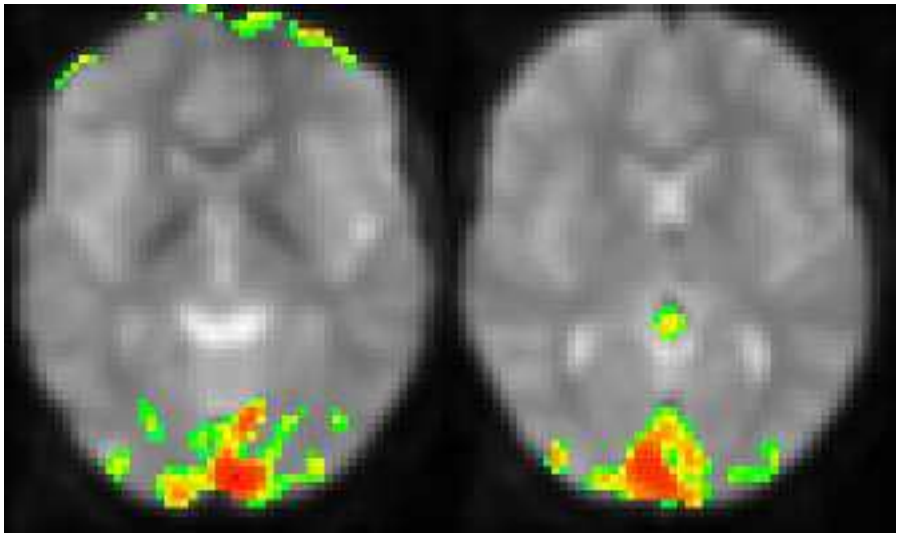


Figura 4.116: Método de sobreposição de ordem 0 (dados não filtrados, sem HRF e com atraso igual a 1).

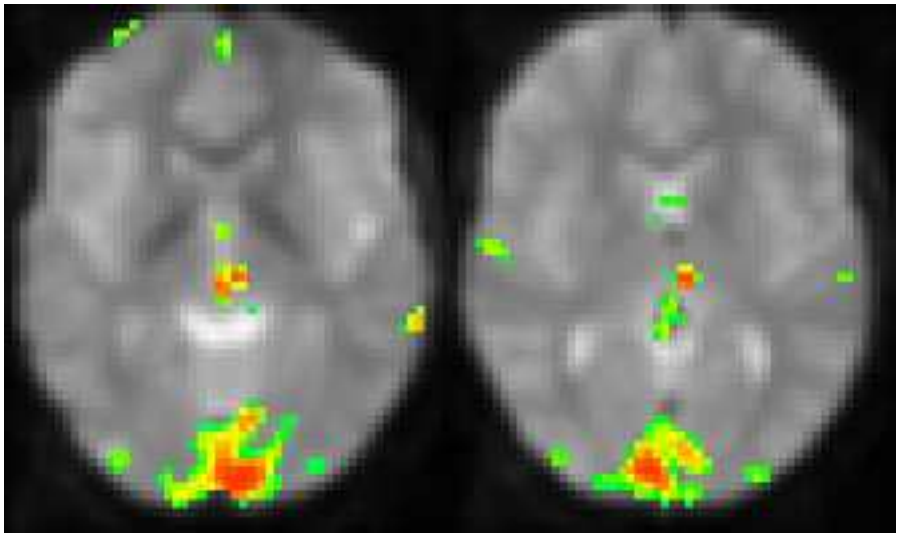


Figura 4.117: Método de sobreposição de ordem 1 (dados não filtrados, sem HRF e com atraso igual a 1).

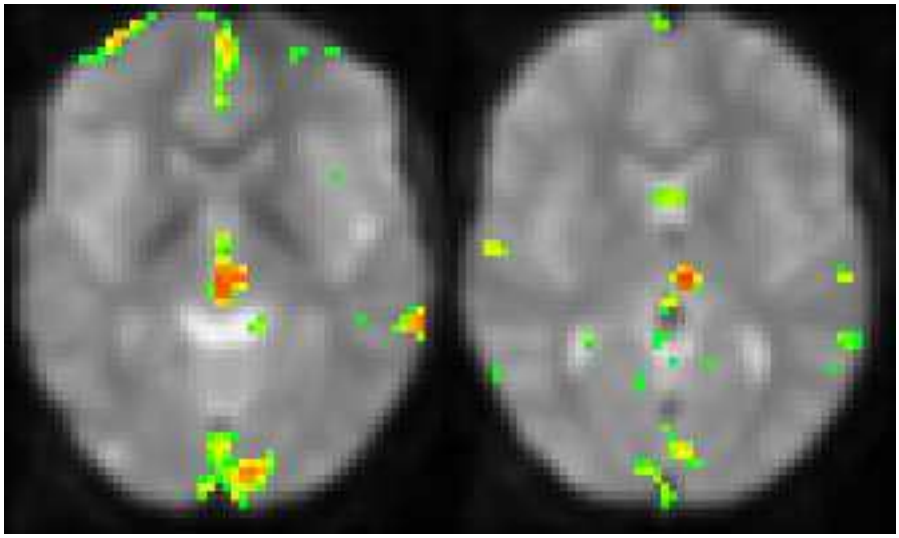


Figura 4.118: Método de sobreposição de ordem 2 (dados não filtrados, sem HRF e com atraso igual a 1).

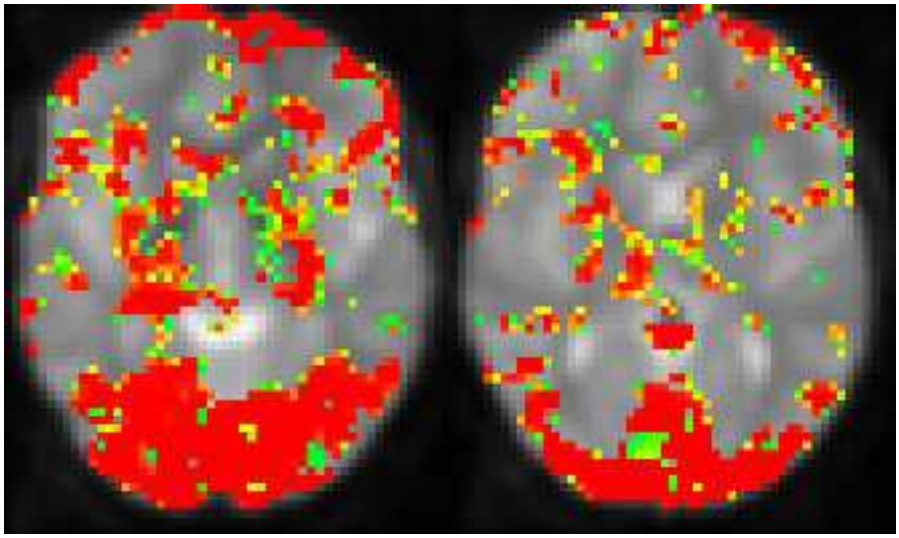


Figura 4.119: Método de t de Student (dados não filtrados, sem HRF e com atraso igual a 2).

4.8.4.4 Não filtrados, sem HRF e com atraso igual a 2

Apresentamos nesta secção as imagens de activação produzidas pelo programa *Cérebro* para cada um dos métodos com dados não filtrados e analisados com uma sequência de modelo em bloco com atraso igual a dois (figuras 4.119, 4.120, 4.121, 4.122, 4.123, 4.124, 4.125 e 4.126).

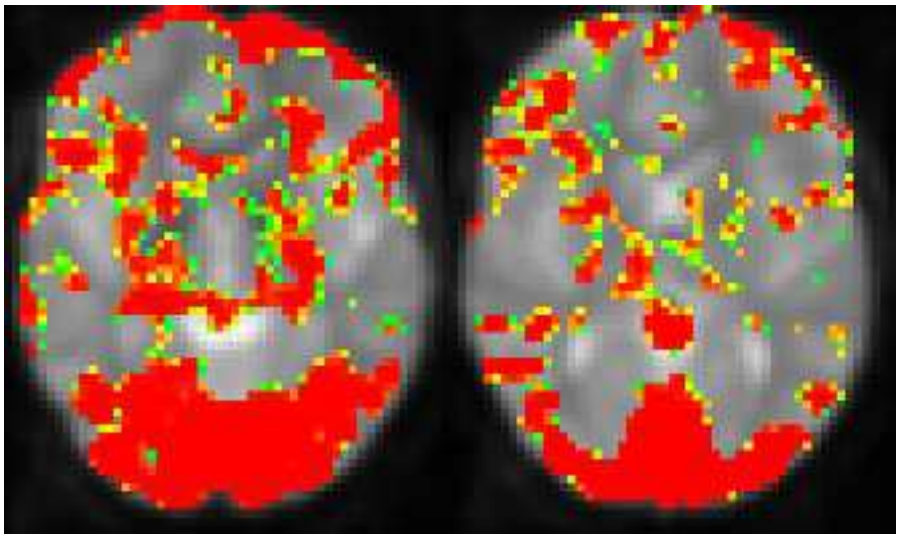


Figura 4.120: Método de coeficiente de correlação (dados não filtrados, sem HRF e com atraso igual a 2).

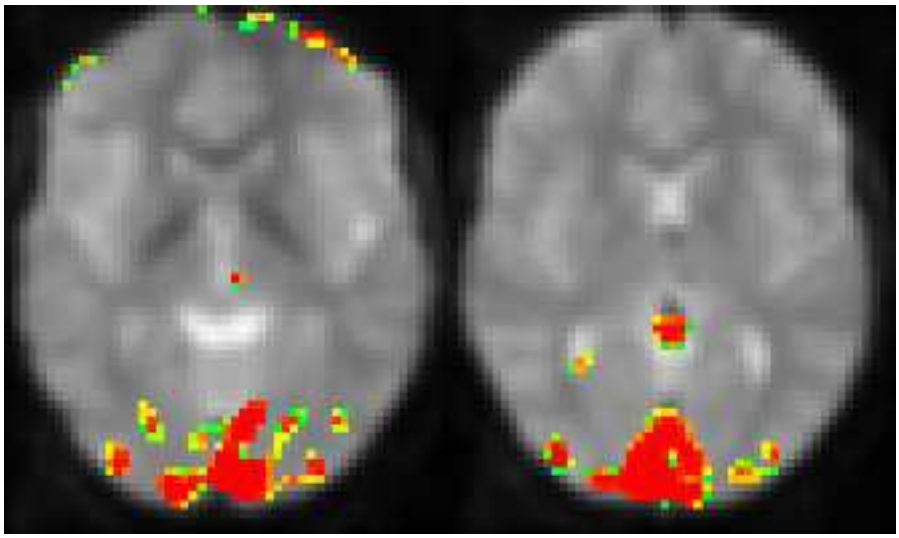


Figura 4.121: Método dos picos de Fourier (dados não filtrados, sem HRF e com atraso igual a 2).

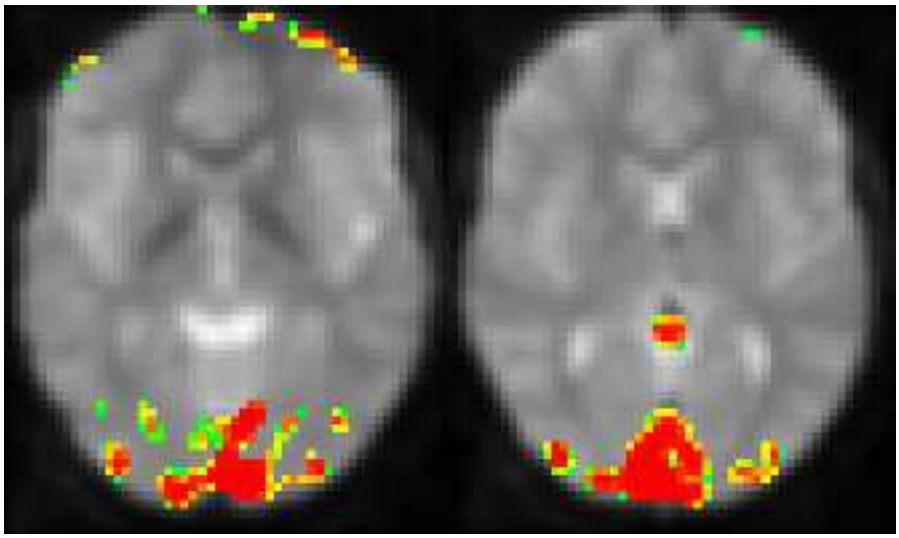


Figura 4.122: Método do modelo linear generalizado (dados não filtrados, sem HRF e com atraso igual a 2).

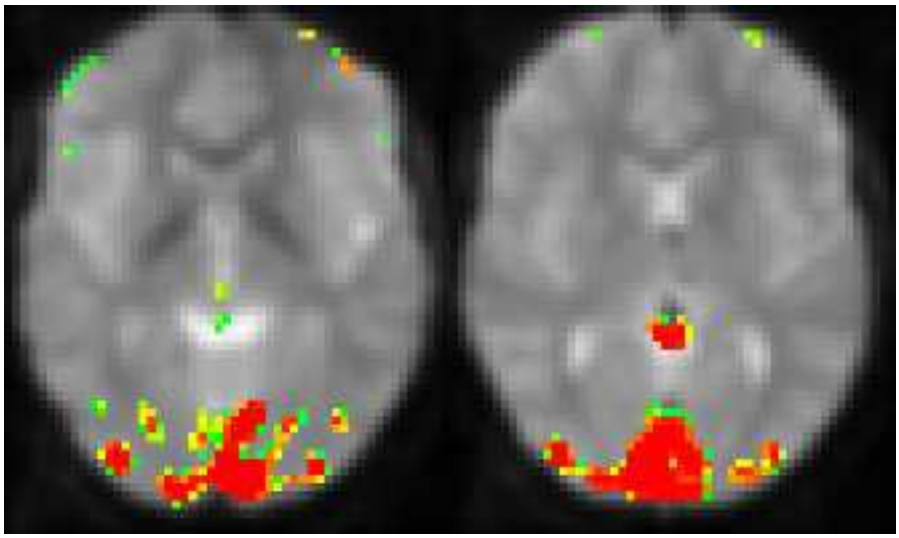


Figura 4.123: Método de amplitude (dados não filtrados, sem HRF e com atraso igual a 2).

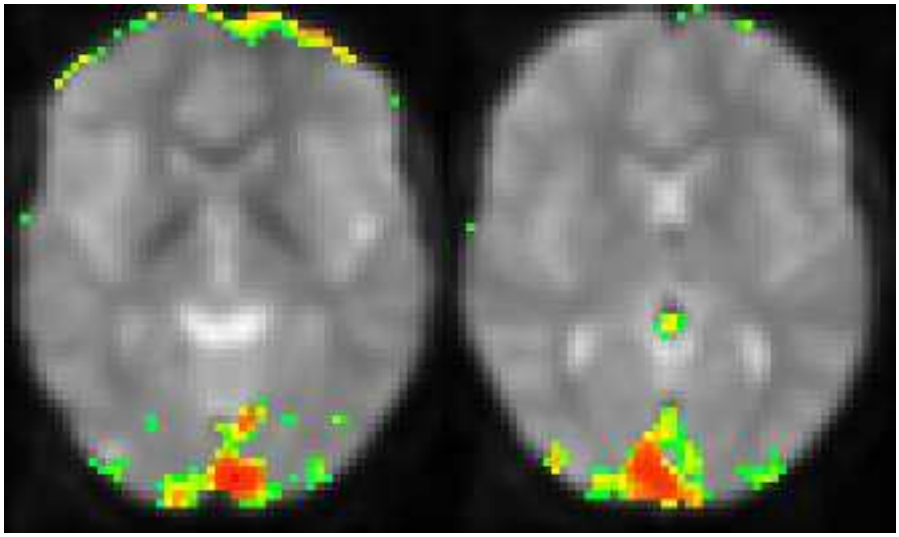


Figura 4.124: Método de sobreposição de ordem 0 (dados não filtrados, sem HRF e com atraso igual a 2).

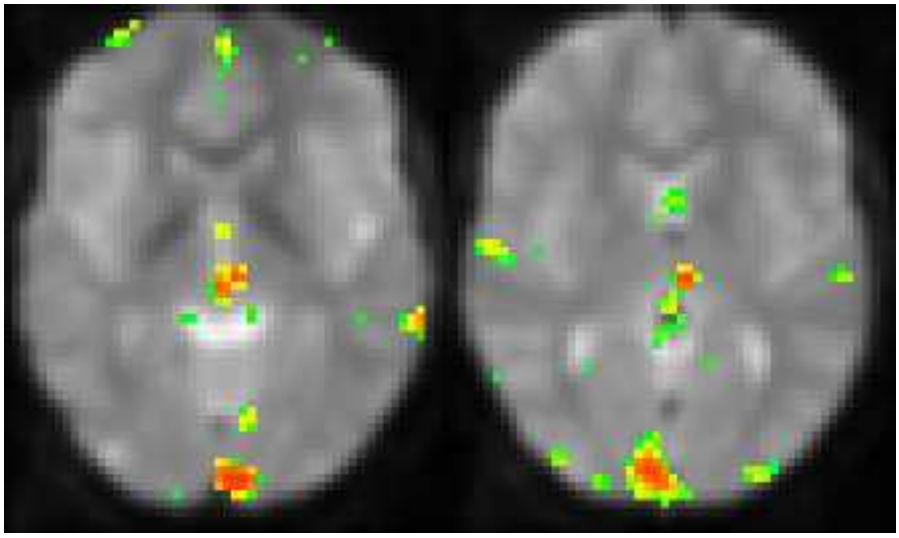


Figura 4.125: Método de sobreposição de ordem 1 (dados não filtrados, sem HRF e com atraso igual a 2).

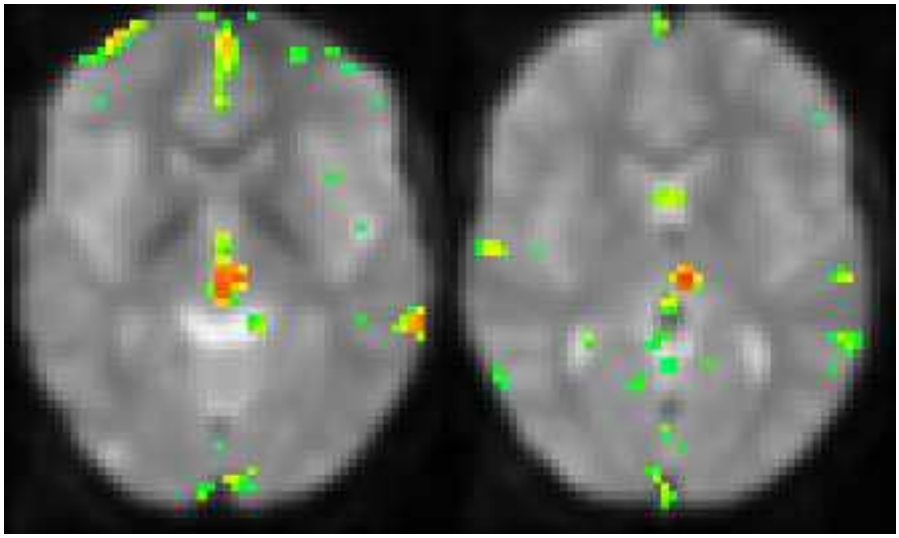


Figura 4.126: Método de sobreposição de ordem 2 (dados não filtrados, sem HRF e com atraso igual a 2).

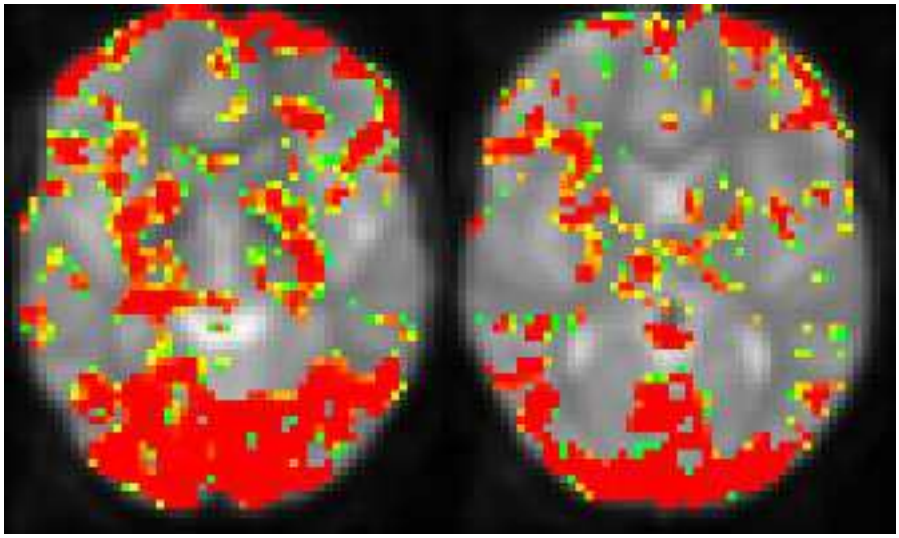


Figura 4.127: Método de t de Student (dados não filtrados, sem HRF e com atraso igual a 3).

4.8.4.5 Não filtrados, sem HRF e com atraso igual a 3

Apresentamos nesta secção as imagens de activação produzidas pelo programa *Cérebro* para cada um dos métodos com dados não filtrados e analisados com uma sequência de modelo em bloco com atraso igual a três (figuras 4.127, 4.128, 4.129, 4.130, 4.131, 4.132, 4.133 e 4.134).

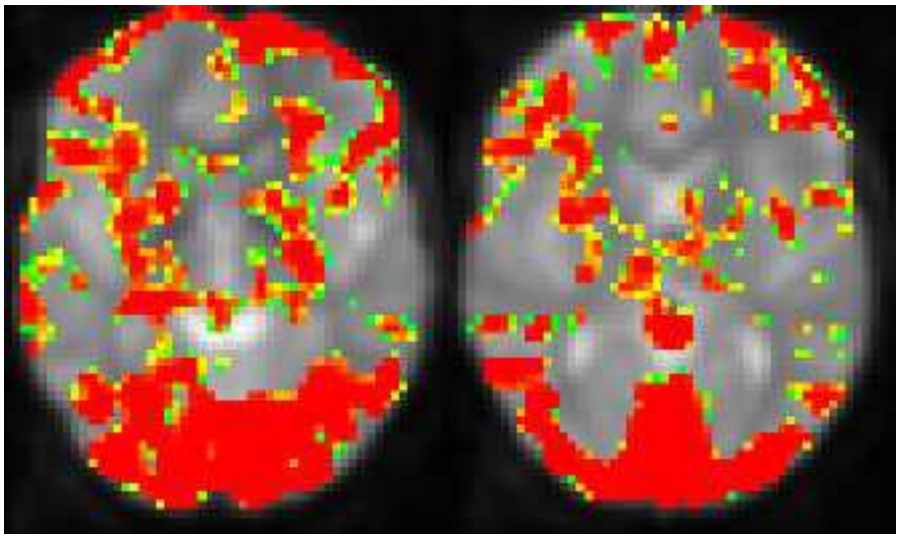


Figura 4.128: Método de coeficiente de correlação (dados não filtrados, sem HRF e com atraso igual a 3).

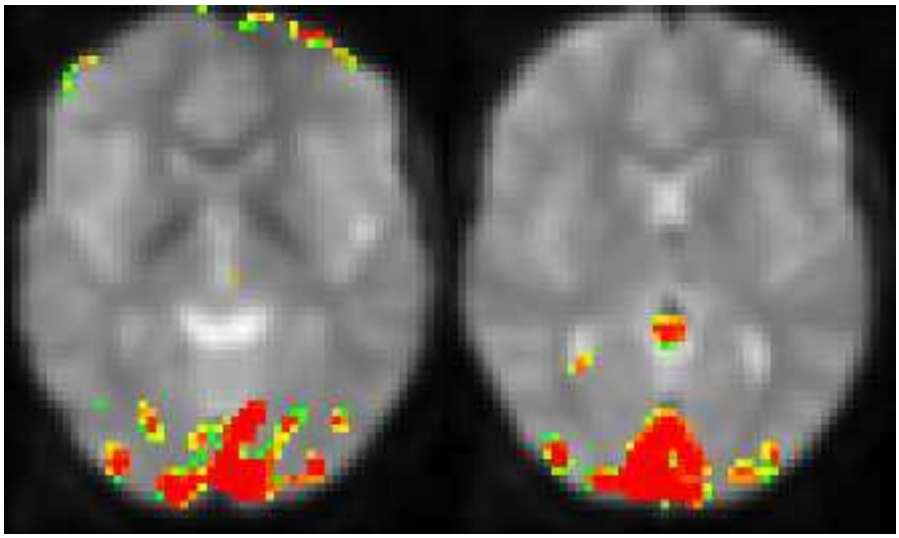


Figura 4.129: Método dos picos de Fourier (dados não filtrados, sem HRF e com atraso igual a 3).

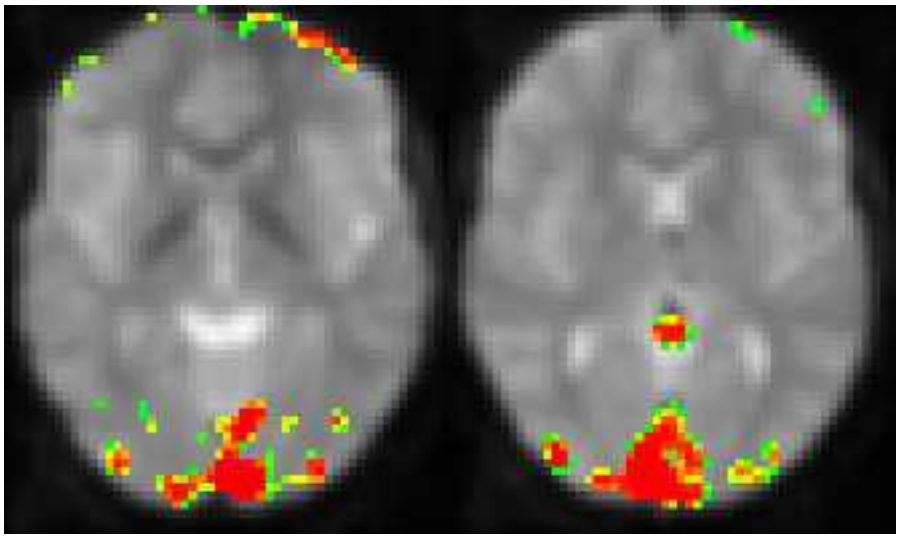


Figura 4.130: Método do modelo linear generalizado (dados não filtrados, sem HRF e com atraso igual a 3).

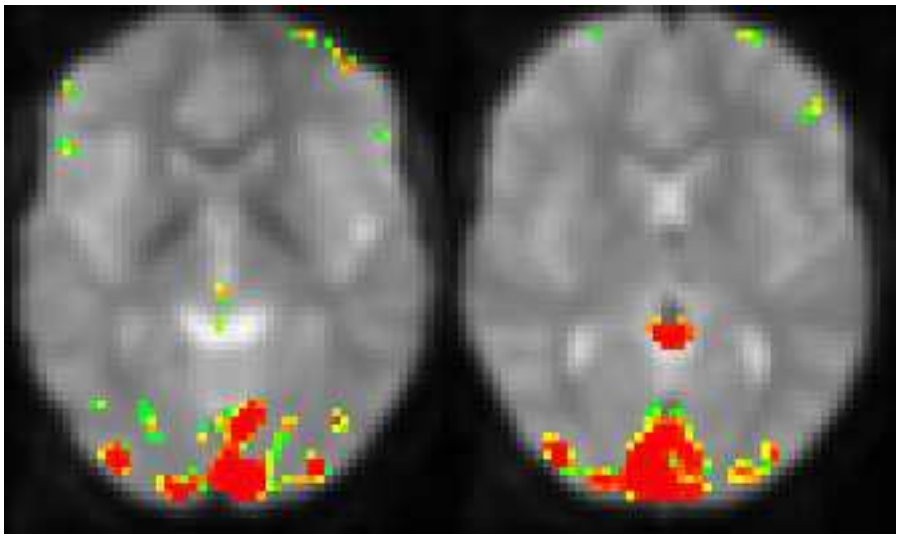


Figura 4.131: Método de amplitude (dados não filtrados, sem HRF e com atraso igual a 3).

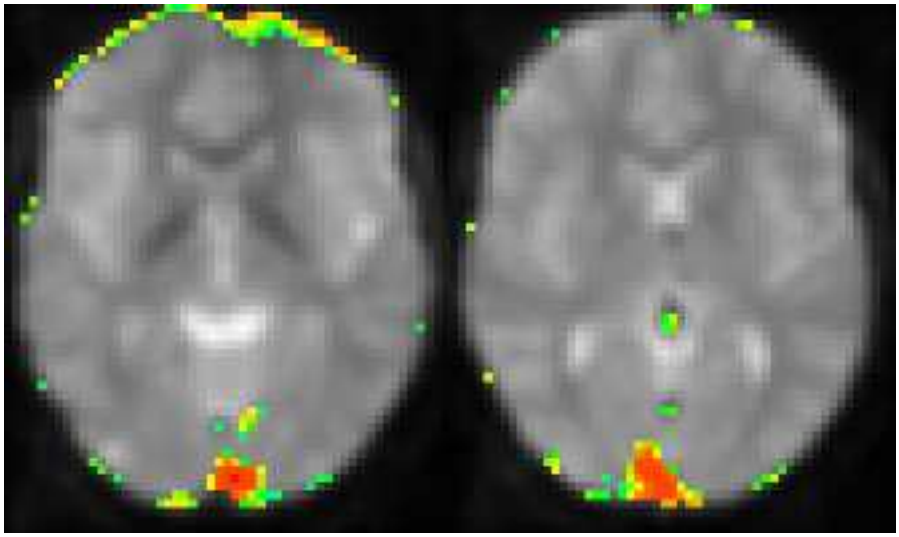


Figura 4.132: Método de sobreposição de ordem 0 (dados não filtrados, sem HRF e com atraso igual a 3).

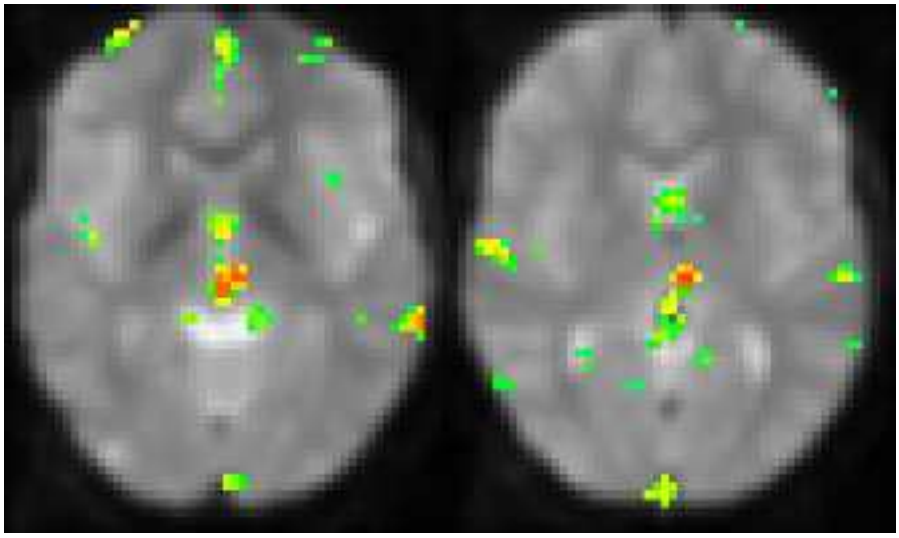


Figura 4.133: Método de sobreposição de ordem 1 (dados não filtrados, sem HRF e com atraso igual a 3).

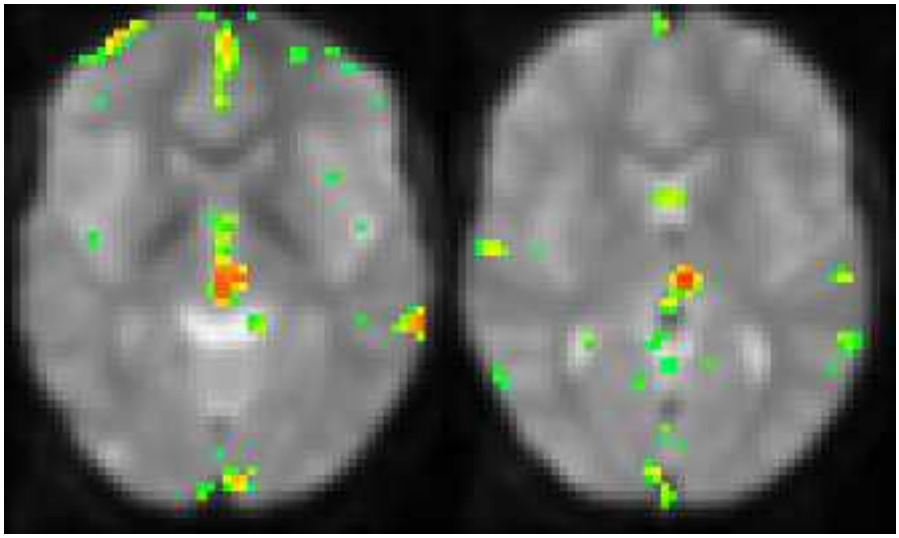


Figura 4.134: Método de sobreposição de ordem 2 (dados não filtrados, sem HRF e com atraso igual a 3).

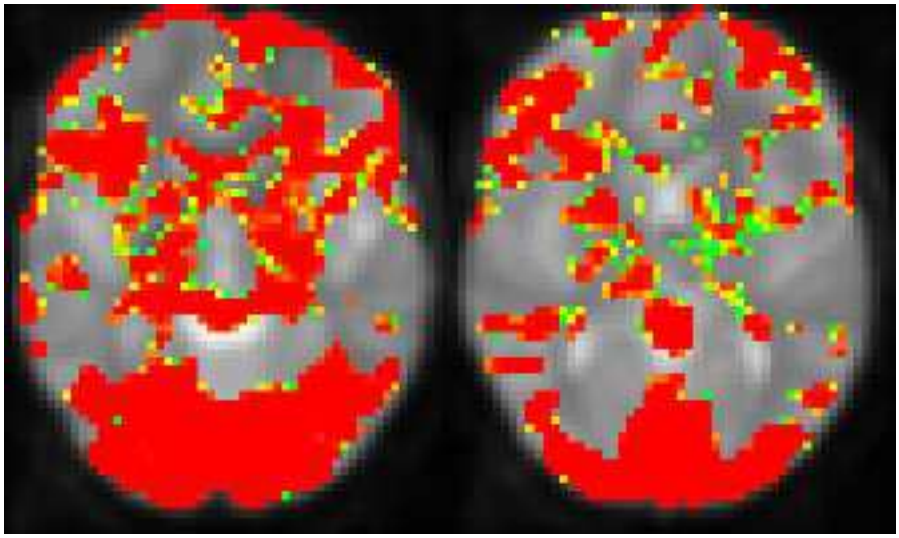


Figura 4.135: Método de coeficiente de correlação (dados filtrados e com HRF).

4.8.4.6 Filtrados e com HRF

Apresentamos nesta secção as imagens de activação produzidas pelo programa *Cérebro* para cada um dos métodos com dados filtrados e analisados com uma sequência de modelo com HRF (figuras 4.135, 4.136, 4.137, 4.138, 4.139 e 4.140).

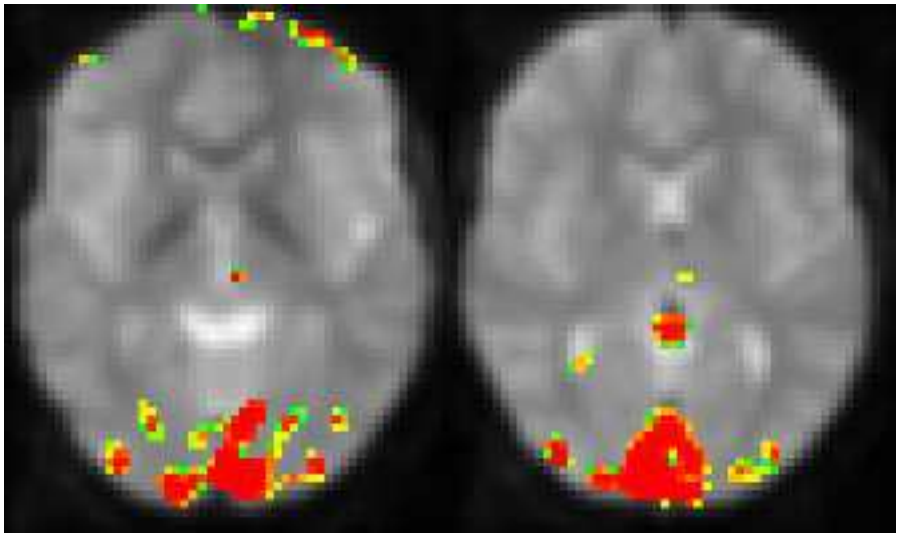


Figura 4.136: Método dos picos de Fourier (dados filtrados e com HRF).

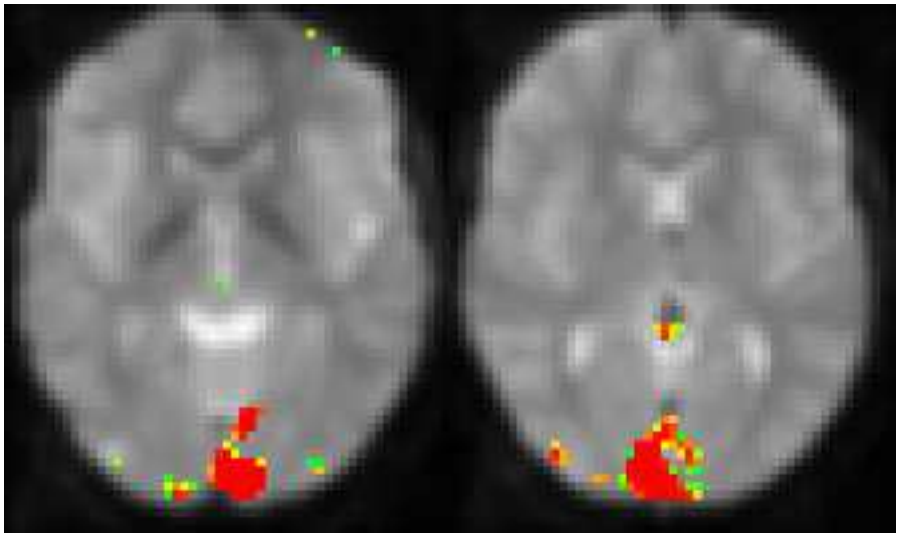


Figura 4.137: Método de amplitude (dados filtrados e com HRF).

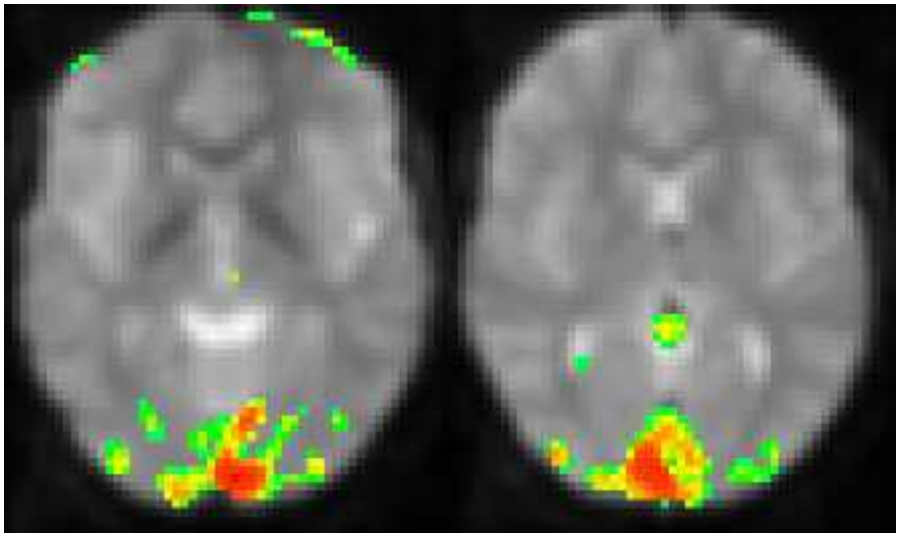


Figura 4.138: Método de sobreposição de ordem 0 (dados filtrados e com HRF).

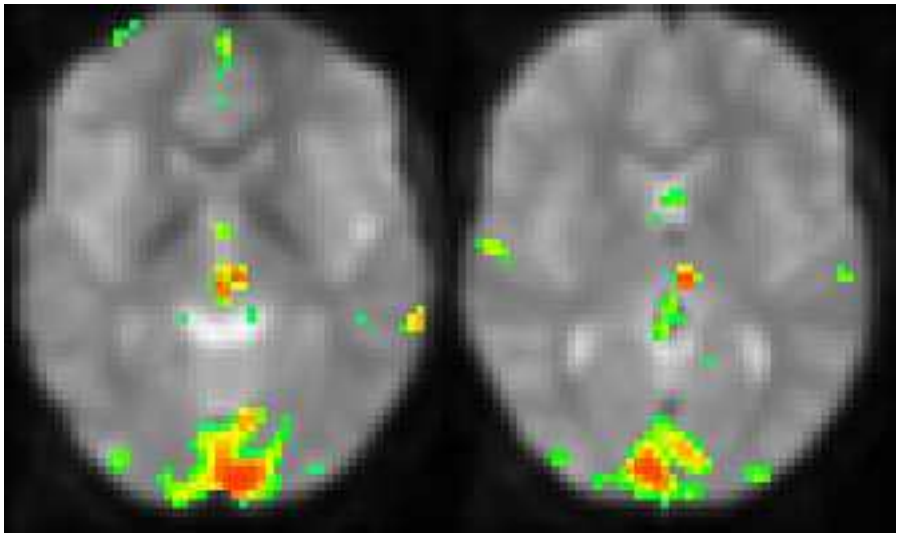


Figura 4.139: Método de sobreposição de ordem 1 (dados filtrados e com HRF).

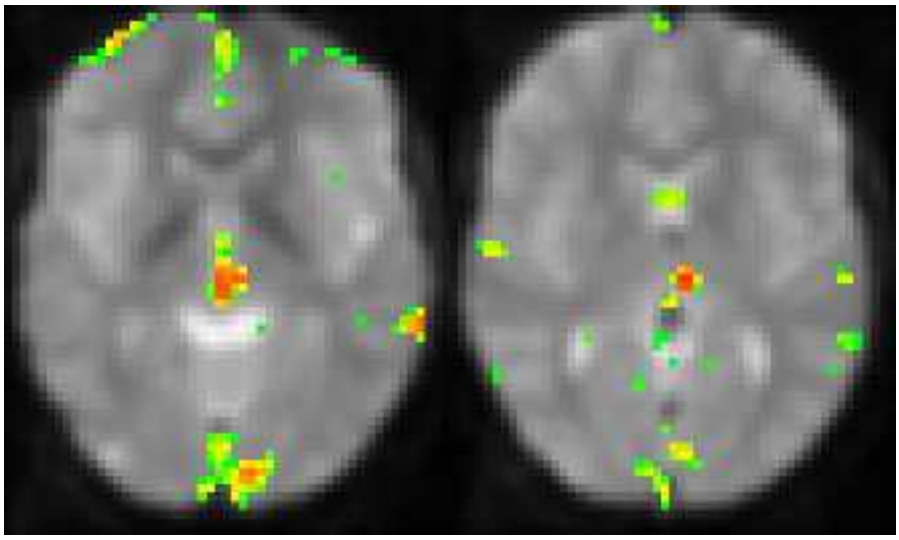


Figura 4.140: Método de sobreposição de ordem 2 (dados filtrados e com HRF).

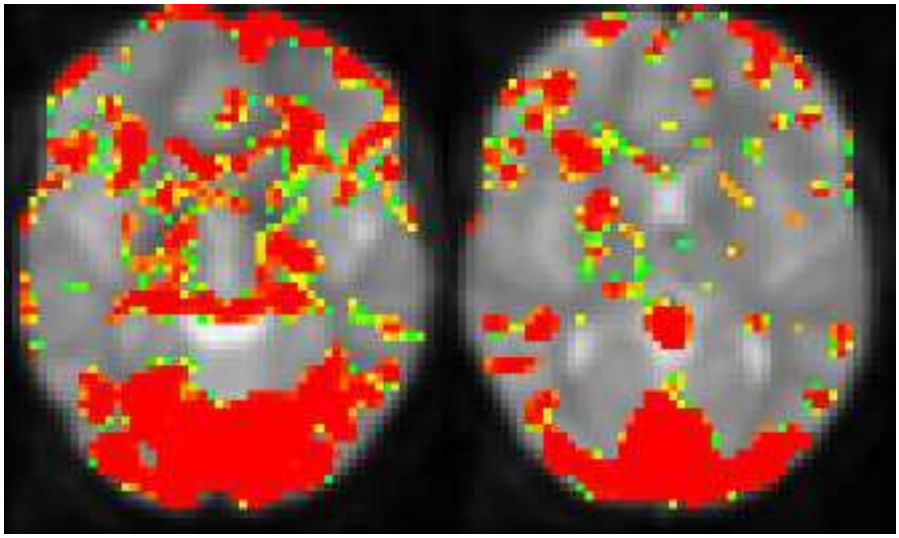


Figura 4.141: Método de coeficiente de correlação (dados não filtrados, com HRF e com atraso igual a 0).

4.8.4.7 Não filtrados, com HRF e com atraso igual a 0

Apresentamos nesta secção as imagens de activação produzidas pelo programa *Cérebro* para cada um dos métodos com dados não filtrados e analisados com uma sequência de modelo com HRF e atraso nulo (figuras 4.141, 4.142, 4.143, 4.144, 4.145 e 4.146).

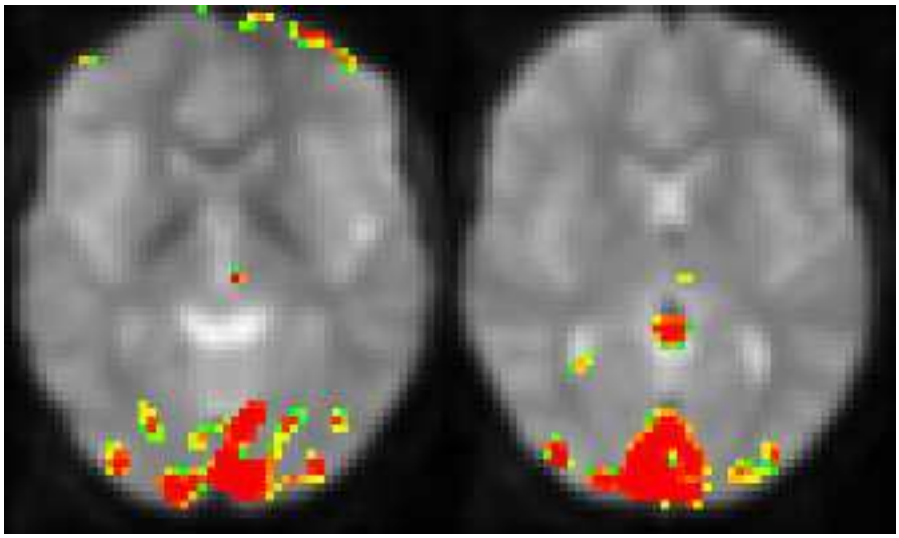


Figura 4.142: Método dos picos de Fourier (dados não filtrados, com HRF e com atraso igual a 0).

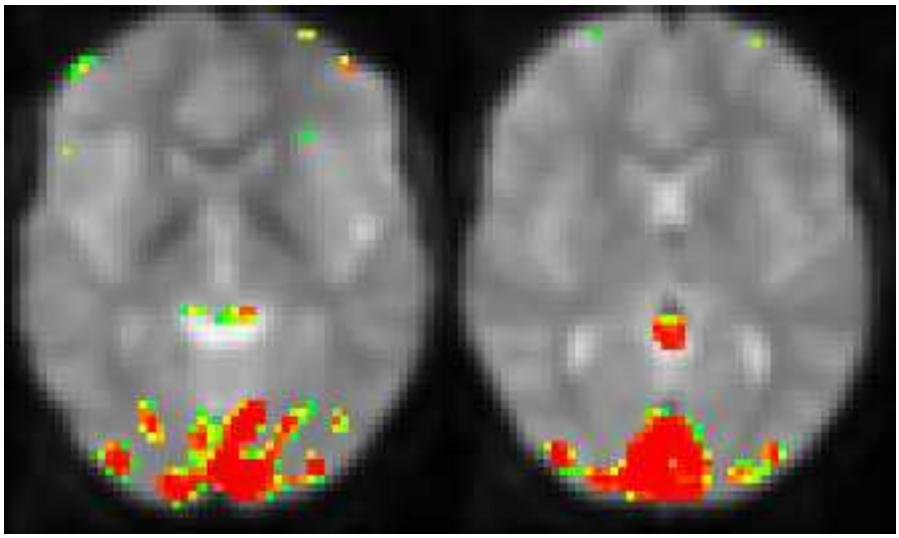


Figura 4.143: Método de amplitude (dados não filtrados, com HRF e com atraso igual a 0).

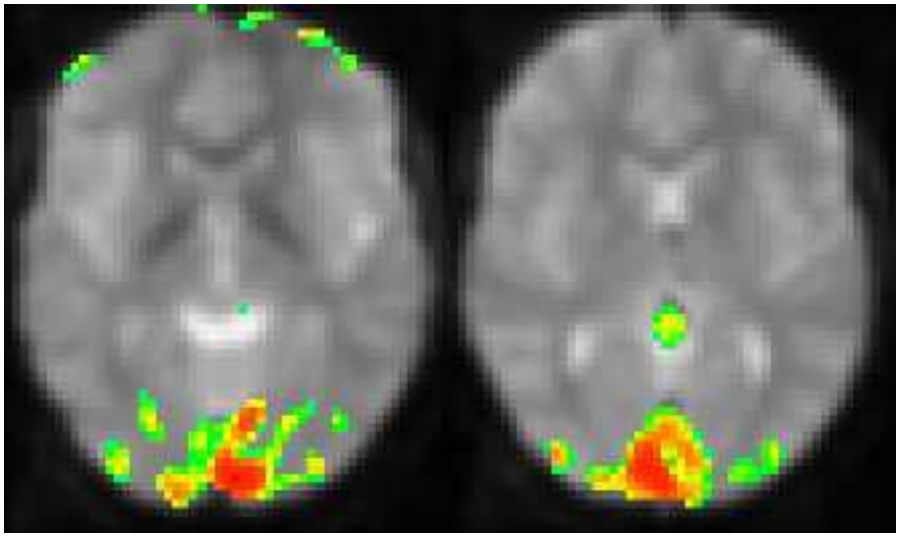


Figura 4.144: Método de sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 0).

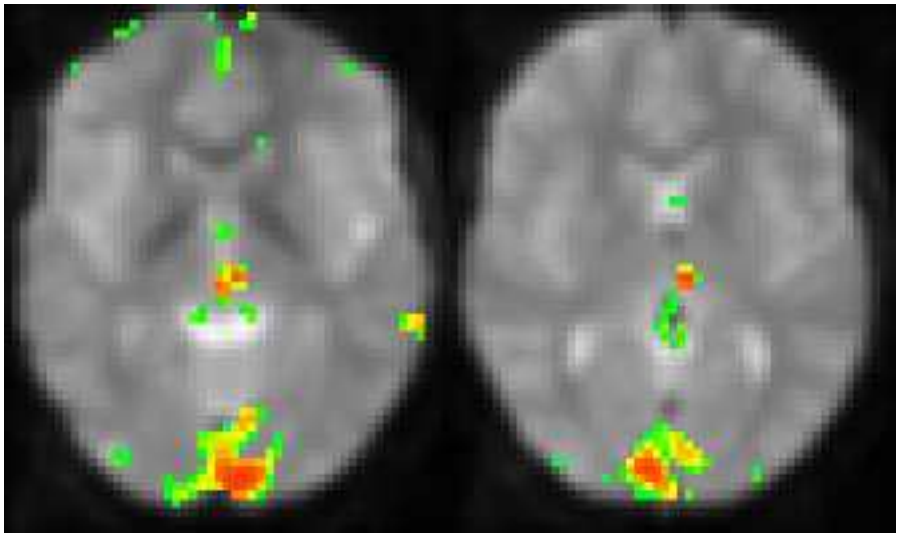


Figura 4.145: Método de sobreposição de ordem 1 (dados não filtrados, com HRF e com atraso igual a 0).

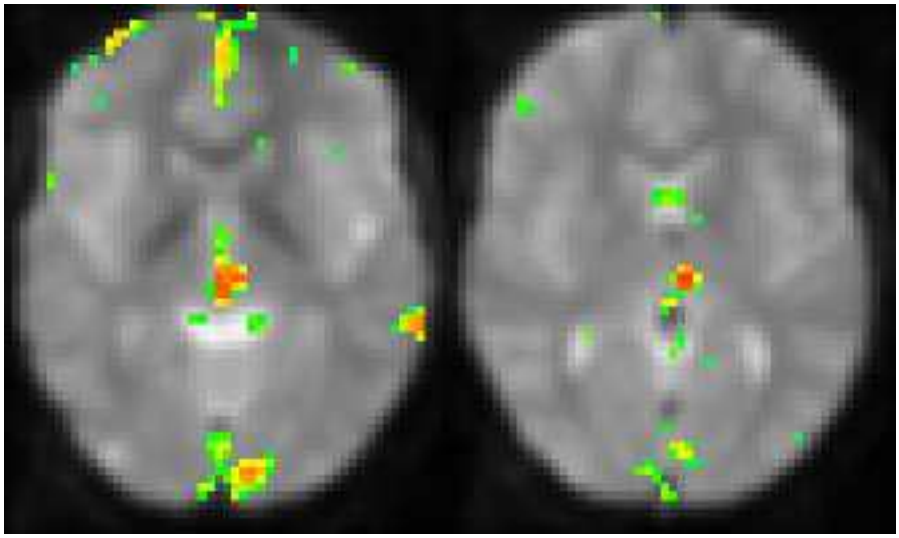


Figura 4.146: Método de sobreposição de ordem 2 (dados não filtrados, com HRF e com atraso igual a 0).

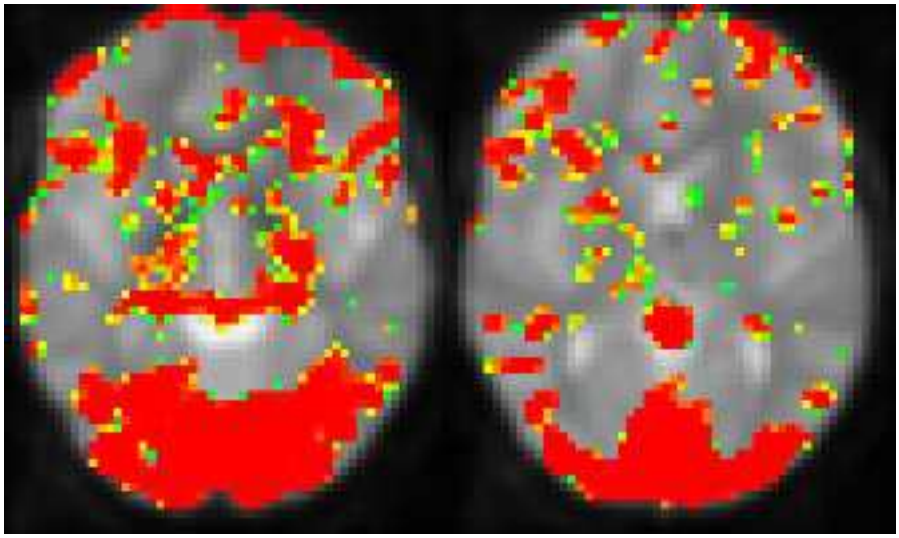


Figura 4.147: Método de coeficiente de correlação (dados não filtrados, com HRF e com atraso igual a 1).

4.8.4.8 Não filtrados, com HRF e com atraso igual a 1

Apresentamos nesta secção as imagens de activação produzidas pelo programa *Cérebro* para cada um dos métodos com dados não filtrados e analisados com uma sequência de modelo com HRF e atraso unitário (figuras 4.147, 4.148, 4.149, 4.150, 4.151 e 4.152).

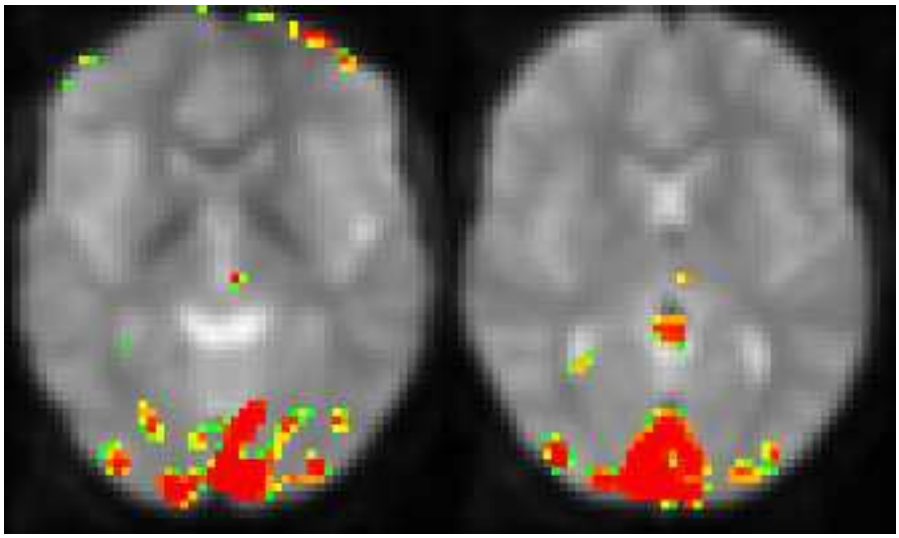


Figura 4.148: Método dos picos de Fourier (dados não filtrados, com HRF e com atraso igual a 1).

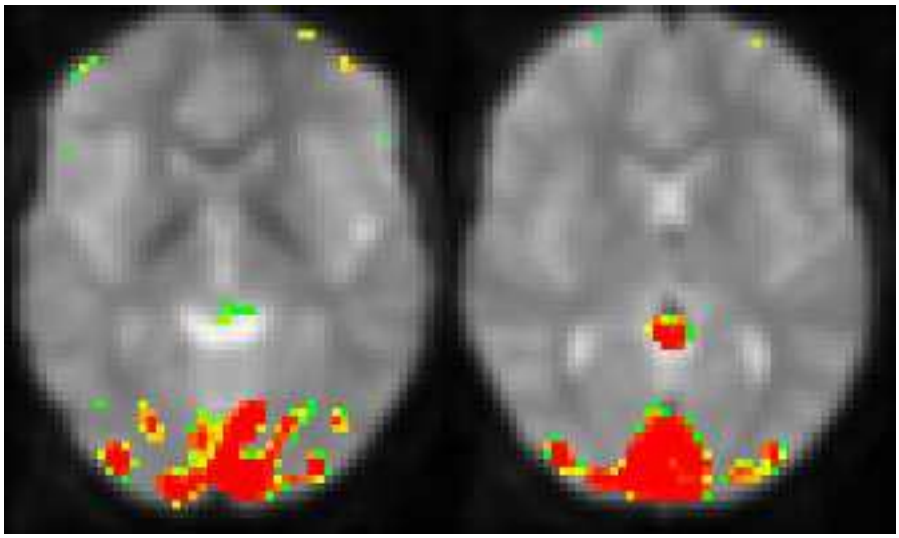


Figura 4.149: Método de amplitude (dados não filtrados, com HRF e com atraso igual a 1).

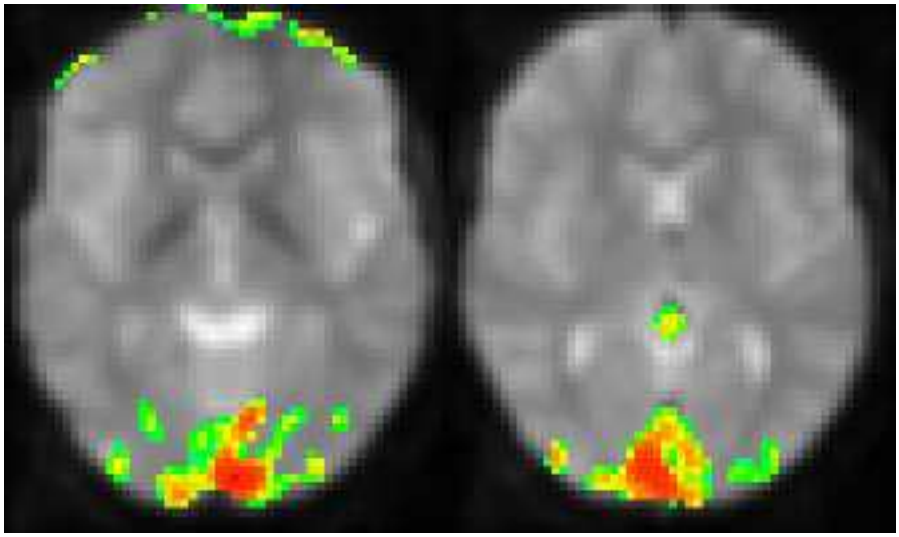


Figura 4.150: Método de sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 1).

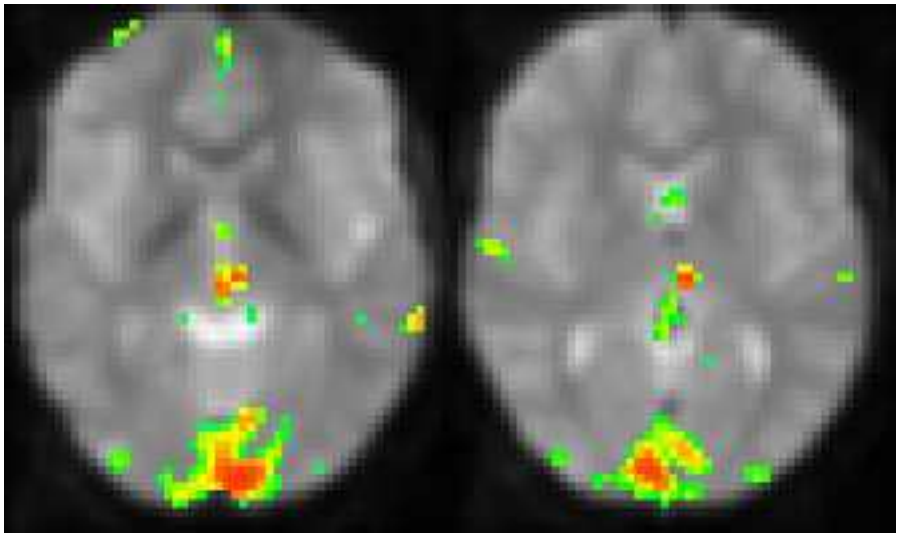


Figura 4.151: Método de sobreposição de ordem 1 (dados não filtrados, com HRF e com atraso igual a 1).

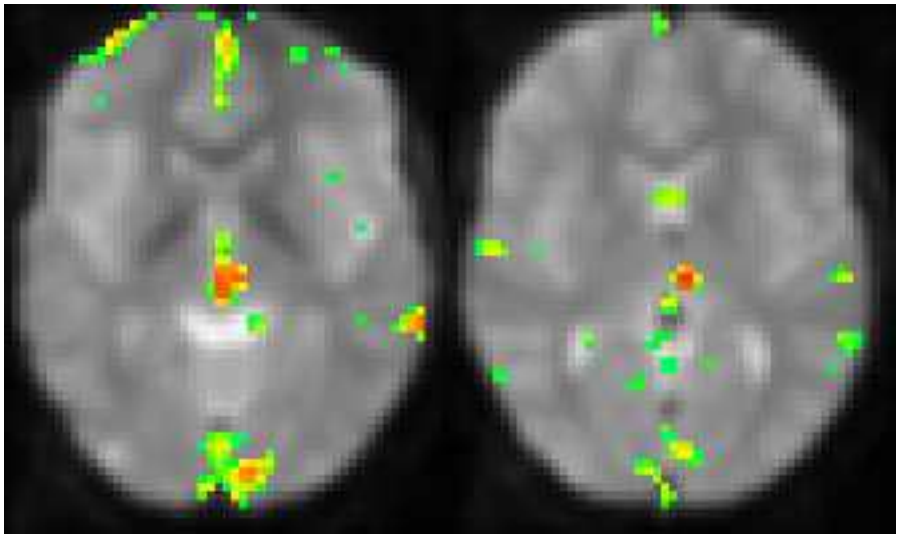


Figura 4.152: Método de sobreposição de ordem 2 (dados não filtrados, com HRF e com atraso igual a 1).

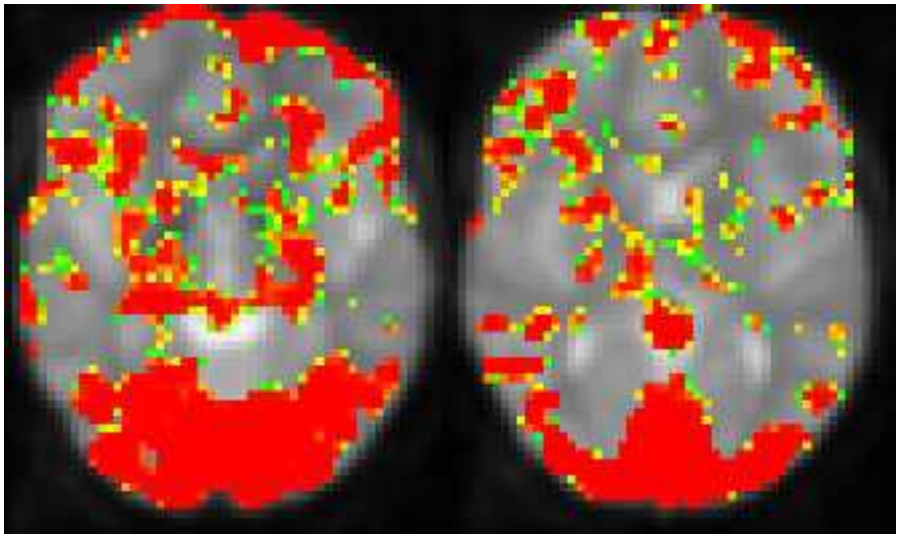


Figura 4.153: Método de coeficiente de correlação (dados não filtrados, com HRF e com atraso igual a 2).

4.8.4.9 Não filtrados, com HRF e com atraso igual a 2

Apresentamos nesta secção as imagens de activação produzidas pelo programa *Cérebro* para cada um dos métodos com dados não filtrados e analisados com uma sequência de modelo com HRF e atraso igual a dois (figuras 4.153, 4.154, 4.155, 4.156, 4.157 e 4.158).

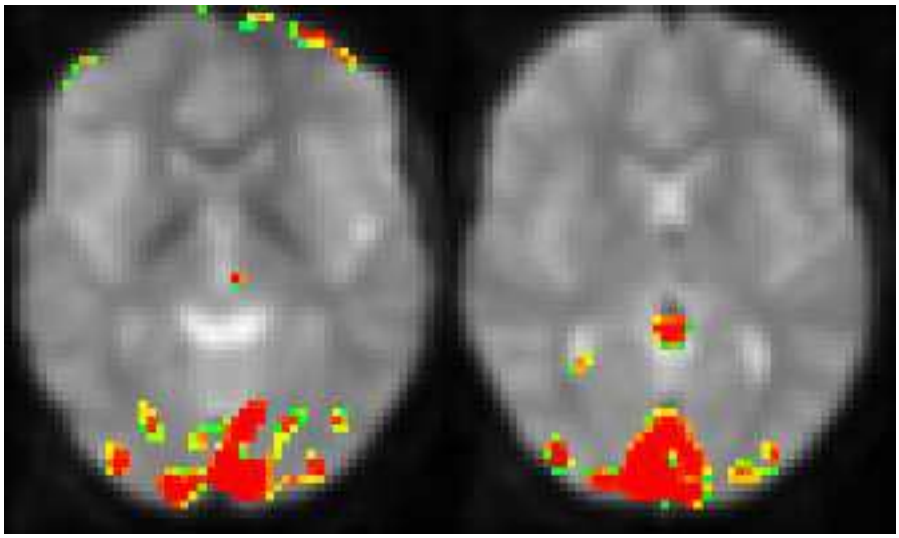


Figura 4.154: Método dos picos de Fourier (dados não filtrados, com HRF e com atraso igual a 2).

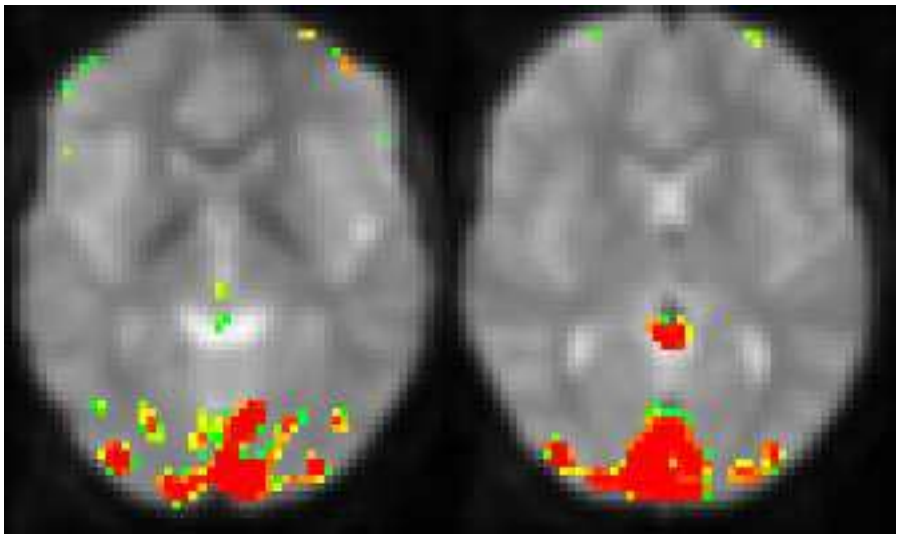


Figura 4.155: Método de amplitude (dados não filtrados, com HRF e com atraso igual a 2).

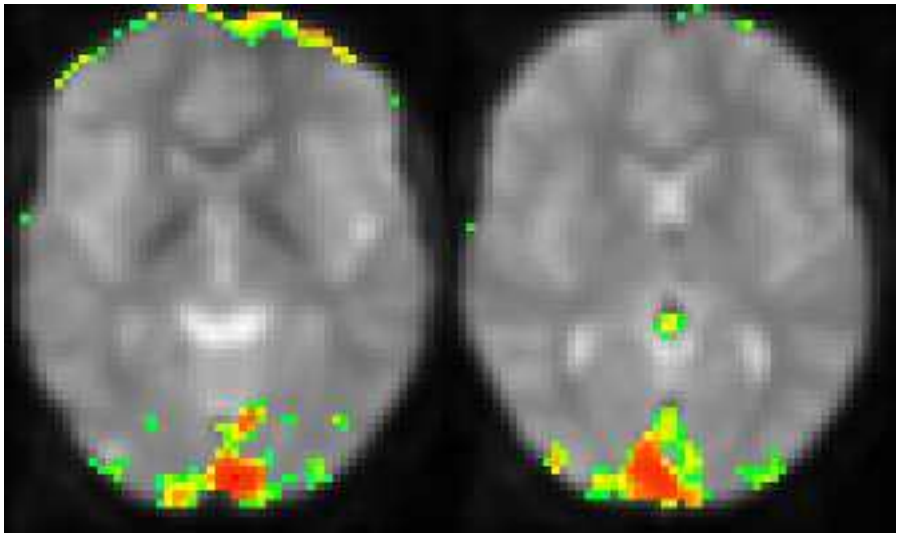


Figura 4.156: Método de sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 2).

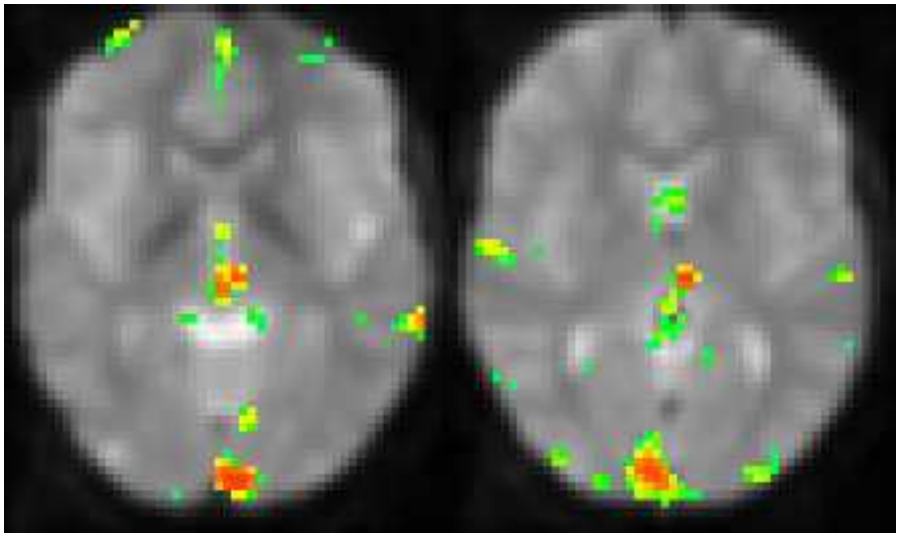


Figura 4.157: Método de sobreposição de ordem 1 (dados não filtrados, com HRF e com atraso igual a 2).

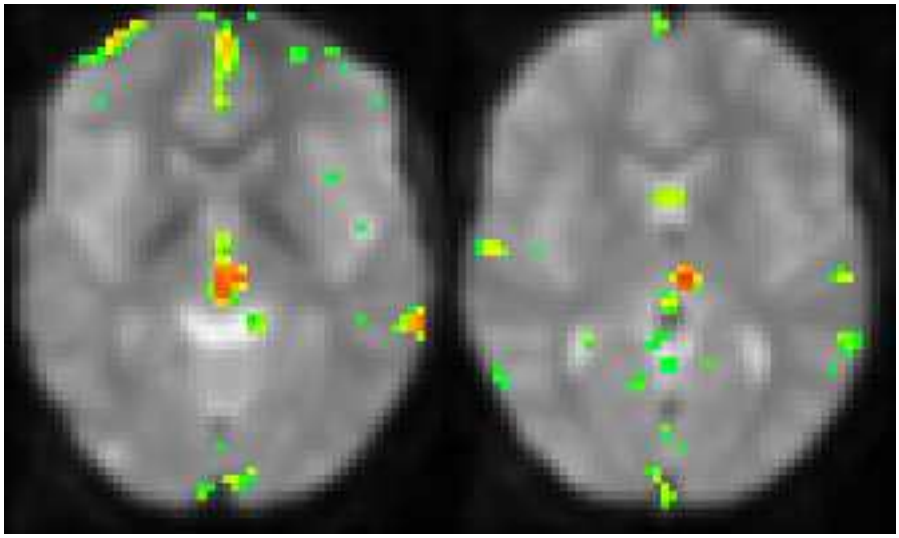


Figura 4.158: Método de sobreposição de ordem 2 (dados não filtrados, com HRF e com atraso igual a 2).

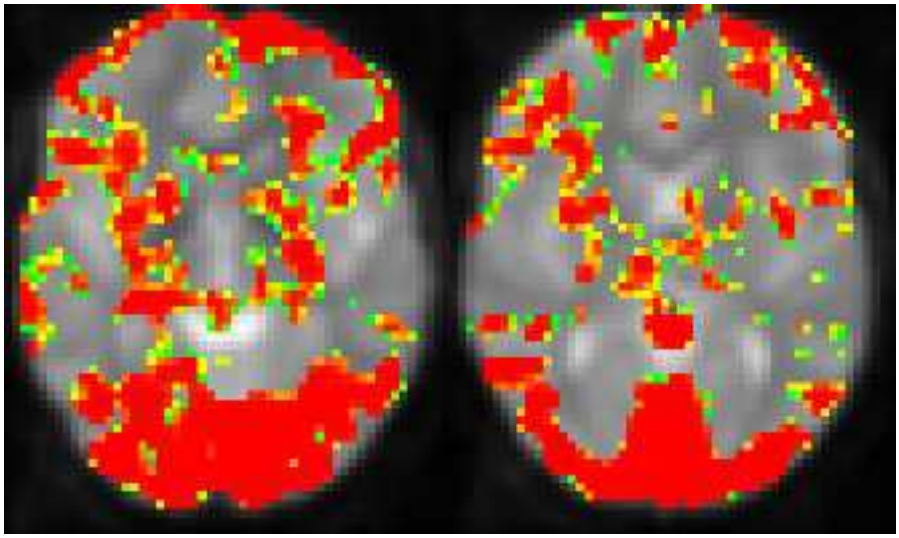


Figura 4.159: Método de coeficiente de correlação (dados não filtrados, com HRF e com atraso igual a 3).

4.8.4.10 Não filtrados, com HRF e com atraso igual a 3

Apresentamos nesta secção as imagens de activação produzidas pelo programa *Cérebro* para cada um dos métodos com dados não filtrados e analisados com uma sequência de modelo com HRF e atraso igual a três (figuras 4.159, 4.160, 4.161, 4.162, 4.163 e 4.164).

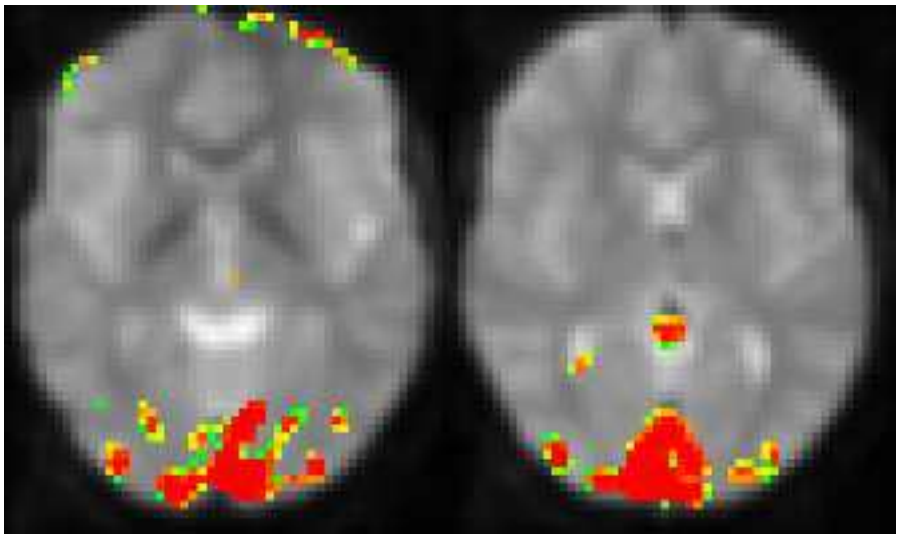


Figura 4.160: Método dos picos de Fourier (dados não filtrados, com HRF e com atraso igual a 3).

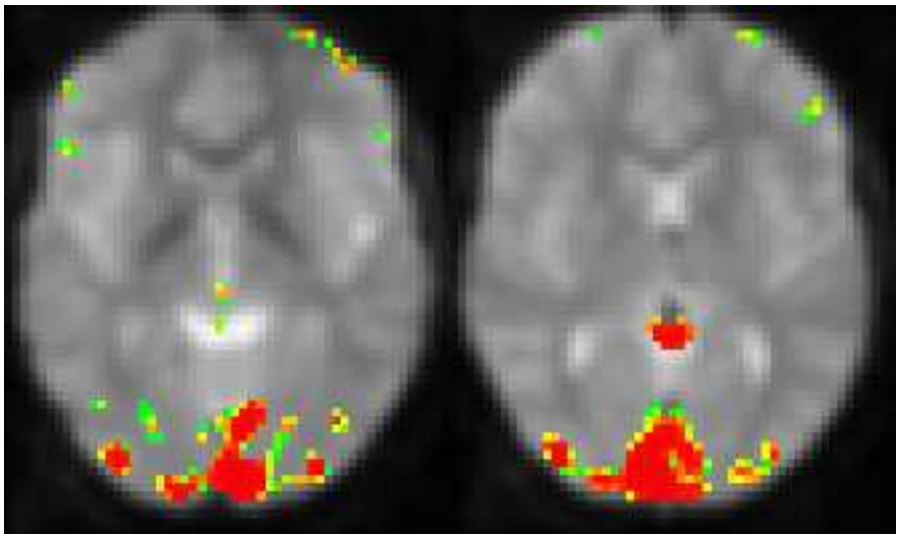


Figura 4.161: Método de amplitude (dados não filtrados, com HRF e com atraso igual a 3).

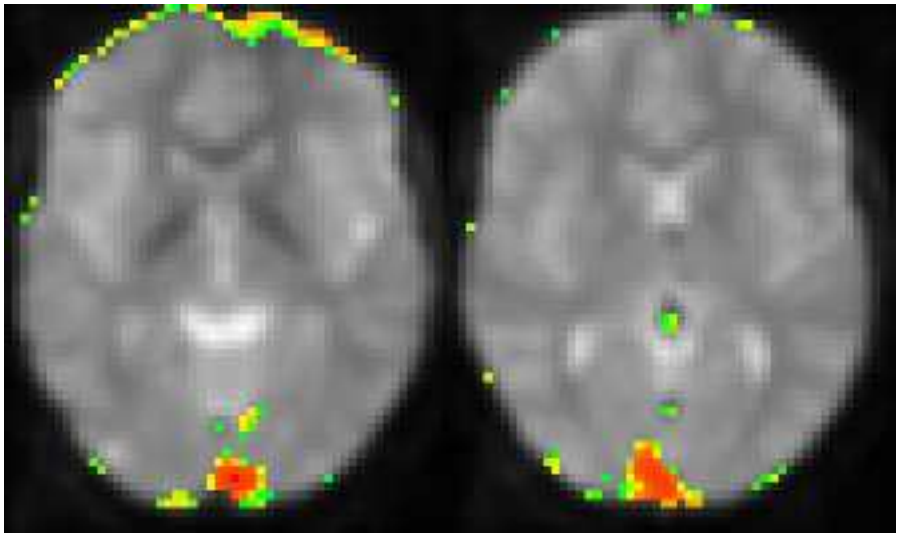


Figura 4.162: Método de sobreposição de ordem 0 (dados não filtrados, com HRF e com atraso igual a 3).

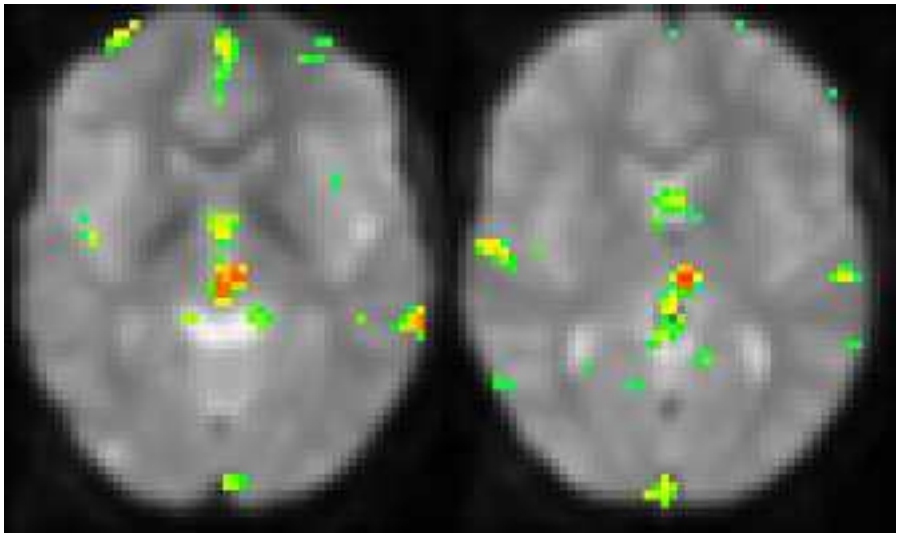


Figura 4.163: Método de sobreposição de ordem 1 (dados não filtrados, com HRF e com atraso igual a 3).

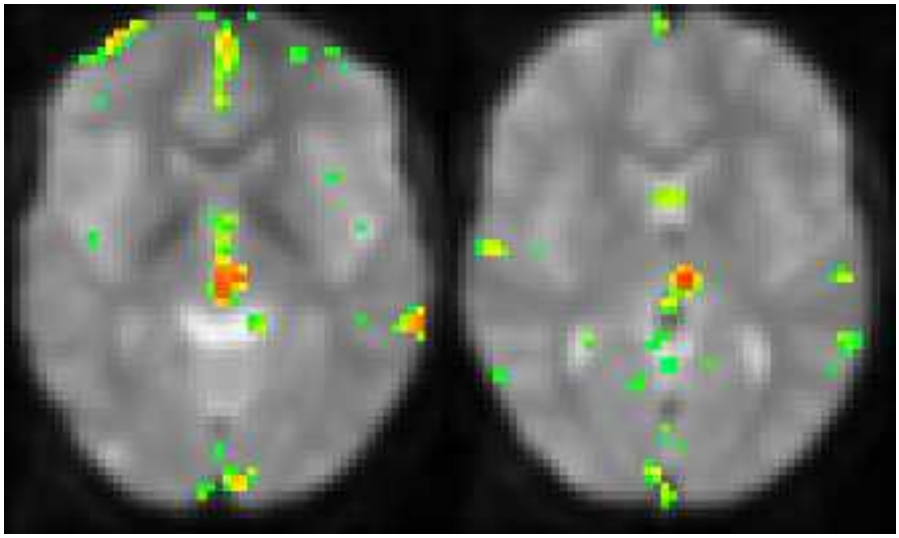


Figura 4.164: Método de sobreposição de ordem 2 (dados não filtrados, com HRF e com atraso igual a 3).

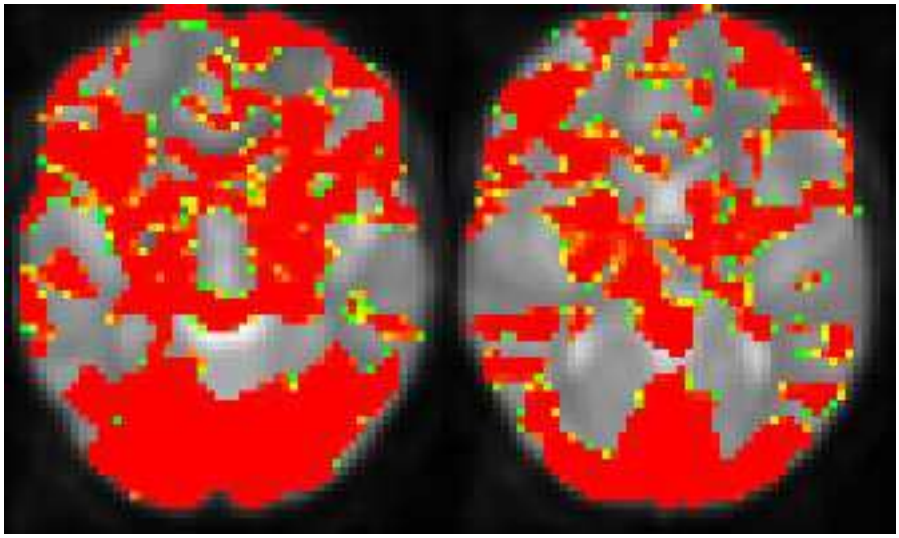


Figura 4.165: Método de coeficiente de correlação com a sequência de (27,6,17) para dados filtrados.

4.8.4.11 Filtrados e com conectividade funcional com (27,6,17)

Apresentamos nesta secção as imagens de activação produzidas pelo programa *Cérebro* para cada um dos métodos com dados filtrados e analisados com uma sequência de modelo igual à do voxel (27,6,17) (figuras 4.165, 4.166, 4.167, 4.168, 4.169 e 4.170).

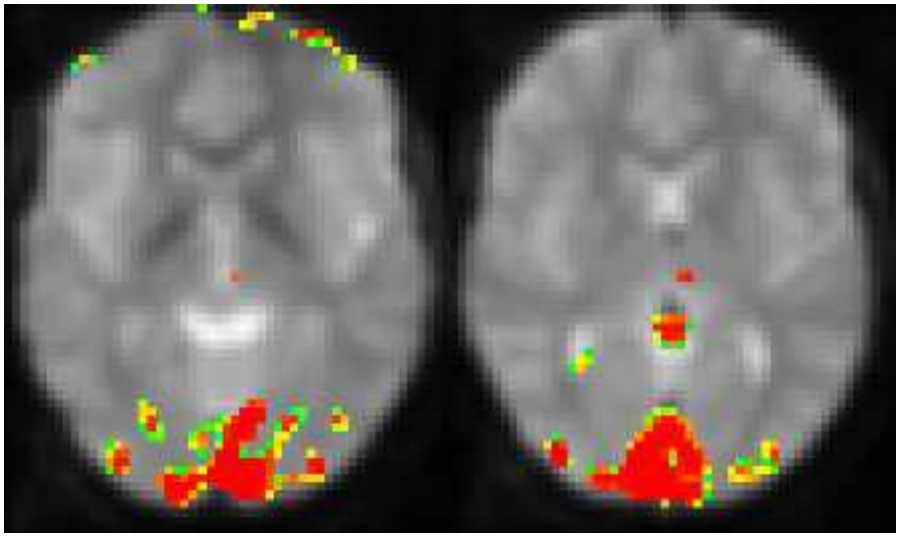


Figura 4.166: Método dos picos de Fourier com a sequência de (27,6,17) para dados filtrados.

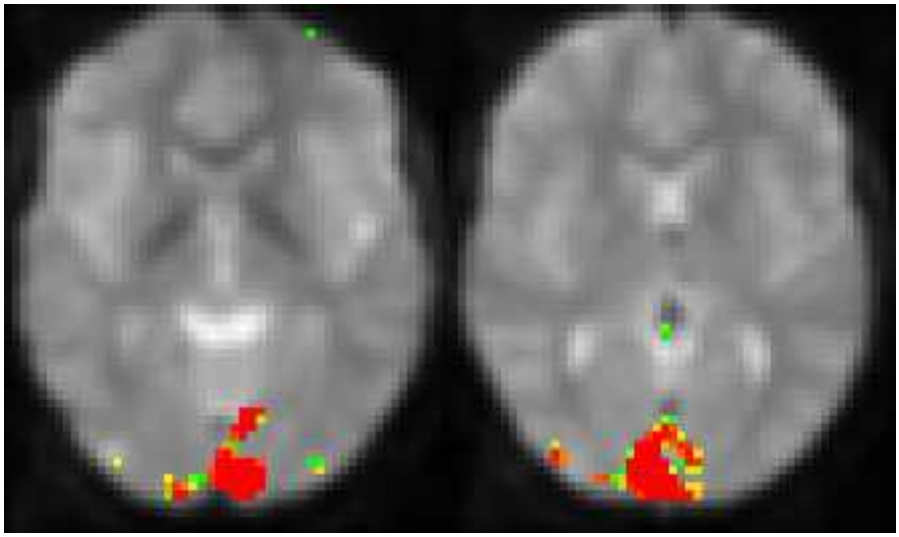


Figura 4.167: Método de amplitude com a sequência de (27,6,17) para dados filtrados.

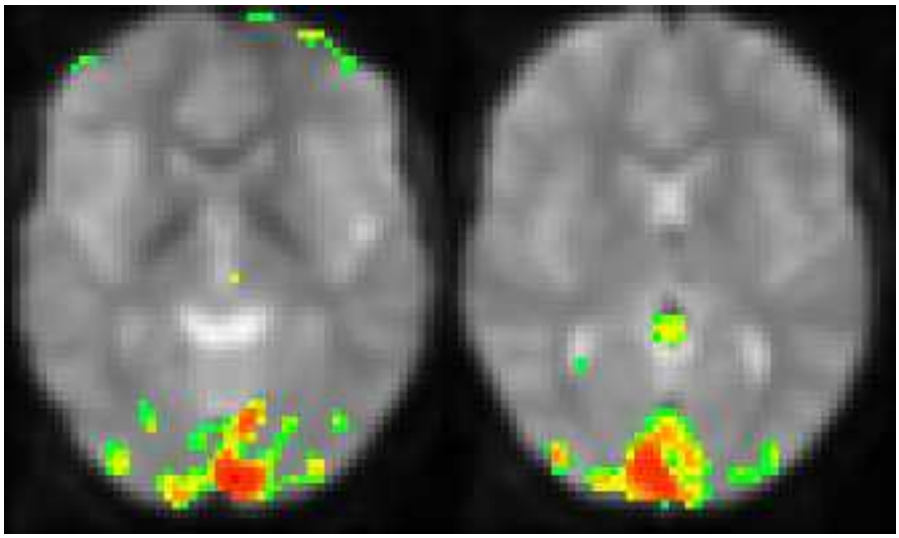


Figura 4.168: Método de sobreposição de ordem 0 com a sequência de (27,6,17) para dados filtrados.

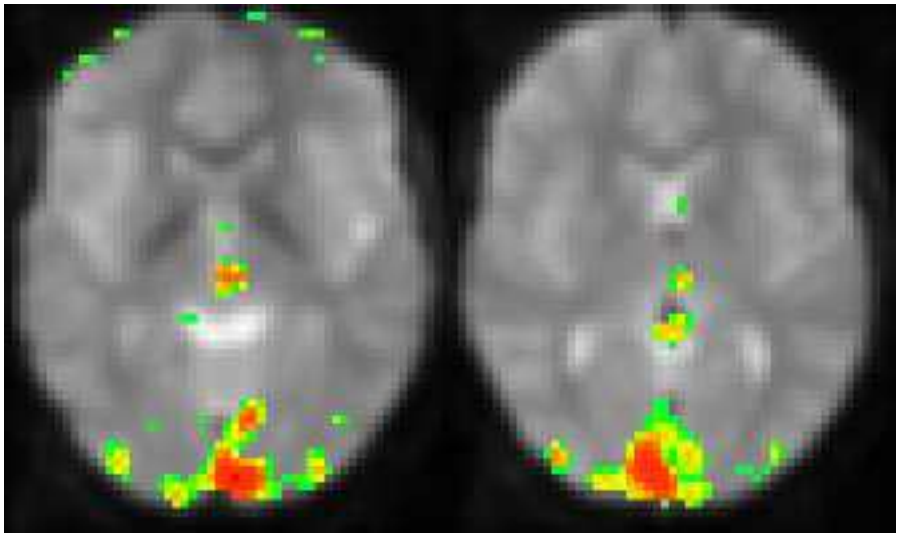


Figura 4.169: Método de sobreposição de ordem 1 com a sequência de (27,6,17) para dados filtrados.

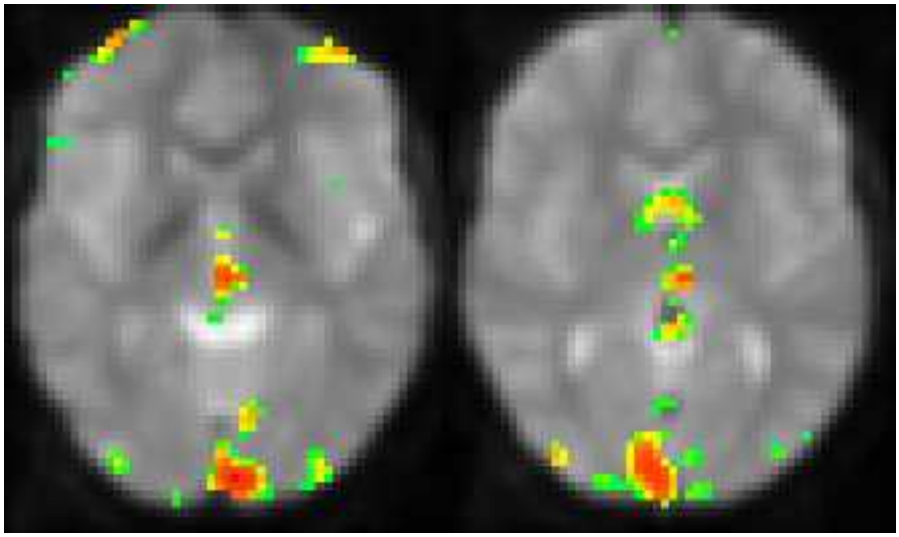


Figura 4.170: Método de sobreposição de ordem 2 com a sequência de (27,6,17) para dados filtrados.

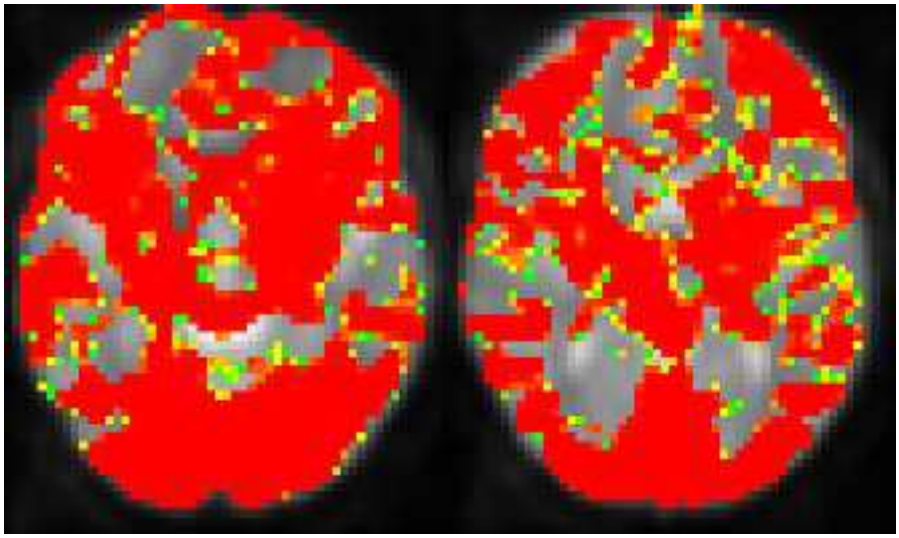


Figura 4.171: Método de coeficiente de correlação com a sequência de (27,6,17) para dados não filtrados.

4.8.4.12 Não filtrados e com conectividade funcional com (27,6,17)

Apresentamos nesta secção as imagens de activação produzidas pelo programa *Cérebro* para cada um dos métodos com dados não filtrados e analisados com uma sequência de modelo igual à do voxel (27,6,17) (figuras 4.171, 4.172, 4.173, 4.174, 4.175 e 4.176).

4.8.4.13 Conclusões das imagens

As imagens reproduzem em parte o que é observado pela análise dos volumes feita em 4.8.1. Os métodos MLG, amplitude e sobreposição de ordem zero apresentam áreas de activação semelhantes.

O método de sobreposição para ordens crescente vê um aumento de activação na região central do cérebro e uma diminuição na zona conhecida de activação.

A introdução da função HRF faz com que as áreas de activação tenham um aumento ligeiro nas áreas de activação.

A filtragem dos dados reduz as áreas de activação.

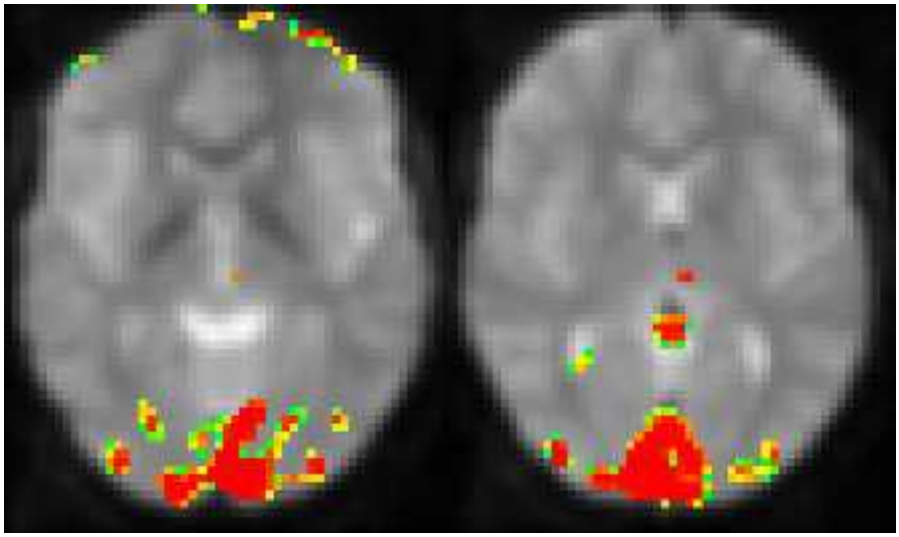


Figura 4.172: Método dos picos de Fourier com a sequência de (27,6,17) para dados não filtrados.

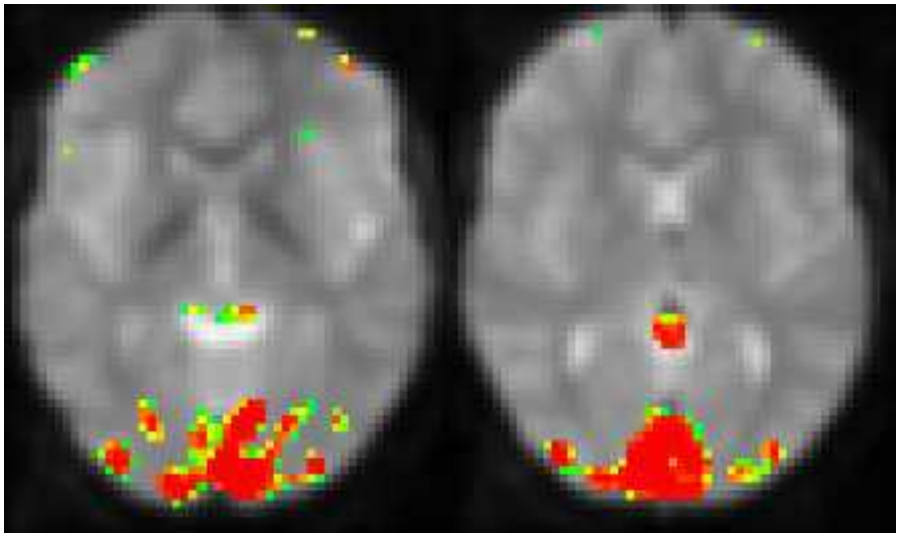


Figura 4.173: Método de amplitude com a sequência de (27,6,17) para dados não filtrados.

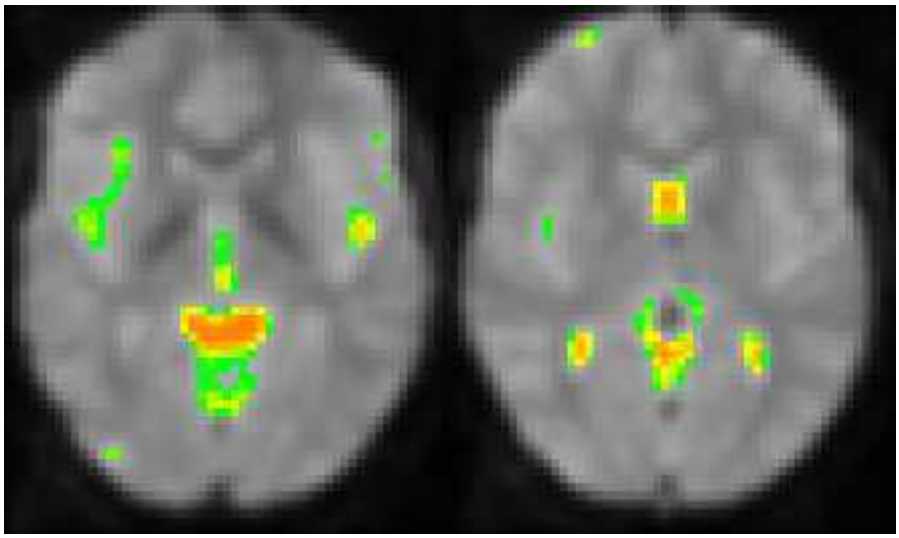


Figura 4.174: Método de sobreposição de ordem 0 com a sequência de (27,6,17) para dados não filtrados.

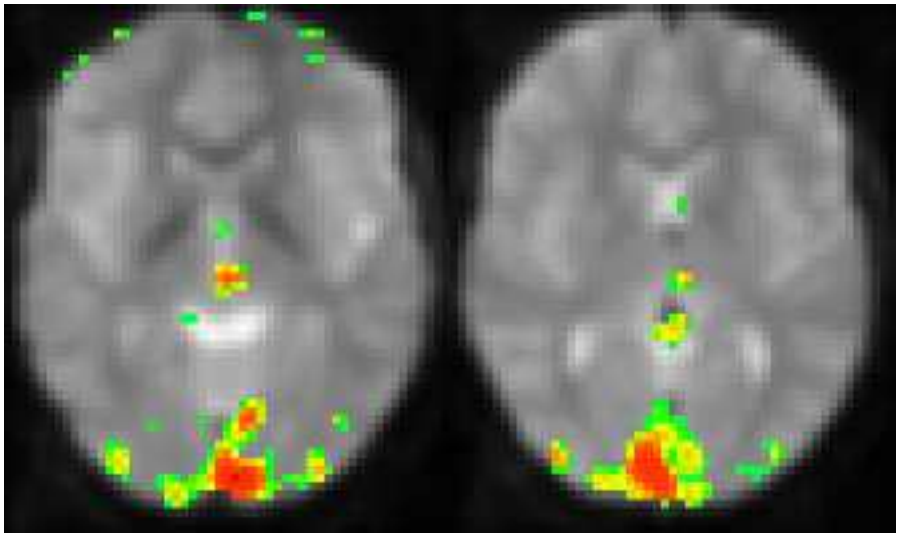


Figura 4.175: Método de sobreposição de ordem 1 com a sequência de (27,6,17) para dados não filtrados.

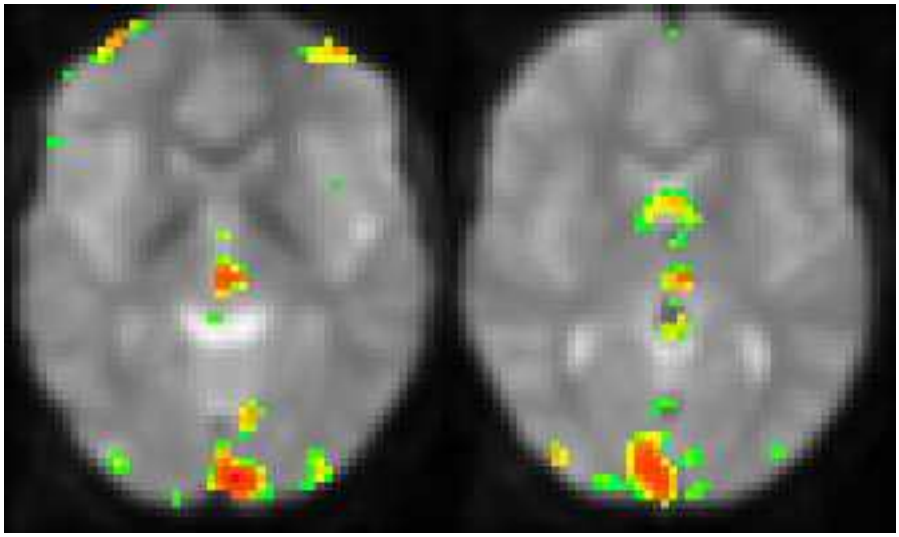


Figura 4.176: Método de sobreposição de ordem 2 com a sequência de (27,6,17) para dados não filtrados.

Capítulo 5

O *Cérebro* e a segmentação

5.1 Teoria

As imagens de ressonância magnética funcional contêm muitas imagens diferentes do cérebro no mesmo estado de activação (e.g. durante o repouso). Seja n_R o número de volumes no estado R. Podemos representar cada voxel como um ponto de um espaço n_R-dimensional. As coordenadas desse voxel são o valor da sua intensidade para cada volume V_i. Este é um *espaço de aglomerados.*

Nas imagens está representado um número finito de tecidos (n_Ξ). Podemos então definir um conjunto finito de *classes*:

$$\Xi = \{\xi_1, \ldots, \xi_{n_\Xi}\} \tag{5.1}$$

Para não confundir as *classes* estatísticas com classes de Java, as primeiras estarão sempre em itálico.

Em teoria de reconhecimento de padrões os valores medidos para cada voxel são as *caractersticas* [18]. Neste caso serão representadas por vectores n_R-dimensionais (ν).

Podemos assumir que todos os pontos do espaço podem ser subdivididos em n_Ξ grupos. Estes grupos são os aglomerados.

A segmentação pretende identificar a que aglomerado pertence cada voxel. Ou seja, assume-se que cada observação foi gerada por uma das *classes* ξ_i.

Neste trabalho optou-se por uma abordagem Bayesiana. Procuramos a probabilidade de um voxel pertencer à *classe* ξ_i se o vector medido foi ν pela equação:

$$\varphi(\xi_i \mid \nu) = \frac{\phi(\nu \mid \xi_i)}{\phi(\nu)} \varphi(\xi_i) \tag{5.2}$$

em que $\phi(\nu \mid \xi_i)$ é a densidade de probabilidade de medir o valor ν se foi gerado pela *classe* ξ_i, $\varphi(\xi_i)$ é a probabilidade da *classe* ξ_i e $\phi(\nu)$ é a densidade de probabilidade de medir ν. Esta última está relacionada com as duas anteriores pela expressão:

$$\phi(\nu) = \sum_{i=0}^{n_\Xi - 1} \phi(\nu \mid \xi_i) \varphi(\xi_i) \tag{5.3}$$

5.2 Aplicação

No programa *Cérebro* assumiu-se que os dados estão distribuidos no espaço de aglomerados de forma normal multivariada:

$$\phi(\nu \mid \xi_i) = \frac{1}{\sqrt{(2\pi)^{n_R} |C|}} exp\left[-\frac{1}{2}(\nu - \bar{\nu}_i)^T C^{-1} (\nu - \bar{\nu}_i)\right] \tag{5.4}$$

Podemos estimar esta distribuição se conhecermos as médias de ν para cada *classe* ξ_i, $\bar{\nu}_i$ e a matriz covariância C. Como n_R é variável ao longo de diferentes estudos, esta equação é implementada na forma matricial. ν são matrizes coluna e C é uma matriz quadrada $n_R \times n_R$.

As médias $\bar{\nu}_i$ são calculadas a partir de *amostras padrão* introduzidas pelo utilizador com um exemplar da classe paramWindowClusterSampler. O utilizador escolhe um mínimo de 10 voxels por *classe*. A partir destes dados criam-se n_Ξ exemplares da classe Cluster. Cada exemplar desta classe contém os dados de imagem relativos ao estado escolhido e uma máscara (exemplar da classe Mask) com a posição dos voxels dessa *classe*.

A classe Cluster tem métodos que calculam as matrizes coluna $\bar{\nu}_i$ (getMeansMatrix()), a matriz covariância C (getCovarianceMatrix()) assim como a distribuição resultante (getDistribution()) sob a forma de um exemplar da classe MultiGaussianDistribution. Esta classe tem o método getProbDensity que toma como argumento um valor ν.

No entanto verificou-se que para um grande número de dimensões o valor da densidade de probabilidade (5.4) pode ser tão pequeno que não é representável na forma de um variável de dupla precisão $\left(\phi\left(\nu \mid \xi_i\right) < 2.2 \times 10^{-308}\right)$. Foi necessário criar a classe LargeNumber. Um exemplar de LargeNumber tem uma mantissa entre 0 e 1 com dupla precisão e um expoente entre $+2.2 \times 10^{308}$ e -2.2×10^{308}. A LargeNumber inclui métodos para todas as operações aritméticas básicas, para potenciação com números não inteiros e de comparação.

A classe MultiGaussianDistribution tem um método capaz de calcular a densidade de probabilidade no formato LargeNumber (getHighPrecProbDensity) que tal como getProbDensity, toma como argumento um valor ν.

Para determinar a probabilidade da *classe* ξ_i é feito um histograma ($h\left(\nu\right)$) de um volume V_j completo. Determina-se o valor do histograma para cada um dos picos. Os picos são identificados a partir das médias das amostras padrão ($h\left(\bar{\nu}_i\right)$). A probabilidade da *classe* ξ_i é estimada por:

$$\varphi\left(\xi_i\right) = h\left(\bar{\nu}_i\right) \cdot \sqrt{\left(C_{jj}\right)_i} \tag{5.5}$$

em que $\left(C_{jj}\right)_i$ é o valor do elemento jj da matriz covariância do aglomerado i. Este cálculo é feito no método getNumberOfClusterVoxels() da classe Cluster.

A densidade de probabilidade total é extraída dos dados das amostras padrão por aplicação da equação 5.3.

Finalmente temos toda a informação para calcular a probabilidade do voxel ν pertencer à *classe* ξ_i a prtir da equação 5.2.

O resultado é apresentado sob a forma de um volume de probabilidade com o valor de $\varphi\left(\xi_i \mid \nu\right)$ para cada voxel e gravável num ficheiro Analyze a partir da janela de resultado.

Capítulo 6

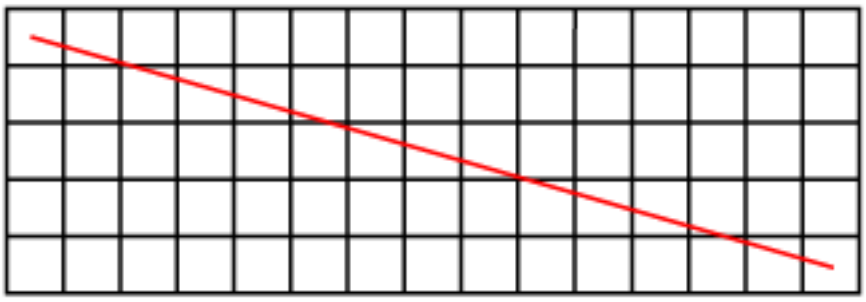
Conjugação de resultados de segmentação com funcionais

6.1 Teoria

Consideremos um volume de probabilidade de activação e outro de segmentação. Por comparação dos perfis é possível avaliar qual o grau de sobreposição. Uma forma imediata de obter perfis é ver a variação da probabilidade ao longo de uma das três direcções ortogonais. Basta manter fixas duas componentes e variar a terceira.

Se tivermos em conta que tanto as áreas de activação como as de tumor não têm em geral uma forma de paralelipípedo é imediato que os perfis ortogonais são insuficientes. Perfis oblíquos permitem completar a avaliação de sobreposição.

Para construir um perfil oblíquo a partir de um volume de probabilidade temos que definir uma linha imaginária ao longo da direcção de perfil pretendida. Uma linha genérica oblíqua irá atravessar os voxels ao longo do seu caminho a uma distância do centro variável. Na figura seguinte temos um exemplo para um perfil sobre um dos três planos ortogonais.



Os valores da probabilidade ao longo da linha têm que ser interpolados. Este objectivo é atingido em duas etapas. Primeiro decidimos quantos pontos a linha de perfil (n_P) irá ter:

$$n_P = 1 + \left[\sqrt{(\Delta X)^2 + (\Delta Y)^2 + (\Delta Z)^2}\right] \tag{6.1}$$

em que os parêntesis rectos representam um arredondamento. ΔX é igual ao número de voxels menos um ao longo da direcção x. ΔY e ΔZ são as quantidades análogas para as direcções y e z respectivamente.

Em segundo lugar interpolamos a probabilidade para cada ponto. A linha de perfill está dividida em $n_P - 1$ segmentos de recta de igual comprimento. Se a linha começa num voxel na posição (X_0, Y_0, Z_0), a posição de um ponto i será dada por:

$$(x_i, y_i, z_i) = (X_0 + i \cdot \delta x, Y_0 + i \cdot \delta y, Z_0 + i \cdot \delta z) \tag{6.2}$$

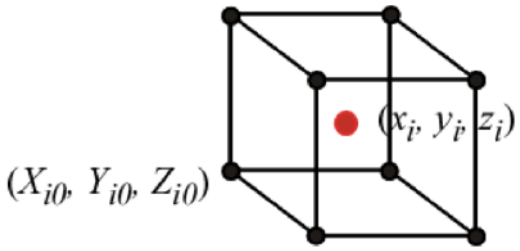
em que:

$$\begin{cases} \delta x = \frac{\Delta X}{n_P - 1} \\ \delta y = \frac{\Delta Y}{n_P - 1} \\ \delta z = \frac{\Delta Z}{n_P - 1} \end{cases} \tag{6.3}$$

O valor interpolado da probabilidade para um ponto i será dado por:

$$v(x_i, y_i, z_i) = \sum_{j=0}^{7} p_j \cdot v(X_{ij}, Y_{ij}, Z_{ij}) \tag{6.4}$$

em que (X_{ij}, Y_{ij}, Z_{ij}) são as coordenadas dos oito pontos mais próximos de (x_i, y_i, z_i):



$$\begin{array}{l} (X_{i0}, Y_{i0}, Z_{i0}) = (\lfloor x_i \rfloor, \lfloor y_i \rfloor, \lfloor z_i \rfloor) \\ (X_{i1}, Y_{i1}, Z_{i1}) = (\lfloor x_i \rfloor + 1, \lfloor y_i \rfloor, \lfloor z_i \rfloor) \\ (X_{i2}, Y_{i2}, Z_{i2}) = (\lfloor x_i \rfloor, \lfloor y_i \rfloor + 1, \lfloor z_i \rfloor) \\ (X_{i3}, Y_{i3}, Z_{i3}) = (\lfloor x_i \rfloor, \lfloor y_i \rfloor, \lfloor z_i \rfloor + 1) \\ (X_{i4}, Y_{i4}, Z_{i4}) = (\lfloor x_i \rfloor + 1, \lfloor y_i \rfloor + 1, \lfloor z_i \rfloor) \\ (X_{i5}, Y_{i5}, Z_{i5}) = (\lfloor x_i \rfloor, \lfloor y_i \rfloor + 1, \lfloor z_i \rfloor + 1) \\ (X_{i6}, Y_{i6}, Z_{i6}) = (\lfloor x_i \rfloor + 1, \lfloor y_i \rfloor, \lfloor z_i \rfloor + 1) \\ (X_{i7}, Y_{i7}, Z_{i7}) = (\lfloor x_i \rfloor + 1, \lfloor y_i \rfloor + 1, \lfloor z_i \rfloor + 1) \end{array} \tag{6.5}$$

e p_j é o peso atribuído a cada dos pontos vizinhos. O seu valor aproxima-se da unidade quando (X_{ij}, Y_{ij}, Z_{ij}) aproxima-se de (x_i, y_i, z_i):

$$p_j = \frac{\frac{1}{7} \sum_{i \neq j} d(i,j)}{\sum_{i=0}^{7} d(i,j)} \tag{6.6}$$

em que:

$$d(i,j) = \sqrt{(x_i - X_j)^2 + (y_i - Y_j)^2 + (z_i - Z_j)^2} \tag{6.7}$$

A soma de todos os oito valore p_j é igual à unidade.
Assim obtemos o perfil de probabilidade pretendido:

$$v(x_i, y_i, z_i), i \in [0, n_P]$$

Se o perfil está sobre um dos três planos ortogonais estes cálculos podem ser simplificados. Por exemplo, se $\Delta X = 0$, a equação 6.1 passa a ser:

$$n_P = 1 + \left[\sqrt{(\Delta Y)^2 + (\Delta Z)^2} \right] \tag{6.8}$$

a equação 6.3 dá:

$$\begin{cases} \delta x = 0 \\ \delta y = \frac{\Delta Y}{n_P - 1} \\ \delta z = \frac{\Delta Z}{n_P - 1} \end{cases} \tag{6.9}$$

e 6.4 altera-se para:

$$v\left(x_i, y_i, z_i\right) = \sum_{j=0}^{3} p_j \cdot v\left(X_{ij}, Y_{ij}, Z_{ij}\right) \tag{6.10}$$

em que (X_{ij}, Y_{ij}, Z_{ij}) são as coordenadas dos quatro pontos mais próximos de (x_i, y_i, z_i):

$$\begin{array}{l} (X_{i0}, Y_{i0}, Z_{i0}) = (\lfloor x_i \rfloor, \lfloor y_i \rfloor, \lfloor z_i \rfloor) \\ (X_{i1}, Y_{i1}, Z_{i1}) = (\lfloor x_i \rfloor, \lfloor y_i \rfloor + 1, \lfloor z_i \rfloor) \\ (X_{i2}, Y_{i2}, Z_{i2}) = (\lfloor x_i \rfloor, \lfloor y_i \rfloor, \lfloor z_i \rfloor + 1) \\ (X_{i3}, Y_{i3}, Z_{i3}) = (\lfloor x_i \rfloor, \lfloor y_i \rfloor + 1, \lfloor z_i \rfloor + 1) \end{array} \tag{6.11}$$

A equação 6.6 simplifica-se:

$$p_j = \frac{\frac{1}{3}\sum_{i \neq j} d\left(i, j\right)}{\sum_{i=0}^{3} d\left(i, j\right)} \tag{6.12}$$

em que:

$$d\left(i, j\right) = \sqrt{\left(y_i - Y_j\right)^2 + \left(z_i - Z_j\right)^2} \tag{6.13}$$

6.2 Aplicação

A implementação prática deste algoritmo de cálculo dos perfis de probabilidade foi feita sob a forma de um programa utilitário separado do *Cérebro* chamado N.

O programa começa por carregar dois volumes de probabilidade (e.g. activação pelo método de picos de Fourier e segmentação) e apresenta os resultados da comparação das probabilidades numa janela exemplar da classe ViewerWindowComparer.

A janela contém um painel onde são apresentadas imagens da diferença algébrica entre os dois volumes de probabilidade. Os valores negativos são apresentados com uma côr com hue entre 0.4 e 0.8 (verde - azul) e os valores positivos com uma cõr com hue entre 0 e 0.4 (vermelho - laranja - amarelo - verde).

Estas são imagens axiais pretendem ser apenas qualitativas. O utilizador observa que se houver muita sobreposição de uma zona de activação com uma de tumor, vai predominar o verde na zona do tumor.

A análise quantitativa é feita pelos perfis que são apresentados sob a forma de um gráfico com duas linhas: uma vermelha e a outra azul. Elas representam a variação da probabilidade para cada um dos volumes ao longo de uma linha definida por dois voxels quaiquer do volume. Os voxels são seleccionados pelo utilizador clicando a imagem na posição pretendida. Se o utilizador clicar com o rato num ponto da imagem, irá definir as posições dos primeiro e segundo voxels alternadamente. Se quiser pode definir em qualquer altura a posição do primeiro voxel clicando na posição desejada mantendo o botão de CTRL pressionado. Para o segundo voxel o procedimento é o mesmo com a diferença que o botão pressionado deve ser o ALT.

O programa calcula a percentagem de voxels sobrepostos entre as duas probabilidades para um valor de p definido pelo utilizador. É construída uma máscara para cada volume de probabilidade. Os voxels têm todos os valor "falso" com excepção naqueles em que o valor da probabilidade é maior ou igual ao valor limiar definido pelo utilizador. Em seguida o programa aplica a operação booleana "e" aos volumes e o resultado é uma nova máscara com o valor "verdadeiro" apenas onde havia sobreposição. A percentagem de sobreposição é apresentada dividindo o número de voxels com o valor "verdadeiro" da terceira máscara pelo menor número de voxels válidos das duas máscaras iniciais.

Capítulo 7

Um caso clínico

7.1 O estudo

Os dados apresentados referem-se a uma situação clínica em que há um tumor perto de uma área do cérebro recrutada no movimento da mão direita. É necessário avaliar se uma remoção total do tumor poderá afectar o movimento da mão direita. Para tal são feitas três estimativas de volume. A primeira é do volume de sobreposição da activação e do tumor, a segunda é da percentagem do volume de activação sobreposto e a terceira é da percentagem do volume de tumor sobreposto.

A aquisição dos dados é feita através de um estudo de ressonância magnética funcional com um design de bloco. O paciente aperta uma bola de borracha periodicamente. Fazem-se 10 aquisições de volume por cada estado (repouso ou aperto) com um tempo de acquisição (TR) de 6.2 s. O número total de volumes é 50.

As imagens das três fatias com tumor estão na fig. 7.1

A segmentação identifica as áreas de tumor (fig. 7.2).

7.2 Análise comparativa

Os resultados da análise funcional variam conforme o método de avaliação de activação escolhido. Segue-se uma análise comparativa com o programa *Cérebro* e utilitário *Comparador*.

7.2.1 Extremos e volumes

Os dados são filtrados com um filtro passa-alto com uma frequência de corte $14mHz$. Em seguida calculam-se os volumes de probabilidade para cada um dos métodos de análise funcional do

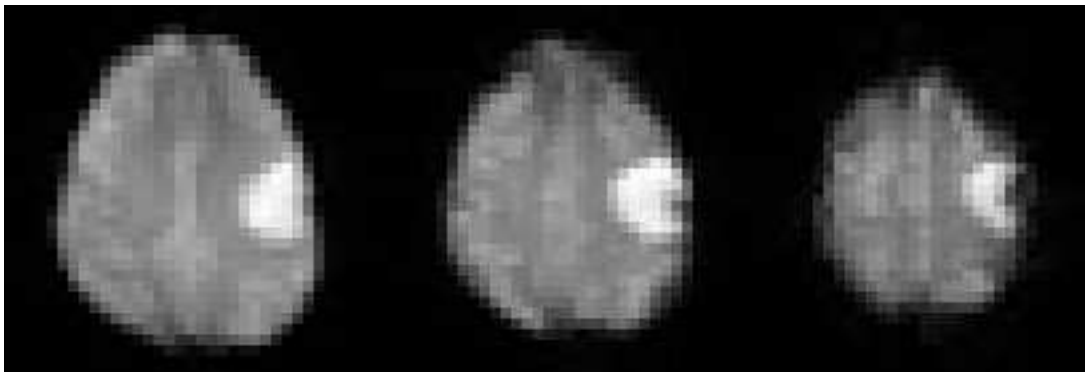


Figura 7.1: Fatias 5, 6 e 7 dos dados

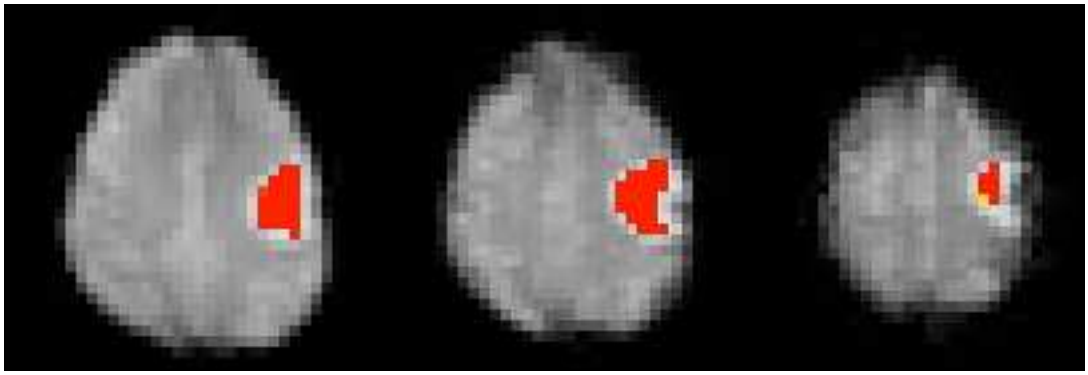


Figura 7.2: Fatias 5, 6 e 7 dos dados com o tumor segmentado

Método	$V\,(cm^3)$	$\%V_A$	$\%V_T$	maxVox
t de Student	2.3	0.8	15.1	(37,33,7)
Coef. de correlação	2.8	0.9	18.9	(37,33,7)
picos de Fourier	1.4	1.8	9.4	(39,35,5)
MLG	0.3	0.4	1.9	(37,33,7)
Amplitude	0.0	0.0	0.0	(37,33,7)
Sobreposição de ordem 0	0.6	1.5	3.8	(39,35,5)
Sobreposição de ordem 1	0.0	0.0	0.0	(39,35,5)
Sobreposição de ordem 2	0.0	0.0	0.0	(40,34,5)

Tabela 7.1: Volume de sobreposição entre os volumes de activação e de tumor, percentagem do volume de activação sobreposto, percentagem do volume de tumor sobreposto e voxel de máxima probabilidade de sobreposição para modelo em bloco

programa *Cérebro* com e sem HRF.

Os volumes estimados para $p < 0.05$ são apresentados nas tabelas seguintes.

Na tabela 7.1 os métodos de amplitude e de sobreposição (ordens 1 e 2) dizem que o volume de sobreposição é nulo para $p < 0.05$.

Os métodos de t de Student, de coeficiente de correlação, de picos de Fourier, de MLG e de sobreposição de ordem 0 dizem o contrário: o volume de sobreposição dos volumes de activação e tumor não é nulo para $p < 0.05$.

Pode-se argumentar que ficou bem visível na secção 4.8 que os métodos de t de Student e de coeficiente de correlação sobrestimam o volume de activação. No entanto os métodos de picos de Fourier, de MLG e o método de sobreposição de ordem 0 corroboram em medida diferente que há de facto sobreposição dos volumes de activação e de tumor.

Se tivermos em conta as dimensões de cada voxel $\left(5.3125mm \times 5.312541mm \times 9.999999mm \approx 0.3cm^3\right)$ o método de picos de Fourier diz que há sobreposição em 5 voxels o de MLG em 1 voxel e o de sobreposição de ordem 0 em 2 voxels.

Os voxels de máxima activação centraram-se na fatia 7 para os métodos de t de Student, de coeficiente de correlação, do MLG e da amplitude e na fatia 5 para os métodos de picos de Fourier e de sobreposição.

Se introduzirmos a correcção com HRF na sequência de modelo (tabela 7.2)

vemos que os métodos que apresentavam um volume não nulo mantiveram-se. O método de picos de Fourier faz uma estimativa de volume inferior em 1 voxel enquanto o de coeficiente de correlação faz uma estimativa inferior em 4 voxels (de 10 para 6 voxels).

O desempenho dos métodos de análise funcional pode ser parciamente avaliado pela posição dos máximos de activação para cada método (tabelas 7.3 e 7.4).

Método	$V\left(cm^3\right)$	$\%V_A$	$\%V_T$	maxVox
Coef. de correlação	1.7	0.6	11.3	(37,30,5)
picos de Fourier	1.1	1.4	7.5	(39,35,5)
Amplitude	0.0	0.0	0.0	(37,33,7)
Sobreposição de ordem 0	0.6	1.4	3.8	(39,35,5)
Sobreposição de ordem 1	0.0	0.0	0.0	(39,35,5)
Sobreposição de ordem 2	0.0	0.0	0.0	(40,34,5)

Tabela 7.2: Volume de sobreposição entre os volumes de activação e de tumor, percentagem do volume de activação sobreposto, percentagem do volume de tumor sobreposto e voxel de máxima probabilidade de sobreposição para modelo com HRF

Método	maxVox
t de Student	(38,26,6)
Coef. de correlação	(38,26,6)
picos de Fourier	(28,29,2)
MLG	(28,29,2)
Amplitude	(28,29,2)
Sobreposição de ordem 0	(28,29,2)
Sobreposição de ordem 1	(27,30,0)
Sobreposição de ordem 2	(27,30,0)

Tabela 7.3: Voxel de activação máxima para cada método de avaliação para dados analisados com um modelo em bloco

Método	maxVox
Coef. de correlação	(38,26,6)
picos de Fourier	(28,30,2)
Amplitude	(28,29,2)
Sobreposição de ordem 0	(28,29,2)
Sobreposição de ordem 1	(27,30,0)
Sobreposição de ordem 2	(27,30,0)

Tabela 7.4: Voxel de activação máxima para cada método de avaliação para dados analisados com um modelo com HRF

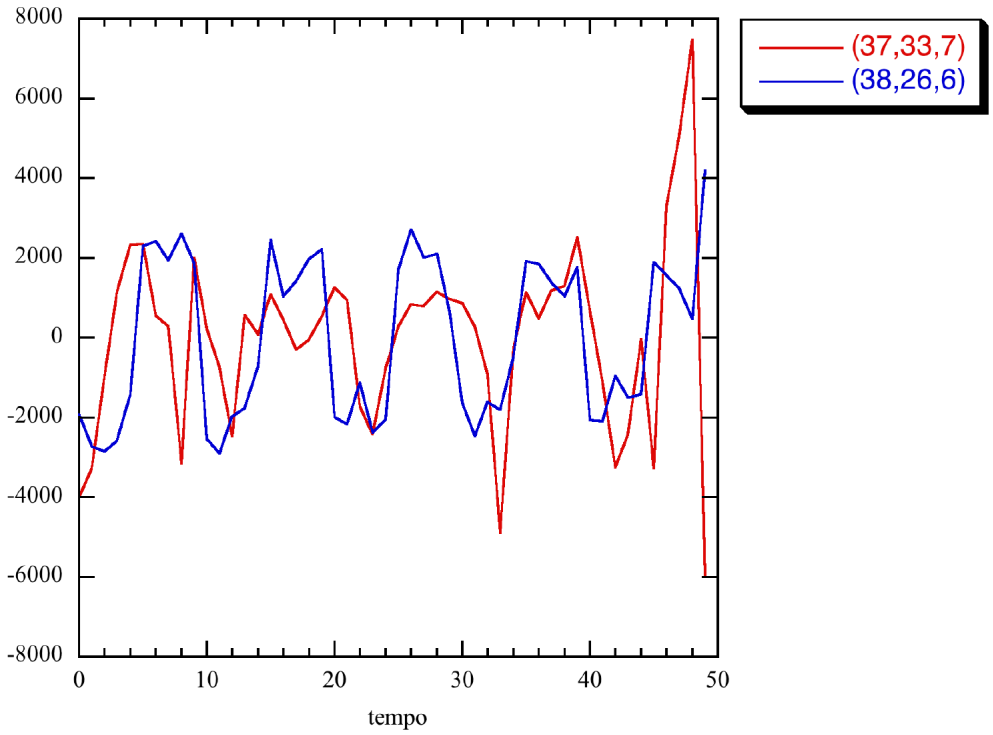


Figura 7.3: Comparação das sequências dos voxels (37,33,7) e (38,26,6).

Os métodos de t de Student, de coeficiente de correlação são os únicos com o máximo na zona de maior activação.

7.2.2 Sequências e espectros

Nesta secção comparam-se as sequências dos voxels de activação máxima.

Na figura 7.3, verifica-se que as amplitudes dos voxels (37,33,7) e (38,28,6) são comparáveis ainda que a igualdade de período possa ser questionada.

A figura 7.4 revela que a frequência das primeiras harmónicas são coincidentes. Além disso há uma elevação na frequência da segunda harmónica com uma amplitude relativa sugestiva. Podemos então concluír que os dados sugerem fortemente uma activação no voxel (37,33,7).

O segundo voxel identificado como de sobreposição é o (39,35,5). Em 7.5 compara-se a sua sequência temporal com a do voxel (38,26,6).

Verifica-se que as amplitudes e os períodos são comparáveis. Há no entanto uma diferença de fase entre os sinais.

A comparação dos espectros (figura 7.6) confirma que as frequências da primeira harmónica são semelhantes.

A ausência do pico de segunda harmónica não impede que admitamos a possibilidade de uma activação válida no voxel (39,35,5).

Quanto ao voxel (40,34,5), a análise temporal (figura 7.7) indica a presença de frequências elevadas.

A análise espectral (figura 7.8) mostra que há de facto um pico junto à primeira harmónica do voxel (38,26,6)

Há também um pico na frequência da segunda harmónica. A presença de frequências superiores de amplitude comparável à da primeira harmónica do voxel (38,26,6) não é suficiente para excluír a hipótese de uma activação válida.

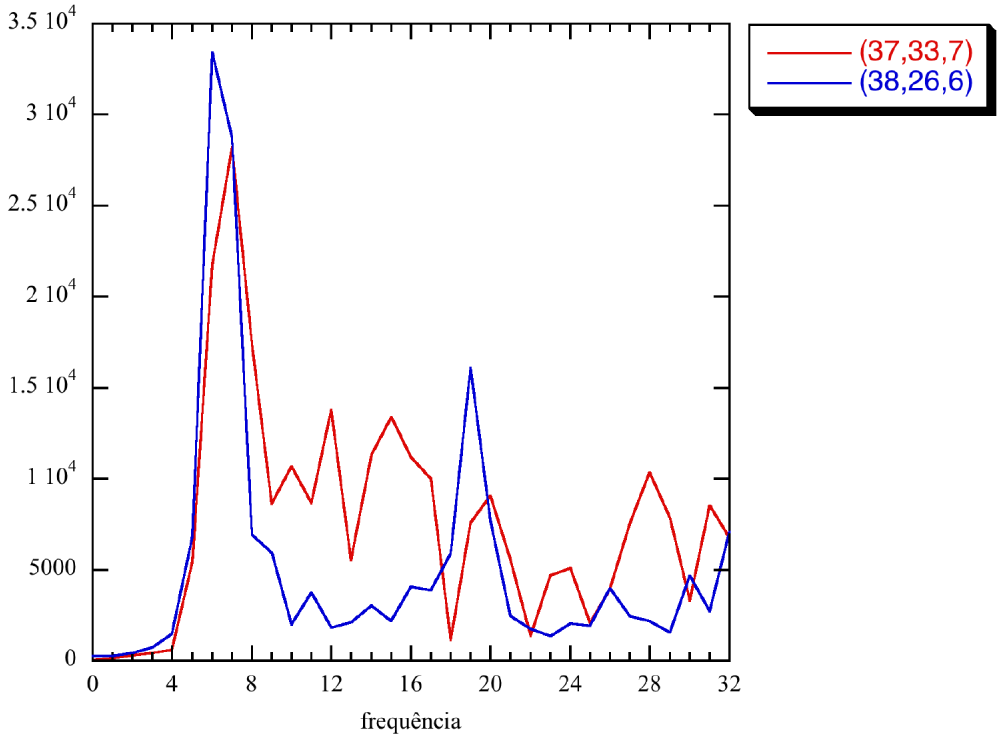


Figura 7.4: Comparação dos espectros das sequências dos voxels (37,33,7) e (38,26,6).

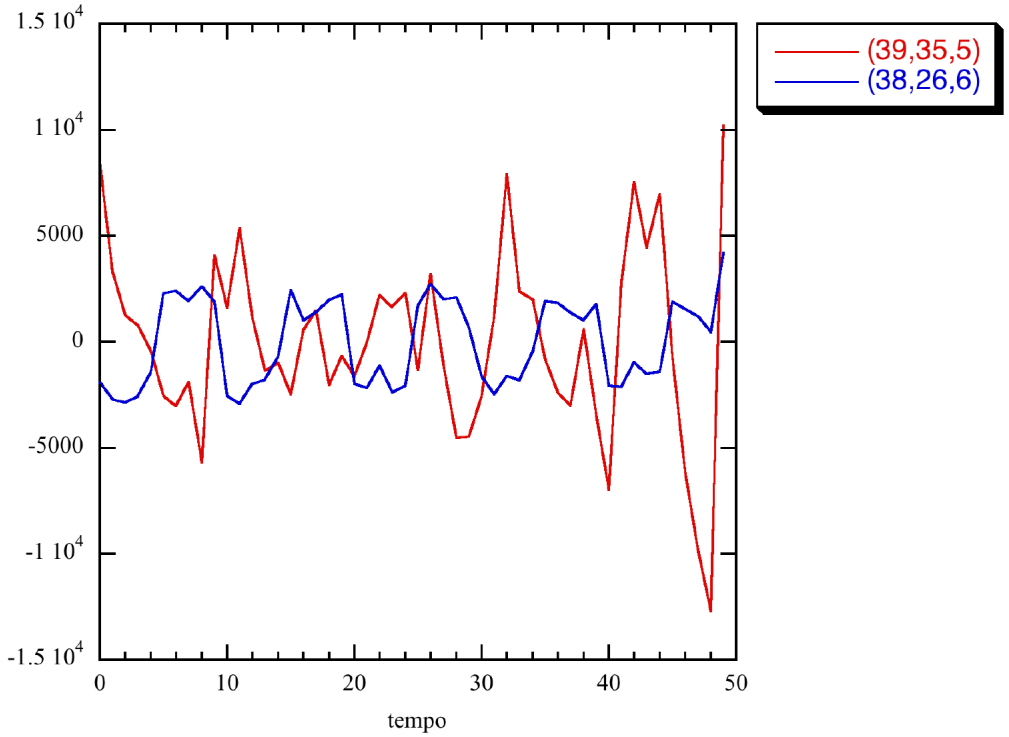


Figura 7.5: Comparação das sequências dos voxels (39,35,5) e (38,26,6)

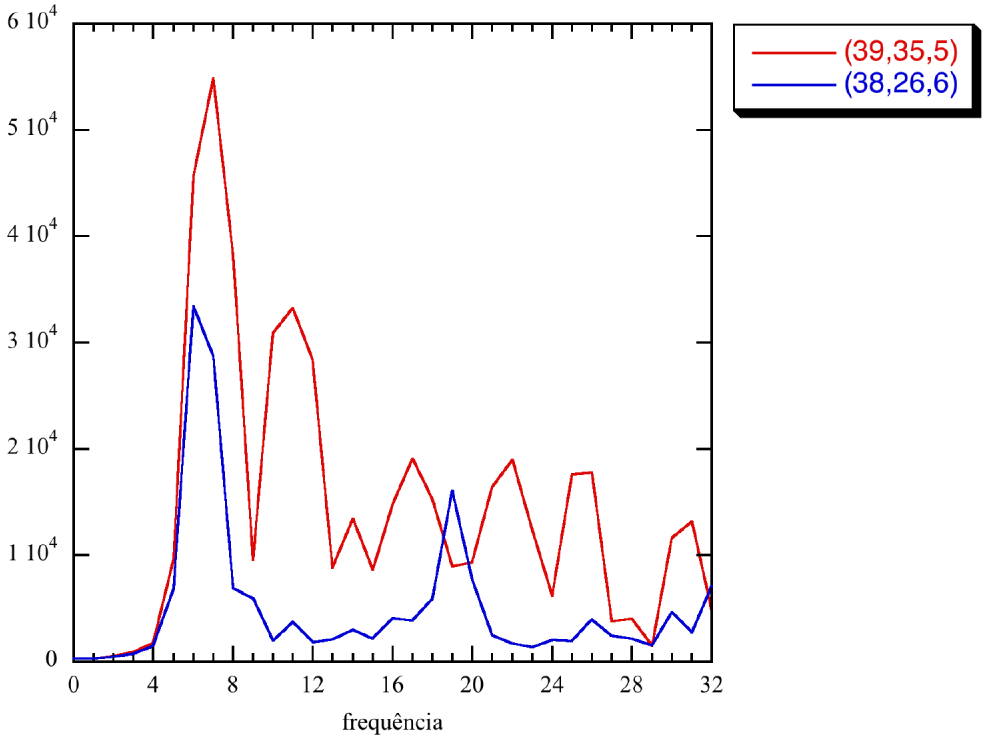


Figura 7.6: Comparação dos espectros das sequências dos voxels (39,35,5) e (38,26,6).

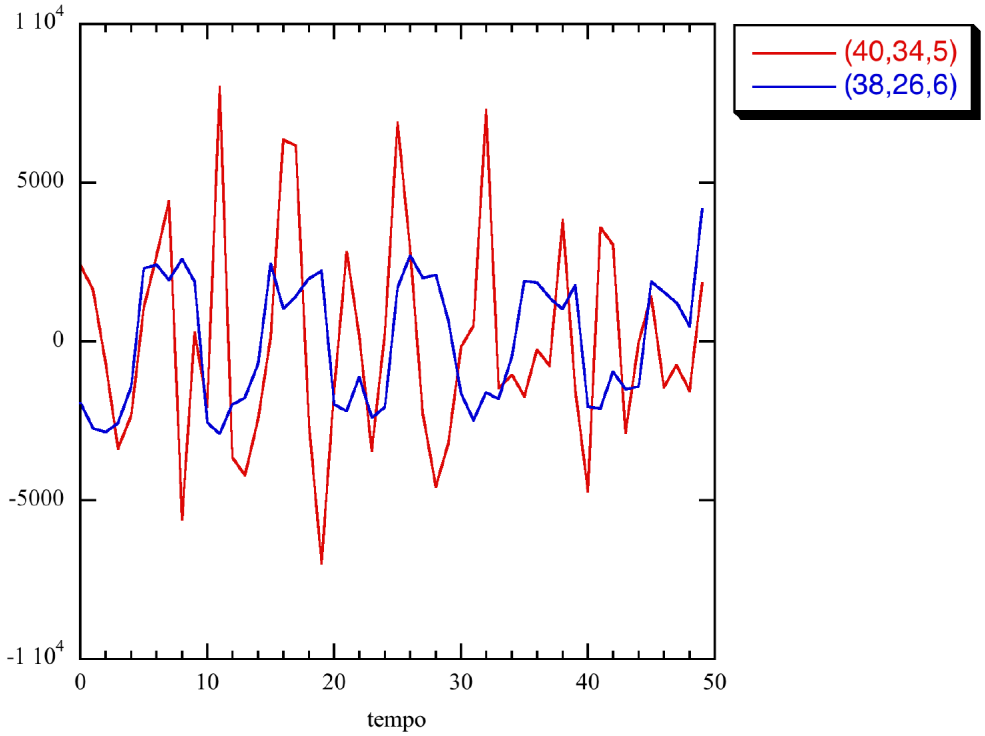


Figura 7.7: Comparação dos espectros das sequências dos voxels (40,34,5) e (38,26,6)

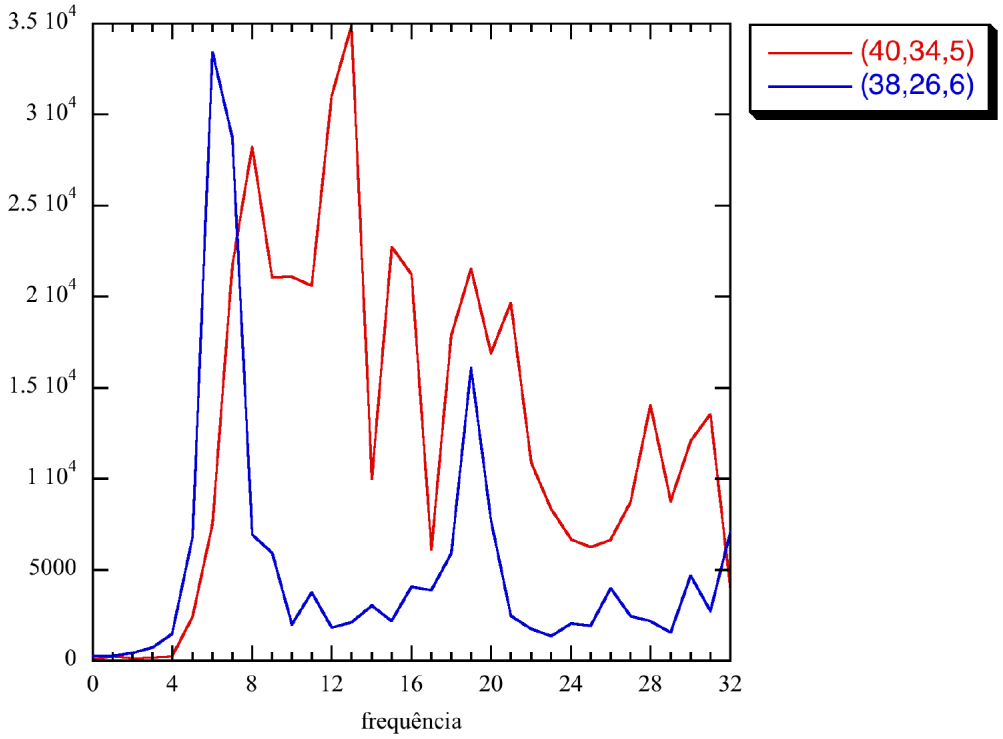


Figura 7.8: Comparação dos espectros das sequências dos voxels (40,34,5) e (38,26,6)

O sinal temporal do voxel (37,30,5) (figura 7.9) apresenta uma amplitude comparável à do voxel (38,26,6).

A partir da análise dos espectros da figura 7.10 é possível identificar um pico coincidente com o da primeira harmónica do sinal do voxel (38,26,6).

Existe também outro pico com amplitude elevada para uma frequência superior (entre as primeira e sgunda harmónicas). Os dados são inconclusivos quanto à validade da activação no voxel (37,30,5).

Quanto aos voxels de activação máxima, uma análise temporal e espectral do principal voxel concorrente (28,29,2) ao voxel (38,26,6) (figuras 7.11 e 7.12) permite-nos observar que a sua amplitude é maior que a do voxel (38,26,6). Porém o contributo de sinais de outra frequência é muito elevado. Não podemos concluír desta forma se o sinal é ou não de activação válida.

O sinal no voxel (27,30,0) visto na figura 7.13

também apresenta uma grande quantidade de sinais de alta frequência (figura 7.14). Estes até ocorrem "preferencialmente" fora das frequências das primeira e segunda harmónicas.

Os testes anteriores ao método da sobreposição de ordem 2 sugerem que este método é sensível a sinais de alta frequência e apresenta erros do tipo I nestes casos.

7.2.3 Perfis

Analisamos nesta secção os perfis de probabilidade referentes aos voxels (37,33,7) e (39,35,5) para os vários métodos que o assinalaram.

As formas dos perfis de activação nas figuras 7.15, 7.16 e 7.17 são consistentes entre si e todas apresentam sobreposição com o tumor. A única diferença reside na probabilidade de activação na zona do tumor.

As formas dos perfis de activação nas figuras 7.18, 7.19 e 7.20 não são tão consistentes entre si mas todas apresentam sobreposição com o tumor. De facto a probabilidade de activação média

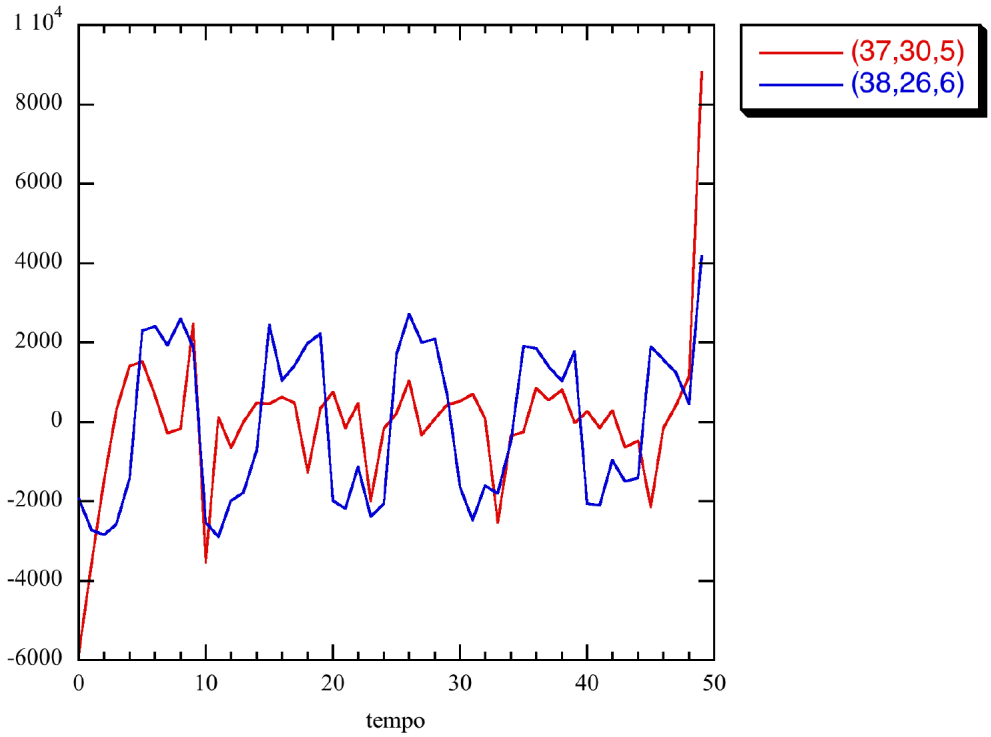


Figura 7.9: Comparação das sequências dos voxels (37,30,5) e (38,26,6).

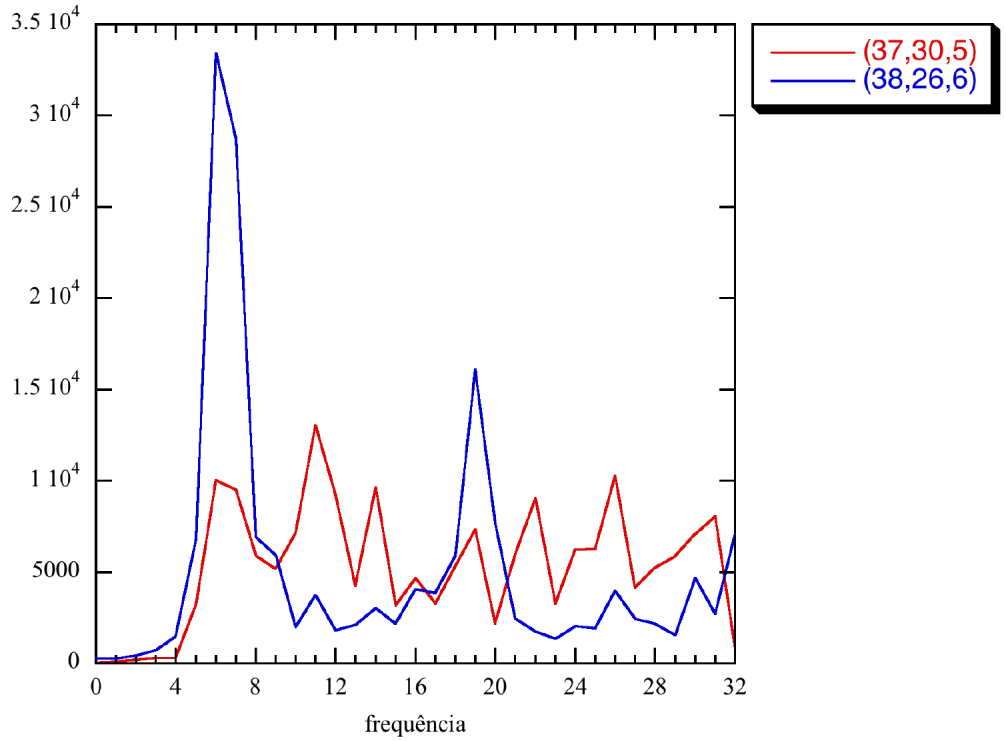


Figura 7.10: Comparação dos espectros das sequências dos voxels (37,30,5) e (38,26,6)

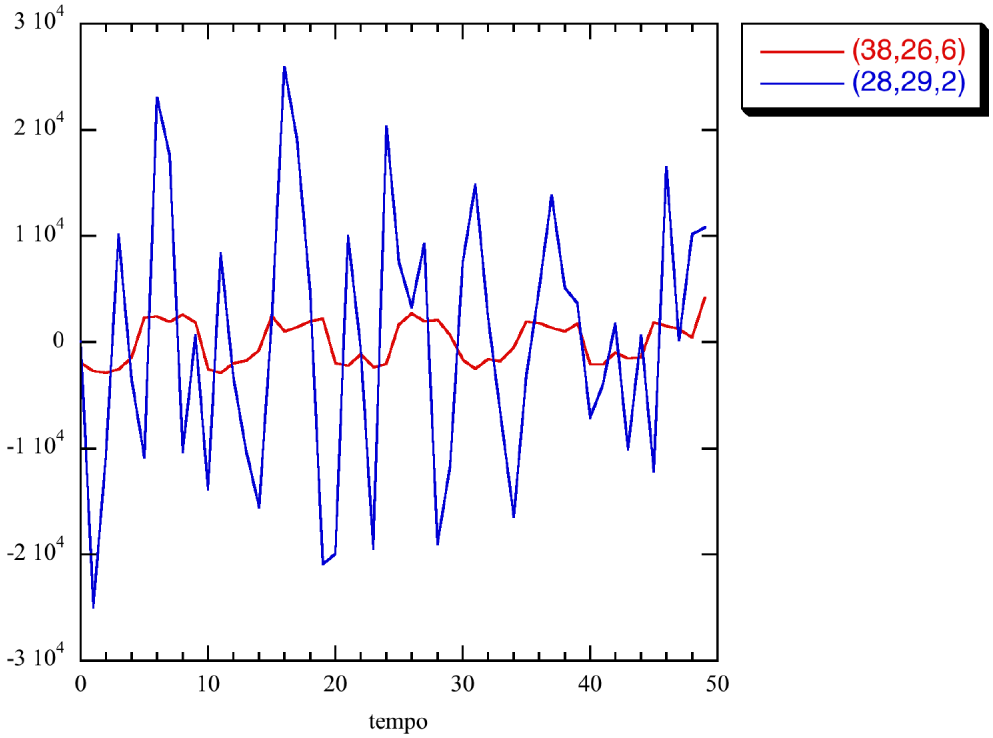


Figura 7.11: Comparação das sequências dos voxels (38,28,6) e (28,29,2).

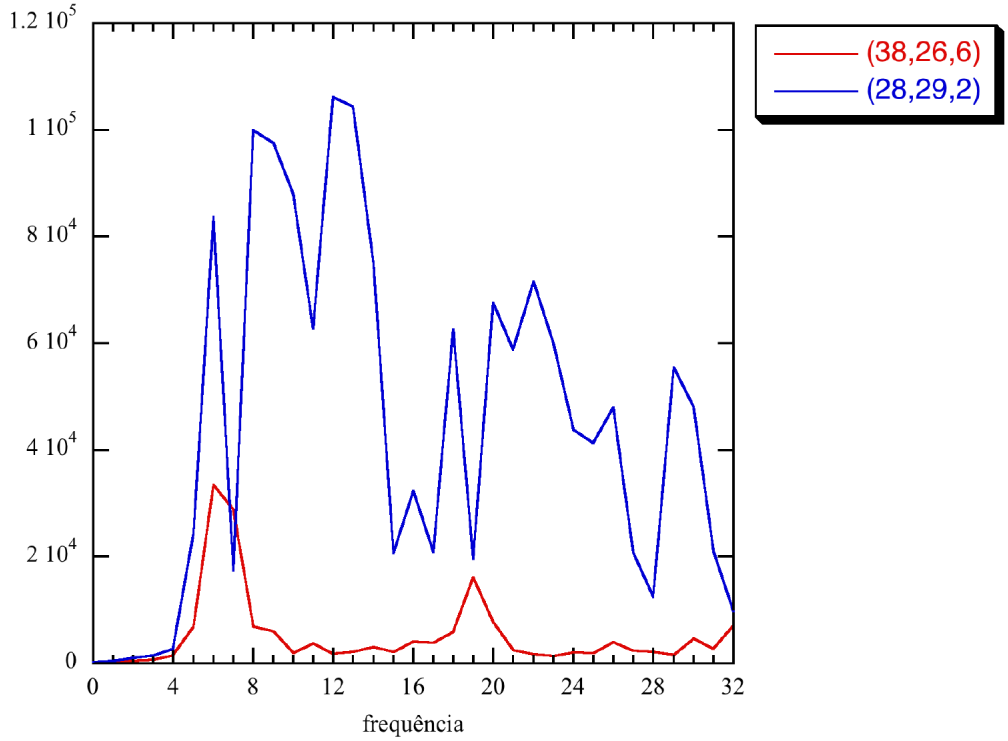


Figura 7.12: Comparação dos espectros das sequências dos voxels (38,26,6) e (28,29,2).

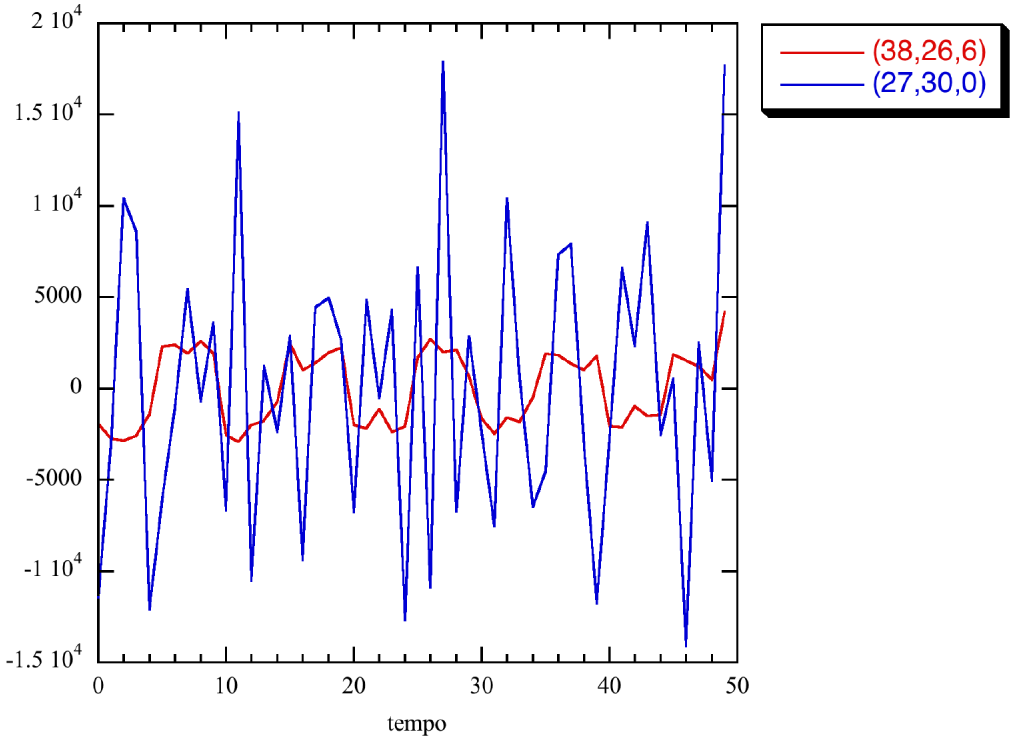


Figura 7.13: Comparação das sequências dos voxels (38,26,6) e (27,30,0).

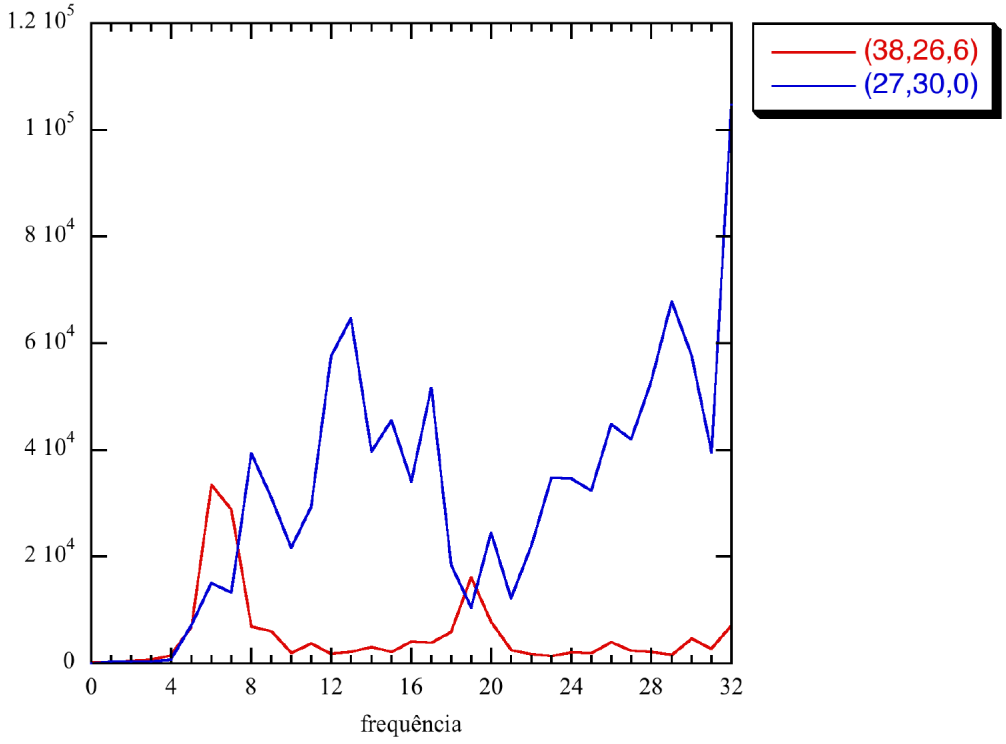


Figura 7.14: Comparação dos espectros das sequências dos voxels (38,26,6) e (27,30,0).

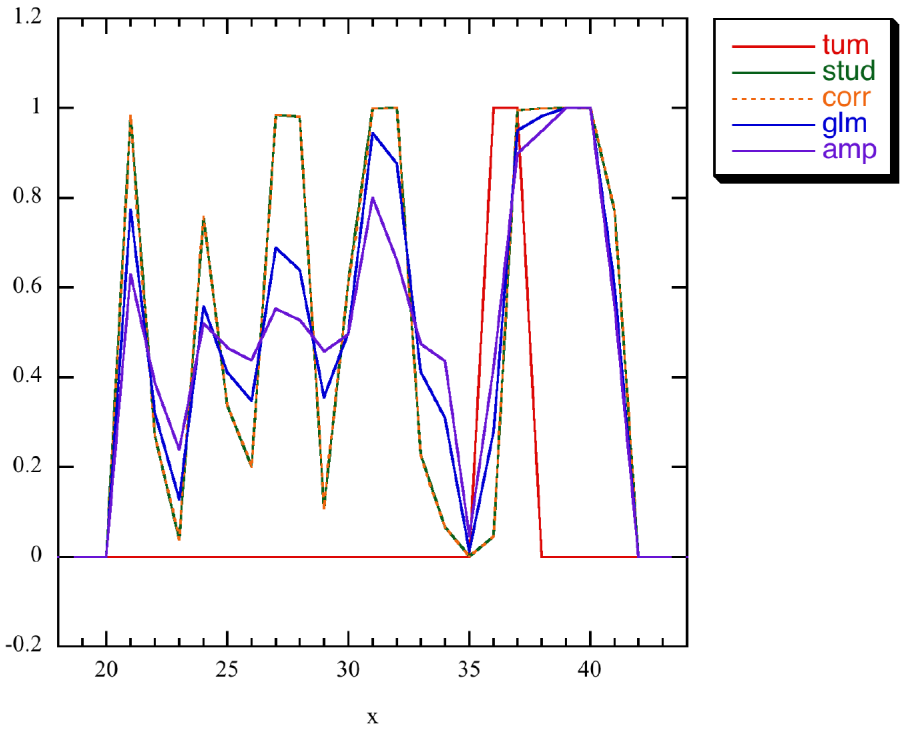


Figura 7.15: Perfil de probabilidade segundo x do voxel (37,33,7) para o tumor e os métodos de Student, coeficiente de correlação, MLG e de amplitude.

na zona do tumor é superior para este voxel na zona de sobreposição.

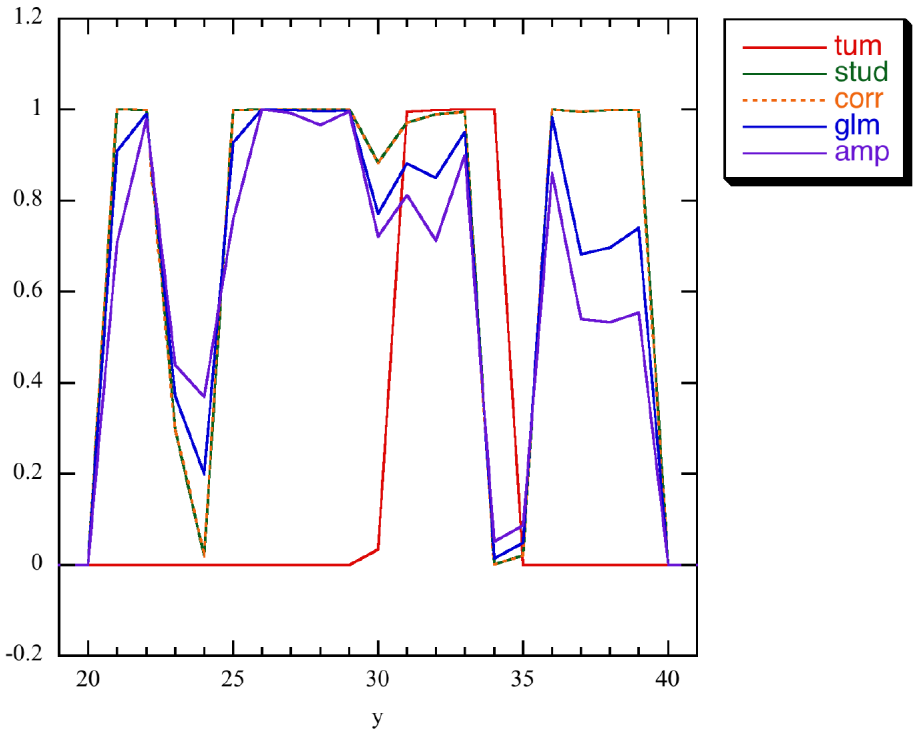


Figura 7.16: Perfil de probabilidade segundo y do voxel (37,33,7) para o tumor e os métodos de Student, coeficiente de correlação, MLG e de amplitude.

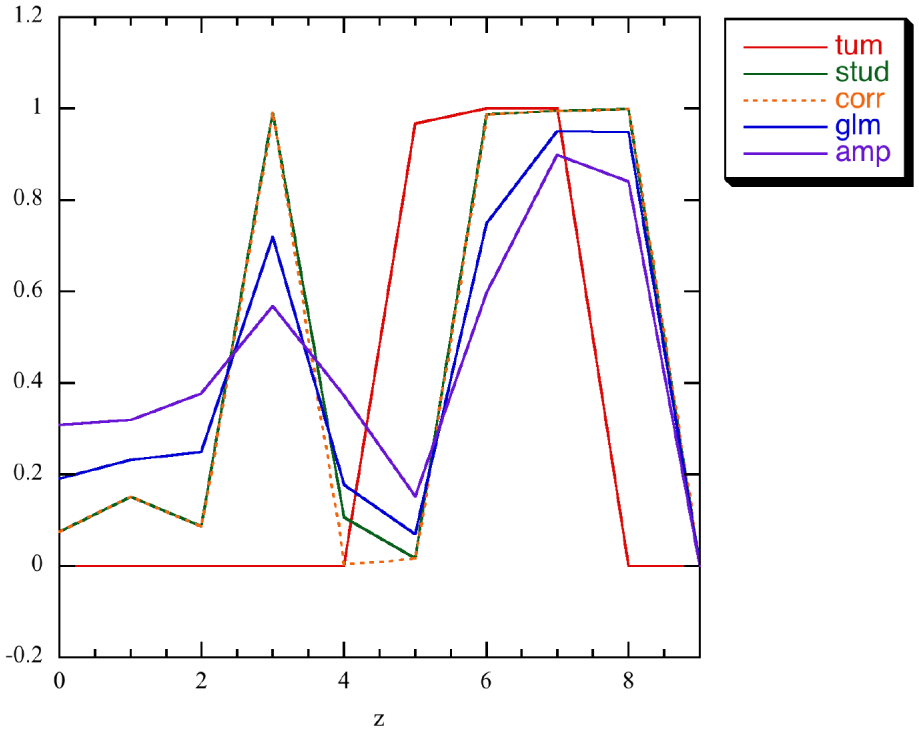


Figura 7.17: Perfil de probabilidade segundo z do voxel (37,33,7) para o tumor e os métodos de Student, coeficiente de correlação, MLG e de amplitude.

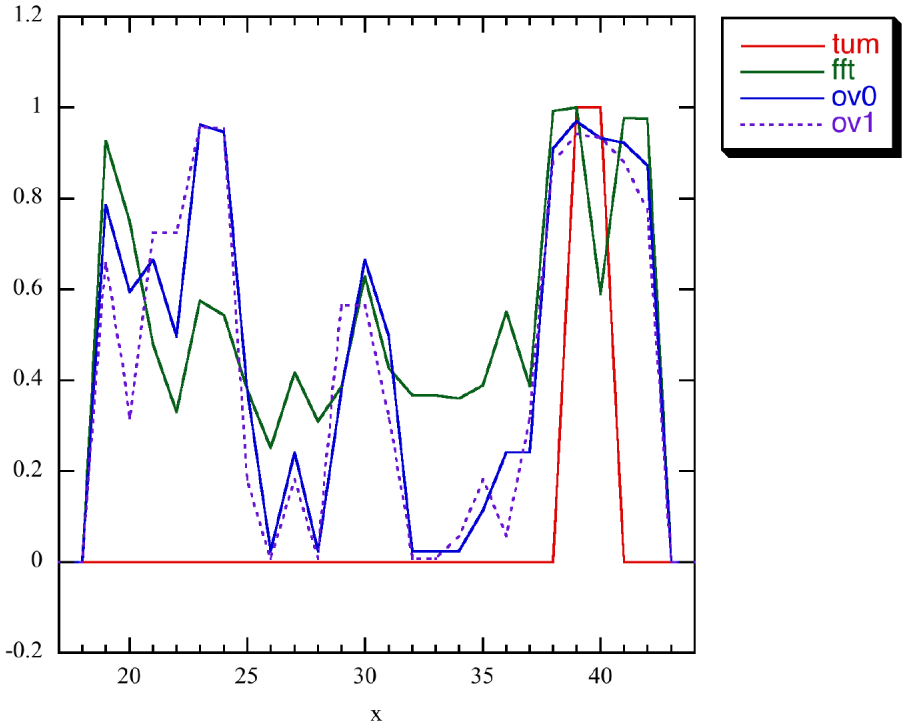


Figura 7.18: Perfil de probabilidade segundo x do voxel (39,35,5) para o tumor e os métodos de picos de Fourier e sobreposição de ordens 0 e 1

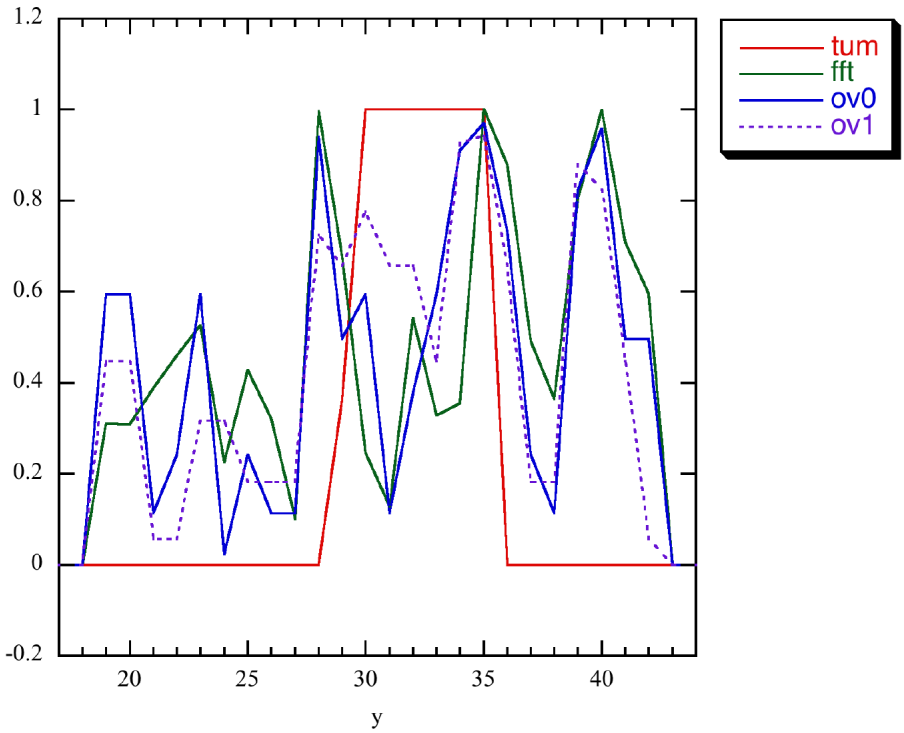


Figura 7.19: Perfil de probabilidade segundo y do voxel (39,35,5) para o tumor e os métodos de picos de Fourier e sobreposição de ordens 0 e 1

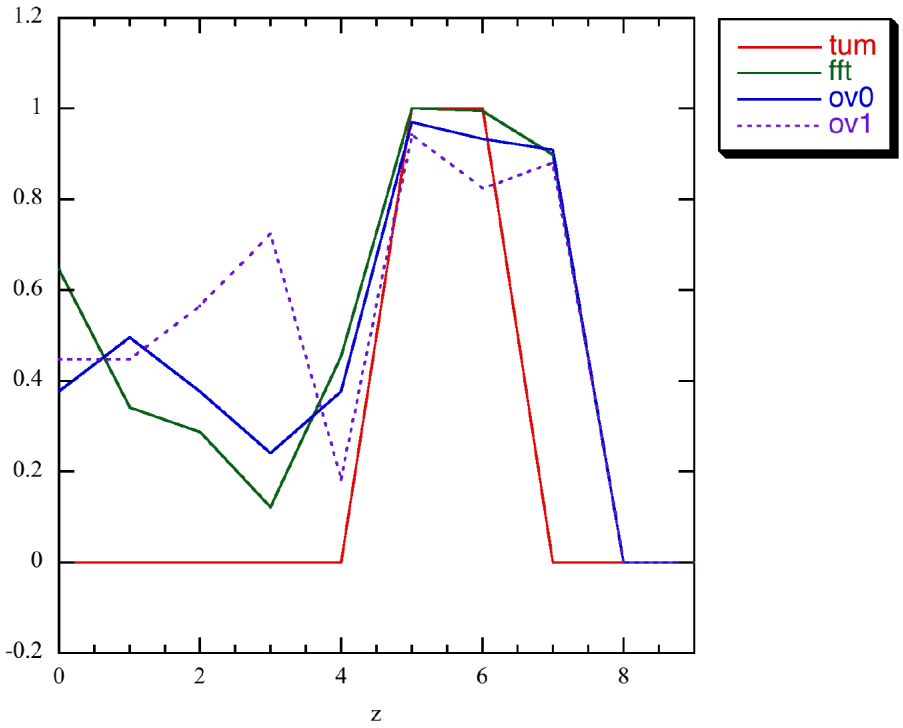


Figura 7.20: Perfil de probabilidade segundo z do voxel (39,35,5) para o tumor e os métodos de picos de Fourier e sobreposição de ordens 0 e 1

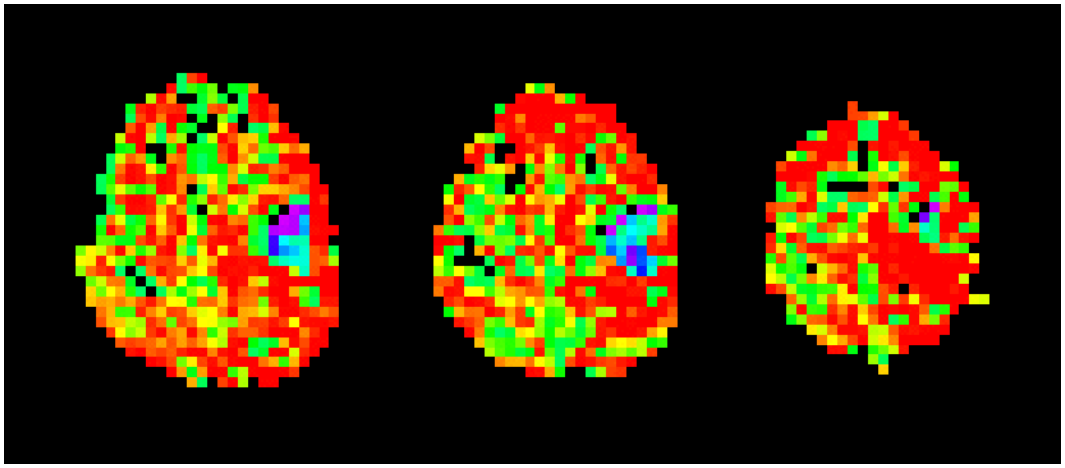


Figura 7.21: Método de t de Student sem HRF

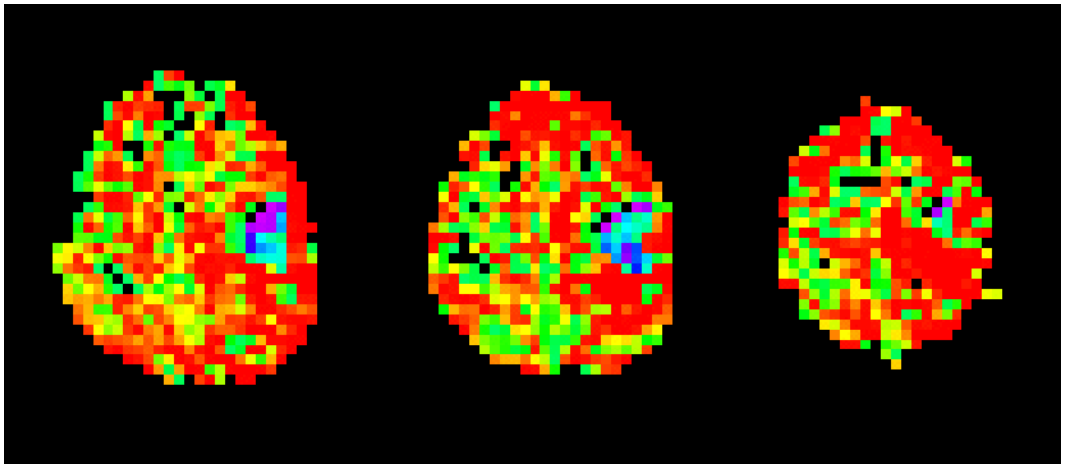


Figura 7.22: Método de coeficiente de correlação sem HRF

7.2.4 Imagens

O programa *Comparador* produz imagens das diferenças de probabilidade entre os volumes de probabilidade de activação e de tumor.

7.2.4.1 Dados sem HRF

Apresentam-se as imagens produzidas pelo programa *Comparador* da diferença entre os volumes de probabilidade de activação e tumor para cada método de análise funcional (figuras 7.21, 7.22, 7.23, 7.24, 7.25, 7.26, 7.27, 7.28). Os dados foram analisados assumindo uma sequência de modelo em bloco.

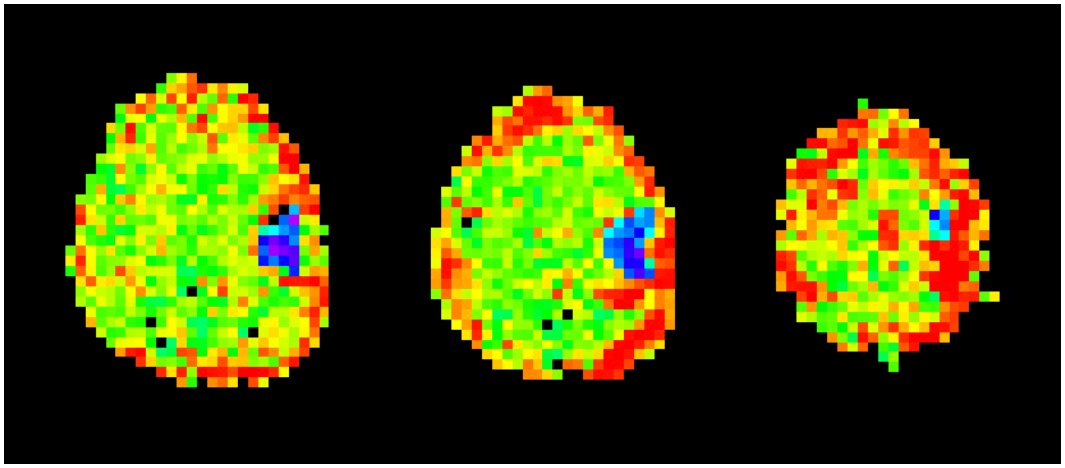


Figura 7.23: Método dos picos de Fourier sem HRF

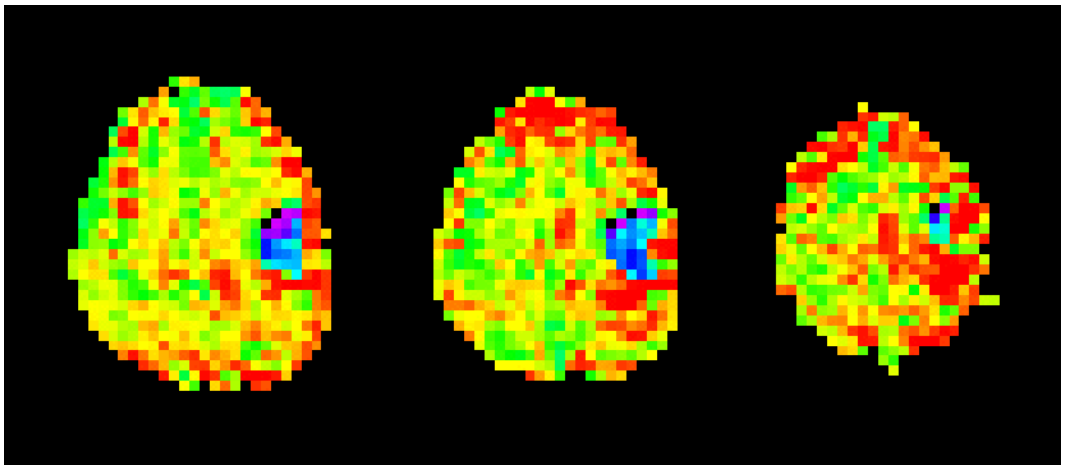


Figura 7.24: Método do modelo linear generalizado sem HRF

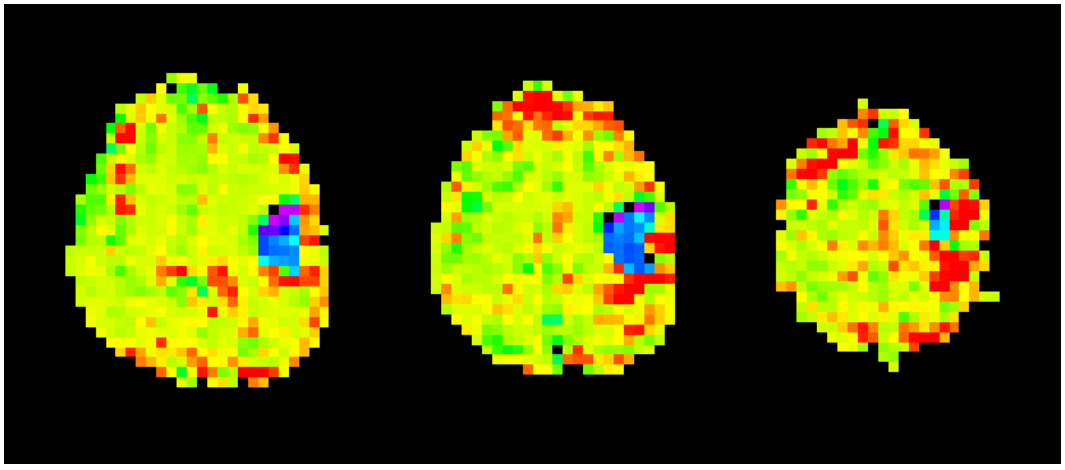


Figura 7.25: Método de amplitude sem HRF

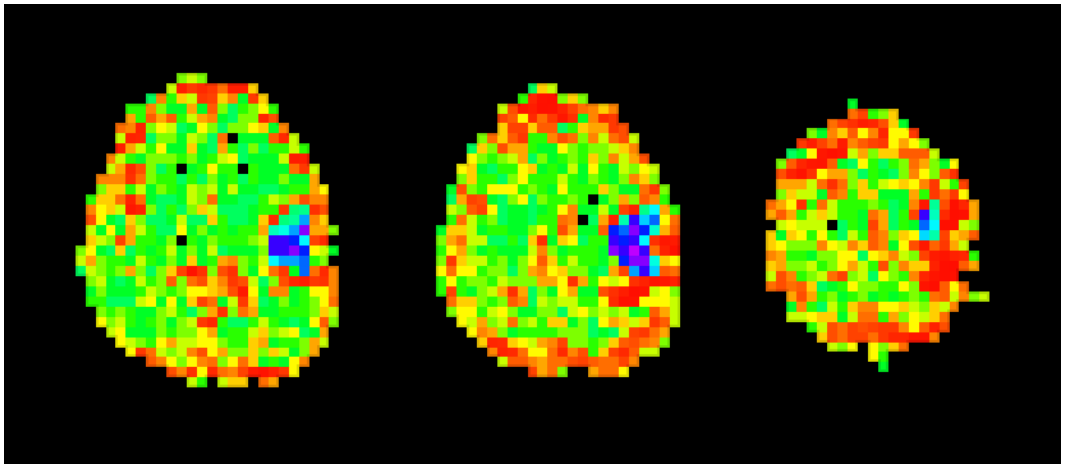


Figura 7.26: Método de sobreposição de ordem 0 sem HRF

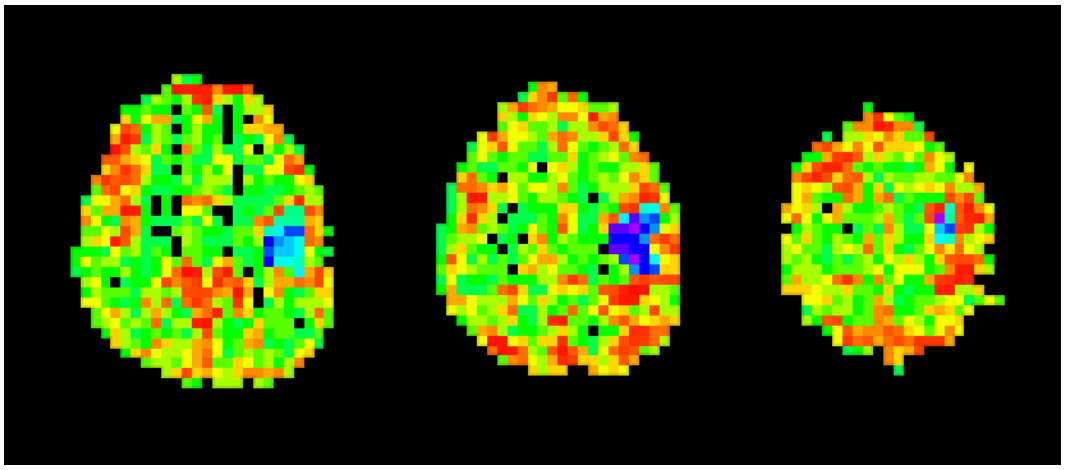


Figura 7.27: Método de sobreposição de ordem 1 sem HRF

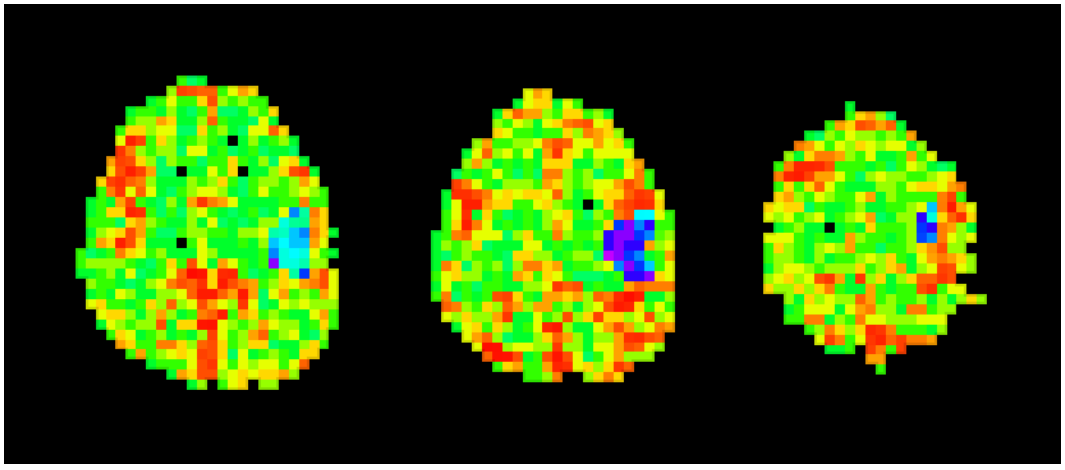


Figura 7.28: Método de sobreposição de ordem 2 sem HRF

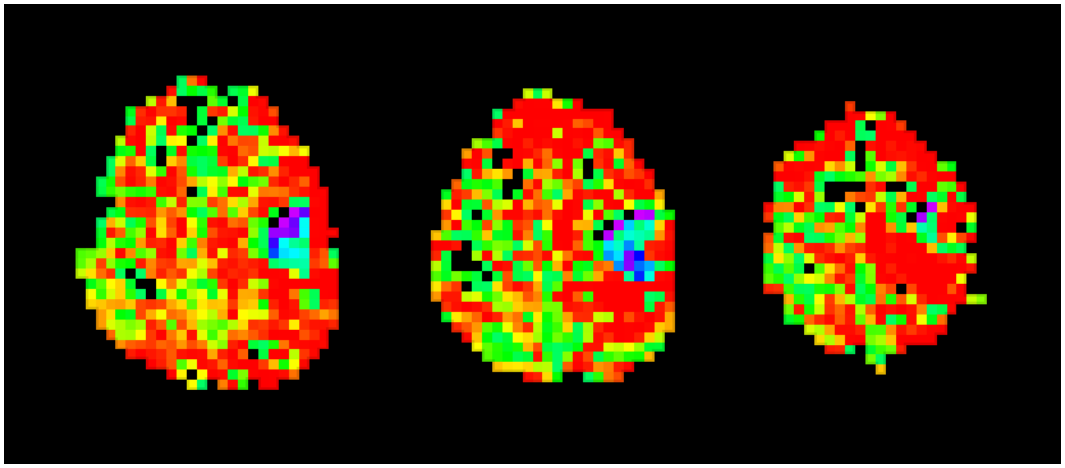


Figura 7.29: Método de coeficiente de correlação

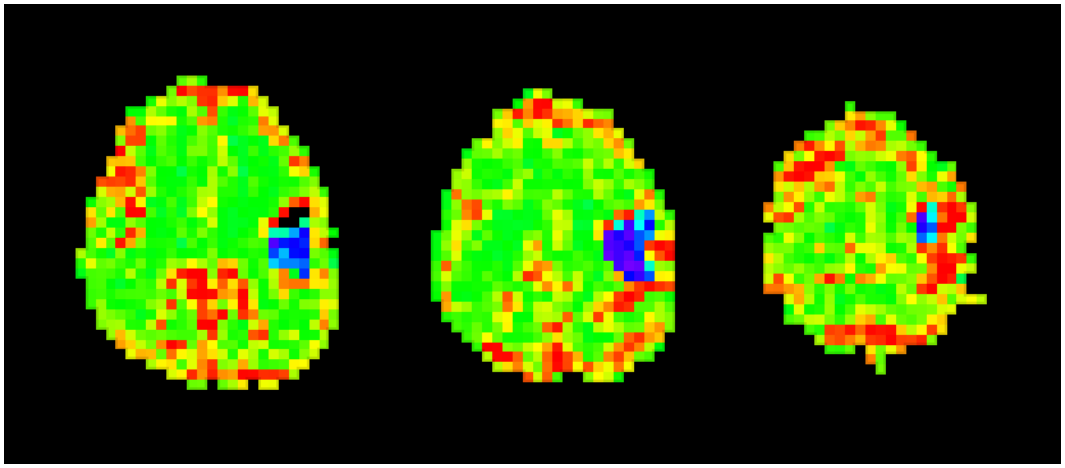


Figura 7.30: Método de picos de Fourier

7.2.4.2 Dados com HRF

Apresentam-se as imagens produzidas pelo programa *Comparador* da diferença entre os volumes de probabilidade de activação e tumor para cada método de análise funcional (figuras 7.29, 7.30, 7.31, 7.32, 7.33, 7.34). Os dados foram analisados assumindo uma sequência de modelo com HRF.

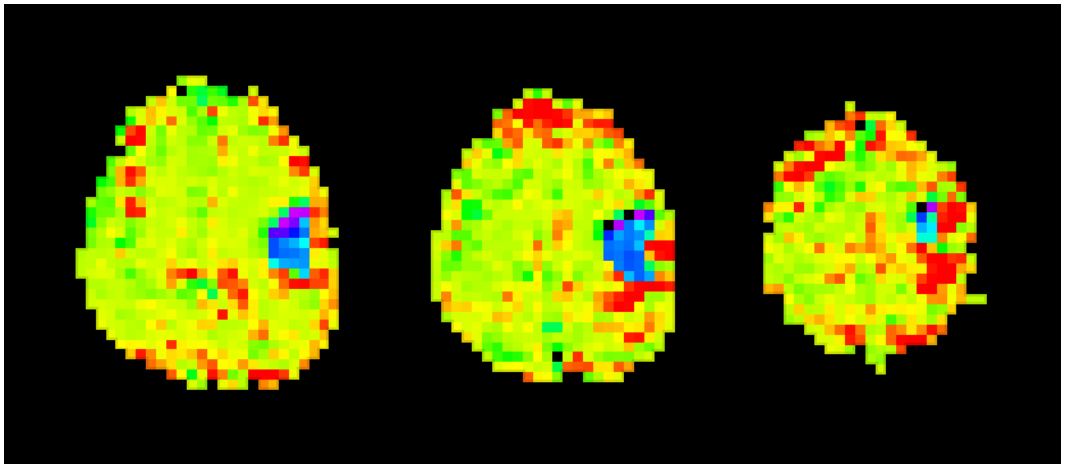


Figura 7.31: Método de amplitude

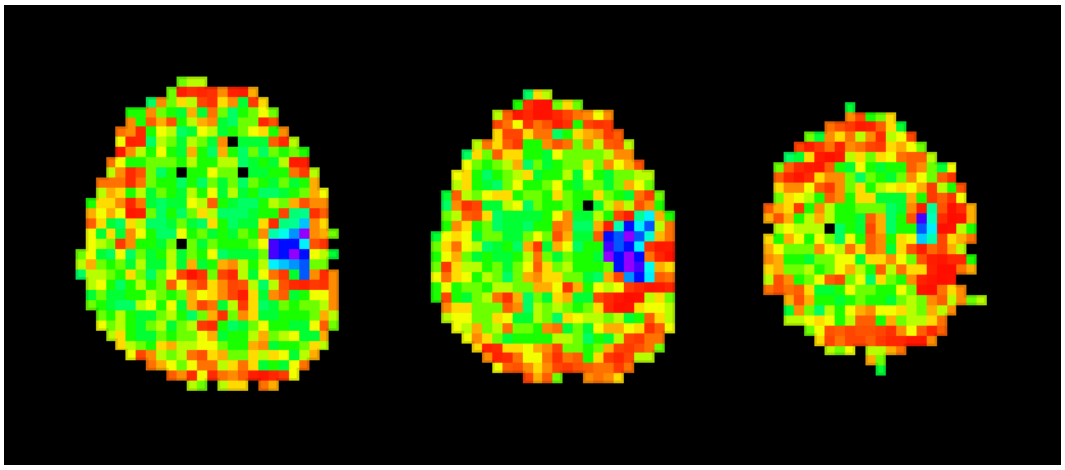


Figura 7.32: Método de sobreposição de ordem 0.

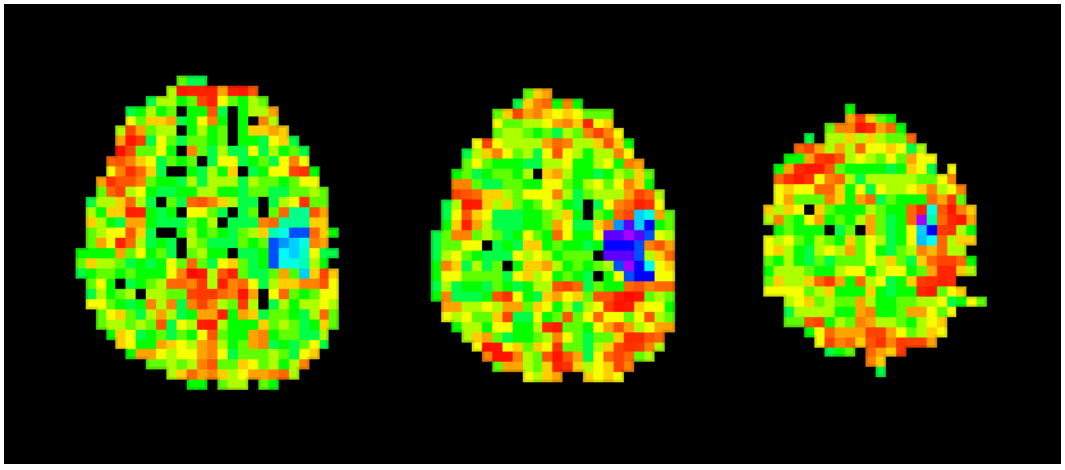


Figura 7.33: Método de sobreposição de ordem 1.

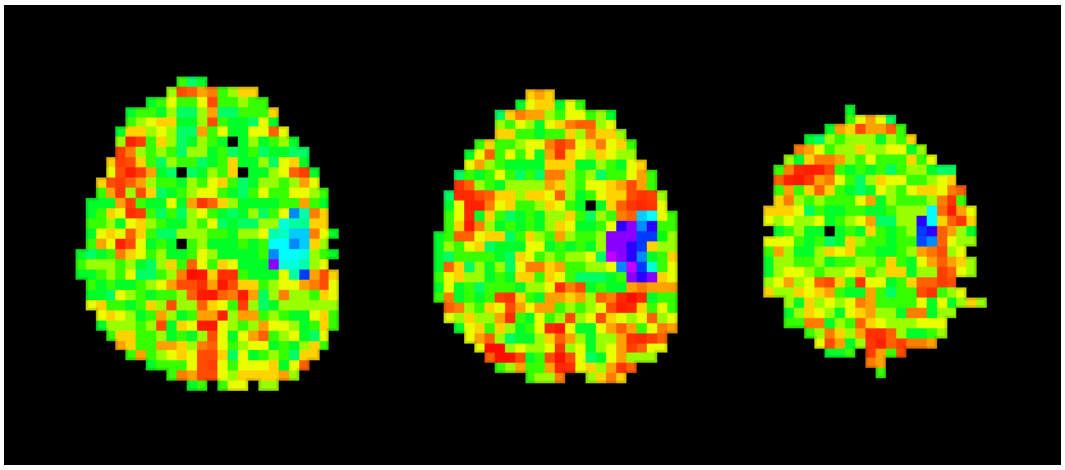


Figura 7.34: Método de sobreposição de ordem 2.

Capítulo 8

Conclusões gerais e trabalho futuro

Foi observado que há uma variação significativa dos resultados das análises funcionais com os métodos. Essa variabilidade tem uma influência directa significativa na avaliação da sobreposição entre as regiões de activação e de tumor.

Os métodos responsáveis por maiores volumes de activação foram (por ordem decrescente de volume): Coeficiente de Correlação, t de Student e Picos de Fourier. Os restantes métodos (Amplitude, MLG e Sobreposição de ordens 0, 1 e 2) formaram um grupo com volumes de activação semelhantes (ver tabela8.1).

As diferenças observadas entre os membros deste grupo não foram suficientes para estabelecer uma ordem. Houve no entanto uma excepção. Para dados filtrados, o método de Amplitude apresentou um volume de activação significativamente mais pequeno.

No estudo de atenção (ver secção4.8) foi identificado o melhor candidato a voxel de activação máxima (27,6,17). Esta identificação foi feita por comparação das sequências temporais dos máximos obtidos pelos vários métodos. Na tabela 8.2 os métodos são ordenados por ordem decrescente de proximidade do voxel de activação máxima em relação ao voxel (27,6,17).

Esta foi uma forma de avaliar a capacidade de cada método identificar o voxel de activação máxima. No topo da tabela está o grupo dos métodos de picos de Fourier, MLG, Amplitude e Sobreposição de ordem 0. No meio da tabela temos o grupo formado pelos métodos do Coeficiente de correlação e de Student. No fundo da tabela está consistentemente o método de sobreposição de ordem 2. Isto sugere que no topo estão os menos propensos a causar erros do tipo I.

É possível que uma filtragem de alta frequência dos dados melhore o desempenho do métodos de Sobreposição de ordens 1 e 2 (ver subsecção 4.8.2).

Na avaliação da distância tumor-zona de activação do caso clínico (ver o capítulo 7) confirmou-se o desempenho dos vários métodos em termos do volume de activação.

Apesar do volume de activação sobreposto com o tumor, obtido pelo método dos Picos de Fourier ser inferior ao obtido pelo método do Coeficiente de Correlação, a percentagem de volume de activação sobreposta com o tumor foi superior para o primeiro. Isto sugere que do primeiro

Filtrados (bloco)	Não filtrados (bloco)	Filtrados (HRF)	Não filtrados (HRF)
Coef. Corr.	Coef. Corr.	Coef. Corr.	Coef. Corr.
Student	Student	Fourier	Fourier
Fourier	Fourier	Sobrep.(0)	Amplitude
MLG	Amplitude	Sobrep.(1)	Sobrep.(0)
Sobrep.(0)	MLG	Sobrep.(2)	Sobrep.(1)
Sobrep.(1)	Sobrep.(0)	Amplitude	Sobrep.(2)
Sobrep.(2)	Sobrep.(1)		
Amplitude	Sobrep.(2)		

Tabela 8.1: Métodos de análise funcional ordenados por ordem decrescente de volume de activação

Filtrados (bloco)	Não filtrados (bloco)	Filtrados (HRF)	Não filtrados (HRF)
Fourier	Fourier	Fourier	Fourier
MLG	MLG	Amplitude	Amplitude
Amplitude	Sobrep.(0)	Sobrep.(0)	Sobrep.(0)
Sobrep.(0)	Amplitude	Coef. Corr.	Coef. Corr.
Coef. Corr.	Coef. Corr.	Sobrep.(1)	Sobrep.(1)
Student	Student	Sobrep.(2)	Sobrep.(2)
Sobrep.(1)	Sobrep.(1)		
Sobrep.(2)	Sobrep.(2)		

Tabela 8.2: Métodos de análise funcional ordenados por ordem decrescente de proximidade do voxel de activação máxima em relação ao voxel (27,6,17)

método resultou um volume de activação mais próximo do tumor.

Quanto aos máximos, foram mais próximos do voxel de activação máxima (38,26,6) para os métodos de Coeficiente de Correlação e de Student!

No futuro, a técnica dos perfis oblíquos poderá ser mais útil na avaliação de distâncias com uma visualização tridimensional do cérebro. Daí que essa será uma evolução natural na próxima versão do *Cérebro*.

Interessa também possibilitar uma filtragem espacial e comparar com os dados deste estudo.

Outro aspecto que estará sempre presente no programa *Cérebro* é procurar introduzir novos métodos de análise funcional (e.g. redes neuronais).

A interface dos parâmetros do modelo linear generalizado também deve melhorar para permitir tirar partido de outros tipos de análises que podem ser feitas com este método (e.g. análise de harmónicas, ANCOVA, etc.)

A filtragem temporal segundo o modelo também é uma área de interesse. Por exemplo, pode construír-se um filtro com valor unitário na zona dos picos do espectro do modelo.

Se subdividirmos os voxels do cérebro em grupos em função do atraso da sua sequência temporal podemos fazer uma avaliação da ordem em que ocorrem várias possíveis activações.

Por exemplo, com o método da sobreposição é possível estimar que a activação máxima do voxel (27,6,17) ocorre com um atraso de 2 TR. Há outros voxels que têm um máximo em outros instantes. Podemos então fazer um pequeno "filme" de activações.

Finalmente, a conjugação de outras fontes de sinal neuronal (tal como sinais de electro-encefalografia ou até de magneto-encefalografia) poderá melhorar ainda a identificação dos polos de activação.

Bibliografia

[1] Dennis Donnelly Bert Rust. The fast fourier transform for experimentalists, part iii: Classical spectral analysis. *Computing in Science and Engineering*, 7(5):74–78, Sep./Oct. 2005.

[2] Dennis Donnelly Bert Rust. The fast fourier transform for experimentalists, part iv: Autoregressive spectral analysis. *Computing in Science and Engineering*, 7(6):85–90, Nov./Dec. 2005.

[3] C. Buchel and K.J. Friston. Modulation of connectivity in visual pathways by attention: Cortical interactions evaluated with structural equation modelling and fmri. *Cerebral Cortex*, (7):768–778, 1997.

[4] Richard B. Buxton. *Introduction to functional magnetic resonance imaging: principles and techniques*. Cambridge University Press, 2002.

[5] G. Rees R. Turner C. Frith K. Friston C. Buchel, O. Josephs. The functional anatomy of attention to visual motion. a functional mri study. *Brain*, (121):1281–1294, 1998.

[6] Bert Rust Denis Donnelly. The fast fourier transform for experimentalists, part i: Concepts. *Computing in Science and Engineering*, 7(2):80–88, Mar./Apr. 2005.

[7] Bert Rust Denis Donnelly. The fast fourier transform for experimentalists, part ii: Convolutions. *Computing in Science and Engineering*, 7(4):92–95, Jul./Aug. 2005.

[8] Denis Donnelly. The fast fourier transform for experimentalists, part v: Filters. *Computing in Science and Engineering*, 8(1):92–95, Jan./Feb. 2006.

[9] M. D'Esposito G. K. Aguire, E. Zarahn. A critique of use of th e kolmogorov-smirnov (ks) statistic for the analysis of bold fmri data. *Magn Reson Med*, 50(4):926–932, 1998.

[10] M. D'Esposito G.K. Aguirre, E. Zarahn. The variability of human, bold hemodynamic responses. *Neuroimage*, 8:360–369, 1998.

[11] Mélanie Pélégrini-Issac Habib Benali Guillaume Marrelec, Philippe Ciuciu. Estimation of hemodinamic response function in event-related functional mri: Bayesian networks as a framework for efficient bayesian modeling and inference. *IEEE Transactions on medical imaging*, 23(8):959–967, August 2004.

[12] K. M. Harris A. H. Friedman T. M. George J. H. Sampson J. S. Pekala J. T. Voyvodic J. R. Petrella, L. M. Shah. Preoperative functional mr imaging localization of language and motor areas: effect on therapeutic decision making in patients with potentially resectable brain tumors. *Radiology*, 3(240):793–802, September 2006.

[13] J. Ashburner K. J. Friston, A. P. Holmes. Statistical parametric mapping (spm).

[14] M. Turner K. J. Friston, P. Jezzard. Analysis of functional mri time-series. *Human Brain Mapping*, 1(2):153–171, 1994.

[15] Stefan J. Kiebel Thomas E. Nichols William D. Penny K. J. Friston, J. T. Ashburner, editor. *Statistical Parametric Mapping. The Analysis of Functional Brain Images.* Elsevier, first edition, 2007.

[16] R. Turner C. J. Price K.J. Friston, A. Mechelli. Nonlinear responses in fmri: The balloon model, volterra kernels, and other hemodynamics. *NeuroImage*, (12):466–477, 2000.

[17] Michael A. Paradiso Mark F. Bear, Barry W. Connors. *Neuroscience: exploring the brain.* Lippincott Williams And Wilkins, 3 edition, 2007.

[18] Jorge S. Marques. *Reconhecimento de padrões: métodos estatísticos e neuronais.* IST Press, 1999.

[19] E. C. Wong J. S. Hyde P. A. Bandettini, A. Jesmanowicz. Processing strategies for time-course data sets in functional mri of the human brain. *Magn Reson Med*, (30):161–173, 1993.

[20] J. C. Gore P. Skudlarski, R. T. Constable. Roc analysis of statistical methods used in functional mri: individual subjects. *Neuroimage*, 9(3):311–329, Mar. 1999.

[21] W.H. Press, S.A. Teukolsky, A.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C++.* Cambridge University Press, second edition, 2002.

[22] W. Richter R Summers M. Jarmasz R. Somorjai R. Baumgartner, L. Ryner. Comparison of two exploratory data analysis methods for fmri: fuzzy clustering vs. principal component analysis. *Magn Reson Imaging*, 18(1):89–94, Jan. 2000.

Apêndice A

O formato Mayo Clinic Analyze

Linha	Byte	Secção	Nome	Tipo	Descrição
1	0	key	sizeof_hdr	int32	# bytes do hdr
2	4		data_type	uchar[10]	
3	14		db_name	uchar[18]	
4	32		extents	int32	
5	36		session_error	int16	
6	38		regular	uchar	r => vol.s iguais
7	39		hkey_un0	uchar	# dims (4)
8	40	dim	dim[0]	int16	# voxels seg. x
9	42		dim[1]	int16	# voxels seg. y
10	44		dim[2]	int16	# voxels seg. z
11	46		dim[3]	int16	# de volumes
12	48		dim[4]	int16	
13	50		dim[5]	int16	
14	52		dim[6]	int16	
15	54		dim[7]	int16	
16	56		vox_units	uchar[4]	unidades voxel
17	60		cal_units	uchar[8]	unidades calibração
18	68		unused1	int16	
19	70		datatype	int16	ver tabela abaixo
20	72		bitpix	int16	bits por voxel
21	74		dim_un0	int16	
22	76		pixdim[0]	float	
23	80		pixdim[1]	float	larg. voxel mm
24	84		pixdim[2]	float	alt. voxel mm
25	88		pixdim[3]	float	esp. fatia mm
26	92		pixdim[4]	float	
27	96		pixdim[5]	float	
28	100		pixdim[6]	float	
29	104		pixdim[7]	float	
30	108		vox_offset	float	
31	112		funused1	float	SPM: factor de escala
32	116		funused2	float	
33	120		funused3	float	
34	124		cal_max	float	valor max calibração
35	128		cal_min	float	valor min calibração

36	132		compressed	float	
37	136		verified	float	
38	140		glmax	int32	valor máximo
39	144		glmin	int32	valor mínimo
40	148	hist	descrip	uchar[80]	
41	228		aux_file	uchar[24]	
42	252		orient	uchar	orientação das fatias
43	253		originator	uchar[10]	x comissura anter.
44					y comissura anter.
45					z comissura anter.
46					
47					
48	263		generated	uchar[10]	
49	273		scannum	uchar[10]	
50	283		patient_id	uchar[10]	
51	293		exp_date	uchar[10]	
52	303		exp_time	uchar[10]	
53	313		hist_un0	uchar[10]	
54	316		views	int32	
55	320		vols_added	int32	
56	324		start_field	int32	
57	328		field_skip	int32	
58	332		omax	int32	
59	336		omin	int32	
60	340		smax	int32	
61	344		smin	int32	

datatype	Valor	Bits por voxel
DT_NONE	0	
DT_UNKNOWN	0	
DT_BINARY	1	1
DT_UNSIGNED_CHAR	2	8
DT_SIGNED_SHORT	4	16
DT_SIGNED_INT	8	32
DT_FLOAT	16	32
DT_COMPLEX	32	2×32
DT_DOUBLE	64	64
DT_RGB	128	
DT_ALL	255	

Apêndice B

A função Beta

$$B_x(a,b) = \int_0^x u^{a-1}(1-u)^{b-1}\,du \tag{B.1}$$

$$B(a,b) = \int_0^1 u^{a-1}(1-u)^{b-1}\,du \tag{B.2}$$

$$I_x(a,b) = \frac{B_x(a,b)}{B(a,b)} \tag{B.3}$$

$$B(a,b) = \frac{\Gamma(a)\,\Gamma(b)}{\Gamma(a+b)} \tag{B.4}$$

$$B(a,b) = B(b,a) \tag{B.5}$$

$$B_x(a,b) = B(b,a) - B_{1-x}(b,a) \tag{B.6}$$

Demonstremos esta igualdade:

$$B_x(a,b) = \int_0^x u^{a-1}(1-u)^{b-1}\,du$$

fazendo a mudança de variável:

$$v = 1-u$$

obtemos

$$\int_1^{1-x} (1-v)^{a-1} v^{b-1} \frac{du}{dv} dv$$

$$= \int_{1-x}^1 v^{b-1}(1-v)^{a-1}\,dv$$

$$= \int_0^1 v^{b-1}(1-v)^{a-1}\,dv - \int_0^{1-x} v^{b-1}(1-v)^{a-1}\,dv$$

$$= B(b,a) - B_{1-x}(b,a)$$

Podemos então concluir também que:

$$I_x(a,b) = 1 - I_{1-x}(b,a) \tag{B.7}$$

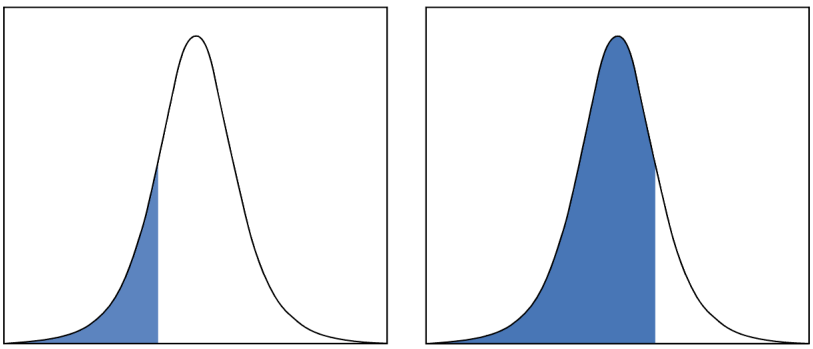
Apêndice C

Cálculo da distribuição cumulativa de Student

Integração da função densidade de probabilidade de Student:

$$\int_{-\infty}^{t} \phi(x,n)\,dx = \int_{-\infty}^{t} \frac{1}{\sqrt{n}B\left(\frac{1}{2},\frac{n}{2}\right)}\left(1+\frac{x^2}{n}\right)^{-\frac{n+1}{2}} dx$$

Como podemos ver na figura a seguir, a solução desta integração dependerá do sinal de t.



Uma forma de simplificar este problema é começar por tomar apenas o integral entre $-t$ e t:

$$\int_{-t}^{t} \phi(x,n)\,dx = \int_{-t}^{t} \frac{1}{\sqrt{n}B\left(\frac{1}{2},\frac{n}{2}\right)}\left(1+\frac{x^2}{n}\right)^{-\frac{n+1}{2}} dx$$

Como $\phi(x,n)$ é par em relação a x (ou seja: $\phi(-x,n) = \phi(x,n)$):

$$\int_{-t}^{t} \phi(x,n)\,dx = 2\int_{0}^{t} \phi(x,n)\,dx$$

eliminando assim a dificuldade do t negativo. O integral fica então:

$$\frac{2}{\sqrt{n}B\left(\frac{1}{2},\frac{n}{2}\right)}\int_{0}^{t}\left(1+\frac{x^2}{n}\right)^{-\frac{n+1}{2}} dx$$

Se realizarmos a seguinte mudança de variável:

$$u = \left(1+\frac{x^2}{n}\right)^{-1}$$

então o integral ficará:

$$\frac{2}{\sqrt{n}B\left(\frac{1}{2},\frac{n}{2}\right)}\int_1^{\frac{n}{n+t^2}} u^{\frac{n+1}{2}}\frac{dx}{du}du$$

$$=\frac{2}{\sqrt{n}B\left(\frac{1}{2},\frac{n}{2}\right)}\int_1^{\frac{n}{n+t^2}} u^{\frac{n+1}{2}}\left(-\frac{1}{2}\sqrt{n}u^{-\frac{3}{2}}(1-u)^{-\frac{1}{2}}\right)du$$

$$=-\frac{1}{B\left(\frac{1}{2},\frac{n}{2}\right)}\int_1^{\frac{n}{n+t^2}} u^{\frac{n}{2}-1}(1-u)^{-\frac{1}{2}}\,du$$

$$=\frac{1}{B\left(\frac{1}{2},\frac{n}{2}\right)}\int_{\frac{n}{n+t^2}}^1 u^{\frac{n}{2}-1}(1-u)^{-\frac{1}{2}}\,du$$

$$=\frac{1}{B\left(\frac{1}{2},\frac{n}{2}\right)}\left(\int_0^1 u^{\frac{n}{2}-1}(1-u)^{-\frac{1}{2}}\,du-\int_0^{\frac{n}{n+t^2}} u^{\frac{n}{2}-1}(1-u)^{-\frac{1}{2}}\,du\right)$$

$$=\frac{1}{B\left(\frac{1}{2},\frac{n}{2}\right)}\left(B\left(\frac{n}{2},\frac{1}{2}\right)-B_{\frac{n}{n+t^2}}\left(\frac{n}{2},\frac{1}{2}\right)\right)$$

recorrendo às propriedades B.5 e B.7:

$$=1-I_{\frac{n}{n+t^2}}\left(\frac{n}{2},\frac{1}{2}\right)$$

$$=I_{\frac{t^2}{n+t^2}}\left(\frac{1}{2},\frac{n}{2}\right)$$

Apêndice D

Soluções das equações normais do modelo linear generalizado

Podemos reformular o problema da forma matricial:

$$\begin{bmatrix} v_0 \\ \vdots \\ v_i \\ \vdots \\ v_{n-1} \quad _1 \end{bmatrix} = \begin{bmatrix} u_{00} & \cdots & u_{0j} & \cdots & u_{0m-1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ u_{i0} & \cdots & u_{ij} & \cdots & u_{im-1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ u_{n-10} & \cdots & u_{n-1j} & \cdots & u_{n-1m-1} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_j \\ \vdots \\ a_{m-1} \end{bmatrix}$$

ou seja:

$$\hat{v}_i = \hat{u}_{ij}\hat{a}_j$$

Para determinar os coeficientes $\hat{a}_j$ temos que minimizar a soma dos quadrados dos resíduos:

$$S = \left(\hat{v}_i - \hat{u}_{ij}\hat{a}_j\right)^T \left(\hat{v}_i - \hat{u}_{ij}\hat{a}_j\right)$$

logo:

$$\frac{\partial S}{\partial a_j} = 0$$

$$2\hat{u}_{ij}^T\hat{v}_i - 2\hat{u}_{ij}^T\hat{u}_{ij}\hat{a}_j = 0$$

A solução é então:

$$\hat{a}_j = \left(\hat{u}_{ij}^T\hat{u}_{ij}\right)^{-1}\hat{u}_{ij}^T\hat{v}_i$$

Apêndice E

O código do *Cérebro*

```java
//
//  xCerebro.java
//  cerebro
//
//  Created by Jose Gabriel Lira Gomes on 6/13/06.
//  Copyright (c) 2006 __FDG__. All rights reserved.
//

import javax.swing.JFrame;

import fMRIWindows.MainWindow;

public class xCerebro extends JFrame
{
    public xCerebro()
    {
        new MainWindow();
    }

    public static void main(String args[])
    {
        new xCerebro();
    }
}
```

```java
package fMRIWindows;

//
//  MainWindow.java
//
//
//  Created by JGLG on 4/27/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

import imageUtils.BlockImageBuilder;

import java.awt.Color;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.io.IOException;

import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;

import matematica.fourier.Convolver;
import estatistica.descriptive.FourDArrayAnalyzer;
//import fMRIWindows.cleaners.ParamWindowBackgroundCleaner;
import fMRIWindows.cleaners.ParamWindowHighPassFilter;
import fMRIWindows.cleaners.ParamWindowOutliersCleaner;
import fMRIWindows.design.TimelineWindowBlockAcquisitionParams;
import fMRIWindows.design.TimelineWindowEventAcquisitionParams;
import fMRIWindows.design.TimelineWindowSetter;
import fMRIWindows.design.TimelineWindowStates;
import fMRIWindows.methodParams.ParamWindowAmplitude;
import fMRIWindows.methodParams.ParamWindowCorr;
import fMRIWindows.methodParams.ParamWindowFFT;
import fMRIWindows.methodParams.ParamWindowGLM;
import fMRIWindows.methodParams.ParamWindowOverlap;
import fMRIWindows.methodParams.ParamWindowStudent;
import fMRIWindows.segmenters.ParamWindowClusterSampler;
import fMRIWindows.viewers.ViewerWindowData;
import fMRIWindows.viewers.ViewerWindowDataArrays;
import fMRIWindows.viewers.ViewerWindowDataMask;
import fMRIWindows.viewers.ViewerWindowDesignMatrix;
import fMRIWindows.viewers.ViewerWindowProfiles;
import fMRIWindows.viewers.ViewerWindowResult;
import fileUtils.ExtensionsFileFilter;
import fileUtils.FilePathGetter;
import functional.data.AnlzHeader;
import functional.data.AnlzReader;
import functional.data.AnlzWriter;
import functional.data.Cluster;
import functional.data.DesignMatrix;
import functional.data.Mask;
import functional.data.Timeline;
import functional.data.Voxel;
import functional.filter.DataFilter;
import functional.hrf.Hrf;
import functional.methods.AmplitudeAnalysis;
import functional.methods.CorrAnalysis;
import functional.methods.FFourierAnalysis;
import functional.methods.GlmAnalysis;
import functional.methods.OverlapAnalysis;
import functional.methods.TStudentAnalysis;
import functional.outliers.DataOutliersCleaner;
import functional.segment.ClusterSegmenter;
import functional.segment.DataMaskBuilder;

public class MainWindow extends MenuedFrame
{
```

```
//
//  Data Members
//

private double[][][][] usedImagingData;
private double[][][][] usedSegmentData;
private double[][][][] usedAnalysisData;
private String currentFilePath;
private boolean backgroundClean = false;
private boolean outliersClean = false;
private boolean filtered = false;

private Container mainContainer;

private JTabbedPane mainTabbedPane;

private JPanel loadDataPanel;
private JPanel getDataMaskPanel;
private JPanel setDesignPanel;
private JPanel cleanDataPanel;
private JPanel viewDataPanel;
private JPanel segmentDataPanel;
private JPanel setMethodPanel;

private JButton loadDataButton;
private JButton loadMaskButton;
private JButton simpleMaskButton;
private JButton timelineButton;
private JButton spectrumButton;
//private JButton cleanBackgroundButton;
private JButton cleanOutliersButton;
private JButton cleanLowFreqsButton;
private JButton viewDataButton;
private JButton viewDesignButton;
private JButton viewDataMaskButton;
private JButton setStatesButton;

private JButton setSequenceButton;
private JButton setAcquisitionParamsButton;

private JButton clustersSegmentButton;
private Cluster[] sampleClusters;
private JCheckBox tumorSelector;
private boolean withTumor;

private JButton setMethodParamsButton;
private JButton calculateButton;

private String[] comboBoxOptions =
                {"Metodo", "t de Student", "Correlacao",
                "Fourier", "GLM", "Amplitude", "Sobreposicao"};
private JComboBox methodComboBox;
private String selectedMethod;

private AnlzReader anlzFile;
private FilePathGetter pathGetter;
private String[] validExtensions = {"hdr"};
private ExtensionsFileFilter extensionsFilter =
            new ExtensionsFileFilter(validExtensions);
private String filePath;
private short[] imageDims;
private double[][][][] imagingData;
private double[][][][] analysisData;
private double[][][] bckgVolume;
private boolean complexData;
private int width;
private int height;
private int numberOfSlices;
private int numberOfInstants;
private double numberOfVoxels;
```

```
private Mask dataMask;
private int numberOfMaskVoxels;
private short[] maskDims;

private double[] timeline;
private boolean timelinePresent = false;
private boolean blockStudy = false;
private DesignMatrix designMatrix;
private double acquisitionTime;
private int samplesPerCycle;

private String[] statesAcronyms;
private String[] statesNames;
private int numberOfStates;

private boolean fftNotPerformed = true;

//private double[][][][] backgroundCleanImagingData;
//private double[][][][] backgroundCleanAnalysisData;
//private int numberOfBackgroundVoxels;
//private double percentageOfBackgroundVoxels;
//private AnlzHeader backgroundCleanDataHeader;

private double outliersThresholdProb;
private double outliersFraction;
private double[][][][] outlierCleanImagingData;
private double[][][][] outlierCleanAnalysisData;
private double[][][][] cleanDataSpectrum;
private int numberOfOutliers;
private double percentageOfOutliers;
private AnlzHeader outlierCleanDataHeader;

private double cutFrequency;
private DataFilter highPassFilter;
private double[][][][] filteredData;
private AnlzHeader filteredDataHeader;

private double[] hrf;

private int[] tStudentSelectedStates = new int[2];
private int tSudentLag;
private int tStudentModelType;
private double[][][] tTestResultData;
private TStudentAnalysis tAnalyzer;

private int[] corrSelectedStates = new int[2];
private int corrLag;
private int corrModelType;
private Voxel corrModelVoxel;
private double[] corrDesignVector;
private double[] corrModelArray;
private double[][][] corrTestResultData;
private CorrAnalysis corrAnalyzer;

private int[] fFourierSelectedStates = new int[2];
private int fFourierLag;
private int fFourierModelType;
private Voxel fFourierModelVoxel;
private int fFourierDataWindow;
private int fFourierModelWindow;
private int fFourierInfMethod;
private double[] fFourierDesignVector;
private double[] fFourierModelArray;
private double[][][] fFourierTestResultData;
private FFourierAnalysis fFourierAnalyzer;

private int[] glmSelectedStates = new int[2];
private int glmLag;
private int glmModelType;
private Voxel glmModelVoxel;
private int glmInferenceType;
```

```
private double[] glmDesignVector;
private double[] glmModelArray;
private double[][][] glmTestResultData;
private GlmAnalysis glmAnalyzer;

private int[] amplitudeSelectedStates = new int[2];
private int amplitudeLag;
private int amplitudeModelType;
private Voxel amplitudeModelVoxel;
private double[] amplitudeDesignVector;
private double[] amplitudeModelArray;
private double[][][] amplitudeTestResultData;
private AmplitudeAnalysis amplitudeAnalyzer;

private int[] overlapSelectedStates = new int[2];
private int overlapLag;
private int overlapModelType;
private Voxel overlapModelVoxel;
private double[] overlapDesignVector;
private double[] overlapModelArray;
private int overlapOrder;
private double[][][] overlapTestResultData;
private OverlapAnalysis overlapAnalyzer;

private long startTime;
private long stopTime;
private long elapsedTime;

private int mainWindowWidth = 520;
//private int mainWindowHeight = 140;
private int mainWindowHeight = 120;
private int mainWindowXPosition = 0;
private int mainWindowYPosition = 21;

//
//  Constructors
//
//      MainWindow()
//

public MainWindow()
{
    createGUI();
}

public double[][][][] getUsedImagingData()
{
    return usedImagingData;
}

public double[][][] getImageDataAtInstant(int instant)
{
    return extractInstantVolume(instant, false);
}

public short[] getImageDims()
{
    return imageDims;
}

public int getVolumesNumberOfVoxels()
{
    return width*height*numberOfSlices;
}

public float[] getPixelsDims()
{
    return anlzFile.getPixelsDims();
}

public double getVoxelVolume()
```

```
{
    double voxelVolume = 1.0;
    float[] pixelsDims;

    pixelsDims = getPixelsDims();
    for(int i=1; i<4; i++)
    {
        voxelVolume = voxelVolume*pixelsDims[i]/10.0;
    }

    return voxelVolume;
}

public float getFUnusedOne()
{
    return anlzFile.getFUnusedOne();
}

public AnlzReader getAnlzFile()
{
    return anlzFile;
}

public Mask getDataMask()
{
    return dataMask;
}

public void setDataMask(Mask dataMask)
{
    this.dataMask = dataMask;
}

public int getNumberOfMaskVoxels()
{
    return numberOfMaskVoxels;
}

public int getNumberOfBckgVoxels()
{
    return dataMask.getNumberOfNonMaskVoxels();
}

public double getBckgProb()
{
    return (double)getNumberOfBckgVoxels()
                    /(double)getVolumesNumberOfVoxels();
}

public void setStatesData(String[][] statesData)
{
    this.statesAcronyms = extractColumnFromDualArray(statesData, 0);
    this.statesNames = extractColumnFromDualArray(statesData, 1);
    this.numberOfStates = statesAcronyms.length;
}

public String[] getStatesAcronyms()
{
    return statesAcronyms;
}

public int getNumberOfStates()
{
    return numberOfStates;
}

public void enableSetSequenceButton()
{
    this.setSequenceButton.setEnabled(true);
}
```

```
public double[] getTimeline()
{
    return timeline;
}

public void setTimeline(double[] timeline)
{
    this.timeline = timeline;
    this.timelinePresent = true;
}

public void setBlockStudy()
{
    this.blockStudy = true;
}

public DesignMatrix getDesignMatrix()
{
    return designMatrix;
}

public void setDesignMatrix(DesignMatrix designMatrix)
{
    this.designMatrix = designMatrix;
}

public void enableSetAcquisitionParamsButton()
{
    this.setAcquisitionParamsButton.setEnabled(true);
}

public void setAcquisitionTime(double acquisitionTime)
{
    this.acquisitionTime = acquisitionTime;
}

public void setSamplesPerCycle(int samplesPerCycle)
{
    this.samplesPerCycle = samplesPerCycle;
}

public void enableTumorSelector()
{
    this.tumorSelector.setEnabled(true);
}

public void enableClustersSegmentButton()
{
    this.clustersSegmentButton.setEnabled(true);
}

public void enableViewDesignButton()
{
    this.viewDesignButton.setEnabled(true);
}

/*
public void enableCleanBackgroundButton()
{
    this.cleanBackgroundButton.setEnabled(true);
}
*/
/*
public void cleanBackground()
{
    System.out.println("Inicio da limpeza de fundo");
    startTime = System.currentTimeMillis();

    backgroundCleanImagingData =
        dataMask.getMaskedData(usedImagingData);
    backgroundCleanAnalysisData =
```

```
    dataMask.getMaskedData(usedAnalysisData);

stopTime = System.currentTimeMillis();
System.out.println("Fim da limpeza de fundo");

numberOfBackgroundVoxels = dataMask.getNumberOfNonMaskVoxels();

elapsedTime = stopTime - startTime;
System.out.println(numberOfBackgroundVoxels
                   + " voxels de fundo detectados");
percentageOfBackgroundVoxels = (double)numberOfBackgroundVoxels
                               *numberOfInstants
                               /numberOfVoxels;
System.out.println(Double.toString(
                  Math.round(percentageOfBackgroundVoxels*100.0))
                  + " % de todos os pontos");
System.out.println("Tempo de limpeza de fundo:");
System.out.println(elapsedTime + " ms");
System.out.println("");

backgroundClean = true;

if(complexData)
{
    usedAnalysisData = backgroundCleanAnalysisData;
}
else
{
    usedAnalysisData = backgroundCleanImagingData;
}
usedImagingData = backgroundCleanImagingData;

backgroundCleanDataHeader = new AnlzHeader();
if(complexData)
{
    backgroundCleanDataHeader.setDataType("cpl");
}
else
{
    backgroundCleanDataHeader.setDataType("dbl");
}
backgroundCleanDataHeader.setNumberOfDims((short)4);
backgroundCleanDataHeader.setDimensions(imageDims);
if(complexData)
{
    backgroundCleanDataHeader.setDataTypeCode((short)32);
}
else
{
    backgroundCleanDataHeader.setDataTypeCode((short)64);
}
backgroundCleanDataHeader.setBitsPerPixel((short)64);
backgroundCleanDataHeader.setPixelsDims(anlzFile.getPixelsDims());
backgroundCleanDataHeader.setFUnusedOne(anlzFile.getFUnusedOne());
FourDArrayAnalyzer dataAnalyzer =
            new FourDArrayAnalyzer(usedImagingData);
backgroundCleanDataHeader.setGLMax((int)Math.round
                                    (dataAnalyzer.getMaximum()));
backgroundCleanDataHeader.setGLMin((int)Math.round
                                    (dataAnalyzer.getMinimum()));

if(backgroundClean)
{
    backgroundCleanDataHeader.addDescrip("bkgClean ");
}
if(outliersClean)
{
    backgroundCleanDataHeader.addDescrip("outClean ");
}
if(filtered)
{
```

```
            backgroundCleanDataHeader.addDescrip("filtered ");
        }

        currentFilePath = concatenateStrings(currentFilePath, "b");

        System.out.println("Inicio da escrita dos dados sem fundo");
        startTime = System.currentTimeMillis();
        try
        {
            if(complexData)
            {
                new AnlzWriter(backgroundCleanDataHeader,
                               usedImagingData,
                               currentFilePath);
            }
            else
            {
                new AnlzWriter(backgroundCleanDataHeader,
                               usedImagingData,
                               usedAnalysisData,
                               currentFilePath);
            }
        }
        catch (IOException e)
        {
            System.out.println("failed analyze file writing");
            e.printStackTrace();
        }
        stopTime = System.currentTimeMillis();
        System.out.println("Fim da escrita dos dados sem fundo");
        elapsedTime = stopTime - startTime;
        System.out.println("Tempo da escrita dos dados sem fundo:");
        System.out.println(elapsedTime + " ms");
        System.out.println("");
    }
    */

    public void enableCleanOutliersButton()
    {
        this.cleanOutliersButton.setEnabled(true);
    }

    public void setOutliersThresholdProb(double outliersThresholdProb)
    {
        this.outliersThresholdProb = outliersThresholdProb;
    }

    public void setOutliersFraction(double outliersFraction)
    {
        this.outliersFraction = outliersFraction;
    }

    public void exportZeroesAnlz()
    {
        double[][][][] zeroesArray =
            new double[width][height][numberOfSlices][1];
        String zeroesFilePath;

        for(int z=0; z< numberOfSlices; z++)
        {
            for(int y=0; y< height; y++)
            {
                for(int x=0; x< width; x++)
                {
                    zeroesArray[x][y][z][0] = 0.0;
                }
            }
        }

        short[] imageDims = new short[8];
        imageDims[0] = 4;
```

```
        imageDims[1] = (short)width;
        imageDims[2] = (short)height;
        imageDims[3] = (short)numberOfSlices;
        imageDims[4] = 1;

        AnlzHeader zeroesHeader = new AnlzHeader();

        zeroesHeader.setDataType("dbl");
        zeroesHeader.setNumberOfDims((short)4);
        zeroesHeader.setDimensions(imageDims);
        zeroesHeader.setDataTypeCode((short)64);
        zeroesHeader.setBitsPerPixel((short)64);
        zeroesHeader.setPixelsDims(anlzFile.getPixelsDims());
        zeroesHeader.setFUnusedOne(anlzFile.getFUnusedOne());
        zeroesHeader.setGLMax(1);
        zeroesHeader.setGLMin(0);
        zeroesHeader.setDescrip("Zeroes");

        zeroesFilePath = concatenateStrings(currentFilePath, "z");

        System.out.println("Inicio da escrita dos dados de zeros");
        startTime = System.currentTimeMillis();
        try
        {
            new AnlzWriter(zeroesHeader, zeroesArray, zeroesFilePath);
        }
        catch (IOException e)
        {
            System.out.println("failed analyze file writing");
            e.printStackTrace();
        }
        stopTime = System.currentTimeMillis();
        System.out.println("Fim da escrita dos dados de zeros");
        elapsedTime = stopTime - startTime;
        System.out.println("Tempo da escrita dos dados de zeros:");
        System.out.println(elapsedTime + " ms");
        System.out.println("");
    }

    public void cleanOutliers()
    {
        DataOutliersCleaner outlierCleaner;

        System.out.println("Inicio da limpeza de outliers");
        startTime = System.currentTimeMillis();
        if(complexData)
        {
            outlierCleaner = new DataOutliersCleaner(usedAnalysisData,
                                                     outliersThresholdProb,
                                                     outliersFraction);
            outlierCleanAnalysisData = outlierCleaner.getCleanData();
        }
        outlierCleaner = new DataOutliersCleaner(usedImagingData,
                                                 outliersThresholdProb,
                                                 outliersFraction);
        outlierCleanImagingData = outlierCleaner.getCleanData();
        numberOfOutliers = outlierCleaner.getNumberOfOutliers();
        stopTime = System.currentTimeMillis();
        System.out.println("Fim da limpeza de outliers");

        elapsedTime = stopTime - startTime;
        System.out.println(numberOfOutliers + " outliers detectados");
        percentageOfOutliers = (double)numberOfOutliers/numberOfVoxels;
        System.out.println(Double.toString(
                           Math.round(percentageOfOutliers*100.0))
                           + " % de todos os pontos");
        System.out.println("Tempo de limpeza de outliers:");
        System.out.println(elapsedTime + " ms");
        System.out.println("");

        outliersClean = true;
```

```
if(complexData)
{
    usedAnalysisData = outlierCleanAnalysisData;
}
else
{
    usedAnalysisData = outlierCleanImagingData;
}
usedImagingData = outlierCleanImagingData;

outlierCleanDataHeader = new AnlzHeader();
if(complexData)
{
    outlierCleanDataHeader.setDataType("cpl");
}
else
{
    outlierCleanDataHeader.setDataType("dbl");
}
outlierCleanDataHeader.setNumberOfDims((short)4);
outlierCleanDataHeader.setDimensions(imageDims);
if(complexData)
{
    outlierCleanDataHeader.setDataTypeCode((short)32);
}
else
{
    outlierCleanDataHeader.setDataTypeCode((short)64);
}
outlierCleanDataHeader.setBitsPerPixel((short)64);
outlierCleanDataHeader.setPixelsDims(anlzFile.getPixelsDims());
outlierCleanDataHeader.setFUnusedOne(anlzFile.getFUnusedOne());
FourDArrayAnalyzer dataAnalyzer =
            new FourDArrayAnalyzer(usedImagingData);
outlierCleanDataHeader.setGLMax(
                (int)Math.round(dataAnalyzer.getMaximum()));
outlierCleanDataHeader.setGLMin(
                (int)Math.round(dataAnalyzer.getMinimum()));

if(backgroundClean)
{
    outlierCleanDataHeader.addDescrip("bkgClean ");
}
if(outliersClean)
{
    outlierCleanDataHeader.addDescrip("outClean ");
}
if(filtered)
{
    outlierCleanDataHeader.addDescrip("filtered ");
}

currentFilePath = concatenateStrings(currentFilePath, "c");

System.out.println("Inicio da escrita dos dados sem outliers");
startTime = System.currentTimeMillis();
try
{
    if(complexData)
    {
        new AnlzWriter(outlierCleanDataHeader,
                       usedImagingData,
                       usedAnalysisData,
                       currentFilePath);
    }
    else
    {
        new AnlzWriter(outlierCleanDataHeader,
                       usedImagingData,
                       currentFilePath);
```

```
            }
        }
        catch (IOException e)
        {
            System.out.println("failed analyze file writing");
            e.printStackTrace();
        }
        stopTime = System.currentTimeMillis();
        System.out.println("Fim da escrita dos dados sem outliers");
        elapsedTime = stopTime - startTime;
        System.out.println("Tempo da escrita dos dados sem outliers:");
        System.out.println(elapsedTime + " ms");
        System.out.println("");
    }

    public void enableCleanLowFreqsButton()
    {
        this.cleanLowFreqsButton.setEnabled(true);
    }

    public void setCutFrequency(double cutFrequency)
    {
        this.cutFrequency = cutFrequency;
    }

    public void cleanLowFreqs()
    {
        System.out.println("Inicio da filtragem de baixas frequencias");
        startTime = System.currentTimeMillis();
        if(!complexData)
        {
            highPassFilter = new DataFilter(usedAnalysisData, acquisitionTime);
            filteredData = highPassFilter.getIdealHPFilteredData(cutFrequency);
        }
        stopTime = System.currentTimeMillis();
        elapsedTime = stopTime - startTime;
        System.out.println("Fim da filtragem de baixas frequencias");
        System.out.println("Tempo de filtragem de baixas frequencias:");
        System.out.println(elapsedTime + " ms");
        System.out.println("");

        filtered = true;

        usedAnalysisData = filteredData;

        filteredDataHeader = new AnlzHeader();
        filteredDataHeader.setDataType("cpl");
        filteredDataHeader.setNumberOfDims((short)4);
        filteredDataHeader.setDimensions(imageDims);
        filteredDataHeader.setDataTypeCode((short)32);
        filteredDataHeader.setBitsPerPixel((short)64);
        filteredDataHeader.setPixelsDims(anlzFile.getPixelsDims());
        filteredDataHeader.setFUnusedOne(anlzFile.getFUnusedOne());
        FourDArrayAnalyzer dataAnalyzer =
                    new FourDArrayAnalyzer(usedImagingData);
        filteredDataHeader.setGLMax((int)Math.round(dataAnalyzer.getMaximum()));
        filteredDataHeader.setGLMin((int)Math.round(dataAnalyzer.getMinimum()));

        if(backgroundClean)
        {
            filteredDataHeader.addDescrip("bkgClean ");
        }
        if(outliersClean)
        {
            filteredDataHeader.addDescrip("outClean ");
        }
        if(filtered)
        {
            filteredDataHeader.addDescrip("filtered ");
        }
```

```
    currentFilePath = concatenateStrings(currentFilePath, "f");

    System.out.println("Inicio da escrita dos dados filtrados");
    startTime = System.currentTimeMillis();
    try
    {
        new AnlzWriter(filteredDataHeader,
                       usedImagingData,
                       usedAnalysisData,
                       currentFilePath);
    }
    catch (IOException e)
    {
        System.out.println("failed analyze file writing");
        e.printStackTrace();
    }
    stopTime = System.currentTimeMillis();
    System.out.println("Fim da escrita dos dados filtrados");
    elapsedTime = stopTime - startTime;
    System.out.println("Tempo da escrita dos dados filtrados:");
    System.out.println(elapsedTime + " ms");
    System.out.println("");
}

public boolean imageHasBackground()
{
    return !backgroundClean;
}

public boolean imageHasOutliers()
{
    return !outliersClean;
}

public boolean imageHasLowFreqs()
{
    return !filtered;
}

public void enableSegmentButton()
{
    this.clustersSegmentButton.setEnabled(true);
}

public Cluster getSampleCluster(int tissueIndex)
{
    return sampleClusters[tissueIndex];
}

public void setSampleCluster(Cluster sampleCluster, int tissueIndex)
{
    this.sampleClusters[tissueIndex] = sampleCluster;
}

public void setUsedSegmentData(double[][][][] usedSegmentData)
{
    this.usedSegmentData = usedSegmentData;
}

public Cluster[] getSampleClusters()
{
    return sampleClusters;
}

public void setSampleClusters(Cluster[] sampleClusters)
{
    this.sampleClusters = sampleClusters;
}

public void segmentImageTissue(int tissueIndex)
{
```

```
        ClusterSegmenter segmenter;

        segmenter = new ClusterSegmenter(usedSegmentData, sampleClusters, this);

        new ViewerWindowResult(segmenter.getClusterMembershipProbs(tissueIndex),
                               bckgVolume,
                               numberOfMaskVoxels,
                               4,
                               "Segmentacao",
                               this);
    }

    public void enableMethodComboBox()
    {
        this.methodComboBox.setEnabled(true);
    }

    public void enableSetMethodParamsButton()
    {
        this.setMethodParamsButton.setEnabled(true);
    }

    public void setTStudentSelectedStates(int[] tStudentSelectedStates)
    {
        this.tStudentSelectedStates = tStudentSelectedStates;
    }

    public void setTStudentLag(int tStudentLag)
    {
        this.tSudentLag = tStudentLag;
    }

    public void setTStudentModelType(int tStudentModelType)
    {
        this.tStudentModelType = tStudentModelType;
    }

    public void setCorrSelectedStates(int[] corrSelectedStates)
    {
        this.corrSelectedStates = corrSelectedStates;
    }

    public void setCorrLag(int corrLag)
    {
        this.corrLag = corrLag;
    }

    public void setCorrModelType(int corrModelType)
    {
        this.corrModelType = corrModelType;
    }

    public void setCorrModelVoxel(Voxel corrModelVoxel)
    {
        this.corrModelVoxel = corrModelVoxel;
    }

    public double[] getCorrModelArray()
    {
        return corrModelArray;
    }

    public void setFFourierSelectedStates(int[] fFourierSelectedStates)
    {
        this.fFourierSelectedStates = fFourierSelectedStates;
    }

    public void setFFourierLag(int fFourierLag)
    {
        this.fFourierLag = fFourierLag;
    }
```

```
public void setFFourierModelType(int fFourierModelType)
{
    this.fFourierModelType = fFourierModelType;
}

public void setFFourierModelVoxel(Voxel fFourierModelVoxel)
{
    this.fFourierModelVoxel = fFourierModelVoxel;
}

public void setFFourierDataWindow(int fFourierDataWindow)
{
    this.fFourierDataWindow = fFourierDataWindow;
}

public void setFFourierModelWindow(int fFourierModelWindow)
{
    this.fFourierModelWindow = fFourierModelWindow;
}

public void setFFourierInfMethod(int fFourierInfMethod)
{
    this.fFourierInfMethod = fFourierInfMethod;
}

public double[] getFFourierModelArray()
{
    return fFourierModelArray;
}

public void setGLMSelectedStates(int[] glmSelectedStates)
{
    this.glmSelectedStates = glmSelectedStates;
}

public void setGLMLag(int glmLag)
{
    this.glmLag = glmLag;
}

public void setGLMModelType(int glmModelType)
{
    this.glmModelType = glmModelType;
}

public void setGLMModelVoxel(Voxel glmModelVoxel)
{
    this.glmModelVoxel = glmModelVoxel;
}

public void setGLMInferenceType(int glmInferenceType)
{
    this.glmInferenceType = glmInferenceType;
}

public double[] getGLMModelArray()
{
    return glmModelArray;
}

public void setAmplitudeSelectedStates(int[] amplitudeSelectedStates)
{
    this.amplitudeSelectedStates = amplitudeSelectedStates;
}

public void setAmplitudeLag(int amplitudeLag)
{
    this.amplitudeLag = amplitudeLag;
}
```

```
public void setAmplitudeModelType(int amplitudeModelType)
{
    this.amplitudeModelType = amplitudeModelType;
}

public void setAmplitudeModelVoxel(Voxel amplitudeModelVoxel)
{
    this.amplitudeModelVoxel = amplitudeModelVoxel;
}

public double[] getAmplitudeModelArray()
{
    return amplitudeModelArray;
}

public void setOverlapSelectedStates(int[] overlapSelectedStates)
{
    this.overlapSelectedStates = overlapSelectedStates;
}

public void setOverlapLag(int overlapLag)
{
    this.overlapLag = overlapLag;
}

public void setOverlapModelType(int overlapModelType)
{
    this.overlapModelType = overlapModelType;
}

public void setOverlapModelVoxel(Voxel overlapModelVoxel)
{
    this.overlapModelVoxel = overlapModelVoxel;
}

public void setOverlapOrder(int overlapOrder)
{
    this.overlapOrder = overlapOrder;
}

public double[] getOverlapModelArray()
{
    return overlapModelArray;
}

public void enableSpectrumButton()
{
    this.spectrumButton.setEnabled(true);
}

public void enableCalculateButton()
{
    this.calculateButton.setEnabled(true);
}

/******* LISTEN BUTTON ACTION EVENT *******/

public void actionPerformed(ActionEvent event)
{
    JButton clickedButton = (JButton) event.getSource();

    if(clickedButton == loadDataButton)
    {
        loadDataButton.setEnabled(false);
        this.setVisible(false);

        pathGetter = new FilePathGetter(extensionsFilter);
        filePath = pathGetter.getPath();
        currentFilePath = removeExtension(filePath);

        try
```

```
{
    System.out.println("Inicio da leitura");
    startTime = System.currentTimeMillis();
    anlzFile = new AnlzReader(filePath);
    imageDims = anlzFile.getDimensions();
    imagingData = anlzFile.getRealImageData();
    complexData = anlzFile.dataIsComplex();
    if(complexData)
    {
        analysisData = anlzFile.getImaginaryImageData();
    }
    stopTime = System.currentTimeMillis();
    System.out.println("Fim da leitura");
    elapsedTime = stopTime - startTime;
    System.out.println("Tempo de leitura:");
    System.out.println(elapsedTime + " ms");
    System.out.println("");

    numberOfVoxels = (double)imageDims[1]
                             *imageDims[2]
                             *imageDims[3]
                             *imageDims[4];
    anlzFile.showHeader();

    backgroundClean = wordIsPresentIn(
                          anlzFile.getDescrip(), "bkgClean");
    outliersClean = wordIsPresentIn(
                          anlzFile.getDescrip(), "outClean");
    filtered = wordIsPresentIn(
                          anlzFile.getDescrip(), "filtered");

    if(complexData)
    {
        usedAnalysisData = analysisData;
    }
    else
    {
        usedAnalysisData = imagingData;
    }
    usedImagingData = imagingData;

    width = imageDims[1];
    height = imageDims[2];
    numberOfSlices = imageDims[3];
    numberOfInstants = imageDims[4];

    bckgVolume = extractInstantVolume(0, false);

    dataMask = getTotalMask();
    numberOfMaskVoxels = dataMask.getNumberOfMaskVoxels();

    loadMaskButton.setEnabled(true);
    simpleMaskButton.setEnabled(true);
    viewDataButton.setEnabled(true);
    viewDataMaskButton.setEnabled(true);
    timelineButton.setEnabled(true);
    setStatesButton.setEnabled(true);
}
catch(Exception e)
{
    if(e.getMessage().equals("not hdr"))
    {
        JOptionPane.showMessageDialog(this,
                                      "Nao escolheu um *.hdr",
                                      "Erro de formato",
                                      JOptionPane.ERROR_MESSAGE);
    }
    else
    {
        JOptionPane.showMessageDialog(this,
                                    "O header nao tem 348 bytes",
```

```
                                        "Erro de formato",
                                        JOptionPane.ERROR_MESSAGE);
            }
            System.out.println("Leitura cancelada");
            loadDataButton.setEnabled(true);
        }

        this.setVisible(true);
    }

    if(clickedButton == loadMaskButton)
    {
        this.setVisible(false);

        pathGetter = new FilePathGetter(extensionsFilter);
        filePath = pathGetter.getPath();
        currentFilePath = removeExtension(filePath);

        try
        {
            System.out.println("Inicio da leitura de mascara");
            startTime = System.currentTimeMillis();
            anlzFile = new AnlzReader(filePath);
            maskDims = anlzFile.getDimensions();
            if(maskDims[1] == width)
            {
                if(maskDims[2] == height)
                {
                    if(maskDims[3] == numberOfSlices)
                    {
                        dataMask =
                        new Mask(anlzFile.getBooleanRealImageData(0, 0.5));
                        numberOfMaskVoxels =
                        dataMask.getNumberOfMaskVoxels();

                        System.out.println("Leitura executada com sucesso");
                    }
                    else
                    {
                        JOptionPane.showMessageDialog(this,
                                                "dimensoes erradas",
                                                "Erro de ficheiro",
                                                JOptionPane.ERROR_MESSAGE);
                    }
                }
                else
                {
                    JOptionPane.showMessageDialog(this,
                                                "dimensoes erradas",
                                                "Erro de ficheiro",
                                                JOptionPane.ERROR_MESSAGE);
                }
            }
            else
            {
                JOptionPane.showMessageDialog(this,
                                                "dimensoes erradas",
                                                "Erro de ficheiro",
                                                JOptionPane.ERROR_MESSAGE);
            }

            stopTime = System.currentTimeMillis();
            System.out.println("Fim da leitura de mascara");
            elapsedTime = stopTime - startTime;
            System.out.println("Tempo de leitura de mascara:");
            System.out.println(elapsedTime + " ms");
            System.out.println("");

            new ViewerWindowDataMask(dataMask, 4, "Dados", this);
            anlzFile.showHeader();
        }
```

```
            catch(Exception e)
            {
                JOptionPane.showMessageDialog(this,
                                              "Mascara",
                                              "Erro de leitura",
                                              JOptionPane.ERROR_MESSAGE);
            }

            this.setVisible(true);
        }

        if(clickedButton == simpleMaskButton)
        {
            simpleMaskButton.setEnabled(false);
            this.setVisible(false);

            try
            {
                System.out.println("Inicio da mascara");
                startTime = System.currentTimeMillis();
                dataMask = new DataMaskBuilder(usedImagingData).getDataMask();
                numberOfMaskVoxels = dataMask.getNumberOfMaskVoxels();
                stopTime = System.currentTimeMillis();
                System.out.println("Fim da mascara");
                elapsedTime = stopTime - startTime;
                System.out.println("Tempo de mascara:");
                System.out.println(elapsedTime + " ms");
                System.out.println("");

                new ViewerWindowDataMask(dataMask, 4, "Dados", this);

                System.out.println("bckg/total= "+getBckgProb());
                System.out.println();
            }
            catch(Exception e)
            {

                JOptionPane.showMessageDialog(this,
                                              "Erro de mascara",
                                              "Erro de execucao",
                                              JOptionPane.ERROR_MESSAGE);

                loadDataButton.setEnabled(true);
            }

            this.setVisible(true);
        }

        if(clickedButton == setStatesButton)
        {
            setStatesButton.setEnabled(false);
            this.setVisible(false);

            new TimelineWindowStates(this);
        }

        if(clickedButton == setSequenceButton)
        {
            setSequenceButton.setEnabled(false);
            this.setVisible(false);

            new TimelineWindowSetter(statesAcronyms, numberOfInstants, this);
        }

        if(clickedButton == setAcquisitionParamsButton)
        {
            setAcquisitionParamsButton.setEnabled(false);
            this.setVisible(false);

            if(blockStudy)
            {
```

```
            new TimelineWindowBlockAcquisitionParams(this);
        }
        else
        {
            new TimelineWindowEventAcquisitionParams(this);
        }
    }

    /*
    if(clickedButton == cleanBackgroundButton)
    {
        cleanBackgroundButton.setEnabled(false);
        this.setVisible(false);

        cleanBackground();
        //new ParamWindowBackgroundCleaner(this);
    }
    */

    if(clickedButton == cleanOutliersButton)
    {
        cleanOutliersButton.setEnabled(false);
        this.setVisible(false);

        new ParamWindowOutliersCleaner(this);
    }
    if(clickedButton == cleanLowFreqsButton)
    {
        this.setVisible(false);
        cleanLowFreqsButton.setEnabled(false);

        new ParamWindowHighPassFilter(this);
    }
    if(clickedButton == clustersSegmentButton)
    {
        this.setVisible(false);
        clustersSegmentButton.setEnabled(false);

        withTumor = tumorSelector.isSelected();

        new ParamWindowClusterSampler(usedImagingData, 4, withTumor, this);
    }

    if(clickedButton == setMethodParamsButton)
    {
        this.setVisible(false);

        selectedMethod = (String) methodComboBox.getSelectedItem();
        if(selectedMethod == "t de Student")
        {
            new ParamWindowStudent(statesAcronyms, this);
        }
        if(selectedMethod == "Correlacao")
        {
            new ParamWindowCorr(statesAcronyms, this);
        }
        if(selectedMethod == "Fourier")
        {
            new ParamWindowFFT(statesAcronyms, this);
        }
        if(selectedMethod == "GLM")
        {
            new ParamWindowGLM(statesAcronyms, this);
        }
        if(selectedMethod == "Amplitude")
        {
            new ParamWindowAmplitude(statesAcronyms, this);
        }
        if(selectedMethod == "Sobreposicao")
        {
            new ParamWindowOverlap(statesAcronyms, this);
```

```
        }
    }

    if(clickedButton == viewDataButton)
    {
        this.setVisible(false);

        if(timelinePresent)
        {
            new ViewerWindowData(usedImagingData, 4, timeline, this);
        }
        else
        {
            new ViewerWindowData(usedImagingData, 4, this);
        }

        this.setVisible(true);
    }

    if(clickedButton == viewDesignButton)
    {
        new ViewerWindowDesignMatrix(designMatrix);
    }

    if(clickedButton == viewDataMaskButton)
    {
        new ViewerWindowDataMask(dataMask, 4, "Dados", this);
    }

    if(clickedButton == timelineButton)
    {
        this.setVisible(false);
        if(timelinePresent)
        {
            BlockImageBuilder bckgBuilder =
            new BlockImageBuilder(2, 250, numberOfStates, (float)0.25);
            bckgBuilder.setHorizBlocks(timeline);
            new ViewerWindowDataArrays(usedAnalysisData,
                                       bckgBuilder.getDataArray(),
                                       "Sequencia",
                                       this);
        }
        else
        {
            new ViewerWindowDataArrays(usedAnalysisData, "Sequencia", this);
        }
        this.setVisible(true);
    }
    if(clickedButton == spectrumButton)
    {
        this.setVisible(false);
        new ViewerWindowDataArrays(cleanDataSpectrum, "Espectro", this);
        this.setVisible(true);
    }
    if(clickedButton == calculateButton)
    {
        if(selectedMethod == "t de Student")
        {
            this.setVisible(false);

            System.out.println("Inicio da Analise de Student");
            startTime = System.currentTimeMillis();
            tAnalyzer = new TStudentAnalysis(usedAnalysisData,
                                             dataMask,
                                             designMatrix,
                                             tStudentSelectedStates,
                                             tSudentLag);
            stopTime = System.currentTimeMillis();
            System.out.println("Fim da Analise de Student");

            tTestResultData = tAnalyzer.getResultData();
```

```
                new ViewerWindowProfiles(tTestResultData, "t de Student", this);

                elapsedTime = stopTime - startTime;
                System.out.println("Tempo de Analise de Student:");
                System.out.println(elapsedTime + " ms");
                System.out.println("");

                new ViewerWindowResult(tTestResultData,
                                       bckgVolume,
                                       numberOfMaskVoxels,
                                       4,
                                       "tStudent",
                                       this);

                this.setVisible(true);
            }

            if(selectedMethod == "Correlacao")
            {
                this.setVisible(false);

                if(corrModelType == 0)
                {
                    corrDesignVector = designMatrix.getDesignVector(
                                                corrSelectedStates[0],
                                                corrSelectedStates[1],
                                                corrLag);
                    corrModelArray = corrDesignVector;
                }

                if(corrModelType == 1)
                {
                    corrDesignVector = designMatrix.getDesignVector(
                                                corrSelectedStates[0],
                                                corrSelectedStates[1],
                                                corrLag);
                    hrf = new Hrf(acquisitionTime).getHRF();
                    corrModelArray = new Convolver(
                                            corrDesignVector, hrf).getConv();
                }

                if(corrModelType == 2)
                {
                    double[] timeArray = extractTimeArray(
                                                corrModelVoxel.getX(),
                                                corrModelVoxel.getY(),
                                                corrModelVoxel.getZ(),
                                                usedAnalysisData);
                    Timeline voxelTimeline = new Timeline(timeArray);
                    double[] selectVector = designMatrix.getSelectVector(
                                                corrSelectedStates[0],
                                                corrSelectedStates[1]);
                    corrModelArray = voxelTimeline.getLaggedData(selectVector,
                                                                 corrLag);
                }

                System.out.println("Inicio da Analise de Correlacao");
                startTime = System.currentTimeMillis();
                corrAnalyzer = new CorrAnalysis(usedAnalysisData,
                                                dataMask,
                                                designMatrix,
                                                corrSelectedStates,
                                                corrLag,
                                                corrModelArray);
                stopTime = System.currentTimeMillis();
                System.out.println("Fim da Analise de Correlacao");

                corrTestResultData = corrAnalyzer.getResultData();
                new ViewerWindowProfiles(corrTestResultData,
                                            "Correlacao",
                                            this);
```

```
            elapsedTime = stopTime - startTime;
            System.out.println("Tempo de Analise de Correlacao:");
            System.out.println(elapsedTime + " ms");
            System.out.println("");

            new ViewerWindowResult(corrTestResultData,
                                   bckgVolume,
                                   numberOfMaskVoxels,
                                   4,
                                   "Correlacao",
                                   this);

            this.setVisible(true);
        }

        if(selectedMethod == "Fourier")
        {
            this.setVisible(false);

            if(fFourierModelType == 0)
            {
                fFourierDesignVector = designMatrix.getDesignVector(
                                                fFourierSelectedStates[0],
                                                fFourierSelectedStates[1],
                                                fFourierLag);
                fFourierModelArray = fFourierDesignVector;
            }

            if(fFourierModelType == 1)
            {
                fFourierDesignVector = designMatrix.getDesignVector(
                                                fFourierSelectedStates[0],
                                                fFourierSelectedStates[1],
                                                fFourierLag);
                hrf = new Hrf(acquisitionTime).getHRF();
                fFourierModelArray =
                    new Convolver(fFourierDesignVector, hrf).getConv();
            }

            if(fFourierModelType == 2)
            {
                double[] timeArray =
                    extractTimeArray(fFourierModelVoxel.getX(),
                                     fFourierModelVoxel.getY(),
                                     fFourierModelVoxel.getZ(),
                                     usedAnalysisData);

                Timeline voxelTimeline = new Timeline(timeArray);
                double[] selectVector = designMatrix.getSelectVector(
                                        fFourierSelectedStates[0],
                                        fFourierSelectedStates[1]);
                fFourierModelArray = voxelTimeline.getLaggedData(
                                                    selectVector,
                                                    fFourierLag);
            }

            calculateFFT();
            cleanDataSpectrum = fFourierAnalyzer.getDataFFTNorm();

            if(fFourierInfMethod == 0)
            {
                System.out.println(
                        "Inicio da Analise de Picos de Fourier");
                startTime = System.currentTimeMillis();
                fFourierTestResultData =
                    fFourierAnalyzer.getPeaksSlopesAnalysisResult();
                stopTime = System.currentTimeMillis();
                System.out.println(
                    "Fim da Analise de Picos de Fourier");
```

```
                new ViewerWindowProfiles(fFourierTestResultData,
                                         "Picos de Fourier",
                                         this);

                elapsedTime = stopTime - startTime;
                System.out.println(
                            "Tempo da Analise de Picos de Fourier:");
                System.out.println(elapsedTime + " ms");
                System.out.println("");
            }
            if(fFourierInfMethod == 1)
            {
                System.out.println(
                            "Inicio da Analise de Comparacao de Fourier");
                startTime = System.currentTimeMillis();
                fFourierTestResultData =
                    fFourierAnalyzer.getDataModelComparisonAnalysisResult();
                stopTime = System.currentTimeMillis();
                System.out.println(
                            "Fim da Analise de Comparacao de Fourier");

                new ViewerWindowProfiles(fFourierTestResultData,
                                         "Comparacao de Fourier",
                                         this);

                elapsedTime = stopTime - startTime;
                System.out.println(
                            "Tempo da Analise de Comparacao de Fourier:");
                System.out.println(elapsedTime + " ms");
                System.out.println("");
            }

            new ViewerWindowResult(fFourierTestResultData,
                                   bckgVolume,
                                   numberOfMaskVoxels,
                                   4,
                                   "Fourier",
                                   this);

            this.setVisible(true);
        }

        if(selectedMethod == "GLM")
        {
            this.setVisible(false);

            if(glmModelType == 0)
            {
                glmDesignVector = designMatrix.getDesignVector(
                                                glmSelectedStates[0],
                                                glmSelectedStates[1],
                                                glmLag);
                glmModelArray = glmDesignVector;
            }

            if(glmModelType == 1)
            {
                glmDesignVector = designMatrix.getDesignVector(
                                                glmSelectedStates[0],
                                                glmSelectedStates[1],
                                                glmLag);
                hrf = new Hrf(acquisitionTime).getHRF();
                glmModelArray =
                        new Convolver(glmDesignVector, hrf).getConv();
            }

            if(glmModelType == 2)
            {
                double[] timeArray = extractTimeArray(
                                                glmModelVoxel.getX(),
                                                glmModelVoxel.getY(),
```

```
                                        glmModelVoxel.getZ(),
                                        usedAnalysisData);
            Timeline voxelTimeline = new Timeline(timeArray);
            double[] selectVector = designMatrix.getSelectVector(
                                        glmSelectedStates[0],
                                        glmSelectedStates[1]);
            glmModelArray = voxelTimeline.getLaggedData(selectVector,
                                                        glmLag);
        }

        System.out.println("Inicio da Analise de GLM");
        startTime = System.currentTimeMillis();
        glmAnalyzer = new GlmAnalysis(usedAnalysisData,
                                      dataMask,
                                      designMatrix,
                                      glmSelectedStates,
                                      glmLag,
                                      glmInferenceType);
        stopTime = System.currentTimeMillis();
        System.out.println("Fim da Analise de GLM");

        glmTestResultData = glmAnalyzer.getGLMAnalysisResult();
        new ViewerWindowProfiles(glmTestResultData, "GLM", this);

        elapsedTime = stopTime - startTime;
        System.out.println("Tempo de Analise de GLM:");
        System.out.println(elapsedTime + " ms");
        System.out.println("");

        new ViewerWindowResult(glmTestResultData,
                               bckgVolume,
                               numberOfMaskVoxels,
                               4,
                               "GLM",
                               this);

        this.setVisible(true);
    }

    if(selectedMethod == "Amplitude")
    {
        this.setVisible(false);

        if(amplitudeModelType == 0)
        {
            amplitudeDesignVector = designMatrix.getDesignVector(
                                        amplitudeSelectedStates[0],
                                        amplitudeSelectedStates[1],
                                        amplitudeLag);
            amplitudeModelArray = amplitudeDesignVector;
        }

        if(amplitudeModelType == 1)
        {
            amplitudeDesignVector = designMatrix.getDesignVector(
                                        amplitudeSelectedStates[0],
                                        amplitudeSelectedStates[1],
                                        amplitudeLag);
            hrf = new Hrf(acquisitionTime).getHRF();
            amplitudeModelArray =
                    new Convolver(amplitudeDesignVector, hrf).getConv();
        }

        if(amplitudeModelType == 2)
        {
            double[] timeArray = extractTimeArray(
                                        amplitudeModelVoxel.getX(),
                                        amplitudeModelVoxel.getY(),
                                        amplitudeModelVoxel.getZ(),
                                        usedAnalysisData);
            Timeline voxelTimeline = new Timeline(timeArray);
```

```
                double[] selectVector = designMatrix.getSelectVector(
                                                amplitudeSelectedStates[0],
                                                amplitudeSelectedStates[1]);
                amplitudeModelArray = voxelTimeline.getLaggedData(
                                                selectVector,
                                                amplitudeLag);
            }

            System.out.println("Inicio da Analise de Amplitude");
            startTime = System.currentTimeMillis();
            amplitudeAnalyzer = new AmplitudeAnalysis(usedAnalysisData,
                                                dataMask,
                                                designMatrix,
                                                amplitudeSelectedStates,
                                                amplitudeLag,
                                                amplitudeModelArray);
            stopTime = System.currentTimeMillis();
            System.out.println("Fim da Analise de Amplitude");

            amplitudeTestResultData =
                amplitudeAnalyzer.getAmplitudeAnalysisResult();
            new ViewerWindowProfiles(amplitudeTestResultData,
                                     "Amplitude",
                                     this);

            elapsedTime = stopTime - startTime;
            System.out.println("Tempo de Analise de Amplitude:");
            System.out.println(elapsedTime + " ms");
            System.out.println("");

            new ViewerWindowResult(amplitudeTestResultData,
                                   bckgVolume,
                                   numberOfMaskVoxels,
                                   4,
                                   "Amplitude",
                                   this);

            this.setVisible(true);
        }

        if(selectedMethod == "Sobreposicao")
        {
            this.setVisible(false);

            if(overlapModelType == 0)
            {
                overlapDesignVector = designMatrix.getDesignVector(
                                                overlapSelectedStates[0],
                                                overlapSelectedStates[1],
                                                overlapLag);
                overlapModelArray = overlapDesignVector;
            }

            if(overlapModelType == 1)
            {
                overlapDesignVector = designMatrix.getDesignVector(
                                                overlapSelectedStates[0],
                                                overlapSelectedStates[1],
                                                overlapLag);
                hrf = new Hrf(acquisitionTime).getHRF();
                overlapModelArray =
                    new Convolver(overlapDesignVector, hrf).getConv();
            }

            if(overlapModelType == 2)
            {
                double[] timeArray = extractTimeArray(
                                        overlapModelVoxel.getX(),
                                        overlapModelVoxel.getY(),
                                        overlapModelVoxel.getZ(),
                                        usedAnalysisData);
```

```
                Timeline voxelTimeline = new Timeline(timeArray);
                double[] selectVector = designMatrix.getSelectVector(
                                            overlapSelectedStates[0],
                                            overlapSelectedStates[1]);
                overlapModelArray = voxelTimeline.getLaggedData(
                                            selectVector,
                                            overlapLag);
            }

            System.out.println("Inicio da Analise de Sobreposicao");
            startTime = System.currentTimeMillis();
            overlapAnalyzer = new OverlapAnalysis(usedAnalysisData,
                                                  dataMask,
                                                  designMatrix,
                                                  overlapSelectedStates,
                                                  overlapLag,
                                                  overlapModelArray,
                                                  10,
                                                  overlapOrder);
            stopTime = System.currentTimeMillis();
            System.out.println("Fim da Analise de Sobreposicao");

            overlapTestResultData =
                overlapAnalyzer.getOverlapAnalysisResult();
            new ViewerWindowProfiles(overlapTestResultData,
                                     "Sobreposicao",
                                     this);

            elapsedTime = stopTime - startTime;
            System.out.println("Tempo de Analise de Sobreposicao:");
            System.out.println(elapsedTime + " ms");
            System.out.println("");

            new ViewerWindowResult(overlapTestResultData,
                                   bckgVolume,
                                   numberOfMaskVoxels,
                                   4,
                                   "Sobreposicao",
                                   this);

            this.setVisible(true);
        }
    }
}

private void calculateFFT()
{
    System.out.println("Inicio do Calculo da FFT");
    startTime = System.currentTimeMillis();
    fFourierAnalyzer = new FFourierAnalysis(usedAnalysisData,
                                            dataMask,
                                            designMatrix,
                                            fFourierSelectedStates,
                                            fFourierLag,
                                            fFourierDataWindow,
                                            fFourierModelArray);
    stopTime = System.currentTimeMillis();
    System.out.println("Fim do Calculo da FFT");

    elapsedTime = stopTime - startTime;
    System.out.println("Tempo do Calculo da FFT:");
    System.out.println(elapsedTime + " ms");
    System.out.println("");

    cleanDataSpectrum = fFourierAnalyzer.getDataFFTNorm();
    fftNotPerformed = false;
    enableSpectrumButton();
}

private void createGUI()
{
```

```
mainContainer = getContentPane();
mainContainer.setLayout(new GridLayout(1, 1));
mainContainer.setBackground(Color.white);

mainTabbedPane = new JTabbedPane();

loadDataPanel = new JPanel();
loadDataButton = new JButton("Anlz 4D");
loadDataPanel.add(loadDataButton);
loadDataButton.setEnabled(true);
loadDataButton.addActionListener(this);

getDataMaskPanel = new JPanel();
loadMaskButton = new JButton("Carregar");
simpleMaskButton = new JButton("Simples");
getDataMaskPanel.add(loadMaskButton);
getDataMaskPanel.add(simpleMaskButton);
loadMaskButton.setEnabled(false);
loadMaskButton.addActionListener(this);
simpleMaskButton.setEnabled(false);
simpleMaskButton.addActionListener(this);

cleanDataPanel = new JPanel();
//cleanBackgroundButton = new JButton("Fundo");
cleanOutliersButton = new JButton("Estranhos");
cleanLowFreqsButton = new JButton("Filtrar");
//cleanDataPanel.add(cleanBackgroundButton);
cleanDataPanel.add(cleanOutliersButton);
cleanDataPanel.add(cleanLowFreqsButton);
//cleanBackgroundButton.setEnabled(false);
//cleanBackgroundButton.addActionListener(this);
cleanOutliersButton.setEnabled(false);
cleanOutliersButton.addActionListener(this);
cleanLowFreqsButton.setEnabled(false);
cleanLowFreqsButton.addActionListener(this);

viewDataPanel = new JPanel();
viewDataButton = new JButton("Imagens");
viewDesignButton = new JButton("Design");
viewDataMaskButton = new JButton("Mascara");
timelineButton = new JButton("Sequencia");
spectrumButton = new JButton("Espectro");
viewDataPanel.add(viewDataButton);
viewDataPanel.add(viewDesignButton);
viewDataPanel.add(viewDataMaskButton);
viewDataPanel.add(timelineButton);
viewDataPanel.add(spectrumButton);
viewDataButton.setEnabled(false);
viewDataButton.addActionListener(this);
viewDesignButton.setEnabled(false);
viewDesignButton.addActionListener(this);
viewDataMaskButton.setEnabled(false);
viewDataMaskButton.addActionListener(this);
timelineButton.setEnabled(false);
timelineButton.addActionListener(this);
spectrumButton.setEnabled(false);
spectrumButton.addActionListener(this);

setDesignPanel = new JPanel();
setStatesButton = new JButton("Estados");
setSequenceButton = new JButton("Sequencia");
setAcquisitionParamsButton = new JButton("Parametros");
setDesignPanel.add(setStatesButton);
setDesignPanel.add(setSequenceButton);
setDesignPanel.add(setAcquisitionParamsButton);
setStatesButton.setEnabled(false);
setStatesButton.addActionListener(this);
setSequenceButton.setEnabled(false);
setSequenceButton.addActionListener(this);
setAcquisitionParamsButton.setEnabled(false);
setAcquisitionParamsButton.addActionListener(this);
```

```
        segmentDataPanel = new JPanel();
        tumorSelector = new JCheckBox("Tumor");
        tumorSelector.setEnabled(false);
        clustersSegmentButton = new JButton("Clusters");
        clustersSegmentButton.setEnabled(false);
        clustersSegmentButton.addActionListener(this);
        segmentDataPanel.add(tumorSelector);
        segmentDataPanel.add(clustersSegmentButton);

        setMethodPanel = new JPanel();
        methodComboBox = new JComboBox(comboBoxOptions);
        setMethodParamsButton = new JButton("Parametros");
        calculateButton = new JButton("Calcular");
        setMethodPanel.add(methodComboBox);
        setMethodPanel.add(setMethodParamsButton);
        setMethodPanel.add(calculateButton);

        methodComboBox.setEnabled(false);
        setMethodParamsButton.setEnabled(false);
        setMethodParamsButton.addActionListener(this);

        calculateButton.setEnabled(false);
        calculateButton.addActionListener(this);

        mainTabbedPane.addTab("Dados", loadDataPanel);
        mainTabbedPane.addTab("Mascara", getDataMaskPanel);
        mainTabbedPane.addTab("Design", setDesignPanel);
        mainTabbedPane.addTab("Limpar", cleanDataPanel);
        mainTabbedPane.addTab("Segmentar", segmentDataPanel);
        mainTabbedPane.addTab("Ver", viewDataPanel);
        mainTabbedPane.addTab("Processar", setMethodPanel);

        mainContainer.add(mainTabbedPane);

        setSize(mainWindowWidth, mainWindowHeight);
        setLocation(mainWindowXPosition, mainWindowYPosition);
        setTitle("Cerebro");
        setVisible(true);
    }

    private String[] extractColumnFromDualArray(String[][] dualArray,
                                                int columnNumber)
    {
        int numberOfLines = dualArray.length;
        int numberOfColumns = dualArray[0].length;
        int i;

        String[] column = new String[numberOfLines];

        if(columnNumber < numberOfColumns)
        {
            for(i=0; i<numberOfLines; i++)
            {
                column[i] = dualArray[i][columnNumber];
            }
        }

        return column;
    }



    /******* CHANGE PATH TO IMG *******/

    private String concatenateStrings(String textOne, String textTwo)
    {
        return textOne+textTwo;
    }

    private String removeExtension(String filePath)
```

```
{
    int dotIndex;
    String pathBase;
    String newFilePath;

    dotIndex = filePath.lastIndexOf(".");

    if(dotIndex==filePath.length()-4)
    {
        pathBase = filePath.substring(0,dotIndex);
        newFilePath = pathBase;
    }
    else
    {
        newFilePath = filePath;
    }

    return newFilePath;
}

private boolean wordIsPresentIn(String text, String word)
{
    int indexOfWord;

    indexOfWord = text.indexOf(word);
    if(indexOfWord<0)
    {
        return false;
    }
    else
    {
        return true;
    }
}



/******* EXTRACT TIME ARRAY ******/

private double[] extractTimeArray(int x,
                                  int y,
                                  int z,
                                  double[][][][] dataMatrix)
{
    int t;
    double[] timeArray = new double[numberOfInstants];

    for(t=0; t< numberOfInstants; t++)
    {
        timeArray[t] = dataMatrix[x][y][z][t];
    }
    return timeArray;
}

private double[][][] extractInstantVolume(int instant, boolean forAnalysis)
{
    int x, y, z;
    double[][][] instantVolume = new double[width][height][numberOfSlices];

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if(forAnalysis)
                {
                    instantVolume[x][y][z] =
                            usedAnalysisData[x][y][z][instant];
                }
                else
```

```
                    {
                        instantVolume[x][y][z] =
                                usedImagingData[x][y][z][instant];
                    }
                }
            }
        }

        return instantVolume;
    }

    private Mask getTotalMask()
    {
        int x, y, z;
        boolean[][][] maskArray =
                        new boolean[width][height][numberOfSlices];

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    maskArray[x][y][z] = true;
                }
            }
        }

        return new Mask(maskArray);
    }

    /******* SHOW ARRAY ******

    private void showArray(double[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */
}
```

```
package functional.methods;

//
//  AmplitudeAnalysis.java
//
//
//  Created by JGLG on 3/22/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import estatistica.descriptive.OneDArrayAnalyzer;
import estatistica.distributions.GaussianDistribution;
import functional.data.DesignMatrix;
import functional.data.Mask;
import functional.data.Timeline;

public class AmplitudeAnalysis
{
    //
    //  Data Members
    //

    private double[][][] amplitudes;
    private double[][][] amplitudesVariance;
    private double[][][] amplitudeAnalysisResult;

    private double[][][][] imageData;
    private Mask dataMask;
    private DesignMatrix designMatrix;
    private int[] statesID;
    private int lag;
    private double[] modelArray;

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfInstants;

    //
    //  Constructors
    //
    //      amplitudeAnalysis(double[][][][] imageData,
    //                        Mask dataMask,
    //                        DesignMatrix designMatrix,
    //                        int[] statesID,
    //                        int lag,
    //                        double[] modelArray)
    //

    public AmplitudeAnalysis(double[][][][] imageData,
                             Mask dataMask,
                             DesignMatrix designMatrix,
                             int[] statesID,
                             int lag,
                             double[] modelArray)
    {
        this.imageData = imageData;
        this.dataMask = dataMask;
        this.designMatrix = designMatrix;
        this.statesID = statesID;
        this.lag = lag;
        this.modelArray = modelArray;

        width = imageData.length;
        height = imageData[0].length;
        numberOfSlices = imageData[0][0].length;
        numberOfInstants = imageData[0][0][0].length;

        performAmplitudeAnalysis();
    }
```

```
//
//  Public Methods
//
//      double[][][] getResultData()
//

public double[][][] getAmplitudes()
{
    return amplitudes;
}

public double[][][] getAmplitudesVariance()
{
    return amplitudesVariance;
}

public double[][][] getAmplitudeAnalysisResult()
{
    return amplitudeAnalysisResult;
}

//
//  Private Methods
//
//      double[] addValueToEndOfArray(double[] array, double value)
//      double[] extractTimeArray(int x,
//                                int y,
//                                int z,
//                                double dataMatrix[][][][])
//      double[] getFirstDRefPattern(int numberOfCycles,
//                                   int initialRest,
//                                   int onsPerBlock,
//                                   int offsPerBlock)
//

private void performAmplitudeAnalysis()
{
    getDataAmplitudes();

    getAmplitudeProbs();
}


private void getDataAmplitudes()
{
    amplitudes = new double[width][height][numberOfSlices];
    amplitudesVariance = new double[width][height][numberOfSlices];

    Timeline voxelTimeline;

    double[] selectVector;
    double[] firstStateVector;
    double[] secndStateVector;

    int firstStateIndex;
    int secndStateIndex;

    firstStateIndex = statesID[0];
    secndStateIndex = statesID[1];

    double[] timelineData;

    double[] dataArray;

    double[] difference = new double[numberOfInstants-lag];

    OneDArrayAnalyzer posAnalyzer;
    OneDArrayAnalyzer negAnalyzer;

    double[] positives;
    double[] negatives;
```

```
    double posMean;
    double negMean;

    double[] scaledModel;

    int x, y, z;

    int i;

    selectVector =
        designMatrix.getSelectVector(firstStateIndex, secndStateIndex);
    firstStateVector = designMatrix.getColumn(firstStateIndex);
    secndStateVector = designMatrix.getColumn(secndStateIndex);

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if(dataMask.getBooleanValue(x, y, z))
                {
                    timelineData = extractTimeArray(x, y, z, imageData);

                    voxelTimeline = new Timeline(timelineData);

                    dataArray =
                        voxelTimeline.getLaggedData(selectVector, lag);

                    positives =
                        voxelTimeline.getLaggedData(firstStateVector, lag);
                    negatives =
                        voxelTimeline.getLaggedData(secndStateVector, lag);

                    posAnalyzer = new OneDArrayAnalyzer(positives);
                    posMean = posAnalyzer.getMean();

                    negAnalyzer = new OneDArrayAnalyzer(negatives);
                    negMean = negAnalyzer.getMean();

                    scaledModel = getScaledModel(posMean, negMean);

                    for(i=0; i<numberOfInstants-lag; i++)
                    {
                        difference[i] = dataArray[i] - scaledModel[i];
                    }

                    amplitudesVariance[x][y][z] =
                        new OneDArrayAnalyzer(difference).getVariance();
                    amplitudes[x][y][z] = (posMean - negMean)
                            /2*Math.sqrt(amplitudesVariance[x][y][z]);
                }
                else
                {
                    amplitudesVariance[x][y][z] = 0.0;
                    amplitudes[x][y][z] = 0.0;
                }
            }
        }
    }
}


private void getAmplitudeProbs()
{
    int x, y, z;

    double[] serialData;
    int counter = 0;
```

```
        double mean;
        double stdDev;

        OneDArrayAnalyzer dataAnalyzer;

        GaussianDistribution amplitudeGaussian;
        double pValue;

        amplitudeAnalysisResult = new double[width][height][numberOfSlices];


        serialData = new double[dataMask.getNumberOfMaskVoxels()];

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    if(dataMask.getBooleanValue(x, y, z))
                    {
                        serialData[counter] = amplitudes[x][y][z];
                        counter = counter + 1;
                    }
                }
            }
        }

        dataAnalyzer = new OneDArrayAnalyzer(serialData);
        mean = dataAnalyzer.getMean();
        stdDev = dataAnalyzer.getStdDev();

        //System.out.println("mean = " + mean);
        //System.out.println("stdDev = " + stdDev);

        //new WriteTextNumberFile(serialData,
        //                        "/Users/gabix/Desktop/hist.txt", 0);

        amplitudeGaussian = new GaussianDistribution(mean, stdDev);

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    if(dataMask.getBooleanValue(x, y, z))
                    {
                        pValue =
                            amplitudeGaussian.getCumProb(amplitudes[x][y][z]);
                    }
                    else
                    {
                        pValue = 0.0;
                    }
                    amplitudeAnalysisResult[x][y][z] = pValue;
                }
            }
        }

        resultMaxValue(amplitudes);
        resultMinValue(amplitudes);
    }

    private double[] getScaledModel(double posMean, double negMean)
    {
        int i;
        double[] scaledModel = new double[numberOfInstants-lag];

        for(i=0; i<numberOfInstants-lag; i++)
        {
```

```
        if(modelArray[i] > 0)
        {
            scaledModel[i] = posMean*modelArray[i];
        }
        if(modelArray[i] < 0)
        {
            scaledModel[i] = negMean*modelArray[i];
        }
    }

    return scaledModel;
}

/******* EXTRACT TIME ARRAY ******/

private double[] extractTimeArray(int x,
                                  int y,
                                  int z,
                                  double dataMatrix[][][][])
{
    int t;
    double[] timeArray = new double[numberOfInstants];

    for(t=0; t< numberOfInstants; t++)
    {
        timeArray[t] = dataMatrix[x][y][z][t];
    }
    return timeArray;
}



/******* RESULT MAXIMUM VALUE *******/

private void resultMaxValue(double resultMatrix[][][])
{
    int x, y, z;
    int xx=0;
    int yy=0;
    int zz=0;
    double elementValue;
    double maxValue=resultMatrix[0][0][0];

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                elementValue = resultMatrix[x][y][z];
                if(elementValue > maxValue)
                {
                    maxValue = elementValue;
                    xx = x;
                    yy = y;
                    zz = z;
                }
            }
        }
    }

    System.out.println("Max( p(x,y,z) ) = p("
                       + xx + "," + yy + "," + zz + ") = "
                       + amplitudeAnalysisResult[xx][yy][zz]);

}



/******* RESULT MINIMUM VALUE *******/
```

```
    private void resultMinValue(double resultMatrix[][][])
    {
        int x, y, z;
        int xx=0;
        int yy=0;
        int zz=0;
        double elementValue;
        double minValue=resultMatrix[0][0][0];

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    elementValue = resultMatrix[x][y][z];
                    if(elementValue < minValue)
                    {
                        minValue = elementValue;
                        xx = x;
                        yy = y;
                        zz = z;
                    }
                }
            }
        }

        System.out.println("Min( p(x,y,z) ) = p("
                           + xx + "," + yy + "," + zz + ") = "
                           + amplitudeAnalysisResult[xx][yy][zz]);

    }
}
```

```java
package functional.data;

//
//  AnlzHeader.java
//
//
//  Created by Jose Gabriel Lira Gomes on 11/22/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import fileUtils.BytesArray;

public class AnlzHeader
{
    private int     defHeaderSize = 348;
    private String  defDataType = "bin";
    private String  defDataBaseName = "";
    private int     defExtents = 0;
    private short   defSessionError = 0;
    private char    defRegular = 114; //r
    private char    defHKey = 48; //0
    private short   defNumberOfDims = 4;
    private short[] defDimensions = {defNumberOfDims, 1, 1, 1, 1, 0, 0, 0};
    private String  defVoxelUnits = "mm";
    private String  defCalUnits = "";
    private short   defUnused = 0;
    private short   defDataTypeCode = 1;
    private short   defBitsPerPixel = 1;
    private short   defDimUn = 0;
    private float[] defPixelsDims = {0, 1, 1, 1, 0, 0, 0, 0};
    private float   defVoxelOffset = 0;
    private float   defFUnusedOne = 1;
    private float   defFUnusedTwo = 0;
    private float   defFUnusedThree = 0;
    private float   defCalMax = 0;
    private float   defCalMin = 0;
    private float   defCompressed = 0;
    private float   defVerified = 0;
    private int     defGlMax = 1;
    private int     defGlMin = 0;
    private String  defDescrip = "cerebro - ";
    private String  defAuxFile = "none";
    private char    defOrient = 0;
    private String  defOriginator = "";
    private String  defGenerated = "";
    private String  defScannum = "";
    private String  defPatientID = "";
    private String  defExpDate = "";
    private String  defExpTime = "";
    private String  defHistUn = "";
    private int     defViews = 0;
    private int     defVolumesAdded = 0;
    private int     defStartField = 0;
    private int     defFieldSkip = 0;
    private int     defOMax = 0;
    private int     defOMin = 0;
    private int     defSMax = 0;
    private int     defSMin = 0;

    private int     headerSize      = defHeaderSize;
    private String  dataType        = defDataType;      //<-------
    private String  dataBaseName    = defDataBaseName;
    private int     extents         = defExtents;
    private short   sessionError    = defSessionError;
    private char    regular         = defRegular;
    private char    hKey            = defHKey;
    private int     numberOfDims    = defNumberOfDims;  //<-------
    private short[] dimensions      = defDimensions;    //<-------
    private String  voxelUnits      = defVoxelUnits;
    private String  calUnits        = defCalUnits;
    private short   unused          = defUnused;
```

```
private short   dataTypeCode    = defDataTypeCode;  //<-------
private short   dimUn            = defDimUn;
private short   bitsPerPixel    = defBitsPerPixel;  //<-------
private float[] pixelsDims      = defPixelsDims;    //<-------
private float   voxelOffset     = defVoxelOffset;
private float   fUnusedOne      = defFUnusedOne;    //<-------
private float   fUnusedTwo      = defFUnusedTwo;
private float   fUnusedThree    = defFUnusedThree;
private float   calMax          = defCalMax;
private float   calMin          = defCalMin;
private float   compressed      = defCompressed;
private float   verified        = defVerified;
private int     glMax           = defGlMax;         //<-------
private int     glMin           = defGlMin;         //<-------
private String  descrip         = defDescrip;       //<-------
private String  auxFile         = defAuxFile;
private char    orient          = defOrient;
private String  originator      = defOriginator;
private String  generated       = defGenerated;
private String  scannum         = defScannum;
private String  patientID       = defPatientID;
private String  expDate         = defExpDate;
private String  expTime         = defExpTime;
private String  histUn          = defHistUn;
private int     views           = defViews;
private int     volumesAdded    = defVolumesAdded;
private int     startField      = defStartField;
private int     fieldSkip       = defFieldSkip;
private int     oMax            = defOMax;
private int     oMin            = defOMin;
private int     sMax            = defSMax;
private int     sMin            = defSMin;

public AnlzHeader()
{

}

public void setHeaderSize(int headerSize)
{
    this.headerSize = headerSize;
}

public void setDataType(String dataType)
{
    this.dataType = dataType;
}

public void setDataBaseName(String dataBaseName)
{
    this.dataBaseName = dataBaseName;
}

public void setExtents(int extents)
{
    this.extents = extents;
}

public void setSessionError(short sessionError)
{
    this.sessionError = sessionError;
}

public void setRegular(char regular)
{
    this.regular = regular;
}

public void setHKey(char hKey)
{
    this.hKey = hKey;
```

```
}

public void setDimensions(short[] dimensions)
{
    this.dimensions = dimensions;
}

public void setNumberOfDims(short numberOfDims)
{
    this.numberOfDims = numberOfDims;
    this.dimensions[0] = numberOfDims;
}

public void setImgWidth(short width)
{
    this.dimensions[1] = width;
}

public void setImgHeight(short height)
{
    this.dimensions[2] = height;
}

public void setImgNumberOfSlices(short numberOfSlices)
{
    this.dimensions[3] = numberOfSlices;
}

public void setImgNumberOfInstants(short numberOfInstants)
{
    this.dimensions[4] = numberOfInstants;
}

public void setVoxelUnits(String voxelUnits)
{
    this.voxelUnits = voxelUnits;
}

public void setCalUnits(String calUnits)
{
    this.calUnits = calUnits;
}

public void setUnused(short unused)
{
    this.unused = unused;
}

public void setDataTypeCode(short dataTypeCode)
{
    this.dataTypeCode = dataTypeCode;
}

public void setBitsPerPixel(short bitsPerPixel)
{
    this.bitsPerPixel = bitsPerPixel;
}

public void setPixelsDims(float[] pixelsDims)
{
    this.pixelsDims = pixelsDims;
}

public void setVoxelWidth(float width)
{
    this.pixelsDims[1] = width;
}

public void setVoxelHeight(float height)
{
    this.pixelsDims[2] = height;
```

```
}

public void setSliceThickness(float sliceThickness)
{
    this.pixelsDims[3] = sliceThickness;
}

public void setVoxelOffset(float voxelOffset)
{
    this.voxelOffset = voxelOffset;
}

public void setFUnusedOne(float fUnusedOne)
{
    this.fUnusedOne = fUnusedOne;
}

public void setFUnusedTwo(float fUnusedTwo)
{
    this.fUnusedTwo = fUnusedTwo;
}

public void setFUnusedThree(float fUnusedThree)
{
    this.fUnusedThree = fUnusedThree;
}

public void setCalMax(float calMax)
{
    this.calMax = calMax;
}

public void setCalMin(float calMin)
{
    this.calMin = calMin;
}

public void setCompressed(float compressed)
{
    this.compressed = compressed;
}

public void setVerified(float verified)
{
    this.verified = verified;
}

public void setGLMax(int glMax)
{
    this.glMax = glMax;
}

public void setGLMin(int glMin)
{
    this.glMin = glMin;
}

public void setDescrip(String descrip)
{
    this.descrip = descrip;
}

public void addDescrip(String text)
{
    this.descrip = this.descrip + text;
}

public void setAuxFile(String auxFile)
{
    this.auxFile = auxFile;
}
```

```
public void setOrient(char orient)
{
    this.orient = orient;
}

public void setOriginator(String originator)
{
    this.originator = originator;
}

public void setGenerated(String generated)
{
    this.generated = generated;
}

public void setScannum(String scannum)
{
    this.scannum = scannum;
}

public void setPatientID(String patientID)
{
    this.patientID = patientID;
}

public void setExpDate(String expDate)
{
    this.expDate = expDate;
}

public void setExpTime(String expTime)
{
    this.expTime = expTime;
}

public void setHistUn(String histUn)
{
    this.histUn = histUn;
}

public void setViews(int views)
{
    this.views = views;
}

public void setVolumesAdded(int volumesAdded)
{
    this.volumesAdded = volumesAdded;
}

public void setStartField(int startField)
{
    this.startField = startField;
}

public void setFieldSkip(int fieldSkip)
{
    this.fieldSkip = fieldSkip;
}

public void setOMax(int oMax)
{
    this.oMax = oMax;
}

public void setOMin(int oMin)
{
    this.oMin = oMin;
}
```

```
public void setSMax(int sMax)
{
    this.sMax = sMax;
}

public void setSMin(int sMin)
{
    this.sMin = sMin;
}

public int getHeaderSize()
{
    return headerSize;
}

public String getDataType()
{
    return dataType;
}

public String getDataBaseName()
{
    return dataBaseName;
}

public int getExtents()
{
    return extents;
}

public short getSessionError()
{
    return sessionError;
}

public char getRegular()
{
    return regular;
}

public char getHKey()
{
    return hKey;
}

public int getNumberOfDims()
{
    return numberOfDims;
}

public short[] getDimensions()
{
    return dimensions;
}

public String getVoxelUnits()
{
    return voxelUnits;
}

public String getCalUnits()
{
    return calUnits;
}

public short getUnused()
{
    return unused;
}

public int getDataTypeCode()
```

```
{
    return dataTypeCode;
}

public int getBitsPerPixel()
{
    return bitsPerPixel;
}

public float[] getPixelsDims()
{
    return pixelsDims;
}

public float getVoxelOffset()
{
    return voxelOffset;
}

public float getFUnusedOne()
{
    return fUnusedOne;
}

public float getFUnusedTwo()
{
    return fUnusedTwo;
}

public float getFUnusedThree()
{
    return fUnusedThree;
}

public float getCalMax()
{
    return calMax;
}

public float getCalMin()
{
    return calMin;
}

public float getCompressed()
{
    return compressed;
}

public float getVerified()
{
    return verified;
}

public int getGLMax()
{
    return glMax;
}

public int getGLMin()
{
    return glMin;
}

public String getDescrip()
{
    return descrip;
}

public String getAuxFile()
{
```

```
    return auxFile;
}

public char getOrient()
{
    return orient;
}

public String getOriginator()
{
    return originator;
}

public String getGenerated()
{
    return generated;
}

public String getScannum()
{
    return scannum;
}

public String getPatientID()
{
    return patientID;
}

public String getExpDate()
{
    return expDate;
}

public String getExpTime()
{
    return expTime;
}

public String getHistUn()
{
    return histUn;
}

public int getViews()
{
    return views;
}

public int getVolumesAdded()
{
    return volumesAdded;
}

public int getStartField()
{
    return startField;
}

public int getFieldSkip()
{
    return fieldSkip;
}

public int getOMax()
{
    return oMax;
}

public int getOMin()
{
    return oMin;
```

```
    }

    public int getSMax()
    {
        return sMax;
    }

    public int getSMin()
    {
        return sMin;
    }

    public byte[] getHdrBytesArray(boolean bigEndian)
    {
        BytesArray hdrBytesArray = new BytesArray(headerSize, bigEndian);
        int i;

        hdrBytesArray.setInt(headerSize, 0);
        hdrBytesArray.setString(dataType, 4, 10);
        hdrBytesArray.setString(dataBaseName, 14, 18);
        hdrBytesArray.setInt(extents, 32);
        hdrBytesArray.setShort(sessionError, 36);
        hdrBytesArray.setChar(regular, 38);
        hdrBytesArray.setChar(hKey, 39);
        for(i=0; i<8; i++)
        {
            hdrBytesArray.setShort(dimensions[i], 40+2*i);
        }
        hdrBytesArray.setString(voxelUnits, 56, 4);
        hdrBytesArray.setString(calUnits, 60, 8);
        hdrBytesArray.setShort(unused, 68);
        hdrBytesArray.setShort(dataTypeCode, 70);
        hdrBytesArray.setShort(bitsPerPixel, 72);
        hdrBytesArray.setShort(dimUn, 74);
        for(i=0; i<8; i++)
        {
            hdrBytesArray.setFloat(pixelsDims[i], 76+4*i);
        }
        hdrBytesArray.setFloat(voxelOffset, 108);
        hdrBytesArray.setFloat(fUnusedOne, 112);
        hdrBytesArray.setFloat(fUnusedTwo, 116);
        hdrBytesArray.setFloat(fUnusedThree, 120);
        hdrBytesArray.setFloat(calMax, 124);
        hdrBytesArray.setFloat(calMin, 128);
        hdrBytesArray.setFloat(compressed, 132);
        hdrBytesArray.setFloat(verified, 136);
        hdrBytesArray.setInt(glMax, 140);
        hdrBytesArray.setInt(glMin, 144);
        hdrBytesArray.setString(descrip, 148, 80);
        hdrBytesArray.setString(auxFile, 228, 24);
        hdrBytesArray.setChar(orient, 252);
        hdrBytesArray.setString(originator, 253, 10);
        hdrBytesArray.setString(generated, 263, 10);
        hdrBytesArray.setString(scannum, 273, 10);
        hdrBytesArray.setString(patientID, 283, 10);
        hdrBytesArray.setString(expDate, 293, 10);
        hdrBytesArray.setString(expTime, 303, 10);
        hdrBytesArray.setString(histUn, 313, 3);
        hdrBytesArray.setInt(views, 316);
        hdrBytesArray.setInt(volumesAdded, 320);
        hdrBytesArray.setInt(startField, 324);
        hdrBytesArray.setInt(fieldSkip, 328);
        hdrBytesArray.setInt(oMax, 332);
        hdrBytesArray.setInt(oMin, 336);
        hdrBytesArray.setInt(sMax, 340);
        hdrBytesArray.setInt(sMin, 344);

        return hdrBytesArray.getBytesArray();
    }
}
```

```
package functional.data;

//
//  AnlzReader.java
//
//
//  Created by JGLG on 4/22/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

import fileUtils.BytesArray;
import fileUtils.FileBytesReader;

public class AnlzReader
{
    //
    //  Data Members
    //

    private String hdrFilePath;

    private byte[] headerBytes;
    private boolean bigEndian;
    private BytesArray headerTag;

    private String imgFilePath;
    private byte[] imageBytes;

    private double[][][][] realImageData;
    private double[][][][] imaginaryImageData;

    //
    //  Constructors
    //
    //      anlzReader(String hdrFilePath)
    //

    public AnlzReader(String hdrFilePath) throws Exception
    {
        this.hdrFilePath = hdrFilePath;
        if(!findExtension(hdrFilePath, "hdr"))
        {
            throw new Exception("not hdr");
        }

        this.headerBytes = new FileBytesReader(hdrFilePath).getData();
        if((headerBytes[0] != 92)&(headerBytes[3] != 92))
        {
            throw new Exception("wrong header size");
        }

        this.bigEndian = isBigEndian();
        headerTag = new BytesArray(headerBytes, this.bigEndian);
    }

    //
    //  Public Methods
    //
    //      boolean isBigEndian()
    //      int getHeaderSize()
    //      String getDataType()
    //      String getDataBaseName()
    //      int getExtents()
    //      int getSessionError()
    //      char getRegular()
    //      char getHKey()
    //      int getNumberOfDims()
    //      short[] getDimensions()
    //      String getVoxelUnits()
    //      String getCalUnits()
    //      int getUnused()
```

```
//      int getDataTypeCode()
//      int getBitsPerPixel()
//      float[] getPixelsDims()
//      float getVoxelOffset()
//      float getFUnusedOne()
//      float getFUnusedTwo()
//      float getFUnusedThree()
//      float getCalMax()
//      float getCalMin()
//      float getCompressed()
//      float getVerified()
//      int getGLMax()
//      int getGLMin()
//      String getDescrip()
//      String getAuxFile()
//      char getOrient()
//      String getOriginator()
//      String getGenerated()
//      String getScannum()
//      String getPatientID()
//      String getExpDate()
//      String getExpTime()
//      String getHistUn()
//      int getViews()
//      int getVolumesAdded()
//      int getStartField()
//      int getFieldSkip()
//      int getOMax()
//      int getOMin()
//      int getSMax()
//      int getSMin()
//      void showHeader()
//      double[][][][] getRealImageData()
//      double[][][][] getImaginaryImageData()
//      boolean dataIsComplex()
//

public boolean isBigEndian()
{
    boolean bEndian;

    if(headerBytes[0] == 92)
    {
        bEndian = false;
    }
    else
    {
        bEndian = true;
    }

    return bEndian;
}

public int getHeaderSize()
{
    return headerTag.getInt(0);
}

public String getDataType()
{
    return headerTag.getString(4, 10);
}

public String getDataBaseName()
{
    return headerTag.getString(14, 18);
}

public int getExtents()
{
    return headerTag.getInt(32);
```

```
}

public int getSessionError()
{
    return headerTag.getShort(36);
}

public char getRegular()
{
    return headerTag.getChar(38);
}

public char getHKey()
{
    return headerTag.getChar(39);
}

public int getNumberOfDims()
{
    return headerTag.getShort(40);
}

public short[] getDimensions()
{
    int i;

    short[] dimensions = new short[8];

    for(i=0; i<8; i++)
    {
        dimensions[i] = headerTag.getShort(40+2*i);
    }

    return dimensions;
}

public String getVoxelUnits()
{
    return headerTag.getString(56, 4);
}

public String getCalUnits()
{
    return headerTag.getString(60, 8);
}

public int getUnused()
{
    return headerTag.getShort(68);
}

public int getDataTypeCode()
{
    return headerTag.getShort(70);
}

public int getBitsPerPixel()
{
    return headerTag.getShort(72);
}

public float[] getPixelsDims()
{
    float[] pixelsDims = new float[8];

    int i;

    for(i=0; i<8; i++)
    {
        pixelsDims[i] = headerTag.getFloat(76+4*i);
    }
```

```
        return pixelsDims;
    }

    public float getVoxelOffset()
    {
        return headerTag.getFloat(108);
    }

    public float getFUnusedOne()
    {
        return headerTag.getFloat(112);
    }

    public float getScaleFactor()
    {
        return getFUnusedOne();
    }

    public float getFUnusedTwo()
    {
        return headerTag.getFloat(116);
    }

    public float getFUnusedThree()
    {
        return headerTag.getFloat(120);
    }

    public float getCalMax()
    {
        return headerTag.getFloat(124);
    }

    public float getCalMin()
    {
        return headerTag.getFloat(128);
    }

    public float getCompressed()
    {
        return headerTag.getFloat(132);
    }

    public float getVerified()
    {
        return headerTag.getFloat(136);
    }

    public int getGLMax()
    {
        return headerTag.getInt(140);
    }

    public int getGLMin()
    {
        return headerTag.getInt(144);
    }

    public String getDescrip()
    {
        return headerTag.getString(148, 80);
    }

    public String getAuxFile()
    {
        return headerTag.getString(228, 24);
    }

    public char getOrient()
    {
```

```
    return headerTag.getChar(252);
}

public String getOriginator()
{
    return headerTag.getString(253, 10);
}

public String getGenerated()
{
    return headerTag.getString(263, 10);
}

public String getScannum()
{
    return headerTag.getString(273, 10);
}

public String getPatientID()
{
    return headerTag.getString(283, 10);
}

public String getExpDate()
{
    return headerTag.getString(293, 10);
}

public String getExpTime()
{
    return headerTag.getString(303, 10);
}

public String getHistUn()
{
    return headerTag.getString(313, 3);
}

public int getViews()
{
    return headerTag.getInt(316);
}

public int getVolumesAdded()
{
    return headerTag.getInt(320);
}

public int getStartField()
{
    return headerTag.getInt(324);
}

public int getFieldSkip()
{
    return headerTag.getInt(328);
}

public int getOMax()
{
    return headerTag.getInt(332);
}

public int getOMin()
{
    return headerTag.getInt(336);
}

public int getSMax()
{
    return headerTag.getInt(340);
```

```
}

public int getSMin()
{
    return headerTag.getInt(344);
}

public void showHeader()
{
    int numberOfDims = getNumberOfDims();
    short[] dimensions = getDimensions();
    float[] pixelsDims = getPixelsDims();

    System.out.println("\t\t******* Header Info *******");
    System.out.println("");
    System.out.println("\tSize of header    \t" + getHeaderSize());
    System.out.println("\tData type         \t" + getDataType());
    System.out.println("\tDatabase Name     \t" + getDataBaseName());
    System.out.println("\tExtents           \t" + getExtents());
    System.out.println("\tSession error     \t" + getSessionError());
    System.out.println("\tRegular           \t" + getRegular());
    System.out.println("\tH Key             \t" + getHKey());
    System.out.println("\tNumber of dims    \t" + getNumberOfDims());
    if(numberOfDims > 0)
    {
        System.out.println("\tWidth             \t" + dimensions[1]);
    }
    if(numberOfDims > 1)
    {
        System.out.println("\tHeight            \t" + dimensions[2]);
    }
    if(numberOfDims > 2)
    {
        System.out.println("\tNumber of slices  \t" + dimensions[3]);
    }
    if(numberOfDims > 3)
    {
        System.out.println("\tNumber of instants\t" + dimensions[4]);
    }
    System.out.println("\tVoxel units       \t" + getVoxelUnits());
    System.out.println("\tCalibration units \t" + getCalUnits());
    System.out.println("\tUnused            \t" + getUnused());
    System.out.println("\tData type code    \t" + getDataTypeCode());
    System.out.println("\tBits per pixel    \t" + getBitsPerPixel());
    if(numberOfDims > 0)
    {
        System.out.println("\tVoxel width       \t" + pixelsDims[1]);
    }
    if(numberOfDims > 1)
    {
        System.out.println("\tVoxel height      \t" + pixelsDims[2]);
    }
    if(numberOfDims > 2)
    {
        System.out.println("\tSlice thickness   \t" + pixelsDims[3]);
    }
    /*
    if(numberOfDims > 3)
    {
        System.out.println("\tAcquisition time  \t" + pixelsDims[4]);
    }
    */
    System.out.println("\tVoxels offset     \t" + getVoxelOffset());
    System.out.println("\tSPM: Scale factor \t" + getFUnusedOne());
    System.out.println("\tSPM2: Int zero int\t" + getFUnusedTwo());
    System.out.println("\tUnused            \t" + getFUnusedThree());
    System.out.println("\tCalibration max   \t" + getCalMax());
    System.out.println("\tCalibration min   \t" + getCalMin());
    System.out.println("\tCompressed        \t" + getCompressed());
    System.out.println("\tVerified          \t" + getVerified());
    System.out.println("\tglMax             \t" + getGLMax());
```

```
        System.out.println("\tglMin            \t" + getGLMin());
        System.out.println("\tDescription      \t" + getDescrip());
        System.out.println("\tAux file         \t" + getAuxFile());
        System.out.println("\tOrientation      \t" + getOrient());
        System.out.println("\tOriginator       \t" + getOriginator());
        System.out.println("\tGenerated        \t" + getGenerated());
        System.out.println("\tScannum          \t" + getScannum());
        System.out.println("\tPatient ID       \t" + getPatientID());
        System.out.println("\tExposure date    \t" + getExpDate());
        System.out.println("\tExposure time    \t" + getExpTime());
        System.out.println("\tHist unused      \t" + getHistUn());
        System.out.println("\tViews            \t" + getViews());
        System.out.println("\tVolumes added    \t" + getVolumesAdded());
        System.out.println("\tStart field      \t" + getStartField());
        System.out.println("\tField skip       \t" + getFieldSkip());
        System.out.println("\to max            \t" + getOMax());
        System.out.println("\to min            \t" + getOMin());
        System.out.println("\ts max            \t" + getSMax());
        System.out.println("\ts min            \t" + getSMin());
        System.out.println("");
    }

    public double[][][][] getRealImageData()
    {
        short[] dimensions = getDimensions();
        int bytesPerVoxel = getBitsPerPixel()/8;
        int dataTypeCode = getDataTypeCode();

        int x, y, z;
        int t;

        int width = dimensions[1];
        int height = dimensions[2];
        int numberOfSlices = dimensions[3];
        int numberOfInstants = dimensions[4];

        BytesArray imageBytesArray;

        realImageData =
            new double[width][height][numberOfSlices][numberOfInstants];

        int index = 0;

        // change path from *.hdr to .img
        imgFilePath = changePathToImage(hdrFilePath);

        imageBytes = new FileBytesReader(imgFilePath).getData();

        imageBytesArray = new BytesArray(imageBytes, bigEndian);

        for(t=0; t< numberOfInstants; t++)
        {
            for(z=0; z< numberOfSlices; z++)
            {
                for(y=0; y< height; y++)
                {
                    //for(x=0; x< width; x++)
                    for(x=width-1; x> -1; x--) // LAI ou RAI?
                    {
                        if(dataTypeCode == 1)   // DT_BINARY
                        {
                            bytesPerVoxel = 1;
                            realImageData[x][y][z][t] =
                            (double)imageBytesArray.getByte(bytesPerVoxel
                                                            *index);
                            index = index + 1;
                        }

                        if(dataTypeCode == 2)   // DT_UNSIGNED_CHAR
                        {
                            realImageData[x][y][z][t] =
```

```
                        (double)imageBytesArray.getByte(bytesPerVoxel
                                                        *index);
                        index = index + 1;
                    }

                    if(dataTypeCode == 4)   // DT_SIGNED_SHORT
                    {
                        realImageData[x][y][z][t] =
                        (double)imageBytesArray.getShort(bytesPerVoxel
                                                        *index);
                        index = index + 1;
                    }

                    if(dataTypeCode == 8)   // DT_SIGNED_INT
                    {
                        realImageData[x][y][z][t] =
                        (double)imageBytesArray.getInt(bytesPerVoxel
                                                        *index);
                        index = index + 1;
                    }

                    if(dataTypeCode == 16)  // DT_FLOAT
                    {
                        realImageData[x][y][z][t] =
                        (double)imageBytesArray.getFloat(bytesPerVoxel
                                                        *index);
                        index = index + 1;
                    }

                    if(dataTypeCode == 32)  // DT_COMPLEX
                    {
                        realImageData[x][y][z][t] =
                        (imageBytesArray.getComplex(bytesPerVoxel
                                                    *index)).getReal();
                        index = index + 1;
                    }

                    if(dataTypeCode == 64)  // DT_DOUBLE
                    {
                        realImageData[x][y][z][t] =
                        imageBytesArray.getDouble(bytesPerVoxel*index);
                        index = index + 1;
                    }
                }
            }
        }
    }

    return realImageData;
}

public boolean[][][] getBooleanRealImageData(int instant, double threshold)
{
    int x, y, z, t;

    short[] dimensions = getDimensions();

    int width = dimensions[1];
    int height = dimensions[2];
    int numberOfSlices = dimensions[3];

    boolean[][][] booleanRealImageData =
                    new boolean[width][height][numberOfSlices];

    double[][][][] dataArray;

    t = instant;

    dataArray = getRealImageData();

    for(z=0; z< numberOfSlices; z++)
```

```
    {
        for(y=0; y< height; y++)
        {
            for(x=width-1; x> -1; x--)
            {
                if(dataArray[x][y][z][t] > threshold)
                {
                    booleanRealImageData[x][y][z] = true;
                }
                else
                {
                    booleanRealImageData[x][y][z] = false;
                }
            }
        }
    }


    return booleanRealImageData;
}

public double[][][] getRealImageDataVolume(int instant)
{
    return getInstantVolume(0, true);
}

public double[][][][] getImaginaryImageData()
{
    short[] dimensions = getDimensions();
    int bytesPerVoxel = getBitsPerPixel()/8;
    int dataTypeCode = getDataTypeCode();

    int x, y, z;
    int t;

    int width = dimensions[1];
    int height = dimensions[2];
    int numberOfSlices = dimensions[3];
    int numberOfInstants = dimensions[4];

    BytesArray imageBytesArray;

    imaginaryImageData =
        new double[width][height][numberOfSlices][numberOfInstants];

    int index = 0;

    // change path from *.hdr to .img
    imgFilePath = changePathToImage(hdrFilePath);

    imageBytes = new FileBytesReader(imgFilePath).getData();

    imageBytesArray = new BytesArray(imageBytes, bigEndian);

    for(t=0; t< numberOfInstants; t++)
    {
        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                //for(x=0; x< width; x++)
                for(x=width-1; x> -1; x--) // LAI ou RAI?
                {
                    if(dataTypeCode == 32)  // DT_COMPLEX
                    {
                        imaginaryImageData[x][y][z][t] =
                        (imageBytesArray.getComplex(bytesPerVoxel
                                                    *index)).getImg();
                        index = index + 1;
                    }
                    else
```

```
                    {
                        imaginaryImageData[x][y][z][t] = 0;
                        index = index + 1;
                    }
                }
            }
        }
    }

    return imaginaryImageData;
}

public double[][][] getImaginaryImageDataVolume(int instant)
{
    return getInstantVolume(0, false);
}

/*
public Complex[][][][] getComplexImageData()
{

}
*/

public boolean dataIsComplex()
{
    int dataTypeCode = getDataTypeCode();

    if(dataTypeCode == 32)
    {
        return true;
    }
    else
    {
        return false;
    }
}

//
//  Private Methods
//
//      boolean findExtension(String text, String extension)
//      String changePathToImage(String filePath)
//

private boolean findExtension(String text, String extension)
{
    boolean extensionIsPresent = false;
    int textSize;
    int extensionSize;
    String textExtension;

    textSize = text.length();
    extensionSize = extension.length();

    textExtension = text.substring(textSize-extensionSize);

    if(textExtension.equals(extension))
    {
        extensionIsPresent = true;
    }

    return extensionIsPresent;
}

/******* CHANGE PATH TO IMG *******/

private String changePathToImage(String filePath)
{
    int indexOfDot;
    String pathBase;
```

```
        String extension;
        String newFilePath;

        indexOfDot = filePath.indexOf(".hdr");
        pathBase = filePath.substring(0,indexOfDot+1);
        extension = "img";
        newFilePath = pathBase + extension;

        return newFilePath;
    }

    private double[][][] getInstantVolume(int instant, boolean real)
    {
        short[] dimensions = getDimensions();

        int width = dimensions[1];
        int height = dimensions[2];
        int numberOfSlices = dimensions[3];

        int x, y, z;
        double[][][] instantVolume =
                        new double[width][height][numberOfSlices];

        double[][][][] usedData;

        if(real)
        {
            usedData = getRealImageData();
        }
        else
        {
            usedData = getImaginaryImageData();
        }

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    instantVolume[x][y][z] =
                            usedData[x][y][z][instant];
                }
            }
        }

        return instantVolume;
    }
}
```

```
package functional.data;

//
//  AnlzWriter.java
//
//
//  Created by JGLG on 11/22/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

import fileUtils.FileBytesWriter;

public class AnlzWriter
{
    //
    //  Data Members
    //

    private boolean defBigEndian = true;
    private boolean bigEndian = defBigEndian;

    private AnlzHeader header;
    private double[][][][] realData;
    private double[][][][] imaginaryData;
    private String filePath;


    //
    //  Constructors
    //
    //      anlzReader(String hdrFilePath)
    //

    public AnlzWriter(AnlzHeader header,
                      double[][][][] realData,
                      String filePath)
                          throws IOException
    {
        this.header = header;
        this.realData = realData;
        this.filePath = filePath;

        writeHdr();
        writeImg();
    }

    public AnlzWriter(AnlzHeader header,
                      double[][][][] realData,
                      double[][][][] imaginaryData,
                      String filePath)
                          throws IOException
    {
        this.header = header;
        this.realData = realData;
        this.imaginaryData = imaginaryData;
        this.filePath = filePath;

        writeHdr();
        writeImg();
    }

    public AnlzWriter(AnlzHeader header,
                      double[][][] realData,
                      String filePath)
                          throws IOException
    {
        this.header = header;
```

```
        this.filePath = filePath;

        this.realData = convertToFourD(realData);

        writeHdr();
        writeImg();
    }

    private void writeHdr()
    {
        String hdrFilePath;

        hdrFilePath = filePath + ".hdr";
        new FileBytesWriter(header.getHdrBytesArray(bigEndian), hdrFilePath);
    }

    private void writeImg() throws IOException
    {
        String imgFilePath;
        FileOutputStream imgFile;
        DataOutputStream imgData;

        int x, y, z;
        int t;

        int width, height, numberOfSlices;
        int numberOfInstants;

        short[] dim = header.getDimensions();
        int dataTypeCode = header.getDataTypeCode();


        width = dim[1];
        height = dim[2];
        numberOfSlices = dim[3];
        numberOfInstants = dim[4];

        imgFilePath = filePath + ".img";


        imgFile = new FileOutputStream(imgFilePath);
        imgData = new DataOutputStream(imgFile);

        for(t=0; t< numberOfInstants; t++)
        {
            for(z=0; z< numberOfSlices; z++)
            {
                for(y=0; y< height; y++)
                {
                    //for(x=0; x< width; x++)
                    for(x=width-1; x> -1; x--) // LAI ou RAI?
                    {
                        if(dataTypeCode == 1.0) // DT_BINARY
                        {
                            boolean value;

                            if(realData[x][y][z][t] == 1.0)
                            {
                                value = true;
                            }
                            else
                            {
                                value = false;
                            }
                            imgData.writeBoolean(value);
                        }

                        if(dataTypeCode == 2)   // DT_UNSIGNED_CHAR
                        {
                            imgData.writeChar
                                ((char)Math.round(realData[x][y][z][t]));
```

```
                    }

                    if(dataTypeCode == 4)   // DT_SIGNED_SHORT
                    {
                        imgData.writeShort
                            ((short)Math.round(realData[x][y][z][t]));
                    }

                    if(dataTypeCode == 8)   // DT_SIGNED_INT
                    {
                        imgData.writeInt
                            ((int)Math.round(realData[x][y][z][t]));
                    }

                    if(dataTypeCode == 16)  // DT_FLOAT
                    {
                        imgData.writeFloat
                            ((float)realData[x][y][z][t]);
                    }

                    if(dataTypeCode == 32)  // DT_COMPLEX
                    {
                        imgData.writeFloat
                            ((float)realData[x][y][z][t]);
                        imgData.writeFloat
                            ((float)imaginaryData[x][y][z][t]);
                    }

                    if(dataTypeCode == 64)  // DT_DOUBLE
                    {
                        imgData.writeDouble
                            (realData[x][y][z][t]);
                    }
                }
            }
        }
    }
}

//
//  Public Methods
//
//      int[] setDimensions()
//      int[][][][] setData()
//



//
//  Private Methods
//
//      byte[] readImageHeader(String filePath)
//      String changePathToImage(String filePath)
//      int[][][][] readDataMatrix(String filePath)
//

private double[][][][] convertToFourD(double[][][] dataArray)
{
    int width = dataArray.length;
    int height = dataArray[0].length;
    int numberOfSlices = dataArray[0][0].length;
    double[][][][] fourDArray =
        new double[width][height][numberOfSlices][1];
    int x, y, z;

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
```

```
                    fourDArray[x][y][z][0] = dataArray[x][y][z];
                }
            }
        }

        return fourDArray;
    }

    /******* CHANGE PATH TO IMG ******

    private String changePathToImage(String filePath)
    {
        int indexOfDot;
        String pathBase;
        String extension;
        String newFilePath;

        indexOfDot = filePath.indexOf(".hdr");
        pathBase = filePath.substring(0,indexOfDot+1);
        extension = "img";
        newFilePath = pathBase + extension;

        return newFilePath;
    }
    */
}
```

```java
package matematica.specialFunctions;

//
//  Beta.java
//
//
//  Created by Jose Gabriel Lira Gomes on 8/1/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

public class Beta
{
    private double z;
    private double w;

    public Beta(double z, double w)
    {
        this.z = z;
        this.w = w;
    }

    public double getBeta()
    {
        double gammaOfZ;
        double gammaOfW;
        double gammaOfZPlusW;

        double beta;

        gammaOfZ = new Gamma(z).getGamma();
        gammaOfW = new Gamma(w).getGamma();
        gammaOfZPlusW = new Gamma(z+w).getGamma();

        beta = gammaOfZ*gammaOfW/gammaOfZPlusW;

        return beta;
    }
}
```

```
package imageUtils;

//
//  BlockImageBuilder.java
//
//
//  Created by José Gabriel Lira Gomes on 3/1/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class BlockImageBuilder
{
    private int rectangleWidth;
    private int rectangleHeight;

    private double[][] blocksDataArray;

    private int numberOfHorizPoints;
    private int numberOfVertiPoints;

    public BlockImageBuilder(int rectangleWidth,
                             int rectangleHeight,
                             int numberOfColors,
                             float intensity)
    {
        this.rectangleWidth = rectangleWidth;
        this.rectangleHeight = rectangleHeight;
    }

    public void setMatrixBlocks(double[][] dataArray)
    {
        this.numberOfVertiPoints = dataArray.length;
        this.numberOfHorizPoints = dataArray[0].length;

        this.blocksDataArray = getBlocksDataArray(dataArray);
    }

    public void setHorizBlocks(double[] dataArray)
    {
        double[][] lineDataArray;

        this.numberOfHorizPoints = dataArray.length;
        this.numberOfVertiPoints = 1;

        lineDataArray = convertToLine(dataArray);
        this.blocksDataArray = getBlocksDataArray(lineDataArray);
    }

    public void updateHorizBlocks(double[] dataArray)
    {
        double[][] lineDataArray;

        lineDataArray = convertToLine(dataArray);
        this.blocksDataArray = getBlocksDataArray(lineDataArray);
    }

    public void setVertiBlocks(double[] dataArray)
    {
        double[][] columnDataArray;

        this.numberOfHorizPoints = 1;
        this.numberOfVertiPoints = dataArray.length;

        columnDataArray = convertToColumn(dataArray);
        this.blocksDataArray = getBlocksDataArray(columnDataArray);
    }

    public void setVertiBlocksData(double[] dataArray)
    {
        double[][] columnDataArray;
```

```
        columnDataArray = convertToColumn(dataArray);
        this.blocksDataArray = getBlocksDataArray(columnDataArray);
    }

    public double[][] getDataArray()
    {
        return blocksDataArray;
    }


    private double[][] getBlocksDataArray(double[][] dataArray)
    {
        double[][] xtendedDataArray =
            new double[numberOfVertiPoints][numberOfHorizPoints
                                            *rectangleWidth];
        double[][] xytendedDataArray =
            new double[numberOfVertiPoints*rectangleHeight]
                            [numberOfHorizPoints*rectangleWidth];

        int i, j;
        int k;

        int line;
        int column;

        for(line=0; line<numberOfVertiPoints; line++)
        {
            k = 0;
            for(i=0; i<numberOfHorizPoints; i++)
            {
                for(j=0; j<rectangleWidth; j++)
                {
                    xtendedDataArray[line][k] = (double)dataArray[line][i];
                    k = k + 1;
                }
            }
        }

        for(column=0; column<numberOfHorizPoints*rectangleWidth; column++)
        {
            k = 0;
            for(i=0; i<numberOfVertiPoints; i++)
            {
                for(j=0; j<rectangleHeight; j++)
                {
                    xytendedDataArray[k][column] =
                                xtendedDataArray[i][column];
                    k = k + 1;
                }
            }
        }

        return xytendedDataArray;
    }


    private double[][] convertToColumn(double[] dataArray)
    {
        int dataSize = dataArray.length;
        double[][] convertedArray = new double[dataSize][1];
        int i;

        for(i=0; i<dataSize; i++)
        {
            convertedArray[i][0] = dataArray[i];
        }

        return convertedArray;
    }

    private double[][] convertToLine(double[] dataArray)
```

```
    {
        int dataSize = dataArray.length;
        double[][] convertedArray = new double[1][dataSize];
        int i;

        for(i=0; i<dataSize; i++)
        {
            convertedArray[0][i] = dataArray[i];
        }

        return convertedArray;
    }
}
```

```java
package imageUtils;

//
//  BlockImagePanel.java
//
//
//  Created by José Gabriel Lira Gomes on 10/31/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.image.MemoryImageSource;

import javax.swing.BorderFactory;
import javax.swing.JPanel;

public class BlockImagePanel extends JPanel
{
    private int panelWidth;
    private int panelHeight;
    private int rectangleWidth;
    private int rectangleHeight;
    private int numberOfColors;

    private double[][] blocksDataArray;

    private int width;
    private int height;

    private int numberOfHorizPoints;
    private int numberOfVertiPoints;

    private Image dataImage;

    public BlockImagePanel(int panelWidth,
                           int panelHeight,
                           int rectangleWidth,
                           int rectangleHeight,
                           int numberOfColors)
    {
        this.panelWidth = panelWidth;
        this.panelHeight = panelHeight;
        this.rectangleWidth = rectangleWidth;
        this.rectangleHeight = rectangleHeight;
        this.numberOfColors = numberOfColors;
    }

    public void setMatrixBlocks(double[][] dataArray)
    {
        this.numberOfVertiPoints = dataArray.length;
        this.numberOfHorizPoints = dataArray[0].length;

        width = numberOfHorizPoints*rectangleWidth;
        height = numberOfVertiPoints*rectangleHeight;

        this.blocksDataArray = getBlocksDataArray(dataArray);

        buildBlocksImage();
        setOptions();
    }

    public void setHorizBlocks(double[] dataArray)
    {
        double[][] lineDataArray;

        this.numberOfHorizPoints = dataArray.length;
        this.numberOfVertiPoints = 1;

        width = numberOfHorizPoints*rectangleWidth;
```

```
        height = rectangleHeight;

        lineDataArray = convertToLine(dataArray);
        this.blocksDataArray = getBlocksDataArray(lineDataArray);

        buildBlocksImage();
        setOptions();
    }

    public void updateHorizBlocks(double[] dataArray)
    {
        double[][] lineDataArray;

        lineDataArray = convertToLine(dataArray);
        this.blocksDataArray = getBlocksDataArray(lineDataArray);

        buildBlocksImage();
        repaint();
    }

    public void setVertiBlocks(double[] dataArray)
    {
        double[][] columnDataArray;

        this.numberOfHorizPoints = 1;
        this.numberOfVertiPoints = dataArray.length;

        width = rectangleWidth;
        height = numberOfVertiPoints*rectangleHeight;

        columnDataArray = convertToColumn(dataArray);
        this.blocksDataArray = getBlocksDataArray(columnDataArray);

        buildBlocksImage();
        setOptions();
    }

    public void setVertiBlocksData(double[] dataArray)
    {
        double[][] columnDataArray;

        columnDataArray = convertToColumn(dataArray);
        this.blocksDataArray = getBlocksDataArray(columnDataArray);

        buildBlocksImage();
        repaint();
    }

    public void paint(Graphics g)
    {
        super.paint(g);

        g.drawImage(dataImage,
                        Math.round((panelWidth-width)/2),
                        Math.round((panelHeight-height)/2),
                        this);
    }


    public void buildBlocksImage()
    {
        MemoryImageSource source;

        int i=0;

        int x, y;

        double colorLevel;
        int red, green, blue;

        int rgb;
```

```
        double hue;

        int pixels[] = new int[height*width];

        for(y=height-1; y >= 0; y--)
        {
            for(x=0; x< width; x++)
            {
                if(blocksDataArray[y][x] == -1.0)
                {
                    rgb = 13421772;
                }
                else
                {
                    colorLevel = blocksDataArray[y][x]*0.7
                                    /((double)numberOfColors-1.0);

                    hue = Math.abs(0.7-colorLevel);
                    rgb = Color.HSBtoRGB((float)hue, (float)0.7, (float)1.0);
                }

                red = (rgb>>16)&0xFF;
                green = (rgb>>8)&0xFF;
                blue = rgb&0xFF;

                pixels[i] = (255 << 24) | (red << 16) | (green << 8) | blue;

                i=i+1;
            }
        }


        source = new MemoryImageSource(width, height, pixels, 0, width);
        this.dataImage = createImage(source);
    }

    private void setOptions()
    {
        setBorder(BorderFactory.createLoweredBevelBorder());
        setBackground(Color.white);
    }

    private double[][] getBlocksDataArray(double[][] dataArray)
    {
        double[][] xtendedDataArray =
                new double[numberOfVertiPoints][numberOfHorizPoints
                                                    *rectangleWidth];
        double[][] xytendedDataArray =
                new double[numberOfVertiPoints*rectangleHeight]
                            [numberOfHorizPoints*rectangleWidth];

        int i, j;
        int k;

        int line;
        int column;

        for(line=0; line<numberOfVertiPoints; line++)
        {
            k = 0;
            for(i=0; i<numberOfHorizPoints; i++)
            {
                for(j=0; j<rectangleWidth; j++)
                {
                    xtendedDataArray[line][k] = (double)dataArray[line][i];
                    k = k + 1;
                }
            }
        }

        for(column=0; column<numberOfHorizPoints*rectangleWidth; column++)
```

```
        {
            k = 0;
            for(i=0; i<numberOfVertiPoints; i++)
            {
                for(j=0; j<rectangleHeight; j++)
                {
                    xytendedDataArray[k][column] =
                                xtendedDataArray[i][column];
                    k = k + 1;
                }
            }
        }

        return xytendedDataArray;
    }


    private double[][] convertToColumn(double[] dataArray)
    {
        int dataSize = dataArray.length;
        double[][] convertedArray = new double[dataSize][1];
        int i;

        for(i=0; i<dataSize; i++)
        {
            convertedArray[i][0] = dataArray[i];
        }

        return convertedArray;
    }

    private double[][] convertToLine(double[] dataArray)
    {
        int dataSize = dataArray.length;
        double[][] convertedArray = new double[1][dataSize];
        int i;

        for(i=0; i<dataSize; i++)
        {
            convertedArray[0][i] = dataArray[i];
        }

        return convertedArray;
    }
}
```

```java
package fileUtils;

//
//  BytesArray.java
//
//
//  Created by José Gabriel Lira Gomes on 11/17/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.nio.ByteBuffer;
import java.nio.ByteOrder;

import matematica.complex.Complex;

public class BytesArray
{
    private byte[] bytesArray;
    private boolean bigEndian;

    public BytesArray(byte[] bytesArray, boolean bigEndian)
    {
        this.bytesArray = bytesArray;
        this.bigEndian = bigEndian;
    }

    public BytesArray(int arraySize, boolean bigEndian)
    {
        this.bytesArray = new byte[arraySize];
        this.bigEndian = bigEndian;
    }

    public byte[] getBytesArray()
    {
        return bytesArray;
    }

    public byte getByte(int byteIndex)
    {
        return bytesArray[byteIndex];
    }

    public short getShort(int firstByteIndex)
    {
        int i;
        short number = 0;
        int numberOfBytes = 2;

        for(i=0; i<numberOfBytes; i++)
        {
            if(bigEndian)
            {
                number |= ( (short)( bytesArray[firstByteIndex+i] & 0xff ) )
                                    << 8*(numberOfBytes-1-i);
            }
            else
            {
                number |= ( (short)( bytesArray[firstByteIndex+i] & 0xff ) )
                                    << 8*i;
            }
        }

        return number;
    }

    public int getInt(int firstByteIndex)
    {
        int i;
        int number = 0;
        int numberOfBytes = 4;
```

```
        for(i=0; i<numberOfBytes; i++)
        {
            if(bigEndian)
            {
                number |= ( (int)( bytesArray[firstByteIndex+i] & 0xff ) )
                                << 8*(numberOfBytes-1-i);
            }
            else
            {
                number |= ( (int)( bytesArray[firstByteIndex+i] & 0xff ) )
                                << 8*i;
            }
        }

        return number;
    }

    public long getLong(int firstByteIndex)
    {
        int i;
        long number = 0;
        int numberOfBytes = 8;

        for(i=0; i<numberOfBytes; i++)
        {
            if(bigEndian)
            {
                number |= ( (long)( bytesArray[firstByteIndex+i] & 0xff ) )
                                << 8*(numberOfBytes-1-i);
            }
            else
            {
                number |= ( (long)( bytesArray[firstByteIndex+i] & 0xff ) )
                                << 8*i;
            }
        }

        return number;
    }

    public float getFloat(int firstByteIndex)
    {
        int i;
        int number = 0;
        int numberOfBytes = 4;

        for(i=0; i<numberOfBytes; i++)
        {
            if(bigEndian)
            {
                number |= ( (int)( bytesArray[firstByteIndex+i] & 0xff ) )
                                << 8*(numberOfBytes-1-i);
            }
            else
            {
                number |= ( (int)( bytesArray[firstByteIndex+i] & 0xff ) )
                                << 8*i;
            }
        }

        return Float.intBitsToFloat(number);
    }

    public Complex getComplex(int firstByteIndex)
    {
        int numberOfBytes = 4;
        double real;
        double imaginary;

        real = (double)getFloat(firstByteIndex);
```

```
        imaginary = (double)getFloat(firstByteIndex+numberOfBytes);

        return new Complex(real, imaginary);
    }

    public double getDouble(int firstByteIndex)
    {
        int i;
        long number = 0;
        int numberOfBytes = 8;

        for(i=0; i<numberOfBytes; i++)
        {
            if(bigEndian)
            {
                number |= ( (long)( bytesArray[firstByteIndex+i] & 0xff ) )
                                    << 8*(numberOfBytes-1-i);
            }
            else
            {
                number |= ( (long)( bytesArray[firstByteIndex+i] & 0xff ) )
                                    << 8*i;
            }
        }

        return Double.longBitsToDouble(number);
    }

    public char getChar(int byteIndex)
    {
        return (char)bytesArray[byteIndex];
    }

    public String getString(int firstByteIndex, int length)
    {
        int i;
        String text = "";

        for(i=0; i<length; i++)
        {
            text = text + (char)bytesArray[firstByteIndex+i];
        }
        return text;
    }

    public void setByte(byte number, int byteIndex)
    {
        bytesArray[byteIndex] = number;
    }

    public void setShort(short number, int firstByteIndex)
    {
        int i;
        byte[] b = shortToByteArray(number);

        for(i=0; i<b.length; i++)
        {
            bytesArray[firstByteIndex+i] = b[i];
        }
    }

    public void setInt(int number, int firstByteIndex)
    {
        int i;
        byte[] b = intToByteArray(number);

        for(i=0; i<b.length; i++)
        {
            bytesArray[firstByteIndex+i] = b[i];
        }
    }
```

```
public void setLong(long number, int firstByteIndex)
{
    int i;
    byte[] b = longToByteArray(number);

    for(i=0; i<b.length; i++)
    {
        bytesArray[firstByteIndex+i] = b[i];
    }
}

public void setFloat(float number, int firstByteIndex)
{
    int i;
    byte[] b = floatToByteArray(number);

    for(i=0; i<b.length; i++)
    {
        bytesArray[firstByteIndex+i] = b[i];
    }
}

public void setDouble(double number, int firstByteIndex)
{
    int i;
    byte[] b = doubleToByteArray(number);

    for(i=0; i<b.length; i++)
    {
        bytesArray[firstByteIndex+i] = b[i];
    }
}

public void setChar(char letter, int byteIndex)
{
    byte b = charToByteArray(letter);

    bytesArray[byteIndex] = b;
}

public void setString(String text, int firstByteIndex, int textSize)
{
    int i;
    byte[] b = stringToByteArray(text);
    int arraySize = b.length;

    if(arraySize >= textSize)
    {
        for(i=0; i<textSize; i++)
        {
            bytesArray[firstByteIndex+i] = b[i];
        }
    }
    else
    {
        if(arraySize==0)
        {
            for(i=0; i<textSize; i++)
            {
                bytesArray[firstByteIndex+i] = 0;
            }
        }
        else
        {
            for(i=0; i<arraySize; i++)
            {
                bytesArray[firstByteIndex+i] = b[i];
            }
            for(i=0; i<textSize-arraySize; i++)
            {
```

```
                bytesArray[firstByteIndex+arraySize+i] = 0;
            }
        }
    }
}

private byte[] shortToByteArray(short number)
{
    ByteBuffer buf = ByteBuffer.allocate(2);
    if(bigEndian)
    {
        buf.order(ByteOrder.BIG_ENDIAN);
    }
    else
    {
        buf.order(ByteOrder.LITTLE_ENDIAN);
    }
    buf.putShort(number);

    return buf.array();
}

private byte[] intToByteArray(int number)
{
    ByteBuffer buf = ByteBuffer.allocate(4);
    if(bigEndian)
    {
        buf.order(ByteOrder.BIG_ENDIAN);
    }
    else
    {
        buf.order(ByteOrder.LITTLE_ENDIAN);
    }
    buf.putInt(number);

    return buf.array();
}

private byte[] longToByteArray(long number)
{
    ByteBuffer buf = ByteBuffer.allocate(8);
    if(bigEndian)
    {
        buf.order(ByteOrder.BIG_ENDIAN);
    }
    else
    {
        buf.order(ByteOrder.LITTLE_ENDIAN);
    }
    buf.putLong(number);

    return buf.array();
}

private byte[] floatToByteArray(float number)
{
    ByteBuffer buf = ByteBuffer.allocate(4);
    if(bigEndian)
    {
        buf.order(ByteOrder.BIG_ENDIAN);
    }
    else
    {
        buf.order(ByteOrder.LITTLE_ENDIAN);
    }
    buf.putFloat(number);

    return buf.array();
}

private byte[] doubleToByteArray(double number)
```

```
    {
        ByteBuffer buf = ByteBuffer.allocate(8);
        if(bigEndian)
        {
            buf.order(ByteOrder.BIG_ENDIAN);
        }
        else
        {
            buf.order(ByteOrder.LITTLE_ENDIAN);
        }
        buf.putDouble(number);

        return buf.array();
    }

    private byte charToByteArray(char letter)
    {
        return (byte)letter;
    }

    private byte[] stringToByteArray(String text)
    {
        return text.getBytes();
    }
}
```

```
package functional.data;

//
//  Cluster.java
//
//
//  Created by José Gabriel Lira Gomes on 2/13/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import matematica.algebra.Matrix;
import estatistica.descriptive.MultipleArrayAnalyzer;
import estatistica.descriptive.OneDArrayAnalyzer;
import estatistica.descriptive.ThreeDArrayAnalyzer;
import estatistica.distributions.Histogram;
import estatistica.distributions.MultiGaussianDistribution;

public class Cluster
{
    private Matrix multipleArrayMatrix;
    private Matrix meansMatrix;
    private Matrix covarianceMatrix;
    private MultiGaussianDistribution clusterDistribution;

    private double[][][][] imageData;
    private Voxel[] sampleVoxels;

    private int numberOfVoxels;

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfVolumes;


    public Cluster(double[][][][] imageData, Voxel[] sampleVoxels)
    {
        this.imageData = imageData;

        getImageDims();

        this.sampleVoxels = sampleVoxels;

        this.numberOfVoxels = sampleVoxels.length;

        getMultipleArray();

        getMeansMatrix();

        getCovarianceMatrix();

        getClusterDistribution();
    }

    public MultiGaussianDistribution getDistribution()
    {
        return clusterDistribution;
    }

    public Voxel[] getSampleVoxels()
    {
        return sampleVoxels;
    }

    public double[] getSampleValues()
    {
        double[] sampleValues = new double[numberOfVoxels];
        int i;

        for(i=0; i<numberOfVoxels; i++)
        {
```

```
            sampleValues[i] = imageData[sampleVoxels[i].getX()]
                                       [sampleVoxels[i].getY()]
                                       [sampleVoxels[i].getZ()]
                                       [0];
        }

        return sampleValues;
    }

    public int getNumberOfClusterVoxels()
    {
        Histogram firstVolumeHist;
        OneDArrayAnalyzer clusterAnalyzer;
        double clusterMean;
        double clusterStdDev;
        double meanProbDens;

        int numberOfClusterVoxels;
        double volumeNumberOfVoxels = (double)width*height*numberOfSlices;

        firstVolumeHist = new ThreeDArrayAnalyzer
                              (extractInstantVolume(0)).getHistogram(1);

        clusterAnalyzer = new OneDArrayAnalyzer(getSampleValues());
        clusterMean = clusterAnalyzer.getMean();
        clusterStdDev = clusterAnalyzer.getStdDev();

        meanProbDens = firstVolumeHist.getProbDensity(clusterMean);

        numberOfClusterVoxels = (int)Math.round(clusterStdDev
                                                *meanProbDens
                                                *volumeNumberOfVoxels);

        return numberOfClusterVoxels;
    }

    public Mask getSimpleMask(double range)
    {
        int x, y, z, t;
        double[] valueArray = new double[numberOfVolumes];
        Matrix value;
        double mahDistance;
        boolean[][][] dataMaskArray =
            new boolean[width][height][numberOfSlices];
        Mask dataMask;


        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    for(t=0; t< numberOfVolumes; t++)
                    {
                        valueArray[t] = imageData[x][y][z][t];
                    }
                    value = new Matrix(valueArray, false);
                    mahDistance = clusterDistribution.getMahDistance(value);

                    if(mahDistance < range)
                    {
                        dataMaskArray[x][y][z] = true;
                        //System.out.println("("+x+","+y+","+z+"): "
                        //                              +mahDistance);
                    }
                    else
                    {
                        dataMaskArray[x][y][z] = false;
                    }
                }
```

```
            }
        }
        dataMask = new Mask(dataMaskArray);

        return dataMask;
    }

    private void getImageDims()
    {
        this.width = imageData.length;
        this.height = imageData[0].length;
        this.numberOfSlices = imageData[0][0].length;
        this.numberOfVolumes = imageData[0][0][0].length;
    }

    private void getMultipleArray()
    {
        int x, y, z, t;
        int i;

        double[][] multipleArray = new double[numberOfVolumes][numberOfVoxels];

        for(i=0; i<numberOfVoxels; i++)
        {
            x = sampleVoxels[i].getX();
            y = sampleVoxels[i].getY();
            z = sampleVoxels[i].getZ();

            for(t=0; t<numberOfVolumes; t++)
            {
                multipleArray[t][i] = imageData[x][y][z][t];
                //System.out.println("mA["+t+"]["+i+"]="+multipleArray[t][i]);
            }
        }

        multipleArrayMatrix = new Matrix(multipleArray);
    }

    private void getMeansMatrix()
    {
        double[] meansArray = new double[numberOfVolumes];
        int i;

        for(i=0; i<numberOfVolumes; i++)
        {
            meansArray[i] = new OneDArrayAnalyzer
                                    (multipleArrayMatrix.getRow(i)).getMean();
            //System.out.println("meanV["+i+"]="+meansArray[i]);
            //System.out.println("var["+i+"]="
            //  +new OneDArrayAnalyzer
            //      (multipleArrayMatrix.getRow(i)).getVariance());
        }

        meansMatrix = new Matrix(meansArray, false);
    }

    private void getCovarianceMatrix()
    {
        covarianceMatrix = new MultipleArrayAnalyzer
                                (multipleArrayMatrix).getCovarianceMatrix();
        //covarianceMatrix.showDiagonalArray();
    }

    private void getClusterDistribution()
    {
        clusterDistribution =
            new MultiGaussianDistribution(meansMatrix, covarianceMatrix);
    }

    private double[][][] extractInstantVolume(int instant)
    {
```

```
        int x, y, z;
        double[][][] instantVolume = new double[width][height][numberOfSlices];

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    instantVolume[x][y][z] = imageData[x][y][z][instant];
                }
            }
        }

        return instantVolume;
    }

    /*
    private Voxel[] getOuterRimVoxels()
    {
        Voxel[] outerRimVoxels = new Voxel[4*width
                                           + 4*(height-2)
                                           + 4*(numberOfSlices-2)];
        int i;

        for(i=0; i<width; i++)
        {
            outerRimVoxels[i] = new Voxel(i,
                                          0,
                                          0);
            outerRimVoxels[i+width] = new Voxel(i,
                                                height-1,
                                                0);
            outerRimVoxels[i+2*width] = new Voxel(i,
                                                  0,
                                                  numberOfSlices-1);
            outerRimVoxels[i+3*width] = new Voxel(i,
                                                  height-1,
                                                  numberOfSlices-1);
        }
        for(i=0; i<height-2; i++)
        {
            outerRimVoxels[i+4*width] = new Voxel(0,
                                                  i+1,
                                                  0);
            outerRimVoxels[i+4*width+(height-2)] = new Voxel(width-1,
                                                             i+1,
                                                             0);
            outerRimVoxels[i+4*width+2*(height-2)] = new Voxel(0,
                                                               i+1,
                                                               numberOfSlices-1);
            outerRimVoxels[i+4*width+3*(height-2)] = new Voxel(width-1,
                                                               i+1,
                                                               numberOfSlices-1);
        }
        for(i=0; i<numberOfSlices-2; i++)
        {
            outerRimVoxels[i+4*width+4*(height-2)] = new Voxel(0,
                                                               0,
                                                               i+1);
            outerRimVoxels[i+4*width+4*(height-2)+(numberOfSlices-2)] =
                                        new Voxel(width-1, 0, i+1);
            outerRimVoxels[i+4*width+4*(height-2)+2*(numberOfSlices-2)] =
                                        new Voxel(0, height-1, i+1);
            outerRimVoxels[i+4*width+4*(height-2)+3*(numberOfSlices-2)] =
                                        new Voxel(width-1, height-1, i+1);
        }

        return outerRimVoxels;
    }
    */
```

```
    /******* SHOW ARRAY ******

    private void showArray(double[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */
}
```

```
package functional.segment;

//
//  ClusterSegmenter.java
//
//
//  Created by José Gabriel Lira Gomes on 2/13/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import matematica.algebra.LargeNumber;
import matematica.algebra.Matrix;
import estatistica.distributions.MultiGaussianDistribution;
import fMRIWindows.MainWindow;
import functional.data.Cluster;
import functional.data.Mask;
import functional.data.Voxel;

public class ClusterSegmenter
{
    private MultiGaussianDistribution[] clusterDistributions;

    private double[][][][] imageData;
    private Cluster[] clusters;
    private MainWindow parentWindow;

    private int numberOfTissues;

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfVolumes;

    double[] numberOfClusterVoxels;
    double totalNumberOfVoxels;
    double[] clusterProbs;


    public ClusterSegmenter(double[][][][] imageData,
                            Cluster[] clusters,
                            MainWindow parentWindow)
    {
        this.imageData = imageData;
        this.clusters = clusters;
        this.parentWindow = parentWindow;
        this.numberOfTissues = clusters.length;

        getImageDims();

        getClusterDistributions();

        getNumberOfClusterVoxels();

        getTotalNumberOfVoxels();

        getClusterProbs();
        System.out.println("clusterProbs");
        showArray(clusterProbs);
        System.out.println();
    }

    public double[][][] getClusterMembershipProbs(int clusterIndex)
    {
        int x, y, z, t;
        int i;

        double[] valueArray = new double[numberOfVolumes];
        Matrix value;

        LargeNumber clusterProbDensity;
        LargeNumber totalProbDensity;
```

```
long startTime;
long stopTime;
long elapsedTime;

LargeNumber clusterIProbDens;

double[][][] clusterMembershipProbs =
        new double[width][height][numberOfSlices];

boolean[][][] dataMaskArray;


System.out.println("Inicio da segmentacao");
startTime = System.currentTimeMillis();

for(z=0; z< numberOfSlices; z++)
{
    for(y=0; y< height; y++)
    {
        for(x=0; x< width; x++)
        {
            for(t=0; t< numberOfVolumes; t++)
            {
                valueArray[t] = imageData[x][y][z][t];
            }
            value = new Matrix(valueArray, false);

            clusterProbDensity =
                clusterDistributions[clusterIndex]
                .getHighPrecProbDensity(value);

            totalProbDensity =
                (clusterDistributions[0].getHighPrecProbDensity(value))
                .multiply(clusterProbs[0]);

            //System.out.println(
            //  "c: " + clusterProbDensity.getString());
            //System.out.println(
            //  "t: " + totalProbDensity.getString());

            for(i=1; i<numberOfTissues; i++)
            {
                clusterIProbDens =
                    clusterDistributions[i]
                    .getHighPrecProbDensity(value);

                totalProbDensity = totalProbDensity.add(
                        clusterIProbDens.multiply(clusterProbs[i]));
            }

            clusterMembershipProbs[x][y][z] =
                (clusterProbDensity.divide(totalProbDensity))
                .getDouble()*clusterProbs[clusterIndex];

            if(clusterMembershipProbs[x][y][z] > 1.0)
            {
                clusterMembershipProbs[x][y][z] = 1.0;
            }
        }
    }
}

//System.out.println(new ThreeDArrayAnalyzer(
//          clusterMembershipProbs).getMaximum());

stopTime = System.currentTimeMillis();
System.out.println("Fim da segmentacao");
elapsedTime = stopTime - startTime;
System.out.println("Tempo de segmentacao:");
System.out.println(elapsedTime + " ms");
```

```
        System.out.println("");

        if(clusterIndex == 0)
        {
            System.out.println("Inicio da actualizacao da mascara");

            dataMaskArray = new boolean[width][height][numberOfSlices];

            for(z=0; z< numberOfSlices; z++)
            {
                for(y=0; y< height; y++)
                {
                    for(x=0; x< width; x++)
                    {
                        if(clusterMembershipProbs[x][y][z] > 0.5)
                        {
                            dataMaskArray[x][y][z] = false;
                        }
                        else
                        {
                            dataMaskArray[x][y][z] = true;
                        }
                    }
                }
            }

            parentWindow.setDataMask(new Mask(dataMaskArray));

            stopTime = System.currentTimeMillis();
            System.out.println("Fim da actualizacao da mascara");
            elapsedTime = stopTime - startTime;
            System.out.println("Tempo de actualizacao da mascara:");
            System.out.println(elapsedTime + " ms");
            System.out.println("");
        }

        return clusterMembershipProbs;
    }

    public double getClusterProb(int clusterIndex)
    {
        return numberOfClusterVoxels[clusterIndex]/totalNumberOfVoxels;
    }

    public void getClusterProbs()
    {
        int i;
        clusterProbs = new double[numberOfTissues];

        for(i=0; i<numberOfTissues; i++)
        {
            clusterProbs[i] = getClusterProb(i);
        }
    }

    public double[][][] getConnectedClusterProbs(int clusterIndex,
                                                 double maxPercentVariation)
    {
        int x, y, z, t;
        double[][][] clusterMenbProbs;
        Voxel[] neighbourVoxels;
        int numberOfNeighbours;
        int i;
        int nx, ny, nz;
        boolean validVoxel;

        clusterMenbProbs = getClusterMembershipProbs(clusterIndex);

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
```

```
        {
            for(x=0; x< width; x++)
            {
                neighbourVoxels = getNeighbourVoxels(x, y, z);
                numberOfNeighbours = neighbourVoxels.length;
                validVoxel = false;
                for(t=0; t< numberOfVolumes; t++)
                {
                    for(i=0; i<numberOfNeighbours; i++)
                    {
                        nx = neighbourVoxels[i].getX();
                        ny = neighbourVoxels[i].getY();
                        nz = neighbourVoxels[i].getZ();
                        if(Math.abs(imageData[x][y][z][t]
                                -imageData[nx][ny][nz][t])
                                /Math.abs(imageData[x][y][z][t])
                                < 1.0-maxPercentVariation)
                        {
                            validVoxel = true;
                        }
                    }
                }
                if(!validVoxel)
                {
                    clusterMenbProbs[x][y][z] = 0;
                }
            }
        }
    }

    return clusterMenbProbs;
}

public Mask getClusterMask(int clusterIndex, double classThreshold)
{
    double[][][] clusterMenbProbs;
    int x, y, z;
    boolean[][][] clusterMask =
        new boolean[width][height][numberOfSlices];

    clusterMenbProbs = getClusterMembershipProbs(clusterIndex);

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if(clusterMenbProbs[x][y][z] > classThreshold)
                {
                    clusterMask[x][y][z] = true;
                }
                else
                {
                    clusterMask[x][y][z] = false;
                }
            }
        }
    }

    return new Mask(clusterMask);
}

public Voxel[] getClusterVoxels(int clusterIndex, double classThreshold)
{
    double[][][] clusterMenbProbs;
    int x, y, z;
    int counter = 0;
    Voxel[] clusterVoxels;

    clusterMenbProbs = getClusterMembershipProbs(clusterIndex);
```

```
        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    if(clusterMenbProbs[x][y][z] > classThreshold)
                    {
                        counter = counter + 1;
                    }
                }
            }
        }

        clusterVoxels = new Voxel[counter];
        counter = 0;

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    if(clusterMenbProbs[x][y][z] > classThreshold)
                    {
                        clusterVoxels[counter] = new Voxel(x, y, z);
                        counter = counter + 1;
                    }
                }
            }
        }

        return clusterVoxels;
    }

    public Mask getConnectedClusterMask(int clusterIndex,
                                        double maxPercentVariation,
                                        double classThreshold)
    {
        double[][][] clusterMenbProbs;
        int x, y, z;
        boolean[][][] clusterMask =
            new boolean[width][height][numberOfSlices];

        clusterMenbProbs = getConnectedClusterProbs(clusterIndex,
                                                    maxPercentVariation);

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    if(clusterMenbProbs[x][y][z] > classThreshold)
                    {
                        clusterMask[x][y][z] = true;
                    }
                    else
                    {
                        clusterMask[x][y][z] = false;
                    }
                }
            }
        }

        return new Mask(clusterMask);
    }

    public Voxel[] getConnectedClusterVoxels(int clusterIndex,
                                             double maxPercentVariation,
```

```
                                                    double classThreshold)
{
    double[][][] clusterMenbProbs;
    int x, y, z;
    int counter = 0;
    Voxel[] clusterVoxels;

    clusterMenbProbs = getConnectedClusterProbs(clusterIndex,
                                                    maxPercentVariation);

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if(clusterMenbProbs[x][y][z] > classThreshold)
                {
                    counter = counter + 1;
                }
            }
        }
    }

    clusterVoxels = new Voxel[counter];
    counter = 0;

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if(clusterMenbProbs[x][y][z] > classThreshold)
                {
                    clusterVoxels[counter] = new Voxel(x, y, z);
                    counter = counter + 1;
                }
            }
        }
    }

    return clusterVoxels;
}

private void getImageDims()
{
    this.width = imageData.length;
    this.height = imageData[0].length;
    this.numberOfSlices = imageData[0][0].length;
    this.numberOfVolumes = imageData[0][0][0].length;
}

private void getClusterDistributions()
{
    int i;
    clusterDistributions =
        new MultiGaussianDistribution[numberOfTissues];

    for(i=0; i<numberOfTissues; i++)
    {
        clusterDistributions[i] = clusters[i].getDistribution();
    }
}

private void getNumberOfClusterVoxels()
{
    int i;
    numberOfClusterVoxels = new double[numberOfTissues];

    for(i=0; i<numberOfTissues; i++)
```

```
        {
            numberOfClusterVoxels[i] =
                (double)clusters[i].getNumberOfClusterVoxels();
        }
    }

    private void getTotalNumberOfVoxels()
    {
        int i;

        totalNumberOfVoxels = 0.0;
        for(i=0; i<numberOfTissues; i++)
        {
            totalNumberOfVoxels = totalNumberOfVoxels
                                    + numberOfClusterVoxels[i];
        }
    }

    private Voxel[] getNeighbourVoxels(int x, int y, int z)
    {
        int i, j, k;
        boolean valid;
        int counter = 0;
        Voxel[] neighbourVoxels;

        for(k=-1; k< 2; k++)
        {
            for(j=-1; j< 2; j++)
            {
                for(i=-1; i< 2; i++)
                {
                    valid = true;
                    if((x+i < 0)||(x+i > width-1))
                    {
                        valid = false;
                    }
                    if((y+j < 0)||(y+j > height-1))
                    {
                        valid = false;
                    }
                    if((z+k < 0)||(z+k > numberOfSlices-1))
                    {
                        valid = false;
                    }
                    if((i==0)&&(j==0)&&(k==0))
                    {
                        valid = false;
                    }
                    if(valid)
                    {
                        counter = counter + 1;
                    }
                }
            }
        }

        neighbourVoxels = new Voxel[counter];
        counter = 0;

        for(k=-1; k< 2; k++)
        {
            for(j=-1; j< 2; j++)
            {
                for(i=-1; i< 2; i++)
                {
                    valid = true;
                    if((x+i < 0)||(x+i > width-1))
                    {
                        valid = false;
                    }
                    if((y+j < 0)||(y+j > height-1))
```

```
                    {
                        valid = false;
                    }
                    if((z+k < 0)||(z+k > numberOfSlices-1))
                    {
                        valid = false;
                    }
                    if((i==0)&&(j==0)&&(k==0))
                    {
                        valid = false;
                    }
                    if(valid)
                    {
                        neighbourVoxels[counter] =
                                    new Voxel(x+i, y+j, z+k);
                        counter = counter + 1;
                    }
                }
            }
        }

        return neighbourVoxels;
    }

    /*
    private double[][][] extractInstantVolume(int instant)
    {
        int x, y, z;
        double[][][] instantVolume =
                new double[width][height][numberOfSlices];

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    instantVolume[x][y][z] =
                        imageData[x][y][z][instant];
                }
            }
        }

        return instantVolume;
    }
    */

    /******* SHOW ARRAY *******/

    private void showArray(double[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }

}
```

```java
package windowUtils;

//
//  ComboBoxesWindows.java
//
//
//  Created by José Gabriel Lira Gomes on 1/22/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import java.awt.Container;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JPanel;

import dataUtils.ComboBoxItems;
import fMRIWindows.MenuedFrame;

public class ComboBoxesWindow extends MenuedFrame implements ItemListener
{
    private int[] boxValues;

    private String windowTitle;
    private String[] labelsText;
    private ComboBoxItems boxItems;

    private int numberOfComboBoxes;

    private Container contentPane;

    private JPanel comboBoxesPanel;
    private JPanel acceptPanel;

    public JButton acceptButton;

    private JLabel[] label;
    private JComboBox[] comboBox;

    private int windowWidth = 200;
    private int windowHeight;
    private int windowXPosition = 0;
    private int windowYPosition = 21;

    public ComboBoxesWindow(String windowTitle,
                            String[] labelsText,
                            ComboBoxItems boxItems)
    {
        this.windowTitle = windowTitle;
        this.labelsText = labelsText;
        this.boxItems = boxItems;

        this.numberOfComboBoxes = labelsText.length;
        boxValues = new int[numberOfComboBoxes];

        createGUI();
    }

    public int[] getSelectedIndexes()
    {
        return boxValues;
    }

    public int getSelectedIndex(int comboBoxIndex)
```

```
{
    return boxValues[comboBoxIndex];
}

public void actionPerformed(ActionEvent event)
{

}

public void itemStateChanged(ItemEvent event)
{
    JComboBox comboBoxSource;
    int i, index;

    if(event.getStateChange() == ItemEvent.SELECTED)
    {
        // Which combo box
        comboBoxSource = (JComboBox)event.getSource();
        // What item in that box
        index = comboBoxSource.getSelectedIndex();

        for(i=0; i<numberOfComboBoxes; i++)
        {
            if(comboBoxSource == comboBox[i])
            {
                boxValues[i] = index;
            }
        }
    }
}

private void createGUI()
{
    int i;
    String[] items;

    windowHeight = 20 + (numberOfComboBoxes - 1)*53 + 45 + 80;

    contentPane = getContentPane();
    contentPane.setLayout(new GridBagLayout());
    GridBagConstraints c = new GridBagConstraints();

    comboBoxesPanel =
            new JPanel(new GridLayout(2*numberOfComboBoxes, 1));

    label = new JLabel[numberOfComboBoxes];
    comboBox = new JComboBox[numberOfComboBoxes];

    for(i=0; i<numberOfComboBoxes; i++)
    {
        items = boxItems.getItems(i);
        label[i] = new JLabel(labelsText[i]);
        label[i].setHorizontalAlignment(JLabel.CENTER);
        comboBox[i] = new JComboBox(items);
        comboBox[i].addActionListener(this);
        comboBox[i].addItemListener(this);
        comboBoxesPanel.add(label[i]);
        comboBoxesPanel.add(comboBox[i]);
    }

    acceptPanel = new JPanel();
    acceptButton = new JButton("Aceitar");
    acceptButton.addActionListener(this);
    acceptPanel.add(acceptButton);

    c.fill = GridBagConstraints.CENTER;

    c.gridx = 0;
    c.gridy = 0;
    c.weightx = 1.00;
    c.weighty = (20 + ((double)numberOfComboBoxes - 1)*53 + 45 + 10)
```

```
                    /(20 + ((double)numberOfComboBoxes - 1)*53 + 45 + 80);

            contentPane.add(comboBoxesPanel, c);

            c.gridx = 0;
            c.gridy = 1;
            c.weightx = 1.00;
            c.weighty = 1
                    - (20 + ((double)numberOfComboBoxes - 1)*53 + 45 + 10)
                    /(20 + ((double)numberOfComboBoxes - 1)*53 + 45 + 80);

            contentPane.add(acceptPanel, c);

            setSize(windowWidth, windowHeight);
            setLocation(windowXPosition, windowYPosition);
            setTitle(windowTitle);
            setVisible(true);
        }
    }
```

```java
package dataUtils;

//
//  ComboBoxItems.java
//
//
//  Created by José Gabriel Lira Gomes on 1/24/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class ComboBoxItems
{
    private String[][] itemsArray;

    private int numberOfGroups;
    private int numberOfItems;

    public ComboBoxItems()
    {
        this.numberOfGroups = 1;
        this.numberOfItems = 1;

        this.itemsArray = startItemsArray();
    }

    public void setItems(int group, String[] items)
    {
        if((items.length > numberOfItems)||(group >= numberOfGroups))
        {
            this.itemsArray = extendItemsArray(group, items);
        }
        updateItemsArray(group, items);
    }

    public void setSequentialItems(int group, int number)
    {
        String[] items =  new String[number];
        int i;

        for(i=0; i<number; i++)
        {
            items[i] = Integer.toString(i);
        }

        if((number > numberOfItems)||(group >= numberOfGroups))
        {
            this.itemsArray = extendItemsArray(group, items);
        }
        updateItemsArray(group, items);
    }

    public String[] getItems(int group)
    {
        String[] items;
        int i;
        int counter = 0;

        for(i=0; i<numberOfItems; i++)
        {
            if(itemsArray[group][i] != null)
            {
                counter = counter + 1;
            }
        }

        items = new String[counter];

        for(i=0; i<counter; i++)
        {
            items[i] = itemsArray[group][i];
        }
```

```
        return items;
    }

    private String[][] startItemsArray()
    {
        String[][] textArray = new String[numberOfGroups][numberOfItems];

        textArray[0][0] = null;

        return textArray;
    }

    private String[][] extendItemsArray(int group, String[] items)
    {
        int oldNumberOfGroups = itemsArray.length;
        int oldNumberOfItems = itemsArray[0].length;

        if(group >= numberOfGroups)
        {
            this.numberOfGroups = group + 1;
        }

        if(items.length > numberOfItems)
        {
            this.numberOfItems = items.length;
        }

        String[][] textArray = new String[numberOfGroups][numberOfItems];
        int i, j;


        for(i=0; i<numberOfGroups; i++)
        {
            for(j=0; j<numberOfItems; j++)
            {
                if((i<oldNumberOfGroups)&&(j<oldNumberOfItems))
                {
                    textArray[i][j] = itemsArray[i][j];
                }
                else
                {
                    textArray[i][j] = null;
                }
            }
        }

        return textArray;
    }

    private void updateItemsArray(int group, String[] items)
    {
        int i;

        for(i=0; i<items.length; i++)
        {
            itemsArray[group][i] = items[i];
        }
    }
}
```

```java
package matematica.complex;

//
//  Complex.java
//
//
//  Created by José Gabriel Lira Gomes on 3/9/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class Complex
{
    double real;
    double img;

    public Complex()
    {
        this.real = 0.0;
        this.img = 0.0;
    }

    public Complex(double real, double img)
    {
        this.real = real;
        this.img = img;
    }

    public double getReal()
    {
        return real;
    }

    public double getImg()
    {
        return img;
    }

    public double getNorm()
    {
        return Math.sqrt(real*real+img*img);
    }

    public Complex plus(Complex z)
    {
        double re;
        double im;

        re = z.getReal();
        im = z.getImg();

        return new Complex(real+re, img+im);
    }

    public Complex minus(Complex z)
    {
        return plus(z.times(-1.0));
    }

    public Complex times(double number)
    {
        return new Complex(real*number, img*number);
    }

    public Complex times(Complex z)
    {
        double re;
        double im;

        re = z.getReal();
        im = z.getImg();
```

```
        return new Complex(real*re-img*im, real*im+img*re);
    }

    public Complex dividedBy(Complex z)
    {
        return times(z.getInverse());
    }

    public Complex getConjugate()
    {
        return new Complex(real, -img);
    }

    public Complex getInverse()
    {
        double norm = getNorm();
        Complex conjugate = getConjugate();

        return conjugate.times(1/(norm*norm));
    }
}
```

```java
package matematica.fourier;

//
//  Convolver.java
//
//
//  Created by Jose Gabriel Lira Gomes on 10/24/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

public class Convolver
{
    //
    //  Data Members
    //

    private double[] convSignal;

    //
    //  Constructors
    //
    //      Convolver(double[] signal, double[] response)
    //

    public Convolver(double[] signal, double[] response)
    {
        int n = signal.length;
        int m = response.length;
        int numberOfDots;

        double[] paddedSignal = new double[n+2*m];
        double[] paddedResp = new double[n+2*m];

        double[] paddedConvSignal;

        FFT signalFFT, responseFFT;
        FFT convInvFFT;

        double[] signalF;
        double[] responseF;

        int k;

        double[] productF;

        paddedSignal = signalPad(signal, m);

        paddedResp = responsePad(response, n);

        signalFFT = new FFT(paddedSignal, true, 1, false);
        responseFFT = new FFT(paddedResp, true, 0, false);

        signalF = signalFFT.getComplex();
        responseF = responseFFT.getComplex();
        //showArray(signalF);
        //showArray(responseF);

        numberOfDots = signalF.length;
        productF = new double[numberOfDots];

        for(k=0; k<numberOfDots; k+=2)
        {
            productF[k] = signalF[k]*responseF[k]
                        - signalF[k+1]*responseF[k+1];
            productF[k+1] = signalF[k]*responseF[k+1]
                        + signalF[k+1]*responseF[k];
        }

        //showArray(productF);

        convInvFFT = new FFT(productF, true);
```

```
        paddedConvSignal = convInvFFT.getReal();
        convSignal = trimConvSignal(paddedConvSignal, n, m);
        //showArray(convSignal);
    }

    //
    //  Public Methods
    //
    //      double[] getConv()
    //

    /******* GET CONVOLUTION *******/

    public double[] getConv()
    {
        return convSignal;
    }

    //
    //  Private Methods
    //
    //      double[] trimConvSignal(double[] initArray,
    //                              int trimmedSignalSize,
    //                              int trimSize)
    //      double[] responsePad(double[] initResponse, int n)
    //      double[] signalPad(double[] initSignal, int m)
    //      double[] rightZeroPad(double[] dArray, int numberOfZeroes)
    //      double[] leftZeroPad(double[] dArray, int numberOfZeroes)
    //      void showArray(double[] array)
    //

    /******* TRIM CONVOLVED SIGNAL *******/

    private double[] trimConvSignal(double[] initArray,
                                    int trimmedSignalSize,
                                    int trimSize)
    {
        int m = trimSize;
        int n = trimmedSignalSize;
        int i;
        int j;
        double[] trimmedSignal = new double[n];

        j=m;
        for(i=0; i<n; i++)
        {
            trimmedSignal[i] = initArray[j];
            j=j+1;
        }

        return trimmedSignal;
    }



    /******* RESPONSE PAD *******/

    private double[] responsePad(double[] initResponse, int n)
    {
        int m = initResponse.length;
        int i;
        double[] paddedArray = new double[n+2*m];

        for(i=0; i<m; i++)
        {
            paddedArray[i] = initResponse[i];
        }
        for(i=m; i<n+2*m; i++)
        {
            paddedArray[i] = 0;
        }
```

```
        return paddedArray;
    }


    /******* SIGNAL PAD *******/

    private double[] signalPad(double[] initSignal, int m)
    {
        int n = initSignal.length;
        int i;
        int j;
        double[] paddedArray = new double[n+2*m];

        j=m-1;
        for(i=m-1; i>=0; i--)
        {
            paddedArray[i] = initSignal[j];
            j=j-1;
        }
        j=0;
        for(i=m-1; i<n+m-1; i++)
        {
            paddedArray[i] = initSignal[j];
            j=j+1;
        }
        j=0;
        for(i=n+m-1; i<n+2*m-1; i++)
        {
            paddedArray[i] = initSignal[j];
            j=j+1;
        }

        //showArray(paddedArray);

        return paddedArray;
    }

    /******* SHOW ARRAY *******/
    /*
    private void showArray(double[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */
}
```

```
package functional.methods;

//
//  CorrAnalysis.java
//
//
//  Created by JGLG on 5/17/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

import estatistica.tests.CorrTest;
import functional.data.DesignMatrix;
import functional.data.Mask;
import functional.data.Timeline;

public class CorrAnalysis
{
    //
    //  Data Members
    //

    private double[][][] corrAnalysisResultData;
    private double[][][] corrCoefResultData;

    private double[][][][] imageData;
    private Mask dataMask;
    private DesignMatrix designMatrix;
    private int[] statesID;
    private int lag;
    private double[] modelArray;

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfInstants;

    //
    //  Constructors
    //
    //      corrAnalysis(double[][][][] imageData,
    //                   Mask dataMask,
    //                   DesignMatrix designMatrix,
    //                   int[] statesID,
    //                   int lag,
    //                   double[] modelArray)
    //


    public CorrAnalysis(double[][][][] imageData,
                        Mask dataMask,
                        DesignMatrix designMatrix,
                        int[] statesID,
                        int lag,
                        double[] modelArray)
    {
        this.imageData = imageData;
        this.dataMask = dataMask;
        this.designMatrix = designMatrix;
        this.statesID = statesID;
        this.lag = lag;
        this.modelArray = modelArray;

        width = imageData.length;
        height = imageData[0].length;
        numberOfSlices = imageData[0][0].length;
        numberOfInstants = imageData[0][0][0].length;

        performCorrelationAnalysis();
    }

    //
```

```
//  Public Methods
//
//      double[][][] getResultData()
//

public double[][][] getResultData()
{
    return corrAnalysisResultData;
}

//
//  Private Methods
//
//      double[] extractTimeArray(int x,
//                                int y,
//                                int z,
//                                int dataMatrix[][][][])
//      void showArray(double[] array)
//      void show3DArray(double[][][] array)
//      void showXProfile(double[][][] array, int x, int y, int z)
//      void showYProfile(double[][][] array, int x, int y, int z)
//      void showZProfile(double[][][] array, int x, int y, int z)
//


private void performCorrelationAnalysis()
{
    corrAnalysisResultData = new double[width][height][numberOfSlices];
    corrCoefResultData = new double[width][height][numberOfSlices];

    Timeline voxelTimeline;

    double[] selectVector;

    int firstStateIndex;
    int secndStateIndex;

    double[] timeArray;

    double[] dataArray;

    int x, y, z;

    CorrTest corrTester;
    double corrProb;
    double corrCoef;

    firstStateIndex = statesID[0];
    secndStateIndex = statesID[1];

    selectVector = designMatrix.getSelectVector(firstStateIndex,
                                                secndStateIndex);

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if(dataMask.getBooleanValue(x, y, z))
                {
                    timeArray = extractTimeArray(x, y, z, imageData);

                    voxelTimeline = new Timeline(timeArray);

                    dataArray = voxelTimeline.getLaggedData(selectVector,
                                                            lag);

                    corrTester = new CorrTest(dataArray, modelArray);
                    corrProb = corrTester.getP();
                    corrCoef = corrTester.getCorrCoef();
```

```
                    corrAnalysisResultData[x][y][z] = corrProb;
                    corrCoefResultData[x][y][z] = corrCoef;
                }
                else
                {
                    corrAnalysisResultData[x][y][z] = 0.0;
                    corrCoefResultData[x][y][z] = 0.0;
                }
            }
        }
    }

    resultMaxValue(corrCoefResultData);
    resultMinValue(corrCoefResultData);

}



/******* EXTRACT TIME ARRAY ******/

private double[] extractTimeArray(int x,
                                  int y,
                                  int z,
                                  double[][][][] dataMatrix)
{
    int t;
    double[] timeArray = new double[numberOfInstants];

    for(t=0; t< numberOfInstants; t++)
    {
        timeArray[t] = dataMatrix[x][y][z][t];
    }
    return timeArray;
}



/******* RESULT MAXIMUM VALUE *******/

private void resultMaxValue(double resultMatrix[][][])
{
    int x, y, z;
    int xx=0;
    int yy=0;
    int zz=0;
    double elementValue;
    double maxValue=resultMatrix[0][0][0];

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                elementValue = resultMatrix[x][y][z];
                if(elementValue > maxValue)
                {
                    maxValue = elementValue;
                    xx = x;
                    yy = y;
                    zz = z;
                }
            }
        }
    }

    System.out.println("Max( p(x,y,z) ) = p("
                       + xx + "," + yy + "," + zz + ") = "
                       + corrAnalysisResultData[xx][yy][zz]);
```

```
    }


    /******* RESULT MINIMUM VALUE *******/

    private void resultMinValue(double resultMatrix[][][])
    {
        int x, y, z;
        int xx=0;
        int yy=0;
        int zz=0;
        double elementValue;
        double minValue=resultMatrix[0][0][0];

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    elementValue = resultMatrix[x][y][z];
                    if(elementValue < minValue)
                    {
                        minValue = elementValue;
                        xx = x;
                        yy = y;
                        zz = z;
                    }
                }
            }
        }

        System.out.println("Min( p(x,y,z) ) = p("
                           + xx + "," + yy + "," + zz + ") = "
                           + corrAnalysisResultData[xx][yy][zz]);

    }


    /******* SHOW ARRAY *******/
    /*
    private void showArray(double[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */
}
```

```java
package estatistica.tests;

//
//  CorrTest.java
//
//
//  Created by JGLG on 5/17/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

import estatistica.descriptive.DualArrayAnalyzer;
import estatistica.distributions.TStudentDistribution;

public class CorrTest
{
    //
    //  Data Members
    //

    private double corrCoeficient;
    private double corrProb;

    private double[] firstArray;
    private double[] secndArray;

    private double sumOfCrossDevProducts;
    private double sumOfFirstDevProducts;
    private double sumOfSecndDevProducts;

    private double TINY=1.0e-20;

    private double t;

    //
    //  Constructors
    //
    //      CorrTest(double sumOfCrossDevProducts,
    //                  double sumOfFirstDevProducts,
    //                  double sumOfSecndDevProducts)
    //      CorrTest(double[] firstArray, double[] secndArray)
    //

    public CorrTest(double sumOfCrossDevProducts,
                    double sumOfFirstDevProducts,
                    double sumOfSecndDevProducts)
    {
        this.sumOfCrossDevProducts = sumOfCrossDevProducts;
        this.sumOfFirstDevProducts = sumOfFirstDevProducts;
        this.sumOfSecndDevProducts = sumOfSecndDevProducts;
    }

    public CorrTest(double[] firstArray, double[] secndArray)
    {
        this.firstArray = firstArray;
        this.secndArray = secndArray;

        getSums();
    }

    //
    //  Public Methods
    //
    //      double getCorrCoef()
    //      double getT()
    //      double getP()
    //

    public double getCorrCoef()
    {
        corrCoeficient = sumOfCrossDevProducts
            /(Math.sqrt(sumOfFirstDevProducts*sumOfSecndDevProducts) + TINY);
```

```
    return corrCoeficient;
}

public double getT()
{
    double n, df;

    n = (double)firstArray.length;
    df = n - 2;

    corrCoeficient = sumOfCrossDevProducts
        /(Math.sqrt(sumOfFirstDevProducts*sumOfSecndDevProducts) + TINY);

    t = corrCoeficient
        *Math.sqrt(df/((1-corrCoeficient+TINY)*(1+corrCoeficient+TINY)));

    return t;
}

public double getP()
{
    double n, df;
    TStudentDistribution tDist;

    n = (double)firstArray.length;
    df = n - 2;

    corrCoeficient = sumOfCrossDevProducts
        /(Math.sqrt(sumOfFirstDevProducts*sumOfSecndDevProducts) + TINY);

    t = corrCoeficient
        *Math.sqrt(df/((1-corrCoeficient+TINY)*(1+corrCoeficient+TINY)));

    tDist = new TStudentDistribution(df);
    corrProb = tDist.getCumProb(t);

    return corrProb;
}


//
//  Private Methods
//
//      void getSums
//      void showArray(double[] array)
//

private void getSums()
{
    DualArrayAnalyzer analyzer;

    analyzer = new DualArrayAnalyzer(firstArray, secndArray);

    sumOfCrossDevProducts = analyzer.getSumOfCrossDevProducts();
    sumOfFirstDevProducts = analyzer.getSumOfFirstDevProducts();
    sumOfSecndDevProducts = analyzer.getSumOfSecndDevProducts();
}

/******* SHOW ARRAY *******/
/*
private void showArray(double[] array)
{
    int size = array.length;
    int i;

    for(i=0; i<size; i++)
    {
        System.out.println(array[i]);
    }
}
```

```
     */
}
```

```java
package dirUtils;

//
//  CreateDirectory.java
//
//
//  Created by José Gabriel Lira Gomes on 1/19/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import java.io.File;

public class CreateDirectory
{
    private String[] dirNames;
    private String[] dirPaths;

    private int numberOfDirs;
    private int numberOfPaths;

    public CreateDirectory(String dirName, String dirPath)
    {
        numberOfDirs = 1;
        numberOfPaths = 1;

        dirNames = new String[numberOfDirs];
        dirPaths = new String[numberOfPaths];

        this.dirNames[0] = dirName;
        this.dirPaths[0] = dirPath;

        createDirs();
    }

    public CreateDirectory(String[] dirNames, String dirPath)
    {
        this.dirNames = dirNames;

        numberOfDirs = dirNames.length;
        numberOfPaths = 1;

        dirPaths = new String[numberOfPaths];
        this.dirPaths[0] = dirPath;

        createDirs();
    }

    public CreateDirectory(String[] dirNames, String[] dirPaths)
    {
        this.dirNames = dirNames;
        this.dirPaths = dirPaths;

        numberOfDirs = dirNames.length;
        numberOfPaths = dirPaths.length;

        if(numberOfDirs == numberOfPaths)
        {
            numberOfDirs = dirNames.length;
            createDirs();
        }
    }

    private void createDirs()
    {
        String[] totalPath = new String[numberOfDirs];
        int i;
        String dirPath;

        for(i=0; i<numberOfDirs; i++)
        {
            if(numberOfPaths == 1)
```

```
                {
                    dirPath = dirPaths[0];
                }
                else
                {
                    dirPath = dirPaths[i];
                }

                if(dirPath.endsWith("/"))
                {
                    totalPath[i] = dirPath + dirNames[i];
                }
                else
                {
                    totalPath[i] = dirPath + "/" + dirNames[i];
                }

                try
                {
                    new File(totalPath[i]).mkdir();
                }
                catch(Exception e)
                {
                    System.out.println("Error: " + e.getMessage());
                }
            }
        }
    }
```

```
package functional.filter;

//
//  DataFilter.java
//
//
//  Created by Jose Gabriel Lira Gomes on 10/16/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import estatistica.descriptive.OneDArrayAnalyzer;
import physics.Filter;

public class DataFilter
{
    //
    //  Data Members
    //

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfInstants;

    private double[][][][] imageData;
    private double deltaT;

    //
    //  Constructors
    //
    //

    public DataFilter(double[][][][] imageData, double deltaT)
    {
        this.imageData = imageData;
        this.deltaT = deltaT;
    }



    public double[][][][] getIdealHPFilteredData(double cutFrequency)
    {
        width = imageData.length;
        height = imageData[0].length;
        numberOfSlices = imageData[0][0].length;
        numberOfInstants = imageData[0][0][0].length;

        double[][][][] filteredImageData =
            new double[width][height][numberOfSlices][numberOfInstants];

        int x, y, z, t;
        double[] dataTimeline;
        double[] variableDataTimeline;
        boolean tooConstant;
        boolean hasConstantSegments;
        int numberOfSegments;
        Filter highPassFilter;
        double[] filteredTimeline;
        double[] nullArray = new double[numberOfInstants];

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    dataTimeline = extractTimeArray(x, y, z, imageData);

                    tooConstant = hasConstantSegment(dataTimeline,
                                    (int)Math.round(1/(cutFrequency*deltaT)));
                    hasConstantSegments = hasConstantSegment(dataTimeline, 3);
```

```
                if(tooConstant)
                {
                    filteredTimeline = nullArray;
                }
                else
                {
                    if(hasConstantSegments)
                    {
                        numberOfSegments =
                            getNumberOfSegments(dataTimeline, 3);
                        if(numberOfSegments > 1)
                        {
                            filteredTimeline = nullArray;
                        }
                        else
                        {
                            filteredTimeline =
                                extractConstant(dataTimeline);
                            //System.out.println
                            //  ("(" + x + "," + y + "," + z + ")");
                            //filteredTimeline = nullArray;
                        }
                    }
                    else
                    {
                        variableDataTimeline =
                            extractConstant(dataTimeline);
                        highPassFilter =
                            new Filter(variableDataTimeline, deltaT);
                        filteredTimeline =
                            highPassFilter.getIdealHPOutput(cutFrequency);
                    }
                }
                for(t=0; t<numberOfInstants; t++)
                {
                    filteredImageData[x][y][z][t] = filteredTimeline[t];
                }
            }
        }
    }

    return filteredImageData;
}


/******* EXTRACT TIME ARRAY ******/

private double[] extractTimeArray(int x,
                                  int y,
                                  int z,
                                  double dataMatrix[][][][])
{
    int t;
    double[] timeArray = new double[numberOfInstants];

    for(t=0; t< numberOfInstants; t++)
    {
        timeArray[t] = dataMatrix[x][y][z][t];
    }
    return timeArray;
}


private double[] extractConstant(double[] dataArray)
{
    int dataSize = dataArray.length;
    int i;
    double mean = new OneDArrayAnalyzer(dataArray).getMean();
```

```
    for(i=0; i<dataSize; i++)
    {
        dataArray[i] = dataArray[i] - mean;
    }

    return dataArray;
}

private boolean hasConstantSegment(double[] dataArray,
                                   int maxNumberOfConstantPoints)
{
    int size = dataArray.length;
    int i;
    int lengthCounter = 0;
    boolean tooConstant = false;

    for(i=1; i<size; i++)
    {
        if(dataArray[i]==dataArray[i-1])
        {
            lengthCounter = lengthCounter + 1;
        }
        else
        {
            if(lengthCounter > maxNumberOfConstantPoints)
            {
                tooConstant = true;
            }
            lengthCounter = 0;
        }
    }

    if(lengthCounter > maxNumberOfConstantPoints)
    {
        tooConstant = true;
    }

    return tooConstant;
}

private int getNumberOfSegments(double[] dataArray,
                                   int maxNumberOfConstantPoints)
{
    int size = dataArray.length;
    int i;
    int lengthCounter = 0;
    int segmentsCounter = 0;

    for(i=1; i<size; i++)
    {
        if(dataArray[i]==dataArray[i-1])
        {
            lengthCounter = lengthCounter + 1;
        }
        else
        {
            if(lengthCounter > maxNumberOfConstantPoints)
            {
                segmentsCounter = segmentsCounter + 1;
            }
            lengthCounter = 0;
        }
    }

    if(lengthCounter > maxNumberOfConstantPoints)
    {
        segmentsCounter = segmentsCounter + 1;
    }

    return segmentsCounter;
}
```

```
    /******* SHOW ARRAY ******

    private void showArray(double[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */
}
```

```
package functional.segment;

import functional.data.Cluster;
import functional.data.Mask;
import functional.data.Voxel;

//
//  DataMaskBuilder.java
//
//
//  Created by José Gabriel Lira Gomes on 2/21/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class DataMaskBuilder
{
    //
    //  Data Members
    //

    private double[][][][] imageData;
    private Cluster backgroundCluster;

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfInstants;

    private Voxel[] sampleBackgroundVoxels;


    public DataMaskBuilder(double[][][][] imageData)
    {
        this.imageData = imageData;

        getImageDims();

        this.sampleBackgroundVoxels = getOuterRimVoxels();

        backgroundCluster = new Cluster(imageData, sampleBackgroundVoxels);
    }

    public Mask getDataMask()
    {
        Mask dataMask;

        dataMask = getBackgroundMask().getComplementary();
        //System.out.println(dataMask.getNumberOfMaskVoxels());

        return dataMask;
    }

    public Voxel[] getDataVoxels()
    {
        Mask dataMask;
        int x, y, z;
        int numberOfDataVoxels;
        Voxel[] dataVoxels;
        int counter = 0;

        dataMask = getDataMask();
        numberOfDataVoxels = width*height*numberOfSlices
                           - getNumberOfBackgroundVoxels();

        dataVoxels = new Voxel[numberOfDataVoxels];

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
```

```
            {
                if(dataMask.getBooleanValue(x, y, z))
                {
                    dataVoxels[counter] = new Voxel(x, y, z);
                    counter = counter + 1;
                }
            }
        }
    }

    return dataVoxels;
}

public int getNumberOfDataVoxels()
{
    int numberOfDataVoxels;

    numberOfDataVoxels = width*height*numberOfSlices
                       - getNumberOfBackgroundVoxels();

    return numberOfDataVoxels;
}

public Mask getBackgroundMask()
{
    return addBackgroundZeroes(backgroundCluster.getSimpleMask(10.0));
}

public Voxel[] getBackgroundVoxels()
{
    Mask backgroundMask;
    int x, y, z;
    int numberOfBackgroundVoxels;
    Voxel[] backgroundVoxels;
    int counter = 0;

    backgroundMask = getBackgroundMask();
    numberOfBackgroundVoxels = getNumberOfBackgroundVoxels();

    backgroundVoxels = new Voxel[numberOfBackgroundVoxels];

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if(backgroundMask.getBooleanValue(x, y, z))
                {
                    backgroundVoxels[counter] = new Voxel(x, y, z);
                    counter = counter + 1;
                }
            }
        }
    }

    return backgroundVoxels;
}

public int getNumberOfBackgroundVoxels()
{
    Mask backgroundMask;
    int x, y, z;
    int counter = 0;

    backgroundMask = getBackgroundMask();

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
```

```
            for(x=0; x< width; x++)
            {
                if(backgroundMask.getBooleanValue(x, y, z))
                {
                    counter = counter + 1;
                }
            }
        }
    }

    return counter;
}

private void getImageDims()
{
    this.width = imageData.length;
    this.height = imageData[0].length;
    this.numberOfSlices = imageData[0][0].length;
    this.numberOfInstants = imageData[0][0][0].length;
}

private Voxel[] getOuterRimVoxels()
{
    Voxel[] outerRimVoxels =
        new Voxel[4*width + 4*(height-2) + 4*(numberOfSlices-2)];
    int i;

    for(i=0; i<width; i++)
    {
        outerRimVoxels[i] = new Voxel(i, 0, 0);
        outerRimVoxels[i+width] = new Voxel(i, height-1, 0);
        outerRimVoxels[i+2*width] = new Voxel(i, 0, numberOfSlices-1);
        outerRimVoxels[i+3*width] =
            new Voxel(i, height-1, numberOfSlices-1);
    }
    for(i=0; i<height-2; i++)
    {
        outerRimVoxels[i+4*width] = new Voxel(0, i+1, 0);
        outerRimVoxels[i+4*width+(height-2)] =
            new Voxel(width-1, i+1, 0);
        outerRimVoxels[i+4*width+2*(height-2)] =
            new Voxel(0, i+1, numberOfSlices-1);
        outerRimVoxels[i+4*width+3*(height-2)] =
            new Voxel(width-1, i+1, numberOfSlices-1);
    }
    for(i=0; i<numberOfSlices-2; i++)
    {
        outerRimVoxels[i+4*width+4*(height-2)] = new Voxel(0, 0, i+1);
        outerRimVoxels[i+4*width+4*(height-2)+(numberOfSlices-2)] =
            new Voxel(width-1, 0, i+1);
        outerRimVoxels[i+4*width+4*(height-2)+2*(numberOfSlices-2)] =
            new Voxel(0, height-1, i+1);
        outerRimVoxels[i+4*width+4*(height-2)+3*(numberOfSlices-2)] =
            new Voxel(width-1, height-1, i+1);
    }

    return outerRimVoxels;
}

private Mask addBackgroundZeroes(Mask dataArray)
{
    int x, y, z;

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if((!dataArray.getBooleanValue(x, y, z))
                        &&(zeroesTimeline(x, y, z)))
```

```
                    {
                        dataArray.setBooleanValue(x, y, z, true);
                    }
                }
            }
        }

        return dataArray;
    }

    private boolean zeroesTimeline(int x, int y, int z)
    {
        int t;
        boolean allZeroes = true;

        for(t=0; t<numberOfInstants; t++)
        {
            if(imageData[x][y][z][t] != 0.0)
            {
                allZeroes = false;
            }
        }

        return allZeroes;
    }
}
```

```
package functional.outliers;

//
//  DataOutliersCleaner.java
//
//
//  Created by Jose Gabriel Lira Gomes on 1/20/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import estatistica.OutlierBrowser;

public class DataOutliersCleaner
{
    //
    //  Data Members
    //

    private double[][][][] cleanVolumesArray;

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfInstants;

    private int numberOfOutliers = 0;

    //
    //  Constructors
    //
    //      DataOutliersCleaner(double[][][][] imageData,
    //                          double outliersThresholdProb,
    //                          double outliersFraction)
    //

    public DataOutliersCleaner(double[][][][] imageData,
                               double outliersThresholdProb,
                               double outliersFraction)
    {
        width = imageData.length;
        height = imageData[0].length;
        numberOfSlices = imageData[0][0].length;
        numberOfInstants = imageData[0][0][0].length;

        double[] dataTimeline = new double[numberOfInstants];
        double[] cleanTimeline = new double[numberOfInstants];

        OutlierBrowser outBrowser;
        boolean outlierIsPresent;
        int[] outliersIndex;

        int x, y, z, t;

        cleanVolumesArray =
            new double[width][height][numberOfSlices][numberOfInstants];

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    dataTimeline = extractTimeArray(x, y, z, imageData);

                    outBrowser = new OutlierBrowser(dataTimeline,
                                                    outliersThresholdProb,
                                                    outliersFraction);
                    outlierIsPresent = outBrowser.outPresent();

                    if(outlierIsPresent)
                    {
```

```
                    numberOfOutliers = numberOfOutliers + 1;

                    outliersIndex = outBrowser.getOutliersIndex();

                    cleanTimeline = correctOutliers(dataTimeline,
                                                    outliersIndex);

                    for(t=0; t< numberOfInstants; t++)
                    {
                        cleanVolumesArray[x][y][z][t] = cleanTimeline[t];
                    }
                }
                else
                {
                    for(t=0; t< numberOfInstants; t++)
                    {
                        cleanVolumesArray[x][y][z][t] = dataTimeline[t];
                    }
                }
            }
        }
    }
}

//
//  Public Methods
//
//      double[][][][] getCleanData()
//

public double[][][][] getCleanData()
{
    return cleanVolumesArray;
}

public int getNumberOfOutliers()
{
    return numberOfOutliers;
}

//
//  Private Methods
//
//      double[] extractTimeArray(int x,
//                                int y,
//                                int z,
//                                double dataMatrix[][][][])
//      double[] correctOutliers(double[] dataArray, int[] outlierIndexPos)
//      int searchNextValid(int ind, int[] outlierInd, int maxInd)
//      int searchPreviousValid(int ind, int[] outlierInd, int minInd)
//

/******* EXTRACT TIME ARRAY ******/

private double[] extractTimeArray(int x,
                                  int y,
                                  int z,
                                  double dataMatrix[][][][])
{
    int t;
    double[] timeArray = new double[numberOfInstants];

    for(t=0; t< numberOfInstants; t++)
    {
        timeArray[t] = dataMatrix[x][y][z][t];
    }
    return timeArray;
}
```

```
/******* CORRECT OUTLIERS *******/

private double[] correctOutliers(double[] dataArray, int[] outlierIndexPos)
{
    int dataLength = dataArray.length;
    int numOfOutliers = outlierIndexPos.length;

    int j;

    //boolean outlier;

    int index = 0;
    int nextValidIndex;
    int previousValidIndex;

    //showArray(outlierIndexPos); // <--------------------

    for(j=0; j<numOfOutliers; j++)
    {
        index = outlierIndexPos[j];

        nextValidIndex = searchNextValid(index,
                                         outlierIndexPos,
                                         dataLength-1);
        previousValidIndex = searchPreviousValid(index,
                                         outlierIndexPos,
                                         0);

        if((nextValidIndex != -1)&(previousValidIndex != -1))
        {
            //System.out.println(index);
            dataArray[index] = (dataArray[nextValidIndex]
                                + dataArray[previousValidIndex])/2;
        }
        else
        {
            if(nextValidIndex != -1)
            {
                dataArray[index] = dataArray[nextValidIndex];
            }
            if(previousValidIndex != -1)
            {
                dataArray[index] = dataArray[previousValidIndex];
            }
        }
    }

    return dataArray;
}



/******* SEARCH NEXT VALID POINT *******/

private int searchNextValid(int ind, int[] outlierInd, int maxInd)
{
    int size = outlierInd.length;
    boolean invalid = true;
    boolean match = false;

    int i;
    int next = ind;

    while(invalid)
    {
        next = next + 1;

        for(i=0; i<size; i++)
        {
            if(next == outlierInd[i])
            {
```

```
                match = true;
            }
        }

        if(!match)
        {
            invalid = false;
        }

        if(next > maxInd)
        {
            next = -1;
            invalid = false;
        }
    }

    return next;
}



/******* SEARCH PREVIOUS VALID POINT *******/

private int searchPreviousValid(int ind, int[] outlierInd, int minInd)
{
    int size = outlierInd.length;
    boolean invalid = true;
    boolean match = false;

    int i;
    int previous = ind;

    while(invalid)
    {
        previous = previous - 1;

        for(i=0; i<size; i++)
        {
            if(previous == outlierInd[i])
            {
                match = true;
            }
        }

        if(!match)
        {
            invalid = false;
        }

        if(previous < minInd)
        {
            previous = -1;
            invalid = false;
        }
    }

    return previous;
}



/******* SHOW ARRAY *******/
/*
private void showArray(int[] array)
{
    int size = array.length;
    int i;

    for(i=0; i<size; i++)
    {
        System.out.println(array[i]);
```

```
            }
        }
        */
    }
```

```
package matematica;

//
//  DataSorter.java
//
//
//  Created by José Gabriel Lira Gomes on 1/23/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

public class DataSorter
{
    //
    //  Data Members
    //

    private double[] sarr;
    private int[] key;

    //
    //  Constructors
    //
    //      dataSorter(double arr[])
    //

    public DataSorter(double arr[])
    {
        sarr = equal(arr);

        int n = sarr.length;
        double[] brr = new double[n];

        int M = 7;
        int NSTACK = 50;

        int i, j, k, l;
        int ir;

        for(i=0; i<n; i++)
        {
            brr[i] = (double)i;
        }

        int[] istack = new int[NSTACK];

        int jstack=-1;

        double a, b, temp;

        l = 0;
        ir = n-1;

        for(;;)
        {
            if(ir-l < M)
            {
                for(j=l+1; j<=ir; j++)
                {
                    a = sarr[j];
                    b = brr[j];

                    for(i=j-1; i>=l; i--)
                    {
                        if(sarr[i] <= a) break;
                        sarr[i+1] = sarr[i];
                        brr[i+1] = brr[i];
                    }
                    sarr[i+1] = a;
                    brr[i+1] = b;
                }
                if(jstack < 0) break;
```

```
        ir = istack[jstack--];
        l = istack[jstack--];
    }
    else
    {
        k = (l+ir) >> 1;

        temp = sarr[k];
        sarr[k] = sarr[l+1];
        sarr[l+1] = temp;

        temp = brr[k];
        brr[k] = brr[l+1];
        brr[l+1] = temp;

        if(sarr[l] > sarr[ir])
        {
            temp = sarr[l];
            sarr[l] = sarr[ir];
            sarr[ir] = temp;

            temp = brr[l];
            brr[l] = brr[ir];
            brr[ir] = temp;
        }
        if(sarr[l+1] > sarr[ir])
        {
            temp = sarr[l+1];
            sarr[l+1] = sarr[ir];
            sarr[ir] = temp;

            temp = brr[l+1];
            brr[l+1] = brr[ir];
            brr[ir] = temp;
        }
        if(sarr[l] > sarr[l+1])
        {
            temp = sarr[l];
            sarr[l] = sarr[l+1];
            sarr[l+1] = temp;

            temp = brr[l];
            brr[l] = brr[l+1];
            brr[l+1] = temp;
        }
        i = l + 1;
        j = ir;
        a = sarr[l+1];
        b = brr[l+1];
        for(;;)
        {
            do i++; while(sarr[i] < a);
            do j--; while(sarr[j] > a);
            if(j < i) break;

            temp = sarr[i];
            sarr[i] = sarr[j];
            sarr[j] = temp;

            temp = brr[i];
            brr[i] = brr[j];
            brr[j] = temp;
        }
        sarr[l+1] = sarr[j];
        sarr[j] = a;
        brr[l+1] = brr[j];
        brr[j] = b;
        jstack = jstack + 2;

        if(jstack> NSTACK)
```

```
                {
                    System.out.println("NSTACK too small.");
                }

                if(ir-i+1 >= j-l)
                {
                    istack[jstack] = ir;
                    istack[jstack-1] = i;
                    ir = j - 1;
                }
                else
                {
                    istack[jstack] = j - 1;
                    istack[jstack-1] = l;
                    l = i;
                }
            }
        }

        key = equalInt(brr);
    }

    //
    //  Public Methods
    //
    //      double[] getSortedArray()
    //      int[] getSwapTable()
    //

    public double[] getSortedArray()
    {
        return sarr;
    }

    public int[] getSwapTable()
    {
        return key;
    }

    //
    //  Private Methods
    //
    //      int[] equalInt(double[] array)
    //      double[] equal(double[] array)
    //

    private int[] equalInt(double[] array)
    {
        int n = array.length;
        int i;
        int temp;

        int[] eqArray = new int[n];

        for(i=0; i<n; i++)
        {
            temp = (int)Math.round(array[i]);
            eqArray[i] = temp;
        }

        return eqArray;
    }


    private double[] equal(double[] array)
    {
        int n = array.length;
        int i;
        double temp;

        double[] eqArray = new double[n];
```

```
        for(i=0; i<n; i++)
        {
            temp = array[i];
            eqArray[i] = temp;
        }

        return eqArray;
    }
}
```

```java
package matematica.difCalculus;

//
//  Derivator.java
//
//
//  Created by Jose Gabriel Lira Gomes on 5/2/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

public class Derivator
{
    //
    //  Data Members
    //

    private double[] firstDer;
    private double[] absFirstDer;
    private double[] secondDer;
    private double[] absSecondDer;

    //
    //  Constructors
    //
    //      derivator(double[] function)
    //

    public Derivator(double[] function)
    {

        int n;
        int i;

        n = function.length;

        firstDer = new double[n-1];
        secondDer = new double[n-2];

        for(i=0; i<n-1; i++)
        {
            firstDer[i] = function[i+1] - function[i];
        }
        absFirstDer = getAbsArray(firstDer);
        for(i=0; i<n-2; i++)
        {
            secondDer[i] = firstDer[i+1] - firstDer[i];
        }
        absSecondDer = getAbsArray(secondDer);
    }

    //
    //  Public Methods
    //
    //      double[] getFirstDer()
    //      double[] getAbsFirstDer()
    //      double[] getSecondDer()
    //      double[] getAbsSecondDer()
    //

    public double[] getFirstDer()
    {
        return firstDer;
    }


    public double[] getAbsFirstDer()
    {
        return absFirstDer;
    }
```

```
    public double[] getSecndDer()
    {
        return secondDer;
    }


    public double[] getAbsSecndDer()
    {
        return absSecondDer;
    }

    //
    //  Private Methods
    //
    //      double[] getAbsArray(double[] array)
    //

    private double[] getAbsArray(double[] array)
    {
        int n;
        int i;
        double[] absArray;

        n = array.length;

        absArray = new double[n];

        for(i=0; i<n; i++)
        {
            absArray[i] = Math.abs(array[i]);
        }

        return absArray;
    }
}
```

```
package functional.data;

//
//  DataOutliersCleaner.java
//
//
//  Created by Jose Gabriel Lira Gomes on 3/22/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import matematica.algebra.Matrix;

public class DesignMatrix extends Matrix
{
    public DesignMatrix(double[][] designMatrixArray)
    {
        super(designMatrixArray);
    }

    public int getNumberOfInstants()
    {
        return getNumberOfRows();
    }

    public int getNumberOfStates()
    {
        return getNumberOfColumns();
    }

    public double[][] getDesignMatrixArray()
    {
        return getMatrixArray();
    }

    public double[] getDesignVector(int firstStateIndex,
                                    int secndStateIndex,
                                    int lag)
    {
        int i;
        double[] firstStateVector;
        double[] secndStateVector;
        double[] designVectorsDif;

        firstStateVector = getStateVector(firstStateIndex);
        secndStateVector = getStateVector(secndStateIndex);

        designVectorsDif = subtractStateVectors(firstStateVector,
                                                secndStateVector);

        int dataSize = designVectorsDif.length - lag;

        double[] modDesignVector = new double[dataSize];

        for(i=0; i<dataSize; i++)
        {
            modDesignVector[i] = (double)designVectorsDif[i];
        }

        return modDesignVector;
    }

    public double[] getSelectVector(int firstStateIndex,
                                    int secndStateIndex)
    {
        double[] firstStateVector;
        double[] secndStateVector;
        double[] designVectorsSum;

        firstStateVector = getStateVector(firstStateIndex);
        secndStateVector = getStateVector(secndStateIndex);
```

```
    designVectorsSum = addStateVectors(firstStateVector,
                                       secndStateVector);

    return designVectorsSum;
}

public double[] getStateVector(int stateIndex)
{
    return getColumn(stateIndex);
}

public double[] getStatesAtInstant(int timeIndex)
{
    return getRow(timeIndex);
}

public int getNumberOfOccurrences(int stateIndex)
{
    int counter = 0;
    double[] stateVector;
    int i;

    stateVector = getColumn(stateIndex);

    for(i=0; i<stateVector.length; i++)
    {
        if(stateVector[i] == 1.0)
        {
            counter = counter + 1;
        }
    }

    return counter;
}

private double[] subtractStateVectors(double[] dVectorOne,
                                      double[] dVectorTwo)
{
    int vectorOneSize = dVectorOne.length;
    int vectorTwoSize = dVectorTwo.length;
    int i;

    double[] dVectorDif = new double[vectorOneSize];

    if(vectorOneSize == vectorTwoSize)
    {
        for(i=0; i<vectorOneSize; i++)
        {
            dVectorDif[i] = dVectorOne[i] - dVectorTwo[i];
        }
    }

    return dVectorDif;
}

private double[] addStateVectors(double[] dVectorOne,
                                 double[] dVectorTwo)
{
    int vectorOneSize = dVectorOne.length;
    int vectorTwoSize = dVectorTwo.length;
    int i;

    double[] dVectorSum = new double[vectorOneSize];

    if(vectorOneSize == vectorTwoSize)
    {
        for(i=0; i<vectorOneSize; i++)
        {
            dVectorSum[i] = dVectorOne[i] + dVectorTwo[i];
        }
    }
```

```
            return dVectorSum;
        }
    }
```

```java
package functional.design;

//
//  DesignMatrixBuilder.java
//
//
//  Created by Jose Gabriel Lira Gomes on 7/24/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import functional.data.DesignMatrix;

public class DesignMatrixBuilder
{
    //
    //  Data Members
    //

    private double[][] designMatrixArray;
    private DesignMatrix desMatrix;

    private int numberOfInstants;
    private int numberOfStates;

    //
    //  Constructors
    //
    //      designMatrixBuilder(int[] timeline, int[] states)
    //

    public DesignMatrixBuilder(double[] timeline, int[] states)
    {
        numberOfInstants = timeline.length;
        numberOfStates = states.length;

        int i, j;

        designMatrixArray = new double[numberOfInstants][numberOfStates+1];

        for(i=0; i<numberOfInstants; i++)
        {
            for(j=0; j<numberOfStates; j++)
            {
                if(timeline[i]==(double)j)
                {
                    designMatrixArray[i][j] = 1.0;
                }
                else
                {
                    designMatrixArray[i][j] = 0.0;
                }
            }
            designMatrixArray[i][numberOfStates] = 1.0;
        }

    }

    //
    //  Public Methods
    //
    //      int[][] getDesignMatrix()
    //

    public double[][] getDesignMatrixArray()
    {
        return designMatrixArray;
    }

    public DesignMatrix getDesignMatrix()
    {
        desMatrix = new DesignMatrix(designMatrixArray);
```

```
        return desMatrix;
    }



    //
    //  Private Methods
    //
    //      void showArray(double[] array)
    //



    /******* SHOW ARRAY *******/
    /*
     private void showArray(double[] array)
     {
         int size = array.length;
         int i;

         for(i=0; i<size; i++)
         {
             System.out.println(array[i]);
         }
     }
     */

}
```

```
package dirUtils;

//
//  Directory.java
//
//
//  Created by José Gabriel Lira Gomes on 1/13/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import java.io.File;

public class Directory
{
    private String dirPath;

    private File folder;
    private String[] filesNames;
    private String[] filesPaths;
    private int numberOfFiles;

    public Directory(String dirPath)
    {
        this.dirPath = dirPath;

        this.folder = new File(dirPath);
        this.filesNames = folder.list();
        this.numberOfFiles = filesNames.length;

        this.filesPaths = addDirPath();
    }

    public int getNumberOfFiles()
    {
        return numberOfFiles;
    }

    public String[] getFilesNames()
    {
        return filesNames;
    }

    public int getNumberOfFilesWithWord(String word)
    {
        int i;
        int counter = 0;
        boolean wordIsPresent;

        for(i=0; i<filesPaths.length; i++)
        {
            wordIsPresent = findWord(filesPaths[i], word);
            if(wordIsPresent)
            {
                counter = counter + 1;
            }
        }

        return counter;
    }

    public int getNumberOfFilesWithExtension(String extension)
    {
        int i;
        int counter = 0;
        boolean extensionIsPresent;

        for(i=0; i<numberOfFiles; i++)
        {
            extensionIsPresent = findExtension(filesPaths[i], extension);
            if(extensionIsPresent)
            {
```

```
            counter = counter + 1;
        }
    }

    return counter;
}

public String[] getFilesPaths()
{
    return filesPaths;
}

public String[] getFilesPathsWithWord(String word)
{
    int size = getNumberOfFilesWithWord(word);
    String[] validFilesPaths = new String[size];
    int i, j;
    boolean wordIsPresent;

    j = 0;

    for(i=0; i<numberOfFiles; i++)
    {
        wordIsPresent = findWord(filesPaths[i], word);
        if(wordIsPresent)
        {
            validFilesPaths[j] = filesPaths[i];
            j = j + 1;
        }
    }

    return validFilesPaths;
}

public String[] getFilesPathsWithExtension(String extension)
{
    int size = getNumberOfFilesWithExtension(extension);
    String[] validFilesPaths = new String[size];
    int i, j;
    boolean extensionIsPresent;

    j = 0;

    for(i=0; i<numberOfFiles; i++)
    {
        extensionIsPresent = findExtension(filesPaths[i], extension);
        if(extensionIsPresent)
        {
            validFilesPaths[j] = filesPaths[i];
            j = j + 1;
        }
    }

    return validFilesPaths;
}

private String[] addDirPath()
{
    int i;
    String[] completeFilesPaths = new String[numberOfFiles];

    for(i=0; i<numberOfFiles; i++)
    {
        completeFilesPaths[i] = dirPath + "/" + filesNames[i];
    }

    return completeFilesPaths;
}

private boolean findWord(String text, String word)
{
```

```
        boolean wordIsPresent = false;

        if(text.indexOf(word) > -1)
        {
            wordIsPresent = true;
        }

        return wordIsPresent;
    }

    private boolean findExtension(String text, String extension)
    {
        boolean extensionIsPresent = false;
        int textSize;
        int extensionSize;
        String textExtension;

        textSize = text.length();
        extensionSize = extension.length();

        textExtension = text.substring(textSize-extensionSize);

        if(textExtension.equals(extension))
        {
            extensionIsPresent = true;
        }

        return extensionIsPresent;
    }

    /******* SHOW ARRAY ******

    private void showArray(String[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */
}
```

```java
package dirUtils;

//
//  DirPathGetter.java
//
//
//  Created by José Gabriel Lira Gomes on 10/23/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.io.File;

import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

public class DirPathGetter
{
    private String path;

    public DirPathGetter()
    {
        path = getDirPath();
    }

    public String getPath()
    {
        return path;
    }

    private String getDirPath()
    {
        JFileChooser chooser;
        File dir;
        int status;
        String dirPath;

        chooser = new JFileChooser();
        chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        status = chooser.showOpenDialog(null);

        if(status == JFileChooser.APPROVE_OPTION)
        {
            dir = chooser.getSelectedFile();
            dirPath = dir.getAbsolutePath();
        }
        else
        {
            JOptionPane.showMessageDialog(null,
                                "Open File dialog canceled");
            dirPath = "-1";
        }

        return dirPath;
    }
}
```

```java
package estatistica.descriptive;

//
//  DualArrayAnalyzer.java
//
//
//  Created by Jose Gabriel Lira Gomes on 7/31/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import matematica.algebra.Matrix;

public class DualArrayAnalyzer
{
    //
    //  Data Members
    //

    private double[] firstArray;
    private double[] secndArray;


    //
    //  Constructors
    //
    //      DualArrayAnalyzer(double[] firstArray, double[] secndArray)
    //

    public DualArrayAnalyzer(double[] firstArray, double[] secndArray)
    {
        this.firstArray = firstArray;
        this.secndArray = secndArray;
    }


    //
    //  Public Methods
    //
    //      double getSumOfCrossDevProducts()
    //      double getSumOfFirstDevProducts()
    //      double getSumOfSecndDevProducts()
    //

    public Matrix getCovarianceMatrix()
    {
        return new Matrix(getCovarianceArray());
    }

    public double[][] getCovarianceArray()
    {
        double firstVariance;
        double secndVariance;
        double covariance;
        double[][] covarianceArray = new double[2][2];

        firstVariance = new OneDArrayAnalyzer(firstArray).getVariance();
        secndVariance = new OneDArrayAnalyzer(secndArray).getVariance();
        covariance = getCovariance();

        covarianceArray[0][0] = firstVariance;
        covarianceArray[1][1] = secndVariance;
        covarianceArray[0][1] = covariance;
        covarianceArray[1][0] = covariance;

        return covarianceArray;
    }

    /*
    public double getCovariance()
    {
        int i;
```

```
        double devOne;
        double devOneSum=0.0;

        double devTwo;
        double devTwoSum=0.0;

        double crossDevSum=0.0;

        int N;
        double dN;
        double meanOne;
        double meanTwo;
        double covariance;

        N = firstArray.length;
        dN = (double)N;

        meanOne = new OneDArrayAnalyzer(firstArray).getMean();
        meanTwo = new OneDArrayAnalyzer(secndArray).getMean();

        for(i=0; i< N; i++)
        {
            devOne = firstArray[i]-meanOne;
            devOneSum = devOneSum + devOne;

            devTwo = firstArray[i]-meanTwo;
            devTwoSum = devTwoSum + devTwo;

            crossDevSum = crossDevSum + devOne*devTwo;
        }

        covariance = (crossDevSum - devOneSum*devTwoSum/dN)/(dN-1);

        return covariance;
    }
    */

    public double getCovariance()
    {
        double dN;

        dN = (double)firstArray.length;

        double sumOfCrossDevProducts = getSumOfCrossDevProducts();

        return sumOfCrossDevProducts/(dN-1);
    }

    public double getSumOfCrossDevProducts()
    {
        int N;
        int i;
        double dataSum=0.0;
        double totalSum;

        N = firstArray.length;

        double firstArrayAverage;
        double secndArrayAverage;

        OneDArrayAnalyzer firstArrayAnalyzer;
        OneDArrayAnalyzer secndArrayAnalyzer;

        firstArrayAnalyzer = new OneDArrayAnalyzer(firstArray);
        firstArrayAverage = firstArrayAnalyzer.getMean();

        secndArrayAnalyzer = new OneDArrayAnalyzer(secndArray);
        secndArrayAverage = secndArrayAnalyzer.getMean();

        for(i=0; i< N; i++)
        {
```

```
            dataSum = dataSum + (firstArray[i]-firstArrayAverage)
                                    *(secndArray[i]-secndArrayAverage);
        }
        totalSum = dataSum;

        return totalSum;
    }


    public double getSumOfFirstDevProducts()
    {
        int N;
        int i;
        double dataSum=0.0;
        double totalSum;

        N = firstArray.length;

        double firstArrayAverage;

        OneDArrayAnalyzer firstArrayAnalyzer;

        firstArrayAnalyzer = new OneDArrayAnalyzer(firstArray);
        firstArrayAverage = firstArrayAnalyzer.getMean();

        for(i=0; i< N; i++)
        {
            dataSum = dataSum + (firstArray[i]-firstArrayAverage)
                                    *(firstArray[i]-firstArrayAverage);
        }
        totalSum = dataSum;

        return totalSum;
    }

    public double getSumOfSecndDevProducts()
    {
        int N;
        int i;
        double dataSum=0.0;
        double totalSum;

        N = secndArray.length;

        double secndArrayAverage;

        OneDArrayAnalyzer secndArrayAnalyzer;

        secndArrayAnalyzer = new OneDArrayAnalyzer(secndArray);
        secndArrayAverage = secndArrayAnalyzer.getMean();

        for(i=0; i< N; i++)
        {
            dataSum = dataSum + (secndArray[i]-secndArrayAverage)
                                    *(secndArray[i]-secndArrayAverage);
        }
        totalSum = dataSum;

        return totalSum;
    }
}
```

```java
package matematica.specialFunctions;

//
//  Erf.java
//
//
//  Created by Jose Gabriel Lira Gomes on 8/1/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

public class Erf
{
    private double z;

    public Erf(double z)
    {
        this.z = z;
    }

    public double getError()
    {
        double t = 1.0 / (1.0 + 0.5 * Math.abs(z));

        // use Horner's method
        double ans = 1 - t * Math.exp( -z*z   -   1.26551223 +
                                             t * ( 1.00002368 +
                                             t * ( 0.37409196 +
                                             t * ( 0.09678418 +
                                             t * (-0.18628806 +
                                             t * ( 0.27886807 +
                                             t * (-1.13520398 +
                                             t * ( 1.48851587 +
                                             t * (-0.82215223 +
                                             t * ( 0.17087277)))))))));
        if (z >= 0) return  ans;
        else        return -ans;
    }
}
```

```java
package fileUtils;

//
//  ExtensionsFileFilter.java
//
//
//  Created by José Gabriel Lira Gomes on 1/29/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import java.io.File;

import javax.swing.filechooser.FileFilter;

public class ExtensionsFileFilter extends FileFilter
{
    private String[] extensions;
    private int numberOfExtensions;

    private static final char DOT = '.';

    public ExtensionsFileFilter(String extension)
    {
        this.numberOfExtensions = 1;
        this.extensions = new String[1];
        this.extensions[0] = extension;
    }

    public ExtensionsFileFilter(String[] extensions)
    {
        this.numberOfExtensions = extensions.length;
        this.extensions = extensions;
    }

    public boolean accept(File f)
    {
        boolean valid = false;

        /*
        if(f.isDirectory())
        {
            valid =  true;
        }
        */

        for(int i=0; i<numberOfExtensions; i++)
        {
            if(extension(f).equalsIgnoreCase(extensions[i]))
            {
                valid =  true;
            }
        }

        return valid;
    }

    public String getDescription()
    {
        String fileFormats = "";
        int i;

        for(i=0; i<numberOfExtensions; i++)
        {
            if(i==numberOfExtensions-1)
            {
                fileFormats = fileFormats + "*." + extensions[i];
            }
            else
            {
                fileFormats = fileFormats + "*." + extensions[i] + ", ";
            }
```

```
        }

        return fileFormats;
    }

    private String extension(File f)
    {
        String fileName = f.getName();
        int dotIndex = fileName.lastIndexOf(DOT);

        if(dotIndex==fileName.length()-4)
        {
            return fileName.substring(dotIndex+1);
        }
        else
        {
            return "";
        }
    }
}
```

```java
package estatistica.distributions;

//
//  FDistribution.java
//
//
//  Created by José Gabriel Lira Gomes on 1/16/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import matematica.specialFunctions.IncBeta;

public class FDistribution
{
    private double dfOne;
    private double dfTwo;

    public FDistribution(double dfOne, double dfTwo)
    {
        this.dfOne = dfOne;
        this.dfTwo = dfTwo;
    }

    public double getCumProb(double value)
    {
        IncBeta incBetaCalcer;
        double prob;

        incBetaCalcer = new IncBeta(0.5*dfTwo,
                                    0.5*dfOne,
                                    dfTwo/(dfTwo+dfOne*value));

        prob = 2.0*(incBetaCalcer.getIncBeta());

        if(prob > 1.0)
        {
            prob = 2.0 - prob;
        }

        return prob;
    }
}
```

```java
package functional.methods;

//
//  FFourierAnalysis.java
//
//
//  Created by Jose Gabriel Lira Gomes on 5/20/05.
//  Copyright 2006 __FDG__. All rights reserved.
//

import physics.SpectrumAnalyzer;
import matematica.fourier.FFT;
import estatistica.descriptive.OneDArrayAnalyzer;
import estatistica.distributions.GaussianDistribution;
import estatistica.fitting.LinFit;
import functional.data.DesignMatrix;
import functional.data.Mask;
import functional.data.Timeline;

public class FFourierAnalysis
{
    //
    //  Data Members
    //

    private double[][][][] imageData;
    private Mask dataMask;
    private DesignMatrix designMatrix;
    private int[] statesID;
    private int lag;
    private int dataWin;
    private double[] modelArray;

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfInstants;

    private int numberOfFrequencies;
    private int numberOfPeaks;

    private FFT modelFFT;
    private SpectrumAnalyzer modelSpecAnalyzer;

    private double[] modelFFTNorm;
    private double[] modelFFTPhase;

    private double[][][][] dataFFTNorm;
    private double[][][][] dataFFTPhase;

    private int[] peaksFreq;

    double[] peakModelValues;
    double[] lowModelValues;

    private double[][][] peaksSlopesVolume;
    private double[][][] peaksSlopesAnalysisResult;

    private double[][][] dataModelComparisonVolume;
    private double[][][] dataModelComparisonAnalysisResult;


    //
    //  Constructors
    //
    //      FFourierAnalysis(double[][][][] imageData,
    //                           Mask dataMask,
    //                           DesignMatrix designMatrix,
    //                           int[] statesID,
    //                           int lag,
    //                           int dataWin,
```

```
//                          double[] modelArray)
//

public FFourierAnalysis(double[][][][] imageData,
                        Mask dataMask,
                        DesignMatrix designMatrix,
                        int[] statesID,
                        int lag,
                        int dataWin,
                        double[] modelArray)
{
    this.imageData = imageData;
    this.dataMask = dataMask;
    this.designMatrix = designMatrix;
    this.statesID = statesID;
    this.lag = lag;
    this.dataWin = dataWin;
    this.modelArray = modelArray;

    width = imageData.length;
    height = imageData[0].length;
    numberOfSlices = imageData[0][0].length;
    numberOfInstants = imageData[0][0][0].length;

    getNumberOfFrequencies();

    getModelFFT();
    getModelSpectrum();
    getModelAnalyzer();
    getNumberOfPeaks();
    getModelPeaksFreqs();
    getModelPeakValues();

    getDataFFT();
}

//
//  Public Methods
//
//      double[][][][] getNorm()
//      double[][][][] getPhase()
//

public double[][][][] getDataFFTNorm()
{
    return dataFFTNorm;
}

public double[][][][] getDataFFTPhase()
{
    return dataFFTPhase;
}

public double[] getModelFFTNorm()
{
    return modelFFTNorm;
}

public double[] getModelFFTPhase()
{
    return modelFFTPhase;
}

public double[][][] getPeaksSlopesAnalysisResult()
{
    getDataPeakSlopes();

    getDataPeakSlopeProbs();

    return peaksSlopesAnalysisResult;
}
```

```
public double[][][] getDataModelComparisonAnalysisResult()
{
    getDataModelComparison();

    getDataModelComparisonProbs();

    return dataModelComparisonAnalysisResult;
}

//
//  Private Methods
//
//      double[] shiftLeft(double[] dataArray, int shiftAmount)
//      double[] extractTimeArray(int x,
//                          int y, int z, int dataMatrix[][][][])
//      double[] intToDoubleArrayConvert(int[] intArray)
//      double[] extractZeroFrequency(double[] dataArray)
//      double getAverage(double dataArray[])
//      int nextMultipleOfTwo(int number)
//      void showArray(double[] array)
//

private void getNumberOfFrequencies()
{
    numberOfFrequencies = nextPowerOfTwo(numberOfInstants);
}

private void getModelFFT()
{
    modelFFT = new FFT(modelArray,true, 1, 1, 1, false);
}

private void getModelSpectrum()
{
    modelFFTNorm = modelFFT.getNorm();
}

private void getModelAnalyzer()
{
    modelSpecAnalyzer = new SpectrumAnalyzer(modelFFTNorm);
}

private void getNumberOfPeaks()
{
    numberOfPeaks = modelSpecAnalyzer.getNumberOfPeaks();
}

private void getModelPeaksFreqs()
{
    peaksFreq = modelSpecAnalyzer.getPeaksFreq();
}

private void getModelPeakValues()
{
    peakModelValues = modelSpecAnalyzer.getPeaks();
}

private void getDataFFT()
{
    dataFFTNorm =
        new double[width][height][numberOfSlices][numberOfFrequencies];
    dataFFTPhase =
        new double[width][height][numberOfSlices][numberOfFrequencies];

    Timeline voxelTimeline;

    double[] selectVector;

    int firstStateIndex;
    int secndStateIndex;
```

```
        firstStateIndex = statesID[0];
        secndStateIndex = statesID[1];

        double[] timeArray;

        double[] dataArray;

        int x, y, z;
        int f;

        double[] zeroedDataArray;

        FFT timelineFFT;
        double[] normOfTimelineFFT;
        double[] phaseOfTimelineFFT;

        selectVector =
            designMatrix.getSelectVector(firstStateIndex, secndStateIndex);

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    if(dataMask.getBooleanValue(x, y, z))
                    {
                        timeArray = extractTimeArray(x, y, z, imageData);

                        voxelTimeline = new Timeline(timeArray);

                        dataArray =
                            voxelTimeline.getLaggedData(selectVector, lag);

                        zeroedDataArray = extractZeroFrequency(dataArray);

                        //(0 - no Win; 1 - Hann; 2 - Hamming)
                        timelineFFT =
                        new FFT(zeroedDataArray, true, dataWin, 1, 1, false);
                        normOfTimelineFFT = timelineFFT.getNorm();
                        phaseOfTimelineFFT = timelineFFT.getPhase();

                        for(f=0; f<numberOfFrequencies; f++)
                        {
                            dataFFTNorm[x][y][z][f] = normOfTimelineFFT[f];
                            dataFFTPhase[x][y][z][f] = phaseOfTimelineFFT[f];
                        }
                    }
                    else
                    {
                        for(f=0; f<numberOfFrequencies; f++)
                        {
                            dataFFTNorm[x][y][z][f] = 0.0;
                            dataFFTPhase[x][y][z][f] = 0.0;
                        }
                    }
                }
            }
        }
    }



    private void getDataPeakSlopes()
    {
        int x, y, z;
        int i;

        double[] peakDataValues = new double[numberOfPeaks];
```

```
        LinFit linFitter;

        double slope;
        double[][] covariance;
        double slopeVar;

        peaksSlopesVolume = new double[width][height][numberOfSlices];

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    if(dataMask.getBooleanValue(x, y, z))
                    {
                        for(i=0; i<numberOfPeaks; i++)
                        {
                            peakDataValues[i] =
                                dataFFTNorm[x][y][z][peaksFreq[i]];
                        }

                        linFitter =
                            new LinFit(peakModelValues, peakDataValues);
                        slope = linFitter.getSlope();
                        covariance = linFitter.getCovArray();
                        slopeVar = covariance[1][1];

                        if(slopeVar != 0)
                        {
                            peaksSlopesVolume[x][y][z] =
                                            slope/Math.sqrt(slopeVar);
                        }
                        else
                        {
                            peaksSlopesVolume[x][y][z] = 0;
                        }
                    }
                    else
                    {
                        peaksSlopesVolume[x][y][z] = 0;
                    }
                }
            }
        }

    private void getDataPeakSlopeProbs()
    {
        int x, y, z;

        double[] serialData;
        int counter = 0;

        double mean;
        double stdDev;

        OneDArrayAnalyzer dataArray;

        GaussianDistribution normalDist;
        double pValue;

        peaksSlopesAnalysisResult =
                        new double[width][height][numberOfSlices];


        serialData = new double[dataMask.getNumberOfMaskVoxels()];

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
```

```
            {
                for(x=0; x< width; x++)
                {
                    if(dataMask.getBooleanValue(x, y, z))
                    {
                        serialData[counter] = peaksSlopesVolume[x][y][z];
                        counter = counter + 1;
                    }
                }
            }
        }

        dataArray = new OneDArrayAnalyzer(serialData);
        mean = dataArray.getMean();
        stdDev = dataArray.getStdDev();

        normalDist = new GaussianDistribution(mean, stdDev);

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    if(dataMask.getBooleanValue(x, y, z))
                    {
                        pValue =
                            normalDist.getCumProb(peaksSlopesVolume[x][y][z]);
                    }
                    else
                    {
                        pValue = 0.0;
                    }
                    peaksSlopesAnalysisResult[x][y][z] = pValue;
                }
            }
        }

        resultMaxValue(peaksSlopesVolume, 0);
        resultMinValue(peaksSlopesVolume, 0);
    }

    private void getDataModelComparison()
    {
        int x, y, z;
        int i;

        double[] scaledModelFFTNorm;
        double[] difference;

        double variance;

        scaledModelFFTNorm = new double[numberOfFrequencies];
        difference = new double[numberOfFrequencies];
        dataModelComparisonVolume = new double[width][height][numberOfSlices];

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    if(dataMask.getBooleanValue(x, y, z))
                    {
                        for(i=0; i<numberOfFrequencies; i++)
                        {
                            scaledModelFFTNorm[i] = modelFFTNorm[i]
                                    *dataFFTNorm[x][y][z][peaksFreq[0]]
                                    /modelFFTNorm[peaksFreq[0]];
                            difference[i] = dataFFTNorm[x][y][z][i]
                                                - scaledModelFFTNorm[i];
```

```
                    }

                    variance =
                        new OneDArrayAnalyzer(difference).getVariance();
                    dataModelComparisonVolume[x][y][z] =
                                    dataFFTNorm[x][y][z][peaksFreq[0]]
                                    /Math.sqrt(variance);
                }
                else
                {
                    dataModelComparisonVolume[x][y][z] = 0.0;
                }
            }
        }
    }
}

private void getDataModelComparisonProbs()
{
    int x, y, z;

    double[] serialData;
    int counter = 0;

    double mean;
    double stdDev;

    OneDArrayAnalyzer dataArray;

    GaussianDistribution normalDist;
    double pValue;


    dataModelComparisonAnalysisResult =
                new double[width][height][numberOfSlices];

    serialData = new double[dataMask.getNumberOfMaskVoxels()];

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if(dataMask.getBooleanValue(x, y, z))
                {
                    serialData[counter] =
                            dataModelComparisonVolume[x][y][z];
                    counter = counter + 1;
                }
            }
        }
    }

    dataArray = new OneDArrayAnalyzer(serialData);
    mean = dataArray.getMean();
    stdDev = dataArray.getStdDev();

    System.out.println("mean = " + mean);
    System.out.println("stdDev = " + stdDev);

    normalDist = new GaussianDistribution(mean, stdDev);

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if(dataMask.getBooleanValue(x, y, z))
                {
```

```
                    pValue =
                        normalDist.getCumProb(
                                dataModelComparisonVolume[x][y][z]);
                }
                else
                {
                    pValue = 0.0;
                }
                dataModelComparisonAnalysisResult[x][y][z] = pValue;
            }
        }
    }

    resultMaxValue(dataModelComparisonVolume, 1);
    resultMinValue(dataModelComparisonVolume, 1);
}



/******* EXTRACT TIME ARRAY ******/

private double[] extractTimeArray(int x,
                                  int y,
                                  int z,
                                  double dataMatrix[][][][])
{
    int t;
    double[] timeArray = new double[numberOfInstants];

    for(t=0; t< numberOfInstants; t++)
    {
        timeArray[t] = dataMatrix[x][y][z][t];
    }
    return timeArray;
}



/******* EXTRACT CONSTANT COMPONENT OF DATA ******/

private double[] extractZeroFrequency(double[] dataArray)
{
    int i;
    int arraySize;

    double[] newData;
    double average;

    arraySize = dataArray.length;

    newData = new double[arraySize];

    average = getAverage(dataArray);

    for(i=0; i<arraySize; i++)
    {
        newData[i] = dataArray[i] - average;
    }

    return newData;
}



private int nextPowerOfTwo(int number)
{
    int powerOfTwo = 2;

    while(powerOfTwo < number)
    {
        powerOfTwo = 2*powerOfTwo;
```

```
    }

    return powerOfTwo;
}


/******* GET AVERAGE ******/

private double getAverage(double dataArray[])
{
    int i;
    int N;

    double dataSum=0.0;
    double average;

    N = dataArray.length;

    for(i=0; i< N; i++)
    {
        dataSum = dataSum + dataArray[i];
    }
    average = dataSum/(double)N;

    return average;
}



/******* RESULT MAXIMUM VALUE *******/

private void resultMaxValue(double resultMatrix[][][], int id)
{
    int x, y, z;
    int xx=0;
    int yy=0;
    int zz=0;
    double elementValue;
    double maxValue=resultMatrix[0][0][0];

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                elementValue = resultMatrix[x][y][z];
                if(elementValue > maxValue)
                {
                    maxValue = elementValue;
                    xx = x;
                    yy = y;
                    zz = z;
                }
            }
        }
    }

    if(id==0)
    {
        System.out.println("Max( p(x,y,z) ) = p("
                    + xx + "," + yy + "," + zz + ") = "
                    + peaksSlopesAnalysisResult[xx][yy][zz]);
    }
    if(id==1)
    {
        System.out.println("Max( p(x,y,z) ) = p("
                    + xx + "," + yy + "," + zz + ") = "
                    + dataModelComparisonAnalysisResult[xx][yy][zz]);
    }
}
```

```
    /******* RESULT MINIMUM VALUE *******/

    private void resultMinValue(double resultMatrix[][][], int id)
    {
        int x, y, z;
        int xx=0;
        int yy=0;
        int zz=0;
        double elementValue;
        double minValue=resultMatrix[0][0][0];

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    elementValue = resultMatrix[x][y][z];
                    if(elementValue < minValue)
                    {
                        minValue = elementValue;
                        xx = x;
                        yy = y;
                        zz = z;
                    }
                }
            }
        }

        if(id==0)
        {
            System.out.println("Min( p(x,y,z) ) = p("
                        + xx + "," + yy + "," + zz + ") = "
                        + peaksSlopesAnalysisResult[xx][yy][zz]);
        }
        if(id==1)
        {
            System.out.println("Min( p(x,y,z) ) = p("
                        + xx + "," + yy + "," + zz + ") = "
                        + dataModelComparisonAnalysisResult[xx][yy][zz]);
        }
    }


    /******* SHOW ARRAY ******

    private void showArray(double[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */
}
```

```java
package matematica.fourier;

//
//  FFT.java
//
//
//  Created by José Gabriel Lira Gomes on 12/23/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

public class FFT
{
    //
    //  Data Members
    //

    private final static double twoPI = 6.28318530717959;

    private double[] complexFFTResult;

    private static int NO_WINDOW = 0;

    private static int ZERO_PAD = 0;
    //private static int SELF_PAD = 1;

    private static int defPaddingFactor = 1;

    private double[] complexData;
    private double[] paddedComplexData;
    private double[] paddedWindowedComplexData;

    private int windowType;
    private int paddingType;
    private int paddingFactor;

    private int dir;

    //
    //  Constructors
    //

    public FFT(double[] complexData, boolean inverse)
    {
        this.windowType = NO_WINDOW;
        this.paddingType = ZERO_PAD;
        this.paddingFactor = defPaddingFactor;

        this.complexData = complexData;
        padData();

        if(inverse)
        {
            dir = -1;
        }
        else
        {
            dir = 1;
        }

        complexFFTResult = fft(paddedComplexData, dir);
    }

    public FFT(double[] data, boolean isReal, boolean inverse)
    {
        this.windowType = NO_WINDOW;
        this.paddingType = ZERO_PAD;
        this.paddingFactor = defPaddingFactor;

        if(isReal)
        {
            this.complexData = convertRealToComplex(data);
```

```
    }
    else
    {
        this.complexData = data;
    }
    padData();

    if(inverse)
    {
        dir = -1;
    }
    else
    {
        dir = 1;
    }

    complexFFTResult = fft(paddedComplexData, dir);
}

public FFT(double[] data,
           boolean isReal,
           int paddingType,
           boolean inverse)
{
    this.windowType = NO_WINDOW;
    this.paddingType = paddingType;
    this.paddingFactor = defPaddingFactor;

    if(isReal)
    {
        this.complexData = convertRealToComplex(data);
    }
    else
    {
        this.complexData = data;
    }
    padData();

    if(inverse)
    {
        dir = -1;
    }
    else
    {
        dir = 1;
    }

    complexFFTResult = fft(paddedComplexData, dir);
}

public FFT(double[] data,
               boolean isReal,
               int windowType,
               int paddingType,
               int paddingFactor,
               boolean inverse)
{
    this.windowType = windowType;
    this.paddingType = paddingType;
    this.paddingFactor = paddingFactor;

    if(isReal)
    {
        this.complexData = convertRealToComplex(data);
    }
    else
    {
        this.complexData = data;
    }
    padData();
```

```
        if(inverse)
        {
            dir = -1;
        }
        else
        {
            dir = 1;
        }

        if(windowType == NO_WINDOW)
        {
            complexFFTResult = fft(paddedComplexData, dir);
        }
        else
        {
            applyWindow();
            complexFFTResult = fft(paddedWindowedComplexData, dir);
        }

    }

    //
    //  Public Methods
    //
    //      double[] getComplex()
    //      double[] getReal()
    //      double[] getImg()
    //      double[] getNorm()
    //      double[] getPhase()
    //

    /******* GET COMPLEX ARRAY *******/

    public double[] getComplex()
    {
        return complexFFTResult;
    }



    /******* GET COMPLEX REAL ARRAY *******/

    public double[] getReal()
    {
        double[] realArray;

        realArray = getReal(complexFFTResult);

        return realArray;
    }



    /******* GET COMPLEX IMAGINARY ARRAY *******/

    public double[] getImg()
    {
        double[] imgArray;

        imgArray = getImg(complexFFTResult);

        return imgArray;
    }



    /******* GET COMPLEX NORM ARRAY *******/

    public double[] getNorm()
    {
        double[] normArray;
```

```
    normArray = getNorm(complexFFTResult);

    return normArray;
}



/******* GET COMPLEX PHASE ARRAY *******/

public double[] getPhase()
{
    double[] phaseArray;

    phaseArray = getPhase(complexFFTResult);

    return phaseArray;
}

//
//  Private Methods
//

private void padData()
{
    int dataArraySize;
    int paddedDataArraySize;

    dataArraySize = complexData.length;

    paddedDataArraySize = paddingFactor*nextPowerOfTwo(dataArraySize);

    paddedComplexData = new double[paddedDataArraySize];


    switch (paddingType)
    {
        case 0:
            {
                paddedComplexData =
                    addZeroPaddingTo(complexData, paddedDataArraySize);
                break;
            }

        case 1:
            {
                paddedComplexData =
                    addSelfPaddingTo(complexData, paddedDataArraySize);
                break;
            }
    }
}

private void applyWindow()
{
    double[] windowedRealData;

    double[] realData;
    double[] imgData;

    realData = getReal(paddedComplexData);
    imgData = getImg(paddedComplexData);

    switch (windowType)
    {
        case 1: windowedRealData = applyHannWindow(realData); break;

        case 2: windowedRealData = applyHammingWindow(realData); break;

        case 3: windowedRealData = applyBlackmanWindow(realData); break;
```

```
            default: windowedRealData = applyHannWindow(realData);
        }

        paddedWindowedComplexData = buildComplexArray(windowedRealData,
                                                      imgData);
    }

    private double[] buildComplexArray(double[] realArray, double[] imgArray)
    {
        int realSize = realArray.length;
        int imgSize = imgArray.length;

        int N = Math.max(realSize, imgSize);

        int i;

        double[] complexArray = new double[2*N];

        for(i=0; i<2*N; i+=2)
        {
            complexArray[i] = realArray[i/2];
        }
        for(i=1; i<2*N; i+=2)
        {
            complexArray[i] = imgArray[i/2];
        }

        return complexArray;
    }

    /******* CONVERT REAL ARRAY TO COMPLEX ARRAY *******/

    private double[] convertRealToComplex(double[] realArray)
    {
        int i, j;
        int arraySize;

        arraySize = realArray.length;
        double[] complexArray = new double[arraySize*2];


        for(i=0; i<2*arraySize; i+=2)
        {
            complexArray[i] = realArray[i/2];
        }
        for(j=1; j<2*arraySize; j+=2)
        {
            complexArray[j] = 0.0;
        }

        return complexArray;
    }


    /******* GET COMPLEX REAL ARRAY *******/

    private double[] getReal(double[] complexArray)
    {
        int i, j;
        int arraySize;

        arraySize = complexArray.length;
        double[] realArray = new double[arraySize/2];

        j = 0;

        for(i=0; i<arraySize; i+=2)
        {
            realArray[j] = complexArray[i];
            j = j + 1;
        }
```

```
    return realArray;
}


/******* GET COMPLEX IMAGINARY ARRAY *******/

private double[] getImg(double[] complexArray)
{
    int i, j;
    int arraySize;

    arraySize = complexArray.length;
    double[] imgArray = new double[arraySize/2];

    j = 0;

    for(i=1; i<arraySize; i+=2)
    {
        imgArray[j] = complexArray[i];
        j = j + 1;
    }
    return imgArray;
}



/******* GET COMPLEX NORM ARRAY *******/

private double[] getNorm(double[] complexArray)
{
    int f;

    double[] realArray;
    double[] imgArray;

    int arraySize;
    int numberOfFrequencies;

    arraySize = complexArray.length;
    numberOfFrequencies = arraySize/2;

    double[] normArray = new double[numberOfFrequencies];

    realArray = getReal(complexArray);
    imgArray = getImg(complexArray);

    for(f=0; f<numberOfFrequencies; f++)
    {
        normArray[f] = Math.sqrt((realArray[f])*(realArray[f])
                        + (imgArray[f])*(imgArray[f]));
    }

    return normArray;
}



/******* GET COMPLEX PHASE ARRAY *******/

private double[] getPhase(double[] complexArray)
{
    int f;

    double[] realArray;
    double[] imgArray;

    int arraySize;
    int numberOfFrequencies;

    arraySize = complexArray.length;
```

```
        numberOfFrequencies = arraySize/2;

        //double temp;

        double[] phaseArray = new double[numberOfFrequencies];

        realArray = getReal(complexArray);
        imgArray = getImg(complexArray);

        for(f=0; f<numberOfFrequencies; f++)
        {
            if((imgArray[f]==0)&(realArray[f]==0))
            {
                phaseArray[f] = 0.0;
            }
            else
            {
                phaseArray[f] = Math.atan(imgArray[f]/realArray[f]);
            }
            /*
            if(phaseArray[f]<0)
            {
                temp = phaseArray[f];
                phaseArray[f] = temp + Math.PI;
            }
            */
        }

        return phaseArray;
    }



    /******* FFT *******/

    private double[] fft(double data[], int isign)
    {
        int i, j, n, m, mmax, istep;
        double wtemp, wr, wpr, wpi, wi, theta;
        double tempr, tempi;

        int nn = data.length/2;

        n = nn*2;
        j = 1;

        for(i=1; i<n; i+=2)
        {
            if(j > i)
            {
                tempr = data[j-1];
                data[j-1] = data[i-1];
                data[i-1] = tempr;

                tempi = data[j];
                data[j] = data[i];
                data[i] = tempi;
            }

            m = nn;

            while((m >= 2)&&(j > m))
            {
                j = j - m;
                m = m/2;
            }

            j = j + m;
        }

        mmax=2;
```

```
    while(n > mmax)
    {
        istep = mmax*2;
        theta = (double)isign*(twoPI/(double)mmax);
        wtemp = Math.sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
        wpi = Math.sin(theta);
        wr = 1.0;
        wi = 0.0;

        for(m=1; m<mmax; m+=2)
        {
            for(i=m; i<=n; i+=istep)
            {
                j = i + mmax;
                tempr = wr*data[j-1] - wi*data[j];
                tempi = wr*data[j] + wi*data[j-1];
                data[j-1] = data[i-1] - tempr;
                data[j] = data[i] - tempi;
                data[i-1]+=tempr;
                data[i]+=tempi;
            }
            wtemp=wr;
            wr=wtemp*wpr-wi*wpi+wr;
            wi=wi*wpr+wtemp*wpi+wi;
        }
        mmax=istep;
    }
    return data;
}



/******* TEST IF NUMBER IS A POWER OF 2 ******

private boolean testIfPowerOfTwo(int number)
{
    float testNumber;
    boolean testValue;

    testNumber = number;

    while(testNumber == Math.round(testNumber))
    {
        testNumber = testNumber/2;
    }

    if(testNumber != 0.5)
    {
        testValue = false;
    }
    else
    {
        testValue = true;
    }

    return testValue;
}
*/


/******* NEXT POWER OF 2 *******/

private int nextPowerOfTwo(int number)
{
    int powerOfTwo = 2;

    while(powerOfTwo < number)
    {
        powerOfTwo = 2*powerOfTwo;
```

```
    }

    return powerOfTwo;
}



/******* ADD ZERO PADDING TO DATA *******/

private double[] addZeroPaddingTo(double[] array,
                                  int paddedNumberOfInstants)
{
    int i;
    int arraySize = array.length;
    double[] paddedArray = new double[paddedNumberOfInstants];

    for(i=0; i<arraySize; i++)
    {
        paddedArray[i] = array[i];
    }

    for(i=arraySize; i<paddedNumberOfInstants; i++)
    {
        paddedArray[i] = 0;
    }

    return paddedArray;
}



/******* ADD SELF PADDING TO DATA *******/

private double[] addSelfPaddingTo(double[] array,
                                  int paddedNumberOfInstants)
{
    int i, j;
    int arraySize = array.length;
    double[] paddedArray = new double[paddedNumberOfInstants];
    int numberOfReps;
    int restOfPoints;

    numberOfReps = (int) Math.floor(paddedNumberOfInstants/arraySize);

    restOfPoints = paddedNumberOfInstants - numberOfReps*arraySize;

    for(j=0; j<numberOfReps; j++)
    {
        for(i=0; i<arraySize; i++)
        {
            paddedArray[i+j*arraySize] = array[i];
        }
    }

    for(i=0; i<restOfPoints; i++)
    {
        paddedArray[i+numberOfReps*arraySize] = array[i];
    }

    return paddedArray;
}



/******* APPLY HANN WINDOW *******/

private double[] applyHannWindow(double[] data)
{
    int i;
    int arraySize;
    double[] newData;
```

```
        arraySize = data.length;


        newData = new double[arraySize];

        for(i=0; i<arraySize; i++)
        {
            newData[i] = (0.5-0.5*Math.cos
                    (twoPI*(double)i/(double)arraySize))*data[i];
        }

        return newData;
    }



    /******* APPLY HAMMING WINDOW *******/

    private double[] applyHammingWindow(double[] data)
    {
        int i;
        int arraySize;
        double[] newData;

        arraySize = data.length;
        newData = new double[arraySize];

        for(i=0; i<arraySize; i++)
        {
            newData[i] = (0.54-0.46*Math.cos
                    (twoPI*(double)i/(double)arraySize))*data[i];
        }

        return newData;
    }

    private double[] applyBlackmanWindow(double[] data)
    {
        int i;
        int arraySize;
        double[] newData;

        arraySize = data.length;
        newData = new double[arraySize];

        for(i=0; i<arraySize; i++)
        {
            newData[i] = (0.42-0.5*Math.cos
                    (twoPI*(double)i/(double)arraySize)
                    +0.08*Math.cos(2.0*twoPI*(double)i
                    /(double)arraySize))*data[i];
        }

        return newData;
    }

    /******* SHOW ARRAY ******

    private void showArray(double[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */
}
```

```java
package fileUtils;

//
//  FileBytesReader.java
//
//
//  Created by José Gabriel Lira Gomes on 11/21/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.io.FileInputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;

public class FileBytesReader
{
    private byte fileBytesData[];

    public FileBytesReader(String filePath)
    {
        FileInputStream fIn;
        FileChannel fChan;
        long fileSize;
        ByteBuffer mBuf;

        byte byteValue;

        try
        {
            /******* abre o ficheiro *******/
            fIn = new FileInputStream(filePath);

            /******* abre canal para o ficheiro *******/
            fChan = fIn.getChannel();

            /******* le tamanho do ficheiro *******/
            fileSize = fChan.size();
            this.fileBytesData = new byte[(int)fileSize];

            /******* cria espaço na memória para os dados *******/
            mBuf = ByteBuffer.allocate((int)fileSize);

            /******* le dados para a memória *******/
            fChan.read(mBuf);

            /******* volta ao início dos dados *******/
            mBuf.rewind();

            /******* guarda dados *******/

            for(int i=0; i< (int)fileSize; i++)
            {
                byteValue = mBuf.get();
                fileBytesData[i] = byteValue;
            }

            /******* fecha o canal *******/
            fChan.close();

            /******* fecha o ficheiro original *******/
            fIn.close();
        }
        catch(IOException exc)
        {
            System.out.println(exc);
            System.exit(1);
        }
    }

    public byte[] getData()
```

```
    {
        return fileBytesData;
    }
}
```

```java
package fileUtils;

//
//  FileBytesWriter.java
//
//
//  Created by Jose Gabriel Lira Gomes on 11/22/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class FileBytesWriter
{
    public FileBytesWriter(byte[] fileBytesData, String filePath)
    {
        FileOutputStream fOut;
        DataOutputStream out;

        try
        {
            /******* abre o ficheiro *******/
            fOut = new FileOutputStream(filePath);
            out = new DataOutputStream(fOut);

            out.write(fileBytesData);

            /******* fecha o ficheiro original *******/
            out.close();
        }
        catch(IOException exc)
        {
            System.out.println(exc);
            System.exit(1);
        }
    }
}
```

```java
package fileUtils;

//
//  FileCopy.java
//
//
//  Created by José Gabriel Lira Gomes on 1/19/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;

public class FileCopy
{
    private File src;
    private File dst;

    public FileCopy(File src, File dst)
    {
        this.src = src;
        this.dst = dst;

        copy();
    }

    private void copy()
    {
        int i;
        int counter=0;
        byte bValue;

        FileInputStream fIn;
        FileOutputStream fOut;
        FileChannel fChan;
        long fSize;
        ByteBuffer mBuf;

        try
        {
            /******* abre o ficheiro final *******/
            fOut = new FileOutputStream(dst);

            /******* abre o ficheiro original *******/
            fIn = new FileInputStream(src);

            /******* abre canal para o ficheiro *******/
            fChan = fIn.getChannel();

            /******* lê tamanho do ficheiro *******/
            fSize = fChan.size();

            /******* cria espaço na memória para os dados *******/
            mBuf = ByteBuffer.allocate((int) fSize);

            /******* lê dados para a memória *******/
            fChan.read(mBuf);

            /******* volta ao início dos dados *******/
            mBuf.rewind();

            /******* imprime dados *******/
            for(i=0; i< fSize; i++)
            {
                counter = counter+1;
                bValue = mBuf.get();
                fOut.write(bValue);
```

```
            }

            /******* fecha o canal *******/
            fChan.close();

            /******* fecha o ficheiro final *******/
            fOut.close();

            /******* fecha o ficheiro original *******/
            fIn.close();
        }
        catch(IOException exc)
        {
            System.out.println(exc);
        }
    }
}
```

```
package fileUtils;

//
//  FilePathGetter.java
//
//
//  Created by JGLG on 6/21/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.io.File;

import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.filechooser.FileFilter;

public class FilePathGetter
{
    private String path;

    public FilePathGetter()
    {
        path = getFilePath();
    }

    public FilePathGetter(FileFilter filter)
    {
        path = getFilePath(filter);
    }

    public String getPath()
    {
        return path;
    }

    private String getFilePath()
    {
        JFileChooser chooser;
        File file;
        int status;
        String filePath;

        chooser = new JFileChooser();
        status = chooser.showOpenDialog(null);

        if(status == JFileChooser.APPROVE_OPTION)
        {
            file = chooser.getSelectedFile();
            filePath = file.getAbsolutePath();
        }
        else
        {
            JOptionPane.showMessageDialog(null,
                                "Open File dialog canceled");
            filePath = "-1";
        }
        return filePath;
    }

    private String getFilePath(FileFilter filter)
    {
        JFileChooser chooser;
        File file;
        int status;
        String filePath;

        chooser = new JFileChooser();
        chooser.setFileFilter(filter);
        status = chooser.showOpenDialog(null);

        if(status == JFileChooser.APPROVE_OPTION)
```

```
        {
            file = chooser.getSelectedFile();
            filePath = file.getAbsolutePath();
        }
        else
        {
            JOptionPane.showMessageDialog(null,
                                "Open File dialog canceled");
            filePath = "-1";
        }
        return filePath;
    }
}
```

```
package physics;

//
//  Filter.java
//
//
//  Created by Jose Gabriel Lira Gomes on 10/12/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import matematica.fourier.FFT;

public class Filter
{
    private int n;
    private double[] paddedInputSignal;

    private double[] transferFunction;

    private int numberOfPoints;
    private double deltaF;

    private double[] signalFFT;

    public Filter(double[] inputSignal, double deltaT)
    {
        n = inputSignal.length;

        this.paddedInputSignal = signalPad(inputSignal, n);

        signalFFT = new FFT(paddedInputSignal, true, 0, 1, 1, false)
                                .getComplex();
        numberOfPoints = signalFFT.length;
        deltaF = 1.0/(numberOfPoints*deltaT);
    }

    public double[] getTransferFunction()
    {
        return transferFunction;
    }

    public double[] getIdealHPOutput(double cutFrequency)
    {
        double[] product;
        double[] outputSignal;

        transferFunction = getIdealHPTransferFunction(cutFrequency);

        product = getComplexProduct(signalFFT, transferFunction);

        outputSignal = new FFT(product, true).getReal();

        return trimConvSignal(outputSignal, n, n);
    }


    public double[] getIdealLPOutput(double cutFrequency)
    {
        double[] product;
        double[] outputSignal;

        transferFunction = getIdealLPTransferFunction(cutFrequency);

        product = getComplexProduct(signalFFT, transferFunction);

        outputSignal = new FFT(product,true).getReal();

        return trimConvSignal(outputSignal, n, n);
    }
```

```
private double[] getIdealHPTransferFunction(double cutFrequency)
{
    int i;
    double frequency;
    double[] tFunction = new double[numberOfPoints];

    for(i=0; i<numberOfPoints; i+=2)
    {
        if(i >= numberOfPoints/2)
        {
            frequency = (numberOfPoints-i-1)*deltaF;
        }
        else
        {
            frequency = (double)i*deltaF;
        }

        if(frequency < cutFrequency)
        {
            tFunction[i] = 0.0;
            tFunction[i+1] = 0.0;
        }
        else
        {
            tFunction[i] = 1.0;
            tFunction[i+1] = 0.0;
        }
    }

    return tFunction;
}

private double[] getIdealLPTransferFunction(double cutFrequency)
{
    int i;
    double frequency;
    double[] tFunction = new double[numberOfPoints];

    for(i=0; i<numberOfPoints; i+=2)
    {
        if(i >= numberOfPoints/2)
        {
            frequency = (numberOfPoints-i-1)*deltaF;
        }
        else
        {
            frequency = (double)i*deltaF;
        }
        if(frequency < cutFrequency)
        {
            tFunction[i] = 1.0;
            tFunction[i+1] = 0.0;
        }
        else
        {
            tFunction[i] = 0.0;
            tFunction[i+1] = 0.0;
        }
    }

    return tFunction;
}



private double[] getComplexProduct(double[] signalF, double[] responseF)
{
    int numberOfDots = signalF.length;
    double[] productF = new double[numberOfDots];
    int k;
```

```
    for(k=0; k<numberOfDots; k+=2)
    {
        productF[k] = signalF[k]*responseF[k]
                    - signalF[k+1]*responseF[k+1];
        productF[k+1] = signalF[k]*responseF[k+1]
                    + signalF[k+1]*responseF[k];
    }

    return productF;
}


/******* SIGNAL PAD *******/

private double[] signalPad(double[] initSignal, int m)
{
    int n = initSignal.length;
    int i;
    int j;
    double[] paddedArray = new double[n+2*m];

    j=n-1;
    for(i=m-1; i>=0; i--)
    {
        paddedArray[i] = initSignal[j];
        if(j==0)
        {
            j=n-1;
        }
        else
        {
            j=j-1;
        }
    }
    j=0;
    for(i=m; i<n+m; i++)
    {
        paddedArray[i] = initSignal[j];
        j=j+1;
    }
    j=0;
    for(i=n+m; i<n+2*m; i++)
    {
        paddedArray[i] = initSignal[j];
        if(j==n-1)
        {
            j=0;
        }
        else
        {
            j=j+1;
        }
    }

    //showArray(paddedArray);

    return paddedArray;
}

/******* TRIM CONVOLVED SIGNAL *******/

private double[] trimConvSignal(double[] initArray,
                                int trimmedSignalSize,
                                int trimStart)
{
    int m = trimStart;
    int n = trimmedSignalSize;
    int i;
    int j;
    double[] trimmedSignal = new double[n];
```

```
            j=m+1;
            for(i=0; i<n; i++)
            {
                trimmedSignal[i] = initArray[j];
                j=j+1;
            }

            return trimmedSignal;
        }


        /******* SHOW ARRAY ******

        private void showArray(double[] array)
        {
            int size = array.length;
            int i;

            for(i=0; i<size; i++)
            {
                System.out.println(array[i]);
            }
        }
        */
}
```

```java
package windowUtils;

//
//  FormatWindow.java
//
//
//  Created by José Gabriel Lira Gomes on 1/17/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class FormatWindow extends RadioButtonsWindow
{
    private static String windowTitle = "Formato";
    private static String title = "Escolha um formato";
    private static String[] possibleFormats = {"Apple", "Unix", "Windows"};

    public FormatWindow()
    {
        super(windowTitle, title, possibleFormats);
    }

    public int getFormat()
    {
        return getSelection();
    }
}
```

```java
package estatistica.descriptive;

//
//  FourDArrayAnalyzer.java
//
//
//  Created by José Gabriel Lira Gomes on 2/8/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import estatistica.distributions.Histogram;

public class FourDArrayAnalyzer
{
    private double[][][][] dataArray;
    private int numberOfLines;
    private int numberOfColumns;
    private int numberOfPlanes;
    private int numberOfVolumes;

    public FourDArrayAnalyzer(double[][][][] dataArray)
    {
        this.dataArray = dataArray;

        this.numberOfLines = dataArray.length;
        this.numberOfColumns = dataArray[0].length;
        this.numberOfPlanes = dataArray[0][0].length;
        this.numberOfVolumes = dataArray[0][0][0].length;
    }

    public double getMean()
    {
        int N;
        int x, y, z, v;
        double dataSum=0;
        double mean;

        N = numberOfColumns*numberOfLines*numberOfPlanes*numberOfVolumes;

        for(v=0; v<numberOfVolumes; v++)
        {
            for(z=0; z<numberOfPlanes; z++)
            {
                for(y=0; y<numberOfLines; y++)
                {
                    for(x=0; x<numberOfColumns; x++)
                    {
                        dataSum = dataSum + dataArray[y][x][z][v];
                    }
                }
            }
        }

        mean = dataSum/(double)N;

        return mean;
    }

    public double getStdDev()
    {
        double stdDev;

        stdDev = Math.sqrt(getVariance());

        return stdDev;
    }

    public double getVariance()
    {
        int x, y, z, v;
        double devSum=0.0;
```

```
        double ep=0.0;
        double s;
        int N;
        double dN;
        double mean;
        double variance;

        N = numberOfColumns*numberOfLines*numberOfPlanes*numberOfVolumes;
        dN = (double)N;

        mean = getMean();

        for(v=0; v<numberOfVolumes; v++)
        {
            for(z=0; z<numberOfPlanes; z++)
            {
                for(y=0; y<numberOfLines; y++)
                {
                    for(x=0; x<numberOfColumns; x++)
                    {
                        s = dataArray[y][x][z][v]-mean;
                        devSum = devSum + s*s;
                        ep = ep + s;
                    }
                }
            }
        }

        variance = (devSum - ep*ep/dN)/(dN-1);

        return variance;
    }

    public double getMaximum()
    {
        int x, y, z, v;
        double element;
        double maximum = dataArray[0][0][0][0];

        for(v=0; v<numberOfVolumes; v++)
        {
            for(z=0; z<numberOfPlanes; z++)
            {
                for(y=0; y<numberOfLines; y++)
                {
                    for(x=0; x<numberOfColumns; x++)
                    {
                        element = dataArray[y][x][z][v];
                        if(element > maximum)
                        {
                            maximum = element;
                        }
                    }
                }
            }
        }

        return maximum;
    }

    public double getMinimum()
    {
        int x, y, z, v;
        double element;
        double minimum = dataArray[0][0][0][0];

        for(v=0; v<numberOfVolumes; v++)
        {
            for(z=0; z<numberOfPlanes; z++)
            {
                for(y=0; y<numberOfLines; y++)
```

```
            {
                for(x=0; x<numberOfColumns; x++)
                {
                    element = dataArray[y][x][z][v];
                    if(element < minimum)
                    {
                        minimum = element;
                    }
                }
            }
        }
    }

    return minimum;
}

public Histogram getHistogram(int binningMethod)
{
    double[][] histogram;

    double[] serialDataArray;
    double N;
    double minimum = getMinimum();
    double maximum = getMaximum();
    int numberOfClasses;
    double classesWidth;

    int i, j;

    serialDataArray = getSerialData();

    N = (double)serialDataArray.length;

    switch (binningMethod)
    {
        case 0: // Sturge
            {
                numberOfClasses =
                        (int)Math.round(Math.log(N)/Math.log(2)) + 1;
                classesWidth = (maximum-minimum)/numberOfClasses;

                break;
            }

        case 1: // Scott
            {
                double s = getStdDev();

                classesWidth = 3.5*s/Math.cbrt(N);
                numberOfClasses =
                        (int)Math.round((maximum-minimum)/classesWidth);

                break;
            }
        default:
            {
                numberOfClasses =
                        (int)Math.round(Math.log(N)/Math.log(2)) + 1;
                classesWidth = (maximum-minimum)/numberOfClasses;

                break;
            }
    }

    histogram = new double[2][numberOfClasses];

    for(j=0; j<numberOfClasses; j++)
    {
        histogram[0][j] = minimum + (j+0.5)*classesWidth;
        histogram[1][j] = 0.0;
    }
```

```
        for(j=0; j<numberOfClasses; j++)
        {
            histogram[1][j] = 0.0;

            for(i=0; i<N; i++)
            {
                if((serialDataArray[i] >= minimum + j*classesWidth)
                    &&(serialDataArray[i] < minimum + (j+1)*classesWidth))
                {
                    histogram[1][j] = histogram[1][j] + 1.0;
                }
            }
        }

        for(j=0; j<numberOfClasses; j++)
        {
            histogram[1][j] = histogram[1][j]/(N*classesWidth);
        }

        return new Histogram(histogram);
    }

    public double[] getSerialData()
    {
        int counter = 0;
        int N = numberOfLines
                *numberOfColumns
                *numberOfPlanes
                *numberOfVolumes;
        int x, y, z, v;
        double[] serialData = new double[N];

        for(v=0; v<numberOfVolumes; v++)
        {
            for(z=0; z<numberOfPlanes; z++)
            {
                for(y=0; y<numberOfLines; y++)
                {
                    for(x=0; x<numberOfColumns; x++)
                    {
                        serialData[counter] = dataArray[y][x][z][v];
                        counter = counter + 1;
                    }
                }
            }
        }

        return serialData;
    }

    public double[][][][] getNormalizedData()
    {
        double[][][][] normalizedData =
        new double
        [numberOfLines][numberOfColumns][numberOfPlanes][numberOfVolumes];
        double maximum;
        double minimum;
        int x, y, z, v;

        maximum = getMaximum();
        minimum = getMinimum();

        for(v=0; v<numberOfVolumes; v++)
        {
            for(z=0; z<numberOfPlanes; z++)
            {
                for(y=0; y<numberOfLines; y++)
                {
                    for(x=0; x<numberOfColumns; x++)
                    {
```

```
                        normalizedData[y][x][z][v] =
                        (dataArray[y][x][z][v]-minimum)/(maximum-minimum);
                    }
                }
            }
        }

        return normalizedData;
    }
}
```

```java
package estatistica.tests;

//
//  FTest.java
//
//
//  Created by José Gabriel Lira Gomes on 1/16/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import estatistica.descriptive.OneDArrayAnalyzer;
import estatistica.distributions.FDistribution;

public class FTest
{
    private double firstVariance;
    private double secndVariance;

    private double N;
    private double M;

    private double dfOne;
    private double dfTwo;


    public FTest(double firstVariance,
                 double secndVariance,
                 double N, double M)
    {
        this.firstVariance = firstVariance;
        this.secndVariance = secndVariance;

        this.N = N;
        this.M = M;
    }

    public FTest(double[] firstArray, double[] secndArray)
    {
        int sizeOfFirstArray = firstArray.length;
        int sizeOfSecndArray = secndArray.length;

        OneDArrayAnalyzer firstAnalyzer;
        OneDArrayAnalyzer secndAnalyzer;

        N = (double)sizeOfFirstArray;
        M = (double)sizeOfSecndArray;

        firstAnalyzer = new OneDArrayAnalyzer(firstArray);
        secndAnalyzer = new OneDArrayAnalyzer(secndArray);

        firstVariance = firstAnalyzer.getVariance();
        secndVariance = secndAnalyzer.getVariance();
    }

    public double getF()
    {
        double f;

        dfOne = N - 1;
        dfTwo = M - 1;

        if(firstVariance == secndVariance)
        {
            f = 1;
        }
        else
        {
            if(firstVariance > secndVariance)
            {
                f = firstVariance/secndVariance;
            }
```

```
            else
            {
                f = secndVariance/firstVariance;
                dfOne = M - 1;
                dfTwo = N - 1;
            }
        }

        return f;
    }

    public double getP()
    {
        double f;
        FDistribution fDist;
        double fTestProb;

        f = getF();

        fDist = new FDistribution(dfOne, dfTwo);
        fTestProb = fDist.getCumProb(f);

        return fTestProb;
    }
}
```

```java
package functional.data;

//
//  FunData.java
//
//
//  Created by José Gabriel Lira Gomes on 1/23/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class FunData
{
    private double[][][][] imagingData;
    private double[][][][] analysisData;

    private boolean real;

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfInstants;

    private int x = 0;
    private int y = 0;
    private int z = 0;

    private String[] states;

    private DesignMatrix desMatrix;

    public FunData(double[][][][] imagingData)
    {
        this.imagingData = imagingData;
        this.analysisData = imagingData;

        this.real = true;

        width = imagingData.length;
        height = imagingData[0].length;
        numberOfSlices = imagingData[0][0].length;
        numberOfInstants = imagingData[0][0][0].length;
    }

    public FunData(double[][][][] imagingData, double[][][][] analysisData)
    {
        this.imagingData = imagingData;
        this.analysisData = analysisData;

        this.real = false;

        width = imagingData.length;
        height = imagingData[0].length;
        numberOfSlices = imagingData[0][0].length;
        numberOfInstants = imagingData[0][0][0].length;
    }

    public void setVoxel(int x, int y, int z)
    {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public void setStates(String[] states)
    {
        this.states = states;
    }

    public void setDesignMatrix(DesignMatrix desMatrix)
    {
        this.desMatrix = desMatrix;
```

```
}


public double[] getTimelineArray()
{
    int t;
    double[] timeArray = new double[numberOfInstants];

    for(t=0; t< numberOfInstants; t++)
    {
        timeArray[t] = imagingData[x][y][z][t];
    }
    return timeArray;
}


public Timeline getTimeline()
{
    return new Timeline(getTimelineArray());
}

public int getWidth()
{
    return width;
}

public int getHeight()
{
    return height;
}

public int getNumberOfSlices()
{
    return numberOfSlices;
}

public int getNumberOfInstants()
{
    return numberOfInstants;
}

public String[] getStates()
{
    return states;
}

public double[] getTwoStateTimelineArray(int[] statesID, int lag)
{
    double[] firstStateVector;
    double[] secndStateVector;
    double[] stateVectorsSum;

    int firstStateIndex;
    int secndStateIndex;

    firstStateIndex = statesID[0];
    secndStateIndex = statesID[1];

    firstStateVector = desMatrix.getStateVector(firstStateIndex);
    secndStateVector = desMatrix.getStateVector(secndStateIndex);

    stateVectorsSum = addArrays(firstStateVector, secndStateVector);

    return getTimeline().getLaggedData(stateVectorsSum, lag);
}


private double[] addArrays(double[] dVectorOne, double[] dVectorTwo)
{
    int vectorOneSize = dVectorOne.length;
    int vectorTwoSize = dVectorTwo.length;
```

```
        int i;

        double[] dVectorSum = new double[vectorOneSize];

        if(vectorOneSize == vectorTwoSize)
        {
            for(i=0; i<vectorOneSize; i++)
            {
                dVectorSum[i] = dVectorOne[i] + dVectorTwo[i];
            }
        }

        return dVectorSum;
    }
}
```

```java
package matematica.specialFunctions;

//
//  gamma.java
//
//
//  Created by Jose Gabriel Lira Gomes on 8/1/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

public class Gamma
{
    private double z;
    private double lnGamma;

    public Gamma(double z)
    {
        this.z = z;

        lnGamma = calcLnGamma();
    }

    public double getGamma()
    {
        return Math.exp(lnGamma);
    }

    public double getLnGamma()
    {
        return lnGamma;
    }

    private double calcLnGamma()
    {
        int j;
        double x, y, tmp, ser, stp;
        double[] cof = new double[6];
        double gammaln;

        cof[0] = 76.18009172947146;
        cof[1] = -86.50532032941677;
        cof[2] = 24.01409824083091;
        cof[3] = -1.231739572450155;
        cof[4] = 0.1208650973866179e-2;
        cof[5] = -0.5395239384953e-5;

        y = x = z;
        tmp = x + 5.5;
        tmp = (x+0.5)*Math.log(tmp) - tmp;
        ser = 1.000000000190015;
        stp = 2.5066282746310005;

        for(j=0; j<6; j++)
        {
            y = y+1;
            ser = ser + cof[j]/y;
        }

        gammaln = tmp + Math.log(stp*ser/x);

        return gammaln;
    }
}
```

```java
package estatistica.distributions;

//
//  GaussianDistribution.java
//
//
//  Created by José Gabriel Lira Gomes on 8/1/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.util.Random;

import matematica.specialFunctions.Erf;

public class GaussianDistribution
{
    //
    //  Data Members
    //

    private double mu;
    private double sigma;


    //
    //  Constructors
    //
    //      GaussianDistribution(double mu, double sigma)
    //

    public GaussianDistribution(double mu, double sigma)
    {
        this.mu = mu;
        this.sigma = sigma;
    }


    //
    //  Public Methods
    //
    //      double getProbDensity(double value)
    //      double getCumProb(double value)
    //

    public double getProbDensity(double value)
    {
        double probDensity;
        double z;

        z = (value-mu)/sigma;

        probDensity = (Math.exp(-z*z/2.0))/(sigma*Math.sqrt(2.0*Math.PI));

        return probDensity;
    }

    public double getCumProb(double value)
    {
        double cumProb;
        double z;

        z = (value-mu)/sigma;

        cumProb = 0.5*(1.0 + new Erf(z/Math.sqrt(2)).getError());

        return cumProb;
    }

    public double getRangeCumProb(double value)
    {
        double z;
```

```
        double rangeCumProb;

        z = (value-mu)/sigma;

        rangeCumProb = new Erf(z/Math.sqrt(2)).getError();

        return rangeCumProb;
    }

    public double getRelRangeCumProb(double z)
    {
        double relRangeCumProb;

        relRangeCumProb = new Erf(z/Math.sqrt(2)).getError();

        return relRangeCumProb;
    }

    public double[] generateData(int numberOfPoints)
    {
        Random r = new Random();
        double[] data = new double[numberOfPoints];
        int i;

        for(i=0; i<numberOfPoints; i++)
        {
            data[i] = r.nextGaussian() + mu;
        }
        for(i=0; i<numberOfPoints; i++)
        {
            data[i] = mu + (data[i]-mu)*sigma;
        }

        return data;
    }
}
```

```java
package matematica.algebra;

//
//  GaussJordanElimination.java
//
//
//  Created by Jose Gabriel Lira Gomes on 10/17/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

public class GaussJordanElimination
{
    //
    //  Data Members
    //

    private double[][] covar;
    private double[][] x;

    //
    //  Constructors
    //
    //      GaussJordanElimination(double[][] a, double[][] b)
    //

    public GaussJordanElimination(double[][] a, double[][] b)
    {
        performGaussJ(a, b);
    }

    public GaussJordanElimination(Matrix a, Matrix b)
    {
        performGaussJ(a.getMatrixArray(), b.getMatrixArray());
    }

    public double[][] getCovarianceArray()
    {
        return covar;
    }

    public Matrix getCovariance()
    {
        return new Matrix(covar);
    }

    public double[][] getSolutionArray()
    {
        return x;
    }

    public Matrix getSolution()
    {
        return new Matrix(x);
    }

    private void performGaussJ(double[][] a, double[][] b)
    {
        int i, j, k, l, ll;
        int iCol = 0;
        int iRow = 0;
        double big, dum, pivinv;
        double swap;

        int n = a.length;
        int m = b[0].length;

        int[] indxc = new int[n];
        int[] indxr = new int[n];
        int[] ipiv = new int[n];

        covar = new double[n][n];
```

```
x = new double[n][m];

for (j=0; j<n; j++)
{
    ipiv[j] = 0;
}
for (i=0; i<n; i++)
{
    big = 0.0;
    for (j=0; j<n; j++)
    {
        if (ipiv[j] != 1);
        {
            for (k=0; k<n; k++)
            {
                if (ipiv[k] == 0)
                {
                    if (Math.abs(a[j][k]) >= big)
                    {
                        big = Math.abs(a[j][k]);
                        iRow = j;
                        iCol = k;
                    }
                }
            }
        }
    }

    ipiv[iCol] = ipiv[iCol] + 1;

    if (iRow != iCol)
    {
        for (l=0; l<n; l++)
        {
            swap = a[iRow][l];
            a[iRow][l] = a[iCol][l];
            a[iCol][l] = swap;
        }
        for (l=0; l<m; l++)
        {
            //System.out.println(iRow);
            //System.out.println(l);

            swap = b[iRow][l];
            b[iRow][l] = b[iCol][l];
            b[iCol][l] = swap;
        }
    }

    indxr[i] = iRow;
    indxc[i] = iCol;

    if (a[iCol][iCol] == 0)
    {
        System.out.println("Singular Matrix");
    }

    pivinv = 1.0/a[iCol][iCol];
    a[iCol][iCol] = 1.0;

    for (l=0; l<n; l++)
    {
        a[iCol][l] = a[iCol][l]*pivinv;
    }
    for (l=0; l<m; l++)
    {
        b[iCol][l] = b[iCol][l]*pivinv;
    }

    for (ll=0; ll<n; ll++)
    {
```

```
                if (ll != iCol)
                {
                    dum = a[ll][iCol];
                    a[ll][iCol] = 0.0;

                    for (l=0; l<n; l++)
                    {
                        a[ll][l] = a[ll][l] - a[iCol][l]*dum;
                    }
                    for (l=0; l<m; l++)
                    {
                        b[ll][l] = b[ll][l] - b[iCol][l]*dum;
                    }
                }
            }
        }
        for (l=n-1; l>=0; l--)
        {
            if (indxr[l] != indxc[l])
            {
                for (k=0; k<n; k++)
                {
                    swap = a[k][indxr[l]];
                    a[k][indxr[l]] = a[k][indxc[l]];
                    a[k][indxc[l]] = swap;
                }
            }
        }


        for(j=0; j<n; j++)
        {
            for(k=0; k<n; k++)
            {
                covar[j][k] = a[j][k];
            }
        }
        for(j=0; j<n; j++)
        {
            for(k=0; k<m; k++)
            {
                x[j][k] = b[j][k];
            }
        }
    }
}
```

```java
package estatistica.fitting;

//
//  GenLinFit.java
//
//
//  Created by José Gabriel Lira Gomes on 10/18/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

import matematica.algebra.GaussJordanElimination;
import matematica.algebra.Matrix;

public class GenLinFit
{
    private double[] coefs;
    private double[][] covariance;
    private double chiSquare;

    private double[] x;
    private double[] y;
    private double[] sigma;
    private double[] initialCoefs;
    private boolean[] validCoefs;

    private int numberOfPoints;
    private int numberOfCoefs;

    private double[][] function;

    private int pointsPerPeriod = 1;

    public GenLinFit(double[] x,
                     double[] y,
                     double[] sigma,
                     double[] initialCoefs,
                     boolean[] validCoefs,
                     int functionID)
    {
        this.x = x;
        this.y = y;
        this.sigma = sigma;
        this.initialCoefs = initialCoefs;
        this.validCoefs = validCoefs;

        this.numberOfPoints = y.length;
        this.numberOfCoefs = initialCoefs.length;

        switch (functionID)
        {
            case 0: this.function = poly(); break;

            case 1: this.function = sin(); break;
        }

        calculateLinFit();
    }

    public GenLinFit(double[] y, Matrix functionMatrix)
    {
        this.y = y;

        this.numberOfPoints = y.length;
        this.numberOfCoefs = functionMatrix.getNumberOfColumns();

        this.x = new double[numberOfPoints];
        this.sigma = new double[numberOfPoints];
        for(int i=0; i<numberOfPoints; i++)
        {
            this.x[i] = i;
            this.sigma[i] = 1.0;
```

```
        }

        this.initialCoefs = new double[numberOfCoefs];
        this.validCoefs = new boolean[numberOfCoefs];
        for(int i=0; i<numberOfCoefs; i++)
        {
            this.initialCoefs[i] = 1.0;
            this.validCoefs[i] = true;
        }

        this.function = functionMatrix.getMatrixArray();

        calculateLinFit();
    }

    public GenLinFit(double[] x,
                     double[] y,
                     int functionID,
                     int numberOfCoefs)
    {
        this.x = x;
        this.y = y;

        this.numberOfPoints = y.length;
        this.numberOfCoefs = numberOfCoefs;

        this.sigma = new double[numberOfPoints];
        for(int i=0; i<numberOfPoints; i++)
        {
            this.sigma[i] = 1.0;
        }

        this.initialCoefs = new double[numberOfCoefs];
        this.validCoefs = new boolean[numberOfCoefs];
        for(int i=0; i<numberOfCoefs; i++)
        {
            this.initialCoefs[i] = 1.0;
            this.validCoefs[i] = true;
        }

        switch (functionID)
        {
            case 0: this.function = poly(); break;

            case 1: this.function = sin(); break;
        }

        calculateLinFit();
    }

    /******* GET COEFICIENTS *******/

    public double[] getCoefs()
    {
        return coefs;
    }



    /******* GET COVARIANCE ARRAY *******/

    public double[][] getCovArray()
    {
        return covariance;
    }

    public Matrix getCovMatrix()
    {
        return new Matrix(covariance);
    }
```

```
/******* GET CHI SQUARE *******/

public double getChiSquare()
{
    return chiSquare;
}



private void calculateLinFit()
{
    int i, j, k, l, m, mfit;
    double ym, wt, sum, sig2i;

    double[][] alpha= new double[numberOfCoefs][numberOfCoefs];
    double[][] beta= new double[numberOfCoefs][1];

    double[][] covar;
    double[][] a;

    GaussJordanElimination solution;

    mfit=0;

    for(j=0; j<numberOfCoefs; j++)
    {
        if(validCoefs[j])
        {
            mfit=mfit+1;
        }
    }
    if(mfit==0)
    {
        System.out.println("No parameters to be fitted");
    }


    /******* Initialize the matrix *******/

    for (j=0; j<mfit; j++)
    {
        for (k=0; k<mfit; k++)
        {
            alpha[j][k]=0.0;
        }
        beta[j][0]=0.0;
    }


    /* Loop over data to accumulate coefficients of the normal equations */

    for (i=0; i<numberOfPoints; i++)
    {
        ym=y[i];
        if(mfit<numberOfCoefs)
        {
            for (j=0; j<numberOfCoefs; j++)
            {
                if(!validCoefs[j])
                {
                    ym = ym - initialCoefs[j]*function[i][j];
                }
            }
        }
        sig2i = 1.0/((sigma[i])*(sigma[i]));

        j=0;
        for (l=0; l< numberOfCoefs; l++)
        {
            if (validCoefs[l])
```

```
            {
                wt = function[i][l]*sig2i;
                k=0;
                for (m=0; m<=l; m++)
                {
                    if (validCoefs[m])
                    {
                        alpha[j][k] = alpha[j][k] + wt*function[i][m];
                        k=k+1;
                    }
                }
                beta[j][0] = beta[j][0] + ym*wt;
                j=j+1;
            }
        }
    }


    /******* Fill in above the diagonal from symmetry *******/

    for (j=1; j<mfit; j++)
    {
        for (k=0; k<j; k++)
        {
            alpha[k][j] = alpha[j][k];
        }
    }


    /******* Matrix solution *******/

    solution = new GaussJordanElimination(alpha, beta);

    covar = solution.getCovarianceArray();
    a = solution.getSolutionArray();

    coefs = asrt(a, initialCoefs, validCoefs);
    covariance = covsrt(covar, validCoefs, mfit);


    /******* Evaluate chi2 of the fit *******/

    chiSquare = 0.0;
    for (i=0; i<numberOfPoints; i++)
    {
        sum = 0.0;
        for (j=0; j<numberOfCoefs; j++)
        {
            sum = sum + coefs[j]*function[i][j];
        }
        chiSquare = chiSquare
                        + ((y[i]-sum)/sigma[i])*((y[i]-sum)/sigma[i]);
    }
}

/******* ASRT *******/

private double[] asrt(double[][] a,
                        double[] initialCoefs,
                        boolean[] validCoefs)
{
    int j, l;
    double[] aFinal = new double[numberOfCoefs];

    j=0;
    for (l=0; l<numberOfCoefs; l++)
    {
        if (validCoefs[l])
        {
            aFinal[l] = a[j][0];
            j=j+1;
```

```
        }
        else
        {
            aFinal[l] = initialCoefs[j];
            j=j+1;
        }
    }
    return aFinal;
}



/******* COVSRT *******/

private double[][] covsrt(double[][] covar,
                          boolean[] validCoefs,
                          int mfit)
{
    int i, j, k;
    double swap;

    int numberOfCoefs = validCoefs.length;

    for (i=mfit; i<numberOfCoefs; i++)
    {
        for(j=0; j<i+1; j++)
        {
            covar[i][j] = covar[j][i] = 0.0;
        }
    }
    k = mfit-1;
    for (j=numberOfCoefs-1; j>=0; j--)
    {
        if (validCoefs[j])
        {
            for (i=0; i<numberOfCoefs; i++)
            {
                swap = covar[i][k];
                covar[i][k] = covar[i][j];
                covar[i][j] = swap;
            }
            for (i=0; i<numberOfCoefs; i++)
            {
                swap = covar[k][i];
                covar[k][i] = covar[j][i];
                covar[j][i] = swap;
            }
            k = k-1;
        }
    }
    return covar;
}


/******* POLYNOMIUM FUNCTION DEFINITION *******/

private double[][] poly()
{
    double[][] yValue = new double[numberOfPoints][numberOfCoefs];

    int i, j;

    for(i=0; i<numberOfPoints; i++)
    {
        yValue[i][0] = 1.0;
        for (j=1; j<numberOfCoefs; j++)
        {
            yValue[i][j] = yValue[i][j-1]*x[i];
        }
    }
```

```
        return yValue;
    }


    /******* SIN SERIES FUNCTION DEFINITION *******/

    private double[][] sin()
    {
        double[][] yValue = new double[numberOfPoints][numberOfCoefs];

        int i, j;

        for(i=0; i<numberOfPoints; i++)
        {
            yValue[i][0] = 1.0;
            for (j=1; j<numberOfCoefs; j++)
            {
                yValue[i][j] = Math.sin(2*Math.PI*x[i]*j/pointsPerPeriod);
            }
        }

        return yValue;
    }


}
```

```java
package functional.methods;

//
//  GlmAnalysis.java
//
//
//  Created by JGLG on 10/24/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

import estatistica.descriptive.OneDArrayAnalyzer;
import estatistica.distributions.GaussianDistribution;
import estatistica.fitting.GenLinFit;
import functional.data.DesignMatrix;
import functional.data.Mask;
import functional.data.Timeline;

public class GlmAnalysis
{
    //
    //  Data Members
    //

    private double[][][] glmAnalysisResult;
    private double[][][] relCoefsVolume;

    private double[][][][] imageData;
    private Mask dataMask;
    private DesignMatrix designMatrix;
    private int[] statesID;
    private int lag;
    //private int inferenceType;

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfInstants;

    private int singCounter = 0;

    //
    //  Constructors
    //
    //      glmAnalysis(double[][][][] imageData,
    //                  Mask dataMask,
    //                  DesignMatrix designMatrix,
    //                  int[] statesID,
    //                  int lag,
    //                  int inferenceType)
    //

    public GlmAnalysis(double[][][][] imageData,
                       Mask dataMask,
                       DesignMatrix designMatrix,
                       int[] statesID,
                       int lag,
                       int inferenceType)
    {
        this.imageData = imageData;
        this.dataMask = dataMask;
        this.designMatrix = designMatrix;
        this.statesID = statesID;
        this.lag = lag;
        //this.inferenceType = inferenceType;

        width = imageData.length;
        height = imageData[0].length;
        numberOfSlices = imageData[0][0].length;
        numberOfInstants = imageData[0][0][0].length;

        performGLMAnalysis();
```

```
}

//
//  Public Methods
//
//      double[][][] getResultData()
//

public double[][][] getGLMAnalysisResult()
{
    return glmAnalysisResult;
}

//
//  Private Methods
//
//      double[] getResiduesArray(double[] origArray,
//                                double[] fittedArray)
//      double[] getFittedArray(double[] intArray,
//                              double[] fittingCoef)
//      double[] extractTimeArray(int x,
//                                int y,
//                                int z,
//                                double dataMatrix[][][][])
//      double getAverage(double dataArray[])
//      double getVariance(double dataArray[])
//      double incBeta(double a, double b, double x)
//      double betacf(double a, double b, double x)
//      double lnGamma(double z)
//

private void performGLMAnalysis()
{
    getGLMRelCoefs();

    getGLMProbs();
}

private void getGLMRelCoefs()
{
    relCoefsVolume = new double[width][height][numberOfSlices];

    Timeline voxelTimeline;

    double[] timeArray;

    double[] dataArray;

    int x, y, z;

    GenLinFit dataLine;
    double[] dataCoefs;
    double[][] covariance;
    double totalVariance;

    double firstStateCoef;
    double secndStateCoef;

    double relCoefValue;

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if(dataMask.getBooleanValue(x, y, z))
                {
                    timeArray = extractTimeArray(x, y, z, imageData);

                    voxelTimeline = new Timeline(timeArray);
```

```
                    dataArray = voxelTimeline.getLaggedData(lag);

                    dataLine = new GenLinFit(dataArray, designMatrix);
                    dataCoefs = dataLine.getCoefs();
                    covariance = dataLine.getCovArray();

                    firstStateCoef = dataCoefs[statesID[0]];
                    secndStateCoef = dataCoefs[statesID[1]];
                    totalVariance = covariance[statesID[0]][statesID[0]]
                                + covariance[statesID[1]][statesID[1]];

                    relCoefValue = (firstStateCoef - secndStateCoef)
                                            /Math.sqrt(totalVariance);

                    if((dataCoefs[1]==0.0)&(totalVariance==0.0))
                    {
                        relCoefValue = -70.0;
                        singCounter = singCounter + 1;
                    }
                    if((dataCoefs[1]!=0.0)&(totalVariance==0.0))
                    {
                        relCoefValue = -70.0;
                        singCounter = singCounter + 1;
                    }

                    relCoefsVolume[x][y][z] = relCoefValue;
                }
                else
                {
                    relCoefsVolume[x][y][z] = 0.0;
                }
            }
        }
    }
}

private void getGLMProbs()
{
    int x, y, z;

    double[] serialData;
    int counter = 0;

    double mean;
    double stdDev;

    OneDArrayAnalyzer dataAnalyzer;

    GaussianDistribution glmGaussian;
    double pValue;

    glmAnalysisResult = new double[width][height][numberOfSlices];

    serialData = new double[dataMask.getNumberOfMaskVoxels()];

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if(dataMask.getBooleanValue(x, y, z))
                {
                    serialData[counter] = relCoefsVolume[x][y][z];
                    counter = counter + 1;
                }
            }
        }
    }
```

```
        dataAnalyzer = new OneDArrayAnalyzer(serialData);
        mean = dataAnalyzer.getMean();
        stdDev = dataAnalyzer.getStdDev();

        System.out.println("mean = " + mean);
        System.out.println("stdDev = " + stdDev);

        glmGaussian = new GaussianDistribution(mean, stdDev);

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    if(dataMask.getBooleanValue(x, y, z))
                    {
                        pValue =
                            glmGaussian.getCumProb(relCoefsVolume[x][y][z]);
                    }
                    else
                    {
                        pValue = 0.0;
                    }
                    glmAnalysisResult[x][y][z] = pValue;
                }
            }
        }

        resultMaxValue(relCoefsVolume);
        resultMinValue(relCoefsVolume);
    }



    /******* EXTRACT TIME ARRAY ******/

    private double[] extractTimeArray(int x,
                                      int y,
                                      int z,
                                      double dataMatrix[][][][])
    {
        int t;
        double[] timeArray = new double[numberOfInstants];

        for(t=0; t< numberOfInstants; t++)
        {
            timeArray[t] = dataMatrix[x][y][z][t];
        }
        return timeArray;
    }

    /******* RESULT MAXIMUM VALUE *******/

    private void resultMaxValue(double resultMatrix[][][])
    {
        int x, y, z;
        int xx=0;
        int yy=0;
        int zz=0;
        double elementValue;
        double maxValue = resultMatrix[0][0][0];

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    elementValue = resultMatrix[x][y][z];
                    if(elementValue > maxValue)
```

```
                {
                    if(elementValue > -70.0)
                    {
                        maxValue = elementValue;
                        xx = x;
                        yy = y;
                        zz = z;
                    }
                }
            }
        }
    }

    System.out.println("Max( p(x,y,z) ) = p("
                        + xx + "," + yy + "," + zz + ") = "
                        + glmAnalysisResult[xx][yy][zz]);

}


/******* RESULT MINIMUM VALUE *******/

private void resultMinValue(double resultMatrix[][][])
{
    int x, y, z;
    int xx=0;
    int yy=0;
    int zz=0;
    double elementValue;
    double minValue = resultMatrix[0][0][0];

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                elementValue = resultMatrix[x][y][z];
                if(elementValue < minValue)
                {
                    if(elementValue > -70.0)
                    {
                        minValue = elementValue;
                        xx = x;
                        yy = y;
                        zz = z;
                    }
                }
            }
        }
    }

    System.out.println("Min( p(x,y,z) ) = p("
                        + xx + "," + yy + "," + zz + ") = "
                        + glmAnalysisResult[xx][yy][zz]);

}


/******* SHOW ARRAY ******

private void showArray(double[] array)
{
    int size = array.length;
    int i;

    for(i=0; i<size; i++)
    {
        System.out.println(array[i]);
```

```
            }
        }
        */
}
```

```java
package graphUtils;

//
//  GraphPanel.java
//
//
//  Created by Jose Gabriel Lira Gomes on 8/20/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.image.MemoryImageSource;

import javax.swing.BorderFactory;
import javax.swing.JPanel;

import estatistica.descriptive.TwoDArrayAnalyzer;

public class GraphPanel extends JPanel
{
    private double[][][] multipleArray;

    private int panelWidth;
    private int panelHeight;
    private int[][][] scaledArray;
    private int dataSize;
    private int numberOfGraphs;
    private boolean withBackground = false;
    private double[][] backgroundData;
    private int width;
    private int height;
    private Image bckgImage;

    public GraphPanel(double[] singleArray, int panelWidth, int panelHeight)
    {
        this.multipleArray = getMultipleArray(singleArray);
        this.panelWidth = panelWidth;
        this.panelHeight = panelHeight;

        this.setPreferredSize(new Dimension(panelWidth, panelHeight));

        dataSize = singleArray.length;
        numberOfGraphs = 1;

        scaledArray = scaleToPanel();

        setOptions();
    }

    public GraphPanel(double[][] dualArray, int panelWidth, int panelHeight)
    {
        this.multipleArray = getMultipleArray(dualArray);
        this.panelWidth = panelWidth;
        this.panelHeight = panelHeight;

        this.setPreferredSize(new Dimension(panelWidth, panelHeight));

        dataSize = dualArray.length;
        numberOfGraphs = 1;

        scaledArray = scaleToPanel();

        setOptions();
    }

    public GraphPanel(double[][][] multipleArray,
                      int panelWidth,
                      int panelHeight)
```

```
{
    this.multipleArray = multipleArray;
    this.panelWidth = panelWidth;
    this.panelHeight = panelHeight;

    this.setPreferredSize(new Dimension(panelWidth, panelHeight));

    dataSize = multipleArray.length;
    numberOfGraphs = multipleArray[0][0].length;

    scaledArray = scaleToPanel();

    setOptions();
}

public void setBackground(double[][] backgroundData)
{
    this.backgroundData = backgroundData;
    withBackground = true;

    height = backgroundData.length;
    width = backgroundData[0].length;

    buildBackgroundImage();
    repaint();
}

public void removeBackground()
{
    if(withBackground)
    {
        withBackground = false;

        repaint();
    }
}

public void resetSingleArray(double[] singleArray)
{
    dataSize = singleArray.length;

    this.multipleArray = getMultipleArray(singleArray);

    scaledArray = scaleToPanel();

    repaint();
}

public void resetTwoArrays(double[] dataArrayOne, double[] dataArrayTwo)
{
    dataSize = dataArrayOne.length;

    this.multipleArray = getMultipleArray(dataArrayOne, dataArrayTwo);

    scaledArray = scaleToPanel();

    repaint();
}

public void updateSingleArray(double[] singleArray)
{
    setMultipleArray(singleArray, numberOfGraphs-1);

    scaledArray = scaleToPanel();

    repaint();
}

public void updateSingleArray(double[] singleArray, int arrayIndex)
{
    setMultipleArray(singleArray, arrayIndex);
```

```
        scaledArray = scaleToPanel();

        repaint();
    }

    public void updateDualArray(double[][] dualArray)
    {
        setMultipleArray(dualArray, numberOfGraphs-1);

        scaledArray = scaleToPanel();

        repaint();
    }

    public void updateDualArray(double[][] dualArray, int arrayIndex)
    {
        setMultipleArray(dualArray, arrayIndex);

        scaledArray = scaleToPanel();

        repaint();
    }

    public void updateMultipleArray(double[][][] multipleArray)
    {
        this.multipleArray = multipleArray;

        scaledArray = scaleToPanel();

        repaint();
    }

    public void addSingleArray(double[] singleArray, int arrayIndex)
    {
        this.multipleArray = addToMultipleArray(singleArray, arrayIndex);

        scaledArray = scaleToPanel();

        repaint();
    }

    public void addSingleArrayOver(double[] singleArray)
    {
        this.multipleArray = addToMultipleArray(singleArray, numberOfGraphs);

        scaledArray = scaleToPanel();

        repaint();
    }

    public void addSingleArrayUnder(double[] singleArray)
    {
        this.multipleArray = addToMultipleArray(singleArray, 0);

        scaledArray = scaleToPanel();

        repaint();
    }

    public void addDualArray(double[][] dualArray, int arrayIndex)
    {
        this.multipleArray = addToMultipleArray(dualArray, arrayIndex);

        scaledArray = scaleToPanel();

        repaint();
    }

    public void addDualArrayOver(double[][] dualArray)
    {
```

```
        this.multipleArray = addToMultipleArray(dualArray, numberOfGraphs);

        scaledArray = scaleToPanel();

        repaint();
    }

    public void addDualArrayUnder(double[][] dualArray)
    {
        this.multipleArray = addToMultipleArray(dualArray, 0);

        scaledArray = scaleToPanel();

        repaint();
    }

    public void removeSingleArray(int arrayIndex)
    {
        this.multipleArray = removeFromMultipleArray(arrayIndex);

        scaledArray = scaleToPanel();

        repaint();
    }

    public int getPanelWidth()
    {
        return panelWidth;
    }

    public int getPanelHeight()
    {
        return panelHeight;
    }

    public void paint(Graphics g)
    {
        super.paint(g);

        if(withBackground)
        {
            g.drawImage(bckgImage, 0, 0, this);
        }
        drawGraphs(g);
    }

    private void drawGraphs(Graphics g)
    {
        int i, j;
        float hue;
        int rgb;
        int red, green, blue;

        for(j=0; j<numberOfGraphs; j++)
        {
            if(numberOfGraphs > 1)
            {
                hue = (0.7f*(float)j)/(float)(numberOfGraphs-1);
                rgb = Color.HSBtoRGB((float)hue, (float)1.0, (float)1.0);

                red = (rgb>>16)&0xFF;
                green = (rgb>>8)&0xFF;
                blue = rgb&0xFF;

                g.setColor(new Color(red, green, blue));
            }
            else
            {
                g.setColor(Color.black);
            }
            for(i=0; i<dataSize-1; i++)
```

```
        {
            g.drawLine(scaledArray[i][0][j],
                        panelHeight-scaledArray[i][1][j],
                        scaledArray[i+1][0][j],
                        panelHeight-scaledArray[i+1][1][j]);
        }
    }
}

private void buildBackgroundImage()
{
    MemoryImageSource source;

    int i=0;

    int x, y;

    double colorLevel;
    int red, green, blue;

    int rgb;
    double hue;

    int pixels[] = new int[height*width];

    double max = new TwoDArrayAnalyzer(backgroundData).getMaximum();
    double min = new TwoDArrayAnalyzer(backgroundData).getMinimum();

    for(y=height-1; y >= 0; y--)
    {
        for(x=0; x< width; x++)
        {
            colorLevel = (backgroundData[y][x]-min)*0.7
                        /(Math.max(Math.abs(max), Math.abs(min))-min);

            hue = Math.abs(0.7-colorLevel);
            rgb = Color.HSBtoRGB((float)hue, (float)0.25, (float)1.0);

            red = (rgb>>16)&0xFF;
            green = (rgb>>8)&0xFF;
            blue = rgb&0xFF;

            pixels[i] = (255 << 24) | (red << 16) | (green << 8) | blue;

            i=i+1;
        }
    }

    source = new MemoryImageSource(width, height, pixels, 0, width);
    this.bckgImage = createImage(source);
}

private double[][][] getMultipleArray(double[] dataArrayOne,
                                      double[] dataArrayTwo)
{
    int size = dataArrayOne.length;
    double[][][] modArray = new double[size][2][2];
    int i;

    for(i=0; i<size; i++)
    {
        modArray[i][0][0] = (double)i;
        modArray[i][1][0] = dataArrayOne[i];
        modArray[i][0][1] = (double)i;
        modArray[i][1][1] = dataArrayTwo[i];
    }

    return modArray;
}

private double[][][] getMultipleArray(double[] dataArray)
```

```
{
    int size = dataArray.length;
    double[][][] modArray = new double[size][2][1];
    int i;

    for(i=0; i<size; i++)
    {
        modArray[i][0][0] = (double)i;
        modArray[i][1][0] = dataArray[i];
    }

    return modArray;
}

private double[][][] getMultipleArray(double[][] dataArray)
{
    int size = dataArray.length;
    double[][][] modArray = new double[size][2][1];
    int i;

    for(i=0; i<size; i++)
    {
        modArray[i][0][0] = dataArray[i][0];
        modArray[i][1][0] = dataArray[i][1];
    }

    return modArray;
}

private void setMultipleArray(double[] dataArray, int arrayIndex)
{
    int arraySize = dataArray.length;
    int i;

    if((arraySize == dataSize)
        &&(arrayIndex < numberOfGraphs)
        &&(arrayIndex >= 0))
    {
        for(i=0; i< arraySize; i++)
        {
            this.multipleArray[i][0][arrayIndex] = (double)i;
            this.multipleArray[i][1][arrayIndex] = dataArray[i];
        }
    }
}

private void setMultipleArray(double[][] dataArray, int arrayIndex)
{
    int arraySize = dataArray.length;
    int i;

    if((arraySize == dataSize)
        &&(arrayIndex < numberOfGraphs)
        &&(arrayIndex >= 0))
    {
        for(i=0; i< arraySize; i++)
        {
            this.multipleArray[i][0][arrayIndex] = dataArray[i][0];
            this.multipleArray[i][1][arrayIndex] = dataArray[i][1];
        }
    }
}

private double[][][] addToMultipleArray(double[] dataArray, int arrayIndex)
{
    int arraySize = dataArray.length;
    double[][][] modArray = new double[arraySize][2][numberOfGraphs+1];
    int i, j, k;
    boolean validArray = false;

    if((arraySize == dataSize)
```

```
            &&(arrayIndex <= numberOfGraphs)
            &&(arrayIndex >= 0))
        {
            if(arrayIndex < numberOfGraphs)
            {
                for(k=0; k<arrayIndex; k++)
                {
                    for(j=0; j<2; j++)
                    {
                        for(i=0; i< arraySize; i++)
                        {
                            modArray[i][j][k] = multipleArray[i][j][k];
                        }
                    }
                }

                for(k=arrayIndex+1; k<numberOfGraphs; k++)
                {
                    for(j=0; j<2; j++)
                    {
                        for(i=0; i< arraySize; i++)
                        {
                            modArray[i][j][k+1] = multipleArray[i][j][k];
                        }
                    }
                }
            }
            else
            {
                for(k=0; k<numberOfGraphs; k++)
                {
                    for(j=0; j<2; j++)
                    {
                        for(i=0; i< arraySize; i++)
                        {
                            modArray[i][j][k] = multipleArray[i][j][k];
                        }
                    }
                }
            }

            for(i=0; i< arraySize; i++)
            {
                modArray[i][0][arrayIndex] = (double)i;
                modArray[i][1][arrayIndex] = dataArray[i];
            }

            validArray = true;
        }


        if(validArray)
        {
            numberOfGraphs = numberOfGraphs + 1;

            return modArray;
        }
        else
        {
            return multipleArray;
        }
    }

    private double[][][] addToMultipleArray(double[][] dataArray,
                                            int arrayIndex)
    {
        int arraySize = dataArray.length;
        double[][][] modArray = new double[arraySize][2][numberOfGraphs+1];
        int i, j, k;
        boolean validArray = false;
```

```
        if((arraySize == dataSize)
            &&(arrayIndex <= numberOfGraphs)
            &&(arrayIndex >= 0))
        {
            if(arrayIndex < numberOfGraphs)
            {
                for(k=0; k<arrayIndex; k++)
                {
                    for(j=0; j<2; j++)
                    {
                        for(i=0; i< arraySize; i++)
                        {
                            modArray[i][j][k] = multipleArray[i][j][k];
                        }
                    }
                }

                for(k=arrayIndex+1; k<numberOfGraphs; k++)
                {
                    for(j=0; j<2; j++)
                    {
                        for(i=0; i< arraySize; i++)
                        {
                            modArray[i][j][k+1] = multipleArray[i][j][k];
                        }
                    }
                }
            }
            else
            {
                for(k=0; k<numberOfGraphs; k++)
                {
                    for(j=0; j<2; j++)
                    {
                        for(i=0; i< arraySize; i++)
                        {
                            modArray[i][j][k] = multipleArray[i][j][k];
                        }
                    }
                }
            }

            for(i=0; i< arraySize; i++)
            {
                modArray[i][0][arrayIndex] = dataArray[i][0];
                modArray[i][1][arrayIndex] = dataArray[i][1];
            }

            validArray = true;
        }


        if(validArray)
        {
            numberOfGraphs = numberOfGraphs + 1;

            return modArray;
        }
        else
        {
            return multipleArray;
        }
    }

    private double[][][] removeFromMultipleArray(int arrayIndex)
    {
        double[][][] modArray = new double[dataSize][2][numberOfGraphs-1];
        int i, j, k;

        for(k=0; k<arrayIndex; k++)
        {
```

```
        for(j=0; j<2; j++)
        {
            for(i=0; i< dataSize; i++)
            {
                modArray[i][j][k] = multipleArray[i][j][k];
            }
        }
    }

    for(k=arrayIndex+1; k<numberOfGraphs; k++)
    {
        for(j=0; j<2; j++)
        {
            for(i=0; i< dataSize; i++)
            {
                modArray[i][j][k-1] = multipleArray[i][j][k];
            }
        }
    }

    numberOfGraphs = numberOfGraphs - 1;

    return modArray;
}


private void setOptions()
{
    setBorder(BorderFactory.createLoweredBevelBorder());
    setBackground(Color.white);
}


private int[][][] scaleToPanel()
{
    double[][] y = new double[dataSize][numberOfGraphs];
    double[][] x = new double[dataSize][numberOfGraphs];

    double[] maxY;
    double[] minY;

    double[] maxX;
    double[] minX;

    double[] slopeY = new double[numberOfGraphs];
    double[] slopeX = new double[numberOfGraphs];

    int i, j;

    int[][] yy = new int[dataSize][numberOfGraphs];
    int[][] xx = new int[dataSize][numberOfGraphs];

    int[][][] modArray = new int[dataSize][2][numberOfGraphs];

    for(j=0; j<numberOfGraphs; j++)
    {
        for(i=0; i<dataSize; i++)
        {
            y[i][j] = multipleArray[i][1][j];
            x[i][j] = multipleArray[i][0][j];
        }
    }

    maxY = getMax(y);
    minY = getMin(y);

    maxX = getMax(x);
    minX = getMin(x);

    for(j=0; j<numberOfGraphs; j++)
    {
```

```
            slopeY[j] = panelHeight/(maxY[j]-minY[j]);
            slopeX[j] = panelWidth/(maxX[j]-minX[j]);

            for(i=0; i<dataSize; i++)
            {
                yy[i][j] = (int)Math.round(slopeY[j]*(y[i][j] - minY[j]));
                xx[i][j] = (int)Math.round(slopeX[j]*(x[i][j] - minX[j]));
            }

            for(i=0; i<dataSize; i++)
            {
                modArray[i][1][j] = yy[i][j];
                modArray[i][0][j] = xx[i][j];
            }
        }

        return modArray;
    }

    private double[] getMax(double[][] dataArray)
    {
        int i, j;
        double elementValue;
        double[] maxValue = new double[numberOfGraphs];

        for(j=0; j<numberOfGraphs; j++)
        {
            maxValue[j] = dataArray[0][j];
        }

        for(j=0; j<numberOfGraphs; j++)
        {
            for(i=1; i<dataSize; i++)
            {
                elementValue = dataArray[i][j];
                if(elementValue > maxValue[j])
                {
                    maxValue[j] = elementValue;
                }
            }
        }

        return maxValue;
    }

    private double[] getMin(double[][] dataArray)
    {
        int i, j;
        double elementValue;
        double[] minValue = new double[numberOfGraphs];

        for(j=0; j<numberOfGraphs; j++)
        {
            minValue[j] = dataArray[0][j];
        }

        for(j=0; j<numberOfGraphs; j++)
        {
            for(i=1; i<dataSize; i++)
            {
                elementValue = dataArray[i][j];
                if(elementValue < minValue[j])
                {
                    minValue[j] = elementValue;
                }
            }
        }

        return minValue;
    }
```

```
    /******* SHOW ARRAY *******/
    /*
    private void showArray(int[][] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i][1]);
        }
    }
    */
}
```

```
package windowUtils;

//
//  GraphPanelsWindow.java
//
//
//  Created by José Gabriel Lira Gomes on 1/30/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import java.awt.Container;
import java.awt.FlowLayout;

import matematica.algebra.Matrix;

import fMRIWindows.MenuedFrame;
import graphUtils.GraphPanel;

public class GraphPanelsWindow extends MenuedFrame
{
    private static final int defaultPanelHeight = 200;

    private GraphPanel[] graphPanels;
    private String windowTitle;
    private int panelsPerRow;

    private Container contentPane;

    private int windowWidth;
    private int windowHeight;
    private int windowXPosition = 0;
    private int windowYPosition = 21;

    public GraphPanelsWindow(GraphPanel[] graphPanels,
                             String windowTitle,
                             int panelsPerRow)
    {
        this.graphPanels = graphPanels;
        this.windowTitle = windowTitle;
        this.panelsPerRow = panelsPerRow;

        createGUI();
    }

    public GraphPanelsWindow(Matrix dataMatrix,
                             String windowTitle,
                             int panelsPerRow)
    {
        this.windowTitle = windowTitle;
        this.panelsPerRow = panelsPerRow;

        int numberOfPanels = dataMatrix.getNumberOfColumns();
        graphPanels = new GraphPanel[numberOfPanels];
        int i;
        double[] dataArray;

        for(i=0; i<numberOfPanels; i++)
        {
            dataArray = dataMatrix.getColumn(i);
            this.graphPanels[i] =
                    new GraphPanel(dataArray,
                                   dataMatrix.getNumberOfRows(),
                                   defaultPanelHeight);
        }

        createGUI();
    }

    private void createGUI()
    {
        int numberOfPanels = graphPanels.length;
```

```
        int numberOfRows = (int)Math.ceil(numberOfPanels/panelsPerRow);
        int i;
        int[] panelsWidths = new int[numberOfPanels];
        int[] panelsHeights = new int[numberOfPanels];

        for(i=0; i<numberOfPanels; i++)
        {
            panelsWidths[i] = graphPanels[i].getPanelWidth();
            panelsHeights[i] = graphPanels[i].getPanelHeight();
        }

        windowWidth = getMaximum(panelsWidths)*panelsPerRow;
        windowHeight = getMaximum(panelsHeights)*numberOfRows;

        contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());

        for(i=0; i<numberOfPanels; i++)
        {
            contentPane.add(graphPanels[i]);
        }

        setSize(windowWidth, windowHeight);
        setLocation(windowXPosition, windowYPosition);
        setTitle(windowTitle);
        setVisible(true);
    }

    private int getMaximum(int[] dataArray)
    {
        int maxValue = dataArray[0];

        int N = dataArray.length;

        int i;

        for(i=1; i<N; i++)
        {
            if(dataArray[i] > maxValue)
            {
                maxValue = dataArray[i];
            }
        }

        return maxValue;
    }
}
```

```java
package matematica.specialFunctions;

//
//  Heaviside.java
//
//
//  Created by José Gabriel Lira Gomes on 3/7/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class Heaviside
{
    private double z;

    public Heaviside(double z)
    {
        this.z = z;
    }

    public double getValue()
    {
        double value;

        if(z >= 0)
        {
            if(z == 0)
            {
                value = 0.5;
            }
            else
            {
                value = 1.0;
            }
        }
        else
        {
            value = 0.0;
        }

        return value;
    }
}
```

```java
package estatistica.distributions;

//
//  Histogram.java
//
//
//  Created by José Gabriel Lira Gomes on 4/3/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import matematica.algebra.LargeNumber;

public class Histogram
{
    private double[][] histogram;
    private int numberOfPoints;

    public Histogram(double[][] histogram)
    {
        this.histogram = histogram;
        this.numberOfPoints = histogram[0].length;
    }

    public int getNumberOfPoints()
    {
        return numberOfPoints;
    }

    public double[] getXValues()
    {
        return histogram[0];
    }

    public double[] getYValues()
    {
        return histogram[1];
    }

    public double getXMin()
    {
        return histogram[0][0];
    }

    public double getXMax()
    {
        return histogram[0][numberOfPoints-1];
    }

    public double getXRange()
    {
        return getXMax()-getXMin();
    }

    public double getXResolution()
    {
        return getXRange()/numberOfPoints;
    }

    public double getProbDensity(double x)
    {
        double probDensity;
        int i;
        double leftProbDensity = 0.0;
        double rightProbDensity = 0.0;

        if(x < getXMin())
        {
            probDensity = histogram[1][0];
        }
        else
        {
```

```
            if(x > getXMax())
            {
                probDensity = histogram[1][numberOfPoints-1];
            }
            else
            {
                for(i=0; i<numberOfPoints; i++)
                {
                    if(x >= histogram[0][i])
                    {
                        leftProbDensity = histogram[1][i];
                    }
                }

                for(i=numberOfPoints-1; i>=0; i--)
                {
                    if(x <= histogram[0][i])
                    {
                        rightProbDensity = histogram[1][i];
                    }
                }

                probDensity = (leftProbDensity + rightProbDensity)/2.0;
            }
        }

        return probDensity;
    }

    public double getCumProb(double x)
    {
        double cumProb = 0.0;
        double[] xValues = getXValues();
        int i;

        if(x < getXMin())
        {
            cumProb = 0.0;
        }
        else
        {
            if(x > getXMax())
            {
                cumProb = 1.0;
            }
            else
            {
                for(i=0; i<numberOfPoints; i++)
                {
                    if(x > xValues[i])
                    {
                        cumProb = cumProb
                            + getProbDensity(xValues[i])*getXResolution();
                    }
                }


            }
        }

        return cumProb;
    }

    public LargeNumber getHighPrecProbDensity(double x)
    {
        return new LargeNumber(getProbDensity(x));
    }

    public double[][] getHistogramArray()
    {
        return histogram;
```

```
        }
    }
```

```java
package functional.hrf;

//
//  Hrf.java
//
//
//  Created by José Gabriel Lira Gomes on 11/2/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

public class Hrf
{
    //
    //  Data Members
    //

    private double[] hrfArray;

    private int numberOfPoints = 16;

    //
    //  Constructors
    //
    //      hrf(double repeatTime, int numberOfPoints)
    //

    public Hrf(double repeatTime)
    {
        //  Parameters:
        //
        //      p[0]    -delay of response
        //      p[1]    -delay of undershoot
        //      p[2]    -dispersion of response
        //      p[3]    -dispersion of undershoot
        //      p[4]    -ratio of response to undershoot
        //      p[5]    -onset
        //      p[6]    -length of kernel

        double[] p = new double[7];// = {6, 16, 1, 1, 6, 0, 32};
        double dt, u;
        int i;
        double totalTime;

        hrfArray = new double[numberOfPoints];

        totalTime = repeatTime*numberOfPoints;
        p[0] = 6*totalTime/32;
        p[1] = 16*totalTime/32;
        p[2] = 1*totalTime/32;
        p[3] = 1*totalTime/32;
        p[4] = 6*totalTime/32;
        p[5] = 0;
        p[6] = totalTime;

        dt = repeatTime/(numberOfPoints-1);
        u = (p[6]-p[5])/dt;

        for(i=0; i<numberOfPoints; i++)
        {
            hrfArray[i] = spmGpdf(u*i*dt, p[0]/p[2], dt/p[2])
                        - spmGpdf(u*i*dt, p[1]/p[3], dt/p[3])/p[4];
        }

        //showArray(hrfArray);
    }

    //
    //  Public Methods
    //
    //      double[] getHRF()
    //
```

```
    public double[] getHRF()
    {
        return hrfArray;
    }

    //
    //  Private Methods
    //
    //      double spmGpdf(double x, double h, double l)
    //      double lnGamma(double z)
    //

    private double spmGpdf(double x, double h, double l)
    {
        double result;

        result = Math.exp( (h-1)*Math.log(x)
                + h*Math.log(l) - l*x - lnGamma(h) );

        return result;
    }

    private double lnGamma(double z)
    {
        int j;
        double x, y, tmp, ser, stp;
        double[] cof = new double[6];
        double gammaln;

        cof[0] = 76.18009172947146;
        cof[1] = -86.50532032941677;
        cof[2] = 24.01409824083091;
        cof[3] = -1.231739572450155;
        cof[4] = 0.1208650973866179e-2;
        cof[5] = -0.5395239384953e-5;

        y = x = z;
        tmp = x + 5.5;
        tmp = (x+0.5)*Math.log(tmp) - tmp;
        ser = 1.000000000190015;
        stp = 2.5066282746310005;

        for(j=0; j<6; j++)
        {
            y = y+1;
            ser = ser + cof[j]/y;
        }

        gammaln = tmp + Math.log(stp*ser/x);

        return gammaln;
    }



    /******* SHOW ARRAY *******/
    /*
    private void showArray(double[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */
}
```

```java
package imageUtils;

//
//  ImagePanel.java
//
//
//  Created by Jose Gabriel Lira Gomes on 10/19/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.image.MemoryImageSource;

import javax.swing.BorderFactory;
import javax.swing.JPanel;

import estatistica.descriptive.TwoDArrayAnalyzer;

public class ImagePanel extends JPanel
{
    private double[][][] multipleImageData;
    private int numberOfImages;

    private int zoomFactor;
    private double[] threshold;
    private boolean[] color;
    private double[] opacity;
    private boolean[] bckgVisible;

    private boolean[] variableExtremes;
    private double[] max;
    private double[] min;

    private int width;
    private int height;

    private int panelWidth;
    private int panelHeight;

    private int pixels[];
    private Image image;


    public ImagePanel(double[][] imageData, int zoomFactor, boolean color)
    {
        this.width = imageData.length;
        this.height = imageData[0].length;
        this.numberOfImages = 1;

        this.multipleImageData = getMultipleImageData(imageData);
        this.zoomFactor = zoomFactor;

        initParams();

        this.threshold[0] = 0.0;
        this.color[0] = color;
        this.opacity[0] = 1.0;
        this.bckgVisible[0] = false;

        variableExtremes[0] = true;
        TwoDArrayAnalyzer dataAnalyzer = new TwoDArrayAnalyzer(imageData);
        this.max[0] = dataAnalyzer.getMaximum();
        this.min[0] = dataAnalyzer.getMinimum();

        this.panelWidth = width*zoomFactor;
        this.panelHeight = height*zoomFactor;

        this.setPreferredSize(new Dimension(panelWidth, panelHeight));
```

```
        initPixels();

        updatePixels();

        buildImage();

        setOptions();
    }

    public ImagePanel(double[][] imageData,
                      int zoomFactor,
                      double threshold,
                      double max,
                      double min,
                      boolean color)
    {
        this.width = imageData.length;
        this.height = imageData[0].length;
        this.numberOfImages =  1;

        this.multipleImageData = getMultipleImageData(imageData);
        this.zoomFactor = zoomFactor;

        initParams();

        this.threshold[0] = threshold;
        this.color[0] = color;
        this.opacity[0] = 1.0;
        this.bckgVisible[0] = false;

        variableExtremes[0] = false;
        this.max[0] = max;
        this.min[0] = min;

        this.panelWidth = width*zoomFactor;
        this.panelHeight = height*zoomFactor;

        this.setPreferredSize(new Dimension(panelWidth, panelHeight));

        initPixels();

        updatePixels();

        buildImage();

        setOptions();
    }

    public ImagePanel(double[][] bckgImageData,
                      double[][] frgdImageData,
                      int zoomFactor,
                      double threshold,
                      double bckgMax,
                      double bckgMin,
                      double frgdMax,
                      double frgdMin,
                      boolean colorBackground,
                      boolean colorForeground)
    {
        this.width = bckgImageData.length;
        this.height = bckgImageData[0].length;
        this.numberOfImages = 2;

        this.multipleImageData = getMultipleImageData(
                                            bckgImageData,
                                            frgdImageData);
        this.zoomFactor = zoomFactor;

        initParams();
```

```
        this.threshold[0] = 0.0;
        this.threshold[1] = threshold;
        this.color[0] = colorBackground;
        this.color[1] = colorForeground;
        this.opacity[0] = 1.0;
        this.opacity[1] = 1.0;
        this.bckgVisible[0] = false;
        this.bckgVisible[1] = true;

        variableExtremes[0] = false;
        variableExtremes[1] = false;
        this.max[0] = bckgMax;
        this.min[0] = bckgMin;
        this.max[1] = frgdMax;
        this.min[1] = frgdMin;

        this.panelWidth = width*zoomFactor;
        this.panelHeight = height*zoomFactor;

        this.setPreferredSize(new Dimension(panelWidth, panelHeight));

        initPixels();

        updatePixels();

        buildImage();

        setOptions();
    }

    public void setImageDataAndThreshold(double[][] imageData,
                                          double threshold,
                                          int layer)
    {
        if(layer < numberOfImages)
        {
            setMultipleImageData(imageData, layer);

            this.threshold[layer] = threshold;

            if(variableExtremes[layer])
            {
                getExtremes(layer);
            }

            updatePixels();

            buildImage();

            repaint();
        }
    }

    public void setImageData(double[][] imageData, int layer)
    {
        if(layer < numberOfImages)
        {
            setMultipleImageData(imageData, layer);

            this.threshold[layer] = 0.0;

            if(variableExtremes[layer])
            {
                getExtremes(layer);
            }
            updatePixels();

            buildImage();

            repaint();
        }
```

```
    }

    public void setBooleanImageData(boolean[][] booleanImageData, int layer)
    {
        if(layer < numberOfImages)
        {
            setMultipleImageData(getImageData(booleanImageData), layer);

            this.threshold[layer] = 0.5;

            this.variableExtremes[layer] = false;
            this.max[layer] = 1.0;
            this.min[layer] = 0.0;

            updatePixels();

            buildImage();

            repaint();
        }
    }

    public void setBckgImageData(double[][] imageData)
    {
        setImageData(imageData, 0);
    }

    public void addLayerImageData(double[][] imageData,
                                  double threshold,
                                  boolean color,
                                  int layer)
    {
        if(layer <= numberOfImages)
        {
            this.multipleImageData = addLayer(imageData, layer);

            this.threshold = incrementArray(this.threshold, threshold, layer);
            this.color = incrementArray(this.color, color, layer);
            this.opacity = incrementArray(opacity, 1.0, layer);
            this.bckgVisible = incrementArray(this.bckgVisible, true, layer);

            this.variableExtremes =
                        incrementArray(this.variableExtremes, true, layer);

            TwoDArrayAnalyzer dataAnalyzer;
            dataAnalyzer = new TwoDArrayAnalyzer(getImageData(layer));

            this.max =
                incrementArray(this.max, dataAnalyzer.getMaximum(), layer);
            this.min =
                incrementArray(this.min, dataAnalyzer.getMinimum(), layer);

            updatePixels();

            buildImage();

            repaint();
        }
    }

    public void addLayerImageData(double[][] imageData,
                                  double threshold,
                                  boolean color,
                                  double max,
                                  double min,
                                  int layer)
    {
        if(layer <= numberOfImages)
        {
            this.multipleImageData = addLayer(imageData, layer);
```

```
            this.threshold = incrementArray(this.threshold, threshold, layer);
            this.color = incrementArray(this.color, color, layer);
            this.opacity = incrementArray(opacity, 1.0, layer);
            this.bckgVisible = incrementArray(this.bckgVisible, true, layer);

            this.variableExtremes =
                        incrementArray(this.variableExtremes, false, layer);
            this.max = incrementArray(this.max, max, layer);
            this.min = incrementArray(this.min, min, layer);

            updatePixels();

            buildImage();

            repaint();
        }
    }

    public void addBooleanLayerImageData(boolean[][] booleanImageData,
                                         boolean color,
                                         int layer)
    {
        if(layer <= numberOfImages)
        {
            this.multipleImageData =
                        addLayer(getImageData(booleanImageData), layer);

            this.threshold = incrementArray(threshold, 0.5, layer);
            this.color = incrementArray(this.color, color, layer);
            this.opacity = incrementArray(opacity, 0.5, layer);
            this.bckgVisible = incrementArray(this.bckgVisible, true, layer);

            this.variableExtremes =
                        incrementArray(this.variableExtremes, false, layer);
            this.max = incrementArray(this.max, 1.0, layer);
            this.min = incrementArray(this.min, 0.0, layer);

            updatePixels();

            buildImage();

            repaint();
        }
    }

    public void removeLayerImageData(int layer)
    {
        if(layer < numberOfImages)
        {
            removeLayer(layer);

            this.threshold = decrementArray(threshold, layer);
            this.color = decrementArray(this.color, layer);
            this.opacity = decrementArray(opacity, layer);
            this.bckgVisible = decrementArray(this.bckgVisible, layer);

            this.variableExtremes =
                        decrementArray(this.variableExtremes, layer);
            this.max = decrementArray(this.max, layer);
            this.min = decrementArray(this.min, layer);

            updatePixels();

            buildImage();

            repaint();
        }
    }

    /*
    public void removeBackground()
```

```
{
    removeLayerImageData(0);
}
*/

public void hideBackground()
{
    if(numberOfImages > 1)
    {
        setBckgVisibility(false, 1);
    }
    else
    {
        resetPixels();

        buildImage();

        repaint();
    }
}

public void showBackground()
{
    for(int i=1; i<numberOfImages; i++)
    {
        setBckgVisibility(true, i);
    }
}

public void setPanelSize(int panelWidth, int panelHeight)
{
    this.panelWidth = panelWidth;
    this.panelHeight = panelHeight;

    this.setPreferredSize(new Dimension(panelWidth, panelHeight));

    repaint();
}

public void setZoomFactor(int zoomFactor)
{
    this.zoomFactor = zoomFactor;

    resetPixels();

    updatePixels();

    buildImage();

    repaint();
}

public void setThreshold(double threshold, int layer)
{
    this.threshold[layer] = threshold;

    updatePixels();

    buildImage();

    repaint();
}

public void setTransparency(double opacity, int layer)
{
    this.opacity[layer] = opacity;

    updatePixels();

    buildImage();
```

```
        repaint();
    }

    public void setBckgVisibility(boolean bckgVisible, int layer)
    {
        this.bckgVisible[layer] = bckgVisible;

        updatePixels();

        buildImage();

        repaint();
    }

    public Image getImage()
    {
        buildImage();

        return image;
    }

    public void paint(Graphics g)
    {
        g.drawImage(image, 0, 0, this);
    }

    private void buildImage()
    {
        MemoryImageSource source;

        source = new MemoryImageSource(zoomFactor*width,
                                       zoomFactor*height,
                                       pixels,
                                       0,
                                       zoomFactor*width);

        image = createImage(source);
    }

    private void initParams()
    {
        threshold = new double[numberOfImages];
        color = new boolean[numberOfImages];
        opacity = new double[numberOfImages];
        bckgVisible = new boolean[numberOfImages];

        variableExtremes = new boolean[numberOfImages];
        max = new double[numberOfImages];
        min = new double[numberOfImages];
    }

    private void initPixels()
    {
        pixels = new int[height*zoomFactor*width*zoomFactor];
    }

    private void resetPixels()
    {
        initPixels();
    }

    private void getExtremes(int layer)
    {
        TwoDArrayAnalyzer dataAnalyzer;

        dataAnalyzer = new TwoDArrayAnalyzer(getImageData(layer));
        this.max[layer] = dataAnalyzer.getMaximum();
        this.min[layer] = dataAnalyzer.getMinimum();
    }

    private void updatePixels()
```

```
{
    int i;
    int j, k, l;

    int x, y;

    double colorLevel;
    double resampledColorLevel;
    int red, green, blue;

    double maxHue = 0.0;
    double medHue = 0.4;
    double minHue = 0.8;

    int alpha; // between 0 and 255
    int rgb;
    double hue;

    for(l=0; l< numberOfImages; l++)
    {
        i=0;
        for(y=height-1; y >= 0; y--)
        {
            for(j=0; j<zoomFactor; j++)
            {
                for(x=0; x< width; x++)
                {
                    if(color[l])
                    {
                        colorLevel = (multipleImageData[x][y][l]-min[l])
                            /(Math.max(Math.abs(max[l]),
                                       Math.abs(min[l]))-min[l]);
                        resampledColorLevel =
                            (Math.abs(colorLevel)-threshold[l])
                                /(1-threshold[l]);

                        if (colorLevel < -threshold[l])
                        {
                            alpha = (int)Math.round(255*opacity[l]);
                            hue = (- medHue + minHue)*resampledColorLevel
                                        + medHue;
                            rgb = Color.HSBtoRGB(
                                    (float)hue, (float)1.0, (float)1.0);
                        }
                        else
                        {
                            if (colorLevel >= threshold[l])
                            {
                                alpha = (int)Math.round(255*opacity[l]);
                                hue = (maxHue - medHue)*resampledColorLevel
                                        + medHue;
                                rgb = Color.HSBtoRGB(
                                        (float)hue, (float)1.0, (float)1.0);
                            }
                            else
                            {
                                if(bckgVisible[l])
                                {
                                    alpha = 0;
                                }
                                else
                                {
                                    alpha = 255;
                                }
                                rgb = Color.HSBtoRGB(
                                        (float)1.0, (float)0.0, (float)0.0);
                            }
                        }
                        red = (rgb>>16)&0xFF;
                        green = (rgb>>8)&0xFF;
                        blue = rgb&0xFF;
```

```
                        }
                        else
                        {
                            if(max == min)
                            {
                                colorLevel = 0.0;
                            }
                            else
                            {
                                colorLevel =
                                    255*(multipleImageData[x][y][l] - min[l])
                                    /Math.abs(max[l]-min[l]);
                            }

                            if (colorLevel < -threshold[l])
                            {
                                alpha = (int)Math.round(255*opacity[l]);
                                red = (int)Math.round(colorLevel);
                                green = red;
                                blue = red;
                            }
                            else
                            {
                                if (colorLevel >= threshold[l])
                                {
                                    alpha = (int)Math.round(255*opacity[l]);
                                    red = (int)Math.round(colorLevel);
                                    green = red;
                                    blue = red;
                                }
                                else
                                {
                                    if(bckgVisible[l])
                                    {
                                        alpha = 0;
                                    }
                                    else
                                    {
                                        alpha = 255;
                                    }
                                    red = 0;
                                    green = red;
                                    blue = red;
                                }
                            }
                        }
                        for(k=0; k<zoomFactor; k++)
                        {
                            if(alpha > 0)
                            {
                                pixels[i] =(alpha << 24)
                                            | (red << 16)
                                            | (green << 8)
                                            | blue;
                            }

                            i=i+1;
                        }
                    }
                }
            }
        }

    private void setOptions()
    {
        setBorder(BorderFactory.createLoweredBevelBorder());
        setBackground(Color.white);
    }
```

```
private double[][][] getMultipleImageData(double[][] dataArray)
{
    double[][][] modData = new double[width][height][1];
    int i, j;

    for(i=0; i<width; i++)
    {
        for(j=0; j<height; j++)
        {
            modData[i][j][0] = dataArray[i][j];
            modData[i][j][0] = dataArray[i][j];
        }
    }

    return modData;
}

private double[][][] getMultipleImageData(double[][] dataArrayOne,
                                          double[][] dataArrayTwo)
{
    double[][][] modData = new double[width][height][2];
    int i, j;

    for(i=0; i<width; i++)
    {
        for(j=0; j<height; j++)
        {
            modData[i][j][0] = dataArrayOne[i][j];
            modData[i][j][0] = dataArrayOne[i][j];
            modData[i][j][1] = dataArrayTwo[i][j];
            modData[i][j][1] = dataArrayTwo[i][j];
        }
    }

    return modData;
}

private void setMultipleImageData(double[][] dataArray, int layer)
{
    int i, j;

    for(i=0; i<width; i++)
    {
        for(j=0; j<height; j++)
        {
            this.multipleImageData[i][j][layer] = dataArray[i][j];
        }
    }
}

private double[][][] addLayer(double[][] dataArray, int layer)
{
    double[][][] modData = new double[width][height][numberOfImages+1];
    int i, j, k;

    for(k=0; k<layer; k++)
    {
        for(i=0; i<width; i++)
        {
            for(j=0; j<height; j++)
            {
                modData[i][j][k] = multipleImageData[i][j][k];
            }
        }
    }

    for(i=0; i<width; i++)
    {
        for(j=0; j<height; j++)
        {
            modData[i][j][layer] = dataArray[i][j];
```

```
            }
        }

        for(k=layer+1; k<numberOfImages+1; k++)
        {
            for(i=0; i<width; i++)
            {
                for(j=0; j<height; j++)
                {
                    modData[i][j][k] = multipleImageData[i][j][k-1];
                }
            }
        }

        numberOfImages = numberOfImages + 1;

        return modData;
    }

    private double[][][] removeLayer(int layer)
    {
        double[][][] modData = new double[width][height][numberOfImages-1];
        int i, j, k;

        for(k=0; k<layer; k++)
        {
            for(i=0; i<width; i++)
            {
                for(j=0; j<height; j++)
                {
                    modData[i][j][k] = multipleImageData[i][j][k];
                }
            }
        }

        for(k=layer+1; k<numberOfImages; k++)
        {
            for(i=0; i<width; i++)
            {
                for(j=0; j<height; j++)
                {
                    modData[i][j][k-1] = multipleImageData[i][j][k];
                }
            }
        }

        numberOfImages = numberOfImages - 1;

        return modData;
    }

    private double[][] getImageData(int layer)
    {
        int i, j;
        double[][] imageData = new double[width][height];

        for(i=0; i<width; i++)
        {
            for(j=0; j<height; j++)
            {
                imageData[i][j] = multipleImageData[i][j][layer];
            }
        }

        return imageData;
    }

    private double[][] getImageData(boolean[][] dataArray)
    {
        int i, j;
        double[][] imageData = new double[width][height];
```

```
        for(i=0; i<width; i++)
        {
            for(j=0; j<height; j++)
            {
                if(dataArray[i][j])
                {
                    imageData[i][j] = 1.0;
                }
                else
                {
                    imageData[i][j] = 0.0;
                }
            }
        }

        return imageData;
    }

    private double[] incrementArray(double[] dataArray,
                                    double value,
                                    int index)
    {
        int size = dataArray.length;
        double[] modData = new double[size+1];
        int i;

        for(i=0; i<index; i++)
        {
            modData[i] = dataArray[i];
        }

        modData[index] = value;

        for(i=index+1; i<size+1; i++)
        {
            modData[i] = dataArray[i-1];
        }

        return modData;
    }

    private boolean[] incrementArray(boolean[] dataArray,
                                     boolean value,
                                     int index)
    {
        int size = dataArray.length;
        boolean[] modData = new boolean[size+1];
        int i;

        for(i=0; i<index; i++)
        {
            modData[i] = dataArray[i];
        }

        modData[index] = value;

        for(i=index+1; i<size+1; i++)
        {
            modData[i] = dataArray[i-1];
        }

        return modData;
    }

    private Image[] incrementArray(Image[] dataArray,
                                   Image value,
                                   int index)
    {
        int size = dataArray.length;
        Image[] modData = new Image[size+1];
```

```
        int i;

        for(i=0; i<index; i++)
        {
            modData[i] = dataArray[i];
        }

        modData[index] = value;

        for(i=index+1; i<size+1; i++)
        {
            modData[i] = dataArray[i-1];
        }

        return modData;
    }

    private double[] decrementArray(double[] dataArray, int index)
    {
        int size = dataArray.length;
        double[] modData = new double[size-1];
        int i;

        for(i=0; i<index; i++)
        {
            modData[i] = dataArray[i];
        }

        for(i=index; i<size-1; i++)
        {
            modData[i] = dataArray[i+1];
        }

        return modData;
    }

    private boolean[] decrementArray(boolean[] dataArray, int index)
    {
        int size = dataArray.length;
        boolean[] modData = new boolean[size-1];
        int i;

        for(i=0; i<index; i++)
        {
            modData[i] = dataArray[i];
        }

        for(i=index; i<size-1; i++)
        {
            modData[i] = dataArray[i+1];
        }

        return modData;
    }

    private Image[] decrementArray(Image[] dataArray, int index)
    {
        int size = dataArray.length;
        Image[] modData = new Image[size-1];
        int i;

        for(i=0; i<index; i++)
        {
            modData[i] = dataArray[i];
        }

        for(i=index; i<size-1; i++)
        {
            modData[i] = dataArray[i+1];
        }
```

```
            return modData;
        }
    }
```

```java
package matematica.specialFunctions;

//
//  incBeta.java
//
//
//  Created by Jose Gabriel Lira Gomes on 11/16/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

import matematica.specialFunctions.Gamma;

public class IncBeta
{
    //
    //  Data Members
    //

    private double incBetaRes;


    //
    //  Constructors
    //
    //      incBeta(double a, double b, double x)
    //

    public IncBeta(double a, double b, double x)
    {
        double bt;
        double lnGammaOfAPlusB;
        double lnGammaOfA;
        double lnGammaOfB;

        lnGammaOfAPlusB = new Gamma(a+b).getLnGamma();
        lnGammaOfA = new Gamma(a).getLnGamma();
        lnGammaOfB = new Gamma(b).getLnGamma();

        if(x < 0.0 || x > 1.0)
        {
            System.out.println("bad argument x in betai");
        }
        if(x == 0.0 || x == 1.0)
        {
            bt = 0.0;
        }
        else
        {
            bt = Math.exp(lnGammaOfAPlusB
                          -lnGammaOfA
                          -lnGammaOfB
                          +a*Math.log(x)
                          +b*Math.log(1.0-x));
        }
        if(x < (a+1.0)/(a+b+2.0))
        {
            incBetaRes = bt*betacf(a, b, x)/a;
        }
        else
        {
            incBetaRes = 1.0 - bt*betacf(b, a, 1.0-x)/b;
        }
    }


    //
    //  Public Methods
    //
    //      double getIncBeta()
    //
```

```
public double getIncBeta()
{
    return incBetaRes;
}


//
//  Private Methods
//
//      double betacf(double a, double b, double x)
//

private double betacf(double a, double b, double x)
{
    int MAXIT=100;
    double EPS=3.0e-7;
    double FPMIN=1.0e-30;
    int m, m2;
    double dm, dm2;
    double aa, c, d, del, h, qab, qam, qap;

    qab = a + b;
    qap = a + 1.0;
    qam = a - 1.0;
    c = 1.0;
    d = 1.0 - qab*x/qap;

    if(Math.abs(d) < FPMIN)
    {
        d = FPMIN;
    }

    d = 1.0/d;
    h = d;
    for(m=1; m<=MAXIT; m++)
    {
        dm = (double)m;
        m2 = 2*m;
        dm2 = (double)m2;
        aa = dm*(b-dm)*x/((qam+dm2)*(a+dm2));
        d = 1.0 + aa*d;
        if(Math.abs(d) < FPMIN)
        {
            d = FPMIN;
        }
        c = 1.0 + aa/c;
        if(Math.abs(c) < FPMIN)
        {
            c = FPMIN;
        }
        d = 1.0/d;
        h = h*d*c;
        aa = -(a+dm)*(qab+dm)*x/((a+dm2)*(qap+dm2));
        d = 1.0 + aa*d;
        if(Math.abs(d) < FPMIN)
        {
            d = FPMIN;
        }
        c = 1.0 + aa/c;
        if(Math.abs(c) < FPMIN)
        {
            c = FPMIN;
        }
        d = 1.0/d;
        del = d*c;
        h = h*del;
        if(Math.abs(del-1.0) <= EPS)
        {
            break;
        }
    }
```

```
            if(m > MAXIT)
            {
                System.out.println(
                    "a or b too big, or MAXIT too small in betacf");
                System.out.println("a: " + a + "; b: " + b + ";");
            }
            return h;
        }
    }
```

```java
package imageUtils;

//
//  JpegEncoder.java
//
//
//  Created by Jose Gabriel Lira Gomes on 12/5/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.image.BufferedImage;
import java.io.FileOutputStream;
import java.io.IOException;

import com.sun.image.codec.jpeg.JPEGCodec;
import com.sun.image.codec.jpeg.JPEGImageEncoder;

public class JpegEncoder
{
    public JpegEncoder(Image imageToEncode, String dirPath, String fileName)
    {
        FileOutputStream out;
        JPEGImageEncoder encoder;
        BufferedImage bimage;
        Graphics2D g2d;

        try
        {
            /* open a file */
            out = new FileOutputStream(dirPath + "/" + fileName + ".jpg");

            bimage = new BufferedImage(imageToEncode.getWidth(null),
                                       imageToEncode.getHeight(null),
                                       BufferedImage.TYPE_INT_RGB);
            g2d = bimage.createGraphics();
            g2d.drawImage(imageToEncode, null, null);

            encoder = JPEGCodec.createJPEGEncoder(out);
            encoder.encode(bimage);

            out.flush();
            out.close();
            /* finish encoding */

        }
        catch (IOException e)
        {
            System.out.println("Error on jpeg encoding of file: "
                               + fileName + ".jpg");
        }
    }
}
```

```java
package functional.export;

//
//  JpegExporter.java
//
//
//  Created by JGLG on 4/22/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

import imageUtils.ImagePanel;
import imageUtils.JpegEncoder;

import java.awt.Image;
import java.text.DecimalFormat;

public class JpegExporter
{
    private double[][][] imageData;
    private double[][][] bckgData;
    private double threshold;
    private int zoomFactor;
    private boolean color;
    private String dirPath;
    private String method;

    private boolean withBackground = false;

    private int width;
    private int height;
    private int numberOfSlices;

    private Image jpegDataImage;

    public JpegExporter(double[][][] imageData,
                        double threshold,
                        int zoomFactor,
                        boolean color,
                        String dirPath,
                        String method)
    {
        this.imageData = imageData;
        this.threshold = threshold;
        this.zoomFactor = zoomFactor;
        this.color = color;
        this.dirPath = dirPath;
        this.method = method;

        exportJpegImages();
    }

    public JpegExporter(double[][][] imageData,
                        double[][][] bckgData,
                        double threshold,
                        int zoomFactor,
                        boolean color,
                        String dirPath,
                        String method)
    {
        this.imageData = imageData;
        this.bckgData = bckgData;
        this.threshold = threshold;
        this.zoomFactor = zoomFactor;
        this.color = color;
        this.dirPath = dirPath;
        this.method = method;

        withBackground = true;

        exportJpegImages();
    }
```

```
private void exportJpegImages()
{
    double frgdMax;
    double frgdMin;
    double bckgMax;
    double bckgMin;
    int z;
    ImagePanel imagePanel;

    width = imageData.length;
    height = imageData[0].length;
    numberOfSlices = imageData[0][0].length;

    frgdMax = dataMaxValue(imageData);
    frgdMin = dataMinValue(imageData);

    if(withBackground)
    {
        bckgMax = dataMaxValue(bckgData);
        bckgMin = dataMinValue(bckgData);
    }
    else
    {
        bckgMax = 0.0;
        bckgMin = 0.0;
    }

    for(z=0; z< numberOfSlices; z++)
    {
        if(withBackground)
        {
            imagePanel = new ImagePanel(getSlice(bckgData, z),
                                        getSlice(imageData, z),
                                        zoomFactor,
                                        threshold,
                                        bckgMax,
                                        bckgMin,
                                        frgdMax,
                                        frgdMin,
                                        false,
                                        color);
        }
        else
        {
            imagePanel = new ImagePanel(getSlice(imageData, z),
                                        zoomFactor,
                                        threshold,
                                        frgdMax,
                                        frgdMin,
                                        color);
        }
        jpegDataImage = imagePanel.getImage();
        new JpegEncoder(jpegDataImage,
                        dirPath,
                        method + addTrailZeroes(z, 4));
    }
}



/******* ADD TRAILING ZEROES *******/

private String addTrailZeroes(int number, int numDigits)
{
    int i;
    String mask = "";
    String modifiedNumber;

    for(i=0; i<numDigits; i++)
    {
```

```
            mask = mask + "0";
        }

        DecimalFormat df = new DecimalFormat(mask);
        modifiedNumber = df.format(number);
        return modifiedNumber;
    }

    private double[][] getSlice(double[][][] volumeDataArray, int z)
    {
        int x, y;
        double[][] sliceDataArray = new double[width][height];

        for(y=0; y<height; y++)
        {
            for(x=0; x<width; x++)
            {
                sliceDataArray[x][y] = volumeDataArray[x][y][z];
            }
        }

        return sliceDataArray;
    }

    /******* DATA MAXIMUM VALUE *******/

    private double dataMaxValue(double dataMatrix[][][])
    {
        int x, y, z;

        double elementValue;
        double maxValue=0.0;

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    elementValue = dataMatrix[x][y][z];
                    if(elementValue > maxValue)
                    {
                        maxValue = elementValue;
                    }
                }
            }
        }

        return maxValue;
    }



    /******* DATA MINIMUM VALUE *******/

    private double dataMinValue(double dataMatrix[][][])
    {
        int x, y, z;

        double elementValue;
        double minValue=0.0;

        for(z=0; z< numberOfSlices; z++)
        {
            for(y=0; y< height; y++)
            {
                for(x=0; x< width; x++)
                {
                    elementValue = dataMatrix[x][y][z];
                    if(elementValue < minValue)
                    {
```

```
                        minValue = elementValue;
                    }
                }
            }
        }

        return minValue;
    }

}
```

```java
package matematica.algebra;

//
//  LargeNumber.java
//
//
//  Created by José Gabriel Lira Gomes on 4/4/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class LargeNumber
{
    private double mantissa;
    private double exponent;

    public LargeNumber()
    {
        this.mantissa = 0.0;
        this.exponent = 0.0;
    }

    public LargeNumber(double mantissa)
    {
        this.mantissa = mantissa;
        this.exponent = 0.0;

        reduce();
    }

    public LargeNumber(double mantissa, double exponent)
    {
        this.mantissa = mantissa;
        this.exponent = exponent;

        reduce();
    }

    public double getMantissa()
    {
        return mantissa;
    }

    public void setMantissa(double mantissa)
    {
        this.mantissa = mantissa;

        reduce();
    }

    public double getExponent()
    {
        return exponent;
    }

    public void setExponent(double exponent)
    {
        this.exponent = exponent;

        reduce();
    }

    public double getDouble()
    {
        if(exponent > -308)
        {
            return mantissa*Math.pow(10, exponent);
        }
        else
        {
            return 0.0;
        }
```

```
}

public String getString()
{
    return Double.toString(mantissa) + "E" + Double.toString(exponent);
}

public LargeNumber add(LargeNumber largeFactor)
{
    double sumMantissa;
    double sumExponent;

    if(exponent == largeFactor.getExponent())
    {
        sumMantissa = (mantissa + largeFactor.getMantissa());
        sumExponent = exponent;
    }
    else
    {
        if(exponent > largeFactor.getExponent())
        {
            if(exponent - largeFactor.getExponent() > 10.0)
            {
                sumMantissa = mantissa;
                sumExponent = exponent;
            }
            else
            {
                sumMantissa = (mantissa*Math.pow(10, exponent
                                - largeFactor.getExponent())
                                + largeFactor.getMantissa());
                sumExponent = largeFactor.getExponent();
            }

        }
        else
        {
            if(largeFactor.getExponent() - exponent > 10.0)
            {
                sumMantissa = largeFactor.getMantissa();
                sumExponent = largeFactor.getExponent();
            }
            else
            {
                sumMantissa = (mantissa + largeFactor.getMantissa()
                    *Math.pow(10, largeFactor.getExponent() - exponent));
                sumExponent = exponent;
            }

        }
    }

    return new LargeNumber(sumMantissa, sumExponent);
}

public LargeNumber subtract(LargeNumber largeFactor)
{
    double differenceMantissa;
    double differenceExponent;

    if(exponent == largeFactor.getExponent())
    {
        differenceMantissa = (mantissa - largeFactor.getMantissa());
        differenceExponent = exponent;
    }
    else
    {
        if(exponent > largeFactor.getExponent())
        {
            if(exponent - largeFactor.getExponent() > 10.0)
            {
```

```
                differenceMantissa = mantissa;
                differenceExponent = exponent;
            }
            else
            {
                differenceMantissa = (mantissa*Math.pow(10, exponent
                - largeFactor.getExponent()) - largeFactor.getMantissa());
                differenceExponent = largeFactor.getExponent();
            }
        }
        else
        {
            if(largeFactor.getExponent() - exponent > 10.0)
            {
                differenceMantissa = largeFactor.getMantissa();
                differenceExponent = largeFactor.getExponent();
            }
            else
            {
                differenceMantissa = (mantissa - largeFactor.getMantissa()
                *Math.pow(10, largeFactor.getExponent() - exponent));
                differenceExponent = exponent;
            }
        }
    }

    return new LargeNumber(differenceMantissa, differenceExponent);
}

public LargeNumber multiply(LargeNumber largeFactor)
{
    double productMantissa;
    double productExponent;

    productMantissa = mantissa * largeFactor.getMantissa();
    productExponent = exponent + largeFactor.getExponent();

    return new LargeNumber(productMantissa, productExponent);
}

public LargeNumber multiply(double dblNumber)
{
    LargeNumber largeFactor = new LargeNumber(dblNumber);

    return this.multiply(largeFactor);
}

public LargeNumber divide(LargeNumber largeFactor)
{
    double ratioMantissa;
    double ratioExponent;

    ratioMantissa = mantissa / largeFactor.getMantissa();
    ratioExponent = exponent - largeFactor.getExponent();

    return new LargeNumber(ratioMantissa, ratioExponent);
}

public LargeNumber divide(double dblNumber)
{
    LargeNumber largeFactor = new LargeNumber(dblNumber);

    return this.divide(largeFactor);
}

public LargeNumber power(double exponent)
{
    double powerMantissa;
    double powerExponent;

    powerMantissa = Math.pow(mantissa, exponent);
```

```
        powerExponent = this.exponent*exponent;

        return new LargeNumber(powerMantissa, powerExponent);
    }

    public LargeNumber power(int exponent)
    {
        return power((double)exponent);
    }

    public LargeNumber abs()
    {
        return new LargeNumber(Math.abs(mantissa), exponent);
    }

    public int compareTo(LargeNumber largeNumber)
    {
        //largeNumber.reduce();

        if(exponent == largeNumber.getExponent())
        {
            if(mantissa == largeNumber.getMantissa())
            {
                return 0;
            }
            else
            {
                if(mantissa > largeNumber.getMantissa())
                {
                    return 1;
                }
                else
                {
                    return -1;
                }
            }
        }
        else
        {
            if(exponent > largeNumber.getExponent())
            {
                return 1;
            }
            else
            {
                return -1;
            }
        }
    }

    public void reduce()
    {
        double logTen = Math.log10(mantissa);

        if((mantissa < 1.0)||(mantissa >= 10.0))
        {
            mantissa = Math.pow(10, logTen - Math.floor(logTen));
            exponent = exponent + Math.floor(logTen);
        }
    }
}
```

```java
package functional.results;

//
//  LineProfiler.java
//
//
//  Created by José Gabriel Lira Gomes on 4/24/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import functional.data.Voxel;

public class LineProfiler
{
    private double[][][] dataArray;
    private Voxel initVox;
    private Voxel finalVox;

    private double segmentLength;
    private int numberOfSections;
    private int numberOfPoints;

    private double xVar;
    private double yVar;
    private double zVar;

    private double deltaX;
    private double deltaY;
    private double deltaZ;

    public LineProfiler(double[][][] dataArray, Voxel initVox, Voxel finalVox)
    {
        this.dataArray = dataArray;
        this.initVox = initVox;
        this.finalVox = finalVox;

        getNumberOfSections();
        getDeltas();
    }

    public double[] getProfile()
    {
        double[] profile;
        int i;
        double x, y, z;

        profile = new double[numberOfPoints];

        for(i=0; i<numberOfPoints; i++)
        {
            if(xVar == 0)
            {
                x = (double)initVox.getX();
            }
            else
            {
                x = (double)initVox.getX() + deltaX*(double)i;
            }

            if(yVar == 0)
            {
                y = (double)initVox.getY();
            }
            else
            {
                y = (double)initVox.getY() + deltaY*(double)i;
            }

            if(zVar == 0)
            {
                z = (double)initVox.getZ();
```

```
            }
            else
            {
                z = (double)initVox.getZ() + deltaZ*(double)i;
            }

            profile[i] = getDataValue(x, y, z);
        }

        return profile;
    }

    public double[] getAbsProfile()
    {
        return getAbs(getProfile());
    }

    private void getNumberOfSections()
    {
        this.xVar = (double)finalVox.getX() - (double)initVox.getX();
        this.yVar = (double)finalVox.getY() - (double)initVox.getY();
        this.zVar = (double)finalVox.getZ() - (double)initVox.getZ();

        this.segmentLength = Math.sqrt(xVar*xVar + yVar*yVar + zVar*zVar);

        this.numberOfSections = (int)Math.round(segmentLength);
        this.numberOfPoints = numberOfSections + 1;

        //System.out.println(segmentLength);
        //System.out.println(numberOfSections);
    }

    private void getDeltas()
    {
        this.deltaX = xVar/(double)numberOfSections;
        this.deltaY = yVar/(double)numberOfSections;
        this.deltaZ = zVar/(double)numberOfSections;
    }

    private double getDataValue(double x, double y, double z)
    {
        double dataValue;

        double[] probs;

        double xFloor = Math.floor(x);
        double yFloor = Math.floor(y);
        double zFloor = Math.floor(z);

        int iX = (int)xFloor;
        int iY = (int)yFloor;
        int iZ = (int)zFloor;

        boolean xIsInt = false;
        boolean yIsInt = false;
        boolean zIsInt = false;

        if(xFloor == x) xIsInt = true;
        if(yFloor == y) yIsInt = true;
        if(zFloor == z) zIsInt = true;

        if((xIsInt)&&(yIsInt)&&(zIsInt))
        {
            dataValue = dataArray[iX][iY][iZ];
        }
        else
        {
            if((!xIsInt)&&(yIsInt)&&(zIsInt))
            {
                probs = getOneDProbs(x, xFloor);
```

```
            dataValue = probs[0]*dataArray[iX][iY][iZ] +
                        probs[1]*dataArray[iX+1][iY][iZ];
        }
        else
        {
            if((xIsInt)&&(!yIsInt)&&(zIsInt))
            {
                probs = getOneDProbs(y, yFloor);

                dataValue = probs[0]*dataArray[iX][iY][iZ] +
                            probs[1]*dataArray[iX][iY+1][iZ];
            }
            else
            {
                if((xIsInt)&&(yIsInt)&&(!zIsInt))
                {
                    probs = getOneDProbs(z, zFloor);

                    dataValue = probs[0]*dataArray[iX][iY][iZ] +
                                probs[1]*dataArray[iX][iY][iZ+1];
                }
                else
                {
                    if((!xIsInt)&&(!yIsInt)&&(zIsInt))
                    {
                        probs = getTwoDProbs(x, y, xFloor, yFloor);

                        dataValue = probs[0]*dataArray[iX][iY][iZ] +
                                    probs[1]*dataArray[iX+1][iY][iZ] +
                                    probs[2]*dataArray[iX][iY+1][iZ] +
                                    probs[3]*dataArray[iX+1][iY+1][iZ];
                    }
                    else
                    {
                        if((!xIsInt)&&(yIsInt)&&(!zIsInt))
                        {
                            probs = getTwoDProbs(x, z, xFloor, zFloor);

                            dataValue = probs[0]
                                        *dataArray[iX][iY][iZ] +
                                        probs[1]
                                        *dataArray[iX+1][iY][iZ] +
                                        probs[2]
                                        *dataArray[iX][iY][iZ+1] +
                                        probs[3]
                                        *dataArray[iX+1][iY][iZ+1];
                        }
                        else
                        {
                            if((xIsInt)&&(!yIsInt)&&(!zIsInt))
                            {
                                probs = getTwoDProbs(y, z, yFloor, zFloor);

                                dataValue = probs[0]
                                            *dataArray[iX][iY][iZ] +
                                            probs[1]
                                            *dataArray[iX][iY+1][iZ] +
                                            probs[2]
                                            *dataArray[iX][iY][iZ+1] +
                                            probs[3]
                                            *dataArray[iX][iY+1][iZ+1];
                            }
                            else
                            {
                                probs = getThrDProbs(x, y, z,
                                                     xFloor, yFloor, zFloor);

                                dataValue = probs[0]
                                            *dataArray[iX][iY][iZ] +
                                            probs[1]
                                            *dataArray[iX+1][iY][iZ] +
```

```
                                                probs[2]
                                                *dataArray[iX][iY+1][iZ] +
                                                probs[3]
                                                *dataArray[iX+1][iY+1][iZ] +
                                                probs[4]
                                                *dataArray[iX][iY][iZ+1] +
                                                probs[5]
                                                *dataArray[iX+1][iY][iZ+1] +
                                                probs[6]
                                                *dataArray[iX][iY+1][iZ+1] +
                                                probs[7]
                                                *dataArray[iX+1][iY+1][iZ+1];
                                    }
                                }
                            }
                        }
                    }
                }
            }

        return dataValue;
    }

    private double[] getOneDProbs(double xOne, double xTwo)
    {
        double[] oneDProbs = new double[2];

        oneDProbs[0] = 1.0 - Math.abs(xOne-xTwo);
        oneDProbs[1] = 1.0 - Math.abs(xOne-xTwo-1);

        return oneDProbs;
    }

    private double[] getTwoDProbs(double xOne,
                                  double yOne,
                                  double xTwo,
                                  double yTwo)
    {
        double[] dists = new double[4];
        double distsSum;
        int i;

        double[] twoDProbs = new double[4];

        dists[0] = Math.sqrt((xOne-xTwo)*(xOne-xTwo)
                             + (yOne-yTwo)*(yOne-yTwo));
        dists[1] = Math.sqrt((xOne-xTwo-1)*(xOne-xTwo-1)
                             + (yOne-yTwo)*(yOne-yTwo));
        dists[2] = Math.sqrt((xOne-xTwo)*(xOne-xTwo)
                             + (yOne-yTwo-1)*(yOne-yTwo-1));
        dists[3] = Math.sqrt((xOne-xTwo-1)*(xOne-xTwo-1)
                             + (yOne-yTwo-1)*(yOne-yTwo-1));

        distsSum = dists[0] + dists[1] + dists[2] + dists[3];

        for(i=0; i<4; i++)
        {
            twoDProbs[i] = 1.0 - dists[i]/distsSum;
            twoDProbs[i] = twoDProbs[i]/3.0;
        }

        return twoDProbs;
    }

    private double[] getThrDProbs(double xOne,
                                  double yOne,
                                  double zOne,
                                  double xTwo,
                                  double yTwo,
                                  double zTwo)
    {
```

```
        double[] dists = new double[8];
        double distsSum;
        int i;

        double[] thrDProbs = new double[8];

        dists[0] = Math.sqrt((xOne-xTwo)*(xOne-xTwo)
                             + (yOne-yTwo)*(yOne-yTwo)
                             + (zOne-zTwo)*(zOne-zTwo));
        dists[1] = Math.sqrt((xOne-xTwo-1)*(xOne-xTwo-1)
                             + (yOne-yTwo)*(yOne-yTwo)
                             + (zOne-zTwo)*(zOne-zTwo));
        dists[2] = Math.sqrt((xOne-xTwo)*(xOne-xTwo)
                             + (yOne-yTwo-1)*(yOne-yTwo-1)
                             + (zOne-zTwo)*(zOne-zTwo));
        dists[3] = Math.sqrt((xOne-xTwo-1)*(xOne-xTwo-1)
                             + (yOne-yTwo-1)*(yOne-yTwo-1)
                             + (zOne-zTwo)*(zOne-zTwo));
        dists[4] = Math.sqrt((xOne-xTwo)*(xOne-xTwo)
                             + (yOne-yTwo)*(yOne-yTwo)
                             + (zOne-zTwo-1)*(zOne-zTwo-1));
        dists[5] = Math.sqrt((xOne-xTwo-1)*(xOne-xTwo-1)
                             + (yOne-yTwo)*(yOne-yTwo)
                             + (zOne-zTwo-1)*(zOne-zTwo-1));
        dists[6] = Math.sqrt((xOne-xTwo)*(xOne-xTwo)
                             + (yOne-yTwo-1)*(yOne-yTwo-1)
                             + (zOne-zTwo-1)*(zOne-zTwo-1));
        dists[7] = Math.sqrt((xOne-xTwo-1)*(xOne-xTwo-1)
                             + (yOne-yTwo-1)*(yOne-yTwo-1)
                             + (zOne-zTwo-1)*(zOne-zTwo-1));

        distsSum = dists[0]
                    + dists[1]
                    + dists[2]
                    + dists[3]
                    + dists[4]
                    + dists[5]
                    + dists[6]
                    + dists[7];

        for(i=0; i<7; i++)
        {
            thrDProbs[i] = 1.0 - dists[i]/distsSum;
            thrDProbs[i] = thrDProbs[i]/7.0;
        }

        return thrDProbs;
    }

    private double[] getAbs(double[] dataArray)
    {
        int dataSize = dataArray.length;
        int i;

        for(i=0; i<dataSize; i++)
        {
            dataArray[i] = Math.abs(dataArray[i]);
        }

        return dataArray;
    }
}
```

```java
package estatistica.fitting;

//
//  LinFit.java
//
//
//  Created by José Gabriel Lira Gomes on 1/27/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class LinFit extends GenLinFit
{
    double[] coefs;

    public LinFit(double[] x, double[] y)
    {
        super(x, y, 0, 2);

        coefs = getCoefs();
    }

    public double getSlope()
    {
        return coefs[1];
    }

    public double getOriginOrdinate()
    {
        return coefs[0];
    }
}
```

```
package estatistica.fitting;

//
//  LinRegr.java
//
//
//  Created by José Gabriel Lira Gomes on 1/29/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class LinRegr
{
    double[] x;
    double[] y;

    int numberOfPoints;

    double slope;
    double origOrd;
    double corrCoef;
    double slopeVar;
    double origOrdVar;
    double covar;

    public LinRegr(double[] x, double[] y)
    {
        this.x = x;
        this.y = y;

        this.numberOfPoints = Math.max(x.length, y.length);

        calcRegression();
    }

    public double getSlope()
    {
        return slope;
    }

    public double getOriginOrdinate()
    {
        return origOrd;
    }

    public double getSlopeVariance()
    {
        return slopeVar;
    }

    public double getOriginOrdinateVariance()
    {
        return origOrdVar;
    }

    public double getCovariance()
    {
        return covar;
    }

    private void calcRegression()
    {
        double Sx = 0.0;
        double Sy = 0.0;
        double Sxx = 0.0;
        double Syy = 0.0;
        double Sxy = 0.0;
        double delta;
        int i;
        int N = numberOfPoints;

        for(i=0; i<N; i++)
```

```
        {
            Sx = Sx + x[i];
            Sy = Sy + y[i];
            Sxx = Sxx + x[i]*x[i];
            Syy = Syy + y[i]*y[i];
            Sxy = Sxy + x[i]*y[i];
        }

        delta = (N*Sxx-Sx*Sx);
        slope = (N*Sxy-Sx*Sy)/delta;
        origOrd = (Sxx*Sy-Sx*Sxy)/delta;
        corrCoef = -Sx/Math.sqrt(N*Sxx);
        slopeVar = Sxx/delta;
        origOrdVar = N/delta;
        covar = -Sx/delta;
    }
}
```

```java
package matematica.algebra;

//
//  LUDecomposition.java
//
//
//  Created by José Gabriel Lira Gomes on 2/18/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class LUDecomposition
{
    private double[][] dataArray;
    private Matrix dataMatrix;

    private double[][] LU;

    private int numberOfRows, numberOfColumns;
    private boolean evenExchanges;

    private int[] pivot;

    public LUDecomposition(double[][] dataArray)
    {
        this.dataArray = dataArray;
        this.dataMatrix = new Matrix(dataArray);
        this.LU = dataArray;
        numberOfRows = dataArray.length;
        numberOfColumns = dataArray[0].length;

        luDecompose();
    }

    public LUDecomposition(Matrix dataMatrix)
    {
        this.dataMatrix = dataMatrix;
        this.dataArray = dataMatrix.getMatrixArray();
        this.LU = dataArray;
        numberOfRows = dataArray.length;
        numberOfColumns = dataArray[0].length;

        luDecompose();
    }


    /* ------------------------
        Public Methods
     * ------------------------ */


    public boolean isSingular()
    {
        int i;

        for (i=0; i<numberOfColumns; i++)
        {
            if (LU[i][i] == 0)
            {
                return true;
            }
        }

        return false;
    }


    public Matrix getL()
    {
        Matrix X = new Matrix(numberOfRows,numberOfColumns);
        double[][] L = X.getMatrixArray();
        int i, j;
```

```
    for(i=0; i<numberOfRows; i++)
    {
        for(j=0; j<numberOfColumns; j++)
        {
            if(i > j)
            {
                L[i][j] = LU[i][j];
            }
            else
            {
                if(i == j)
                {
                    L[i][j] = 1.0;
                }
                else
                {
                    L[i][j] = 0.0;
                }
            }
        }
    }

    return X;
}


public Matrix getU()
{
    Matrix X = new Matrix(numberOfColumns,numberOfColumns);
    double[][] U = X.getMatrixArray();
    int i, j;

    for(i=0; i<numberOfColumns; i++)
    {
        for(j=0; j<numberOfColumns; j++)
        {
            if (i <= j)
            {
                U[i][j] = LU[i][j];
            }
            else
            {
                U[i][j] = 0.0;
            }
        }
    }

    return X;
}


public int[] getPivot()
{
    int[] p = new int[numberOfRows];
    int i;

    for(i=0; i<numberOfRows; i++)
    {
        p[i] = pivot[i];
    }

    return p;
}


public double[] getDoublePivot()
{
    double[] vals = new double[numberOfRows];
    int i;
```

```
        for (i=0; i<numberOfRows; i++)
        {
            vals[i] = (double)pivot[i];
        }

        return vals;
    }

    public double getDeterminant()
    {
        if(numberOfRows != numberOfColumns)
        {
            throw new IllegalArgumentException("Matrix must be square.");
        }

        double determinant;

        if(evenExchanges)
        {
            determinant = 1.0;
        }
        else
        {
            determinant = -1.0;
        }

        int i;

        for (i=0; i<numberOfColumns; i++)
        {
            determinant = determinant*LU[i][i];
        }

        return determinant;
    }

    public LargeNumber getHighPrecDeterminant()
    {
        if(numberOfRows != numberOfColumns)
        {
            throw new IllegalArgumentException("Matrix must be square.");
        }

        LargeNumber determinant;

        if(evenExchanges)
        {
            determinant = new LargeNumber(1.0);
        }
        else
        {
            determinant = new LargeNumber(-1.0);
        }

        int i;

        for (i=0; i<numberOfColumns; i++)
        {
            determinant = determinant.multiply(new LargeNumber(LU[i][i]));
        }

        return determinant;
    }

    /*
    public LargeNumber getHighPrecInvDeterminant()
    {
        if(numberOfRows != numberOfColumns)
        {
            throw new IllegalArgumentException("Matrix must be square.");
        }
```

```
        LargeNumber invDeterminant;

        if(evenExchanges)
        {
            invDeterminant = new LargeNumber(1.0);
        }
        else
        {
            invDeterminant = new LargeNumber(-1.0);
        }

        int i;

        for (i=0; i<numberOfColumns; i++)
        {
            invDeterminant =
                invDeterminant.multiply(new LargeNumber(1.0/LU[i][i]));
        }

        return invDeterminant;
    }
    */

    public Matrix getInverse()
    {
        int i, j;
        double[][] identity = new double[numberOfColumns][numberOfColumns];
        Matrix identityMatrix;
        Matrix inverseMatrix;

        for(j=0; j<numberOfColumns; j++)
        {
            for(i=0; i<numberOfRows; i++)
            {
                if(i == j)
                {
                    identity[i][j] = 1.0;
                }
                else
                {
                    identity[i][j] = 0.0;
                }
            }
        }
        identityMatrix = new Matrix(identity);
        inverseMatrix = solve(identityMatrix);

        return inverseMatrix;
    }

    public Matrix solve(Matrix B)
    {
        if(B.getNumberOfRows() != numberOfRows)
        {
            throw new IllegalArgumentException(
                    "Matrix row dimensions must agree.");
        }
        if(isSingular())
        {
            throw new RuntimeException(
                    "Matrix is singular.");
        }

        // Copy right hand side with pivoting
        int nx = B.getNumberOfColumns();
        Matrix Xmat = B.getSubMatrix(pivot,0,nx-1);
        double[][] X = Xmat.getMatrixArray();
        int i, j, k;

        // Solve L*Y = B(pivot,:)
```

```
        for(k=0; k<numberOfColumns; k++)
        {
            for(i=k+1; i<numberOfColumns; i++)
            {
                for (j=0; j<nx; j++)
                {
                    X[i][j] = X[i][j] - X[k][j]*LU[i][k];
                }
            }
        }

        // Solve U*X = Y;
        for(k=numberOfColumns-1; k>=0; k--)
        {
            for(j=0; j<nx; j++)
            {
                X[k][j] = X[k][j]/LU[k][k];
            }
            for(i=0; i<k; i++)
            {
                for(j=0; j<nx; j++)
                {
                    X[i][j] = X[i][j] - X[k][j]*LU[i][k];
                }
            }
        }
        return Xmat;
    }

    private void luDecompose()
    {
        evenExchanges = true;
        pivot = new int[numberOfRows];

        double[] iRow;
        double[] jCol;

        int kmax;
        double croutSum;

        int i, j, k;

        for (i=0; i<numberOfRows; i++)
        {
            pivot[i] = i;
        }

        for (j=0; j<numberOfColumns; j++)
        {
            jCol = dataMatrix.getColumn(j);

            for (i=0; i<numberOfRows; i++)
            {
                iRow = dataMatrix.getRow(i);

                kmax = Math.min(i,j);
                croutSum = 0.0;
                for (k=0; k<kmax; k++)
                {
                    croutSum = croutSum + iRow[k]*jCol[k];
                }

                iRow[j] = iRow[j] - croutSum;
                jCol[i] = jCol[i] - croutSum;
            }

            // Find pivot and exchange if necessary.

            int p = j;
            for (i=j+1; i<numberOfRows; i++)
            {
```

```
                    if (Math.abs(jCol[i]) > Math.abs(jCol[p]))
                    {
                        p = i;
                    }
                }
                if (p != j)
                {
                    for (k=0; k<numberOfColumns; k++)
                    {
                        double t = LU[p][k];
                        LU[p][k] = LU[j][k];
                        LU[j][k] = t;
                    }
                    k = pivot[p];
                    pivot[p] = pivot[j];
                    pivot[j] = k;
                    evenExchanges = !evenExchanges;
                }

                // Compute multipliers.

                if ((j < numberOfRows)&(LU[j][j] != 0.0))
                {
                    for (i=j+1; i<numberOfRows; i++)
                    {
                        LU[i][j] = LU[i][j]/LU[j][j];
                    }
                }
            }
        }
    }
```

```
package functional.data;

//
//  Mask.java
//
//
//  Created by José Gabriel Lira Gomes on 3/12/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class Mask
{
    boolean[][][] maskArray;

    private int maskWidth;
    private int maskHeight;
    private int maskNumberOfSlices;

    public Mask(boolean[][][] maskArray)
    {
        this.maskArray = maskArray;

        getMaskDims();
    }

    public boolean getBooleanValue(int x, int y, int z)
    {
        return maskArray[x][y][z];
    }

    public void setBooleanValue(int x, int y, int z, boolean value)
    {
        this.maskArray[x][y][z] = value;
    }

    public void negateBooleanValue(int x, int y, int z)
    {
        this.maskArray[x][y][z] = !maskArray[x][y][z];
    }

    public void negate()
    {
        int x, y, z;

        for(z=0; z< maskNumberOfSlices; z++)
        {
            for(y=0; y< maskHeight; y++)
            {
                for(x=0; x< maskWidth; x++)
                {
                    maskArray[x][y][z] = !maskArray[x][y][z];
                }
            }
        }
    }

    public boolean[][][] getMaskArray()
    {
        return maskArray;
    }

    public Mask getComplementary()
    {
        int x, y, z;
        boolean[][][] dataArray =
            new boolean[maskWidth][maskHeight][maskNumberOfSlices];

        for(z=0; z< maskNumberOfSlices; z++)
        {
            for(y=0; y< maskHeight; y++)
            {
```

```
            for(x=0; x< maskWidth; x++)
            {
                dataArray[x][y][z] = !maskArray[x][y][z];
            }
        }
    }

    return new Mask(dataArray);
}

public Mask getAnd(Mask otherMask)
{
    int otherMaskWidth = otherMask.getMaskWidth();
    int otherMaskHeight = otherMask.getMaskHeight();
    int otherMaskNumberOfSlices = otherMask.getMaskNumberOfSlices();

    int x, y, z;

    boolean[][][] andMaskArray =
        new boolean[maskWidth][maskHeight][maskNumberOfSlices];
    boolean[][][] otherMaskArray = otherMask.getMaskArray();

    if((otherMaskWidth == maskWidth)
        &&(otherMaskHeight == maskHeight)
        &&(otherMaskNumberOfSlices == maskNumberOfSlices))
    {
        for(z=0; z< maskNumberOfSlices; z++)
        {
            for(y=0; y< maskHeight; y++)
            {
                for(x=0; x< maskWidth; x++)
                {
                    andMaskArray[x][y][z] =
                        maskArray[x][y][z]&&otherMaskArray[x][y][z];
                }
            }
        }
    }

    return new Mask(andMaskArray);
}

public Voxel[] getMaskVoxels()
{
    int x, y, z;
    int numberOfMaskVoxels;
    int counter;
    Voxel[] maskVoxels;

    numberOfMaskVoxels = getNumberOfMaskVoxels();

    maskVoxels = new Voxel[numberOfMaskVoxels];
    counter = 0;

    for(z=0; z< maskNumberOfSlices; z++)
    {
        for(y=0; y< maskHeight; y++)
        {
            for(x=0; x< maskWidth; x++)
            {
                if(maskArray[x][y][z])
                {
                    maskVoxels[counter] = new Voxel(x, y, z);
                    counter = counter + 1;
                }
            }
        }
    }

    return maskVoxels;
}
```

```
public double[][][][] getMaskedData(double[][][][] functionalData)
{
    int dataWidth = functionalData.length;
    int dataHeight = functionalData[0].length;
    int dataNumberOfSlices = functionalData[0][0].length;
    int dataNumberOfInstants = functionalData[0][0][0].length;

    int x, y, z, t;

    if((dataWidth == maskWidth)
        &&(dataHeight == maskHeight)
        &&(dataNumberOfSlices == maskNumberOfSlices))
    {
        for(z=0; z< dataNumberOfSlices; z++)
        {
            for(y=0; y< dataHeight; y++)
            {
                for(x=0; x< dataWidth; x++)
                {
                    if(!maskArray[x][y][z])
                    {
                        for(t=0; t< dataNumberOfInstants; t++)
                        {
                            functionalData[x][y][z][t] = 0.0;
                        }
                    }
                }
            }
        }
    }

    return functionalData;
}

public double[][][][] getComplementaryMaskedData(
                            double[][][][] functionalData)
{
    int dataWidth = functionalData.length;
    int dataHeight = functionalData[0].length;
    int dataNumberOfSlices = functionalData[0][0].length;
    int dataNumberOfInstants = functionalData[0][0][0].length;

    int x, y, z, t;

    if((dataWidth == maskWidth)
        &&(dataHeight == maskHeight)
        &&(dataNumberOfSlices == maskNumberOfSlices))
    {
        for(z=0; z< dataNumberOfSlices; z++)
        {
            for(y=0; y< dataHeight; y++)
            {
                for(x=0; x< dataWidth; x++)
                {
                    if(maskArray[x][y][z])
                    {
                        for(t=0; t< dataNumberOfInstants; t++)
                        {
                            functionalData[x][y][z][t] = 0.0;
                        }
                    }
                }
            }
        }
    }

    return functionalData;
}

public int getNumberOfMaskVoxels()
```

```
{
    int x, y, z;
    int counter = 0;

    for(z=0; z< maskNumberOfSlices; z++)
    {
        for(y=0; y< maskHeight; y++)
        {
            for(x=0; x< maskWidth; x++)
            {
                if(maskArray[x][y][z])
                {
                    counter = counter + 1;
                }
            }
        }
    }

    return counter;
}

public int getNumberOfNonMaskVoxels()
{
    int x, y, z;
    int counter = 0;

    for(z=0; z< maskNumberOfSlices; z++)
    {
        for(y=0; y< maskHeight; y++)
        {
            for(x=0; x< maskWidth; x++)
            {
                if(!maskArray[x][y][z])
                {
                    counter = counter + 1;
                }
            }
        }
    }

    return counter;
}

public double[][][] getDataNumberArray()
{
    int x, y, z;
    double[][][] dataNumberArray =
        new double[maskWidth][maskHeight][maskNumberOfSlices];

    for(z=0; z< maskNumberOfSlices; z++)
    {
        for(y=0; y< maskHeight; y++)
        {
            for(x=0; x< maskWidth; x++)
            {
                if(maskArray[x][y][z])
                {
                    dataNumberArray[x][y][z] = 1.0;
                }
                else
                {
                    dataNumberArray[x][y][z] = 0.0;
                }
            }
        }
    }

    return dataNumberArray;
}

public int getMaskWidth()
```

```
    {
        return maskWidth;
    }

    public int getMaskHeight()
    {
        return maskHeight;
    }

    public int getMaskNumberOfSlices()
    {
        return maskNumberOfSlices;
    }

    private void getMaskDims()
    {
        this.maskWidth = maskArray.length;
        this.maskHeight = maskArray[0].length;
        this.maskNumberOfSlices = maskArray[0][0].length;
    }
}
```

```java
package matematica.algebra;

//
//  Matrix.java
//
//
//  Created by José Gabriel Lira Gomes on 2/26/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class Matrix
{
    private double[][] dataArray;
    private int numberOfRows;
    private int numberOfColumns;

    public Matrix(double[][] dataArray)
    {
        this.dataArray = dataArray;

        numberOfRows = getNumberOfRows();
        numberOfColumns = getNumberOfColumns();
    }

    public Matrix(int numberOfRows, int numberOfColumns)
    {
        this.dataArray = new double[numberOfRows][numberOfColumns];

        this.numberOfRows = numberOfRows;
        this.numberOfColumns = numberOfColumns;
    }

    public Matrix(double[] singleArray, boolean isRow)
    {
        int size = singleArray.length;
        int i;

        if(isRow)
        {
            this.dataArray = new double[1][size];
            for(i=0; i<size; i++)
            {
                this.dataArray[0][i] = singleArray[i];
            }
        }
        else
        {
            this.dataArray = new double[size][1];
            for(i=0; i<size; i++)
            {
                this.dataArray[i][0] = singleArray[i];
            }
        }

        numberOfRows = getNumberOfRows();
        numberOfColumns = getNumberOfColumns();
    }

    public double[][] getMatrixArray()
    {
        return dataArray;
    }

    public double[] getColumn(int columnIndex)
    {
        int i;
        double[] column = new double[numberOfRows];

        for(i=0; i<numberOfRows; i++)
        {
            column[i] = dataArray[i][columnIndex];
```

```
        }

        return column;
    }

    public double[] getRow(int rowIndex)
    {
        int i;
        double[] row = new double[numberOfColumns];

        for(i=0; i<numberOfColumns; i++)
        {
            row[i] = dataArray[rowIndex][i];
        }

        return row;
    }

    public int getNumberOfRows()
    {
        return dataArray.length;
    }

    public int getNumberOfColumns()
    {
        return dataArray[0].length;
    }

    public double getValue(int row, int column)
    {
        return dataArray[row][column];
    }

    public Matrix getSubMatrix(int firstRow,
                               int lastRow,
                               int firstColumn,
                               int lastColumn)
    {
        int i, j;

        double[][] subMatrixArray =
            new double[lastRow-firstRow+1][lastColumn-firstColumn+1];

        for(i=firstRow; i<=lastRow; i++)
        {
            for(j=firstColumn; j<=lastColumn; j++)
            {
                subMatrixArray[i-firstRow][j-firstColumn] = dataArray[i][j];
            }
        }

        return new Matrix(subMatrixArray);
    }

    public Matrix getSubMatrix(int[] rows, int[] columns)
    {
        int i, j;

        double[][] subMatrixArray = new double[rows.length][columns.length];

        for(i=0; i<rows.length; i++)
        {
            for(j=0; j<columns.length; j++)
            {
                subMatrixArray[i][j] = dataArray[rows[i]][columns[j]];
            }
        }

        return new Matrix(subMatrixArray);
    }
```

```
public Matrix getSubMatrix(int[] rows, int firstColumn, int lastColumn)
{
    int i, j;

    double[][] subMatrixArray =
        new double[rows.length][lastColumn-firstColumn+1];

    for(i=0; i<rows.length; i++)
    {
        for(j=firstColumn; j<=lastColumn; j++)
        {
            subMatrixArray[i][j-firstColumn] = dataArray[rows[i]][j];
        }
    }

    return new Matrix(subMatrixArray);
}

public Matrix getSubMatrix(int firstRow, int lastRow, int[] columns)
{
    int i, j;

    double[][] subMatrixArray =
        new double[lastRow-firstRow+1][columns.length];

    for(i=firstRow; i<=lastRow; i++)
    {
        for(j=0; j<columns.length; j++)
        {
            subMatrixArray[i-firstRow][j] = dataArray[i][columns[j]];
        }
    }

    return new Matrix(subMatrixArray);
}

public Matrix getInverse()
{
    return new LUDecomposition(new Matrix(dataArray)).getInverse();
}

public double getDeterminant()
{
    return new LUDecomposition(new Matrix(dataArray)).getDeterminant();
}

public LargeNumber getHighPrecDeterminant()
{
    return new LUDecomposition(new Matrix(dataArray))
                                .getHighPrecDeterminant();
}

/*
public LargeNumber getHighPrecInvDeterminant()
{
    return new LUDecomposition(new Matrix(dataArray))
                                .getHighPrecInvDeterminant();
}
*/

public Matrix getTranspose()
{
    double[][] transposeArray = new double[numberOfColumns][numberOfRows];
    int i, j;

    for(i=0; i<numberOfRows; i++)
    {
        for(j=0; j<numberOfColumns; j++)
        {
            transposeArray[j][i] = dataArray[i][j];
        }
```

```
        }

        return new Matrix(transposeArray);
    }

    public Matrix plus(Matrix otherMatrix)
    {
        double[][] otherMatrixArray = otherMatrix.getMatrixArray();
        double[][] sumArray = new double[numberOfRows][numberOfColumns];
        int i, j;

        for(i=0; i<numberOfRows; i++)
        {
            for(j=0; j<numberOfColumns; j++)
            {
                sumArray[i][j] = dataArray[i][j] + otherMatrixArray[i][j];
            }
        }

        return new Matrix(sumArray);
    }

    public Matrix minus(Matrix otherMatrix)
    {
        otherMatrix = otherMatrix.times(-1.0);

        return plus(otherMatrix);
    }

    public Matrix times(double value)
    {
        double[][] productArray = new double[numberOfRows][numberOfColumns];
        int i, j;

        for(i=0; i<numberOfRows; i++)
        {
            for(j=0; j<numberOfColumns; j++)
            {
                productArray[i][j] = value*dataArray[i][j];
            }
        }

        return new Matrix(productArray);
    }

    public Matrix times(Matrix otherMatrix)
    {
        int numberOfColumns = otherMatrix.getNumberOfColumns();
        double[][] productArray =
                new double[numberOfRows][numberOfColumns];
        int i, j;
        double[] dataRow;
        double[] otherColumn;

        for(i=0; i<numberOfRows; i++)
        {
            dataRow = getRow(i);
            for(j=0; j<numberOfColumns; j++)
            {
                otherColumn = otherMatrix.getColumn(j);
                productArray[i][j] =
                        getInternalProduct(dataRow, otherColumn);
            }
        }

        return new Matrix(productArray);
    }

    public void showArray()
    {
        int i, j;
```

```
        for(i=0; i<numberOfRows; i++)
        {
            for(j=0; j<numberOfColumns; j++)
            {
                System.out.println("matrixArray["+i+"]["+j+"] = "
                    + dataArray[i][j]);
            }
        }
    }

    public void showDiagonalArray()
    {
        int i, j;

        for(i=0; i<numberOfRows; i++)
        {
            for(j=0; j<numberOfColumns; j++)
            {
                if(i==j)
                {
                    System.out.println("matrixArray["+i+"]["+j+"] = "
                        + dataArray[i][j]);
                }
            }
        }
    }

    private double getInternalProduct(double[] vectorOne,
                                      double[] vectorTwo)
    {
        int i;
        double internalProduct = 0;

        for(i=0; i<vectorOne.length; i++)
        {
            internalProduct = internalProduct + vectorOne[i]*vectorTwo[i];
        }

        return internalProduct;
    }
}
```

```java
package fMRIWindows;

//
//  MenuedFrame.java
//
//
//  Created by JGLG on 4/29/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;

public class MenuedFrame extends JFrame implements ActionListener
{
    //
    //  Data Members
    //

    private JMenuBar menuBar;

    private JMenu fileMenu;
    private JMenu editMenu;
    private JMenu dataMenu;


    //
    //  Constructors
    //
    //      menuedFrame()
    //

    public MenuedFrame()
    {
        //java.lang.System.setProperty(
        //          "apple.laf.useScreenMenuBar", "true");

        menuBar = new JMenuBar();
        setJMenuBar(menuBar);
    }


    public void createFileMenu(String menuName)
    {
        fileMenu = new JMenu("Ficheiro");

        fileMenu.add("Novo");

        fileMenu.addActionListener(this);

        menuBar.add(fileMenu);
    }

    public void createEditMenu()
    {
        editMenu = new JMenu("Editar");

        editMenu.add("Cortar");

        editMenu.addActionListener(this);

        menuBar.add(editMenu);
    }

    public void createDataMenu()
    {
        dataMenu = new JMenu("Dados");
```

```
        dataMenu.add("Sequencia");
        dataMenu.add("Espectro");

        dataMenu.addActionListener(this);

        menuBar.add(dataMenu);

    }

    public void actionPerformed(ActionEvent event)
    {
        String itemName;

        itemName = event.getActionCommand();

        if(event.getSource() == fileMenu)
        {
            System.out.println(
                "Escolheu o menu Ficheiro e o item " + itemName);
        }
        if(event.getSource() == editMenu)
        {
            System.out.println(
                "Escolheu o menu Editar e o item " + itemName);
        }
        if(event.getSource() == dataMenu)
        {
            System.out.println(
                "Escolheu o menu Dados e o item " + itemName);
        }
    }
}
```

```java
package estatistica.distributions;

//
//  MultiGaussianDistribution.java
//
//
//  Created by José Gabriel Lira Gomes on 8/1/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import matematica.algebra.LargeNumber;
import matematica.algebra.Matrix;

public class MultiGaussianDistribution
{
    //
    //  Data Members
    //

    private Matrix mean;
    private Matrix covariance;
    private Matrix covarianceI;
    private double covarianceD;
    private LargeNumber covarianceHPD;

    private double invSqrtOfTwoPi;

    private int numberOfDims;

    long startTime;
    long stopTime;
    long elapsedTime;

    //
    //  Constructors
    //
    //      MultiGaussianDistribution(double mean, double covariance)
    //

    public MultiGaussianDistribution(Matrix mean, Matrix covariance)
    {
        this.mean = mean;
        this.covariance = covariance;

        this.covarianceI = covariance.getInverse();
        this.covarianceD = covariance.getDeterminant();
        this.covarianceHPD = covariance.getHighPrecDeterminant();

        invSqrtOfTwoPi = Math.sqrt(1.0/(2.0*Math.PI));

        numberOfDims = mean.getNumberOfRows();
    }


    //
    //  Public Methods
    //
    //      double getProbDensity(double value)
    //      double getCumProb(double value)
    //

    public Matrix getMean()
    {
        return mean;
    }

    public Matrix getCovariance()
    {
        return covariance;
    }
```

```
    public double getMaxProbDensity()
    {
        return getProbDensity(mean);
    }

    public double getProbDensity(Matrix value)
    {
        double probDensity;
        double z;

        Matrix delta;
        Matrix deltaT;

        delta = value.minus(mean);
        deltaT = delta.getTranspose();

        z = ((deltaT.times(covarianceI)).times(delta)).getValue(0, 0);

        probDensity = Math.pow(invSqrtOfTwoPi, numberOfDims)
                        *(Math.exp(-z/2.0))/Math.sqrt(covarianceD);

        return probDensity;
    }

    public LargeNumber getHighPrecProbDensity(Matrix value)
    {
        LargeNumber probDensity;
        LargeNumber exponentialPart;
        LargeNumber invTwoPiToN;
        LargeNumber coefficientPart;
        double z;

        Matrix delta;
        Matrix deltaT;

        delta = value.minus(mean);
        deltaT = delta.getTranspose();

        z = ((deltaT.times(covarianceI)).times(delta)).getValue(0, 0);
        exponentialPart = new LargeNumber(Math.exp(-z/2.0));

        invTwoPiToN = (new LargeNumber(invSqrtOfTwoPi))
                            .power((double)numberOfDims);

        coefficientPart = invTwoPiToN.divide(covarianceHPD.power(0.5));

        probDensity = coefficientPart.multiply(exponentialPart);

        return probDensity;
    }

    public double getMahDistance(Matrix value)
    {
        Matrix delta;
        double mahDistance;
        Matrix deltaT;

        delta = value.minus(mean);
        deltaT = delta.getTranspose();

        mahDistance = Math.sqrt(((deltaT.times(covarianceI))
                                    .times(delta)).getValue(0, 0));

        return mahDistance;
    }
}
```

```
package estatistica.descriptive;

//
//  MultipleArrayAnalyzer.java
//
//
//  Created by Jose Gabriel Lira Gomes on 2/15/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import matematica.algebra.Matrix;

public class MultipleArrayAnalyzer
{
    //
    //  Data Members
    //

    private double[][] dataArray;
    private int numberOfArrays;
    private int arraysSize;

    //
    //  Constructors
    //
    //      DualArrayAnalyzer(double[] firstArray, double[] secndArray)
    //

    public MultipleArrayAnalyzer(double[][] dataArray)
    {
        this.dataArray = dataArray;

        this.numberOfArrays = dataArray.length;
        this.arraysSize = dataArray[0].length;
    }

    public MultipleArrayAnalyzer(Matrix dataArrayMatrix)
    {
        this.dataArray = dataArrayMatrix.getMatrixArray();

        this.numberOfArrays = dataArrayMatrix.getNumberOfRows();
        this.arraysSize = dataArrayMatrix.getNumberOfColumns();
    }


    //
    //  Public Methods
    //
    //      double getSumOfCrossDevProducts()
    //      double getSumOfFirstDevProducts()
    //      double getSumOfSecndDevProducts()
    //

    public Matrix getCovarianceMatrix()
    {
        return new Matrix(getCovarianceArray());
    }

    public double[][] getCovarianceArray()
    {
        double[][] covarianceArray =
                new double[numberOfArrays][numberOfArrays];
        int i, j;
        double[] array;
        double[] arrayOne;
        double[] arrayTwo;

        for(i=0; i<numberOfArrays; i++)
        {
            array = getArray(i);
            covarianceArray[i][i] =
```

```
                new OneDArrayAnalyzer(array).getVariance();
        }

        for(i=0; i<numberOfArrays-1; i++)
        {
            for(j=i+1; j<numberOfArrays; j++)
            {
                arrayOne = getArray(i);
                arrayTwo = getArray(j);

                covarianceArray[i][j] =
                new DualArrayAnalyzer(arrayOne, arrayTwo).getCovariance();
                covarianceArray[j][i] = covarianceArray[i][j];
            }
        }

        return covarianceArray;
    }

    private double[] getArray(int index)
    {
        double[] array = new double[arraysSize];
        int i;

        for(i=0; i<arraysSize; i++)
        {
            array[i] = dataArray[index][i];
        }

        return array;
    }
}
```

```java
package estatistica.descriptive;

//
//  OneDArrayAbsAnalyzer.java
//
//
//  Created by José Gabriel Lira Gomes on 8/1/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import matematica.DataSorter;

public class OneDArrayAbsAnalyzer
{
    private double[] dataArray;
    private int dataSize;
    private double[] sortedArray;

    public OneDArrayAbsAnalyzer(double[] dataArray)
    {
        this.dataArray = dataArray;

        dataSize = getDataSize();
        sortedArray = sortData();
    }

    public double[] getSortedArray()
    {
        return sortedArray;
    }

    /******* GET MEDIAN *******/

    public double getMedian()
    {
        int medianIndex;
        double median;

        double halfSize = dataSize/2;

        medianIndex = (int)Math.floor(halfSize);

        if(isEven(halfSize))
        {
            median = (sortedArray[medianIndex]+sortedArray[medianIndex-1])/2;
        }
        else
        {
            median = sortedArray[medianIndex];
        }

        return median;
    }



    /******* GET MEAN ABSOLUTE DEVIATION *******/

    public double getMeanAbsDev()
    {
        int i;
        double devSum=0.0;
        double s;
        double N;
        double median;
        double mad;

        N = (double)dataSize;

        median = getMedian();
```

```
        for(i=0; i< dataSize; i++)
        {
            s = Math.abs(dataArray[i]-median);
            devSum = devSum + s;
        }

        mad = devSum/N;

        return mad;
    }



    /******* NUMBER IS EVEN *******/

    private boolean isEven(double number)
    {
        double half;
        double nextInt;

        boolean even;

        half = number/2;
        nextInt = Math.round(half);

        if(half == nextInt)
        {
            even = true;
        }
        else
        {
            even = false;
        }

        return even;
    }

    private int getDataSize()
    {
        return dataArray.length;
    }

    private double[] sortData()
    {
        DataSorter sorter;

        sorter = new DataSorter(dataArray);

        return sorter.getSortedArray();
    }
}
```

```
package estatistica.descriptive;

//
//  OneDArrayAnalyzer.java
//
//
//  Created by Jose Gabriel Lira Gomes on 7/31/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import estatistica.distributions.GaussianDistribution;
import estatistica.distributions.Histogram;

public class OneDArrayAnalyzer
{
    private double[] dataArray;

    public OneDArrayAnalyzer(double[] dataArray)
    {
        this.dataArray = dataArray;
    }


    /******* GET MEAN ******/

    public double getMean()
    {
        int N;
        int i;
        double dataSum=0;
        double mean;

        N = dataArray.length;

        for(i=0; i<N; i++)
        {
            dataSum = dataSum + dataArray[i];
        }
        mean = dataSum/(double)N;

        return mean;
    }

    public double getStdDev()
    {
        double stdDev;

        stdDev = Math.sqrt(getVariance());

        return stdDev;
    }

    /******* GET VARIANCE ******/

    public double getVariance()
    {
        int i;
        double dev;
        double devSum=0.0;
        double devSquareSum=0.0;
        int N;
        double dN;
        double mean;
        double variance;

        N = dataArray.length;
        dN = (double)N;

        mean = getMean();

        for(i=0; i< N; i++)
```

```
    {
        dev = dataArray[i]-mean;
        devSum = devSum + dev;
        devSquareSum = devSquareSum + dev*dev;
    }

    variance = (devSquareSum - devSum*devSum/dN)/(dN-1);

    return variance;
}

public double getMaximum()
{
    double maxValue = dataArray[0];

    int N = dataArray.length;

    int i;

    for(i=1; i<N; i++)
    {
        if(dataArray[i] > maxValue)
        {
            maxValue = dataArray[i];
        }
    }

    return maxValue;
}

public double[] getMaxima()
{
    double maximum;
    double[] maxima;

    int N = dataArray.length;

    int i, j;

    int counter = 0;

    maximum = getMaximum();

    for(i=0; i<N; i++)
    {
        if(dataArray[i] == maximum)
        {
            counter = counter + 1;
        }
    }

    maxima = new double[counter];
    j = 0;

    for(i=0; i<N; i++)
    {
        if(dataArray[i] == maximum)
        {
            maxima[j] = dataArray[i];
        }
    }

    return maxima;
}

public double getMinimum()
{
    double minValue = dataArray[0];

    int N = dataArray.length;
```

```
    int i;

    for(i=1; i<N; i++)
    {
        if(dataArray[i] < minValue)
        {
            minValue = dataArray[i];
        }
    }

    return minValue;
}

public double[] getMinima()
{
    double minimum;
    double[] minima;

    int N = dataArray.length;

    int i, j;

    int counter = 0;

    minimum = getMinimum();

    for(i=0; i<N; i++)
    {
        if(dataArray[i] == minimum)
        {
            counter = counter + 1;
        }
    }

    minima = new double[counter];
    j = 0;

    for(i=0; i<N; i++)
    {
        if(dataArray[i] == minimum)
        {
            minima[j] = dataArray[i];
        }
    }

    return minima;
}

public double[] getPositives()
{
    int N = dataArray.length;
    int i, j;
    int numberOfPositives = 0;

    double[] positives;

    for(i=0; i<N; i++)
    {
        if(dataArray[i] > 0)
        {
            numberOfPositives = numberOfPositives + 1;
        }
    }

    positives = new double[numberOfPositives];

    j = 0;

    for(i=0; i<N; i++)
    {
        if(dataArray[i] > 0)
```

```
        {
            positives[j] = dataArray[i];
            j = j + 1;
        }
    }

    return positives;
}

public double[] getNegatives()
{
    int N = dataArray.length;
    int i, j;
    int numberOfNegatives = 0;

    double[] negatives;

    for(i=0; i<N; i++)
    {
        if(dataArray[i] < 0)
        {
            numberOfNegatives = numberOfNegatives + 1;
        }
    }

    negatives = new double[numberOfNegatives];

    j = 0;

    for(i=0; i<N; i++)
    {
        if(dataArray[i] < 0)
        {
            negatives[j] = dataArray[i];
            j = j + 1;
        }
    }

    return negatives;
}

public Histogram getHistogram(int binningMethod)
{
    double[][] histogram;

    double N = (double)dataArray.length;
    double minimum = getMinimum();
    double maximum = getMaximum();
    int numberOfClasses;
    double classesWidth;

    int i, j;

    switch (binningMethod)
    {
        case 0: // Sturge
            {
                numberOfClasses =
                    (int)Math.round(Math.log(N)/Math.log(2)) + 1;
                classesWidth = (maximum-minimum)/numberOfClasses;

                break;
            }

        case 1: // Scott
            {
                double s = getStdDev();

                classesWidth = 3.5*s/Math.cbrt(N);
                numberOfClasses =
                    (int)Math.round((maximum-minimum)/classesWidth);
```

```
                break;
            }
        default:
            {
                numberOfClasses =
                    (int)Math.round(Math.log(N)/Math.log(2)) + 1;
                classesWidth = (maximum-minimum)/numberOfClasses;

                break;
            }
    }

    histogram = new double[2][numberOfClasses];

    for(j=0; j<numberOfClasses; j++)
    {
        histogram[0][j] = minimum + (j+0.5)*classesWidth;
        histogram[1][j] = 0.0;
    }

    for(j=0; j<numberOfClasses; j++)
    {
        histogram[1][j] = 0.0;

        for(i=0; i<N; i++)
        {
            if((dataArray[i] >= minimum + j*classesWidth)
                &&(dataArray[i] < minimum + (j+1)*classesWidth))
            {
                histogram[1][j] = histogram[1][j] + 1.0;
            }
        }
    }

    for(j=0; j<numberOfClasses; j++)
    {
        histogram[1][j] = histogram[1][j]/(N*classesWidth);
    }

    return new Histogram(histogram);
}


public double[][] getProbDensity(int binningMethod, int kernelType)
{
    double[][] histogram;

    double N = (double)dataArray.length;
    double minimum = getMinimum();
    double maximum = getMaximum();
    int numberOfClasses;
    double classesWidth;

    int i, j;

    switch (binningMethod)
    {
        case 0: // Sturge
            {
                numberOfClasses =
                    (int)Math.round(Math.log(N)/Math.log(2)) + 1;
                classesWidth = (maximum-minimum)/numberOfClasses;

                break;
            }

        case 1: // Scott
            {
                double s = getStdDev();
```

```
                    classesWidth = 3.5*s/Math.cbrt(N);
                    numberOfClasses =
                        (int)Math.round((maximum-minimum)/classesWidth);

                    break;
                }
            default:
                {
                    numberOfClasses =
                        (int)Math.round(Math.log(N)/Math.log(2)) + 1;
                    classesWidth = (maximum-minimum)/numberOfClasses;

                    break;
                }
        }

        histogram = getHistogram(binningMethod).getHistogramArray();

        switch (kernelType)
        {
            case 0:
                {
                    return histogram;
                }

            case 1:
                {
                    for(j=0; j<numberOfClasses; j++)
                    {
                        histogram[1][j] = 0.0;

                        for(i=0; i<N; i++)
                        {
                            histogram[1][j] = histogram[1][j]
                            + new GaussianDistribution(dataArray[i],
                            classesWidth).getProbDensity(histogram[0][j]);
                        }
                    }

                    for(j=0; j<numberOfClasses; j++)
                    {
                        histogram[1][j] = histogram[1][j]/(N*classesWidth);
                    }

                    return histogram;
                }
            default:
                {
                    return histogram;
                }
        }
    }
}
```

```java
package estatistica;

//
//  OutlierBrowser.java
//
//
//  Created by Jose Gabriel Lira Gomes on 1/26/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import estatistica.descriptive.OneDArrayAbsAnalyzer;
import estatistica.distributions.GaussianDistribution;

public class OutlierBrowser
{
    private int[] outliersIndex;
    private boolean outlierPresent;

    public OutlierBrowser(double[] data,
                          double thresholdProb,
                          double outliersRatio)
    {
        int dataSize = data.length;

        double[] sortedData;
        double[] bestSortedData;
        double bestMedian;
        double bestMeanAbsDev;
        double prob;
        double outProb;

        OneDArrayAbsAnalyzer arrayAnalyzer;
        OneDArrayAbsAnalyzer bestArrayAnalyzer;
        GaussianDistribution gaussian;

        int[] outliers = new int[dataSize];

        int i;

        arrayAnalyzer = new OneDArrayAbsAnalyzer(data);
        sortedData = arrayAnalyzer.getSortedArray();

        bestSortedData = subSelectFirstGroup(sortedData, 1-outliersRatio);
        bestArrayAnalyzer = new OneDArrayAbsAnalyzer(bestSortedData);
        bestMedian = bestArrayAnalyzer.getMedian();
        bestMeanAbsDev = bestArrayAnalyzer.getMeanAbsDev();

        gaussian = new GaussianDistribution(bestMedian, bestMeanAbsDev);

        for(i=0; i<dataSize; i++)
        {
            prob = gaussian.getCumProb(data[i]);
            outProb = 1 - prob;

            if(outProb<thresholdProb)
            {
                outliers[i] = 1;
                outlierPresent = true;
            }
            else
            {
                outliers[i] = 0;
            }
        }
        if(outlierPresent)
        {
            outliersIndex = getOutliersIndexArray(outliers);
        }
    }
```

```
public int[] getOutliersIndex()
{
    return outliersIndex;
}



public boolean outPresent()
{
    return outlierPresent;
}



private int[] getOutliersIndexArray(int[] dataArray)
{
    int dataSize = dataArray.length;
    int i, j, k;

    int[] outputArray;

    j = 0;

    for(i=0; i<dataSize; i++)
    {
        if(dataArray[i] == 1)
        {
            j = j + 1;
        }
    }

    outputArray = new int[j];

    k = 0;

    for(i=0; i<dataSize; i++)
    {
        if(dataArray[i] == 1)
        {
            outputArray[k] = i;
            k = k + 1;
        }
    }
    //showArray(outputArray);  // <-------------
    return outputArray;
}



/******* SUB SELECT FIRST GROUP *******/

private double[] subSelectFirstGroup(double[] dataArray, double percentage)
{
    int dataSize;
    int selectionSize;
    int i;

    double[] subGroup;

    dataSize = dataArray.length;

    selectionSize = (int)Math.round(percentage*dataSize);

    subGroup = new double[selectionSize];

    for(i=0; i<selectionSize; i++)
    {
        subGroup[i] = dataArray[i];
    }
```

```
    return subGroup;
}



/******* SUB SELECT END GROUP *******/
/*
private double[] subSelectEndGroup(double[] dataArray, double percentage)
{
    int dataSize;
    int selectionSize;
    int firstIndex;
    int i;

    double[] subGroup;

    dataSize = dataArray.length;
    selectionSize = (int)Math.round(percentage*dataSize);
    firstIndex = dataSize - selectionSize;

    subGroup = new double[selectionSize];

    for(i=firstIndex; i<dataSize; i++)
    {
        subGroup[i] = dataArray[i];
    }

    return subGroup;
}
*/


/******* MAXIMUM SLOPE *******/
/*
private double maxSlope(double[] sortedArray)
{
    double slope;
    int dataSize = sortedArray.length;

    slope = sortedArray[dataSize] - sortedArray[0];

    return slope;
}
*/


/******* GET OUTLIERS *******/
/*
private int[] getOutliers(double[] data,
                          int[] oldIndex,
                          double maxSlope,
                          int percentage)
{
    int size = data.length;
    int firstTestIndex = (int)Math.round(percentage*size);
    int initSize = size - firstTestIndex;

    int[] tempOut = new int[initSize];

    int i, j;

    int originalIndex;
    int next;
    int previous;

    boolean first, last;

    double derivative;

    int[] outliers;
```

```
    first = false;
    last = false;

    j=0;

    for(i=firstTestIndex; i<size; i++)
    {
        originalIndex = oldIndex[i];

        next = originalIndex+1;
        previous = originalIndex-1;

        if(next == size)
        {
            last = true;
        }
        if(previous == -1)
        {
            first = true;
        }

        if(first)
        {
            derivative = Math.abs(data[next] - data[originalIndex]);
        }
        else
        {
            if(last)
            {
                derivative = Math.abs(data[originalIndex]
                                      - data[previous]);
            }
            else
            {
                derivative = Math.max(Math.abs(data[next]
                                      - data[originalIndex]),
                                      Math.abs(data[originalIndex]
                                      - data[previous]));
            }
        }

        if(derivative > 5*maxSlope)
        {
            tempOut[j] = originalIndex;
            j=j+1;
        }
    }

    for(i=j; i<initSize; i++)
    {
        tempOut[i] = 0;
    }

    outliers = trimArray(tempOut, j);

    return outliers;
}
*/


/******* TRIM ARRAY *******/
/*
private int[] trimArray(int[] initArray, int trimmedArraySize)
{
    int n = trimmedArraySize;
    int i;
    int[] trimmedSignal = new int[n];

    for(i=0; i<n; i++)
    {
```

```
            trimmedSignal[i] = initArray[i];
        }

        return trimmedSignal;
    }
    */


    /******* SHOW ARRAY *******/
    /*
    private void showArray(int[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */
}
```

```
package functional.methods;

//
//  OverlapAnalysis.java
//
//
//  Created by JGLG on 3/22/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import matematica.difCalculus.Derivator;
import estatistica.descriptive.OneDArrayAnalyzer;
import estatistica.distributions.Histogram;
import functional.data.DesignMatrix;
import functional.data.Mask;
import functional.data.Timeline;
import functional.design.VectorOverlapper;

public class OverlapAnalysis
{
    //
    //  Data Members
    //

    private double[][][] maxInternalProductData;
    private double[][][] phaseData;
    private double[][][] overlapAnalysisResult;

    private double[][][][] imageData;
    private Mask dataMask;
    private DesignMatrix designMatrix;
    private int[] statesID;
    private int lag;
    private double[] modelArray;
    private int maxShift;
    private int order;

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfInstants;

    //
    //  Constructors
    //
    //      overlapAnalysis(double[][][][] imageData,
    //                      Mask dataMask,
    //                      DesignMatrix designMatrix,
    //                      int[] statesID,
    //                      int lag,
    //                      double[] modelArray,
    //                      int maxShift,
    //                      int order)
    //

    public OverlapAnalysis(double[][][][] imageData,
                           Mask dataMask,
                           DesignMatrix designMatrix,
                           int[] statesID,
                           int lag,
                           double[] modelArray,
                           int maxShift,
                           int order)
    {
        this.imageData = imageData;
        this.dataMask = dataMask;
        this.designMatrix = designMatrix;
        this.statesID = statesID;
        this.lag = lag;
        this.modelArray = modelArray;
        this.maxShift = maxShift;
```

```
    this.order = order;

    width = imageData.length;
    height = imageData[0].length;
    numberOfSlices = imageData[0][0].length;
    numberOfInstants = imageData[0][0][0].length;

    performOverlapAnalysis();
}

//
//  Public Methods
//
//      double[][][] getResultData()
//

public double[][][] getOverlapAnalysisResult()
{
    return overlapAnalysisResult;
}

public double[][][] getPhaseData()
{
    return phaseData;
}

//
//  Private Methods
//

private void performOverlapAnalysis()
{
    getOverlapCoefs();

    getOverlapProbs();
}


private void getOverlapCoefs()
{
    maxInternalProductData = new double[width][height][numberOfSlices];
    phaseData = new double[width][height][numberOfSlices];

    Timeline voxelTimeline;

    double[] selectVector;

    int firstStateIndex;
    int secndStateIndex;

    firstStateIndex = statesID[0];
    secndStateIndex = statesID[1];

    double[] timelineData;

    double[] dataArray;

    int x, y, z;

    Derivator modelTimelineDerivator;
    double[] modelFirstDerivative;
    double[] modelSecndDerivative;

    Derivator timelineDerivator;
    double[] timelineFirstDerivative;
    double[] timelineSecndDerivative;

    VectorOverlapper zeroOverlapper;
    VectorOverlapper firstOverlapper;
    VectorOverlapper secndOverlapper;
```

```
double maxZeroOverlapValue;
double maxFirstOverlapValue;
double maxSecndOverlapValue;

double maxZeroOverlapPosition;
double maxFirstOverlapPosition;
double maxSecndOverlapPosition;

selectVector = designMatrix.getSelectVector(firstStateIndex,
                                            secndStateIndex);

modelTimelineDerivator = new Derivator(modelArray);
modelFirstDerivative = modelTimelineDerivator.getFirstDer();
modelSecndDerivative = modelTimelineDerivator.getSecndDer();

for(z=0; z< numberOfSlices; z++)
{
    for(y=0; y< height; y++)
    {
        for(x=0; x< width; x++)
        {
            if(dataMask.getBooleanValue(x, y, z))
            {
                timelineData = extractTimeArray(x, y, z, imageData);

                voxelTimeline = new Timeline(timelineData);

                dataArray = voxelTimeline.getLaggedData(selectVector,
                                                        lag);

                timelineDerivator = new Derivator(dataArray);

                switch(order)
                {
                    default:
                        try
                        {
                            zeroOverlapper =
                            new VectorOverlapper(modelArray,
                                                 dataArray,
                                                 maxShift);
                            maxZeroOverlapValue =
                                zeroOverlapper.getMaxOverlapValue();
                            maxZeroOverlapPosition =
                                zeroOverlapper.getMaxOverlapPosition();

                            maxInternalProductData[x][y][z] =
                                              maxZeroOverlapValue;
                            phaseData[x][y][z] =
                                              maxZeroOverlapPosition;
                        }
                        catch(Exception e)
                        {
                            if(e.getMessage().equals(
                                    "differently sized vectors"))
                            {
                                System.out.println(
                                    "Vectores de diferente tamanho");
                            }
                        }
                        break;
                    case 1:
                        try
                        {
                            timelineFirstDerivative =
                                timelineDerivator.getFirstDer();

                            firstOverlapper =
                            new VectorOverlapper(modelFirstDerivative,
                                                 timelineFirstDerivative,
                                                 maxShift);
```

```
                                maxFirstOverlapValue =
                                    firstOverlapper.getMaxOverlapValue();
                                maxFirstOverlapPosition =
                                    firstOverlapper.getMaxOverlapPosition();

                                maxInternalProductData[x][y][z] =
                                    maxFirstOverlapValue;
                                phaseData[x][y][z] =
                                    maxFirstOverlapPosition;
                            }
                            catch(Exception e)
                            {
                                if(e.getMessage().equals(
                                    "differently sized vectors"))
                                {
                                    System.out.println(
                                        "Vectores de diferente tamanho");
                                }
                            }
                            break;
                        case 2:
                            try
                            {
                                timelineSecndDerivative =
                                    timelineDerivator.getSecndDer();

                                secndOverlapper =
                                    new VectorOverlapper(
                                                modelSecndDerivative,
                                                timelineSecndDerivative,
                                                maxShift);
                                maxSecndOverlapValue =
                                    secndOverlapper.getMaxOverlapValue();
                                maxSecndOverlapPosition =
                                    secndOverlapper.getMaxOverlapPosition();

                                maxInternalProductData[x][y][z] =
                                    maxSecndOverlapValue;
                                phaseData[x][y][z] =
                                    maxSecndOverlapPosition;
                            }
                            catch(Exception e)
                            {
                                if(e.getMessage().equals(
                                    "differently sized vectors"))
                                {
                                    System.out.println(
                                        "Vectores de diferente tamanho");
                                }
                            }
                            break;
                    }
                }
                else
                {
                    maxInternalProductData[x][y][z] = 0.0;
                    phaseData[x][y][z] = 0.0;
                }
            }
        }
    }
}


private void getOverlapProbs()
{
    int x, y, z;

    double[] serialData;
    int counter = 0;
```

```
    OneDArrayAnalyzer dataAnalyzer;
    Histogram dataHistogram;

    double pValue;

    overlapAnalysisResult = new double[width][height][numberOfSlices];


    serialData = new double[dataMask.getNumberOfMaskVoxels()];

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if(dataMask.getBooleanValue(x, y, z))
                {
                    serialData[counter] = maxInternalProductData[x][y][z];
                    counter = counter + 1;
                }
            }
        }
    }

    //new WriteTextNumberFile(serialData,
    //                        "/Users/gabix/Desktop/hist.txt",
    //                        0);

    dataAnalyzer = new OneDArrayAnalyzer(serialData);
    dataHistogram = dataAnalyzer.getHistogram(1);


    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if(dataMask.getBooleanValue(x, y, z))
                {
                    pValue =
                        dataHistogram.getCumProb(
                            maxInternalProductData[x][y][z]);
                }
                else
                {
                    pValue = 0.0;
                }
                overlapAnalysisResult[x][y][z] = pValue;
            }
        }
    }

    resultMaxValue(maxInternalProductData);
    resultMinValue(maxInternalProductData);
}

/******* EXTRACT TIME ARRAY ******/

private double[] extractTimeArray(int x,
                                  int y,
                                  int z,
                                  double dataMatrix[][][][])
{
    int t;
    double[] timeArray = new double[numberOfInstants];

    for(t=0; t< numberOfInstants; t++)
    {
        timeArray[t] = dataMatrix[x][y][z][t];
```

```
    }
    return timeArray;
}



/******* RESULT MAXIMUM VALUE *******/

private void resultMaxValue(double resultMatrix[][][])
{
    int x, y, z;
    int xx=0;
    int yy=0;
    int zz=0;
    double elementValue;
    double maxValue=resultMatrix[0][0][0];

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                elementValue = resultMatrix[x][y][z];
                if(elementValue > maxValue)
                {
                    maxValue = elementValue;
                    xx = x;
                    yy = y;
                    zz = z;
                }
            }
        }
    }

    System.out.println("Max( p(x,y,z) ) = p("
                       + xx + "," + yy + "," + zz + ") = "
                       + overlapAnalysisResult[xx][yy][zz]);

}



/******* RESULT MINIMUM VALUE *******/

private void resultMinValue(double resultMatrix[][][])
{
    int x, y, z;
    int xx=0;
    int yy=0;
    int zz=0;
    double elementValue;
    double minValue=resultMatrix[0][0][0];

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                elementValue = resultMatrix[x][y][z];
                if(elementValue < minValue)
                {
                    minValue = elementValue;
                    xx = x;
                    yy = y;
                    zz = z;
                }
            }
        }
    }
```

```
            System.out.println("Min( p(x,y,z) ) = p("
                               + xx + "," + yy + "," + zz + ") = "
                               + overlapAnalysisResult[xx][yy][zz]);

        }


        /******* SHOW ARRAY ******

        private void showArray(double[] array)
        {
            int size = array.length;
            int i;

            for(i=0; i<size; i++)
            {
                System.out.println(array[i]);
            }
        }
        */
}
```

```java
package fMRIWindows.methodParams;

//
//  ParamWindowAmplitude.java
//
//
//  Created by José Gabriel Lira Gomes on 9/16/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.Color;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

import dataUtils.ComboBoxItems;

import fMRIWindows.MainWindow;
import fMRIWindows.MenuedFrame;
import fMRIWindows.params.ParamWindowVoxel;
//import fMRIWindows.viewers.ViewerWindowModelArray;
import functional.data.Voxel;

public class ParamWindowAmplitude extends MenuedFrame implements ItemListener
{
    //
    //  Data Members
    //

    private String[] statesAcronyms;
    private MainWindow parentWindow;

    private Container mainContainer;

    private JPanel setFirstStatePanel;
    private JLabel firstStateTitle;
    private JComboBox firstStateComboBox;
    private int firstState;

    private JPanel setSecndStatePanel;
    private JLabel secndStateTitle;
    private JComboBox secndStateComboBox;
    private int secndState;

    private int[] selectedStates = new int[2];

    private JPanel setLagPanel;
    private JLabel lagParamTitle;
    private JTextField lagParamTextField;
    private int lagValue;

    private JPanel modelTypePanel;
    private JLabel modelParamTitle;
    private String[] modelBoxOptions = {"Bloco", "HRF", "Voxel"};
    private JComboBox modelComboBox;
    private int modelType;  // 0 - Block; 1 - HRF;

    private JPanel submitParamsPanel;
    public JButton submitAmplitudeParamsButton;

    private short[] imageDims;
    private int width;
```

```
private int height;
private int numberOfSlices;

private ComboBoxItems voxelsGroups;
private ParamWindowVoxel voxelWindow;
private JButton selectVoxelButton;
private Voxel modelVoxel;

private int windowWidth = 200;
private int windowHeight = 250;
private int windowXPosition = 0;
private int windowYPosition = 21;

//
//  Constructors
//
//      paramWindowAmplitude()
//

public ParamWindowAmplitude(String[] statesAcronyms,
                            MainWindow parentWindow)
{
    this.statesAcronyms = statesAcronyms;
    this.parentWindow = parentWindow;

    createGUI();
}

public int getFirstState()
{
    return firstState;
}

public int getSecndState()
{
    return secndState;
}

public int[] getSelectedStates()
{
    selectedStates[0] = getFirstState();
    selectedStates[1] = getSecndState();

    return selectedStates;
}

public int getLag()
{
    return lagValue;
}

public int getModelType()
{
    return modelType;
}

public Voxel getModelVoxel()
{
    return modelVoxel;
}

//
//  Public Methods
//
//      void actionPerformed(ActionEvent event)
//

/******* LISTEN BUTTON ACTION EVENT *******/

public void actionPerformed(ActionEvent event)
{
```

```
        JButton clickedButton = (JButton) event.getSource();

        if(clickedButton == submitAmplitudeParamsButton)
        {
            this.setVisible(false);

            try
            {
                firstState = firstStateComboBox.getSelectedIndex();
                secndState = secndStateComboBox.getSelectedIndex();
                lagValue = Integer.parseInt(lagParamTextField.getText());
                modelType = modelComboBox.getSelectedIndex();

                if(firstState == secndState)
                {
                    JOptionPane.showMessageDialog(this,
                            "Os estados nao podem ser iguais",
                            "Erro de design",
                            JOptionPane.ERROR_MESSAGE);
                    this.setVisible(true);
                }
                else
                {
                    parentWindow.setAmplitudeSelectedStates(
                                                getSelectedStates());
                    parentWindow.setAmplitudeLag(getLag());
                    parentWindow.setAmplitudeModelType(getModelType());
                    parentWindow.setAmplitudeModelVoxel(getModelVoxel());
                    parentWindow.enableCalculateButton();
                    parentWindow.setVisible(true);

                    //new ViewerWindowModelArray(
                    //          parentWindow.getAmplitudeModelArray());
                }
            }
            catch(NumberFormatException numberFormatException)
            {
                JOptionPane.showMessageDialog(this,
                        "Tem que inserir um inteiro",
                        "Erro de formato",
                        JOptionPane.ERROR_MESSAGE);
                this.setVisible(true);
            }
        }

        if(clickedButton == selectVoxelButton)
        {
            this.setVisible(true);
            voxelWindow.setVisible(false);
            modelVoxel = voxelWindow.getVoxel();
        }
    }

    public void itemStateChanged(ItemEvent event)
    {
        int index;

        if(event.getStateChange() == ItemEvent.SELECTED)
        {
            index = modelComboBox.getSelectedIndex();

            if(index == 2)
            {
                this.setVisible(false);

                imageDims = parentWindow.getImageDims();
                width = imageDims[1];
                height = imageDims[2];
                numberOfSlices = imageDims[3];

                voxelsGroups = new ComboBoxItems();
```

```
                voxelsGroups.setSequentialItems(0, width);
                voxelsGroups.setSequentialItems(1, height);
                voxelsGroups.setSequentialItems(2, numberOfSlices);

                voxelWindow = new ParamWindowVoxel(voxelsGroups);
                selectVoxelButton = voxelWindow.acceptButton;
                selectVoxelButton.addActionListener(this);
            }
        }
    }

    private void createGUI()
    {
        mainContainer = getContentPane();
        mainContainer.setLayout(new GridLayout(5, 1));
        mainContainer.setBackground(Color.white);

        setFirstStatePanel = new JPanel();
        setSecndStatePanel = new JPanel();
        setLagPanel = new JPanel();
        submitParamsPanel = new JPanel();
        modelTypePanel = new JPanel();

        firstStateTitle = new JLabel("Estado 1");
        firstStateComboBox = new JComboBox(statesAcronyms);

        secndStateTitle = new JLabel("Estado 2");
        secndStateComboBox = new JComboBox(statesAcronyms);

        lagParamTitle = new JLabel("Atraso");
        lagParamTextField = new JTextField(2);

        modelParamTitle = new JLabel("Modelo");
        modelComboBox = new JComboBox(modelBoxOptions);
        modelComboBox.addItemListener(this);

        submitAmplitudeParamsButton = new JButton("Aceitar");

        setFirstStatePanel.add(firstStateTitle);
        setFirstStatePanel.add(firstStateComboBox);
        setSecndStatePanel.add(secndStateTitle);
        setSecndStatePanel.add(secndStateComboBox);
        setLagPanel.add(lagParamTitle);
        setLagPanel.add(lagParamTextField);
        modelTypePanel.add(modelParamTitle);
        modelTypePanel.add(modelComboBox);
        submitParamsPanel.add(submitAmplitudeParamsButton);

        mainContainer.add(setFirstStatePanel);
        mainContainer.add(setSecndStatePanel);
        mainContainer.add(setLagPanel);
        mainContainer.add(modelTypePanel);
        mainContainer.add(submitParamsPanel);

        submitAmplitudeParamsButton.setEnabled(true);
        submitAmplitudeParamsButton.addActionListener(this);
        submitAmplitudeParamsButton.addActionListener(parentWindow);

        setSize(windowWidth, windowHeight);
        setLocation(windowXPosition, windowYPosition);
        setTitle("Amplitude");
        setVisible(true);
    }
}
```

```java
package fMRIWindows.segmenters;

//
//  ParamWindowClusterSampler.java
//
//
//  Created by JGLG on 3/16/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import imageUtils.ImagePanel;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.JTextField;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import fMRIWindows.MainWindow;
import fMRIWindows.MenuedFrame;
import functional.data.Cluster;
import functional.data.DesignMatrix;
import functional.data.Mask;
import functional.data.Timeline;
import functional.data.Voxel;

public class ParamWindowClusterSampler extends MenuedFrame
                implements ChangeListener, MouseListener, MouseMotionListener
{
    //
    //  Data Members
    //

    private double[][][][] imageData;
    private int zoomFactor;
    private boolean withTumor;
    private MainWindow parentWindow;

    private String[] fourTissueAcronyms = {"fnd", "brn", "cza", "csf"};
    private String[] fiveTissueAcronyms = {"fnd", "brn", "cza", "csf", "tmr"};

    private String[] tissueNames = {"Fundo", "Mat branca",
                                    "Mat cinzenta", "csf", "Tumor"};

    private boolean[][][] sampleMask;
    private Voxel[] sampleVoxels;

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfInstants;

    private double[][] instantImageSliceData;
    private boolean[][] sampleMaskSliceData;
```

```
    private int numberOfTissues;
    private int tissueCounter = 0;

    private int minNumberOfSampleVoxels = 10;

    private int xInitialValue;
    private int yInitialValue;
    private int zInitialValue;

    private int x;
    private int y;
    private int z;

    private int instant;

    private int middleZ;

    private int windowWidth;
    private int windowHeight;
    private int windowXPosition = 0;
    private int windowYPosition = 21;

    private Container viewerContainer;
    private JPanel zSliderPanel;
    private ImagePanel imagePanel;
    private JPanel paramsPanel;
    private JPanel dataPanel;
    private JPanel tSliderPanel;
    private JPanel sampleAcceptPanel;

    private JSlider timeSlider;
    private JButton sampleAcceptButton;
    private JSlider zSlider;

    private JPanel tissueTitlePanel;
    private JPanel tissueComboPanel;
    private JPanel stateTitlePanel;
    private JPanel stateComboPanel;
    private JPanel lagTitlePanel;
    private JPanel lagParamPanel;

    private JLabel tissueTitle;
    private JComboBox tissueComboBox;
    private int tissueIndex;
    private JLabel stateTitle;
    private JComboBox stateComboBox;
    private int state;
    private JLabel lagParamTitle;
    private JTextField lagParamTextField;
    private int lag;

    private double[][][][] imageDataSubset;
    private Cluster[] sampleClusters;

    private JLabel voxelPosition;
    private JLabel voxelInstant;

    private int voxelCounter;

    private int mouseX = 0;
    private int mouseY = 0;

    //
    //  Constructors
    //
    //      ParamWindowClusterSampler(double[][][][] imageData,
    //                                int zoomFactor,
    //                                boolean withTumor,
    //                                MainWindow parentWindow)
    //
```

```
public ParamWindowClusterSampler(double[][][][] imageData,
                                 int zoomFactor,
                                 boolean withTumor,
                                 MainWindow parentWindow)
{
    this.imageData = imageData;
    this.zoomFactor = zoomFactor;
    this.withTumor = withTumor;
    this.parentWindow = parentWindow;

    getImageDims();
    getNumberOfTissues();
    initSampleClusters();
    initSampleMask();
    getInitialValues();

    createGUI();

    imagePanel.addMouseListener(this);
    imagePanel.addMouseMotionListener(this);
}

/******* LISTEN TO SLIDER CHANGE EVENT *******/

public void stateChanged(ChangeEvent event)
{
    this.instant = timeSlider.getValue();

    this.z = zSlider.getValue();

    voxelInstant.setText("t = " + instant);

    instantImageSliceData = getInstantSlice(imageData, z, instant);
    sampleMaskSliceData = getSlice(sampleMask, z);

    imagePanel.setImageData(instantImageSliceData, 0);
    imagePanel.setBooleanImageData(sampleMaskSliceData, 1);
}

/******* LISTEN TO BUTTON ACTION EVENT *******/

public void actionPerformed(ActionEvent event)
{
    JButton clickedButton = (JButton) event.getSource();

    if(clickedButton == sampleAcceptButton)
    {
        this.setVisible(false);

        try
        {
            tissueIndex = tissueComboBox.getSelectedIndex();
            state = stateComboBox.getSelectedIndex();
            lag = Integer.parseInt(lagParamTextField.getText());
        }
        catch(NumberFormatException numberFormatException)
        {
            JOptionPane.showMessageDialog(this,
                            "Tem que inserir um inteiro",
                            "Erro de formato",
                            JOptionPane.ERROR_MESSAGE);
            this.setVisible(true);
        }

        imageDataSubset = getDataSubset(state, lag);
        sampleVoxels = new Mask(sampleMask).getMaskVoxels();
        sampleClusters[tissueCounter] = new Cluster(imageDataSubset,
                                                    sampleVoxels);

        tissueCounter = tissueCounter + 1;
```

```
            if(tissueCounter > 0)
            {
                tissueComboBox.setEnabled(false);
                stateComboBox.setEnabled(false);
                lagParamTextField.setEnabled(false);
                initSampleMask();
                sampleMaskSliceData = getSlice(sampleMask, z);
                imagePanel.setBooleanImageData(sampleMaskSliceData, 1);
                sampleAcceptButton.setEnabled(false);
                voxelCounter = 0;
                if(tissueCounter < numberOfTissues)
                {
                    setTitle("Amostra de " + tissueNames[tissueCounter]);
                }
            }

            if(tissueCounter == numberOfTissues-1)
            {
                sampleAcceptButton.setText("Finalizar");
            }

            if(tissueCounter == numberOfTissues)
            {
                System.out.println("Tecido a segmentar: "
                                            + tissueNames[tissueIndex]);
                System.out.println();

                parentWindow.setUsedSegmentData(imageDataSubset);
                parentWindow.setSampleClusters(sampleClusters);
                parentWindow.segmentImageTissue(tissueIndex);
                parentWindow.enableTumorSelector();
                parentWindow.enableClustersSegmentButton();
                parentWindow.setVisible(true);
            }
            else
            {
                this.setVisible(true);
            }
        }
    }

    public void paint(Graphics g)
    {
        super.paint(g);
    }


    public void mouseClicked(MouseEvent me)
    {
        mouseX = me.getX();
        mouseY = me.getY();

        if(me.isMetaDown())
        {

        }
        else
        {
            if(me.isAltDown())
            {
                if(zoomFactor>1)
                {
                    zoomFactor = zoomFactor/2;
                    imagePanel.setZoomFactor(zoomFactor);
                }
            }
            else
            {
                if(me.isControlDown())
                {
```

```
                zoomFactor = zoomFactor*2;
                imagePanel.setZoomFactor(zoomFactor);
            }
            else
            {
                updateSampleMask();
            }
        }
    }

}

public void mouseEntered(MouseEvent me)
{

}


public void mouseExited(MouseEvent me)
{

}


public void mousePressed(MouseEvent me)
{

}


public void mouseReleased(MouseEvent me)
{

}


public void mouseDragged(MouseEvent me)
{
    mouseX = me.getX();
    mouseY = me.getY();

    updateSampleMask();
}


public void mouseMoved(MouseEvent me)
{

}

private void getImageDims()
{
    this.width = imageData.length;
    this.height = imageData[0].length;
    this.numberOfSlices = imageData[0][0].length;
    this.numberOfInstants = imageData[0][0][0].length;
}

private void getNumberOfTissues()
{
    if(withTumor)
    {
        this.numberOfTissues = 5;
    }
    else
    {
        this.numberOfTissues = 4;
    }
}

private void initSampleClusters()
```

```
{
    this.sampleClusters = new Cluster[numberOfTissues];
}

private void initSampleMask()
{
    sampleMask = new boolean[width][height][numberOfSlices];
    int x, y, z;

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                sampleMask[x][y][z] = false;
            }
        }
    }
}

private void getInitialValues()
{
    this.xInitialValue = (int)Math.round(width/2);
    this.yInitialValue = (int)Math.round(height/2);
    this.zInitialValue = (int)Math.round(numberOfSlices/2);

    this.x = xInitialValue;
    this.y = yInitialValue;
    this.z = zInitialValue;

    middleZ = Math.round(numberOfSlices/2) -1;
}

private void createGUI()
{
    windowWidth = 85 + zoomFactor*width + 85;
    windowHeight = zoomFactor*height + 85 + 20;

    createZSliderPanel();
    createImagePanel();
    createParamsPanel();
    createDataPanel();
    createTSliderPanel();
    createSampleAcceptPanel();

    addPanelsToContainer();

    setSize(windowWidth, windowHeight);
    setLocation(windowXPosition, windowYPosition);
    setTitle("Amostra de " + tissueNames[tissueCounter]);
    setVisible(true);
}

private void createZSliderPanel()
{
    zSliderPanel = new JPanel(new BorderLayout());
    zSliderPanel.setBorder(BorderFactory.createTitledBorder("Z"));
    zSlider = createZSlider(middleZ);
    zSliderPanel.add(zSlider, BorderLayout.CENTER);
}

private void createImagePanel()
{
    instantImageSliceData = getInstantSlice(imageData, middleZ, 0);
    sampleMaskSliceData = getSlice(sampleMask, middleZ);

    imagePanel = new ImagePanel(instantImageSliceData, zoomFactor, false);
    imagePanel.addBooleanLayerImageData(sampleMaskSliceData, true, 1);
}
```

```
private void createParamsPanel()
{
    paramsPanel = new JPanel(new GridLayout(9,1));

    tissueTitlePanel = new JPanel();
    tissueTitle = new JLabel("Tecido");
    tissueComboPanel = new JPanel();
    if(withTumor)
    {
        tissueComboBox = new JComboBox(fiveTissueAcronyms);
    }
    else
    {
        tissueComboBox = new JComboBox(fourTissueAcronyms);
    }

    stateTitlePanel = new JPanel();
    stateTitle = new JLabel("Estado");
    stateComboPanel = new JPanel();
    stateComboBox = new JComboBox(parentWindow.getStatesAcronyms());

    lagTitlePanel = new JPanel();
    lagParamTitle = new JLabel("Atraso");
    lagParamPanel = new JPanel();
    lagParamTextField = new JTextField(2);

    lagParamTextField.setText("0");

    tissueTitlePanel.add(tissueTitle);
    tissueComboPanel.add(tissueComboBox);
    stateTitlePanel.add(stateTitle);
    stateComboPanel.add(stateComboBox);
    lagTitlePanel.add(lagParamTitle);
    lagParamPanel.add(lagParamTextField);

    paramsPanel.add(tissueTitlePanel);
    paramsPanel.add(tissueComboPanel);
    paramsPanel.add(stateTitlePanel);
    paramsPanel.add(stateComboPanel);
    paramsPanel.add(lagTitlePanel);
    paramsPanel.add(lagParamPanel);
}

private void createDataPanel()
{
    dataPanel = new JPanel(new GridLayout(2,1));
    dataPanel.setPreferredSize(new Dimension(85,85));
    voxelPosition = new JLabel("("+ x + ", " + y + ", " + z + ")");
    voxelInstant = new JLabel("t = " + instant);
    dataPanel.add(voxelPosition);
}

private void createTSliderPanel()
{
    tSliderPanel = new JPanel(new BorderLayout());
    tSliderPanel.setBorder(BorderFactory.createTitledBorder("Tempo"));
    timeSlider = createTSlider(0);
    tSliderPanel.add(timeSlider, BorderLayout.CENTER);
    if(numberOfInstants == 1)
    {
        timeSlider.setEnabled(false);
    }
}

private void createSampleAcceptPanel()
{
    sampleAcceptPanel = new JPanel(new GridLayout(2,1));
    sampleAcceptPanel.setPreferredSize(new Dimension(85,85));

    sampleAcceptButton = new JButton("Seguinte >");
    sampleAcceptButton.setEnabled(false);
```

```
        sampleAcceptButton.addActionListener(this);

        sampleAcceptPanel.add(sampleAcceptButton);
    }

    private void addPanelsToContainer()
    {
        viewerContainer = getContentPane();
        viewerContainer.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();

        c.fill = GridBagConstraints.BOTH;

        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewerContainer.add(zSliderPanel, c);

        c.gridx = 1;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 1;
        c.weighty = 1;

        viewerContainer.add(imagePanel, c);

        c.gridx = 2;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewerContainer.add(paramsPanel, c);

        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewerContainer.add(dataPanel, c);

        c.gridx = 1;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewerContainer.add(tSliderPanel, c);

        c.gridx = 2;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewerContainer.add(sampleAcceptPanel, c);
    }

    /******* CREATE T SLIDER *******/

    private JSlider createTSlider(int initialValue)
```

```
{
    JSlider tSlider = new JSlider();

    tSlider.setOrientation(JSlider.HORIZONTAL);
    tSlider.setPaintLabels(true);
    tSlider.setPaintTicks(true);
    tSlider.setMinimum(0);
    tSlider.setMaximum(numberOfInstants-1);
    tSlider.setValue(initialValue);
    tSlider.setMajorTickSpacing(50);
    tSlider.setMinorTickSpacing(25);

    tSlider.addChangeListener(this);

    return tSlider;
}

/******* CREATE Z SLIDER *******/

private JSlider createZSlider(int initialValue)
{
    JSlider zSlider = new JSlider();

    zSlider.setOrientation(JSlider.VERTICAL);
    zSlider.setPaintLabels(true);
    zSlider.setPaintTicks(true);
    zSlider.setMinimum(0);
    zSlider.setMaximum(numberOfSlices-1);
    zSlider.setValue(initialValue);
    zSlider.setMajorTickSpacing(10);
    zSlider.setMinorTickSpacing(5);

    zSlider.addChangeListener(this);

    return zSlider;
}

private void updateSampleMask()
{
    x = (int)Math.floor(mouseX/zoomFactor);
    y = height - 1 - (int)Math.floor(mouseY/zoomFactor);
    if((x<width)&&(x>=0)&&(y<height)&&(y>=0))
    {
        voxelPosition.setText("("+ x + ", " + y + ", " + z + ")");

        sampleMask[x][y][z] = !sampleMask[x][y][z];
        sampleMaskSliceData = getSlice(sampleMask, z);
        imagePanel.setBooleanImageData(sampleMaskSliceData, 1);

        voxelCounter = new Mask(sampleMask).getNumberOfMaskVoxels();

        if(voxelCounter >= minNumberOfSampleVoxels)
        {
            sampleAcceptButton.setEnabled(true);
        }
        else
        {
            sampleAcceptButton.setEnabled(false);
        }
    }
}

private double[][] getInstantSlice(double[][][][] volumeDataArray,
                                   int z,
                                   int t)
{
    int x, y;
    double[][] sliceDataArray = new double[width][height];

    for(y=0; y<height; y++)
    {
```

```
        for(x=0; x<width; x++)
        {
            sliceDataArray[x][y] = volumeDataArray[x][y][z][t];
        }
    }

    return sliceDataArray;
}

private boolean[][] getSlice(boolean[][][] volumeDataArray, int z)
{
    int x, y;
    boolean[][] sliceDataArray = new boolean[width][height];

    for(y=0; y<height; y++)
    {
        for(x=0; x<width; x++)
        {
            sliceDataArray[x][y] = volumeDataArray[x][y][z];
        }
    }

    return sliceDataArray;
}

private double[][][][] getDataSubset(int state, int lag)
{
    DesignMatrix designMatrix;
    int numberOfStateOccurrences;
    double[] selectVector;
    double[][][][] dataSubset;

    int x, y, z, t;
    double[] timelineData;
    Timeline voxelTimeline;
    double[] dataArray;

    designMatrix = parentWindow.getDesignMatrix();
    numberOfStateOccurrences = designMatrix.getNumberOfOccurrences(state);
    selectVector = designMatrix.getStateVector(state);
    dataSubset =
    new double[width][height][numberOfSlices][numberOfStateOccurrences];

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                timelineData = extractTimeArray(x, y, z, imageData);

                voxelTimeline = new Timeline(timelineData);

                dataArray = voxelTimeline.getLaggedData(selectVector,
                                                        ag);

                for(t=0; t< numberOfStateOccurrences; t++)
                {
                    dataSubset[x][y][z][t] = dataArray[t];
                }
            }
        }
    }

    return dataSubset;
}

private double[] extractTimeArray(int x,
                                  int y,
                                  int z,
                                  double dataMatrix[][][][])
```

```
    {
        int t;
        double[] timeArray = new double[numberOfInstants];

        for(t=0; t< numberOfInstants; t++)
        {
            timeArray[t] = dataMatrix[x][y][z][t];
        }
        return timeArray;
    }

    /******* SHOW ARRAY ******

    private void showArray(double[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */
}
```

```java
package fMRIWindows.methodParams;

//
//  ParamWindowCorr.java
//
//
//  Created by Jose Gabriel Lira Gomes on 1/3/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.Color;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

import dataUtils.ComboBoxItems;

import fMRIWindows.MainWindow;
import fMRIWindows.MenuedFrame;
import fMRIWindows.params.ParamWindowVoxel;
//import fMRIWindows.viewers.ViewerWindowModelArray;
import functional.data.Voxel;

public class ParamWindowCorr extends MenuedFrame implements ItemListener
{
    //
    //  Data Members
    //

    private String[] statesAcronyms;
    private MainWindow parentWindow;

    private Container mainContainer;

    private JPanel setFirstStatePanel;
    private JLabel firstStateTitle;
    private JComboBox firstStateComboBox;
    private int firstState;

    private JPanel setSecndStatePanel;
    private JLabel secndStateTitle;
    private JComboBox secndStateComboBox;
    private int secndState;

    private int[] selectedStates = new int[2];

    private JPanel setLagPanel;
    private JLabel lagParamTitle;
    private JTextField lagParamTextField;
    private int lag;

    private JPanel modelTypePanel;
    private JLabel modelParamTitle;
    private String[] modelBoxOptions = {"Bloco", "HRF", "Voxel"};
    private JComboBox modelComboBox;
    private int modelType;  // 0 - Block; 1 - HRF; 2 - Voxel;

    private short[] imageDims;
    private int width;
    private int height;
    private int numberOfSlices;
```

```
private ComboBoxItems voxelsGroups;
private ParamWindowVoxel voxelWindow;
private JButton selectVoxelButton;
private Voxel modelVoxel;

private JPanel submitParamsPanel;
public JButton submitCorrParamsButton;

private int windowWidth = 200;
private int windowHeight = 250;
private int windowXPosition = 0;
private int windowYPosition = 21;

//
//  Constructors
//
//      paramWindowCorr()
//

public ParamWindowCorr(String[] statesAcronyms, MainWindow parentWindow)
{
    this.statesAcronyms = statesAcronyms;
    this.parentWindow = parentWindow;

    createGUI();
}

public int getFirstState()
{
    return firstState;
}

public int getSecndState()
{
    return secndState;
}

public int[] getSelectedStates()
{
    selectedStates[0] = getFirstState();
    selectedStates[1] = getSecndState();

    return selectedStates;
}

public int getLag()
{
    return lag;
}

public int getModelType()
{
    return modelType;
}

public Voxel getModelVoxel()
{
    return modelVoxel;
}

//
//  Public Methods
//
//      void actionPerformed(ActionEvent event)
//

/******* LISTEN BUTTON ACTION EVENT *******/

public void actionPerformed(ActionEvent event)
{
    JButton clickedButton = (JButton) event.getSource();
```

```
        if(clickedButton == submitCorrParamsButton)
        {
            this.setVisible(false);

            try
            {
                firstState = firstStateComboBox.getSelectedIndex();
                secndState = secndStateComboBox.getSelectedIndex();
                lag = Integer.parseInt(lagParamTextField.getText());
                modelType = modelComboBox.getSelectedIndex();

                if(firstState == secndState)
                {
                    JOptionPane.showMessageDialog(this,
                                    "Os estados nao podem ser iguais",
                                    "Erro de design",
                                    JOptionPane.ERROR_MESSAGE);
                    this.setVisible(true);
                }
                else
                {
                    parentWindow.setCorrSelectedStates(getSelectedStates());
                    parentWindow.setCorrLag(getLag());
                    parentWindow.setCorrModelType(getModelType());
                    parentWindow.setCorrModelVoxel(getModelVoxel());
                    parentWindow.enableCalculateButton();
                    parentWindow.setVisible(true);

                    //new ViewerWindowModelArray(
                    //          parentWindow.getCorrModelArray());
                }
            }
            catch(NumberFormatException numberFormatException)
            {
                JOptionPane.showMessageDialog(this,
                                    "Tem que inserir um inteiro",
                                    "Erro de formato",
                                    JOptionPane.ERROR_MESSAGE);
                this.setVisible(true);
            }
        }

        if(clickedButton == selectVoxelButton)
        {
            this.setVisible(true);
            voxelWindow.setVisible(false);
            modelVoxel = voxelWindow.getVoxel();
        }
    }

    public void itemStateChanged(ItemEvent event)
    {
        int index;

        if(event.getStateChange() == ItemEvent.SELECTED)
        {
            index = modelComboBox.getSelectedIndex();

            if(index == 2)
            {
                this.setVisible(false);

                imageDims = parentWindow.getImageDims();
                width = imageDims[1];
                height = imageDims[2];
                numberOfSlices = imageDims[3];

                voxelsGroups = new ComboBoxItems();
                voxelsGroups.setSequentialItems(0, width);
                voxelsGroups.setSequentialItems(1, height);
```

```
                voxelsGroups.setSequentialItems(2, numberOfSlices);

                voxelWindow = new ParamWindowVoxel(voxelsGroups);
                selectVoxelButton = voxelWindow.acceptButton;
                selectVoxelButton.addActionListener(this);
            }
        }
    }

    private void createGUI()
    {
        mainContainer = getContentPane();
        mainContainer.setLayout(new GridLayout(5, 1));
        mainContainer.setBackground(Color.white);

        setFirstStatePanel = new JPanel();
        setSecndStatePanel = new JPanel();
        setLagPanel = new JPanel();
        submitParamsPanel = new JPanel();
        modelTypePanel = new JPanel();

        firstStateTitle = new JLabel("Estado 1");
        firstStateComboBox = new JComboBox(statesAcronyms);

        secndStateTitle = new JLabel("Estado 2");
        secndStateComboBox = new JComboBox(statesAcronyms);

        lagParamTitle = new JLabel("Atraso");
        lagParamTextField = new JTextField(2);

        modelParamTitle = new JLabel("Modelo");
        modelComboBox = new JComboBox(modelBoxOptions);
        modelComboBox.addItemListener(this);

        submitCorrParamsButton = new JButton("Aceitar");

        setFirstStatePanel.add(firstStateTitle);
        setFirstStatePanel.add(firstStateComboBox);
        setSecndStatePanel.add(secndStateTitle);
        setSecndStatePanel.add(secndStateComboBox);
        setLagPanel.add(lagParamTitle);
        setLagPanel.add(lagParamTextField);
        modelTypePanel.add(modelParamTitle);
        modelTypePanel.add(modelComboBox);
        submitParamsPanel.add(submitCorrParamsButton);

        mainContainer.add(setFirstStatePanel);
        mainContainer.add(setSecndStatePanel);
        mainContainer.add(setLagPanel);
        mainContainer.add(modelTypePanel);
        mainContainer.add(submitParamsPanel);

        submitCorrParamsButton.setEnabled(true);
        submitCorrParamsButton.addActionListener(this);
        submitCorrParamsButton.addActionListener(parentWindow);

        setSize(windowWidth, windowHeight);
        setLocation(windowXPosition, windowYPosition);
        setTitle("Correlacao");
        setVisible(true);
    }
}
```

```
package fMRIWindows.methodParams;

//
//  ParamWindowFFT.java
//
//
//  Created by Jose Gabriel Lira Gomes on 1/5/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.Color;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

import dataUtils.ComboBoxItems;

import fMRIWindows.MainWindow;
import fMRIWindows.MenuedFrame;
import fMRIWindows.params.ParamWindowVoxel;
//import fMRIWindows.viewers.ViewerWindowModelArray;
import functional.data.Voxel;

public class ParamWindowFFT extends MenuedFrame implements ItemListener
{
    //
    //  Data Members
    //

    private String[] statesAcronyms;
    private MainWindow parentWindow;

    private Container mainContainer;

    private JPanel setFirstStatePanel;
    private JLabel firstStateTitle;
    private JComboBox firstStateComboBox;
    private int firstState;

    private JPanel setSecndStatePanel;
    private JLabel secndStateTitle;
    private JComboBox secndStateComboBox;
    private int secndState;

    private int[] selectedStates = new int[2];

    private JPanel setLagPanel;
    private JLabel lagParamTitle;
    private JTextField lagParamTextField;
    private int lag;

    public int modelType; // 0 - Block; 1 - HRF; 2 - Voxel
    public int dataWin; // 0 - NoWin; 1 - Hann; 2 - Hamming;
    public int modelWin; // 0 - NoWin; 1 - Hann; 2 - Hamming;
    public int infMethod; // 0 - Peak; 1 - Correlation;

    private JPanel modelTypePanel;
    private JPanel dataWinTypePanel;
    private JPanel modelWinTypePanel;
    private JPanel infMethodPanel;
    private JPanel submitParamsPanel;
```

```
public JButton submitFFTParamsButton;

private short[] imageDims;
private int width;
private int height;
private int numberOfSlices;

private ComboBoxItems voxelsGroups;
private ParamWindowVoxel voxelWindow;
private JButton selectVoxelButton;
private Voxel modelVoxel;

private int windowWidth = 200;
private int windowHeight = 400;
private int windowXPosition = 0;
private int windowYPosition = 21;

private String[] modelTypeComboBoxOptions = {"Bloco", "HRF", "Voxel"};
private String[] winTypeComboBoxOptions = {"No Window","Hann", "Hamming"};
private String[] infMethodComboBoxOptions = {"Picos"/*,"Comparacao"*/};

private JComboBox modelComboBox;
private JComboBox dataWinComboBox;
private JComboBox modelWinComboBox;
private JComboBox infMethodComboBox;

//
//  Constructors
//
//      paramWindowFFT()
//

public ParamWindowFFT(String[] statesAcronyms, MainWindow parentWindow)
{
    this.statesAcronyms = statesAcronyms;
    this.parentWindow = parentWindow;

    createGUI();
}

public int getFirstState()
{
    return firstState;
}

public int getSecndState()
{
    return secndState;
}

public int[] getSelectedStates()    //
{
    selectedStates[0] = getFirstState();
    selectedStates[1] = getSecndState();

    return selectedStates;
}

public int getLag()
{
    return lag;
}

public int getModelType()
{
    return modelType;
}

public int getDataWindow()
{
    return dataWin;
```

```
}

public int getModelWindow()
{
    return modelWin;
}

public int getInfMethod()
{
    return infMethod;
}

public Voxel getModelVoxel()
{
    return modelVoxel;
}


/******* LISTEN BUTTON ACTION EVENT *******/

public void actionPerformed(ActionEvent event)
{
    JButton clickedButton = (JButton) event.getSource();

    if(clickedButton == submitFFTParamsButton)
    {
        this.setVisible(false);

        try
        {
            firstState = firstStateComboBox.getSelectedIndex();
            secndState = secndStateComboBox.getSelectedIndex();
            lag = Integer.parseInt(lagParamTextField.getText());
            modelType = modelComboBox.getSelectedIndex();
            dataWin = dataWinComboBox.getSelectedIndex();
            modelWin = modelWinComboBox.getSelectedIndex();
            infMethod = infMethodComboBox.getSelectedIndex();

            if(firstState == secndState)
            {
                JOptionPane.showMessageDialog(this,
                                "Os estados nao podem ser iguais",
                                "Erro de design",
                                JOptionPane.ERROR_MESSAGE);
                this.setVisible(true);
            }
            else
            {
                parentWindow.setFFourierSelectedStates(getSelectedStates());
                parentWindow.setFFourierLag(getLag());
                parentWindow.setFFourierModelType(getModelType());
                parentWindow.setFFourierDataWindow(getDataWindow());
                parentWindow.setFFourierModelWindow(getModelWindow());
                parentWindow.setFFourierInfMethod(getInfMethod());
                parentWindow.setFFourierModelVoxel(getModelVoxel());
                parentWindow.enableCalculateButton();
                parentWindow.setVisible(true);

                //new ViewerWindowModelArray(
                //              parentWindow.getFFourierModelArray());
            }
        }
        catch(NumberFormatException numberFormatException)
        {
            JOptionPane.showMessageDialog(this,
                                "Tem que inserir um inteiro",
                                "Erro de formato",
                                JOptionPane.ERROR_MESSAGE);
            this.setVisible(true);
        }
    }
```

```
        if(clickedButton == selectVoxelButton)
        {
            this.setVisible(true);
            voxelWindow.setVisible(false);
            modelVoxel = voxelWindow.getVoxel();
        }
    }

    public void itemStateChanged(ItemEvent event)
    {
        int index;

        if(event.getStateChange() == ItemEvent.SELECTED)
        {
            index = modelComboBox.getSelectedIndex();

            if(index == 2)
            {
                this.setVisible(false);

                imageDims = parentWindow.getImageDims();
                width = imageDims[1];
                height = imageDims[2];
                numberOfSlices = imageDims[3];

                voxelsGroups = new ComboBoxItems();
                voxelsGroups.setSequentialItems(0, width);
                voxelsGroups.setSequentialItems(1, height);
                voxelsGroups.setSequentialItems(2, numberOfSlices);

                voxelWindow = new ParamWindowVoxel(voxelsGroups);
                selectVoxelButton = voxelWindow.acceptButton;
                selectVoxelButton.addActionListener(this);
            }
        }
    }

    private void createGUI()
    {
        mainContainer = getContentPane();
        mainContainer.setLayout(new GridLayout(8, 1));
        mainContainer.setBackground(Color.white);

        setFirstStatePanel = new JPanel();
        setSecndStatePanel = new JPanel();
        setLagPanel = new JPanel();
        modelTypePanel = new JPanel();
        dataWinTypePanel = new JPanel();
        modelWinTypePanel = new JPanel();
        infMethodPanel = new JPanel();
        submitParamsPanel = new JPanel();

        firstStateTitle = new JLabel("Estado 1");
        firstStateComboBox = new JComboBox(statesAcronyms);

        secndStateTitle = new JLabel("Estado 2");
        secndStateComboBox = new JComboBox(statesAcronyms);

        lagParamTitle = new JLabel("Atraso");
        lagParamTextField = new JTextField(2);

        setFirstStatePanel.add(firstStateTitle);
        setFirstStatePanel.add(firstStateComboBox);
        setSecndStatePanel.add(secndStateTitle);
        setSecndStatePanel.add(secndStateComboBox);
        setLagPanel.add(lagParamTitle);
        setLagPanel.add(lagParamTextField);

        modelComboBox = new JComboBox(modelTypeComboBoxOptions);
        dataWinComboBox = new JComboBox(winTypeComboBoxOptions);
```

```
            modelWinComboBox = new JComboBox(winTypeComboBoxOptions);
            infMethodComboBox = new JComboBox(infMethodComboBoxOptions);
            modelComboBox.addItemListener(this);

            submitFFTParamsButton = new JButton("Aceitar");
            modelTypePanel.add(modelComboBox);
            dataWinTypePanel.add(dataWinComboBox);
            modelWinTypePanel.add(modelWinComboBox);
            infMethodPanel.add(infMethodComboBox);
            submitParamsPanel.add(submitFFTParamsButton);

            mainContainer.add(setFirstStatePanel);
            mainContainer.add(setSecndStatePanel);
            mainContainer.add(setLagPanel);
            mainContainer.add(modelTypePanel);
            mainContainer.add(dataWinTypePanel);
            mainContainer.add(modelWinTypePanel);
            mainContainer.add(infMethodPanel);
            mainContainer.add(submitParamsPanel);

            submitFFTParamsButton.setEnabled(true);
            submitFFTParamsButton.addActionListener(this);
            submitFFTParamsButton.addActionListener(parentWindow);

            setSize(windowWidth, windowHeight);
            setLocation(windowXPosition, windowYPosition);
            setTitle("FFT");
            setVisible(true);
        }
    }
```

```java
package fMRIWindows.methodParams;

//
//  ParamWindowGLM.java
//
//
//  Created by Jose Gabriel Lira Gomes on 1/3/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.Color;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

import dataUtils.ComboBoxItems;

import fMRIWindows.MainWindow;
import fMRIWindows.MenuedFrame;
import fMRIWindows.params.ParamWindowVoxel;
//import fMRIWindows.viewers.ViewerWindowModelArray;
import functional.data.Voxel;

public class ParamWindowGLM extends MenuedFrame implements ItemListener
{
    //
    //  Data Members
    //

    private String[] statesAcronyms;
    private MainWindow parentWindow;

    private Container mainContainer;

    private JPanel setFirstStatePanel;
    private JLabel firstStateTitle;
    private JComboBox firstStateComboBox;
    private int firstState;

    private JPanel setSecndStatePanel;
    private JLabel secndStateTitle;
    private JComboBox secndStateComboBox;
    private int secndState;

    private int[] selectedStates = new int[2];

    private JPanel setLagPanel;
    private JLabel lagParamTitle;
    private JTextField lagParamTextField;
    private int lag;

    private JPanel modelTypePanel;
    private JLabel modelParamTitle;
    private String[] modelBoxOptions = {"Bloco"/*, "HRF"*/};
    private JComboBox modelComboBox;
    private int modelType;  // 0 - Block; 1 - HRF;

    private JPanel inferenceTypePanel;
    private JLabel inferenceParamTitle;
    private String[] inferenceBoxOptions = {"t de Student", "Harmonicas"};
    private JComboBox inferenceComboBox;
    private int inferenceType;
```

```
private short[] imageDims;
private int width;
private int height;
private int numberOfSlices;

private ComboBoxItems voxelsGroups;
private ParamWindowVoxel voxelWindow;
private JButton selectVoxelButton;
private Voxel modelVoxel;

private JPanel submitParamsPanel;
public JButton submitGLMParamsButton;

private int windowWidth = 200;
private int windowHeight = 300;
private int windowXPosition = 0;
private int windowYPosition = 21;

//
//  Constructors
//
//      paramWindowGLM()
//

public ParamWindowGLM(String[] statesAcronyms, MainWindow parentWindow)
{
    this.statesAcronyms = statesAcronyms;
    this.parentWindow = parentWindow;

    createGUI();
}

public int getFirstState()
{
    return firstState;
}

public int getSecndState()
{
    return secndState;
}

public int[] getSelectedStates()
{
    selectedStates[0] = getFirstState();
    selectedStates[1] = getSecndState();

    return selectedStates;
}

public int getLag()
{
    return lag;
}

public int getModelType()
{
    return modelType;
}

public Voxel getModelVoxel()
{
    return modelVoxel;
}

public int getInferenceType()
{
    return inferenceType;
}
```

```
//
//  Public Methods
//
//      void actionPerformed(ActionEvent event)
//

/******* LISTEN BUTTON ACTION EVENT *******/

public void actionPerformed(ActionEvent event)
{
    JButton clickedButton = (JButton) event.getSource();

    if(clickedButton == submitGLMParamsButton)
    {
        this.setVisible(false);

        try
        {
            firstState = firstStateComboBox.getSelectedIndex();
            secndState = secndStateComboBox.getSelectedIndex();
            lag = Integer.parseInt(lagParamTextField.getText());
            modelType = modelComboBox.getSelectedIndex();
            inferenceType = inferenceComboBox.getSelectedIndex();

            if(firstState == secndState)
            {
                JOptionPane.showMessageDialog(this,
                                "Os estados nao podem ser iguais",
                                "Erro de design",
                                JOptionPane.ERROR_MESSAGE);
                this.setVisible(true);
            }
            else
            {
                parentWindow.setGLMSelectedStates(getSelectedStates());
                parentWindow.setGLMLag(getLag());
                parentWindow.setGLMModelType(getModelType());
                parentWindow.setGLMModelVoxel(getModelVoxel());
                parentWindow.setGLMInferenceType(getInferenceType());
                parentWindow.enableCalculateButton();
                parentWindow.setVisible(true);

                //new ViewerWindowModelArray(
                //                  parentWindow.getGLMModelArray());
            }
        }
        catch(NumberFormatException numberFormatException)
        {
            JOptionPane.showMessageDialog(this,
                                "Tem que inserir um inteiro",
                                "Erro de formato",
                                JOptionPane.ERROR_MESSAGE);
            this.setVisible(true);
        }
    }

    if(clickedButton == selectVoxelButton)
    {
        this.setVisible(true);
        voxelWindow.setVisible(false);
        modelVoxel = voxelWindow.getVoxel();
    }
}

public void itemStateChanged(ItemEvent event)
{
    int index;

    if(event.getStateChange() == ItemEvent.SELECTED)
    {
        index = modelComboBox.getSelectedIndex();
```

```
            if(index == 2)
            {
                this.setVisible(false);

                imageDims = parentWindow.getImageDims();
                width = imageDims[1];
                height = imageDims[2];
                numberOfSlices = imageDims[3];

                voxelsGroups = new ComboBoxItems();
                voxelsGroups.setSequentialItems(0, width);
                voxelsGroups.setSequentialItems(1, height);
                voxelsGroups.setSequentialItems(2, numberOfSlices);

                voxelWindow = new ParamWindowVoxel(voxelsGroups);
                selectVoxelButton = voxelWindow.acceptButton;
                selectVoxelButton.addActionListener(this);
            }
        }
    }

    private void createGUI()
    {
        mainContainer = getContentPane();
        mainContainer.setLayout(new GridLayout(6, 1));
        mainContainer.setBackground(Color.white);

        setFirstStatePanel = new JPanel();
        setSecndStatePanel = new JPanel();
        setLagPanel = new JPanel();
        submitParamsPanel = new JPanel();
        modelTypePanel = new JPanel();
        inferenceTypePanel = new JPanel();

        firstStateTitle = new JLabel("Estado 1");
        firstStateComboBox = new JComboBox(statesAcronyms);

        secndStateTitle = new JLabel("Estado 2");
        secndStateComboBox = new JComboBox(statesAcronyms);

        lagParamTitle = new JLabel("Atraso");
        lagParamTextField = new JTextField(2);

        modelParamTitle = new JLabel("Modelo");
        modelComboBox = new JComboBox(modelBoxOptions);
        //modelComboBox.getItemAt(1).setEnabled(false);

        inferenceParamTitle = new JLabel("Estudo");
        inferenceComboBox = new JComboBox(inferenceBoxOptions);

        submitGLMParamsButton = new JButton("Aceitar");

        setFirstStatePanel.add(firstStateTitle);
        setFirstStatePanel.add(firstStateComboBox);
        setSecndStatePanel.add(secndStateTitle);
        setSecndStatePanel.add(secndStateComboBox);
        setLagPanel.add(lagParamTitle);
        setLagPanel.add(lagParamTextField);
        modelTypePanel.add(modelParamTitle);
        modelTypePanel.add(modelComboBox);
        inferenceTypePanel.add(inferenceParamTitle);
        inferenceTypePanel.add(inferenceComboBox);
        submitParamsPanel.add(submitGLMParamsButton);

        mainContainer.add(setFirstStatePanel);
        mainContainer.add(setSecndStatePanel);
        mainContainer.add(setLagPanel);
        mainContainer.add(modelTypePanel);
        mainContainer.add(inferenceTypePanel);
        mainContainer.add(submitParamsPanel);
```

```
            submitGLMParamsButton.setEnabled(true);
            submitGLMParamsButton.addActionListener(this);
            submitGLMParamsButton.addActionListener(parentWindow);

            setSize(windowWidth, windowHeight);
            setLocation(windowXPosition, windowYPosition);
            setTitle("GLM");
            setVisible(true);
        }
    }
```

```java
package fMRIWindows.cleaners;

//
//  ParamWindowCutFrequency.java
//
//
//  Created by José Gabriel Lira Gomes on 1/31/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import java.awt.event.ActionEvent;

import javax.swing.JButton;
import javax.swing.JOptionPane;

import windowUtils.TextFieldsWindow;

import fMRIWindows.MainWindow;

public class ParamWindowHighPassFilter extends TextFieldsWindow
{
    static String windowTitle = "Frequencia de Corte";
    static String[] possibleFormats = {"fc (Hz)"};
    static String[] defaultValues = {"0.013"};

    double cutFrequency;

    private MainWindow parentWindow;

    public ParamWindowHighPassFilter(MainWindow parentWindow)
    {
        super(windowTitle, possibleFormats);
        setTextValues(defaultValues);

        this.parentWindow = parentWindow;
    }

    public void actionPerformed(ActionEvent event)
    {
        JButton clickedButton = (JButton) event.getSource();

        if(clickedButton == acceptButton)
        {
            this.setVisible(false);

            try
            {
                cutFrequency = Double.parseDouble(getTextValue(0));

                if(cutFrequency>=0)
                {
                    System.out.println("fc = " + cutFrequency + " Hz");
                    System.out.println();

                    parentWindow.setCutFrequency(cutFrequency);
                    parentWindow.cleanLowFreqs();
                    parentWindow.setVisible(true);
                }
                else
                {
                    JOptionPane.showMessageDialog(this,
                                "A frequencia de corte tem que ser positiva",
                                "Erro fisico",
                                JOptionPane.ERROR_MESSAGE);
                    this.setVisible(true);
                }
            }
            catch(NumberFormatException numberFormatException)
            {
                JOptionPane.showMessageDialog(this,
                                        "Tem que inserir um duplo",
```

```
                                                "Erro de formato",
                                                JOptionPane.ERROR_MESSAGE);
                        this.setVisible(true);
                    }
                }
            }
        }
```

```
package fMRIWindows.cleaners;

//
//  ParamWindowCutFrequency.java
//
//
//  Created by José Gabriel Lira Gomes on 1/31/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import java.awt.event.ActionEvent;

import javax.swing.JButton;
import javax.swing.JOptionPane;

import windowUtils.TextFieldsWindow;

import fMRIWindows.MainWindow;

public class ParamWindowOutliersCleaner extends TextFieldsWindow
{
    static String windowTitle = "Limpeza de estranhos";
    static String[] possibleFormats = {"maxProb", "maxFraccao"};
    static String[] defaultValues = {"1e-5", "0.01"};

    double outliersThresholdProb;
    double outliersFraction;

    private MainWindow parentWindow;

    public ParamWindowOutliersCleaner(MainWindow parentWindow)
    {
        super(windowTitle, possibleFormats);
        setTextValues(defaultValues);

        this.parentWindow = parentWindow;
    }

    public void actionPerformed(ActionEvent event)
    {
        JButton clickedButton = (JButton) event.getSource();

        if(clickedButton == acceptButton)
        {
            this.setVisible(false);

            try
            {
                outliersThresholdProb = Double.parseDouble(getTextValue(0));
                outliersFraction = Double.parseDouble(getTextValue(1));

                if((outliersThresholdProb>0.0)
                    &&(outliersThresholdProb<1.0)
                    &&(outliersFraction>0.0)
                    &&(outliersFraction<1.0))
                {
                    System.out.println("outliersThresholdProb = "
                                       + outliersThresholdProb);
                    System.out.println("outliersFraction = "
                                       + outliersFraction);
                    System.out.println();

                    parentWindow.setOutliersThresholdProb(outliersThresholdProb);
                    parentWindow.setOutliersFraction(outliersFraction);
                    parentWindow.cleanOutliers();
                    parentWindow.setVisible(true);
                }
                else
                {
                    JOptionPane.showMessageDialog(this,
                                                  "Os valores devem estar entre 0 e 1"
```

```
                                                "Erro matematico",
                                                JOptionPane.ERROR_MESSAGE);
                        this.setVisible(true);
                    }
                }
                catch(NumberFormatException numberFormatException)
                {
                    JOptionPane.showMessageDialog(this,
                                                "Tem que inserir dois duplos",
                                                "Erro de formato",
                                                JOptionPane.ERROR_MESSAGE);
                    this.setVisible(true);
                }
            }
        }
    }
```

```java
package fMRIWindows.methodParams;

//
//  ParamWindowOverlap.java
//
//
//  Created by José Gabriel Lira Gomes on 1/22/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import java.awt.Color;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

import dataUtils.ComboBoxItems;


import fMRIWindows.MainWindow;
import fMRIWindows.MenuedFrame;
import fMRIWindows.params.ParamWindowVoxel;
//import fMRIWindows.viewers.ViewerWindowModelArray;
import functional.data.Voxel;

public class ParamWindowOverlap extends MenuedFrame implements ItemListener
{
    //
    //  Data Members
    //

    private String[] statesAcronyms;
    private MainWindow parentWindow;

    private Container mainContainer;

    private JPanel setFirstStatePanel;
    private JLabel firstStateTitle;
    private JComboBox firstStateComboBox;
    private int firstState;

    private JPanel setSecndStatePanel;
    private JLabel secndStateTitle;
    private JComboBox secndStateComboBox;
    private int secndState;

    private int[] selectedStates = new int[2];

    private JPanel setLagPanel;
    private JLabel lagParamTitle;
    private JTextField lagParamTextField;
    private int lagValue;

    private JPanel modelTypePanel;
    private JLabel modelParamTitle;
    private String[] modelBoxOptions = {"Bloco", "HRF", "Voxel"};
    private JComboBox modelComboBox;
    private int modelType;  // 0 - Block; 1 - HRF;

    private JPanel orderPanel;
    private JLabel orderParamTitle;
    private String[] orderBoxOptions = {"0", "1", "2"};
    private JComboBox orderComboBox;
```

```
    private int order;

    private JPanel submitParamsPanel;
    public JButton submitOverlapParamsButton;

    private short[] imageDims;
    private int width;
    private int height;
    private int numberOfSlices;

    private ComboBoxItems voxelsGroups;
    private ParamWindowVoxel voxelWindow;
    private JButton selectVoxelButton;
    private Voxel modelVoxel;

    private int windowWidth = 200;
    private int windowHeight = 300;
    private int windowXPosition = 0;
    private int windowYPosition = 21;

    //
    //  Constructors
    //
    //      paramWindowOverlap()
    //

    public ParamWindowOverlap(String[] statesAcronyms, MainWindow parentWindow)
    {
        this.statesAcronyms = statesAcronyms;
        this.parentWindow = parentWindow;

        createGUI();
    }

    public int getFirstState()
    {
        return firstState;
    }

    public int getSecndState()
    {
        return secndState;
    }

    public int[] getSelectedStates()
    {
        selectedStates[0] = getFirstState();
        selectedStates[1] = getSecndState();

        return selectedStates;
    }

    public int getLag()
    {
        return lagValue;
    }

    public int getModelType()
    {
        return modelType;
    }

    public Voxel getModelVoxel()
    {
        return modelVoxel;
    }

    public int getOrder()
    {
        return order;
    }
```

```
//
//  Public Methods
//
//      void actionPerformed(ActionEvent event)
//

/******* LISTEN BUTTON ACTION EVENT *******/

public void actionPerformed(ActionEvent event)
{
    JButton clickedButton = (JButton) event.getSource();

    if(clickedButton == submitOverlapParamsButton)
    {
        this.setVisible(false);

        try
        {
            firstState = firstStateComboBox.getSelectedIndex();
            secndState = secndStateComboBox.getSelectedIndex();
            lagValue = Integer.parseInt(lagParamTextField.getText());
            modelType = modelComboBox.getSelectedIndex();
            order = orderComboBox.getSelectedIndex();

            if(firstState == secndState)
            {
                JOptionPane.showMessageDialog(this,
                                    "Os estados nao podem ser iguais",
                                    "Erro de design",
                                    JOptionPane.ERROR_MESSAGE);
                this.setVisible(true);
            }
            else
            {
                parentWindow.setOverlapSelectedStates(getSelectedStates());
                parentWindow.setOverlapLag(getLag());
                parentWindow.setOverlapModelType(getModelType());
                parentWindow.setOverlapModelVoxel(getModelVoxel());
                parentWindow.setOverlapOrder(getOrder());
                parentWindow.enableCalculateButton();
                parentWindow.setVisible(true);

                //new ViewerWindowModelArray(
                //                    parentWindow.getOverlapModelArray());
            }
        }
        catch(NumberFormatException numberFormatException)
        {
            JOptionPane.showMessageDialog(this,
                                    "Tem que inserir um inteiro",
                                    "Erro de formato",
                                    JOptionPane.ERROR_MESSAGE);
            this.setVisible(true);
        }
    }

    if(clickedButton == selectVoxelButton)
    {
        this.setVisible(true);
        voxelWindow.setVisible(false);
        modelVoxel = voxelWindow.getVoxel();
    }
}

public void itemStateChanged(ItemEvent event)
{
    int index;

    if(event.getStateChange() == ItemEvent.SELECTED)
    {
```

```
            index = modelComboBox.getSelectedIndex();

            if(index == 2)
            {
                this.setVisible(false);

                imageDims = parentWindow.getImageDims();
                width = imageDims[1];
                height = imageDims[2];
                numberOfSlices = imageDims[3];

                voxelsGroups = new ComboBoxItems();
                voxelsGroups.setSequentialItems(0, width);
                voxelsGroups.setSequentialItems(1, height);
                voxelsGroups.setSequentialItems(2, numberOfSlices);

                voxelWindow = new ParamWindowVoxel(voxelsGroups);
                selectVoxelButton = voxelWindow.acceptButton;
                selectVoxelButton.addActionListener(this);
            }
        }
    }

    private void createGUI()
    {
        mainContainer = getContentPane();
        mainContainer.setLayout(new GridLayout(6, 1));
        mainContainer.setBackground(Color.white);

        setFirstStatePanel = new JPanel();
        setSecndStatePanel = new JPanel();
        setLagPanel = new JPanel();
        submitParamsPanel = new JPanel();
        modelTypePanel = new JPanel();
        orderPanel = new JPanel();

        firstStateTitle = new JLabel("Estado 1");
        firstStateComboBox = new JComboBox(statesAcronyms);

        secndStateTitle = new JLabel("Estado 2");
        secndStateComboBox = new JComboBox(statesAcronyms);

        lagParamTitle = new JLabel("Atraso");
        lagParamTextField = new JTextField(2);

        modelParamTitle = new JLabel("Modelo");
        modelComboBox = new JComboBox(modelBoxOptions);
        modelComboBox.addItemListener(this);

        orderParamTitle = new JLabel("Ordem");
        orderComboBox = new JComboBox(orderBoxOptions);

        submitOverlapParamsButton = new JButton("Aceitar");

        setFirstStatePanel.add(firstStateTitle);
        setFirstStatePanel.add(firstStateComboBox);
        setSecndStatePanel.add(secndStateTitle);
        setSecndStatePanel.add(secndStateComboBox);
        setLagPanel.add(lagParamTitle);
        setLagPanel.add(lagParamTextField);
        modelTypePanel.add(modelParamTitle);
        modelTypePanel.add(modelComboBox);
        orderPanel.add(orderParamTitle);
        orderPanel.add(orderComboBox);
        submitParamsPanel.add(submitOverlapParamsButton);

        mainContainer.add(setFirstStatePanel);
        mainContainer.add(setSecndStatePanel);
        mainContainer.add(setLagPanel);
        mainContainer.add(modelTypePanel);
        mainContainer.add(orderPanel);
```

```
            mainContainer.add(submitParamsPanel);

            submitOverlapParamsButton.setEnabled(true);
            submitOverlapParamsButton.addActionListener(this);
            submitOverlapParamsButton.addActionListener(parentWindow);

            setSize(windowWidth, windowHeight);
            setLocation(windowXPosition, windowYPosition);
            setTitle("Sobreposicao");
            setVisible(true);
        }
    }
```

```java
package fMRIWindows.methodParams;

//
//  ParamWindowStudent.java
//
//
//  Created by Jose Gabriel Lira Gomes on 1/4/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.Color;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

import fMRIWindows.MainWindow;
import fMRIWindows.MenuedFrame;

public class ParamWindowStudent extends MenuedFrame
{
    //
    //  Data Members
    //

    private String[] statesAcronyms;
    private MainWindow parentWindow;

    private Container mainContainer;

    private JPanel setFirstStatePanel;
    private JLabel firstStateTitle;
    private JComboBox firstStateComboBox;
    private int firstState;

    private JPanel setSecndStatePanel;
    private JLabel secndStateTitle;
    private JComboBox secndStateComboBox;
    private int secndState;

    private int[] selectedStates = new int[2];  //

    private JPanel setLagPanel;
    private JLabel lagParamTitle;
    private JTextField lagParamTextField;
    private int lag;

    private JPanel modelTypePanel;
    private JLabel modelParamTitle;
    private String[] modelBoxOptions = {"Bloco"/*, "HRF"*/};
    private JComboBox modelComboBox;
    private int modelType; // 0 - Block; 1 - HRF;

    private JPanel submitParamsPanel;
    public JButton submitTStudentParamsButton;

    private int windowWidth = 200;
    private int windowHeight = 250;
    private int windowXPosition = 0;
    private int windowYPosition = 21;

    //
    //  Constructors
    //
    //      paramWindowStudent()
```

```
//

public ParamWindowStudent(String[] statesAcronyms, MainWindow parentWindow)
{
    this.statesAcronyms = statesAcronyms;
    this.parentWindow = parentWindow;

    createGUI();
}

public int getFirstState()
{
    return firstState;
}

public int getSecndState()
{
    return secndState;
}

public int[] getSelectedStates()    //
{
    selectedStates[0] = getFirstState();
    selectedStates[1] = getSecndState();

    return selectedStates;
}

public int getLag()
{
    return lag;             //
}

public int getModelType()   //
{
    return modelType;
}

/******* LISTEN BUTTON ACTION EVENT *******/

public void actionPerformed(ActionEvent event)
{
    JButton clickedButton = (JButton) event.getSource();

    if(clickedButton == submitTStudentParamsButton)
    {
        this.setVisible(false);

        try
        {
            firstState = firstStateComboBox.getSelectedIndex();
            secndState = secndStateComboBox.getSelectedIndex();
            lag = Integer.parseInt(lagParamTextField.getText());
            modelType = modelComboBox.getSelectedIndex();

            if(firstState == secndState)
            {
                JOptionPane.showMessageDialog(this,
                                "Os estados nao podem ser iguais",
                                "Erro de design",
                                JOptionPane.ERROR_MESSAGE);
                this.setVisible(true);
            }
            else
            {
                parentWindow.setTStudentSelectedStates(getSelectedStates());
                parentWindow.setTStudentLag(getLag());
                parentWindow.setTStudentModelType(getModelType());
                parentWindow.enableCalculateButton();
                parentWindow.setVisible(true);
            }
```

```
            }
            catch(NumberFormatException numberFormatException)
            {
                JOptionPane.showMessageDialog(this,
                                              "Tem que inserir um inteiro",
                                              "Erro de formato",
                                              JOptionPane.ERROR_MESSAGE);
                this.setVisible(true);
            }
        }
    }

    private void createGUI()
    {
        mainContainer = getContentPane();
        mainContainer.setLayout(new GridLayout(5, 1));
        mainContainer.setBackground(Color.white);

        setFirstStatePanel = new JPanel();
        setSecndStatePanel = new JPanel();
        setLagPanel = new JPanel();
        modelTypePanel = new JPanel();
        submitParamsPanel = new JPanel();

        firstStateTitle = new JLabel("Estado 1");
        firstStateComboBox = new JComboBox(statesAcronyms);

        secndStateTitle = new JLabel("Estado 2");
        secndStateComboBox = new JComboBox(statesAcronyms);

        lagParamTitle = new JLabel("Atraso");
        lagParamTextField = new JTextField(2);

        modelParamTitle = new JLabel("Modelo");
        modelComboBox = new JComboBox(modelBoxOptions);

        submitTStudentParamsButton = new JButton("Aceitar");

        setFirstStatePanel.add(firstStateTitle);
        setFirstStatePanel.add(firstStateComboBox);
        setSecndStatePanel.add(secndStateTitle);
        setSecndStatePanel.add(secndStateComboBox);
        setLagPanel.add(lagParamTitle);
        setLagPanel.add(lagParamTextField);
        modelTypePanel.add(modelParamTitle);
        modelTypePanel.add(modelComboBox);
        submitParamsPanel.add(submitTStudentParamsButton);

        mainContainer.add(setFirstStatePanel);
        mainContainer.add(setSecndStatePanel);
        mainContainer.add(setLagPanel);
        mainContainer.add(modelTypePanel);
        mainContainer.add(submitParamsPanel);

        submitTStudentParamsButton.setEnabled(true);
        submitTStudentParamsButton.addActionListener(this);
        submitTStudentParamsButton.addActionListener(parentWindow);

        setSize(windowWidth, windowHeight);
        setLocation(windowXPosition, windowYPosition);
        setTitle("t Student");
        setVisible(true);
    }
}
```

```java
package fMRIWindows.params;

//
//  ParamWindowVoxel.java
//
//
//  Created by José Gabriel Lira Gomes on 1/17/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import dataUtils.ComboBoxItems;
import windowUtils.ComboBoxesWindow;
import functional.data.Voxel;

public class ParamWindowVoxel extends ComboBoxesWindow
{
    static private String windowTitle = "Voxel";
    static private String[] voxelCoordinates = {"x", "y", "z"};

    public ParamWindowVoxel(ComboBoxItems boxItems)
    {
        super(windowTitle, voxelCoordinates, boxItems);
    }

    public Voxel getVoxel()
    {
        int x;
        int y;
        int z;

        x = getSelectedIndex(0);
        y = getSelectedIndex(1);
        z = getSelectedIndex(2);

        return new Voxel(x, y, z);
    }
}
```

```java
package functional.results;

//
//  ProbVolumesAnalyzer.java
//
//
//  Created by José Gabriel Lira Gomes on 5/16/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import estatistica.descriptive.ThreeDArrayAnalyzer;
import functional.data.Mask;
import functional.data.Voxel;

public class ProbVolumesAnalyzer
{
    //
    //  Data Members
    //

    private double[][][] firstArray;
    private double[][][] secndArray;

    private int xSize;
    private int ySize;
    private int zSize;

    //
    //  Constructors
    //
    //      ProbVolumeAnalyzer(double[][][] firstArray,
    //                         double[][][] secondArray)
    //

    public ProbVolumesAnalyzer(double[][][] firstArray,
                               double[][][] secndArray)
    {
        this.firstArray = firstArray;
        this.secndArray = secndArray;

        this.xSize = firstArray.length;
        this.ySize = firstArray[0].length;
        this.zSize = firstArray[0][0].length;
    }

    //
    //  Public Methods
    //
    //      double[][][] getDifference()
    //

    public double[][][] getDifference()
    {
        double[][][] differenceArray = new double[xSize][ySize][zSize];

        int x, y, z;

        for(x=0; x<xSize; x++)
        {
            for(y=0; y<ySize; y++)
            {
                for(z=0; z<zSize; z++)
                {
                    differenceArray[x][y][z] =
                            firstArray[x][y][z] - secndArray[x][y][z];
                }
            }
        }

        return differenceArray;
    }
```

```
public double[][][] getSum()
{
    double[][][] sumArray = new double[xSize][ySize][zSize];

    int x, y, z;

    for(x=0; x<xSize; x++)
    {
        for(y=0; y<ySize; y++)
        {
            for(z=0; z<zSize; z++)
            {
                sumArray[x][y][z] =
                        firstArray[x][y][z] + secndArray[x][y][z];
            }
        }
    }

    return sumArray;
}

public Voxel getMaxOvlpVoxel()
{
    return (new ThreeDArrayAnalyzer(getSum()).getMaximumVoxel());
}

public double getOverlapPerc(double threshold, boolean firstData)
{
    Mask firstMask;
    Mask secndMask;
    Mask overlapMask;

    double numberOfFirstMVoxels;
    double numberOfSecndMVoxels;
    double numberOfOverlVoxels;
    double overlapPerc;

    firstMask = getMask(firstArray, threshold);
    secndMask = getMask(secndArray, threshold);
    numberOfFirstMVoxels = (double)firstMask.getNumberOfMaskVoxels();
    numberOfSecndMVoxels = (double)secndMask.getNumberOfMaskVoxels();

    overlapMask = getOverlapMask(threshold);
    numberOfOverlVoxels = (double)overlapMask.getNumberOfMaskVoxels();

    if(firstData)
    {
        overlapPerc = numberOfOverlVoxels/numberOfFirstMVoxels;
    }
    else
    {
        overlapPerc = numberOfOverlVoxels/numberOfSecndMVoxels;
    }

    return overlapPerc;
}

public double getNumberOfOverlVoxels(double threshold)
{
    Mask overlapMask;

    double numberOfOverlVoxels;

    overlapMask = getOverlapMask(threshold);
    numberOfOverlVoxels = (double)overlapMask.getNumberOfMaskVoxels();

    return numberOfOverlVoxels;
}

private Mask getMask(double[][][] dataArray, double threshold)
```

```
    {
        boolean[][][] maskArray = new boolean[xSize][ySize][zSize];

        int x, y, z;

        Mask dataMask;

        for(x=0; x<xSize; x++)
        {
            for(y=0; y<ySize; y++)
            {
                for(z=0; z<zSize; z++)
                {
                    if(dataArray[x][y][z] >= threshold)
                    {
                        maskArray[x][y][z] = true;
                    }
                    else
                    {
                        maskArray[x][y][z] = false;
                    }
                }
            }
        }

        dataMask = new Mask(maskArray);

        return dataMask;
    }

    private Mask getOverlapMask(double threshold)
    {
        Mask firstMask;
        Mask secndMask;
        Mask overlapMask;

        firstMask = getMask(firstArray, threshold);
        secndMask = getMask(secndArray, threshold);

        overlapMask = firstMask.getAnd(secndMask);

        return overlapMask;
    }
}
```

```java
package windowUtils;

//
//  ProgressBarWindow.java
//
//
//  Created by JGLG on 5/5/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

import javax.swing.*;

import fMRIWindows.MenuedFrame;

import java.awt.*;

public class ProgressBarWindow extends MenuedFrame
{
    //
    //  Data Members
    //

    public JProgressBar progressBar;
    private Container progressBarContainer;
    //private JPanel progressBarPanel;

    //
    //  Constructors
    //
    //      progressBarWindow()
    //

    public ProgressBarWindow()
    {
        setTitle("A carregar");
        setSize(400, 160);
        setLocation(250, 250);
        setVisible(true);

        progressBarContainer = this.getContentPane();
        progressBarContainer.setBackground(Color.white);
        progressBarContainer.setLayout(null);
        /*
        progressBarPanel = new JPanel();
        progressBarPanel.setLayout(new BorderLayout());
        */
        progressBar = new JProgressBar();
        progressBar.setPreferredSize(new Dimension(200, 20));
        progressBar.setMinimum(0);
        progressBar.setMaximum(100);
        progressBar.setValue(0);
        progressBar.setBounds(100, 60, 200, 20);

        progressBarContainer.add(progressBar);
        //progressBarPanel.add(progressBar, BorderLayout.CENTER);

    }
}
```

```
package windowUtils;

//
//  RadioButtonsWindow.java
//
//
//  Created by José Gabriel Lira Gomes on 1/17/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JRadioButton;

import fMRIWindows.MenuedFrame;

public class RadioButtonsWindow extends MenuedFrame implements ItemListener
{
    private int selection;

    private String windowTitle;
    private String titleText;
    private String[] buttonsText;
    private int numberOfButtons;

    private JRadioButton[] radioButton;

    private Container contentPane;

    private JPanel radioPanel;
    private JPanel acceptPanel;

    public JButton acceptButton;

    ButtonGroup optionsGroup;

    private int windowWidth = 200;
    private int windowHeight;
    private int windowXPosition = 0;
    private int windowYPosition = 21;

    public RadioButtonsWindow(String windowTitle,
                              String titleText,
                              String[] buttonsText)
    {
        this.windowTitle = windowTitle;
        this.titleText = titleText;
        this.buttonsText = buttonsText;
        this.numberOfButtons = buttonsText.length;

        createGUI();
    }

    public int getSelection()
    {
        return selection;
    }

    public void itemStateChanged(ItemEvent event)
    {
        JRadioButton source = (JRadioButton) event.getSource();
```

```
        if(event.getStateChange() == ItemEvent.SELECTED)
        {
            for(int i=0; i<numberOfButtons; i++)
            {
                if(buttonsText[i].equals(source.getText()))
                {
                    selection = i;
                }

            }
        }
    }

    private void createGUI()
    {
        windowHeight = numberOfButtons*50 + 50;

        contentPane = getContentPane();
        contentPane.setLayout(new BorderLayout());
        contentPane.setBackground(Color.white);

        radioPanel = new JPanel(new GridLayout(numberOfButtons, 1));
        radioPanel.setBorder(BorderFactory.createTitledBorder(titleText));

        optionsGroup = new ButtonGroup();

        radioButton = new JRadioButton[numberOfButtons];

        for(int i=0; i<numberOfButtons; i++)
        {
            radioButton[i] = new JRadioButton(buttonsText[i]);
            radioButton[i].addItemListener((ItemListener) this);
            optionsGroup.add(radioButton[i]);
            radioPanel.add(radioButton[i]);
        }

        radioButton[0].setSelected(true);

        acceptPanel = new JPanel();
        acceptButton = new JButton("Aceitar");
        acceptButton.addActionListener((ActionListener) this);
        acceptPanel.add(acceptButton);

        contentPane.add(radioPanel, BorderLayout.CENTER);
        contentPane.add(acceptPanel, BorderLayout.SOUTH);

        setSize(windowWidth, windowHeight);
        setLocation(windowXPosition, windowYPosition);
        setTitle(windowTitle);
        setVisible(true);
    }
}
```

```
package fileUtils;

//
//  ReadTextNumberFile.java
//
//
//  Created by JGLG on 1/11/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.io.*;
import java.nio.*;
import java.nio.channels.*;

public class ReadTextNumberFile
{
    private String filePath;

    private double[] number;

    public ReadTextNumberFile(String filePath, String fileExtension)
                                        throws Exception
    {
        this.filePath = filePath;
        if(!findExtension(filePath, fileExtension))
        {
            throw new Exception("not " +  fileExtension);
        }

        readData();
    }

    public double[] getData()
    {
        return number;
    }

    public int getNumberOfLines()
    {
        return number.length;
    }

    private void readData()
    {
        int i;
        int line = 0;
        int numLines = 0;

        byte bValue;
        int intValue;
        char asciiChar;

        FileInputStream fIn;
        FileChannel fChan;
        long fSize;
        ByteBuffer mBuf;

        boolean valid = true;

        String[] numberString;

        // Apple => nextLineChar = cr       * supported
        // Unix => nextLineChar = lf        * unsupported
        // DOS => nextLineChar = cr + lf    * supported

        int carriageReturn = 13;    //cr
        int lineFeed = 10;          //lf

        int numberOfCR = 0;
        int numberOfLF = 0;
        int nextLineChar = carriageReturn;  // Apple or DOS
```

```
try
{
    /******* abre o ficheiro *******/
    fIn = new FileInputStream(filePath);

    /******* abre canal para o ficheiro *******/
    fChan = fIn.getChannel();

    /******* le tamanho do ficheiro *******/
    fSize = fChan.size();

    /******* cria espaco na memoria para os dados *******/
    mBuf = ByteBuffer.allocate((int) fSize);

    /******* le dados para a memoria *******/
    fChan.read(mBuf);

    /******* volta ao inicio dos dados *******/
    mBuf.rewind();

    /******* conta o numero de cr e lf *******/
    for(i=0; i< fSize; i++)
    {
        bValue = mBuf.get();
        intValue = (int)bValue;

        if (intValue == carriageReturn)
        {
            numberOfCR = numberOfCR + 1;
        }

        if (intValue == lineFeed)
        {
            numberOfLF = numberOfLF + 1;
        }
    }

    if((numberOfCR > 0)&&(numberOfLF == 0)) // Apple
    {
        System.out.println("Formato Apple");
    }

    if((numberOfCR == 0)&&(numberOfLF > 0)) // Unix
    {
        nextLineChar = lineFeed;
        System.out.println("Formato Unix");
    }

    if((numberOfCR > 0)&&(numberOfLF > 0)
            &&(numberOfCR == numberOfLF))   // DOS
    {
        System.out.println("Formato DOS");
    }

    /******* volta ao inicio dos dados *******/
    mBuf.rewind();

    /******* conta o numero de linhas *******/
    for(i=0; i< fSize; i++)
    {
        bValue = mBuf.get();
        intValue = (int)bValue;

        if (intValue == nextLineChar)
        {
            numLines = numLines + 1;
        }
    }

    numLines = numLines + 1;
```

```
        numberString = new String[numLines];
        number = new double[numLines];

        for(i=0; i<numLines; i++)
        {
            numberString[i] = "";
        }

        /******* volta ao inicio dos dados *******/
        mBuf.rewind();


        /******* le texto *******/
        for(i=0; i< fSize; i++)
        {

            bValue = mBuf.get();
            intValue = (int)bValue;
            asciiChar = (char)bValue;

            if (bValue == 0)
            {
                valid = false;
            }

            if (intValue == carriageReturn)
            {
                valid = false;
            }

            if (intValue == lineFeed)
            {
                valid = false;
            }

            if(valid)
            {
                numberString[line] = numberString[line] + asciiChar;
            }
            else
            {
                if (intValue == carriageReturn)
                {
                    line = line + 1;
                }
            }

            valid = true;
        }

        for(i=0; i<numLines; i++)
        {
            number[i] = Double.parseDouble(numberString[i]);
        }

        /******* fecha o canal *******/
        fChan.close();

        /******* fecha o ficheiro original *******/
        fIn.close();
    }
    catch(FileNotFoundException exc)
    {
        System.out.println("File Not Found");
        return;
    }
    catch(IOException exc)
    {
        System.out.println(exc);
        System.exit(1);
```

```
        }
    }

    private boolean findExtension(String text, String extension)
    {
        boolean extensionIsPresent = false;
        int textSize;
        int extensionSize;
        String textExtension;

        textSize = text.length();
        extensionSize = extension.length();

        textExtension = text.substring(textSize-extensionSize);

        if(textExtension.equals(extension))
        {
            extensionIsPresent = true;
        }

        return extensionIsPresent;
    }
}
```

```
package matematica.specialFunctions;

//
//  Sinc.java
//
//
//  Created by José Gabriel Lira Gomes on 10/11/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

public class Sinc
{
    private double frequency;
    private static double PI = 3.14159265358979;
    private double[] functionArray;

    public Sinc(double frequency)
    {
        this.frequency = frequency;
    }


    public double[] getFunctionArray(double startPoint,
                                     double deltaT,
                                     int numberOfPoints)
    {
        int i, j;

        functionArray = new double[numberOfPoints];

        //double deltaT = (lastPoint - startPoint)/(numberOfPoints - 1);
        double time;

        j = 0;

        for(i=0; i<numberOfPoints; i++)
        {
            time = startPoint + (i*deltaT);
            if(time==0)
            {
                functionArray[j] = 1.0;
            }
            else
            {
                functionArray[j] = Math.sin(2*PI*frequency*time)
                                   /(2*PI*frequency*time);
            }
            j = j + 1;
        }

        return functionArray;
    }

    public double[] getInvFunctionArray(double startPoint,
                                     double deltaT,
                                     int numberOfPoints)
    {
        int i, j;

        functionArray = new double[numberOfPoints];

        double time;

        j = 0;

        for(i=0; i<numberOfPoints; i++)
        {
            time = startPoint + (i*deltaT);
            if(time==0)
            {
                functionArray[j] = 1.0;
```

```
            }
            else
            {
                functionArray[j] = -Math.sin(2*PI*frequency*time)
                                        /(2*PI*frequency*time);
            }
            j = j + 1;
        }

        return functionArray;
    }
}
```

```java
package physics;

//
//  SpectrumAnalyzer.java
//
//
//  Created by José Gabriel Lira Gomes on 1/28/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import matematica.difCalculus.Derivator;

public class SpectrumAnalyzer
{
    private double[] spectrum;

    private int spectrumSize;
    private Derivator specDerivator;


    public SpectrumAnalyzer(double[] spectrum)
    {
        this.spectrum = spectrum;
        this.spectrumSize = spectrum.length;

        specDerivator = new Derivator(spectrum);
    }


    public int getNumberOfPeaks()
    {
        int i;
        int counter = 0;
        double[] specFirstDer;

        specFirstDer = specDerivator.getFirstDer();

        for(i=2; i<(spectrumSize)/2-1; i++)
        {
            if((spectrum[i]!=0)
                &&(specFirstDer[i-2]>0)
                &&(specFirstDer[i-1]>0)
                &&(specFirstDer[i]<0)
                &&(specFirstDer[i+1]<0))
            {
                counter = counter + 1;
            }
        }

        return counter;
    }

    public int getNumberOfLows()
    {
        return getNumberOfPeaks();
    }

    public double[] getPeaks()
    {
        int[] peaksFreq;
        int numberOfPeaks;
        int i;

        peaksFreq = getPeaksFreq();
        numberOfPeaks = peaksFreq.length;

        double[] peaks = new double[numberOfPeaks];

        for(i=0; i<numberOfPeaks; i++)
        {
            peaks[i] = spectrum[peaksFreq[i]];
```

```
        }

        return peaks;
    }

    public double[] getLows()
    {
        int[] lowsFreq;
        int numberOfLows;
        int i;

        lowsFreq = getLowsFreq();
        numberOfLows = lowsFreq.length;

        double[] lows = new double[numberOfLows];

        for(i=0; i<numberOfLows; i++)
        {
            lows[i] = spectrum[lowsFreq[i]];
        }

        return lows;
    }

    public int[] getPeaksFreq()
    {
        int i;
        int counter = 0;
        int numberOfPeaks = getNumberOfPeaks();
        int[] peaksFreq = new int[numberOfPeaks];
        double[] specFirstDer;

        specFirstDer = specDerivator.getFirstDer();

        for(i=2; i<(spectrumSize)/2-1; i++)
        {
            if((specFirstDer[i-2]>0)
                &&(specFirstDer[i-1]>0)
                &&(specFirstDer[i]<0)
                &&(specFirstDer[i+1]<0))
            {
                peaksFreq[counter] = i;
                counter = counter + 1;
            }
        }

        return peaksFreq;
    }

    public int[] getLowsFreq()
    {
        int[] peaksFreq;
        int numberOfLows;
        int[] lowsFreq;
        int i;

        peaksFreq = getPeaksFreq();
        numberOfLows = peaksFreq.length;
        lowsFreq = new int[numberOfLows];

        lowsFreq[0] = 0;

        for(i=1; i<numberOfLows; i++)
        {
            lowsFreq[i] = Math.round((peaksFreq[i]+peaksFreq[i-1])/2);
        }

        return lowsFreq;
    }
}
```

```java
package functional.design;

//
//  StateRecord.java
//
//
//  Created by José Gabriel Lira Gomes on 7/5/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

public class StateRecord
{
    //
    //  Data Members
    //

    protected String acronym;
    protected String name;

    //
    //  Constructors
    //
    //      stateRecord()
    //

    public StateRecord()
    {
        acronym = "";
        name = "";
    }

    public String getAcronym()
    {
        return acronym;
    }

    public void setAcronym(String acronym)
    {
        this.acronym = acronym;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }
 }
```

```java
package windowUtils;

//
//  TextFieldsWindow.java
//
//
//  Created by José Gabriel Lira Gomes on 1/18/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import java.awt.Container;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.TextEvent;
import java.awt.event.TextListener;

import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;

import fMRIWindows.MenuedFrame;

public class TextFieldsWindow extends MenuedFrame implements TextListener
{
    String[] textValues;

    private String windowTitle;
    private String[] labelsText;
    private int numberOfTextFields;

    private Container contentPane;

    private JPanel textFieldsPanel;
    private JPanel acceptPanel;

    public JButton acceptButton;

    int i, j;

    private JLabel[] label;
    private TextField[] textField;
    private static int defaultTFSize = 4;

    private int windowWidth = 200;
    private int windowHeight;
    private int windowXPosition = 0;
    private int windowYPosition = 21;

    public TextFieldsWindow(String windowTitle, String[] labelsText)
    {
        this.windowTitle = windowTitle;
        this.labelsText = labelsText;

        this.numberOfTextFields = labelsText.length;
        textValues = new String[numberOfTextFields];

        createGUI();
    }

    public String[] getTextValues()
    {
        return textValues;
    }

    public void setTextValues(String[] textValues)
    {
        if(textValues.length == numberOfTextFields)
        {
```

```
        this.textValues = textValues;

        for(int i=0; i<numberOfTextFields; i++)
        {
            textField[i].setText(textValues[i]);
        }
    }
}

public String getTextValue(int index)
{
    return textValues[index];
}

public void actionPerformed(ActionEvent event)
{

}

public void textValueChanged(TextEvent tEvent)
{
    Object source = tEvent.getSource();

    for(int i=0; i<numberOfTextFields; i++)
    {
        if(source == textField[i])
        {
            textValues[i] = textField[i].getText();
        }

        if(textValues[i] == null)
        {
            j = 0;
        }
    }
}

private void createGUI()
{
    windowHeight = 20 + (numberOfTextFields - 1)*55 + 47 + 80;

    contentPane = getContentPane();
    contentPane.setLayout(new GridBagLayout());
    GridBagConstraints c = new GridBagConstraints();

    textFieldsPanel =
                new JPanel(new GridLayout(2*numberOfTextFields, 1));

    label = new JLabel[numberOfTextFields];
    textField = new TextField[numberOfTextFields];

    for(i=0; i<numberOfTextFields; i++)
    {
        label[i] = new JLabel(labelsText[i]);
        label[i].setHorizontalAlignment(JLabel.CENTER);
        textField[i] = new TextField(defaultTFSize);
        textField[i].addTextListener(this);
        textFieldsPanel.add(label[i]);
        textFieldsPanel.add(textField[i]);
    }

    acceptPanel = new JPanel();
    acceptButton = new JButton("Aceitar");
    acceptButton.addActionListener(this);
    acceptPanel.add(acceptButton);

    c.fill = GridBagConstraints.CENTER;

    c.gridx = 0;
    c.gridy = 0;
    c.weightx = 1.00;
```

```
            c.weighty = (20 + ((double)numberOfTextFields - 1)*55 + 47 + 10)
                        /(20 + (double)(numberOfTextFields - 1)*55 + 47 + 80);

            contentPane.add(textFieldsPanel, c);

            c.gridx = 0;
            c.gridy = 1;
            c.weightx = 1.00;
            c.weighty = 1
                        - (20 + ((double)numberOfTextFields - 1)*55 + 47 + 10)
                        /(20 + (double)(numberOfTextFields - 1)*55 + 47 + 80);

            contentPane.add(acceptPanel, c);

            setSize(windowWidth, windowHeight);
            setLocation(windowXPosition, windowYPosition);
            setTitle(windowTitle);
            setVisible(true);
        }
    }
```

```java
package estatistica.descriptive;

//
//  ThreeDArrayAnalyzer.java
//
//
//  Created by José Gabriel Lira Gomes on 2/8/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import estatistica.distributions.Histogram;
import functional.data.Voxel;

public class ThreeDArrayAnalyzer
{
    private double[][][] dataArray;
    private int numberOfLines;
    private int numberOfColumns;
    private int numberOfPlanes;

    public ThreeDArrayAnalyzer(double[][][] dataArray)
    {
        this.dataArray = dataArray;

        this.numberOfLines = dataArray.length;
        this.numberOfColumns = dataArray[0].length;
        this.numberOfPlanes = dataArray[0][0].length;
    }

    public double getMean()
    {
        int N;
        int x, y, z;
        double dataSum=0;
        double mean;

        N = numberOfColumns*numberOfLines*numberOfPlanes;

        for(z=0; z<numberOfPlanes; z++)
        {
            for(y=0; y<numberOfLines; y++)
            {
                for(x=0; x<numberOfColumns; x++)
                {
                    dataSum = dataSum + dataArray[y][x][z];
                }
            }
        }

        mean = dataSum/(double)N;

        return mean;
    }

    public double getStdDev()
    {
        double stdDev;

        stdDev = Math.sqrt(getVariance());

        return stdDev;
    }

    public double getVariance()
    {
        int x, y, z;
        double devSum=0.0;
        double ep=0.0;
        double s;
        int N;
        double dN;
```

```
        double mean;
        double variance;

        N = numberOfColumns*numberOfLines*numberOfPlanes;
        dN = (double)N;

        mean = getMean();

        for(z=0; z<numberOfPlanes; z++)
        {
            for(y=0; y<numberOfLines; y++)
            {
                for(x=0; x<numberOfColumns; x++)
                {
                    s = dataArray[y][x][z]-mean;
                    devSum = devSum + s*s;
                    ep = ep + s;
                }
            }
        }

        variance = (devSum - ep*ep/dN)/(dN-1);

        return variance;
    }

    public double getMaximum()
    {
        int x, y, z;
        double element;
        double maximum = dataArray[0][0][0];

        for(z=0; z<numberOfPlanes; z++)
        {
            for(y=0; y<numberOfLines; y++)
            {
                for(x=0; x<numberOfColumns; x++)
                {
                    element = dataArray[y][x][z];
                    if(element > maximum)
                    {
                        maximum = element;
                    }
                }
            }
        }

        return maximum;
    }


    public double getMinimum()
    {
        int x, y, z;
        double element;
        double minimum = dataArray[0][0][0];

        for(z=0; z<numberOfPlanes; z++)
        {
            for(y=0; y<numberOfLines; y++)
            {
                for(x=0; x<numberOfColumns; x++)
                {
                    element = dataArray[y][x][z];
                    if(element < minimum)
                    {
                        minimum = element;
                    }
                }
            }
        }
```

```
    return minimum;
}

public Voxel getMaximumVoxel()
{
    int x, y, z;
    double element;
    double maximum = dataArray[0][0][0];
    Voxel maxVoxel = new Voxel(0, 0, 0);

    for(z=0; z<numberOfPlanes; z++)
    {
        for(y=0; y<numberOfLines; y++)
        {
            for(x=0; x<numberOfColumns; x++)
            {
                element = dataArray[y][x][z];
                if(element > maximum)
                {
                    maximum = element;
                    maxVoxel = new Voxel(y, x, z);
                }
            }
        }
    }

    return maxVoxel;
}

public Voxel getMinimumVoxel()
{
    int x, y, z;
    double element;
    double minimum = dataArray[0][0][0];
    Voxel minVoxel = new Voxel(0, 0, 0);

    for(z=0; z<numberOfPlanes; z++)
    {
        for(y=0; y<numberOfLines; y++)
        {
            for(x=0; x<numberOfColumns; x++)
            {
                element = dataArray[y][x][z];
                if(element < minimum)
                {
                    minimum = element;
                    minVoxel = new Voxel(y, x, z);
                }
            }
        }
    }

    return minVoxel;
}

public Histogram getHistogram(int binningMethod)
{
    double[][] histogram;

    double[] serialDataArray;
    double N;
    double minimum = getMinimum();
    double maximum = getMaximum();
    int numberOfClasses;
    double classesWidth;

    int i, j;

    serialDataArray = getSerialData();
```

```
        N = (double)serialDataArray.length;

        switch (binningMethod)
        {
            case 0: // Sturge
                {
                    numberOfClasses =
                            (int)Math.round(Math.log(N)/Math.log(2)) + 1;
                    classesWidth = (maximum-minimum)/numberOfClasses;

                    break;
                }

            case 1: // Scott
                {
                    double s = getStdDev();

                    classesWidth = 3.5*s/Math.cbrt(N);
                    numberOfClasses =
                            (int)Math.round((maximum-minimum)/classesWidth);

                    break;
                }
            default:
                {
                    numberOfClasses =
                            (int)Math.round(Math.log(N)/Math.log(2)) + 1;
                    classesWidth = (maximum-minimum)/numberOfClasses;

                    break;
                }
        }

        histogram = new double[2][numberOfClasses];

        for(j=0; j<numberOfClasses; j++)
        {
            histogram[0][j] = minimum + (j+0.5)*classesWidth;
            histogram[1][j] = 0.0;
        }

        for(j=0; j<numberOfClasses; j++)
        {
            histogram[1][j] = 0.0;

            for(i=0; i<N; i++)
            {
                if((serialDataArray[i] >= minimum + j*classesWidth)
                    &&(serialDataArray[i] < minimum + (j+1)*classesWidth))
                {
                    histogram[1][j] = histogram[1][j] + 1.0;
                }
            }
        }

        for(j=0; j<numberOfClasses; j++)
        {
            histogram[1][j] = histogram[1][j]/(N*classesWidth);
        }

        return new Histogram(histogram);
    }

    public double[] getSerialData()
    {
        int counter = 0;
        int N = numberOfLines*numberOfColumns*numberOfPlanes;
        int x, y, z;
        double[] serialData = new double[N];

        for(z=0; z<numberOfPlanes; z++)
```

```
        {
            for(y=0; y<numberOfLines; y++)
            {
                for(x=0; x<numberOfColumns; x++)
                {
                    serialData[counter] = dataArray[y][x][z];
                    counter = counter + 1;
                }
            }
        }

        return serialData;
    }
}
```

```java
package functional.data;

//
//  Timeline.java
//
//
//  Created by José Gabriel Lira Gomes on 2/15/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class Timeline
{
    private double[] dataArray;

    public Timeline(double[] dataArray)
    {
        this.dataArray = dataArray;
    }

    public double[] getLaggedData(double[] statesVector, int delay)
    {
        int N = dataArray.length;
        int i, j;
        double[] subset;
        int subsetSize = 0;

        for(i=0; i<N; i++)
        {
            if((statesVector[i]==1)&(i+delay<N))
            {
                subsetSize = subsetSize + 1;
            }
        }

        subset = new double[subsetSize];

        j = 0;

        for(i=0; i<N; i++)
        {
            if((statesVector[i]==1)&(i+delay<N))
            {
                subset[j] = dataArray[i+delay];
                j = j + 1;
            }
        }

        return subset;
    }

    public double[] getLaggedData(int delay)
    {
        int N = dataArray.length;
        int i, j;
        double[] subset;
        int subsetSize = 0;

        for(i=0; i<N; i++)
        {
            if(i+delay<N)
            {
                subsetSize = subsetSize + 1;
            }
        }

        subset = new double[subsetSize];

        j = 0;

        for(i=0; i<N; i++)
        {
```

```
            if(i+delay<N)
            {
                subset[j] = dataArray[i+delay];
                j = j + 1;
            }
        }

        return subset;
    }
}
```

```java
package functional.design;

//
//  TimelineBuilder.java
//
//
//  Created by José Gabriel Lira Gomes on 7/3/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

public class TimelineBuilder
{
    private double[] timeline;

    public TimelineBuilder(int state, int start, int duration, int totalTime)
    {
        int i;

        timeline = new double[totalTime];

        if(start != 0)
        {
            for(i=0; i<start; i++)
            {
                timeline[i] = -1.0;
            }
        }
        for(i=start; i<start+duration; i++)
        {
            timeline[i] = (double)state;
        }
        for(i=start+duration; i<totalTime; i++)
        {
            timeline[i] = -1.0;
        }
    }

    public double[] getTimeline()
    {
        return timeline;
    }
}
```

```java
package fMRIWindows.design;


//
//  TimelineStateTableModel.java
//
//
//  Created by José Gabriel Lira Gomes on 7/5/06.
//  Copyright 2006 __FDG__. All rights reserved.
//


import java.util.Vector;

import javax.swing.table.AbstractTableModel;

import functional.design.StateRecord;

public class TimelineStateTableModel extends AbstractTableModel
{
    //
    //  Data Members
    //

    public static final int ACRONYM_INDEX = 0;
    public static final int NAME_INDEX = 1;

    protected String[] columnNames;
    protected Vector dataVector;

    //
    //  Constructors
    //
    //      TimelineStateTableModel(String[] columnNames)
    //

    public TimelineStateTableModel(String[] columnNames)
    {
        this.columnNames = columnNames;
        dataVector = new Vector();
    }

    public String getColumnName(int column)
    {
        return columnNames[column];
    }

    public boolean isCellEditable(int row, int column)
    {
        return false;
    }

    public Class getColumnClass(int column)
    {
        switch (column)
        {
        case ACRONYM_INDEX:
        case NAME_INDEX:
            return String.class;
        default:
            return Object.class;
        }
    }

    public Object getValueAt(int row, int column)
    {
        StateRecord record = (StateRecord)dataVector.get(row);
        switch (column)
        {
        case ACRONYM_INDEX:
            return record.getAcronym();
```

```
            case NAME_INDEX:
                return record.getName();
            default:
                return new Object();
            }
        }

        public void setValueAt(Object value, int row, int column)
        {
            StateRecord record = (StateRecord)dataVector.get(row);
            switch (column)
            {
            case ACRONYM_INDEX:
                record.setAcronym((String)value);
                break;
            case NAME_INDEX:
                record.setName((String)value);
                break;
            default:
                System.out.println("invalid index");
            }
            fireTableCellUpdated(row, column);
        }

        public int getRowCount()
        {
            return dataVector.size();
        }

        public int getColumnCount()
        {
            return columnNames.length;
        }

        public boolean hasEmptyRow()
        {
            if (dataVector.size() == 0) return false;

            StateRecord sttRecord =
            (StateRecord)dataVector.get(dataVector.size() - 1);
            if (sttRecord.getAcronym().trim().equals("")
                && sttRecord.getName().trim().equals(""))
            {
                return true;
            }
            else return false;
        }

        public void addEmptyRow()
        {
            dataVector.add(new StateRecord());
            fireTableRowsInserted(dataVector.size() - 1,
                                  dataVector.size() - 1);
        }
    }
```

```java
package fMRIWindows.design;

//
//  TimelineWindowAcquisitionParams.java
//
//
//  Created by Jose Gabriel Lira Gomes on 10/16/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.event.ActionEvent;

import javax.swing.JButton;
import javax.swing.JOptionPane;

import windowUtils.TextFieldsWindow;
import fMRIWindows.MainWindow;

public class TimelineWindowBlockAcquisitionParams extends TextFieldsWindow
{
    static String windowTitle = "Aquisicao";
    static String[] possibleFormats = {"TR (s)", "amostras/ciclo"};

    double acquisitionTime;
    int samplesPerCycle;

    private MainWindow parentWindow;

    public TimelineWindowBlockAcquisitionParams(MainWindow parentWindow)
    {
        super(windowTitle, possibleFormats);

        this.parentWindow = parentWindow;
    }

    public double getAcquisitionTime()
    {
        return acquisitionTime;
    }

    public int getSamplesPerCycle()
    {
        return samplesPerCycle;
    }

    public void actionPerformed(ActionEvent event)
    {
        JButton clickedButton = (JButton) event.getSource();

        if(clickedButton == acceptButton)
        {
            this.setVisible(false);

            try
            {
                acquisitionTime = Double.parseDouble(getTextValue(0));
                samplesPerCycle = Integer.parseInt(getTextValue(1));

                if((acquisitionTime > 0.0) && (samplesPerCycle > 0))
                {
                    System.out.println("TR = " + acquisitionTime + " s");
                    System.out.println("spc = " + samplesPerCycle);
                    System.out.println();

                    parentWindow.setAcquisitionTime(acquisitionTime);
                    parentWindow.setSamplesPerCycle(samplesPerCycle);
                    /*
                    if(parentWindow.imageHasBackground())
                    {
                        parentWindow.enableCleanBackgroundButton();
                    }
```

```
                    */
                    if(parentWindow.imageHasOutliers())
                    {
                        parentWindow.enableCleanOutliersButton();
                    }
                    if(parentWindow.imageHasLowFreqs())
                    {
                        parentWindow.enableCleanLowFreqsButton();
                    }
                    parentWindow.enableTumorSelector();
                    parentWindow.enableClustersSegmentButton();
                    parentWindow.enableMethodComboBox();
                    parentWindow.enableSetMethodParamsButton();
                    parentWindow.setVisible(true);
                }
                else
                {
                    JOptionPane.showMessageDialog(this,
                        "Os valores tem que ser positivos",
                        "Erro fisico",
                        JOptionPane.ERROR_MESSAGE);
                    this.setVisible(true);
                }
            }
            catch(NumberFormatException numberFormatException)
            {
                JOptionPane.showMessageDialog(this,
                        "Tem que inserir um duplo e um inteiro",
                        "Erro de formato",
                        JOptionPane.ERROR_MESSAGE);
                this.setVisible(true);
            }
        }
    }
}
```

```java
package fMRIWindows.design;

//
//  TimelineWindowAcquisitionParams.java
//
//
//  Created by Jose Gabriel Lira Gomes on 10/16/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.event.ActionEvent;

import javax.swing.JButton;
import javax.swing.JOptionPane;

import windowUtils.TextFieldsWindow;
import fMRIWindows.MainWindow;

public class TimelineWindowEventAcquisitionParams extends TextFieldsWindow
{
    static String windowTitle = "Aquisicao";
    static String[] possibleFormats = {"TR (s)"};

    double acquisitionTime;

    private MainWindow parentWindow;

    public TimelineWindowEventAcquisitionParams(MainWindow parentWindow)
    {
        super(windowTitle, possibleFormats);

        this.parentWindow = parentWindow;
    }

    public double getAcquisitionTime()
    {
        return acquisitionTime;
    }

    public void actionPerformed(ActionEvent event)
    {
        JButton clickedButton = (JButton) event.getSource();

        if(clickedButton == acceptButton)
        {
            this.setVisible(false);

            try
            {
                acquisitionTime = Double.parseDouble(getTextValue(0));

                if(acquisitionTime > 0.0)
                {
                    System.out.println("TR = " + acquisitionTime + " s");
                    System.out.println();

                    parentWindow.setAcquisitionTime(acquisitionTime);
                    /*
                    if(parentWindow.imageHasBackground())
                    {
                        parentWindow.enableCleanBackgroundButton();
                    }
                    */
                    if(parentWindow.imageHasOutliers())
                    {
                        parentWindow.enableCleanOutliersButton();
                    }
                    if(parentWindow.imageHasLowFreqs())
                    {
                        parentWindow.enableCleanLowFreqsButton();
                    }
```

```
                    parentWindow.enableMethodComboBox();
                    parentWindow.enableSetMethodParamsButton();
                    parentWindow.setVisible(true);
                }
                else
                {
                    JOptionPane.showMessageDialog(this,
                            "O tempo de aquisicao tem que ser positivo",
                            "Erro fisico",
                            JOptionPane.ERROR_MESSAGE);
                    this.setVisible(true);
                }
            }
            catch(NumberFormatException numberFormatException)
            {
                JOptionPane.showMessageDialog(this,
                            "Tem que inserir um duplo", "Erro de formato",
                            JOptionPane.ERROR_MESSAGE);
                this.setVisible(true);
            }
        }
    }
}
```

```java
package fMRIWindows.design;

//
//  TimelineWindowPeriod.java
//
//
//  Created by Jose Gabriel Lira Gomes on 6/26/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

import fMRIWindows.MenuedFrame;
import functional.design.TimelineBuilder;

public class TimelineWindowPeriod extends MenuedFrame
{
    //
    //  Data Members
    //

    private double[] periodTimeline;
    private double[] timeline;

    private Container periodContainer;

    private JPanel statePanel;
    private String[] stateBoxOptions;
    private JComboBox stateComboBox;
    private int stateValue;

    private JPanel startPanel;
    private JLabel startTitle;
    private JTextField startTextField;
    private int startValue;

    private JPanel durationPanel;
    private JLabel durationTitle;
    private JTextField durationTextField;
    private int durationValue;

    private JPanel acceptButtonPanel;
    private JButton acceptDataButton;

    private TimelineBuilder timelineBuild;

    private int totalDuration;

    private TimelineWindowSetter parentWindow;

    private int windowWidth = 200;
    private int windowHeight = 230;
    private int windowXPosition = 0;
    private int windowYPosition = 151;

    //
    //  Constructors
    //
    //      timelineWindowPeriod(String[] stateBoxOptions, int totalDuration)
    //

    public TimelineWindowPeriod(String[] stateBoxOptions,
```

```
                              int totalDuration,
                              TimelineWindowSetter parentWindow)
{
    this.stateBoxOptions = stateBoxOptions;
    this.totalDuration = totalDuration;
    this.parentWindow = parentWindow;

    createGUI();
}


/******* LISTEN BUTTON ACTION EVENT *******/

public void actionPerformed(ActionEvent event)
{
    JButton clickedButton = (JButton) event.getSource();

    if(clickedButton == acceptDataButton)
    {
        this.setVisible(false);

        try
        {
            timeline = parentWindow.getTimeline();

            stateValue = stateComboBox.getSelectedIndex();
            startValue = Integer.parseInt(startTextField.getText());
            durationValue = Integer.parseInt(durationTextField.getText());

            timelineBuild = new TimelineBuilder(stateValue,
                                                startValue,
                                                durationValue,
                                                totalDuration);
            periodTimeline = timelineBuild.getTimeline();
            timeline = overlapPeriodTimeline(periodTimeline, timeline);

            parentWindow.setTimeline(timeline);
            parentWindow.updateImage();
        }
        catch(NumberFormatException numberFormatException)
        {
            JOptionPane.showMessageDialog(this,
                                          "Tem que inserir dois inteiros",
                                          "Erro de formato",
                                          JOptionPane.ERROR_MESSAGE);
            this.setVisible(true);
        }
    }
}



private double[] overlapPeriodTimeline(double[] timelineToAdd,
                                       double[] originalTimeline)
{
    int timelineToAddSize = timelineToAdd.length;
    int originalTimelineSize = originalTimeline.length;

    double[] modifiedTimeline = new double[originalTimelineSize];

    int i;
    boolean noOverlap = true;

    if(timelineToAddSize==originalTimelineSize)
    {
        for(i=0; i<originalTimelineSize; i++)
        {
            if((timelineToAdd[i] >= 0)&(originalTimeline[i] >= 0))
            {
                noOverlap = false;
            }
```

```
            }
            if(noOverlap)
            {
                for(i=0; i<originalTimelineSize; i++)
                {
                    modifiedTimeline[i] = timelineToAdd[i]
                                          + originalTimeline[i] + 1;
                }
            }
            else
            {
                for(i=0; i<timelineToAddSize; i++)
                {
                    modifiedTimeline[i] = originalTimeline[i];
                }
                System.out.println
                ("erro na adicao de periodo: sobreposicao de estados");
            }
        }
        else
        {
            for(i=0; i<timelineToAddSize; i++)
            {
                modifiedTimeline[i] = originalTimeline[i];
            }

            System.out.println(
            "erro na adicao de periodo: arrays de diferente tamanho");
        }

        return modifiedTimeline;
    }



    private void createGUI()
    {
        statePanel = new JPanel();
        stateComboBox = new JComboBox(stateBoxOptions);
        statePanel.add(stateComboBox);

        startPanel = new JPanel();
        startTitle = new JLabel();
        startTitle.setText("Comeco");
        startTextField = new JTextField(2);
        startPanel.add(startTitle);
        startPanel.add(startTextField);

        durationPanel = new JPanel();
        durationTitle = new JLabel();
        durationTitle.setText("Duracao");
        durationTextField = new JTextField(2);
        durationPanel.add(durationTitle);
        durationPanel.add(durationTextField);

        acceptButtonPanel = new JPanel();
        acceptDataButton = new JButton("Aceitar");
        acceptButtonPanel.add(acceptDataButton);

        periodContainer = getContentPane();
        periodContainer.setLayout(new GridLayout(4, 1));

        periodContainer.add(statePanel);
        periodContainer.add(startPanel);
        periodContainer.add(durationPanel);
        periodContainer.add(acceptButtonPanel);

        acceptDataButton.addActionListener(this);

        setSize(windowWidth, windowHeight);
        setLocation(windowXPosition, windowYPosition);
```

```
            setTitle("Periodo");
            setVisible(true);
        }

    }
```

```java
package fMRIWindows.design;

//
//  TimelineWindowSetter.java
//
//
//  Created by Jose Gabriel Lira Gomes on 5/9/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

import imageUtils.BlockImagePanel;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.event.ActionEvent;

import javax.swing.JButton;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

import dirUtils.DirPathGetter;

import windowUtils.FormatWindow;
import windowUtils.RadioButtonsWindow;

import fMRIWindows.MainWindow;
import fMRIWindows.MenuedFrame;
import fileUtils.FilePathGetter;
import fileUtils.ReadTextNumberFile;
import fileUtils.WriteTextNumberFile;
import functional.data.DesignMatrix;
import functional.design.DesignMatrixBuilder;

public class TimelineWindowSetter extends MenuedFrame
{
    //
    //  Data Members
    //

    private DesignMatrix designMatrix;

    private Container timelineContainer;

    private BlockImagePanel timelinePanel;
    private JPanel acceptSequencePanel;

    private JButton loadTimelineButton;
    private JButton addPeriodButton;
    private JButton repeatBlockButton;
    private JButton cleanAllButton;
    private JButton saveTimelineButton;
    private JButton acceptSequenceButton;

    private FilePathGetter filePathGetter;
    private String filePath;
    private double[] fileData;

    private TimelineWindowPeriod periodWindow;
    private double[] timeline;
    private int blockStart;
    private int blockDuration;
    private double[] timelineBlock;

    private int numberOfStates;
    private int numberOfInstants;
    private String[] states;
    private int[] statesIndexes;

    private DesignMatrixBuilder designBuilder;

    private boolean firstTime = true;
```

```
private int numberOfTimelines = 1;
private int rectangleHeight = 25;
private int rectangleWidth = 4;

private RadioButtonsWindow formatWindow;
private JButton setFormatButton;

private DirPathGetter dirPathGetter;
private String dirPath;
private String outputPath;

private MainWindow parentWindow;

private int minWindowWidth = 150;
private int minWindowHeight = 150;
private int windowXPosition = 0;
private int windowYPosition = 21;

private int windowWidth;
private int windowHeight;

private int imagePanelWidth;
private int imagePanelHeight;
private int buttonsPanelHeight = 25;

//
//  Constructors
//
//      timelineWindowSetter(String[] states, int numberOfInstants)
//

public TimelineWindowSetter(String[] states,
                            int numberOfInstants,
                            MainWindow parentWindow)
{
    this.states = states;
    this.numberOfInstants = numberOfInstants;
    this.parentWindow = parentWindow;

    createGUI();
}

public double[] getTimeline()
{
    return timeline;
}

public void setTimeline(double[] timeline)
{
    this.timeline = timeline;
}

public DesignMatrix getDesignMatrix()
{
    return designMatrix;
}

public String[] getStatesAcronyms()
{
    return states;
}

public void updateImage()
{
    timelinePanel.updateHorizBlocks(timeline);
}

public void actionPerformed(ActionEvent event)
{
    JButton clickedButton = (JButton) event.getSource();
```

```
if(clickedButton == loadTimelineButton)
{
    this.setVisible(false);

    filePathGetter = new FilePathGetter();
    filePath = filePathGetter.getPath();

    try
    {
        fileData = new ReadTextNumberFile(filePath, "tml").getData();
        if(validTimeline())
        {
            JOptionPane.showMessageDialog(this,
                                "Numero de instantes diferente",
                                "Erro de ficheiro",
                                JOptionPane.ERROR_MESSAGE);
        }
        else
        {
            timeline = fileData;

            updateImage();
        }
    }
    catch(Exception e)
    {
        if(e.getMessage().equals("not tml"))
        {
            JOptionPane.showMessageDialog(this,
                                "Nao escolheu um *.tml",
                                "Erro de formato",
                                JOptionPane.ERROR_MESSAGE);
        }
    }

    this.setVisible(true);
}

if(clickedButton == addPeriodButton)
{
    if(firstTime)
    {
        periodWindow = new TimelineWindowPeriod(states,
                                                numberOfInstants,
                                                this);

        firstTime = false;
    }
    else
    {
        periodWindow.setVisible(true);
    }
}

if(clickedButton == repeatBlockButton)
{
    blockStart = getBlockStart(timeline);
    blockDuration = getBlockDuration(timeline, blockStart);
    timelineBlock = getBlock(timeline, blockStart, blockDuration);

    timeline = repeatBlock(timeline, blockStart, timelineBlock);

    updateImage();
}

if(clickedButton == cleanAllButton)
{
    timeline = resetTimeline(timeline);

    updateImage();
```

```
        }

        if(clickedButton == saveTimelineButton)
        {
            formatWindow = new FormatWindow();

            setFormatButton = formatWindow.acceptButton;
            setFormatButton.addActionListener(this);

            this.setVisible(false);
        }
        if(clickedButton == setFormatButton)
        {
            formatWindow.setVisible(false);

            dirPathGetter = new DirPathGetter();
            dirPath = dirPathGetter.getPath();

            outputPath = dirPath + "/timeline.tml";

            new WriteTextNumberFile(timeline,
                                    outputPath,
                                    formatWindow.getSelection());

            this.setVisible(true);
        }

        if(clickedButton == acceptSequenceButton)
        {
            this.setVisible(false);

            designBuilder = new DesignMatrixBuilder(timeline, statesIndexes);
            designMatrix = designBuilder.getDesignMatrix();

            parentWindow.setTimeline(timeline);
            parentWindow.setDesignMatrix(designMatrix);
            if(isBlockStudy())
            {
                parentWindow.setBlockStudy();
            }
            parentWindow.enableViewDesignButton();
            parentWindow.enableSetAcquisitionParamsButton();
            parentWindow.setVisible(true);
        }
    }


    private void createGUI()
    {
        imagePanelWidth = rectangleWidth*numberOfInstants + 40;
        imagePanelHeight = rectangleHeight*numberOfTimelines + 40;

        if(imagePanelWidth<=minWindowWidth)
        {
            windowWidth = minWindowWidth;
        }
        else
        {
            windowWidth = imagePanelWidth;
        }

        if(imagePanelHeight + buttonsPanelHeight<=minWindowHeight)
        {
            windowHeight = minWindowHeight - 20;
        }
        else
        {
            windowHeight = imagePanelHeight + buttonsPanelHeight - 20;
        }

        numberOfStates = states.length;
```

```
        int i;
        statesIndexes = new int[numberOfStates];

        for(i=0; i<numberOfStates; i++)
        {
            statesIndexes[i] = i;
        }

        timeline = new double[numberOfInstants];

        for(i=0; i<numberOfInstants; i++)
        {
            timeline[i] = -1;
        }

        timelineContainer = getContentPane();
        timelineContainer.setLayout(new BorderLayout());

        timelinePanel = new BlockImagePanel(imagePanelWidth,
                                            imagePanelHeight,
                                            rectangleWidth,
                                            rectangleHeight,
                                            numberOfStates);
        timelinePanel.setHorizBlocks(timeline);
        acceptSequencePanel = new JPanel();

        loadTimelineButton = new JButton("Carregar");
        addPeriodButton = new JButton("Adicionar");
        repeatBlockButton = new JButton("Repetir");
        cleanAllButton = new JButton("Limpar");
        saveTimelineButton = new JButton("Guardar");
        acceptSequenceButton = new JButton("Aceitar dados");

        loadTimelineButton.addActionListener(this);
        addPeriodButton.addActionListener(this);
        repeatBlockButton.addActionListener(this);
        cleanAllButton.addActionListener(this);
        saveTimelineButton.addActionListener(this);
        acceptSequenceButton.addActionListener(this);

        acceptSequencePanel.add(loadTimelineButton);
        acceptSequencePanel.add(addPeriodButton);
        acceptSequencePanel.add(repeatBlockButton);
        acceptSequencePanel.add(cleanAllButton);
        acceptSequencePanel.add(saveTimelineButton);
        acceptSequencePanel.add(acceptSequenceButton);

        timelineContainer.add(timelinePanel, BorderLayout.CENTER);
        timelineContainer.add(acceptSequencePanel, BorderLayout.SOUTH);

        setSize(windowWidth, windowHeight);
        setLocation(windowXPosition, windowYPosition);
        setTitle("Sequencia temporal");
        setVisible(true);
    }

    private boolean validTimeline()
    {
        boolean valid = false;
        int numberOfStates = states.length;
        double[] statesPossibleValues = new double[numberOfStates];
        int i, j;
        boolean validSize = false;
        boolean validValues = true;

        for(i=0; i<numberOfStates; i++)
        {
            statesPossibleValues[i] = (double)i;
        }
```

```
        if(fileData.length == numberOfInstants)
        {
            validSize = true;
            for(j=0; j<numberOfInstants; j++)
            {
                for(i=0; i<numberOfStates; i++)
                {
                    if(fileData[j] != statesPossibleValues[i])
                    {
                        validValues = false;
                    }
                }
            }
        }

        if((validSize)&&(validValues))
        {
            valid = true;
        }

        return valid;
    }

    private int getBlockStart(double[] timelineArray)
    {
        int arraySize = timelineArray.length;
        int i;
        boolean noFlag = true;
        int startIndex = 0;

        for(i=0; i<arraySize; i++)
        {
            if(timelineArray[i]>-1)
            {
                if(noFlag)
                {
                    noFlag = false;
                    startIndex = i;
                }
            }
        }

        return startIndex;
    }

    private int getBlockDuration(double[] timelineArray, int start)
    {
        int arraySize = timelineArray.length;
        int i;
        boolean lastFlag = false;
        int duration = 0;

        for(i=start; i<arraySize; i++)
        {
            if((timelineArray[i]!=-1)&(!lastFlag))
            {
                duration = duration + 1;
            }
            else
            {
                lastFlag = true;
            }
        }
        return duration;
    }


    private double[] getBlock(double[] timelineArray,
                              int start,
                              int duration)
    {
```

```
    int i;
    double[] block = new double[duration];

    for(i=0; i<duration; i++)
    {
        block[i] = timelineArray[i+start];
    }

    return block;
}


private double[] repeatBlock(double[] timelineArray,
                             int start,
                             double[] block)
{
    int arraySize = timelineArray.length;
    int duration = block.length;
    int i;
    int j = 0;
    double[] modTimeline = new double[arraySize];
    //int[] invertedBlock;

    int currentIndex;

    //int leftLength;
    int rightLength;
    int numberOfLoops;

    //leftLength = start;
    rightLength = arraySize - start;
    //invertedBlock = invertArray(block);

    numberOfLoops = (int) Math.ceil((double)rightLength/(double)duration);

    if(start != 0)
    {
        for(i=0; i<start; i++)
        {
            modTimeline[i] = timelineArray[i];
        }
        for(j=0; j<numberOfLoops; j++)
        {
            for(i=0; i<duration; i++)
            {
                currentIndex = j*duration+i+start;
                if(currentIndex < arraySize)
                {
                    modTimeline[currentIndex] = block[i];
                }
            }
        }
    }
    else
    {
        for(j=0; j<Math.ceil(rightLength/duration); j++)
        {
            for(i=0; i<duration; i++)
            {
                currentIndex = j*duration+i;
                if(currentIndex < arraySize)
                {
                    modTimeline[currentIndex] = block[i];
                }
            }
        }
    }

    return modTimeline;
}
```

```
private double[] resetTimeline(double[] timelineArray)
{
    int i;

    for(i=0; i<numberOfInstants; i++)
    {
        timelineArray[i] = -1;
    }

    return timelineArray;
}

private boolean isBlockStudy()
{
    int firstStateDuration = 0;
    int i, j;
    double firstState = timeline[0];
    boolean stateChanged = false;
    double numberOfCycles;
    boolean blockStudy = true;

    for(i=0; i<numberOfInstants; i++)
    {
        if((timeline[i] == firstState)&&(!stateChanged))
        {
            firstStateDuration = firstStateDuration + 1;
        }
        else
        {
            stateChanged = true;
        }
    }

    numberOfCycles = (double)numberOfInstants/(double)firstStateDuration;

    if(numberOfCycles == Math.round(numberOfCycles))
    {
        for(j=0; j<(int)numberOfCycles; j++)
        {
            firstState = timeline[j*firstStateDuration];
            for(i=0; i<firstStateDuration; i++)
            {
                if((timeline[i+j*firstStateDuration]==firstState)
                    &&(blockStudy))
                {
                    blockStudy = true;
                }
                else
                {
                    blockStudy = false;
                }
            }
        }
    }
    else
    {
        blockStudy = false;
    }

    return blockStudy;
}

/******* SHOW ARRAY ******

private void showArray(int[] array)
{
    int size = array.length;
    int i;

    for(i=0; i<size; i++)
```

```
            {
                System.out.println(array[i]);
            }
        }
        */
    }
```

```java
package fMRIWindows.design;

//
//  TimelineWindowStates.java
//
//
//  Created by Jose Gabriel Lira Gomes on 6/26/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.ActionEvent;

import javax.swing.JButton;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.DefaultTableCellRenderer;

import fMRIWindows.MainWindow;
import fMRIWindows.MenuedFrame;

public class TimelineWindowStates extends MenuedFrame
{
    //
    //  Data Members
    //

    private String[][] statesData;

    private Container setStatesContainer;

    private JPanel acceptStatesPanel;
    private JTextField stateAcronTextField;
    private JTextField stateNameTextField;
    public JButton acceptStatesButton;
    private JButton addStateButton;

    protected JTable statesTable;
    protected JScrollPane listStatesPane;
    protected TimelineStateTableModel statesTableModel;
    private timelineStateTableModelListener statesTableModelListener;

    private String[] columnNames = {"Sigla", "Nome"};

    private String acronValue;
    private String nameValue;
    private int acronNumberOfLetters;
    private int nameNumberOfLetters;
    private String[] stateLine = new String[2];
    private int currentLine = 0;

    private boolean validState = false;

    private MainWindow parentWindow;

    private int windowWidth = 500;
    private int windowHeight = 250;
    private int windowXPosition = 0;
    private int windowYPosition = 21;

    //
    //  Constructors
    //
```

```
//      timelineWindowStates()
//

public TimelineWindowStates(MainWindow parentWindow)
{
    this.parentWindow = parentWindow;

    createGUI();
}

public String[][] getStatesData()
{
    return statesData;
}

/******* LISTEN BUTTON ACTION EVENT *******/

public void actionPerformed(ActionEvent event)
{
    JButton clickedButton = (JButton) event.getSource();

    if(clickedButton == addStateButton)
    {
        acronValue = stateAcronTextField.getText();
        nameValue = stateNameTextField.getText();

        acronNumberOfLetters = getNumberOfLetters(acronValue);
        nameNumberOfLetters = getNumberOfLetters(nameValue);

        if((acronNumberOfLetters == 3)
            &&(nameNumberOfLetters<26)
            &&(nameNumberOfLetters>0))
        {
            validState = stateIsValid();

            if(validState)
            {
                statesTableModel.setValueAt(acronValue, currentLine, 0);
                statesTableModel.setValueAt(nameValue, currentLine, 1);
                statesTableModel.addEmptyRow();

                stateLine[0] = acronValue;
                stateLine[1] = nameValue;

                if(currentLine == 0)
                {
                    statesData = addFirstLineToDualArray(2, stateLine);
                }
                else
                {
                    statesData = addLineToDualArray(statesData, stateLine);
                }
                currentLine = currentLine + 1;
            }
            else
            {
                JOptionPane.showMessageDialog(this,
                                        "Sigla ou nome ja existente",
                                        "Erro de design",
                                        JOptionPane.ERROR_MESSAGE);
            }
        }
        else
        {
            JOptionPane.showMessageDialog(this,
                "A sigla tem que ter 3 letras e o nome de 1 a 25 letras",
                "Erro de design",
                JOptionPane.ERROR_MESSAGE);
        }
    }
    if(clickedButton == acceptStatesButton)
```

```
    {
        if(statesData.length>=2)
        {
            this.setVisible(false);

            parentWindow.setStatesData(statesData);
            parentWindow.enableSetSequenceButton();
            parentWindow.setVisible(true);
        }
        else
        {
            JOptionPane.showMessageDialog(this,
                "Tem que ter pelo menos 2 estados",
                "Erro de design",
                JOptionPane.ERROR_MESSAGE);
        }
    }
}


public void highlightLastRow(int row)
{
        int lastrow = statesTableModel.getRowCount();
        if (row == lastrow - 1)
        {
            statesTable.setRowSelectionInterval(lastrow - 1, lastrow - 1);
        }
        else
        {
            statesTable.setRowSelectionInterval(row + 1, row + 1);
        }
        statesTable.setColumnSelectionInterval(0, 0);
    }

class interactiveRenderer extends DefaultTableCellRenderer
{
        /**
     *
     */
    private static final long serialVersionUID = 1L;

    protected int interactiveColumn;

        public interactiveRenderer(int interactiveColumn)
        {
            this.interactiveColumn = interactiveColumn;
        }

        public Component getTableCellRendererComponent(JTable table,
        Object value,
        boolean isSelected,
        boolean hasFocus,
        int row,
        int column)
        {
            Component c = super.getTableCellRendererComponent(table,
                                                              value,
                                                              isSelected,
                                                              hasFocus,
                                                              row,
                                                              column);
            if (column == interactiveColumn && hasFocus)
            {
                if ((TimelineWindowStates.
                this.statesTableModel.getRowCount() - 1)
                == row && !TimelineWindowStates.
                 this.statesTableModel.hasEmptyRow())
                {
                    TimelineWindowStates.
                    this.statesTableModel.addEmptyRow();
                }
```

```
                highlightLastRow(row);
            }

            return c;
        }
    }

public class timelineStateTableModelListener implements TableModelListener
{
    public void tableChanged(TableModelEvent evt)
    {
        if (evt.getType() == TableModelEvent.UPDATE)
        {
            int column = evt.getColumn();
            int row = evt.getFirstRow();

            //System.out.println("col: " + column);
            //System.out.println("row: " + row);
        }
    }
}

private void createGUI()
{
    statesTableModel = new TimelineStateTableModel(columnNames);
    statesTableModelListener = new timelineStateTableModelListener();
    statesTableModel.addTableModelListener(statesTableModelListener);

    statesTable = new JTable();
    statesTable.setModel(statesTableModel);
    statesTable.setSurrendersFocusOnKeystroke(true);

    if (!statesTableModel.hasEmptyRow())
    {
        statesTableModel.addEmptyRow();
    }

    listStatesPane = new JScrollPane(statesTable);
    statesTable.setPreferredScrollableViewportSize(new Dimension(250, 300));

    setStatesContainer = getContentPane();
    setStatesContainer.setLayout(new BorderLayout());

    acceptStatesPanel = new JPanel();
    stateAcronTextField = new JTextField(2);
    stateNameTextField = new JTextField(14);
    addStateButton = new JButton("Adicionar");
    acceptStatesButton = new JButton("Aceitar");
    acceptStatesPanel.add(stateAcronTextField);
    acceptStatesPanel.add(stateNameTextField);
    acceptStatesPanel.add(addStateButton);
    acceptStatesPanel.add(acceptStatesButton);
    addStateButton.addActionListener(this);
    acceptStatesButton.addActionListener(this);
    acceptStatesButton.addActionListener(parentWindow);

    setStatesContainer.add(listStatesPane, BorderLayout.CENTER);
    setStatesContainer.add(acceptStatesPanel, BorderLayout.NORTH);

    setSize(windowWidth, windowHeight);
    setLocation(windowXPosition, windowYPosition);
    setTitle("Estados");
    setVisible(true);
}

private boolean stateIsValid()
{
    boolean validState = true;
    int i;
```

```
        if(currentLine > 0)
        {
            for(i=0; i<currentLine; i++)
            {
                if(statesData[i][0].equals(acronValue))
                {
                    validState = false;
                }
                if(statesData[i][1].equals(nameValue))
                {
                    validState = false;
                }
            }
        }

        return validState;
    }

    private int getNumberOfLetters(String word)
    {
        int numberOfLetters;
        char letter;
        int i;

        numberOfLetters = word.length();

        for(i=0; i<numberOfLetters; i++)
        {
            letter = word.charAt(i);
            if(letter == ' ')
            {
                numberOfLetters = numberOfLetters - 1;
            }
        }

        return numberOfLetters;
    }

    private String[][] addFirstLineToDualArray(int dualArrayNumberOfColumns,
                                               String[] line)
    {
        String[][] doubleArray = new String[1][dualArrayNumberOfColumns];
        int i;

        for(i=0; i<dualArrayNumberOfColumns; i++)
        {
            doubleArray[0][i] = line[i];
        }
        return doubleArray;
    }

    private String[][] addLineToDualArray(String[][] dualArray, String[] line)
    {
            int dualArrayNumberOfLines = dualArray.length;
            int dualArrayNumberOfColumns = dualArray[0].length;
            int lineNumberOfColumns = line.length;
            int i, j;

            String[][] modDualArray =
            new String[dualArrayNumberOfLines+1][dualArrayNumberOfColumns];

            if(dualArrayNumberOfColumns == lineNumberOfColumns)
            {
                for(j=0; j<dualArrayNumberOfColumns; j++)
                {
                    for(i=0; i<dualArrayNumberOfLines; i++)
                    {
                        modDualArray[i][j] = dualArray[i][j];
                    }
                }
```

```
                for(j=0; j<dualArrayNumberOfColumns; j++)
                {
                    modDualArray[dualArrayNumberOfLines][j] = line[j];
                }
            }

            return modDualArray;
    }
}
```

```
package windowUtils;

//
//  TripleSliderPanel.java
//
//
//  Created by Jose Gabriel Lira Gomes on 10/21/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.Dimension;

import javax.swing.JPanel;
import javax.swing.JSlider;

public class TripleSliderPanel extends JPanel
{
    public JSlider xSlider;
    public JSlider ySlider;
    public JSlider zSlider;

    private int x, y, z;

    private int width;
    private int height;
    private int numberOfSlices;


    public TripleSliderPanel(int width, int height, int numberOfSlices)
    {
        this.width = width;
        this.height = height;
        this.numberOfSlices = numberOfSlices;

        this.setPreferredSize(new Dimension(180,200));

        x = (int)Math.round(width/2);
        y = (int)Math.round(height/2);
        z = (int)Math.round(numberOfSlices/2);

        createSlidersPanel();
    }


    private void createSlidersPanel()
    {
        xSlider = new JSlider();
        xSlider.setOrientation(JSlider.VERTICAL);
        xSlider.setPaintLabels(true);
        xSlider.setPaintTicks(true);
        xSlider.setMinimum(0);
        xSlider.setMaximum(width-1);
        xSlider.setValue(x);
        xSlider.setMajorTickSpacing(10);
        xSlider.setMinorTickSpacing(5);

        ySlider = new JSlider();
        ySlider.setOrientation(JSlider.VERTICAL);
        ySlider.setPaintLabels(true);
        ySlider.setPaintTicks(true);
        ySlider.setMinimum(0);
        ySlider.setMaximum(height-1);
        ySlider.setValue(y);
        ySlider.setMajorTickSpacing(10);
        ySlider.setMinorTickSpacing(5);

        zSlider = new JSlider();
        zSlider.setOrientation(JSlider.VERTICAL);
        zSlider.setPaintLabels(true);
        zSlider.setPaintTicks(true);
        zSlider.setMinimum(0);
```

```
            zSlider.setMaximum(numberOfSlices-1);
            zSlider.setValue(z);
            zSlider.setMajorTickSpacing(10);
            zSlider.setMinorTickSpacing(5);

            this.add(xSlider);
            this.add(ySlider);
            this.add(zSlider);
        }
    }
```

```java
package functional.methods;

//
//  TStudentAnalysis.java
//
//
//  Created by Jose Gabriel Lira Gomes on 7/26/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import estatistica.tests.TStudentTest;
import functional.data.DesignMatrix;
import functional.data.Mask;
import functional.data.Timeline;

public class TStudentAnalysis
{
    //
    //  Data Members
    //

    private double[][][] tAnalysisResultData;
    private double[][][] tValueResultData;

    private double[][][][] imageData;
    private Mask dataMask;
    private DesignMatrix designMatrix;
    private int[] statesID;
    private int lag;

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfInstants;

    //
    //  Constructors
    //

    public TStudentAnalysis(double[][][][] imageData,
                            Mask dataMask,
                            DesignMatrix designMatrix,
                            int[] statesID,
                            int lag)
    {
        this.imageData = imageData;
        this.dataMask = dataMask;
        this.designMatrix = designMatrix;
        this.statesID = statesID;
        this.lag = lag;

        width = imageData.length;
        height = imageData[0].length;
        numberOfSlices = imageData[0][0].length;
        numberOfInstants = imageData[0][0][0].length;

        performTStudentAnalysis();
    }

    //
    //  Public Methods
    //
    //      double[][][] getResultData()
    //

    public double[][][] getResultData()
    {
        return tAnalysisResultData;
    }

    //
```

```
//  Private Methods
//

private void performTStudentAnalysis()
{
    tAnalysisResultData = new double[width][height][numberOfSlices];
    tValueResultData = new double[width][height][numberOfSlices];

    Timeline voxelTimeline;

    double[] firstStateVector;
    double[] secndStateVector;

    int firstStateIndex;
    int secndStateIndex;

    firstStateIndex = statesID[0];
    secndStateIndex = statesID[1];

    double[] timeArray;
    double[] firstStateDataArray;
    double[] secndStateDataArray;

    int x, y, z;

    TStudentTest tStudentTester;
    double tProb;
    double tValue;

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                if(dataMask.getBooleanValue(x, y, z))
                {
                    timeArray = extractTimeArray(x, y, z, imageData);

                    voxelTimeline = new Timeline(timeArray);

                    firstStateVector =
                        designMatrix.getStateVector(firstStateIndex);
                    secndStateVector =
                        designMatrix.getStateVector(secndStateIndex);

                    firstStateDataArray =
                        voxelTimeline.getLaggedData(firstStateVector, lag);
                    secndStateDataArray =
                        voxelTimeline.getLaggedData(secndStateVector, lag);

                    tStudentTester = new TStudentTest(firstStateDataArray,
                                                      secndStateDataArray);
                    tProb = tStudentTester.getP();
                    tValue = tStudentTester.getT();

                    tAnalysisResultData[x][y][z] = tProb;
                    tValueResultData[x][y][z] = tValue;
                }
                else
                {
                    tAnalysisResultData[x][y][z] = 0.0;
                    tValueResultData[x][y][z] = 0.0;
                }
            }
        }
    }
    resultMaxValue(tValueResultData);
    resultMinValue(tValueResultData);
}
```

```
/******* EXTRACT TIME ARRAY ******/

private double[] extractTimeArray(int x,
                                  int y,
                                  int z,
                                  double dataMatrix[][][][])
{
    int t;
    double[] timeArray = new double[numberOfInstants];

    for(t=0; t< numberOfInstants; t++)
    {
        timeArray[t] = dataMatrix[x][y][z][t];
    }
    return timeArray;
}

/******* RESULT MAXIMUM VALUE *******/

private void resultMaxValue(double resultMatrix[][][])
{
    int x, y, z;
    int xx=0;
    int yy=0;
    int zz=0;
    double elementValue;
    double maxValue=resultMatrix[0][0][0];

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
            for(x=0; x< width; x++)
            {
                elementValue = resultMatrix[x][y][z];
                if(elementValue > maxValue)
                {
                    maxValue = elementValue;
                    xx = x;
                    yy = y;
                    zz = z;
                }
            }
        }
    }

    System.out.println("Max( p(x,y,z) ) = p("
                       + xx + "," + yy + "," + zz + ") = "
                       + tAnalysisResultData[xx][yy][zz]);

}



/******* RESULT MINIMUM VALUE *******/

private void resultMinValue(double resultMatrix[][][])
{
    int x, y, z;
    int xx=0;
    int yy=0;
    int zz=0;
    double elementValue;
    double minValue=resultMatrix[0][0][0];

    for(z=0; z< numberOfSlices; z++)
    {
        for(y=0; y< height; y++)
        {
```

```
                for(x=0; x< width; x++)
                {
                    elementValue = resultMatrix[x][y][z];
                    if(elementValue < minValue)
                    {
                        minValue = elementValue;
                        xx = x;
                        yy = y;
                        zz = z;
                    }
                }
            }
        }

        System.out.println("Min( p(x,y,z) ) = p("
                           + xx + "," + yy + "," + zz + ") = "
                           + tAnalysisResultData[xx][yy][zz]);

    }

    /******* SHOW ARRAY *******/
    /*
    private void showArray(double[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */
}
```

```java
package estatistica.distributions;

//
//  TStudentDistribution.java
//
//
//  Created by José Gabriel Lira Gomes on 8/1/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import matematica.specialFunctions.Gamma;
import matematica.specialFunctions.IncBeta;

public class TStudentDistribution
{
    private double df;

    public TStudentDistribution(double df)
    {
        this.df = df;
    }

    public double getProbDensity(double value)
    {
        double gammaOne;
        double gammaTwo;

        double probDensity;

        gammaOne = new Gamma((df+1.0)/2.0).getGamma();
        gammaTwo = new Gamma(df/2.0).getGamma();

        probDensity = gammaOne/(gammaTwo*Math.pow(1.0 + value*value/df,
                                    (df+1.0)/2.0)*Math.sqrt(df*Math.PI));

        return probDensity;
    }

    public double getCumProb(double value)
    {
        return getCumProb(value, 0);
    }

    public double getCumProb(double value, int type)
    {
        IncBeta incBetaCalcer;
        double prob;

        //System.out.println("df: " + df + "; value: " + value + ";");

        if(type == 0)
        {
            incBetaCalcer = new IncBeta(0.5*df,
                                        0.5,
                                        df/(df+value*value));
        }
        else
        {
            incBetaCalcer = new IncBeta(0.5*df,
                                        0.5,
                                        df/(df+Math.sqrt(Math.abs(value))));
        }
        if(value >= 0)
        {
            prob = 1.0 - 0.5*(incBetaCalcer.getIncBeta());
        }
        else
        {
            prob = 0.5*(incBetaCalcer.getIncBeta());
        }
```

```
            return prob;
        }
    }
```

```java
package estatistica.tests;

//
//  TStudentTest.java
//
//
//  Created by Jose Gabriel Lira Gomes on 7/26/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import estatistica.descriptive.OneDArrayAnalyzer;
import estatistica.distributions.TStudentDistribution;

public class TStudentTest
{
    private double firstAverage;
    private double firstVariance;

    private double secndAverage;
    private double secndVariance;

    private double N;
    private double M;

    private double TINY=1.0e-20;



    public TStudentTest(double firstAverage,
                        double secndAverage,
                        double firstVariance,
                        double secndVariance,
                        double N,
                        double M)
    {
        this.firstAverage = firstAverage;
        this.secndAverage = secndAverage;

        this.firstVariance = firstVariance;
        this.secndVariance = secndVariance;

        this.N = N;
        this.M = M;
    }

    public TStudentTest(double[] firstArray, double[] secndArray)
    {
        int sizeOfFirstArray = firstArray.length;
        int sizeOfSecndArray = secndArray.length;

        OneDArrayAnalyzer firstAnalyzer;
        OneDArrayAnalyzer secndAnalyzer;

        N = (double)sizeOfFirstArray;
        M = (double)sizeOfSecndArray;

        firstAnalyzer = new OneDArrayAnalyzer(firstArray);
        secndAnalyzer = new OneDArrayAnalyzer(secndArray);

        firstAverage = firstAnalyzer.getMean();
        secndAverage = secndAnalyzer.getMean();

        firstVariance = firstAnalyzer.getVariance();
        secndVariance = secndAnalyzer.getVariance();
    }

    public double getT()
    {
        double t;

        if((firstVariance == 0)&(secndVariance == 0))
```

```
        {
            if(firstAverage == secndAverage)
            {
                t = 0;
            }
            else
            {
                t = (firstAverage - secndAverage)
                    /Math.sqrt(firstVariance/N + secndVariance/M + TINY);
            }
        }
        else
        {
            t = (firstAverage - secndAverage)
                /Math.sqrt(firstVariance/N + secndVariance/M);
        }


        return t;
    }

    public double getP()
    {
        double t;
        double df;
        int type;
        TStudentDistribution tDist;
        boolean variancesAreDifferent;
        double tTestProb;

        t = getT();
        variancesAreDifferent = compareVariances();

        if(variancesAreDifferent)
        {
            df = Math.sqrt(firstVariance/N + secndVariance/M)
            /(Math.sqrt(firstVariance/N)/(N-1)
                + Math.sqrt(secndVariance/M)/(M-1));
            type = 1;
        }
        else
        {
            df = N + M - 2;
            type = 0;
        }
        tDist = new TStudentDistribution(df);
        tTestProb = tDist.getCumProb(t, type);

        return tTestProb;
    }

    private boolean compareVariances()
    {
        double fProb;
        boolean different;
        double criticalProb = 0.05;

        fProb = new FTest(firstVariance, secndVariance, N, M).getP();

        if(fProb < criticalProb)
        {
            different = true;
        }
        else
        {
            different = false;
        }

        return different;
    }
}
```

```
package estatistica.descriptive;

//
//  TwoDArrayAnalyzer.java
//
//
//  Created by José Gabriel Lira Gomes on 2/8/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import estatistica.distributions.Histogram;

public class TwoDArrayAnalyzer
{
    private double dataArray[][];
    private int numberOfLines;
    private int numberOfColumns;

    public TwoDArrayAnalyzer(double dataArray[][])
    {
        this.dataArray = dataArray;
        this.numberOfLines = dataArray.length;
        this.numberOfColumns = dataArray[0].length;
    }

    /******* GET MEAN ******/

    public double getMean()
    {
        int N;
        int x, y;
        double dataSum=0;
        double mean;

        N = numberOfLines*numberOfColumns;

        for(y=0; y<numberOfLines; y++)
        {
            for(x=0; x<numberOfColumns; x++)
            {
                dataSum = dataSum + dataArray[y][x];
            }
        }
        mean = dataSum/(double)N;

        return mean;
    }

    public double getStdDev()
    {
        double stdDev;

        stdDev = Math.sqrt(getVariance());

        return stdDev;
    }

    /******* GET VARIANCE ******/

    public double getVariance()
    {
        int x, y;
        double dev;
        double devSum=0.0;
        double devSquareSum=0.0;
        int N;
        double dN;
        double mean;
        double variance;

        N = numberOfColumns*numberOfLines;
```

```
    dN = (double)N;

    mean = getMean();

    for(y=0; y<numberOfLines; y++)
    {
        for(x=0; x<numberOfColumns; x++)
        {
            dev = dataArray[y][x]-mean;
            devSum = devSum + dev;
            devSquareSum = devSquareSum + dev*dev;
        }
    }

    variance = (devSquareSum - devSum*devSum/dN)/(dN-1);

    return variance;
}


/******* RESULT MAXIMUM VALUE *******/

public double getMaximum()
{
    int y, x;
    double elementValue;
    double maxValue=dataArray[0][0];

    for(x=0; x< numberOfColumns; x++)
    {
        for(y=0; y< numberOfLines; y++)
        {
            elementValue = dataArray[y][x];
            if(elementValue > maxValue)
            {
                maxValue = elementValue;
            }
        }
    }

    return maxValue;
}



/******* RESULT MINIMUM VALUE *******/

public double getMinimum()
{
    int y, x;
    double elementValue;
    double minValue=dataArray[0][0];

    for(x=0; x< numberOfColumns; x++)
    {
        for(y=0; y< numberOfLines; y++)
        {
            elementValue = dataArray[y][x];
            if(elementValue < minValue)
            {
                minValue = elementValue;
            }
        }
    }

    return minValue;
}

public Histogram getHistogram(int binningMethod)
{
    double[][] histogram;
```

```
double[] serialDataArray;
double N;
double minimum = getMinimum();
double maximum = getMaximum();
int numberOfClasses;
double classesWidth;

int i, j;

serialDataArray = getSerialData();

N = (double)serialDataArray.length;

switch (binningMethod)
{
    case 0: // Sturge
        {
            numberOfClasses =
                (int)Math.round(Math.log(N)/Math.log(2)) + 1;
            classesWidth = (maximum-minimum)/numberOfClasses;

            break;
        }

    case 1: // Scott
        {
            double s = getStdDev();

            classesWidth = 3.5*s/Math.cbrt(N);
            numberOfClasses =
                (int)Math.round((maximum-minimum)/classesWidth);

            break;
        }
    default:
        {
            numberOfClasses =
                (int)Math.round(Math.log(N)/Math.log(2)) + 1;
            classesWidth = (maximum-minimum)/numberOfClasses;

            break;
        }
}

histogram = new double[2][numberOfClasses];

for(j=0; j<numberOfClasses; j++)
{
    histogram[0][j] = minimum + (j+0.5)*classesWidth;
    histogram[1][j] = 0.0;
}

for(j=0; j<numberOfClasses; j++)
{
    histogram[1][j] = 0.0;

    for(i=0; i<N; i++)
    {
        if((serialDataArray[i] >= minimum + j*classesWidth)
            &&(serialDataArray[i] < minimum + (j+1)*classesWidth))
        {
            histogram[1][j] = histogram[1][j] + 1.0;
        }
    }
}

for(j=0; j<numberOfClasses; j++)
{
    histogram[1][j] = histogram[1][j]/(N*classesWidth);
}
```

```
        return new Histogram(histogram);
    }

    public double[] getSerialData()
    {
        int counter = 0;
        int N = numberOfLines*numberOfColumns;
        int x, y;
        double[] serialData = new double[N];

        for(y=0; y<numberOfLines; y++)
        {
            for(x=0; x<numberOfColumns; x++)
            {
                serialData[counter] = dataArray[y][x];
                counter = counter + 1;
            }
        }

        return serialData;
    }
}
```

```java
package functional.design;

import estatistica.descriptive.OneDArrayAnalyzer;

//
//  VectorOverlapper.java
//
//
//  Created by José Gabriel Lira Gomes on 2/15/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class VectorOverlapper
{
    //
    //  Data Members
    //

    private double maxOverlapPosition;
    private double maxOverlapValue;

    private double[] vectorOne;
    private double[] vectorTwo;
    private int maxShift;

    private int vectorOneDim;
    private int vectorTwoDim;

    //
    //  Constructors
    //
    //      VectorOverlapper(double[] vectorOne,
    //                       double[] vectorTwo,
    //                       int maxShift)
    //

    public VectorOverlapper(double[] vectorOne,
                            double[] vectorTwo,
                            int maxShift)
                                throws Exception
    {
        this.vectorOne = vectorOne;
        this.vectorTwo = vectorTwo;
        this.maxShift = maxShift;

        this.vectorOneDim = vectorOne.length;
        this.vectorTwoDim = vectorTwo.length;

        if(vectorOneDim != vectorTwoDim)
        {
            throw new Exception("differently sized vectors");
        }

        getMaxOverlapData();
    }

    //
    //  Public Methods
    //
    //      double getMaxOverlapValue()
    //      double getMaxOverlapPosition()
    //

    public double getMaxOverlapValue()
    {
        return maxOverlapValue;
    }

    public double getMaxOverlapPosition()
    {
        return maxOverlapPosition;
```

```
}

public double getStdDev()
{
    double[] shiftedVectorOne;
    double[] shiftedVectorTwo;

    double vectorOneMin;
    double vectorTwoMin;

    double[] differenceVector = new double[vectorOneDim];
    double[] sumVector = new double[vectorOneDim];
    double[] dispersionVector = new double[vectorOneDim];
    int i;

    vectorOneMin = new OneDArrayAnalyzer(vectorOne).getMinimum();
    vectorTwoMin = new OneDArrayAnalyzer(vectorTwo).getMinimum();

    shiftedVectorOne = shiftRight(vectorOne,
                                  (int)Math.round(maxOverlapPosition));
    if(vectorTwoMin > -vectorOneMin)
    {
        shiftedVectorTwo = vectorTwo;
    }
    else
    {
        shiftedVectorTwo = shiftUp(vectorTwo, -vectorOneMin+1.0);
    }

    for(i=0; i<vectorOneDim; i++)
    {
        differenceVector[i] = shiftedVectorOne[i] - shiftedVectorTwo[i];
        sumVector[i] = shiftedVectorOne[i] + shiftedVectorTwo[i];
        dispersionVector[i] = differenceVector[i]/sumVector[i];
    }

    return new OneDArrayAnalyzer(dispersionVector).getStdDev();
}

//
//  Private Methods
//
//      double[] shiftRight(double[] dataArray, int shiftAmount)
//

private void getMaxOverlapData()
{
    double[] productSum;
    int i, j;

    double elementValue;
    double maxValue;
    int position = 0;

    double[] shiftedVectorOne;

    productSum = new double[maxShift+1];

    for(i=0; i<maxShift+1; i++)
    {
        productSum[i] = 0;
    }

    for(i=0; i<maxShift+1; i++)
    {
        shiftedVectorOne = shiftRight(vectorOne, i);

        for(j=0; j<vectorOneDim; j++)
        {
            productSum[i] = productSum[i]
                            + shiftedVectorOne[j]*vectorTwo[j];
```

```
            }
        }

        maxValue = productSum[0];

        for(i=0; i< maxShift+1; i++)
        {
            elementValue = productSum[i];
            if(elementValue > maxValue)
            {
                maxValue = elementValue;
                position = i;
            }
        }

        maxOverlapValue = maxValue;
        maxOverlapPosition = (double)position;
    }

    private double[] shiftRight(double[] dataArray, int shiftAmount)
    {
        int arraySize;
        int i, j;

        arraySize = dataArray.length;

        double[] shiftedData = new double[arraySize];

        if(shiftAmount != 0)
        {
            j = 0;

            for(i=shiftAmount; i<arraySize; i++)
            {
                shiftedData[i] = dataArray[j];
                j = j + 1;
            }

            for(i=0; i<shiftAmount; i++)
            {
                //shiftedData[i] = dataArray[j];    // wrap
                shiftedData[i] = 0.0;               // zero
                j = j + 1;
            }

            return shiftedData;
        }
        else
        {
            return dataArray;
        }
    }

    private double[] shiftUp(double[] dataArray, double shift)
    {
        int dataSize = dataArray.length;
        int i;

        for(i=0; i<dataSize; i++)
        {
            dataArray[i] = dataArray[i] + shift;
        }

        return dataArray;
    }

    /******* SHOW ARRAY ******

    private void showArray(double[] array)
    {
        int size = array.length;
```

```
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */
}
```

```java
package fMRIWindows.viewers;

//
//  ViewerWindowArray.java
//
//
//  Created by José Gabriel Lira Gomes on 4/25/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import matematica.algebra.Matrix;
import windowUtils.GraphPanelsWindow;

public class ViewerWindowArray extends GraphPanelsWindow
{
    public ViewerWindowArray(double[] dataArray)
    {
        super(new Matrix(dataArray, false), "Perfil", 1);
    }
}
```

```
package fMRIWindows.viewers;

//
//  ViewerWindowData.java
//
//
//  Created by JGLG on 4/22/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

import imageUtils.BlockImageBuilder;
import imageUtils.ImagePanel;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Graphics;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import windowUtils.FormatWindow;
import windowUtils.RadioButtonsWindow;

import dirUtils.DirPathGetter;
import estatistica.descriptive.OneDArrayAnalyzer;
import fMRIWindows.MainWindow;
import fMRIWindows.MenuedFrame;
import fileUtils.WriteTextNumberFile;
import functional.export.JpegExporter;
import graphUtils.GraphPanel;

public class ViewerWindowData extends MenuedFrame
            implements ChangeListener, MouseListener, MouseMotionListener
{
    //
    //  Data Members
    //

    private double[][][][] imageData;
    private int zoomFactor;
    private MainWindow parentWindow;

    private double[] bckgData;
    private boolean timelineWithBackground = false;

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfInstants;

    private double[][] instantImageSliceData;

    private int xInitialValue;
    private int yInitialValue;
    private int zInitialValue;

    private int x;
```

```
    private int y;
    private int z;

    private int middleZ;

    private int arrayImgWidth;
    private int arrayImgHeight=200;

    private int windowWidth;
    private int windowHeight;
    private int windowXPosition = 0;
    private int windowYPosition = 142;

    private double[] timelineArray;

    private Container viewerContainer;
    private GraphPanel timelinePanel;
    private JPanel timelineDataPanel;
    private JPanel tSliderPanel;
    private JPanel dataExportPanel;
    private JPanel zSliderPanel;
    private ImagePanel imagePanel;
    private JPanel dataPanel;
    private JPanel imageExportPanel;

    private JLabel dataVoxel;
    private JSlider timeSlider;
    private JButton dataExportButton;
    private JSlider zSlider;
    private JLabel dataDims;
    private JLabel dataInstant;
    private JLabel dataZValue;
    private JButton jpegExportButton;

    private RadioButtonsWindow formatWindow;
    private JButton setFormatButton;

    private double amplitude;

    private int mouseX = 0;
    private int mouseY = 0;



    //
    //  Constructors
    //
    //      viewerWindowData(double[][][][] imageData,
    //                       int zoomFactor,
    //                       MainWindow parentWindow)
    //

    public ViewerWindowData(double[][][][] imageData,
                            int zoomFactor,
                            MainWindow parentWindow)
    {
        this.imageData = imageData;
        this.zoomFactor = zoomFactor;
        this.parentWindow = parentWindow;

        getImageDims();
        getInitialValues();

        createGUI();

        imagePanel.addMouseListener(this);
        imagePanel.addMouseMotionListener(this);
    }

    public ViewerWindowData(double[][][][] imageData,
                            int zoomFactor,
```

```
                        double[] bckgData,
                        MainWindow parentWindow)
{
    this.imageData = imageData;
    this.zoomFactor = zoomFactor;
    this.bckgData = bckgData;
    this.timelineWithBackground = true;
    this.parentWindow = parentWindow;

    getImageDims();
    getInitialValues();

    createGUI();

    imagePanel.addMouseListener(this);
    imagePanel.addMouseMotionListener(this);
}

/******* LISTEN TO SLIDER CHANGE EVENT *******/

public void stateChanged(ChangeEvent event)
{
    int time;

    time = timeSlider.getValue();

    this.z = zSlider.getValue();

    dataInstant.setText("t = " + Integer.toString(time));
    dataZValue.setText("Z = " + Integer.toString(z));

    instantImageSliceData = getInstantSlice(imageData, z, time);
    imagePanel.setImageData(instantImageSliceData, 0);
}

/******* LISTEN TO BUTTON ACTION EVENT *******/

public void actionPerformed(ActionEvent event)
{
    int time;
    double[][][] instantImageData;

    DirPathGetter pathGetter;
    String path;
    String outputPath;

    JButton clickedButton = (JButton) event.getSource();

    time = timeSlider.getValue();
    instantImageData = extractInstantVolume(time);

    if(clickedButton == dataExportButton)
    {
        formatWindow = new FormatWindow();

        setFormatButton = formatWindow.acceptButton;
        setFormatButton.addActionListener(this);

        parentWindow.setVisible(false);
        this.setVisible(false);
    }
    if(clickedButton == setFormatButton)
    {
        formatWindow.setVisible(false);

        pathGetter = new DirPathGetter();
        path = pathGetter.getPath();

        outputPath = path + "/Sequencia"+"("+Integer.toString(x)
                        +","+Integer.toString(y)+","
                        +Integer.toString(z)+").txt";
```

```
            new WriteTextNumberFile(timelineArray,
                                    outputPath,
                                    formatWindow.getSelection());

            this.setVisible(true);
            parentWindow.setVisible(true);
        }
        if(clickedButton == jpegExportButton)
        {
            pathGetter = new DirPathGetter();
            path = pathGetter.getPath();
            new JpegExporter(instantImageData,
                             0.0,
                             zoomFactor,
                             false,
                             path,
                             "Dados");
        }
    }

    public void paint(Graphics g)
    {
        super.paint(g);
    }


    public void mouseClicked(MouseEvent me)
    {
        mouseX = me.getX();
        mouseY = me.getY();

        if(me.isMetaDown())
        {

        }
        else
        {
            if(me.isAltDown())
            {
                if(zoomFactor>1)
                {
                    zoomFactor = zoomFactor/2;
                    imagePanel.setZoomFactor(zoomFactor);
                }
            }
            else
            {
                if(me.isControlDown())
                {
                    zoomFactor = zoomFactor*2;
                    imagePanel.setZoomFactor(zoomFactor);
                }
                else
                {
                    x = (int)Math.floor(mouseX/zoomFactor);
                    y = height - 1 - (int)Math.floor(mouseY/zoomFactor);
                    if((x<width)&&(x>=0)&&(y<height)&&(y>=0))
                    {
                        timelineArray = extractDataArray(x, y, z, imageData);
                        timelinePanel.updateSingleArray(timelineArray);

                        updateAmplitude();
                        dataVoxel.setText("V = ("+x+","+y+","+z+"); A(V) = "
                                          + amplitude);
                    }
                }
            }
        }

    }
```

```
public void mouseEntered(MouseEvent me)
{

}


public void mouseExited(MouseEvent me)
{

}


public void mousePressed(MouseEvent me)
{

}


public void mouseReleased(MouseEvent me)
{

}


public void mouseDragged(MouseEvent me)
{

}


public void mouseMoved(MouseEvent me)
{

}

private void getImageDims()
{
    this.width = imageData.length;
    this.height = imageData[0].length;
    this.numberOfSlices = imageData[0][0].length;
    this.numberOfInstants = imageData[0][0][0].length;
}

private void getInitialValues()
{
    this.xInitialValue = (int)Math.round(width/2);
    this.yInitialValue = (int)Math.round(height/2);
    this.zInitialValue = (int)Math.round(numberOfSlices/2);

    this.x = xInitialValue;
    this.y = yInitialValue;
    this.z = zInitialValue;

    middleZ = Math.round(numberOfSlices/2) -1;
}

private void createGUI()
{
    arrayImgWidth = 2*numberOfInstants;

    windowWidth = /*20 +*/ Math.max(arrayImgWidth, zoomFactor*width+70);
    windowHeight = 200 + 42 + 85 + zoomFactor*height + 100 + 20;

    createTimelinePanel();
    createTimelineDataPanel();
    createTSliderPanel();
    createExportDataPanel();
    createZSliderPanel();
    createImagePanel();
```

```
        createDataPanel();
        createExportImagePanel();

        addPanelsToContainer();

        setSize(windowWidth, windowHeight);
        setLocation(windowXPosition, windowYPosition);
        setTitle("Dados");
        setVisible(true);
    }

    private void createTimelinePanel()
    {
        timelineArray = extractDataArray(xInitialValue,
                                          yInitialValue,
                                          zInitialValue,
                                          imageData);
        timelinePanel = new GraphPanel(timelineArray,
                                       arrayImgWidth,
                                       arrayImgHeight);
        if(timelineWithBackground)
        {
            BlockImageBuilder bckgBuilder = new BlockImageBuilder(2,
                                    arrayImgHeight,
                                    parentWindow.getNumberOfStates(),
                                    (float)0.25);
            bckgBuilder.setHorizBlocks(parentWindow.getTimeline());
            timelinePanel.setBackground(bckgBuilder.getDataArray());
        }
    }

    private void createTSliderPanel()
    {
        tSliderPanel = new JPanel(new BorderLayout());
        tSliderPanel.setBorder(BorderFactory.createTitledBorder("Tempo"));
        timeSlider = createTSlider(0);
        tSliderPanel.add(timeSlider, BorderLayout.CENTER);
        if(numberOfInstants == 1)
        {
            timeSlider.setEnabled(false);
        }
    }

    private void createTimelineDataPanel()
    {
        timelineDataPanel = new JPanel(new FlowLayout());
        timelineDataPanel.setMinimumSize(new Dimension(50,42));
        updateAmplitude();
        dataVoxel = new JLabel("V = ("
                        +xInitialValue+","+yInitialValue+","+zInitialValue
                        +"); A(V) = " + amplitude);
        timelineDataPanel.add(dataVoxel);
    }

    private void createExportDataPanel()
    {
        dataExportPanel = new JPanel(new GridLayout(1,1));
        dataExportPanel.setPreferredSize(new Dimension(79,42));
        dataExportButton = new JButton("Dados");
        dataExportButton.addActionListener(this);
        dataExportPanel.add(dataExportButton);
    }

    private void createImagePanel()
    {
        instantImageSliceData = getInstantSlice(imageData, middleZ, 0);
        imagePanel = new ImagePanel(instantImageSliceData, zoomFactor, false);
    }

    private void createZSliderPanel()
    {
```

```
        zSliderPanel = new JPanel(new BorderLayout());
        zSliderPanel.setBorder(BorderFactory.createTitledBorder("Z"));
        zSlider = createZSlider(middleZ);
        zSliderPanel.add(zSlider, BorderLayout.CENTER);
    }

    private void createDataPanel()
    {
        dataPanel = new JPanel(new GridLayout(3,1));
        dataPanel.setMinimumSize(new Dimension(100,100));
        dataPanel.setBorder(BorderFactory.createTitledBorder("Dados"));
        dataDims = new JLabel(Integer.toString(width) +"x"
                            + Integer.toString(height) +"x"
                            + Integer.toString(numberOfSlices)
                            +"x"+ Integer.toString(numberOfInstants));
        dataInstant = new JLabel("t = 0");
        dataZValue = new JLabel("Z = " + middleZ);
        dataPanel.add(dataDims);
        dataPanel.add(dataInstant);
        dataPanel.add(dataZValue);
    }

    private void createExportImagePanel()
    {
        imageExportPanel = new JPanel(new GridLayout(3,1));
        imageExportPanel.setPreferredSize(new Dimension(79,100));
        jpegExportButton = new JButton("JPEG");
        jpegExportButton.addActionListener(this);
        imageExportPanel.add(jpegExportButton);
    }

    private void addPanelsToContainer()
    {
        viewerContainer = getContentPane();
        viewerContainer.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();

        c.fill = GridBagConstraints.BOTH;

        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 3;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewerContainer.add(timelinePanel, c);

        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 2;
        c.gridheight = 1;
        c.weightx = 1;
        c.weighty = 0;

        viewerContainer.add(timelineDataPanel, c);

        c.gridx = 2;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewerContainer.add(dataExportPanel, c);

        c.gridx = 0;
        c.gridy = 2;
        c.gridwidth = 3;
        c.gridheight = 1;
        c.weightx = 0;
```

```
        c.weighty = 0;

        viewerContainer.add(tSliderPanel, c);

        c.gridx = 0;
        c.gridy = 3;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewerContainer.add(zSliderPanel, c);

        c.gridx = 1;
        c.gridy = 3;
        c.gridwidth = 2;
        c.gridheight = 1;
        c.weightx = width*zoomFactor;
        c.weighty = height*zoomFactor;

        viewerContainer.add(imagePanel, c);

        c.gridx = 0;
        c.gridy = 4;
        c.gridwidth = 2;
        c.gridheight = 1;
        c.weightx = 1;
        c.weighty = 0;

        viewerContainer.add(dataPanel, c);

        c.gridx = 2;
        c.gridy = 4;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewerContainer.add(imageExportPanel, c);
    }

    /******* CREATE T SLIDER *******/

    private JSlider createTSlider(int initialValue)
    {
        JSlider tSlider = new JSlider();

        tSlider.setOrientation(JSlider.HORIZONTAL);
        tSlider.setPaintLabels(true);
        tSlider.setPaintTicks(true);
        tSlider.setMinimum(0);
        tSlider.setMaximum(numberOfInstants-1);
        tSlider.setValue(initialValue);
        tSlider.setMajorTickSpacing(50);
        tSlider.setMinorTickSpacing(25);

        tSlider.addChangeListener(this);

        return tSlider;
    }

    /******* CREATE Z SLIDER *******/

    private JSlider createZSlider(int initialValue)
    {
        JSlider zSlider = new JSlider();

        zSlider.setOrientation(JSlider.VERTICAL);
        zSlider.setPaintLabels(true);
        zSlider.setPaintTicks(true);
        zSlider.setMinimum(0);
```

```
        zSlider.setMaximum(numberOfSlices-1);
        zSlider.setValue(initialValue);
        zSlider.setMajorTickSpacing(10);
        zSlider.setMinorTickSpacing(5);

        zSlider.addChangeListener(this);

        return zSlider;
    }

    private void updateAmplitude()
    {
        OneDArrayAnalyzer timelineArrayAnalyzer;
        double maxArrayValue;
        double minArrayValue;

        timelineArrayAnalyzer = new OneDArrayAnalyzer(timelineArray);
        maxArrayValue = timelineArrayAnalyzer.getMaximum();
        minArrayValue = timelineArrayAnalyzer.getMinimum();
        amplitude = Math.round((maxArrayValue-minArrayValue)*100)/100;
    }

    private double[] extractDataArray(int x,
                                      int y,
                                      int z,
                                      double[][][][] dataMatrix)
    {
        int i;
        double[] dataArray = new double[numberOfInstants];

        for(i=0; i< numberOfInstants; i++)
        {
            dataArray[i] = dataMatrix[x][y][z][i];
        }
        return dataArray;
    }

    private double[][][] extractInstantVolume(int t)
    {
        double[][][] volume = new double[width][height][numberOfSlices];
        int x, y, z;

        for(x=0; x<width; x++)
        {
            for(y=0; y<height; y++)
            {
                for(z=0; z<numberOfSlices; z++)
                {
                    volume[x][y][z] = imageData[x][y][z][t];
                }
            }
        }

        return volume;
    }

    private double[][] getInstantSlice(double[][][][] volumeDataArray,
                                       int z,
                                       int t)
    {
        int x, y;
        double[][] sliceDataArray = new double[width][height];

        for(y=0; y<height; y++)
        {
            for(x=0; x<width; x++)
            {
                sliceDataArray[x][y] = volumeDataArray[x][y][z][t];
            }
        }
```

```
        return sliceDataArray;
    }

    /******* SHOW ARRAY ******

    private void showArray(double[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */
}
```

```
package fMRIWindows.viewers;

//
//  ViewerWindowDataArrays.java
//
//
//  Created by José Gabriel Lira Gomes on 9/12/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.Container;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Graphics;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;

import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import windowUtils.FormatWindow;
import windowUtils.RadioButtonsWindow;
import windowUtils.TripleSliderPanel;
import dirUtils.DirPathGetter;
import estatistica.descriptive.OneDArrayAnalyzer;
import fMRIWindows.MainWindow;
import fMRIWindows.MenuedFrame;
import fileUtils.WriteTextNumberFile;
import graphUtils.GraphPanel;

public class ViewerWindowDataArrays extends MenuedFrame
                                        implements ChangeListener
{
    private double[][][][] imageData;
    private String titleText;
    private MainWindow parentWindow;

    private Container viewerContainer;

    private GraphPanel arrayGraphPanel;
    private TripleSliderPanel arraySlidersPanel;
    private JPanel arrayExportPanel;
    private JPanel dataPanel;

    private boolean withBackground = false;

    private RadioButtonsWindow formatWindow;
    private JButton setFormatButton;

    private JButton dataExportButton;

    private JSlider xSlider;
    private JSlider ySlider;
    private JSlider zSlider;

    private JLabel dataLabel;

    private int width;
    private int height;
    private int numberOfSlices;
    private int numberOfPoints;

    private int xInitialValue;
    private int yInitialValue;
    private int zInitialValue;
```

```
private int x;
private int y;
private int z;

private double[] array;
private OneDArrayAnalyzer arrayAnalyzer;
private double maxArrayValue;
private double minArrayValue;
private double amplitude;

private int arrayImgWidth;
private int arrayImgHeight;
private int slidersPanelWidth;

private int windowWidth;
private int windowHeight;
private int windowXPosition;
private int windowYPosition = 21;

public ViewerWindowDataArrays(double[][][][] imageData,
                              String titleText,
                              MainWindow parentWindow)
{
    this.imageData = imageData;
    this.titleText = titleText;
    this.parentWindow = parentWindow;

    getDims();

    getInitialValues();

    array = extractDataArray(xInitialValue,
                              yInitialValue,
                              zInitialValue,
                              imageData);

    createGUI();
}

public ViewerWindowDataArrays(double[][][][] imageData,
                              double[][] backgroundData,
                              String titleText,
                              MainWindow parentWindow)
{
    this.imageData = imageData;
    this.titleText = titleText;
    this.parentWindow = parentWindow;

    getDims();

    getInitialValues();

    array = extractDataArray(xInitialValue,
                              yInitialValue,
                              zInitialValue,
                              imageData);

    createGUI();

    setBackground(backgroundData);
}

public void setBackground(double[][] backgroundData)
{
    arrayGraphPanel.setBackground(backgroundData);
    withBackground = true;
}

public void removeBackground()
{
    if(withBackground)
```

```
        {
            arrayGraphPanel.removeBackground();
            withBackground = false;
        }
    }

    public void setPosition(int xPos, int yPos, int zPos)
    {
        x = xPos;
        y = yPos;
        z = zPos;

        array = extractDataArray(x, y, z, imageData);
    }

    public void actionPerformed(ActionEvent event)
    {
        DirPathGetter pathGetter;
        String path;
        String outputPath;

        JButton clickedButton = (JButton) event.getSource();

        if(clickedButton == dataExportButton)
        {
            formatWindow = new FormatWindow();

            setFormatButton = formatWindow.acceptButton;
            setFormatButton.addActionListener(this);

            parentWindow.setVisible(false);
            this.setVisible(false);
        }
        if(clickedButton == setFormatButton)
        {
            formatWindow.setVisible(false);

            pathGetter = new DirPathGetter();
            path = pathGetter.getPath();

            outputPath = path + "/"+titleText+"("+Integer.toString(x)
                            +","+Integer.toString(y)
                            +","+Integer.toString(z)+").txt";

            new WriteTextNumberFile(array,
                                    outputPath,
                                    formatWindow.getSelection());

            this.setVisible(true);
            parentWindow.setVisible(true);
        }
    }

    public void stateChanged(ChangeEvent event)
    {
        int xPos, yPos, zPos;

        xPos = xSlider.getValue();
        yPos = ySlider.getValue();
        zPos = zSlider.getValue();

        setPosition(xPos, yPos, zPos);

        arrayGraphPanel.updateSingleArray(array);

        arrayAnalyzer = new OneDArrayAnalyzer(array);
        maxArrayValue = arrayAnalyzer.getMaximum();
        minArrayValue = arrayAnalyzer.getMinimum();
        amplitude = Math.round((maxArrayValue-minArrayValue)*100)/100;

        dataLabel.setText("V = (" + xPos + ", " + yPos + ", " + zPos
```

```
                                        + "); A(V) = " + amplitude);
    }

    public void paint(Graphics g)
    {
        super.paint(g);
    }

    private void getDims()
    {
        this.width = imageData.length;
        this.height = imageData[0].length;
        this.numberOfSlices = imageData[0][0].length;
        this.numberOfPoints = imageData[0][0][0].length;
    }

    private void getInitialValues()
    {
        this.xInitialValue = (int)Math.round(width/2);
        this.yInitialValue = (int)Math.round(height/2);
        this.zInitialValue = (int)Math.round(numberOfSlices/2);

        this.x = xInitialValue;
        this.y = yInitialValue;
        this.z = zInitialValue;
    }

    private void createGUI()
    {
        arrayImgWidth = 2*numberOfPoints;
        arrayImgHeight = 200;

        slidersPanelWidth = 180;

        windowWidth = arrayImgWidth + slidersPanelWidth;
        windowHeight = arrayImgHeight + 100;
        windowXPosition = parentWindow.getWidth() + 1;

        createArrayGraphPanel();
        createArraySlidersPanel();
        createDataPanel();
        createArrayExportPanel();

        viewerContainer = getContentPane();

        viewerContainer.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();

        c.fill = GridBagConstraints.BOTH;

        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = arrayImgWidth;
        c.weighty = arrayImgHeight;

        viewerContainer.add(arrayGraphPanel, c);

        c.gridx = 1;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewerContainer.add(arraySlidersPanel, c);

        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 1;
```

```
        c.gridheight = 1;
        c.weightx = arrayImgWidth;
        c.weighty = 0;

        viewerContainer.add(dataPanel, c);

        c.gridx = 1;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewerContainer.add(arrayExportPanel, c);

        setSize(windowWidth, windowHeight);
        setLocation(windowXPosition, windowYPosition);
        setTitle(titleText);
        setVisible(true);
    }

    private void createArrayGraphPanel()
    {
        arrayGraphPanel = new GraphPanel(array,
                                         arrayImgWidth,
                                         arrayImgHeight);
    }
    private void createArraySlidersPanel()
    {
        arraySlidersPanel = new TripleSliderPanel(width,
                                                  height,
                                                  numberOfSlices);

        xSlider = arraySlidersPanel.xSlider;
        xSlider.addChangeListener(this);

        ySlider = arraySlidersPanel.ySlider;
        ySlider.addChangeListener(this);

        zSlider = arraySlidersPanel.zSlider;
        zSlider.addChangeListener(this);
    }
    private void createDataPanel()
    {
        dataPanel = new JPanel(new FlowLayout());
        dataPanel.setMinimumSize(new Dimension(50,42));

        arrayAnalyzer = new OneDArrayAnalyzer(array);
        maxArrayValue = arrayAnalyzer.getMaximum();
        minArrayValue = arrayAnalyzer.getMinimum();
        amplitude = Math.round((maxArrayValue-minArrayValue)*100)/100;
        dataLabel = new JLabel("V = ("
                               + xInitialValue + ", "
                               + yInitialValue + ", "
                               + zInitialValue + "); A(V) = "
                               + amplitude);

        dataPanel.add(dataLabel);
    }
    private void createArrayExportPanel()
    {
        arrayExportPanel = new JPanel();
        arrayExportPanel.setPreferredSize(new Dimension(79,42));

        dataExportButton = new JButton("Dados");
        dataExportButton.setEnabled(true);
        dataExportButton.addActionListener(this);

        arrayExportPanel.add(dataExportButton);
    }
```

```
    private double[] extractDataArray(int x,
                                      int y,
                                      int z,
                                      double[][][][] dataMatrix)
    {
        int i;
        double[] dataArray = new double[numberOfPoints];

        for(i=0; i< numberOfPoints; i++)
        {
            dataArray[i] = dataMatrix[x][y][z][i];
        }
        return dataArray;
    }

}
```

```java
package fMRIWindows.viewers;

//
//  ViewerWindowDataMask.java
//
//
//  Created by Jose Gabriel Lira Gomes on 4/10/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import imageUtils.ImagePanel;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.io.IOException;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.JToggleButton;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import windowUtils.FormatWindow;
import windowUtils.RadioButtonsWindow;
import dirUtils.DirPathGetter;
import estatistica.descriptive.ThreeDArrayAnalyzer;
import fMRIWindows.MainWindow;
import fMRIWindows.MenuedFrame;
import fileUtils.WriteTextNumberFile;
import functional.data.AnlzHeader;
import functional.data.AnlzWriter;
import functional.data.Mask;
import functional.export.JpegExporter;
import graphUtils.GraphPanel;

public class ViewerWindowDataMask extends MenuedFrame
    implements ChangeListener, MouseListener, MouseMotionListener, ItemListener
{
    //
    //  Data Members
    //

    private double[][][] imageData;
    private int zoomFactor;
    private String method;
    MainWindow parentWindow;

    private int width;
    private int height;
    private int numberOfSlices;
    private short[] imageDims;

    private int xInitialValue;
    private int yInitialValue;
    private int zInitialValue;

    private int x;
```

```
    private int y;
    private int z;

    private int middleZ;
    private double middleThres;

    private int xArrayImgWidth;
    private int yArrayImgWidth;
    private int zArrayImgWidth;

    private int profilesPanelWidth;
    private int profilePanelsHeight;

    private int dataPanelHeight;

    private int windowWidth;
    private int windowHeight;
    private int windowXPosition;
    private int windowYPosition = 21;

    double[] xProfileArray;
    double[] yProfileArray;
    double[] zProfileArray;

    private Container viewContainer;
    private GraphPanel xProfilePanel;
    private GraphPanel yProfilePanel;
    private GraphPanel zProfilePanel;
    private JPanel profilesPanel;
    private ImagePanel imagePanel;
    private JPanel toolsPanel;
    private JPanel dataPanel;
    private JPanel zSliderPanel;
    private JPanel thresSliderPanel;
    private JPanel exportPanel;

    private boolean bckgVisible = false;
    private double[][][] bckgData;
    private double[][] bckgSliceData;
    private double[][] imageSliceData;

    private JButton xDataButton;
    private JButton yDataButton;
    private JButton zDataButton;
    private JSlider zSlider;
    private JSlider thresSlider;
    private JToggleButton bckgButton;
    private JLabel dataThres;
    private JLabel dataZValue;
    private JButton jpegExportButton;
    private JButton analyzeExportButton;

    private boolean xAxis = false;
    private boolean yAxis = false;
    private boolean zAxis = false;

    private RadioButtonsWindow formatWindow;
    private JButton setFormatButton;

    private AnlzHeader resultDataHeader;

    private long startTime;
    private long stopTime;
    private long elapsedTime;

    private int mouseX = 0;
    private int mouseY = 0;

    //
    //  Constructors
    //
```

```
//      ViewerWindowDataMask(Mask dataMask,
//                           int zoomFactor,
//                           String method,
//                           MainWindow parentWindow)
//

public ViewerWindowDataMask(Mask dataMask,
                            int zoomFactor,
                            String method,
                            MainWindow parentWindow)
{
    this.imageData = dataMask.getDataNumberArray();
    this.zoomFactor = zoomFactor;
    this.method = method;
    this.parentWindow = parentWindow;

    getImageDims();
    getInitialValues();

    createGUI();

    imagePanel.addMouseListener(this);
    imagePanel.addMouseMotionListener(this);
}


public ViewerWindowDataMask(Mask dataMask,
                            double[][][] bckgData,
                            int zoomFactor,
                            String method,
                            MainWindow parentWindow)
{
    this.imageData = dataMask.getDataNumberArray();
    this.bckgData = bckgData;
    this.zoomFactor = zoomFactor;
    this.method = method;
    this.parentWindow = parentWindow;

    getImageDims();
    getInitialValues();

    bckgVisible = true;
    createGUI();

    imagePanel.addMouseListener(this);
    imagePanel.addMouseMotionListener(this);
}

/******* LISTEN TO SLIDER CHANGE EVENT *******/

public void stateChanged(ChangeEvent event)
{
    int intThres;
    double threshold;
    String pValueText;
    String zPositionText;

    intThres = thresSlider.getValue();
    threshold = (double)intThres/100.0;
    pValueText = Double.toString(Math.round((1.0 - threshold)*100.0)/100.0);

    this.z = zSlider.getValue();
    zPositionText = Integer.toString(z);

    dataThres.setText("p = " + pValueText);
    dataZValue.setText("Z = " + zPositionText);

    if(bckgVisible)
    {
        this.bckgSliceData = getSlice(bckgData, z);
        this.imagePanel.setBckgImageData(bckgSliceData);
```

```
            this.imageSliceData = getSlice(imageData, z);
            this.imagePanel.setImageDataAndThreshold(imageSliceData,
                                                     threshold,
                                                     1);
        }
        else
        {
            this.imageSliceData = getSlice(imageData, z);
            this.imagePanel.setImageDataAndThreshold(imageSliceData,
                                                     threshold,
                                                     0);
        }

    }


    public void itemStateChanged(ItemEvent event)
    {
        JToggleButton source = (JToggleButton) event.getSource();

        if(source == bckgButton)
        {
            if(event.getStateChange() == ItemEvent.SELECTED)
            {
                this.bckgVisible = true;

                this.bckgSliceData = getSlice(bckgData, z);
                this.imagePanel.setBckgImageData(bckgSliceData);
                this.imagePanel.showBackground();
            }
            else
            {
                this.bckgVisible = false;

                this.imagePanel.hideBackground();
            }
        }
    }


    /******* LISTEN TO BUTTON ACTION EVENT *******/

    public void actionPerformed(ActionEvent event)
    {
        int intThres;
        double threshold;
        DirPathGetter dirPathGetter;
        String dirPath;
        String xPath;
        String yPath;
        String zPath;

        JButton clickedButton = (JButton) event.getSource();

        intThres = thresSlider.getValue();
        threshold = (double)intThres/100;

        if(clickedButton == xDataButton)
        {
            xAxis = true;
            formatWindow = new FormatWindow();

            setFormatButton = formatWindow.acceptButton;
            setFormatButton.addActionListener(this);

            parentWindow.setVisible(false);
            this.setVisible(false);
        }
        if(clickedButton == yDataButton)
        {
            yAxis = true;
```

```
            formatWindow = new FormatWindow();

            setFormatButton = formatWindow.acceptButton;
            setFormatButton.addActionListener(this);

            parentWindow.setVisible(false);
            this.setVisible(false);
        }
        if(clickedButton == zDataButton)
        {
            zAxis = true;
            formatWindow = new FormatWindow();

            setFormatButton = formatWindow.acceptButton;
            setFormatButton.addActionListener(this);

            parentWindow.setVisible(false);
            this.setVisible(false);
        }
        if(clickedButton == setFormatButton)
        {
            formatWindow.setVisible(false);

            dirPathGetter = new DirPathGetter();
            dirPath = dirPathGetter.getPath();

            if(xAxis)
            {
                xPath = dirPath + "/xPerfil("+Integer.toString(x)+","
                                              +Integer.toString(y)+","
                                              +Integer.toString(z)+").txt";
                new WriteTextNumberFile(xProfileArray,
                                        xPath,
                                        formatWindow.getSelection());
                xAxis = false;
            }
            if(yAxis)
            {
                yPath = dirPath + "/yPerfil("+Integer.toString(x)
                                +","+Integer.toString(y)
                                +","+Integer.toString(z)+").txt";
                new WriteTextNumberFile(yProfileArray,
                                            yPath,
                                            formatWindow.getSelection());
                yAxis = false;
            }
            if(zAxis)
            {
                zPath = dirPath + "/zPerfil("+Integer.toString(x)+","
                                              +Integer.toString(y)+","
                                              +Integer.toString(z)+").txt";
                new WriteTextNumberFile(zProfileArray,
                                        zPath,
                                        formatWindow.getSelection());
                zAxis = false;
            }

            this.setVisible(true);
            parentWindow.setVisible(true);
        }
        if(clickedButton == jpegExportButton)
        {
            dirPathGetter = new DirPathGetter();
            dirPath = dirPathGetter.getPath();
            if(bckgVisible)
            {
                new JpegExporter(imageData,
                                 bckgData,
                                 threshold,
                                 4,
                                 true,
```

```
                                dirPath,
                                method);
            }
            else
            {
                new JpegExporter(imageData,
                                threshold,
                                4,
                                true,
                                dirPath,
                                method);
            }
        }
        if(clickedButton == analyzeExportButton)
        {
            dirPathGetter = new DirPathGetter();
            dirPath = dirPathGetter.getPath();

            resultDataHeader = new AnlzHeader();

            resultDataHeader.setDataType("bin");
            resultDataHeader.setNumberOfDims((short)4);
            resultDataHeader.setDimensions(imageDims);
            resultDataHeader.setDataTypeCode((short)1);
            resultDataHeader.setBitsPerPixel((short)1);
            resultDataHeader.setPixelsDims(parentWindow.getPixelsDims());
            resultDataHeader.setFUnusedOne(parentWindow.getFUnusedOne());
            resultDataHeader.setGLMax(1);
            resultDataHeader.setGLMin(0);
            resultDataHeader.setDescrip("Mascara " + method);

            System.out.println("Inicio da escrita dos resultados");
            startTime = System.currentTimeMillis();
            try
            {
                new AnlzWriter(resultDataHeader,
                               imageData,
                               dirPath + "/" + method);
            }
            catch (IOException e)
            {
                System.out.println("failed analyze file writing");
                e.printStackTrace();
            }
            stopTime = System.currentTimeMillis();
            System.out.println("Fim da escrita dos resultados");
            elapsedTime = stopTime - startTime;
            System.out.println("Tempo da escrita dos resultados:");
            System.out.println(elapsedTime + " ms");
            System.out.println("");
        }
    }

    public void paint(Graphics g)
    {
        super.paint(g);
    }


    public void mouseClicked(MouseEvent me)
    {
        mouseX = me.getX();
        mouseY = me.getY();

        if(me.isMetaDown())
        {

        }
        else
        {
            if(me.isAltDown())
```

```
        {
            if(zoomFactor>1)
            {
                zoomFactor = zoomFactor/2;
                imagePanel.setZoomFactor(zoomFactor);
            }
        }
        else
        {
            if(me.isControlDown())
            {
                zoomFactor = zoomFactor*2;
                imagePanel.setZoomFactor(zoomFactor);
            }
            else
            {
                x = (int)Math.floor(mouseX/zoomFactor);
                y = height - 1 - (int)Math.floor(mouseY/zoomFactor);
                if((x<width)&&(x>=0)&&(y<height)&&(y>=0))
                {
                    getProfiles();

                    setAll();
                }
            }
        }
    }
}


public void mouseEntered(MouseEvent me)
{

}


public void mouseExited(MouseEvent me)
{

}


public void mousePressed(MouseEvent me)
{

}


public void mouseReleased(MouseEvent me)
{

}


public void mouseDragged(MouseEvent me)
{

}


public void mouseMoved(MouseEvent me)
{

}

private void getImageDims()
{
    this.width = imageData.length;
    this.height = imageData[0].length;
    this.numberOfSlices = imageData[0][0].length;
```

```
        imageDims = new short[8];
        imageDims[0] = 4;
        imageDims[1] = (short)width;
        imageDims[2] = (short)height;
        imageDims[3] = (short)numberOfSlices;
        imageDims[4] = 1;
    }


    private void getInitialValues()
    {
        this.xInitialValue = (int)Math.round(width/2);
        this.yInitialValue = (int)Math.round(height/2);
        this.zInitialValue = (int)Math.round(numberOfSlices/2);

        this.x = xInitialValue;
        this.y = yInitialValue;
        this.z = zInitialValue;

        middleZ = Math.round(numberOfSlices/2) -1;
        middleThres = 0.50;
    }


    private void createGUI()
    {
        xArrayImgWidth = 2*width;
        yArrayImgWidth = 2*height;
        zArrayImgWidth = 2*numberOfSlices;
        profilesPanelWidth = xArrayImgWidth+yArrayImgWidth+zArrayImgWidth;
        profilePanelsHeight = 200;
        dataPanelHeight = 42;

        windowXPosition = parentWindow.getWidth() + 1;
        windowWidth = 26 + Math.max(profilesPanelWidth, zoomFactor*width+79);
        windowHeight = 200 + dataPanelHeight + zoomFactor*height + 85 + 80 + 20;

        getProfiles();

        createProfilesPanel();
        createZSliderPanel();
        createImagePanel();
        createThresSliderPanel();
        createToolsPanel();
        createDataPanel();
        createExportPanel();

        addPanelsToContainer();

        setSize(windowWidth, windowHeight);
        setLocation(windowXPosition, windowYPosition);
        setTitle(method);
        setVisible(true);
    }


    private void createProfilesPanel()
    {
        createProfilePanels();
        createProfileDataLabels();

        profilesPanel = new JPanel(new GridBagLayout());
        profilesPanel.setPreferredSize(new Dimension(profilesPanelWidth,
                                                     profilePanelsHeight+42));

        GridBagConstraints c = new GridBagConstraints();


        c.fill = GridBagConstraints.BOTH;

        c.gridx = 0;
```

```
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        profilesPanel.add(xProfilePanel, c);

        c.gridx = 1;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        profilesPanel.add(yProfilePanel, c);

        c.gridx = 2;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        profilesPanel.add(zProfilePanel, c);

        c.fill = GridBagConstraints.CENTER;

        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        profilesPanel.add(xDataButton, c);

        c.gridx = 1;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        profilesPanel.add(yDataButton, c);

        c.gridx = 2;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        profilesPanel.add(zDataButton, c);
    }

    private void createProfilePanels()
    {
        xProfilePanel = new GraphPanel(xProfileArray,
                                       xArrayImgWidth,
                                       profilePanelsHeight);
        yProfilePanel = new GraphPanel(yProfileArray,
                                       yArrayImgWidth,
                                       profilePanelsHeight);
        zProfilePanel = new GraphPanel(zProfileArray,
                                       zArrayImgWidth,
                                       profilePanelsHeight);
    }

    private void createProfileDataLabels()
    {
```

```
        xDataButton = new JButton();
        yDataButton = new JButton();
        zDataButton = new JButton();

        xDataButton.addActionListener(this);
        yDataButton.addActionListener(this);
        zDataButton.addActionListener(this);

        setProfilesDataLabels();
    }


    private void createZSliderPanel()
    {
        zSliderPanel = new JPanel(new BorderLayout());
        zSliderPanel.setBorder(BorderFactory.createTitledBorder("Z"));
        zSlider = createZSlider(middleZ);
        zSliderPanel.add(zSlider, BorderLayout.CENTER);
    }


    private void createImagePanel()
    {
        //ThreeDArrayAnalyzer imageDataAnalyzer =
        //          new ThreeDArrayAnalyzer(imageData);
        //double imageMax = imageDataAnalyzer.getMaximum();
        //double imageMin = imageDataAnalyzer.getMinimum();
        double imageMax = 1.0;
        double imageMin = 0.0;

        double bckgMax;
        double bckgMin;

        imageSliceData = getSlice(imageData, middleZ);

        if(bckgVisible)
        {
            ThreeDArrayAnalyzer bckgDataAnalyzer =
                        new ThreeDArrayAnalyzer(bckgData);
            bckgMax = bckgDataAnalyzer.getMaximum();
            bckgMin = bckgDataAnalyzer.getMinimum();

            bckgSliceData = getSlice(bckgData, middleZ);
            imagePanel = new ImagePanel(bckgSliceData,
                                        imageSliceData,
                                        zoomFactor,
                                        middleThres,
                                        bckgMax,
                                        bckgMin,
                                        imageMax,
                                        imageMin,
                                        false,
                                        true);
        }
        else
        {
            imagePanel = new ImagePanel(imageSliceData,
                                        zoomFactor,
                                        middleThres,
                                        imageMax,
                                        imageMin,
                                        true);
        }
    }


    private void createThresSliderPanel()
    {
        thresSliderPanel = new JPanel(new BorderLayout());
        thresSliderPanel.setBorder(
                    BorderFactory.createTitledBorder("Threshold"));
```

```
        thresSlider = createThresSlider((int)(middleThres*100));
        thresSliderPanel.add(thresSlider, BorderLayout.CENTER);
    }

    private void createToolsPanel()
    {
        toolsPanel = new JPanel(new GridLayout(2,1));
        toolsPanel.setPreferredSize(new Dimension(79,79));
        bckgButton = new JToggleButton("Fundo");
        bckgButton.setSelected(true);
        bckgButton.addItemListener(this);
        toolsPanel.add(bckgButton);
    }

    private void createDataPanel()
    {
        dataPanel = new JPanel(new GridLayout(2,1));
        dataPanel.setMinimumSize(new Dimension(100,70));
        //dataPanel.setPreferredSize(new Dimension(100,80));
        dataPanel.setBorder(BorderFactory.createTitledBorder("Dados"));
        dataThres = new JLabel("p = 0.50");
        dataZValue = new JLabel("Z = " + middleZ);
        dataPanel.add(dataThres);
        dataPanel.add(dataZValue);
    }


    private void createExportPanel()
    {
        exportPanel = new JPanel(new GridLayout(2,1));
        exportPanel.setPreferredSize(new Dimension(79,79));
        jpegExportButton = new JButton("JPEG");
        analyzeExportButton = new JButton("ANLZ");
        exportPanel.add(jpegExportButton);
        exportPanel.add(analyzeExportButton);
        jpegExportButton.addActionListener(this);
        analyzeExportButton.addActionListener(this);
    }

    private void addPanelsToContainer()
    {
        viewContainer = getContentPane();
        viewContainer.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();

        c.fill = GridBagConstraints.BOTH;

        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 3;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewContainer.add(profilesPanel, c);

        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewContainer.add(zSliderPanel, c);

        c.gridx = 1;
        c.gridy = 1;
        c.gridwidth = 2;
        c.gridheight = 1;
        c.weightx = width*zoomFactor;
        c.weighty = height*zoomFactor;
```

```
        viewContainer.add(imagePanel, c);

        c.gridx = 0;
        c.gridy = 2;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewContainer.add(toolsPanel, c);

        c.gridx = 1;
        c.gridy = 2;
        c.gridwidth = 2;
        c.gridheight = 1;
        c.weightx = 1;
        c.weighty = 0;

        viewContainer.add(thresSliderPanel, c);

        c.gridx = 0;
        c.gridy = 3;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewContainer.add(exportPanel, c);

        c.gridx = 1;
        c.gridy = 3;
        c.gridwidth = 2;
        c.gridheight = 1;
        c.weightx = 1;
        c.weighty = 0;

        viewContainer.add(dataPanel, c);
    }

    private void setProfilesDataLabels()
    {
        xDataButton.setText("V(x," + y + "," + z + ")");
        yDataButton.setText("V(" + x + ",y," + z + ")");
        zDataButton.setText("V(" + x + "," + y + ",z)");
    }


    private JSlider createThresSlider(int initialValue)
    {
        JSlider thresSlider = new JSlider();

        thresSlider.setOrientation(JSlider.HORIZONTAL);
        thresSlider.setPaintLabels(true);
        thresSlider.setPaintTicks(true);
        thresSlider.setMinimum(0);
        thresSlider.setMaximum(100);
        thresSlider.setValue(initialValue);
        thresSlider.setMajorTickSpacing(25);
        thresSlider.setMinorTickSpacing(5);

        thresSlider.addChangeListener(this);

        return thresSlider;
    }


    private JSlider createZSlider(int initialValue)
    {
        JSlider zSlider = new JSlider();
```

```
        zSlider.setOrientation(JSlider.VERTICAL);
        zSlider.setPaintLabels(true);
        zSlider.setPaintTicks(true);
        zSlider.setMinimum(0);
        zSlider.setMaximum(numberOfSlices-1);
        zSlider.setValue(initialValue);
        zSlider.setMajorTickSpacing(10);
        zSlider.setMinorTickSpacing(5);

        zSlider.addChangeListener(this);

        return zSlider;
    }

    private void getProfiles()
    {
        xProfileArray = getXProfileArray(x, y, z);
        yProfileArray = getYProfileArray(x, y, z);
        zProfileArray = getZProfileArray(x, y, z);
    }

    /******* GET VOXEL PROFILE ALONG X *******/

    private double[] getXProfileArray(int x, int y, int z)
    {
        double[] xProfile = new double[width];
        int i;

        for(i=0; i<width; i++)
        {
            xProfile[i] = imageData[i][y][z];
        }

        return xProfile;
    }


    /******* GET VOXEL PROFILE ALONG Y *******/

    private double[] getYProfileArray(int x, int y, int z)
    {
        double[] yProfile = new double[height];
        int i;

        for(i=0; i<height; i++)
        {
            yProfile[i] = imageData[x][i][z];
        }

        return yProfile;
    }

    /******* GET VOXEL PROFILE ALONG Z *******/

    private double[] getZProfileArray(int x, int y, int z)
    {
        double[] zProfile = new double[numberOfSlices];
        int i;

        for(i=0; i<numberOfSlices; i++)
        {
            zProfile[i] = imageData[x][y][i];
        }

        return zProfile;
    }

    private double[][] getSlice(double[][][] volumeDataArray, int z)
    {
        int x, y;
        double[][] sliceDataArray = new double[width][height];
```

```
            for(y=0; y<height; y++)
            {
                for(x=0; x<width; x++)
                {
                    sliceDataArray[x][y] = volumeDataArray[x][y][z];
                }
            }

            return sliceDataArray;
        }

        private void setAll()
        {
            setProfilesImages();

            setProfilesDataLabels();
        }

        private void setProfilesImages()
        {
            xProfilePanel.updateSingleArray(xProfileArray);
            yProfilePanel.updateSingleArray(yProfileArray);
            zProfilePanel.updateSingleArray(zProfileArray);
        }
}
```

```
package fMRIWindows.viewers;

//
//  ViewerWindowDesignMatrix.java
//
//
//  Created by Jose Gabriel Lira Gomes on 6/26/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import imageUtils.BlockImagePanel;

import java.awt.Container;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;

import javax.swing.JButton;
import javax.swing.JPanel;

import fMRIWindows.MenuedFrame;
import functional.data.DesignMatrix;

public class ViewerWindowDesignMatrix extends MenuedFrame
{
    //
    //  Data Members
    //

    private double[][] designMatrixArray;

    private JButton closeWindowButton;

    private Container viewContainer;
    private BlockImagePanel imagePanel;
    private JPanel closePanel;

    private int numberOfStates;
    private int numberOfInstants;
    private int rectangleHeight = 2;
    private int rectangleWidth = 25;

    private int minWindowWidth = 180;
    private int minWindowHeight = 150;
    private int windowXPosition = 0;
    private int windowYPosition = 142;

    private int windowWidth;
    private int windowHeight;

    private int imagePanelWidth;
    private int imagePanelHeight;
    private int closePanelHeight = 25;


    //
    //  Constructors
    //
    //      viewerWindowDesignMatrix(int[][] localDesignMatrix)
    //

    public ViewerWindowDesignMatrix(DesignMatrix designMatrix)
    {
        this.designMatrixArray = designMatrix.getDesignMatrixArray();

        createGUI();
    }
```

```
/******* LISTEN TO BUTTON ACTION EVENT *******/

public void actionPerformed(ActionEvent event)
{

    JButton clickedButton = (JButton) event.getSource();

    if(clickedButton == closeWindowButton)
    {
        this.setVisible(false);
    }
}


private void createGUI()
{
    numberOfInstants = designMatrixArray.length;
    numberOfStates = designMatrixArray[0].length;

    imagePanelWidth = rectangleWidth*numberOfStates + 40;
    imagePanelHeight = rectangleHeight*numberOfInstants + 40;

    if(imagePanelWidth<=minWindowWidth)
    {
        windowWidth = minWindowWidth;
    }
    else
    {
        windowWidth = imagePanelWidth;
    }

    if(imagePanelHeight + closePanelHeight<=minWindowHeight)
    {
        windowHeight = minWindowHeight;
    }
    else
    {
        windowHeight = 20 + imagePanelHeight + 20 + closePanelHeight;
    }


    createImagePanel();
    createClosePanel();

    addPanelsToContainer();

    setSize(windowWidth, windowHeight);
    setLocation(windowXPosition, windowYPosition);
    setTitle("Matriz Design");
    setVisible(true);
}

private void createImagePanel()
{
    imagePanel = new BlockImagePanel(imagePanelWidth,
                                     imagePanelHeight,
                                     rectangleWidth,
                                     rectangleHeight,
                                     designMatrixArray[0].length);
    imagePanel.setPreferredSize(new Dimension(imagePanelWidth,
                                              imagePanelHeight));
    imagePanel.setMatrixBlocks(designMatrixArray);
}

private void createClosePanel()
{
    closePanel = new JPanel();

    closeWindowButton = new JButton("Fechar");
    closeWindowButton.addActionListener(this);
```

```
            closePanel.add(closeWindowButton);
        }

        private void addPanelsToContainer()
        {
            viewContainer = getContentPane();
            viewContainer.setLayout(new GridBagLayout());
            GridBagConstraints c = new GridBagConstraints();

            c.fill = GridBagConstraints.BOTH;

            c.gridx = 0;
            c.gridy = 0;
            c.gridwidth = 1;
            c.gridheight = 1;
            c.weightx = imagePanelWidth;
            c.weighty = imagePanelHeight;

            viewContainer.add(imagePanel, c);

            c.gridx = 0;
            c.gridy = 1;
            c.gridwidth = 1;
            c.gridheight = 1;
            c.weightx = 0;
            c.weighty = 0;

            viewContainer.add(closePanel, c);

        }
    }
```

```java
package fMRIWindows.viewers;

//
//  ViewerWindowModelArray.java
//
//
//  Created by José Gabriel Lira Gomes on 1/30/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import matematica.algebra.Matrix;
import windowUtils.GraphPanelsWindow;

public class ViewerWindowModelArray extends GraphPanelsWindow
{
    public ViewerWindowModelArray(double[] modelData)
    {
        super(new Matrix(modelData, false), "Modelo", 1);
    }
}
```

```java
package fMRIWindows.viewers;

//
//  ViewerWindowProfiles.java
//
//
//  Created by José Gabriel Lira Gomes on 9/16/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.awt.Container;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;

import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import windowUtils.FormatWindow;
import windowUtils.RadioButtonsWindow;
import windowUtils.TripleSliderPanel;
import dirUtils.DirPathGetter;
import fMRIWindows.MainWindow;
import fMRIWindows.MenuedFrame;
import fileUtils.WriteTextNumberFile;
import graphUtils.GraphPanel;

public class ViewerWindowProfiles extends MenuedFrame implements ChangeListener
{
    private double[][][] imageData;
    private String titleText;
    private MainWindow parentWindow;

    private int width;
    private int height;
    private int numberOfSlices;

    private int x;
    private int y;
    private int z;

    double[] xProfileArray;
    double[] yProfileArray;
    double[] zProfileArray;

    private Container viewerContainer;

    private GraphPanel xProfilePanel;
    private GraphPanel yProfilePanel;
    private GraphPanel zProfilePanel;
    private TripleSliderPanel arraySlidersPanel;
    private JPanel xDataPanel;
    private JPanel yDataPanel;
    private JPanel zDataPanel;
    private JPanel arrayExportPanel;

    private RadioButtonsWindow formatWindow;
    private JButton setFormatButton;

    private JButton dataExportButton;

    private JSlider xSlider;
    private JSlider ySlider;
    private JSlider zSlider;
```

```
private JLabel xDataLabel;
private JLabel yDataLabel;
private JLabel zDataLabel;

private int arrayImgsHeight = 200;
private int dataPanelHeight = 30;

private int xArrayImgWidth;
private int yArrayImgWidth;
private int zArrayImgWidth;

private int slidersPanelWidth;

private int windowWidth;
private int windowHeight;
private int windowXPosition;
private int windowYPosition = 413;

public ViewerWindowProfiles(double[][][] imageData,
                            String titleText,
                            MainWindow parentWindow)
{
    this.imageData = imageData;
    this.titleText = titleText;
    this.parentWindow = parentWindow;

    width = imageData.length;
    height = imageData[0].length;
    numberOfSlices = imageData[0][0].length;

    x = (int)Math.round(width/2);
    y = (int)Math.round(height/2);
    z = (int)Math.round(numberOfSlices/2);

    createGUI();

    setAll();
}

public void setPosition(int xPos, int yPos, int zPos)
{
    x = xPos;
    y = yPos;
    z = zPos;
}

/******* PAINT *******/

public void paint(Graphics g)
{
    super.paint(g);
}

public void stateChanged(ChangeEvent event)
{
    int xPos, yPos, zPos;

    xPos = xSlider.getValue();
    yPos = ySlider.getValue();
    zPos = zSlider.getValue();

    setPosition(xPos, yPos, zPos);
    getProfiles();

    setAll();
}

public void actionPerformed(ActionEvent event)
{
    DirPathGetter dirPathGetter;
    String dirPath;
```

```
        String xPath;
        String yPath;
        String zPath;

        JButton clickedButton = (JButton) event.getSource();

        if(clickedButton == dataExportButton)
        {
            formatWindow = new FormatWindow();

            setFormatButton = formatWindow.acceptButton;
            setFormatButton.addActionListener(this);

            parentWindow.setVisible(false);
            this.setVisible(false);
        }
        if(clickedButton == setFormatButton)
        {
            formatWindow.setVisible(false);

            dirPathGetter = new DirPathGetter();
            dirPath = dirPathGetter.getPath();

            xPath = dirPath + "/xPerfil("+Integer.toString(x)+","
                                         +Integer.toString(y)+","
                                         +Integer.toString(z)+").txt";
            yPath = dirPath + "/yPerfil("+Integer.toString(x)+","
                                         +Integer.toString(y)+","
                                         +Integer.toString(z)+").txt";
            zPath = dirPath + "/zPerfil("+Integer.toString(x)+","
                                         +Integer.toString(y)+","
                                         +Integer.toString(z)+").txt";
            new WriteTextNumberFile(xProfileArray,
                                    xPath,
                                    formatWindow.getSelection());
            new WriteTextNumberFile(yProfileArray,
                                    yPath,
                                    formatWindow.getSelection());
            new WriteTextNumberFile(zProfileArray,
                                    zPath,
                                    formatWindow.getSelection());

            this.setVisible(true);
            parentWindow.setVisible(true);
        }
    }

    private void setAll()
    {
        setProfilesImages();

        setProfilesDataLabels();
    }

    private void getProfiles()
    {
        xProfileArray = getXProfileArray(x, y, z);
        yProfileArray = getYProfileArray(x, y, z);
        zProfileArray = getZProfileArray(x, y, z);
    }

    private void setProfilesImages()
    {
        xProfilePanel.updateSingleArray(xProfileArray);
        yProfilePanel.updateSingleArray(yProfileArray);
        zProfilePanel.updateSingleArray(zProfileArray);
    }

    private void setProfilesDataLabels()
    {
        xDataLabel.setText("V(x," + y + "," + z + ")");
```

```
        yDataLabel.setText("V(" + x + ",y," + z + ")");
        zDataLabel.setText("V(" + x + "," + y + ",z)");
    }

    private void createGUI()
    {
        xArrayImgWidth = 2*width;
        yArrayImgWidth = 2*height;
        zArrayImgWidth = 2*numberOfSlices;
        slidersPanelWidth = 180;

        windowWidth = xArrayImgWidth + yArrayImgWidth
                        + zArrayImgWidth + slidersPanelWidth;
        windowHeight = arrayImgsHeight + dataPanelHeight + 50;
        windowXPosition = parentWindow.getWidth() + 1;

        setPosition(x, y, z);
        getProfiles();

        createProfilePanels();
        createSliderPanel();
        createDataPanels();
        createExportPanel();

        addPanelsToContainer();

        setSize(windowWidth, windowHeight);
        setLocation(windowXPosition, windowYPosition);
        setTitle(titleText);
        setVisible(true);
    }

    private void createProfilePanels()
    {
        xProfilePanel = new GraphPanel(xProfileArray,
                                       xArrayImgWidth,
                                       arrayImgsHeight);
        yProfilePanel = new GraphPanel(yProfileArray,
                                       yArrayImgWidth,
                                       arrayImgsHeight);
        zProfilePanel = new GraphPanel(zProfileArray,
                                       zArrayImgWidth,
                                       arrayImgsHeight);

        xProfilePanel.setPreferredSize(new Dimension(xArrayImgWidth,
                                                     arrayImgsHeight));
        yProfilePanel.setPreferredSize(new Dimension(yArrayImgWidth,
                                                     arrayImgsHeight));
        zProfilePanel.setPreferredSize(new Dimension(zArrayImgWidth,
                                                     arrayImgsHeight));
    }

    private void createSliderPanel()
    {
        arraySlidersPanel = new TripleSliderPanel(width,
                                                  height,
                                                  numberOfSlices);

        this.xSlider = arraySlidersPanel.xSlider;
        this.xSlider.addChangeListener(this);

        this.ySlider = arraySlidersPanel.ySlider;
        this.ySlider.addChangeListener(this);

        this.zSlider = arraySlidersPanel.zSlider;
        this.zSlider.addChangeListener(this);
    }

    private void createDataPanels()
    {
        xDataPanel = new JPanel();
```

```
        yDataPanel = new JPanel();
        zDataPanel = new JPanel();

        xDataLabel = new JLabel();
        yDataLabel = new JLabel();
        zDataLabel = new JLabel();

        setProfilesDataLabels();

        xDataPanel.add(xDataLabel);
        yDataPanel.add(yDataLabel);
        zDataPanel.add(zDataLabel);
    }

    private void createExportPanel()
    {
        arrayExportPanel = new JPanel();

        dataExportButton = new JButton("Dados");
        dataExportButton.setEnabled(true);
        dataExportButton.addActionListener(this);

        arrayExportPanel.add(dataExportButton);
    }

    private void addPanelsToContainer()
    {
        viewerContainer = getContentPane();
        viewerContainer.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();

        c.fill = GridBagConstraints.BOTH;

        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = xArrayImgWidth;
        c.weighty = arrayImgsHeight;

        viewerContainer.add(xProfilePanel, c);

        c.gridx = 1;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = yArrayImgWidth;
        c.weighty = arrayImgsHeight;

        viewerContainer.add(yProfilePanel, c);

        c.gridx = 2;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = zArrayImgWidth;
        c.weighty = arrayImgsHeight;

        viewerContainer.add(zProfilePanel, c);

        c.gridx = 3;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewerContainer.add(arraySlidersPanel, c);

        c.gridx = 0;
        c.gridy = 1;
```

```
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = xArrayImgWidth;
        c.weighty = 0;

        viewerContainer.add(xDataPanel, c);

        c.gridx = 1;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = yArrayImgWidth;
        c.weighty = 0;

        viewerContainer.add(yDataPanel, c);

        c.gridx = 2;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = zArrayImgWidth;
        c.weighty = 0;

        viewerContainer.add(zDataPanel, c);

        c.gridx = 3;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewerContainer.add(arrayExportPanel, c);
    }

    /******* SHOW ARRAY ******

    private void showArray(double[] array)
    {
        int size = array.length;
        int i;

        for(i=0; i<size; i++)
        {
            System.out.println(array[i]);
        }
    }
    */


    /******* GET VOXEL PROFILE ALONG X *******/

    private double[] getXProfileArray(int x, int y, int z)
    {
        double[] xProfile = new double[width];
        int i;

        for(i=0; i<width; i++)
        {
            xProfile[i] = imageData[i][y][z];
        }

        return xProfile;
    }

    /******* GET VOXEL PROFILE ALONG Y *******/

    private double[] getYProfileArray(int x, int y, int z)
    {
        double[] yProfile = new double[height];
        int i;
```

```
        for(i=0; i<height; i++)
        {
            yProfile[i] = imageData[x][i][z];
        }

        return yProfile;
    }

    /******* GET VOXEL PROFILE ALONG Z *******/

    private double[] getZProfileArray(int x, int y, int z)
    {
        double[] zProfile = new double[numberOfSlices];
        int i;

        for(i=0; i<numberOfSlices; i++)
        {
            zProfile[i] = imageData[x][y][i];
        }

        return zProfile;
    }


}
```

```java
package fMRIWindows.viewers;

//
//  ViewerWindowResult.java
//
//
//  Created by Jose Gabriel Lira Gomes on 5/17/05.
//  Copyright 2005 __FDG__. All rights reserved.
//

import imageUtils.ImagePanel;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.io.IOException;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.JToggleButton;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import windowUtils.FormatWindow;
import windowUtils.RadioButtonsWindow;
import dirUtils.DirPathGetter;
import estatistica.descriptive.ThreeDArrayAnalyzer;
import fMRIWindows.MainWindow;
import fMRIWindows.MenuedFrame;
import fileUtils.WriteTextNumberFile;
import functional.data.AnlzHeader;
import functional.data.AnlzWriter;
import functional.export.JpegExporter;
import graphUtils.GraphPanel;

public class ViewerWindowResult extends MenuedFrame
    implements ChangeListener, MouseListener, MouseMotionListener, ItemListener
{
    //
    //  Data Members
    //

    private double[][][] imageData;
    private int numberOfVoxels;
    private int zoomFactor;
    private String method;
    MainWindow parentWindow;

    private int width;
    private int height;
    private int numberOfSlices;
    private short[] imageDims;

    private int x;
    private int y;
    private int z;

    private double threshold;
```

```
    private double voxelVolume;
    private String volumeText;
    private String activeVoxelsPercText;

    private int xArrayImgWidth;
    private int yArrayImgWidth;
    private int zArrayImgWidth;

    private int profilesPanelWidth;
    private int profilePanelsHeight;

    private int dataPanelHeight;

    private int windowWidth;
    private int windowHeight;
    private int windowXPosition;
    private int windowYPosition = 21;

    double[] xProfileArray;
    double[] yProfileArray;
    double[] zProfileArray;

    private Container viewContainer;
    private GraphPanel xProfilePanel;
    private GraphPanel yProfilePanel;
    private GraphPanel zProfilePanel;
    private JPanel profilesPanel;
    private ImagePanel imagePanel;
    private JPanel toolsPanel;
    private JPanel dataPanel;
    private JPanel zSliderPanel;
    private JPanel thresSliderPanel;
    private JPanel exportPanel;

    private boolean bckgVisible = false;
    private double[][][] bckgData;
    private double[][] bckgSliceData;
    private double[][] imageSliceData;

    private JButton xDataButton;
    private JButton yDataButton;
    private JButton zDataButton;
    private JSlider zSlider;
    private JSlider thresSlider;
    private JToggleButton bckgButton;
    private JLabel dataZValue;
    private JLabel dataThres;
    private JLabel dataVValue;
    private JButton jpegExportButton;
    private JButton analyzeExportButton;

    private boolean xAxis = false;
    private boolean yAxis = false;
    private boolean zAxis = false;

    private RadioButtonsWindow formatWindow;
    private JButton setFormatButton;

    private AnlzHeader resultDataHeader;

    private long startTime;
    private long stopTime;
    private long elapsedTime;

    private int mouseX = 0;
    private int mouseY = 0;



    //
    //  Constructors
```

```
//
//      ViewerWindowResult(double[][][] imageData,
//                         int numberOfVoxels,
//                         int zoomFactor,
//                         String method,
//                         MainWindow parentWindow)
//

public ViewerWindowResult(double[][][] imageData,
                          int numberOfVoxels,
                          int zoomFactor,
                          String method,
                          MainWindow parentWindow)
{
    this.imageData = imageData;
    this.numberOfVoxels = numberOfVoxels;
    this.zoomFactor = zoomFactor;
    this.method = method;
    this.parentWindow = parentWindow;

    getImageDims();
    getInitialValues();

    createGUI();

    imagePanel.addMouseListener(this);
    imagePanel.addMouseMotionListener(this);
}

public ViewerWindowResult(double[][][] imageData,
                          double[][][] bckgData,
                          int numberOfVoxels,
                          int zoomFactor,
                          String method,
                          MainWindow parentWindow)
{
    this.imageData = imageData;
    this.bckgData = bckgData;
    this.numberOfVoxels = numberOfVoxels;
    this.zoomFactor = zoomFactor;
    this.method = method;
    this.parentWindow = parentWindow;

    getImageDims();
    getInitialValues();

    bckgVisible = true;
    createGUI();

    imagePanel.addMouseListener(this);
    imagePanel.addMouseMotionListener(this);
}

public void disableBckgButton()
{
    this.bckgButton.setEnabled(false);
}

/******* LISTEN TO SLIDER CHANGE EVENT *******/

public void stateChanged(ChangeEvent event)
{
    int intThres;
    String pValueText;
    String zPositionText;

    intThres = thresSlider.getValue();
    threshold = (double)intThres/100.0;
    pValueText = Double.toString(Math.round((1.0 - threshold)*100.0)/100.0);
    setVolumeInfo(threshold);
```

```
        this.z = zSlider.getValue();
        zPositionText = Integer.toString(z);

        dataThres.setText("p = " + pValueText);
        dataZValue.setText("Z = " + zPositionText);
        dataVValue.setText("V = " + volumeText
                           + " cm3 (" + activeVoxelsPercText + " %)");

        if(bckgVisible)
        {
            this.bckgSliceData = getSlice(bckgData, z);
            this.imagePanel.setBckgImageData(bckgSliceData);
            this.imageSliceData = getSlice(imageData, z);
            this.imagePanel.setImageDataAndThreshold(imageSliceData,
                                                     threshold,
                                                     1);
        }
        else
        {
            this.imageSliceData = getSlice(imageData, z);
            this.imagePanel.setImageDataAndThreshold(imageSliceData,
                                                     threshold,
                                                     0);
        }

    }


    public void itemStateChanged(ItemEvent event)
    {
        JToggleButton source = (JToggleButton) event.getSource();

        if(source == bckgButton)
        {
            if(event.getStateChange() == ItemEvent.SELECTED)
            {
                this.bckgVisible = true;

                this.bckgSliceData = getSlice(bckgData, z);
                this.imagePanel.setBckgImageData(bckgSliceData);
                this.imagePanel.showBackground();
            }
            else
            {
                this.bckgVisible = false;

                this.imagePanel.hideBackground();
            }
        }
    }


    /******* LISTEN TO BUTTON ACTION EVENT *******/

    public void actionPerformed(ActionEvent event)
    {
        DirPathGetter dirPathGetter;
        String dirPath;
        String xPath;
        String yPath;
        String zPath;

        JButton clickedButton = (JButton) event.getSource();

        if(clickedButton == xDataButton)
        {
            xAxis = true;
            formatWindow = new FormatWindow();

            setFormatButton = formatWindow.acceptButton;
            setFormatButton.addActionListener(this);
```

```
            parentWindow.setVisible(false);
            this.setVisible(false);
        }
        if(clickedButton == yDataButton)
        {
            yAxis = true;
            formatWindow = new FormatWindow();

            setFormatButton = formatWindow.acceptButton;
            setFormatButton.addActionListener(this);

            parentWindow.setVisible(false);
            this.setVisible(false);
        }
        if(clickedButton == zDataButton)
        {
            zAxis = true;
            formatWindow = new FormatWindow();

            setFormatButton = formatWindow.acceptButton;
            setFormatButton.addActionListener(this);

            parentWindow.setVisible(false);
            this.setVisible(false);
        }
        if(clickedButton == setFormatButton)
        {
            formatWindow.setVisible(false);

            dirPathGetter = new DirPathGetter();
            dirPath = dirPathGetter.getPath();

            if(xAxis)
            {
                xPath = dirPath + "/xPerfil("+Integer.toString(x)+","
                                             +Integer.toString(y)+","
                                             +Integer.toString(z)+").txt";
                new WriteTextNumberFile(xProfileArray,
                                        xPath,
                                        formatWindow.getSelection());
                xAxis = false;
            }
            if(yAxis)
            {
                yPath = dirPath + "/yPerfil("+Integer.toString(x)+","
                                             +Integer.toString(y)+","
                                             +Integer.toString(z)+").txt";
                new WriteTextNumberFile(yProfileArray,
                                        yPath,
                                        formatWindow.getSelection());
                yAxis = false;
            }
            if(zAxis)
            {
                zPath = dirPath + "/zPerfil("+Integer.toString(x)+","
                                             +Integer.toString(y)+","
                                             +Integer.toString(z)+").txt";
                new WriteTextNumberFile(zProfileArray,
                                        zPath,
                                        formatWindow.getSelection());
                zAxis = false;
            }

            this.setVisible(true);
            parentWindow.setVisible(true);
        }
        if(clickedButton == jpegExportButton)
        {
            dirPathGetter = new DirPathGetter();
            dirPath = dirPathGetter.getPath();
```

```
            if(bckgVisible)
            {
                new JpegExporter(imageData,
                                 bckgData,
                                 threshold,
                                 4,
                                 true,
                                 dirPath,
                                 method);
            }
            else
            {
                new JpegExporter(imageData,
                                 threshold,
                                 4,
                                 true,
                                 dirPath,
                                 method);
            }
        }
        if(clickedButton == analyzeExportButton)
        {
            dirPathGetter = new DirPathGetter();
            dirPath = dirPathGetter.getPath();

            resultDataHeader = new AnlzHeader();

            resultDataHeader.setDataType("dbl");
            resultDataHeader.setNumberOfDims((short)4);
            resultDataHeader.setDimensions(imageDims);
            resultDataHeader.setDataTypeCode((short)64);
            resultDataHeader.setBitsPerPixel((short)64);
            resultDataHeader.setPixelsDims(parentWindow.getPixelsDims());
            resultDataHeader.setFUnusedOne(parentWindow.getFUnusedOne());
            resultDataHeader.setGLMax(1);
            resultDataHeader.setGLMin(0);
            resultDataHeader.setDescrip("Resultado " + method);

            System.out.println("Inicio da escrita dos resultados");
            startTime = System.currentTimeMillis();
            try
            {
                new AnlzWriter(resultDataHeader,
                               imageData,
                               dirPath + "/" + method);
            }
            catch (IOException e)
            {
                System.out.println("failed analyze file writing");
                e.printStackTrace();
            }
            stopTime = System.currentTimeMillis();
            System.out.println("Fim da escrita dos resultados");
            elapsedTime = stopTime - startTime;
            System.out.println("Tempo da escrita dos resultados:");
            System.out.println(elapsedTime + " ms");
            System.out.println("");
        }
    }

    public void paint(Graphics g)
    {
        super.paint(g);
    }


    public void mouseClicked(MouseEvent me)
    {
        mouseX = me.getX();
        mouseY = me.getY();
```

```
        if(me.isMetaDown())
        {

        }
        else
        {
            if(me.isAltDown())
            {
                if(zoomFactor>1)
                {
                    zoomFactor = zoomFactor/2;
                    imagePanel.setZoomFactor(zoomFactor);
                }
            }
            else
            {
                if(me.isControlDown())
                {
                    zoomFactor = zoomFactor*2;
                    imagePanel.setZoomFactor(zoomFactor);
                }
                else
                {
                    x = (int)Math.floor(mouseX/zoomFactor);
                    y = height - 1 - (int)Math.floor(mouseY/zoomFactor);
                    if((x<width)&&(x>=0)&&(y<height)&&(y>=0))
                    {
                        getProfiles();

                        setAll();
                    }
                }
            }
        }
    }


    public void mouseEntered(MouseEvent me)
    {

    }


    public void mouseExited(MouseEvent me)
    {

    }


    public void mousePressed(MouseEvent me)
    {

    }


    public void mouseReleased(MouseEvent me)
    {

    }


    public void mouseDragged(MouseEvent me)
    {

    }


    public void mouseMoved(MouseEvent me)
    {

    }
```

```
private void getImageDims()
{
    this.width = imageData.length;
    this.height = imageData[0].length;
    this.numberOfSlices = imageData[0][0].length;

    imageDims = new short[8];
    imageDims[0] = 4;
    imageDims[1] = (short)width;
    imageDims[2] = (short)height;
    imageDims[3] = (short)numberOfSlices;
    imageDims[4] = 1;

    voxelVolume = parentWindow.getVoxelVolume();
}


private void getInitialValues()
{
    int xInitialValue;
    int yInitialValue;
    int zInitialValue;

    double initialThres;

    xInitialValue = (int)Math.round(width/2);
    yInitialValue = (int)Math.round(height/2);
    zInitialValue = (int)Math.round(numberOfSlices/2);

    initialThres = 0.95;

    this.x = xInitialValue;
    this.y = yInitialValue;
    this.z = zInitialValue;

    this.threshold = initialThres;
    setVolumeInfo(threshold);
}


private void createGUI()
{
    xArrayImgWidth = 2*width;
    yArrayImgWidth = 2*height;
    zArrayImgWidth = 2*numberOfSlices;
    profilesPanelWidth = xArrayImgWidth+yArrayImgWidth+zArrayImgWidth;
    profilePanelsHeight = 200;
    dataPanelHeight = 42;

    windowXPosition = parentWindow.getWidth() + 1;
    windowWidth = 26 + Math.max(profilesPanelWidth, zoomFactor*width+79);
    windowHeight = 200 + dataPanelHeight + zoomFactor*height + 85 + 80 + 20;

    getProfiles();

    createProfilesPanel();
    createZSliderPanel();
    createImagePanel();
    createThresSliderPanel();
    createToolsPanel();
    createDataPanel();
    createExportPanel();

    addPanelsToContainer();

    setSize(windowWidth, windowHeight);
    setLocation(windowXPosition, windowYPosition);
    setTitle(method);
    setVisible(true);
}
```

```
private void createProfilesPanel()
{
    createProfilePanels();
    createProfileDataLabels();

    profilesPanel = new JPanel(new GridBagLayout());
    profilesPanel.setPreferredSize(new Dimension(profilesPanelWidth,
                                                 profilePanelsHeight+42));

    GridBagConstraints c = new GridBagConstraints();


    c.fill = GridBagConstraints.BOTH;

    c.gridx = 0;
    c.gridy = 0;
    c.gridwidth = 1;
    c.gridheight = 1;
    c.weightx = 0;
    c.weighty = 0;

    profilesPanel.add(xProfilePanel, c);

    c.gridx = 1;
    c.gridy = 0;
    c.gridwidth = 1;
    c.gridheight = 1;
    c.weightx = 0;
    c.weighty = 0;

    profilesPanel.add(yProfilePanel, c);

    c.gridx = 2;
    c.gridy = 0;
    c.gridwidth = 1;
    c.gridheight = 1;
    c.weightx = 0;
    c.weighty = 0;

    profilesPanel.add(zProfilePanel, c);

    c.fill = GridBagConstraints.CENTER;

    c.gridx = 0;
    c.gridy = 1;
    c.gridwidth = 1;
    c.gridheight = 1;
    c.weightx = 0;
    c.weighty = 0;

    profilesPanel.add(xDataButton, c);

    c.gridx = 1;
    c.gridy = 1;
    c.gridwidth = 1;
    c.gridheight = 1;
    c.weightx = 0;
    c.weighty = 0;

    profilesPanel.add(yDataButton, c);

    c.gridx = 2;
    c.gridy = 1;
    c.gridwidth = 1;
    c.gridheight = 1;
    c.weightx = 0;
    c.weighty = 0;

    profilesPanel.add(zDataButton, c);
```

```
}

private void createProfilePanels()
{
    xProfilePanel = new GraphPanel(xProfileArray,
                                   xArrayImgWidth,
                                   profilePanelsHeight);
    yProfilePanel = new GraphPanel(yProfileArray,
                                   yArrayImgWidth,
                                   profilePanelsHeight);
    zProfilePanel = new GraphPanel(zProfileArray,
                                   zArrayImgWidth,
                                   profilePanelsHeight);
}

private void createProfileDataLabels()
{
    xDataButton = new JButton();
    yDataButton = new JButton();
    zDataButton = new JButton();

    xDataButton.addActionListener(this);
    yDataButton.addActionListener(this);
    zDataButton.addActionListener(this);

    setProfilesDataLabels();
}


private void createZSliderPanel()
{
    zSliderPanel = new JPanel(new BorderLayout());
    zSliderPanel.setBorder(BorderFactory.createTitledBorder("Z"));
    zSlider = createZSlider(z);
    zSliderPanel.add(zSlider, BorderLayout.CENTER);
}


private void createImagePanel()
{
    //ThreeDArrayAnalyzer imageDataAnalyzer =
    //                  new ThreeDArrayAnalyzer(imageData);
    //double imageMax = imageDataAnalyzer.getMaximum();
    //double imageMin = imageDataAnalyzer.getMinimum();
    double imageMax = 1.0;
    double imageMin = 0.0;

    double bckgMax;
    double bckgMin;

    imageSliceData = getSlice(imageData, z);

    if(bckgVisible)
    {
        ThreeDArrayAnalyzer bckgDataAnalyzer =
                            new ThreeDArrayAnalyzer(bckgData);
        bckgMax = bckgDataAnalyzer.getMaximum();
        bckgMin = bckgDataAnalyzer.getMinimum();

        bckgSliceData = getSlice(bckgData, z);
        imagePanel = new ImagePanel(bckgSliceData,
                                    imageSliceData,
                                    zoomFactor,
                                    threshold,
                                    bckgMax,
                                    bckgMin,
                                    imageMax,
                                    imageMin,
                                    false,
                                    true);
    }
```

```
        else
        {
            imagePanel = new ImagePanel(imageSliceData,
                                        zoomFactor,
                                        threshold,
                                        imageMax,
                                        imageMin,
                                        true);
        }
    }


    private void createThresSliderPanel()
    {
        thresSliderPanel = new JPanel(new BorderLayout());
        thresSliderPanel.setBorder(BorderFactory.createTitledBorder(
                                                            "Threshold"));
        thresSlider = createThresSlider((int)(threshold*100));
        thresSliderPanel.add(thresSlider, BorderLayout.CENTER);
    }

    private void createToolsPanel()
    {
        toolsPanel = new JPanel(new GridLayout(2,1));
        toolsPanel.setPreferredSize(new Dimension(79,79));
        bckgButton = new JToggleButton("Fundo");
        bckgButton.setSelected(true);
        bckgButton.addItemListener(this);
        toolsPanel.add(bckgButton);
    }

    private void createDataPanel()
    {
        dataZValue = new JLabel("Z = " + z);
        dataThres = new JLabel("p = 0.05");
        dataVValue = new JLabel("V = " + volumeText + " cm3 ("
                                        + activeVoxelsPercText + " %)");

        dataPanel = new JPanel(new GridBagLayout());
        dataPanel.setMinimumSize(new Dimension(100,70));
        dataPanel.setBorder(BorderFactory.createTitledBorder("Dados"));

        GridBagConstraints c = new GridBagConstraints();

        c.fill = GridBagConstraints.BOTH;

        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0.5;
        c.weighty = 0.4;

        dataPanel.add(dataZValue, c);

        c.gridx = 1;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0.5;
        c.weighty = 0.4;

        dataPanel.add(dataThres, c);

        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 2;
        c.gridheight = 1;
        c.weightx = 1;
        c.weighty = 0.6;
```

```
        dataPanel.add(dataVValue, c);
    }

    private void createExportPanel()
    {
        exportPanel = new JPanel(new GridLayout(2,1));
        exportPanel.setPreferredSize(new Dimension(79,79));
        jpegExportButton = new JButton("JPEG");
        analyzeExportButton = new JButton("ANLZ");
        exportPanel.add(jpegExportButton);
        exportPanel.add(analyzeExportButton);
        jpegExportButton.addActionListener(this);
        analyzeExportButton.addActionListener(this);
    }

    private void addPanelsToContainer()
    {
        viewContainer = getContentPane();
        viewContainer.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();

        c.fill = GridBagConstraints.BOTH;

        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 3;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewContainer.add(profilesPanel, c);

        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewContainer.add(zSliderPanel, c);

        c.gridx = 1;
        c.gridy = 1;
        c.gridwidth = 2;
        c.gridheight = 1;
        c.weightx = width*zoomFactor;
        c.weighty = height*zoomFactor;

        viewContainer.add(imagePanel, c);

        c.gridx = 0;
        c.gridy = 2;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewContainer.add(toolsPanel, c);

        c.gridx = 1;
        c.gridy = 2;
        c.gridwidth = 2;
        c.gridheight = 1;
        c.weightx = 1;
        c.weighty = 0;

        viewContainer.add(thresSliderPanel, c);

        c.gridx = 0;
        c.gridy = 3;
        c.gridwidth = 1;
```

```
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        viewContainer.add(exportPanel, c);

        c.gridx = 1;
        c.gridy = 3;
        c.gridwidth = 2;
        c.gridheight = 1;
        c.weightx = 1;
        c.weighty = 0;

        viewContainer.add(dataPanel, c);
    }

    private void setProfilesDataLabels()
    {
        xDataButton.setText("V(x," + y + "," + z + ")");
        yDataButton.setText("V(" + x + ",y," + z + ")");
        zDataButton.setText("V(" + x + "," + y + ",z)");
    }


    private JSlider createThresSlider(int initialValue)
    {
        JSlider thresSlider = new JSlider();

        thresSlider.setOrientation(JSlider.HORIZONTAL);
        thresSlider.setPaintLabels(true);
        thresSlider.setPaintTicks(true);
        thresSlider.setMinimum(0);
        thresSlider.setMaximum(100);
        thresSlider.setValue(initialValue);
        thresSlider.setMajorTickSpacing(25);
        thresSlider.setMinorTickSpacing(5);

        thresSlider.addChangeListener(this);

        return thresSlider;
    }


    private JSlider createZSlider(int initialValue)
    {
        JSlider zSlider = new JSlider();

        zSlider.setOrientation(JSlider.VERTICAL);
        zSlider.setPaintLabels(true);
        zSlider.setPaintTicks(true);
        zSlider.setMinimum(0);
        zSlider.setMaximum(numberOfSlices-1);
        zSlider.setValue(initialValue);
        zSlider.setMajorTickSpacing(10);
        zSlider.setMinorTickSpacing(5);

        zSlider.addChangeListener(this);

        return zSlider;
    }

    private void getProfiles()
    {
        xProfileArray = getXProfileArray(x, y, z);
        yProfileArray = getYProfileArray(x, y, z);
        zProfileArray = getZProfileArray(x, y, z);
    }

    /******* GET VOXEL PROFILE ALONG X *******/

    private double[] getXProfileArray(int x, int y, int z)
```

```
{
    double[] xProfile = new double[width];
    int i;

    for(i=0; i<width; i++)
    {
        xProfile[i] = imageData[i][y][z];
    }

    return xProfile;
}


/******* GET VOXEL PROFILE ALONG Y *******/

private double[] getYProfileArray(int x, int y, int z)
{
    double[] yProfile = new double[height];
    int i;

    for(i=0; i<height; i++)
    {
        yProfile[i] = imageData[x][i][z];
    }

    return yProfile;
}

/******* GET VOXEL PROFILE ALONG Z *******/

private double[] getZProfileArray(int x, int y, int z)
{
    double[] zProfile = new double[numberOfSlices];
    int i;

    for(i=0; i<numberOfSlices; i++)
    {
        zProfile[i] = imageData[x][y][i];
    }

    return zProfile;
}

private double[][] getSlice(double[][][] volumeDataArray, int z)
{
    int x, y;
    double[][] sliceDataArray = new double[width][height];

    for(y=0; y<height; y++)
    {
        for(x=0; x<width; x++)
        {
            sliceDataArray[x][y] = volumeDataArray[x][y][z];
        }
    }

    return sliceDataArray;
}

private void setAll()
{
    setProfilesImages();

    setProfilesDataLabels();
}

private void setProfilesImages()
{
    xProfilePanel.updateSingleArray(xProfileArray);
    yProfilePanel.updateSingleArray(yProfileArray);
    zProfilePanel.updateSingleArray(zProfileArray);
```

```
    }

    private void setVolumeInfo(double threshold)
    {
        int x, y, z;

        int numberOfActiveVoxels = 0;
        double volume;
        double activeVoxelsPerc;

        for(x=0; x<width; x++)
        {
            for(y=0; y<height; y++)
            {
                for(z=0; z<numberOfSlices; z++)
                {
                    if(imageData[x][y][z] >= threshold)
                    {
                        numberOfActiveVoxels = numberOfActiveVoxels + 1;
                    }
                }
            }
        }

        volume = (double)numberOfActiveVoxels*voxelVolume;
        activeVoxelsPerc = (double)numberOfActiveVoxels/numberOfVoxels;

        this.volumeText = Double.toString(Math.round(volume*10.0)/10.0);
        this.activeVoxelsPercText =
            Double.toString(Math.round(activeVoxelsPerc*1000.0)/10.0);
    }
}
```

```java
package functional.data;

//
//  Voxel.java
//
//
//  Created by José Gabriel Lira Gomes on 1/19/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

public class Voxel
{
    private int x;
    private int y;
    private int z;

    public Voxel(int x, int y, int z)
    {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public int getX()
    {
        return x;
    }

    public int getY()
    {
        return y;
    }

    public int getZ()
    {
        return z;
    }
}
```

```java
package fileUtils;

//
//  WriteTextFile.java
//
//
//  Created by JGLG on 1/13/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.io.*;

public class WriteTextFile
{
    public WriteTextFile(String[] data, String filePath, int format)
    {
        int i;

        FileOutputStream fOut;
        DataOutputStream out;

        int dataArraySize = data.length;

        boolean notLastLine = true;

        byte[] textBytes;
        int j;

        try
        {
            fOut = new FileOutputStream(filePath);
            out = new DataOutputStream(fOut);

            /******* imprime dados *******/
            for(i=0; i< dataArraySize; i++)
            {
                if(i == dataArraySize-1)
                {
                    notLastLine = false;
                }

                textBytes = data[i].getBytes();
                for(j=0; j<textBytes.length; j++)
                {
                    out.writeByte(textBytes[j]);
                }

                if(notLastLine)
                {
                    if(format == 0) // Apple
                    {
                        out.writeByte(13);
                    }
                    if(format == 1) // Unix
                    {
                        out.writeByte(10);
                    }
                    if(format == 2) // DOS
                    {
                        out.writeByte(13);
                        out.writeByte(10);
                    }
                }
            }

            /******* fecha o ficheiro final *******/
            out.close();
        }
        catch(FileNotFoundException exc)
        {
            System.out.println("File Not Found");
```

```
                return;
            }
            catch(IOException exc)
            {
                System.out.println(exc);
                System.exit(1);
            }
        }
    }
```

```
package fileUtils;

//
//  WriteTextNumberFile.java
//
//
//  Created by JGLG on 1/11/06.
//  Copyright 2006 __FDG__. All rights reserved.
//

import java.io.*;

public class WriteTextNumberFile
{
    private double[] data;
    private String filePath;
    private int format;

    public WriteTextNumberFile(double[] data, String filePath, int format)
    {
        this.data = data;
        this.filePath = filePath;
        this.format = format;

        writeFile();
    }

    private void writeFile()
    {
        int i;

        FileOutputStream fOut;
        DataOutputStream out;

        int dataArraySize = data.length;

        boolean notLastLine = true;

        String numberText;
        byte[] textBytes;
        int j;

        try
        {
            fOut = new FileOutputStream(filePath);
            out = new DataOutputStream(fOut);

            /******* imprime dados *******/
            for(i=0; i< dataArraySize; i++)
            {
                if(i == dataArraySize-1)
                {
                    notLastLine = false;
                }

                numberText = Double.toString(data[i]);
                textBytes = numberText.getBytes();
                for(j=0; j<textBytes.length; j++)
                {
                    out.writeByte(textBytes[j]);
                }

                if(notLastLine)
                {
                    if(format == 0) // Apple
                    {
                        out.writeByte(13);
                    }
                    if(format == 1) // Unix
                    {
                        out.writeByte(10);
```

```
                    }
                    if(format == 2) // DOS
                    {
                        out.writeByte(13);
                        out.writeByte(10);
                    }
                }
            }

            /******* fecha o ficheiro final *******/
            out.close();
        }
        catch(FileNotFoundException exc)
        {
            System.out.println("File Not Found");
            return;
        }
        catch(IOException exc)
        {
            System.out.println(exc);
            System.exit(1);
        }
    }
}
```

```java
//
//  Comparador.java
//
//
//  Created by José Gabriel Lira Gomes on 4/24/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import windows.MainWindow;

public class Comparador
{
    public Comparador()
    {
        new MainWindow();
    }

    public static void main(String args[])
    {
        new Comparador();
    }
}
```

```java
package windows;

//
//  MainWindow.java
//
//
//  Created by José Gabriel Lira Gomes on 4/24/07.
//  Copyright 2007 __FDG__. All rights reserved.
//

import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;

import javax.swing.JButton;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

import windows.viewers.ViewerWindowComparer;
import fileUtils.ExtensionsFileFilter;
import fileUtils.FilePathGetter;
import functional.data.AnlzReader;

public class MainWindow extends MenuedFrame
{
    private Container mainContainer;

    private JPanel loadFilesPanel;

    private JButton loadAnlzOneButton;
    private JButton loadAnlzTwoButton;
    private JButton viewButton;

    private FilePathGetter pathGetter;
    private String[] validExtensions = {"hdr"};
    private ExtensionsFileFilter extensionsFilter =
                new ExtensionsFileFilter(validExtensions);

    private String fileOnePath;
    private String fileTwoPath;

    private AnlzReader anlzFileOne;
    private AnlzReader anlzFileTwo;

    private boolean validFileOne = false;
    private boolean validFileTwo = false;

    private short[] probArrayOneDims;
    private short[] probArrayTwoDims;

    private float[] probArrayOnePixDims;
    private float[] probArrayTwoPixDims;
    private float[] pixelsDims;

    private float scaleFactorOne;
    private float scaleFactorTwo;
    private float scaleFactor;

    private double[][][] probArrayOne;
    private double[][][] probArrayTwo;

    private double[][][] probArraysDif;

    private long startTime;
    private long stopTime;
    private long elapsedTime;

    private int mainWindowWidth = 300;
    private int mainWindowHeight = 80;
    private int mainWindowXPosition = 0;
    private int mainWindowYPosition = 21;
```

```
public MainWindow()
{
    createGUI();
}

public float[] getPixelsDims()
{
    return pixelsDims;
}

public double getVoxelVolume()
{
    double voxelVolume = 1.0;

    for(int i=1; i<4; i++)
    {
        voxelVolume = voxelVolume*pixelsDims[i]/10.0;
    }

    return voxelVolume;
}

public float getFUnusedOne()
{
    return scaleFactor;
}

public void actionPerformed(ActionEvent event)
{
    JButton clickedButton = (JButton) event.getSource();

    if(clickedButton == loadAnlzOneButton)
    {
        loadAnlzOneButton.setEnabled(false);
        this.setVisible(false);

        pathGetter = new FilePathGetter(extensionsFilter);
        fileOnePath = pathGetter.getPath();

        try
        {
            System.out.println("Inicio da leitura");
            startTime = System.currentTimeMillis();
            anlzFileOne = new AnlzReader(fileOnePath);
            probArrayOneDims = anlzFileOne.getDimensions();
            probArrayOnePixDims = anlzFileOne.getPixelsDims();
            scaleFactorOne = anlzFileOne.getFUnusedOne();
            probArrayOne = anlzFileOne.getRealImageDataVolume(0);

            stopTime = System.currentTimeMillis();
            System.out.println("Fim da leitura");
            elapsedTime = stopTime - startTime;
            System.out.println("Tempo de leitura:");
            System.out.println(elapsedTime + " ms");
            System.out.println("");

            if(validFileTwo)
            {
                if(areEqual(probArrayOneDims, probArrayTwoDims))
                {
                    if(areEqual(probArrayOnePixDims, probArrayTwoPixDims))
                    {
                        if(scaleFactorOne == scaleFactorTwo)
                        {
                            pixelsDims = probArrayOnePixDims;
                            scaleFactor = scaleFactorOne;
                            validFileOne = true;
                        }
                        else
```

```
                        {
                            validFileOne = false;
                            loadAnlzTwoButton.setEnabled(true);
                            JOptionPane.showMessageDialog(this,
                                "Os factores de escala sao diferentes",
                                "Erro de ficheiro",
                                JOptionPane.ERROR_MESSAGE);
                        }
                    }
                    else
                    {
                        validFileOne = false;
                        loadAnlzTwoButton.setEnabled(true);
                        JOptionPane.showMessageDialog(this,
                                "As dimensoes sao diferentes",
                                "Erro de ficheiro",
                                JOptionPane.ERROR_MESSAGE);
                    }
                }
                else
                {
                    validFileOne = false;
                    loadAnlzTwoButton.setEnabled(true);
                    JOptionPane.showMessageDialog(this,
                                "As dimensoes sao diferentes",
                                "Erro de ficheiro",
                                JOptionPane.ERROR_MESSAGE);
                }
            }
            else
            {
                validFileOne = true;
            }

            if((validFileOne)&&(validFileTwo))
            {
                viewButton.setEnabled(true);
            }
        }
        catch(Exception e)
        {
            if(e.getMessage().equals("not hdr"))
            {
                JOptionPane.showMessageDialog(this,
                                "Nao escolheu um *.hdr",
                                "Erro de formato",
                                JOptionPane.ERROR_MESSAGE);
            }
            else
            {
                JOptionPane.showMessageDialog(this,
                                "O header nao tem 348 bytes",
                                "Erro de formato",
                                JOptionPane.ERROR_MESSAGE);
            }
            loadAnlzOneButton.setEnabled(true);
        }

        this.setVisible(true);
    }
    if(clickedButton == loadAnlzTwoButton)
    {
        loadAnlzTwoButton.setEnabled(false);
        this.setVisible(false);

        pathGetter = new FilePathGetter(extensionsFilter);
        fileTwoPath = pathGetter.getPath();

        try
        {
            System.out.println("Inicio da leitura");
```

```
            startTime = System.currentTimeMillis();
            anlzFileTwo = new AnlzReader(fileTwoPath);
            probArrayTwoDims = anlzFileTwo.getDimensions();
            probArrayTwoPixDims = anlzFileTwo.getPixelsDims();
            scaleFactorTwo = anlzFileTwo.getFUnusedOne();
            probArrayTwo = anlzFileTwo.getRealImageDataVolume(0);

            stopTime = System.currentTimeMillis();
            System.out.println("Fim da leitura");
            elapsedTime = stopTime - startTime;
            System.out.println("Tempo de leitura:");
            System.out.println(elapsedTime + " ms");
            System.out.println("");

            if(validFileOne)
            {
                if(areEqual(probArrayOneDims, probArrayTwoDims))
                {
                    if(areEqual(probArrayOnePixDims, probArrayTwoPixDims))
                    {
                        if(scaleFactorOne == scaleFactorTwo)
                        {
                            pixelsDims = probArrayOnePixDims;
                            scaleFactor = scaleFactorOne;
                            validFileTwo = true;
                        }
                        else
                        {
                            validFileTwo = false;
                            loadAnlzTwoButton.setEnabled(true);
                            JOptionPane.showMessageDialog(this,
                                    "Os factores de escala sao diferentes",
                                    "Erro de ficheiro",
                                    JOptionPane.ERROR_MESSAGE);
                        }
                    }
                    else
                    {
                        validFileTwo = false;
                        loadAnlzTwoButton.setEnabled(true);
                        JOptionPane.showMessageDialog(this,
                                    "As dimensoes sao diferentes",
                                    "Erro de ficheiro",
                                    JOptionPane.ERROR_MESSAGE);
                    }
                }
                else
                {
                    validFileTwo = false;
                    loadAnlzTwoButton.setEnabled(true);
                    JOptionPane.showMessageDialog(this,
                                    "As dimensoes sao diferentes",
                                    "Erro de ficheiro",
                                    JOptionPane.ERROR_MESSAGE);
                }
            }
            else
            {
                validFileTwo = true;
            }

            if((validFileOne)&&(validFileTwo))
            {
                viewButton.setEnabled(true);
            }
        }
        catch(Exception e)
        {
            if(e.getMessage().equals("not hdr"))
            {
                JOptionPane.showMessageDialog(this,
```

```
                                    "Nao escolheu um *.hdr",
                                    "Erro de formato",
                                    JOptionPane.ERROR_MESSAGE);
            }
            else
            {
                JOptionPane.showMessageDialog(this,
                                    "O header nao tem 348 bytes",
                                    "Erro de formato",
                                    JOptionPane.ERROR_MESSAGE);
            }
            loadAnlzOneButton.setEnabled(true);
        }

        this.setVisible(true);
    }
    if(clickedButton == viewButton)
    {
        new ViewerWindowComparer(probArrayOne,
                                    probArrayTwo,
                                    4,
                                    "Comparador",
                                    this);
        viewButton.setEnabled(false);
        loadAnlzOneButton.setEnabled(true);
        loadAnlzTwoButton.setEnabled(true);
    }
}

private void createGUI()
{
    mainContainer = getContentPane();

    loadFilesPanel = new JPanel(new GridLayout(1, 3));

    loadAnlzOneButton = new JButton("Anlz 4D 1");
    loadAnlzTwoButton = new JButton("Anlz 4D 2");
    viewButton = new JButton("Ver");

    loadAnlzOneButton.setEnabled(true);
    loadAnlzTwoButton.setEnabled(true);
    viewButton.setEnabled(false);

    loadAnlzOneButton.addActionListener(this);
    loadAnlzTwoButton.addActionListener(this);
    viewButton.addActionListener(this);

    loadFilesPanel.add(loadAnlzOneButton);
    loadFilesPanel.add(loadAnlzTwoButton);
    loadFilesPanel.add(viewButton);

    mainContainer.add(loadFilesPanel);

    setSize(mainWindowWidth, mainWindowHeight);
    setLocation(mainWindowXPosition, mainWindowYPosition);
    setTitle("Comparador");
    setVisible(true);
}

private boolean areEqual(short[] dataArrayOne, short[] dataArrayTwo)
{
    int dataSize = dataArrayOne.length;
    int i;
    boolean equal = true;

    for(i=0; i<dataSize; i++)
    {
        if(dataArrayOne[i] != dataArrayTwo[i])
        {
            equal = false;
        }
```

```
        }

        return equal;
    }

    private boolean areEqual(float[] dataArrayOne, float[] dataArrayTwo)
    {
        int dataSize = dataArrayOne.length;
        int i;
        boolean equal = true;

        for(i=0; i<dataSize; i++)
        {
            if(dataArrayOne[i] != dataArrayTwo[i])
            {
                equal = false;
            }
        }

        return equal;
    }
}
```

```
package windows.viewers;

//
//  ViewerWindowComparer.java
//
//
//  Created by Jose Gabriel Lira Gomes on 4/25/07.
//  Copyright 2005 __FDG__. All rights reserved.
//

import imageUtils.ImagePanel;

import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.JToggleButton;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import windowUtils.FormatWindow;
import windowUtils.RadioButtonsWindow;
import windows.MainWindow;
import windows.MenuedFrame;
import dirUtils.DirPathGetter;
import estatistica.descriptive.ThreeDArrayAnalyzer;
import fileUtils.WriteTextNumberFile;
import functional.data.Voxel;
import functional.export.JpegExporter;
import functional.results.LineProfiler;
import functional.results.ProbVolumesAnalyzer;
import graphUtils.GraphPanel;

public class ViewerWindowComparer extends MenuedFrame
    implements ChangeListener, MouseListener, MouseMotionListener, ItemListener
{
    //
    //  Data Members
    //

    private double[][][] imageDataOne;
    private double[][][] imageDataTwo;
    private ProbVolumesAnalyzer pVolsAnalyzer;
    private double[][][] imageDataDifference;
    private int zoomFactor;
    private String method;
    MainWindow parentWindow;

    private int width;
    private int height;
    private int numberOfSlices;
    private short[] imageDims;

    private int x;
    private int y;
    private int z;
```

```
    private double threshold;
    private double voxelVolume;
    private String overlVolumeText;
    private String overlPercText;

    private int arrayImgWidth;

    private int profilesPanelWidth;
    private int profilePanelsHeight;

    private int dataPanelHeight;

    private int windowWidth;
    private int windowHeight;
    private int windowXPosition;
    private int windowYPosition = 21;

    private double[] profileArrayOne;
    private double[] profileArrayTwo;

    private Voxel maxVoxel;

    private Container viewContainer;
    private GraphPanel profilePanel;
    private JPanel profileDataPanel;
    private JPanel profilesPanel;
    private ImagePanel imagePanel;
    private JPanel toolsPanel;
    private JPanel dataPanel;
    private JPanel zSliderPanel;
    private JPanel thresSliderPanel;
    private JPanel exportPanel;

    private boolean bckgVisible = false;
    private double[][][] bckgData;
    private double[][] bckgSliceData;
    private double[][] imageSliceData;

    private JButton profileOneDataButton;
    private JButton profileTwoDataButton;
    private JSlider zSlider;
    private JSlider thresSlider;
    private JToggleButton bckgButton;
    private JLabel dataZValue;
    private JLabel dataThres;
    private JLabel dataVValue;
    private JButton jpegExportButton;

    private RadioButtonsWindow formatWindow;
    private JButton setFormatButton;

    private int iX;
    private int iY;
    private int iZ;

    private int fX;
    private int fY;
    private int fZ;

    private Voxel firstVoxel;
    private Voxel secndVoxel;

    private JLabel firstVoxelLabel;
    private JLabel secndVoxelLabel;

    private boolean lastChangedWasFirst = false;

    private LineProfiler profilerOne;
    private LineProfiler profilerTwo;
```

```
    private boolean isProfileOne;

    private int mouseX = 0;
    private int mouseY = 0;



    //
    //  Constructors
    //
    //      ViewerWindowComparer(double[][][] imageDataOne,
    //                           double[][][] imageDataTwo,
    //                           int zoomFactor,
    //                           String method,
    //                           MainWindow parentWindow)
    //

    public ViewerWindowComparer(double[][][] imageDataOne,
                                double[][][] imageDataTwo,
                                int zoomFactor,
                                String method,
                                MainWindow parentWindow)
    {
        this.imageDataOne = imageDataOne;
        this.imageDataTwo = imageDataTwo;
        this.pVolsAnalyzer =
                new ProbVolumesAnalyzer(imageDataOne, imageDataTwo);
        this.imageDataDifference = pVolsAnalyzer.getDifference();
        this.zoomFactor = zoomFactor;
        this.method = method;
        this.parentWindow = parentWindow;

        getImageDims();
        getInitialValues();

        maxVoxel = pVolsAnalyzer.getMaxOvlpVoxel();
        System.out.println("maxVoxel = (" + maxVoxel.getX() + ","
                                          + maxVoxel.getY() + ","
                                          + maxVoxel.getZ() + ")");

        createGUI();

        imagePanel.addMouseListener(this);
        imagePanel.addMouseMotionListener(this);
    }

    public ViewerWindowComparer(double[][][] imageDataOne,
                                double[][][] imageDataTwo,
                                double[][][] bckgData,
                                int zoomFactor,
                                String method,
                                MainWindow parentWindow)
    {
        this.imageDataOne = imageDataOne;
        this.imageDataTwo = imageDataTwo;
        this.pVolsAnalyzer =
                new ProbVolumesAnalyzer(imageDataOne, imageDataTwo);
        this.imageDataDifference = pVolsAnalyzer.getDifference();
        this.bckgData = bckgData;
        this.zoomFactor = zoomFactor;
        this.method = method;
        this.parentWindow = parentWindow;

        getImageDims();
        getInitialValues();

        maxVoxel = pVolsAnalyzer.getMaxOvlpVoxel();
        System.out.println("maxVoxel = (" + maxVoxel.getX() + ","
                                          + maxVoxel.getY() + ","
                                          + maxVoxel.getZ() + ")");
```

```
        bckgVisible = true;
        createGUI();

        imagePanel.addMouseListener(this);
        imagePanel.addMouseMotionListener(this);
    }

    public void disableBckgButton()
    {
        this.bckgButton.setEnabled(false);
    }

    /******* LISTEN TO SLIDER CHANGE EVENT *******/

    public void stateChanged(ChangeEvent event)
    {
        int intThres;

        String pValueText;
        String zPositionText;

        intThres = thresSlider.getValue();
        this.threshold = (double)intThres/100.0;
        pValueText =
                Double.toString(Math.round((1.0 - threshold)*100.0)/100.0);
        setVolumeInfo(threshold);

        this.z = zSlider.getValue();
        zPositionText = Integer.toString(z);

        dataThres.setText("p = " + pValueText);
        dataZValue.setText("Z = " + zPositionText);
        dataVValue.setText("V = " + overlVolumeText + " cm3 ("
                                  + overlPercText + ")");

        if(bckgVisible)
        {
            this.bckgSliceData = getSlice(bckgData, z);
            this.imagePanel.setBckgImageData(bckgSliceData);
            this.imageSliceData = getSlice(imageDataDifference, z);
            this.imagePanel.setImageDataAndThreshold(
                                  imageSliceData, threshold, 1);
        }
        else
        {
            this.imageSliceData = getSlice(imageDataDifference, z);
            this.imagePanel.setImageDataAndThreshold(
                                  imageSliceData, threshold, 0);
        }

    }

    public void itemStateChanged(ItemEvent event)
    {
        Object source = (Object) event.getSource();

        if(source == bckgButton)
        {
            if(event.getStateChange() == ItemEvent.SELECTED)
            {
                this.bckgVisible = true;

                this.bckgSliceData = getSlice(bckgData, z);
                this.imagePanel.setBckgImageData(bckgSliceData);
                this.imagePanel.showBackground();
            }
            else
            {
                this.bckgVisible = false;

                this.imagePanel.hideBackground();
```

```
        }
    }
}


/******* LISTEN TO BUTTON ACTION EVENT *******/

public void actionPerformed(ActionEvent event)
{
    //int intThres;
    DirPathGetter dirPathGetter;
    String dirPath;
    String path;

    JButton clickedButton = (JButton) event.getSource();

    //intThres = thresSlider.getValue();
    //this.threshold = (double)intThres/100;

    if(clickedButton == profileOneDataButton)
    {
        formatWindow = new FormatWindow();

        isProfileOne = true;

        setFormatButton = formatWindow.acceptButton;
        setFormatButton.addActionListener(this);

        parentWindow.setVisible(false);
        this.setVisible(false);
    }
    if(clickedButton == profileTwoDataButton)
    {
        formatWindow = new FormatWindow();

        isProfileOne = false;

        setFormatButton = formatWindow.acceptButton;
        setFormatButton.addActionListener(this);

        parentWindow.setVisible(false);
        this.setVisible(false);
    }
    if(clickedButton == setFormatButton)
    {
        formatWindow.setVisible(false);

        dirPathGetter = new DirPathGetter();
        dirPath = dirPathGetter.getPath();

        if(isProfileOne)
        {
            path = dirPath + "/PerfilUm("+Integer.toString(iX)+","
                                         +Integer.toString(iY)+","
                                         +Integer.toString(iZ)+")-("
                                         +Integer.toString(fX)+","
                                         +Integer.toString(fY)+","
                                         +Integer.toString(fZ)+").txt";

            new WriteTextNumberFile(profileArrayOne,
                                    path,
                                    formatWindow.getSelection());
        }
        else
        {
            path = dirPath + "/PerfilDois("+Integer.toString(iX)+","
                                           +Integer.toString(iY)+","
                                           +Integer.toString(iZ)+")-("
                                           +Integer.toString(fX)+","
                                           +Integer.toString(fY)+","
                                           +Integer.toString(fZ)+").txt";
```

```
            new WriteTextNumberFile(profileArrayTwo,
                                    path,
                                    formatWindow.getSelection());
        }
        this.setVisible(true);
        parentWindow.setVisible(true);
    }
    if(clickedButton == jpegExportButton)
    {
        dirPathGetter = new DirPathGetter();
        dirPath = dirPathGetter.getPath();
        if(bckgVisible)
        {
            new JpegExporter(imageDataDifference,
                             bckgData,
                             threshold,
                             4,
                             true,
                             dirPath,
                             method);
        }
        else
        {
            new JpegExporter(imageDataDifference,
                             threshold,
                             4,
                             true,
                             dirPath,
                             method);
        }
    }
}

public void paint(Graphics g)
{
    super.paint(g);
}


public void mouseClicked(MouseEvent me)
{
    mouseX = me.getX();
    mouseY = me.getY();

    if(me.isMetaDown())
    {

    }
    else
    {
        if(me.isAltDown())
        {
            x = (int)Math.floor(mouseX/zoomFactor);
            y = height - 1 - (int)Math.floor(mouseY/zoomFactor);
            if((x<width)&&(x>=0)&&(y<height)&&(y>=0))
            {
                secndVoxel = new Voxel(x, y, z);
                lastChangedWasFirst = false;

                iX = firstVoxel.getX();
                iY = firstVoxel.getY();
                iZ = firstVoxel.getZ();

                fX = secndVoxel.getX();
                fY = secndVoxel.getY();
                fZ = secndVoxel.getZ();

                getProfiles();

                setAll();
```

```
            }
        }
        else
        {
            if(me.isControlDown())
            {
                x = (int)Math.floor(mouseX/zoomFactor);
                y = height - 1 - (int)Math.floor(mouseY/zoomFactor);
                if((x<width)&&(x>=0)&&(y<height)&&(y>=0))
                {
                    firstVoxel = new Voxel(x, y, z);
                    lastChangedWasFirst = true;

                    iX = firstVoxel.getX();
                    iY = firstVoxel.getY();
                    iZ = firstVoxel.getZ();

                    fX = secndVoxel.getX();
                    fY = secndVoxel.getY();
                    fZ = secndVoxel.getZ();

                    getProfiles();

                    setAll();
                }
            }
            else
            {
                x = (int)Math.floor(mouseX/zoomFactor);
                y = height - 1 - (int)Math.floor(mouseY/zoomFactor);
                if((x<width)&&(x>=0)&&(y<height)&&(y>=0))
                {
                    if(lastChangedWasFirst)
                    {
                        secndVoxel = new Voxel(x, y, z);
                        lastChangedWasFirst = false;
                    }
                    else
                    {
                        firstVoxel = new Voxel(x, y, z);
                        lastChangedWasFirst = true;
                    }

                    iX = firstVoxel.getX();
                    iY = firstVoxel.getY();
                    iZ = firstVoxel.getZ();

                    fX = secndVoxel.getX();
                    fY = secndVoxel.getY();
                    fZ = secndVoxel.getZ();

                    //System.out.println(iX+","+iY+","+iZ);
                    //System.out.println(fX+","+fY+","+fZ);

                    getProfiles();

                    setAll();
                }
            }
        }
    }
}


public void mouseEntered(MouseEvent me)
{

}


public void mouseExited(MouseEvent me)
```

```
{

}


public void mousePressed(MouseEvent me)
{

}


public void mouseReleased(MouseEvent me)
{

}


public void mouseDragged(MouseEvent me)
{

}


public void mouseMoved(MouseEvent me)
{

}

private void getImageDims()
{
    this.width = imageDataOne.length;
    this.height = imageDataOne[0].length;
    this.numberOfSlices = imageDataOne[0][0].length;

    imageDims = new short[8];
    imageDims[0] = 4;
    imageDims[1] = (short)width;
    imageDims[2] = (short)height;
    imageDims[3] = (short)numberOfSlices;
    imageDims[4] = 1;

    voxelVolume = parentWindow.getVoxelVolume();
}


private void getInitialValues()
{
    int xInitialValue;
    int yInitialValue;
    int zInitialValue;

    double initialThres;

    xInitialValue = (int)Math.round(width/2);
    yInitialValue = (int)Math.round(height/2);
    zInitialValue = (int)Math.round(numberOfSlices/2);

    initialThres = 0.5;

    this.x = xInitialValue;
    this.y = yInitialValue;
    this.z = zInitialValue;

    this.threshold = initialThres;
    setVolumeInfo(threshold);
}


private void createGUI()
{
    arrayImgWidth = 2*width;
```

```
        profilesPanelWidth = arrayImgWidth;
        profilePanelsHeight = 200;
        dataPanelHeight = 42;

        windowXPosition = parentWindow.getWidth() + 1;
        windowWidth = 26 + Math.max(profilesPanelWidth, zoomFactor*width+79);
        windowHeight = 200 + dataPanelHeight + zoomFactor*height + 85 + 80 + 20;

        firstVoxel = new Voxel(x, 0, z);
        secndVoxel = new Voxel(x, height-1, z);

        iX = firstVoxel.getX();
        iY = firstVoxel.getY();
        iZ = firstVoxel.getZ();

        fX = secndVoxel.getX();
        fY = secndVoxel.getY();
        fZ = secndVoxel.getZ();

        getProfiles();

        createProfilesPanel();
        createZSliderPanel();
        createImagePanel();
        createThresSliderPanel();
        createToolsPanel();
        createDataPanel();
        createExportPanel();

        addPanelsToContainer();

        setSize(windowWidth, windowHeight);
        setLocation(windowXPosition, windowYPosition);
        setTitle(method);
        setVisible(true);
    }


    private void createProfilesPanel()
    {
        createProfilePanel();
        createProfileDataPanel();

        profilesPanel = new JPanel(new GridBagLayout());
        profilesPanel.setPreferredSize(new Dimension(profilesPanelWidth,
                                                     profilePanelsHeight+42));

        GridBagConstraints c = new GridBagConstraints();


        c.fill = GridBagConstraints.BOTH;

        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        profilesPanel.add(profilePanel, c);

        c.gridx = 1;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0;
        c.weighty = 0;

        profilesPanel.add(profileDataPanel, c);
    }
```

```
private void createProfilePanel()
{
    profilePanel = new GraphPanel(profileArrayOne,
                                  2*arrayImgWidth,
                                  profilePanelsHeight);
    profilePanel.addSingleArrayOver(profileArrayTwo);
}

private void createProfileDataPanel()
{
    profileDataPanel = new JPanel(new GridLayout(4,1));
    profileDataPanel.setPreferredSize(
            new Dimension(70, profilePanelsHeight));

    firstVoxelLabel = new JLabel();
    secndVoxelLabel = new JLabel();

    profileOneDataButton = new JButton("D 1");
    profileOneDataButton.addActionListener(this);
    profileTwoDataButton = new JButton("D 2");
    profileTwoDataButton.addActionListener(this);

    setProfilesDataLabels();

    profileDataPanel.add(firstVoxelLabel);
    profileDataPanel.add(secndVoxelLabel);
    profileDataPanel.add(profileOneDataButton);
    profileDataPanel.add(profileTwoDataButton);
}

private void createZSliderPanel()
{
    zSliderPanel = new JPanel(new BorderLayout());
    zSliderPanel.setBorder(BorderFactory.createTitledBorder("Z"));
    zSlider = createZSlider(z);
    zSliderPanel.add(zSlider, BorderLayout.CENTER);
}


private void createImagePanel()
{
    double imageMax = 1.0;
    double imageMin = 0.0;

    double bckgMax;
    double bckgMin;

    imageSliceData = getSlice(imageDataDifference, z);

    if(bckgVisible)
    {
        ThreeDArrayAnalyzer bckgDataAnalyzer =
                new ThreeDArrayAnalyzer(bckgData);
        bckgMax = bckgDataAnalyzer.getMaximum();
        bckgMin = bckgDataAnalyzer.getMinimum();

        bckgSliceData = getSlice(bckgData, z);
        imagePanel = new ImagePanel(bckgSliceData,
                                    imageSliceData,
                                    zoomFactor,
                                    threshold,
                                    bckgMax,
                                    bckgMin,
                                    imageMax,
                                    imageMin,
                                    false,
                                    true);
    }
    else
    {
```

```
            imagePanel = new ImagePanel(imageSliceData,
                                        zoomFactor,
                                        threshold,
                                        imageMax,
                                        imageMin,
                                        true);
        }
    }


    private void createThresSliderPanel()
    {
        thresSliderPanel = new JPanel(new BorderLayout());
        thresSliderPanel.setBorder(
                BorderFactory.createTitledBorder("Threshold"));
        thresSlider = createThresSlider((int)(threshold*100));
        thresSliderPanel.add(thresSlider, BorderLayout.CENTER);
    }

    private void createToolsPanel()
    {
        toolsPanel = new JPanel(new GridLayout(2,1));
        toolsPanel.setPreferredSize(new Dimension(79,79));
        bckgButton = new JToggleButton("Fundo");
        bckgButton.setSelected(true);
        bckgButton.addItemListener(this);
        //toolsPanel.add(bckgButton);
    }

    private void createDataPanel()
    {
        dataZValue = new JLabel("Z = " + z);
        dataThres = new JLabel("p = 0.5");
        dataVValue = new JLabel("V = " + overlVolumeText + " cm3 ("
                                       + overlPercText + ")");

        dataPanel = new JPanel(new GridBagLayout());
        dataPanel.setMinimumSize(new Dimension(100,70));
        dataPanel.setBorder(BorderFactory.createTitledBorder("Dados"));

        GridBagConstraints c = new GridBagConstraints();

        c.fill = GridBagConstraints.BOTH;

        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0.5;
        c.weighty = 0.4;

        dataPanel.add(dataZValue, c);

        c.gridx = 1;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.weightx = 0.5;
        c.weighty = 0.4;

        dataPanel.add(dataThres, c);

        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 2;
        c.gridheight = 1;
        c.weightx = 1;
        c.weighty = 0.6;

        dataPanel.add(dataVValue, c);
    }
```

```
private void createExportPanel()
{
    exportPanel = new JPanel(new GridLayout(2,1));
    exportPanel.setPreferredSize(new Dimension(79,79));
    jpegExportButton = new JButton("JPEG");
    exportPanel.add(jpegExportButton);
    jpegExportButton.addActionListener(this);
}

private void addPanelsToContainer()
{
    viewContainer = getContentPane();
    viewContainer.setLayout(new GridBagLayout());
    GridBagConstraints c = new GridBagConstraints();

    c.fill = GridBagConstraints.BOTH;

    c.gridx = 0;
    c.gridy = 0;
    c.gridwidth = 3;
    c.gridheight = 1;
    c.weightx = 0;
    c.weighty = 0;

    viewContainer.add(profilesPanel, c);

    c.gridx = 0;
    c.gridy = 1;
    c.gridwidth = 1;
    c.gridheight = 1;
    c.weightx = 0;
    c.weighty = 0;

    viewContainer.add(zSliderPanel, c);

    c.gridx = 1;
    c.gridy = 1;
    c.gridwidth = 2;
    c.gridheight = 1;
    c.weightx = width*zoomFactor;
    c.weighty = height*zoomFactor;

    viewContainer.add(imagePanel, c);

    c.gridx = 0;
    c.gridy = 2;
    c.gridwidth = 1;
    c.gridheight = 1;
    c.weightx = 0;
    c.weighty = 0;

    viewContainer.add(toolsPanel, c);

    c.gridx = 1;
    c.gridy = 2;
    c.gridwidth = 2;
    c.gridheight = 1;
    c.weightx = 1;
    c.weighty = 0;

    viewContainer.add(thresSliderPanel, c);

    c.gridx = 0;
    c.gridy = 3;
    c.gridwidth = 1;
    c.gridheight = 1;
    c.weightx = 0;
    c.weighty = 0;
```

```
        viewContainer.add(exportPanel, c);

        c.gridx = 1;
        c.gridy = 3;
        c.gridwidth = 2;
        c.gridheight = 1;
        c.weightx = 1;
        c.weighty = 0;

        viewContainer.add(dataPanel, c);
    }

    private void setProfilesDataLabels()
    {
        firstVoxelLabel.setText("("+iX+","+iY+","+iZ+")");
        secndVoxelLabel.setText("("+fX+","+fY+","+fZ+")");
    }


    private JSlider createThresSlider(int initialValue)
    {
        JSlider thresSlider = new JSlider();

        thresSlider.setOrientation(JSlider.HORIZONTAL);
        thresSlider.setPaintLabels(true);
        thresSlider.setPaintTicks(true);
        thresSlider.setMinimum(0);
        thresSlider.setMaximum(100);
        thresSlider.setValue(initialValue);
        thresSlider.setMajorTickSpacing(25);
        thresSlider.setMinorTickSpacing(5);

        thresSlider.addChangeListener(this);

        return thresSlider;
    }


    private JSlider createZSlider(int initialValue)
    {
        JSlider zSlider = new JSlider();

        zSlider.setOrientation(JSlider.VERTICAL);
        zSlider.setPaintLabels(true);
        zSlider.setPaintTicks(true);
        zSlider.setMinimum(0);
        zSlider.setMaximum(numberOfSlices-1);
        zSlider.setValue(initialValue);
        zSlider.setMajorTickSpacing(10);
        zSlider.setMinorTickSpacing(5);

        zSlider.addChangeListener(this);

        return zSlider;
    }

    private void getProfiles()
    {
        profilerOne = new LineProfiler(imageDataOne, firstVoxel, secndVoxel);
        profileArrayOne = profilerOne.getProfile();
        profilerTwo = new LineProfiler(imageDataTwo, firstVoxel, secndVoxel);
        profileArrayTwo = profilerTwo.getProfile();
    }

    private double[][] getSlice(double[][][] volumeDataArray, int z)
    {
        int x, y;
        double[][] sliceDataArray = new double[width][height];

        for(y=0; y<height; y++)
        {
```

```
                for(x=0; x<width; x++)
                {
                    sliceDataArray[x][y] = volumeDataArray[x][y][z];
                }
            }

            return sliceDataArray;
        }

        private void setAll()
        {
            setProfilesImages();

            setProfilesDataLabels();
        }

        private void setProfilesImages()
        {
            profilePanel.resetTwoArrays(profileArrayOne, profileArrayTwo);
        }

        private void setVolumeInfo(double threshold)
        {
            double numberOfVoxels;
            double overlVolume;
            double firstOverlPerc;
            double secndOverlPerc;

            numberOfVoxels = pVolsAnalyzer.getNumberOfOverlVoxels(threshold);
            overlVolume = numberOfVoxels*voxelVolume;
            this.overlVolumeText =
                    Double.toString(Math.round(overlVolume*10.0)/10.0);

            firstOverlPerc = pVolsAnalyzer.getOverlapPerc(threshold, true);
            secndOverlPerc = pVolsAnalyzer.getOverlapPerc(threshold, false);
            this.overlPercText = Double.toString(
                Math.round(firstOverlPerc*1000.0)/10.0) + " %.V1; "
                + Double.toString(Math.round(secndOverlPerc*1000.0)/10.0)
                + " %.V2";
        }
}
```