

# Use Software R to do Survival Analysis and Simulation. A tutorial

MAI ZHOU

Department of Statistics, University of Kentucky  
©GPL 2.0 copyrighted

In this short tutorial we suppose you already have R (version 2.0.0 or later) installed, and know how to start and quit R. If not, please look at my notes *Install and Use R on Windows PC*. This notes should work for all versions of R (Windows, Mac and Linux). Now start R and continue

## 1 Load the package Survival

A lot of functions (and data sets) for survival analysis are in the package `survival`, so we need to load it first. This is a package in the recommended list, so if you downloaded the binary when installing R, most likely it is included with the base package. If for some reason you do not have the package `survival`, you need to install it first. The easiest way is to start R and click the button **Install package(s)** and follow instruction from there.

R commands:

```
library()          # see the list of available packages
library(survival) # load it. You can also
                  # click the pull-down manual for packages and load it.
library(help=survival) # see the version number, list of available functions and data sets.
data(aml)          # load the data set aml
aml               # see the data
```

One feature of survival analysis is that the data are subject to (right) *censoring*.

Example: 2.2; 3+; 8.4; 7.5+.

This means the second observation is larger than 3 but we do not know by how much, etc. These often happen when subjects are still alive when we terminate the study. To handle the two types of observations, we use two vectors, one for the numbers, another one to indicate if the number is a right censored one. In R, we represent the data by

```
stime <- c(2.2, 3, 8.4, 7.5)
status <- c(1,0,1,0)
```

In R they are later used as in `Surv( stime, status )`. Right censoring happens for the case of `aml` data set, variable `time` and `status`. Try

```
Surv(aml$time, aml$status)
```

To simulate the (randomly right) censored observations, we need to first simulate a lifetime vector and, independently, a termination time (=censoring time) vector. Then we observe whichever comes first (`ztimes`) and related indicator (`status`).

```
lifetimes <- rexp( 25, rate = 0.2)
censtimes <- 5 + 5*runif(25)
ztimes <- pmin(lifetimes, censtimes)
status <- as.numeric(censtime > lifetimes)
```

The estimation goal is to recover the exponential distribution of the lifetimes from the censored observations `ztimes` and `status` only.

## 2 The Kaplan-Meier and Nelson-Aalen estimator

To estimate the distribution of lifetimes nonparametrically, based on right censored observations, we use the Kaplan-Meier estimator. The R function to do that is `survfit()` (part of the survival package).

```
summary(survfit(Surv(aml$time[1:11], aml$status[1:11])))
```

This will give (print) the Kaplan-Meier estimator and its estimated sd (square root of Greenwood formula), and the 95% (Wald) confidence interval using the log transform. If you want to get a nice plot, then do

```
fit1 <- survfit(Surv(aml$time[1:11], aml$status[1:11]))
plot(fit1)
```

The plot show, along with the Kaplan-Meier curve, the (point-wise) 95% confidence interval and ticks for the censored observations.

To estimate the cumulative hazard function by the Nelson-Aalen estimator we need to compute a slightly different version (use option `type="fh"` for Fleming and Harrington) and save the output then do some computation afterwards.

```
temp <- summary(survfit(Surv(aml$time[1:11], aml$status[1:11]), type="fh"))
list(temp$time, -log(temp$surv))
```

Or we may also use `coxph()` and then `basehaz()`.

So, you say we can easily construct confidence intervals from the estimator and its estimated sd — Wrong. Since the finite sample distribution are usually quite skewed, (especially at the tail) we should construct the Wald confidence interval ( $\hat{\theta} \pm 1.96\hat{\sigma}$ ) on a transformed scale, (for this, `survfit` provides two: log scale, and log-log scale.) (SAS v.9 proc `lifetest` provide 4 transformations.) But nobody knows which transformation is the best scale, though log scale is often recommended. A consensus is that without transformation the confidence intervals are not very good.

Another approach (my favorite) is to use the empirical likelihood ratio to construct confidence interval, which do not need to specify a transformation. It *automatically* uses the best transformation! See examples in the section 8 later.

Another often needed function is to return the value of the Kaplan-Meier (Nelson-Aalen) estimator at a given time `t`. The function was `survsum()`, but not in R anymore? If not, it should be available at some other web site, include mine. Or you may write one yourself, make use of `approxfun`.

### 3 The Log-rank test and relatives

1. One sample log-rank test. Test if the sample follows a specific distribution (for example exponential with  $\lambda = 0.02$ ). See an R function on my web side for the one sample log-rank test.

2. Two or more sample log-rank test. To test if the two samples are coming from the same distribution or two different distributions. (power is best for proportional hazard/Lehmann alternatives.)

The relevant R function is `survdiff()`. There is also an option for 'rho'. Rho = 0 (default) gives the log-rank test, rho=1 gives the Wilcoxon test. A rho between 0 and 1 is possible. (Is anybody using a rho strickly between 0 and 1 in practice? How are the two compare and is there other tests?)

```
survdiff(Surv(time, status)~x, data=aml)
```

This does a two sample log-rank test, since there are two different values of `x` in the data set `aml`.

Remark: log-rank test can fail completely (i.e. have no power) for two samples that have cross hazard functions. How to deal with such case is one of the class project.

### 4 Parametric regression models

1. Exponential regression

This does an exponential regression:

```
survreg(Surv(aml$time, aml$status)~aml$x, dist="exponential")
```

Or

```
survreg( Surv(time, status)~x, data=aml, dist="exponential")
```

For an iid model, just say `Surv(time, status) ~ 1`. The covariate,  $x$ , can be a continuous variable like age.

2. Weibull regression

Similar to the above except `dist="weibull"`.

**The interpretations of the parameters in the `survreg`:** the estimated coefficients (when specify exponential or weibull model) are actually those for the extreme value distribution, i.e. the log of weibull random variable. Therefore the MLE of the usual exponential distribution,  $\hat{\lambda}$  and the R output estimator is related by  $\hat{\mu} = \log(1/\hat{\lambda}) = -\log(\hat{\lambda})$ .

On the other hand, the log likelihood in the R output is obtained using truly Weibull density. In SAS `proc lifereg`, however, the log likelihood is actually obtained with the extreme value density.

When you use likelihood ratio test, only the difference of two log likelihoods matter. So stick with one definition.

You may predict the quantiles of patients with same covariates in the data used to fit. [Or you may supply your new `data.frame()` with specific covariates.]

```
fit1 <- survreg( Surv(time, status)~x, data=aml, dist="exponential")
predict(fit1, type="quantile", p=c(0.1, 0.5, 0.9) )
```

## 4.1 Confidence regions

In general the confidence region for parameters  $\theta$  (a vector of p-dim) can be obtained as the set

$$\{\theta^0 \mid -2 \log \frac{\max L(\theta^0)}{\max L(\cdot)} < C = \chi_p^2(0.95)\} .$$

To get confidence region/interval via likelihood ratio, we want to fit (=maximize) the regression model with some parameters fixed at our desired value ( $=\theta^0$ ), and obtain the log likelihood value. This can be done by:

1. in the Weibull regression, you can fix a scale by specify `scale=2`. When `scale=1` this goes back to exponential.

2. Use `offset()` command to fix one or more of the regressors, and max over other regressor(s).

First, we need to convert  $x$  to numerical.

```
amlnew <- data.frame(aml, x1=c(rep(0,11), rep(1,12)) )
survreg(Surv(time, status)~offset(0.3*x1), data=amlnew, dist="weibull")
```

This fix the slope (=0.3) for `x1` (no max over it), but try to max over scale and intercept.

3. When you try to fix all the regressors, and there is nothing to fit, `survreg()` refuses to do the maximization since there is nothing to fit. In this case, it is easy to write a log likelihood function yourself, all you need is to compute the log likelihood value (no max needed). For example, with exponential distribution with `lam` as parameter, this will compute the log likelihood value at `lam`.

```
exploglik <- function(time, status, lam) {
  sum(status*log(lam)) - lam*sum(time)
}
exploglik(time=aml$time, status=aml$status, lam= exp(-3.628))
```

The last line above verify the log likelihood value given by R. To verify SAS proc `lifereg`'s calculation of `loglik` value, see below. It is obtained by using extreme value distribution with `log(obs)`. No matter which log likelihood definition you use, you always get the same likelihood ratio (difference of two log likelihoods).

```
extremeloglik <- function(time, status, mu) {
  sum(status*(time-mu)) - sum( exp(time -mu))
}
extremeloglik(time= log(aml$time), status=aml$status, mu= 3.628776)
```

We can find a 95% confidence interval by collecting all those theta values that `- 2*( exploglik(time, status, lam=theta) - max value from survreg)` is less then 3.84. (left to reader).

Here is another example:

```
library(survival)
data(cancer)
summary( survreg(Surv(time, status) ~ age+sex, data=cancer, dist="weibull") )
betas <- 1:95/100
```

```

llik <- rep(0, 95)
for(i in 1:95) {
llik[i] <- survreg(Surv(time, status) ~ age+offset(betas[i]*sex),
                  data=cancer, dist="weibull")$loglik[2]
}
plot( betas, llik, type="l" )
abline( h = max(lik) - 3.84/2 )

```

the interval of the betas values, with its llik value above the line, is the 95% confidence interval. (compare this with the Wald confidence interval)

## 4.2 Interval censored data

The parametric regression function `survreg` in R and `proc lifereg` in SAS can handle interval censored data.

The model specification and the output interpretations are the same. The only thing different is the input of the data. (the left hand side of the equation)

In R the interval censored data is handled by the `Surv` function. A single interval censored observation [2, 3] is entered as

```
Surv(time=2,time2=3, event=3, type = "interval")
```

When `event = 0`, then it is a left censored observation at 2. When `event = 2`, then it is a right censored observation at 2. When `event = 1`, then it is an uncensored observation at 2.

The R function `survreg` can take a user defined distribution. See the help page for `survreg.distributions` for details. It is a class project to define a piecewise exponential distribution. (for people good with R). (Warning, this seem to be a hard project)

## 5 Cox regression models.

### 5.1 Simple Cox regression model. I

The assumption of the Cox model are: the hazard function of  $Y_i$  is given as  $h_0(t) \exp(\beta z_i)$  for  $i = 1, \dots, n$ . The baseline hazard,  $h_0(t)$ , is common to all  $i$  but otherwise is arbitrary and unknown.

Cox proportional hazards regression model can be thought of as an exponential regression model under a “crazy” (common but unknown) clock. So, there is no intercept for the regression, it becomes part of this crazy clock. See Zhou 2000. The (partial likelihood) estimators are rank based, therefore do not depend on the clock.

Due to censoring, we observe  $T_i = \min(Y_i, C_i)$ ,  $\delta_i = I[Y_i \leq C_i]$ .

Let  $\mathfrak{R}_i = \{j : T_j \geq T_i\}$ , the risk set at time  $T_i$ . Define

$$\ell(\beta) = \sum_{i=1}^n \delta_i \left[ z_i - \frac{\sum_{j \in \mathfrak{R}_i} z_j \exp(\beta z_j)}{\sum_{j \in \mathfrak{R}_i} \exp(\beta z_j)} \right]. \quad (1)$$

If  $\hat{\beta}_c$  is the solution of (1), i.e.  $\ell(\hat{\beta}_c) = 0$ , then  $\hat{\beta}_c$  is called the Cox partial likelihood estimator for  $\beta$ .

Cox said we may treat  $\ell(\beta)$  like the first derivative of a log likelihood function – the score function. The R command to compute  $\hat{\beta}_c$  etc. is

```
coxfit1 <- coxph(Surv(time, status)~x, data=aml)
summary(coxfit1)
```

To obtain the (cumulative) baseline hazard estimator:

```
basehaz(coxph(Surv(time, status)~x, data=aml))
```

Notice this is the cumulative hazard for a hypothetical subject with the covariate value equal to the mean values. Here the  $x$  values are converted to be 0 (Maintained) or 1 (Nonmaintained).

To obtain the (model based) survival function of a (imaginary) particular subject with specific covariate values ( $x = 1$ ):

```
coxfit1 <- coxph(Surv(time, status)~x, data=aml)
survfit(coxfit1, newdata=data.frame(x=1))
```

And the predict function ?

## 5.2 Cox regression model with time change covariates. II

More flexible Cox model have time-change covariates. The time-change covariates can be thought of as the piecewise exponential under a crazy clock. For interpretation of the time-changing covariates see Zhou (2000).

In R, we handle observations with time-change covariate as multiple observations.

For an observation  $T$ , assume  $T > t_1, \delta = 1$ , with covariate value  $A_1$  in time interval  $[0, t_1]$  and changed to covariate value  $A_2$  in time interval  $(t_1, \infty)$  is treated like two observations: one observation entering at time zero and censored at  $t_1$  with covariate  $A_1$ , the second observation entering/starting at time  $t_1$  and die at time  $T$  with covariate  $A_2$ .

The R `coxph()` Cox model command is changed to

```
Surv(start, stop, status)~x
```

Otherwise it is the same as before.

## 5.3 Cox regression model III; Partially known baseline.

Regular Cox model has an arbitrary baseline hazard function. Here the Cox model have additional information on the baseline. For example, the baseline hazard may have median equal to a known constant, but otherwise arbitrary. We take advantage of this additional information in the inference. The (regression) estimator should be more accurate.

This is a new procedure, the computation function is available from the package: `coxEL` (on my web site only), and the function `coxphEL()`.

How to use the function? Compared to `coxph()` there are two more inputs: `gfun` and `lam`.

These are used to impose a constraint on the baseline hazard. This can be thought of as extra information on the baseline hazard.

$$\int g(t)d\Lambda_0(t) = C$$

where  $\Lambda_0$  is the baseline cumulative hazard, we need to specify the function  $g(t)$  as `gfun`. The output of the function `coxphEL` contains the value  $C$  corresponding to the (Lagrange multiplier) `lam`. To get the integral to equal to a given  $C$  we must (manually) adjust the `lam` value. In other words, if you use `lam = 0.5` then the actual constraint you imposed on the baseline hazard is the above equation but with  $C = \text{constr.value}$  in your output of `coxphEL`. This `constr.value` will change with `lam`. When `lam = 0` then it is equivalent to no constraint, and the output `constr.value` correspond to the integral of the baseline hazard in a regular Cox model.

```
fit <- coxphEL(Surv(time, status)~x, data=testdata, lam=0.5, gfun=g)
fit$constr.value
```

Due to the different definition of the baseline hazard in softwares and books, we need to be careful about what is the baseline. The constraint is imposed on (softwares' definition of) baseline hazard — in R (and Splus and SAS) this is the hazard for the subject with the average covariates (instead of zero covariates). :( If you need to impose the constraint on (books' definition of) baseline hazard — those with zero covariate, you need to do this:

```
z0 <- fit$means          # get the mean of the covariates
bz0 <- sum(z0 * coef(fit)) # compute inner product beta*z0
fit$constr.value <- fit$constr.value * exp(-bz0) # minus bz0 in exponent
```

You can see all the component names of a `coxphEL()` fit by `names()`.

Now suppose you need to set the  $C$  value in the constraint to a specific value, you may use `uniroot()`.

## 6 Simulation: Model comparison.

You can do simulations in R much easier than in SAS.

### 6.1 Comparing regression models

We look at one example: Exponential/Weibull regression model versus the Cox regression model.

Some facts:

1. Cox model is applicable to a wider class of distributions. (semi-parametric)
2. If exponential/Weibull model is applicable then Cox model is also applicable. (but not the other way around)
3. When both models are applicable, the Cox model estimate is less accurate, tests are less powerful.

We use simulation to assess how much power/accuracy is lost by using the Cox model instead of Weibull model when both model are correct. Also, the comparison (3) above is only a statement of “on average”. It is not clear whether Weibull will beat Cox on every single occasion or not.

We first generate the observations that follow an exponential regression model, then we fit two models.

```
x <- (1:30)/20 - 3      # create the covariates, 30 of them
myrates <- exp(3*x+1)  # the risk exp(beta*x), parameters for exp r.v.
```

```

y <- rexp(30, rate = myrates) # generates the r.v.
survreg(Surv(y,rep(1,30))~x,dist="weibull")$coef[2]
      #slope estimate by weibull regression
      #we do not have any censoring in the data
coxph(Surv(y,rep(1,30))~x)$coef # estimate from Cox regression

```

If that is successful, we are ready to replicate. To repeat this many times we perhaps want to put it into a function (no need to repeat `myrates` since we use a fixed design.)

```

Simu2reg <- function(x , inputrates){
  y <- rexp(length(inputrates), rate=inputrates)
  temp1 <- survreg(Surv(y, rep(1, length(y)))~x,dist="weibull")
  temp2 <- coxph(Surv(y, rep(1, length(y)))~x)
  return(c(temp1$coef[2], temp2$coef))
}

```

After that you can try to do many simulations in a loop. I recommend you try small number of runs in a loop to start with. You do not want to do a loop that will take days or even month to finish on the first try. I feel a loop that took a few minutes to complete works best for me (before patience ran out).

```

result <- matrix(NA, nrow=2, ncol=5000) #creat a matrix to hold outcome
for(i in 1:100) result[,i]<-Simu2reg(x,myrates) #run the simulation 100 times

```

If that took only a few seconds on your computer, then you can try more

```

for(i in 101:500) result[i,]<-Simu2reg(x,myrates) #run the simulation 400 times

```

or even more, etc. Because of the memory problem, it is often better to do (say) 5 loops each of 1000 runs than to do one loop of 5000 runs. R will save the random seed automatically so that the two computations will have same result, provided you set the same seed in the beginning (but do not reset it in the middle).

After all 5000 simulations are done, we can look at their sampling distributions:

```

mean(result[1,])
mean(result[2,])
sd(result[1,])
sd(result[2,])
hist(result[1,], xlim=c(0,6))
hist(result[2,], xlim=c(0,6))

```

We should see the means are about the same, but the sd from weibull is a bit smaller than the Cox's.

Since exponential is a special case of weibull, we may repeat the above with `dist="exponential"` in the `survreg()`. Another variation is to use right censored data.

## 6.2 Censored data simulation

To do the simulation with censored data, we need to generate censored observations that follow the weibull regression model.

```
x <- (1:30)/2 - 3      # create the covariates, 30 of them
myrates <- exp(3*x+1) # the risk exp(beta*x), parameters for exp r.v.
y <- rexp(30, rate = myrates) # generates the r.v.
cen <- rexp(30, rate = 0.5 )
ycen <- pmin(y, cen)
di <- as.numeric(y <= cen)
```

The inference has to be based on `x`, `ycen`, `di`

```
survreg(Surv(ycen, di)~x, dist="weibull")$coef[2]
coxph(Surv(ycen, di)~x)$coef
```

Or putting this into a function as before:

```
Simu2regC <- function(x , inputrates){
  y <- rexp(length(inputrates), rate=inputrates)
  cen <- rexp(length(inputrates), rate= mean(inputrates)/2 )
  obs <- pmin(y, cen)
  di <- as.numeric(y <= cen)
  temp1 <- survreg(Surv(obs, di)~x, dist="weibull")
  temp2 <- coxph(Surv(obs, di)~x)
  return(c(temp1$coef[2], temp2$coef))
}
```

I wanted to repeat the point: the Cox model is more general than Weibull model:

If  $g(t)$  is an arbitrary increasing function (power, exponential, log), then  $g(obs)$ ,  $di$ ,  $x$  still follows the Cox model, the estimation/testing results will be identical. (since it only uses the ranks of  $obs$ ). But the Weibull model is going to be completely nonsense (unless  $g(t)$  is a power function).

Stratified Cox model: all observations split into several groups, each group uses its own  $g(t)$  function.

Time-change covariates: Before the  $g(obs)$  transformation,  $obs$  are from a piecewise exponential distribution.

## 7 Using the internal data sets

We already saw and used the data set `aml`. Here is a longer example. The package `survival` contains US population hazard rates from census. It is called `ratetables`. We can build a life table, a survival curve, etc. from it, or compare your study population against this population.

We illustrate the use of it by plotting the hazard curve and survival curve for the US female population, in year 2000.

```

library(survival) # attach the package, if not already did.
data(ratetables) # load the data
ls()             # see what has been loaded
temp <- survexp.us[, "female", "2000"] # extract a vector

```

Now the vector `temp` is the *daily hazard rates* for different age groups of US females in the year 2000. There are 113 rates, most of them are applicable for the interval of one year (for example, the daily hazard rate for a US female between 25 and 26 years of age is 1.697337e-06, but the first 4 rates only apply to shorter intervals (since the rates changes faster there).

To plot the hazard against age, we need to create an age vector of length 113, the first 4 ages needs some extra care. (were the ages are equally spaced, we need not to do that).

```

tempage <- c(0.5/365.25, 4/365.25, 17.5/365.25, 196.6/365.25, 1:109+0.5)
plot(tempage, temp, type="l")

```

To plot the survival curve, we need to convert the hazard rates to survival probability. First, we get the cumulative hazard.

```

tempsurv <- 365.25*temp[5:113] # yearly cumulative hazard
tempsurv <- c(temp[1], 6*temp[2], 21*temp[3], 337.25*temp[4], tempsurv)
tempsurv <- cumsum(tempsurv)
tempage <- c(1/365.25, 7/365.25, 28/365.25, 1:110)
plot(tempage, exp(-tempsurv), xlab="age", ylab="Survival, Female") # plot

```

## 8 More accurate P-values and better confidence intervals by Empirical Likelihood.

Since censoring makes the exact distribution of the statistic too complicated to get (in most cases), people almost always use approximations. (Approximate P-value, approximate 95% confidence interval, etc.)

There are basically 3 types of approximate test: Wald test, (Rao's) Score test, and (Wilks) likelihood ratio test. It is widely accepted that the likelihood ratio tests have better approximation than the other two. (more accurate P-value, confidence interval has coverage probability closer to the nominal value).

We describe here some likelihood ratio tests related to the Kaplan-Meier and Nelson-Aalen estimators. For this purpose we need the package `emplik`. This is a recent package I wrote for R. If you do not have it then download and install it first.

The basic functions in the package `emplik` let you test hypothesis about two types of parameters:

$$\int g(t, \theta) dF(t) = \mu \quad \text{or} \quad \int g(t, \theta) d\Lambda(t) = K ,$$

with  $g(t)$  defined by you. Confidence interval/region (95%) can be obtained as the collection of parameter values that the corresponding test results a P-value larger than 0.05. We illustrate the construction of a confidence interval now.

```
> library(emplik) # load the package
```

1. Confidence interval for the survival probabilities and quantiles.

Probability/Quantile can be expressed as a (explicit/implicit) functional of the CDF or a functional of the cumulative hazard. So, we can do it in two different ways.

For a given  $T$ , the probability  $P(X \leq T) = F(T)$ , and it can be written as  $F(T) = \int I_{[s \leq T]} dF(s)$ .

In the example below I took  $T = 1$ . Also, here (P-value > 0.05) iff (-2LLR < 3.841). So, instead of looking at the P-value, we examine the -2LLR value.

```
set.seed(123)
xvec <- rexp(25)
myfun <- function(x) { as.numeric(x <= 1)}
el.cen.EM(x=xvec, d=rep(1,25), fun=myfun, mu=0.5)$"-2LLR"
```

You should get 3.313886.

The input `d=rep(1,25)` is for handling the censoring. Since there is no censoring in the `xvec`, `d` should be all equal to 1.

After repeat the last line of the above code with `mu=0.6`, `mu=0.4` etc. we find that

```
> el.cen.EM(x=xvec, d=rep(1,25), fun=myfun, mu=0.4859)$"-2LLR"
[1] 3.84151
> el.cen.EM(x=xvec, d=rep(1,25), fun=myfun, mu=0.8389)$"-2LLR"
[1] 3.840775
```

So the approximate confidence interval for  $F(1)$  is  $[0.4859, 0.8389]$ .

If we want the confidence interval for the median,  $F^{-1}(0.5)$ , then we should keep `mu=0.5` and change the definition of the `myfun` as follows.

```
> set.seed(123)
> xvec <- rexp(250)
> myfun <- function(x) { as.numeric(x <= 0.97242)}
> el.cen.EM(x=xvec, d=rep(1,250), fun=myfun, mu=0.5)$"-2LLR"
[1] 3.60869
> myfun <- function(x) { as.numeric(x <= 0.6069)}
> el.cen.EM(x=xvec, d=rep(1,250), fun=myfun, mu=0.5)$"-2LLR"
[1] 3.60869
```

Remark: We do not get 3.84 exactly, due to discreteness nature of the -2LLR. Some smoothing would make things better. But we do not implemented it here.

So the 95% confidence interval for the median based on 250 observations is  $[0.6069, 0.97242]$ .

As you can see this is a bit cumbersome. I have wrote a function that try to find the confidence interval automatically.

```
findUL( )
```

2. confidence interval for the weighted hazards.

For continuous distributions, we have  $\log[1 - F(T)] = -\Lambda(T) = -\int I_{[s \leq T]} d\Lambda(s)$ .

So testing  $H_0 : F(4) = 0.5$  is equivalent to testing  $H_0 : \int I_{[s \leq 4]} d\Lambda(s) = -\log(1 - 0.5)$  for continuous distributions. So the above two examples can also be done in terms of hazard.

This is a special case of the next example.

There is a function in the package `emplik` that can test

$$H_0 : \int g(t, \theta) d\Lambda(t) = K$$

If  $\theta$  is known then the integral value  $K$  is a parameter. If  $K$  is known then  $\theta$  is an implicit parameter.

If the (right censored) data are

( )

then to test the above hypothesis with  $K =$

For more examples, Look at the help pages of `emplik.H1.test` or `emplikdisc.test`

The function `emplikH.disc()` tests the hypothesis

$$H_0 : \int g(t, \theta) \log(1 - d\Lambda(t)) = K$$

If  $g(t, \theta) = I_{[t \leq \theta]}$ , then  $\int g(t, \theta) \log(1 - d\Lambda(t)) = \sum_{t \leq \theta} \log[1 - d\Lambda(t)]$ . This is  $\log \prod_{t \leq \theta} [1 - d\Lambda(t)] = \log[1 - F(\theta)]$  by the product limit formula.

Therefore a test of  $H_0 : F(\theta) = 0.5$  (median is  $\theta$ ) can be obtained with the above using function  $g(t, \theta) = I_{[t \leq \theta]}$  and  $K = \log 0.5$ .

For more examples, see the next session with left truncated data.

### 3. Confidence interval for the weighted mean.

There are also functions in the package to test the hypothesis of the type

$$H_0 : \int g(t, \theta) dF(t) = \mu$$

We first use  $g(t, \theta) = t$  (the default), imply  $\mu$  is the usual mean of the (unknown)  $F(\cdot)$ .

```
library(emplik)
x <- c(1, 1.5, 2, 3, 4, 5, 6, 5, 4, 1, 2, 4.5)
d <- c(1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1)
el.cen.EM(x,d,mu=3.5) # use the default g(t)=t.
```

We get (-2LLR = 1.246634). The quantity (-2LLR) should be distributed as chi-square with one degree of freedom, when  $H_0$  is true. So the P-value here is 0.2641963

```
1-pchisq(1.246634, df=1)
[1] 0.2641963
```

We can repeat the same test except change `mu=3.5` to some other value. After some trial and error we found when `mu=4.93` we get (-2LLR = 3.841911), with P-value approximately 0.05. Also when `mu=3.0497` we get (-2LLR = 3.841662), with P-value again approximately 0.05.

Guideline to find the two cut-off points: the P-value when `mu=NPMLE` will be 1. When the value assigned to `mu` gets further away from the NPMLE, the P-value will be smaller.

Thus the approximate 95% confidence interval for mean is [3.0497, 4.93] since the `mu` value inside this interval result a P-value larger then 0.05.

Now for other ‘means’. We try the parameter  $F(4)$  and the hypothesis  $H_0 : F(4) = 0.5$ . For this we need to define a function.

(Notice  $\int I_{[t \leq 4]} dF(t) = F(4)$ .) Either define it on the fly

```
> el.cen.EM(x,d,fun=function(x){as.numeric(x <= 4)}, mu=0.5)
```

or do it in two lines:

```
myfun <- function(x) { as.numeric(x <= 4) }  
el.cen.EM(x,d,fun=myfun, mu=0.5)
```

We get (-2LLR = 0.2920808). We can get a 95% confidence interval for  $F(4)$  by change the value `mu=0.5` above to some other values, until we get -2LLR = 3.841.... (two occasions, as lower and upper confidence limit).

We can also get a confidence interval for median (if  $M$  is median, then  $F(M) = 0.5$ ) if we keep `mu=0.5` fixed in the above but let  $F(4)$  change to  $F(3)$  (by redefine `myfun`), etc. to get an interval of  $T$  values such that the test for  $F(T) = 0.5$  have P-value larger then 0.05.

Look at the help pages of `el.cen.EM` or `el.cen.test` for further information.

Again, you can start testing from the NPMLE value (where the P-value is large) and push the null hypothesis value away from it until you get a P-value of 0.05.

Finally, an example of testing for the equality of two medians.

We take the AML data from the `survival` package. We can use either `el.cen.EM` or `empH.disc`. The only difference: `empH.disc` can handle left truncated data.

Here is the outline: We first test the median of both samples are equal to  $10 = \theta$  (say). Take the -2LLR from output of both test and calculate the sum. Last step is to minimize the sum over the value  $\theta$ .

Notice the minimization over  $\theta$  need only be over those values that are in between the two sample medians. Furthermore, -2LLR is constant between two data points. So the search of minimization is only over finitely many points.

The minimized sum of two -2LLR should have chi square with one degree of freedom under null hypothesis.

## 9 Left-truncated and right censored data

Left truncation happens when patients do not enter the study from the very beginning of the disease (late entry). The first time doctor sees them, the disease is already several weeks. The idea is that if they die within one week (say) then they will never enter the study. So the observation we have is conditional on the fact that they at least survive beyond the first week (or whatever the entering time). When the truncation/entering time is 0, then there is no truncation.

Left truncation can happen together with right censoring.

**Example:**  $(y_i, x_i) = (6, 17), (3, 13), (2, 9+), (0, 16)$ .

This means the first subject enters the study at 6 month after infection and die at 17 month after infection. etc. The third subject enters the study at 2 month and is right censored at 9 month. Notice the observations must have  $x_i > y_i$ .

In the package `survival`, the Cox regression function, `coxph()` can handle those data. You enter the data as `start, stop` format. And you can get the Nelson-Aalen or Product-limit estimator as

the baseline function for a null Cox model. See help page there for detail. Other functions in package `survival` do not seem to have this ability yet.

Several functions in package `emplik` can also handle left truncated, and right censored data. The function `e1.ltrc.EM()` in the package can handle left truncated, right censored observations and test a mean parameter. The function `emplikdisc.test()` can handle left truncated, right censored observations and test a weighted hazard parameter. See example below.

Suppose we have the data as below.

```
xpsy <- c(52, 59, 57, 50, 57, 59, 61, 61, 62, 67, 68, 69, 69, 65, 76)
ypsy <- c(51, 58, 55, 28, 25, 48, 47, 25, 31, 30, 33, 43, 45, 35, 36)
dpsy <- c(1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1)
fun4 <- function(x, theta) { as.numeric( x <= theta ) }
```

We want to test if  $\log[1 - F(T)]$  at  $T = 60$  is equal to  $-0.7$ , i.e.  $H_0 : 1 - F(60) = \exp(-0.7)$ , we do `emplikdisc.test(x=xpsy, d=dpsy, y=ypsy, K=-0.7, fun=fun4, theta=60)`

we should get  $-2 \log \text{lik ratio} = 0.3362712$ . Under  $H_0$  this should be distributed as  $\chi_1^2$ .

Now suppose we want to test if the mean of the survival times is 64 ( $H_0 : \mu = 64$ ). The function to use is `e1.ltrc.EM()`.

```
e1.ltrc.EM(y=ypsy, x=xpsy, d=dpsy, mu=64)
```

To get a confidence interval, we test different values as  $H_0$ , we find that whenever the value of  $\mu$  is within  $[58.78936, 67.81304]$ , we get a P-value  $> 0.05$ . This imply that the 95% confidence interval for the mean is the interval  $[58.78936, 67.81304]$ .

## 10 Accelerated Failure Time models

In the exponential regression model, take the formulation with log transformation.

$$\log Y_i = \beta Z_i + \epsilon_i.$$

If we do not assume the (standard) extreme-value distribution for the error term, and instead, let it be iid but arbitrary and unspecified, we get the (nonparametric) AFT model. This is similar to the usual linear model except the responses are censored and error distribution is arbitrary.

**Remark:** The Cox model was a generalization of exponential regression model in the hazard formulation, without using the log transform.

We want to estimation the parameter  $\beta$  with censored observations of  $T_i$ :  $T_i = \min(Y_i, C_i)$ ,  $\delta_i = I_{[Y_i \leq C_i]}$ .

In both estimation method below, we need the definition of residuals:

$$e_i(b) = \log T_i - bZ_i$$

Notice the order of the residuals depend on the  $b$ .

## 10.1 Buckley-James estimator

The Buckley-James estimator of the parameter  $\beta$  can be thought of as the nonparametric version of the EM algorithm: where the censored residual is replaced by expected values (E-step). Then followed by the usual regression M-estimation procedure.

The non-parametric nature of this procedure appears in both the E-step (where you do not have a parametric distribution for the residuals); and M-step (where you do not maximize a parametric likelihood, but use least squares etc.).

The calculation of least squares Buckley-James estimator can be found in the R function `bj()`, inside the `Design` library. The trustworthiness of the variance estimation from `bj()` is in doubt. Instead we recommend use empirical likelihood. The calculation of quantile Buckley-James estimator is available from the author.

The statistical inference can be obtained via empirical likelihood. I wrote some R functions for this purpose. See `bjtest()` etc. inside the `emplik` package.

## 10.2 Rank based estimator for AFT models

A nice summary of the rank based estimation can be found in Chapter 7 of Kalbfleisch & Prentice (2002); without the computing software and the empirical likelihood (which we shall discuss in the following).

The rank based estimator  $\hat{\beta}$  is the solution of the following estimating equation: (see Jin, Lin, Wei and Ying (2003))

$$0 = \sum_{i=1}^n \delta_i \phi(e_i(b)) [Z_i - \bar{Z}(e_i(b))]$$

where  $\bar{Z}(e_i(b))$  is the average of the covariates,  $Z_j$ , that the corresponding  $e_j(b)$  is still at risk at time  $e_i(b)$ , i.e.  $e_j(b) \geq e_i(b)$ . We assume the function  $\phi(\cdot)$  be either a constant (resulting a log-rank type estimator) or equal to the number of subjects at risk (Gehan type estimator) or other predictable functions.

**Remark:** For two sample case, rank estimation equation is same as the logrank test. Recall for Buckley-James estimator, it is equivalent to comparing the two means from two Kaplan-Meier estimators (t-test).

The numeric solution of this equation can be difficult, except with Gehan type estimator. In the Gehan case, the estimating functions are monotone in  $b$ , so the solution is easy.

I wrote an R function, based on the Splus code of Z. Jin, to compute the Gehan estimator. Download the package

```
rankreg_0.2-2
```

from CRAN.

Use the function `rankaft()` inside the package to obtain the solution/estimator. Caution: this function is memory hungry. For sample size around 600, you need 1G RAM.

For statistical inference after the estimator is computed, we can use empirical likelihood: This is very similar to the exponential regression case when we used the `offset()` command to get the confidence interval or P-value; except here the likelihood been replaced by the empirical likelihood.

I also wrote an R function for this purpose too. Use function `RankRegTest()` inside the package `emplik`; you need version 0.9-2 or later.

There is also the possibility of using re-sampling method to obtain inference, but is relatively slower compared to the likelihood based method.

**Score test:** The “score” test for the  $\hat{\beta}$  in the rank based estimation is also simple:

The idea is that if  $b = \beta_0$  then the residuals are iid. Then the covariate values carried by the residuals should have no relation to how large a residual is.

Condition on the residual values that are  $> t$ , suppose there are  $k$  such residuals. They carry  $k$  covariates. The covariate that corresponding to the smallest residual out of the  $k$  remaining residuals should be equally likely to be any one of the  $k$  corresponding covariates.

By the above argument, the null distribution of the  $Z_i$  is uniform among  $Z_j$  such that  $e_j \geq e_i$ . The second moment of the term  $Z_i - \bar{Z}_i$  is then

$$\sigma_i^2 = 1/k \sum_j (Z_j - \bar{Z}_i)^2$$

This is the variance of (one term of) the “score”. Successive term may be view as (conditionally) uncorrelated.

This variance of the score is computed, also a related chi square (score test) statistic is returned by the function `RankRegV` inside the `rankreg` package.

Remark: A similar argument also works for Cox ‘score’. We only need to treat the observations there as exponential random variables.

### 10.3 Case-weighted regression estimation

If we assume a random design, we may view the  $(q+1)$  dimension vector  $(Y_i, X_i)$  as iid. In this case we can use the weighted estimation method for the slope of the linear model. Sometimes the name IPCW (inverse-probability-of-censor-weight) is used to describe this weighting scheme.

See the help file for the functions `WRegEst` and `WRegTest` inside `emplik` package, you need version  $\geq 0.9-3$ .

## 11 Residual mean and residual median

For long term survival patients, the residual mean and/or residual median are of interest.

Inside `emplik` version 0.9-4 or later there is a function `MMres()` that does the estimation. (based on the Kaplan-Meier. To get the confidence interval for the residual mean or median, we again use empirical likelihood.

First the packages `emplik` and `survival` need to be loaded into R [package `survival` is only needed here to supply the data set `cancer`].

```
> data(cancer)
> time <- cancer$time
> status <- cancer$status-1
> MMRtime(x=time, d=status, age=365.25)
$MeanResidual
[1] 275.9997
```

```
$MedianResidual
[1] 258.75
```

The following is the result from testing the mean residual times through the confidence interval approach. First we need to define the  $g$  function for the mean residual life.

```
> mygfun <- function(s, age, muage) {as.numeric(s >= age)*(s-(age+muage))}
> el.cen.EM2(x=time, d=status, fun=mygfun, mu=0, age=365.25, muage=234.49389)$Pval
[1] 0.1000000
> el.cen.EM2(x=time, d=status, fun=mygfun, mu=0, age=365.25, muage=323.1998)$Pval
[1] 0.1
```

Therefore the 90% confidence interval for mean residual time at 365.25 days is [234.49389, 323.1998].

For testing of the median residual time, we first need to code the  $g_\theta$  function (defined in the paper by Kim and Zhou) and then use `el.cen.EM2` to test.

```
> mygfun2 <- function(s, age, Mdage) {as.numeric(s<=(age+Mdage))-0.5*as.numeric(s<=age) -0.5}
> el.cen.EM2(x=time, d=status, fun=mygfun2, mu=0, age=365.25, Mdage=184.75)$Pval
[1] 0.1135797
> el.cen.EM2(x=time, d=status, fun=mygfun2, mu=0, age=365.25, Mdage=321.7499)$Pval
[1] 0.1192006
```

This implies a 90% confidence interval for the median residual time is [184.75, 321.7499]. Note we do not get an exact p-value of 0.1 here. To overcome this we, suggest smoothing. For the smoothed quantile, first define a (linearly) smoothed  $g$  function, then find the confidence limits.

```
> mygfun22 <- function(s, age, Mdage) {
myfun7(s, theta=(age+Mdage), epi=1/20)-0.5*myfun7(s, theta=age, epi=1/20) -0.5 }
> myfun7 <- function(x, theta=0, epi) {
if(epi <= 0) stop("epi must > 0")
u <- (x-theta)/epi
return( pmax(0, pmin(1-u, 1)) ) }
> el.cen.EM2(x=time, d=status, fun=mygfun22, mu=0, age=365.25, Mdage=184.7416765)$Pval
[1] 0.1000000
> el.cen.EM2(x=time, d=status, fun=mygfun22, mu=0, age=365.25, Mdage=321.71153607)$Pval
[1] 0.1000000
```

The result is very similar.

## Problems

1: If I use the following code to generate right censored data, (as one student used in earlier course) what is the Kaplan-Meier estimator computed from the data suppose to estimate?

```
xcen <- rexp(100)
dcen <- sample(c(0,1), replace=TRUE)
```

2: Write some R code that can generate piecewise exponential r.v. (the number of pieces, the cut-off points and value of the hazards of each piece as input).

## References

Theaune, T. and Grambach. (2001) *Modeling Survival Data - Extending the Cox model* Springer.

When using the Cox model to predicting a specific person's survival, see P. 265 of Theaune and Grambach.

Zhou, Mai (2000) Understanding the Cox model with time-change covariates. *The American Statistician*,

Kalbfleisch & Prentice, *The Statistical analysis of failure time data*. Wiley. second edition (2002)