Recurrent Event Analysis with `reda` and `reReg`

第三屆統計科學與大數據國際論壇

Sy Han (Steven) Chiou
邱思翰

Department of Mathematical Sciences,
The University of Texas at Dallas
美國德州大學達拉斯分校

THE UNIVERSITY OF TEXAS AT DALLAS
School of Natural Sciences and Mathematics

Table of contents

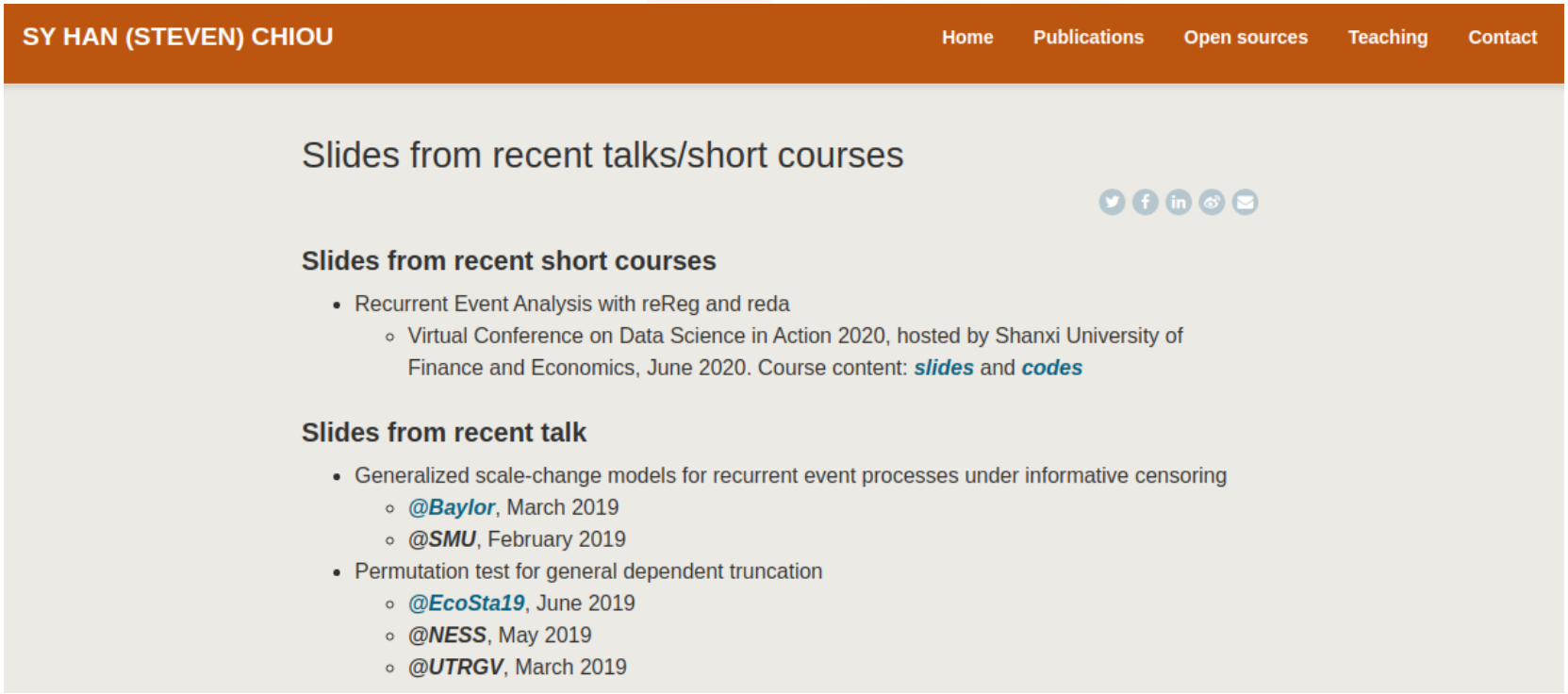
Outline

Outline

- The course provides a survey of modern statistical methodology for analysis of recurrent event data.
- A key feature of this course is the integration of the `R` statistical software:
  - Demonstrate the use of `R` packages `reda` and `reReg`
  - Illustrate how survival quantities are computed
  - Interpret the results
  - Many examples

Outline

- Slides and `R` codes can be downloaded from `https://www.sychiou.com/software/talk/`



SY HAN (STEVEN) CHIOU

Home | Publications | Open sources | Teaching | Contact

Slides from recent talks/short courses

Slides from recent short courses

- Recurrent Event Analysis with reReg and reda
  - Virtual Conference on Data Science in Action 2020, hosted by Shanxi University of Finance and Economics, June 2020. Course content: ***slides*** and ***codes***

Slides from recent talk

- Generalized scale-change models for recurrent event processes under informative censoring
  - ***@Baylor***, March 2019
  - ***@SMU***, February 2019
- Permutation test for general dependent truncation
  - ***@EcoSta19***, June 2019
  - ***@NESS***, May 2019
  - ***@UTRGV***, March 2019

Outline

- The course is divided into four parts:
  1. Introduction to recurrent event data, notation and basic quantities
  2. Exploring recurrent event data with event plots and cumulative sample mean functions with `reda` and `reReg`
  3. Simulating recurrent event data with `reda` and `reReg`
  4. Fitting regression models with `reReg`

Outline

- After taking the course, students will be able to
  - Understand the features of recurrent event data and their implications in drawing inference
  - Use proper functions in `reda` and `reReg` to solve real-world problems
  - Simulate recurrent data under different assumptions
  - Interpret and present the analytic results in a clear and coherent way to answer substantive questions

Introduction

Univariate event time

- Survival analysis is the study of survival times and of the factors that influence them.
- Subjects are often followed from a well-defined starting point until the event of interest occurs or the study ends, whichever occurs first.
- Examples of starting point and outcome events are:

  Starting points: study entry time, date of birth, treatment randomization.
  Outcome events: hospitalization, tumor occurrences, or death.

Recurrent event

- Non-fatal outcome may recur multiple times over the course of the study.
- The recurrent event times are recorded until a censoring point.
- The simplest way to analyze a recurrent event data is to focus on time to the first occurrence, reducing the problem to that of a univariate event time.
  - Throw out a big chunk of data and is inefficient (e.g. Claggett et al., 2018)
- Special techniques are needed to fully capitalize on the recurrent-event information

Example: Chronic Granulomatous Disease (CGD) Study

- Below we give a brief overview of the types of data we will be dealing with in this course, illustrated with real-world examples.
- The International Chronic Granulomatous Disease (CGD) data is public available from the `survival` package (Therneau, 2020).

```
> data(cgd, package = "survival")
> dim(cgd)
[1] 203  16
> head(subset(cgd,
+       select = c(id, tstart, tstop, status, enum, sex, age, random, treat)))
  id tstart tstop status enum    sex age     random   treat
1  1      0   219      1    1 female  12 1989-06-07  rIFN-g
2  1    219   373      1    2 female  12 1989-06-07  rIFN-g
3  1    373   414      0    3 female  12 1989-06-07  rIFN-g
4  2      0     8      1    1   male  15 1989-06-07 placebo
5  2      8    26      1    2   male  15 1989-06-07 placebo
6  2     26   152      1    3   male  15 1989-06-07 placebo
```

- The outcome of interest is repeated CGD infections

Example: Chronic Granulomatous Disease (CGD) Study

- The dataset contains the time to serious infections observed through the end of study for 128 CGD patients.
- The full description of the data is available from the documentation page:

```
> ?survival::cgd
```

- The important variables are:

id	subject identification number
tstart	start of each time interval
tstop	end of each time interval
status	1 = the interval ends with an infection
enum	observation number within subject
sex	gender
age	age (in years) at study entry
random	date of randomization
treat	placebo or gamma interferon

Example: Chronic Granulomatous Disease (CGD) Study

- The aim is to assess the effect of gamma interferon (`treat`) on incidence of repeated CGD infections.
- Patients were followed from the randomization to the end of study or dropout (no death).
- The median length of follow-up time is 293 days:

```
> summary(tapply(cgd$tstop, cgd$id, max))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   91.0   264.8   293.0   292.8   343.0   439.0
```

- Number of infections per patients ranges from 0 to 7;

```
> table(tapply(cgd$status, cgd$id, sum))

 0  1  2  3  4  5  7
84 27  9  5  1  1  1
```

- Correlation of recurrent events within patients

Example: Rehospitalization data

- Another example is the rehospitalization data (González et al., 2005) from the `frailtypack` package (Rondeau et al., 2019)

```
> data(readmission, package = "frailtypack")
> dim(readmission)
[1] 861  11
> head(readmission)
  id enum t.start t.stop time event      chemo    sex dukes charlson death
1  1    1       0     24   24     1    Treated Female     D        3     0
2  1    2      24    457  433     1    Treated Female     D        0     0
3  1    3     457   1037  580     0    Treated Female     D        0     0
4  2    1       0    489  489     1 NonTreated   Male     C        0     0
5  2    2     489   1182  693     0 NonTreated   Male     C        0     0
6  3    1       0     15   15     1 NonTreated   Male     C        3     0
```

- The `frailtypack` package can be installed with following:

```
> install.package("frailtypack")
```

- The outcome of interest is repeated rehospitalization.

Example: Rehospitalization data

- The dataset contains rehospitalization times after surgery in 403 patients diagnosed with colorectal cancer
- The important variables are

id	subject identification number
t.start	start of each time interval
t.stop	end of each time interval
event	1 = the interval ends with a hospitalization
enum	observation number within subject
time	interocurrence (`t.stop - t.start`)
death	1 = died at the end of the interval (0 = alive)
chemo	1 = the patient did not receive chemotherapy (2 = received)
sex	1 = male and 2 = female

Example: Rehospitalization data

- The aim is to investigate gender-based inequalities in hospital readmission among patients diagnosed with colorectal cancer.
- Patients were followed from the date of surgery to the end of study or death.
- The median length of follow-up time is 1128 days.

```
> with(readmission, summary(tapply(t.stop, id, max)))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1     524    1128    1026    1460    2176
```

- Number of rehospitalization per patients ranges from 0 to 22;

```
> with(readmission, table(tapply(event, id, sum)))

  0   1   2   3   4   5   6   8   9  10  11  16  22
199 105  45  21  15   8   4   1   1   1   1   1   1
```

- Correlation of recurrent events within patients

Notations: Recurrent process

- For simplicity, we assume a single recurrent event process starts at $t = 0$
- Let $T_1, T_2, \ldots,$ denote the recurrent event times, where T_k is the time of the kth event
- The associated counting process $\{N^*(t), t \geq 0\}$ records the cumulative number of events generated by the process, or

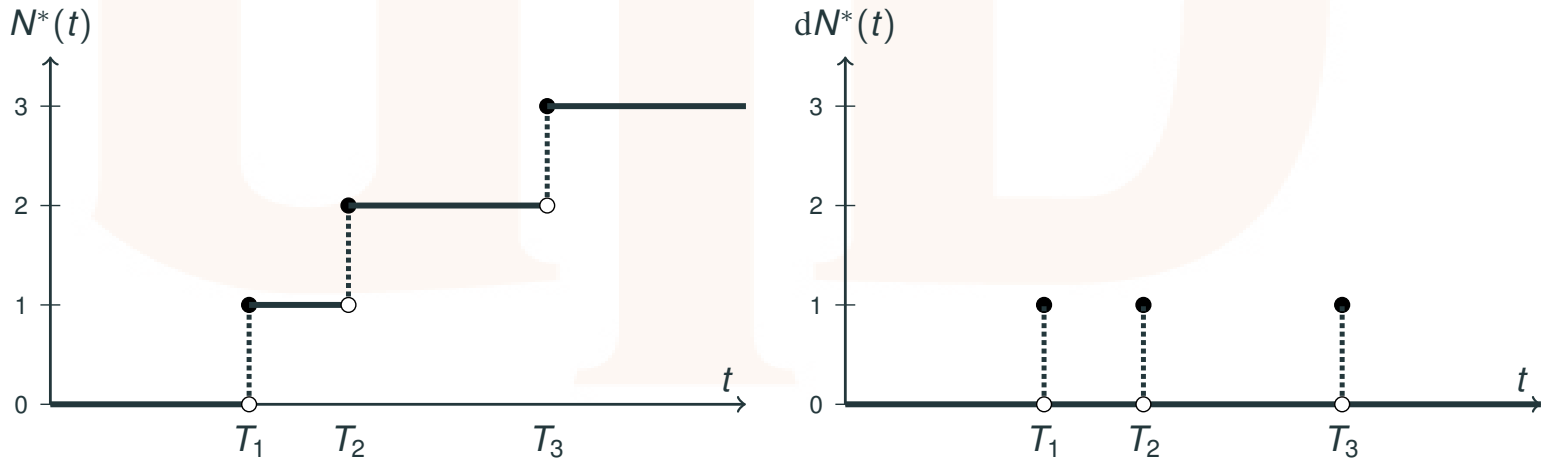
$$N^*(t) = \sum_{k=1}^{\infty} I(T_k \leq t)$$

- $N^*(t)$ also represents the number of events occurring over the interval $[0, t]$.
- More generally, the number of events occurring over the interval $(s, t]$ is

$$N^*(s, t) = N^*(t) - N^*(s).$$

Notations: Recurrent process

- The counting processes are right-continuous; $N^*(t) = N^*(t^+)$.
- Models for recurrent events can be specified by considering the probability distribution for the number of events in short intervals $[t, t + \mathrm{d}t)$
- Define $\mathrm{d}N^*(t) = N^*(t + \mathrm{d}t) - N^*(t)$ be the number of events in $[t, t + \mathrm{d}t)$.
- The following portrays a realization of an event process in terms of its counting process



Notations: Intensity function

- There are two fundamental ways to describe recurrent event process:
  - intensity function (conditional)
  - mean function (marginal)
- The intensity function gives the instantaneous rate of an event occurring at t, conditional on the process history.
- The intensity is defined formally as

$$\lambda\{t|H(t)\} = \lim_{\mathrm{d}t \to 0} \frac{P\{\mathrm{d}N^*(t) = 1|H(t)\}}{\mathrm{d}t}, \tag{1}$$

where $H(t)$ is the process history.

Notations: Intensity function

- We will make a convenient assumption that events occurring in continuous time and two events cannot occur simultaneously.
- Since $\mathrm{d}N^*(t)$ is either 0 or 1, the relationship in (1) is sometimes expressed as

$$\lambda\{t|H(t)\}\mathrm{d}t := \mathrm{E}\{\mathrm{d}N^*(t)|H(t)\}.$$

- Thus, the intensity function can be thought of as an extension of the univariate hazard function to the recurrent event setting.

Notations: Intensity function

- A special case where the incidence of a new event is independent of previous events, then the intensity function is reduced to

$$\lambda\{t|H(t)\} = \lim_{\mathrm{d}t \to 0} \frac{P\{\mathrm{d}N^*(t) = 1\}}{\mathrm{d}t} = r(t).$$

- This implies the incidence of a new event is independent of previous events (independent increments)
- Homogeneous Poisson process gives $r(t) \equiv r$
- The assumption is too strong for medical studies

Notations: Mean function

- The other useful quantity is the mean function, defined as the marginal average frequency of the recurrent event process, or

$$\mu(t) = \mathrm{E}\{N^*(t)\}.$$

- This has a close associated with the rate function

$$r(t)\mathrm{d}t := \mathrm{d}\mu(t) = \mathrm{E}\{\mathrm{d}N^*(t)\}.$$

- Rate function is averaged version of intensity function and is by definition non-random
- Rate function is easier to interpret than intensity
- The relationship between intensity and mean functions is

$$\mu(t) = \int_0^t \mathrm{E}[\lambda\{t|H(t)\}]\mathrm{d}t.$$

Notations: Mean function

- To many applied researchers, the mean function as the average cumulative frequency is more intuitive and easier to interpret than the intensity function.
- The mean function alone does not completely determine the likelihood function of the recurrent event process.

Notations: Observed data

- In practice, the recurrent event processes are subject to censoring.
- Let C denote the censoring time, independent of $N^*(\cdot)$ given covariates.
- The observed part of the outcome data is the recurrent event process curtailed by C;

$$N(t) := N^*\{\min(t, C)\}$$

- When there is no terminal event (death), C is always observable.
- Thus, for a random sample of size n, the observed data consist of

$$\{N_i(\cdot), C_i, Z_i\}, i = 1, \ldots, n,$$

where Z_i is the covariate vector.

Notations: Observed data

- When there is a terminal event in addition to the censoring time C.
- Denote the terminal event Y_i^*, the observed data is

$$\{N_i(\cdot), Y_i, Z_i, \Delta_i\}, i = 1, \ldots, n,$$

  where $N_i(t) := N^*\{\min(t, Y)\}$, $Y = \min(C, Y^*)$, and $\Delta = I(Y^* \leq C)$.
- It is important to keep in mind that our interest lies in the distribution of the latent counting process $N^*(t)$.

Notations: Sample mean function

- For a homogeneous sample without covariates, there is a simple estimator for the mean function of $N^*(t)$ called the cumulative sample mean (CSM) function defined as

$$\widehat{\mu}(t) = \frac{1}{n}\sum_{i=1}^{n} N_i(t). \tag{2}$$

- In the presence of censoring, the CSM function is a Nelsen-Aalen-type estimator given by

$$\widehat{\mu}(t) = \int_0^t \frac{\sum_{i=1}^{n} \mathrm{d}N_i(u)}{\sum_{i=1}^{n} I(C_i \geq u)}. \tag{3}$$

- As $n \to \infty$, we have

$$\widehat{\mu}(t) \to \int_0^t \frac{\mathrm{E}\{\mathrm{d}N(u)\}}{P(C \geq u)} = \int_0^t \frac{P(C \geq u)\mathrm{E}[\mathrm{d}N^*\{(u)\}]}{P(C \geq u)} = \mu(t)$$

Recurrent event objects;

`Recur()`

Recurrent event objects

- The `reReg` package offers simple ways to create plots allowing users to understand recurrent data at glance
- To install the latest version of the `reReg` package is by

```
> ## install.packages("devtools")
> devtools::install_github("stc04003/reReg")
```

- Once installed, load it with `library()`:

```
> library(reReg)
> packageVersion("reReg")
[1] '1.2.1'
```

- Note that, as of June 13, 2020, the version on my GitHub repository is one version ahead of that on CRAN.

Recurrent event objects

- To cite `reReg`, use

```
> citation("reReg")

To cite package 'reReg' in publications use:

  Sy Han (Steven) Chiou and Chiung-Yu Huang (2020). reReg: Recurrent Event
  Regression. R package version 1.2.1. http://github.com/stc04003/reReg

A BibTeX entry for LaTeX users is

  @Manual{,
    title = {reReg: Recurrent Event Regression},
    author = {Sy Han (Steven) Chiou and Chiung-Yu { Huang}},
    year = {2020},
    note = {R package version 1.2.1},
    url = {http://github.com/stc04003/reReg},
  }
```

- The citation will be updated once the next version is submitted to CRAN (summer 2020).

Recurrent event objects

- Depending on whether there is a terminal event, the observed data is

$$\{N_i(\cdot), C_i, Z_i\} \text{ or } \{N_i(\cdot), Y_i, Z_i, \Delta_i\}.$$

- From the example data, variables like `id`, `tstart`, `tstop`, `status`, and `death`, are needed to specify the recurrent process.
- Just like the `Surv()` function in the `Survival` package, to use functions in `reReg` and `reda` , we must create a formula response to represent the recurrent object.

Recurrent event objects with `Recur()`

- The `Recur()`() is used to create such recurrent object
- The arguments of `Recur()` is

```
> args(Recur)
function (time, id, event, terminal, origin, check = c("hard",
    "soft", "none"), ...)
NULL
```

- The `Recur()` function is a successor function of the deprecated functions `Survr()` from `reda` and `reSurv()` from `reReg`.

Recurrent event objects with `Recur()`

- A short description of the six arguments are:

`time`	specifies the time of recurrent event or censoring
`id`	specifies the subject identification
`event`	specifies the status or types of the recurrent events
`terminal`	specifies the status or types of the terminal events
`origin`	specifies the time origin of each subject
`check`	specifies how to perform checks

- The `Recur()` function is very flexible and not all these six arguments are required.
- When a recurrent event object is created without specifying all six arguments, the above arguments have different roles.

Recurrent event objects with `Recur()`

- We will first explore `Recur()` with a small example

```
> (dat0 <- subset(readmission, subset = id <= 4,
+                 select = c(id, t.start, t.stop, event, death)))
   id t.start t.stop event death
1   1       0     24     1     0
2   1      24    457     1     0
3   1     457   1037     0     0
4   2       0    489     1     0
5   2     489   1182     0     0
6   3       0     15     1     0
7   3      15    783     0     1
8   4       0    163     1     0
9   4     163    288     1     0
10  4     288    638     1     0
11  4     638    686     1     0
12  4     686   2048     0     0
```

Recurrent event objects with `Recur()`

- A recurrent event object can be created with:

```
> with(dat0, Recur(time = t.stop, id = id, event = event, terminal = death))
[1] 1: (0, 24], (24, 457], (457, 1037+]
[2] 2: (0, 489], (489, 1182+]
[3] 3: (0, 15], (15, 783*]
[4] 4: (0, 163], (163, 288], ..., (686, 2048+]
```

- From the outputted list we can see that subject 1 experienced
  - recurrent events (rehospitalization) 24 days and 457 days after surgery
  - a censored event (end of follow-up, with no death) 1037 days after surgery
  - the censored event is indicated by `+`
- Similarly, subject 3 experienced
  - a recurrent event 15 days after surgery
  - a terminal event (death) 783 days after surgery
  - the terminal event is indicated by `*`
- In `readmission`, all subjects start with `time = 0`, so the argument `origin` does not need to be specified.

`Recur()`, the `time` argument

- The `time` argument is a numerical vector representing the
  - time of recurrent event, or censoring, or
  - a list of time intervals created by `time1 %2to% time2`

```
> with(dat0, Recur(time = t.start %2% t.stop, id = id,
+                  event = event, terminal = death))
[1] 1: (0, 24], (24, 457], (457, 1037+]
[2] 2: (0, 489], (489, 1182+]
[3] 3: (0, 15], (15, 783*]
[4] 4: (0, 163], (163, 288], ..., (686, 2048+]
```

- When specifying an interval, `Date` and `difftime` are allowed and converted to numeric values.

`Recur()`, the `time` argument

- The `time1 %to% time2` returns a list of two elements
- `%2%` is an alias of `%to%`

```
> str(dat0$t.start %to% dat0$t.stop)
List of 2
 $ time1: int [1:12] 0 24 457 0 489 0 15 0 163 288 ...
 $ time2: int [1:12] 24 457 1037 489 1182 15 783 163 288 638 ...
> str(dat0$t.start %2% dat0$t.stop)
List of 2
 $ time1: int [1:12] 0 24 457 0 489 0 15 0 163 288 ...
 $ time2: int [1:12] 24 457 1037 489 1182 15 783 163 288 638 ...
```

- This function is convenient when subjects start with different origins.

`Recur()`, the `id` argument

- The `id` argument is a subject identificator
- It can be numeric vector, character vector, or a factor vector
- If it is left unspecified, `Recur()` will assume that each row represents a subject
- If each row is assumed to be a subject, then `Recur()` becomes like `Surv()`

`Recur()`, the `event` argument

- The `event` argument is a numeric vector that may represent the status or types of recurrent events.
- A logical vector is allowed and converted to a numeric vector.
- Non-positive values are internally converted to zero indicating censoring status (values ≥ 0 indicate recurrent events)
- If `event` is not specified, `Recur()` assumes times before the last event times are recurrent events

```
> with(dat0, Recur(time = t.stop, id = id, terminal = death))
[1] 1: (0, 24], (24, 457], (457, 1037+]
[2] 2: (0, 489], (489, 1182+]
[3] 3: (0, 15], (15, 783*]
[4] 4: (0, 163], (163, 288], ..., (686, 2048+]
```

`Recur()`, the `terminal` argument

- The `terminal` argument is a numeric vector that may represent the status or types of the terminal events
- A logical vector is allowed and converted to a numeric vector.
- Non-positive values are internally converted to zero indicating censoring status (values ≥ 0 indicate terminal events)
- If a scalar value is specified, all subjects will have the same status of terminal events at their last recurrent episodes.
- The length of the specified `terminal` should be equal to the number of subjects, or number of data rows.
- In the latter case, each subject may have at most one positive entry of `terminal` at the last recurrent episode.

`Recur()`, the `terminal` argument

- When `terminal` is omitted or set `terminal = 0`, `Recur()` assumes the last event times are censoring times.
- This is equivalent to assume there is no terminal times (e.g., `cgd`)

```
> with(dat0, Recur(time = t.stop, id = id, event = event))
[1] 1: (0, 24], (24, 457], (457, 1037+]
[2] 2: (0, 489], (489, 1182+]
[3] 3: (0, 15], (15, 783+]
[4] 4: (0, 163], (163, 288], ..., (686, 2048+]
> with(dat0, Recur(time = t.stop, id = id, event = event, terminal = 0))
[1] 1: (0, 24], (24, 457], (457, 1037+]
[2] 2: (0, 489], (489, 1182+]
[3] 3: (0, 15], (15, 783+]
[4] 4: (0, 163], (163, 288], ..., (686, 2048+]
```

- Similarly, if all recurrent events are terminated by a terminal events, one can set `terminal = 1`:

```
> with(dat0, Recur(time = t.stop, id = id, event = event, terminal = 1))
[1] 1: (0, 24], (24, 457], (457, 1037*]
[2] 2: (0, 489], (489, 1182*]
[3] 3: (0, 15], (15, 783*]
[4] 4: (0, 163], (163, 288], ..., (686, 2048*]
```

Recur(), the origin argument

- The origin argument specifies the time origin of each subject.
- If a scaler value is specified, all subjects will have the same origin at the specified value.

```
> with(dat0, Recur(time = t.stop, id = id, event = event,
+                  terminal = death, origin = 0))
[1] 1: (0, 24], (24, 457], (457, 1037+]
[2] 2: (0, 489], (489, 1182+]
[3] 3: (0, 15], (15, 783*]
[4] 4: (0, 163], (163, 288], ..., (686, 2048+]
> with(dat0, Recur(time = t.stop, id = id, event = event,
+                  terminal = death, origin = 10))
[1] 1: (10, 24], (24, 457], (457, 1037+]
[2] 2: (10, 489], (489, 1182+]
[3] 3: (10, 15], (15, 783*]
[4] 4: (10, 163], (163, 288], ..., (686, 2048+]
```

`Recur()`, calling without argument names

- As in many `R` functions, when calling `Recur()`, the users can specify the arguments by complete name (as before), or by position.
- Arguments are matched by the position.

```
> with(dat0, Recur(t.stop))
 [1] 1: (0, 24+]    2: (0, 457+]   3: (0, 1037+]  4: (0, 489+]   5: (0, 1182+]
 [6] 6: (0, 15+]    7: (0, 783+]   8: (0, 163+]   9: (0, 288+]   10: (0, 638+]
[11] 11: (0, 686+]  12: (0, 2048+]
> with(dat0, Recur(t.stop, id))
[1] 1: (0, 24], (24, 457], (457, 1037+]
[2] 2: (0, 489], (489, 1182+]
[3] 3: (0, 15], (15, 783+]
[4] 4: (0, 163], (163, 288], ..., (686, 2048+]
```

`Recur()`, calling without argument names

- More examples:

```
> with(dat0, Recur(t.stop, id, event))
[1] 1: (0, 24], (24, 457], (457, 1037+]
[2] 2: (0, 489], (489, 1182+]
[3] 3: (0, 15], (15, 783+]
[4] 4: (0, 163], (163, 288], ..., (686, 2048+]
> with(dat0, Recur(t.stop, id, event, death))
[1] 1: (0, 24], (24, 457], (457, 1037+]
[2] 2: (0, 489], (489, 1182+]
[3] 3: (0, 15], (15, 783*]
[4] 4: (0, 163], (163, 288], ..., (686, 2048+]
```

- The above functions calls were execute without errors though returns different results.

`Recur()`, the `check` argument

- The `check` argument is a character string to specify the checking rule
- There are three possible values can be specified
  - `hard` An error will be thrown and a recurrent event object will not be returned
  - `soft` A warning will be thrown and a recurrent event object will be returned
  - `none` no data checking procedure will be ran
- The checking rule include
  - Subject identification (`id`), event times, censoring time (`time`), and event indicator (`event`) cannot contain missing values.
  - There has to be only one censoring time $\leq$ recurrent event time.
  - The time origin has to be the same and not later than any event time.

`Recur()`, the `check` argument

- In the following example, an option is used to limit the maximum number of `Recur()` object to be printed

```
> options(reda.Recur.maxPrint = 3 )
> with(cgd, Recur(tstop, id, status, check = "hard"))
Error:  Subjects censored before events:  87.
> head(with(cgd, Recur(tstop, id, status, check = "soft")))
Warning:  Subjects censored before events:  87.
     time1 time2 id event terminal origin
[1,]     0   219  1     1        0      0
[2,]   219   373  1     1        0      0
[3,]   373   414  1     0        0      0
[4,]     0     8  2     1        0      0
[5,]     8    26  2     1        0      0
[6,]    26   152  2     1        0      0
> head(with(cgd, Recur(tstop, id, status, check = "none")))
     time1 time2 id event terminal origin
[1,]     0   219  1     1        0      0
[2,]   219   373  1     1        0      0
[3,]   373   414  1     0        0      0
[4,]     0     8  2     1        0      0
[5,]     8    26  2     1        0      0
[6,]    26   152  2     1        0      0
```

`Recur()`, the `check` argument

- The message indicates the 87th subject does not have a censoring time.

```
> subset(cgd, id == 87)
    id center     random   treat  sex age height weight  inherit steroids propyla
149 87    NIH 1989-11-03 placebo male  19    170   71.2 X-linked        0
150 87    NIH 1989-11-03 placebo male  19    170   71.2 X-linked        0
    hos.cat tstart enum tstop status
149  US:NIH      0    1    99      1
150  US:NIH     99    2   306      1
> with(subset(cgd, id == 87), Recur(tstop, id, status, check = "soft"))
Warning:  Subjects censored before events:  87.
[1] 87: (0, 99], (99, 306+]
```

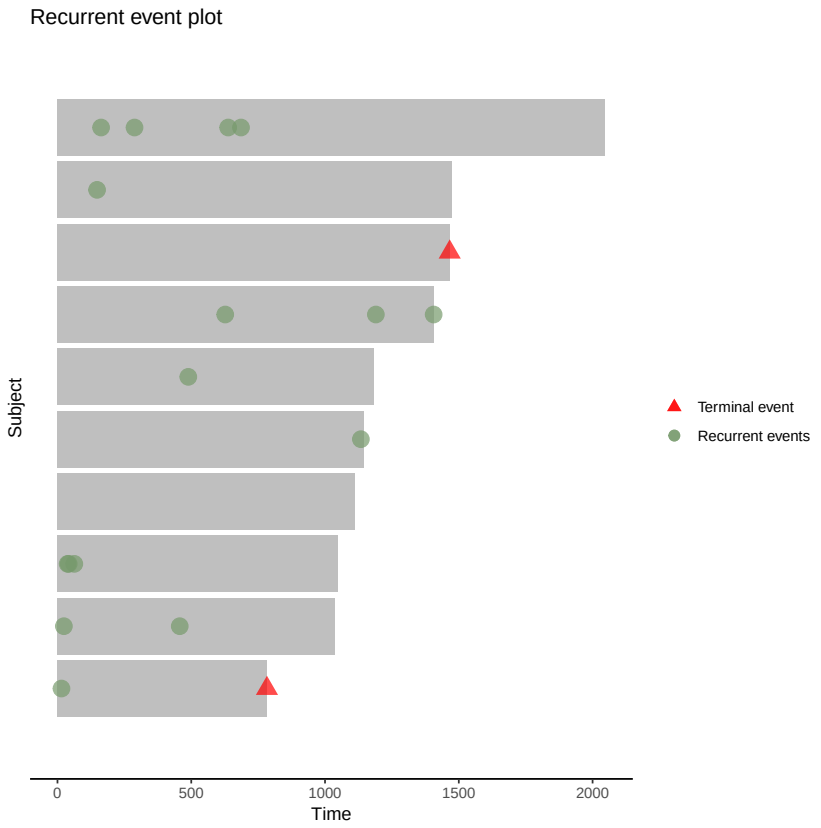
- When `check = "soft"` or `check = "none"` the `Recur()` function "guessed" the last row (time 306) is a censoring time.

Visualization; `plotEvents()` and `plotCSM()`

Event plots: plotting `Recur()`

- An event plot is a quick and easy way to glance at recurrent event data.
- The easiest way to create a event plot is by plotting the `Recur()` object with `R` 's generic function `plot()` when `reReg` is loaded.
- We will start with small examples,

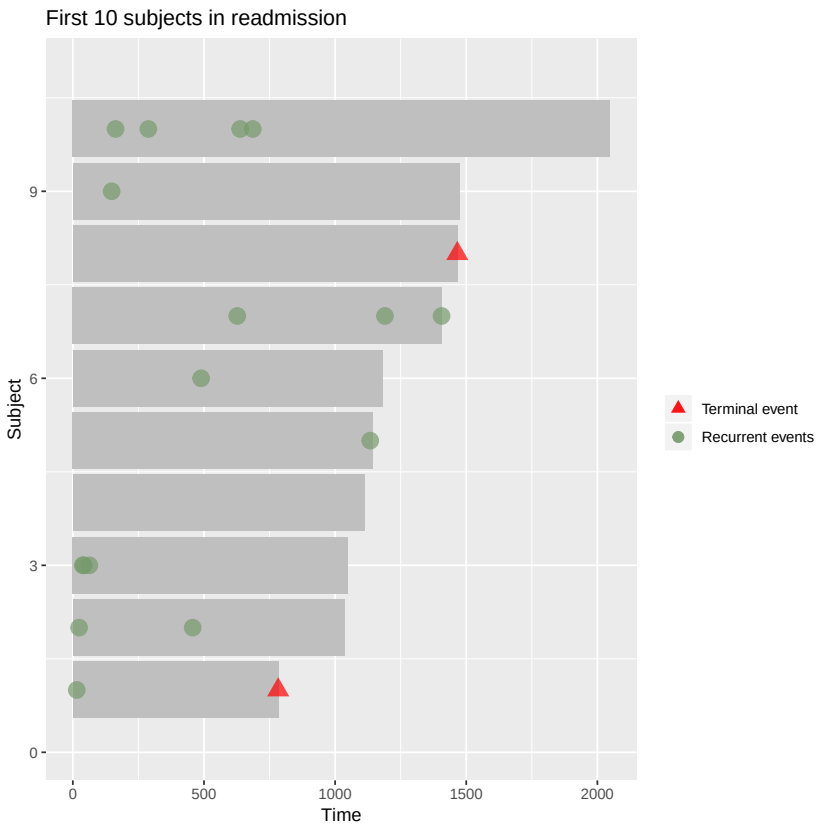
```
> reObj <- with(subset(readmission, subset = id <= 10),
+               Recur(t.stop, id, event, death))
> plot(reObj)
```



Event plots: extension with `ggplot2`

- The event plot is in a `ggplot` object and is capable with `ggplot` layers.
- For example

```
> library(ggplot2)
> plot(reObj) + ggtitle("First 10 subjects in readmission") + theme_gray()
```



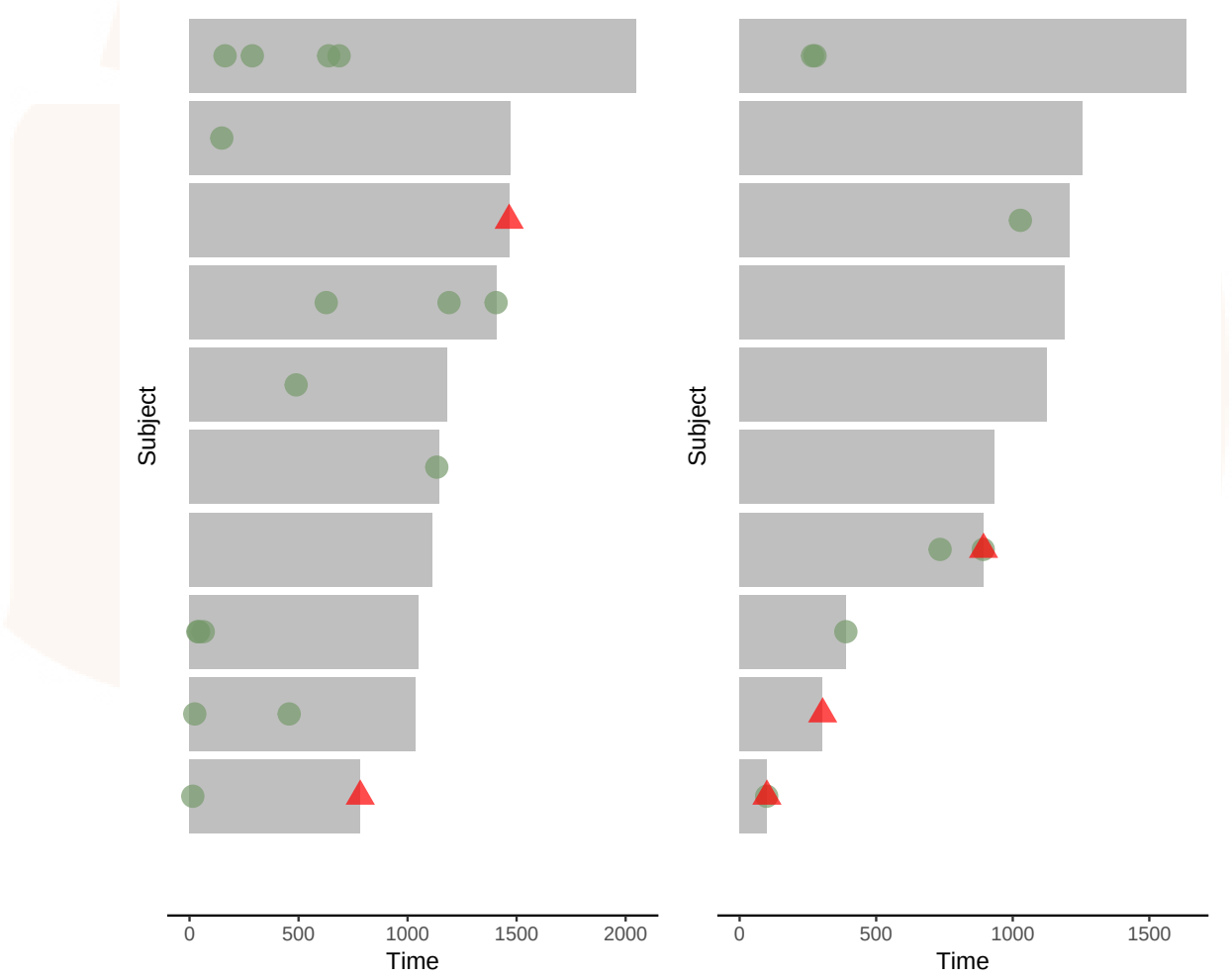
Event plots: extension with `gridExtra`

- Extensions to `ggplot2` can also be applied:

```
> reObj2 <- with(subset(readmission, subset = id %in% 11:20),
+                Recur(t.stop, id, event, death))
> library(gridExtra)
> grid.arrange(plot(reObj) + ggtitle("") + theme(legend.position = "none"),
+              plot(reObj2) + ggtitle("") + theme(legend.position = "none"),
+              ncol = 2)
```

Event plots: extension with `gridExtra`

- The `grid.arrange()` gives



Event plots: more options

- A list of common graphical options can be passed down to `plot()` as arguments when plotting a `Recur()` object.
- Some of these arguments are:

xlab customizable x-label, default value is “Time”

ylab customizable y-label, default value is “Subject”

main customizable title, default value is “Recurrent event plot”

cex size of the points

alpha is between 0 and 1, used to control the transparency of points

terminal.name customizable label for terminal event, default value is “Terminal event”

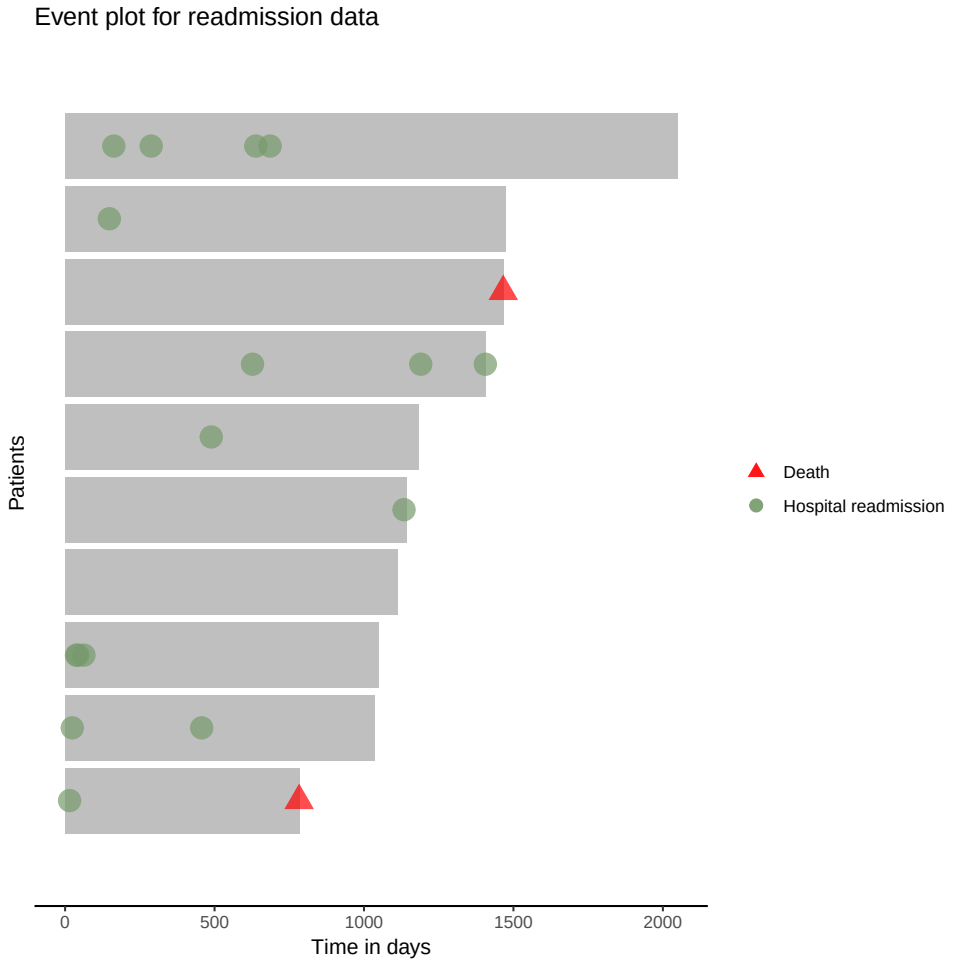
Recurrent.name customizable legend title for recurrent event, default value is “Recurrent events”

Recurrent.type customizable label for recurrent event type, default value is `NULL`

Event plots: more options

- Here is a modified event plot:

```
> plot(reObj, cex = 5, xlab = "Time in days", ylab = "Patients",
+      main = "Event plot for readmission data",
+      terminal.name = "Death", recurrent.name = "Hospital readmission")
```



Event plots: more options

- We will see how the event plot looks for the `readmission` data

```
> plot(with(readmission, Recur(t.stop, id, event, death)))
Error:  Subjects having multiple terminal events:  60, 109, 280.
> subset(readmission, id %in% c(60, 109, 280))
     id enum t.start t.stop time event      chemo  sex dukes charlson death
119  60    1       0    799  799     1 NonTreated Male     C        0     1
120  60    2     799    800    1     0 NonTreated Male     C        0     1
204 109    1       0    226  226     1 NonTreated Male     D        3     1
205 109    2     226    227    1     0 NonTreated Male     D        3     1
579 280    2       0      3    3     1 NonTreated Male   A-B        0     0
580 280    3       3    166  163     1 NonTreated Male   A-B        0     0
581 280    4     166    383  217     1 NonTreated Male   A-B        0     1
582 280    5     383    390    7     1 NonTreated Male   A-B        0     0
583 280    6     390    391    1     0 NonTreated Male   A-B        0     1
```

- The `Recur()` finds errors in subjects # 60, # 109, and # 280.
- We will create a new `readmission` without these subjects:

```
> readmission0 <- subset(readmission, !(id %in% c(60, 109, 280)))
```

Event plots: more options

- Removing the problematic subjects, we have

```
> reObj <- with(readmission0, Recur(t.stop, id, event, death))
> plot(reObj, xlab = "Time in days", ylab = "Patients",
+           main = "Event plot for readmission data",
+           terminal.name = "Death", recurrent.name = "Hospital readmission")
```

Event plots: `plotEvents()`

- The `plotEvents()` function is a more specialized function for event plots.

```
> args(plotEvents)
function (formula, data, result = c("increasing", "decreasing",
    "asis"), control = list(), ...)
NULL
```

`formula` is a formula object, with the response on the left of a "$\sim$" operator, and the predictors on the right. The response must be a recurrent event survival object as returned by function `Recur()`.

`data` is an optional data frame in which to interpret the variables occurring in the `formula`

Event plots: `plotEvents()`

- More arguments follow:

  `result` is an optional character string specifying whether the event plot is sorted by the subjects' terminal time. The available options are

  - `"increasing"` sort the terminal time from in increasing order (default). This places longer terminal times on top.
  - `"decreasing"` sort the terminal time from in decreasing order (default). This places shorter terminal times on top.
  - `"asis"` present the event plot without sorting.

  `control` a list of control parameters (graphical parameters that can be passed to `plotEvents()` without including in the control list)

Event plots: `plotEvents()`

- The `formula` argument allows user to include covariates to be used to stratify event plots.
- When there is no covariates, `plotEvents()` reduces to `plot()`.
- The following gives the same event plot.

```
> plot(reObj)
> plotEvents(reObj)
> plotEvents(reObj ~ 1)
> plotEvents(reObj ~ 1, data = readmission0)
```

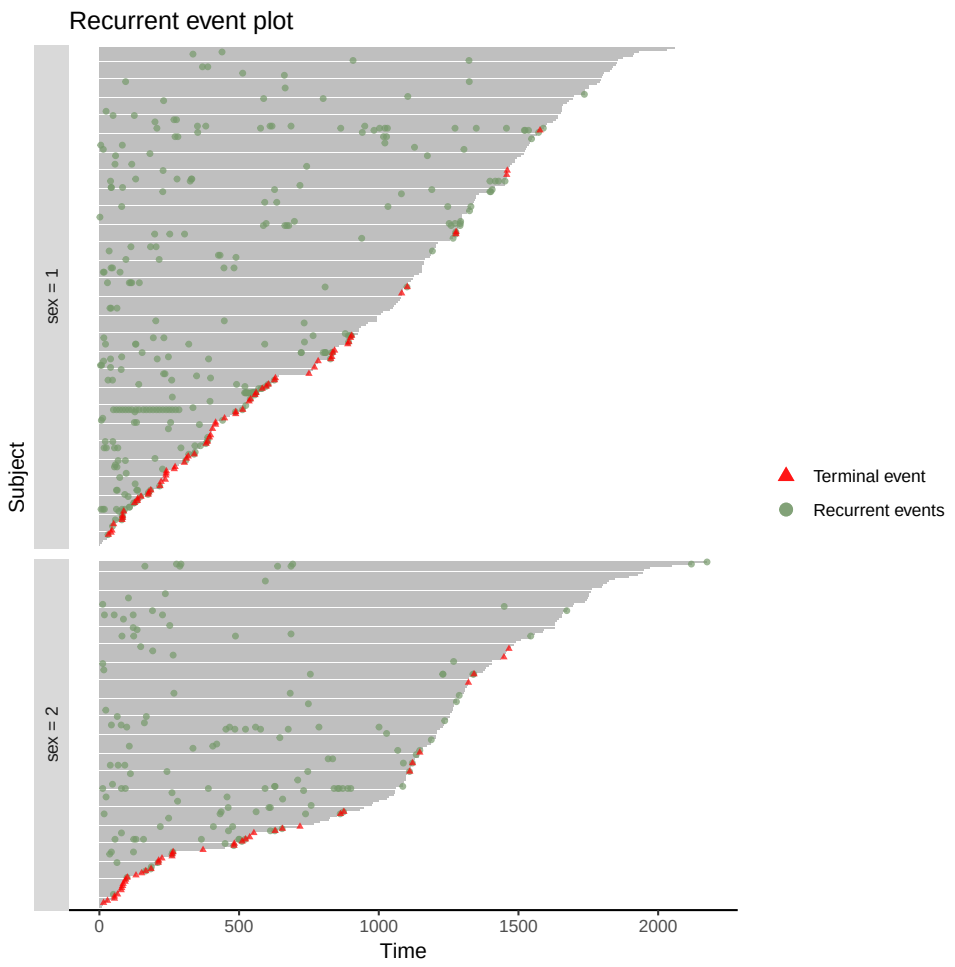
- The `Recur()` object can be created in `plotEvents()`.
- The event plot above can also be created without defining `reObj` by calling

```
> plotEvents(Recur(t.stop, id, event, death) ~ 1, data = readmission0)
```

Event plots: `plotEvents()`

- Suppose we want to stratify the event plot by `sex`, we can call

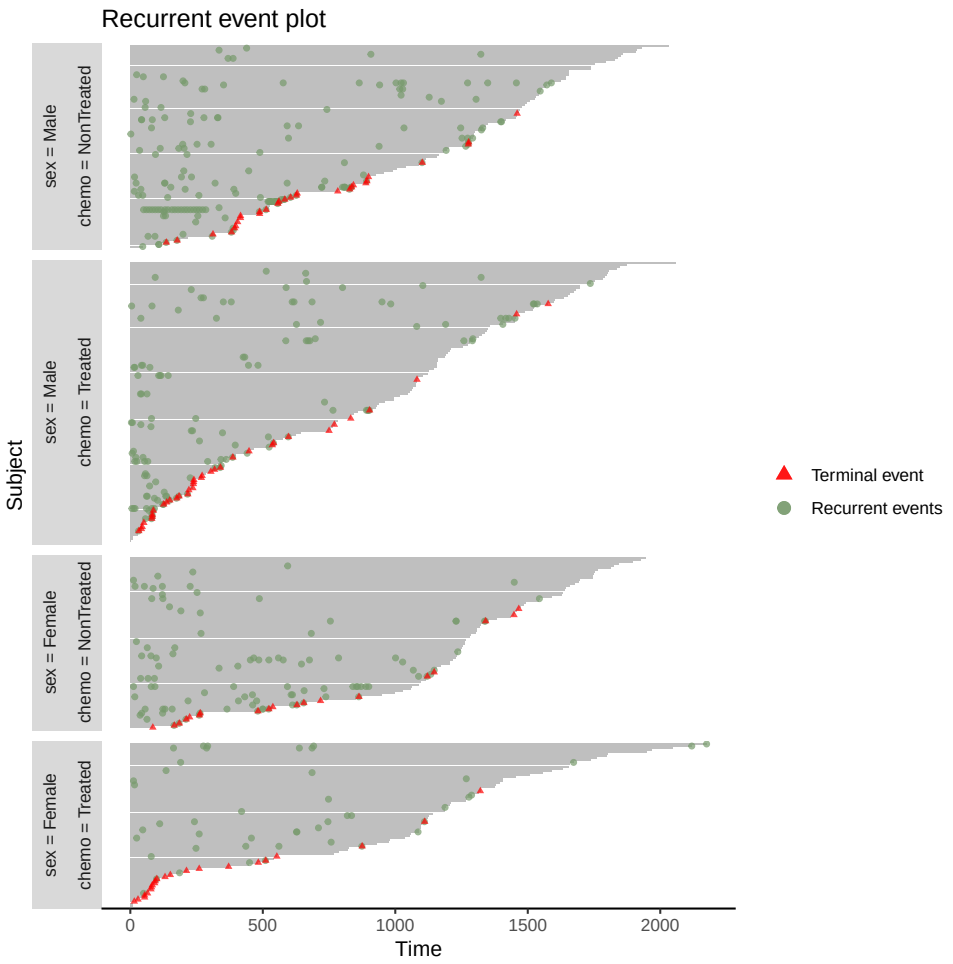
```
> plotEvents(reObj ~ sex, data = readmission0) ## or
> plotEvents(Recur(t.stop, id, event, death) ~ sex, data = readmission0)
```



Event plots: `plotEvents()`

- The `plotEvents()` can handle more complicated stratification.

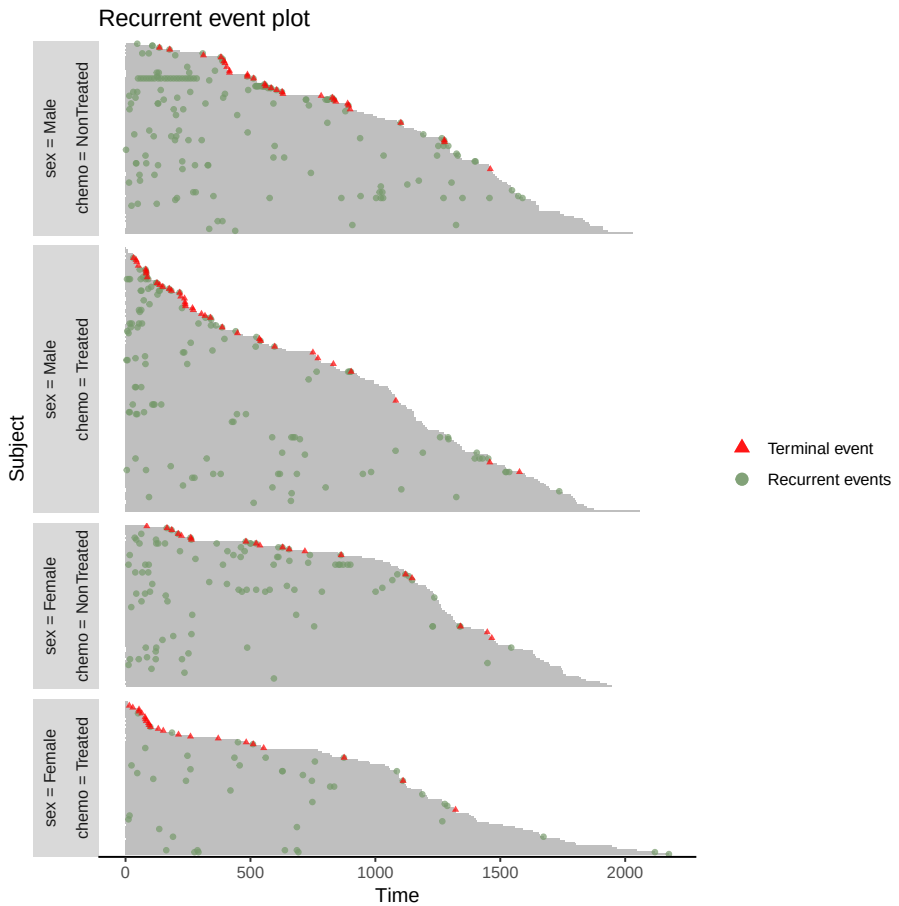
```
> plotEvents(Recur(t.stop, id, event, death) ~ sex + chemo, data = readmission0)
```

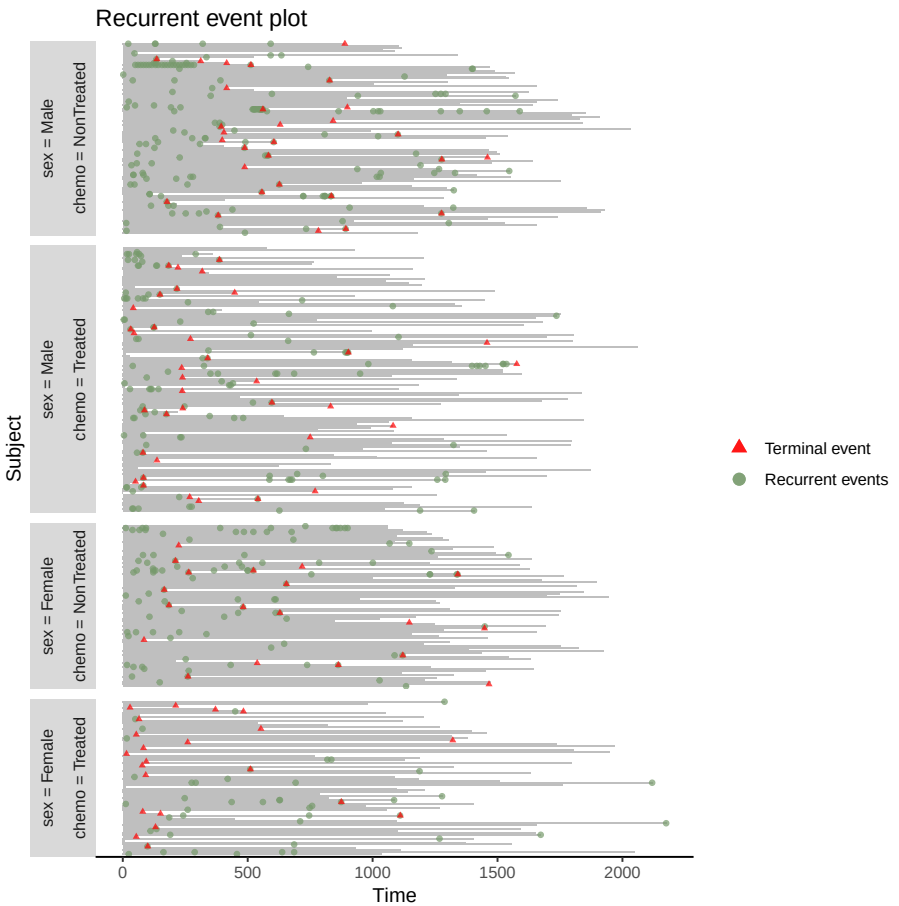


Event plots: `plotEvents()`

- The different sorting of the event times

```
> plotEvents(Recur(t.stop, id, event, death) ~ sex + chemo, result = "dec",
+            data = readmission0)
> plotEvents(Recur(t.stop, id, event, death) ~ sex + chemo, result = "asis",
+            data = readmission0)
```

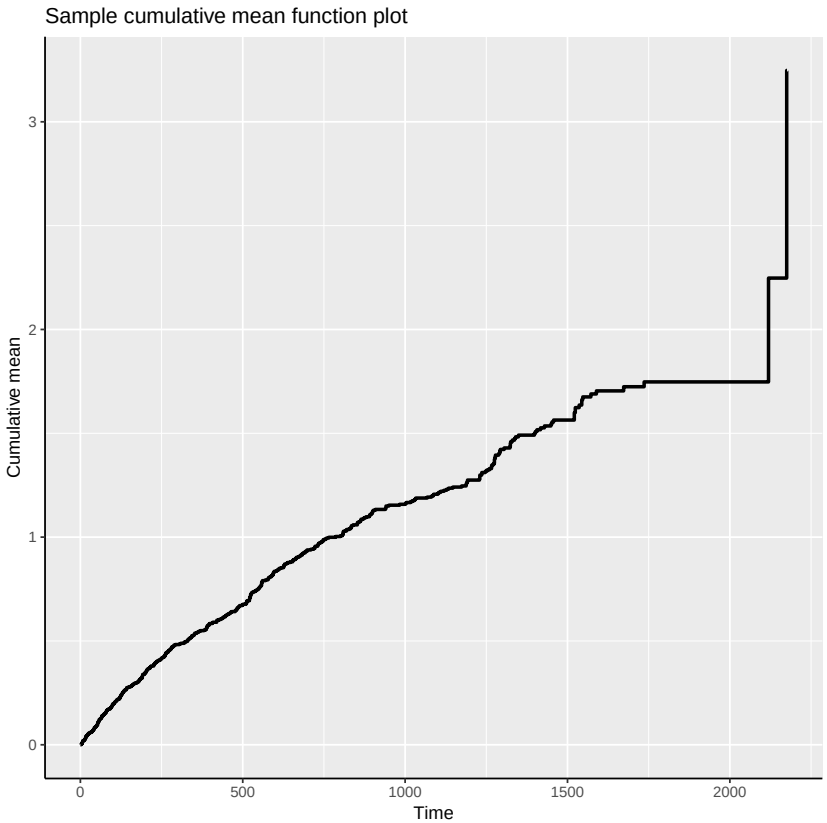




Cumulative sample mean function: plotting `Recur()`

- The Nelson-Aalen estimator is implemented in both the `reda` (as `mcf()`) and `reReg` packages, but we will focus on the implementation in `reReg`.
- The CSM plot can be called by specifying the `CSM = TRUE` in `plot()`

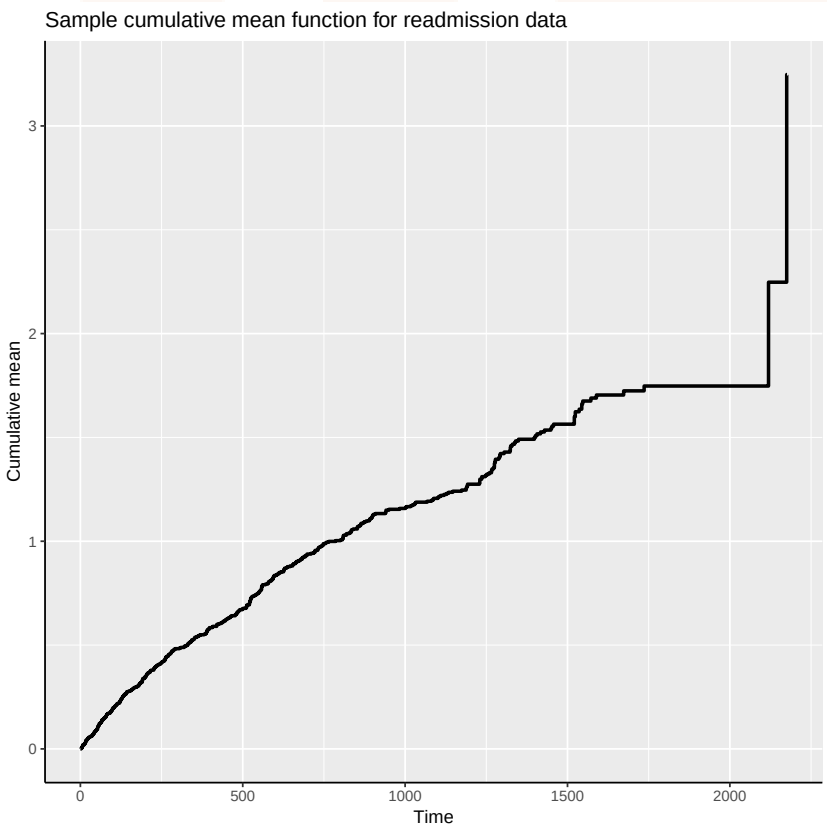
```
> reObj <- with(readmission0, Recur(t.stop, id, event, death))
> plot(reObj, CSM = TRUE)
```



Cumulative sample mean function: plotting `Recur()`

- Calling `plot()` directly computes (2), without adjusting to the censoring times.
- Just like the event plots, basic graphical options can be added directly.

```
> plot(reObj, CSM = TRUE,
+      main = "Sample cumulative mean function for readmission data")
```



Cumulative sample mean function: `plotCSM()`

- The CSM plot can also be created with the more specialized function, `plotCSM()`
- The arguments are

```
> args(plotCSM)
function (formula, data, onePanel = FALSE, adjrisk = TRUE, smooth = FALSE,
    control = list(), ...)
NULL
```

formula is a formula object; similar to that in `plotEvents()`

data is an optional data frame; similar to that in `plotEvents()`

onePanel is an optional logical value indicating whether the CSM will be plotted in the same panel. This is useful when there are multiple recurrent event types or in the presence of (discrete) covariates.

Cumulative sample mean function: `plotCSM()`

- More arguments:

  adjrisk is an optional logical value indicating whether risk set will be adjusted, e.g., if 'TRUE', subjects leave the risk set after terminal times and the Nelson-Aalen estimator will be plotted.

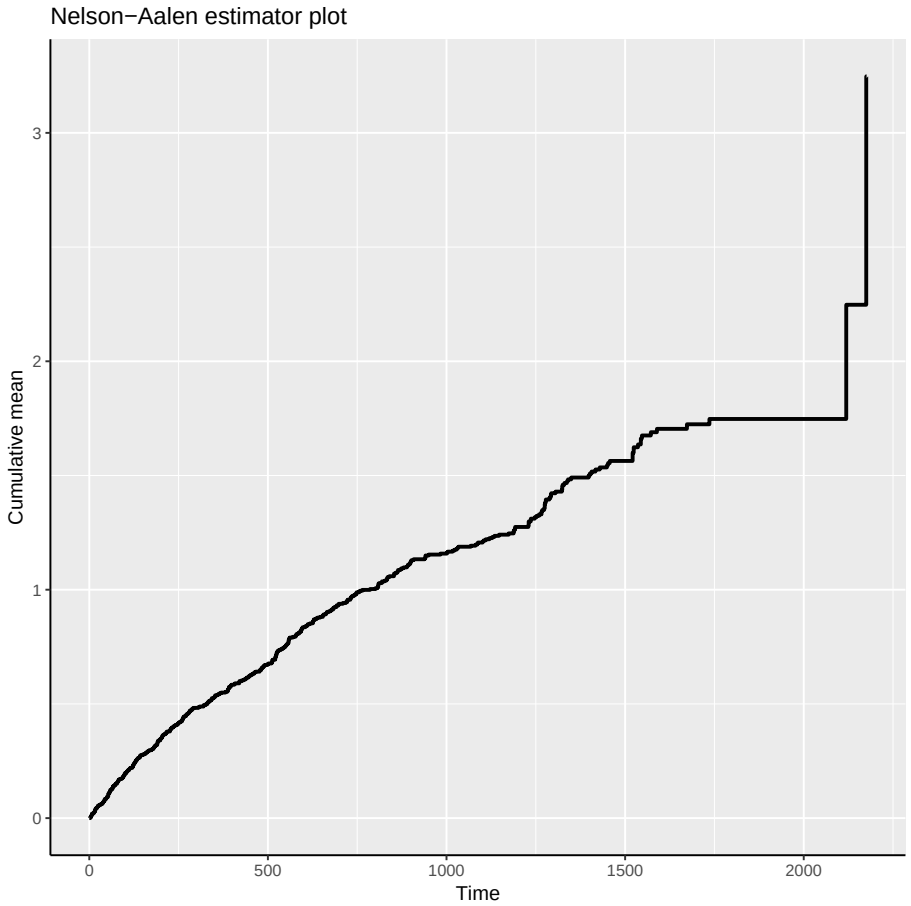
  smooth is an optional logical value indicating whether to add a smooth curve obtained from a monotone increasing *p*-splines implemented in package `scam`. This feature only works for data with one recurrent event type.

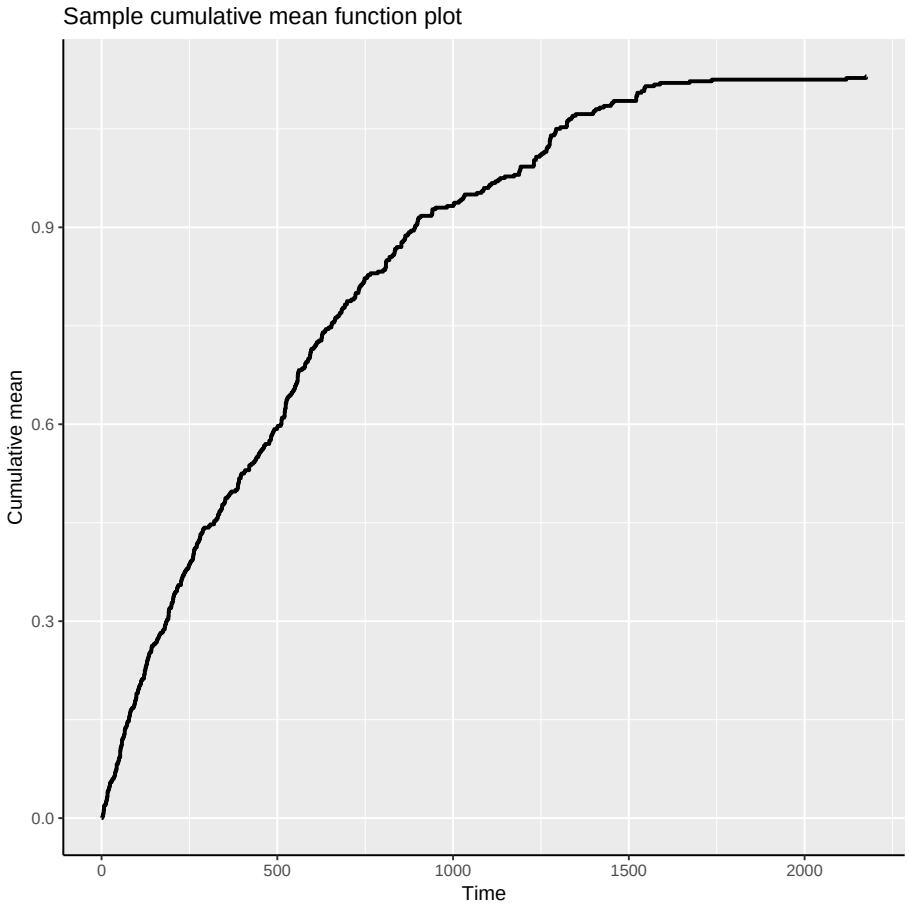
  control a list of control parameters; similar to that in `plotEvents()`

Cumulative sample mean function: `plotCSM()`

- The functions `plotCSM()` and `plotEvents()` share a lot of similarities
- A side-by-side CSM plot, showing the CSM functions with and without risk adjustment:

```
> plotCSM(reObj) + ggtitle("Nelson-Aalen estimator plot")
> plotCSM(reObj, adjrisk = FALSE)
```

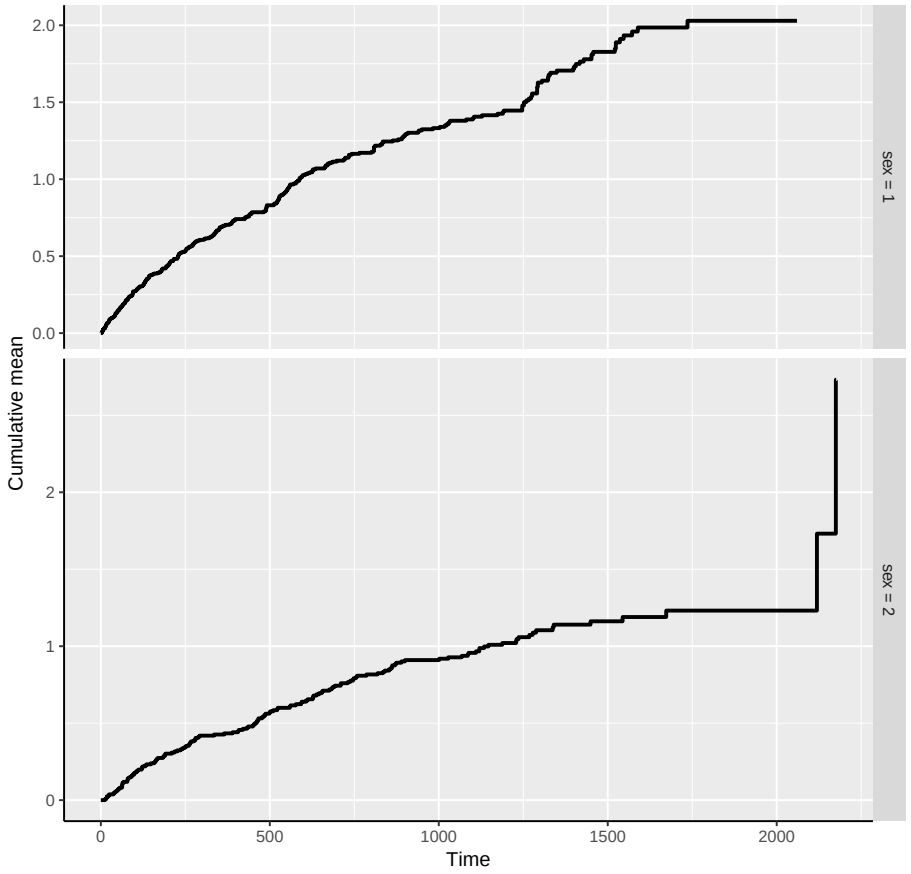


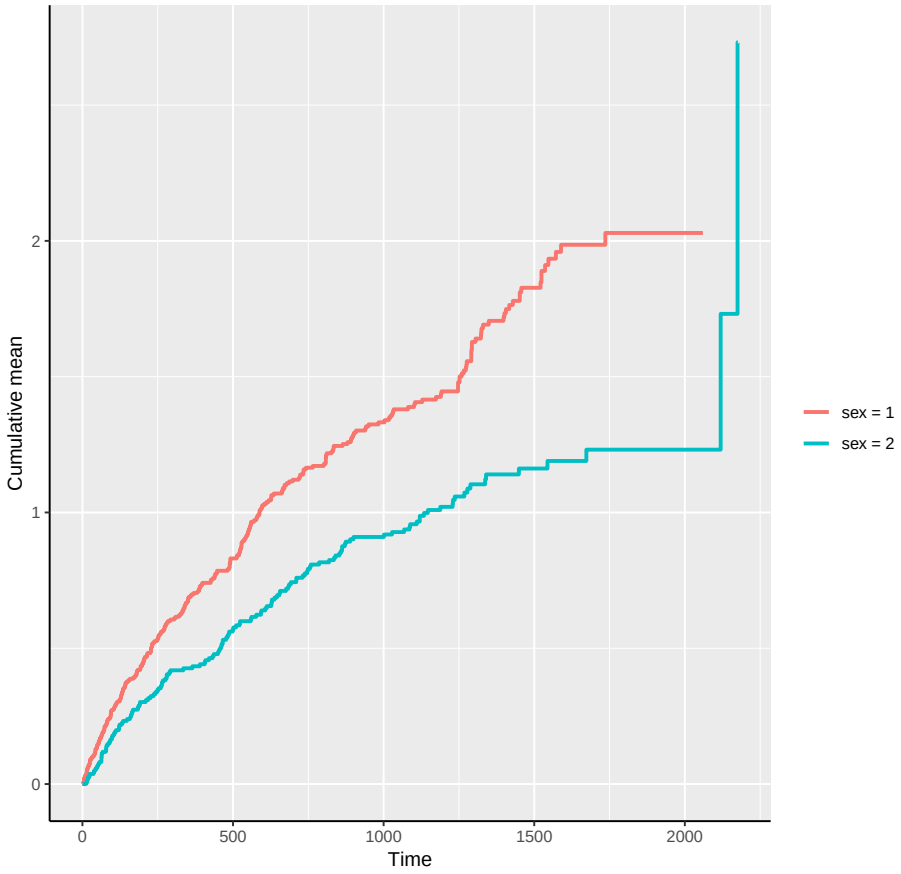


Cumulative sample mean function: `plotCSM()`

- Like the `plotEvents()`, `plotCSM()` can generate CSM plots given covariates.

```
> plotCSM(reObj ~ sex, data = readmission0, main = "")
> plotCSM(reObj ~ sex, onePanel = TRUE, data = readmission0, main = "")
```





More `plot` arguments

- The `plot()` function can be used to create event plots and CSM plots by directly applying to a `Recur()` object.
- Some useful arguments from `plotEvents()` and `plotCSM()` can be called with `plot()`

```
> args(reReg:::plot.Recur)
function (x, CSM = FALSE, event.result = c("increasing", "decreasing",
    "asis"), csm.adjrisk = TRUE, csm.smooth = FALSE, control = list(),
    ...)
NULL
```

`plotEvents()` and `plotCSM()` with multiple event types

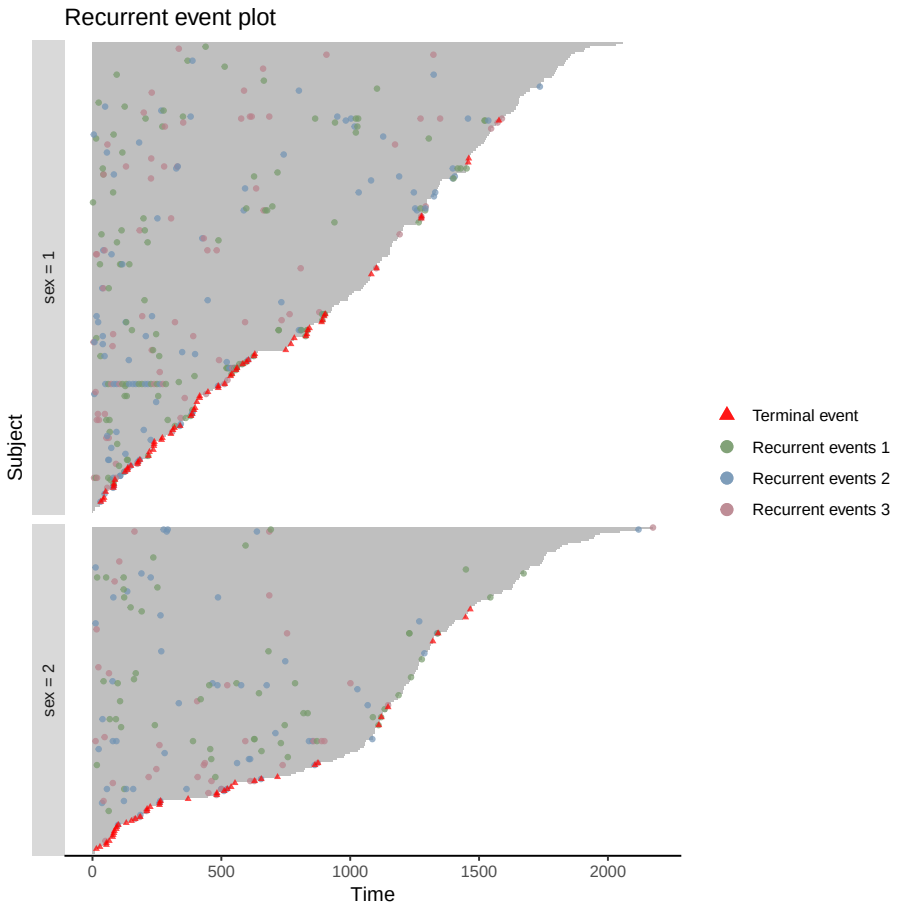
- Both functions `plotEvents()` and `plotCSM()` can be used to accommodate recurrent event data with multiple recurrent types.
- For illustration, we generate hypothetical event types for the `readmission` data and store the corresponding indicator in `event`.

```
> set.seed(0); readmission0$event <- readmission0$event * sample(1:3, 852, TRUE)
> head(readmission0)
  id enum t.start t.stop time event      chemo    sex dukes charlson death
1  1    1       0     24   24     2    Treated Female     D        3     0
2  1    2      24    457  433     1    Treated Female     D        0     0
3  1    3     457   1037  580     0    Treated Female     D        0     0
4  2    1       0    489  489     1 NonTreated   Male     C        0     0
5  2    2     489   1182  693     0 NonTreated   Male     C        0     0
6  3    1       0     15   15     1 NonTreated   Male     C        3     0
```

`plotEvents()` and `plotCSM()` with multiple event types

- The `plotEvents()` distinguishes the different recurrent event types by color:

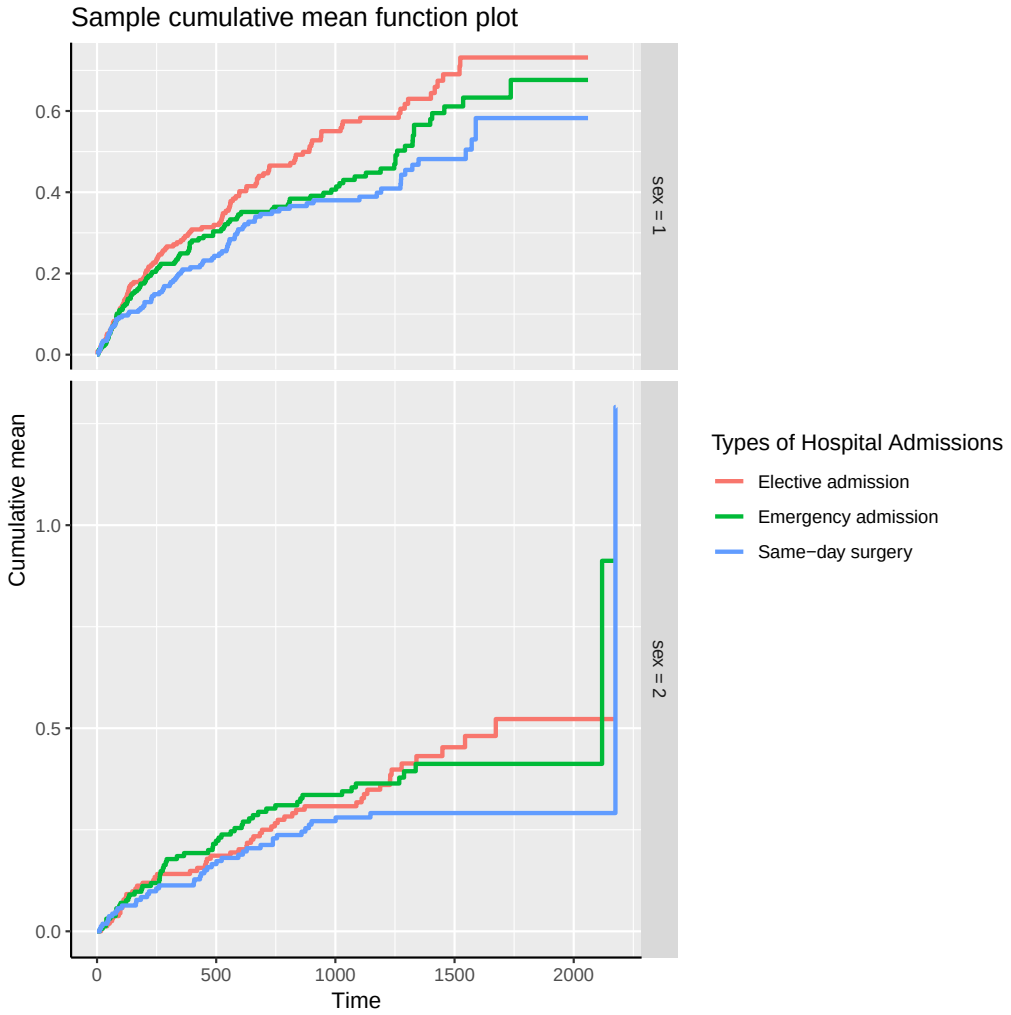
```
> plotEvents(Recur(t.stop, id, event, death) ~ sex, data = readmission0)
```



plotEvents() and plotCSM() with multiple event types

- The legend labels can be modified.

```
> rTypes <- c("Elective admission", "Emergency admission", "Same-day surgery")
> plotCSM(Recur(t.stop, id, event, death) ~ sex, data = readmission0,
+         recurrent.name = "Types of Hospital Admissions",
+         recurrent.type = rTypes)
```



Simulating recurrent event data; `simSC()`

Simulating recurrent event data

- The function `simSC()` is used to generate recurrent times data by specifying the
  - The rate function for the recurrent event process is $\lambda(t)$
  - The hazard function for the terminal event is $h(t)$
- The arguments are:

```
> args(simSC)
function (n, a, b, type = "cox", zVar = 0.25, tau = 60, summary = FALSE)
NULL
```

- The argument `type` controls the structure of the rate function and the hazard function.

Simulating recurrent event data

- The rate function, $\lambda(t)$, of the recurrent process can be specified as
  - Cox-type model:

$$\lambda(t) = Z\lambda_0(t)e^{X^\top a}$$

  - Accelerated mean model

$$\lambda(t) = Z\lambda_0(te^{X^\top a})e^{X^\top a}$$

  - Scale-change model

$$\lambda(t) = Z\lambda_0(te^{X^\top a})e^{X^\top b}$$

Simulating recurrent event data

- The notations are:
  - $\lambda_0(t)$ is the baseline rate function
  - $Z \sim \text{Gamma}(\gamma, \gamma)$ is a frailty variable
  - $X = (X_1, X_2)$ is the covariate vector, where
    - X_1 follows a Bernoulli distribution with probability 0.5
    - X_2 follows a standard normal distribution
  - a and b are the coefficients

Simulating recurrent event data

- The hazard function, $h(t)$, of the terminal event can be specified as:
  - Cox-type model:
$$h(t) = Zh_0(t)e^{X^\top a}$$
  - Accelerated mean model
$$h(t) = Zh_0(te^{X^\top a})e^{X^\top a}$$
  - Scale-change model
$$h(t) = Zh_0(te^{X^\top a})e^{X^\top b}$$
- The baseline hazard function is denoted by $h_0(t)$

Simulating recurrent event data

- The type of rate function and the hazard function can be specified by the `type` argument.
- The type is distinguished by a vertical bar ('`|`'), with rate function on the left.
  - Setting `type = "cox|am"` generates the recurrent process from a Cox model and the terminal event from an accelerated mean model.
- When only one type is specified, both the recurrent process and the terminal event will be generated from that model.
  - Setting `type = "cox|cox"` is the same as setting `type = "cox"`

Simulating recurrent event data

- The essential arguments for `simSC()` are

  n is the number of observation

  a & b are numeric vectors of parameter of length two

  type a character string specifying the underlying model.

- In addition to the terminal events, a potentially informative censoring time, C, is generated separately from an exponential distribution with rate

$$I(X_1 = 1) \cdot \frac{1}{60} + I(X_1 = 0) \cdot \frac{Z^2}{30}.$$

- The argument that control the strength of the informative censoring time is

  zVar a numeric variable specifying the variance of Z (default = 0.25). When `zVar` = 0, the frailty variable is set to $Z = 1$.

Simulating recurrent event data

- With the above configuration, the `simSC()` generates recurrent events up to the minimum of τ, censoring time, and terminal event.
- The administrative censoring, τ, is controlled by the argument `tau`.
- A quick summary about the simulated data will be reported by specifying `summary = TRUE`:

```
> set.seed(0); datCox <- simSC(200, c(1, 1), c(1, 1), summary = TRUE)

Summary results for number of recurrent event per subject:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   1.000   4.000   7.465   9.250  59.000

Number of failures: 96 (48%); Number of censored events: 104 (52%)

Number of x1 == 1: 107 (53.5%); Number of x1 == 0: 93 (46.5%)
Summary results for x2:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-3.00805 -0.73887 -0.06734 -0.06081  0.59361  3.81028
```

Simulating recurrent event data

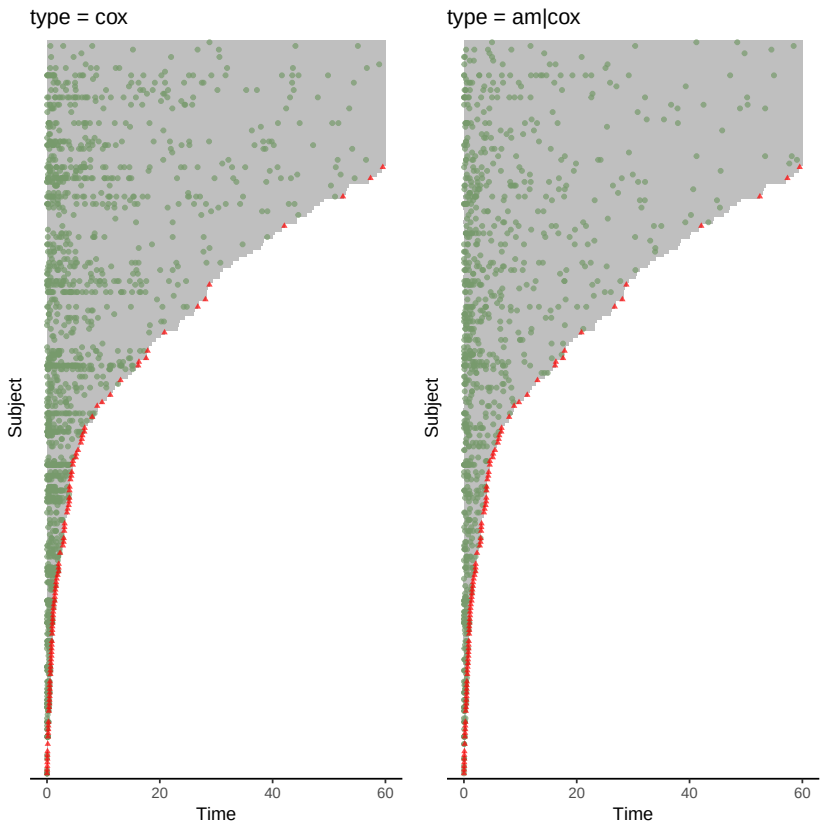
- The following example generates recurrent events from an accelerated mean model.

```
> set.seed(0)
> datAmCox <- simSC(200, c(1, 1), c(1, 1), type = "am|cox", summary = TRUE)

Summary results for number of recurrent event per subject:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   2.000   4.000   4.525   6.250  19.000

Number of failures: 96 (48%); Number of censored events: 104 (52%)

Number of x1 == 1: 107 (53.5%); Number of x1 == 0: 93 (46.5%)
Summary results for x2:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-3.00805 -0.73887 -0.06734 -0.06081  0.59361  3.81028
```

Simulating recurrent event data

- A side-by-side event plot shows the difference between `datCox` and `datAmCox` is in the recurrent event process.

```
> grid.arrange(plotEvents(Recur(Time, id, event, status) ~ 1, data = datCox,
+                         main = "type = cox") + theme(legend.position = "none"),
+              plotEvents(Recur(Time, id, event, status) ~ 1, data = datAmCox,
+                         main = "type = am|cox") + theme(legend.position = "none
+              ncol = 2)
```



Fitting regression model with `reReg()`

Fitting regression model with `reReg()`

- A regression model is needed to assess the covariate effects on the recurrent event process and terminal events.
- The `reReg()` function in `reReg` provides different approaches to fit semiparametric regression model to recurrent event data.
- The arguments of the `reReg()` function are

```
> args(reReg)
function (formula, data, B = 200, method = c("cox.LWYY", "cox.GL",
    "cox.HW", "am.GL", "am.XCHWY", "sc.XCYH"), se = c("NULL",
    "bootstrap", "resampling"), control = list())
NULL
```

Fitting regression model with `reReg()`

- The description of the arguments are

  `formula` is a formula object, created by `Recur()`.

  `data` is an optional data frame.

  `B` is a numeric value specifies the number of resampling or bootstrap for the variance estimation. When `B = 0`, variance estimation will not be carried out and standard error will be returned as `NA`.

  `method` a character string specifying the underlying model.

  `se` a character string specifying the method for the variance estimation.

  `control` a list of control parameters.

- Most arguments of `reReg()` are straightforward, and we will focus our discussion on `method`, `se`, and `control`.

Fitting regression model with `reReg()`

- There are currently six available methods in `reReg()`; they each have their own strength and limitation.
- These methods are generally different by
  - model assumptions, e.g., Cox-type model, accelerated mean model, or scale-change model
  - the presence of a terminal event
- In the following, we will list important properties of these methods with some discussion on the estimating techniques.

`reReg()` with `method = "cox.LWYY"`

- When `method = "cox.LWYY"`, `reReg()` implements the Andersen-Gill intensity model (Self et al., 1982), whose inference procedure was discussed in Pepe and Cai (1993); Lin et al. (2000).
- The model assumes the covariates are associated with the rate function via

$$\lambda(t|X) = \lambda_0(t)e^{X^\top \alpha},$$

where $\lambda_0(t)$ is the baseline rate function, X is a time independent p-dimensional covariate vector, and α is the regression coefficient.

`reReg()` with `method = "cox.LWYY"`

- The standard estimation procedures can be derived in more than one way (e.g. Cook and Lawless, 2007, Section 3.4.2).
- In particular, the partial likelihood score function for α is

$$\sum_{i=1}^{n} \int_0^t \{X_i - \bar{X}(t, \alpha)\} \, \mathrm{d}N_i(u),$$

where $N_i(\cdot)$ is the observed counting process as defined before, $\bar{X}(t, \alpha) = \left[\sum_{i=1}^{n} I(C_i \geq t) X_i e^{X_i^\top \alpha}\right] / \left[\sum_{i=1}^{n} I(C_i \geq t) e^{X_i^\top \alpha}\right]$.

`reReg()` with `method = "cox.LWYY"`

- Advantage:
  - Model is computational efficient because the estimating equation can be solved by the `coxph()` from the `survival` package.
  - Robust variance estimation is available
- Limitation:
  - The model does not incorporate terminal events

reReg() with method = "cox.LWYY"

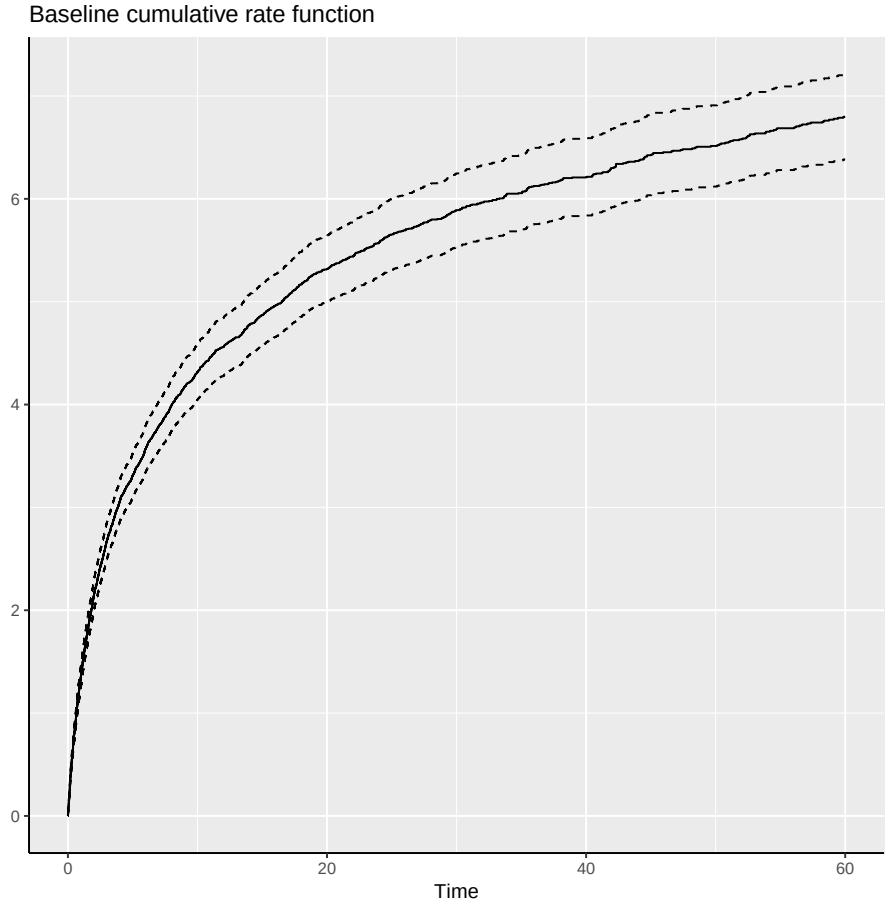
- Suppose we generate a simulated data from a Cox model:

```
> set.seed(0); datCox <- simSC(500, c(1, -1), c(1, -1))
> fit <- reReg(Recur(Time, id, event, status) ~ x1 + x2, data = datCox)
> summary(fit)
Call: reReg(formula = Recur(Time, id, event, status) ~ x1 + x2, data = datCox)

Method: Lin-Wei-Yang-Ying method (fitted with coxph with robust variance)

Coefficients effect:
   Estimate StdErr z.value   p.value
x1    1.005  0.072  13.958 < 2.2e-16 ***
x2   -0.890  0.042 -21.322 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

reReg() with method = "cox.LWYY"

- The estimated baseline rate function λ_0 can be plotted with R's generic function plot().

```
> plot(fit)
Baseline cumulative hazard function is not available for method = cox.LWYY.
Only the baseline cumulative rate function is plotted.
```



`reReg()` with `method = "cox.GL"`

- When `method = "cox.GL"`, `reReg()` gives the methods proposed in Ghosh and Lin (2002).
- Ghosh and Lin (2002) assumes the covaraites are associated with the mean function via

$$\mu(t|X) = \mu_0(t)e^{X^\top \alpha},$$

  where $mu_0(t)$ is the baseline mean function.
- The formulation on the mean function reduces to

$$\mathrm{d}\mu(t|X) = \mathrm{d}\mu_0(t)e^{X^\top \alpha},$$

  when X is time-independent.

`reReg()` with `method = "cox.GL"`

- Ghosh and Lin (2002) first considered an inverse probability censoring weighting (IPCW) approach to account for the presence of a terminal event.
- The IPCW approach requires modeling the censoring distribution, which is a nuisance.
- Ghosh and Lin (2002) then proposed an alternative method that involves modeling the survival distribution of the terminal event times.
- Ghosh and Lin (2002) specifies a proportional hazards model for the terminal events:

$$h(t|X) = h_0(t)e^{X^\top \beta},$$

where $h_0(t)$ is the baseline hazard function.

`reReg()` with `method = "cox.GL"`

- The estimation of $h_0(t)$ and β can be obtained via the partial likelihood approach derived for Cox models, e.g., via `coxph()`.
- Once $h_0(t)$ and β are estimated, an estimation of $S(t|X) = \exp\left(-\int_0^t e^{X^\top \beta} \mathrm{d}h_0\right)$ can be calculated.
- The regression coefficient for the rate function is then the root of the estimating equation:

$$\sum_{i=1}^{n} \int_0^t \left\{X_i - \bar{X}_\omega(t,\alpha)\right\} \omega_i(t) \mathrm{d}N_i(u),$$

where $\bar{X}_\omega(t,\alpha) = \left[\sum_{i=1}^n I(C_i \geq t)\omega_i(t) X_i e^{X_i^\top \alpha}\right] / \left[\sum_{i=1}^n I(C_i \geq t)\omega_i(t) e^{X_i^\top \alpha}\right]$, and $\omega_i(t) = I(X_i \geq t)/\widehat{S}(t|X)$.

reReg() with method = "cox.GL"

- The proposed model can be fitted with

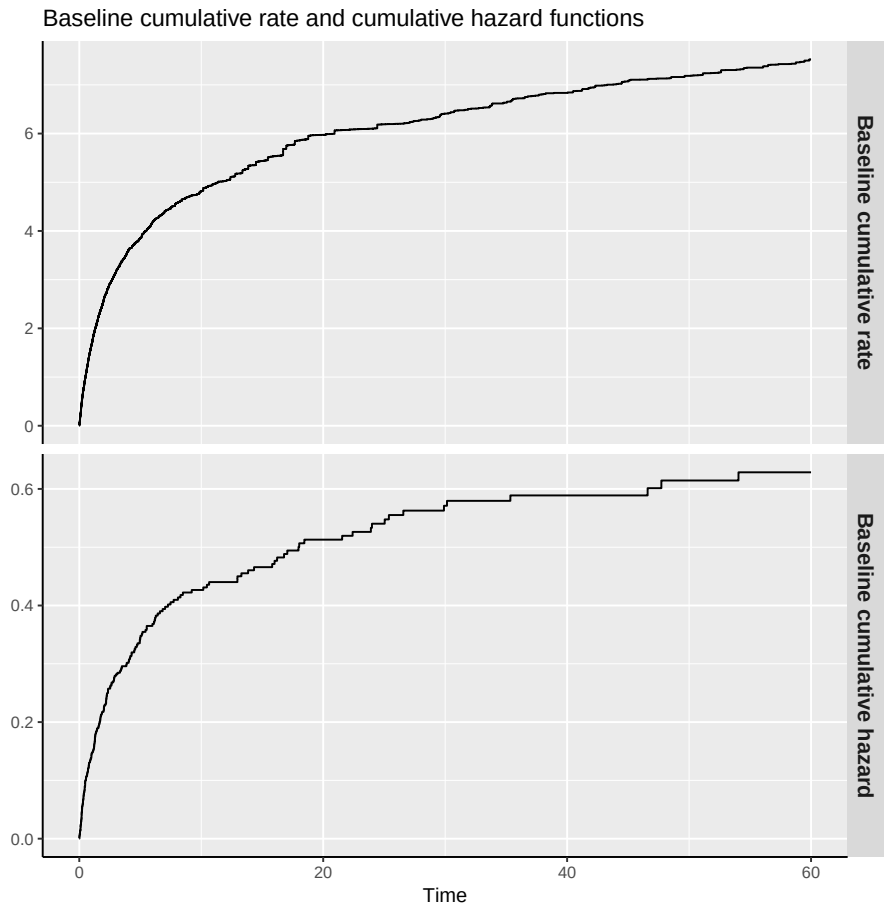
```
> fit <- reReg(Recur(Time, id, event, status) ~ x1 + x2, data = datCox,
+              method = "cox.GL")
> summary(fit)
Call: reReg(formula = Recur(Time, id, event, status) ~ x1 + x2, data = datCox,
    method = "cox.GL")

Coefficients (rate):
   Estimate StdErr z.value p.value
x1    0.917     NA      NA      NA
x2   -0.739     NA      NA      NA

Coefficients (hazard):
   Estimate StdErr z.value p.value
x1    0.758     NA      NA      NA
x2   -0.867     NA      NA      NA
```

- The variance estimation is not provided by default.

reReg() with method = "cox.GL"

- The baseline functions can be plotted easily

```
> plot(fit)
```



reReg() with method = "cox.GL"

- The functions `plotRate()` and `plotHaz()` can be applied to extract the baseline rate function and the baseline hazard function, respectively.

```
> args(plotRate)
function (x, smooth = FALSE, control = list(), ...)
NULL
> args(plotHaz)
function (x, smooth = FALSE, control = list(), ...)
NULL
```

- The arguments are

x is an object of class `reReg` returned by the `reReg()` function

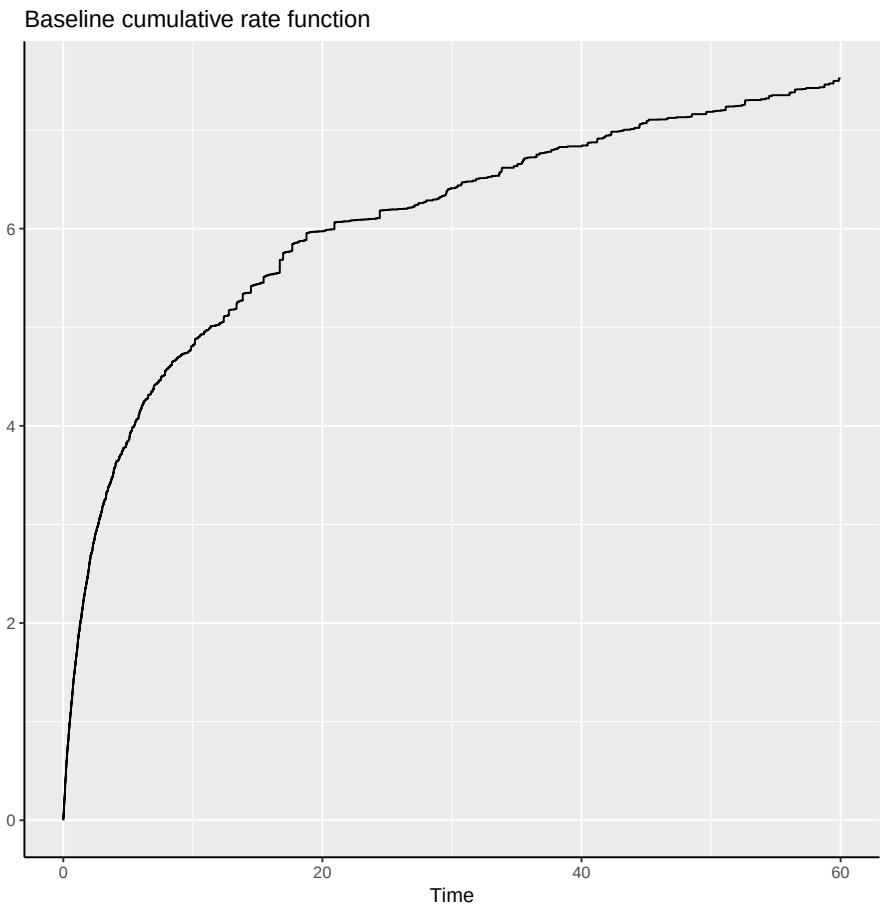
smooth is an optional logical value indicating whether to add a smooth curve obtained from a monotone increasing *p*-spline.

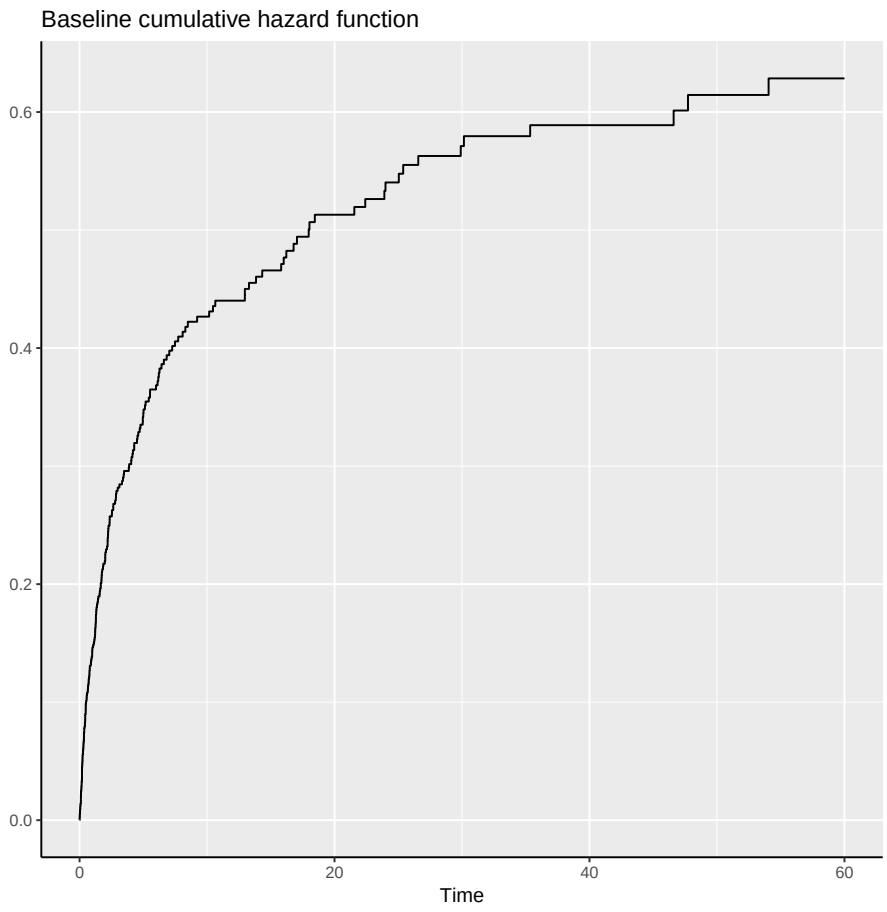
control is a list of control parameters; this is similar to the `control` argument in `plotEvents`. Graphical parameters like `xlab`, `ylab`, `main`, etc, can also be specified outside of the `control` list.

reReg() with method = "cox.GL"

- The baseline functions can be plotted easily

```
> plotRate(fit)
> plotHaz(fit)
```

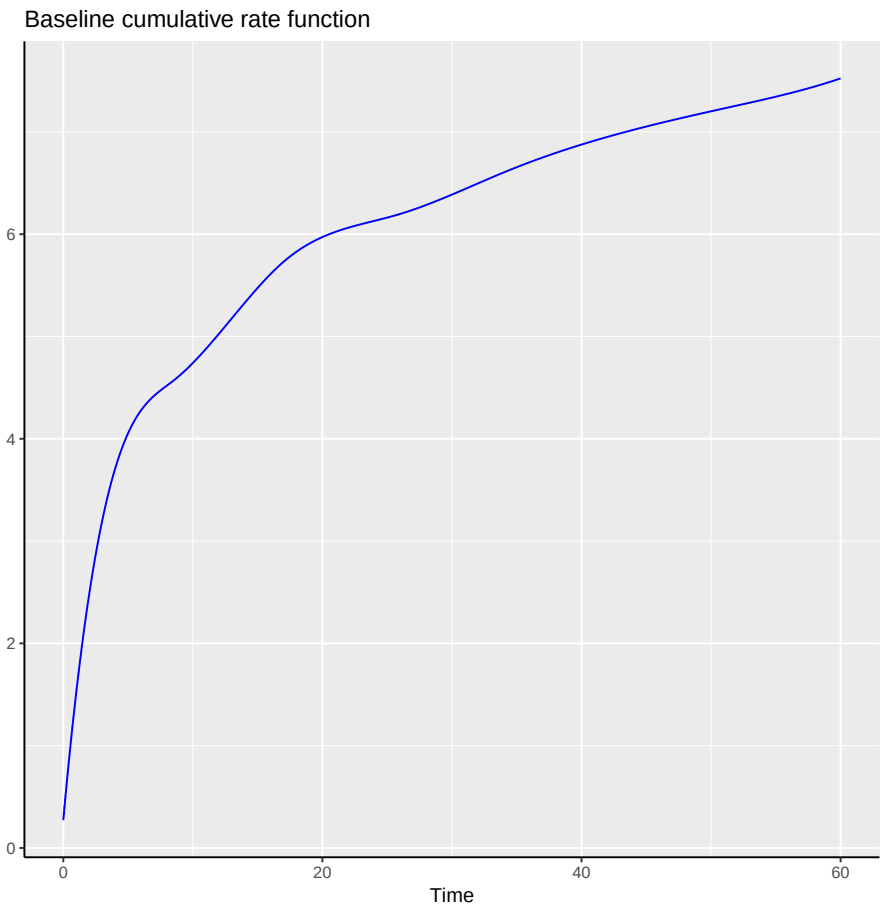


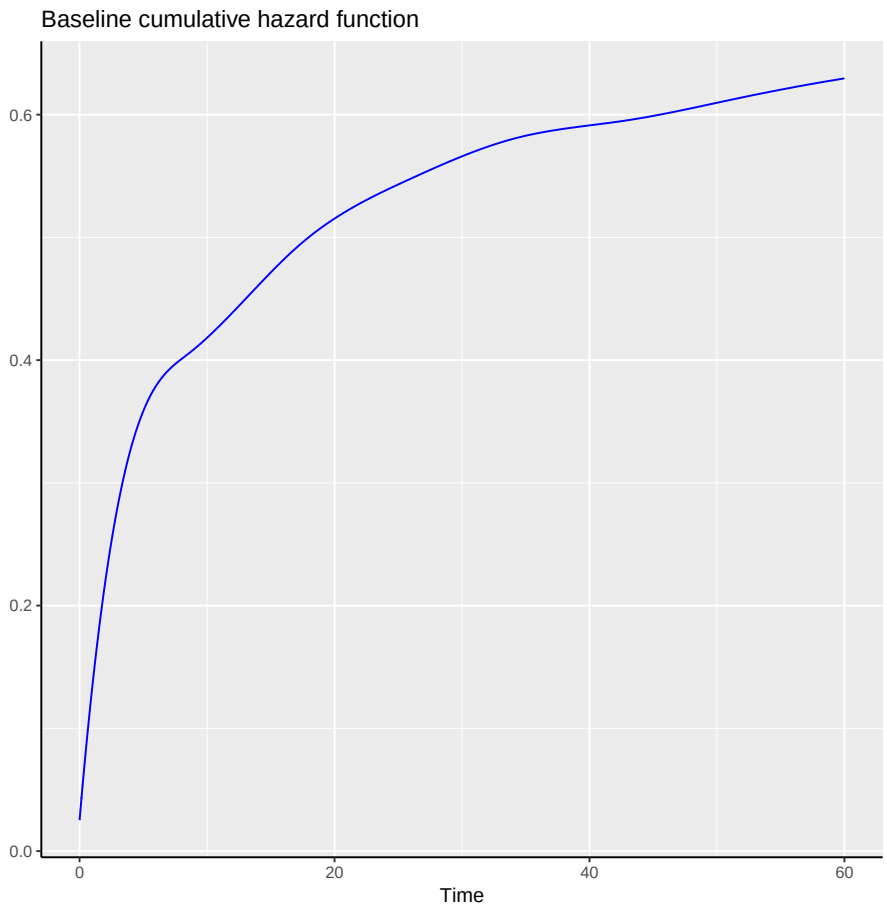


reReg() with method = "cox.GL"

- The baseline functions with smooth splines

```
> plotRate(fit, smooth = TRUE)
> plotHaz(fit, smooth = TRUE)
```





`reReg()` with `method = "cox.GL"`

- Advantage:
  - The estimating equation can be solved by the `coxph()` from the `survival` package.
  - A joint model that allows terminal event
- Limitation:
  - Assumes the recurrent event process is independent of the terminal events given covariates

`reReg()` with `method = "cox.HW"`

- When `method = "cox.HW"`, `reReg()` gives the method proposed in Huang and Wang (2004).
- Huang and Wang (2004) considered a similar joint model:

$$\begin{cases} \text{Rate:} & \lambda(t) = Z\lambda_0(t)e^{X^\top \alpha} \\ \text{Hazard:} & h(t) = Zh_0(t)e^{X^\top \beta}. \end{cases}$$

- The main difference is in the inclusion of a nonnegative frailty variable, Z.

`reReg()` with `method = "cox.HW"`

- One advantage of the frailty is that it accounts for heterogeneity that cannot be explained by the observed covariates.
- Different from many frailty models, the model proposed by Huang and Wang (2004) does not require a parametric assumption in the estimation procedure.
- For identifiability, Huang and Wang (2004) assumes $\Lambda_0(\tau) = 1$ and $\mathrm{E}(Z_i|X_i) = \mathrm{E}(Z_i) = \mu_Z$ is constant.

`reReg()` with `method = "cox.HW"`

- The estimation of α, β, $\lambda_0(t)$, and $h_0(t)$ can be divided into the following steps:
  1. Following Wang et al. (2001), the baseline cumulative rate function is estimated by
  $$\widehat{\Lambda}_0(t) = \prod_{s_{(l)} > t} \left\{ 1 - \frac{\sum_{i=1}^{n} \sum_{j=1}^{m_i} I(t_{ij} = s_{(l)})}{\sum_{i=1}^{n} \sum_{j=1}^{m_i} I(t_{ij} \leq s_{(l)} \leq Y_i)} \right\},$$
  where $\{s_{(l)}\}$'s are ordered, distinct values of $\{t_{ij}\}$.
  2. Estimate α and μ_z through
  $$U_{1n}(\gamma) = \frac{1}{n} \sum_{i=1}^{n} \bar{X}_i^\top \left\{ \frac{m_i}{\widehat{\Lambda}_0(Y_i)} - e^{\bar{X}_i^\top \gamma} \right\},$$
  where $\bar{X}_i = \{1, X_i\}$, $\gamma = (\log(\mu_z), \alpha)^\top$.

`reReg()` with `method = "cox.HW"`

3. Estimate β through

$$U_{2n}(\beta) = \frac{1}{n}\sum_{i=1}^{n}\Delta_i\left\{X_i - \frac{\sum_{j=1}^{n} X_j Z_j e^{X_j^\top \beta} I(Y_j \geq Y_i)}{\sum_{j=1}^{n} Z_j e^{X_j^\top \beta} I(Y_j \geq Y_i)}\right\},$$

   where Z_j is estimated by

$$\widehat{Z}_i = \frac{m_i}{\Lambda_0(Y_i)e^{X_i^\top \widehat{\alpha}}},$$

   and is replaced by 0 if $0/0$ occurs.

4. The hazard function is estimated with

$$\widehat{H}_0(t) = \sum_{i=1}^{n}\frac{\Delta_i I(Y_i \leq t)}{\sum_{j=1}^{n}\widehat{Z}_j e^{X_j \widehat{\beta}} I(Y_j \geq Y_i)}.$$

reReg() with method = "cox.HW"

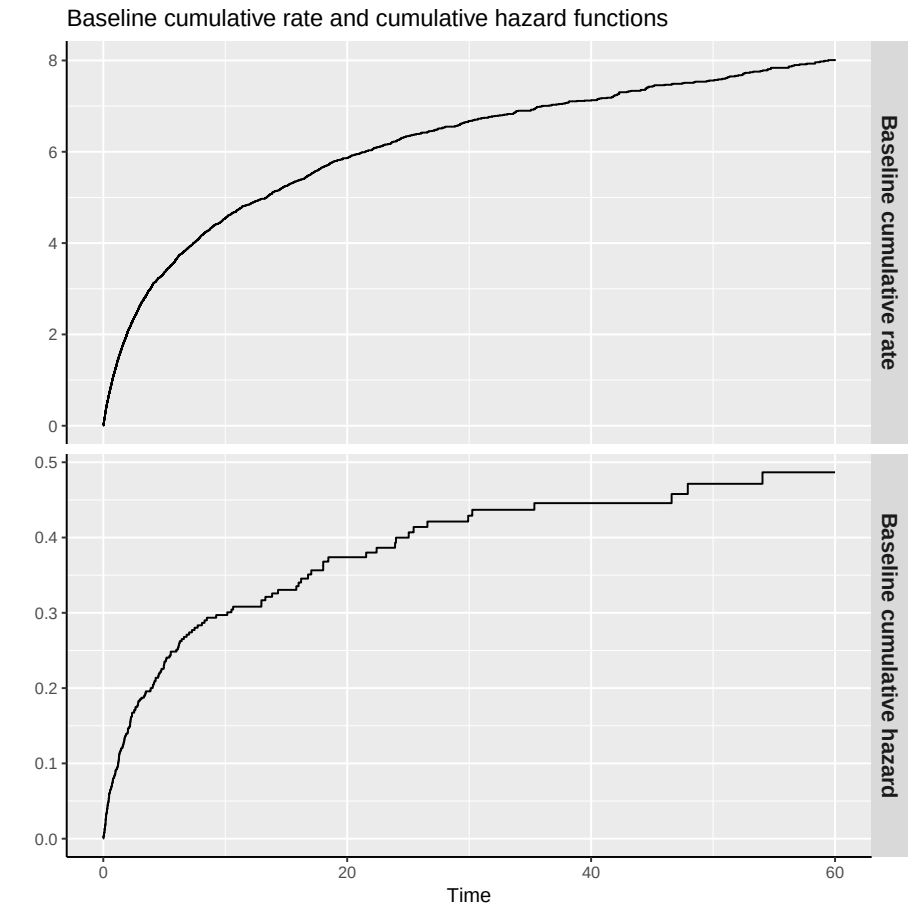
- The method can be fitted with

```
> fit <- reReg(Recur(Time, id, event, status) ~ x1 + x2, data = datCox,
+              method = "cox.HW")
> summary(fit)
Call: reReg(formula = Recur(Time, id, event, status) ~ x1 + x2, data = datCox,
    method = "cox.HW")

Method: Huang-Wang Model

Coefficients (rate):
   Estimate StdErr z.value p.value
x1    1.143     NA      NA      NA
x2   -1.016     NA      NA      NA

Coefficients (hazard):
   Estimate StdErr z.value p.value
x1    0.893     NA      NA      NA
x2   -0.984     NA      NA      NA
```

reReg() with method = "cox.HW"

- The baseline functions are given by

```
> plot(fit)
```



`reReg()` with `method = "cox.HW"`

- Advantage:
  - A joint model that allows terminal event
  - Allows informative censoring
  - A resampling based variance estimation can be adopted to facilitate variance estimation
  - The estimation procedure does not require the strong Poisson assumption.
- Limitation:
  - The variance estimation with bootstrap procedure could be slow when sample size is very large

`reReg()` with `method = "am.GL"`

- When `method = "am.GL"`, `reReg()` gives the method proposed in Ghosh and Lin (2003)
- The general idea is similar to the proposed method in Ghosh and Lin (2002).
- Ghosh and Lin (2003) assumes the joint model:

$$\begin{cases} \text{Rate:} & \lambda(t) = \lambda_0(te^{X^\top \alpha})e^{X^\top \alpha} \\ \text{Hazard:} & h(t) = h_0(te^{X^\top \beta})e^{X^\top \beta}. \end{cases}$$

- The marginal model is in an accelerated failure time model.
- Ghosh and Lin (2003) refers their model as an accelerated rate time model.

`reReg()` with `method = "am.GL"`

- The estimating procedure consists of the following steps:
  1. Estimating β:

$$U_{1n}(\beta) = \frac{1}{n}\sum_{i=1}^{n}\Delta_i\left\{X_i - \frac{\sum_{j=1}^{n} Z_j I(Y_j e^{X_j^\top \beta} \geq Y_i e^{X_i^\top \beta})}{\sum_{j=1}^{n} I(Y_j e^{X_j^\top \beta} \geq Y_i e^{X_i^\top \beta})}\right\}.$$

  2. Estimating α with artificial censoring:

$$U_{2n}(\alpha) = \sum_{i=1}^{n}\sum_{k=1}^{m_i} I(t_{ik} e^{X_i^\top \alpha} \leq Y_i e^{X_i^\top \alpha - d})\left[X_i - \frac{\sum_{j=1}^{n} X_j I(t_{ik} e^{X_i^\top \alpha} \leq Y_j e^{X_j^\top \alpha - d})}{X_j I(t_{ik} e^{X_i^\top \alpha} \leq Y_j e^{X_j^\top \alpha - d})}\right]$$

  where $d = \max_i X_i^\top(\widehat{\beta} - \alpha)$.

`reReg()` with `method = "am.GL"`

3. Baseline cumulative rate function:

$$\Lambda_0(t;\alpha) = \sum_{i=1}^{n} I(Y_i e^{X_i^\top \alpha - d} \leq t) \sum_{k=1}^{m_i} \frac{I(t_{ik} e^{X_i^\top \alpha} \leq Y_i e^{X_i^\top \alpha - d})}{\sum_{j=1}^{n} I(t_{ik} e^{X_i^\top \alpha} \leq Y_j e^{X_j^\top \alpha - d})}.$$

4. Baseline cumulative hazard function:

$$\Lambda_0(t;\beta) = \sum_{i=1}^{n} \frac{\Delta_i I(Y_i e^{X_i^\top \beta} \leq t)}{\sum_{j=1}^{n} I(Y_j e^{X_j^\top \beta} \geq Y_i e^{X_i^\top \beta})}.$$

reReg() with method = "am.GL"

- The reReg() fit gives:

```
> fit <- reReg(Recur(Time, id, event, status) ~ x1 + x2, data = datCox,
+              method = "am.GL")
> summary(fit)
Call: reReg(formula = Recur(Time, id, event, status) ~ x1 + x2, data = datCox,
    method = "am.GL")

Method: Ghosh-Lin Model

Coefficients (rate):
   Estimate StdErr z.value p.value
x1   -1.232     NA      NA      NA
x2  -77.590     NA      NA      NA

Coefficients (hazard):
   Estimate StdErr z.value p.value
x1    1.907     NA      NA      NA
x2   -2.103     NA      NA      NA
```

- A large bias is observed because datCox was generated under the Cox assumption.

reReg() with method = "am.GL"

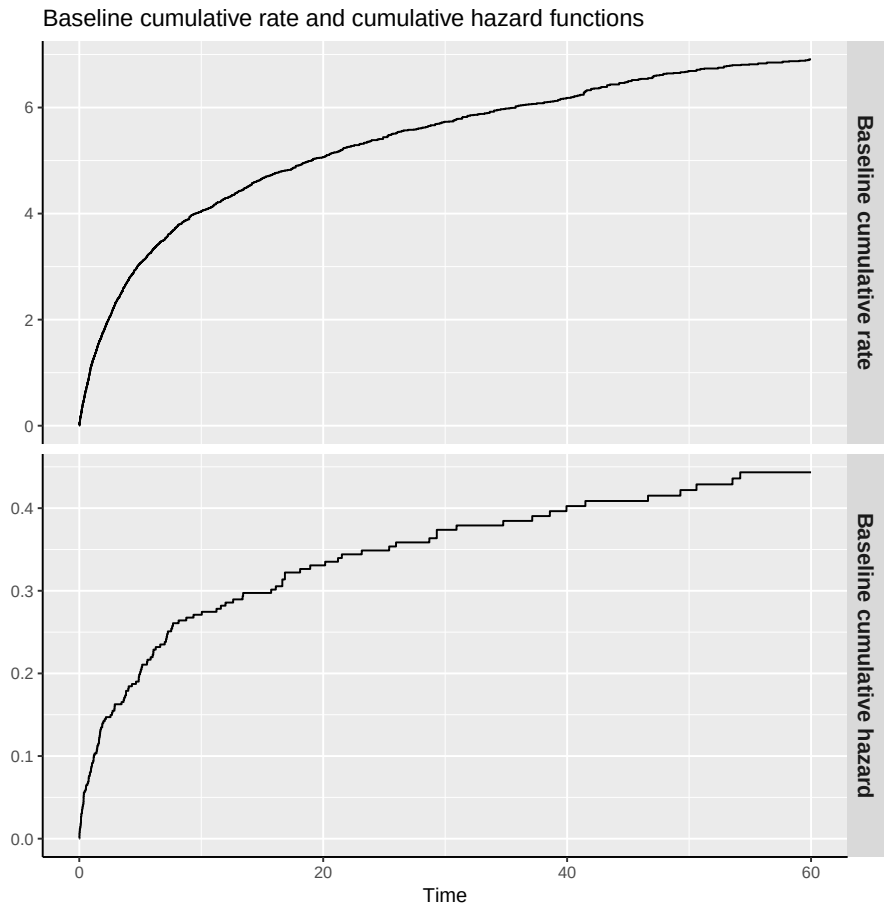
- The bias went down when generating under the correct model

```
> set.seed(0); datAM <- simSC(500, c(1, -1), c(1, -1), type = "am")
> fit <- reReg(Recur(Time, id, event, status) ~ x1 + x2, data = datAM,
+              method = "am.GL")
> summary(fit)
Call: reReg(formula = Recur(Time, id, event, status) ~ x1 + x2, data = datAM,
    method = "am.GL")

Method: Ghosh-Lin Model

Coefficients (rate):
   Estimate StdErr z.value p.value
x1    1.523     NA      NA      NA
x2   -1.105     NA      NA      NA

Coefficients (hazard):
   Estimate StdErr z.value p.value
x1    0.631     NA      NA      NA
x2   -1.146     NA      NA      NA
```

reReg() with method = "am.GL"

- The baseline functions are given by

```
> plot(fit)
```



`reReg()` with `method = "cox.HW"`

- Advantage:
  - A joint model that allows terminal event
- Limitation:
  - Does not account for informative censoring
  - Could be inefficient when the censoring rate is high
  - The variance estimation with bootstrap procedure could be slow when sample size is very large

`reReg()` with `method = "am.XCHWY"`

- When `method = "am.XCHWY"`, `reReg()` gives the method proposed in Xu et al. (2017)
- Xu et al. (2017) assumes the joint model:

$$\begin{cases} \text{Rate:} & \lambda(t) = Z\lambda_0(te^{X^\top \alpha})e^{X^\top \alpha} \\ \text{Hazard:} & h(t) = Zh_0(te^{X^\top \beta})e^{X^\top \beta}. \end{cases}$$

- As in the work of Huang and Wang (2004), the identifiability assumptions require $\Lambda_0(\tau) = 1$ and $E(Z|X) = E(Z) = \mu_Z$ is a constant.

`reReg()` with `method = "am.XCHWY"`

- The estimation procedure consists of the following steps:
  1. Consider the transformed time, $t_{ij}^*(a) = t_{ij}e^{X^\top a}$ and $Y_i^*(a) = Y_i e^{X^\top a}$, the baseline rate function can be estimated from

$$\widehat{\Lambda}_0(t,a) = \prod_{s_{(l)}>t}\left\{1 - \frac{\sum_{i=1}^n \sum_{j=1}^{m_i} I[t_{ij}^*(a) = s_{(l)}]}{\sum_{i=1}^n \sum_{j=1}^{m_i} I[t_{ij}^*(a) \le s_{(l)} \le Y_i^*(a)]}\right\},$$

     where $\{s_{(l)}\}$'s are ordered, distinct values of $\{t_{ij}^*(a)\}$.
  2. The regression coefficient α is estimated through solving

$$U_{1n}(a) = \frac{1}{n}\sum_{i=1}^n X_i\left\{\frac{m_i}{\widehat{\Lambda}_0[Y_i^*(a)]} - \frac{1}{n}\sum_{j=1}^n \frac{m_j}{\widehat{\Lambda}_0[Y_j^*(a)]}\right\}.$$

`reReg()` with `method = "am.XCHWY"`

3. Using the borrowing-strength technique from Wang et al. (2001); Huang and Wang (2004), β can be estimated by solving

$$U_{2n}(b) = \frac{1}{n}\sum_{i=1}^{n}\Delta_i\left\{X_i - \frac{\sum_{j=1}^{n} X_j Z_j I[Y_j^*(b) \geq Y_j^*(b)]}{\sum_{j=1}^{n} Z_j I[Y_j^*(b) \geq Y_j^*(b)]}\right\},$$

where Z_i is estimated by

$$\widehat{Z}_i = \frac{m_i}{\widehat{\Lambda}_0(Y_i^*(\widehat{\alpha}))}.$$

4. Lastly, the baseline hazard function can be estimated via

$$\widehat{H}_0(t,b) = \sum_{i=1}^{n}\frac{\Delta_i I[Y_i^*(\hat{\beta}) \leq t]}{\sum_{j=1}^{n}\hat{Z}_i I[Y_j^*(\hat{\beta}) \geq Y_i^*(\hat{\beta})]}.$$

reReg() with method = "am.XCHWY"

- The reReg() fit gives:

```
> fit <- reReg(Recur(Time, id, event, status) ~ x1 + x2, data = datAM,
+              method = "am.XCHWY")
> summary(fit)
Call: reReg(formula = Recur(Time, id, event, status) ~ x1 + x2, data = datAM,
    method = "am.XCHWY")

Method: Xu et al. (2016) Model

Coefficients (rate):
   Estimate StdErr z.value p.value
x1    1.180     NA      NA      NA
x2   -1.091     NA      NA      NA

Coefficients (hazard):
   Estimate StdErr z.value p.value
x1    0.334     NA      NA      NA
x2   -1.088     NA      NA      NA
```

`reReg()` with `method = "am.XCHWY"`

- The key feature of the work of Xu et al. (2017) is similar to that of the Huang and Wang (2004)
- Advantage:
  - A joint model that allows terminal event
  - Allows informative censoring
  - A resampling based variance estimation can be adopted to facilitate variance estimation
  - The estimation procedure does not require the strong Poisson assumption.
  - Easy interpretation
- Limitation:
  - The variance estimation with bootstrap procedure could be slow when sample size is very large

`reReg()` with `method = "sc.XCYH"`

- When `method = "sc.XCYH"`, `reReg()` gives the methods proposed in Xu et al. (2019).
- Xu et al. (2019) models the rate function of the recurrent event process via a scale-change model

$$\lambda_i(t) = Z_i \lambda_0(te^{X_i^\top \alpha})e^{X_i^\top \beta}, t \in [0, \tau].$$

- The model reduces to the following special cases:
  - Cox model when $\alpha = 0$
  - Accelerated rate model when $\beta = 0$
  - Accelerated mean model when $\alpha = \beta$
- Allows model selection through testing $H_o : \alpha = 0$, $H_o : \beta = 0$, and $H_o : \alpha = \beta$.

`reReg()` with `method = "sc.XCYH"`

- The estimation procedure is based on the transformed time; similar to that in Xu et al. (2017).
- Consider the transformation $t_{ij}^* = t_{ij} e^{X_i^\top \alpha}$, $Y_i^* = Y_i e^{X_i^\top \alpha}$
- The estimation procedure consists of the following steps:
  1. The regression coefficient α can be estimated by solving

$$\frac{1}{n}\sum_{i=1}^{n}\int_0^\infty \left\{ X_i - \frac{\sum_{j=1}^n X_j \sum_{j=1}^{m_i} I\{t_{ij}^* \le t \le Y_i^*\}}{\sum_{j=1}^n \sum_{j=1}^{m_i} I\{t_{ij}^* \le t \le Y_i^*\}} \right\} dN_i^*(t) = 0.$$

  2. The baseline hazard function can be estimated via $\widehat{\lambda}_0(t) = \exp\{\widehat{H}(t)\}$, where

$$\widehat{H}_n(t, a) = -\int_t^\infty \frac{\sum_{i=1}^n dN_i^*(u)}{\sum_{i=1}^n \sum_{j=1}^{m_i} I\{t_{ij}^* \le t \le Y_i^*\}}.$$

`reReg()` with `method = "sc.XCYH"`

3. Lastly, the regression coefficient β is estimated by solving

$$n^{-1}\sum_{i=1}^{n} X_i \left[m_i \widehat{\Lambda}_n^{-1}\{Y_i^*(\widehat{\alpha}_n)\} - e^{X_i^\top(\beta-\widehat{\alpha}_n)} \right] = 0,$$

where $\widehat{\alpha}_n$ is obtained from Step 1.

reReg() with method = "sc.XCYH"

- Since the scale-change model includes the Cox model and the accelerated mean model as special cases, it is expected to have a low bias when applying it to datCox and datAM that we generated before.
- Applying to datCox

```
> fit1 <- reReg(Recur(Time, id, event, status) ~ x1 + x2, data = datCox,
+               method = "sc.XCYH")
> summary(fit1)
Call: reReg(formula = Recur(Time, id, event, status) ~ x1 + x2, data = datCox,
    method = "sc.XCYH")

Method: Generalized Scale-Change Model

Scale-change effect:
   Estimate StdErr z.value p.value
x1   -0.125     NA      NA      NA
x2   -0.006     NA      NA      NA

Multiplicative coefficients:
   Estimate StdErr z.value p.value
x1    1.088     NA      NA      NA
x2   -1.007     NA      NA      NA
```

reReg() with method = "sc.XCYH"

- Applying to datAM

```
> fit2 <- reReg(Recur(Time, id, event, status) ~ x1 + x2, data = datAM,
+               method = "sc.XCYH")
> summary(fit2)
Call: reReg(formula = Recur(Time, id, event, status) ~ x1 + x2, data = datAM,
    method = "sc.XCYH")

Method: Generalized Scale-Change Model

Scale-change effect:
   Estimate StdErr z.value p.value
x1    0.955     NA      NA      NA
x2   -1.009     NA      NA      NA

Multiplicative coefficients:
   Estimate StdErr z.value p.value
x1    1.055     NA      NA      NA
x2   -1.046     NA      NA      NA
```

`reReg()` with `method = "sc.XCYH"`

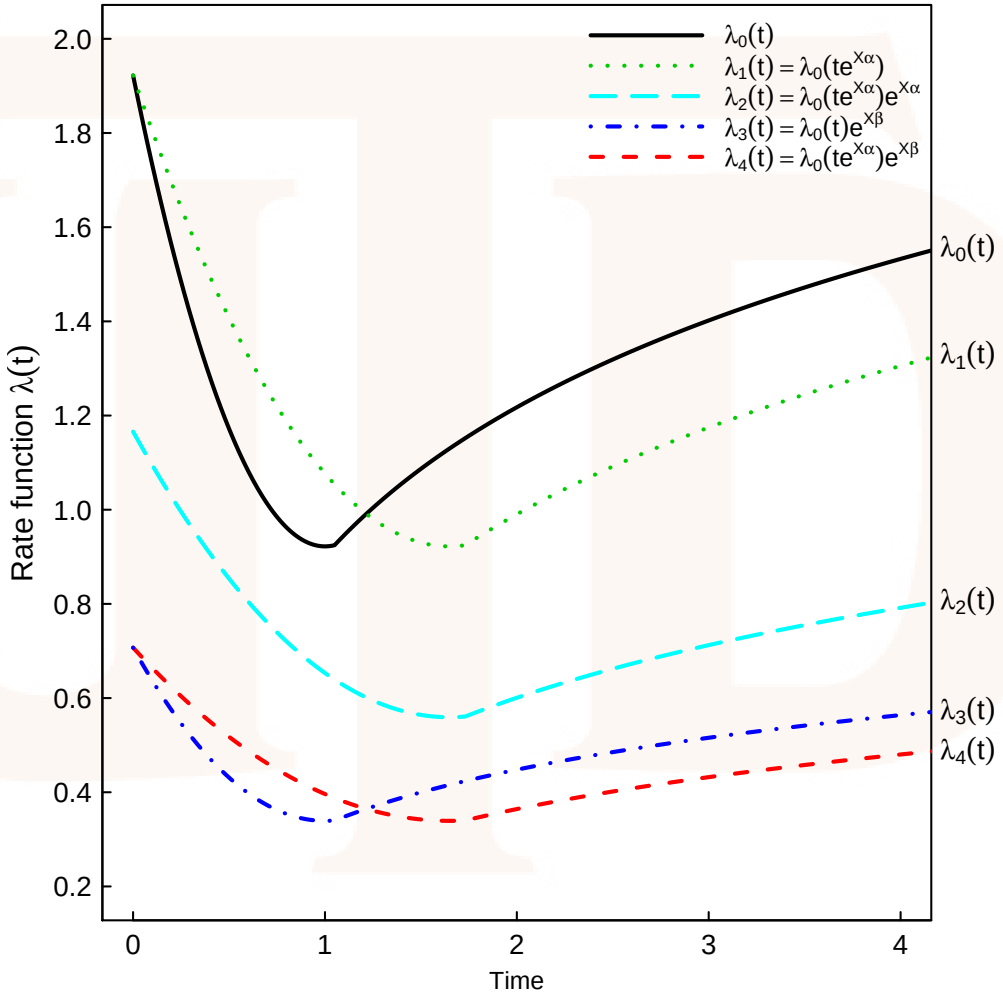
- The key feature of the work of Xu et al. (2017) is similar to that of the Huang and Wang (2004)
- Advantage:
  - More robust to model misspecification
  - Model selection via hypothesis test
- Limitation:
  - Bias-variance trade off
  - Can be extended to a joint model setting

`reReg()`; Interpretation

- The interpretation of the covariate effect in the scale-change model involves two types of modification on the rate function:
  - a scale-change effect that alters the time scale by a factor of $e^{X_i^\top \alpha}$.
  - a multiplicative effect that modifies the magnitude of the rate function by a factor of $e^{X_i^\top \beta}$.

`reReg()`; Interpretation

- Consider hypothetical rate functions with both α and β negative.



`reReg()`; Variance estimation and control

- The variance estimation is controlled by the argument `se`.
- Currently, there are three options:

  NULL variance estimation will not be performed. This is equivalent to setting `B = 0`

  bootstrap performs nonparametric bootstrap

  resampling performs the efficient resampling-based sandwich estimator.
- Generally speaking, the resampling-based sandwich estimator is faster than the bootstrap approach, as it does not require solving estimating equations repeatedly.

`reReg()`; Variance estimation and control

- The `control` list consists of the following parameters:

  `tol` absolute error tolerance used in solving estimating equations; default at 0.001.

  `a0` & `b0` initial guesses used for root search; default at zero vectors.

  `solver` the equation solver used for root search. The available options are `BB::BBsolve` (default), `BB::dfsane`, `BB:BBoptim`, and `optim`.

  `parallel` is an logical value indicating whether parallel computing will be applied when `se = "bootstrap"`.

  `parCl` is an integer value specifying the number of CPU cores to be used when "parallel = TRUE". The default value is half the CPU cores on the current host.

reReg(); Variance estimation and control

- Here is an example that shows the advantage of parallel computing:

```
> system.time(fit1 <- reReg(Recur(Time, id, event, status) ~ x1 + x2,
+                           data = datCox, B = 50, se = "b", method = "cox.HW",
+                           control = list(parallel = FALSE)))
   user  system elapsed
 91.968   0.000  91.980
> parallel::detectCores()
[1] 8
> system.time(fit2 <- reReg(Recur(Time, id, event, status) ~ x1 + x2,
+                           data = datCox, B = 50, se = "b", method = "cox.HW",
+                           control = list(parallel = TRUE, parCl = 8)))
   user  system elapsed
 25.024   0.144  43.389
```

`reReg()`; Variance estimation and control

- The parallel computing does improve the overall computational speed, but it is not the only factor.

```
> summary(fit1)
Call: reReg(formula = Recur(Time, id, event, status) ~ x1 + x2, data = datCox,
    B = 50, method = "cox.HW", se = "b", control = list(parallel = FALSE))

Method: Huang-Wang Model

Coefficients (rate):
   Estimate StdErr z.value   p.value
x1    1.143  0.105  10.895 < 2.2e-16 ***
x2   -1.016  0.047 -21.576 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Coefficients (hazard):
   Estimate StdErr z.value   p.value
x1    0.893  0.163   5.480 < 2.2e-16 ***
x2   -0.984  0.080 -12.333 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- `summary(fit2)` gives similar results.

`reReg()`; summary

- Currently, the `reReg()` allows users to choose from one of the six methods `"cox.LWYY"`, `"cox.GL"`, `"cox.HW"`, `"am.GL"`, `"am.XCHWY"`, `"sc.XCYH"`.
- These methods can be divided based on the presence of the terminal events and the ability to handle informative censoring.

`reReg()`; new features coming up

- Informative censoring exists in many medical applications, e.g., informed dropouts, patient visits in a as-needed basis.
- Approaches that accounts for informative censoring is more appropriate (Huang and Wang, 2004; Xu et al., 2017, 2019).
- The borrowing-strength technique (Wang et al., 2001) made these approach even more appropriate by not requiring a parametric assumption on the frailty variable used to account for informative censoring.
- Our next step is to extend the method `sc.XCYH` to a joint model setting, that will eventually allow users to choose the underlying model assumption for the rate function and the hazard function for the recurrent process and the terminal event, respectively.

Online resource

- Updates on the `reReg` will be posted at `www.sychiou.com/reReg/`

reReg 1.2.1 | Reference | Articles | Changelog

reReg

Regression models for recurrent event data

reReg implements a collection of regression models for recurrent event process and failure time. The package is still under active development.

Installation

You can install and load **reReg** from CRAN using

```
install.packages("reReg")
library(reReg)
```

You can install reReg from github with:

```
## install.packages("devtools")
devtools::install_github("stc04003/reReg")
```

Online documentation

Online document includes:

- Package vignette on visualization of recurrent event data.
- Package vignette on simulating recurrent event data.
- Package vignette on regression analysis for recurrent event data (still under development).

Links

Download from CRAN at
https://cloud.r-project.org/package=reReg

Browse source code at
https://github.com/stc04003/reReg

Report a bug at
http://github.com/stc04003/reReg/issues

License

GPL (>= 3)

Developers

Sy Han (Steven) Chiou
Author, maintainer

Chiung-Yu Huang
Author

Dev status

repo status Active
R>= 3.4.0
CRAN 1.2.1
Package version 1.2.0
build passing
build passing
last change 2020-06-11

Online resource

- Updates on the `reda` are posted at `www.wenjie-stat.me/reda/`

reda 0.5.1.9000 | Reference | Articles | Changelog

reda

Overview

The R package **reda** provides functions for

- simulating survival, recurrent event, and multiple event data from stochastic process point of view;
- exploring and modeling recurrent event data through the mean cumulative function (MCF) by the Nelson-Aalen estimator of the cumulative hazard rate function, and gamma frailty model with spline rate function;
- comparing two-sample recurrent event responses with the pseudo-score tests.

Installation

You can install the released version from CRAN.

```
install.packages("reda")
```

Getting Started

Online documentation provides function documentations and includes package vignettes for

- exploring and modeling recurrent event data.
- introduction to formula response function Recur()
- simulating survival and recurrent event data.

Links

Download from CRAN at
https://cloud.r-project.org/package=reda

Browse source code at
https://github.com/wenjie2wang/reda

Report a bug at
https://github.com/wenjie2wang/reda/issues

License

GPL (>= 3)

Citation

Citing reda

Developers

Wenjie Wang
Author, maintainer

Haoda Fu
Author

All authors...

Dev status

CRAN 0.5.0

build passing

Instructor's contact information

- If you have any questions (especially if you spot bugs), please feel free to contact me at `schiou@utdallas.edu`.
- You can also find relevant works from my website `www.sychiou.com`

Sy Han (Steven) Chiou

Assistant Professor of Statistics

The University of Texas at Dallas

cv

About me

I am an assistant professor in the Department of Mathematical Sciences at the University of Texas at Dallas (UTD). I joined UTD in 2017 after being a postdoctoral research fellow in the Department of Biostatistics at the Harvard T.H. Chan School of Public Health during 2015-2017 and an assistant professor in the Department of Mathematicas at the University of Minnesota Duluth during 2013-2015.

I am an International Statistical Institute (ISI) Elected member.

I am also the faculty advisor of the UTD Go Association.

Download my CV.

Interests

- Survival analysis
- Statistical computing
- Enivironment applications
- Public health applications

Education

Ph.D. in Statistics, 2013
University of Connecticut

B.S. in Statistics, 2008
University of Connecticut

B.S. in Applied Mathematics, 2008
University of Connecticut

Many many thank you

- Shanxi University of Finance and Economics for providing this platform to present my package
- The planning committee for organizing the conference during this unprecedented time
- My collaborators for supporting me with the projects:
  - Prof. Chiung-Yu Huang, University of California, San Francisco
  - Prof. Gongjun Xu, University of Michigan
  - Prof. Jun Yan, University of Connecticut

Reference

References

Claggett, B., Pocock, S., Wei, L., Pfeffer, M. A., McMurray, J. J., and Solomon, S. D. (2018). Comparison of time-to-first event and recurrent-event methods in randomized clinical trials. *Circulation* **138,** 570–577.

Cook, R. J. and Lawless, J. (2007). *The statistical analysis of recurrent events*. Springer Science & Business Media.

Ghosh, D. and Lin, D. (2003). Semiparametric analysis of recurrent events data in the presence of dependent censoring. *Biometrics* **59,** 877–885.

Ghosh, D. and Lin, D. Y. (2002). Marginal regression models for recurrent and terminal events. *Statistica Sinica* pages 663–688.

González, J. R., Fernandez, E., Moreno, V., Ribes, J., Peris, M., Navarro, M., Cambray, M., and Borràs, J. M. (2005). Sex differences in hospital readmission among colorectal cancer patients. *Journal of Epidemiology & Community Health* **59,** 506–511.

Huang, C.-Y. and Wang, M.-C. (2004). Joint modeling and estimation for recurrent event processes and failure time data. *Journal of the American Statistical Association* **99,** 1153–1165.

Lin, D. Y., Wei, L.-J., Yang, I., and Ying, Z. (2000). Semiparametric regression for the mean and rate functions of recurrent events. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **62,** 711–730.

Pepe, M. S. and Cai, J. (1993). Some graphical displays and marginal regression analyses for recurrent failure times and time dependent covariates. *Journal of the American statistical Association* **88,** 811–820.

Rondeau, V., Gonzalez, J. R., Mazroui, Y., Mauguen, A., Diakite, A., Laurent, A., Lopez, M., Król, A., and Sofeu, C. L. (2019). *frailtypack: General Frailty Models: Shared, Joint and Nested Frailty Models with Prediction; Evaluation of Failure-Time Surrogate Endpoints*. R package version 3.0.3.

Self, S. G., Prentice, R. L., et al. (1982). Commentary on andersen and gill's" cox's regression model for counting processes: A large sample study". *The Annals of Statistics* **10,** 1121–1124.

Therneau, T. M. (2020). *A Package for Survival Analysis in R*. R package version 3.1-12.

Wang, M.-C., Qin, J., and Chiang, C.-T. (2001). Analyzing recurrent event data with informative censoring.

Xu, G., Chiou, S. H., Huang, C.-Y., Wang, M.-C., and Yan, J. (2017). Joint scale-change models for recurrent events and failure time. *Journal of the American Statistical Association* **112,** 794–805.

Xu, G., Chiou, S. H., Yan, J., Marr, K., and Huang, C.-Y. (2019). Generalized scale-change models for recurrent event processes under informative censoring. *Statistica Sinica* .