Analysing seed germination and emergence data with R: a tutorial

Andrea Onofri

Update: v. 0.651 (2023-01-16), compil. 2023-04-06

Contents

1 Introduction **5**
- 1.1 The R packages . . . 6
- 1.2 Our purpose . . . 6

2 What is time-to-event data? **9**
- 2.1 Time-to-event data are censored data . . . 9
- 2.2 Motivating example: a simulated dataset . . . 10
- 2.3 Neglecting censoring . . . 13
- 2.4 Accounting for censoring . . . 17
- 2.5 Neglecting or accounting for censoring? . . . 19
- 2.6 Take-home message . . . 21

3 Reshaping time-to-event data **23**
- 3.1 Motivating example . . . 24
- 3.2 Transforming from WIDE to LONG GROUPED . . . 25
- 3.3 Transforming from WIDE to LONG UNGROUPED . . . 26
- 3.4 From LONG GROUPED to LONG UNGROUPED (and vice-versa) 27
- 3.5 From NONLINEAR REGRESSION to LONG GROUPED . . . 28

4 Time-to-event models for seed germination/emergence **31**
- 4.1 Parametric time-to-event models . . . 32
- 4.2 Nonparametric time-to-event models . . . 34
- 4.3 Kernel density estimators . . . 36
- 4.4 ML, NPMLE or KDE? . . . 38

5 Comparing the time course of events for several groups **39**
- 5.1 A motivating example . . . 40
- 5.2 Fitting several time-to-event curves . . . 41
- 5.3 Comparing non-parametric curves . . . 45

6 Fitting time-to-event models with environmental covariates **49**
- 6.1 Hydro-time-to-event models . . . 50
- 6.2 A better modelling approach . . . 53
- 6.3 Another modelling approach . . . 54

6.4 Further detail 56

7 Exploring the results of a time-to-event fit: model parameters 59

8 Predictions from time-to-event models 63
8.1 Parametric time-to-event model 63
8.2 Non-parametric time-to-event models 66
8.3 Predictions from a time-to-event model from literature 67

9 Quantiles from time-to-event models 69
9.1 Peculiarities of seed science data 69
9.2 Getting the quantiles with 'drcte' 71
9.3 Quantiles and Effective Doses (ED) 75

10 Fitting germination models in two steps 77
10.1 Fitting linear models in two steps 78
10.2 Fitting non-linear threshold models in two steps 78
10.3 Carrying over information to the 2^{nd} step 92

11 Examples and case-studies 93
11.1 Fitting hydro-thermal-time-models to seed germination data . . . 93
11.2 Fitting thermal-time-models to seed germination data 104

12 References 111

Chapter 1

Introduction

Germination/emergence assays are relatively easy to perform, by following standardised procedures, as described, e.g., by the International Seed Testing Association (see here). In short, we take a sample of seeds and we put them in an appropriate container. We put the container in the right environmental conditions (e.g., relating to humidity content and temperature) and we inspect the seeds according to a regular schedule (e.g., daily). At each inspection, we count the number of germinated/emerged seeds and remove them from the containers; inspections are performed until no new germinations/emergences are observed for a sufficient amount of time.

We see that these assays are rather simple, but, in spite of such simplicity, the process of data analysis still presents several grey areas. How should we quantify the germination/emergence process? How should we compare the germination/emergence of different seed lots?

A brief survey of literature shows that a plethora of methods is used, which is certainly encouraged by the wide availability of computer packages. Some of these methods have been around for quite a while (e.g., the use of germination indices or nonlinear regression), some others are relatively new (e.g., methods from survival analysis). It is clear that not all these methods are efficient or reliable, especially when they are used with little concern about the basic assumptions that each method makes.

Furthermore, the use of a lot of different methods of data analysis by different research groups may serve the purpose of creativity, but are we really sure that it also serves the purpose of advancing science? Wouldn't it be better if we could use the same reliable methods of data analysis, so that we could better understand each other, compare our results and pool them together?

Therefore, together with some collegues, we decided to start defining a framework for the analysis of germination and emergence data, which might help the readers to select efficient and reliable methods and, lately, improve comparisons

and communication of results within the scientific community. We decided to structure this framework as the combination of:

1. a step-by-step procedure
2. the methods to accomplish it
3. a user friendly software interface, based on a new R package.

1.1 The R packages

As we anticipated, the analyses we propose for seed germination/emergence assays can be performed by using our new R package, namely `drcte`. This package is heavily based on the 'drc' package (Ritz et al., 2015) that is a very flexible software for general model fitting purposes. Although `drc` contains, already, several basic functions for time-to-event analyses, we felt that, in order to meet the specific needs of agricultural research, it would be useful to make some further customisation and develop some additional functions, which we implemented in the 'drcte' package. Furthermore, we also created the `drcSeedGerm` package, that contains specific functions for seed germination/emergence assays.

Both the `drcte` and `drcSeedGerm` packages can be downloaded and installed from gitHub, by using the code proposed in the box below. It requires the 'devtools' package, that needs to be as well installed, if it is not already included in the system. Please, make sure you always have the latest version of both packages.

```
install.packages("devtools")
library(devtools)
install_github("OnofriAndreaPG/drcte")
install_github("OnofriAndreaPG/drcSeedGerm")
```

Once you are ready with the above packages, you can proceed to work with the rest of this tutorial.

1.2 Our purpose

We have already presented the procedure and the R package in a recent paper in the Journal Weed Science (Onofri et al., 2022; if you are interested please follow this link to the paper). The purpose of this tutorial is to expand on that manuscript and present several realistic examples, relating to seed germination/emergence assays. We would like to emphasize that these methods can be as well useful for other types of time-to-event or censored data in agriculture.

Building this site is still (and will always be…) an ongoing task, so, please forgive us if you do not find what you are looking for. In the meantime, you can take a look at the published papers from my group and at my blog, at this page.

We look forward to your comments to improve our approach. Please, drop your notes to me:

Prof. Andrea Onofri
Department of Agricultural, Food and Environmental Sciences
University of Perugia (Italy)
andrea.onofri@unipg.it

If you want to be updated, you can follow me at:

Follow

Chapter 2

What is time-to-event data?

In general, time-to-event data describe the amount of time elapsed before an event of interest occurs. In medicine, such an event is, e.g., the death, or the remission from a disease; in agriculture we are more often interested in other types of events, such as germination, emergence or flowering. Regardless the discipline, studying the time to en event of interest requires periodical inspections on a population of individuals (people, animals, seeds, plants…) to record, for each individual, the date when the event is achieved. The data resulting from such a series of periodical inspections is usually called **time-to-event data**.

Please, note that we have been talking about periodical inspections; in some cases, germination/emergence assays are performed with only one inspection at a pre-defined time. With this type of assays we can only record the proportion of germinated seeds (in general: the proportion of individuals with the event) at a given time point, while we cannot record the event time for all individuals. Therefore, this type of data are NOT time-to-event data and they will not be the object of this tutorial.

2.1 Time-to-event data are censored data

You may wonder: what's the matter with time-to-event data? Do they have anything special that needs our attention? The answer is, definitely, yes!

Indeed, with very few exceptions, time-to-event data are affected by a peculiar form of uncertainty, which takes the name of **censoring**. It relates to the fact that, due to the typical monitoring schedule, the exact time-to-event may not always be determined. Think about a germination assay, where we put, e.g., 100 seeds in a Petri dish and make daily inspections. At each inspection, we count the number of germinated seeds. In the end, what have we learnt about

the germination time of each seed? It is easy to note that we do not have exact values, we only have a set of intervals. Let's consider three possible situations.

1. If we found a germinated seed at the first inspection time, we only know that germination took place before the inspection (left-censoring).
2. If we find a germinated seed at the second (or later) inspection time, we only know that germination took place somewhere between the previous and the present inspection (interval-censoring).
3. If we find an ungerminated seed at the end of the experiment, we only know that its germination time, if any, is higher than the duration of the experiment (right-censoring).

Censoring implies a lot of uncertainty, which is additional to other more common sources of uncertainty, such as the individual-to-individual variability or random errors in the manipulation process. The problem of censoring has been widely recognised in other disciplines, such as medicine, in relation to survival data (time-to-death data). In order to address that problem, a vast body of methods has developed, which goes under the name of 'survival analysis'.

In principle, time-to-germination and time-to-emergence data are totally similar to time-to-death data, apart from the fact that we usually deal with different (and less sad) events. Unfortunately, such a connection has been largely overlooked by plant biologists and, as the consequence, the problem of censoring has been most often neglected. This is not an unfortunate practice and it is important that we become aware about the possible consequences of neglecting censoring during the data analysis step.

2.2 Motivating example: a simulated dataset

It may be useful to think about a possible mechanism by which time-to-event data arise. Let's imagine that we have a population with 85% of germinable seeds (the other ones are dormant and, therefore, they are not immediately able to germinate). The germinable fraction is composed by seeds with variables germination times, as dictated by their genotypes and environmental conditions. Let's assume that such a variability can be described by using a log-logistic distribution, with a median germination time $e = 4.5$ days and a shape parameter $b = 1.6$.

If we sample 100 seeds from that population, the sample will not necessarily reflect the characteristics of the whole population. We can do this sampling in R, by using a three-steps approach.

2.2.1 Step 1: the ungerminated fraction

First, let's simulate the number of germinated seeds, assuming a binomial distribution with a proportion of successes equal to 0.85. We use the random number

generator `rbinom()`:

```
#Monte Carlo simulation - Step 1
d <- 0.85
set.seed(1234)
nGerm <- rbinom(1, 100, d)
nGerm
## [1] 89
```

We see that, in this instance, 89 seeds germinated out of 100, which is not the expected 85%. This may be regarded as an expression of the typical random sampling variability.

2.2.2 Step 2: germination times for the germinated fraction

Second, let's simulate the germination times for these 89 germinable seeds, by drawing from a log-logistic distribution with $b = 1.6$ and $e = 4.5$. To this aim, we use the `rllogis()` function in the `actuar` package (Dutang et al., 2008):

```
#Monte Carlo simulation - Step 2
library(actuar)
b <- 1.6; e <- 4.5
Gtimes <- rllogis(nGerm, shape = b, scale = e)
Gtimes <- c(Gtimes, rep(NA, 100 - nGerm))
Gtimes
##   [1]  3.2936708  3.4089762  3.2842199  1.4401630  3.1381457
##   [6] 82.1611955  9.4906364  2.9226745  4.3424551  2.7006042
##  [11]  4.0202158  8.0519663  0.9492013  7.8199588  1.6163588
##  [16]  7.9661485  8.4641154 11.2879041  9.5014360  7.2786264
##  [21]  7.5809838 12.7421713 32.7999661  9.9691944  1.8137333
##  [26]  4.2197542  1.0218849  1.6604417 30.0352308  5.0235265
##  [31]  8.5085067  7.5367739  4.4185382 11.5555259  2.1919263
##  [36] 10.6509339  8.6857151  0.2185902  1.8377033  3.9362727
##  [41]  3.0864702  7.3804164  3.2978782  7.0100360  4.4775843
##  [46]  2.8328842  4.6721090  9.1258796  2.1485568 21.8749808
##  [51]  7.4265984  2.5148724  4.4491466 13.1132301  4.4559642
##  [56]  4.5684584  2.2556488 11.8783556  1.5338755  1.4106592
##  [61] 31.8419420  7.2666641 65.0154287  9.2798476  2.5988399
##  [66]  7.4612907  4.4048509 27.7439121  3.8257187 15.4967751
##  [71]  1.1960785 62.5152642  2.0169970 19.1134899  4.2891084
##  [76]  6.0420938 22.6521417  7.1946293  2.9028993  0.9241876
##  [81]  4.8277336 13.8068124  4.0273655 10.8651761  1.1509735
##  [86]  5.9593534  7.4009589 12.6839405  1.1698335         NA
##  [91]         NA         NA         NA         NA         NA
##  [96]         NA         NA         NA         NA         NA
```

Now, we have a vector hosting 100 germination times ('Gtimes'). Please, note that I also added 11 NA values, to represent non-germinable seeds.

2.2.3 Step 3: counts of germinated seeds

Unfortunately, due to the monitoring schedule, we cannot observe the exact germination time for each single seed in the sample; we can only count the seeds which have germinated between two assessment times. Therefore, as the third step, we simulate the observed counts, by assuming daily monitoring for 40 days; what we do, is a sort of 'binning' process, where we assign each seed to the time interval during which it germinated.

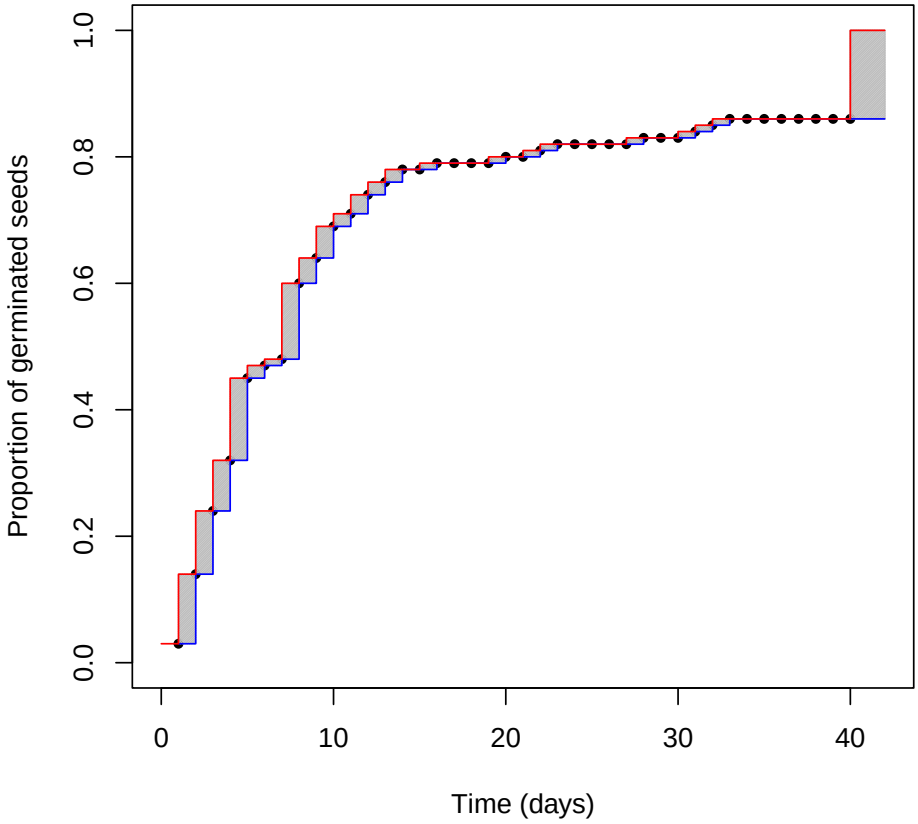
```
obsT <- seq(1, 40, by=1) #Observation schedule
count <- table( cut(Gtimes, breaks = c(0, obsT)) )
count
##
##   (0,1]   (1,2]   (2,3]   (3,4]   (4,5]   (5,6]   (6,7]   (7,8]
##       3      11      10       8      13       2       1      12
##   (8,9]  (9,10] (10,11] (11,12] (12,13] (13,14] (14,15] (15,16]
##       4       5       2       3       2       2       0       1
## (16,17] (17,18] (18,19] (19,20] (20,21] (21,22] (22,23] (23,24]
##       0       0       0       1       0       1       1       0
## (24,25] (25,26] (26,27] (27,28] (28,29] (29,30] (30,31] (31,32]
##       0       0       0       1       0       0       1       1
## (32,33] (33,34] (34,35] (35,36] (36,37] (37,38] (38,39] (39,40]
##       1       0       0       0       0       0       0       0
```

We can see that, e.g., 11 germinated seeds were counted at day 2; therefore they germinated between day 1 and day 2 and their real germination time is unknown, but included in the range between 1 and 2 (left-open and right-closed). This is a typical example of interval-censoring (see above).

We can also see that, in total, we counted 86 germinated seeds and, therefore, 14 seeds were still ungerminated at the end of the assay. For this simulation exercise, we know that 11 seeds were non-germinable and three seeds were germinable, but they were not allowed enough time to germinate (look at the table above: there are 3 seeds with germination times higher than 40). In real life, this is another source of uncertainty: we might be able to ascertain whether these 14 seeds are viable or not (e.g. by using a tetrazolium test), but, if they are viable, we would never be able to tell whether they are dormant or their germination time is simply longer than the duration of the assay. In real life, we can only reach an uncertain conclusion: the germination time of the 14 ungerminated seeds is comprised between 40 days to infinity; this is an example of right-censoring.

The above uncertainty affects our capability of describing the germination time-course from the observed data. We can try to picture the situation in the graph

below.



What is the real germination time-course? The red one? The blue one? Something in between? We cannot really say this from our dataset, we are uncertain. The grey areas represent the uncertainty due to censoring. Do you think that we can reasonably neglect it?

2.3 Neglecting censoring

For a moment, let's forget about those grey uncertainty areas. Let's forget about censoring; this is what it is often done in literature for germination data: we associate the observed proportion of germinated seeds to the exact time when it was observed. It is, indeed, an abuse, as the observed data tell us a different story: the observed proportion of germinated seeds might have been attained before the moment of inspection (and not necessarily at the moment of inspection). But we disregard this and fit a nonlinear regression model, i.e. a log-logistic function:

$$G(t) = \frac{d}{1 + exp\left\{-b[\log(t) - \log(e)]\right\}}$$

where G is the fraction of germinated seeds at time t, d is the germinable fraction, e is the median germination time for the germinable fraction and b is the slope around the inflection point. The above model is sygmoidally shaped and it is symmetric on a log-time scale. The three parameters are biologically relevant, as they describe the three main features of seed germination, i.e. capability (d), speed (e) and uniformity (b).

In order to fit a nonlinear regression model, we need to transform the observed data, as follows:

```
counts <- as.numeric( table( cut(Gtimes, breaks = c(0, obsT)) ) )
propCum <- cumsum(counts)/100
df <- data.frame(time = obsT, counts = counts, propCum = propCum)
df
##    time counts propCum
## 1     1      3    0.03
## 2     2     11    0.14
## 3     3     10    0.24
## 4     4      8    0.32
## 5     5     13    0.45
## 6     6      2    0.47
## 7     7      1    0.48
## 8     8     12    0.60
## 9     9      4    0.64
## 10   10      5    0.69
## 11   11      2    0.71
## 12   12      3    0.74
## 13   13      2    0.76
## 14   14      2    0.78
## 15   15      0    0.78
## 16   16      1    0.79
## 17   17      0    0.79
## 18   18      0    0.79
## 19   19      0    0.79
## 20   20      1    0.80
## 21   21      0    0.80
## 22   22      1    0.81
## 23   23      1    0.82
## 24   24      0    0.82
## 25   25      0    0.82
## 26   26      0    0.82
## 27   27      0    0.82
## 28   28      1    0.83
```

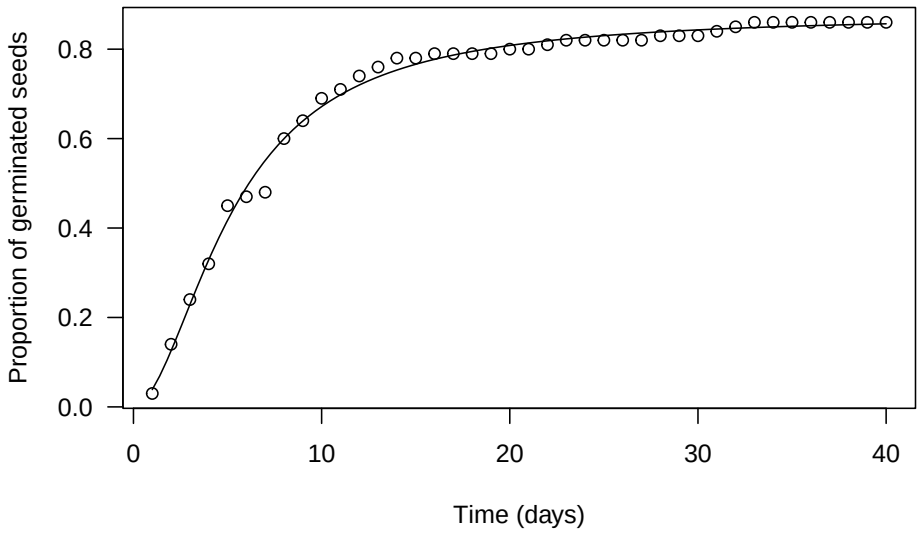
```
## 29   29      0    0.83
## 30   30      0    0.83
## 31   31      1    0.84
## 32   32      1    0.85
## 33   33      1    0.86
## 34   34      0    0.86
## 35   35      0    0.86
## 36   36      0    0.86
## 37   37      0    0.86
## 38   38      0    0.86
## 39   39      0    0.86
## 40   40      0    0.86
```

In practice, we determine the cumulative proportion (or percentage) of germinated seeds ('propCum'), which is used as the response variable, while the observation time ('time') is used as the independent variable. Then, we fit the log-logistic function by non-linear least squares regression. I'll say this again: **you can clearly see that, by doing so, we totally neglect the grey areas in the figure above, we only look at the observed points**.

Let's use the 'drm' function, in the 'drc' package (Ritz et al., 2015). The argument 'fct' is used to set the fitted function to log-logistic with three parameters (the equation above).

The `plot()` and `summary()` methods can be used to plot a graph and to retrieve the estimated parameters.

```
library(drc)
mod <- drm(propCum ~ time, data = df,
           fct = LL.3() )
plot(mod, log = "",
      xlab = "Time (days)",
      ylab = "Proportion of germinated seeds")
```




```
summary(mod)
##
## Model fitted: Log-logistic (ED50 as parameter) with lower limit at 0 (3 parms)
##
## Parameter estimates:
##
##                 Estimate Std. Error t-value   p-value
## b:(Intercept) -1.8497771  0.0702626 -26.327 < 2.2e-16 ***
## d:(Intercept)  0.8768793  0.0070126 125.044 < 2.2e-16 ***
## e:(Intercept)  5.2691575  0.1020457  51.635 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error:
##
##  0.01762168 (37 degrees of freedom)
```

We see that our estimates are very close to the real values ($b = 1.85$ vs. 1.6; $e = 5.27$ vs. 4.5; $d = 0.88$ vs. 0.86) and we also see that standard errors are rather small (the coefficient of variability goes from 1 to 4%). There is a difference in sign for b, which relates to the fact that the `LL.3()` function in 'drc' removes the minus sign in the equation above. Please, disregard this aspect, which stems from the fact that the 'drc' package is rooted in pesticide bioassays.

Do you think that this analysis is not that bad? Wait a moment. Let's see what happens if we account for censoring.

2.4 Accounting for censoring

How can we account for censoring? The answer is simple: we should use fitting methods which are specifically devised to incorporate the uncertainty due to censoring. We said that, in medicine, the body of these methods goes under the name of survival analysis and, from this framework, we can borrow a parametric survival model. However, I will not use the name 'survival model' as we are not dealing with a survival process. Instead, I will use the name **parametric time-to-event model**.

My colleagues and I have extensively talked about parametric time-to-event models in two of our recent papers and related appendices (Onofri et al., 2019; Onofri et al., 2018). Therefore, I will not go into detail, now. I will just say that time-to-event models directly consider the observed counts as the response variable. As the independent variable, they consider the extremes of each time interval ('timeBef' and 'timeAf'; see below). We immediately see two differences with nonlinear regression: (1) we do not need to transform the observed counts into cumulative proportions, and (2) by using an interval as the independent variable, we inject into the model the uncertainty due to censoring (the grey areas in the figure above). Let's reshape the dataset as shown below.

```
df <- data.frame(timeBef = c(0, obsT), timeAf = c(obsT, Inf), counts = c(as.numeric(counts), 100
df
##    timeBef timeAf counts
## 1        0      1      3
## 2        1      2     11
## 3        2      3     10
## 4        3      4      8
## 5        4      5     13
## 6        5      6      2
## 7        6      7      1
## 8        7      8     12
## 9        8      9      4
## 10       9     10      5
## 11      10     11      2
## 12      11     12      3
## 13      12     13      2
## 14      13     14      2
## 15      14     15      0
## 16      15     16      1
## 17      16     17      0
## 18      17     18      0
## 19      18     19      0
## 20      19     20      1
## 21      20     21      0
## 22      21     22      1
```

```
## 23      22     23      1
## 24      23     24      0
## 25      24     25      0
## 26      25     26      0
## 27      26     27      0
## 28      27     28      1
## 29      28     29      0
## 30      29     30      0
## 31      30     31      1
## 32      31     32      1
## 33      32     33      1
## 34      33     34      0
## 35      34     35      0
## 36      35     36      0
## 37      36     37      0
## 38      37     38      0
## 39      38     39      0
## 40      39     40      0
## 41      40    Inf     14
```

Can you see the difference? Please, note that we also added the ungerminated seeds at the end, with an uncertainty interval going from 40 days to infinity.

Time-to-event models can be easily fitted by using the `drm()` function in the 'drc' package, although we should specify that we want to use a time-to-event method, by setting the `type = "event"` argument. More simply, we can use the `drmte()` function in the 'drcte' package, that is a specific package for time-to-event methods. Both the functions use the same syntax and, with respect to nonlinear regression, there are some important differences in the model call. In particular, a nonlinear regression model is defined as:

```
CumulativeProportion ~ timeAf
```

On the other hand, a time-to-event model is defined as:

```
Count ~ timeBef + timeAf
```

The full model call is shown below, both with `drm()` and with `drmte()`

```
#Time-to-event model
library(drcte)
# modTE <- drm(counts ~ timeBef + timeAf, data = df,
#              fct = LL.3(), type = "event")
modTE <- drmte(counts ~ timeBef + timeAf, data = df,
               fct = LL.3())
summary(modTE)
##
## Model fitted: Log-logistic (ED50 as parameter) with lower limit at 0
##
```

```
## Robust estimation: no
##
## Parameter estimates:
##
##                Estimate Std. Error t-value   p-value
## b:(Intercept) -1.826006   0.194579 -9.3844 < 2.2e-16 ***
## d:(Intercept)  0.881476   0.036928 23.8701 < 2.2e-16 ***
## e:(Intercept)  5.302109   0.565273  9.3797 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

With respect to the nonlinear regression fit, the estimates from a time-to-event fit are very similar, but the standard errors are much higher (the coefficient of variability now goes from 4 to 11%). That was expected: we have added the uncertainty due to censoring.

2.5 Neglecting or accounting for censoring?

You may wonder which of the two analysis is right and which is wrong. We cannot say this from just one dataset. However, we can repeat the Monte Carlo simulation above to extract 1000 samples, fit the model by using the two methods and retrieve parameter estimates and standard errors for each sample and method. We do this by using the code below (please, be patient… it may take some time).

```
GermSampling <- function(nSeeds, timeLast, stepOss, e, b, d){

    #Draw a sample as above
    nGerm <- rbinom(1, nSeeds, d)
    Gtimes <- rllogis(nGerm, shape = b, scale = e)
    Gtimes <- c(Gtimes, rep(Inf, 100 - nGerm))


    #Generate the observed data
    obsT <- seq(1, timeLast, by=stepOss)
    counts <- as.numeric( table( cut(Gtimes, breaks = c(0, obsT)) ) )
    propCum <- cumsum(counts)/nSeeds
    timeBef <- c(0, obsT)
    timeAf <- c(obsT, Inf)
    counts <- c(counts, nSeeds - sum(counts))

    #Calculate the T50 with two methods
    mod <- drm(propCum ~ obsT, fct = LL.3() )
    modTE <- drm(counts ~ timeBef + timeAf,
```

```
            fct = LL.3(), type = "event")
    c(b1 = summary(mod)[[3]][1,1],
      ESb1 = summary(mod)[[3]][1,2],
      b2 = summary(modTE)[[3]][1,1],
      ESb2 = summary(modTE)[[3]][1,2],
      d1 = summary(mod)[[3]][2,1],
      ESd1 = summary(mod)[[3]][2,2],
      d2 = summary(modTE)[[3]][2,1],
      ESd2 = summary(modTE)[[3]][2,2],
      e1 = summary(mod)[[3]][3,1],
      ESe1 = summary(mod)[[3]][3,2],
      e2 = summary(modTE)[[3]][3,1],
      ESe2 = summary(modTE)[[3]][3,2] )
}

set.seed(1234)
result <- data.frame()
for (i in 1:1000) {
  res <- GermSampling(100, 40, 1, 4.5, 1.6, 0.85)
  result <- rbind(result, res)
}
names(result) <- c("b1", "ESb1", "b2", "ESb2",
                   "d1", "ESd1", "d2", "ESd2",
                   "e1", "ESe1", "e2", "ESe2")
result <- result[result$d2 > 0,]
```

We have stored our results in the data frame 'result'. The means of estimates obtained for both methods should be equal to the real values that we used for the simulation, which will ensure that estimators are unbiased. The means of standard errors (in brackets, below) should represent the real sample-to-sample variability, which may be obtained from the Monte Carlo standard deviation, i.e. from the standard deviation of the 1000 estimates for each parameter and method.

	b	d	e
Nonlinear regression	1.63 (0.051)	0.85 (0.006)	4.55 (0.086)
Time-to-event method	1.62 (0.187)	0.85 (0.041)	4.55 (0.579)
Real values	1.60 (0.188)	0.85 (0.041)	4.55 (0.593)

We clearly see that both nonlinear regression and the time-to-event method lead to unbiased estimates of model parameters. However, standard errors from nonlinear regression are severely biased and underestimated. On the contrary, standard errors from time-to-event method are unbiased.

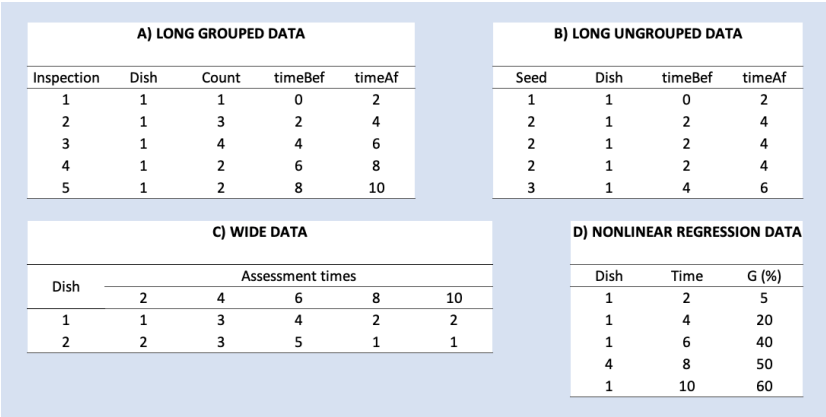
2.6 Take-home message

Censoring is peculiar to germination assays and other time-to-event studies. It may have a strong impact on the reliability of our standard errors and, consequently, on hypotheses testing. Therefore, censoring should never be neglected and time-to-event methods should necessarily be used for data analyses. The body of time-to-event methods often goes under the name of 'survival analysis', which creates a direct connection between survival data and germination/emergence data.

Chapter 3

Reshaping time-to-event data

Let's now open a parenthesis about the structure of germination/emergence data. To our experience, seed scientists are used to storing their datasets in several formats, that may not be immediately usable with the 'drcte' and 'drc' packages, which this tutorial is built upon. The figure below shows some of the possible formats that I have often encountered in my consulting work.



A) LONG GROUPED DATA

Inspection	Dish	Count	timeBef	timeAf
1	1	1	0	2
2	1	3	2	4
3	1	4	4	6
4	1	2	6	8
5	1	2	8	10

B) LONG UNGROUPED DATA

Seed	Dish	timeBef	timeAf
1	1	0	2
2	1	2	4
2	1	2	4
2	1	2	4
3	1	4	6

C) WIDE DATA

Dish	Assessment times				
	2	4	6	8	10
1	1	3	4	2	2
2	2	3	5	1	1

D) NONLINEAR REGRESSION DATA

Dish	Time	G (%)
1	2	5
1	4	20
1	6	40
4	8	50
1	10	60

Figure 3.1: Possible data structures for seed germination/emergence data

Both the 'drcte' and 'drc' packages require that the data is stored in LONG GROUPED format, as shown in the figure above (panel A, top left). We have already mentioned in the previous section that each row represents a time interval, the columns 'timeBef' and 'timeAf' represent the beginning and ending of the interval, while the column 'count' represents the seeds that germinated/emerged

during that interval of time. Other columns may be needed, to represent, e.g., the randomisation unit (e.g., Petri dish) within which the count was made and the experimental treatment level that was allocated to that unit.

Apart from the LONG GROUPED format, other time-to-event packages, such as 'survival' (Therneau, 2021) or 'interval' (Fay and Shaw, 2010) require a different data format, which could be named as LONG UNGROUPED (Figure above, panel B, top right). In this format, each row represents a seed, while the columns 'timeBef' and 'timeAf' represent the beginning and ending of the interval during which that seed germinated/emerged. The column 'count' is not necessary, while other columns may be necessary, as for the LONG GROUPED format. Apart from the above mentioned 'survival' and 'interval' packages, this format is also compatible with the 'drcte' package.

Both the GROUPED and UNGROUPED formats have two basic advantages: (i) they can be used with all types of time-to-event data and (ii) they obey to the principles of tidy data (Wickham, 2016). However, to my experience, seed scientists often use other formats, such as the WIDE format (Figure above, panel C, bottom left) or the NONLINEAR REGRESSION format (Figure above, panel D, bottom right), which need to be preliminary transformed into one of the LONG GROUPED or LONG UNGROUPED formats.

In order to ease the transition from traditional methods of data analysis to time-to-event methods, we decided to develop a couple of service functions that can be used to reshape the data to a format that is more suitable for time-to-event analyses. Let's see how to do this by using a 'real-life' example.

3.1 Motivating example

Seeds of *L. ornithopodioides* were collected from natural populations, at three different maturation stages, i.e. at 27 Days After Anthesis (DAA), when seeds were still green (Stage A), at 35 DAA, when seeds were brown and soft (Stage B) and at 41 DAA, when seeds were brown and moderately hard (Stage C). Germination assays were performed by placing four replicates of 25 seeds on filter paper (Whatman no. 3) in 9-cm diameter Petri dishes, in the dark and at a constant temperature of 20°C. The filter paper was initially moistened with 5 mL of distilled water and replenished as needed during the assay. Germinated seeds were counted daily over 15 days and removed from the Petri dishes. This dataset is a subset of a bigger dataset, aimed at assessing the time when the hard coat imposes dormancy in seeds of different legume species (Gresta et al., 2011).

The authors of the above study sent me the above dataset in WIDE format, where the rows represented each Petri dish and the columns represented all information about each dish, including the counts of germinated seeds, which were listed in different columns. The dataset is shown in the table below and it

is available as the 'lotusOr' dataframe in the 'drcte' package.

```
##  Stage Dish 1  2 3 4 5 6 7 8 9 10 11 12 13 14 15
##      A    1 0  0 0 0 0 1 2 1 1  1  3  1  1  4  2
##      A    2 0  0 0 0 0 1 1 1 1  2  1  2  3  1  1
##      A    3 0  0 0 0 0 3 4 4 3  1  2  1  1  0  1
##      A    4 0  0 0 0 0 1 3 1 2  1  2  1  3  1  1
##      B    5 0  0 0 4 1 2 1 1 0  1  2  2  3  2  1
##      B    6 0  0 0 4 2 2 1 0 1  1  3  4  3  4  0
##      B    7 0  0 0 4 0 4 1 2 1  1  3  3  2  2  0
##      B    8 0  0 0 4 3 2 2 3 1  1  1  1  1  0  0
##      C    9 1 10 1 2 1 2 3 2 0  0  0  0  1  0  0
##      C   10 1 10 4 1 4 0 1 1 0  0  0  0  1  0  1
##      C   11 1 11 5 1 2 2 1 0 1  0  0  1  0  0  0
##      C   12 0 16 1 2 2 1 1 0 0  1  0  0  0  1  0
```

The WIDE format is handy for swift calculations with a spreadsheet, but, in general, it is not ok, as: (1) it does not obey to the principle of tidy data (Wickham, 2016); (2) it is not generally efficient and it cannot be used with a set of germination assays with different monitoring schedules. Therefore, facing a dataset in the WIDE format, we need to reshape it into a LONG format.

3.2 Transforming from WIDE to LONG GROUPED

For such a reshaping, we could use one of the available R functions, such us `pivot_longer()` in the 'tidyverse' or `melt()` in the 'reshape' package. However, when reshaping the data, it is also useful to make e few transformations, such as producing cumulative counts and proportions, that might be useful to plot graphs, or for other purposes. Therefore, we developed the `melt_te()` function; let's look at its usage, in the box below.

```
datasetG <- melt_te(lotusOr, count_cols = 3:17, treat = Stage,
                    monitimes = 1:15, n.subjects = rep(25,12))
head(datasetG, 16)
##    Stage Units timeBef timeAf count nCum propCum
## 1      A     1       0      1     0    0    0.00
## 2      A     1       1      2     0    0    0.00
## 3      A     1       2      3     0    0    0.00
## 4      A     1       3      4     0    0    0.00
## 5      A     1       4      5     0    0    0.00
## 6      A     1       5      6     1    1    0.04
## 7      A     1       6      7     2    3    0.12
## 8      A     1       7      8     1    4    0.16
## 9      A     1       8      9     1    5    0.20
## 10     A     1       9     10     1    6    0.24
## 11     A     1      10     11     3    9    0.36
```

```
## 12     A     1      11     12     1    10    0.40
## 13     A     1      12     13     1    11    0.44
## 14     A     1      13     14     4    15    0.60
## 15     A     1      14     15     2    17    0.68
## 16     A     1      15    Inf     8    NA      NA
```

The `melt_te()` function requires the following arguments:

1. data: the original dataframe
2. count_cols: the positions of the columns in 'data', listing, for each dish, the counts of germinated seeds at each assessment time (columns)
3. treat: the columns in 'data', listing, for each dish, the levels of each experimental treatment
4. monitimes: a vector of monitoring times that needs to be of the same length as the argument 'count_cols'
5. subjects: a vector with the number of viable seeds per dish, at the beginning of the assay. If this argument is omitted, the function assumes that such a number is equal to the total count of germinated seeds in each dish

The functions outputs a dataframe in LONG format, where the initial columns code for the experimental treatments, the column 'Units' represents the original rows (Petri dishes), 'timeAf' represents the time at which the observation was made, 'timeBef' represents the time at which the previous observation was made, 'count' represents the number of seeds that germinated between 'timeBef' and 'timeAf', 'nCum' is the cumulative count and 'propCum' is the cumulative proportion of germinated seeds. An extra row is added for the ungerminated seeds at the end of the assay, with 'timeBef' equal to the final assessment time and 'timeAf' equal to ∞.

3.3 Transforming from WIDE to LONG UNGROUPED

The LONG GROUPED format is good for the packages 'drc' and 'drcte'. However, we might be interested in performing data analyses within the framework of survival analysis, e.g. with the 'survival' or 'interval' packages, that require the LONG UNGROUPED format. In order to reshape the original dataset into the LONG UNGROUPED format, we can use the same `melt_te()` function, by setting the argument `grouped = FALSE`. An example is given in the box below.

```
datasetU <- melt_te(lotusOr, count_cols = 3:17, treat = 1,
                    monitimes = 1:15, n.subjects = rep(25,12), grouped = F)
head(datasetU, 16)
##    Stage Units timeBef timeAf
## 1      A     1       5      6
## 2      A     1       6      7
```

```
## 3      A     1       6      7
## 4      A     1       7      8
## 5      A     1       8      9
## 6      A     1       9     10
## 7      A     1      10     11
## 8      A     1      10     11
## 9      A     1      10     11
## 10     A     1      11     12
## 11     A     1      12     13
## 12     A     1      13     14
## 13     A     1      13     14
## 14     A     1      13     14
## 15     A     1      13     14
## 16     A     1      14     15
```

3.4 From LONG GROUPED to LONG UNGROUPED (and vice-versa)

If necessary, we can easily reshape back and forth from the GROUPED and UNGROUPED formats, by using the funtions `ungroup_te` and `group_te()`. See the sample code below.

```
# From LONG GROUPED to LONG UNGROUPED
datasetU2 <- ungroup_te(datasetG, count)[,-c(5, 6)]
head(datasetU2, 16)
##    Stage Units timeBef timeAf
## 1      A     1       5      6
## 2      A     1       6      7
## 3      A     1       6      7
## 4      A     1       7      8
## 5      A     1       8      9
## 6      A     1       9     10
## 7      A     1      10     11
## 8      A     1      10     11
## 9      A     1      10     11
## 10     A     1      11     12
## 11     A     1      12     13
## 12     A     1      13     14
## 13     A     1      13     14
## 14     A     1      13     14
## 15     A     1      13     14
## 16     A     1      14     15
```

```
# From LONG UNGROUPED to LONG GROUPED
datasetG2 <- group_te(datasetU)
head(datasetG2, 16)
##    Stage Units timeBef timeAf count
## 1      A     1       5      6     1
## 2      A     1       6      7     2
## 3      A     1       7      8     1
## 4      A     1       8      9     1
## 5      A     1       9     10     1
## 6      A     1      10     11     3
## 7      A     1      11     12     1
## 8      A     1      12     13     1
## 9      A     1      13     14     4
## 10     A     1      14     15     2
## 11     A     1      15    Inf     8
## 12     A     2       5      6     1
## 13     A     2       6      7     1
## 14     A     2       7      8     1
## 15     A     2       8      9     1
## 16     A     2       9     10     2
```

3.5 From NONLINEAR REGRESSION to LONG GROUPED

In other instances, it may happen that the dataset was prepared as required by nonlinear regression analysis, i.e. listing the cumulative number of germinated seeds at each inspection time. The following Table shows an example for the first Petri dish, as available in the 'lotusCum' dataframe in the 'drcte' package.

```
##  Stage Dish Time nCum
##      A    1    1    0
##      A    1    2    0
##      A    1    3    0
##      A    1    4    0
##      A    1    5    0
##      A    1    6    1
##      A    1    7    3
##      A    1    8    4
##      A    1    9    5
##      A    1   10    6
##      A    1   11    9
##      A    1   12   10
##      A    1   13   11
##      A    1   14   15
```

```
##      A    1   15   17
```

In this situation, we need to 'decumulate' the counts and add the beginning of each inspection interval (e.g., 'timeBef'). We can easily do this by using the `decumulate_te()` function in 'drcte'. An example is given in the box below.

```
dataset_sd <- decumulate_te(lotusCum,
                            resp = nCum,
                            treat = Stage,
                            monitimes = Time,
                            units = Dish,
                            n.subjects = rep(25, 12),
                            type = "count")
dataset_sd <- decumulate_te(lotusCum,
                            resp = Prop,
                            treat = Stage,
                            monitimes = Time,
                            units = Dish,
                            n.subjects = rep(25, 12),
                            type = "proportion")
```

The `decumulate_te()` function requires the following arguments:

1. resp: the column containing the counts/proportions of germinated seeds
2. treat: the columns listing, for each row of data, the corresponding level of experimental factors (one factor per column)
3. monitimes: the column listing monitoring times
4. units: the column listing the randomisation units
5. subjects: a vector listing the number of viable seeds, at the beginning of the assay, for each randomisation unit.
6. type: a value specifying if 'resp' contains 'counts' or 'proportions'

We do hope that, with these functions, we can menage to ease your transition from traditional methods of data analysis to time-to-event methods, for seed germination/emergence assays.

Chapter 4

Time-to-event models for seed germination/emergence

The individual seeds within a population do not germinate/emerge altogether at the same moment; this is an undisputed fact, resulting from seed-to-seed variability in germination/emergence time. Accordingly, the primary reason why we organise germination assays is to describe the progress to germination for the whole population, by using some appropriate time-to-event model.

What is a time-to-event model? It is a model that describes the probability that the event (germination/emergence, in our case) occurs at any time t or before that time:

$$P(t) = \Phi(T \leq t)$$

In practice, it may be easier to think that $P(t)$ is the proportion of germinated/emerged seeds at time t. In statistical terms, Φ is a Cumulative Distribution Function (CDF), with the following characteristics:

1. the time t is constrained from 0 to $+\infty$
2. the response $P(t)$ is constrained from 0 to 1
3. the response $P(t)$ is monotonically increasing
4. due to the possible presence of a final fraction of individuals without the event (e.g., ungerminated seeds), $P(t)$ may not necessarily reach 1.

The first task to fit a time-to-event model is to select a form for Φ. In general, we have three possibilities:

1. a parametric maximum likelihood model (ML)

2. a non-parametric maximum likelihood model (NPMLE)
3. a kernel density estimator (KDE)

Let's have a closer look at those three options.

4.1 Parametric time-to-event models

The CDF for parametric time-to-event models is characterised by a pre-defined, usually sigmoidal, shape. Right-skewed CDFs have proven useful, such as the log-normal, log-logistic or Weibull, which are only defined for $t > 0$ (see #1 above). These CDFs contain a location (e) and a scale (b) parameter: the former is a measure of central tendency (e.g., the median for the log-logistic and log-normal CDFs) while the latter is a measure of how fast the curve grows during time. For seed germination/emergence, most often, a third parameter is necessary, to describe the fraction of dormant or nonviable seeds ($0 < d \leq 1$). As an example, we show a log-logistic CDF, that is also used in ecotoxicology, for dose-response studies:

$$P(t) = \frac{d}{1 + exp\left\{b[\log(t) - log(e)]\right\}}$$

Fitting the above parametric model implies that, based on the observed data, we need to assign a specific value to the parameters b, d and e, so that an appropriate likelihood function is maximised (Maximum Likelihood estimation). Relating to the estimation process, we should necessarily take into account that germination/emergence data are censored data; neglecting this fact has some important consequences, as I have described in this post.

In practice, parametric time-to-event models can be fitted by the `drmte()` function in the 'drcte' package. As an example, let's consider a factitious dataset relating to an assay where the germinations of 30 seeds were counted daily for 15 days. A log-logistic time-to-event model can be easily fit as follows:

```
dataset <- read.csv("https://www.casaonofri.it/_datasets/oneFlush.csv")
head(dataset)
##   timeBef timeAf counts
## 1       0      1      3
## 2       1      2      2
## 3       2      3      3
## 4       3      4      4
## 5       4      5      3
## 6       5      6      2
library(drcte)
te.mod <- drmte(counts ~ timeBef + timeAf, fct = LL.3(),
                data = dataset)
# Alternative
```

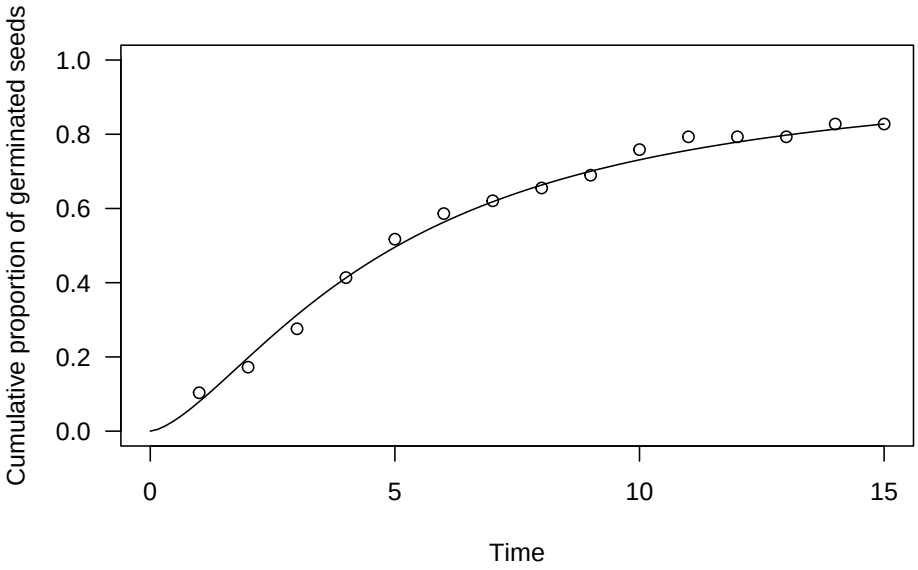
```
# te.mod <- drm(counts ~ timeBef + timeAf, fct = LL.3(),
#                 data = dataset, type = "event")
```

We can see that we have directly considered the observed counts as the response variable, with no preliminary transformation. Furthermore, we have two predictors that represent the extremes of each time interval ('timeBef' and 'timeAf'), by which we associate each count to the whole uncertainty interval during which germinations took place and not to a precise time instant. In other words, we fully respect our censored data.

The very same model can be fitted by using the `drm()` function in the 'drc' package, although we should add the argument `type = "event"`, as this latter package is not specific to time-to-event methods.

The usual `coef()`, `summary()`, `print()` and `plot()` methods can be used for 'drcte' objects as for any other model object in R.

```
summary(te.mod)
##
## Model fitted: Log-logistic (ED50 as parameter) with lower limit at 0
##
## Robust estimation: no
##
## Parameter estimates:
##
##               Estimate Std. Error t-value   p-value
## b:(Intercept) -1.52553    0.44166 -3.4541 0.0005521 ***
## d:(Intercept)  0.97842    0.16027  6.1048  1.03e-09 ***
## e:(Intercept)  4.91400    1.68058  2.9240 0.0034558 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
plot(te.mod, ylim = c(0, 1), xlim = c(0, 15),
     xlab = "Time", ylab = "Cumulative proportion of germinated seeds")
```



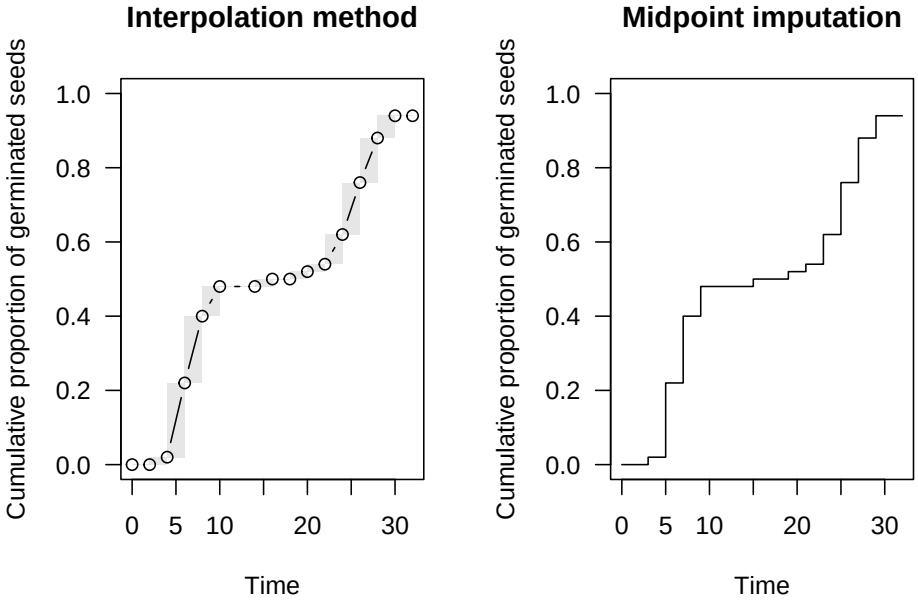
4.2 Nonparametric time-to-event models

In some cases we are not willing to assume that the germination curve has a certain predefined shape, but we need an extra degree of flexibility. For example, emergences may proceed in successive flushes that are not easily described by using a sigmoidal curve. In these cases, we can fit a non-parametric model, whose shape is not pre-defined, but it is built by closely following the observed data. In survival analyses, time-to-event curves for interval censored data are estimated by using Nonparametric Maximum Likelihood Estimators (NPMLE), that are potentially interesting also for plant science.

For example, we can consider another factitious dataset, where the germination took place in two distinct flushes; a non-parametric maximum likelihood model can be fit by using the `NPMLE()` function, as shown in the box below.

```
dataset <- read.csv("https://www.casaonofri.it/_datasets/twoFlushes.csv")
head(dataset)
##   timeBef timeAf nEmerg
## 1       0      2      0
## 2       2      4      1
## 3       4      6     10
## 4       6      8      9
## 5       8     10      4
## 6      10     12      0
te.npmle <- drmte(nEmerg ~ timeBef + timeAf, fct = NPMLE(),
                  data = dataset)
summary(te.npmle)
```

```
##
## Model fitted: NPML estimator for time-to-event data
##
## Robust estimation: no
##
## Turnbull's intervals and masses:
##
##             count   pdf   cdf Naive.SE
## 1.(2,4]      1.00  0.02  0.02   0.0198
## 1.(4,6]     10.00  0.20  0.22   0.0586
## 1.(6,8]      9.00  0.18  0.40   0.0693
## 1.(8,10]     4.00  0.08  0.48   0.0707
## 1.(14,16]    1.00  0.02  0.50   0.0707
## 1.(18,20]    1.00  0.02  0.52   0.0707
## 1.(20,22]    1.00  0.02  0.54   0.0705
## 1.(22,24]    4.00  0.08  0.62   0.0686
## 1.(24,26]    7.00  0.14  0.76   0.0604
## 1.(26,28]    6.00  0.12  0.88   0.0460
## 1.(28,30]    3.00  0.06  0.94   0.0336
## 1.(32,Inf)   3.00  0.06  1.00   0.0000
par(mfrow = c(1,2))

plot(te.npmle, ylim = c(0, 1),
     xlab = "Time", ylab = "Cumulative proportion of germinated seeds",
     main = "Interpolation method", npmle.points = T)
plot(te.npmle, ylim = c(0, 1),
     xlab = "Time", ylab = "Cumulative proportion of germinated seeds",
     npmle.type = "midpoint", shading = F,
     main = "Midpoint imputation")
```



With NPMLE, the time-to-event curve is only defined at the end of each time interval, while it is undefined elsewhere and it is (optionally) represented by a shaded area (Figure above, left). This shaded area reflects the uncertainty due to censoring.

Although we cannot know at what moment the nonparametric curve went up within the grey interval, we can make some reasonable assumptions. For example, we could assume that events are evenly spread within the interval, which is the approach taken in the 'interval' package (Figure above, left panel). In the 'survival' package and, most commonly, in survival analysis, it is assumed that the curve goes up in the middle of the interval (midpoint imputation; Figure above, right panel).

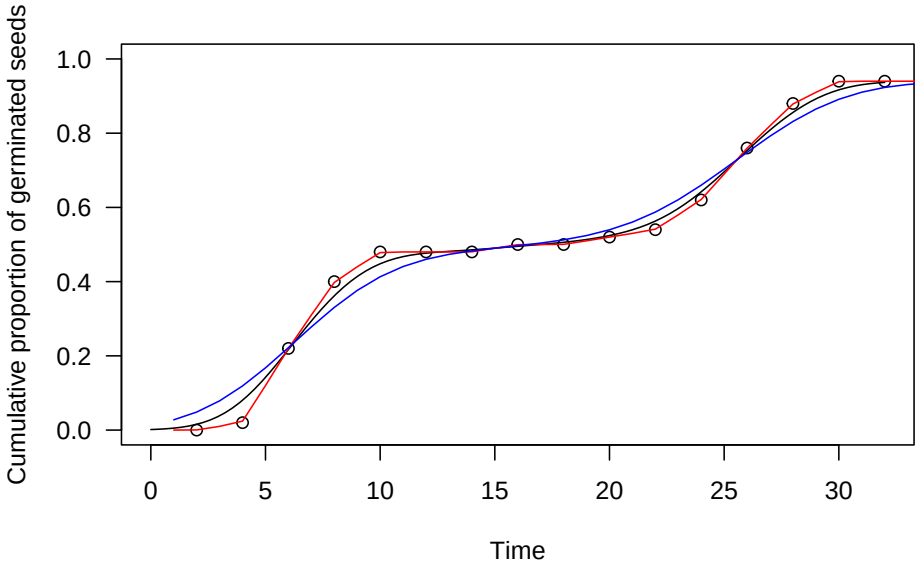
4.3 Kernel density estimators

NPMLEs are very flexible and they can be used to describe the progress to germination/emergence with, virtually, every type of datasets, also when the events took place in distinct flushes. However, we may be reluctant to accept a time-to-event curve with a stairstep shape, especially for prediction purposes. A further possibility to describe the time to event curve with a smooth and flexible model is to use a so-called Kernel Density Estimator (KDE). A KDE is built by considering the observed data, a Kernel function (usually gaussian) and a bandwith value, that controls the degree of smoothing; a nice post to see how Kernel density estimation works in practice can be found at this link here.

In the box below we show that a KDE can be fitted to the observed data (same

example as above) by using the `drmte()` function and setting the 'fct' argument to `KDE()`. Please, note that this type of KDE is specifically modified to comply with censoring.

```
te.kde <- drmte(nEmerg ~ timeBef + timeAf, fct = KDE(),
                data = dataset)
summary(te.kde)
##
## Model fitted: Kernel estimator for the distribution function
##
## Robust estimation: no
##
## Bandwidth estimates:
##
##                Estimate
## h:(bandwidth)   1.7871
plot(te.kde, ylim = c(0, 1),
     xlab = "Time", ylab = "Cumulative proportion of germinated seeds")
paf <- KDE.fun(seq(1,35, 1), dataset$timeBef, dataset$timeAf, dataset$nEmerg,
               h = 0.5)
lines(paf ~ seq(1,35, 1), col = "red")
paf <- KDE.fun(seq(1,35, 1), dataset$timeBef, dataset$timeAf, dataset$nEmerg,
               h = 3.0)
lines(paf ~ seq(1,35, 1), col = "blue")
```



We have augmented the above graph with three estimators: the blue one has a bandwidth $h = 3$, the red one has $b = 0.5$ and the black one has $b = 1.7871$. Our aim was to show the effect of bandwidth selection on the resulting time-to-event

curve, although it is usually necessary to apply an appropriate algorithm for the selection. The function `drmte()`, by default, uses the AMISE method and, unless you know what you are doing, we recommend that you stick to such an algorithm.

4.4 ML, NPMLE or KDE?

There is no rule to select the type of time-to-event model for seed germination/emergence and you will have to make your own choice and defend it at the publication stage. As a swift suggestion, I would say that a parametric model is to be preferred, unless it shows some visible signs of lack of fit.

Whatever model you select, fitting a time-to-event model may be the most unambiguous way to describe the progress to germination/emergence. In a future post, we will see that we can also compare time-to-event models for different experimental treatments and or environmental conditions, which is, most often, the central step in our process of data analyses, for seed germination/emergence assays.

Chapter 5

Comparing the time course of events for several groups

Very often, seed scientists need to compare the germination behavior of different seed populations, e.g., different plant species, or one single plant species submitted to different temperatures, light conditions, priming treatments and so on. How should such a comparison be performed? For example, if we have submitted several seed samples to different environmental conditions, how do we decide whether the germinative response is affected by those environmental conditions?

If the case that we have replicates for all experimental treatments, e.g. several Petri dishes, one possible line of attack is to take a summary measure for each dish and use that for further analyses, in a two-steps fashion. For example, we could take the total number of germinated seeds (Pmax) in each dish and use the resulting values to parameterise some sort of ANOVA-like model and test the significance of all effects.

This method of data analysis is known as 'response feature analysis' and it may be regarded as 'very traditional', in the sense that it is often found in literature; although it is not wrong, it is, undoubtedly, sub-optimal. Indeed, two seed lots submitted to different treatments may show the same total number of germinated seeds, but a different velocity or uniformity of germination. In other words, if we only consider, e.g., the Pmax, we can answer the question: "do the seed lots differ for their germination capability?", but not the more general question: "are the seed lots different?".

In order to answer this latter question, we should consider the entire time-course of germination and not only one single summary statistic. **In other words, we need a method to fit and compare several time-to-event curves.**

5.1 A motivating example

Let's take a practical approach and start from an example: a few years ago, some colleagues of mine studied the germination behavior of a plant species (*Verbascum arcturus*), in different conditions. In detail, they considered the factorial combination of two storage periods (LONG and SHORT storage) and two temperature regimes (FIX: constant daily temperature of 20°C; ALT: alternating daily temperature regime, with 25°C during daytime and 15°C during night time, with a 12:12h photoperiod). If you are a seed scientist and are interested in this experiment, you'll find detail in Catara *et al.* (2016).

If you are not a seed scientist, you may wonder why my colleagues made such an assay; well, there is evidence that, for some plant species, the germination ability improves over time, after seed maturation. Therefore, if we take seeds and store them for different periods of time, there might be an effect on their germination traits. Likewise, there is also evidence that germinations may be hindered if seed is not submitted to daily temperature fluctuations. For seed scientists, all these mechanisms are very important, as they permit to trigger the germinations when the environmental conditions are favorable for seedling survival.

Let's go back to our assay: the experimental design consisted of four experimental 'combinations' (LONG-FIX, LONG-ALT, SHORT-FIX and SHORT-ALT) and four replicates for each combination. One replicate consisted of a Petri dish, that is a small plastic box containing humid blotting paper, with 25 seeds of *V. arcturus*. In all, there were 16 Petri dishes, which were put in climatic chambers with the appropriate conditions. During the assay, my colleagues made daily inspections: germinated seeds were counted and removed from the dishes. Inspections were made for 15 days, until no more germinations could be observed.

The original dataset is available from a gitHub repository: let's load and have a look at it.

```
rm(list = ls())
datasetOr <- read.csv("https://raw.githubusercontent.com/OnofriAndreaPG/agroBioData/mas
head(datasetOr)
##   Dish Storage Temp 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
## 1    1     Low  Fix 0 0 0 0 0 0 0 0 3  4  6  0  1  0  3
## 2    2     Low  Fix 0 0 0 0 1 0 0 0 2  7  2  3  0  5  1
## 3    3     Low  Fix 0 0 0 0 1 0 0 1 3  5  2  4  0  1  3
## 4    4     Low  Fix 0 0 0 0 1 0 3 0 0  3  1  1  0  4  4
## 5    5    High  Fix 0 0 0 0 0 0 0 0 1  2  5  4  2  3  0
## 6    6    High  Fix 0 0 0 0 0 0 0 0 2  2  7  8  1  2  1
```

We have one row per Petri dish; the first three columns show, respectively, the dish number, storage and temperature conditions. The next 15 columns represent the inspection times (from 1 to 15) and contain the counts of germinated

seeds. The research question is:

Is germination behavior affected by storage and temperature conditions?

5.2 Fitting several time-to-event curves

The original dataset for our example is in a WIDE format and, as we have shown earlier in this tutorial, it is necessary to reshape it in LONG GROUPED format, by using the `melt_te()` function in the 'drcte' package.

The `melt_te()` function needs to receive the columns storing the counts ('count_cols = 4:18'), the columns storing the factor variables ('treat_cols = c("Dish", "Storage", "Temp")'), a vector of monitoring times ('monitimes = 1:15') and a vector with the total number of seeds in each Petri dish ('n.subjects = rep(25,16)').

```
library(drcte)
dataset <- melt_te(datasetOr, count_cols = 4:18,
                   treat_cols = c("Dish", "Storage", "Temp"),
                   monitimes = 1:15, n.subjects = rep(25, 16))
head(dataset, 16)
##    Dish Storage Temp Units timeBef timeAf count nCum propCum
## 1     1     Low  Fix     1       0      1     0    0    0.00
## 2     1     Low  Fix     1       1      2     0    0    0.00
## 3     1     Low  Fix     1       2      3     0    0    0.00
## 4     1     Low  Fix     1       3      4     0    0    0.00
## 5     1     Low  Fix     1       4      5     0    0    0.00
## 6     1     Low  Fix     1       5      6     0    0    0.00
## 7     1     Low  Fix     1       6      7     0    0    0.00
## 8     1     Low  Fix     1       7      8     0    0    0.00
## 9     1     Low  Fix     1       8      9     3    3    0.12
## 10    1     Low  Fix     1       9     10     4    7    0.28
## 11    1     Low  Fix     1      10     11     6   13    0.52
## 12    1     Low  Fix     1      11     12     0   13    0.52
## 13    1     Low  Fix     1      12     13     1   14    0.56
## 14    1     Low  Fix     1      13     14     0   14    0.56
## 15    1     Low  Fix     1      14     15     3   17    0.68
## 16    1     Low  Fix     1      15    Inf     8   NA      NA
```

In the resulting data frame, the column 'timeAf' contains the time when the inspection was made and the column 'count' contains the number of germinated seeds (e.g. 9 seeds were counted at day 9). These seeds did not germinate exactly at day 9; they germinated within the interval between two inspections, that is between day 8 and day 9. The beginning of the interval is given as the variable 'timeBef'. Apart from these three columns, we have the columns for the blocking factor ('Dish' and 'Units'; this latter column is added by the R function, but it

is not useful in this case) and for the treatment factors ('Storage' and 'Temp') plus two other additional columns ('nCum' and 'propCum'), which we are not going to use for our analyses.

In this case, we have reasons to believe that the germination time-course can be described by using a parametric log-logistic time-to-event model, which can be estimated by using either the `drm()` function in the 'drc' package (Ritz et al., 2019) or the `drmte()` function in the 'drcte' package (Onofri et al., 2022). In both cases, we have to include the experimental factor ('curveid' argument), to specify that we want to fit a different curve for each combination of storage and temperature.

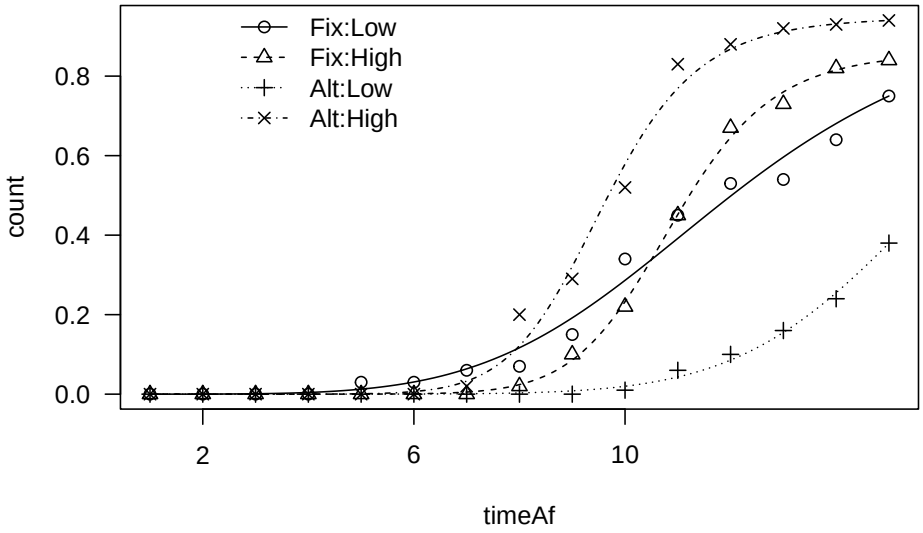
```
library(tidyverse)
library(drcte)

dataset <- dataset %>%
  mutate(across(1:3, .fns = factor))
mod1 <- drmte(count ~ timeBef + timeAf, fct = loglogistic(),
              data = dataset,
              curveid = Temp:Storage)
summary(mod1)
##
## Model fitted: Log-logistic distribution of event times
##
## Robust estimation: no
##
## Parameter estimates:
##
##              Estimate Std. Error t-value   p-value
## b:Fix:Low    4.974317   0.819632  6.0690 1.287e-09 ***
## b:Fix:High 11.476618   1.254439  9.1488 < 2.2e-16 ***
## b:Alt:Low    7.854558   5.239825  1.4990    0.1339
## b:Alt:High 10.600439   1.014061 10.4534 < 2.2e-16 ***
## d:Fix:Low    0.998474   0.150189  6.6481 2.968e-11 ***
## d:Fix:High   0.861711   0.038987 22.1027 < 2.2e-16 ***
## d:Alt:Low    1.405930   5.607576  0.2507    0.8020
## d:Alt:High   0.948113   0.024298 39.0208 < 2.2e-16 ***
## e:Fix:Low   12.009974   0.987039 12.1677 < 2.2e-16 ***
## e:Fix:High 10.906963   0.190532 57.2447 < 2.2e-16 ***
## e:Alt:Low   17.014976  13.214513  1.2876    0.1979
## e:Alt:High   9.585255   0.166937 57.4183 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In a previous post I have described a log-logistic time-to-event model (see here), which has a sygmoidal shape, with the three parameters b, d and e. These parameters represent, respectively, the slope at inflection point, the higher asymptote

(i.e. the maximum proportion of germinated seeds) and the median germination time. As we have four curves, we have a number of 12 estimated parameters.

We see that the optimization routines returns an unreasonable value for the higher asymptote for one of the curves ($d = 1.40$ with Alt:Low); it is unreasonable because the maximum proportion of germinated seeds may not exceed 1. Therefore, we should refit the model by adding a constraint ($d \leq 1$) for all the four curves. We can do so by setting the 'upperl' argument to 1 for the 5^{th} through 8^{th} estimands.

```
mod1 <- drmte(count ~ timeBef + timeAf, fct = loglogistic(),
              data = dataset,
              curveid = Temp:Storage,
              upperl = c(NA, NA, NA, NA, 1, 1, 1, 1, NA, NA, NA, NA))
summary(mod1)
##
## Model fitted: Log-logistic distribution of event times
##
## Robust estimation: no
##
## Parameter estimates:
##
##              Estimate Std. Error t-value   p-value
## b:Fix:Low    4.979665   0.818414  6.0845 1.168e-09 ***
## b:Fix:High 11.471625   1.254025  9.1478 < 2.2e-16 ***
## b:Alt:Low    8.408186   2.232712  3.7659 0.0001659 ***
## b:Alt:High 10.605807   1.014523 10.4540 < 2.2e-16 ***
## d:Fix:Low    0.997284   0.149235  6.6826 2.347e-11 ***
## d:Fix:High   0.861709   0.038999 22.0955 < 2.2e-16 ***
## d:Alt:Low    1.000000   0.881644  1.1342 0.2566920
## d:Alt:High   0.948132   0.024282 39.0468 < 2.2e-16 ***
## e:Fix:Low   12.004534   0.981279 12.2336 < 2.2e-16 ***
## e:Fix:High 10.907108   0.190613 57.2213 < 2.2e-16 ***
## e:Alt:Low   15.903079   2.872327  5.5367 3.083e-08 ***
## e:Alt:High   9.585297   0.166873 57.4406 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
plot(mod1, log = "", legendPos = c(6, 1))
```



From the graph we see that there are visible differences between the fitted curves (the legend considers the curves in alphabetical order, i.e. 1: Fix-Low, 2: Fix-High, 3: Alt-Low and 4: Alt-High). Now, the question is: could we say that those differences are only due to chance (null hypothesis)?

In order to make such a test, we could compare the logarithm of the likelihood for the fitted model with the logarithm of the likelihood for a 'reduced' model, where all curves have been pooled into one common curve for all treatment levels. The higher the log-likelihood difference, the lowest the probability that the null is true (Likelihood Ratio Test; LRT).

A LRT for parametric models can be done with the `compCDF()` function in the 'drcte' package, as shown in the box below.

```
compCDF(mod1)
##
##
## Likelihood ratio test
## NULL: time-to-event curves are equal
##
## Observed LR value:  202.8052
## Degrees of freedom:  9
## P-value:  8.551556e-39
```

We see that the LR value, that relates to the difference between the two log-likelihoods, is rather high and equal to 202; when the null is true, this LR value has an approximate Chi-square distribution; accordingly, we see that the P-level is very low and, thus, the null should be rejected.

It general, the results of LRTs should be taken with care, particularly when the observed data are not independent from one another. Unfortunately, the lack

of independence is an intrinsic characteristic of germination/emergence assays, where seeds are, most often, clustered within Petri dishes or other types of containers.

In this example, we got a very low p-level, which leaves us rather confident that the difference between time-to-event curves is significant. More generally, instead of relying on a chi-square approximation, we should better use a grouped-permutation approach. This technique is based on the idea that, when the difference between curves is not significant, we should be able to freely permute the labels (treatment level) among Petri dishes (clustering units) and, consequently, build an empirical distribution for the LR statistic under the null (permutation distribution). The p-level is related to the proportion of LR values in the permutation distribution that are higher than the observed value (i.e.: 202.8)

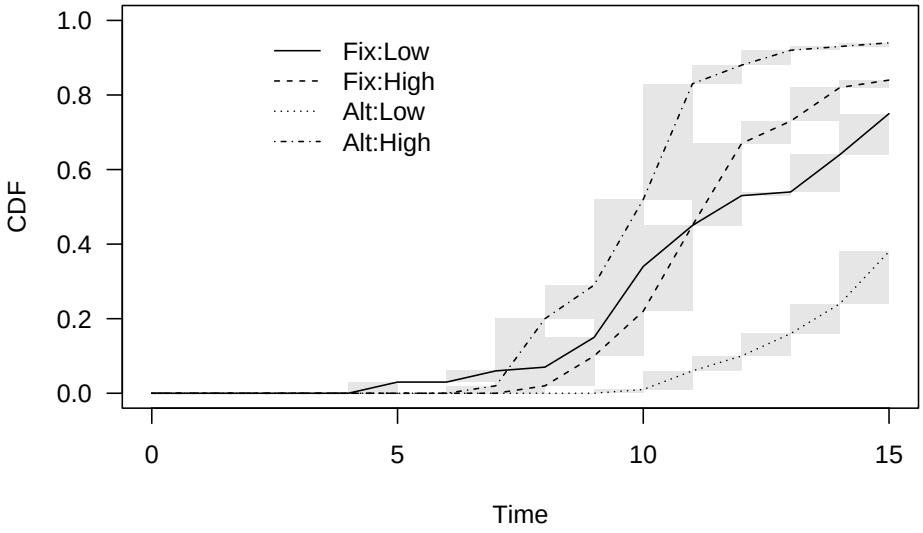
In the code below, we show how we can do this. The code is rather slow and, therefore, we should not use a very high number of permutation; the default is 199, that gives us a minimum p-value of 0.005. We see that we can confirm that the difference between curves is highly significant.

```
compCDF(mod1, type = "permutation", units = dataset$Dishes)
## Likelihood ratio test (permutation based)
## NULL: time-to-event curves are equal
##
## Observed LR value:  202.8052
## Degrees of freedom:  9
## Naive P-value:  8.551556e-39
## Permutation P-value (B = 199): 0.005
```

5.3 Comparing non-parametric curves

In the above example, we have decided to fit a parametric time-to-event model. However, in other situations, we might be interested in fitting a non-parametric time-to-event model (NPMLE; see here) and compare the curves for different treatment levels. In practice, nothing changes with respect to the approach I have shown above: first of all, we fit the NPMLEs with the following code:

```
modNP <- drmte(count ~ timeBef + timeAf, fct = NPMLE(),
               data = dataset,
               curveid = Temp:Storage)
plot(modNP, log = "", legendPos = c(6, 1))
```



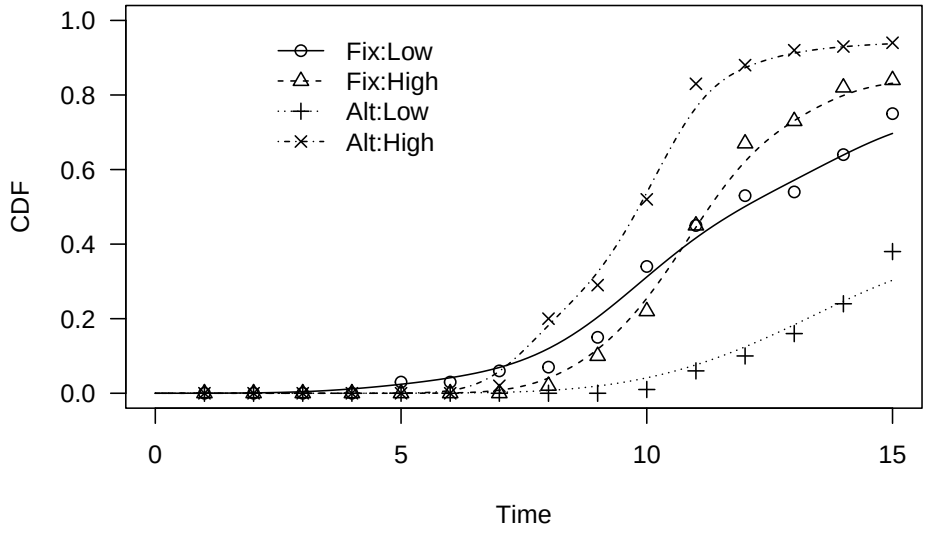
Next, we compare the non-parametric curves, in the very same fashion as above:

```
compCDF(modNP, units = dataset$Units)
## Exact Wilcoxon test (permutation form)
## NULL: time-to-event curves are equal
##
##      level   n   Scores
## 1  Fix:Low 100   2.5350
## 2 Fix:High 100   6.4600
## 3  Alt:Low 100 -51.2275
## 4 Alt:High 100  42.2325
##
## Observed T value:  44.56
## Permutation P-value (B = 199): 0.005
```

Obviously, with NPMLEs, a different test statistic is used in the background; the default one is the Wilcoxon rank sum score, although two types of log-rank scores are also implemented (Sun's scores and Finkelstein's scores; see Fay and Shaw 2010). Permutation based P-values are calculated and reported.

The approach is exactly the same with Kernel Density Estimators (KDE; see here). First we fit the four curves curves, by including the experimental factor as the 'curveid' argument:

```
modKD <- drmte(count ~ timeBef + timeAf, fct = KDE(),
               data = dataset,
               curveid = Temp:Storage)
plot(modKD, log = "", legendPos = c(6, 1))
```



Second, we compare those curves, by using the `compCDF()` function:

```
compCDF(modKD, units = dataset$Units)
## Permuting groups
## 1% 1% 2% 2% 3% 3% 4% 4% 5% 5% 6% 6% 7% 7% 8% 8% 9% 9% 10% 10% 11% 11%
## 12% 12% 13% 13% 14% 14% 15% 15% 16% 16% 17% 17% 18% 18% 19% 19% 20%
## 20% 21% 21% 22% 22% 23% 23% 24% 24% 25% 25% 26% 26% 27% 27% 28% 28%
## 29% 29% 30% 30% 31% 31% 32% 32% 33% 33% 34% 34% 35% 35% 36% 36% 37%
## 37% 38% 38% 39% 39% 40% 40% 41% 41% 42% 42% 43% 43% 44% 44% 45% 45%
## 46% 46% 47% 47% 48% 48% 49% 49% 50% 50% 51% 51% 52% 52% 53% 53% 54%
## 54% 55% 55% 56% 56% 57% 57% 58% 58% 59% 59% 60% 60% 61% 61% 62% 62%
## 63% 63% 64% 64% 65% 65% 66% 66% 67% 67% 68% 68% 69% 69% 70% 70% 71%
## 71% 72% 72% 73% 73% 74% 74% 75% 75% 76% 76% 77% 77% 78% 78% 79% 79%
## 80% 80% 81% 81% 82% 82% 83% 83% 84% 84% 85% 85% 86% 86% 87% 87% 88%
## 88% 89% 89% 90% 90% 91% 91% 92% 92% 93% 93% 94% 94% 95% 95% 96% 96%
## 97% 97% 98% 98% 99% 99% 100%
## Permutation test based on a Cramer-von-Mises type distance (Barreiro-Ures et al., 2019)
## NULL HYPOTHESIS: time-to-event curves are equal
##
##      level   n         D
## 1  Fix:Low 100 0.0579661
## 2 Fix:High 100 0.1347005
## 3  Alt:Low 100 4.1378209
## 4 Alt:High 100 3.8581487
##
## Observed D value =  2.0472
## P value =  0.005
```

In this case, a Cramér-von Mises type distance among curves is used (Barreiro-Ures et al., 2019), which, roughly speaking, is based on the integrated distance

between the KDEs for the different groups and the pooled KDE for all groups. Permutation based P-values are also calculated and reported.

Chapter 6

Fitting time-to-event models with environmental covariates

We have seen that time-to-event curves (e.g., germination or emergence curves) can be used to describe the time course of germinations/emergences for a seed lot and we have also seen that the effects of experimental factors on seed germination can be accounted for by coding a different time-to-event curve for each factor level.

In some cases, we might be interested in considering environmental variables, that are, perhaps, the most important factors to trigger germination/emergence. For example, let's consider either humidity content in the substrate, or temperature, or oxygen availability; it is clear that these variables play a fundamental role in determining germination extent and velocity and, therefore, they are very much studied by seed scientists. In principle, germination assays with environmental variables are straightforward to set up: several Petri dishes are submitted to different environmental conditions and germinations are inspected over time. What is the best method to analyse the resulting data and retrieve some important parameters, such as threshold temperatures (base, optimal or ceiling temperature) or base water potential?

It is important to anticipate that most environmental variables can be expressed on a quantitative scale; obviously when we make an experiment we are forced into selecting a subset of all possible, e.g., temperatures, such as 15, 20, 30°C, but that does not mean that we are not interested to what happens at, e.g., 18 or 22°C. From this point of view, quantitative variables are very different from qualitative variables, such as the different plant species that we have compared in previous sections of this tutorial.

In this post we will see an example of how we can account for the effects of water content in the substrate and include it in our time-to-event models. Of course, the same approach can be followed also with other types of environmental variables and, more generally, quantitative variables.

6.1 Hydro-time-to-event models

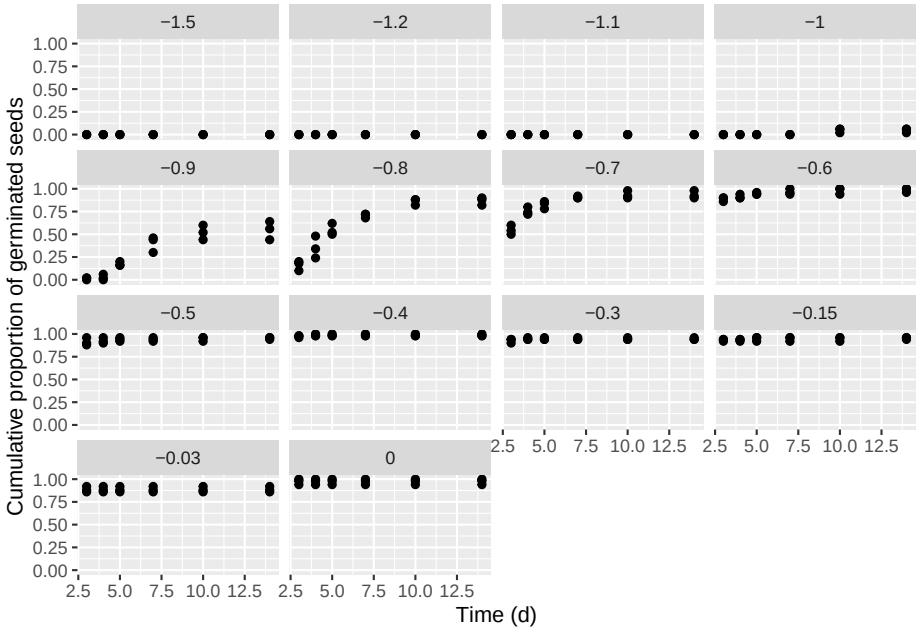
Let's consider the following example: the germination of rapeseed (*Brassica napus* L. var. *oleifera*, cv. Excalibur) was tested at fourteen different water potential levels (0, -0.03, -0.15, -0.3, -0.4, -0.5, -0.6, -0.7, -0.8, -0.9, -1, -1.1, -1.2, -1.5 MPa), which were created by using a polyethylene glycol solution (PEG 6000). For each water potential level, three replicated Petri dishes with 50 seeds each were incubated at 20°C. Germinated seeds were counted every 2-3 days for 14 days and they were removed from the dishes after germination.

The dataset was published by Pace et al. (2012) and it is available as `rape` in the `drcSeedGerm` package, which needs to be installed from github (see below). Furthermore, the package `drcte` is necessary to fit time-to-event models and it should also be installed from gitHub. The following code loads the necessary packages, loads the dataset `rape` and shows the first six lines.

```
# library(devtools)
# install_github("OnofriAndreaPG/drcSeedGerm")
# install_github("OnofriAndreaPG/drcte")
library(drcSeedGerm)
library(drcte)
library(ggplot2)
data(rape)
head(rape)
##   Psi Dish timeBef timeAf nSeeds nCum propCum
## 1   0    1       0      3     49   49    0.98
## 2   0    1       3      4      0   49    0.98
## 3   0    1       4      5      0   49    0.98
## 4   0    1       5      7      0   49    0.98
## 5   0    1       7     10      0   49    0.98
## 6   0    1      10     14      0   49    0.98
```

In the above data.frame, 'timeAf' represents the moment when germinated seeds were counted, while 'timeBef' represents the previous inspection time (or the beginning of the assay). The column 'nSeeds' is the number of seeds that germinated during the time interval between 'timeBef' and 'timeAf. The 'propCum' column contains the cumulative proportions of germinated seeds and it is not necessary for time-to-event model fitting, although we can use it for plotting purposes.

```
ggplot(rape, aes(timeAf, propCum)) +
  geom_point() +
  facet_wrap(~Psi) +
  scale_x_continuous(name = "Time (d)") +
  scale_y_continuous(name = "Cumulative proportion of germinated seeds")
```



The germination time-course is strongly affected by the water potential in the substrate, as this determines the ability of seeds to absorb water and, consequently, trigger the germination and emergence processes. Therefore, our obvious interest is to understand how the environmental factor affects the time-course of germination.

We have shown that a parametric time-to-event curve is defined as a cumulative probability function (Φ), with three parameters:

$$P(t) = \Phi\left(b, d, e\right)$$

Considering our previous post, the most obvious extension of the above model is to allow for different b, d and e value for each water potential level:

$$P(t, \Psi_i) = \Phi\left(b_i, d_i, e_i\right)$$

The first problem is that, for some water potential levels, germination did not occur and, for other levels, it occurred very quickly, so that no time-course of events could be observed (e.g., see the graph at 0 or -0.03 MPa). We say that we have 'degenerated' time-to-event curves.

If we fit those curves by using the 'curveid' argument, we are forced into fitting the same time-to-event model to all water potential levels (as shown in our previous post), and, therefore, the presence of degenerated curves provokes an error.

```
# Not run
# mod0 <- drmte(nSeeds ~ timeBef + timeAf, data = rape,
#               curveid = Psi, fct = loglogistic())
```

This problem can be circumvented by using the `separate = TRUE` argument; in this case, the different curves are fitted independent of one another and we are not tied to fitting the same model for all water potential levels. Errors are raised when trying to fit parametric time-to-event models, but they do not stop the execution in R.

```
mod1 <- drmte(nSeeds ~ timeBef + timeAf, data = rape,
              curveid = Psi, fct = loglogistic(),
              separate = TRUE)
## Error in optim(startVec, opfct, hessian = TRUE, method = optMethod, control = list(
##   non-finite finite-difference value [3]
## Error in optim(startVec, opfct, hessian = TRUE, method = optMethod, control = list(
##   non-finite value supplied by optim
## Error in optim(startVec, opfct, hessian = TRUE, method = optMethod, control = list(
##   non-finite value supplied by optim
## Error in optim(startVec, opfct, hessian = TRUE, method = optMethod, control = list(
##   non-finite value supplied by optim
coef(mod1)
##      d:-1.5      d:-1.2      d:-1.1        b:-1        d:-1
##  0.00000000  0.00000000  0.00000000 64.39202307  0.04666194
##        e:-1      b:-0.9      d:-0.9      e:-0.9      b:-0.8
##  8.36474946  5.83290900  0.55010281  5.87022056  3.73294999
##      d:-0.8      e:-0.8      b:-0.7      d:-0.7      e:-0.7
##  0.87889318  4.47105672  3.59940891  0.93592886  2.73025095
##      b:-0.6      d:-0.6      e:-0.6      b:-0.5      d:-0.5
##  1.46594380  1.00000000  0.73867461  1.09308015  0.96293123
##      e:-0.5      d:-0.4      d:-0.3     b:-0.15     d:-0.15
##  0.20862176  0.98444444  0.94333333  0.82546511  0.96143597
##     e:-0.15     d:-0.03         d:0
##  0.04345269  0.88666667  0.97333333
```

In particular, for the cases where a time-course of events cannot be estimated, the `drmte()` function resorts to fitting a simpler model, where only the d parameter is estimated (that is the maximum fraction of germinated seeds). In the box above, we can see the estimated parameters but no standard errors, which can be obtained by using the `summary()` method, although there are some statistical issues that we will consider in a following post.

6.2 A better modelling approach

The previous approach is clearly sub-optimal. First of all, the different water potential levels are assumed as independent, with no ordering and distances. In other words, we have a time-to-event curve for, e.g. -0.9 MPa and -0.8 MPa, but we have no information about the time-to-event curve for any water potential levels in between. Furthermore, we have no estimates of some relevant hydro-time parameters, such as the *base water potential*, that is fundamental to predict the germination/emergence in field conditions.

In order to account for the very nature of the water potential variable, we could code a time-to-event model where the three parameters are continuous functions of Ψ:

$$P(t, \Psi) = \Phi\left(b(\Psi), d(\Psi), e(\Psi)\right)$$

We followed such an approach in a relatively recent publication (Onofri et al., 2018) and we also spoke about this in a recent post. In detail, we considered a log-logistic cumulative distribution function:

$$P(t) = \frac{d}{1 + \exp\left\{b\left[\log(t) - \log(e)\right]\right\}}$$

where e is the median germination time, b is the slope at the inflection point and d is the maximum germinated proportion. Considering that the germination rate is the inverse of germination time, we replaced $e = 1/GR_{50}$ and wrote the three parameters as functions of Ψ:

$$P(t, \Psi) = \frac{d(\Psi)}{1 + \exp\left\{b(\Psi)\left[\log(t) - \log(1/\left[GR_{50}(\Psi)\right])\right]\right\}}$$

where:

$$\begin{aligned}
&GR_{50}(\Psi) = \max\left(\tfrac{\Psi - \Psi_b}{\theta_H}; 0\right)\\
&d(\Psi) = \max\left\{G\ \left[1 - \exp\left(\tfrac{\Psi - \Psi_b}{\sigma_{\Psi_b}}\right)\right]; 0\right\}\\
&b(\Psi) = b
\end{aligned}$$

The parameters are:

1. Ψ_b, that is the median base water potential in the seed lot (in *MPa*),
2. θ_H, that is the hydro-time constant (in *MPa day* or *MPa hour*)
3. σ_{Ψ_b}, that represents the variability of Ψ_b within the population,
4. G, that is the germinable fraction, accounting for the fact that d may not reach 1, regardless of time and water potential.
5. b (slope parameter) that is assumed to be constant and independent on Ψ.

In the end, our hydro-time model is composed by four sub-models:

1. a cumulative probability function (log-logistic, in our example), based on the three parameters d, b and $e = 1/GR50$;
2. a sub-model expressing d as a function of Ψ;
3. a sub-model expressing $GR50$ as a function of Ψ;
4. a sub-model expressing b as a function of Ψ, although, this was indeed a simple identity model $b(\Psi) = b$.

This hydro-time-to-event model was implemented in R as the `HTE1()` function, and it is available within the `drcSeedGerm` package, together with the appropriate self-starting routine. It can be fitted by using the `drmte()` function in the `drcte` package and the `coef()` function can be used to retrieve the parameter estimates.

```
modHTE <- drmte(nSeeds ~ timeBef + timeAf + Psi,
                data = rape, fct = HTE1())
coef(modHTE)
##         G:(Intercept)      Psib:(Intercept) sigmaPsib:(Intercept)
##             0.9577918           -1.0397239             0.1108891
##    thetaH:(Intercept)         b:(Intercept)
##             0.9061385             4.0273963
```

As we said before, we are also interested in standard errors for model parameters; we will address this issue in another post. It is important to note that this model gives us the ability of predicting germination at any water potential levels and it is not restrained to the values that we included in the experimental design. Furthermore, we have reliable estimates of Ψ_b and θ_H, which we can use for prediction purposes in field conditions.

6.3 Another modelling approach

Another type of hydro-time model was proposed by Bradford (2002) and later extended by Mesgaran et al., (2013). These authors, instead of modifying a traditional log-logistic distribution to include the environmental covariate, wrote a totally new cumulative distribution function, based on theoretical underpinnings relating to the distribution of base water potential within a seed population. Their model is:

$$P(t, \Psi) = \Phi\left\{\frac{\Psi - (\theta_H/t) - \Psi_b}{\sigma_{\Psi_b}}\right\}$$

where Φ is a gaussian cumulative distribution function for base water potential. More information on how this model can be obtained from the original papers; it is, however, important to highlight that it is assumed that base water potential changes from seed to seeds within the population, according to a gaussian

distribution function. The cumulative distribution function of event times is indirectly modelled, but it is not, in itself, gaussian (you see that t is at the denominator).

Mesgaran et al (2013) suggested that Φ may not be gaussian and proposed several alternatives, so that, in all, we have six possible hydro-time-to-event models, which we have implemented within the `drcSeedGerm` package:

1. gaussian (function `HTnorm()`)
2. logistic (function `HTL()`)
3. Gumbel (function `HTG()`)
4. log-logistic (function `HTLL()`)
5. Weibull (Type I) (function `HTW1()`)
6. Weibull (Type II) (function `HTW2()`)

These equations are given at the end of this post. The code to fit those models is given below:

```
mod1 <- drmte(nSeeds ~ timeBef + timeAf + Psi,
              data = rape, fct = HTnorm())
mod2 <- drmte(nSeeds ~ timeBef + timeAf + Psi,
              data = rape, fct = HTL())
mod3 <- drmte(nSeeds ~ timeBef + timeAf + Psi,
              data = rape, fct = HTG())
mod4 <- drmte(nSeeds ~ timeBef + timeAf + Psi,
            data = rape, fct = HTLL())
mod5 <- drmte(nSeeds ~ timeBef + timeAf + Psi,
            data = rape, fct = HTW1())
mod6 <- drmte(nSeeds ~ timeBef + timeAf + Psi,
            data = rape, fct = HTW2())
```

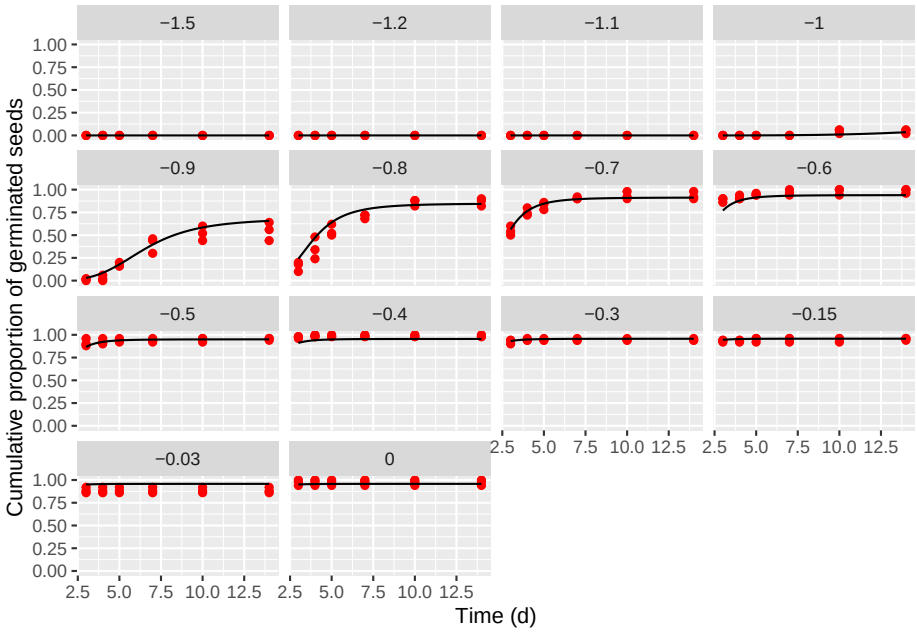
What is the best model for this dataset? Let's use the Akaike's Information Criterion (AIC: the lowest, the best) to decide; we see that `modHTE` was the best fitting one, followed by `mod4`.

```
AIC(mod1, mod2, mod3, mod4, mod5, mod6, modHTE)
##          df      AIC
## mod1    291 3516.914
## mod2    291 3300.824
## mod3    291 3097.775
## mod4    290 2886.608
## mod5    290 2889.306
## mod6    290 3009.023
## modHTE 289 2832.481
```

It is important not to neglect a graphical inspection of model fit. The `plot()` method does not work with time-to-event curves with additional covariates (apart from time). However, we can retrieve the fitted data by using the `plotData()` function and use those predictions within the `ggplot()` function.

The box below shows the appropriate coding.

```
library(ggplot2)
tab <- plotData(modHTE)

ggplot() +
  geom_point(data = rape, mapping = aes(x = timeAf, y = propCum),
             col = "red") +
  geom_line(data = tab$plotFits, mapping = aes(x = timeAf, y = CDF)) +
  facet_wrap(~ Psi) +
  scale_x_continuous(name = "Time (d)") +
  scale_y_continuous(name = "Cumulative proportion of germinated seeds")
## Warning: Removed 42 rows containing missing values (`geom_point()`).
```



6.4 Further detail

Let us conclude this section by giving some detail on all other models in Mesgaran et al (2913; slightly reparameterised). In same cases Ψ_b has been replaced by μ, that is the location parameter of the cumulative distribution function of base water potential, but it is not the median value. On the other hand, δ is the shifting parameter for all logarithm based distributions; indeed, logarithm based distribution are only defined for strictly positive variables, while we know that water potential usually assumes negative values. The shifting parameters is used to shift the cumulative distribution function to the right, so that negative

values are allowed.

6.4.1 HTL()

$$G(t,\Psi) = \frac{1}{1 + exp\left[-\frac{\Psi-(\theta_H/t)-\Psi_b}{\sigma}\right]}$$

6.4.2 HTG()

$$G(t,\Psi) = \exp\left\{-\exp\left[-\left(\frac{\Psi-(\theta_H/t)-\mu}{\sigma}\right)\right]\right\}$$

6.4.3 HTLL()

$$G(t,\Psi) = \frac{1}{1+\exp\left\{\frac{\log\left[\Psi-\left(\frac{\theta_H}{t}\right)+\delta\right]-\log(\Psi_b+\delta)}{\sigma}\right\}}$$

6.4.4 HTW1()

$$G(t,\Psi) = exp\left\{-\exp\left[-\frac{\log\left[\Psi-\left(\frac{\theta_H}{t}\right)+\delta\right]-\log(\Psi_b+\delta)}{\sigma}\right]\right\}$$

6.4.5 HTW2()

$$G(t,\Psi) = 1 - exp\left\{-\exp\left[\frac{\log\left[\Psi-\left(\frac{\theta_H}{t}\right)+\delta\right]-\log(\Psi_b+\delta)}{\sigma}\right]\right\}$$

Chapter 7

Exploring the results of a time-to-event fit: model parameters

Once we have fit a time-to-event model, we are usually interested in exploring the results, to get all possible information from the fitted model. If we have fitted a parametric model, the value of the estimated parameters is usually of extreme interest, as it gives information on the main traits of germination/emergence (e.g., capability, speed and uniformity).

For example, let's consider the hydro-time model we have fitted in our previous post at this link:

$$P(t, \Psi) = \Phi \left\{ \frac{\Psi - (\theta_H / t) - \Psi_b}{\sigma_{\Psi_b}} \right\}$$

where P is the cumulative proportion of germinated seeds at time t and water potential Ψ, Φ is a gaussian cumulative distribution function for base water potential, Ψ_b is the median base water potential in the seed lot (in MPa), θ_H is the hydro-time constant (in MPa day or MPa hour) and σ_{Ψ_b} represents the variability of Ψ_b within the population. Clearly, these parameters have a clear biological meaning and getting to know about their value represents the reason why we have fitted such a model. The box below shows the code we used in our previous post:

```
library(drcSeedGerm)
library(drcte)
data(rape)
modHTE <- drmte(nSeeds ~ timeBef + timeAf + Psi,
```

```
                data = rape, fct = HTnorm())
coef(modHTE)
##    ThetaH:(Intercept)    Psib50:(Intercept) sigmaPsib:(Intercept)
##             0.7510691            -0.9069810             0.2369954
```

Indeed, we got parameter estimates, but we are not happy with this. We also need standard errors, to present along with estimates in our papers. The easiest way to obtain parameters and their standard errors altogether is to use the `summary()` method for 'drcte' objects:

```
summary(modHTE)
##
## Model fitted: Hydrotime model with normal distribution of Psib (Bradford et al., 20
##
## Robust estimation: no
##
## Parameter estimates:
##
##                         Estimate Std. Error t-value   p-value
## ThetaH:(Intercept)     0.7510691  0.0394604  19.034 < 2.2e-16 ***
## Psib50:(Intercept)    -0.9069810  0.0118081 -76.810 < 2.2e-16 ***
## sigmaPsib:(Intercept)  0.2369954  0.0071406  33.190 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Unfortunately, the above standard errors are not correct: indeed, they are obtained assuming that the observational units (i.e., the seeds) are independent, while they are clustered within randomisation units (Petri dishes, in this case). Consequently, seeds in the same Petri dish are more similar than seeds in different Petri dishes (there is intra-class correlation, we say). How can we obtain standard errors that account for such lack of independence?

If we look at the literature about survival analysis (that is where we borrowed our methods from), we can see that cluster robust sandwich estimators of standard errors have proven useful and reliable (Yu and Peng, 2008). Therefore, we have implemented them in 'drcte'; the 'units' argument in the `summary()` method can be used to provide a variable for the Petri dishes and calculate cluster-robust SEs, by way of the facilities provided in the 'sandwich' package (Zeileis et al. 2020).

```
summary(modHTE, robust = T, units = Dish)
##
## Model fitted: Hydrotime model with normal distribution of Psib (Bradford et al., 20
##
## Robust estimation: Cluster robust sandwich SEs
##
## Parameter estimates:
```

```
##
##                         Estimate Std. Error  t value  Pr(>|t|)
## ThetaH:(Intercept)      0.751069   0.131968   5.6913 3.075e-08 ***
## Psib50:(Intercept)     -0.906981   0.039530 -22.9444 < 2.2e-16 ***
## sigmaPsib:(Intercept)  0.236995   0.031309   7.5696 4.974e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see that the difference between 'naive' and cluster robust SEs is remarkable.

There might be other methods to obtain cluster robust standard errors (e.g., jackknife and bootstrap methods) and we are looking for ways to implement them in a reliable way. So far, we recommend that you make sure that your standard errors for model parameters do not neglect the clustering of seeds within randomisation units (petri dishes, pots, boxes or plots).

Chapter 8

Predictions from time-to-event models

8.1 Parametric time-to-event model

Another key aspect is to use a fitted model to make predictions: what fraction of germinated/emerged seeds will we find in, e.g., one/two weeks? And in one month? For example, let's consider the hydro-time model we have fitted in some previous posts (the first one is at this link):

$$P(t, \Psi) = \Phi\left\{\frac{\Psi - (\theta_H/t) - \Psi_b}{\sigma_{\Psi_b}}\right\}$$

In the above model, P is the cumulative proportion of germinated seeds at time t and water potential Ψ, Φ is a gaussian cumulative distribution function for base water potential, Ψ_b is the median base water potential in the seed lot (in MPa), θ_H is the hydro-time constant (in MPa day or MPa hour) and σ_{Ψ_b} represents the variability of Ψ_b within the population.

The code below can be used to fit the above model to the 'rape' dataset in the 'drcSeedGerm' package and retreive the estimated parameters, with robust standard errors:

```
library(drcSeedGerm)
library(drcte)
data(rape)
modHTE <- drmte(nSeeds ~ timeBef + timeAf + Psi,
                data = rape, fct = HTnorm())
summary(modHTE, units = Dish)
##
```

```
## Model fitted: Hydrotime model with normal distribution of Psib (Bradford et al., 20
##
## Robust estimation: Cluster robust sandwich SEs
##
## Parameter estimates:
##
##                         Estimate Std. Error  t value  Pr(>|t|)
## ThetaH:(Intercept)      0.751069   0.131968   5.6913 3.075e-08 ***
## Psib50:(Intercept)     -0.906981   0.039530 -22.9444 < 2.2e-16 ***
## sigmaPsib:(Intercept)  0.236995   0.031309   7.5696 4.974e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Now, we may wonder: if we have a seed lot with the above characteristics ($\theta = 0.751$ MPa d, $\Psi_{b_{50}} = -0.907$ MPa and $\sigma_{\Psi_b} = 0.237$), what will the proportion of germinated seeds be, e.g., at 1, 3, 5, 7 days after water imbibition, when the base water potential in the substrate is, e.g., 0, -0.25 and -0.5 MPa? To predict this from the model object we can build a data frame with the values of predictors (see below the use of the `expand.grid()` function) and use it as the 'newdata' argument to the `predict()` method.

```
newd <- expand.grid(time = c(1, 3, 5, 7),
                    psi = c(0, -0.25, -0.5))
predict(modHTE, newdata = newd)
##    time   psi Prediction
## 1     1  0.00 0.74468884
## 2     3  0.00 0.99720253
## 3     5  0.00 0.99929640
## 4     7  0.00 0.99962993
## 5     1 -0.25 0.34568239
## 6     3 -0.25 0.95689600
## 7     5 -0.25 0.98375378
## 8     7 -0.25 0.98981312
## 9     1 -0.50 0.07326801
## 10    3 -0.50 0.74565419
## 11    5 -0.50 0.86069050
## 12    7 -0.50 0.89697826
```

With time-to-event models, the 'newdata' argument takes a data frame, where the first column is always time and the succeeding columns, wherever needed, represent the environmental covariates, in the same order as they appear in the model definition. If several models have been simultaneously fitted by using the 'curveid' argument (not in this case, though), predictions are made for all models, always using the same 'newdata'.

We can also obtain standard errors and confidence intervals for the predictions, by adding the `se.fit = TRUE` and `interval = TRUE` arguments. We also rec-

ommend to add the `robust = T` argument, so that we obtain robust standard errors, accounting for the clustering of seeds within Petri dishes (lack of independence). With parametric time-to-event models, robust standard errors are obtained by using a cluster-robust sandwich variance-covariance matrix (Zeileis et al. 2020); in this case, a clustering variable needs to be provided with the `units` argument.

```
# Naive standard errors and confidence intervals
predict(modHTE, newdata = newd, se.fit = T, interval = T)
##    time   psi Prediction           SE      Lower     Upper
## 1     1  0.00 0.74468884 0.0452433821 0.65601344 0.8333642
## 2     3  0.00 0.99720253 0.0008053309 0.99562411 0.9987809
## 3     5  0.00 0.99929640 0.0002384085 0.99882913 0.9997637
## 4     7  0.00 0.99962993 0.0001358778 0.99936362 0.9998963
## 5     1 -0.25 0.34568239 0.0488012921 0.25003362 0.4413312
## 6     3 -0.25 0.95689600 0.0060636345 0.94501149 0.9687805
## 7     5 -0.25 0.98375378 0.0027953655 0.97827496 0.9892326
## 8     7 -0.25 0.98981312 0.0019555787 0.98598025 0.9936460
## 9     1 -0.50 0.07326801 0.0182524957 0.03749378 0.1090422
## 10    3 -0.50 0.74565419 0.0143630866 0.71750306 0.7738053
## 11    5 -0.50 0.86069050 0.0098002073 0.84148245 0.8798986
## 12    7 -0.50 0.89697826 0.0084761895 0.88036524 0.9135913
```

```
# Cluster robust standard errors and confidence intervals
predict(modHTE, newdata = newd, se.fit = T, interval = T,
        robust = T, units = Dish)
##    time   psi Prediction           SE      Lower     Upper
## 1     1  0.00 0.74468884 0.1450176755 0.46045941 1.0289183
## 2     3  0.00 0.99720253 0.0035012373 0.99034023 1.0040648
## 3     5  0.00 0.99929640 0.0011070141 0.99712670 1.0014661
## 4     7  0.00 0.99962993 0.0006427237 0.99837022 1.0008897
## 5     1 -0.25 0.34568239 0.1576339911 0.03672545 0.6546393
## 6     3 -0.25 0.95689600 0.0251911861 0.90752218 1.0062698
## 7     5 -0.25 0.98375378 0.0129567170 0.95835908 1.0091485
## 8     7 -0.25 0.98981312 0.0093141326 0.97155775 1.0080685
## 9     1 -0.50 0.07326801 0.0622953823 0.00000000 0.1953647
## 10    3 -0.50 0.74565419 0.0498559815 0.64793826 0.8433701
## 11    5 -0.50 0.86069050 0.0424570614 0.77747619 0.9439048
## 12    7 -0.50 0.89697826 0.0388178876 0.82089660 0.9730599
```

We are currently studying a way to avoid that confidence intervals return unrealistic predictions (see above some values that are higher than 1). We may note that cluster robust standard errors are higher than naive standard errors: the seed in the same Petri dish are correlated and, thus, they do not contribute full information.

8.2 Non-parametric time-to-event models

The `predict()` method can also be used to make predictions from NPMLE and KDE fits. In this case, no environmental covariates are admissible and, therefore, we can provide 'newdata' as a vector of times to make predictions. In the code below we fit the NPMLE of a time-to-event model to four species of the genus *Verbascum*, for which the data are available as the 'verbascum' data frame. We also make predictions relating to the proportion of germinated seeds at 1, 3, 5, and 7 days from water imbibition.

```
data(verbascum)
mod <- drmte(nSeeds ~ timeBef + timeAf, fct = NPMLE(),
             curveid = Species, data = verbascum)

# Define the values for predictions
newd <- c(1, 3, 5, 7)
predict(mod, newdata = newd, se.fit = T, interval = T,
        robust = T, units = Dish)
##      Species newdata Prediction         SE    Lower Upper
## 1   arcturus       1       0.00 0.00000000 0.000000  0.00
## 2   arcturus       3       0.00 0.00000000 0.000000  0.00
## 3   arcturus       5       0.00 0.00000000 0.000000  0.00
## 4   arcturus       7       0.00 0.00000000 0.000000  0.00
## 5  blattaria       1       0.00 0.00000000 0.000000  0.00
## 6  blattaria       3       0.09 0.05594901 0.010000  0.24
## 7  blattaria       5       0.73 0.06893809 0.600000  0.86
## 8  blattaria       7       0.80 0.06154616 0.690000  0.91
## 9   creticum       1       0.00 0.00000000 0.000000  0.00
## 10  creticum       3       0.33 0.08076917 0.167625  0.48
## 11  creticum       5       0.97 0.02382806 0.910000  1.00
## 12  creticum       7       0.97 0.02382806 0.910000  1.00
```

Standard errors are estimated by using a resampling (bootstrap) approach, that is performed at the group level, whenever a grouping variable is provided, by way of the 'units' argument (Davison and Hinkley, 1997).

For KDE models, we can make predictions in the very same way, although we are still unsure about the most reliable way to obtain standard errors. For this reason, the use of the 'predict' method with this type of non-parametric models does not yet provide standard errors and confidence intervals.

8.3 Predictions from a time-to-event model from literature

In some cases, we do not have a fitted model, but we have some literature information. For example, we have seen a manuscript where the authors say that, for a certain species, emergences appeared to follow a log-logistic time-course with the following parameters: 'b' (the slope at inflection point) equal to -1, 'd' (maximum germinated proportion) equal to 0.83 and 'e' (median germination time for the germinated fraction) equal to 12.3. Considering that a log-logistic time-to-event model is represented as LL.3(), we can make predictions by using the following code:

```
predict(LL.3(), coefs = c(-1, 0.83, 12.3),
        newdata = c(1, 3, 5, 7, 10))
##   newdata prediction
## 1       1 0.06240602
## 2       3 0.16274510
## 3       5 0.23988439
## 4       7 0.30103627
## 5      10 0.37219731
```

Chapter 9

Quantiles from time-to-event models

A time-to-event model is, indeed, a cumulative probability function for germination time and, therefore, we might be interested in finding the quantiles. But, what are the 'quantiles'? It is a set of 'cut-points' that divide the distribution of event-times into a set of q intervals with equal probability. For example the 100-quantiles, also named as the percentiles, divide the distribution of event-times into $q = 100$ groups. Some of these cut-off points may be particularly relevant: for example the 50-th percentile corresponds to the time required to reach 50% germination (T50) and it is regarded as a good measure of germination velocity. Other common percentiles are the T10, or the T30, which are used to express the germination velocity for the quickest seeds in the lot.

Extracting some relevant percentiles from the time-to-event curve is regarded as an important task, to sintetically describe the germination/emergence velocity of seed populations. To this aim, we have included the `quantile()` method in the `drcte` package, that addresses most of the peculiarities of seed germination/emergence data. In this post, we will show the usage of this function.

9.1 Peculiarities of seed science data

I know that you are looking forward to getting the quantiles for your time-to-event curve. Please, hang on for a while... we need to become aware of a couple of issues, that are specific to germination/emergence data and are not covered in literature for other types of time-to-event data (e.g., survival data).

9.1.1 Quantiles and 'restricted' quantiles

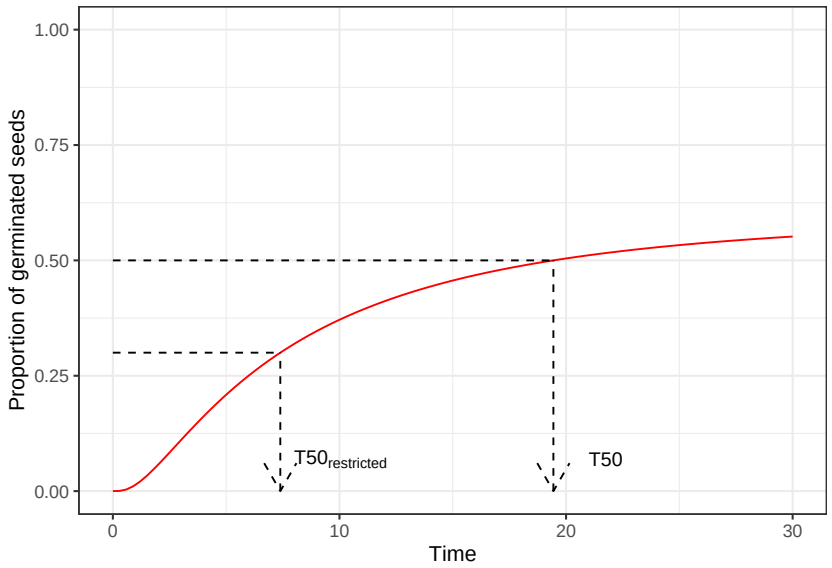
Due to the presence of the ungerminated/unemerged fraction, quantiles suffer from the intrinsic ambiguity that we could calculate them either for the whole sample, or for the germinated fraction. For example, let's think that we have a seed lot where the maximum percentage of germination is 60% and thus 40% of seeds are dormant. How do we define the 50^{th} percentile?

In general, we should consider the whole population, including the ungerminated fraction, where the event is not observed; accordingly, the, e.g., 50^{th} percentile (T50) should be defined as the time to 50% germination. Obviously, with such a definition the, e.g., T50 cannot be estimated when the maximum germinated fraction is lower than 50%.

On the other hand, for certain applications, it might be ok to remove the ungerminated fraction prior to estimating the quantiles; in this case, for our example where the maximum germinated fracion is 60%, the T50 would be defined as the time to 30% germination, that is 50% of the maximum germinated fraction.

Due to such an ambiguity, we should talk about quantiles and 'restricted' quantiles. The graph below should help clarify such a difference.

As a general suggestion, we should never use restricted quantiles for seed germination/emergence studies, especially when the purpose is to make comparisons across treatment groups (Bradford, 2002; Keshtkar et al. 2021).



9.1.2 Germination/emergence rates

The quantiles of germination times (e.g., T10, T30 or T50) are very common measures of germination velocity, although they may be rather counterintuitive, because a high germination time implies low velocity. Another common measure of velocity is the Germination Rate, that is the inverse of germination time (e.g., GR10 = 1/T10).

The quantiles of germination rates (e.g., GR10, GR30, GR50...) represents the daily progress to germination for a given subpopulation and they are used as the basis for hydro-thermal-time modelling. Therefore, their determination for a seed lot is also very relevant.

9.2 Getting the quantiles with 'drcte'

9.2.1 Parametric models

In a previous post we have used the code below to fit a log-logistic time-to-event model to the germination data for three species of the Verbascum genus:

```
library(drcte)
library(drcSeedGerm)
data(verbascum)
mod1 <- drmte(nSeeds ~ timeBef + timeAf, fct = LL.3(),
              curveid = Species, data = verbascum)
summary(mod1, units = Dish)
##
## Model fitted: Log-logistic (ED50 as parameter) with lower limit at 0
##
## Robust estimation: Cluster robust sandwich SEs
##
## Parameter estimates:
##
##                Estimate Std. Error  t value  Pr(>|t|)
## b:arcturus   -9.930005   1.111135  -8.9368 4.286e-16 ***
## b:blattaria  -7.198025   0.575290 -12.5120 < 2.2e-16 ***
## b:creticum  -11.033749   0.943374 -11.6961 < 2.2e-16 ***
## d:arcturus    0.356648   0.047915   7.4433 3.715e-12 ***
## d:blattaria   0.840064   0.025593  32.8242 < 2.2e-16 ***
## d:creticum    0.969990   0.017290  56.1015 < 2.2e-16 ***
## e:arcturus   12.059189   0.486010  24.8126 < 2.2e-16 ***
## e:blattaria   4.031763   0.199590  20.2002 < 2.2e-16 ***
## e:creticum    3.200655   0.103161  31.0259 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It may be useful to rank the species in terms of their germination velocity and, to that purpose, we could estimate the times to 30% and 50% germination (T30 and T50), that are the 30^{th} and 50^{th} percentiles of the time-to-event distribution. We can use the `quantile()` method, where the probability levels are passed in as the vector 'probs':

```
quantile(mod1, probs = c(0.3, 0.5))
##
## Estimated quantiles
##
##                Estimate       SE
## arcturus:30%    14.2634 0.992685
## arcturus:50%        NaN      NaN
## blattaria:30%    3.7156 0.106048
## blattaria:50%    4.2536 0.118515
## creticum:30%     2.9759 0.058279
## creticum:50%     3.2187 0.059480
```

We may note that the T50 is not estimable with *Verbascum arcturus*, as the maximum germinated proportion (d parameter for the time-to-event model above) is 0.36. Standard errors are obtained by using the delta method and they are invalid whenever the experimental units (seeds) are clustered within containers, such as the Petri dishes.

For all these cases, we should prefer cluster-robust standard errors (Zeileis et al. 2020), which can be obtained by setting the extra argument 'robust = TRUE' and providing a clustering variable as the `units` argument. By setting 'interval = TRUE' we can also obtain confidence intervals for the desired probability level (0.95, by default).

```
quantile(mod1, probs = c(0.3, 0.5), robust = T,
         units = Dish,
         interval = T)
##
## Estimated quantiles
##
##                Estimate       SE   Lower   Upper
## arcturus:30%    14.2634 0.755294 12.7830 15.7437
## arcturus:50%        NaN      NaN     NaN     NaN
## blattaria:30%    3.7156 0.187503  3.3481  4.0831
## blattaria:50%    4.2536 0.209853  3.8423  4.6649
## creticum:30%     2.9759 0.085151  2.8090  3.1428
## creticum:50%     3.2187 0.104743  3.0134  3.4240
```

We may note that cluster robust standard errors are higher than naive standard errors: the seed in the same Petri dish are correlated and, thus, they do not contribute full information.

If we are interested in the germination rates G30 and G50, we can set the

argument 'rate = T', as shown in the box below.

```
quantile(mod1, probs = c(0.3, 0.5), robust = T,
         units = Dish,
         interval = T, rate = T)
##
## Estimated quantiles
##
##                Estimate        SE    Lower    Upper
## arcturus:30%    0.07011 0.0037126 0.062833 0.077386
## arcturus:50%    0.00000 0.0000000 0.000000 0.000000
## blattaria:30%   0.26914 0.0135819 0.242519 0.295759
## blattaria:50%   0.23510 0.0115987 0.212364 0.257830
## creticum:30%    0.33604 0.0096153 0.317191 0.354882
## creticum:50%    0.31069 0.0101106 0.290872 0.330505
```

9.2.2 Parametric models with environmental covariates

If we have fitted a hydro-thermal time model or other models with an environmental covariate, we can also use the `quantile()` method, and pass a value for that covariate, as shown in the code below.

```
# Parametric model with environmental covariate
data(rape)
modTE <- drmte(nSeeds ~ timeBef + timeAf + Psi,
           data = rape, fct = HTLL())
quantile(modTE, Psi = 0,
         probs = c(0.05, 0.10, 0.15, 0.21),
         restricted = F, rate = T, robust = T,
         interval = T)
##
## Estimated quantiles
##
##       Estimate      SE  Lower  Upper
## 1:5%    1.5861 0.22621 1.1428 2.0295
## 1:10%   1.5562 0.21060 1.1435 1.9690
## 1:15%   1.5331 0.20063 1.1399 1.9263
## 1:21%   1.5090 0.19183 1.1330 1.8850
```

The environmental covariate only accepts a single value; in order to vectorise, we need to use the `lapply()` function, as shown below.

```
# This is to vectorise on Psi
psiList <- seq(-1, 0, 0.25)
names(psiList) <- as.character(psiList)
lapply(psiList,
       function(x) quantile(modTE, Psi = x,
```

```
                          probs = c(0.05, 0.10, 0.15, 0.21),
                          restricted = F, rate = T,
                          interval = "delta",
                          units = rape$Dish,
                          display = F))
## $`-1`
##          Estimate         SE         Lower      Upper
## 1:5%  0.10860038 0.05566722 -0.0005053734 0.21770613
## 1:10% 0.07869462 0.03741665  0.0053593346 0.15202991
## 1:15% 0.05555279 0.02674483  0.0031338819 0.10797171
## 1:21% 0.03145638 0.01993608 -0.0076176237 0.07053039
##
## $`-0.75`
##        Estimate         SE     Lower     Upper
## 1:5%  0.4779833 0.08302599 0.3152553 0.6407112
## 1:10% 0.4480775 0.06331212 0.3239880 0.5721670
## 1:15% 0.4249357 0.05037871 0.3261952 0.5236762
## 1:21% 0.4008393 0.03876416 0.3248629 0.4768157
##
## $`-0.5`
##        Estimate         SE     Lower     Upper
## 1:5%  0.8473662 0.11816778 0.6157616 1.0789708
## 1:10% 0.8174604 0.09946283 0.6225169 1.0124040
## 1:15% 0.7943186 0.08738918 0.6230390 0.9655982
## 1:21% 0.7702222 0.07670819 0.6198769 0.9205675
##
## $`-0.25`
##       Estimate        SE     Lower    Upper
## 1:5%  1.216749 0.1559175 0.9111565 1.522342
## 1:10% 1.186843 0.1380349 0.9162999 1.457387
## 1:15% 1.163702 0.1265389 0.9156899 1.411713
## 1:21% 1.139605 0.1163701 0.9115239 1.367686
##
## $`0`
##       Estimate        SE    Lower    Upper
## 1:5%  1.586132 0.1947646 1.204400 1.967864
## 1:10% 1.556226 0.1774564 1.208418 1.904034
## 1:15% 1.533084 0.1663240 1.207095 1.859073
## 1:21% 1.508988 0.1564488 1.202354 1.815622
```

9.2.3 Non-parametric (NPMLE) models

The `quantile()` method can also be used to make predictions from NPMLE fits. This method works by assuming that the events are evenly scattered within

each inspection interval ('interpolation method'). Inferences need to be explicitly requested by using setting 'interval = T'; in this case, standard errors are estimated by using a resampling approach, that is performed at the group level, whenever a grouping variable is provided, by way of the 'units' argument (Davison and Hinkley, 1997).

```
mod2 <- drmte(nSeeds ~ timeBef + timeAf, fct = NPMLE(),
              curveid = Species, data = verbascum)
quantile(mod2, probs = c(0.3, 0.5), robust = T, units = Dish,
         interval = T, rate = T)
##
##
##
##
## Estimated quantiles
## (cluster robust bootstrap-based inference)
##
##                   n       Mean   Median        SE   Lower    Upper
## arcturus.30%  999 0.04215854 0.067385 0.0367409 0.00000 0.085837
## arcturus.50%  999 0.00047593 0.000000 0.0056696 0.00000 0.000000
## blattaria.30% 999 0.26933051 0.268456 0.0180207 0.23961 0.308415
## blattaria.50% 999 0.23279718 0.230530 0.0131552 0.21341 0.263740
## creticum.30%  999 0.34397841 0.343750 0.0195091 0.31087 0.379032
## creticum.50%  999 0.30314443 0.302198 0.0117692 0.28139 0.326935
```

For KDE models, quantiles are calculated from the time-to-event curve by using a bisection method. However, we are still unsure about the most reliable way to obtain standard errors and, for this reason, inferences are not provided with this type of non-parametric models.

9.3 Quantiles and Effective Doses (ED)

Quantiles for time-to-event data resamble Effective Doses (ED) for dose-response data, although we discourage the use of this latter term, as the time-to-event curve is a cumulative probability function based on time, that is not a 'dose' in strict terms. However, the concept is similar: we need to find the stimulus (time) that permits to obtain a certain response (germination/emergence). Considering such similarity, we decided to define the `ED()` method for 'drcte' objects, that is compatible with the `ED()` method for 'drc' objects. However, for seed germination/emergence data, we strongly favor the use of the `quantile()` method.

Chapter 10

Fitting germination models in two steps

In the previous sections I have shown how to fit time-to-event curves to seed germination data. I have also shown that these curves can be modified to include the effect of external factors (e.g., the species) or covariates (e.g., temperature or humidity content in the substrate), which brought us to the definition of hydro-time, thernal-time or hydro-thermal-time models. So far, in this tutorial, we have fitted these modified time-to-event curves in 'one-step', i.e., we started from the germination data, fitted the selected model and retrieved the estimates of model parameters. In such a one-step approach, we showed that one problem was that, most of the times, seeds were not independent, but, they were clustered within Petri dishes, trays, boxes or other containers. Such a clustering, if not appropriately accounted for, brakes the independence assumption that is made by most linear and nonlinear models, invalidating the inferences based on those models.

So far, we have accounted for the above mentioned clustering by using cluster robust standard errors, which are easily calculated within the 'drcte' package. However, we should also say that one-step model fitting may often be complex, particularly when we have to deal with a high number of seeds, complex models and multi-tier experimental designs. In order to avoid such complexities, we might like to follow a different approach, that is accomplished in two-steps. Let's consider an example where we have performed germination assays where seeds have been submitted to 10 different treatment levels: instead of building a time-to-event model that contains the treatment as an external factor, we could:

1. independently fit a different time-to-event curve to the germination data for each treatment level, so that we have ten independent time-to-event curves;
2. derive from each curve a summary statistic of interest, such as the germi-

nation rate for the 50-th percentile (GR50; see earlier in this tutorial);
3. fit, e.g., a thermal-time model to those derived statistics (second step of data analyses).

Fitting models in two-steps has always been a common practice in agriculture/biology (see, e.g., multienvironment genotype experiments): it is usually simpler, quicker and requires lower computing power than one-step fitting. The drawback is that some infomation may be lost between the two steps, and, for this reason, one-step and two-steps model fitting do not necessarily lead to the same results. But, we'll make this point later.

Let's show some examples of this two-steps approach.

10.1 Fitting linear models in two steps

(yet to be done)

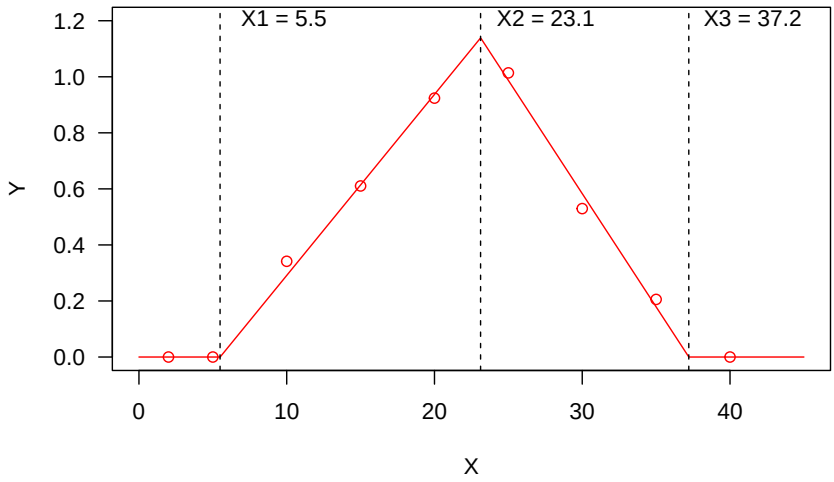
10.2 Fitting non-linear threshold models in two steps

10.2.1 Definition of threshold models

Threshold models are used to describe a relationship where the response variable changes abruptly, following a small change in the predictor. A typical threshold model looks like that in the Figure below, where we see three threshold levels:

1. $X1 = 5.5$: at this threshold, the response changes abruptly from 'flat' to linearly increasing;
2. $X2 = 23.1$: at this threshold, the response changes abruptly from linearly increasing to linearly decreasing;
3. $X3 = 37.2$: at this threshold, the response changes abruptly from linearly decreasing to 'flat'.

You may recognise a 'broken-stick' pattern, although threshold models can also be curvilinear, as we will see later.



I have already considered threshold models in a previous post (see here) and I have already mentioned that thermal-time, hydro-time and hydro-thermal-time models for seed germination can also be cast as threshold models; if we consider, e.g., the Germination Rate (GR) as the response variable and the environmental temperature as the predictor, the relationship could be very close to that represented in Figure 1 and the three thresholds would, respectively, be the *base temperature* (T_b_), the *optimal temperature* (T_o_) and the *ceiling temperature* (T_c_). On the other hand, if we consider the effect of soil humidity on GR, we should expect a response pattern with only one threshold, i.e. the *base water potential* level (e.g. the first half of the figure above, up to the maximum response level).

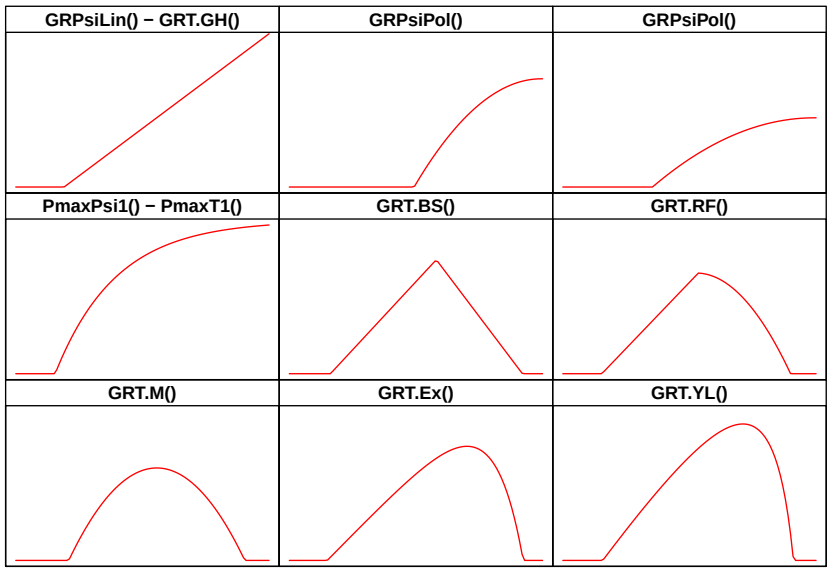
10.2.2 A (possibly incomplete) list of threshold models

I have made a review of literature, searching for all threshold models that have been used so far in seed germination studies. For all those models, I have built the related R functions, together with self-starting routines, which can be used for nonlinear regression fitting with the `drm()` function in the `drc` package (Ritz et al., 2019). The availability of self-starting routines will free you from the hassle of having to provide initial guesses for model parameters. All these R functions are available within the `drcSeedGerm` package (Onofri et al., 2018) and their names, with links to the relevant parts of the appendix to this post are:

1. GRPsi.Lin() - GRT.GH()
2. GRPsi.Pol() - GRPsi.Pol2()
3. PmaxPsi1() - PmaxT1()
4. GRT.BS()
5. GRT.RF()
6. GRT.M()

7. GRT.Ex()
8. GRT.YL()

It may be helpful to look at the shapes of the above models in the Figure below, while the equations are motivated in the appendix, at the end of this post.



Now, let's look at a few examples of two-steps fitting. But, before working through this, you will need to install and load the `drcSeedGerm` package, by using the code below.

```
# installing drcSeedGerm package, if not yet available
# library(devtools)
# install_github("onofriandreapg/drcSeedGerm")

# loading package
library(drcSeedGerm)
library(tidyverse)
library(lmtest)
library(sandwich)
```

10.2.3 Description of models

10.2.3.1 GRPsiLin() - GRT.GH()

The equation behind `GRPsiLin()` has been used to describe the effect of environmental humidity (Ψ, in MPa) on germination rate (Bradford, 2002):

$$GR = \frac{\max\left[\Psi, \Psi_b\right] - \Psi_b}{\theta_H}$$

The parameter Ψ_b is the *base water potential* (in MPa), representing the minimum level of humidity in the substrate to trigger the germination process. The other parameter θ_H (in MPa day or MPa hour) is the hydro-time constant.

A totally similar equation (with different parameter names) has been used by Garcia-Huidobro et al (1982), to describe the effect of sub-optimal temperatures (T in °C) on the germination rate:

$$GR = \frac{\max\left[T, T_b\right] - T_b}{\theta_T}$$

where T_b is the base temperature and θ_T is the thermal time (in °C d). This second model is available `GRT.GH()`.

```
# sample code (not executed)
# Tlev <- c(2, 5, 10, 15, 20, 25)
# GR <- c(0, 0, 0.21, 0.49, 0.68, 0.86)
# modGH <- drm(GR ~ Tlev, fct = GRT.GH())
# library(sandwich); library(lmtest)
# coeftest(modGH, vcov = sandwich)
# plot(modGH, log="", xlim = c(0, 25), legendPos = c(5, 1.2),
#      xlab = "Temperature (°C)")
```

10.2.3.2 GRPsiPol() - GRPsiPol2()

In my experience, I have found that the relationship between GR and water potential in the substrate may, sometimes, be curvilinear. For these situations, I have successfully used the following equations:

$$GR = \frac{\max\left[\Psi, \Psi_b\right]^2 - \Psi_b^2}{\theta_H}$$

and:

$$GR = \frac{\left(\max\left[\Psi, \Psi_b\right] - \Psi_b\right)^2}{\theta_H}$$

Both models can be fitted in R, by using the two functions `GRPsi.Pol()` and `GRPsi.Pol2()`

```
# sample code (not executed)
# Psi <- c(-2, -1.5, -1.2, -1, -0.8, -0.6, -0.4, -0.25,
#          -0.12, -0.06, -0.03, 0)
# GR <- c(0, 0, 0, 0, 0.0585, 0.094, 0.1231, 0.1351,
#         0.1418, 0.1453, 0.1458, 0.1459)
# Psi2 <- c(-0.5, -0.6, -0.7, -0.8, -0.9, -1, -1.1, -1.2,
#           -1.5)
# GR2 <- c(1.4018, 1.0071, 0.5614, 0.3546, 0.2293, 0, 0,
#          0, 0)
#
#
# modHT <- drm(GR ~ Psi, fct = GRPsiPol())
# modHT2 <- drm(GR2 ~ Psi2, fct = GRPsiPol2())
# par(mfrow = c(1,2))
# plot(modHT, log="", legendPos = c(-1.5, 0.15),
#      ylim = c(0, 0.20), xlab = "Water potential (MPa)")
# plot(modHT2, log="", legendPos=c(-1.3, 1),
#      xlab = "Water potential (MPa)")
```

10.2.3.3 PmaxPsi1() and PmaxT1()

All the previous models tend to go up to infinity when the predictor value (temperature or water potential) goes to infinity. In some instances, we may need an asymptotic model, to describe the response of the maximum proportion of germinated seeds to soil humidity (Onofri et al., 2018).

In practice, we could use a shifted exponential model:

$$\pi = G\left[1 - exp\left(\frac{\max\left[\Psi, \Psi_b\right] - \Psi_b}{\sigma}\right)\right]$$

where π is the proportion of germinated seeds, G is the fraction of non-germinable seeds (e.g., dormant seeds) and σ describes how quickly the population of seeds responds to increased humidity in the substrate. This model can be fitted by using the R function the self-starters `PmaxPsi1()`.

If we reverse the sign of σ in the previous equation, we obtain a decreasing trend, which might be useful to describe the effect of super-optimal temperatures on the proportion of germinated seeds, going down to 0 at the ceiling temperature threshold. Also this model is available in R, under the name `PmaxT1()`. `PmaxPsi1()` `PmaxT1()` are two equivalent R functions, apart from the name of model parameters.

```
# sample code (not executed)
# par(mfrow = c(1,2))
# # Pmax vs Psi
```

```
# Psi <- seq(-2.2, 0, by = 0.2)
# Pmax <- c(0, 0, 0.076, 0.413, 0.514, 0.643, 0.712,
#           0.832, 0.865, 0.849, 0.89, 0.90)
# mod <- drm(Pmax ~ Psi, fct = PmaxPsi1())
# plot(mod, log = "", xlab = "Water potential (MPa)",
#      ylab = "Proportion of germinating seeds")
#
# # Pmax vs temperature
# Tval <- c(0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5,
#           20, 22.5, 25, 27.5, 30, 32.5, 35)
# Pmax2 <- c(0.79, 0.81, 0.807, 0.776, 0.83,
#            0.73, 0.744, 0.73, 0.828, 0.818,
#            0.805, 0.706, 0.41, 0.002, 0)
# mod2 <- drm(Pmax2 ~ Tval, fct = PmaxT1())
# plot(mod2, log = "", xlab = "Temperature (°C)",
#      ylab = "Proportion of germinating seeds")
```

10.2.3.4 GRT.BS()

A broken-stick trend, as the one depicted in the first Figure above was used by Alvarado and Bradford (2002) to model the effect of temperature on the germination rate. Their equation is:

$$GR = \frac{\max\left[\min\left[T, T_o\right], Tb\right] - T_b}{\theta_T} \left\{1 - k\left(\max\left[T, T_o\right] - T_o\right)\right\}$$

The right factor is only meaningful when it is positive, that happens when $T < T_c$, i.e. when the environmental temperature is lower than the ceiling temperature. On this basis, the ceiling temperature is equal to:

$$T_c = \frac{1}{k} + T_o$$

The above equation can be easily fitted with the `GRT.BS()` function in the 'drcSeedGerm' package. We have also implemented the reparameterised equation, where the parameter 'k' is replaced with $1/(T_c - T_b)$:

$$GR = \frac{\min\left[T, T_o\right] - T_b}{\theta_T} \left\{1 - \frac{\min\left[\max\left[T, T_o\right], T_c\right] - T_o}{T_c - T_o}\right\}$$

This reparameterised equation is available as `GRT.BSb()`; it is handy, because the ceiling temperature is included as a parameter, but its fitting properties are not as good as those of the previous equation.

```
# sample code
# Tval <- c(2, 5, 10, 15, 20, 25, 30, 35, 40)
# GR <- c(0, 0, 0.209, 0.435, 0.759, 0.821, 0.417, 0.145, 0)
#
# modBS <- drm(GR ~ Tval, fct = GRT.BS())
# plot(modBS, log="", xlim = c(0, 40), ylim=c(0,1.2),
#      legendPos = c(5, 1.0), xlab = "Temperature (°C)")
# coeftest(modBS, vcov = sandwich)
#
# # Reparameterised (self-starter is less accurate)
# modBS <- drm(GR ~ Tval, fct = GRT.BSb())
# plot(modBS, log="", xlim = c(0, 40), ylim=c(0,1.2),
#      legendPos = c(5, 1.0), xlab = "Temperature (°C)")
# coeftest(modBS, vcov = sandwich)
```

10.2.3.5 GRT.RF()

Broken-stick trends may not be reasonable for biological processes, which might be better described by curvilinear equations. Rowse and Finch-Savage (2003) proposed another equation with two components: the first one depicts a linear increase of the GR value with temperature, which is off-set by the second component, starting from $T = T_d$, which is close to (but not coincident with) T_o. The equation is:

$$GR = \frac{\max(T, T_b) - T_b}{\theta_T} \{1 - k[\max(T, T_d) - T_d]\}$$

The optimal temperature can be derived as:

$$T_o = \frac{1 + kT_b + kT_d}{2k}$$

while the ceiling temperature is:

$$T_c = \frac{1}{k} + T_d$$

For this equation, you will find the `GRT.RF()` self-starter in the 'drcSeedGerm' package. We also provide the self-starter `GRT.RFb()`, where the parameters 'k' is replaced by $ 1/(T_c - T_d)$:

$$GR = \frac{\max[T, T_b] - T_b}{\theta_T} \left\{1 - \frac{[\max(T, T_d) - T_d]}{T_c - T_d}\right\}$$

This reparameterised equation contains the ceiling temperature as a parameter, but its fitting properties are as good as those pf the previous equation.

10.2.3.6 GRT.M()

According to Mesgaran et al (2017), the negative and positive effects of temperature coexist for all temperatures above T_b. Their proposed equation is:

$$GR = \frac{\max(T, T_b) - T_b}{\theta_T} \{1 - k[\min(T, T_c) - T_b]\}$$

This equation is only defined from base to ceiling temperature, while it is 0 elsewhere. The ceiling temperature is:

$$T_c = \frac{1}{k} + T_b$$

It is also easy to see that the GR value is a second-order polynomial function of $T - T_b$ and, therefore, the curve is symmetric around the optimal temperature value, which can be derived as:

$$T_o = \frac{T_c - T_b}{2} + T_b$$

For this model, the self-starting function in `drcSeedGerm` is `GRT.M()`. The model can also be reparameterised to include the ceiling temperature as an explicit parameter:

$$GR = \frac{\max[T, T_b] - T_b}{\theta_T} \left[1 - \frac{\min[T, T_c] - T_b}{T_c - T_b}\right]$$

This reparameterised model is available as `GRT.Mb()`.

```
# Sample code (not executed)
# Tval <- c(2, 5, 10, 15, 20, 25, 30, 35, 40)
# GR <- c(0, 0, 0.209, 0.435, 0.759, 0.821, 0.417, 0.145, 0)
# modM <- drm(GR ~ Tval, fct = GRT.Mb())
# plot(modM, log="", xlim = c(0, 40), ylim=c(0,1.2),
#      legendPos = c(5, 1.0), xlab = "Temperature (°C)")
# coeftest(modM, vcov. = sandwich)
```

10.2.3.7 GRT.Ex()

All the equations above use a product, wherein the first term represents the accumulation of thermal time and the second term may be seen as a switch-off term that is 1 either when $T < T_o$ (Alvarado-Bradford equation) or $T < T_d$ (Rowse-Fintch-Savage equation) or $T = T_b$ (Mesgaran equation) and decreases progressively as temperature increases. In all the above equations, the switch-off term is linear, although we can use other types of switch-off terms, to obtain

more flexible models. One possibility is to use an exponential switch-off term, as in the equation below:

$$GR = \frac{\max[T, T_b] - T_b}{\theta_T} \left\{ \frac{1 - \exp[k(\min[T, T_c] - T_c)]}{1 - \exp[k(T_b - T_c)]} \right\}$$

where k is the switch-off parameter: the lower the value, the higher the negative effect of temperature at super-optimal levels. The response of GR to temperature is highly asymmetric with a slow increase below T_o and a steep drop afterwards.

I have successfully used this model in Catara et al (2016) and Masin et al (2017). The self-starting function in R is `GRT.Ex()`.

```
# Sample code
# Tval <- c(2, 5, 10, 15, 20, 25, 30, 35, 40)
# GR <- c(0, 0, 0.209, 0.435, 0.759, 0.821, 0.917, 0.445, 0)
#
# modExb <- drm(GR ~ Tval, fct = GRT.Ex())
# summary(modExb)
# plot(modExb, log="", xlim = c(0, 40), ylim=c(0,1.2),
#     legendPos = c(5, 1.0), xlab = "Temperature (°C)")
```

Go up

10.2.3.8 GRT.YL()

Another switch-off function can be derived from the simple yield loss function devised by Kropff and van Laar (1993). It is very flexible, as it may depict different types of relationships between temperature and base water potential, according to the value taken by the parameter q.

$$GR(g,T) = \frac{\max[T, T_b] - T_b}{\theta_T} \left(1 - \frac{q^{\frac{\min[T, T_c] - T_b}{T_c - T_b}}}{1 + (q-1)^{\frac{T - T_b}{T_c - T_b}}} \right)$$

In R, this model can be fitted by using the self-starter `GRT.YL()`.

```
# sample code
# Tval <- c(2, 5, 10, 15, 20, 25, 30, 35, 40)
# GR <- c(0, 0, 0.209, 0.435, 0.759, 0.821, 0.917, 0.445, 0)
# modYL <- drm(GR ~ Tval, fct = GRT.YL())
# plot(modYL, log="", xlim = c(0, 40), ylim=c(0,1.2),
#      legendPos = c(5, 1.0), xlab = "Temperature (°C)")
```

Go up

10.2.4 Example 1

This dataset describes the germination of rapeseed (cv. Excalibur) at different water potential levels in the substrate. It has been already used for fitting a hydro-time model in one step (see here); in this present post, we try a different line of attack.

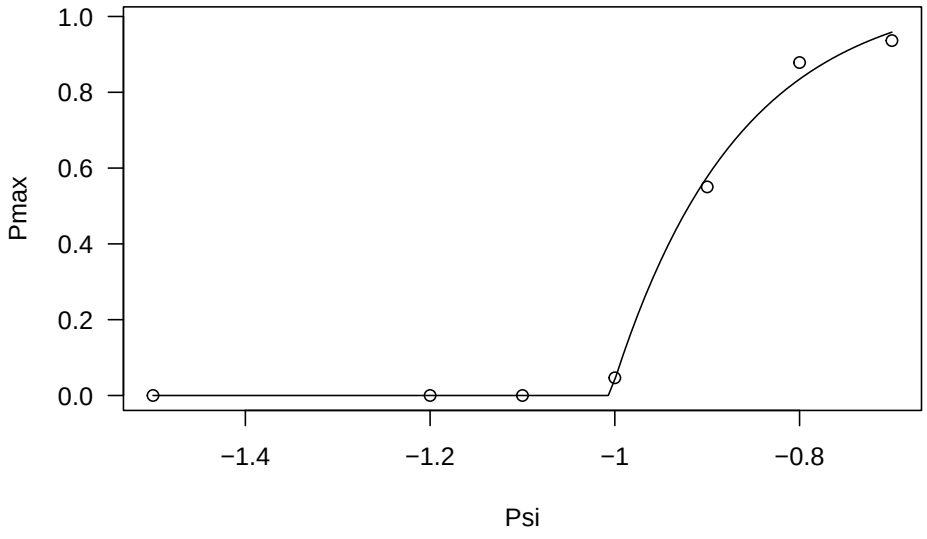
First of all, we remove all dishes with water potential levels higher than -0.7 MPa, because the germinations were too quick to obtain a reliable estimate of the whole time-to-event curve. Next, we independently ('separate = T') fit a time to event model to the data observed in each dish. Lately, for each time to event curve, we retrieve the maximum proportion of germinated seeds (Pmax, i.e. the 'd' parameter of the time-to-event curve) and the germination rates for the 10^{th}, 30^{th} and 50^{th} percentile.

```
# First-step of data analyses
data(rape2G)
rape2G <- rape2G %>%
  dplyr::filter(Psi <=-0.7 & CV == "Excalibur") %>%
  mutate(Dish2 = paste(Dish, Psi, sep = ":"))

# model fit
mod.first <- drmte(nSeeds ~ timeBef + timeAf,
                   data = rape2G,
                   fct = LL.3(), curveid = Dish2,
                   separate = T)

# Retrieve maximum proportion of germinated seeds
Pmax <- coef(mod.first)[substr(names(coef(mod.first)), 1, 1) == "d"]
PmaxList <- tibble(Pmax =  Pmax) %>%
  mutate(temp = names(Pmax), .before = Pmax) %>%
  separate(col = "temp", into = c("n", "Dish", "Psi"),
           sep = ":") %>%
  mutate(Psi = as.numeric(Psi)) %>%
  select(-1)
head(PmaxList)
## # A tibble: 6 x 3
##   Dish    Psi  Pmax
##   <chr> <dbl> <dbl>
## 1 64     -0.7 0.901
## 2 65     -0.7 0.986
## 3 66     -0.7 0.922
## 4 67     -0.8 0.914
## 5 68     -0.8 0.887
```

```
## 6 69     -0.8 0.835
# Retrieve the GR values
GR <- quantile(mod.first, rate = T, probs = c(0.1, 0.3, 0.5))
GRlist <- tibble(temp = row.names(GR), GR, row.names = NULL) %>%
  separate(col = "temp", into = c("Dish", "Psi", "g"),
           sep = ":") %>%
  mutate(Psi = as.numeric(Psi)) %>%
  remove_rownames()
head(GRlist)
## # A tibble: 6 x 5
##   Dish    Psi g     Estimate     SE
##   <chr> <dbl> <chr>    <dbl>  <dbl>
## 1 64     -0.7 10%      0.581 0.0895
## 2 64     -0.7 30%      0.416 0.0397
## 3 64     -0.7 50%      0.333 0.0239
## 4 65     -0.7 10%      0.718 0.149
## 5 65     -0.7 30%      0.468 0.0595
## 6 65     -0.7 50%      0.357 0.0330
```

Now, we are ready to move on to the second step of data analysis. Relating to the Pmax value, we can see that this values stays constant and equal to 0 up to -1 MPa and increases steadily afterwords. We can model this behaviour by using the `PmaxPsi1()` function, as shown in the box below.

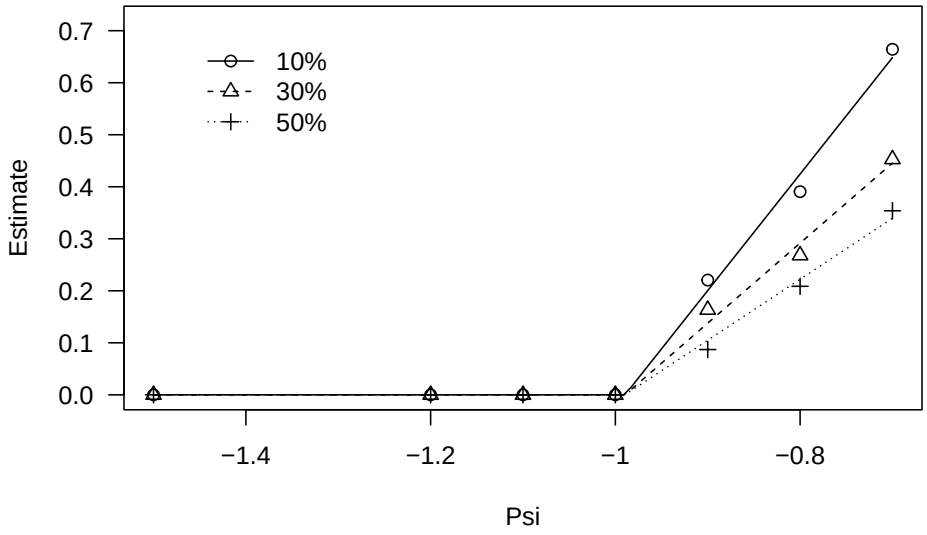
```
modPmax <- drm(Pmax ~ Psi, data = PmaxList,
               fct = PmaxPsi1())
plot(modPmax, log = "")
```



```
coeftest(modPmax, vcov. = sandwich)
##
## t test of coefficients:
##
##                      Estimate Std. Error   t value  Pr(>|t|)
## G:(Intercept)       1.0737381  0.0746708   14.3796 2.608e-11 ***
## Psib:(Intercept)   -1.0053956  0.0016495 -609.5109 < 2.2e-16 ***
## sigma:(Intercept)   0.1367469  0.0272873    5.0114 9.058e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Regarding the germination percentiles, a look at the data shows that, for all percentiles, germination rates stay constant up to -1 MPa and, afterwords, they increase linearly. We can model this behaviour by using the `GRPsiLin()` equation and, following Bradford (2002), we code a common base water potential level for the different germination percentiles. The code is given in the box below.

```
modGR <- drm(Estimate ~ Psi, data = GRlist,
               fct = GRPsi.Lin(), curveid = g,
             pmodels = list(~1, ~g - 1))
plot(modGR, log = "",
             legendPos = c(-1.3, 0.7))
```




```
coeftest(modGR, vcov. = sandwich)
##
## t test of coefficients:
##
##                    Estimate Std. Error  t value  Pr(>|t|)
```

```
## Psib:(Intercept) -0.989504   0.031937 -30.9832 < 2.2e-16 ***
## thetaH:g10%       0.446550   0.081366   5.4881 8.977e-07 ***
## thetaH:g30%       0.649827   0.099213   6.5498 1.557e-08 ***
## thetaH:g50%       0.852974   0.134058   6.3627 3.209e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Usually, we are interested in the base osmotic potential level (Ψ_b) that is given in the output of the `coeftest()` method. We used `coeftest()` in the `lmtest` package for reasons that will be clearer later on.

10.2.5 Example 2

This second dataset was obtained from a germination assays with barley, where three replicates of 50 seeds were placed in Petri dishes and assayed at 9 constant temperature levels (1, 3, 7, 10, 15, 20, 25, 30, 35, 40 °C). Germinated seeds were counted and removed daily for 10 days. We have already presented this analysis in a previous paper (Onofri et al., 2018), although in this post we use a different (and updated) coding.

Also in this second example, the first step of data analysis is based on loading the data and fitting a separate time-to-event curve to the data at each temperature level.

```
data(barley)
barley <- barley %>%
  mutate(TempF = factor(Temp))

mod1 <- drmte(nSeeds ~ timeBef + timeAf, fct=W2.3(),
      curveid = TempF,
      data = barley,
      separate = T)
```

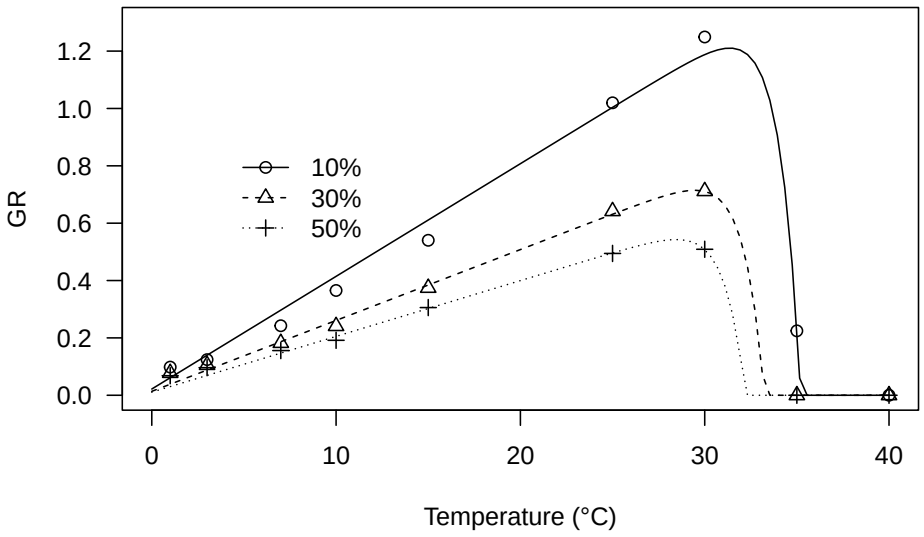
Now, we retrieve the germination rates for the 10^th, 30th and 50th percentile; for analogy with the published paper, we restrict the percentiles to the germinated fraction, althoug it might be better to avoid such a restriction.

```
GR <- quantile(mod1, rate = T, restricted = T,
               probs = c(0.1, 0.3, 0.5),
                display = F)
GRlist <- tibble(temp = row.names(GR), GR, row.names = NULL) %>%
  separate(col = "temp", into = c("Temp", "g"),
           sep = ":") %>%
  mutate(Temp = as.numeric(Temp)) %>%
  remove_rownames()
head(GRlist)
```

```
## # A tibble: 6 x 4
##    Temp g     Estimate      SE
##   <dbl> <chr>    <dbl>   <dbl>
## 1     1 10%     0.0982 0.00328
## 2     1 30%     0.0777 0.00177
## 3     1 50%     0.0682 0.00125
## 4     3 10%     0.124  0.00307
## 5     3 30%     0.105  0.00173
## 6     3 50%     0.0962 0.00126
```

The behaviour of germination rates against temperature can be described, e.g., by using the threshold model proposed by Masin et al. (2017), that is implemented in the R function `GRT.Ex()`, as shown in the box below. Preliminary trials show that the three percentiles share the same 'k' parameter and base temperature level, which we can request by using the 'pmodels' argument.

```
modGR <- drm(Estimate ~ Temp, data = GRlist,
               fct = GRT.Ex(),
             curveid = g, pmodels = list(~1, ~1, ~g - 1, ~g -1))
coeftest(modGR, vcov. = sandwich)
##
## t test of coefficients:
##
##                  Estimate Std. Error   t value  Pr(>|t|)
## k:(Intercept)    0.876849   0.138731    6.3205 4.577e-06 ***
## Tb:(Intercept) -0.553318   0.774412   -0.7145    0.4836
## Tc:g10%        35.200942   0.021761 1617.5850 < 2.2e-16 ***
## Tc:g30%        33.235421   0.321386  103.4129 < 2.2e-16 ***
## Tc:g50%        32.198489   0.225161  143.0021 < 2.2e-16 ***
## ThetaT:g10%    25.454497   1.788195   14.2347 1.378e-11 ***
## ThetaT:g30%    40.458899   2.001762   20.2116 2.630e-14 ***
## ThetaT:g50%    51.413920   2.415663   21.2836 1.025e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
plot(modGR, log = "", type = "all", xlim = c(0, 40),
     ylim = c(0, 1.3),
     ylab = "GR", xlab = "Temperature (°C)",
     legendPos = c(12, 0.9))
```



10.2.6 Warning message!

When we collect data about the response of germination rates to temperature and use them to parameterise nonlinear regression models by using nonlinear least squares, the basic assumption of homoscedasticity is rarely tenable. **We should not forget this!**. In the above examples I used a robust variance-covariance sandwich estimator (Zeileis, 2006; see the use of the `coeftest()` method, instead of the `summary()` method), although other techniques can be successfully used to deal with this problem.

10.3 Carrying over information to the 2nd step

(yet to be done)

Chapter 11

Examples and case-studies

11.1 Fitting hydro-thermal-time-models to seed germination data

This dataset was obtained from previously published work (Mesgaran et al., 2017) with *Hordeum spontaneum* [C. Koch] Thell. The germination assay was conducted using four replicates of 20 seeds tested at six different water potential levels (0, −0.3, −0.6, −0.9, −1.2 and −1.5 MPa). Osmotic potentials were produced using variable amount of polyethylene glycol (PEG, molecular weight 8000) adjusted for the temperature level. Petri dishes were incubated at six constant temperature levels (8, 12, 16, 20, 24 and 28 °C), under a photoperiod of 12 h. Germinated seeds (radicle protrusion $>$ 3 mm) were counted and removed daily for 20 days.

This dataset is available as `hordeum` in the `drcSeedGerm` package, which needs to be installed from github (see below), together the package `drcte`, which is necessary to fit time-to-event models. The following code loads the necessary packages, loads the dataset `rape` and shows the first six lines.

```
# Installing packages (only at first instance)
# library(devtools)
# install_github("OnofriAndreaPG/drcSeedGerm")
# install_github("OnofriAndreaPG/drcte")
library(drcSeedGerm)
library(tidyverse)
data(hordeum)
head(hordeum)
##   temp water Dish timeBef timeAf nViable nSeeds nCum
## 1    8  -1.5    1       0     24      20      0    0
## 2    8  -1.5    1      24     48      20      0    0
```

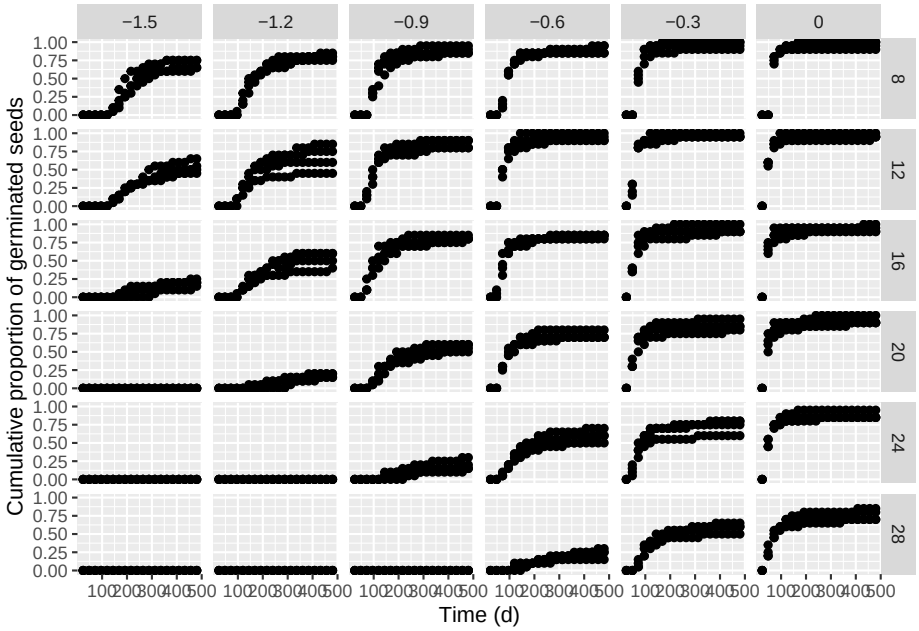
```
## 3    8  -1.5    1      48     72       20       0    0
## 4    8  -1.5    1      72     96       20       0    0
## 5    8  -1.5    1      96    120       20       0    0
## 6    8  -1.5    1     120    144       20       1    1
```

11.1.1 Preliminary analyses

First of all, it is necessary to mention that **this dataset was already analysed in Onofri et al. (2018; Example 2) by using the same methodology, although, in that paper, the R implementation was different (see Supplemental Material) and it is now outdated.**

In the above data frame, 'timeAf' represents the moment when germinated seeds were counted, while 'timeBef' represents the previous inspection time (or the beginning of the assay). The column 'nSeeds' is the number of seeds that germinated during the time interval between 'timeBef' and 'timeAf. The 'nCum' column contains the cumulative number of germinated seeds and it is not necessary for time-to-event model fitting, although we can use it for plotting purposes.

```
hordeum <- hordeum %>%
  mutate(propCum = nCum/nViable)

ggplot(data = hordeum, mapping = aes(timeAf, propCum)) +
  geom_point() +
  facet_grid(temp ~ water) +
  scale_x_continuous(name = "Time (d)") +
  scale_y_continuous(name = "Cumulative proportion of germinated seeds")
```



We see that the germination time-course is strongly affected by both temperature and water potential in the substrate and, consequently, our obvious interest is to model the effects of those environmental covariates. In our manuscript, we started from the idea that a parametric time-to-event curve is defined as a cumulative probability function (Φ), with three parameters:

$$P(t) = \Phi\left(b, d, e\right)$$

where $P(t)$ is the cumulative probability of germination at time t, e is the median germination time, b is the slope at the inflection point and d is the maximum germinated proportion. The most obvious extension is to allow for different b, d and e values for each of the i$^{\text{th}}$ combinations of water potential (Ψ) and temperature level (T):

$$P(t, \Psi, T) = \Phi\left(b_i, d_i, e_i\right)$$

From the graph above, we see several 'degenerated' time-to-event curves, where no germinations occurred (e.g., see the graph at -1.5 MPa and 28°C). In order to avoid problems with those curves, we can use the `drmte()` function and set the `separate = TRUE` argument, so that the different curves are fitted independently of one another and the degenerated curves are recognised and skipped, without stopping the execution in R. In particular, where no time-course of events can be estimated, it is assumed that there is no progress to germination during the study-period and that the cumulative proportion of germinated seeds remains constant across that period. Consequently, the `drmte()` function resorts to

fitting a simpler model, where the only d parameter is estimated (that is the maximum fraction of germinated seeds).

```
hordeum <- hordeum %>%
  mutate(comb = factor( factor(water):factor(temp)))
mod1 <- drmte(nSeeds ~ timeBef + timeAf, data = hordeum,
              curveid = comb, fct = loglogistic(),
              separate = TRUE)
summary(mod1)
##
## Model fitted: Separate fitting of several time-to-event curves
##
## Robust estimation: no
##
## Parameter estimates:
##
##              Estimate Std. Error t-value   p-value
## b:-1.5:8    5.577670   0.671086  8.3114 < 2.2e-16 ***
## d:-1.5:8    0.705845   0.051801 13.6260 < 2.2e-16 ***
## e:-1.5:8  203.532986   8.746552 23.2701 < 2.2e-16 ***
## b:-1.5:12   4.036639   0.633750  6.3694 1.897e-10 ***
## d:-1.5:12   0.566163   0.061406  9.2201 < 2.2e-16 ***
## e:-1.5:12 232.198040  17.347102 13.3854 < 2.2e-16 ***
## b:-1.5:16   4.130712   1.174395  3.5173 0.0004359 ***
## d:-1.5:16   0.226248   0.057013  3.9683 7.238e-05 ***
## e:-1.5:16 293.584384  41.294139  7.1096 1.164e-12 ***
## d:-1.5:20   0.000000   0.000000     NaN       NaN
## d:-1.5:24   0.000000   0.000000     NaN       NaN
## d:-1.5:28   0.000000   0.000000     NaN       NaN
## b:-1.2:8    4.761454   0.531157  8.9643 < 2.2e-16 ***
## d:-1.2:8    0.803436   0.044972 17.8653 < 2.2e-16 ***
## e:-1.2:8  152.794391   7.123662 21.4489 < 2.2e-16 ***
## b:-1.2:12   4.165847   0.518344  8.0368 9.050e-16 ***
## d:-1.2:12   0.667099   0.053329 12.5092 < 2.2e-16 ***
## e:-1.2:12 145.571114   8.562685 17.0006 < 2.2e-16 ***
## b:-1.2:16   3.848053   0.556564  6.9140 4.713e-12 ***
## d:-1.2:16   0.536017   0.057474  9.3262 < 2.2e-16 ***
## e:-1.2:16 175.692653  13.009048 13.5054 < 2.2e-16 ***
## b:-1.2:20   4.679284   1.377885  3.3960 0.0006838 ***
## d:-1.2:20   0.191881   0.049814  3.8520 0.0001172 ***
## e:-1.2:20 291.173467  34.870683  8.3501 < 2.2e-16 ***
## d:-1.2:24   0.000000   0.000000     NaN       NaN
## d:-1.2:28   0.000000   0.000000     NaN       NaN
## b:-0.9:8    5.237479   0.550871  9.5076 < 2.2e-16 ***
## d:-0.9:8    0.887920   0.035345 25.1218 < 2.2e-16 ***
## e:-0.9:8  111.384281   4.427060 25.1599 < 2.2e-16 ***
```

```
## b:-0.9:12   4.368990   0.475050  9.1969 < 2.2e-16 ***
## d:-0.9:12   0.850876   0.039968 21.2889 < 2.2e-16 ***
## e:-0.9:12  99.501087   4.833892 20.5841 < 2.2e-16 ***
## b:-0.9:16   3.581796   0.400722  8.9383 < 2.2e-16 ***
## d:-0.9:16   0.816041   0.043892 18.5922 < 2.2e-16 ***
## e:-0.9:16 105.233306   6.461986 16.2850 < 2.2e-16 ***
## b:-0.9:20   3.536663   0.502112  7.0436 1.874e-12 ***
## d:-0.9:20   0.572734   0.056895 10.0665 < 2.2e-16 ***
## e:-0.9:20 154.543645  11.924187 12.9605 < 2.2e-16 ***
## b:-0.9:24   3.113060   0.947598  3.2852 0.0010191 **
## d:-0.9:24   0.233165   0.062870  3.7087 0.0002084 ***
## e:-0.9:24 269.509412  55.515112  4.8547 1.206e-06 ***
## d:-0.9:28   0.000000   0.000000     NaN       NaN
## b:-0.6:8    5.077994   0.538321  9.4330 < 2.2e-16 ***
## d:-0.6:8    0.900202   0.033548 26.8334 < 2.2e-16 ***
## e:-0.6:8   92.146203   3.725399 24.7346 < 2.2e-16 ***
## b:-0.6:12   5.564019   0.586966  9.4793 < 2.2e-16 ***
## d:-0.6:12   0.937528   0.027059 34.6476 < 2.2e-16 ***
## e:-0.6:12  74.982399   2.818636 26.6024 < 2.2e-16 ***
## b:-0.6:16   4.144136   0.458813  9.0323 < 2.2e-16 ***
## d:-0.6:16   0.837853   0.041271 20.3014 < 2.2e-16 ***
## e:-0.6:16  75.109147   3.897277 19.2722 < 2.2e-16 ***
## b:-0.6:20   4.399408   0.510297  8.6213 < 2.2e-16 ***
## d:-0.6:20   0.725331   0.049946 14.5224 < 2.2e-16 ***
## e:-0.6:20  83.735884   4.468212 18.7404 < 2.2e-16 ***
## b:-0.6:24   3.269465   0.443121  7.3783 1.603e-13 ***
## d:-0.6:24   0.607528   0.055700 10.9071 < 2.2e-16 ***
## e:-0.6:24 125.859897   9.985513 12.6042 < 2.2e-16 ***
## b:-0.6:28   2.959772   0.767672  3.8555 0.0001155 ***
## d:-0.6:28   0.265633   0.059199  4.4871 7.220e-06 ***
## e:-0.6:28 233.440197  40.981613  5.6962 1.225e-08 ***
## b:-0.3:8    6.489283   0.700089  9.2692 < 2.2e-16 ***
## d:-0.3:8    0.962474   0.021243 45.3069 < 2.2e-16 ***
## e:-0.3:8   72.248403   2.326579 31.0535 < 2.2e-16 ***
## b:-0.3:12   5.571444   0.614984  9.0595 < 2.2e-16 ***
## d:-0.3:12   0.962476   0.021243 45.3075 < 2.2e-16 ***
## e:-0.3:12  56.804335   2.154440 26.3662 < 2.2e-16 ***
## b:-0.3:16   3.759741   0.406837  9.2414 < 2.2e-16 ***
## d:-0.3:16   0.925252   0.029452 31.4157 < 2.2e-16 ***
## e:-0.3:16  53.997403   3.004606 17.9715 < 2.2e-16 ***
## b:-0.3:20   3.455788   0.382078  9.0447 < 2.2e-16 ***
## d:-0.3:20   0.863032   0.038527 22.4004 < 2.2e-16 ***
## e:-0.3:20  56.589306   3.525940 16.0494 < 2.2e-16 ***
## b:-0.3:24   3.219012   0.384905  8.3631 < 2.2e-16 ***
## d:-0.3:24   0.739176   0.049321 14.9869 < 2.2e-16 ***
```

```
## e:-0.3:24  72.448658   5.145402 14.0803 < 2.2e-16 ***
## b:-0.3:28   3.384884   0.449578  7.5290 5.110e-14 ***
## d:-0.3:28   0.591722   0.055517 10.6583 < 2.2e-16 ***
## e:-0.3:28 111.482975   8.597283 12.9672 < 2.2e-16 ***
## b:0:8       8.055166   0.890908  9.0415 < 2.2e-16 ***
## d:0:8       0.962496   0.021229 45.3391 < 2.2e-16 ***
## e:0:8      64.579539   1.810276 35.6739 < 2.2e-16 ***
## b:0:12      4.597148   0.515642  8.9154 < 2.2e-16 ***
## d:0:12      0.962502   0.021236 45.3239 < 2.2e-16 ***
## e:0:12     45.258150   2.124034 21.3076 < 2.2e-16 ***
## b:0:16      4.519281   0.504988  8.9493 < 2.2e-16 ***
## d:0:16      0.937575   0.027033 34.6821 < 2.2e-16 ***
## e:0:16     41.805944   2.052607 20.3672 < 2.2e-16 ***
## b:0:20      3.833745   0.419743  9.1335 < 2.2e-16 ***
## d:0:20      0.925095   0.029446 31.4163 < 2.2e-16 ***
## e:0:20     43.588297   2.434550 17.9040 < 2.2e-16 ***
## b:0:24      4.103341   0.467512  8.7770 < 2.2e-16 ***
## d:0:24      0.875057   0.036978 23.6641 < 2.2e-16 ***
## e:0:24     47.161280   2.550713 18.4894 < 2.2e-16 ***
## b:0:28      2.895784   0.341099  8.4896 < 2.2e-16 ***
## d:0:28      0.764633   0.047737 16.0176 < 2.2e-16 ***
## e:0:28     63.034588   4.939641 12.7610 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

11.1.2 A better modelling approach

The previous approach is clearly sub-optimal, as the temperature and water potential levels are regarded as factors, i.e. as nominal classes with no intrinsic orderings and distances. It would be much better to recognise that temperature and water potential are continuous variables and, consequently, code a time-to-event model where the three parameters are expressed as continuous functions of Ψ and T:

$$P(t, \Psi, T) = \Phi\left[b(\Psi, T), d(\Psi, T), e(\Psi, T)\right]$$

In the above mentioned manuscript (Onofri et al., 2018; example 2) we used a log-logistic cumulative distribution function:

$$P(t, \Psi, T) = \frac{d(\Psi, T)}{1 + \exp\left\{b\left[\log(t) - \log(e(\Psi, T))\right]\right\}}$$

Considering that the germination rate is the inverse of germination time, we replaced $e(\Psi, T) = 1/GR_{50}(\Psi, T)$ and used the following sub-models:

$$d(\Psi, T) = \max\left\{G\left[1 - \exp\left(\frac{\Psi - \Psi_b - k(T - T_b)}{\sigma_{\Psi_b}}\right)\right]; 0\right\}$$

$$GR_{50}(\Psi, T) = \max\left\{\frac{T - T_b}{\theta_{HT}}\left[\Psi - \Psi_b - k(T - T_b)\right]; 0\right\}$$

Please, note that the shape parameter b has been regarded as independent from the environmental covariates. It may be useful to note that the the parameters are:

1. Ψ_b, that is the median base water potential in the seed lot,
2. T_b, that is the base temperature for germination,
3. $\theta_H T$, that is the hydro-thermal-time parameter,
4. σ_{Ψ_b}, that represents the variability of Ψ_b within the population,
5. G, that is the germinable fraction, accounting for the fact that d may not reach 1, regardless of time and water potential.
6. k and b that are parameters of shape.

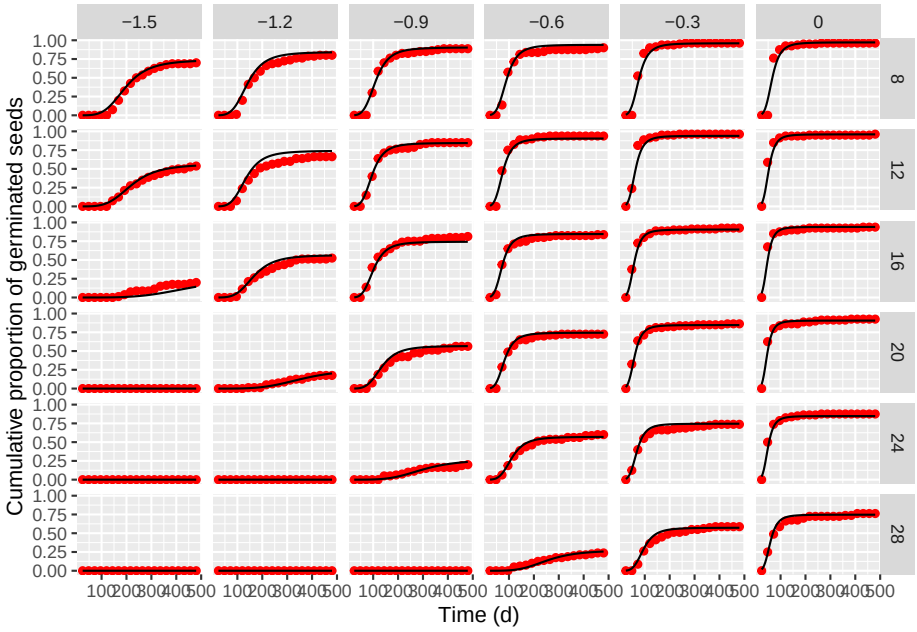
You can get more information from our original paper (Onofri et al., 2018). This hydro-thermal-time model was implemented in R as the `HTTEM()` function, and it is available within the `drcSeedGerm` package; we can fit it by using the `drmte()` function in the `drcte` package, but we need to provide starting values for model parameters, because the self-starting routine is not yet available. Finally, the `summary()` method can be used to retrieve the parameter estimates.

```
# Complex model and slow fitting
modHTTE <- drmte(nSeeds ~ timeBef + timeAf + water + temp,
                 data=hordeum,
                 fct = HTTEM(),
  start=c(0.8,-2, 0.05, 3, 0.2, 2000, 0.5))
summary(modHTTE, robust = T, units = Dish)
##
## Model fitted: Hydro-thermal-time-model (Mesgaran et al., 2017)
##
## Robust estimation: Cluster robust sandwich SEs
##
## Parameter estimates:
##
##                           Estimate  Std. Error  t value Pr(>|t|)
## G:(Intercept)           9.8820e-01  1.1576e-02  85.3692   <2e-16 ***
## Psib:(Intercept)       -2.9133e+00  3.4812e-02 -83.6874   <2e-16 ***
## kt:(Intercept)          7.4228e-02  1.2666e-03  58.6015   <2e-16 ***
## Tb:(Intercept)         -7.4525e-01  3.5254e-01  -2.1139   0.0346 *
## sigmaPsib:(Intercept)  5.5284e-01  2.8976e-02  19.0790   <2e-16 ***
## ThetaHT:(Intercept)    1.3091e+03  4.0638e+01  32.2130   <2e-16 ***
## b:(Intercept)          4.1650e+00  1.1332e-01  36.7548   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It is always important not to neglect a graphical inspection of model fit. The `plot()` method does not work with time-to-event curves with additional covariates (apart from time). However, we can retrieve the fitted data by using the `plotData()` function and use those predictions within the `ggplot()` function. The box below shows the appropriate coding.

```
tab <- plotData(modHTTE)

ggplot() +
  geom_point(data = tab$plotPoints, mapping = aes(x = timeAf, y = CDF),
             col = "red") +
  geom_line(data = tab$plotFits, mapping = aes(x = timeAf, y = CDF)) +
  facet_grid(temp ~ water) +
  scale_x_continuous(name = "Time (d)") +
  scale_y_continuous(name = "Cumulative proportion of germinated seeds")
```



11.1.3 Other HTT models

Mohsen Mesgaran, in an earlier paper (Mesgaran et al., 2017), used a different modelling approach, where he did not directly consider the distribution of germination times within the seed lot, but modelled the distribution of base water potential levels within the lot, under the idea that different germination times

arise as the result of the fact that each seed has its 'intrinsic' water potential level (Bradford, 1992). We do not give any further biological detail, because this is only meant to be a post about R coding; you can refer to Bradford (1992), Mesgaran et al. (2013) and Mesgaran et al. (2017) for this detail. The following equation describes the time-course of germination depending on temperature and water potential level, assuming that the distribution of base water potential in the seed lot is gaussian:

$$P(t,T,\Psi) = \Phi\left\{\frac{\Psi - \left[\frac{\theta_{HT}}{t(T-T_b)}\right] - [\Psi_b + K_t(T-T_b)]}{\sigma_{\Psi_b}}\right\} \qquad \text{(HTTnorm.M)}$$

where P is the proportion of germinated seeds, depending on the time t, temperature T and base water potential Ψ, Φ is a gaussian cumulative distribution function, $\theta_H T$ is the hydro-thermal-time parameter, Ψ_b is the median base water potential in the seed lot, K_t is a parameter that measures the effect of temperature on base water potential, T_b is the base temperature for germination and σ_{Ψ_b} represents the variability of Ψ_b within the population.

This model was coded in the `drcSeedGerm` package as the `HTTnorm.M()` function, which was also modified to fit alternative biological assumptions. For example, if Φ is not gaussian, but it is log-logistic, the model becomes:

$$P(t,T,\Psi) = \frac{1}{1+\exp\left\{-\frac{\log\left[\Psi-\left(\frac{\theta_{HT}}{t(T-T_b)}\right)+\delta\right]-\log[\Psi_b+K_t(T-T_b)+\delta]}{\sigma_{\Psi_b}}\right\}} \qquad \text{(HTTLL.M)}$$

where δ is a shifting parameter, to allow for negative water potential levels. This model was coded in the `drcSeedGerm` package as the `HTTLL.M()` function.

In both models the term $K_T(T-T_b)$ can be modified as $K_t[max(T,T_o)-T_o]$, which changes the way how the germination rate is affected by increasing temperature levels. In the two modified models, the optimal temperature T_o is included as an explicit parameter; the corresponding R functions are `HTTnorm.BS()` and `HTTLL.BS()`.

Let's fit these four models.

```
# HTTnorm.M
modHTTnorm.M <- drmte(nSeeds ~ timeBef + timeAf + water + temp,
                  data=hordeum,
                  fct = HTTnorm.M(),
  start=c(932,-2.5, -3, 0.07, 0.5))
summary(modHTTnorm.M)
##
## Model fitted: Hydrothermal-time model with normal distribution of Psib (Bradford et al., 2002)
```

```
##
## Robust estimation: no
##
## Parameter estimates:
##
##                          Estimate  Std. Error  t-value p-value
## thetaHT:(Intercept)   850.5394623  33.7608043  25.1931  <2e-16 ***
## Tb:(Intercept)          0.5713169   0.3710553   1.5397  0.1236
## Psib50:(Intercept)     -2.3777692   0.0385119 -61.7412  <2e-16 ***
## Kt:(Intercept)          0.0738569   0.0020037  36.8600  <2e-16 ***
## sigmaPsib:(Intercept)   0.4049049   0.0094797  42.7130  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
# HTTnorm.BS
modHTTnorm.BS <- drmte(nSeeds ~ timeBef + timeAf + water + temp,
                 data=hordeum,
                 fct = HTTnorm.BS(),
  start=c(932,-2.5, 15, -3, 0.07, 0.5))
summary(modHTTnorm.BS)
##
## Model fitted: Hydrothermal-time model with normal distribution of Psib (Bradford et
##
## Robust estimation: no
##
## Parameter estimates:
##
##                          Estimate  Std. Error  t-value p-value
## thetaHT:(Intercept)   906.3324894  43.7165251  20.7320  <2e-16 ***
## Tb:(Intercept)         -0.2197905   0.5362506  -0.4099  0.6819
## To:(Intercept)          9.2700969   0.4781515  19.3874  <2e-16 ***
## Psib50:(Intercept)     -1.7649662   0.0357594 -49.3567  <2e-16 ***
## Kt:(Intercept)          0.0755795   0.0021308  35.4696  <2e-16 ***
## sigmaPsib:(Intercept)   0.4042633   0.0094506  42.7767  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
# HTTLL.M
modHTTLL.M <- drmte(nSeeds ~ timeBef + timeAf + water + temp,
                 data=hordeum,
                 fct = HTTLL.M(),
  start=c(832,-2.5, -3, 0.07, 3, 0.5))
summary(modHTTLL.M)
##
## Model fitted: Hydrothermal-time model (Mesgaran et al., 2017)
##
## Robust estimation: no
```

```
##
## Parameter estimates:
##
##                           Estimate  Std. Error  t-value   p-value
## thetaHT:(Intercept)   942.3812703  41.8239049  22.5321 < 2.2e-16 ***
## Tb:(Intercept)         -0.6678997   0.4900833  -1.3628    0.1729
## Psib50:(Intercept)     -2.4978431   0.0383909 -65.0635 < 2.2e-16 ***
## Kt:(Intercept)          0.0732105   0.0020206  36.2313 < 2.2e-16 ***
## delta:(Intercept)       3.4981909   0.2535617  13.7962 < 2.2e-16 ***
## sigmaPsib:(Intercept)   0.1038773   0.0131166   7.9195 2.414e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
# HTTLL.BS
modHTTLL.BS <- drmte(nSeeds ~ timeBef + timeAf + water + temp,
                 data=hordeum,
                 fct = HTTLL.BS(),
  start=c(932,-2.5, 15, -3, 0.07, 3, 0.5))
summary(modHTTLL.BS)
##
## Model fitted: Hydrothermal-time model (Mesgaran et al., 2017)
##
## Robust estimation: no
##
## Parameter estimates:
##
##                           Estimate  Std. Error  t-value   p-value
## thetaHT:(Intercept)    1.4770e+03  8.7763e+01  16.8292 < 2.2e-16 ***
## Tb:(Intercept)        -7.8060e+00  1.1189e+00  -6.9765 3.027e-12 ***
## To:(Intercept)         1.3159e+01  3.5960e-01  36.5937 < 2.2e-16 ***
## Psib50:(Intercept)    -1.6545e+00  1.9710e-02 -83.9418 < 2.2e-16 ***
## Kt:(Intercept)         8.7877e-02  3.8001e-03  23.1247 < 2.2e-16 ***
## delta:(Intercept)      2.6333e+00  9.6670e-02  27.2400 < 2.2e-16 ***
## sigmaPsib:(Intercept)  1.8527e-01  1.6898e-02  10.9641 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The AIC value tells us that, for this dataset, the `HTTLL.BS()` function produced the best fit, confirming its underlying biological assumptions.

```
AIC(modHTTE, modHTTnorm.M, modHTTnorm.BS, modHTTLL.M, modHTTLL.BS)
##                  df      AIC
## modHTTE        3017 14714.05
## modHTTnorm.M   3019 14737.48
## modHTTnorm.BS  3018 14728.87
## modHTTLL.M     3018 14582.64
## modHTTLL.BS    3017 14548.58
```

11.2 Fitting thermal-time-models to seed germination data

11.2.1 A motivating examples

In recent times, we wanted to model the effect of temperature on seed germination for *Hordeum vulgare* and we made an assay with three replicated Petri dishes (50 seeds each) at 9 constant temperature levels (1, 3, 7, 10, 15, 20, 25, 30, 35, 40 °C). Germinated seeds were counted and removed daily for 10 days. This unpublished dataset is available as `barley` in the `drcSeedGerm` package, which needs to be installed from github (see below), together with the `drcte` package for time-to-event model fitting. The following code loads the necessary packages, loads the datasets and shows the first six lines.

```
# Installing packages (only at first instance)
# library(devtools)
# install_github("OnofriAndreaPG/drcSeedGerm")
# install_github("OnofriAndreaPG/drcte")
library(drcSeedGerm)
library(tidyverse)
data(barley)
head(barley)
##   Dish Temp timeBef timeAf nSeeds nCum propCum
## 1    1    1       0      1      0    0       0
## 2    1    1       1      2      0    0       0
## 3    1    1       2      3      0    0       0
## 4    1    1       3      4      0    0       0
## 5    1    1       4      5      0    0       0
## 6    1    1       5      6      0    0       0
```

In this dataset, 'timeAf' represents the moment when germinated seeds were counted, while 'timeBef' represents the previous inspection time (or the beginning of the assay). The column 'nSeeds' is the number of seeds that germinated during each time interval between 'timeBef' and 'timeAf. For the analyses, we will also make use of the 'Temp' (temperature) and 'Dish' (Petri dish) columns.

This dataset was already analysed in Onofri et al. (2018; Example 3 and 4) by using the same methodology, but a different R coding (see the Supplemental Material in that paper), that is now outdated. This post will show you the updated coding.

11.2.2 A first thermal-time model

As we have shown in previous posts (see here and here), the effect of environmental covariates (temperature, in this case) can be simply included by independently coding a different time-to-event curve to each level of that covariate.

In other words, considering that a parametric time-to-event curve is defined as a cumulative probability function (Φ), with three parameters:

$$P(t) = \Phi\left(b, d, e\right)$$

where $P(t)$ is the cumulative probability of germination at time t, e is the median germination time, b is the slope at the inflection point and d is the maximum germinated proportion, the most obvious extension is to allow for different b, d and e values for each of the i^{th} temperature levels (T):

$$P(t, T) = \Phi\left(b_i, d_i, e_i\right)$$

Although the above approach is possible, we will not purse it here, as it is clearly sub-optimal; indeed, such an approach wrongly considers the temperature as a factor, i.e. a set of nominal classes with no intrinsic orderings and distances. Obviosly, we should better regard the temperature as a continuous variable, by coding a time-to-event model where the three parameters are expressed as continuous functions of T:

$$P(t, T) = \Phi\left[b(T), d(T), e(T)\right]$$

For the 'phalaris' dataset (Onofri et al., 2018; example 3) we used a log-logistic cumulative distribution function, with the following sub-models:

$$\begin{cases} P(t,T) = \frac{d(T)}{1+\exp\{b[\log(t)-\log(e(T))]\}} \\ d(T) = G\left[1-\exp\left(-\frac{T_c-T}{\sigma_{T_c}}\right)\right] \\ 1/[e(T)] = GR_{50}(T) = \frac{T-T_b}{\theta_T}\left[1-\frac{T-T_b}{T_c-T_b}\right] \end{cases} \qquad \text{(TTEM)}$$

Please, note that the shape parameter b has been regarded as independent from temperature; the parameters are:

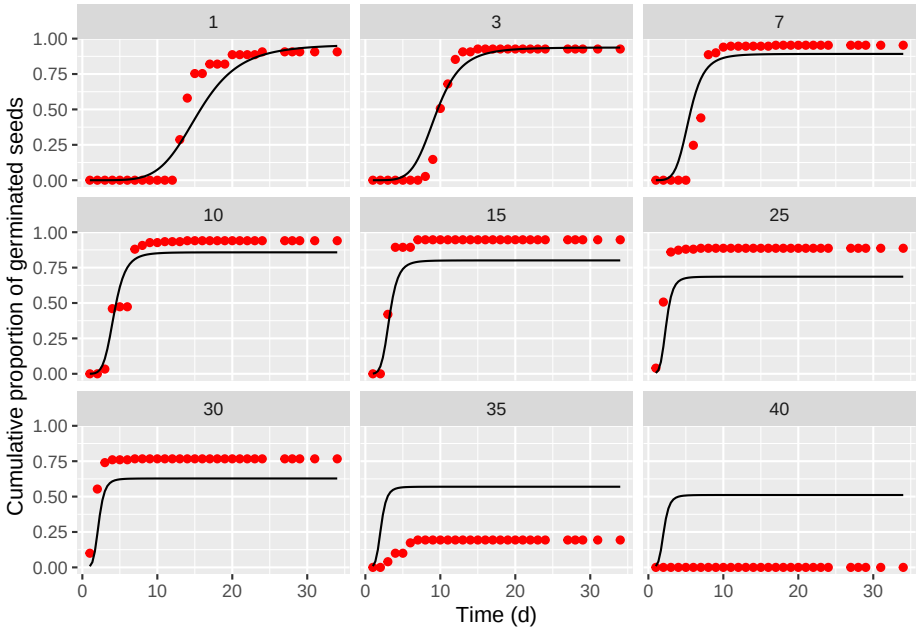
1. G, that is the germinable fraction, accounting for the fact that d may not reach 1, regardless of time and temperature;
2. T_c, that is the ceiling temperature for germination;
3. σ_{T_c}, that represents the variability of T_c within the seed lot;
4. T_b, that is the base temperature for germination;
5. θ_T, that is the thermal-time parameter;
6. b, that is the scale parameters for the log-logistic distribution.

You can get more information from our original paper (Onofri et al., 2018). This thermal-time model was implemented in R as the `TTEM()` function, and it is available within the `drcSeedGerm` package; we can fit it by using the `drmte()` function in the `drcte` package, with no need to provide starting values for model parameters, because a self-starting routine is available. The `summary()` method can be used to retrieve the parameter estimates.

```
# Thermal-Time-to-event model fit
modTTEM <- drmte( nSeeds ~ timeBef + timeAf + Temp, data=barley,
              fct=TTEM())
summary(modTTEM, units = Dish)
##
## Model fitted: Thermal-time model with shifted exponential for Pmax and Mesgaran mod
##
## Robust estimation: Cluster robust sandwich SEs
##
## Parameter estimates:
##
##                        Estimate Std. Error t value  Pr(>|t|)
## G:(Intercept)          12.46391    2.40502  5.1825 2.772e-07 ***
## Tc:(Intercept)         82.66899   23.47042  3.5223  0.000452 ***
## sigmaTc:(Intercept) 1018.92708  322.67791  3.1577  0.001649 **
## Tb:(Intercept)         -1.89327    0.38741 -4.8870 1.236e-06 ***
## ThetaT:(Intercept)     43.41025    4.37185  9.9295 < 2.2e-16 ***
## b:(Intercept)           5.62759    0.69149  8.1383 1.520e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It is always important not to neglect a graphical inspection of model fit. The `plot()` method does not work with time-to-event curves with additional covariates (apart from time). However, we can retrieve the fitted data by using the `plotData()` function and use those predictions within the `ggplot()` function. The box below shows the appropriate coding; the red circles in the graphs represent the observed means, while the black lines are model predictions).

```
tab <- plotData(modTTEM)

ggplot() +
  geom_point(data = tab$plotPoints, mapping = aes(x = timeAf, y = CDF),
             col = "red") +
  geom_line(data = tab$plotFits, mapping = aes(x = timeAf, y = CDF)) +
  facet_wrap( ~ Temp) +
  scale_x_continuous(name = "Time (d)") +
  scale_y_continuous(name = "Cumulative proportion of germinated seeds")
```



The previous graph shows that the `TTEM()` model, that we had successfully used with other datasets, failed to provide an adequate description of the germination time-course for barley. Therefore, we modified one of the sub-models, by adopting the approach highlighted in Rowse and Finch-Savage (2003), where:

$$1/[e(T)] = GR_{50}(T) = \begin{cases} \frac{T-T_b}{\theta_T} & \text{if } T_b < T < T_d \\ \frac{T-T_b}{\theta_T}\left[1 - \frac{T-T_d}{T_c-T_d}\right] & \text{if } T_d < T < T_c \\ 0 & \text{if } T \leq T_b \text{ or } T \geq T_c \end{cases}$$

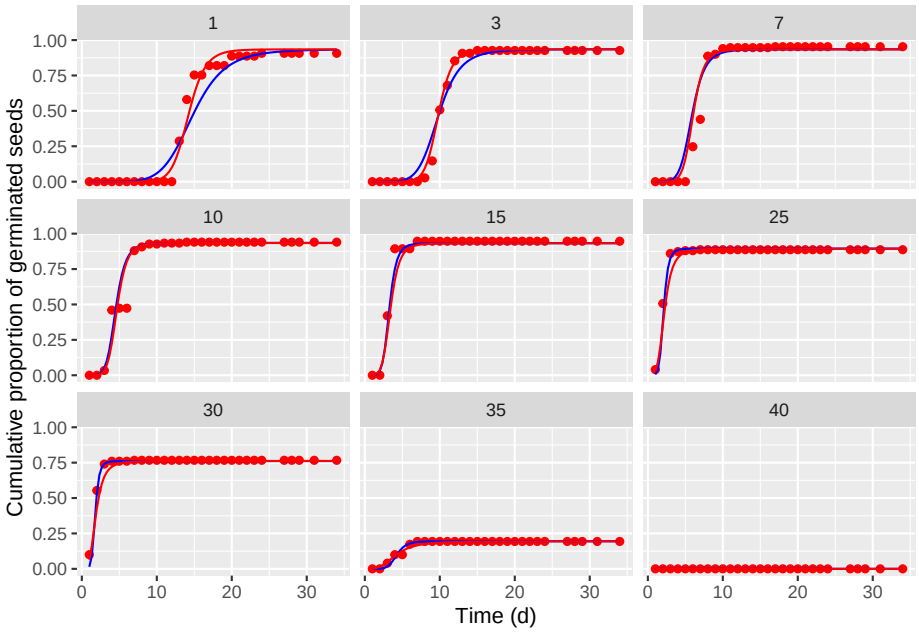
Furthermore, we also made a further improvement, by coding another model where also the b parameter was allowed to change with temperature, according to the following equation:

$$\sigma(T) = \frac{1}{b(T)} = \frac{1}{b_0} + s(T - T_b)$$

These two improved models were coded as the `TTERF()` function (with the only change in the 'e' submodel), and as the `TTERFc()` function (with a change in both the 'b' and 'e' submodels), which are available within the `drcSeedGerm` package. The code below is used to fit both models and explore the resulting fits: the symbols show the observed means, the blue line represents the 'TTERF' fit and the red line represents the 'TTERFc' fit.

```
modTTERF <- drmte( nSeeds ~ timeBef + timeAf + Temp, data=barley,
                   fct=TTERF())
```

```
modTTERFc <- drmte(nSeeds ~ timeBef + timeAf + Temp, data=barley,
                   fct=TTERFc())
AIC(modTTEM, modTTERF, modTTERFc)
##             df      AIC
## modTTEM    804 6478.917
## modTTERF   803 5730.622
## modTTERFc 802 5602.532
tab2 <- plotData(modTTERF)
tab3 <- plotData(modTTERFc)

ggplot() +
  geom_point(data = tab$plotPoints, mapping = aes(x = timeAf, y = CDF),
             col = "red") +
  geom_line(data = tab2$plotFits, mapping = aes(x = timeAf, y = CDF),
             col = "blue") +
  geom_line(data = tab3$plotFits, mapping = aes(x = timeAf, y = CDF),
             col = "red") +
  facet_wrap( ~ Temp) +
  scale_x_continuous(name = "Time (d)") +
  scale_y_continuous(name = "Cumulative proportion of germinated seeds")
```



We see that, with respect to the TTEM model, both the two improved models provide a better fit.

It should be clear that this modelling approach is rather flexible, by appropriately changing one or more submodels, and it can fit the germination pattern

of several species in several environmental conditions.

Chapter 12

References

1. Barreiro-Ures D, Francisco-Fernández M, Cao R, Fraguela BB, Doallo R, González-Andújar JL, Reyes M (2019) Analysis of interval-grouped data in weed science: The binnednp Rcpp package. Ecol Evol 9:10903–10915
2. Bradford KJ (2002) Applications of hydrothermal time to quantifying and modeling seed germination and dormancy. Weed Sci 50:248–260
3. Brown, RF, DG Mayer (1988a) Representing Cumulative Germination. 1. A Critical Analysis of Single-value Germination Indices. Annals of Botany 61:117–125
4. Brown, RF, DG Mayer (1988b) Representing Cumulative Germination.: 2. The Use of the Weibull Function and Other Empirically Derived Curves. Annals of Botany 61:127–138
5. Catara, S., Cristaudo, A., Gualtieri, A., Galesi, R., Impelluso, C., Onofri, A. (2016). Threshold temperatures for seed germination in nine species of Verbascum (Scrophulariaceae). Seed Science Research 26, 30–46.
6. Davison, A.C., Hinkley, D.V. (1997). Bootstrap methods and their application. Cambridge University Press, UK.
7. Dutang, C, Goulet V and M. Pigeon (2008). actuar: An R Package for Actuarial Science. Journal of Statistical Software, vol. 25, no. 7, 1-37.
8. Fay, MP, PA Shaw (2010) Exact and Asymptotic Weighted Logrank Tests for Interval Censored Data: The interval R Package. Journal of Statistical Software 36:1–34
9. Gresta, F, G Avola, A Onofri, U Anastasi, A Cristaudo (2011) When Does Hard Coat Impose Dormancy in Legume Seeds? Lotus and Scorpiurus Case Study. Crop Science 51:1739–1747
10. Keshtkar E, Kudsk P, Mesgaran MB (2021) Perspective: Common errors in dose–response analysis and how to avoid them. Pest Manag Sci 77:2599–2608
11. Mesgaran, M.B., Mashhadi, H.R., Alizadeh, H., Hunt, J., Young, K.R., Cousens, R.D., 2013. Importance of distribution function selection for hydrothermal time models of seed germination. Weed Research 53, 89–

101. https://doi.org/10.1111/wre.12008
12. Michael P. Fay, Pamela A. Shaw (2010). Exact and Asymptotic Weighted Logrank Tests for Interval Censored Data: The interval R Package. Journal of Statistical Software, 36(2), 1-34. URL https://www.jstatsoft.org/v36/i02/.
13. Onofri, A, F Gresta, F Tei (2010) A new method for the analysis of germination and emergence data of weed species. Weed Research 50:187–198
14. Onofri, A, MB Mesgaran, F Tei, RD Cousens (2011) The cure model: an improved way to describe seed germination? Weed Research 51:516–524
15. Onofri, A, MB Mesgaran, P Neve, RD Cousens (2014) Experimental design and parameter estimation for threshold models in seed germination. Weed Research 54:425–435
16. Onofri, A., Benincasa, P., Mesgaran, M.B., Ritz, C. (2018). Hydrothermal-time-to-event models for seed germination. European Journal of Agronomy 101, 129–139.
17. Onofri, A., Mesgaran, M., & Ritz, C. (2022). A unified framework for the analysis of germination, emergence, and other time-to-event data in weed science. Weed Science, 1-13. doi:10.1017/wsc.2022.8
18. Onofri, Andrea, Hans Peter Piepho, and Marcin Kozak (2019). Analysing Censored Data in Agricultural Research: A Review with Examples and Software Tips. Annals of Applied Biology, 174, 3-13.
19. Onofri, Andrea, Paolo Benincasa, M B Mesgaran, and Christian Ritz (2018). Hydrothermal-Time-to-Event Models for Seed Germination. European Journal of Agronomy 101: 129–39.
20. Pace, R., Benincasa, P., Ghanem, M.E., Quinet, M., Lutts, S. (2012). Germination of untreated and primed seeds in rapeseed (brassica napus var oleifera del.) under salinity and low matric potential. Experimental Agriculture 48, 238–251.
21. Ritz C, Jensen SM, Gerhard D, Streibig JC (2019). Dose-response analysis using R CRC Press. USA
22. Ritz, C., Baty, F., Streibig, J. C., Gerhard, D. (2015). Dose-Response Analysis Using R PLOS ONE, 10(12)
23. Therneau T (2021). *A Package for Survival Analysis in R*. R package version 3.2-11, <URL: https://CRAN.R-project.org/package=survival>.
24. Wickham, H, G Grolemund (2016) R for data science: import, tidy, transform, visualize, and model data. First edition. Sebastopol, CA: O'Reilly. 492 pp.
25. Yu, B., Peng, Y. (2008). Mixture cure models for multivariate survival data. Computational Statistics and Data Analysis 52, 1524–1532.
26. Zeileis, A., Köll, S., Graham, N. (2020). Various Versatile Variances: An Object-Oriented Implementation of Clustered Covariances in R. J. Stat. Soft. 95. https://doi.org/10.18637/jss.v095.i01
27. Alvarado, V., Bradford, K.J., 2002. A hydrothermal time model explains the cardinal temperatures for seed germination. Plant, Cell and Environment 25, 1061–1069.
28. Baty, F., Ritz, C., Charles, S., Brutsche, M., Flandrois, J. P., Delignette-

Muller, M.-L., 2014. A toolbox for nonlinear regression in R: the package nlstools. Journal of Statistical Software, 65, 5, 1-21.
29. Bradford, K.J., 2002. Applications of hydrothermal time to quantifying and modelling seed germination and dormancy. Weed Science 50, 248–260.
30. Catara, S., Cristaudo, A., Gualtieri, A., Galesi, R., Impelluso, C., Onofri, A., 2016. Threshold temperatures for seed germination in nine species of Verbascum (Scrophulariaceae). Seed Science Research 26, 30–46.
31. Garcia-Huidobro, J., Monteith, J.L., Squire, R., 1982. Time, temperature and germination of pearl millet (*Pennisetum typhoides* S & H.). 1. Constant temperatures. Journal of Experimental Botany 33, 288–296.
32. Kropff, M.J., van Laar, H.H. 1993. Modelling crop-weed interactions. CAB International, Books.
33. Masin, R., Onofri, A., Gasparini, V., Zanin, G., 2017. Can alternating temperatures be used to estimate base temperature for seed germination? Weed Research 57, 390–398.
34. Onofri, A., Benincasa, P., Mesgaran, M.B., Ritz, C., 2018. Hydrothermal-time-to-event models for seed germination. European Journal of Agronomy 101, 129–139.
35. Ritz, C., Jensen, S. M., Gerhard, D., Streibig, J. C., 2019. Dose-Response Analysis Using R. CRC Press
36. Rowse, H.R., Finch-Savage, W.E., 2003. Hydrothermal threshold models can describe the germination response of carrot (Daucus carota) and onion (Allium cepa) seed populations across both sub- and supra-optimal temperatures. New Phytologist 158, 101–108.
37. Zeileis, A., 2006. Object-oriented computation of sandwich estimators. Journal of Statistical Software, 16, 9, 1-16.