A Quick Guide for the recurrentR Package

Wush Wu[1,2]

[1]National Taiwan University,
Taipei, Taiwan

[2]Taiwan R User Group,
Taipei, Taiwan

Contents

Acknowledgement

I want to thank Chiung-Yu Huang for supporting this work. I also thank Hsiang Yu for testing the package.

1. Introduction

The package **recurrentR** is encouraged by Chiung-Yu Huang since May, 2013. We want to provide a fast implementation of Wang *et al.* (2001); Huang and Wang (2004); Huang *et al.* (2010) to the researchers who are interesting in semi-parametric recurrent data analysis. They studied how to estimate the recurrence process and survival function without assuming the independence between the recurrent process and the failure rate.

2. Installation

The **recurrentR** depends on the **Rcpp** and the **nleqslv**. Please install them before the installation of **recurrentR**.

One can install **recurrentR** from github at `https://github.com/wush978/recurrentR`, and the intallation can be done with the following commands

```
library(devtools)
install_github(repo='recurrentR', username='wush978',
               subdir='recurrentR', build_vignettes = FALSE)
```

3. Data Import

In the **recurrentR**, the imported dataset is stored in a S4 object 'recurrent-data'. Without loss of generality, we assume that there is a dataset of recurrent event data containing n instances. Each instance, i, must include the following information.

Notation 1 (Notation)**.** *The following data are required in the* ***recurrentR****:*

1. *Failure or censored time,* y_i*. The failure time is the termination of the recurrent process of instance* i *(failure time). The censored time is the termination of the observation.*
2. *Indicator,* D_i*.* D_i *indicates whether the* y_i *is failure time or not.*
3. *The time period of the research interest:* $[0, T_0]$*.*
4. *Time of the recurrent events,* $t_{i,1}$*,* $t_{i,2}$*, ...,* t_{i,m_i}*. These are the realization of poisson process* $N_i(.)$ *whose cumulative intensity function* $\Lambda_i(.)$ *is estimated by the* ***recurrentR****.*
5. *q-dim vector of time independent covariates,* W_i*. We assume that* $W_i \in \mathbb{R}^{q \times 1}$*. For simplicity, we denote* $W \in \mathbb{R}^{q \times n}$ *as the corresponding matrix for all instances.*
6. *p-dim time-dependent covariate process,* $X_i(t)$*. We assme that* $X_i(t) \in \mathbb{R}^{p \times 1}$*. This quantity is only used in Huang* et al. *(2010).*

The **recurrentR** provides a S3 method 'create_recurrent_data' to directly construct the 'recurrent-data' object.

3.1. Dataset MMC in the survrec

	id	time	event	group
1	1.00	112.00	1.00	Males
2	1.00	145.00	1.00	Males
3	1.00	39.00	1.00	Males
4	1.00	52.00	1.00	Males
5	1.00	21.00	1.00	Males
6	1.00	34.00	1.00	Males
7	1.00	33.00	1.00	Males
8	1.00	51.00	1.00	Males
9	1.00	54.00	0.00	Males

Table 1: This is the head 20 rows of the Migratory Motor Complex data from **survrec**.

Here is the ‘MMC‘ data from **survrec**. Since the MMC puts all event times into a single data.frame, we take the instance with ‘id == 1‘ as an example (Table. 1).

1. The ‘id‘ indicates the row is corresponding to which instance. Here we list all rows with ‘id‘ 1.
2. The ‘time‘ indicates the recurrence or censoring time. This is the difference between each recurrence or censoring.
3. The ‘event‘ indicates the time is recurrence or censoring.
4. The ‘group‘ is the covariates of the instance.

The numeric vector 112, 145, 39, 52, 21, 34, 33, 51, 54 is the difference of recurrence and censoring, so its ‘cumsum‘, which is 112, 257, 296, 348, 369, 403, 436, 487, 541, is the recurrence and censoring time(the last one). Similarly, we could extract all recurrence and censoring time for all instance and put them seperately into a ‘list‘ of ‘numeric vector‘ and a ‘numeric vector‘. Note that there is no failure time in ‘MMC‘ dataset.

The API ‘create_recurrent_data‘ provides an helper function to quickly construct the ‘recurrent-data‘ for you:

```
str(recurrentR:::create_recurrent_data.data.frame)

## function (src, id, time, time_type = c("absolutely", "relatively"),
##     indicator, indicator_value, covariate, T_0 = NULL)
```

1. The parameter ‘src‘ is the input data.frame.
2. The parameter ‘id‘ is the column name of ‘src‘ to specify the instance. Here, the ‘id‘ should be ‘"id"‘.
3. The parameter ‘time‘ is the column name of ‘src‘ to specify the even time. Here, the ‘time‘ should be ‘"time"‘.

4. The parameter ‘time_type‘ is the type of the value recorded in ‘time‘. If the value is ‘”absolutely”‘, then the ‘time‘ is direct realization of the recurrent process. If the value is ‘”relatively”‘, the time is the length between two continuous events. Here, the ‘time_type‘ should be ‘”relatively”‘.

5. The parameter ‘indicator‘ is the column name of ‘src‘ to specify the event type. In the **recurrentR**, there are three types of events: ”recurrence”, ”censoring” and ”failure”. The ”recurrence” represents the sample from the recurrent process. The ”censoring” represents the termination time of the observation where the failure has not been occurred. The ”failure” represents the failure time of the instance. Here, the ‘indicator‘ should be ‘”event”‘.

6. The paramter ‘indicator_value‘ is a named list indicate the meaning of the value in the ‘src[[indicator]]‘. Here, the ‘indicator_value‘ should be ‘list(”recurrence” = 1, ”censoring” = 0)‘.

7. The parameter ‘covariate‘ is the column names of ‘src‘ to indicate the covariates. They should be the time independent covariate.

8. The parameter ‘T_0‘ is a numeric value indicate the upper bound of research interest. The default value is the maximal observed failure time or censor time. Sometimes there are some outlier observed failure time or censor time, so the estimated intensity of recurrent process becomes ill-behaved. In such case, using a smaller ‘T_0‘ might help.

```
obj <- create_recurrent_data(head(MMC, 12),
    id = "id", time = "time", time_type = "relatively",
    indicator = "event", indicator_value = list(recurrence = 1,
        censoring = 0), covariate = "group")
obj

## An object of class "recurrent-data"
## Slot "y":
## [1] 541 383
##
## Slot "D":
## [1] FALSE FALSE
##
## Slot "t":
## $`1`
##   1   2   3   4   5   6   7   8
## 112 257 296 348 369 403 436 487
##
## $`2`
##  10  11
## 206 353
##
##
```

```
## Slot "T_0":
## [1] 541
##
## Slot "W":
##    groupFemales
## 1             0
## 10            0
##
## Slot "X":
## list()
##
## Slot "X_dim":
## [1] 0
##
## Slot "n":
## [1] 2
##
## Slot "eval":
## numeric(0)
##
## Slot "s":
##  [1] 112 206 257 296 348 353 369 403 436 487
##
## Slot "d":
##  [1] 1 1 1 1 1 1 1 1 1 1
##
## Slot "cache":
## <environment: 0x3101f70>
##
## Slot "tol":
## [1] 1e-04
```

The name of slot should be directly mapped to the notations in Notation. 1.

3.2. Dataset colon in the survrec

Here is the ‘colon‘ dataset in the **survrec**. The first 2 instances are shown in Table. 2. To transform the ‘colon‘ to ‘recurrent-data‘, we must specify the following arguments for ‘create_recurrent_data‘:

1. The ‘src‘ should be ‘colon‘.
2. The ‘id‘ should be ‘hc‘ that is the identificator of each subject.
3. The ‘time‘ should be ‘”time”‘.
4. The ‘time_type‘ should be ‘”relative”‘. The ‘cumsum(src[time])‘ should be the realization of the recurrent process.

	hc	time	event	chemoter	dukes	distance
1	5634.00	24.00	1.00	2.00	3.00	1.00
2	5634.00	433.00	1.00	2.00	3.00	1.00
3	5634.00	580.00	0.00	2.00	3.00	1.00
4	10767.00	489.00	1.00	1.00	2.00	1.00
5	10767.00	693.00	0.00	1.00	2.00	1.00

Table 2: This is the first two instances in ‘colon‘ dataset.

5. The ‘indicator‘ should be ‘”event”‘.
6. The ‘indicator_value‘ should be ‘list(”recurrence” = 1, ”censoring” = 0)‘. There is no data related to ”failure”.
7. The ‘covariate‘ should be ‘c(”chemoter”, ”dukes”, ”distance”)‘.

```
obj <- create_recurrent_data(head(colon,
    5), id = "hc", time = "time", time_type = "relatively",
    indicator = "event", indicator_value = list(recurrence = 1,
        censoring = 0), covariate = c("chemoter",
        "dukes", "distance"))
obj

## An object of class "recurrent-data"
## Slot "y":
## [1] 1037 1182
##
## Slot "D":
## [1] FALSE FALSE
##
## Slot "t":
## $`5634`
## [1]  24 457
##
## $`10767`
## [1] 489
##
##
## Slot "T_0":
## [1] 1182
##
## Slot "W":
##   chemoter dukes distance
## 1        2     3        1
## 4        1     2        1
##
## Slot "X":
```

```
## list()
##
## Slot "X_dim":
## [1] 0
##
## Slot "n":
## [1] 2
##
## Slot "eval":
## numeric(0)
##
## Slot "s":
## [1]  24 457 489
##
## Slot "d":
## [1] 1 1 1
##
## Slot "cache":
## <environment: 0x19ecf60>
##
## Slot "tol":
## [1] 1e-04
```

Note that there is an ‘NA‘ in ‘colon[766,]‘, so the user should remove that row before analyzing.

3.3. Dataset readmission in the frailtypack

Here is the readmission dataset in the **frailtypack**. The first two instances are shown in Table. 3.

	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

Table 3: The first two instance in the readmission dataset

1. The ‘src‘ should be ‘readmission‘.
2. The ‘id‘ should be ‘id‘ that is the identificator of each subject.
3. The ‘time‘ should be ‘”t.stop”‘.
4. The ‘time_type‘ should be ‘”absolutely”‘. The ‘readmission$t.stop‘ should be the realization of the recurrent process.

5. The ‘indicator‘ should be created by both column ‘”event”‘ and ‘”death”‘. The value 1 of ‘event‘ represents recurrence. The value 0 of ‘event‘ and 0 of ‘death‘ represents censoring. The value 0 of ‘event‘ and 1 of ‘death‘ represents failure. Therefore, we should create a new column to be the indicator. Here we concatenate the column ‘event‘ and ‘death‘ to produce a new column ‘indicator‘.

6. The ‘indicator_value‘ should be ‘list(”recurrence” = ”1-0”, ”censoring” = ”0-0”, ”failure” = ”0-1”)‘.

7. The ‘covariate‘ should be ‘c(”chemo”, ”sex”, ”dukes”, ”charlson”)‘.

```
readmission$indicator <- paste(readmission$event,
    readmission$death, sep = "-")
which(readmission$indicator == "1-1")

## [1] 119 204 581
```

Note that there are three rows record recurrence and death simultaneously. We modify the ‘death‘ to ‘0‘ on these rows since the following censoring rows record the death as ‘1‘, too.

```
readmission$death[which(readmission$indicator ==
    "1-1")] <- 0
readmission$indicator <- paste(readmission$event,
    readmission$death, sep = "-")
obj <- create_recurrent_data(readmission,
    id = "id", time = "t.stop", time_type = "absolutely",
    indicator = "indicator", indicator_value = list(recurrence = "1-0",
        censoring = "0-0", failure = "0-1"),
    covariate = c("chemo", "sex", "dukes",
        "charlson"))
```

4. Demo

In this section, we will introduce how to analysis a ‘recurrent-data‘ object with the method developed in Wang *et al.* (2001); Huang and Wang (2004); Huang *et al.* (2010).

4.1. Wang *et al.* (2001)

For each instance i, Wang *et al.* (2001) assmues that the occurrence of recurrent event follows a inhomogenous poisson process with the following intensity:

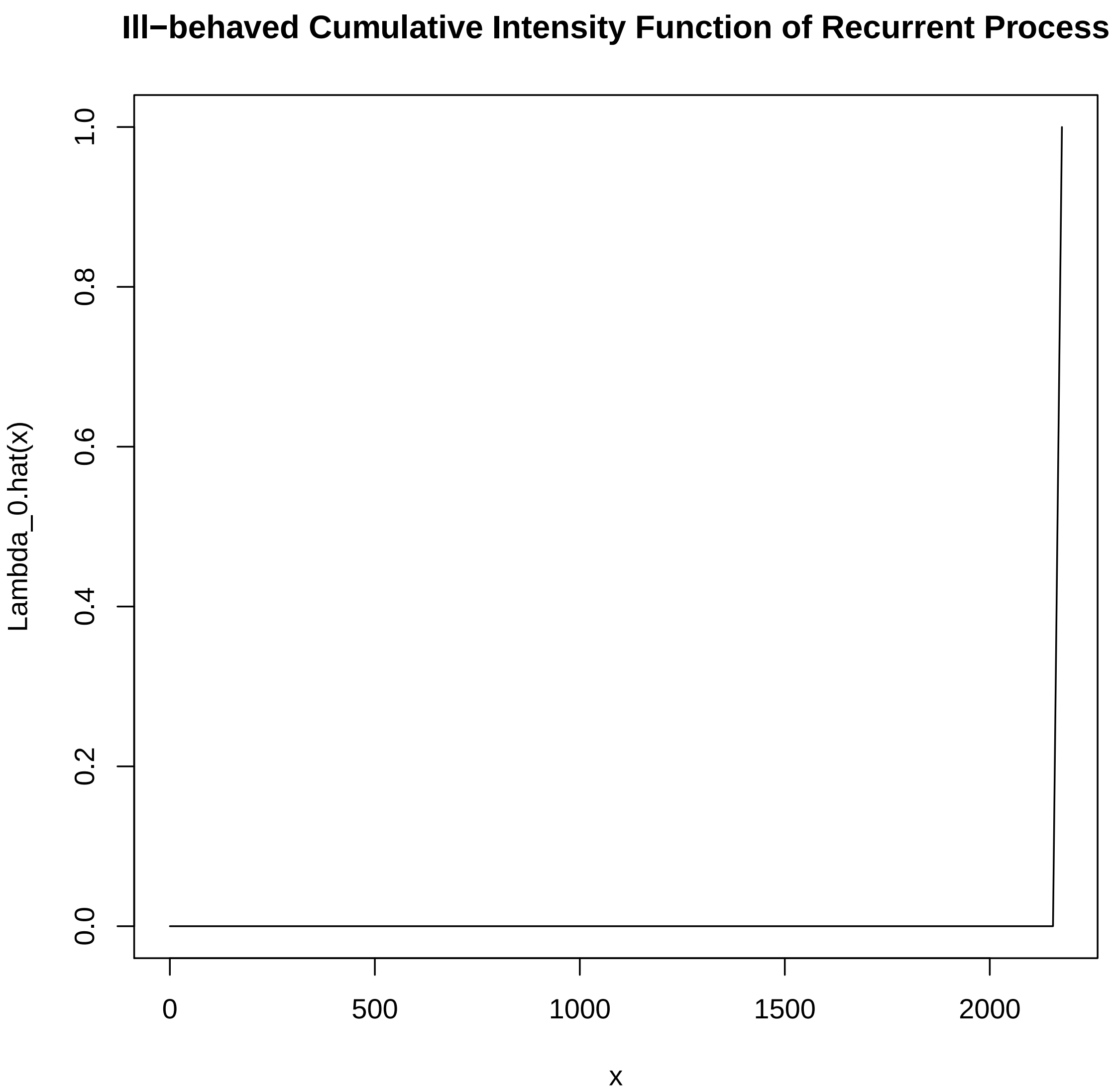
$$\lambda_i(t) = \lambda_0(t) z_i exp(W_i \gamma)$$

where:

- z_i is a nonnegative-valued latent variable such that $E(z_i|W_i) = E(z_i)$. - The baseline intensity function $\lambda_0(t)$ is a probability function: - $\lambda_0(t) \neq 0$ - $\Lambda_0(T_0) = \int_0^{T_0} \lambda_0(u)du = 1$ - γ is a $\mathbb{R}^{1\times q}$ vector.

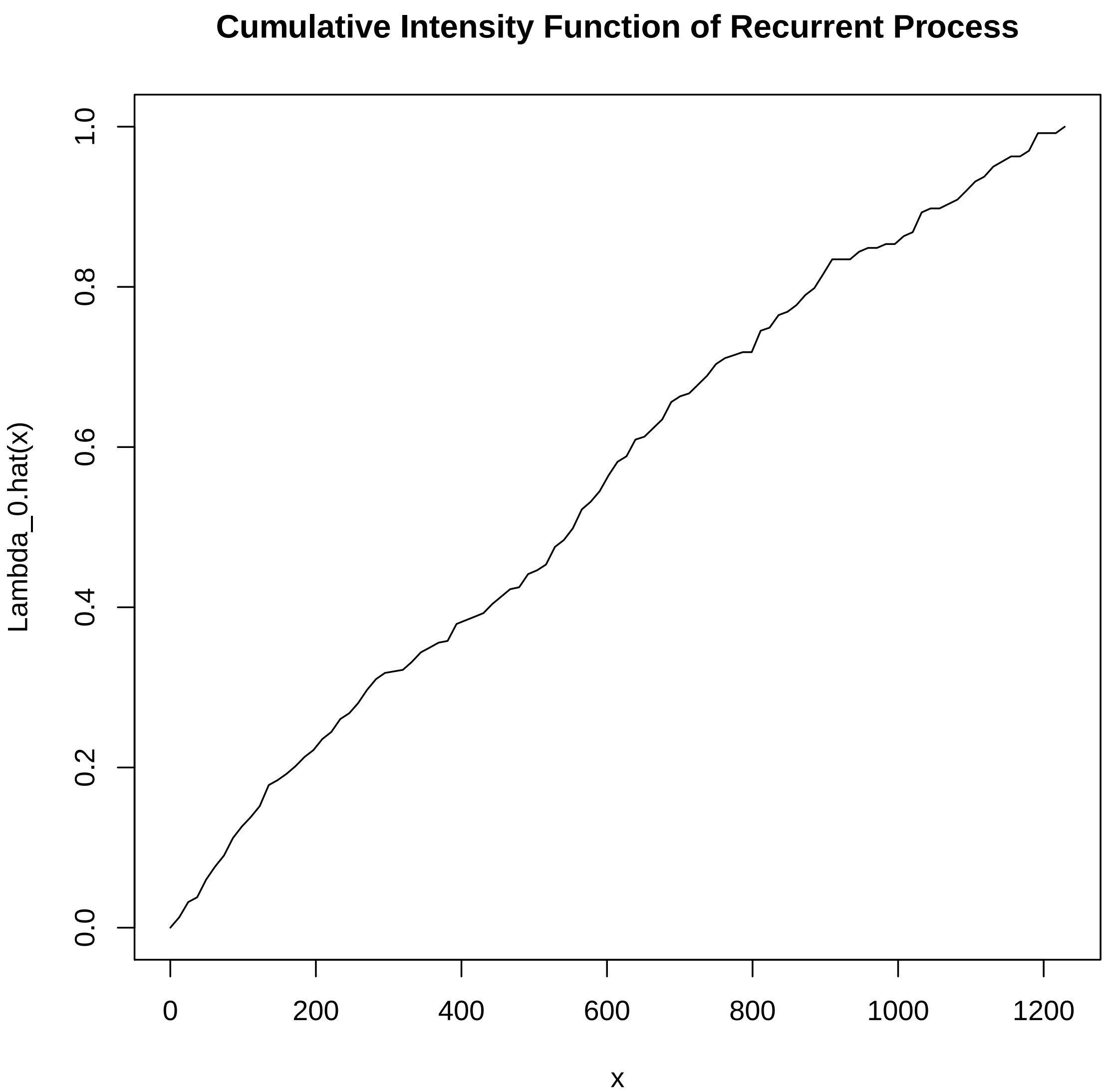
Here we takes the 'recurrent-data' object transformed from the 'readmission' data as indicated in Sec. 3.3:

```
r <- Wang2001(obj)
names(r)

## [1] "Lambda_0.hat"  "gamma.bar.hat"

Lambda_0.hat <- r$Lambda_0.hat
curve(Lambda_0.hat, 0, max(readmission$t.stop), main="Ill-behaved Cumulative Intensity Fun
```



The curve is ill-behaved because there are instance whose recurrent time and failure/censor time is totally larger than others. A possitive correction is to limit the range of research interest. For example, setting the parameter ‘T_0‘ in ‘create_recurrent_data‘ or use the API ‘truncate_T_0‘ to limit the research interest properly. The correction is changing the failure/censor time after new T_0 to censor time at T_0 and ignore the recurrent time after T_0.

```
T_0 <- quantile(readmission$t.stop, 0.75)
obj <- truncate_T_0(obj, T_0)
r <- Wang2001(obj)
Lambda_0.hat <- r$Lambda_0.hat
curve(Lambda_0.hat, 0, T_0, main = "Cumulative Intensity Function of Recurrent Process")
```

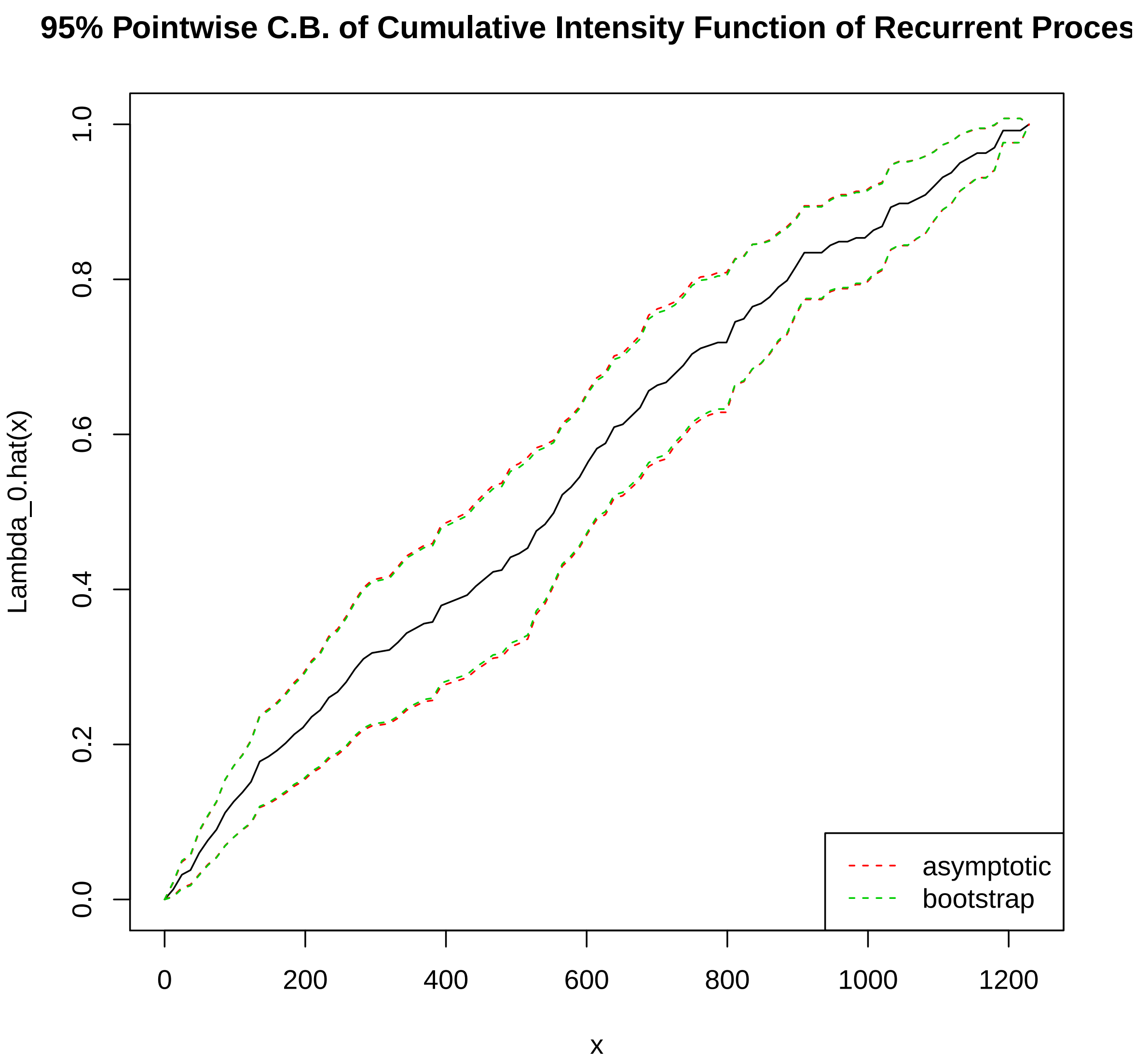


The estimated regression coefficient could be retrieve by:

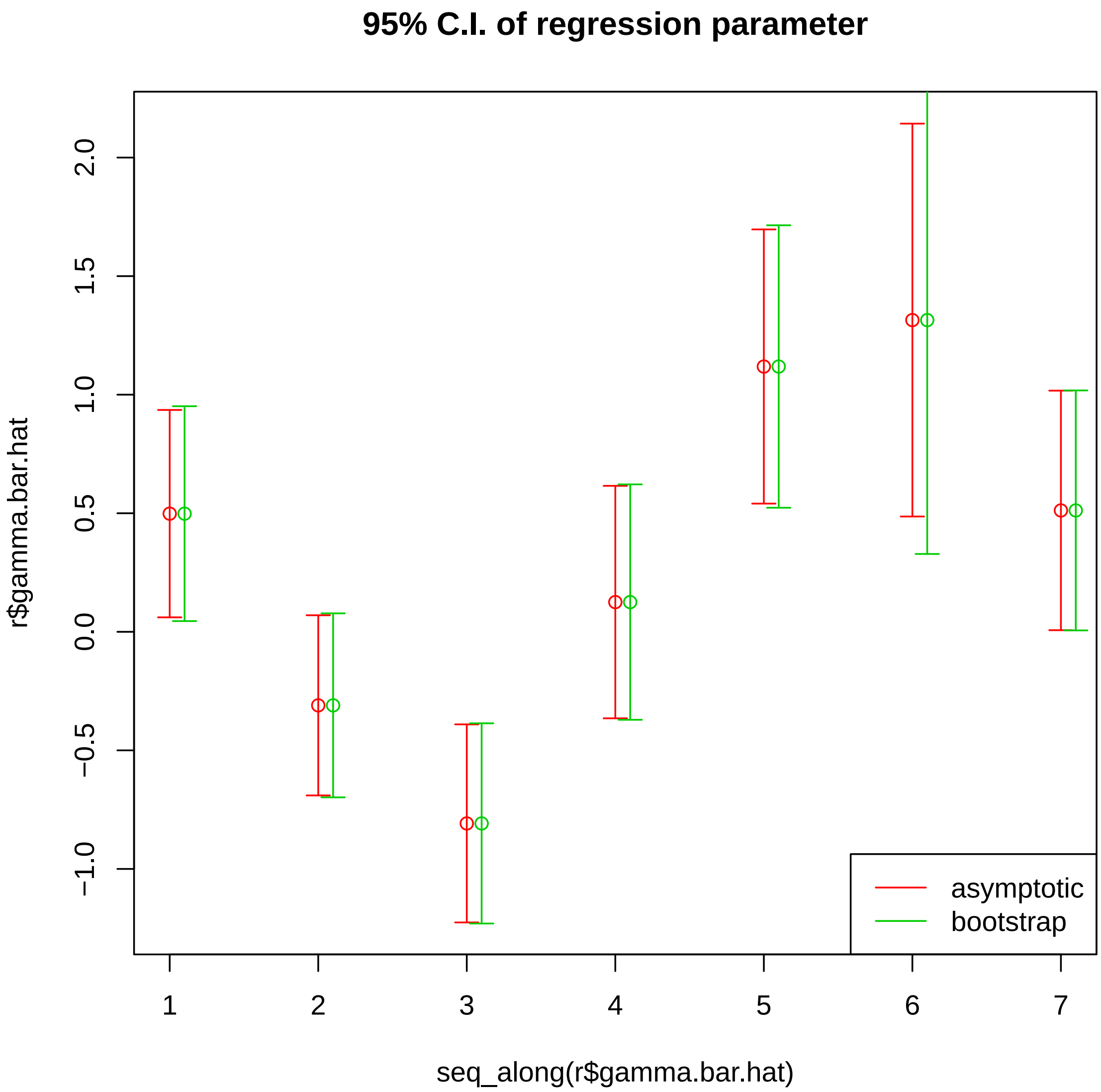
```
r$gamma.bar.hat

##  (Intercept) chemoTreated    sexFemale       dukesC       dukesD
##       0.4983      -0.3102      -0.8081       0.1254       1.1188
##  charlson1-2    charlson3
##       1.3147       0.5119
```

To estimate the variance of the regression coefficient and the recurrent process, the user should specify the methodology. We provide '"asymptotic"' estimation and '"bootstrap"' estimation. The '"asymptotic"' is the plug-in estimator based on the appendix in Wang *et al.* (2001). The bootstrap estimation is based on the non-parametric bootstrap and the parameter 'B' specifies the number of bootstrapping.

```
r.asym <- Wang2001(obj, methods="asymptotic")
Lambda_0.hat <- r.asym$Lambda_0.hat
curve(Lambda_0.hat, 0, obj@T_0, main="95% Pointwise C.B. of Cumulative Intensity Function
Lambda_0.hat.var <- r.asym$Lambda_0.hat.var
Lambda_0.hat.upper <- function(x) {
  Lambda_0.hat(x) + qnorm(0.975) * sqrt(Lambda_0.hat.var(x))
}
curve(Lambda_0.hat.upper, col = 2, add = TRUE, lty = 2)
Lambda_0.hat.lower <- function(x) {
  Lambda_0.hat(x) - qnorm(0.975) * sqrt(Lambda_0.hat.var(x))
}
curve(Lambda_0.hat.lower, col = 2, add = TRUE, lty = 2)
r.boot <- Wang2001(obj, method="bootstrap", B=1000)
Lambda_0.hat.var <- r.boot$Lambda_0.hat.var
Lambda_0.hat.upper <- function(x) {
  Lambda_0.hat(x) + qnorm(0.975) * sqrt(Lambda_0.hat.var(x))
}
curve(Lambda_0.hat.upper, col = 3, add = TRUE, lty = 2)
Lambda_0.hat.lower <- function(x) {
  Lambda_0.hat(x) - qnorm(0.975) * sqrt(Lambda_0.hat.var(x))
}
curve(Lambda_0.hat.lower, col = 3, add = TRUE, lty = 2)
legend('bottomright', c("asymptotic", "bootstrap"), lty = 2, col = 2:3)
```




```
library(plotrix)
plotCI(seq_along(r$gamma.bar.hat), r$gamma.bar.hat, uiw=qnorm(0.975) * sqrt(diag(r.asym$ga
plotCI(seq_along(r$gamma.bar.hat) + 0.1, r$gamma.bar.hat, uiw=qnorm(0.975) * sqrt(diag(r.b
legend('bottomright', c("asymptotic", "bootstrap"), col = 2:3, lty = 1)
```



4.2. Huang and Wang (2004)

Compared to Wang *et al.* (2001), the model intensity of recurrent process is the same:

$$\lambda_i(t) = \lambda_0(t) z_i exp(W_i \gamma)$$

where:

- z_i is a nonnegative-valued latent variable such that $E(z_i|W_i) = E(z_i)$. - The baseline intensity function $\lambda_0(t)$ is a probability function: - $\lambda_0(t) \neq 0$ - $\Lambda_0(T_0) = \int_0^{T_0} \lambda_0(u) du = 1$ - γ is a $\mathbb{R}^{1 \times q}$ vector.

Moreover, the hazard function of the censor time is modeled as

$$h_i(t) = h_0(t) z_i exp(W_i \alpha)$$

where:

- α is a $\mathbb{R}^{1 \times q}$ vector. - We let $H_0(t) = \int_0^t h_0(u)du$

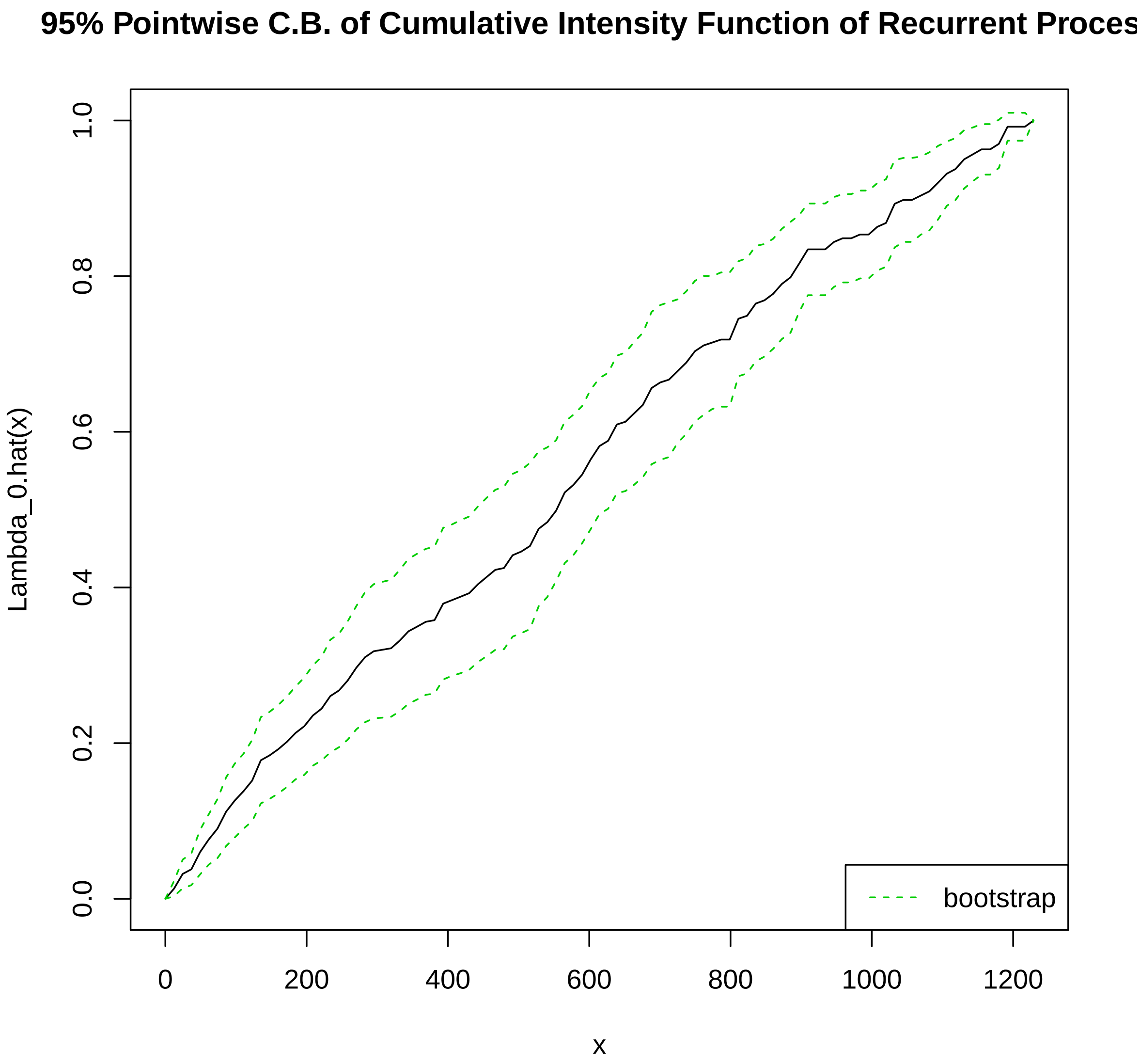
Conditional on (W_i, z_i), $N_i(.)$ and y_i are independent.

The API ‘Huang2004‘ makes the inference based on the model proposed in Huang and Wang (2004). It not only return the estimated $\Lambda_0(t)$, but also privide the estimated hazard function $H_0(t)$ and the regression parameter α:

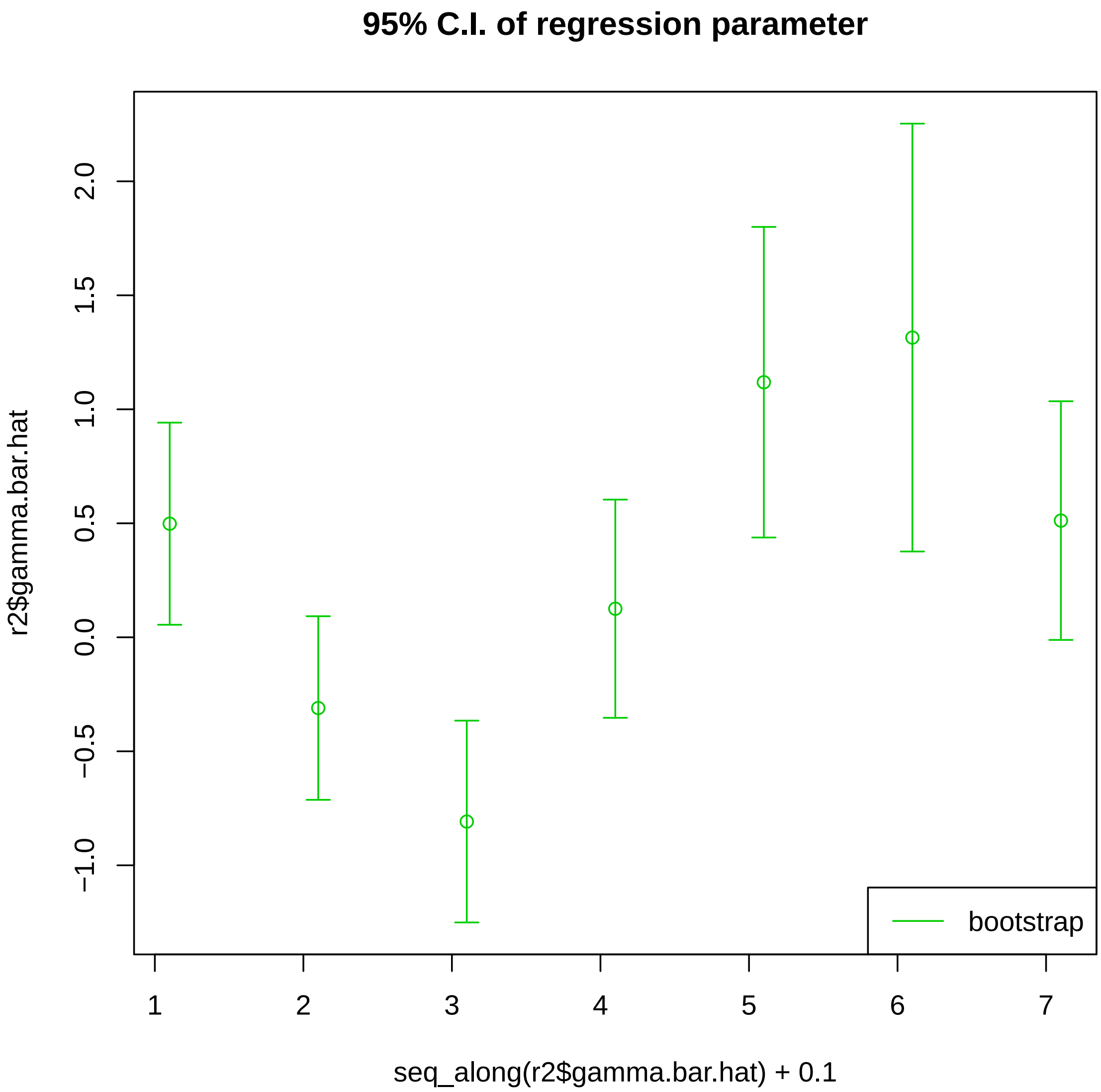
```
r2 <- Huang2004(obj)
names(r2)

## [1] "Lambda_0.hat"  "gamma.bar.hat" "H0.hat"        "alpha.hat"

r2$gamma.bar.hat

##  (Intercept) chemoTreated    sexFemale       dukesC       dukesD
##       0.4983      -0.3102      -0.8081       0.1254       1.1188
##  charlson1-2    charlson3
##       1.3147       0.5119

r2$alpha.hat

## chemoTreated    sexFemale       dukesC       dukesD  charlson1-2
##      0.99812     -0.48074      1.70792      3.79641      0.48987
##    charlson3
##      0.09391
```

Given the parameter of ‘method‘ asks the API ‘Huang2004‘ estimates the variance of the regression coefficient of recurrent process, the intensity of the recurrent process, the coefficient of hazard function and cumulative hazard function. So far we only provide non-parametric bootstrap estimation. The asymptotic estimator is still under development.

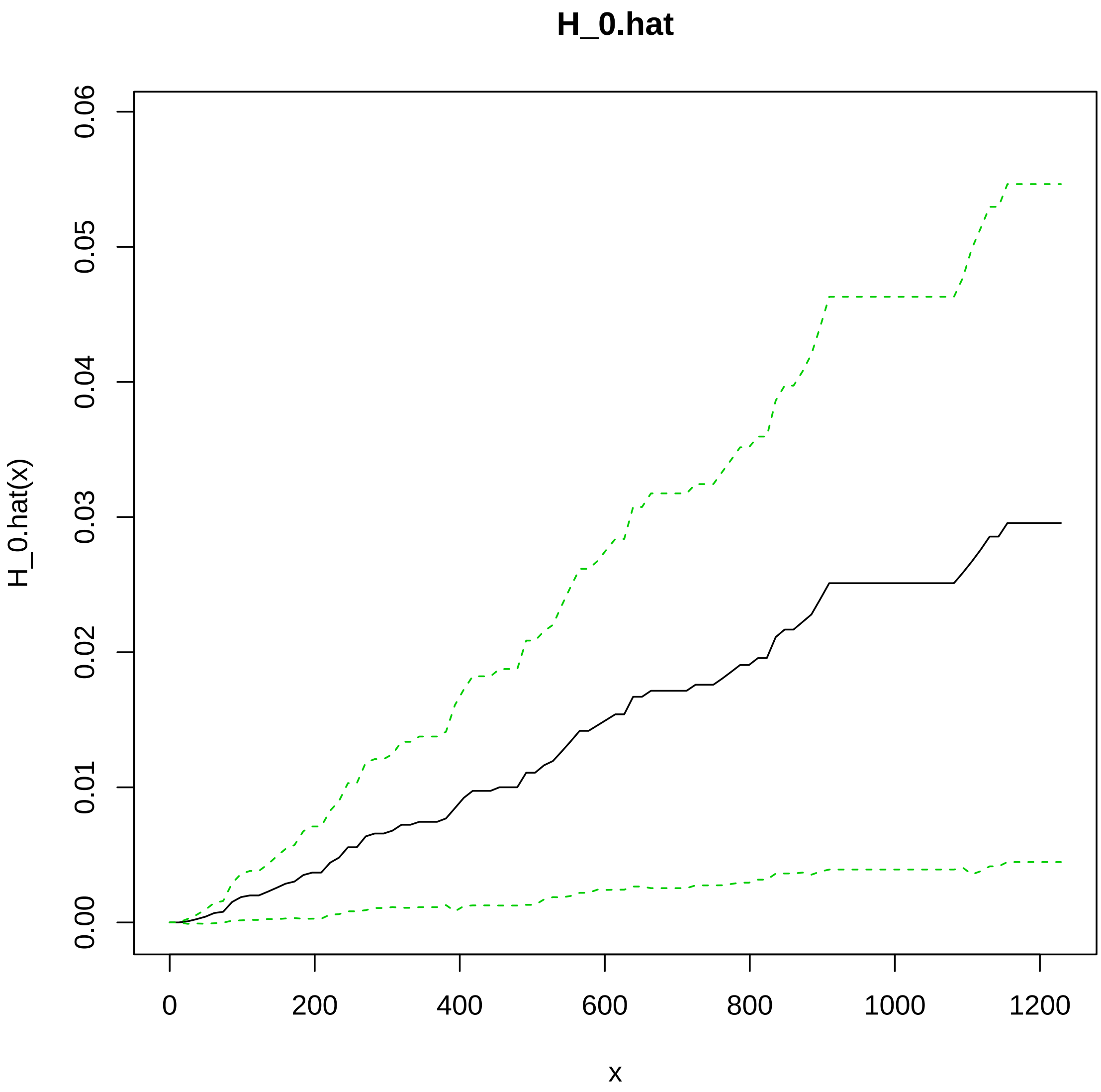
```
  r2.boot <- Huang2004(obj, methods="bootstrap", B = 100)
Lambda_0.hat <- r2.boot$Lambda_0.hat
curve(Lambda_0.hat, 0, obj@T_0, main="95% Pointwise C.B. of Cumulative Intensity Function
Lambda_0.hat.var <- r2.boot$Lambda_0.hat.var
Lambda_0.hat.upper <- function(x) {
  Lambda_0.hat(x) + qnorm(0.975) * sqrt(Lambda_0.hat.var(x))
}
curve(Lambda_0.hat.upper, col = 3, add = TRUE, lty = 2)
Lambda_0.hat.lower <- function(x) {
  Lambda_0.hat(x) - qnorm(0.975) * sqrt(Lambda_0.hat.var(x))
}
curve(Lambda_0.hat.lower, col = 3, add = TRUE, lty = 2)
legend('bottomright', c("bootstrap"), lty = 2, col = 3)
```



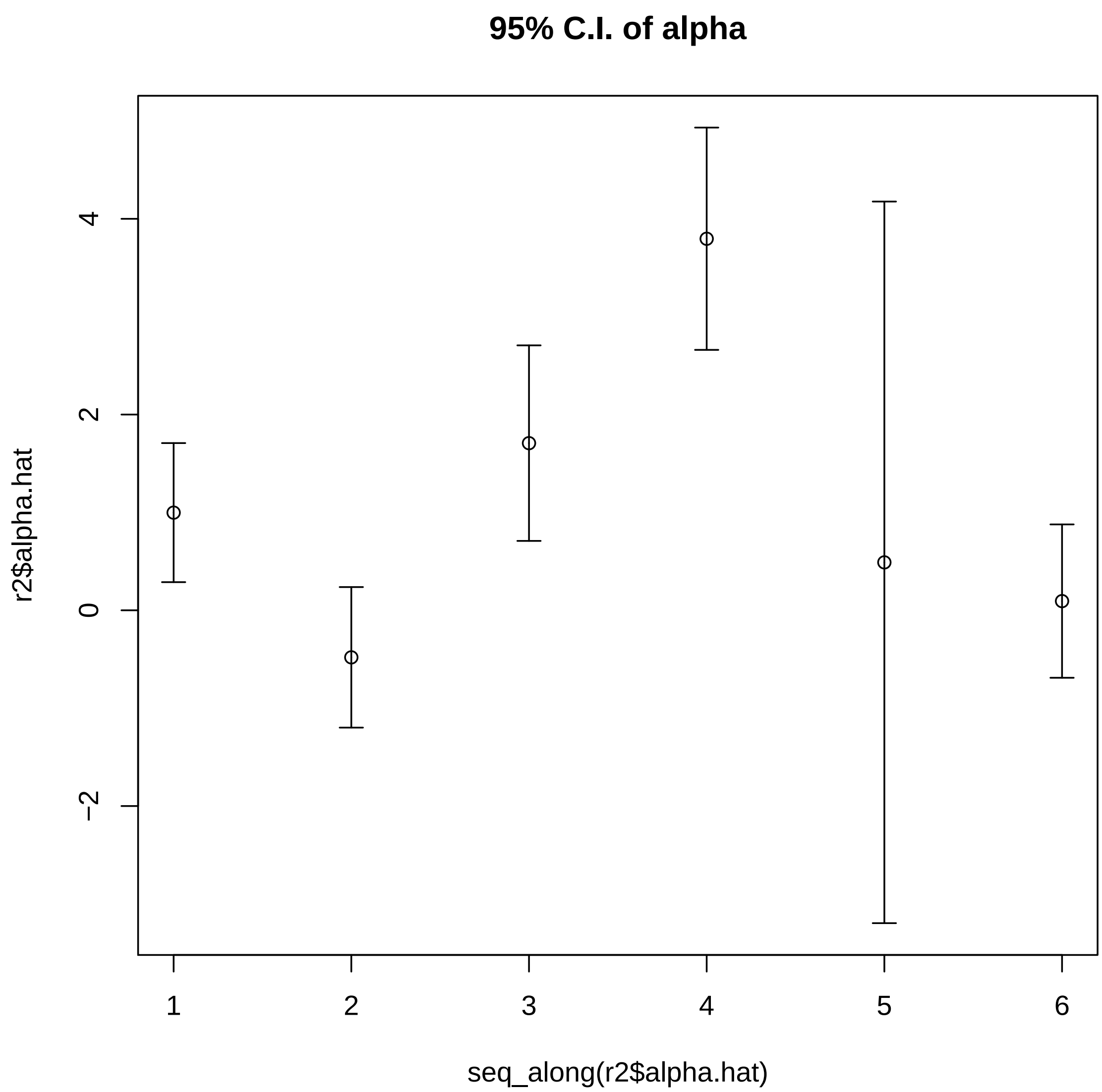

```
library(plotrix)
plotCI(seq_along(r2$gamma.bar.hat) + 0.1, r2$gamma.bar.hat, uiw=qnorm(0.975) * sqrt(diag(r
legend('bottomright', c("bootstrap"), col = 3, lty = 1)
```




```
H_0.hat <- r2$H0.hat
curve(H_0.hat, 0, obj@T_0, ylim = c(0, H_0.hat(obj@T_0)) * 2,
      main = "H_0.hat")
H_0.hat.var <- r2.boot$H0.hat.var
H_0.hat.upper <- function(x) {
  H_0.hat(x) + qnorm(0.975) * sqrt(H_0.hat.var(x))
}
curve(H_0.hat.upper, col = 3, add = TRUE, lty = 2)
H_0.hat.lower <- function(x) {
  H_0.hat(x) - qnorm(0.975) * sqrt(H_0.hat.var(x))
}
curve(H_0.hat.lower, col = 3, add = TRUE, lty = 2)
```




```
library(plotrix)
plotCI(seq_along(r2$alpha.hat), r2$alpha.hat, uiw=qnorm(0.975) * sqrt(diag(r2.boot$alpha.h
```



4.3. Huang *et al.* (2010)

The intensity is:

$$\lambda_i(t) = \lambda_0(t) z_i exp(X_i(t)\beta + \gamma W_i)$$

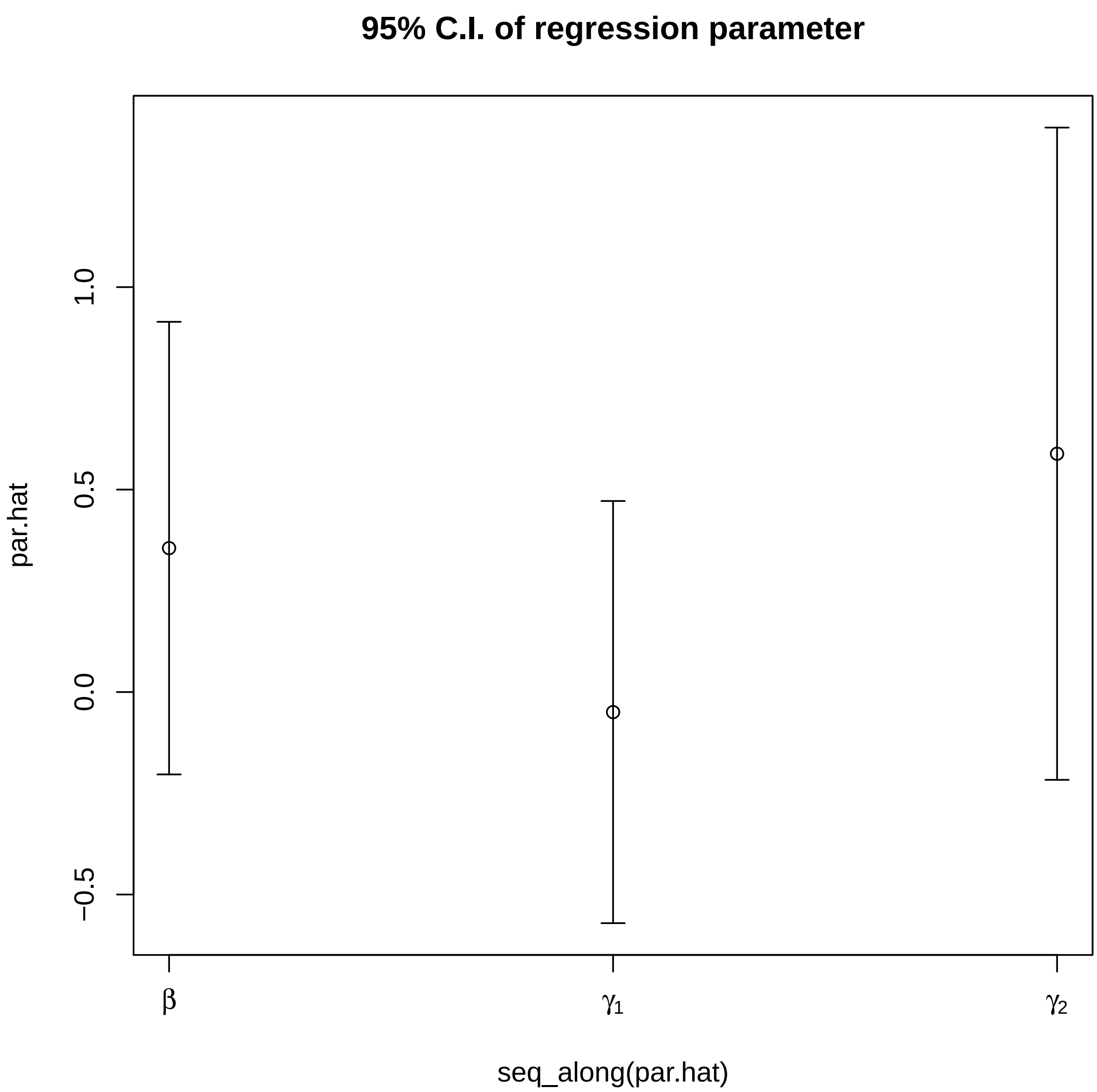
where:

- z_i is a nonnegative-valued latent variable such that $E(z_i|W_i) = E(z_i)$. - The baseline intensity function $\lambda_0(t)$ is a probability function: - $\lambda_0(t) \neq 0$ - $\Lambda_0(T_0) = \int_0^{T_0} \lambda_0(u)du = 1$ - γ is a $\mathbb{R}^{1\times q}$ vector. - β is a $(R)^{1\times p}$ vector.

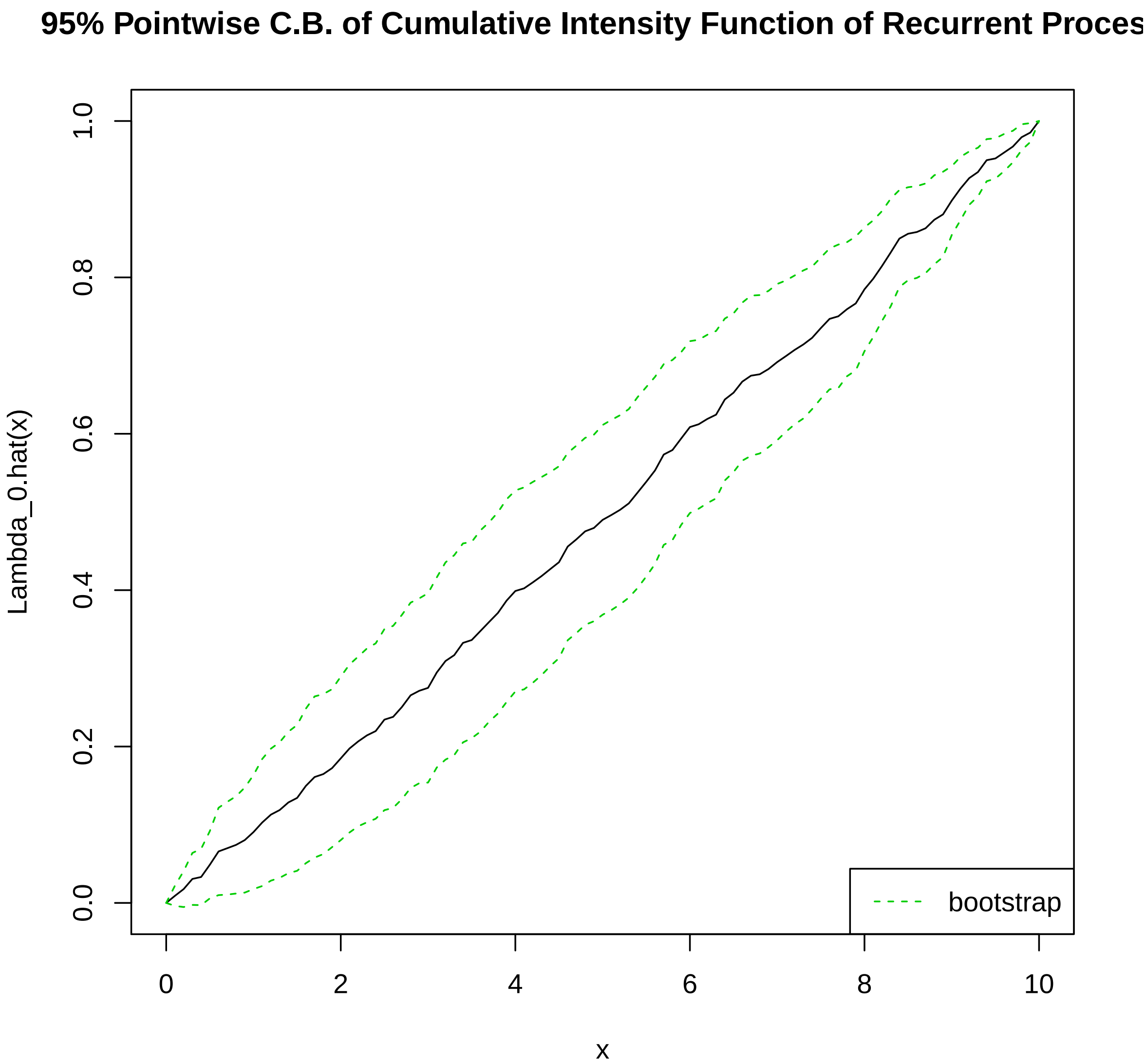
Conditional on (W_i, z_i, X_i), $N_i(.)$ and y_i are independent.

The API 'Huang2010' estimates the Λ_0, γ and β according to the method proposed in Huang *et al.* (2010). To use the API, we must provide the time dependent covariates. We demo an example with synthetic dataset as follow.

```
  data(Huang2010Demo)
r3 <- Huang2010(Huang2010Demo, methods="bootstrap", 100)
# Please wait a while...
names(r3)

## [1] "beta.hat"         "gamma.hat"        "Lambda_0.hat"
## [4] "Lambda_0.hat.var" "gamma.hat.var"    "beta.hat.var"

par.hat <- c(r3$beta.hat, r3$gamma.hat)
par.sd <- c(sqrt(r3$beta.hat.var), sqrt(diag(r3$gamma.hat.var)))
plotCI(seq_along(par.hat), par.hat, uiw = qnorm(0.975) * par.sd, xaxt='n')
axis(side=1, at = 1:3, labels=c(expression(beta), expression(gamma[1]), expression(gamma[2
title(main = "95% C.I. of regression parameter")
```




```
Lambda_0.hat <- r3$Lambda_0.hat
curve(Lambda_0.hat, 0, Huang2010Demo@T_0, main="95% Pointwise C.B. of Cumulative Intensity
Lambda_0.hat.var <- r3$Lambda_0.hat.var
Lambda_0.hat.upper <- function(x) {
  Lambda_0.hat(x) + qnorm(0.975) * sqrt(Lambda_0.hat.var(x))
}
curve(Lambda_0.hat.upper, col = 3, add = TRUE, lty = 2)
Lambda_0.hat.lower <- function(x) {
  Lambda_0.hat(x) - qnorm(0.975) * sqrt(Lambda_0.hat.var(x))
}
curve(Lambda_0.hat.lower, col = 3, add = TRUE, lty = 2)
legend('bottomright', c("bootstrap"), lty = 2, col = 3)
```



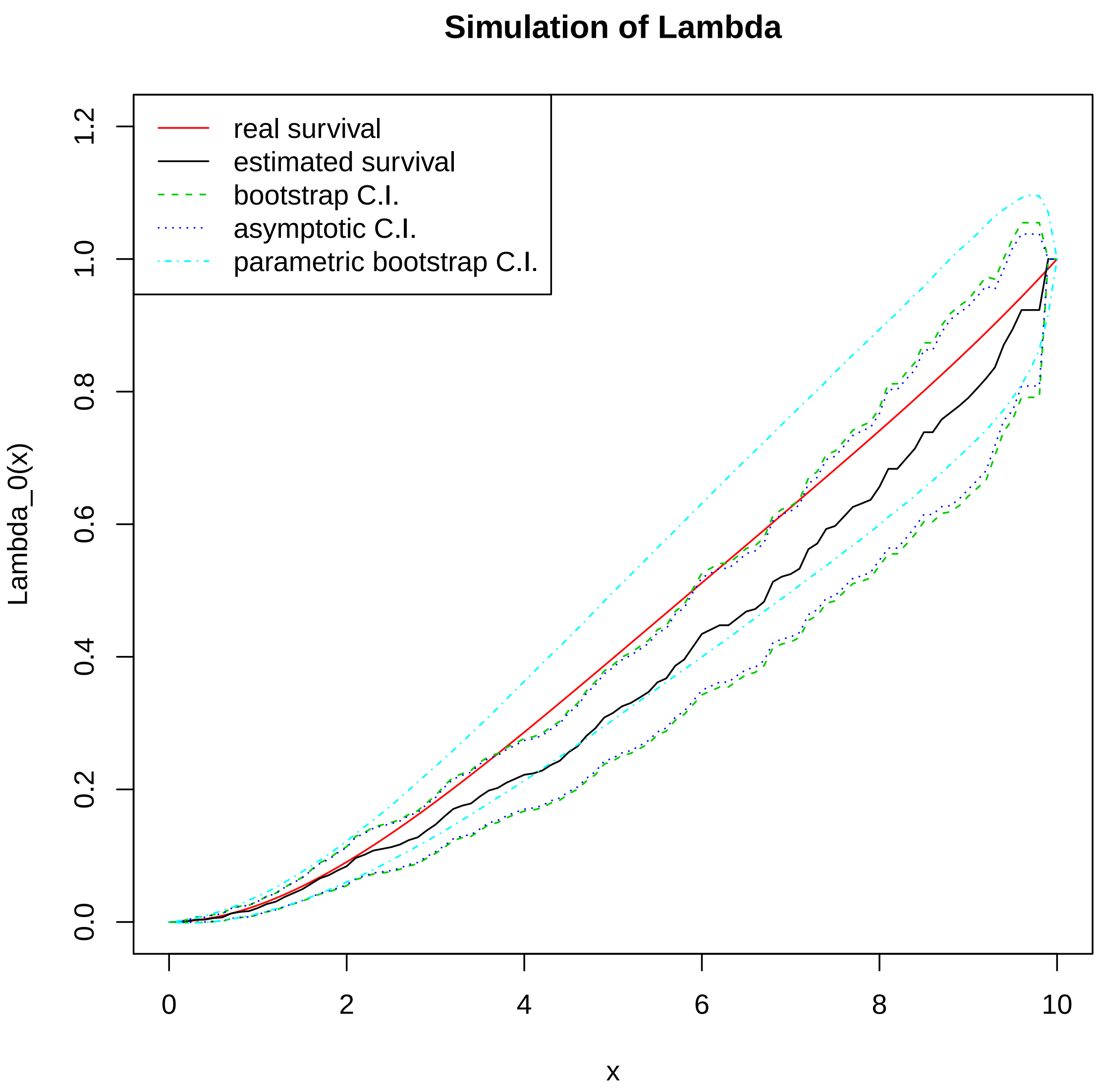
5. Simulation

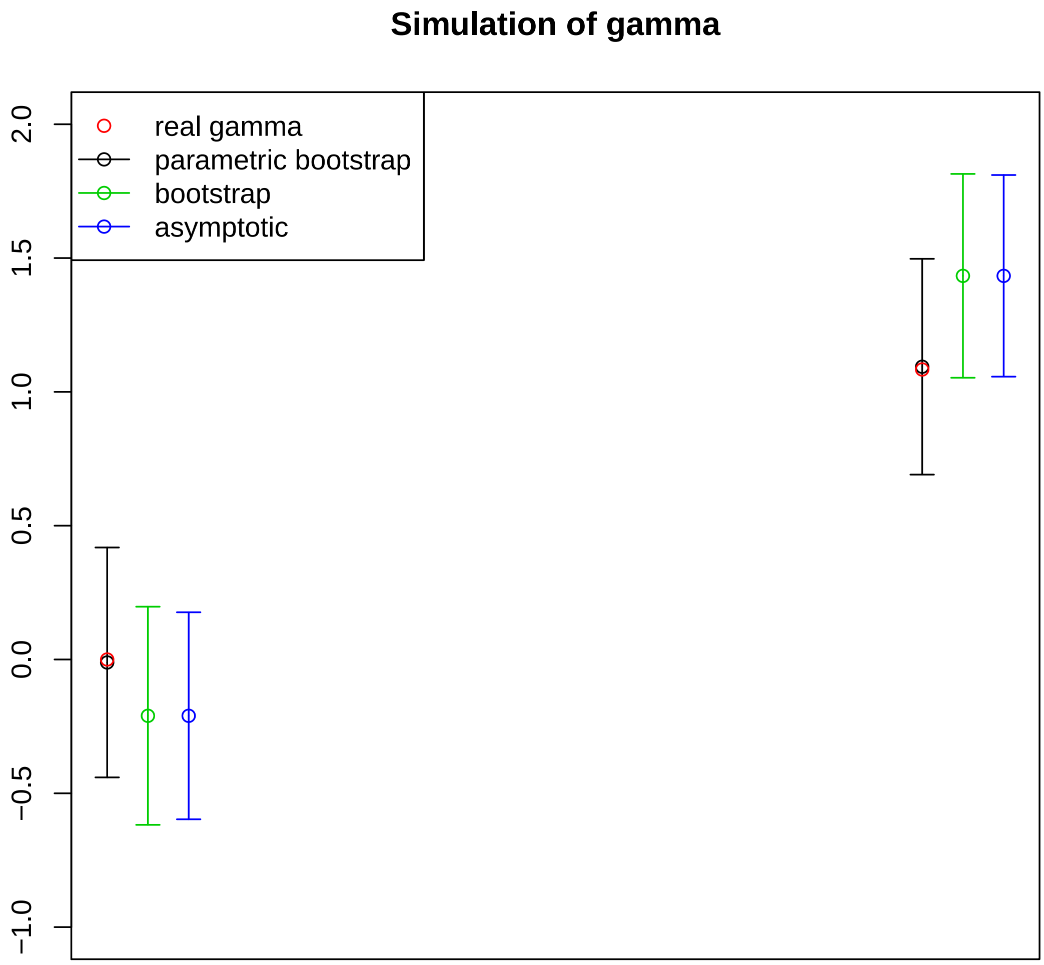
```
library(knitr)
opts_chunk$set(cache = TRUE, echo = FALSE, cache.path = "cache_simulation/")
```

We conduct some simulation studies to test the correctness of the package. The source codes are not showed here, but the reader could find them under 'vignettes/'. We use a self developed package to generate the realization of inhomogenous poisson process. Please install the package if you want to re-run the simulation.

5.1. Wang *et al.* (2001)

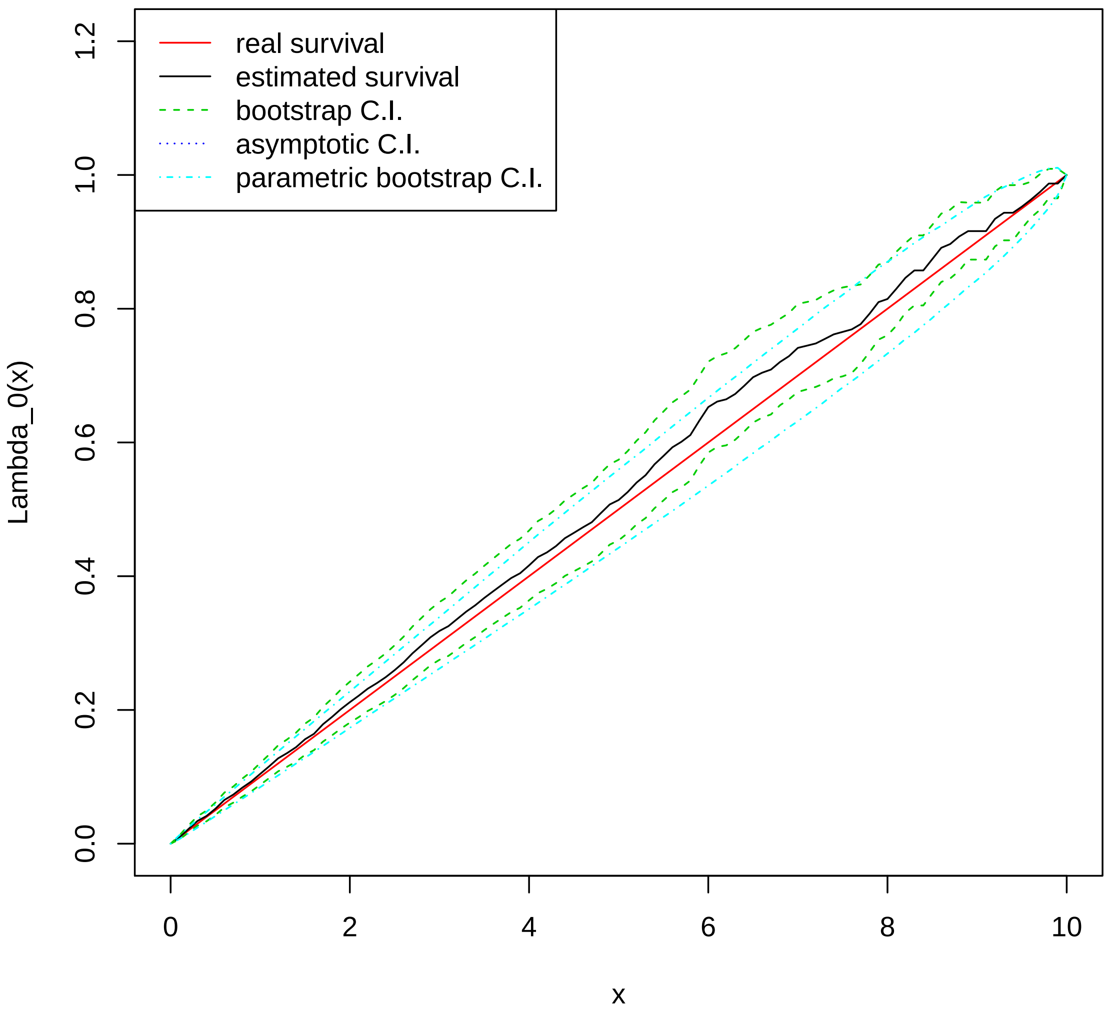
```
library(knitr)
opts_chunk$set(cache = TRUE, echo = FALSE, cache.path = "cache_simulation_wang2001/")
```

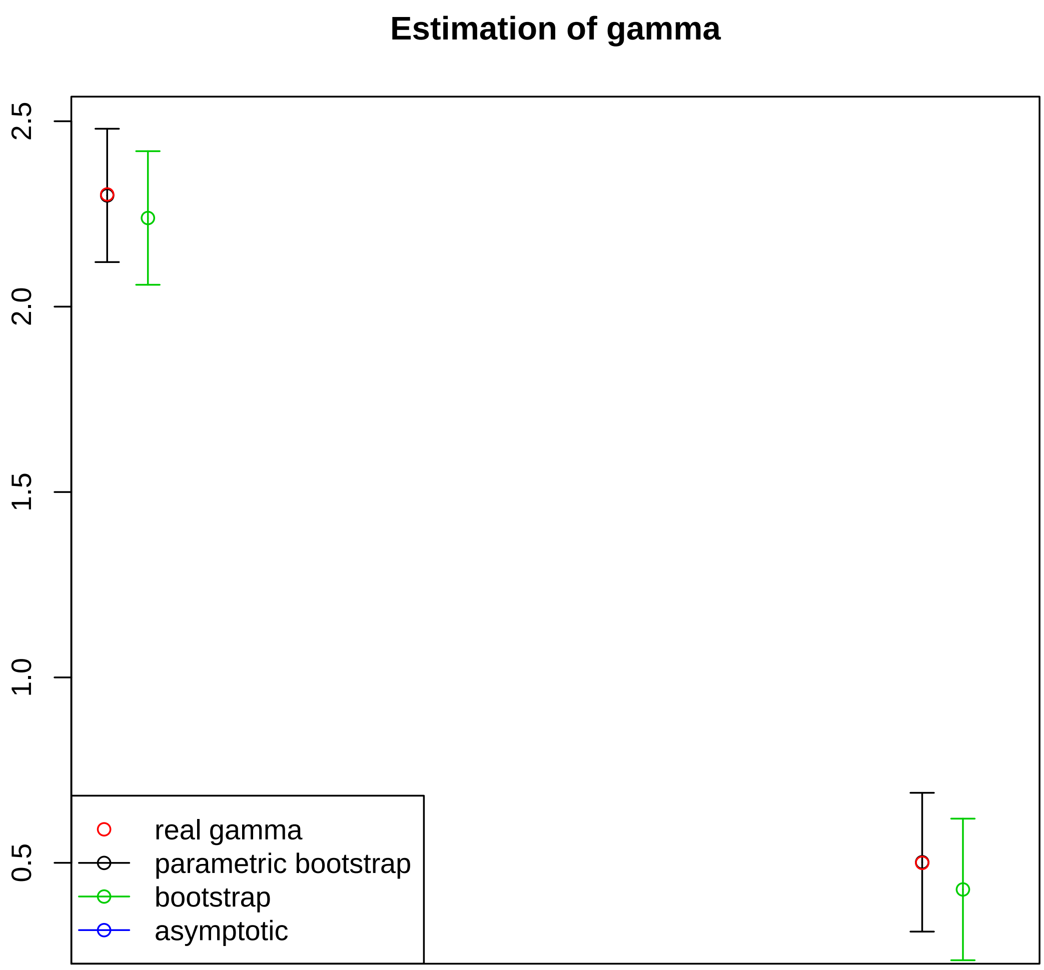


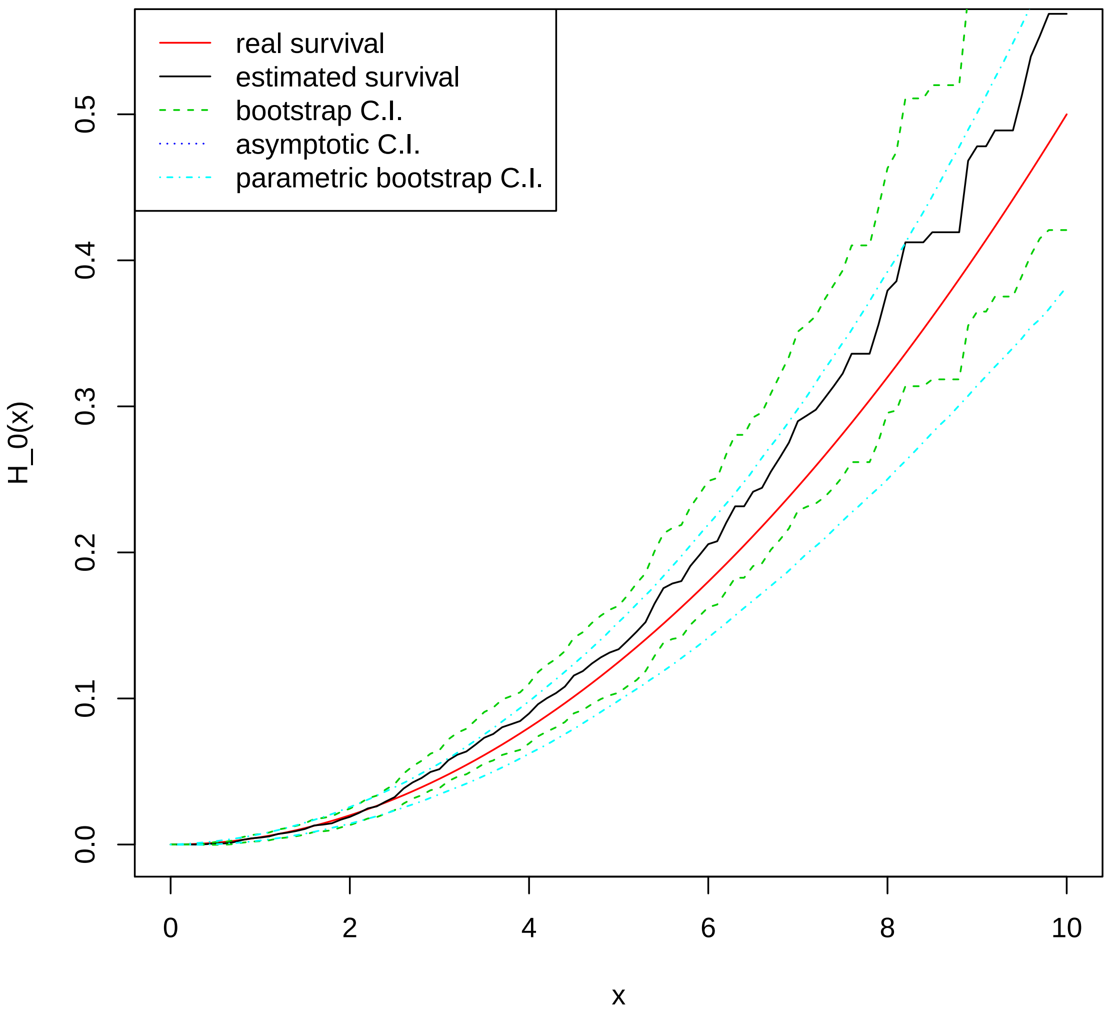


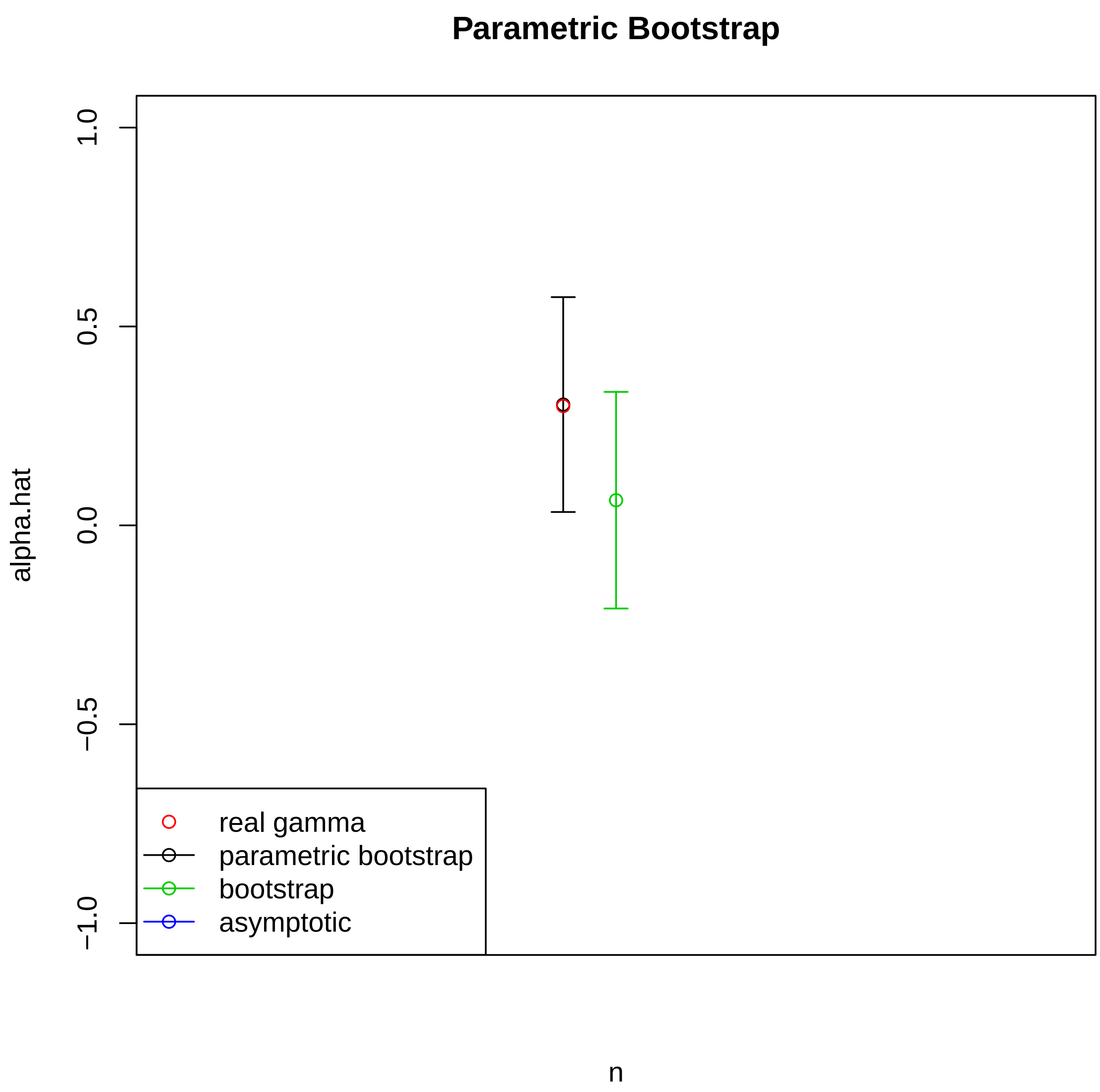
5.2. Huang and Wang (2004)

```
library(knitr)
opts_chunk$set(cache = TRUE, echo = FALSE, cache.path = "cache_simulation_huang2004/")
```


real survival
estimated survival
bootstrap C.I.
asymptotic C.I.
parametric bootstrap C.I.
Lambda_0(x)
x
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
2
4
6
8
10

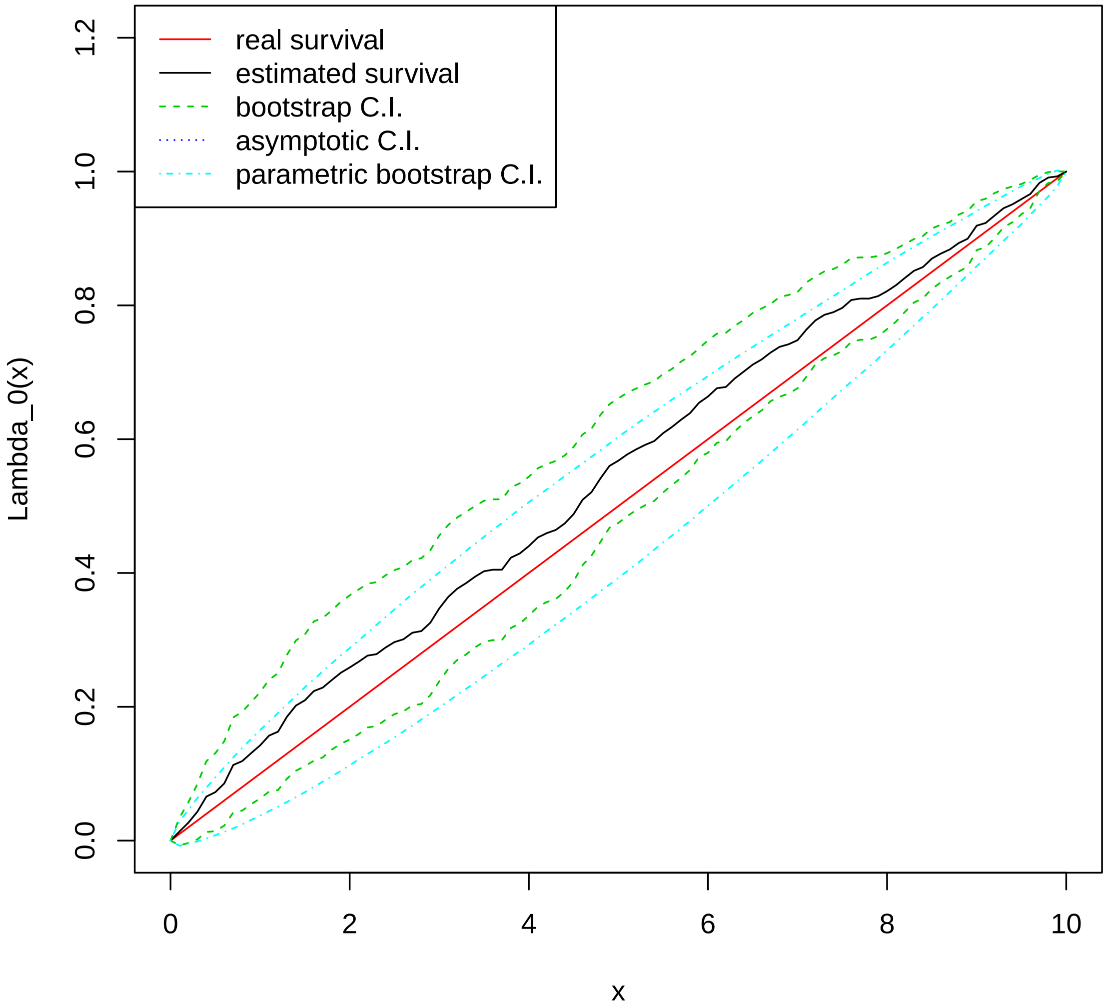

Estimation of gamma
2.5
2.0
1.5
1.0
0.5
real gamma
parametric bootstrap
bootstrap
asymptotic


real survival
estimated survival
bootstrap C.I.
asymptotic C.I.
parametric bootstrap C.I.
H_0(x)
0.0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8
10
x

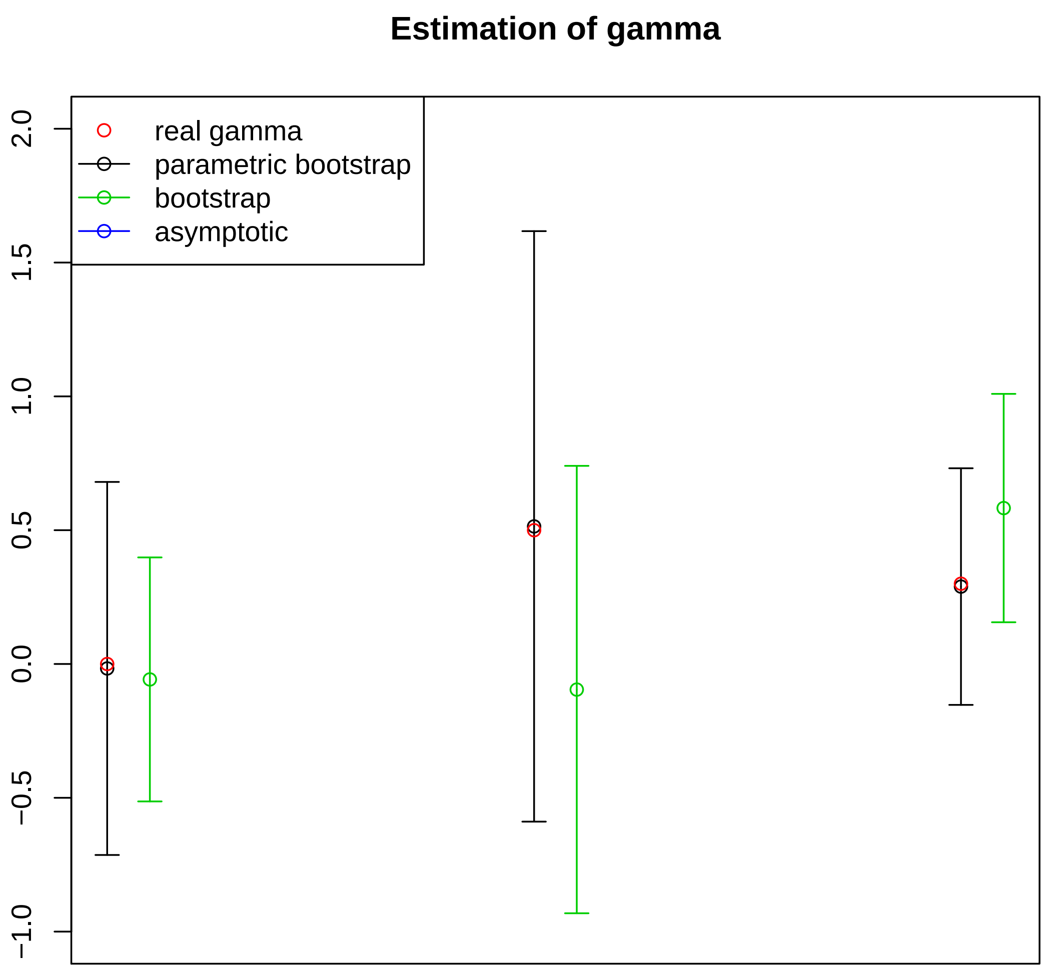


5.3. Huang *et al.* (2010)

```
library(knitr)
opts_chunk$set(cache = TRUE, echo = FALSE, cache.path = "cache_simulation/")
```



real survival
estimated survival
bootstrap C.I.
asymptotic C.I.
parametric bootstrap C.I.
Lambda_0(x)
x
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
2
4
6
8
10


Estimation of gamma
real gamma
parametric bootstrap
bootstrap
asymptotic
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0

References

Huang CY, Qin J, Wang MC (2010). "Semiparametric Analysis for Recurrent Event Data with Time-Dependent Covariates and Informative Censoring." *Biometrics*, **66**(1), pp. 39–49. ISSN 0006341X. URL `http://www.jstor.org/stable/40663150`.

Huang CY, Wang MC (2004). "Joint Modeling and Estimation for Recurrent Event Processes and Failure Time Data." *Journal of the American Statistical Association*, **99**(468), pp. 1153–1165. ISSN 01621459. URL `http://www.jstor.org/stable/27590493`.

Wang MC, Qin J, Chiang CT (2001). "Analyzing Recurrent Event Data with Informative Censoring." *Journal of the American Statistical Association*, **96**(455), pp. 1057–1065. ISSN 01621459. URL `http://www.jstor.org/stable/2670251`.