DM

Um Novo Pacote no R para Visualizar Dados em Análise de Sobrevivência

DISSERTAÇÃO DE MESTRADO

Pedro Daniel Ponte Camacho

MESTRADO EM MATEMÁTICA, ESTATÍSTICA E APLICAÇÕES

UNIVERSIDADE da MADEIRA

A Nossa Universidade

www.uma.pt

dezembro | 2020

Um Novo Pacote no R para Visualizar Dados em Análise de Sobrevivência

DISSERTAÇÃO DE MESTRADO

Pedro Daniel Ponte Camacho
MESTRADO EM MATEMÁTICA, ESTATÍSTICA E APLICAÇÕES

ORIENTAÇÃO
Ana Maria Cortesão Pais Figueira da Silva Abreu

Um Novo Pacote no R para Visualizar Dados em Análise de Sobrevivência

DISSERTAÇÃO DE MESTRADO

Pedro Daniel Ponte Camacho

MESTRADO EM MATEMÁTICA, ESTATÍSTICA E APLICAÇÕES

JÚRI

Maribel Gomes Gonçalves Gordon

Cristina Maria Tristão Simões Rocha

Ana Maria Cortesão Pais Figueira da Silva Abreu

Agradecimentos

Palavras pessoais nunca foram o meu forte, mas tenho muitos para agradecer por me ter levado a este ponto e ter, em alguma maneira, me permitido ter a força e convicção para realizar este trabalho.

À minha orientadora, que para além de fornecer um tema ideal para mim e interessante, que sem o seu suporte e motivação eu tenho a certeza que nunca teria feito mais do que começar, e por me auxiliar nas áreas que eu tinha menor experiência.

À comunidade por detrás do CRAN e do desenvolvimento do R, pelas ferramentas e documentação oferecidas e constantemente iteradas. Um dos pilares do código aberto, um que espero que a minha contribuição minúscula seja de uso também.

Aos meus professores, que assumo não deve ter sido fácil ensinar a uma turma composta por tantos estudantes de tantas outras licenciaturas, minha incluída. Fizeram aprender as intrincarias do campo da Matemática recompensador.

Aos meus colegas, tanto aos do curso de Matemática, tanto aos de Informática, especialmente a Carina, Tatiana e o Yuri. Espero que estejam a fazer bem aí.

À minha família, pela sua paciência e suporte desde sempre. Obrigado.

Finally, but not least, to my dearest friends and tabletop companions, Blue, Lee, and Zach, for being an ongoing respite to a disorientating reality, and for allowing me to take part in exploring your worlds during many hijinks and emotional moments. I always feel welcome and safe in your midst. Another special thanks for Lee, once more, for all his academic know-how and support as we both fought to finish our dissertations this year.

Resumo

Existe uma plenitude de funcionalidades gráficas presentes no ecossistema de pacotes da linguagem de programação R para a área da Estatística e no ramo da Análise de Sobrevivência, de tal modo que um utilizador pode não saber onde começar tendo presente uma estrutura de dados ou um modelo específico. Neste trabalho detalha-se a funcionalidade de um novo pacote, `vsd`, desenvolvido no R para visualizar dados em Análise de Sobrevivência para suavizar esta problemática.

Começa-se com uma introdução ao processo de criação de pacotes para o R, com foco na criação do esqueleto de um pacote através do pacote `devtools`, na documentação de um pacote e consequente declaração de funcionalidades através do pacote `roxygen2`, nalgumas componentes extras como o uso de *linting* para correção do código e a publicação do pacote, e o uso do sistema de controlo de versões do código Git para fácil publicação do pacote sob licença aberta, como requerido pelo CRAN.

Em seguida, faz-se uma breve abordagem a conceitos em Análise de Sobrevivência relevantes para o pacote `vsd`.

O pacote `vsd`, que aproveita funcionalidades doutros pacotes já presentes no ecossistema do R como os pacotes `survminer` e `flexsurv`, permite visualizar vários gráficos para dados contendo censura à direita como, por exemplo: da estimativa de Kaplan-Meier da função de sobrevivência, da estimativa suavizada da função de risco, dos resíduos padronizados e dos gráficos de floresta associados aos coeficientes do modelo de Cox, bem como das estimativas de modelos paramétricos. O pacote `vsd` foi construído com um *design* modular para permitir funcionalidade mais extensiva no futuro.

Os gráficos criados pelo pacote `vsd` podem ser explorados interativamente, através do modo interativo com recurso opcional ao pacote `plotly`, ou após geração modificados pelo utilizador, devido ao uso do pacote gráfico `ggplot2`, que permite reutilização e customização fácil das estruturas gráficas criadas.

Palavras Chave— Análise de Sobrevivência, `ggplot2`, pacotes do R.

Abstract

There's a plethora of graphical packages on the R language ecosystem, specifically in the area of Statistics and within the field of Survival Analysis, so much so that an user might not know where to start when considering a specific model or data structure. This dissertation details the functionality of a newly developed R package, vsd, meant to visualize data in Survival Analysis to ease this problem.

It starts with an introduction on how to create R packages, with a focus on: building a package's skeleton through the devtools package, in its documentation and consequent added functionality though the roxygen2 package, in some extra developmental concepts like linting for code correctness and package publication, and finally in how to use the version control system Git for easy availability of the package under an open license, as required by CRAN.

Afterwards the discussion moves to a brief overview of any relevant concepts under Survival Analysis for the developed package vsd.

The vsd package, taking use of other packages on R's ecosystem such as the packages survminer and flexsurv, allows the user to visualize several plots for right-censored data, as per example: the Kaplan-Meier estimate of the survival function, the smoothed estimate of the hazard function, the padronized residuals and forest plots associated to the coefficients of a Cox model, as well as the survival function estimates for parametric models. The vsd package has a modular design to allow adding more extensive functionality in the future.

The plots created by the vsd package can be explored interactively, through an interactive mode with optional aid of the plotly package, or customized after creation by the user thanks to the usage of the ggplot2 graphical package, that allows for the easy re-utilization and customization of all created plots.

Keywords— `ggplot2`, R packages, Survival Analysis.

Índice

Lista de Figuras **ix**

Lista de Blocos de Código **xiii**

1 Como construir um pacote no R **1**
1.1 O esqueleto do pacote através do `devtools` 2
1.1.1 Licenças de código aberto 3
1.1.2 A diretoria `R/` 4
1.1.3 O ficheiro `DESCRIPTION` 6
1.1.4 O ficheiro `NAMESPACE` 8
1.2 Documentação com `roxygen2` 8
1.2.1 Documentação para o pacote 12
1.3 Extras no desenvolvimento de um pacote 13
1.3.1 *Linting* e formatação 13
1.3.2 *Vignettes* 15
1.3.3 Ficheiros de dados 15
1.3.4 Testes 16
1.3.5 Publicação no CRAN 18
1.4 Controlo de revisões com `git` 19
1.4.1 Plataforma *host* 20
1.4.2 `git init` 22
1.4.3 `git pull` 23
1.4.4 `git add` 24
1.4.5 `git commit` 25
1.4.6 `git push` 26
1.4.7 Sumário do processo Git 26

2 Algumas noções sobre Análise de Sobrevivência **29**
2.1 Censura 30
2.2 Caracterização do tempo de vida 31
2.3 Modelação paramétrica 33

2.4 Modelação não paramétrica 36
2.4.1 Estimador da função de risco 41
2.5 Modelo de regressão de Cox 43
2.5.1 Estudo dos resíduos do modelo de Cox 47

3 Um novo pacote no R **51**
3.1 Sintaxe da função `vsd` 52
3.2 Gráficos personalizáveis com o `ggplot2` 54
3.3 Gráficos da estimativa de Kaplan-Meier da função de sobrevivência 58
3.4 Gráficos específicos ao modelo Cox 59
3.5 Função de risco estimada e `muhaz` 63
3.6 Modelos paramétricos 65
3.7 Modo interativo e `plotly` 68

4 Considerações finais **73**

A Ficheiros de código do pacote `vsd` **77**

Bibliografia **97**

Lista de Figuras

1.1 Documentação da função exemplo `coolSum`, automaticamente gerada pelo pacote `roxygen2` através do comentário redigido. . 11

1.2 *Screenshot* da página de criação do repositório no Github, com as opções recomendadas já preenchidas. 21

2.1 Estimativa de Kaplan-Meier da função de sobrevivência e estimativa da função de sobrevivência obtida através do modelo Weibull ($\alpha = 1.108$, $\lambda^{-1} = 1225.419$) dos dados do ficheiro `ovarian` do pacote `survival`. 36

2.2 Estimativa de Kaplan-Meier da função de sobrevivência (e respetivo intervalo de confiança) correspondente ao dados do ficheiro `ovarian` no pacote `survival`. 39

2.3 Estimativas de Kaplan-Meier da função de sobrevivência para as categorias de `x`, do ficheiro de dados `aml` do pacote `survival`. 40

2.4 Estimativa de Kaplan-Meier da função de sobrevivência, juntamente com o seu intervalo de confiança para $\alpha = 0.05$, quando `x = Nonmaintained`, no ficheiro de dados `aml` do pacote `survival` 41

2.5 Estimativa da função de risco suavizada para os dados do ficheiro `ovarian` do pacote `survival`. 42

2.6 Estimativa da função de sobrevivência obtida através do modelo de Cox, obtido do ficheiro de dados `lung` do pacote `survival`. 47

2.7 Gráficos dos resíduos de Schoenfeld padronizados, obtidos do modelo de Cox criado do ficheiro de dados `lung` no pacote `survival`. 48

3.1 Gráfico obtido pelo pacote gráfico `graphics` para as estimativas de Kaplan-Meier da função de sobrevivência para as categorias de `x`, do ficheiro de dados `aml` do pacote `survival`. . . . 56

3.2 Gráfico obtido pelo pacote gráfico `ggplot2`, após pré-processamento da estrutura de dados, para as estimativas de Kaplan-Meier da função de sobrevivência para as categorias de `x`, do ficheiro de dados `aml` no pacote `survival`. 56
3.3 Gráfico obtido após adicionar componentes ao presente na Figura 3.2. 58
3.4 Gráfico da estimativa de Kaplan-Meier da função de sobrevivência do modelo obtido do ficheiro de dados `ovarian` do pacote `survival`. 59
3.5 Gráficos da estimativa de Kaplan-Meier da função de sobrevivência do modelo obtido do ficheiro de dados `ovarian` do pacote `survival` estratificado pela covariável `rx`, sem (à esquerda) e com (à direita) os intervalos de confiança. 60
3.6 Gráficos dos resíduos de Schoenfeld padronizados do modelo Cox, por covariável, obtido do ficheiro de dados `lung` do pacote `survival` através da função `vsd`. 61
3.7 Gráfico de floresta das estimativas dos coeficientes para o modelo de Cox baseado no ficheiro de dados `colon` do pacote `survival`. 62
3.8 Os quatro gráficos criados pela função `vsd` quando a covariável `strata(rx)` é usada no modelo de Cox criado no ficheiro de dados `colon` do pacote `survival`: para os dados na globalidade e para cada um dos três níveis da variável (Obs, Lev, Lev+5FU). 63
3.9 Gráfico da estimativas das funções de risco para o modelo de Cox para o ficheiro de dados `colon` do pacote `survival`, estratificado pela variável `rx`. 64
3.10 Gráfico da estimativa da função de risco do ficheiro de dados `lung` no pacote `survival`, delineada pela covariável `ph.ecog`. 65
3.11 Estimativa de Kaplan-Meier da função de sobrevivência e estimativa da função de sobrevivência obtida através do modelo Weibull dos dados do ficheiro `ovarian` do pacote `survival` pela função `vsd`. 66
3.12 Gráfico criado pelo pacote `survminer` no ficheiro de dados `bc` do pacote `flexsurvreg` para a estimativa do modelo Gengamma mais adequado. 67
3.13 Gráfico criado pelo `vsd` no ficheiro de dados `bc` do pacote `flexsurvreg` para a estimativa do modelo Gengamma mais adequado. 67

3.14 O gráfico para os resíduos da covariável `adhere` no modelo de Cox estimado do ficheiro de dados `colon`, obtido do modo interativo da função `vsd` sobre uma visualização interativa oferecido pelo pacote `plotly`. 70
3.15 Uso interativo do pacote `plotly` na Figura 3.14. 70
3.16 Gráfico equivalente ao da Figura 3.13, mas com um tema base do `ggplot2` inserido posteriormente após o modo interativo da função `vsd`. 71

Lista de Blocos de Código

A.1 Ficheiro `DESCRIPTION` do pacote `vsd` 77
A.2 Ficheiro `LICENSE` do pacote `vsd` 78
A.3 Ficheiro `NAMESPACE` do pacote `vsd` 78
A.4 Ficheiro `R/options.R` do pacote `vsd` 78
A.5 Ficheiro `R/plot_forest.R` do pacote `vsd` 81
A.6 Ficheiro `R/plot_hazard.R` do pacote `vsd` 82
A.7 Ficheiro `R/plot_parametric.R` do pacote `vsd` 84
A.8 Ficheiro `R/util.R` do pacote `vsd` 85
A.9 Ficheiro `R/vsd.R` do pacote `vsd` 87
A.10 Ficheiro `R/vsd-package.R` do pacote `vsd` 95

Capítulo 1

Como construir um pacote no R

O R é uma linguagem de programação, e seu respetivo conjunto de *software*, compiladores, documentação e interface (consola interativa), com o foco em criar um *environment* para análise estatística computacional e criação de gráficos [19]. Para tal, este oferece uma gama ampla de processos estatísticos e gráficos, com cuidado extra para que nas ferramentas presentes, as opções predefinidas levem à criação de *outputs* com rigor académico e prontos para publicação [17].

O R está disponível como *Software* Livre, sob licença *GNU General Public*, onde qualquer utilizador pode verificar o código, compilar e até contribuir para o seu desenvolvimento. Este espírito de contribuição verifica-se em como o R divide as suas funcionalidades em módulos discretos, cada uma com um objetivo particular: os *packages*.

Criar *packages*, ou pacotes, no R é relativamente simples, embora requeira algum esforço para além de escrever o código pretendido. Para auxiliar na configuração, manutenção e documentação de um pacote, existem dois pacotes, que se recomenda utilizar: `devtools` [21] e `roxygen2` [11], o primeiro focado em quaisquer tarefas relacionadas com o desenvolvimento de pacotes e o segundo com a sua documentação. Estes serão usados frequentemente neste trabalho.

Para os instalar e ter acesso a estes numa instalação corrente do R, como estão publicados no CRAN [4], o repositório compreensivo de pacotes públicos do R, basta correr os seguintes comandos na consola do R:

```
> install.packages("devtools", "roxygen2")
```

Para o desenvolvimento do pacote `vsd`, foi também usado o editor RStudio devido à sua integração com as ferramentas de desenvolvimento e facilidade

de uso, como por exemplo em *debugging*. Porém, nesta secção ir-se-á explicar como criar um pacote do R sem auxílio deste *software*, para universalidade destes conteúdos [25].

1.1 O esqueleto do pacote através do `devtools`

Neste documento, foi assumido que o pacote está a ser desenvolvido no Windows, na localização `C:/Repos/pacote`. O R e o `devtools`, em geral, são agnósticos em relação ao sistema operativo e podem ser usados sem problemas no Mac OS X ou em muitos plataformas UNIX ou similares (FreeBSD, Linux, ...).

Para começar a desenvolver um pacote, usa-se uma função no `devtools` que, quando corrida, estabelece os ficheiros mínimos necessários para que o R possa reconhecer que o seu código está num pacote, e como o correr. Para isso, dever-se-á carregar o pacote para o seu *workspace* primeiro, e então correr a função `wizard`, `devtools::create`:

```
> library(devtools)
> dir.create("C:/Repos/")
> devtools::create("C:/Repos/pacote")
```

Embora não seja necessário, a sintaxe `devtools::create` executa a função `create` do pacote `devtools`, sem haver necessidade que esta seja importada com `library()`[1] para o ambiente corrente. Isto é importante no código dentro dos pacotes, daí já se usar esta sintaxe.

Este fará um conjunto de operações, que se pode verificar pelo *output* da função, como se exemplifica em seguida:

```
✓ Creating 'C:/Repos/pacote'
✓ Setting active project to 'C:/Repos/pacote'
✓ Creating 'R/'
✓ Writing 'DESCRIPTION'
Package: pacote
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R (parsed):
* First Last <first.last@example.com> [aut, cre] (YOUR-ORCID-ID)
Description: What the package does (one paragraph).
License: `use_mit_license()`, `use_gpl3_license()` or friends to
pick a license
Encoding: UTF-8
LazyData: true
```

[1] library() também importa quaisquer dependências do pacote, como o `usethis` no caso do `devtools`, que é requerido para alguns dos comandos seguintes.

```
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.1.1
✓ Writing 'NAMESPACE'
✓ Changing working directory to 'C:/Repos/pacote'
```

1. Cria a diretoria `C:/Repos/pacote`, se esta ainda não existir (mas não cria as diretorias recursivamente, daí ser necessário usar um comando adicional para `C:Repos/`);

2. Cria uma subdiretoria `R/`, que é onde o código do pacote deve ser colocado;

3. Cria o ficheiro `DESCRIPTION`, um meta-ficheiro que descreve o projeto, cujos conteúdos iniciais são mostrados no *output*;

4. Cria o ficheiro `NAMESPACE`, outro meta-ficheiro que lida com o que o pacote importa e exporta (que será manipulado pelo `roxygen2` na Secção 1.2);

5. Define a diretoria ativa no R como `C:/Repos/pacote`, isto é, usa `setwd()` para que todos os comandos seguintes na sessão corram na pasta do projeto.

Embora um esqueleto deva ser o mínimo para ter um pacote funcional, existe um último passo crucial a fazer que precisa da decisão do utilizador: a escolha de uma licença de *software* para o pacote, preferencialmente uma licença de código aberto.

1.1.1 Licenças de código aberto

As licenças são fundamentais para estabelecer os direitos legais sob itens de autoria, neste caso o código redigido, definindo as possibilidades de partilha e direitos dos utilizadores, *developers* e programas de *software* que se baseiam neste código.

Por norma, na UE e nos EUA, se não estiver definida qualquer licença, o item fica sob *copyright* do(s) autor(es) original(is), não permitindo a partilha livre, manipulação ou manutenção por outrém [22]. Embora possa ser essa a intenção de alguns agentes, o espírito colaborativo do R implica que ninguém poderá manter ou submeter um pacote sem licença e, até em alguns casos, ser barrado de o usar devido à licença ulterior ou outras razões legais.

Recomenda-se então uma licença MIT (detalhes sobre esta e outras licenças podem ser encontradas *online* [12]) a qual, em geral, permite o uso,

re-uso e alteração livre do código (*copyleft*). Felizmente, o R, através do pacote `usethis`, importado automaticamente pelo `devtools`, vem com comandos delimitados para acrescentar licenças abertas ao seu código corrente com um simples comando. Para uma licença MIT, corra:

```
> use_mit_license("Nome Autor")
```

```
✓ Setting License field in DESCRIPTION to 'MIT + file LICENSE'
✓ Writing 'LICENSE.md'
✓ Adding '^LICENSE\.md$' to '.Rbuildignore'
✓ Writing 'LICENSE'
```

substituindo `Nome Autor` pelo seu nome legal, e a licença estará aplicada ao seu pacote.

São criados dois ficheiros de licença, `LICENSE` e `LICENSE.md`. O primeiro é usado pelo R para identificar a licença e os seus detalhes (ano da criação da licença, autor), e o último é um ficheiro Markdown com os conteúdos da licença em si.

```
YEAR: 2020
COPYRIGHT HOLDER: Nome Autor
```

LICENSE

```
# MIT License

Copyright (c) 2020 Nome Autor

Permission is hereby granted, free of charge, to any person
    obtaining a copy
```

LICENSE.md (primeiras cinco linhas)

Note-se que também foi criado o ficheiro `.Rbuildignore`. Este é um meta-ficheiro que inclui uma lista de ficheiros e diretorias que serão ignoradas na compilação do código do pacote, ou seja, quaisquer ficheiros que não sejam código (ou parte do esqueleto).

Finalmente, note-se que o ficheiro `DESCRIPTION` foi alterado com o comando, pois agora contém `License: MIT + file LICENSE`.

Agora já é possível importar o pacote para o R, embora não seja útil visto que ainda não tem qualquer código para executar.

1.1.2 A diretoria `R/`

Como referido anteriormente, é na diretoria `R/` onde se deve colocar o código do pacote, dentro de ficheiros de texto com a extensão `.R`, onde tanto o nome

como o número de ficheiros presentes é arbitrário. Por enquanto, para criar o ficheiro `main.R` é necessário o seguinte código:

```r
coolSum <- function(left, right = 0) {
  return(left + right)
}

coolFibbonaci <- function(n) {
  stop("Este ainda não funciona, desculpe!")
}
```

R/main.R

Não esquecer de guardar o ficheiro após o criar. Ao fazer:

```
> devtools::load_all()
```

todos os ficheiros presentes na diretoria `R/` são recarregados na sessão no R, permitindo correr o código como se tivesse carregado o pacote normalmente. Tente correr:

```
> coolSum(exp(1), pi)
```

```
[1] 5.859874
```

Agora, embora seja possível colocar qualquer código do R como parte de um pacote, espera-se que os pacotes sejam modulares e isolados, isto é, que estes não toquem nas definições e objetos do utilizador. Assim, nos ficheiros presentes dentro da diretoria `R/` só se pode ter declaração de funções e não quaisquer outros elementos do R, como a (re)definição de objetos de dados, ao contrário do que é habitual em ficheiros *script* do R. Os pacotes podem conter ficheiros de dados, mas deve existir uma diretoria reservada para os alocar (como irá ser detalhado na Secção 1.3.3).

Outra consideração a ter em mente, para assegurar que o pacote não tem *side effects*, é que um pacote não deverá ocupar o *namespace* do utilizador com nomes inesperados ou indesejados, que ocorre aquando o uso de funções como `library(package)` para ter acesso à funcionalidade de outros pacotes. Então, para contornar este problema, recomenda-se:

- Nunca usar `library(package)` ou `require(package)` para aceder (*attach*) a outros pacotes mas sim `package::function` descrita anteriormente para aceder a funções de outros pacotes; esta sintaxe também carrega o pacote de uma maneira segura, equivalente a `loadNamespace`;
- Alternativamente, usar `requireNamespace(..., quietly = T)` para verificar se um utilizador tem um pacote opcional presente;

- Correr `use_package("package")` para alterar o ficheiro `DESCRIPTION` para que o pacote desejado (`"package"`) seja listado como uma dependência para importar (e assegurar que esta está presente, ou é instalada juntamente com o pacote).

Como exemplo, imagine-se que se quer acrescentar a seguinte função ao ficheiro `main.R` que usaria alguma funcionalidade do pacote `ggplot2`:

```
coolGraph <- function() {
  ggplot(mpg, aes(x = displ, y = hwy, color = class)) + geom_
      point() + geom_smooth(se = FALSE)
}
```

Então deve-se primeiro acrescentar `ggplot2::` a todas as funções deste pacote, da forma que se indica:

```
9  coolGraph <- function() {
10   ggplot2::ggplot(mpg, aes(x = displ, y = hwy, color = class)) +
         ggplot2::geom_point() + ggplot2::geom_smooth(se = FALSE)
11 }
```

R/main.R

Em seguida, na consola R que foi usada para criar o pacote deve-se correr:

```
> use_package("ggplot2")
```

```
✓ Adding 'ggplot2' to Imports field in DESCRIPTION
• Refer to functions with `ggplot2::fun()`
```

Mais à frente ir-se-á ver uma maneira de comprovar se o pacote tem *side effects* ou se falta algum elemento recomendado (*linting* na Secção 1.3.1) mas, por enquanto, desde que só se defina funções nos ficheiros `.R` e se siga as recomendações acima, não haverá problemas imediatos.

1.1.3 O ficheiro `DESCRIPTION`

O esqueleto do meta-ficheiro `DESCRIPTION`, escrito no formato DFC (*Debian control format*), embora funcional, possui vários campos que têm de ser alterados para que a informação associada ao pacote esteja correta. Após as secções anteriores, este seria o ficheiro correspondente:

```
1  Package: package.create
2  Title: What the Package Does (One Line, Title Case)
3  Version: 0.0.0.9000
4  Authors@R:
5      person(given = "First",
6             family = "Last",
```

```
7               role = c("aut", "cre"),
8               email = "first.last@example.com",
9               comment = c(ORCID = "YOUR-ORCID-ID"))
10 Description: What the package does (one paragraph).
11 License: MIT + file LICENSE
12 Encoding: UTF-8
13 LazyData: true
14 Roxygen: list(markdown = TRUE)
15 RoxygenNote: 7.1.1
```

DESCRIPTION

O ficheiro é composto por uma série de campos, que são:

Package O nome do pacote, que deve ser válido como quaisquer outros nomes para objetos no R;

Title Título do pacote, que como indicado no ficheiro esqueleto, deverá ser só uma linha Em Letras Maiúsculas;

Version A versão do pacote, uma sequência de números separados por `.` (ou `-`) onde os números são incrementados após cada iteração do pacote, semelhante ao *Semantic Versioning* [16]; pacotes em desenvolvimento por norma usam `0.0.0.X` onde `X` é um número igual ou superior a `9000` mas é arbitrário;

Author@R Um bloco de código R que identifica o(s) autor(es) do pacote [10];

Description Uma descrição sucinta do projeto, que não pode ser maior do que um parágrafo, já que informação mais extensa deve ir para a documentação ou estar, por exemplo, num `README.md`;

License Referente ao ficheiro de licença do projeto, como automaticamente definido na Secção 1.1.1;

Imports Lista de pacotes para importar (e instalar, se em falta) juntamente com este pacote, como automaticamente adicionado na Secção 1.1.2;

Um ficheiro `DESCRIPTION` pode ter campos adicionais com nomes arbitrários para uso no R, como `Encoding` e `LazyData`, ou outros pacotes como `Roxygen` ou `RoxygenNote`. Os criados pelo esqueleto do ficheiro são suficientes, e recomenda-se deixá-los como estão e verificar a documentação para mais detalhes.

O ficheiro `DESCRIPTION` do pacote `vsd` (Ficheiro A.1), o cerne deste trabalho, possui alguns elementos adicionais não presentes no esqueleto, como o

`Collate` (diz ao R a ordem de importação dos ficheiros de código do pacote) ou `Suggests` (refere dependências do pacote que não são indispensáveis à sua execução, não sendo por norma instaladas com `install_packages()`).

1.1.4 O ficheiro `NAMESPACE`

Se o pacote for importado localmente para o R sem `devtools::load_all()`, nenhuma das funções estarão disponíveis para ser usadas. Isto deve-se ao facto de que o R, não sabendo que funções o pacote pretende disponibilizar externamente, não as carrega para o *namespace* do utilizador.

O meta-ficheiro `NAMESPACE` é onde se indica ao R o que importar (pacotes externos) e exportar (funções) num pacote, mas como indicado anteriormente e no cabeçalho do ficheiro criado pelo esqueleto do projeto:

```
1 # Generated by roxygen2: do not edit by hand
```

NAMESPACE

ou seja, não é para ser modificado pelo utilizador, se este estiver a usar `roxygen2` para obter a documentação, como explicado na Secção 1.2. Para ver um exemplo mais completo de um ficheiro NAMESPACE consulte o Bloco de Código A.3.

1.2 Documentação com `roxygen2`

Tal como noutras linguagens de computação, o utilizador final de qualquer código produzido não é o computador, mas um ser humano. Logo, não basta que o código corra (e faça o que é pretendido), mas que este seja legível, agora e para o futuro, tanto para o programador como para o seu utilizador (i.e., exemplos e documentação).

No R, para criar comentários, código ignorados pelo compilador, basta começar a linha com um `#`. Mas o pacote `roxygen2` permite dar funcionalidade adicional a comentários, usando-os para criar a documentação de funções e pacotes automaticamente e até definir o que o pacote exporta e importa.

Para tal, antes de uma função, usa-se um bloco de comentários que inicia com `#'` para indicar que este é um comentário especial para o pacote `roxygen2` interpretar depois. Este bloco deve seguir uma sintaxe específica, como documentada pelo pacote [11]. Segue-se um exemplo para a função `coolSum` no código apresentado na Secção 1.1.2:

```
1 #' Acrescenta dois números
2 #'
```

```
#' Acrescenta dois números, ou retorna o número único.
#'
#' Não há muito mais que se possa dizer! É uma função gira.
#'
#' @param left Um número.
#' @param right Um número, opcional.
#'
#' @return A soma dos dois números.
#' @export
#'
#' @examples
#' coolSum(-3,5)
#' coolSum(pi)
coolSum <- function(left, right = 0) {
```

R/main.R (primeira função com comentário)

O bloco é composto por um bloco de texto, seguido por uma lista de *tags* como `@param` e `@return`. O bloco de texto consiste em:

1. A primeira linha será o título da função, devendo ser uma linha única e sucinta, e não acabar com pontuação;

2. A segunda linha/parágrafo será a descrição da função, ou seja, o sumário que a função faz;

3. O terceiro e subsequentes parágrafos deverão acrescentar mais detalhe sobre como a função funciona.

De seguida vem uma lista de *tags*, marcadas pelo `@`, que denotam funcionalidades da função numa sintaxe que o `roxygen2` consegue interpretar. Neste exemplo, estas são:

@param Um parâmetro da função, seguido pelo seu nome (igual à definição da função) e por uma descrição sucinta;

@return Uma descrição do resultado da função;

@examples Código do R que exemplifica como a função corre.

Se este *layout* parecer familiar, é porque o `roxygen2` usa estas linhas e *tags* para automaticamente criar a documentação desta função, ou seja, o que se obteria com `?coolSum`. Se fizer:

```
> devtools::document()
```

```
Updating pacote documentation
Loading pacote
Writing NAMESPACE
Writing coolSum.Rd
```

o `roxygen2` (que é chamado pelo `devtools`) cria a pasta `man`, para onde vão a documentação do pacote e respetivas componentes como, por exemplo, a função pretendida no ficheiro `coolSum.Rd`.

Ficheiros `.Rd` são o que o R usa para gerar a sua documentação, tendo uma sintaxe similar à do LaTeX:

```
% Generated by roxygen2: do not edit by hand
% Please edit documentation in R/main.R
\name{coolSum}
\alias{coolSum}
\title{Acrescenta dois números}
\usage{
coolSum(left, right = 0)
}
\arguments{
\item{left}{Um número.}

\item{right}{Um número, opcional.}
}
\value{
A soma dos dois números.
}
\description{
Acrescenta dois números, ou retorna o número único.
}
\details{
Não há muito mais que se possa dizer! É uma função gira.
}
\examples{
coolSum(-3,5)
coolSum(pi)
}
```

man/coolSum.Rd

Então origina a documentação patente na Figura 1.1:

```
> ?coolSum
```

coolSum {pacote} R Documentation

Acrescenta dois números

Description

Acrescenta dois números, ou retorna o número único.

Usage

```
coolSum(left, right = 0)
```

Arguments

`left`	Um número.
`right`	Um número, opcional.

Details

Não há muito mais que se possa dizer! É uma função gira.

Value

A soma dos dois números.

Examples

```
coolSum(-3,5)
coolSum(pi)
```

[Package *pacote* version 0.0.0.9000 Index]

Figura 1.1: Documentação da função exemplo `coolSum`, automaticamente gerada pelo pacote `roxygen2` através do comentário redigido.

Mais, como se usou `@examples`, `example(coolSum)` ficou também definido, já que esta função primitiva usa a documentação diretamente. Porém, se tentar correr este código, é provável que falhe, já que `devtools::load_all` não carrega a biblioteca da maneira usual, levando a que `example()` não consiga encontrar o pacote. Como alternativa, pode-se usar:

```
> devtools::run_examples()
```

```
Updating pacote documentation
Loading pacote
Writing NAMESPACE
Writing NAMESPACE
Writing coolSum.Rd
Writing pacote.Rd
-- Running 2 example files ------------------------------ pacote
   --
Loading pacote
```

```
> ### Name: coolSum
> ### Title: Acrescenta dois números
> ### Aliases: coolSum
>
> ### ** Examples
>
> coolSum(-3,5)
[1] 2

> coolSum(pi)
[1] 3.141593
Loading pacote
```

O `roxygen2` também alterou o ficheiro `NAMESPACE`:

```
# Generated by roxygen2: do not edit by hand

export(coolSum)
```

NAMESPACE

indicando ao R, com a *tag* `@export`, que esta função deve ser exportada, ficando assim disponível aos utilizadores deste pacote. Funções não documentadas, ou sem `@export`, não estarão acessíveis fora do pacote, o que é ideal para funções internas ou sub-funções.

Finalmente, no `roxygen2` existe também a possibilidade de utilizar a *tag* `@importFrom package function` como alternativa a `package::function` dentro dessa função, para um pequeno aumento de *performance* e facilidade de leitura. No entanto, não se recomenda usar esta forma para toda a importação individual de uma função externa.

1.2.1 Documentação para o pacote

Para criar a documentação para o pacote em si, `help("pacote")`, ou definir *tags* no `roxygen2` que devem aplicar ao pacote por completo, como não existe qualquer objeto num pacote que refira ao pacote por completo, documenta-se NULL. Pode-se fazer isto em qualquer ficheiro de código, embora recomenda-se fazer num ficheiro dedicado, como por exemplo `pacote.R`.

```
#' pacote: Um pacote como exemplo
#'
#' Este pacote possui algumas funções como exemplo em como fazer
    um pacote no R.
#'
#' @section coolSum: soma fantástica
#' A função coolSum ...
```

```
#'
#' @docType package
#' @name pacote
NULL
```

pacote.R

onde `@section` é uma *tag* para indicar uma subsecção na documentação (e o seu título correspondente).

1.3 Extras no desenvolvimento de um pacote

1.3.1 *Linting* e formatação

Não basta apenas ter código documentado, este tem de ser legível e não conter erros comuns. Felizmente, existe o conceito de programas *lint*, que verificam se o código escrito numa língua específica de programação segue um conjunto de normas de sintaxe, tanto em legibilidade como em correção.

No `devtools`, pode-se correr `devtools::lint()`, que usa o pacote `lintr` (que, na falta deste, permite-o instalar):

```
> devtools::lint()
```

```
'lintr' >= 2.0.1 must be installed for this functionality.
Would you like to install it?

1: Yes
2: No
```

```
Linting pacote..
R/main.R:4:3: style: Trailing whitespace is superfluous.
#'
  ^
R/main.R:4:3: style: Trailing whitespace is superfluous.
#'
  ^
R/main.R:16:1: style: Variable and function name style should be snake_case.
coolSum <- function(left, right = 0) {
  ^~~~~~~
R/main.R:20:1: style: Variable and function name style should be snake_case.
coolFibbonaci <- function(n) {
  ^~~~~~~~~~~~~
R/main.R:24:1: style: Variable and function name style should be snake_case.
coolGraph <- function() {
  ^~~~~~~~~
R/main.R:25:1: style: Lines should not be more than 80 characters.
ggplot2::ggplot(mpg, aes(x = displ, y = hwy, color = class)) + ggplot2::geom_point() +
    ggplot2::geom_smooth(se = FALSE, span = 1)
^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
R/main.R:25:19: warning: no visible binding for global variable 'mpg'
ggplot2::ggplot(mpg, aes(x = displ, y = hwy, color = class)) + ggplot2::geom_point() +
    ggplot2::geom_smooth(se = FALSE, span = 1)
^~~
R/main.R:25:24: warning: no visible global function definition for 'aes'
ggplot2::ggplot(mpg, aes(x = displ, y = hwy, color = class)) + ggplot2::geom_point() +
    ggplot2::geom_smooth(se = FALSE, span = 1)
^~~
```

```
R/main.R:25:32: warning: no visible binding for global variable 'displ'
ggplot2::ggplot(mpg, aes(x = displ, y = hwy, color = class)) + ggplot2::geom_point() +
    ggplot2::geom_smooth(se = FALSE, span = 1)
                               ^~~~~
R/main.R:25:43: warning: no visible binding for global variable 'hwy'
ggplot2::ggplot(mpg, aes(x = displ, y = hwy, color = class)) + ggplot2::geom_point() +
    ggplot2::geom_smooth(se = FALSE, span = 1)
                                          ^~~
R/pacote.R:4:3: style: Trailing whitespace is superfluous.
#'
  ^
```

A função imprime uma lista dos problemas encontrados no projeto em geral, catalogando-os pela localização (ficheiro, número da linha, número do carácter), o tipo de problema (pode ser só um aviso ou até erro no código), o problema em si, a linha de código problemático e a localização específica do problema na linha de código.

Problemas de estilo podem ir de espaço em branco a mais, falta de linhas em branco, más convenções nos nomes das variáveis (é usual usar `este_estilo` em vez de `esteEstilo` para nome de variáveis), linhas demasiado compridas, entre outros. Este tipo de problemas não quebram a funcionalidade do pacote, mas facilitam a leitura e manutenção de pacotes no ecossistema CRAN.

Avisos, embora não necessariamente indicativos de algo errado no pacote, muitas vezes são sinais de algo que poderá causar problemas na execução do pacote, como a redefinição do *workspace* do utilizador (uso erróneo de variáveis globais).

Neste exemplo, o primeiro aviso é que se recomenda usar `ggplot2::mpg` para o ficheiro de dados, e os seguintes são sobre as referências das colunas do ficheiro de dados; embora não esteja tecnicamente errado, pois não são referências externas, recomenda-se usar `.data$` para referências internas da seguinte maneira:

```
24 cool_graph <- function() {
25   ggplot2::ggplot(ggplot2::mpg,
26                   aes(x = .data$displ, y = .data$hwy, color = .
                        data$class)) +
27     ggplot2::geom_point() + ggplot2::geom_smooth(se = FALSE)
28 }
```

Erros, têm como consequência que o pacote ou não corre, ou corre numa maneira definitivamente indesejada. Estes devem ser resolvidos de imediato.

Software como o RStudio, ou IDE como o VSCode, permitem fazer *linting* do seu código enquanto o escreve, sugerir correções e até fazer as mais triviais por si, mas são opcionais na criação de qualquer pacote.

`lintr`, embora faça um teste compreensivo à certidão de um pacote, foca-se mais na correção do código em si e não, por exemplo, na estrutura do pacote, ou se este pode ser compilado agnosticamente relativamente ao

sistema operativo. O `devtools::check()` permite correr um bloco de testes completos e exaustivos ao pacote (`R CMD`), que se recomenda fazer sempre antes de disponibilizar ao público uma versão nova de um pacote. Mais, só depois de um pacote passar por estes testes sem quaisquer reportes (em especial avisos e erros) é que fica apto a ser submetido no CRAN.

1.3.2 *Vignettes*

Vignettes, obtidas com `vignette(pacote)`, são os guias extensivos de um pacote, indo para além da documentação do pacote em si, normalmente escritas usando a sintaxe LaTeX ou Markdown mas possuindo funcionalidades adicionais, onde código do R pode ser executado e o seu *output* inserido nestes documentos (através do pacote `knitr`, preferencialmente).

Essencialmente, uma *vignette* é um artigo académico escrito sobre o pacote, usualmente descrevendo a problemática que este pretende resolver, como o faz e detalhes particulares sob a sua implementação bem como a teoria matemática subjacente. Além disso, o R também possui o conceito de *articles*, que são *vignettes* que não são incluídas no pacote em si (sendo imediatamente referidas no ficheiro `.Rbuildignore` na sua criação).

O esqueleto de uma *vignette* pode ser criada com o comando:

```
> use_vignette("nome")
```

onde `"nome"` é o nome do ficheiro onde a *vignette* será criada. Este será colocado na recém-criada diretoria `vignettes/`.

1.3.3 Ficheiros de dados

Quando se deseja partilhar ficheiros de dados do R através de um pacote, estes têm de estar separados do código do pacote, para não poluir o *namespace* do utilizador e permitir o carregamento otimizado deste. Tal é possível colocando os objetos dentro de ficheiros `.Rdata` na pasta `data/`. A maneira mais fácil consiste em correr o comando `use_data()` no objeto R que se pretende criar:

```
> dados <- sample(1000)
> use_data(dados)
```

```
✓ Saving 'dados' to 'data/dados.rda'
• Document your data (see 'https://r-pkgs.org/data.html')
```

Como esta é também uma componente do pacote escrito no R, deverá ser documentada com um bloco de comentários do `roxygen2` para facilidade

dos utilizadores. Para tal, documenta-se a *string* com o nome do ficheiro de dados num dos ficheiros `.R`, por norma `R/data.R`, embora não se deva usar `@export`, pois estes são sempre exportados e não se quer exportar a *string* em si.

```
#' Dados arbitrários
#'
#' Dados arbitrários como exemplo, obtidos com \code{sample
    (1000)}.
#'
#' @format Um array com uma sequência aleatória (1-1000).
#'
#' @source \url{https://www.r-project.org/}
"dados"
```

R/data.R

Se estes dados externos são para uso interno do pacote (como, por exemplo, a inicialização de uma função), use o argumento `internal` para que estes não sejam incluídos no ficheiro `NAMESPACE`.

```
> use_data(dados, internal = T)
```

1.3.4 Testes

Testes, na diretoria `tests/`, são ficheiros do R que usam pacotes e sintaxes específicas e que correm o código do pacote sobre uma série de assunções criados pelos *devs* do pacote, para verificar a correta funcionalidade do *software*, ou para ajudar a acrescentar novas funcionalidades.

Assim, em vez de importar o código do pacote e verificar a funcionalidade à mão, como por exemplo verificar que `cool_sum(2, -5)` é igual a `-3`, os testes permitem automatizar o processo de verificação (se um pacote faz o que se pretende), ou confirmar que, nalgum desenvolvimento futuro, os pressupostos do pacote não foram removidos.

No R, para o desenvolvimento de testes, é habitual usar o pacote `testthat`; o `devtools` possui uma função para inicializar um pacote para testes (através do `usethis`):

```
> use_testthat()
```

```
✓ Adding 'testthat' to Suggests field in DESCRIPTION
✓ Creating 'tests/testthat/'
✓ Writing 'tests/testthat.R'
• Call `use_test()` to initialize a basic test file and open it
    for editing.
```

Como o *output* indica, usa-se:

```
> use_test("main.R")
```

```
✓ Writing 'tests/testthat/test-main.R'
• Modify 'tests/testthat/test-main.R'
```

para criar um ficheiro de testes para o ficheiro `.R` pretendido. Como os testes não são criados automaticamente, o código que se segue é apenas um exemplo predefinido da sintaxe necessária para criar um teste,

```
1 test_that("multiplication works", {
2   expect_equal(2 * 2, 4)
3 })
```

tests/testthat/test-main.R (como criado)

podendo-se acrescentar o seguinte para correr a assunção mencionada anteriormente:

```
5 test_that("soma gira funciona", {
6   expect_equal(pacote::cool_sum(2, -5), -3)
7 })
```

tests/testthat/test-main.R (teste acrescentado)

Os testes são corridos no seu próprio *workspace*, para confirmar se o pacote não tem *side effects* e que um utilizador que venha a usar o pacote não tenha problemas com pressupostos não comunicados pelo pacote como, por exemplo, dependências em falta.

Finalmente, os testes de um pacote serão automaticamente corridos usando a função `devtools::check()`, mas podem ser executados isoladamente com:

```
> devtools::test()
```

```
Loading pacote
Testing pacote
✓ |  OK F W S | Context
✓ |   2       | main

== Results =====================================================
OK:       2
Failed:   0
Warnings: 0
Skipped:  0
```

mostrando que os dois testes presentes no ficheiro correram sem problemas.

Como os testes usam assunções concretas ou que podem ser julgadas por um algoritmo, conceitos mais subjetivos como se o estilo de um gráfico é

adequado não são algo fáceis e, eventualmente, nem possíveis de testar. É esta a razão da falta de testes no pacote `vsd`, embora se pretenda vir a acrescentar para confirmação de funcionalidade.

1.3.5 Publicação no CRAN

São necessários alguns extras para publicar um pacote no CRAN, para além dos descritos em termos da correção do pacote (na Secção 1.3.1).

Estes consistem em:

1. Escolher um indicador de versão para a versão pública do pacote e alterar o ficheiro `DESCRIPTION` de acordo;

2. Criar e/ou atualizar os ficheiros `README.md` e `NEWS.md`;

3. Correr testes (com `R CMD`, ou seja `devtools::check()`) em pelo menos dois sistemas operativos[2] e na versão mais corrente do R;

4. Corrigir quaisquer erros e avisos, e anotar quaisquer notas reportadas no ficheiro `cran-comments.md`;

5. Confirmar manualmente que o pacote segue os critérios de publicação recomendados do CRAN [5];

6. Publicar com `devtools::release()`;

7. Aumentar a versão do pacote para uma versão em desenvolvimento (i.e., que acabe em `.9000` ou superior.).

Embora o ponto 2 seja opcional, é altamente recomendado a criação de dois ficheiros no pacote: um ficheiro para novos utilizadores que lista o que o pacote faz e como o utilizar (o `README.md`) e um ficheiro para os restantes utilizadores que lista as alterações feitas entre versões (o `NEWS.md`), com especial ênfase em alterações que quebram o API como, por exemplo, a remoção de uma função ou a alteração dos seus argumentos. Mais informação em como formatar estes documentos, incluindo a sintaxe da linguagem de *markup* Markdown, e como ter exemplos funcionais de código R no `README.md` estão presentes na bibliografia [25].

Testar pacotes para correção, com `devtools::check()`, em vários sistemas operativos e na versão mais recente do R pode ser complicado para o *developer*, mas o CRAN permite usar os seus servidores para correr a gama

[2]Pode ser só um, mas não se recomenda.

de testes completa no sistema operativo Windows com as versões mais recentes do R (publicadas, em desenvolvimento, ou na versão imediatamente anterior). A função `devtools::check_win()` automatiza este processo para a versão pública mais recente do R, havendo outras funções neste pacote para versões alternativas do R. Para outros sistemas operativos, é possível usar máquinas virtuais ou ambientes de compilação como o Travis.

Um pacote não poderá ser publicado se tiver quaisquer erros ou avisos, mas quaisquer notas obtidas nestes testes exaustivos deverão ser explicadas o porquê da sua inclusão (em inglês) para inspeção manual pelos voluntários do CRAN no ficheiro `cran-comments.md`. Todos os pacotes que são submetidos pela primeira vez no CRAN criam um aviso de tal, logo a presença deste ficheiro para facilitar a tarefa dos voluntários é praticamente obrigatória.

Se um pacote não passar na submissão, o *feedback* do CRAN, enviado para o email indicado no ficheiro `DESCRIPTION`, será curto. Calma. Deve-se tentar uma nova submissão, tendo um cuidado extra para confirmar que nada está em falta e que o que causou a falha foi corrigido, indicando a resolução no topo do ficheiro `cran-comments.md`.

Se o pacote for submetido com sucesso, parabéns! Só falta, opcionalmente, divulgar a sua publicação. Sempre que se deseja fazer uma nova publicação de um pacote, o processo deverá ser repetido, recomendado-se não fazer mais do que uma submissão a cada 1-2 meses. Tome-se cuidado adicional com modificações que alteram as funções ou argumentos do pacote (alterações ao API), recomendando-se usar avisos de *deprecation* e enviar, com antecedência, correspondência pessoal aos autores de pacotes dependentes antes da nova submissão.

Todos os ficheiros adicionais acrescentados nesta secção deverão ser incluídos no `.Rbuildignore`, para não serem incluídos na compilação do código do pacote, mas haverá um aviso nos testes `R CMD` se tal não ocorrer.

1.4 Controlo de revisões com `git`

O desenvolvimento de um pacote do R, sob licença aberta, requer que o código fonte do pacote esteja disponível livremente e que outros utilizadores o possam iterar e alterar. Embora o autor de um pacote, em teoria, pudesse apenas disponibilizar o código do pacote e não ter problemas, para facilitar a colaboração direta com outros indivíduos e para permitir desenvolvimento paralelo e regressões, é habitual usar um sistema de controlo de versões do código (SCM), sendo que a norma na indústria [3] é *software* Git.

O Git mantém um registo das alterações feitas num pacote durante o seu desenvolvimento. Este histórico pode ter várias instâncias em paralelo

(ramos, ou *branches*) que podem ser criadas arbitrariamente e depois sincronizadas (*merge*) quando desejado.

No entanto, o verdadeiro poder do Git é a sua ubiquidade, incluindo a gama de plataformas *online* que podem alojar o código de um pacote, e o facto de este ficar disponível *online* para uso e modificação por outrém. Este último aspeto será o foco desta secção, abordado de uma forma breve, mas o suficiente para que o código de um pacote seja colocado *online* e que as alterações no seu desenvolvimento sejam publicadas.

Embora o Git seja um programa de terminal, este pode ser acompanhado com um software de interface gráfica, ou pode ser instalado em separado, ou até estar integrado no IDE usado para redigir o pacote. Como os comandos no Git têm nomes específicos que todas as outras interfaces tentam seguir, ir-se-á descrever o processo como se os comandos fossem corridos da consola do utilizador, deixando ao leitor o exercício de encontrar o respetivo equivalente na sua metodologia de preferência.

1.4.1 Plataforma *host*

Em primeiro lugar, é necessário uma plataforma que suporta o Git. Para o pacote `vsd` usou-se o Github, embora existam outras alternativas públicas, ou até serviços privados por outrém.

Após criar uma conta na plataforma, deve-se criar um repositório, o termo para uma instância de um projeto sob Git, público para o pacote do R (Figura 1.2). Coloca-se o nome do repositório igual ao do pacote e recomenda-se acrescentar um ficheiro `.gitignore` adequado à linguagem R para filtrar ficheiros indesejados do repositório (como o `.Rbuildignore`, mas para o Git).

Um ficheiro `README.md` também é sugerido, mas recomenda-se não selecionar uma licença através do Github, pois o R usa um formato específico para organizar os ficheiros da licença (como visto na Secção 1.1), já estando tratada. Note também o nome do ramo por defeito na Figura 1.2, aqui sendo `main`, que será necessário depois.

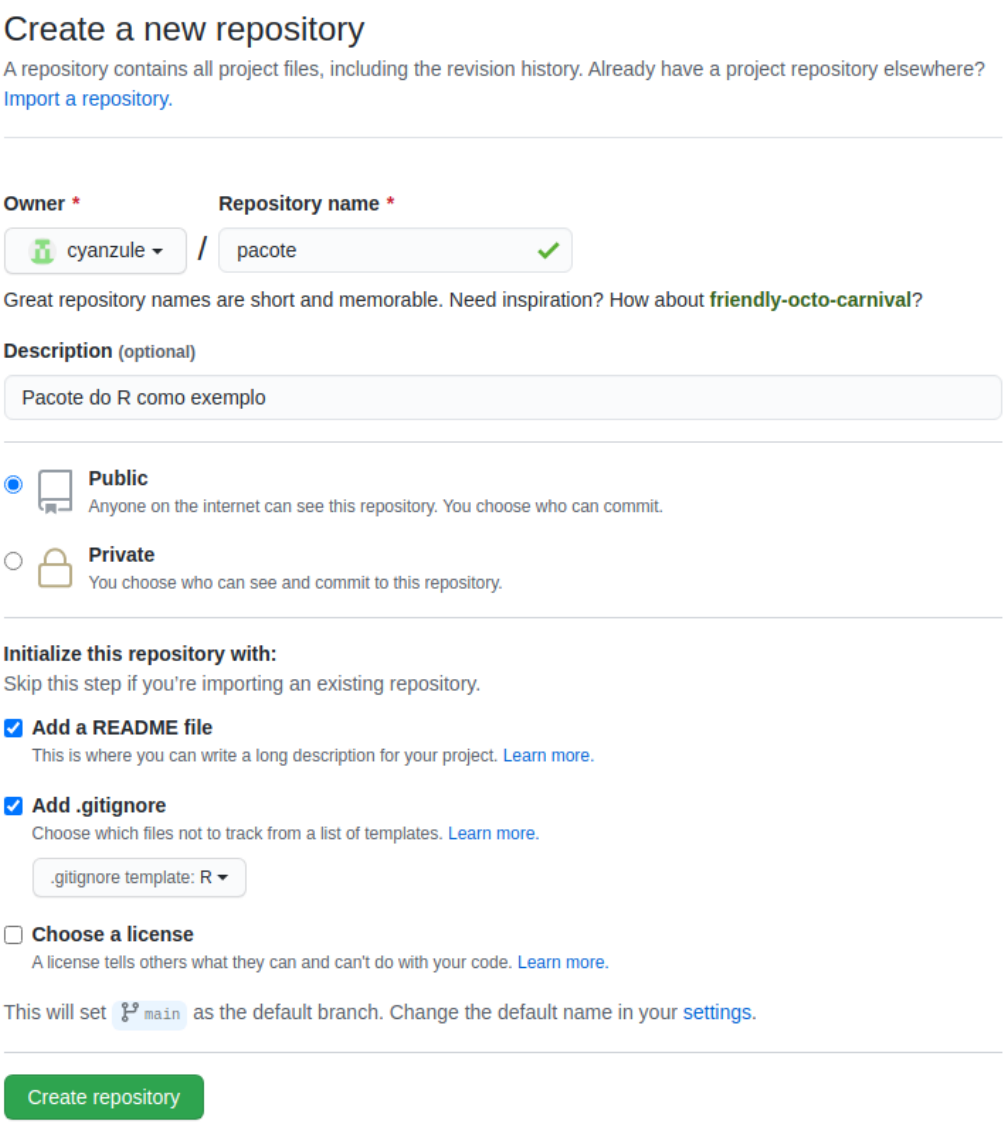

Figura 1.2: *Screenshot* da página de criação do repositório no Github, com as opções recomendadas já preenchidas.

1.4.2 `git init`

Agora que o repositório está criado *online*, é necessário o criar localmente e conectar os dois. Para tal, podem ser seguidos dois métodos:

1. Correr `git clone` com o link HTTPS/SSH fornecido pela plataforma Git para criar uma cópia do repositório localmente e mover todos os ficheiros do pacote já lá existentes;
2. Criar o repositório Git localmente e, em seguida, fazer a conexão manualmente com o *remote* (repositório não local ou externo).

Ir-se-á seguir o segundo método para exemplificar o mais recomendado para bases de código já existentes[3]. Assim, com um terminal definido para a diretoria `C:/Repos/pacote`, corre-se:

```
$ git init
```

```
Initialized empty Git repository in C:/Repos/pacote/
```

Este comando cria, por norma, uma pasta escondida ao utilizador, `.git`, onde o histórico de alterações no código e várias outras estruturas e configurações estão guardadas. Para que um projeto de código esteja sob um repositório Git, basta que esta pasta exista.

Para confirmar se está tudo bem (e consequentemente para ver o estado de um repositório em caso de problemas), use:

```
$ git status
```

```
On branch master

No commits yet

Untracked files:
(use "git add <file>..." to include in what will be committed)
  .Rbuildignore
  DESCRIPTION
  LICENSE
  LICENSE.md
  NAMESPACE
  R/
  data/
  man/
```

[3]Se usar o Git desde o início do desenvolvimento de um pacote, `git clone` antes de fazer `devtools::create()` na diretoria criada é suficiente, podendo, nesse caso, ignorar quaisquer comandos de consola até à Secção 1.4.4.

```
  tests/

nothing added to commit but untracked files present (use "git
    add" to track)
```

que daria erro se a diretoria não estivesse sob um repositório Git, mas também avisa que todos os ficheiros presentes na diretoria estão *untracked*, ou não presentes no histórico.

Antes de mais, para associar este repositório local ao do Github, corre-se o comando:

```
$ git remote add origin <url>
```

onde `<url>` é o *link* fornecido pela plataforma Git como o *remote URL*, sendo o mesmo usado no `git clone`.

Note-se que, por norma, o Git chama o ramo inicial de `master`, como indicado no `git status` na primeira linha, sendo que esta designação pode ser diferente da criada na plataforma *online*, como é o caso no Github, que por norma designa por `main`. Se tal ocorrer, deve-se correr o seguinte comando para alterar o ramo corrente:

```
$ git checkout -b main
```

com `main` a indicar o nome do ramo na plataforma externa. Como não se fez ainda qualquer histórico com o Git, este resume-se, essencialmente, a renomear o ramo base. Na versão 2.28.0 (e seguintes) do Git, pode-se imediatamente definir o nome do ramo inicial (e principal) com a *flag* `-b`:

```
$ git init -b main
```

1.4.3 `git pull`

Como ao criar o repositório no Github, potencialmente, também já foram criados os ficheiros `README.md` e `.gitignore`, o Github já criou um histórico inicial para o repositório, para já com uma única entrada (*commit*).

Se forem feitos *commits* locais após estas alterações remotas, ou se houver conflitos nas alterações realizadas, o Git, sob possível confusão da ordem de operações, irá evitar danificar o histórico presente, pedindo ao utilizador para tentar resolver o problema criado. Esta é a razão de, anteriormente, se ter recomendado não criar a licença de código aberta no Github, para evitar um conflito ao sincronizar (*merge conflict*).

Portanto, antes de tentar estabelecer quaisquer alterações no pacote e para sincronizar o local com o repositório remoto, transfere-se os `commits` do `origin` para o ramo `main` com:

```
$ git pull origin main
```

```
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 986 bytes | 493.00 KiB/s, done.
From https://github.com/cyanzule/pacote
* branch            main       -> FETCH_HEAD
* [new branch]      main       -> origin/main
```

(isto só tem de ser feito uma vez para ramos novos, podendo apenas fazer `git pull` no futuro para atualizar o ramo corrente).

Se estes ficheiros (`.gitignore` e o `README.md`) não estiverem criados e nenhum *commit* estiver presente no repositório remoto, o comando irá dar erro, pois não existirá o ramo desejado no remoto (ver como criar este ramo na Secção 1.4.6).

Mesmo assim, para evitar problemas que, se resolúveis, requerem esforço extra do utilizador ou no pior caso possível progresso perdido, deve-se sempre sincronizar o repositório local com o remoto antes de publicar (como na Secção 1.4.5) e antes de fazer quaisquer alterações locais no código.

1.4.4 `git add`

Agora que o Git está a manter localmente um histórico das alterações do código, este espera que o utilizador indique as alterações que são para ser guardadas. Para o Git, existe uma diferença entre como o código do pacote está presentemente e como o histórico está assinalado até esse momento (`HEAD`). Por exemplo, pode publicar parte das alterações efetuadas, enquanto se estão a realizar outras.

Para assinalar as alterações no pacote que são para ser inscritas, usa-se `git add`, seguido por uma *wildcard* dos ficheiros (ou até linhas) que se pretende seguir [9]. Se o objetivo for incluir todos os ficheiros, a *wildcard* `.` engloba quaisquer ficheiros, diretorias e ficheiros dentro destas, recursivamente. Então, com:

```
$ git add .
$ git status
```

```
On branch main

Changes to be committed:
(use "git restore --staged <file>..." to unstage)
```

```
  new file:   .Rbuildignore
  new file:   DESCRIPTION
  new file:   LICENSE
  new file:   LICENSE.md
  new file:   NAMESPACE
  new file:   R/data.R
  new file:   R/main.R
  new file:   R/pacote.R
  new file:   data/dados.rda
  new file:   man/cool_sum.Rd
  new file:   man/dados.Rd
  new file:   man/pacote.Rd
  new file:   tests/testthat.R
  new file:   tests/testthat/test-main.R
```

verifica-se que o Git (corretamente) identifica estes ficheiros como completamente novos, sendo essas as alterações comparadas ao `HEAD`.

1.4.5 `git commit`

Agora que se indicou ao Git as alterações desejadas, estas são submetidas ao histórico sobre um *commit*, um registo das alterações complementado com uma mensagem obrigatória de *commit*, funcionando como etiqueta das alterações juntamente com espaço extra onde o utilizador pode colocar o que quiser como, por exemplo, listar o que foi alterado e porquê.

```
$ git commit
```

Dependendo da configuração do Git, este abrirá um editor de texto (ou externamente, ou no terminal em si) para o utilizador redigir a mensagem do *commit*. À semelhança dos comentários de bloco para o R, a primeira linha é reservada como o título do *commit* (recomendando-se que esta não tenha mais do que 70 caracteres), e as subsequentes para uma descrição.

Após guardar o ficheiro, se este não estiver vazio, o Git cria o `commit`, movendo o histórico para a frente para acompanhar as alterações (desejadas) pelo utilizador.

Se o objetivo for apenas fazer um commit sem se preocupar com a mensagem, as alterações podem ser submetidas rapidamente só com um título através da sintaxe seguinte:

```
$ git commit -m "Título do commit"
```

```
[main 8e1a6e5] Título do commit
14 files changed, 166 insertions(+)
  create mode 100644 .Rbuildignore
  create mode 100644 DESCRIPTION
```

```
 create mode 100644 LICENSE
 create mode 100644 LICENSE.md
 create mode 100644 NAMESPACE
 create mode 100644 R/data.R
 create mode 100644 R/main.R
 create mode 100644 R/pacote.R
 create mode 100644 data/dados.rda
 create mode 100644 man/cool_sum.Rd
 create mode 100644 man/dados.Rd
 create mode 100644 man/pacote.Rd
 create mode 100644 tests/testthat.R
 create mode 100644 tests/testthat/test-main.R
```

A primeira linha refere o ramo, se este é o seu primeiro *commit* neste repositório (se for o caso, indica que é um *root-commit*), o começo do identificador único SHA256 do *commit*, e o seu título.

1.4.6 `git push`

Por último, sincroniza-se o histórico local para o repositório `origin` com:

```
$ git push origin main
```

```
Username for 'https://github.com': cyanzule
Password for 'https://cyanzule@github.com':
Enumerating objects: 22, done.
Counting objects: 100% (22/22), done.
Delta compression using up to 8 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (21/21), 5.36 KiB | 1.79 MiB/s, done.
Total 21 (delta 0), reused 0 (delta 0)
To https://github.com/cyanzule/pacote.git
  9c9654d..8e1a6e5  main -> main
```

que, como no `git pull`, na primeira execução tem-se de indicar que este ramo é para publicar para o repositório remoto `origin`.

O Git irá perguntar pelas credenciais de *login* da plataforma e, após as introduzir, fará *upload* do novo *commit*.

1.4.7 Sumário do processo Git

Em resumo, o *workflow* dentro de um único ramo no Git resume-se a:

1. `git pull` para obter e juntar (`merge`) as alterações remotas localmente;
2. Fazer alterações locais ao código;

3. `git add` para adicionar as alterações locais que se pretende publicar;

4. `git commit` para submeter as alterações adicionadas, juntamente com uma mensagem de publicação;

5. `git push` para enviar o *commit* para o repositório remoto.

Quando se acrescenta mais utilizadores (e ramos) a um projeto, o itinerário torna-se mais complexo, mas resume-se sempre a estes conceitos: sincronizar com alterações do *remote*, escolher o que publicar, *commit*, e mandar as alterações *upstream*. O código do pacote desenvolvido nesta secção está publicado aqui [6].

Capítulo 2

Algumas noções sobre Análise de Sobrevivência

A Análise de Sobrevivência é um ramo da Estatística que se refere ao estudo do tempo entre um instante inicial e um acontecimento específico, chamado de tempo de sobrevivência, podendo incluir o estudo de fatores de influência.

Apesar do nome, este pode envolver acontecimentos positivos ou menos mórbidos, como o tempo até obter uma promoção, até à regressão de um tumor após tratamento ou, em termos de sobrevivência de elementos não humanos, ao tempo de vida de uma lâmpada.

Estes cenários possuem características específicas, tanto na sua definição como na sua análise, que os separam de outros ramos da Estatística e aos quais tem de ser dada atenção especial e modelação única.

No R, o pacote *core*, contendo a plenitude de funções e estruturas de dados para lidar com a Análise de Sobrevivência, é o pacote `survival`. Este é parte dos pacotes base do R, ou seja, já vem incluído com a linguagem de programação, e assim não é necessário instalar separadamente, podendo-se importar de imediato para a sua utilização.

```
> library(survival)
```

Para visualizar a documentação descritiva deste pacote (ou como referência quaisquer pacotes), usa-se a função `vignette`:

```
> vignette("survival", package = "survival")
```

Para obter informação suplementar usa-se a função `browseVignettes`:

```
> browseVignettes(package = "survival")
```

Para visualizar a informação sobre funções, usa-se `?` antes do nome desta, ou seja:

```
> ?Surv
```

Ao longo deste trabalho ir-se-á referir outros pacotes, assim como funções incluídas nestes, numa forma sucinta, logo recomenda-se a leitura da respetiva a documentação para verificar detalhes adicionais e funções mais avançadas não referidas.

2.1 Censura

Um dos conceitos principais da Análise de Sobrevivência, especialmente em contraste com outros métodos de análise estatística, é a ocorrência de censura nos dados, onde a variável em estudo (o tempo) pode não ser observada por completo para todos os indivíduos.

Existem vários tipos de censura:

- Censura à direita: tempo de sobrevivência é superior ao observado;
- Censura à esquerda: tempo de sobrevivência é inferior ao observado;
- Censura intervalar: tempo de sobrevivência pertence a um intervalo mas não é observado.

Mais especificamente, alguns tipos de censura à direita são:

- Tipo I: O momento de censura é predeterminado no estudo como, por exemplo, num estudo com duração de dois anos ir-se-á censurar os tempos de todos os indivíduos para os quais não tenha sido observado o acontecimento neste período;
- Tipo II: O momento de censura ocorre aquando a ocorrência do acontecimento em estudo numa fração predeterminada da amostra. Por exemplo, num estudo do tempo de vida de lâmpadas industriais, o estudo acaba quando 25% da amostra falhar, sendo a restante 75% constituída por tempos censurados;
- Aleatória: A censura é aleatória, não sendo predeterminada.

Em regra, assume-se que a censura é não informativa, isto é, o mecanismo de censura é independente do mecanismo de sobrevivência. Deverão ser tidos cuidados extras na presença de censura informativa, pois pode inserir vieses nos modelos estatísticos e complicar a modelação na Análise de Sobrevivência.

No R, a estrutura de dados e a sua função correspondente, `Surv`, presente no pacote `survival`[20] é a preferida para inserir dados no âmbito da Análise de Sobrevivência, com a ocorrência de censura.

A função `Surv` possui a seguinte sintaxe, simplificada,

```
> Surv(time, event, type)
```

onde:

time Estrutura de dados com o tempo de estudo dos indivíduos em questão;

event Estrutura de dados que indica se na variável `time` houve censura e, em caso afirmativo, qual o tipo: onde em geral `0` é assumida como censura à direita, `1` como a ocorrência do evento, `2` como censura à esquerda, e `3` como censura intervalar;

type *String* especificando o tipo de censura, como `"right"` ou `"left"`. Pode ser omitido, pois é deduzido com base nos valores de entrada e nas suas estruturas (ver documentação para mais detalhes).

Em caso de se ter, para cada indivíduo, a data de entrada no estudo e a data de ocorrência do acontecimento ou da censura, usa-se a sintaxe alternativa

```
> Surv(time, time2, event, type)
```

com `time` e `time2` sendo estrutura de dados, mas agora, respetivamente, com o tempo inicial e final para formar os intervalos (`time`, `time2`] para cada indivíduo. Sugestivamente, também é usada a estrutura `start` e `stop`.

O pacote `survival` possui também várias ficheiros de dados como exemplo para as suas componentes, que são usadas nos exemplos na documentação desta função. Estes exemplos podem ser executados com

```
> example(Surv)
```

que irão ser usados em exemplos posteriores de Análise de Sobrevivência, neste e no capítulo seguinte.

2.2 Caracterização do tempo de vida

Na Análise de Sobrevivência pretende-se modelar a variável aleatória que representa o tempo de vida dos indivíduos, T.

Seja $S(t)$ a função de sobrevivência que é definida pela probabilidade de sobrevivência para além de um instante t, ou seja:

$$S(t) = P(T > t), \text{ com } 0 \leq t < \infty$$

Esta função é contínua, tem o valor 1 no momento inicial $t = 0$, e é monótona decrescente, nunca sendo inferior a 0, ou seja, respetivamente:

$$S(0) = 1$$
$$S(t + \Delta t) \leq S(t),\ \forall \Delta t, t \geq 0$$
$$\lim_{t \to \infty} S(t) = 0$$

A variável T pode ser definida também através da função de risco, que define a taxa de ocorrência do acontecimento num instante específico, ou seja, a probabilidade da ocorrência do acontecimento em questão, se o sujeito sobreviveu até t, num intervalo de tempo pequeno logo após t. Formalmente:

$$h(t) = \lim_{dt \to 0^+} \frac{P(t \leq T < t + dt \mid T \geq t)}{dt}$$

Um valor maior na função de risco implica maior probabilidade de um acontecimento ocorrer nesse instante específico, e vice-versa, daí ser também chamada de função intensidade, ou da força da mortalidade. A função de risco verifica as seguintes propriedades:

$$h(t) \geq 0$$
$$\int_0^\infty h(t)dt = \infty$$

Tendo a função de risco definida, podemos obter a função de sobrevivência da seguinte maneira:

$$S(t) = \exp\left(-\int_0^t h(u)du\right) = \exp(-H(t)) \tag{2.1}$$

onde

$$H(t) = \int_0^t h(u)du \tag{2.2}$$

é designada por função de risco cumulativa.

Para além das funções de sobrevivência e risco, uma distribuição do tempo de vida pode ser ainda caracterizada pela função densidade de probabilidade:

$$f(t) = -S'(t) = \lim_{dt \to 0^+} \frac{P(t \leq T < t + dt)}{dt} \tag{2.3}$$

Isto é relevante pois oferece uma maneira alternativa de obter a função de risco,

$$h(t) = \frac{f(t)}{S(t)} \tag{2.4}$$

tendo em conta que o risco de um indivíduo em t é a probabilidade de um acontecimento ocorrer suficientemente perto desse instante dividida pela probabilidade de não ter ocorrido o acontecimento até este instante para esse indivíduo (ainda estar vivo).

A censura complica o cálculo do valor esperado, ou a média, de T. Esta é em regra definido por:

$$\mu = E(T) = \int_0^\infty t f(t) dt$$

que, por integração por partes, pode ser rescrita como

$$\mu = \int_0^\infty S(t) dt$$

mas só pode ser calculada se $S(\infty) = 0$, isto é, se o último valor observado não for censurado.

Além disso T, habitualmente, tem uma distribuição assimétrica à direita, por isso, é mais adequado usar a mediana e não a média como medida de localização para T, mas a presença de censura nos dados também implica um cálculo alternativo desta medida.

Assim, define-se a mediana de T como o valor de t mais pequeno onde a função de sobrevivência é menor ou igual a 0.5, isto é:

$$m = \min\{t : S(t) \leq 0.5\}$$

e, mais genericamente, o quantil de probabilidade p é dado por:

$$\chi_p = \min\{t : S(t) \leq 1 - p\}$$

2.3 Modelação paramétrica

Para obter a estimativa da função de sobrevivência que mais se adequa a um conjunto de dados, tanto pode ser usada a modelação não paramétrica ou a paramétrica. Nesta secção será abordada esta última.

Como exemplo, suponha-se que a taxa de risco numa amostra evidencie-se constante, $h(t) = \lambda$. Podemos deduzir a função de sobrevivência respetiva através de (2.2) e (2.1):

$$H(t) = \int_0^t \lambda du = \lambda [u]_0^t = \lambda t$$
$$S(t) = \exp(-\lambda t)$$

Obtemos assim a família de distribuições de sobrevivência mais simples possível, a distribuição exponencial, com λ como parâmetro do modelo, representando o risco (constante) de um indivíduo. Ora, este é um caso particular da distribuição de Weibull, que oferece maior flexibilidade na modelação:

$$\begin{aligned} h(t) &= \alpha\lambda(\lambda t)^{\alpha-1} = \alpha\lambda^{\alpha}t^{\alpha-1} \\ S(t) &= \exp(-(\lambda t)^{\alpha}) \end{aligned} \tag{2.5}$$

sendo o modelo exponencial a sub-família de distribuições quando $\alpha = 1$.

Sejam $(t_1, \delta_1), (t_2, \delta_2), \ldots, (t_n, \delta_n)$ os valores observados numa amostra de dimensão n, e δ_i o indicador de censura que toma o valor 1 quando o acontecimento é observado e 0 caso contrário.

Habitualmente a função de verosimilhança é definida à custa da função densidade (2.3) da seguinte forma:

$$L = \prod_{i=1}^{n} f(t_i)$$

mas a existência de censura à direita obriga a que seja definida por

$$L = \prod_{i=1}^{n} [f(t_i)]^{\delta_i} [S(t_i)]^{1-\delta_i}$$

Usando (2.4), podemos calcular L sem ter de usar $f(t)$:

$$\begin{aligned} L &= \prod_{i=1}^{n} [h(t_i)S(t_i)]^{\delta_i} [S(t_i)]^{1-\delta_i} \\ &= \prod_{i=1}^{n} [h(t_i)]^{\delta_i} S(t_i) \end{aligned} \tag{2.6}$$

Esta pode ser generalizada para os outros tipos de censura, considerando que cada indivíduo na amostra possui uma contribuição diferente nesta consoante o tipo de censura:

$$L = \prod_{i \in D} f(t_i) \prod_{i \in C} S(t_i) \prod_{i \in L} [1 - S(t_i)] \prod_{i \in I} [S(l_i) - S(r_i)]$$

onde

- D é o conjunto de índices associados aos indivíduos com tempos de vida exatos;

- C o conjunto de índices onde ocorre censura à direita;
- L o conjunto de índices onde ocorre censura à esquerda;
- I o conjunto de índices onde ocorre censura intervalar.

Existe no CRAN um pacote que facilita a obtenção dos valores dos parâmetros que maximizam a verosimilhança de um modelo paramétrico com um conjunto de dados, o pacote `flexsurv`.

```
> install.packages("flexsurv")
> library(flexsurv)
```

Como exemplo, ir-se-á considerar uma distribuição de Weibull, como explorada em (2.5). Note-se que o `flexsurvreg` usa internamente `dweibull`, que utiliza os argumentos a e b (*shape* e *scale* do `print`, respetivamente), assim:

$$f(t) = \frac{a}{b}\left(\frac{t}{b}\right)^{a-1} \exp\left(-\left(\frac{t}{b}\right)^{a}\right)$$

ou seja, $a = \alpha$ e $b = \lambda^{-1}$.

Ao tentar modelar os dados do ficheiro `ovarian` com uma distribuição Weibull resulta no seguinte modelo:

```
> flex_ovarian <- flexsurvreg(Surv(futime, fustat) ~ 1, data =
    ovarian, dist="weibull")
> print(flex_ovarian)
```

```
Call:
flexsurvreg(formula = Surv(futime, fustat) ~ 1, data = ovarian
    ,
dist = "weibull")

Estimates:
est         L95%        U95%        se
shape       1.108       0.674       1.822       0.281
scale    1225.419    690.421    2174.979    358.714

N = 26,  Events: 12,  Censored: 14
Total time at risk: 15588
Log-likelihood = -97.9539, df = 2
AIC = 199.9078
```

Este modelo não aparenta se adequar aos dados, como se pode ver graficamente na Figura 2.1:

```
> plot(flex_ovarian)
```

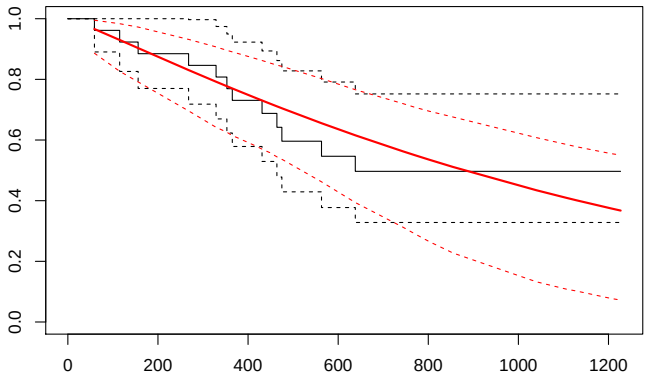

Figura 2.1: Estimativa de Kaplan-Meier da função de sobrevivência e estimativa da função de sobrevivência obtida através do modelo Weibull ($\alpha = 1.108$, $\lambda^{-1} = 1225.419$) dos dados do ficheiro `ovarian` do pacote `survival`.

2.4 Modelação não paramétrica

Embora na Análise de Sobrevivência seja possível a modelação paramétrica dos tempos (ver Secção 2.3), é frequente a utilização de modelação não (ou semi-) paramétrica para obter a estimativa da função de sobrevivência que mais se adequa a um conjunto de dados de sobrevivência.

Esta abordagem oferece vários benefícios, pois evita a necessidade de selecionar o modelo matemático mais adequado e estimar os seus parâmetros. A Análise de Sobrevivência é usada regularmente no contexto da medicina e outras aplicações na saúde, que pode oferecer conjunto de dados irregulares ou orgânicos, que dificulta a escolha da família de funções paramétricas adequada ao problema.

Se existe interesse em estimar os parâmetros, estes costumam estar relacionados com as covariáveis em estudo, ou seja, no vetor $\boldsymbol{\beta}$, onde covariáveis designam quaisquer variáveis independentes do estudo, que podem caracterizar a amostra e/ou constituir um fator de risco para a ocorrência do acontecimento de interesse (como a idade de um indivíduo).

Ora, quando não existe censura, a cada dado instante t a função de sobrevivência pode ser intuitivamente estimada pela proporção de indivíduos na amostra aos quais ainda não ocorreu o acontecimento em estudo, isto é, quantos indivíduos que ainda sobrevivem para além do instante t,

$$\hat{S}(t) = \frac{\text{número de observações} > t}{n} \tag{2.7}$$

Kaplan e Meier [14] propuseram em 1958 um estimador não paramétrico para a função de sobrevivência, baseado no princípio anterior, mas incluindo a ocorrência de censura. Este estimador é designado por estimador produto-limite, ou estimador de Kaplan-Meier.

Sejam $t_{(1)}, t_{(2)}, \ldots, t_{(k)}$, $k \leq n$, os instantes onde se observam o evento em estudo na amostra n, n_i o número de indivíduos em risco em $t_{(i)}$, e d_i o número de indivíduos para os quais ocorreu o acontecimento de interesse nesse instante. O estimador de Kaplan-Meier para a função de sobrevivência é

$$\hat{S}(t) = \prod_{t_{(i)} \leq t} \frac{n_i - d_i}{n_i} = \prod_{t_{(i)} \leq t} \left(1 - \frac{n_i}{d_i}\right) \tag{2.8}$$

sendo $\hat{S}(t) = 1$ para $0 \leq t < t_{(1)}$.

Note-se que, na inexistência de censura, isto é, $n_{i+1} = n_i - d_i, \forall i$, e considerando $t_{(j)}$ como o maior valor de $t_{(i)}$ menor ou igual a t:

$$\begin{aligned}
\hat{S}(t) &= 1 \left(\frac{n_1 - d_1}{n_1}\right) \left(\frac{n_2 - d_2}{n_2}\right) \left(\frac{n_3 - d_3}{n_3}\right) \cdots \left(\frac{n_j - d_j}{n_j}\right) \\
&= \frac{n_2}{n_1} \frac{n_3}{n_2} \frac{n_4}{n_3} \cdots \frac{n_j - d_j}{n_j} \\
&= \frac{n_j - d_j}{n_1} = \frac{n_{j+1}}{n}
\end{aligned}$$

obtendo-se a estimativa da função de sobrevivência em (2.7).

Para obter o intervalo de confiança para $S(t_0)$, onde t_0 é um instante arbitrário, é necessário estimar a variância do estimador de Kaplan-Meier da função de sobrevivência, onde se pode usar a fórmula de Greenwood para o calcular:

$$\text{vâr}\left(\hat{S}(t)\right) = \left[\hat{S}(t)\right]^2 \sum_{t_{(i)} \leq t} \frac{d_i}{n_i(n_i - d_i)} \tag{2.9}$$

Com (2.9), o intervalo de confiança de $100(1-\alpha)\%$ em t_0 é então:

$$\left(\hat{S}(t_0) - z_{1-\alpha/2}\sqrt{\text{vâr}\left(\hat{S}(t_0)\right)}, \hat{S}(t_0) + z_{1-\alpha/2}\sqrt{\text{vâr}\left(\hat{S}(t_0)\right)}\right)$$

Porém, intervalos de confiança calculados com base na variância obtida pela fórmula de Greenwood podem exceder os limites da função de sobrevivência, $0 \leq \hat{S}(t) \leq 1$, logo um cálculo alternativo que evita esta anomalia

consiste em usar a transformação log-log do estimador de Kaplan-Meier:

$$\hat{\text{var}}\left(\log\left[-\log \hat{S}(t)\right]\right) = \frac{1}{\left[\log \hat{S}(t)\right]^2} \sum_{t_{(i)} \leq t} \frac{d_i}{n_i(n_i - d_i)}$$

No R, a família de funções `survfit` permitem fazer a estimação da função de sobrevivência, usando a estimativa de Kaplan-Meier por norma quando passada uma fórmula como o primeiro argumento (usando `survfit.formula` por detrás), que é de interesse neste capítulo.

Assim, um resumo da sintaxe de uso é:

```
> survfit(formula, data, subset, conf.type, conf.int)
```

onde

formula Expressão na forma de `Surv(...)` $\sim$ `1` quando não se inclui covariáveis, ou `Surv(...)` $\sim$ `term1 + term2 + ...` quando se inclui;

data Uma *data frame* de onde as variáveis usadas nos outros argumentos são extraídos, ou seja, os dados da amostra em questão;

subset Expressão opcional onde só as linhas onde esta é verdadeira são usadas na estimativa;

conf.type *String* que indica o tipo de estimativa para o intervalo de confiança a usar, como o `"log"` (predefinido) ou `"log-log"`;

conf.int Valor para o nível de confiança do intervalo, estando por regra 0.95.

Como primeiro exemplo:

```
> fit_ovarian <- survfit(formula = Surv(futime, fustat) ~ 1,
    data = ovarian)
> print(fit_ovarian)
```

```
 n  events  median 0.95LCL 0.95UCL
26      12     638     464      NA
```

mas falta o limite superior do intervalo de confiança (porque as observações com valores mais elevados correspondem a observações censuradas, como se pode confirmar com `summary(fit_ovarian)`).

Ora, se usarmos a função `plot`, podemos obter um gráfico rudimentar da função de sobrevivência obtida pelo estimador de Kaplan-Meier, e o seu intervalo de confiança, juntamente com símbolos `+` onde houve censura se escolhermos o argumento `mark.time` como verdadeiro, como na Figura 2.2:

```
> plot(fit_ovarian, mark.time = T)
```

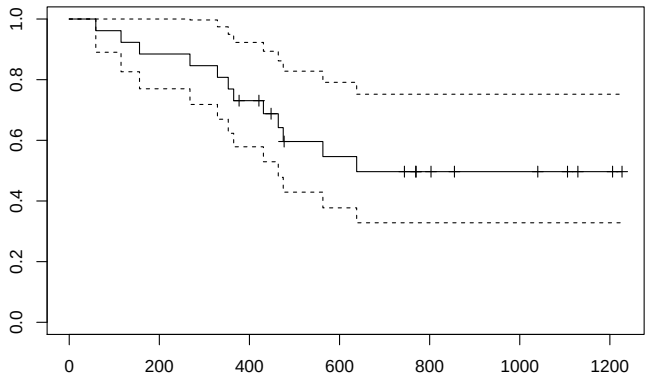

Figura 2.2: Estimativa de Kaplan-Meier da função de sobrevivência (e respetivo intervalo de confiança) correspondente ao dados do ficheiro `ovarian` no pacote `survival`.

As linhas a tracejado são os limites de confiança da curva de sobrevivência estimada, com a linha sólida sendo a estimativa Kaplan-Meier (evidente pois esta é uma estimativa em escada).

Como outro exemplo, para mostrar sub-curvas, usaremos um obtido da documentação da função,

```
> fit_aml <- survfit(Surv(time, status) ~ x, data = aml, conf.
    type = "log-log")
> print(fit_aml)
```

```
                 n events median 0.95LCL 0.95UCL
x=Maintained    11      7     31      13      NA
x=Nonmaintained 12     11     23       5      33
```

Como se pode verificar, foram estimadas duas curvas de sobrevivência, delineadas pela coluna `x` no ficheiro de dados `aml`, uma para quando `x = Maintained` e outra para os restantes indivíduos, onde `time` é o tempo de estudo e `status` indica se houve censura ou não no tempo do indivíduo. Ao fazer `print` do objeto obtido, consegue-se ver a mediana de cada sub-curva e os seus intervalos de confiança, verificando-se que o limite superior não está definido para a primeira curva, como aconteceu no ficheiro `ovarian`, mas está

para a sub-curva `x=Nonmaintained` (porque o último tempo corresponde a um tempo de vida observado).

Podemos obter um gráfico rudimentar destas curvas, como o na Figura 2.3, usando `lty` para variar o estilo das linhas e assim as distinguir (a linha mais sólida neste caso corresponde a `x=Maintained`):

```
> plot(fit_aml, lty=1:2)
```

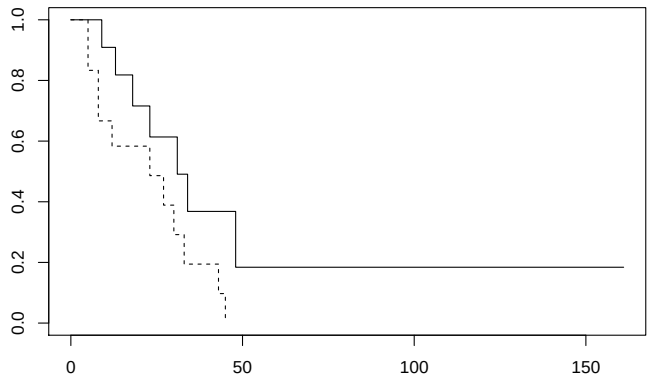

Figura 2.3: Estimativas de Kaplan-Meier da função de sobrevivência para as categorias de `x`, do ficheiro de dados `aml` do pacote `survival`.

Quando há mais do que uma curva, o R, por norma, não mostra os limites dos intervalos de confiança, logo se queremos ver temos de usar um método alternativo, tal como fazer cada uma das curvas individualmente, como demonstrado na Figura 2.4:

```
> plot(survfit(Surv(time, status) ~ 1, data = aml, subset = x
    == "Nonmaintained", conf.type = "log-log"), mark.time = T)
```

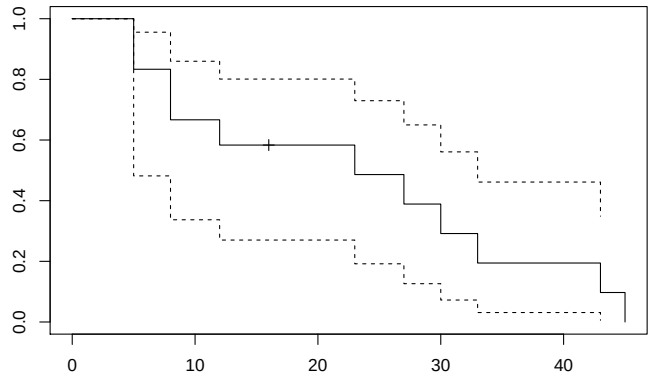

Figura 2.4: Estimativa de Kaplan-Meier da função de sobrevivência, juntamente com o seu intervalo de confiança para $\alpha = 0.05$, quando `x = Nonmaintained`, no ficheiro de dados `aml` do pacote `survival`

Após a estimação da função de sobrevivência, podem ser realizados vários testes estatísticos, nomeadamente para comparar grupos ou para verificar a proporcionalidade das funções de risco (consultar [18]).

2.4.1 Estimador da função de risco

Tendo em conta (2.1), um estimador natural de $H(t)$ é $\hat{H}(t) = -\log(\hat{S}(t))$, mas não é a melhor opção para amostras pequenas. Assim, um estimador alternativo, baseado nos d_i e n_i em (2.8) é o estimador de Nelson-Aalen, que calcula a função de risco cumulativa empírica da seguinte forma:

$$\hat{H}(t) = \sum_{t_{(i)} \leq t} \frac{d_i}{n_i}$$

cuja variância é dada por

$$\text{vâr}\left(\hat{H}(t)\right) = \sum_{t_{(i)} \leq t} \frac{d_i}{n_i^2}$$

Porém, é bastante instável quando se deseja obter $\hat{h}(t)$, recomendando-se usar um método de suavização usando uma função centralizada *kernel*, $K(u)$, para suavizar as várias instâncias de $\frac{d_i}{n_i}$,

$$\hat{h}(t) = \frac{1}{b} \sum_{t_{(i)} \leq t} K\left(\frac{t - t_{(i)}}{b}\right) \frac{d_i}{n_i}$$

onde b é o parâmetro de suavização que controla a granularidade desta estimativa.

No R existe o pacote `muhaz` que estima a função de risco por métodos não paramétricos, usando um conceito de alisamento, como descrito atrás.

```
> install.package("muhaz")
> library("muhaz")
> plot(muhaz(ovarian$futime, ovarian$fustat))
```

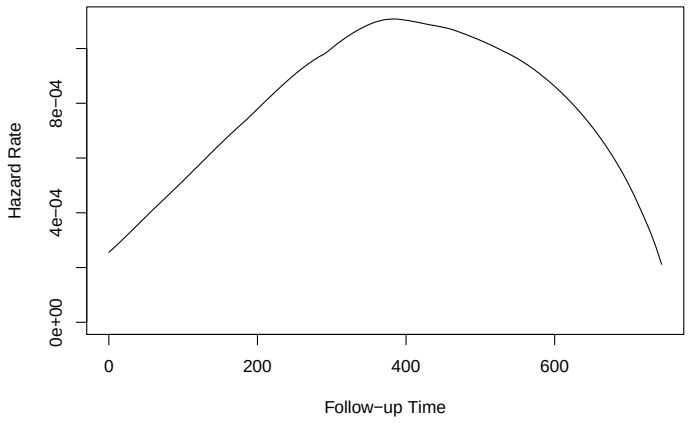

Figura 2.5: Estimativa da função de risco suavizada para os dados do ficheiro `ovarian` do pacote `survival`.

Ao comparar as Figura 2.2 e 2.5 podem surgir algumas dúvidas, tais como a razão de nesta última a escala do eixo das abcissas terminar num valor mais pequeno. Ora, isto é devido a alguns parâmetros deixados em branco na função `muhaz`, que passaremos a explicar, sucintamente, a sua sintaxe:

```
> muhaz(time, event, subset, min.time, max.time, bw.grid)
```

time, event Descrito na função `Surv`;

subset Descrito na função `survfit`;

min.time O limite inferior do tempo usado na suavização. Zero é o valor predefinido;

max.time O limite superior do tempo usado na suavização. Está predefinido que este corresponde ao instante em que ainda restam dez pacientes em risco ($t_{(i)} : n_i = 10$);

bw.grid O coeficiente de alisamento b, que se não for definido é calculado através do intervalo de tempo e o número de observações não censuradas (ver documentação, `?muhaz`).

2.5 Modelo de regressão de Cox

O modelo de regressão semi-paramétrico de Cox, descrito em 1972, é o modelo mais usado na análise de tempos de vida, permitindo também a análise e incorporação das covariáveis do estudo [18].

Seja T uma variável aleatória contínua, e $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_p)$ um vetor de p parâmetros associados aos efeitos das covariáveis de interesse ao estudo do modelo. Se considerarmos $\boldsymbol{z} = (z_1, z_2, \ldots, z_p)'$ como o valor destas covariáveis para um indivíduo, Cox propõe um modelo de regressão em que as covariáveis têm um efeito multiplicativo na função de risco

$$h(t|\boldsymbol{z}) = h_0(t)\exp(\boldsymbol{z}\boldsymbol{\beta}) = h_0(t)\exp(z_1\beta_1 + z_2\beta_2 + \cdots + z_p\beta_p) \tag{2.10}$$

onde $h_0(t)$, designada por função de risco subjacente, é uma função arbitrária (que pode ser estimada ou modelada posteriormente com outro método) não negativa que representa a função de risco para um indivíduo padrão, ou seja, um indivíduo com vetor de covariáveis nulo ($\boldsymbol{z} = \mathbf{0}$).

Como a função $h_0(t)$ será a mesma para todos os indivíduos, verifica-se que para quaisquer dois indivíduos (i e j) em estudo com os correspondente vetores de covariáveis $\boldsymbol{z}_i$ e $\boldsymbol{z}_j$, a razão entre os seus riscos

$$\frac{h(t|\boldsymbol{z}_i)}{h(t|\boldsymbol{z}_j)} = \frac{\exp(\boldsymbol{z}_i\boldsymbol{\beta})}{\exp(\boldsymbol{z}_j\boldsymbol{\beta})} = \exp((\boldsymbol{z}_i - \boldsymbol{z}_j)\boldsymbol{\beta}) \tag{2.11}$$

não depende de t, e é assim constante ao longo do tempo. Por outras palavras, este modelo assume que o efeito das covariáveis não sofre qualquer alteração durante o período de observação.

Assim, é possível estimar os efeitos das covariáveis sem ter de modelar o tempo de sobrevivência diretamente, daí o modelo de Cox ser chamado semi-paramétrico, ignorando-se o cálculo do $h_0(t)$.

Podemos então também definir a função de risco cumulativa, e através de (2.1), a função de sobrevivência:

$$H(t|\boldsymbol{z}) = H_0(t)\exp(\boldsymbol{z}\boldsymbol{\beta})$$
$$S(t|\boldsymbol{z}) = S_0(t)^{\exp(\boldsymbol{z}\boldsymbol{\beta})} = \exp(-H_0(t))^{\exp(\boldsymbol{z}\boldsymbol{\beta})} \tag{2.12}$$

Para estimar os coeficientes de regressão, Cox usa uma função de verosimilhança parcial, começando pela função de verosimilhança parcial para um indivíduo i, e o seu tempo de vida $t_{(i)}$.

Seja $R_i = \{j : t_j \geq t_{(i)}\}$ o conjunto dos índices associados aos indivíduos ainda em risco no instante $t_{(i)}$; então

$$L_i = L(h(t|\boldsymbol{z}_i)) = \frac{h_i(t_{(i)})}{\sum_{j\in R_i} h_j(t_j)}$$

que, devido aos riscos serem proporcionais, como demonstrado em (2.11), leva a que esta não precise de depender de $h_0(t)$. Consequentemente, pode ser expressa unicamente pelas covariáveis, ou seja,

$$L_i = \frac{\exp(\boldsymbol{z}_i\boldsymbol{\beta})}{\sum_{j\in R_i}\exp(\boldsymbol{z}_j\boldsymbol{\beta})}$$

e assim a verosimilhança parcial $L(\boldsymbol{\beta})$ é o produto das contribuições individuais:

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{k} \frac{\exp(\boldsymbol{z}_i\boldsymbol{\beta})}{\sum_{j\in R_i}\exp(\boldsymbol{z}_j\boldsymbol{\beta})} \tag{2.13}$$

onde $\{i : 1, \ldots, k\}$ corresponde ao conjunto dos índices associados aos indivíduos com tempos de vida não censurados.

Para facilidade de cálculo, para obter $\hat{\boldsymbol{\beta}}$, usa-se o logaritmo da verosimilhança parcial, $LL(\boldsymbol{\beta})$,

$$LL(\boldsymbol{\beta}) = \sum_{i=1}^{k}\left(\boldsymbol{z}_i\boldsymbol{\beta} - \log\left(\sum_{j\in R_i}\exp(\boldsymbol{z}_j\boldsymbol{\beta})\right)\right) \tag{2.14}$$

derivando-o e igualando-o a zero.

Note-se que (2.13) não tem em conta observações empatadas, pois considera que estas têm probabilidade nula num modelo contínuo, mas podem ser observadas na prática. Assim, é feito o seguinte ajuste à função de verosimilhança parcial, proposta por Breslow em 1974 [2]:

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{k} \frac{\exp(\boldsymbol{s}_i\boldsymbol{\beta})}{\left[\sum_{j\in R_i}\exp(\boldsymbol{z}_j\boldsymbol{\beta})\right]^{d_i}}$$

onde d_i é o número de indivíduos que morrem em $t_{(i)}$, $\boldsymbol{z}_{ij}$ o vetor de covariáveis para o j-ésimo indivíduo cuja morte ocorre em $t_{(i)}$, e $\boldsymbol{s}_i = \sum_{j=1}^{d_i} \boldsymbol{z}_{ij}$. Esta aproximação é aceitável desde que d_i seja suficientemente pequena comparado ao número de indivíduos pertencentes a R_i em cada i.

O R permite escolher o método para lidar com empates, juntamente com outros critérios a considerar, na estimativa de $\boldsymbol{\beta}$ com a função `coxph` (*Cox's proportional hazards*) no pacote `survival`. Esta possui uma sintaxe similar à função `survfit`:

```
> coxph(formula, data, subset, ties)
```

formula Expressão na forma `Surv(...)` $\sim$ `term1 + term2 + ...`, com `term1`, `term2`, ... como as covariáveis;

data Descrito na função `survfit`;

subset Descrito na função `survfit`;

ties *String* que identifica o método a usar em empates: `"breslow"`, `"efron"` ou `"exact"` (ver respetiva documentação para mais detalhes). Por norma o R usa o `"efron"` mas se houver um número reduzido de empates, produz resultados semelhantes ao `"breslow"`.

Como exemplo, podemos usar o ficheiro de dados `lung`:

```
1 > cox_lung <- coxph(Surv(time, status) ~ age + sex, lung)
2 > summary(cox_lung)
```

```
Call:
coxph(formula = Surv(time, status) ~ age + sex, data = lung)

n= 228, number of events= 165

        coef exp(coef)  se(coef)      z Pr(>|z|)
age  0.017045  1.017191  0.009223  1.848  0.06459 .
sex -0.513219  0.598566  0.167458 -3.065  0.00218 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

    exp(coef) exp(-coef) lower .95 upper .95
age    1.0172     0.9831    0.9990    1.0357
sex    0.5986     1.6707    0.4311    0.8311

Concordance= 0.603  (se = 0.025 )
Likelihood ratio test= 14.12  on 2 df,   p=9e-04
Wald test            = 13.47  on 2 df,   p=0.001
Score (logrank) test = 13.72  on 2 df,   p=0.001
```

sendo a primeira metade do *input* equivalente ao obtido com `print(cox_lung)`. Ver-se-à uma maneira alternativa de visualizar os coeficientes do modelo de Cox na Secção 3.4, com gráficos de floresta.

Tal como as estimativas de Kaplan-Meier da função de sobrevivência, podemos usar o modelo Cox para comparar grupos, mas, além disso, também para verificar se o efeito das covariáveis é significativo. Note-se a presença de valores p e testes estatísticos como o teste de Wald.

No entanto, a função de verosimilhança para o modelo de Cox, usando (2.6) com (2.10) e (2.12), é especificamente:

$$\begin{aligned} L(\boldsymbol{\beta}, h_0(t)) &= \prod_{i=1}^{n} \left[h_0(t_i)\exp(\boldsymbol{z}_i\boldsymbol{\beta})S_0(t_i)^{\exp(\boldsymbol{z}_i\boldsymbol{\beta})}\right]^{\delta_i} \left[S_0(t_i)^{\exp(\boldsymbol{z}_i\boldsymbol{\beta})}\right]^{1-\delta_i} \\ &= \prod_{i=1}^{k} \frac{\exp(\boldsymbol{z}_i\boldsymbol{\beta})}{\sum_{j\in R_i}\exp(\boldsymbol{z}_j\boldsymbol{\beta})} \prod_{i=1}^{k}\left(h_0(t_i)\sum_{j\in R_i}\exp(\boldsymbol{z}_j\boldsymbol{\beta})\right)\prod_{i=1}^{n} S_0(t_i)^{\exp(\boldsymbol{z}_i\boldsymbol{\beta})} \end{aligned}$$

requerendo a estimativa de $h_0(t)$ e $S_0(t)$.

Existem várias abordagens para as estimar, mas uma possível é proposta por Breslow, onde

$$\hat{h}_i = \frac{d_i}{\left(t_{(i)} - t_{(i-1)}\right)\sum_{j\in R_i}\exp(\boldsymbol{z}_i\hat{\boldsymbol{\beta}})}$$

e assim, considerando (2.1),

$$\begin{aligned} \hat{h}_0(t) &= \frac{d_i}{\sum_{j\in R_i}\exp(\boldsymbol{z}_j\hat{\boldsymbol{\beta}})} \\ \hat{H}_0 &= \sum_{t_{(i)}\le t}\hat{h}_0(t_{(i)}) \\ \hat{S}_o &= \prod_{t_{(i)}\le t}\left\lfloor \exp\left(\frac{-d_i}{\sum_{j\in R_i}\exp(\boldsymbol{z}_j\hat{\boldsymbol{\beta}})}\right)\right\rfloor \end{aligned}$$

No R, para obter a estimativa de $S(t)$, é possível usar `survfit` como na estimativa de Kaplan-Meier, fornecendo o objeto retornado por `coxph` em vez de usar uma fórmula (por detrás, é usado a função `survfit.coxph`), que pode ser representada graficamente como na Figura 2.6:

```
> fit_lung <- survfit(cox_lung)
> plot(fit_lung)
```

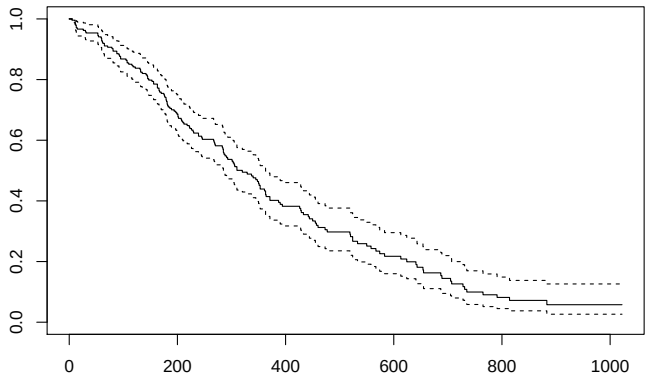

Figura 2.6: Estimativa da função de sobrevivência obtida através do modelo de Cox, obtido do ficheiro de dados `lung` do pacote `survival`.

2.5.1 Estudo dos resíduos do modelo de Cox

Como em qualquer modelo de regressão, uma forma possível para averiguar se o modelo de Cox é adequado aos dados consiste em analisar os resíduos.

Existem vários resíduos que podem ser de uso, como o do indicador de censura d_i, que se calcula da seguinte maneira, se nenhuma das covariáveis depende do tempo e a censura é exclusivamente à direita:

$$m_i = \delta_i - \hat{H}_0(t_i) \exp(\boldsymbol{z}_i \hat{\boldsymbol{\beta}})$$

Estes resíduos são também denominados como resíduos martingala, sendo usados para avaliar a qualidade do ajustamento.

Para verificar a proporcionalidade das funções de risco, usa-se os resíduos de Schoenfeld, que são baseadas na derivada de (2.14):

$$\hat{\boldsymbol{r}}_i = \boldsymbol{z}_i - \sum_{j \in R_i} \left(\boldsymbol{z}_j \frac{\exp(\boldsymbol{z}_j \hat{\boldsymbol{\beta}})}{\sum_{k \in R_i} \exp(\boldsymbol{z}_k \hat{\boldsymbol{\beta}})} \right) = \boldsymbol{z}_i - \bar{\boldsymbol{z}}(t_i)$$

Se for feita uma transformação para escalar com a estimativa da sua variância, isto é,

$$\boldsymbol{r}_i^* = \boldsymbol{r}_i \cdot \sum d_i \cdot \mathrm{var}(\hat{\boldsymbol{\beta}})$$

obtêm-se os resíduos de Schoenfeld padronizados.

Podemos usar a função `cox.zph` para obter estes últimos resíduos e fazer a sua análise visual, assim:

```
> plot(cox.zph(cox_lung))
```

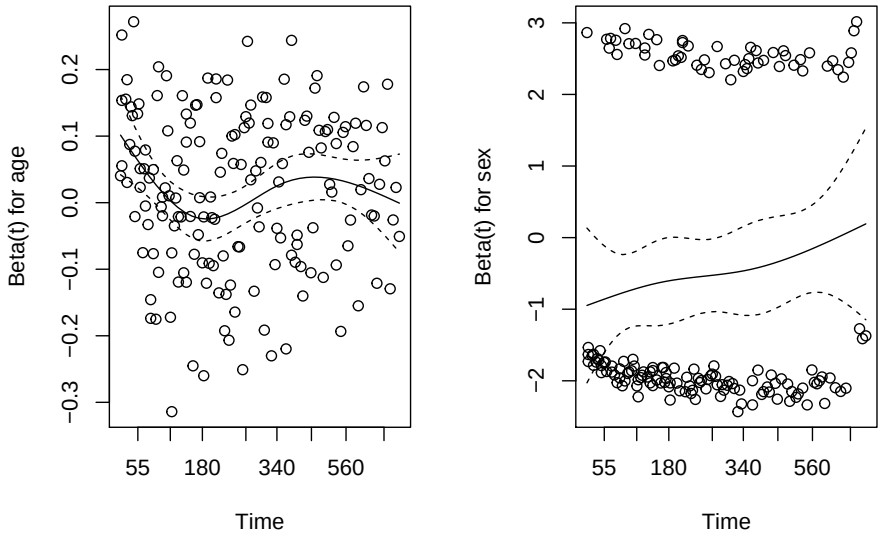

Figura 2.7: Gráficos dos resíduos de Schoenfeld padronizados, obtidos do modelo de Cox criado do ficheiro de dados `lung` no pacote `survival`.

As linhas sólidas presentes nos gráficos da Figura 2.7 são as curvas de suavização *spline* dos resíduos e as linhas a tracejado os limites dos respetivos intervalos de confiança.

Estes gráficos facilitam a análise visual, uma vez que os resíduos se distribuem aleatoriamente em torno do zero quando os riscos são proporcionais (como é o caso no gráfico do lado esquerdo). Note-se que no caso da variável `sex`, por ter duas categorias, os correspondentes resíduos dispõem-se ao longo de duas faixas.

Chama-se a atenção que na aplicação gráfica R GUI, que é o interpretador interativo por norma do R nos sistemas operativos Windows e Mac OS X, quando mais do que um gráfico (com `plot`) é dado como *output* pela mesma função só o último gráfico fica visível, pois quaisquer gráficos anteriores são rapidamente substituídos em sequência. Isto acontece com o `cox.zph` quando

existe mais do que uma covariável, como no exemplo anterior: o gráfico para `age` é apagado só ficando visível o *output* do gráfico para a covariável `sex`.

Para conseguir visualizar individualmente o gráfico da covariável `age`, a alternativa mais simples consiste em só fazer `plot` dessa covariável, ou seja:

```
> plot(cox.zph(cox_lung)[1])
```

Contudo, a longo prazo, o método mais aconselhado é, ou abrir uma janela gráfica separada com `x11()`, ou ir a **Histórico → Recordar...**, para que possa usar `PgUp` e `PgDn` no seu teclado para percorrer os gráficos criados nessa sessão.

Capítulo 3

Um novo pacote no R

O cerne deste trabalho envolve o desenvolvimento do pacote `vsd` para o R sobre a versão 4.0.3, sendo o seu nome um acrónimo para *Visualizing Survival Data*. Esta ideia, apresentada numa proposta de tese, teve como fonte de inspiração o pacote `vcd` (Visualizing Categorical Data) [23]. Assim, o objetivo deste pacote é permitir a fácil criação de um conjunto de gráficos relevantes aos dados/modelos específicos à Análise de Sobrevivência.

Para além deste objetivo principal, foram tidos em conta três princípios:

- Que os gráficos com as opções definidas por norma estejam prontos para publicação, mas que permitam uma fácil customização pelo utilizador (como o R em geral);
- Que o pacote seja modular, ou seja, que permita adicionar novas funcionalidades após publicação inicial;
- Que outros pacotes presentes no ecossistema do R relativos à Análise de Sobrevivência sejam usados como suporte ao pacote (não reinventar a roda).

Pretende-se começar com um *slice* horizontal dos temas mais elementares da Análise de Sobrevivência, os descritos no Capítulo 2, já abordados no R no pacote base `survival`, mas acrescentando alguns conceitos de outros pacotes mais especializados.

O pacote `vsd`, por estar no inicio do seu desenvolvimento, ainda não foi publicado no CRAN logo para o utilizar localmente terá de o instalar do repositório Github onde este está publicado através de:

```
> install.packages("devtools")
> devtools::install_github("cyanzule/vsd")
> library(vsd)
```

Em sistemas Windows isto pode também requerer a instalação das ferramentas `Rtools`.

3.1 Sintaxe da função `vsd`

O pacote `vsd` exportará uma única função[1], com o mesmo nome do pacote, com a seguinte sintaxe:

```
> vsd(model, data, .interactive, .include, .arguments, ...)
```

model Ou um modelo dos dados de Análise de Sobrevivência (por exemplo, `survfit.object`), ou uma estrutura de dados do R a partir da qual se constrói um modelo (p.e., um objeto `Surv`);

data Estrutura de dados utilizada para a criação do `model`, a qual, se não for fornecida, será extraída do modelo (se possível);

.interactive Valor lógico que indica se o utilizador deseja ver os gráficos interativamente (explicado em maior detalhe na Secção 3.7), sendo por norma `FALSE`;

.include Lista de *strings* dos tipos de gráficos desejados, tomando por norma todos os gráficos relevantes ao modelo fornecido;

.arguments Lista de lista de argumentos para passar especificamente a cada um dos tipos de gráficos, sendo uma lista vazia por norma;

... Argumentos para serem passados, por norma, a todos os tipos de gráficos gerados pela função.

Embora este protótipo seja extensivo, os únicos argumentos indispensáveis para utilizar esta função são o `model` e o `data`, este último podendo até ser desnecessário já que o `vsd` tenta extrair os dados usados para criar um modelo (mas recomenda-se passar).

No argumento `model` pode-se então utilizar um modelo de sobrevivência dado pelas funções `survfit` (mais detalhes em `?survfit.object`), `coxph` e `flexsurv` (mais detalhes na Secção 3.6), ou potenciais modelos como:

- Um objeto `Surv`, como `Surv(time, status)`;

[1] Na verdade, é um conjunto de funções que, consoante o tipo de modelo disponibilizado, através do sistema de classes de objetos S3 é escolhido o processamento correto do modelo.

- Uma fórmula contendo um objeto `Surv`, como `Surv(time, status) ~ x`.

onde estes são empacotados dentro de um modelo `survfit` (Kaplan-Meier) com opções por norma estimadas pelo R.

Por razões de processamento, objetos `coxph` também são empacotados num `survfit.object` para obtenção da estimativa da função de sobrevivência, mesmo que estes já sejam um modelo de sobrevivência.

O argumento `.include` é uma lista de *strings* dos tipos de gráficos desejados, que, à data, são:

- "fit": o gráfico da estimativa da(s) função(ões) de sobrevivência, especificamente para modelos não paramétricos (de momento, apenas utilizando a estimativa de Kaplan-Meier);
- "parametric": o gráfico da estimativa da(s) função(ões) de sobrevivência associada(s) a um modelo paramétrico, onde os parâmetros são estimados segundo a definição do objeto `flexsurv`;
- "forest": os gráficos de floresta das covariáveis consideradas no modelo de Cox;
- "residuals": os resíduos das covariáveis consideradas no modelo de Cox;
- "hazard": o gráfico da estimativa suavizada da(s) função(ões) de risco.

Por norma, todos os gráficos possíveis num modelo são exportados, isto é, mesmo que `"forest"` esteja especificado como um *output*, este não será criado a não ser num modelo Cox. A lista de *strings* não tem de ter o nome do tipo de gráfico por completo, sendo o nome auto-completado se possível, ou seja:

```
> vsd(model, data, .include = c("fo", "haz"))
```

retornará apenas os gráficos de floresta e da função de risco, se o modelo o permitir.

Os argumentos `.arguments` e `...` (*ellipsis*, a sintaxe no R para passar argumentos extras que não estejam diretamente definidos na definição da função) requer alguma explicação extra. Se um utilizador pretende passar argumentos ao processamento dos dados e à criação dos gráficos que não podem ser alterados (facilmente) posteriormente à criação do gráfico, estes dois argumentos da função `vsd` estão disponíveis para tal mecanismo, onde argumentos presentes dentro de `.arguments` têm prioridade sobre os argumentos ditos gerais (estes últimos passados a todos os gráficos).

Por exemplo, a seguinte função:

```
> vsd(model, data,
      .arguments = list(fit = (size = 3, xlab = "Dias")),
      xlab = "Tempo")
```

fará com que a legenda do eixo das abcissas seja "Tempo" em vez do que está instituído por norma, "Time", mas no caso da estimação não-paramétrica, essa legenda será "Dias", para além de que a linha será mais grossa (por norma, é `1`).

Argumentos não relevantes, tanto gerais como no específico no argumento `.arguments` não são tidos em conta, onde o pacote `vsd` compara estes com uma lista de argumentos válidos para cada um dos tipos de gráficos. Estes argumentos padronizados estão listados na documentação da função, podendo ser explorados com `?vsd`.

A função retorna uma lista, indexada pelo tipo de gráfico, onde cada índice é, ou um gráfico individual, ou uma lista de gráficos, consoante o número de gráficos relevantes a esse tipo.

3.2 Gráficos personalizáveis com o `ggplot2`

Para desenvolver um pacote com foco gráfico, é necessário verificar que ferramentas e extensões (os pacotes) o R oferece para a criação dos mesmos e quais serão os mais adequados aos objetivos do pacote em desenvolvimento.

Os gráficos apresentados anteriormente neste trabalho foram todos obtidos com `plot()`, tendo isto sido possível devido à funcionalidade gráfica embebida no pacote `survival` para a criação de gráficos através dos pacotes base e sistema de gráficos `graphics` do R [13].

Estes gráficos preliminares oferecem algumas desvantagens, como sejam:

- O seu estilo gráfico, embora razoável, requer *setup* pelo utilizador para ficar apresentável, especialmente se o gráfico tiver múltiplas curvas, legendas, sub-gráficos ou *layouts* mais complexos;

- Mais, o estilo gráfico por norma depende da implementação pelo pacote que está a ser usado, podendo levar à diferença nos estilos como verificado entre as Figuras 2.5 e 2.1, com os pacotes `survival` e `flexsurv`, respetivamente;

- Os gráficos são exportados como imagens imediatamente após a sua criação, não podendo ser reutilizados, ou seja, só é possível repetir a série de funções que levou à sua existência;

- Embora seja possível alterar os gráficos após a sua criação, em `graphics` isto reduz-se a acrescentar novos elementos, o que limita as possibilidades de customização.

Isto não implica que os gráficos base do R sejam de pouco utilidade para o utilizador, muito pelo contrário, tendo em conta a facilidade com que são criados e a extensa compatibilidade com outros pacotes. Porém, existe uma extensa gama de outras implementações gráficas no R, incluindo outros sistemas distintos de pacotes gráficos como os baseados no sistema de gráficos `grid` [15].

Para este projeto, decidiu-se utilizar o pacote `ggplot2` [24]. Este oferece quase uma gramática de gráficos, onde elementos de um gráfico (p.e., dados, escala, estrutura, tema) são somados, com o operador de soma do R, a um objeto gráfico, que o utilizador pode guardar para extensão posterior, ou mostrar com um `print()`.

Como exemplo, compare-se os dois blocos de código que se seguem, os quais dão origem, respetivamente, às Figuras 3.1 e 3.2:

```
> library(survival)
> fit_aml <- survfit(Surv(time, status) ~ x, data = aml, conf.
    type = "log-log")
> plot(fit_aml, xlab = "time", ylab="estimate", lty = 1:2,
    mark.time = T)
> legend("topright", inset = .02 , legend=levels(aml$x), box.
    lty = 0, lty=1:2)
```

```
> install.packages("ggplot2", "broom", "tibble")
> library(ggplot2)
> tidy_fit_aml <- tibble::add_row(broom::tidy(fit_aml), time =
    0, estimate = 1, n.censor = 0, strata = c("x=Maintained",
    "x=Nonmaintained"), .before = 0)
> gridplot_aml <- ggplot(tidy_fit_aml, aes(time, estimate,
    group = strata)) + geom_step(aes(linetype = strata)) +
    theme(legend.position = "top")
> print(gridplot_aml)
```

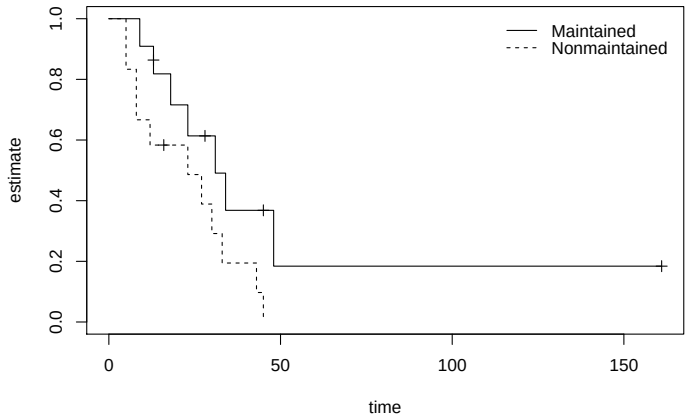

Figura 3.1: Gráfico obtido pelo pacote gráfico `graphics` para as estimativas de Kaplan-Meier da função de sobrevivência para as categorias de `x`, do ficheiro de dados `aml` do pacote `survival`.

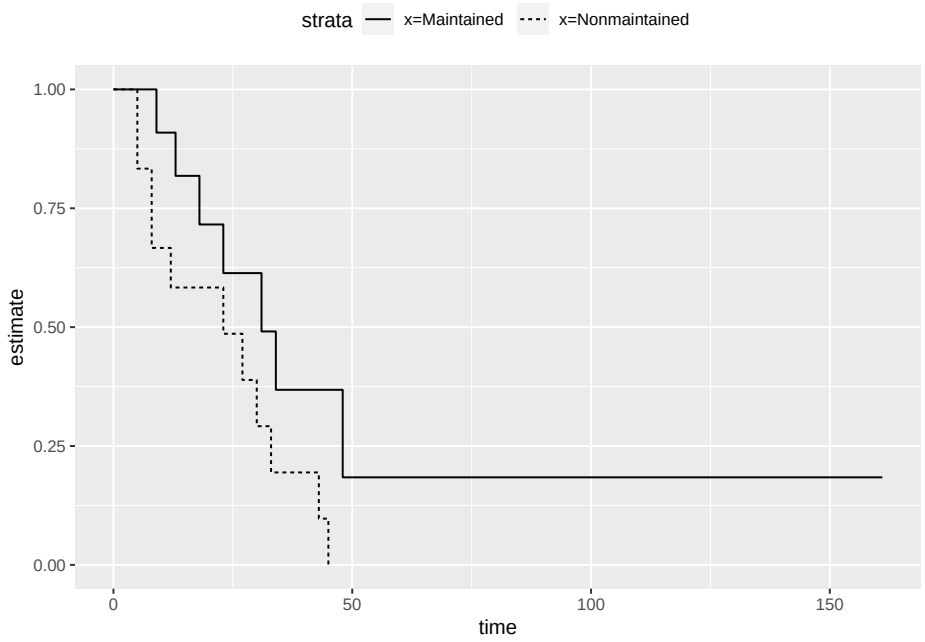

Figura 3.2: Gráfico obtido pelo pacote gráfico `ggplot2`, após pré-processamento da estrutura de dados, para as estimativas de Kaplan-Meier da função de sobrevivência para as categorias de `x`, do ficheiro de dados `aml` no pacote `survival`.

Note-se que a Figura 3.1 é semelhante à Figura 2.3, mas em que se acrescentou uma legenda para distinguir as categorias da covariável `x`, legendas nos eixos e os símbolos para as observações censuradas.

Através do código, pode-se comprovar como a criação de um gráfico no pacote `ggplot2` consiste no acrescento funcional das componentes do gráfico: começa-se pela estrutura de dados e declarações das variáveis, depois passa-se ao estilo da visualização dos gráficos e acaba-se em detalhes estilísticos como a posição do legenda. No `ggplot2` há ainda muito mais possibilidades de customização embora, em geral, não seja necessário pois os valores considerados por norma já são adequados.

No entanto, é visível a complexidade extra necessária para a construção do gráfico na Figura 3.2. Este facto mostra a grande fraqueza do ambiente gráfico `ggplot2` neste projeto: não possui comandos inatos em como processar estruturas de dados no contexto da Análise de Sobrevivência, ao contrário das programadas no `plot.survfit` pelo pacote `survival`. Por esta razão, é necessário o uso de código adicional, como o uso do pacote `broom`, para reformatar os dados e ajustes adicionais, como acrescentar o ponto temporal 0 na estimativa em todas as covariáveis com o pacote `tibble`.

Porém esta complexidade é de menor importância comparada à fácil extensão dos gráficos após a sua criação. No pacote `vsd`, a tarefa de lidar com esta complexidade fica a cargo do programador, pelo que o utilizador nem se apercebe dela. Afinal, mesmo após a aparente finalização de um gráfico, se se quiser acrescentar texto, ou um tema externo pronto para publicação, ou até componentes adicionais (como os símbolos de censura em falta neste exemplo), basta somar estas componentes, como se exemplifica de seguida obtendo-se a Figura 3.3:

```
> install.packages("ggpubr")
> gridplot_aml + labs(title = "Kaplan-Meier survival
    estimation for Acute Myelogenous Leukemia", subtitle = "
    With/without extended (maintained) chemotherapy", caption =
     "From data file survival::aml") + geom_point(data = subset
    (tidy_fit_aml, n.censor > 0), shape = "+", size = 6) +
    ggpubr::theme_pubr()
```

Note-se que existe já um pacote no CRAN, `survminer` [8], que usa outros pacotes (como o `broom`, referenciado anteriormente, e o `survMisc`) para a criação de gráficos através do `ggplot2` com vários tipos de dados de sobrevivência, tratando da parte do processamento prévio dos dados. Este é usado como dependência do pacote `vsd`.

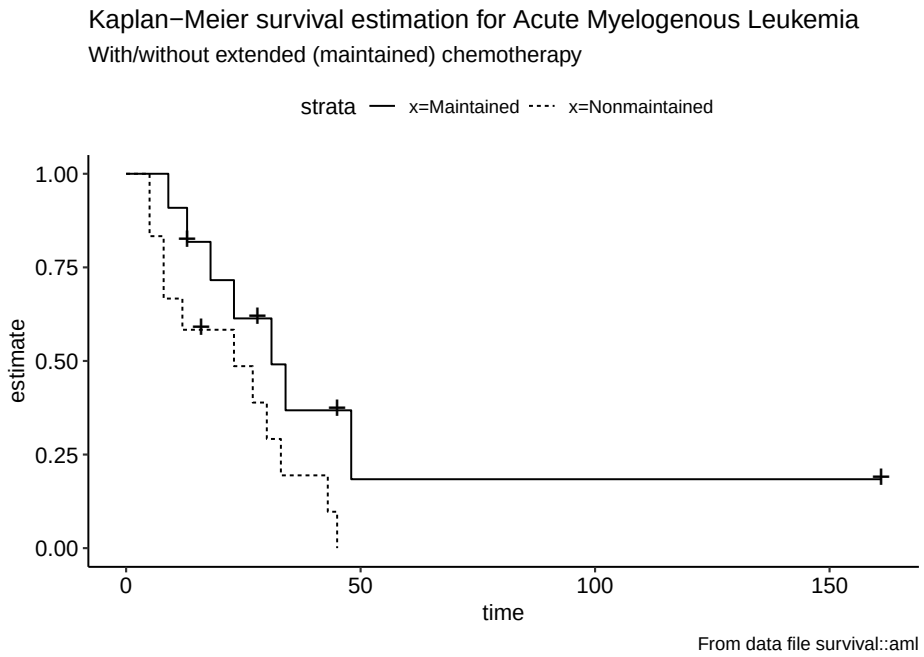

Figura 3.3: Gráfico obtido após adicionar componentes ao presente na Figura 3.2.

3.3 Gráficos da estimativa de Kaplan-Meier da função de sobrevivência

Em termos de gráficos, o pacote `survminer` faz o devido pré-processamento, apenas sendo internamente necessário passar as opções (filtradas) juntamente com o modelo. Este pacote tenta também extrair os dados do modelo se estes não forem fornecidos, mas como a função `vsd` necessita destes para o processamento dos outros tipos de gráficos, o `survminer` nunca é chamado se a função `vsd` não conseguir extrair os dados bem formatados.

Porém, o pacote `vsd`, no espírito do R e nos objetivos deste projeto, escolhe algumas opções, por norma, ao contrário das esperadas ao `survminer`, incluindo a escolha de um tema mais próprio para publicação.

Os três comandos seguintes são equivalentes, mostrando como o pacote `vsd` reformula objetos sem modelos num modelo (simples) adequado ao seu processamento. Qualquer um deles origina a Figura 3.4:

```
> with(ovarian, vsd(Surv(futime, fustat)))$fit
```

```
> vsd(Surv(futime, fustat) ~ 1, ovarian)$fit
```

```
> vsd(survfit(Surv(futime, fustat) ~ 1, data = ovarian))$fit
```

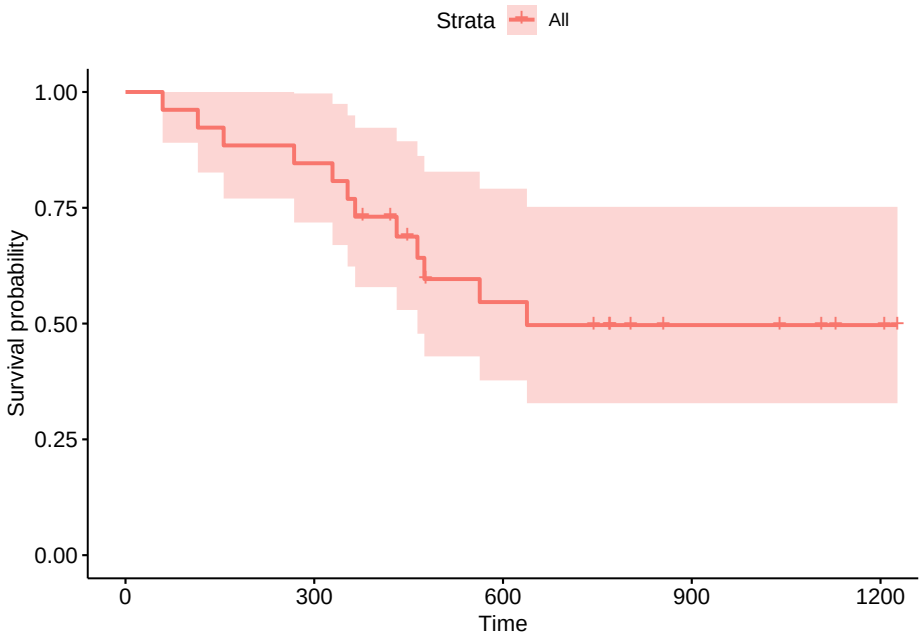

Figura 3.4: Gráfico da estimativa de Kaplan-Meier da função de sobrevivência do modelo obtido do ficheiro de dados `ovarian` do pacote `survival`.

Por regra, a função `vsd` inclui o intervalo de confiança e os símbolos relativos à censura nos gráficos da estimativa de Kaplan-Meier.

O `survminer` também não apresenta problemas com modelos com covariáveis, embora, por norma, esconde os limites dos intervalos de confiança quando existe mais do que uma curva, como presente no lado esquerdo da Figura 3.5:

```
> vsd(survfit(Surv(futime, fustat) ~ rx, data = ovarian))$fit
> vsd(survfit(Surv(futime, fustat) ~ rx, data = ovarian), conf
    .int = T)$fit
```

3.4 Gráficos específicos ao modelo Cox

A função `vsd`, na presença de um objeto com classe `survfitcox`, ou seja, quando um modelo criado pela função `survfit` provém de um modelo de Cox (`coxph`), acrescenta ao *output* mais dois tipos de gráficos, para além

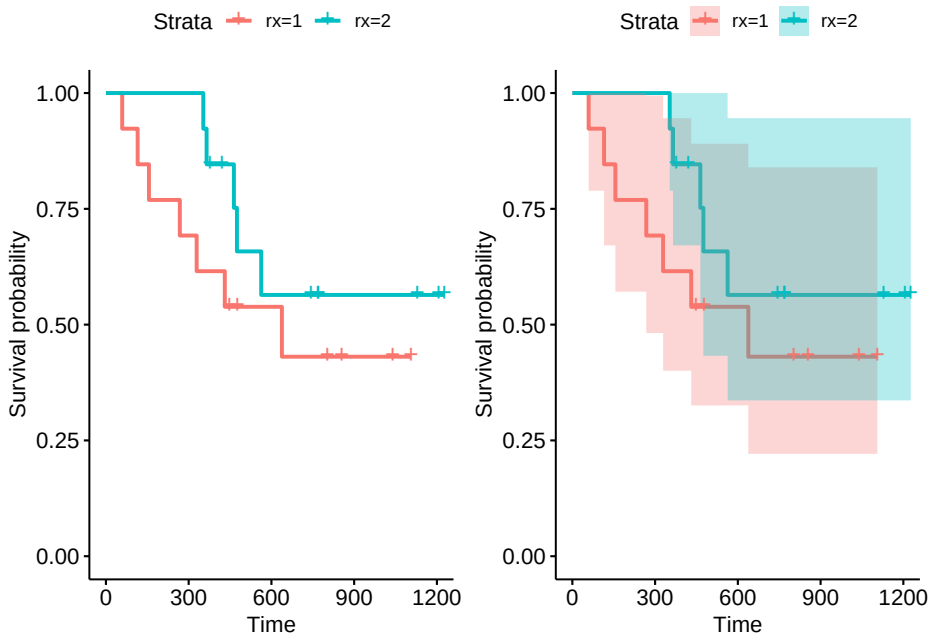

Figura 3.5: Gráficos da estimativa de Kaplan-Meier da função de sobrevivência do modelo obtido do ficheiro de dados `ovarian` do pacote `survival` estratificado pela covariável `rx`, sem (à esquerda) e com (à direita) os intervalos de confiança.

dos criados após processamento de um modelo `survfit`: o(s) gráfico(s) para os resíduos de Schoenfeld, e o(s) gráfico(s) de floresta (que apresenta(m) graficamente as estimativas dos coeficientes das covariáveis).

Os resíduos de Schoenfeld padronizados são obtidos diretamente do pacote `survminer` que, por sua vez, usa a função `plot.cox.zph` do pacote `survival`.

```
> lung_residuals <- vsd(coxph(Surv(time, status) ~ age + sex,
    lung))$residuals
> print(lung_residuals)
```

Os gráficos de resíduos, os da Figura 3.6, são retornados todos na mesma página, através de uma lista de gráficos `ggplot2`, onde cada um pode ser acedido isoladamente com um índice numérico, ou seja:

```
> lung_residuals[[1]]
> lung_residuals[[2]]
```

para obter os gráficos individuais referentes às covariáveis `age` e `sex` respetivamente. Note-se a ausência dos intervalos de confiança, ao contrário dos

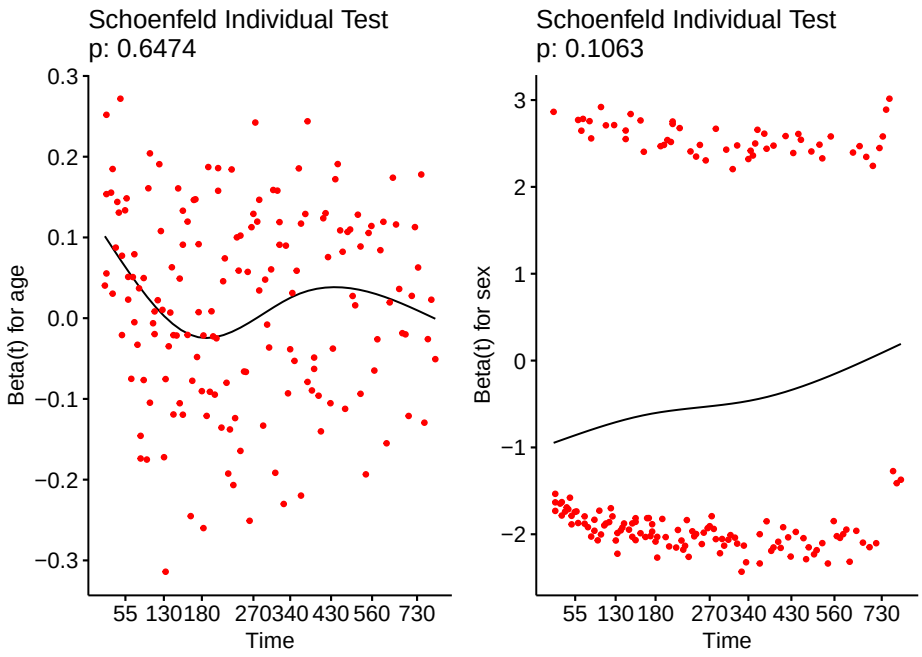

Figura 3.6: Gráficos dos resíduos de Schoenfeld padronizados do modelo Cox, por covariável, obtido do ficheiro de dados `lung` do pacote `survival` através da função `vsd`.

gráficos obtidos na Secção 2.5.1. Esta opção deve-se à existência de um *bug* no pacote `survminer` que, à data, não está a calcular adequadamente estes intervalos de confiança. Por esta razão, no pacote `vsd`, não são representados por norma.

Os gráficos de floresta provêm da função `survminer::ggforest`, que requer o modelo `coxph` diretamente, mas que o `vsd` extrai do modelo dado, e retorna um gráfico `ggplot2` com um estilo único, como visualizado na Figura 3.7:

```
> vsd(survfit(coxph(Surv(time, status) ~ sex + rx + adhere,
    colon)))$forest
```

Porém, se for usada pelo menos uma das covariáveis para estratificar os dados, aparecem problemas adicionais:

```
> survminer::ggforest(coxph(Surv(time, status) ~ sex + strata(
    rx) + adhere, colon)
```

```
Error in `[.data.frame`(data, , var) : undefined columns
    selected
```

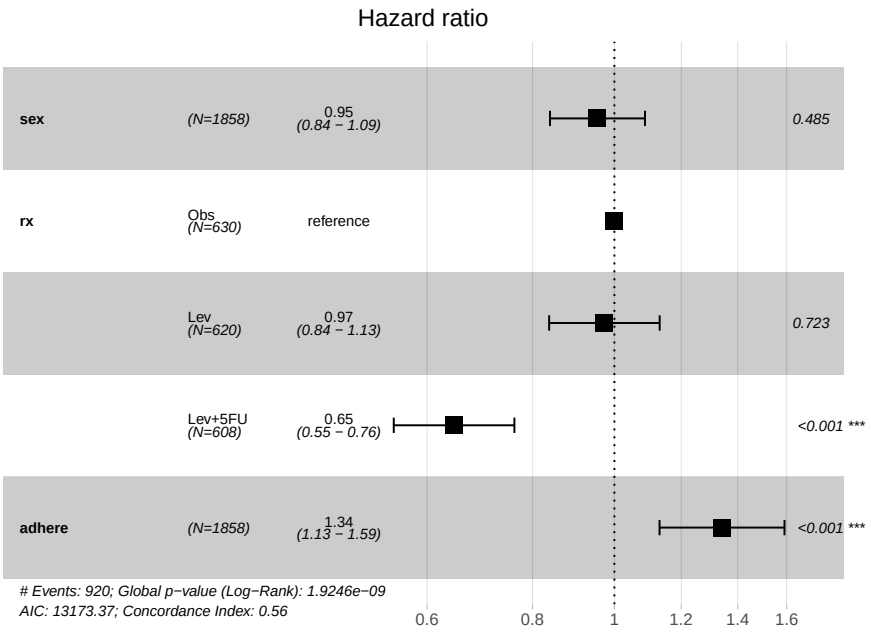

Figura 3.7: Gráfico de floresta das estimativas dos coeficientes para o modelo de Cox baseado no ficheiro de dados `colon` do pacote `survival`.

Para evitar este erro e para que o utilizador consiga obter os gráficos pretendidos, a função `vsd` usa o `strata()` para fazer subconjuntos dos dados (como no `survfit`), criando gráficos de floresta separados, um para a globalidade dos dados e um por nível da(s) covariável(is) problemática(s). Assim, dá origem à Figura 3.8:

```
> vsd(coxph(Surv(time, status) ~ sex + strata(rx) + adhere,
    colon))$forest
```

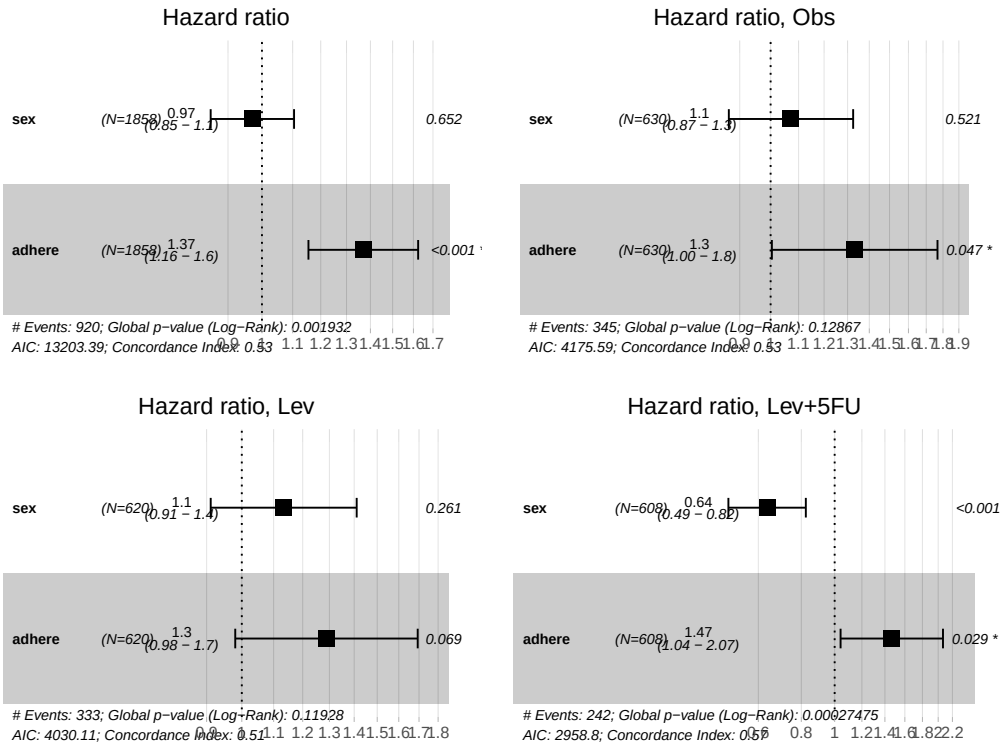

Figura 3.8: Os quatro gráficos criados pela função `vsd` quando a covariável `strata(rx)` é usada no modelo de Cox criado no ficheiro de dados `colon` do pacote `survival`: para os dados na globalidade e para cada um dos três níveis da variável (Obs, Lev, Lev+5FU).

3.5 Função de risco estimada e `muhaz`

Quando a função de risco é estimada a partir de um conjunto de dados com censura à direita, a função `vsd` tenta criar também um gráfico `ggplot2` com a respetiva estimativa, através da função `muhaz`.

Como a função `muhaz` não tem em conta as covariáveis de um modelo (apenas usando uma estrutura equivalente ao `Surv` para estimar a função de risco como nos gráficos de floresta na presença de estratificação), os dados são separados pelos níveis definidos, e usados separadamente para obter as estimativas da função de risco, as quais são apresentadas num gráfico `ggplot2`, como demonstrado na Figura 3.9.

```
> vsd(coxph(Surv(time, status) ~ sex + strata(rx) + adhere,
    colon))$hazard
```

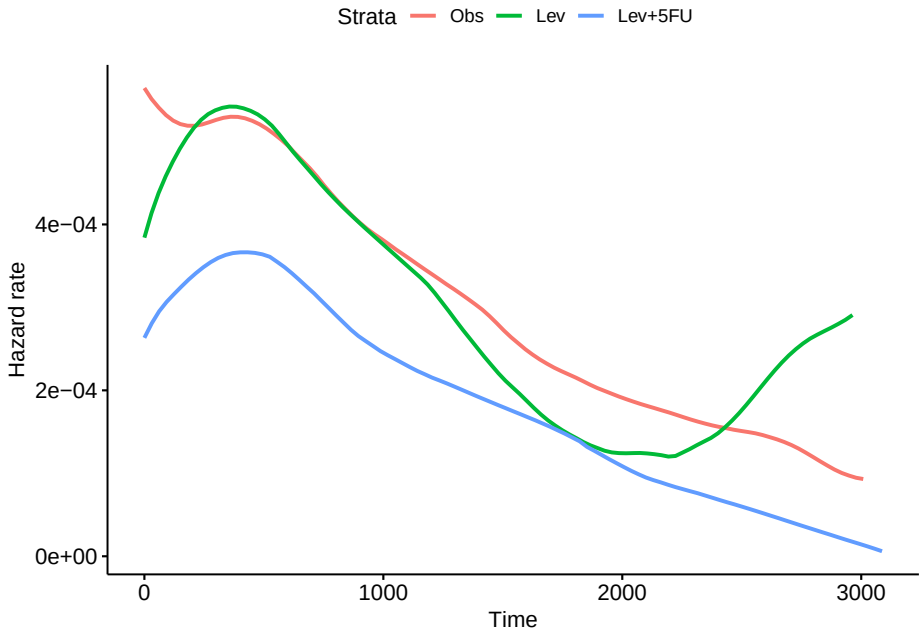

Figura 3.9: Gráfico da estimativas das funções de risco para o modelo de Cox para o ficheiro de dados `colon` do pacote `survival`, estratificado pela variável `rx`.

Como indicado na Secção 2.4.1, o `muhaz` estima internamente o intervalo de tempo a usar para o cálculo da estimativa da função de risco, estando por norma definida entre o instante 0 e o instante em que ainda restam dez pacientes em risco. Para além deste último instante, a função `vsd` ignora esse nível para que o gráfico continue a ser criado para os restantes níveis, informando o utilizador desta lacuna:

```
> vsd(Surv(time, status) ~ ph.ecog, data = lung)$hazard
```

```
Warning: Level 3 doesn't have enough datapoints to estimate
    the hazard function
```

Note-se que, na Figura 3.10, a variável `ph.ecog` possui quatro níveis, 0-3, mas como indicado no aviso o terceiro nível não tem observações suficientes para o cálculo da estimativa e é descartado. Além disso, pelo mesmo motivo verifica-se que as estimativas dos níveis presentes cessam em diferentes instantes.

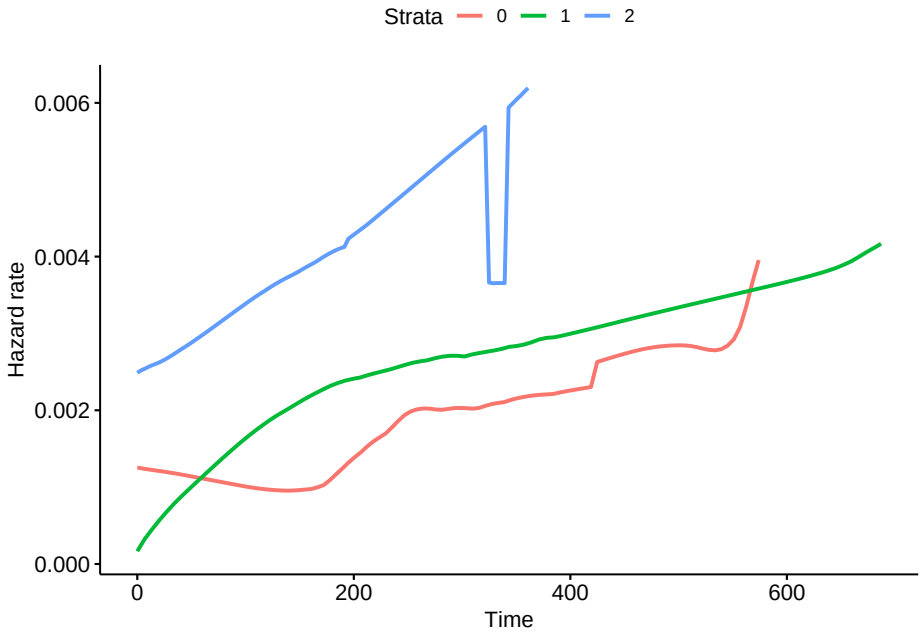

Figura 3.10: Gráfico da estimativa da função de risco do ficheiro de dados `lung` no pacote `survival`, delineada pela covariável `ph.ecog`.

3.6 Modelos paramétricos

O pacote `vsd` também permite usar modelos paramétricos, através dos objetos obtidos pela função `flexsurvreg`, tanto para apresentar o gráficos da estimativa da função de sobrevivência relativa ao modelo paramétrico usado, como o da correspondente estimativa não-paramétrica de Kaplan-Meier, para efeitos de comparação.

Assim, para criar gráficos relativos a modelos paramétricos é usada a função `survminer::ggflexsurvplot`, contrastando-se o gráfico obtido na Figura 3.11 com o presente na Figura 2.1:

```
> library(flexsurv)
> vsd(flexsurvreg(Surv(futime, fustat) ~ 1, data = ovarian,
    dist = "weibull"))$parametric
```

No entanto, à data, na presença de covariáveis no modelo paramétrico, o pacote `survminer` não insere a estimativa de Kaplan-Meier uma vez que ignora os níveis das covariáveis. O código que se segue, que origina o gráfico da Figura 3.12, exemplifica este aspeto.

```
> survminer::ggflexsurvplot(flexsurvreg(Surv(rectime, censrec)
    ~ group, data = bc, dist = "gengamma"))
```

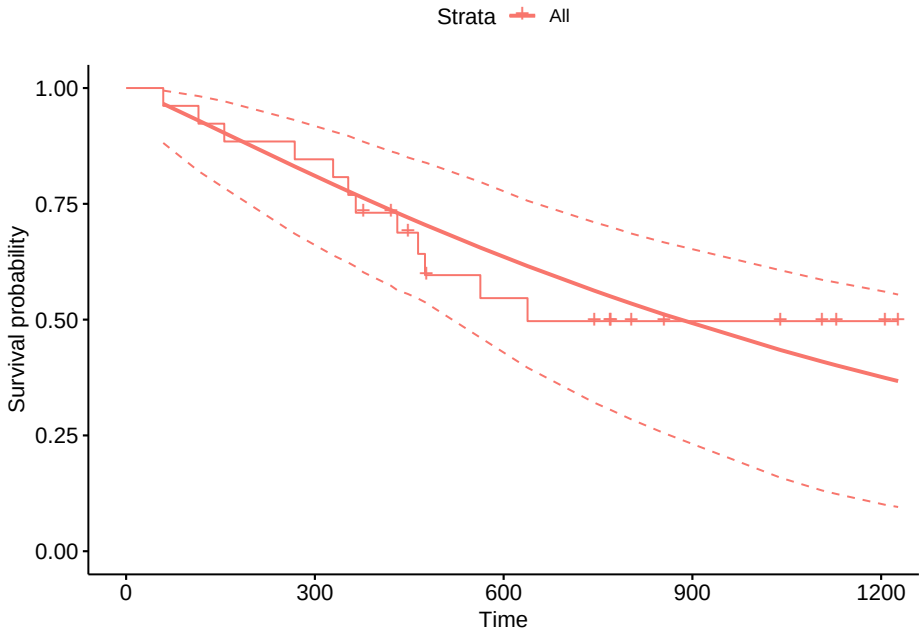

Figura 3.11: Estimativa de Kaplan-Meier da função de sobrevivência e estimativa da função de sobrevivência obtida através do modelo Weibull dos dados do ficheiro `ovarian` do pacote `survival` pela função `vsd`.

Para evitar essa lacuna, a função `vsd` reconstrói o gráfico do modelo paramétrico, e depois acrescenta a estimativa de Kaplan-Meier manualmente, corrigindo esta gralha e simplificando a tarefa do utilizador. Com esta correção, obtêm-se o gráfico adequado como se observa na Figura 3.13:

```
> vsd(flexsurvreg(Surv(rectime, censrec) ~ group, data = bc,
    dist = "gengamma"))$parametric
```

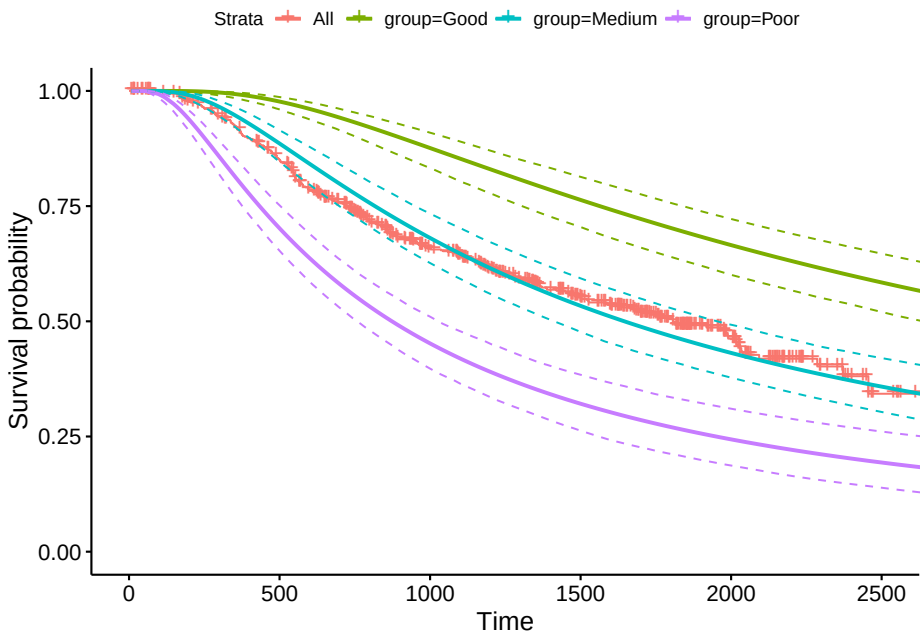

Figura 3.12: Gráfico criado pelo pacote `survminer` no ficheiro de dados `bc` do pacote `flexsurvreg` para a estimativa do modelo Gengamma mais adequado.

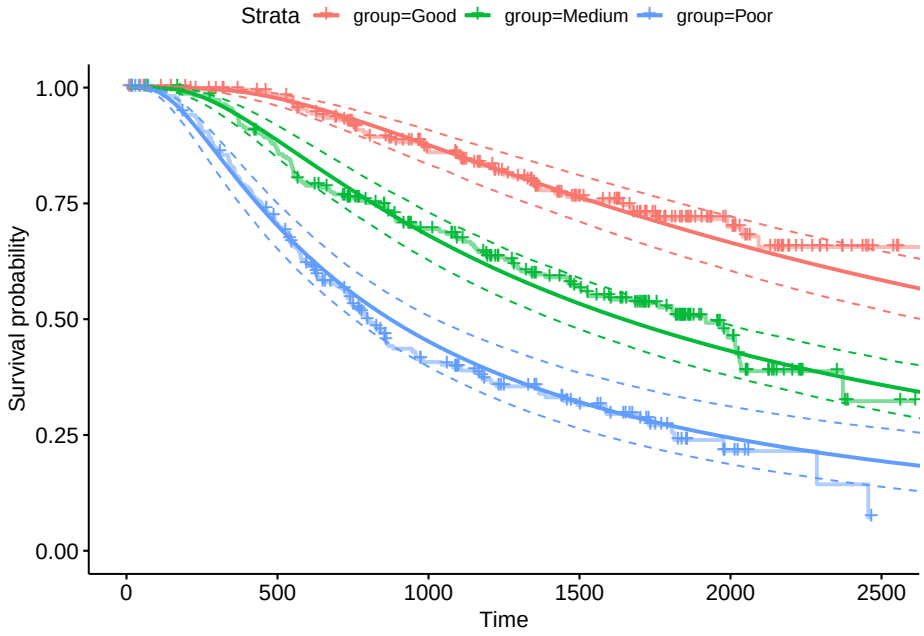

Figura 3.13: Gráfico criado pelo `vsd` no ficheiro de dados `bc` do pacote `flexsurvreg` para a estimativa do modelo Gengamma mais adequado.

3.7 Modo interativo e `plotly`

Até agora todos os exemplos têm consistido em delinear um tipo de gráfico específico. O *output* da função `vsd` é uma lista de gráficos logo, se for apenas apresentado e não guardado, todos os gráficos criados são apresentados um a seguir ao outro, apresentando o mesmo problema do que o ocorrido na Secção 2.5.1 (quando a função `cox.ph` originava mais do que um gráfico de resíduos).

Os dois códigos seguintes levariam, cada um deles, à obtenção de vários gráficos consecutivos:

```
> vsd(coxph(Surv(time, status) ~ sex + strata(rx) + adhere,
    colon))
```

```
$fit

$forest
$all

$strata
$strata$Obs

$strata$Lev

$strata$`Lev+5FU`


attr(,"class")
[1] "vsdstrata"

$residuals

$hazard
```

```
> vsd(flexsurvreg(Surv(rectime, censrec) ~ group, data = bc,
    dist = "gengamma"))
```

```
$fit

$parametric

$hazard
```

Felizmente, para a visualização manual dos gráficos pelo utilizador, a função `vsd` oferece o argumento `.interactive`. Assim, quando lhe é atribuído o valor `TRUE`, em vez de exportar a lista de gráficos imediatamente, a consola

do R pede o *input* do utilizador, apresentando uma lista de gráficos criados para o modelo dado:

```
> vsd(coxph(Surv(time, status) ~ sex + strata(rx) + adhere,
    colon), .interactive = T)
```

```
Pick a graphic (or 0 to exit)

1: fit
2: forest$all
3: forest$strata.Obs
4: forest$strata.Lev
5: forest$strata.Lev+5FU
6: residuals$1
7: residuals$2
8: hazard

Selection:
```

A lista indica os vários gráficos disponíveis, permitindo ao utilizador escolher um gráfico específico como, por exemplo:

```
Selection: 7
```

Como exemplificado na Figura 3.14, se o pacote `plotly` já estiver instalado no R, a função `vsd` usa este pacote para criar uma janela interativa para explorar o gráfico escolhido, se possível, permitindo, por exemplo, fazer *zoom* numa área específica do gráfico ou ver as coordenadas específicas, como exemplificado na Figura 3.15:

Nem todos os tipos de gráficos (como os de floresta) permitem esta visualização interativa. Neste caso, tal como na ausência do pacote `plotly`, é apresentada ao utilizador uma imagem estática do gráfico escolhido.

Sempre que um gráfico é escolhido, a lista de gráficos é exportada outra vez, e o utilizador pode escolher outro gráfico ou sair deste menu com o *input* 0.

Após o fim do modo interativo, a função `vsd` retorna a lista de gráficos como se o argumento não fosse ativado, ou seja, o comando seguinte é válido e origina a Figura 3.16:

```
> model <- vsd(flexsurvreg(Surv(rectime, censrec) ~ group,
    data = bc, dist = "gengamma"), .interactive = T)$param
Selection: 2
Selection: 1
Selection: 0
> model + theme_minimal()
```

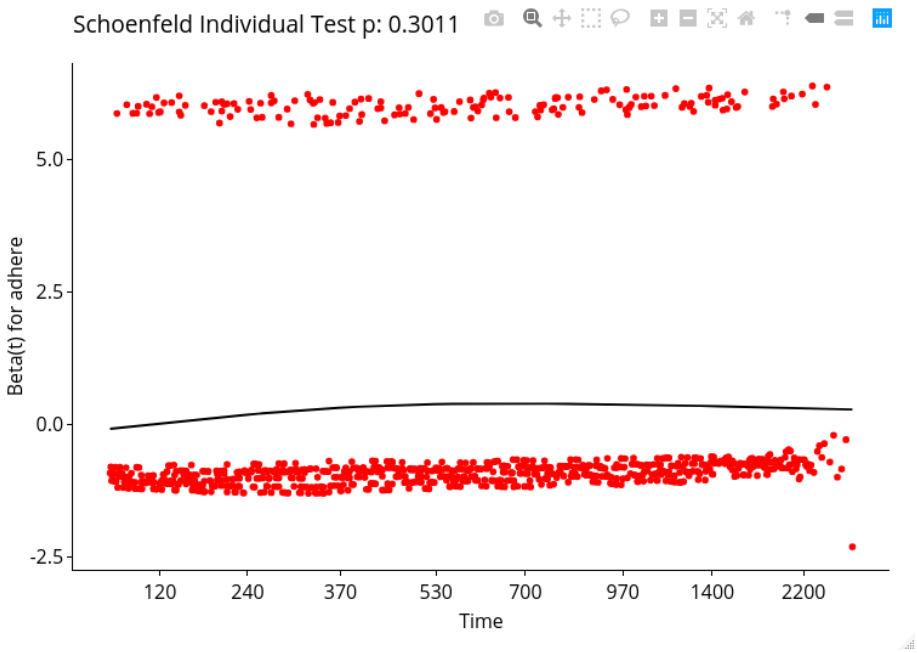

Figura 3.14: O gráfico para os resíduos da covariável `adhere` no modelo de Cox estimado do ficheiro de dados `colon`, obtido do modo interativo da função `vsd` sobre uma visualização interativa oferecido pelo pacote `plotly`.

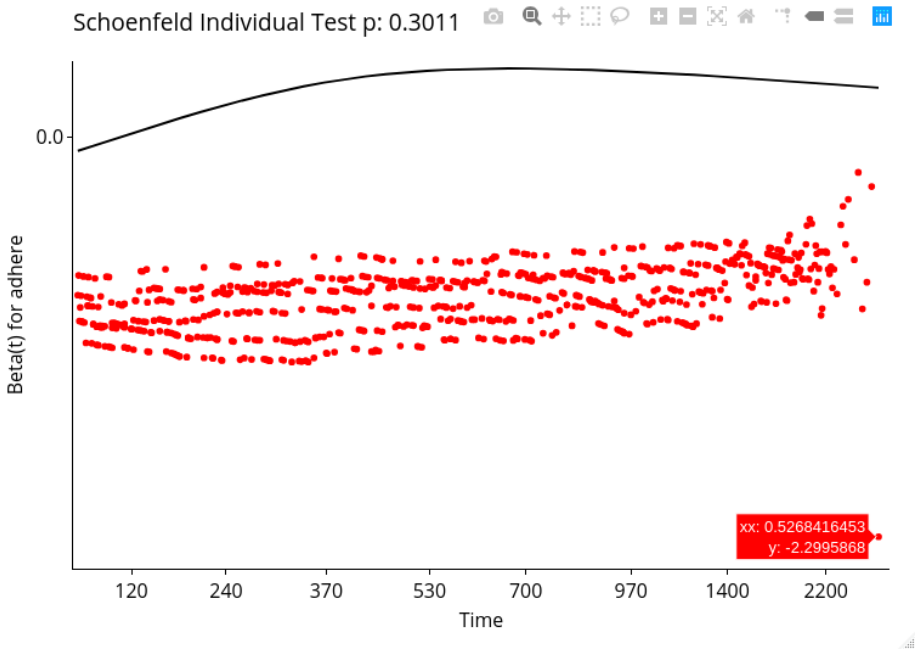

Figura 3.15: Uso interativo do pacote `plotly` na Figura 3.14.

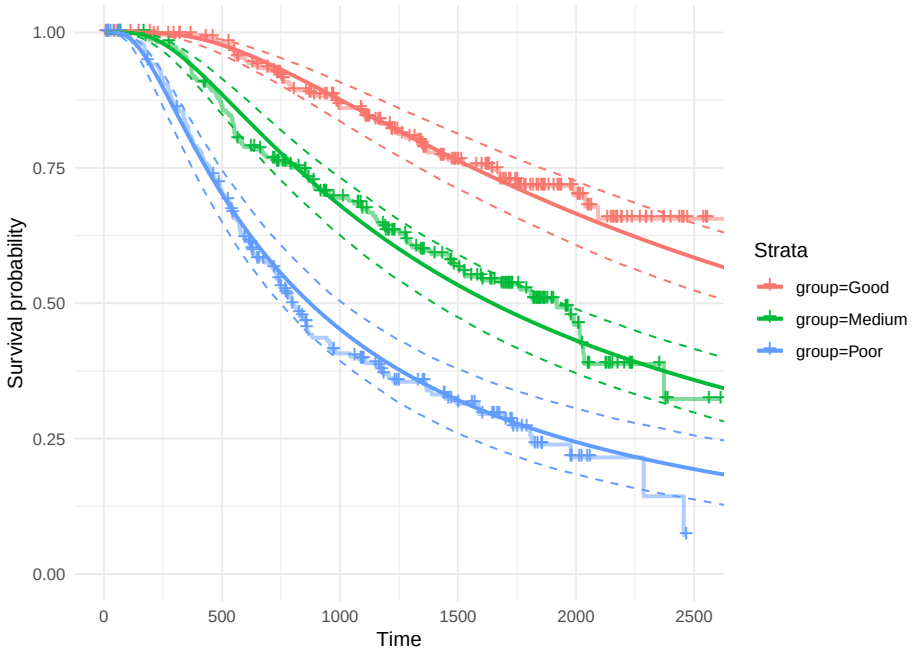

Figura 3.16: Gráfico equivalente ao da Figura 3.13, mas com um tema base do `ggplot2` inserido posteriormente após o modo interativo da função `vsd`.

Capítulo 4

Considerações finais

Este trabalho foi o primeiro projeto de programação de cariz profissional por parte do autor, tendo participado noutros projetos não lançados ou de carácter *hobbyist*, para além de não ter muita experiência na área da Matemática. Por esta razões o pacote `vsd` foi rescrito várias vezes, tendo inicialmente começado a ser desenvolvido num ambiente gráfico diferente.

Não obstante, conseguiu-se um *slice* horizontal dos objetivos propostos para este projeto, havendo ampla margem para expansão e desenvolvimento futuro no âmbito da Análise de Sobrevivência. A natureza modular do programa, onde cada segmento de modelação e criação dos gráficos está separada o quanto possível, facilitará o desenvolvimento de novas funcionalidades. Além disso, o uso da pacote gráfico `ggplot2` permite ao utilizador fazer pós-processamento nos gráficos criados para obter detalhes extras que podem escapar numa ferramenta automatizada.

Um programa informático nunca está realmente acabado, isto é, está em constante construção. Como potenciais pontos a desenvolver no futuro sugere-se, sem ordem específica, as seguintes funcionalidades:

- Suporte mais dinâmico para não-modelos, isto é, na criação e extensão de modelos pelo utilizador (p.e., quando um objeto `Surv` é dado à função `vsd`);
- Acréscimo de componentes aos modelos já abordados neste trabalho, como:
  - Apresentação de gráficos como, por exemplo, da estimativa da função de risco, para situações menos comuns (como sejam a existência de censura à esquerda ou intervalar);
  - Inclusão de outros tipos de modelos não paramétricos;

  - Inclusão de outros modelos de regressão semi-paramétricos (p.e., os abordados no pacote `rms`);
  - Extensão do suporte para modelos paramétricos, através do pacote `flexsurv` ou outros (p.e., `rms`), incluindo até possível sugestão de modelos paramétricos apropriados;

- Suporte para existência de truncatura nos dados;
- Suporte para tipos de modelos na Análise de Sobrevivência não considerados neste trabalho (multivariados, Bayesianos, entre outros);
- Suporte para modelos multimodal, ora em pacotes já envolvidos (como o `survival` e as suas implementações no `survminer`, `flexsurv`), ora noutros pacotes no ecossistema do R;
- Possível retorno de estruturas de dados tratadas, como por exemplo, no processamento dos resíduos, ou o uso de estruturas `tiddle` como retornadas pela função `survminer::ggsurvplot`.
- Uso doutros pacotes gráficos para criação de gráficos, como o `survCurve`;
- Continuar a melhorar as opções por regra para cada um dos modelos e estruturas de dados suportadas.

Todos os pontos anteriores, como indicado individualmente em alguns dos exemplos, podem ser abordados com os pacotes já considerados no pacote `vsd`. No entanto o R possui um ecossistema enorme de pacotes constantemente atualizados [1] que deverão ser suportados e até usados em vez dos pacotes mais usuais se estes apresentarem funcionalidades mais especializadas ou indicadas para os dados em estudo. Destes, cita-se os pacotes `rms`, `prodlim` e o `flexsurv` (este último, embora já seja suportado minimamente no pacote `vsd`, possui muito mais funcionalidades que poderiam ser suportadas e usadas) como os de maior importância para suporte futuro.

Outras áreas mais gerais que podem ser melhoradas no pacote, em conjunto com as funcionalidades apontadas anteriormente, são:

- Submissão do pacote no CRAN;
- Maior quantidade de avisos e erros informativos (face ao utilizador) do pacote;
- Internacionalização (tradução do texto e componentes), no mínimo para português mas permitindo expansão para outras línguas;

- Expandir a documentação, incluindo escrever uma *vignette* em inglês com conteúdos baseados nesta dissertação;
- Acrescentar testes em componentes não gráficas (e até nas componentes gráficas, se possível);
- Toques estéticos indicados para publicação como, por exemplo, a escolha de paletes;
- Submissão de pedidos aos autores de pacotes gráficos já existentes, de acordo com as necessidades do pacote `vsd` (p.e., relativos a *bugs*).

Um ponto a dar ênfase é o modo interativo da função `vsd`. Inicialmente, esta funcionalidade surgiu como forma de contornar a problemática específica do *output* simultâneo de vários gráficos em alguns ambientes gráficos da consola do R. O uso do pacote `plotly` mostra grande potencial, pois é usado no pacote `vsd` de modo a permitir ao utilizador não só escolher os gráficos que pretende visualizar como ainda explorar dinamicamente cada um deles. Porém, é opinião do autor que essa componente do pacote também está no início do seu desenvolvimento, devendo ser expandida no futuro como, por exemplo, usando uma interface gráfica separada.

Por poucas palavras, embora o pacote `vsd` já esteja suficientemente funcional para uma gama abrangente na análise de dados no âmbito da Análise de Sobrevivência, existe ainda muito mais que se pode fazer. Este é só um começo, mas um começo que o leitor pode tentar agora mesmo.

Apêndice A

Ficheiros de código do pacote `vsd`

De seguida apresenta-se uma cópia do código do pacote `vsd`, excluindo alguns dos ficheiros auto-generados como, por exemplo, os presentes dentro da diretoria `man/`, aquando a publicação deste trabalho. Para obter a versão mais atual do pacote, consulte o repositório no GitHub respetivo [7].

```
Package: vsd
Title: Graphical Shim For Visual Survival Data Analysis
Version: 0.1.0.9013
Authors@R:
    c(person(given = "Daniel",
             family = "Camacho",
             role = c("aut", "cre"),
             email = "2016711@student.uma.pt",
             comment = c(ORCID = "YOUR-ORCID-ID")),
    person(given = "Ana",
             family = "Abreu",
             role = c("aut", "ths"),
             email = "abreu@staff.uma.pt",
             comment = c(ORCID = "0000-0002-6155-8492")))
Description: Provides a shim command for survival analysis
    graphic generation.
License: MIT + file LICENSE
Encoding: UTF-8
LazyData: true
Depends:
    survival,
    ggplot2
Imports:
    dplyr,
    survminer,
    flexsurv,
```

```
    magrittr,
    ggpubr,
    muhaz
Suggests:
    plotly
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.1.1
Collate:
    'options.R'
    'plot_forest.R'
    'plot_hazard.R'
    'plot_parametric.R'
    'util.R'
    'vsd-package.R'
    'vsd.R'
```

Bloco de Código A.1: Ficheiro `DESCRIPTION` do pacote `vsd`

```
YEAR: 2019
COPYRIGHT HOLDER: Daniel Camacho
```

Bloco de Código A.2: Ficheiro `LICENSE` do pacote `vsd`

```
# Generated by roxygen2: do not edit by hand

S3method(vsd,Surv)
S3method(vsd,coxph)
S3method(vsd,flexsurvreg)
S3method(vsd,formula)
S3method(vsd,survfit)
S3method(vsd,survfitcox)
export(vsd)
import(ggplot2)
import(survival)
importFrom(magrittr,"%>%")
importFrom(stats,model.frame)
```

Bloco de Código A.3: Ficheiro `NAMESPACE` do pacote `vsd`

```
#' VSD Options (Internal)
#'
#' Generates lists of lists of options for each graph type, from
    function call
#'
#' @param arguments Graph-specific arguments
#' @param ... General arguments
#' @keywords internal
#'
#' @return List of list of graph options
```

```
.options <- function(arguments = list(), ...)  {
  preset_ggpar <-
    list(
      main = NULL,
      title = NULL,
      submain = NULL,
      subtitle = NULL,
      xlab = "Time",
      legend.title = "Strata",
      size = 1,
      linetype = NULL,
      alpha = 1,
      color = NULL,
      palette = NULL,
      ggtheme = ggpubr::theme_pubr()
    )
  preset_ggsurv <-
    append(preset_ggpar,
           list(
             ylab = NULL,
             censor = NULL,
             censor.shape = NULL,
             censor.size = 4.5,
             conf.int = NULL
           ))
  preset_fit <- append(preset_ggsurv, list(
    conf.int.style = NULL
  ))
  preset_parametric <- append(preset_ggsurv, list(
    conf.int.km = FALSE
  ))
  preset_forest <-
    list(
      main = "Hazard ratio",
      title = NULL,
      cpositions = NULL,
      fontsize = NULL,
      refLabel = NULL,
      noDigits = NULL
    )
  preset_residuals <-
    append(
      preset_ggpar,
      list(
        ylab = NULL,
        resid = NULL,
        se = F,
        df = NULL,
        nsmo = NULL,
```

```
59          var = NULL,
60          point.col = NULL,
61          point.size = NULL,
62          point.shape = NULL,
63          point.alpha = NULL,
64          caption = NULL
65        )
66      )
67    preset_hazard <-
68      append(preset_ggpar,
69             list(ylab = "Hazard rate"))
70
71    preset <- list(
72      fit = preset_fit,
73      parametric = preset_parametric,
74      forest = preset_forest,
75      residuals = preset_residuals,
76      hazard = preset_hazard
77    )
78
79    ellipsis <- list(...)
80
81    for (type in names(preset)) {
82      preset[[type]] <- .subset_options(preset[[type]], ellipsis,
          arguments[[type]])
83    }
84
85    return(preset)
86  }
87
88
89  #' VSD (Sub)Options (Internal)
90  #'
91  #' Agglutinates preset, ellipsis, and arguments under a graph
      type
92  #'
93  #' @param subset Graph-specific preset (and allowed) values
94  #' @param ellipsis General arguments
95  #' @param subarguments Graph-specific arguments
96  #' @keywords internal
97  #'
98  #' @return subset
99  .subset_options <- function(subset, ellipsis, subarguments =
      NULL) {
100
101
102   # replaces preset with ellipsis arguments, only if they're
        already named with presets
103
```

```
104   to_replace <- ellipsis[names(ellipsis) %in% names(subset)]
105   subset[names(to_replace)] <- to_replace
106
107   # does the same for the sublist in arguments named after the
          object
108   if (is.list(subarguments)) {
109     to_replace <- subarguments[names(subarguments) %in% names(
            subset)]
110     subset[names(to_replace)] <- to_replace
111   }
112
113   # cleanup: main/submain to title/subtitle
114   # ggsurv requires it to be title, ggpar allows title by
          default
115   if (!is.null(subset$main) && "title" %in% names(subset)) {
116     subset$title <- subset$main
117     subset$main <- NULL
118   }
119   if (!is.null(subset$submain) && "subtitle" %in% names(subset))
          {
120     subset$subtitle <- subset$submain
121     subset$submain <- NULL
122   }
123
124   # cleanup: remove NULL values
125   subset[sapply(subset, is.null)] <- NULL
126
127   return(subset)
128 }
```

Bloco de Código A.4: Ficheiro `R/options.R` do pacote `vsd`

```
1 # Generates forest plots for coxph model (more than one if
      strata isn't null)
2 plot_forest <-
3   function(formula, data, strata = NULL, title, ...) {
4     plots <- list()
5
6     if (!is.null(strata)) {
7       # strategy: remake formula coxph with strata removed (
            using call and grep, 'optional:(+\w*)?strata\(.+\)'
8       # with '') then do several ggforests, using the strata to
            filter the *data* off
9
10      fit_expression <- deparse(formula$call)
11      fit_expression <-
12        str2expression(gsub("\\+?\\s?(strata\\(.+\\)) ", "", fit
              _expression))
13
```

```
14        fit_strataless <- eval(fit_expression, data)
15
16        forest_plots <- list()
17        class(forest_plots) <- "vsdstrata"
18
19        forest_plots$all <-
20          survminer::ggforest(fit_strataless, data, main = title,
              ...)
21
22        forest_plots$strata <- list()
23
24        for (i in levels(strata)) {
25          # does a forest for each strata, separatedly!
26          subdata <- data[strata == i, ]
27          fit_expression[[1]]$data <- subdata
28
29          forest_plots$strata[[i]] <-
30            survminer::ggforest(eval(fit_expression),
31                                subdata,
32                                main = paste(title, i, sep = ", ")
                                      ,
33                                ...)
34        }
35
36        plots$forest <- forest_plots
37      } else {
38        plots$forest <- survminer::ggforest(formula, data, main =
            title)
39      }
40
41      return(plots)
42    }
```

Bloco de Código A.5: Ficheiro `R/plot_forest.R` do pacote `vsd`

```
1  # Generates hazard plot
2  plot_hazard <- function(surv, strata = NULL, size, ...) {
3    plots <- list()
4
5    if (is.null(strata)) {
6      # make a simple muhaz graphic
7      hazard <- muhaz::muhaz(surv$time, surv$status)
8      hazard_df <-
9        data.frame(
10         x = hazard$est.grid,
11         y = hazard$haz.est,
12         strata = factor(rep("All", length(
13           hazard$est.grid
14         )))
```

```
15          )
16      } else {
17        # make several separate hazard maps
18        hazard_df <-
19          data.frame(x = numeric(),
20                     y = numeric(),
21                     strata = numeric())
22
23        hazard_count <- table(strata)
24
25        for (i in levels(strata)) {
26          # TODO: is it always ten?
27          if (hazard_count[[i]] < 10) {
28            warning(
29              "Level ",
30              i,
31              " doesn't have enough datapoints to estimate the
                    hazard function",
32              call. = FALSE,
33              immediate. = TRUE
34            )
35          } else {
36            # creates a sub-table with each muhaz graphic, appends
                  the corresponding strata
37            hazard <-
38              muhaz::muhaz(surv$time, surv$status, strata == i)
39            hazard_df_level <-
40              data.frame(
41                x = hazard$est.grid,
42                y = hazard$haz.est,
43                strata = rep(i, length(hazard$est.grid))
44              )
45            hazard_df <- rbind(hazard_df, hazard_df_level)
46          }
47        }
48
49        hazard_df$strata <-
50          factor(hazard_df$strata, levels(strata))
51      }
52
53      plot <-
54        ggplot(hazard_df, aes(.data$x, .data$y, color = .data$strata
              )) +
55        geom_line(size = size)
56
57      plots$hazard <- ggpubr::ggpar(plot, ...)
58
59      return(plots)
60  }
```

Bloco de Código A.6: Ficheiro `R/plot_hazard.R` do pacote `vsd`

```
# Generates parametric graph, with KM on the background
#' @importFrom magrittr %>%
plot_parametric <-
  function(model,
           km_fit,
           strata = NULL,
           data,
           alpha,
           conf.int,
           conf.int.km,
           size,
           ...) {
    if (missing(conf.int)) {
      conf.int <- TRUE
    }

    plots <- list()

    summary <- summary(model)
    if (!is.factor(strata)) {
      plots$parametric <- do.call(survminer::ggflexsurvplot,
                                  append(
                                    list(
                                      model,
                                      data,
                                      size = size,
                                      alpha = alpha,
                                      conf.int = conf.int,
                                      conf.int.km = conf.int.km
                                    ),
                                    list(...)
                                  ))

      # summary <- summary[[1]] %>% dplyr::mutate(strata = "All")
    } else {
      for (level in levels(strata)) {
        summary[[level]] <-
          summary[[level]] %>% dplyr::mutate(strata = level)
      }

      summary <- do.call(rbind, summary)

      plot_fit <-
        do.call(survminer::ggsurvplot, append(
          list(
```

```
46              km_fit,
47              data,
48              alpha = alpha / 2,
49              size = size,
50              conf.int = conf.int.km
51            ),
52            list(...)
53          ))$plot
54
55        plot_parametric <-
56          plot_fit + geom_line(aes(.data$time, .data$est, color =
                .data$strata),
57                               data = summary,
58                               size = size)
59
60        if (conf.int) {
61          plot_parametric <- plot_parametric +
62            geom_line(
63              aes(.data$time, .data$lcl, color = .data$strata),
64              data = summary,
65              size = size / 2,
66              linetype = "dashed"
67            ) +
68            geom_line(
69              aes(.data$time, .data$ucl, color = .data$strata),
70              data = summary,
71              size = size / 2,
72              linetype = "dashed"
73            )
74        }
75        plots$parametric <- plot_parametric
76
77      }
78
79      return(plots)
80    }
```

Bloco de Código A.7: Ficheiro `R/plot_parametric.R` do pacote `vsd`

```
1  # gets strata factor from formula and its respective model
2  .get_strata <- function(formula, model) {
3      if (inherits(model, "Surv")) {
4          # in case the model is JUST a surv object (hey! it
               happens)
5          return()
6      }
7
8      if (inherits(formula, "coxph")) {
9          # discards any columns not starting with strata()
```

```
            # assumedly it's only one but...
        columns <- which(grepl("^strata\\(", colnames(model)))
        if (length(columns) > 1) {
            return(survival::strata(model[, columns]))
        } else if (length(columns) == 1) {
            return(model[, columns])
        }
    } else {
        if (ncol(model) >= 2) {
            # the whole right side of the formula IS the strata
                discard the left side
            # (which is always the Surv object)
            if (ncol(model) == 2 && !is.factor(model[, 2])) {
                return(as.factor(model[, 2]))
            }
            return(survival::strata(model[, -1, drop = FALSE]))
        }
    }
}

# flattens list of graphs (as ggplots are also lists, can't just
    use unlist)
.unlist_plots <- function(plots) {
    result <- list()

    for (type in names(plots)) {
        item <- plots[[type]]
        name <- type

        sublist <- NULL
        if (is.list(item) && !inherits(item, c("ggplot", "
            ggsurvplot"))) {
            if (inherits(item, "vsdstrata")) {
                sublist <- list(all = list(item$all), strata =
                    item$strata)
                sublist <- unlist(sublist, recursive = FALSE)
            } else {
                sublist <- item
            }
        }

        if (is.list(sublist)) {
            for (subname in names(sublist)) {
                result[[paste(name, subname, sep = "$")]] <-
                    list(plot = sublist[[subname]], type = type)
            }
        } else {
            result[[name]] <- list(plot = item, type = type)
        }
```

```
53
54      }
55
56      return(result)
57  }
```

Bloco de Código A.8: Ficheiro `R/util.R` do pacote `vsd`

```
1  #' Visualizing Survival Data
2  #'
3  #' This function outputs renders of the inputted survival data
       analysis data
4  #' and/or model, and their components, into a graphically
       pleasing output, under
5  #' the [ggplot2] format.
6  #'
7  #' Depending on the kind of model passed to the function, the
       kind of generated
8  #' graphics might vary, but usually an estimation of the
       survival and risk
9  #' curves (depending if the model has covariables) is expected.
       The kinds of
10 #' graphics that can be created according to a specific R object
        are detailed on
11 #' Usage on `.include`'s definition, but non-relevant graphics
       can be requested
12 #' without error, as the function ignores them silently.
13 #'
14 #' Extra options for each graph kind can be passed to either all
        created
15 #' graphics, by having them as generic arguments, or to specific
        graphic types,
16 #' using a list in `.arguments`. Arguments are filtered, so that
        generic
17 #' arguments aren't applied if a graphic kind wouldn't use them.
       As an example,
18 #' ```
19 #' vsd(model, data,
20 #'     .arguments = list(fit = (size = 3, xlab = "Weeks")),
21 #'     xlab = "Days")
22 #' ```
23 #' would set all graphics that have an label on the x axis to
24 #' "Days", except the `fit` graph, which would have "Weeks"
       instead.
25 #'
26 #' # Generic graphical arguments
27 #'
28 #' Unless specified, all graphics are created under [ggpubr::
       ggpar()] and have
```

```
29 #' as additional options 'palette', 'main', 'submain', 'xlab', '
       ylab',
30 #' 'legend.title' and 'ggtheme'. Most line graphics also allow
       to set the
31 #' options 'size', 'linetype', 'alpha' and 'color' to determine
       line styles, as
32 #' detailed on [survminer::ggsurvplot()].
33 #'
34 #' ## fit
35 #'
36 #' Line graphic, with a further subset of the options present in
37 #' [survminer::ggsurvplot()]: 'censor', 'censor.shape', 'censor.
       size',
38 #' 'conf.int', 'conf.int.style'.
39 #'
40 #' ## parametric
41 #'
42 #' Line graphic, with a further subset of the options present in
43 #' [survminer::ggflexsurvplot()]: 'conf.int.km'.
44 #'
45 #' ## forest
46 #'
47 #' Non-standard graphic(s), using all options within [survminer
       ::ggforest()]:
48 #' 'main', 'cpositions', 'fontsize', 'refLabel', 'noDigits'.
49 #'
50 #' ## residuals
51 #'
52 #' Line graphic(s), with a further subset of options present in
53 #' [survminer::ggcoxzph()]: 'resid', 'se', 'df', 'nsmo', 'var',
       'caption'; and
54 #' point style customization options as 'point.col', 'point.size
       ',
55 #' 'point.shape', and 'point.alpha'.
56 #'
57 #' ## hazard
58 #'
59 #' Line graphics, using the generic graphical arguments.
60 #'
61 #' @param model The survival model, or data structure, to
       generate graphics from
62 #' @param data Dataframe from where the model fetches its
       variables, if left
63 #'   blank will be extracted from the model, if possible
64 #' @param .interactive Allows to explore the generated graphs
       before returning
65 #'   (use with [plotly](https://plotly.com/r/) package for best
       results)
66 #' @param .include Graph types to output if relevant, defaults
```

```
    to all possible
#' @param .arguments Collection of list of arguments, indexed by the specific
#'   type of graph they should be passed to, has priority over \dots
#' @param ... Miscellaneous arguments, passed to ALL graphs
#'
#' @import survival
#' @import ggplot2
#' @importFrom stats model.frame
#' @export
#' @return A list of ggplot2 graphs and/or list of graphs, relevant to the model
#'
#' @examples
#' # non-models are cohersed into a survfit object with default arguments
#' \dontrun{
#' vsd(with(aml, Surv(time, status)))
#' vsd(Surv(time, status) ~ ph.ecog, data=lung)
#'
#' # survival fit model
#' vsd(survfit(Surv(futime, fustat) ~ rx, data = ovarian))
#'
#' # coxph (with and without strata)
#' vsd(coxph(Surv(time, status) ~ sex + rx + adhere, data = colon))
#' }
#' vsd(survfit(coxph(Surv(time, status) ~ sex + strata(rx) + adhere, data = colon)))
#'
#' # parametric models
#' vsd(flexsurv::flexsurvreg(Surv(rectime, censrec) ~ group, data = flexsurv::bc, dist = 'gengamma'),
#'   .include = c("par"))
vsd <- function(model,
                data = NULL,
                .interactive = FALSE,
                .include =
                  c("fit", "parametric", "forest", "residuals",
                    "hazard"),
                .arguments = list(),
                ...) {
    UseMethod("vsd")
}

#' @describeIn vsd Wraps `Surv(...) ~ (...)` in a survfit object (Kaplan-Meier
#' model)
```

```
106 #' @export
107 vsd.formula <- function(model,
108                         data = NULL,
109                         .interactive = FALSE,
110                         .include = c("fit", "hazard"),
111                         .arguments = list(),
112                         ...) {
113   # (Assumedly) '~' call (TODO: fail first?)
114   if (is.null(data)) {
115     stop("Data structure required with fit object of type call."
          )
116   }
117   new_model <- survfit(model, data)
118   new_model$call$formula <- eval(model, data)
119
120   vsd(new_model, data, .interactive, .include, .arguments, ...)
121 }
122
123 #' @describeIn vsd Wraps [Surv()] in a survfit object (Kaplan-
      Meier
124 #' model)
125 #' @export
126 vsd.Surv <- function(model,
127                      data = NULL,
128                      .interactive = FALSE,
129                      .include = c("fit", "hazard"),
130                      .arguments = list(),
131                      ...) {
132   # Surv object TODO: more than just right-censored survival
133   data <- as.data.frame(as.matrix(model))
134   warning("Fetched `data`: ", data, call. = FALSE, immediate. =
        TRUE)
135   new_model <- survfit(Surv(time, status) ~ 1, data)
136   warning("New model: ", new_model, call. = FALSE, immediate. =
        TRUE)
137
138   vsd(new_model, data, .interactive, .include, .arguments, ...)
139 }
140
141 #' @describeIn vsd Wraps \code{coxph(...)} in a survfit object (
      Kaplan-Meier
142 #' model)
143 #' @export
144 vsd.coxph <- function(model,
145                       data = NULL,
146                       .interactive = FALSE,
147                       .include = c("fit", "forest", "residuals",
                            "hazard"),
148                       .arguments = list(),
```

```
149                         ...) {
150   if (is.null(data)) {
151     data <- eval(model$call$data)
152     if (is.null(data)) {
153       stop("Original data structure couldn't be extracted, ",
154            "supply it to function call instead")
155     }
156   }
157
158   # http://adv-r.had.co.nz/Expressions.html#capturing-call
159   new_model <- survfit(model)
160   new_model$call$formula <- substitute(model)
161   vsd(new_model, data, .interactive, .include, .arguments, ...)
162 }
163
164 #' @describeIn vsd Graphical output for survfit objects (Kaplan-
      Meier model)
165 #' @export
166 vsd.survfit <- function(model,
167                         data = NULL,
168                         .interactive = FALSE,
169                         .include = c("fit", "hazard"),
170                         .arguments = list(),
171                         ...) {
172   plots <- list()
173   options <- .options(.arguments, ...)
174   include <-
175     as.vector(match.arg(.include, several.ok = TRUE))
176
177   # retrieving mid-objects
178   if (is.null(data)) {
179     if (!is.null(model$call$data)) {
180       data <- eval(model$call$data)
181     } else if (is.call(model$call$formula) &&
182                model$call$formula[[1]] == "coxph") {
183       data <- eval(eval(model$call$formula)$call$data)
184     }
185
186     if (is.null(data)) {
187       stop("Original data structure couldn't be extracted,",
188            " supply it to function call instead")
189     }
190   }
191
192   formula <- model$call$formula
193   model_frame <- model.frame(formula, data)
194   strata <- .get_strata(formula, model_frame)
195
196   surv <- as.data.frame(as.matrix(model_frame[, 1]))
```

```
197
198   #### PLOT$FIT
199   if (("fit" %in% include)) {
200     fit_plots <- do.call(survminer::ggsurvplot, append(
201       list(model, data), options$fit))
202     plots$fit <- fit_plots$plot
203   }
204
205   #### PLOT$HAZARD
206   if (("hazard" %in% include)) {
207     hazard_plots <-
208       do.call(plot_hazard, append(list(surv, strata), options$
              hazard))
209     plots <- append(plots, hazard_plots)
210   }
211
212   if (.interactive && interactive()) {
213     .do_interactive(plots)
214   }
215   return(plots)
216 }
217
218 #' @describeIn vsd Graphical output for survfit objects (Cox
      model)
219 #' @export
220 vsd.survfitcox <- function(model,
221                            data = NULL,
222                            .interactive = FALSE,
223                            .include = c("fit", "forest", "
                               residuals", "hazard"),
224                            .arguments = list(),
225                            ...) {
226   plots <- list()
227   options <- .options(.arguments, ...)
228   include <- as.vector(match.arg(.include, several.ok = TRUE))
229
230   # retrieving mid-objects
231   if (is.null(data)) {
232     data <- eval(eval(model$call$formula)$call$data)
233
234     if (is.null(data)) {
235       stop("Original data structure couldn't be extracted, ",
236            "supply it to function call instead")
237     }
238   }
239
240   cox_model <- eval(model$call$formula, data)
241   formula <- cox_model$formula
242   model_frame <- model.frame(formula, data)
```

```
243   strata <- .get_strata(cox_model, model_frame)
244
245   surv <- as.data.frame(as.matrix(model_frame[, 1]))
246
247   #### PLOT$FIT
248   if (("fit" %in% include)) {
249     fit_plots <- do.call(survminer::ggsurvplot, append(
250       list(model, data), options$fit))
251     plots$fit <- fit_plots$plot
252   }
253
254   #### PLOT$FOREST
255   if (("forest" %in% include)) {
256     forest_plots <- do.call(plot_forest, append(
257       list(cox_model, data, strata), options$forest))
258     plots <- append(plots, forest_plots)
259   }
260
261   #### PLOT$RESIDUALS (for coxph)
262   if (("residuals" %in% include)) {
263     plots$residuals <- do.call(survminer::ggcoxzph, append(
264       list(cox.zph(cox_model)), options$residuals))
265   }
266
267   #### PLOT$HAZARD
268   if (("hazard" %in% include)) {
269     hazard_plots <- do.call(plot_hazard, append(
270       list(surv, strata), options$hazard))
271     plots <- append(plots, hazard_plots)
272   }
273
274   if (.interactive && interactive()) {
275     .do_interactive(plots)
276   }
277   return(plots)
278 }
279
280 #' @describeIn vsd Graphical output for flexsurvreg objects (
      various parametric models)
281 #' @export
282 vsd.flexsurvreg <- function(model,
283                             data = NULL,
284                             .interactive = FALSE,
285                             .include = c("fit", "parametric", "
                                hazard"),
286                             .arguments = list(),
287                             ...) {
288   plots <- list()
289   options <- .options(.arguments, ...)
```

```
290   include <- as.vector(match.arg(.include, several.ok = TRUE))
291
292   if (is.null(data)) {
293     if (!is.null(model$call$data)) {
294       data <- eval(model$call$data)
295     }
296
297     if (is.null(data)) {
298       stop("Original data structure couldn't be extracted, ",
299            "supply it to function call instead")
300     }
301   }
302
303   formula <- eval(model$call$formula, data)
304   model_frame <- model.frame(model)
305   strata <-
306     .get_strata(formula, model_frame[, !(names(model_frame) == "
          (weights)")])
307   surv <- as.data.frame(as.matrix(model_frame[, 1]))
308
309   km_fit <- survfit(formula, data)
310   km_fit$call$formula <- eval(km_fit$call$formula, data)
311
312
313   #### PLOT$FIT
314   if ("fit" %in% include) {
315     plot_fit <- do.call(survminer::ggsurvplot, append(
316       list(km_fit, data), options$fit))
317     plots$fit <- plot_fit$plot
318   }
319
320   #### PLOT$PARAMETRIC
321   if (("parametric" %in% include)) {
322     parametric_plots <-
323       do.call(plot_parametric, append(
324         list(model, km_fit, strata, data), options$parametric))
325     plots <- append(plots, parametric_plots)
326   }
327
328   #### PLOT$HAZARD
329   if (("hazard" %in% include)) {
330     hazard_plots <- do.call(plot_hazard, append(
331       list(surv, strata), options$hazard))
332     plots <- append(plots, hazard_plots)
333   }
334
335   if (.interactive && interactive()) {
336     .do_interactive(plots)
337   }
```

```
338   return(plots)
339 }
340
341
342 .do_interactive <- function(plots) {
343   # TODO: make choices into two lists: plots, types ?
344   choices <- .unlist_plots(plots)
345   whitelist <- c("fit", "parametric", "residuals", "hazard")
346
347   repeat {
348     choice <- utils::menu(names(choices),
349                           title = "Pick a graphic (or 0 to exit)
                                ")
350     if (choice <= 0)
351       break
352     choice <- choices[[choice]]
353     plot <- choice$plot
354     type <- choice$type
355
356     if (type %in% whitelist) {
357       if (requireNamespace("plotly", quietly = TRUE)) {
358         if (inherits(plot, "ggsurvplot")) {
359           print(plotly::ggplotly(plot$plot))
360         } else {
361           print(plotly::ggplotly(plot))
362         }
363       } else {
364         print(plot)
365       }
366     } else {
367       print(plot)
368     }
369   }
370 }
```

Bloco de Código A.9: Ficheiro `R/vsd.R` do pacote `vsd`

```
1 #' @details The only function you're likely to need is [vsd],
      all other
2 #' functions are either private or subfunctions that shouldn't
      be called
3 #' directly.
4 #'
5 #' @include options.R
6 #' @include util.R
7 #' @keywords internal
8 #' @aliases vsd-package
9 "_PACKAGE"
```

Bloco de Código A.10: Ficheiro `R/vsd-package.R` do pacote `vsd`

Bibliografia

[1] Aurelien Latouche Arthur Allignol. *CRAN Task View: Survival Analysis*. [Online; acedido em 16 de novembro de 2020]. Nov. de 2020. URL: https://cran.r-project.org/web/views/Survival.html.

[2] N. E. Breslow. «Discussion of Professor Cox's Paper». Em: *Journal of the Royal Statistical Society* 34.2 (1972).

[3] Scott Chacon. *Pro Git*. Berkeley, CA New York, NY: Apress, Distributed to the Book trade worldwide by Spring Science+Business Media, 2014. ISBN: 978-1-484-20077-3.

[4] *CRAN - Mirrors*. [Online; acedido em 5 de outubro de 2020]. Out. de 2020. URL: https://cran.r-project.org/mirrors.html.

[5] *CRAN Repository Policy*. [Online; acedido em 16 de novembro de 2020]. Out. de 2020. URL: https://cran.r-project.org/web/packages/policies.html.

[6] cyanzule. *pacote*. [Online; acedido em 19 de outubro 2020]. Out. de 2020. URL: https://github.com/cyanzule/pacote.

[7] cyanzule. *vsd*. [Online; acedido em 18 de outubro de 2020]. Out. de 2020. URL: https://github.com/cyanzule/vsd.

[8] *Drawing Survival Curves using 'ggplot2' [R package survminer version 0.4.8]*. [Online; acedido 26 de outubro de 2020]. Jul. de 2020. URL: https://CRAN.R-project.org/package=survminer.

[9] *Git - git-add Documentation*. [Online; acedido em 19 de outubro de 2020]. Out. de 2020. URL: https://git-scm.com/docs/git-add.

[10] Kurt Hornik, Duncan Murdoch e Achim Zeileis. «Who Did What? The Roles of R Package Authors and How to Refer to Them». Em: *The R Journal* 4 (nov. de 2011). DOI: 10.32614/RJ-2012-009.

[11] *In-Line Documentation for R - Roxygen*. [Online; acedido em 12 de outubro de 2020]. Jul. de 2020. URL: https://roxygen2.r-lib.org/.

[12] Open Source Initiative. *Licenses & Standards*. [Online; acedido em 20 de novembro de 2020]. 2016. URL: http://opensource.org/licenses.

[13] Robert Kabacoff. *R in action : data analysis and graphics with R*. Shelter Island: Manning, 2015. ISBN: 978-1-617-29138-8.

[14] Edward L Kaplan e Paul Meier. «Nonparametric estimation from incomplete observations». Em: *Journal of the American Statistical Association* 53.282 (1958), pp. 457–481.

[15] Paul Murrell. *R graphics*. Boca Raton: CRC Press, Taylor & Francis Group, 2019. ISBN: 978-0-429-42276-8.

[16] Tom Preston-Werner. *Semantic Versioning 2.0.0*. [Online; acedido em 18 de outubro de 2020]. Out. de 2020. URL: https://semver.org.

[17] *R: What is R?* Jul. de 2019. URL: https://www.r-project.org/about.html.

[18] C. Rocha e A. L. Papoila. *Análise de Sobrevivência. XVII Congresso da Sociedade Portuguesa de Estatística*. SPE, 2009. ISBN: 978-972-8890-22-3.

[19] R Core Team. *R: A language and environment for statistical computing*. Vienna, Austria, 2020.

[20] Terry M. Therneau. *Survival Analysis [R package survival version 3.2-3]*. [Online; acedido em 12 de outubro de 2020]. Jul. de 2020. URL: https://CRAN.R-project.org/package=survival.

[21] *Tools to Make Developing R Packages Easier [R package devtools version 2.3.2]*. [Online; acedido em 12 de outubro de 2020]. Set. de 2020. URL: https://CRAN.R-project.org/package=devtools.

[22] *Understanding Copyright and Related Rights*. Geneva, Switzerland: World Intellectual Property Organization (WIPO, 2016. ISBN: 978-92-805-2799-5.

[23] *Visualizing Categorical Data [R package vcd version 1.4-8]*. [Online; acedido em 26 de outubro de 2020]. Set. de 2020. URL: https://CRAN.R-project.org/package=vcd.

[24] Hadley Wickham. *ggplot2: elegant graphics for data analysis*. Switzerland: Springer, 2016. ISBN: 978-3-319-24275-0.

[25] Hadley Wickham. *R packages*. First edition. OCLC: ocn898161451. Sebastopol, CA: O'Reilly Media, 2015. ISBN: 978-149-1910-59-7. URL: http://r-pkgs.had.co.nz/.