Saemix 3 - time-to-event data models

Emmanuelle

09/2022

Version

Use saemix version $\geq$ 3.2

Objective

Run TTE and RTTE models in **saemix**

This notebook uses additional code from the **saemix** development github (https://github.com/saemixdevelopment/saemixextension), not yet integrated in the package. The *workDir* folder in the next chunk of code points to the folder where the user stored this code, and is needed to run the notebook (*workDir* defaults to the current working directory). Specifically, the notebook loads:

- code for the MC/AGQ provided by Sebastian Ueckert (Ueckert et al. 2017)
  - if memory issues arise the code can be run in a separate script.
- the results for the bootstrap runs performed using different approaches (see Comets et al. Pharm Res 2021)
  - bootstraps can be run instead by switching the *runBootstrap* variable to TRUE in the first chunk of code
  - in the code, the number of bootstraps is set to 10 for speed but we recommend to use at least 200 for a 90% CI.
  - this can be changed in the following change of code by uncommenting the line *nboot<-200* and setting the number of bootstrap samples (this may cause memory issues in **Rstudio** with older machines, if this is the case we recommend executing the code in a separate script)

The current notebook can be executed to create an HMTL or PDF output with comments and explanations. A script version containing only the R code is also given as *saemix3_tteModel.R* in the same folder.

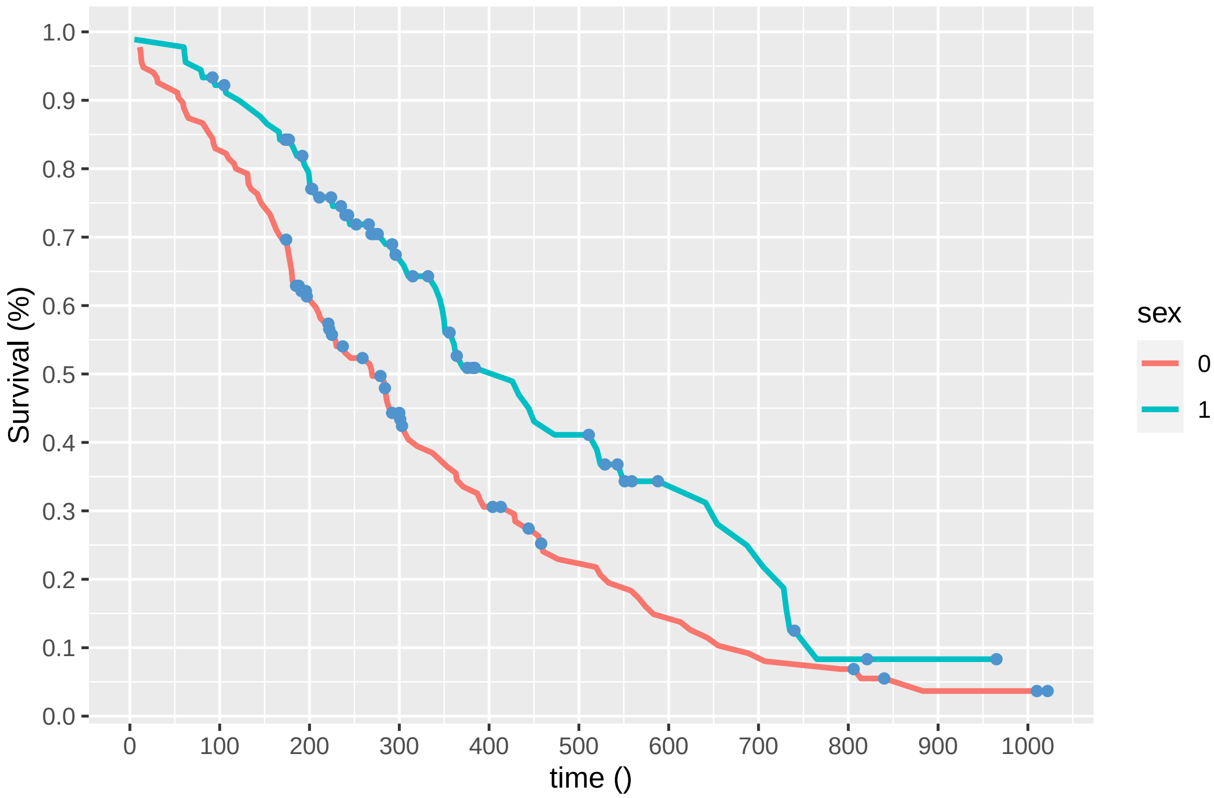
TTE data

Data description - lung cancer The example chosen to illustrate the analysis of time-to-event data in **saemix** is the NCCTG Lung Cancer Data, describing the survival in patients with advanced lung cancer from the North Central Cancer Treatment Group (Loprinzi et al. 1994). Covariates measured in the study include performance scores rating how well the patient can perform usual daily activities. We reformatted the *cancer* dataset provided in the **survival** package in R in SAEM format: patients with missing age, sex, institution or physician assessments were removed from the dataset. Status was recoded as 1 for death and 0 for a censored event, and a censoring column was added to denote whether the patient was dead or alive at the time of the last observation. A line at time=0 was added for all subjects. Finally, subjects were numbered consecutively from 0 to 1.

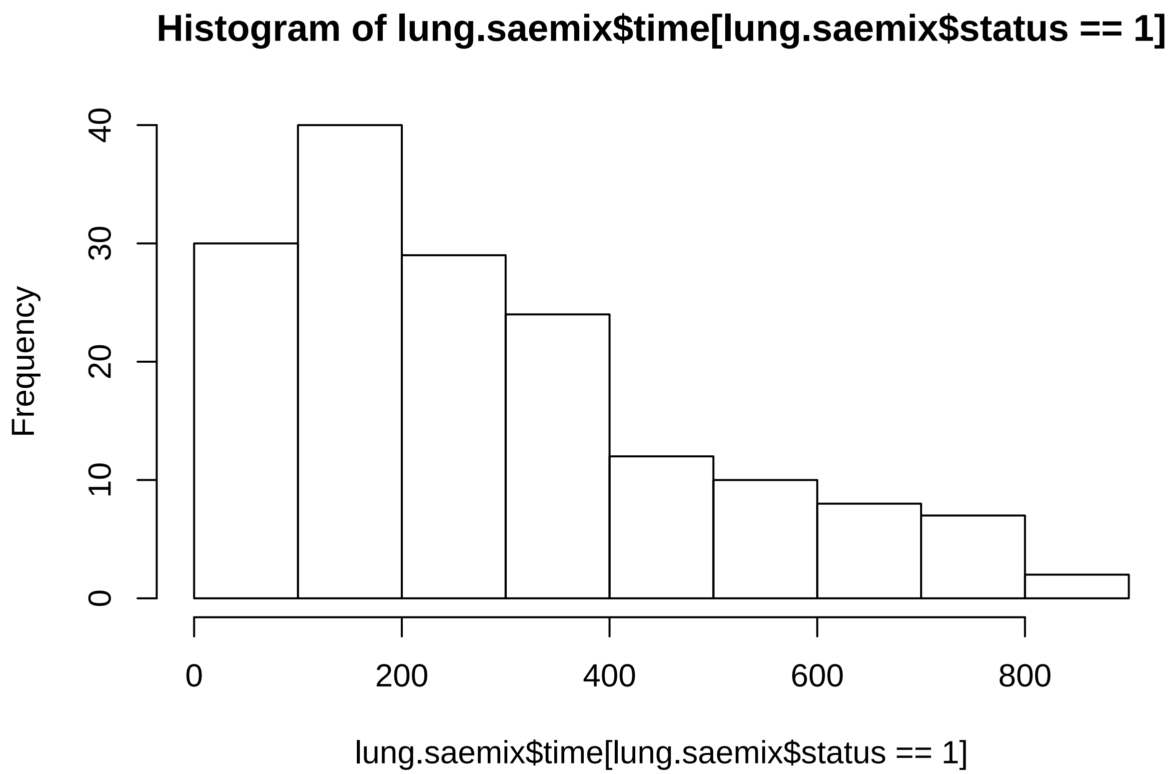
We can plot the distribution of times as a histogram.

```
data(lung.saemix)
saemix.data<-saemixData(name.data=lung.saemix,header=TRUE,name.group=c("id"),
      name.predictors=c("time","status","cens"),name.response=c("status"),
```

```
      name.covariates=c("age", "sex", "ph.ecog", "ph.karno", "pat.karno", "wt.loss","meal.cal"),
      units=list(x="days",y="",covariates=c("yr","","-","%","%","cal","pounds")), verbose=FALSE)

plotDiscreteData(saemix.data, outcome="tte", which.cov="sex")
```




```
# Histogram
hist(lung.saemix$time[lung.saemix$status==1])
```



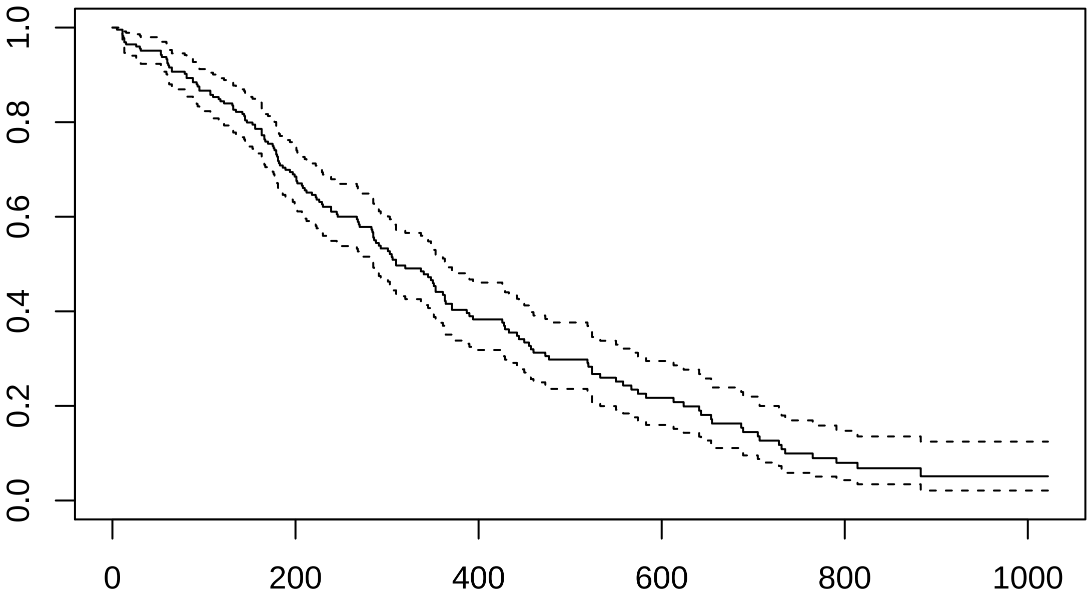

```
# Note: missing data in pat.karno, wt.loss and meal.cal
if(FALSE)
    print(summary(lung.saemix))
```

```
lung.surv<-lung.saemix[lung.saemix$time>0,]
lung.surv$status<-lung.surv$status+1
Surv(lung.surv$time, lung.surv$status) # 1=censored, 2=dead
```

Kaplan-Meier plot

```
##   [1]  306   455  1010+  210   883  1022+  310   361   218   166   170   654
##  [13]  728   567   144   613   707    61    88   301    81   624   371   394
##  [25]  520   574   118   390    12   473    26   533   107    53   122   814
##  [37]  965+   93   731   460   153   433   145   583    95   303   519   643
##  [49]  765   735   189    53   246   689    65     5   132   687   345   444
##  [61]  223   175    60   163    65   208   821+  428   230   840+  305    11
##  [73]  132   226   426   705   363    11   176   791    95   196+  167   806+
##  [85]  284   641   147   740+  163   655   239    88   245   588+   30   179
##  [97]  310   477   166   559+  450   364   107   177   156   529+   11   429
## [109]  351    15   181   283   201   524    13   212   524   288   363   442
## [121]  199   550    54   558   207    92    60   551+  543+  293   202   353
## [133]  511+  267   511+  371   387   457   337   201   404+  222    62   458+
## [145]  356+  353   163    31   340   229   444+  315+  182   156   364+  291
## [157]  179   376+  384+  268   292+  142   413+  266+  194   320   181   285
## [169]  301+  348   197   382+  303+  296+  180   186   145   269+  300+  284+
## [181]  350   272+  292+  332+  285   259+  110   286   270    81   131   225+
## [193]  269   225+  243+  279+  276+  135    79    59   240+  202+  235+  224+
## [205]  239   237+  173+  252+  221+  185+   92+   13   222+  192+  183   211+
## [217]  175+  197+  203+  116   188+  191+  105+  174+  177+
```

```
nonpar.fit <- survfit(Surv(time, status) ~ 1, data = lung.surv)
plot(nonpar.fit)
```



Model for TTE data We can use a Weibull model for the hazard, parameterised as λ and β. For individual i, the hazard function of this model is:

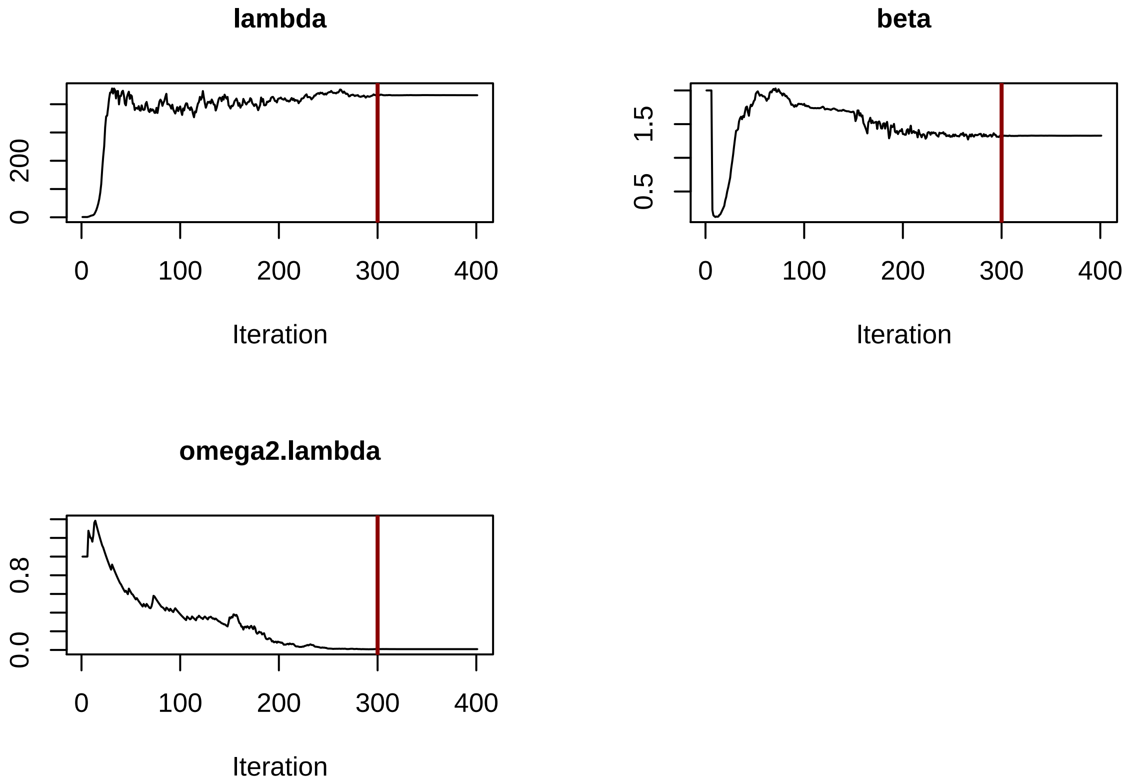
$$h(t) = \frac{\beta}{\lambda} \left(\frac{t}{\lambda}\right)^{\beta-1}$$

And the parametric survival function is given by:

$$S(t) = e^{-\left(\frac{t}{\lambda}\right)^{\beta}}$$

```
weibulltte.model<-function(psi,id,xidep) {
  T<-xidep[,1]
  y<-xidep[,2] # events (1=event, 0=no event)
  cens<-which(xidep[,3]==1) # censoring times (subject specific)
  init <- which(T==0)
  lambda <- psi[id,1] # Parameters of the Weibull model
  beta <- psi[id,2]
  Nj <- length(T)

  ind <- setdiff(1:Nj, append(init,cens)) # indices of events
  hazard <- (beta/lambda)*(T/lambda)^(beta-1) # ln(H')
  H <- (T/lambda)^beta # ln(H)
  logpdf <- rep(0,Nj) # ln(l(T=0))=0
  logpdf[cens] <- -H[cens] + H[cens-1] # ln(l(T=censoring time))
  logpdf[ind] <- -H[ind] + H[ind-1] + log(hazard[ind]) # ln(l(T=event time))
  return(logpdf)
}

saemix.model<-saemixModel(model=weibulltte.model,description="time model",modeltype="likelihood",
  psi0=matrix(c(1,2),ncol=2,byrow=TRUE,dimnames=list(NULL,  c("lambda","beta"))),
  transform.par=c(1,1),covariance.model=matrix(c(1,0,0,0),ncol=2, byrow=TRUE), verbose=FALSE)
saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE, displayProgress=FALSE, print=FALSE)
```

```r
tte.fit<-saemix(saemix.model,saemix.data,saemix.options)
plot(tte.fit, plot.type="convergence")
```




```r
summary(tte.fit)
```

```
## ----------------------------------------------------
## -----------------  Fixed effects  ------------------
## ----------------------------------------------------
##   Parameter Estimate    SE CV(%)
## 1    lambda   431.81 51.60 11.95
## 2      beta     1.33  0.19 14.27
## ----------------------------------------------------
## -----------  Variance of random effects  -----------
## ----------------------------------------------------
##            Parameter Estimate   SE   CV(%)
## lambda omega2.lambda    0.009 0.17 1857.95
## ----------------------------------------------------
## ------  Correlation matrix of random effects  ------
## ----------------------------------------------------
##               omega2.lambda
## omega2.lambda 1.00
## ----------------------------------------------------
## ---------------  Statistical criteria  -------------
## ----------------------------------------------------
## Likelihood computed by linearisation
##       -2LL= 5189.352
##       AIC = 5197.352
##       BIC = 5211.017
##
```

```
## Likelihood computed by importance sampling
##       -2LL= 2269.357
##       AIC = 2277.357
##       BIC = 2291.021
## ----------------------------------------------------
```

Simulation function Simulating from a TTE model is slightly more complicated than for the other non Gaussian models. When the hazard function has an inverse, we can use the inverse CDF technique (or inverse transformation algorithm) for generating a random sample. The method uses the fact that a continuous cumulative density function, F, is a one-to-one mapping of the domain of the cdf into the interval (0,1). Therefore, if U is a uniform random variable on (0,1), then $X = F^{-1}(U)$ has the distribution F.

For the single event Weibull model:

$$F = 1 - e^{-\int_0^T h(u)du} = 1 - e^{-\left(\frac{T}{\lambda}\right)^\beta} \sim \mathcal{U}(0,1)$$

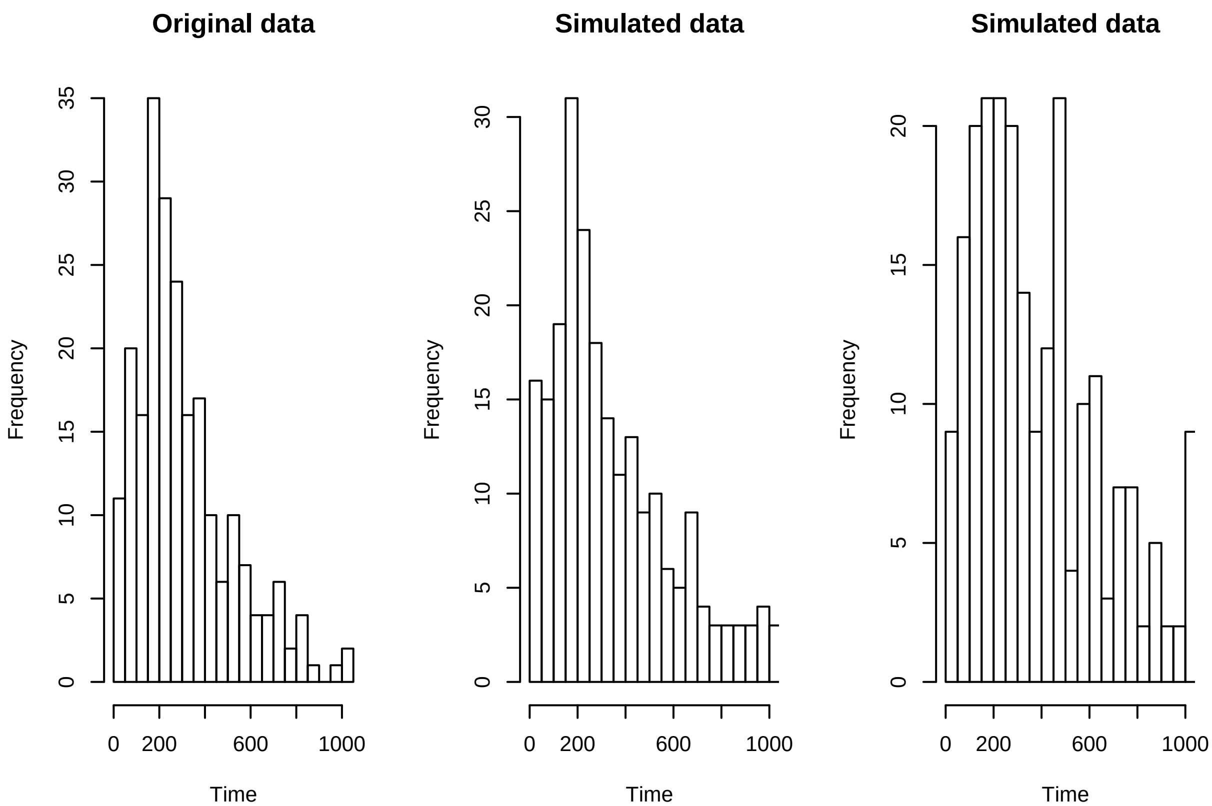
Assuming we simulate $U = 1 - V$ from $\mathcal{U}(0,1)$, we can obtain a sample from the Weibull parametric model as:

$$T = \lambda \left(-\ln(V) + \left(\frac{T}{\lambda}\right)^\beta \right)^{1/\beta}$$

In the following we assume the first column of *xidep* contains the observed times, and that there is a common censoring time (the maximum observed time). We could also assume a common censoring (function *simulateWeibullTTE.maxcens()* below) but simulating from this function shows an excess of times simulated at the censoring limit compared to the original dataset.

```
# Simulate events based on the observed individual censoring time
simulateWeibullTTE <- function(psi,id,xidep) {
  T<-xidep[,1]
  y<-xidep[,2] # events (1=event, 0=no event)
  cens<-which(xidep[,3]==1) # censoring times (subject specific)
  init <- which(T==0)
  lambda <- psi[,1] # Parameters of the Weibull model
  beta <- psi[,2]
  Nj <- length(T)
  ind <- setdiff(1:Nj, append(init,cens)) # indices of events
  tevent<-T
  Vj<-runif(dim(psi)[1])
  tsim<-lambda*(-log(Vj))^(1/beta) # nsuj events
  tevent[T>0]<-tsim
  tevent[tevent[cens]>T[cens]] <- T[tevent[cens]>T[cens]]
  return(tevent)
}

# Checking the simulation function
xidep1<-saemix.data@data[,saemix.data@name.predictors]
nsuj<-saemix.data@N
psiM<-data.frame(lambda=rnorm(nsuj, mean=tte.fit@results@fixed.effects[1], sd=2), beta=tte.fit@results@f
id1<-rep(1:nsuj, each=2)
simtime<-simulateWeibullTTE(psiM, id1, xidep1)

par(mfrow=c(1,2))
hist(saemix.data@data$time[saemix.data@data$time>0], breaks=30, xlab="Time", main="Original data")
hist(simtime[simtime>0], breaks=30, xlim=c(0,1000), xlab="Time", main="Simulated data")
```

```
# Ignoring the cens column and assuming a common censoring time instead
simulateWeibullTTE.maxcens <- function(psi,id,xidep) {
  etime<-xidep[,1]
  censoringtime <- max(etime)
  lambda <- psi[,1]
  beta <- psi[,2]
  N<-dim(psi)[1]
  Vj<-runif(N)
  T<-lambda*(-log(Vj))^(1/beta)
  T[T>censoringtime]<-censoringtime
  etime[etime>0]<-T
  return(etime)
}
simtime.maxcens<-simulateWeibullTTE.maxcens(psiM, id1, xidep1)

par(mfrow=c(1,3))
hist(saemix.data@data$time[saemix.data@data$time>0], breaks=30, xlab="Time", main="Original data")
hist(simtime[simtime>0], breaks=30, xlim=c(0,1000), xlab="Time", main="Simulated data")
hist(simtime.maxcens[simtime.maxcens>0], breaks=30, xlim=c(0,1000), xlab="Time", main="Simulated data")
```



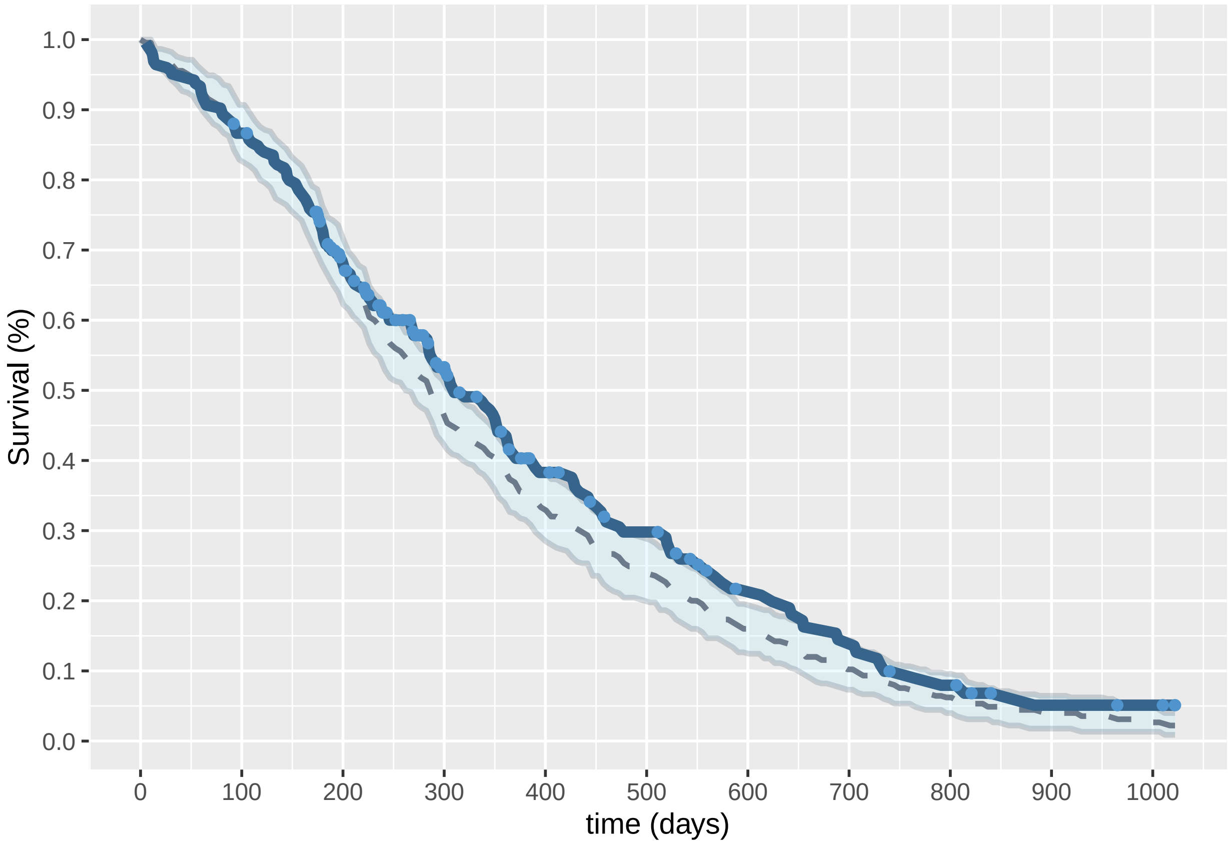
We then use the simulation function defined above to simulate from the fitted model, adding it first to the model component, and plot VPC (we can also include the simulation function when creating the model by adding the argument *simulate.function=simulateWeibullTTE* to saemixModel in the code above).

```
tte.fit@model@simulate.function <- simulateWeibullTTE
simtte.fit <- simulateDiscreteSaemix(tte.fit, nsim=100)
```

```
gpl <- discreteVPC(simtte.fit, outcome="TTE")
```

```
## Error in exists(object) : premier argument incorrect
```

```
plot(gpl)
```



Note that there are some specialised packages such as the **survsim** and the **simsurv** package that could be leveraged for this exercise.

VPC using Ron's package A recent package was developed by Ron Keizer to implement VPC for different types of data. For survival data, we can also use the *vpc_tte()* function from this package to produce the KM-VPC plot.

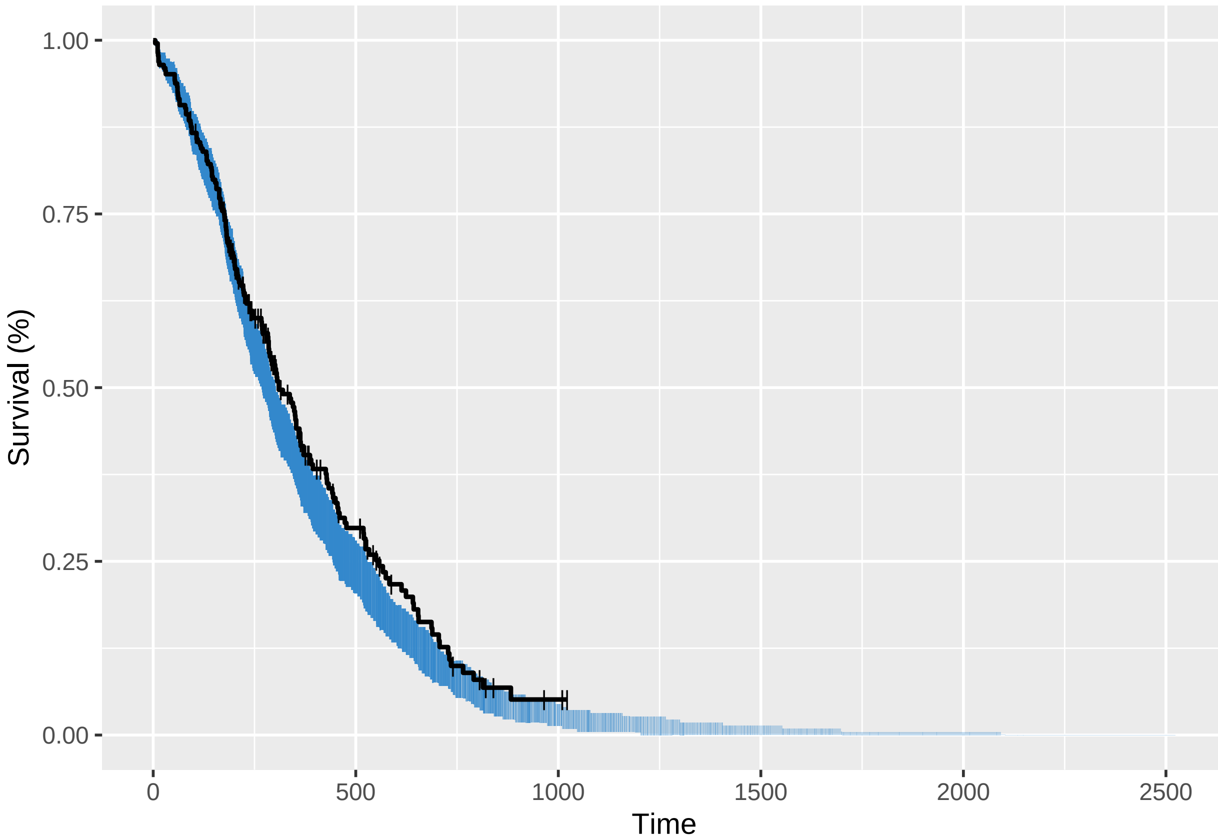
```
library(vpc)

n<-dim(simtte.fit@sim.data@datasim)[1]
simron<-cbind(simtte.fit@sim.data@datasim,dv=rep(c(0,1), n/2))
colnames(simron)<-c("id","sim","time","dv")

vpc_tte(sim = simron,
        obs = lung.saemix,
#        rtte = FALSE,
        sim_cols=list(id="id", dv = "dv", idv = "time"), obs_cols=list(id="id", dv="status", idv = "tim
```

```
## Initializing.
```

```
## Detected column 'cens' with censoring information in observation data, assuming 1=censored event, 0=
```

```
## Warning: `group_by_()` was deprecated in dplyr 0.7.0.
```

```
## Please use `group_by()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.

## Calculating simulation stats.

## Warning: `arrange_()` was deprecated in dplyr 0.7.0.
## Please use `arrange()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.

## ================================================================================

##
## Plotting.

## Warning: Removed 1 rows containing missing values (geom_rect).
```




```
if(FALSE) { # Setting late simulated times to the maximum event time (a censored event)
  simron$time[simron$time>max(lung.saemix$time)]<-max(lung.saemix$time)
  vpc_tte(sim = simron,
          obs = lung.saemix,
          #        rtte = FALSE,
          sim_cols=list(id="id", dv = "dv", idv = "time"), obs_cols=list(id="id", dv="status", idv = "ti
}
```

Exact FIM by AGQ (code by Sebastian Ueckert)

For non-Gaussian models, the exact FIM should be computed, and two approaches have been proposed using either numerical integration by a combination of MC and adaptive Gaussian quadrature (MC/AGQ, Ueckert et al 2017) or stochastic integration by MCMC (Rivière et al. 2017).

Both these approaches are computationally intensive.

Eco pas sûre que la méthode pour calculer la FIM exacte dans un protocole de population soit utilisable pour calculer la FIM empirique d'un jeu de données…

Eco no example for TTE models in Sebastian's code so need to define the different functions by hand…

Here we use code provided by Sebastian Ueckert implementing the MC/AGQ approach, as the MCMC requires the installation of rStan. In this approach, the information matrix (FIM) over the population is first decomposed the sum of the individual FIM:

$$FIM(\Psi, \Xi) = \sum_{i=1}^{N} FIM(\Psi, \xi_i)$$

where ξ_i denotes the individual design in subject i. Assuming Q different elementary designs, the FIM can also be summed over the different designs weighted by the number of subjects N_q in design q as:

$$FIM(\Psi, \Xi) = \sum_{q=1}^{Q} N_q FIM(\Psi, \xi_q)$$

In the following, we first load the functions needed to compute the exact FIM. We then define a model object with the following components:

- *parameter_function*: a function returning the list of parameters as the combination of fixed and random effects
- *log_likelihood_function*: using the parameters, computes the log-likelihood for all y in the dataset
- *simulation_function*: using the parameters, computes the log-likelihood and produces a random sample from the corresponding distribution
- *inverse_simulation_function*: supposed to be the quantile function but not quite sure :-/ (here, returns the category in which is urand)
- *mu*: the fixed parameters
- *omega*: the variance-covariance matrix

For *mu* and *omega*, we use the results from the saemix fit.

TODO Alexandra ?

```
# Code Sebastian
source(file.path(dirAGQ,"default_settings.R"))
source(file.path(dirAGQ,"helper_functions.R"))
source(file.path(dirAGQ,"integration.R"))
source(file.path(dirAGQ,"model.R"))

saemix.fit <- tte.fit

# TODO - adapt to TTE model ???
# model <- Model$new(
#   parameter_function = function(mu, b) list(alp1=mu[1]+b[1], alp2=mu[2], alp3=mu[3], alp4=mu[4], beta
#   log_likelihood_function = function(y, design, alp1, alp2, alp3, alp4, beta) {
#     logit1<-alp1 + beta*design$time
#     logit2<-logit1+alp2
#     logit3<-logit2+alp3
#     logit4<-logit3+alp4
#     pge1<-exp(logit1)/(1+exp(logit1))
```

```
#     pge2<-exp(logit2)/(1+exp(logit2))
#     pge3<-exp(logit3)/(1+exp(logit3))
#     pge4<-exp(logit4)/(1+exp(logit4))
#     pobs = (y==1)*pge1+(y==2)*(pge2 - pge1)+(y==3)*(pge3 - pge2)+(y==4)*(pge4 - pge3)+(y==5)*(1 - pge
#     log(pobs)
#   },
#   simulation_function = function(design, alp1, alp2, alp3, alp4, beta) {
#     logit1<-alp1 + beta*design$time
#     logit2<-logit1+alp2
#     logit3<-logit2+alp3
#     logit4<-logit3+alp4
#     pge1<-exp(logit1)/(1+exp(logit1))
#     pge2<-exp(logit2)/(1+exp(logit2))
#     pge3<-exp(logit3)/(1+exp(logit3))
#     pge4<-exp(logit4)/(1+exp(logit4))
#     x<-runif(length(time))
#     ysim<-1+as.integer(x>pge1)+as.integer(x>pge2)+as.integer(x>pge3)+as.integer(x>pge4)
#   },
#   inverse_simulation_function = function(design, urand,alp1, alp2, alp3, alp4, beta) {
#     if(is.null(urand)) return(seq_along(design$time))
#     logit1<-alp1 + beta*design$time
#     logit2<-logit1+alp2
#     logit3<-logit2+alp3
#     logit4<-logit3+alp4
#     pge1<-exp(logit1)/(1+exp(logit1))
#     pge2<-exp(logit2)/(1+exp(logit2))
#     pge3<-exp(logit3)/(1+exp(logit3))
#     pge4<-exp(logit4)/(1+exp(logit4))
#     1+as.integer(urand>pge1)+as.integer(urand>pge2)+as.integer(urand>pge3)+as.integer(urand>pge4)
#   },
#   mu = saemix.fit@results@fixed.effects,
#   omega = saemix.fit@results@omega[c(1,5),c(1,5)]
#   )

# TBContinued
```

Diagnostics

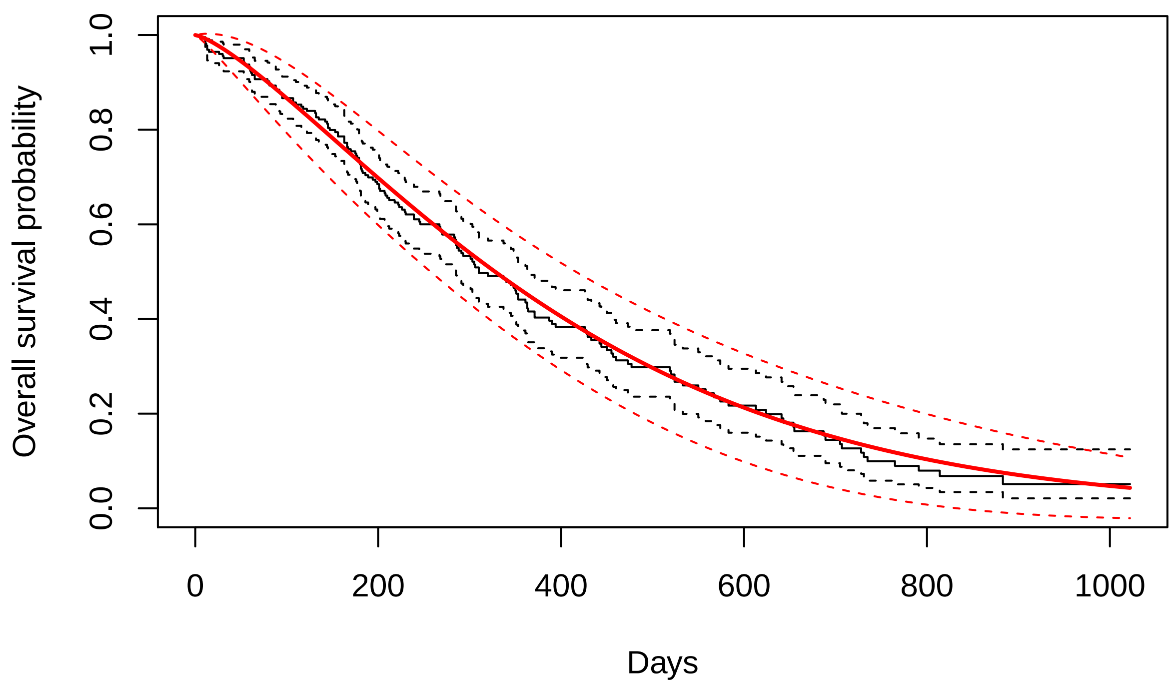
Comparison to the KM fit With TTE data the First-Order approximation for the FIM doesn't seem to perform too badly. We can use the delta-method to obtain standard errors around the value of the survival function, using the following vector of derivatives:

$$\begin{pmatrix} \frac{\delta S}{\delta \lambda} \\ \frac{\delta S}{\delta \beta} \end{pmatrix} = \begin{pmatrix} \frac{\beta}{\lambda} \left(\frac{t}{\lambda}\right)^{\beta} e^{-\left(\frac{t}{\lambda}\right)^{\beta}} \\ -\ln\left(\frac{t}{\lambda}\right) \left(\frac{t}{\lambda}\right)^{\beta} e^{-\left(\frac{t}{\lambda}\right)^{\beta}} \end{pmatrix}$$

We overlay the parametric fit and its confidence interval in red over the previous non-parametric KM estimate, and find a good concordance between the two.

```
ypred<-predict(tte.fit)

# Use survival package to assess Survival curve
xtim<-seq(0,max(lung.saemix$time), length.out=200)
estpar<-tte.fit@results@fixed.effects
```

```
estse<-tte.fit@results@se.fixed
ypred<-exp(-(xtim/estpar[1])^(estpar[2]))

# Computing SE for the survival curve based on linearised FIM (probably not a good idea) through the de
invfim<-solve(tte.fit@results@fim[1:2,1:2])
xcal<- (xtim/estpar[1])^estpar[2]
dsdbeta<- -log(xtim/estpar[1]) * xcal *exp(-xcal)
dsdalpha<- estpar[2]/estpar[1] * xcal *exp(-xcal)
xmat<-rbind(dsdalpha, dsdbeta)
#    x1<-t(xmat[,1:3]) %*% invfim %*% xmat[,1:3]
sesurv<-rep(0,length(xcal))
for(i in 1:length(xcal))
  sesurv[i]<-sqrt(t(xmat[,i]) %*% invfim %*% xmat[,i])

# Comparison between KM and parametric fit
plot(nonpar.fit, xlab = "Days", ylab = "Overall survival probability")
lines(xtim,ypred, col="red",lwd=2)
lines(xtim,ypred+1.96*sesurv, col="red",lwd=1, lty=2)
lines(xtim,ypred-1.96*sesurv, col="red",lwd=1, lty=2)
```



RTTE model

In this section we simulate repeated time-to-event data from a Weibull model and fit it. To simulate from a RTTE model, we simulate repeated events starting from the previous one using the inverse CDF technique. Because we don't know in advance the number of events in each subject, we lose the efficient vectorisation from **R** and this function can be considerably slower than the single event TTE.

```
# Simulating RTTE data by simulating from U(0,1) and inverting the cdf
simul.rtte.unif<-function(psi) { # xidep, id not important, we only use psi
  censoringtime <- 3
  maxevents <- 30
  lambda <- psi[,1]
```

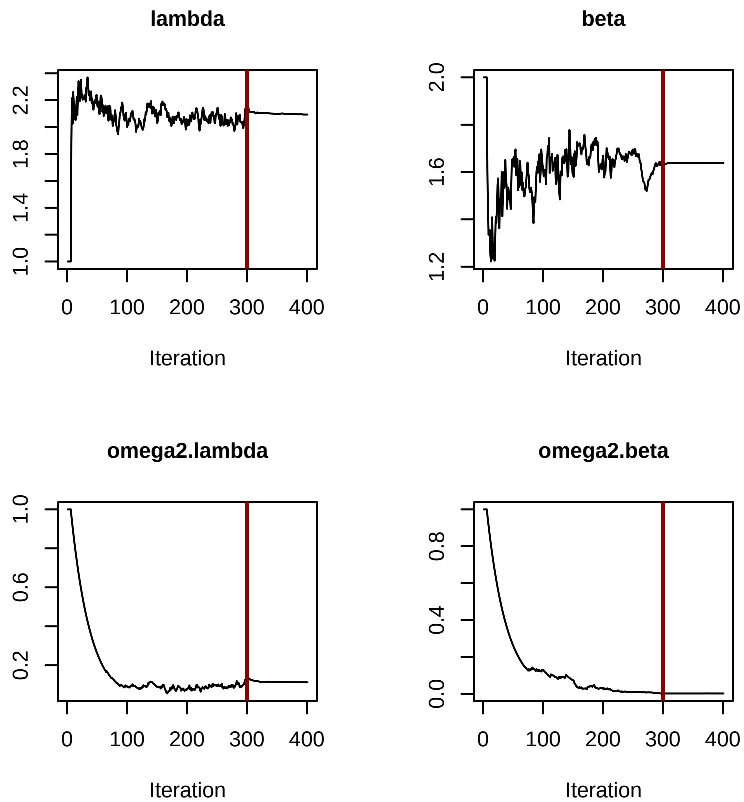
```r
  beta <- psi[,2]
  simdat<-NULL
  N<-nrow(psi)
  for(i in 1:N) {
    eventTimes<-c(0)
    T<-0
    Vj<-runif(1)
    #    T <- (-log(Vj)*lambda[i])^(beta[i])
    T<-lambda[i]*(-log(Vj))^(1/beta[i])
    nev<-0
    while (T < censoringtime & nev<maxevents){
      eventTimes <- c(eventTimes, T)
      nev<-nev+1
      Vj<-runif(1)
      #      T <- T+(-log(Vj)*lambda[i])^(beta[i])
      #      T<-(-log(Vj)*lambda[i] + T^(1/beta[i]))^(beta[i])
      T<-lambda[i]*(-log(Vj) + (T/lambda[i])^(beta[i]))^(1/beta[i])
    }
    if(nev==maxevents) {
      message("Reached maximum number of events\n")
    }
    eventTimes<-c(eventTimes, censoringtime)
    cens<-rep(1,length(eventTimes))
    cens[1]<-cens[length(cens)]<-0
    simdat<-rbind(simdat,
                  data.frame(id=i, T=eventTimes, status=cens))
  }
  return(simdat)
}

# Subjects
set.seed(12345)
param<-c(2, 1.5, 0.5)
# param<-c(4, 1.2, 0.3)
omega<-c(0.25,0.25)
nsuj<-200
risk<-rep(0,nsuj)
risk[(nsuj/2+1):nsuj]<-1
psiM<-data.frame(lambda=param[1]*exp(rnorm(nsuj,sd=omega[1])), beta=param[2]*exp(param[3]*risk+rnorm(nsu
simdat <- simul.rtte.unif(psiM)
```

```
## Reached maximum number of events
```

```r
simdat$risk<-as.integer(simdat$id>(nsuj/2))
```

```r
saemix.data<-saemixData(name.data=simdat, name.group=c("id"), name.predictors=c("T"), name.response="sta

rtte.model<-function(psi,id,xidep) {
  T<-xidep[,1]
  N <- nrow(psi) # nb of subjects
  Nj <- length(T) # nb of events (including 0 and censoring times)
  # censoringtime = 6
  censoringtime = max(T) # same censoring for everyone
  lambda <- psi[id,1]
```

```
  beta <- psi[id,2]
  tinit <- which(T==0) # indices of beginning of observation period
  tcens <- which(T==censoringtime) # indices of censored events
  tevent <- setdiff(1:Nj, append(tinit,tcens)) # indices of non-censored event times
  hazard <- (beta/lambda)*(T/lambda)^(beta-1)
  H <- (T/lambda)^beta
  logpdf <- rep(0,Nj)
  logpdf[tcens] <- -H[tcens] + H[tcens-1]
  logpdf[tevent] <- -H[tevent] + H[tevent-1] + log(hazard[tevent])
  return(logpdf)
}

saemix.model.base<-saemixModel(model=rtte.model,description="Repeated TTE model",modeltype="likelihood"
                               psi0=matrix(c(1,2),ncol=2,byrow=TRUE,dimnames=list(NULL,  c("lambda","be
                               transform.par=c(1,1),covariance.model=matrix(c(1,0,0,1),ncol=2, byrow=TRU
saemix.model<-saemixModel(model=rtte.model,description="Repeated TTE model",modeltype="likelihood",
                          psi0=matrix(c(1,2),ncol=2,byrow=TRUE,dimnames=list(NULL,  c("lambda","beta"))
                          transform.par=c(1,1),covariate.model=matrix(c(0,1),ncol=2),
                          covariance.model=matrix(c(1,0,0,1),ncol=2, byrow=TRUE), verbose=FALSE)
saemix.options<-list(seed=632545,save=FALSE,save.graphs=FALSE, fim=FALSE, displayProgress=FALSE, print=
rtte.fit<-saemix(saemix.model,saemix.data,saemix.options)
plot(rtte.fit, plot.type="convergence")
```




```
print(rtte.fit@results)
```

```
## ----------------------------------------------------
## ------------------  Fixed effects  ------------------
## ----------------------------------------------------
```

```
##      Parameter       Estimate
## [1,] lambda          2.1
## [2,] beta            1.6
## [3,] beta_risk(beta) 0.4
## ----------------------------------------------------
## -----------  Variance of random effects  -----------
## ----------------------------------------------------
##        Parameter     Estimate
## lambda omega2.lambda 0.1125
## beta   omega2.beta   0.0015
## ----------------------------------------------------
## ------  Correlation matrix of random effects  ------
## ----------------------------------------------------
##               omega2.lambda omega2.beta
## omega2.lambda 1             0
## omega2.beta   0             1
## ----------------------------------------------------
## ---------------  Statistical criteria  -------------
## ----------------------------------------------------
##
## Likelihood computed by importance sampling
##       -2LL= 690.2485
##       AIC = 702.2485
##       BIC = 722.0384
## ----------------------------------------------------
```

Work in progress: currently, no diagnostic plots available for RTTE, stay tuned for progress.

Statistical model A nice review of the more frequent hazard functions used in parametric models of TTE data has recently been van Wijk and Simonsson (*CPT:PSP* 2022), including a Shiny app to explore their shape and how to set initial parameters. These models are very sensitive to the initial parameter estimates and their variance, therefore using

References

Comets E, Rodrigues C, Jullien V, Ursino M (2021). Conditional non-parametric bootstrap for non-linear mixed effect models. *Pharmaceutical Research*, 38: 1057-66.

Keizer R (2021). vpc: Create Visual Predictive Checks. *R package* version 1.2.2. https://CRAN.R-project.org/package=vpc

Morina D, Navarro A (2014). The R package survsim for the simulation of simple and complex survival Data. *Journal of Statistical Software*, 59(2), 1–20.

Ueckert S, Mentré F (2017). A new method for evaluation of the Fisher information matrix for discrete mixed effect models using Monte Carlo sampling and adaptive Gaussian quadrature. *Computational Statistics and Data Analysis*, 111: 203-19. 10.1016/j.csda.2016.10.011

van Wijk R, Simonsson U (2022). Finding the right hazard function for time-to-event modeling: A tutorial and Shiny application. *Clinical Pharmacokinetics and Therapeutics: Pharmacometrics and Systems Pharmacology*