Writing technical reports of any economic analysis in R with **bookdown**

A short tutorial for *R for HTA* using a tutorial by Alvarez and Ribero

FW Thielen

2022-05-22

Contents

Preface

This book was created to demonstrate how the online "Tutorial to Create a Discrete Event Simulation Model in R Using descem", written by *Javier Sanchez Alvarez* and *Valerie Aponte Ribero* and published on **R for HTA** can be converted into a **bookdown** report/book.[1]

This report, together with its code can be found on github and was written for the *R for HTA workshop* on Thursday 19th, Friday 20th, and Monday 23rd May 2022. It is part of the presentation on Monday 23rd May 2022: "Copy and paste code, not output. Writing technical reports of any economic analysis in R with Bookdown" and "Estimating Dutch costs of productivity losses in R using up-to-date data from Statistics Netherlands (CBS) with the R-package cbsodataR". More on costs of producitivty losses can be found in Krol and Brouwer (2014).[2]

Setup

The following R-code chunk is only visible in the `index.Rmd` file, unless the chunk option `echo` is set to `TRUE`.

This chunk also controls the options for all other chunks in the document. These options can be overruled in each chunk separately.

Be aware that options like `warnings = FALSE` can be undesirable and should be used with caution.

Other

The standard approach of **bookdown** is to first merge all `.Rmd` files you specified in the `_bookdown.yml` file. After that, these files are rendered. Therefore, all object used in later chapter need to be created somewhere in a previous one.

Rmarkdowns behave as they usually do:

- One `#` is for first level headings
- Two `##` is for second level headings, and so on
- A `{-}` after a heading means that it is not automatically numbered

You can render this book either:

- as a PDF file with:
- or as a HTML file with:

1 Introduction

This document runs a discrete event simulation model in the context of early breast cancer to show how the functions can be used to generate a model in only a few steps.

When running a DES, it's important to consider speed. Simulation based models can be computationally expensive, which means that using efficient coding can have a substantial impact on performance. Because the model is using parallel computing, in order to debug the model one can simply use browser() where needed and set the number of patients to be simulated equal to 1.

1.1 Main options

```
# The following was adapted from the original code because the github
# path and names changed
# devtools::install_github("Roche/descem")

library(descem)

library(dplyr)

#> Attaching package: 'dplyr'
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
library(flexsurv)
#> Loading required package: survival
library(ggplot2)
library(kableExtra)
#>
#> Attaching package: 'kableExtra'
#> The following object is masked from 'package:dplyr':
#>
#>     group_rows
library(purrr)
#>
```

```
#> Attaching package: 'purrr'
#> The following object is masked from '.my.env':
#>
#>     keep
library(tidyr)

# Additional packages ----
library("bookdown")

# Write citations from packages
knitr::write_bib(file = here::here("references/packages.bib"))
```

```
options(scipen = 999)
options(tibble.print_max = 50)
```

2 General inputs with delayed execution

Initial inputs and flags that will be used in the model can be defined below. We can define inputs that are common to all patients (`common_all_inputs`) within a simulation, inputs that are unique to a patient independently of the treatment (e.g. natural death, defined in `common_pt_inputs`), and inputs that are unique to that patient and that treatment (`unique_pt_inputs`). Items can be included through the add_item function, and can be used in subsequent items. All these inputs are generated before the events and the reaction to events are executed. Furthermore, the program first executes common_all_inputs, then `common_pt_inputs` and then `unique_pt_inputs`. So one could use the items generated in `common_all_inputs` in `unique_pt_inputs`.

3 Events

3.1 Add Initial Events

Events are added below through the `add_tte` function. We use this function twice, one per intervention. We must define several arguments: one to indicate the intervention, one to define the names of the events used, one to define the names of other objects created that we would like to store (optional, maybe we generate an intermediate input which is not an event but that we want to save) and the actual input in which we generate the time to event. Events and other objects will be automatically initialized to `Inf`. We draw the times to event for the patients. This chunk is a bit more complex, so it's worth spending a bit of time explaining it.

The `init_event_list` object is populated by using the `add_tte` function twice, one for the "int" strategy and other for the "noint" strategy. We first declare the `start` time to be `0`.

We then proceed to generate the actual time to event. We use the `draw_tte()` function to generate the time to event. One should always be aware of how the competing risks interact with each other. While we have abstracted from these type of corrections here, it is recommended to have an understanding about how these affect the results and have a look at the competing risks/semi-competing risks literature.

3.2 Add Reaction to Those Events

Once the initial times of the events have been defined, we also need to declare how events react and affect each other. To do so, we use the `evt_react_list` object and the `add_reactevt` function. This function just needs to state which event is affected, and the actual reaction (usually setting flags to 1 or 0, or creating new/adjusting events).

There are a series of objects that can be used in this context to help with the reactions. Apart from the global objects and flags defined above, we can also use `curtime` for the current event time, `prevtime` for the time of the previous event, `cur_evtlist` for the named vector of events that is yet to happen for that patient, trt for the current treatment in the loop, `evt` for the current event being processed, `i` expresses the patient iteration, and `simulation` the specific simulation (relevant when the number of simulations is greater than 1). Furthermore, one can also call any other input/item that has been created before or create new ones. For example, we could even modify a cost/utility item by changing it directly, e.g. through `modify_item(list(cost.idfs.tx=500))`.

Table 3.1: Table without caption in original tutorial

Item	What does it do
curtime	Current event time (numeric)
prevtime	Time of the previous event (numeric)
cur_evtlist	Named vector of events that is yet to happen for that patient (named numeric vector)
evt	Current event being processed (character)
i	Patient being iterated (character)
simulation	Simulation being iterated (numeric)

At this place, a Table is rendered in the HTML version of the tutorial. In this document, Tables are automatically numbered and cross-referenced. For this to work, the R-code chunk need to have a:

- *unique* name (avoid underscores _)
- the table needs to have a caption, passed in the `kintr::kable()` option.

Table 3.1 below shows a standard output with a pre-specified theme form the package **kableExtra** (Version 1.3.4).[1]

```
tab1 <- openxlsx::read.xlsx(xlsxFile = here::here("raw/table1.xlsx"),
                            sheet = "table1")

knitr::kable(x = tab1,
             col.names = c("Item", "What does it do"),
             caption = "Table without caption in original tutorial") %>%
        kableExtra::kable_classic()
```

The problem with rendering tables in that way it that they sometimes do not render at the exact place where you want them. Also, they don't look very nice and when they are wide, they don't fit.

Below I share two functions (after a lot of trial and error) that work quite well for both HTML and PDF output. Feel free to use and re-use:

```
# A wrapper for the kintr::kable() function
fkbl <- function (df,
                  digi = 2,
                  big_mark = TRUE,
                  col.names = names(df),
                  ...)
{
        require("dplyr")
        require("kableExtra")
```

```
        if (knitr::pandoc_to("docx")) {
                df %>%
                        flextable::flextable() %>%
                        flextable::colformat_double(x = .,
                                                    big.mark = ifelse(
                                                            big_mark,
                                                            ",",
                                                            ""),
                                                    digits = digi
                        )
        } else {
                df %>%
                        kableExtra::kbl(col.names = col.names,
                                        digits = digi,
                                        booktabs = TRUE,
                                        linesep = "",
                                        format.args = list(
                                                big.mark = ifelse(big_mark,
                                                                  ",",
                                                                  "")
                                                ),
                                        ...)
        }
}
```

```
# A wrapper for kable::Extra::kable_styling()

fkbl_style <- function(fkbl,
                       scale = FALSE,
                       boots_opts = c("striped",
                                      "condensed",
                                      "hover"),
                       latx_opts = c("striped", "HOLD_position"), ...)
{
        require("kableExtra")
        if (knitr::pandoc_to("docx")) {
                fkbl
        }
        else {
                fkbl %>% kableExtra::kable_styling(
                        bootstrap_options = boots_opts,
                        latex_options = if (scale) {
                                c(latx_opts, "scale_down")
                        }
```

```
                        else {
                                latx_opts
                        }, full_width = F, ...) %>%
                        kableExtra::scroll_box(.,
                                               width = "100%",
                                               box_css = "border: 0px;")
        }
}
```

```
tab1 %>%
        fkbl(col.names = c("Item", "What does it do"),
             caption = "Table without caption in original tutorial2") %>%
        fkbl_style(scale = T)
```

Table 3.2: Table without caption in original tutorial2

Item	What does it do
curtime	Current event time (numeric)
prevtime	Time of the previous event (numeric)
cur_evtlist	Named vector of events that is yet to happen for that patient (named numeric vector)
evt	Current event being processed (character)
i	Patient being iterated (character)
simulation	Simulation being iterated (numeric)

The functions to add/modify events and inputs use lists. Whenever several inputs/events are added or modified, it's recommended to group them within one function, as it reduces the computation cost. So rather than use two `modify_item` with a list of one element, it's better to group them into a single `modify_item` with a list of two elements.

`new_event` allows to generate events and add them to the vector of events. It accepts more than one event. `modify_event` allows to modify events (e.g. delay death). When adding an event, the name of the events and the time of the events must be defined. When using `modify_event`, one must indicate which events are affected and what are the new times of the events. If the event specified does not exist or has already occurred, it will be ignored. Note that one could potentially omit part of the modeling set in `init_event_list` and actually define new events dynamically through the reactions (we do that below for the `"ae"` event). However, this can have an impact in computation time, so if possible it's always better to use `init_event_list`. `modify_item` allows to modify and add items. Elements defined within this function are not evaluated sequentially (i.e. defining `modify_item(list(fl.new = 1, var1 = fl.new * 5))` will give an error if `fl.new` was not defined outside this function).

The list of relevant functions to be used within `add_reactevt` are (shown in Table 3.3):

Table 3.3: Another table with no caption

Function	What does it do	How to use it
modify_item()	Adds & Modifies items/flags/variables for future events	modify_item(list("fl.idfs.ontx"=0,"fl.tx.beva"=0))
new_event()	Adds events to the vector of events for that patient	new_event(rep(list("ae"=curtime + 0.001),5))
modify_event()	Modifies existing events by changing their time	modify_event(list("os"=curtime +5, "ttot"=curtime+0.0001))

The model will run until `curtime` is set to `Inf`, so the event that terminates the model (in this case, `os`), should modify `curtime` and set it to `Inf`.

4 Costs and Utilities

Costs and utilities are introduced below. However, it's worth noting that the model is able to run without costs or utilities.

4.1 Utilities

Utilities are defined using pipes with the `add_util` function. The first argument says which events are affected, the second argument which treatments are affected, and the third one describe the utilities. Instant utilities (e.g. AE disutilities) and cycle utilities can be defined in a similar fashion. Note that one can write expressions and objects whose execution will be delayed until the model runs.

```
util_ongoing <- add_util(evt = c("sick", "sicker","death"),
                         trt = c("int", "noint"),
                         util = util.sick * fl.sick +
                                 util.sicker * (1-fl.sick)
)
```

4.2 Costs

Costs are defined using pipes with the `add_cost` function, in a similar fashion to the utilities.

```
cost_ongoing <-
        add_cost(
                evt = c("sick", "sicker","death") ,
                trt = "noint",
                cost = cost.sick * fl.sick +
                        cost.sicker * (1-fl.sick) ) %>%
        add_cost(
                evt = c("sick", "sicker","death") ,
                trt = "int",
                cost = cost.sick * fl.sick + cost.sicker * (1-fl.sick) +
                        cost.int * fl.sick )
```

5 Model

5.1 Model Execution

The model can be run using the function `RunSim` below. We must define the number of patients to be simulated, the number of simulations, whether we want to run a PSA or not, the strategy list, the inputs, events and reactions defined above, the number of cores to be used (by default uses 1 core), the discount rate for costs and the discount rate for qalys. It is recommended not to use all the cores in the machine.

It is worth noting that the `psa_bool` argument does not run a PSA automatically, but is rather an additional input/flag of the model that we use as a reference to determine whether we want to use a deterministic or stochastic input. As such, it could also be defined in `common_all_inputs` as the first item to be defined, and the result would be the same. However, we recommend it to be defined in `RunSim`.

Note that the distribution chosen, the number of events and the interaction between events can have a substantial impact on the running time of the model.

```
## [1] "Simulation number: 1"
## [1] "Time to run iteration 1: 0.71s"
## [1] "Total time to run: 0.71s"
```

6 Post-processing of Model Outputs

6.1 Summary of Results

Once the model has been run, we can use the results and summarize them using the `summary_results_det` to print the results of the last simulation (if nsim=1, it's the deterministic case), and `summary_results_psa` to show the PSA results (with the confidence intervals). We can also use the individual patient data generated by the simulation, which we collect here to plot in the `psa_ipd` object.

The last simulation is summarised in Table 6.1, and the results of the PSA are summarised in Table 6.2.

Table 6.1: Last simulation results

Item	Intervention	No intervention
costs	54,768.48	52,502.48
lys	9.74	9.74
qalys	6.21	6.05
ICER	NA	Inf
ICUR	NA	13,667.14

Table 6.2: PSA results

Item	Intervention	No intervention
costs	54768(54768, 54768)	52502(52502, 52502)
lys	9.74(9.74, 9.74)	9.74(9.74, 9.74)
qalys	6.21(6.21, 6.21)	6.05(6.05, 6.05)
ICER	NaN(NA, NA)	Inf(Inf, Inf)
ICUR	NaN(NA, NA)	13667(13667, 13667)

Finally, the Table with the first ten rows of the simulation of the original tutorial is summarised in Table 6.3 (with the original column names).

Table 6.3: Last table

evtname	evttime	cost	qaly	ly	pat_id	trt	total_costs	total_qalys	total_lys	simulation
sick	0.00	0.00	0.00	0.00	1	int	30,374.96	2.52	4.63	1
sicker	0.68	2,701.69	0.54	0.68	1	int	30,374.96	2.52	4.63	1
death	5.04	27,673.26	1.98	3.95	1	int	30,374.96	2.52	4.63	1
sick	0.00	0.00	0.00	0.00	2	int	52,039.41	5.12	8.61	1
sicker	2.87	10,927.88	2.19	2.73	2	int	52,039.41	5.12	8.61	1
death	10.20	41,111.53	2.94	5.87	2	int	52,039.41	5.12	8.61	1
sick	0.00	0.00	0.00	0.00	3	int	56,227.85	6.34	9.97	1
sicker	4.91	18,074.04	3.61	4.52	3	int	56,227.85	6.34	9.97	1
death	12.21	38,153.81	2.73	5.45	3	int	56,227.85	6.34	9.97	1
sick	0.00	0.00	0.00	0.00	4	int	56,617.71	9.29	12.46	1

6.2 Plots

We now use the data output to plot the histograms/densities of the simulation.

With **bookdown**, Plots and Figures can also be cross-referenced. Be aware that for those, the caption needs to be provided in the R-code chunk through `fig.cap` (see chunk below in the `.Rmd` file). In-text, Figures are referenced similar to Tables. Figure 6.1 summarises the event times.

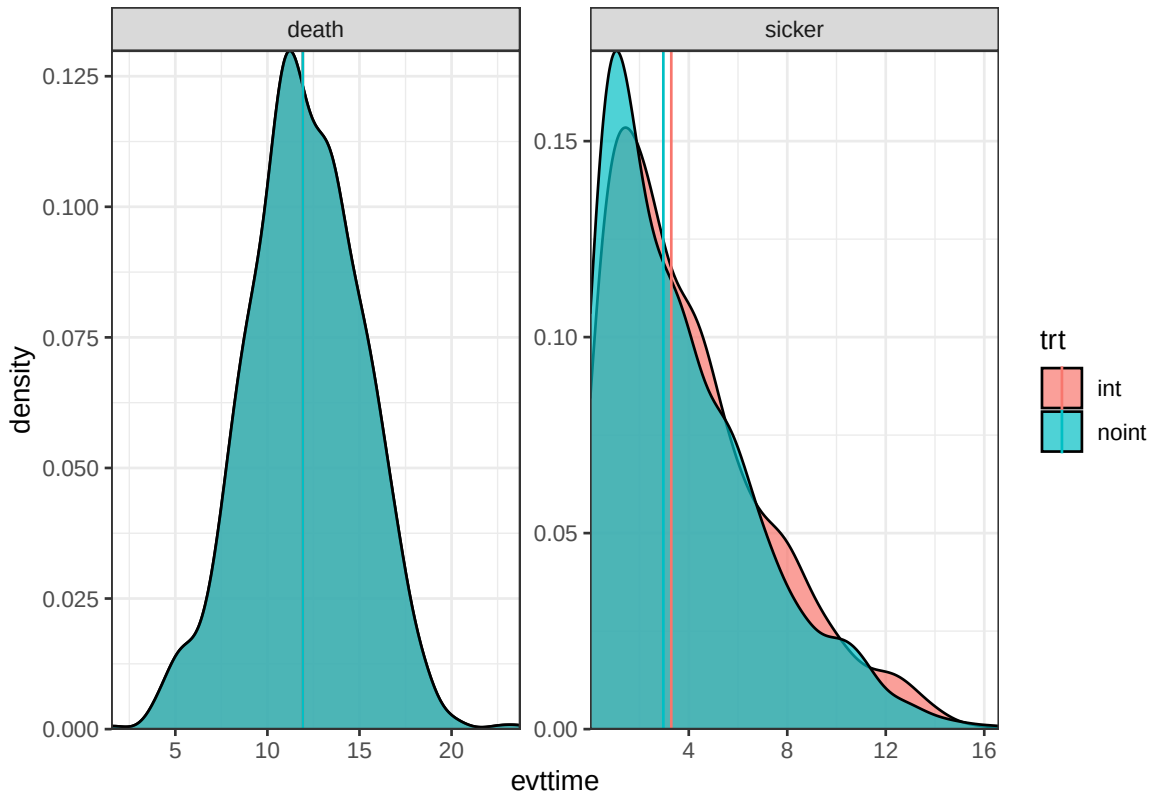

Figure 6.1: First Figure

We can also plot the patient level incremental QALY/costs. Note that there are several clusters in the distribution of patients according to their QALY/costs based on the pathway they took (early metastatic vs. remission and cure or recurrence, see Figure 6.2).

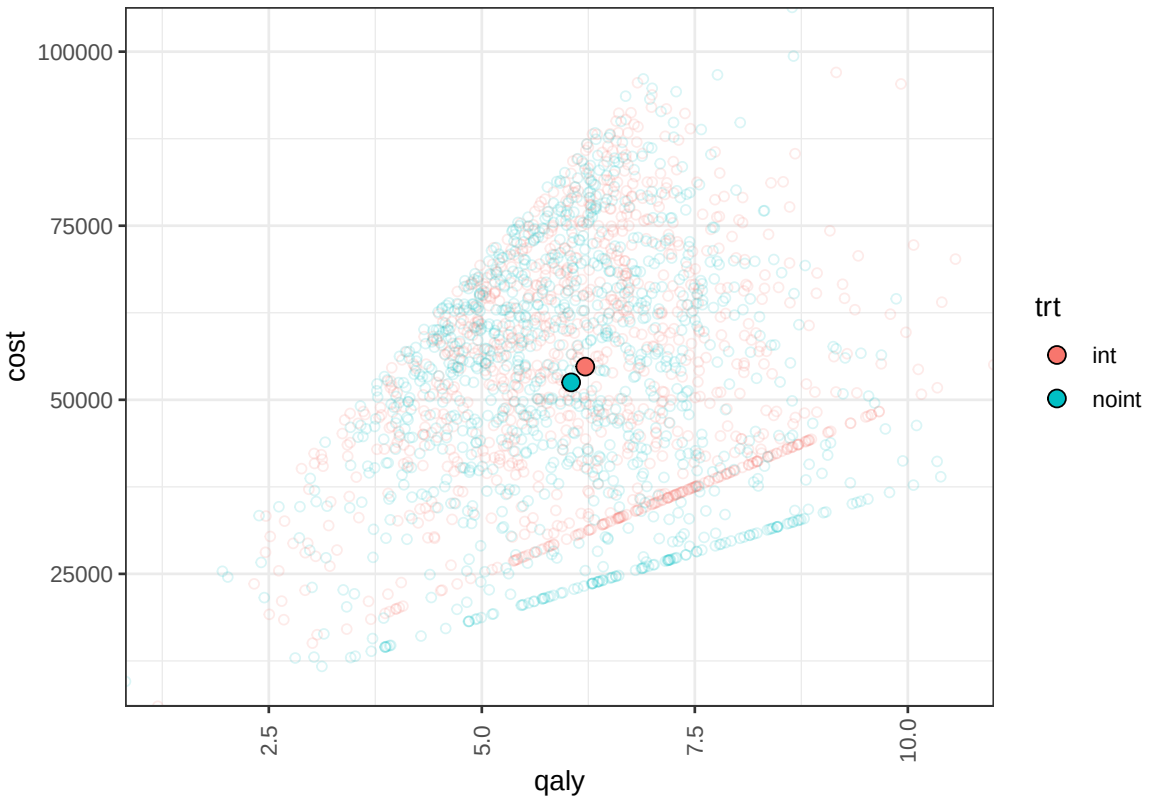

Figure 6.2: Last Figure of Tutorial

You can also cross-reference and link to previous chapter like so: see Section 1 for a short introduction to this tutorial.

7 References

1. Xie, Y. *Bookdown: Authoring books and technical documents with R markdown.* (Chapman; Hall/CRC, 2016).
2. Krol, M. & Brouwer, W. How to Estimate Productivity Costs in Economic Evaluations. *PharmacoEconomics* **32**, 335–344 (2014).