Survival Analysis in R

David Diez

This document is intended to assist an individual who has familiarity with R and who is taking a survival analysis course. Specifically, this was constructed for a biostatistics course at UCLA. Many theoretical details have been intentionally omitted for brevity; it is assumed the reader is familiar with the theory of the topics presented. Likewise, it is assumed the reader has basic understanding of R including working with data frames, vectors, matrices, plotting, and linear model fitting and interpretation.

Functions that are introduced will only have the key arguments mentioned and discussed. Most functions have several other (optional) arguments, however, many of these will not be useful for an introductory course. The functions, with the exception of those I wrote, have well-written descriptions that specify each of the potential arguments and their use. The functions I have written include documentation on the following web site:

`<http://www.stat.ucla.edu/~david/teac/surv/>`

Ideally, this survival analysis document would be printed front-to-back and bound like a book. No topics run over two pages and those that are two pages would then be on opposing pages, making a topic's introduction available without the need to flip back and forth between pages (unless the reader forgets a previous topic, then some flipping may be necessary). A more thorough look at Cox PH models beyond what is discussed here is available in a guide constructed by John Fox, which is listed in the References.

Table of Contents

R packages :: `survival` **&** `KMsurv` **::** The package `survival` is used in each example in this document. Every data set used is found in the package `KMsurv`, which are the data sets from Klein and Moeschberger’s book. To obtain one or both of these packages (if they were not previously installed), use

```
> install.packages(’survival’)
> install.packages(’KMsurv’)
```

To load the libraries, use

```
> library(survival)
> library(KMsurv)
```

To view available data sets, use `library(help=KMsurv)`. To load a data set, use the function `data()`:

```
> data(aids)
> aids
    infect induct adult
1     0.00   5.00     1
2     0.25   6.75     1
...
294   7.00   0.75     0
295   7.25   0.25     0
```

The ’`...`’ denotes output omitted for brevity. Occasionally the ’`...`’ will itself be omitted. If the packages `survival` and `KMsurv` are both loaded, typing in the sample code in this document’s examples will allow the reader to reproduce the results. [[As mentioned, this assumes the libraries are loaded. Also, any variables already in the workspace with names common with any commands, variables, or column names in the examples must be removed using `rm()` to ensure the examples will run.]]

To make columns available for use as variables, use `attach()`:

```
> attach(aids)
> infect
  [1] 0.00 0.25 0.75 0.75 0.75 1.00 1.00 1.00 1.00 1.25 1.25 1.25 1.25 1.50
...
[281] 5.25 5.25 5.50 5.50 5.50 5.75 6.00 6.00 6.25 6.25 6.50 6.75 6.75 7.00
[295] 7.25
```

Detaching the data set when done is a good habit and can prevent errors within R (since two data sets may have common column names):

```
> detach(aids)
```

Survival object :: `Surv(time, time2, event, type)` **::** Before complex functions may be performed, the data has to be put into the proper format: a survival object. In particular, the constructions that will be outlined here are based on the data that is right-censored or left-truncated and right-censored, and the function `Surv()` will be used to construct these survival objects.

Right-censored :: For right-censored data, only the `time` and `time2` arguments need be filled:

```
> data(tongue); attach(tongue)   # the following will not affect computations

The following object(s) are masked from package:stats :

 time

> # subset for just the first group by using [type==1]
> my.surv.object <- Surv(time[type==1], delta[type==1])
> my.surv.object
 [1]    1     3     3     4    10    13    13    16    16    24    26    27    28    30
...
[43] 101+ 104+ 108+ 109+ 120+ 131+ 150+ 231+ 240+ 400+
> detach(tongue)
```

Provided the arguments of `Surv()` are filled in order, the argument labels need not be specified. Here `time` is a vector of the event or censoring times (whichever occurs first) and `delta` is a vector of $\{\delta_i\}$, the indicator variable denoting if the event was observed (`1`) or censored (`0`). For this indicator variable, `1` and `0` may be replaced with `TRUE` and `FALSE`, respectively, if that is preferable.

Left-truncated and right-censored :: For left-truncated and right-censored data, the first three arguments in `Surv()` will be filled:

```
> data(psych); attach(psych)
> my.surv.object <- Surv(age, age+time, death)
> my.surv.object
 [1] (51,52 ] (58,59 ] (55,57 ] (28,50 ] (21,51+] (19,47 ] (25,57 ]
...
[22] (29,63+] (35,65+] (32,67 ] (36,76 ] (32,71+]
> detach(psych)
```

The left-truncation time is entered first as the variable `time`; the event time (or censoring time) is `time2`; the indicator variable for whether the event was observed, $\{\delta_i\}$, is assigned to `event`.

Other options :: To do interval censoring, use `time` for the left ends of the intervals, `time2` for the right ends of the intervals, and `type="interval2"`; `event` is not used for interval censoring. There are more types of survival data that may be transformed into a survival object, however, they will not be discussed here. Note that not all functions will accept all types of data. For example, interval-censored data will not be accepted by the majority of the functions in `survival`.

Kaplan-Meier estimate and pointwise bounds :: The Kaplan-Meier estimate of the survival function, $S(t)$, corresponds to the non-parametric MLE estimate of $S(t)$. The resulting estimate is a step function that has jumps at observed event times, t_i. In general, it is assumed the t_i are ordered: $0 < t_1 < t_2 < \cdots < t_D$. If the number of individuals with an observed event time t_i is d_i, and the number of individuals at risk (ie, who have not experienced the event) at a time *before* t_i is Y_i, then the Kaplan-Meier estimate of the survival function and its estimated variance is given by

$$\begin{aligned} \hat{S}(t) &= \begin{cases} 1 & \text{if } t < t_1 \\ \prod_{t_i \leq t} \left[1 - \frac{d_i}{Y_i}\right] & \text{if } t_1 \leq t \end{cases} \\ \widehat{V}[\hat{S}(t)] &= \left[\hat{S}(t)\right]^2 \hat{\sigma}_S^2(t) = \left[\hat{S}(t)\right]^2 \sum_{t_i \leq t} \frac{d_i}{Y_i(Y_i - d_i)} \end{aligned}$$

The pointwise confidence bounds (not the confidence bands) for the `"plain"` (linear) and `"log-log"` options provided in R are given by

$$\left(\hat{S} - Z_{1-\alpha/2}\hat{\sigma}_S(t)\hat{S}(t), \hat{S} - Z_{1+\alpha/2}\hat{\sigma}_S(t)\hat{S}(t)\right)$$

$$\left(\hat{S}^{1/\theta}(t), \hat{S}^{\theta}(t)\right), \text{ where } \theta = \exp\left\{\frac{Z_{1-\alpha/2}\hat{\sigma}_S(t)}{\log \hat{S}(t)}\right\}$$

R code :: `survfit(formula, conf.int = 0.95, conf.type = "log")` **::** The function `survfit()` is used to find the Kaplan-Meier estimate of the survival function. There are three arguments of particular interest: `formula`, `conf.int`, and `conf.type`. `formula` will be a survival object (and can be made more complex), and it is the only required input:

```
> data(tongue); attach(tongue)
> my.surv <- Surv(time[type==1], delta[type==1])
> survfit(my.surv)
Call: survfit(formula = my.surv)

      n  events  median 0.95LCL 0.95UCL
     52      31      93      67     Inf
```

The argument `conf.int` is the confidence interval level and ranges between `0` and `1` with the default `0.95`. `conf.type` is the type of confidence interval or, more accurately, the transformation used to construct the confidence interval. The default is `'log'`, which equates to the transformation function $g(t) = \log(t)$, **not** $g(t) = \log(-\log(t))$, which is `'log-log'`. A linear confidence interval is created by using the argument `conf.type='plain'`. At this time, there is not a simple way to compute the confidence interval for the arcsine-squareroot transformation except by using output from `survfit()`. [[Using the output information from `survfit()` (with any confidence interval type), which is discussed below, and the formula provided in Klein and Moeschberger's text should make this computation rather straightforward.]]

The simple commands above would yield a survival function fit, which may be obtained either by looking at `summary(survfit(my.surv))` or by looking at the function's hidden output. This hidden information is where the Kaplan-Meier estimate, a 95% confidence bound, along with the $\{t_i\}$, $\{d_i\}$, and $\{Y_i\}$ are saved. All of this data will be output by looking at the summary. To get the output in individual vectors, use the following commands (the actual outputs are omitted for brevity):

```
> my.fit <- survfit(my.surv)
> summary(my.fit)$surv     # outputs the Kaplan-Meier estimate at each t_i
> summary(my.fit)$time     # {t_i}
> summary(my.fit)$n.risk   # {Y_i}
> summary(my.fit)$n.event  # {d_i}
> summary(my.fit)$std.err  # standard error of the K-M estimate at {t_i}
> summary(my.fit)$lower    # lower pointwise estimates (alternatively, $upper)
```

The Kaplan-Meier estimate may be plotted using `plot(my.fit)`. Typical arguments in the plot function may be used to improve the graphical aesthetics:

```
> plot(my.fit, main="Kaplan-Meier estimate with 95% confidence bounds",
+    xlab="time", ylab="survival function")
```

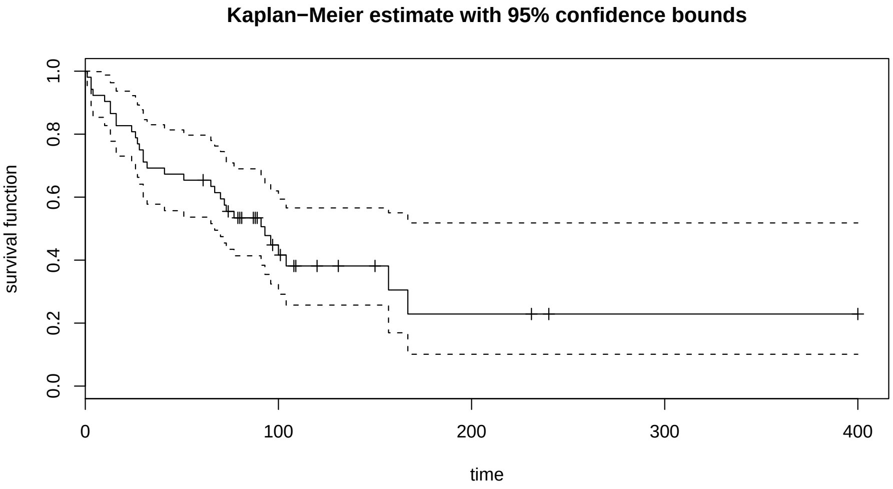

Figure 1: Sample output where only the title, x-axis and y-axis labels have been specified.

One potential issue is when different groups have their data mixed together with a separate vector containing the 'key' to which event times and censoring values correspond to which groups. There are two good options; subset the data like in `my.surv` or specify groups in the formula argument:

```
> my.fit1 <- survfit( Surv(time, delta) ~ type )
```

If `formula` is made more complex in this way, the output vectors for each type are merged into one and an output vector, `summary(my.fit)$strata`, is created and designates which components of the output correspond to which types. This vector may be used to do manual computations within a group via subsetting.

```
> detach(tongue)
```

Confidence bands :: The confidence intervals constructed on the previous pages are only pointwise confidence intervals. Confidence bands, which are a bit more generalized, can also be constructed. These bands would be bounds on an entire range of time. That is, for a 95% confidence band, the probability that any part of the true curve is out of the confidence bands is 0.05. No functions within the package `survival` will create confidence bands (in the sense that is being discussed here), however, a function that will create the bands can be downloaded:

```
> source('http://www.stat.ucla.edu/~david/teac/surv/conf-bands.R')
```

The form of the function is given as

```
conf.bands(surv.object, conf.type='plain', type='ep', tL=NA, tU=NA)
```

Here, `surv.object` is a survival object, `conf.type` may be `'plain'`, `'log-log'`, or `'asin-sqrt'`, `type` may be either `'ep'` or `'hall'` (for Hall-Wellner bands), and finally `tL` and `tU` are optional to limit the scope of the confidence bands' meaning to hold from `tL` to `tU`. The appropriate value, $c_\alpha(a_L, a_U)$ for EP or $k_\alpha(a_L, a_U)$ for Hall-Wellner, must be input when requested by `conf.bands()`:

```
> data(bmt); attach(bmt)
> my.surv <- Surv(t2[group==1], d3[group==1])
> my.cb <- conf.bands(my.surv, type='hall', 100, 600)
   a_L: 0.1052632   |   a_U: 0.594237
   Enter confidence coefficent: 1.3211
> plot(survfit(my.surv), xlim=c(100, 600), xlab='time',
+   ylab='Estimated Survival Function', main='Reproducing
+   Confidence Bands for Example 4.2 in Klein/Moeschberger')
```

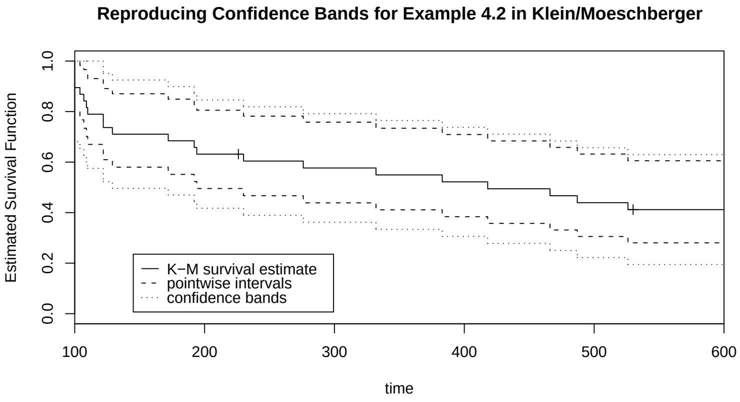

Figure 2: Recall that the default for the pointwise confidence bands is a log transformation, so the pointwise confidence intervals will not be symmetric.

```
> lines(my.cb$time, my.cb$lower, lty=3, type='s')
> lines(my.cb$time, my.cb$upper, lty=3, type='s')
> legend(locator(1), legend=c('K-M survival estimate',
+   'pointwise intervals','confidence bands'), lty=1:3)
> detach(bmt)
```

Cumulative Hazard :: The cumulative hazard function and the survival function are related in the following way for continuous data:

$$S(t) = \exp\{-H(t)\}$$

This offers an immediate estimation method of $H(t)$ by taking the negative of the log of $\hat{S}(t)$: $\hat{H}(t) = -\log \hat{S}(t)$. A second method to estimate $H(t)$ is using the Nelson-Aalen estimator and its variance:

$$\tilde{H}(t) = \sum_{t_i \leq t} \frac{d_i}{Y_i} \text{ (assuming } t_1 \leq t\text{, otherwise it is 0)}, \qquad \sigma_H^2(t) = \sum_{t_i \leq t} \frac{d_i}{Y_i^2}$$

R code :: There is not a function from `survival` that will automatically compute either form of the cumulative hazard function, but this can be done by hand using output from `survfit()`:

```
> data(tongue); attach(tongue)
> my.surv <- Surv(time[type==1], delta[type==1])
> my.fit <- summary(survfit(my.surv))
> H.hat <- -log(my.fit$surv); H.hat <- c(H.hat, H.hat[length(H.hat)])
```

Then, using `H.hat` with `my.fit$time`, a plot (or table) can be made. Alternatively, the Nelson-Aalen estimator may be constructed nearly as easily:

```
> h.sort.of <- my.fit$n.event / my.fit$n.risk
> H.tilde <- vector()
> for(i in 1:length(h.sort.of))   H.tilde[i] <- sum(h.sort.of[1:i])
> H.tilde <- c(H.tilde, H.tilde[length(H.tilde)])
> plot(c(my.fit$time, 250), H.hat, xlab='time', ylab='cumulative hazard',
+   main='comparing cumulative hazards', ylim=range(c(H.hat, H.tilde)), type='s')
> points(c(my.fit$time, 250), H.tilde, lty=2, type='s')
> legend(locator(1), legend=c("H.hat","H.tilde"), lty=1:2)
```

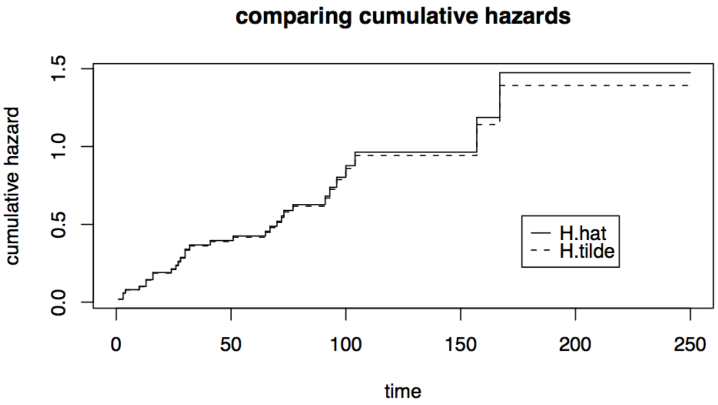

Figure 3: Comparing the two cumulative hazard function estimates. 250 was appended to the time and the hazard functions were extended for aesthetics.

```
> detach(tongue)
```

Mean and median estimates with bounds :: The median survival time is defined to be the time $t_{0.5}$ such that $S(t_{0.5}) = 0.5$. Given an estimate of the survival function using Kaplan-Meier, this may be obtained graphically by drawing a horizontal line at 0.5. The estimate is where $\hat{S}(t)$ crosses 0.5, and the confidence bounds for $t_{0.5}$ are given by the points at which this horizontal line crosses over the confidence bounds of $\hat{S}(t)$.

The mean survival time (and its respective estimate) is given by

$$\mu = \int_0^\infty S(t)dt, \quad \hat{\mu} = \int_0^\infty \hat{S}(t)dt$$

Because $S(t)$ (and/or $\hat{S}(t)$) may not converge to zero, the estimate may diverge. A more commonly used definition is $\mu_\tau = \int_0^\tau S(t)dt$ with the corresponding estimate $\hat{\mu}_\tau = \int_0^\tau \hat{S}(t)dt$, where τ is a finite positive number. A possible choice of τ is the largest observed or censored time. Letting t_i, Y_i, d_i, and D be as described in the Kaplan-Meier estimate, the estimated variance of $\hat{\mu}_\tau$ is

$$\hat{V}(\hat{\mu}_\tau) = \sum_{i=1}^{D} \left[\int_{t_i}^{\tau} \hat{S}(t)dt \right]^2 \frac{d_i}{Y_i(Y_i - d_i)}$$

R code :: `survfit(formula, conf.int = 0.95, conf.type = "log")` **::** The median and its bounds may be estimated using `survfit()` in the same manner as finding $\hat{S}(t)$, however, this time the immediate output is viewed instead of the summary output:

```
> data(drug6mp); attach(drug6mp)
> my.surv <- Surv(t1, rep(1, 21))   # all placebo patients observed
> survfit(my.surv)
Call: survfit(formula = my.surv)

      n  events  median 0.95LCL 0.95UCL
     21      21       8       4      12
```

Using `survfit()` in conjunction with `print()`, the mean survival time and its standard error may be obtained:

```
> print(survfit(my.surv), show.rmean=TRUE)
Call: survfit(formula = my.surv)

         n     events     rmean se(rmean)    median   0.95LCL   0.95UCL
     21.00      21.00      8.67      1.38      8.00      4.00     12.00
```

The `show.rmean=TRUE` argument is necessary to obtain the mean and its standard error. The computed estimate automatically sets τ, the integral's upper bound, as the largest observed or censored time.

```
> detach(drug6mp)
```

Tests for two or more samples :: Given two or more samples, is there a difference between the survival times? Setting up hypotheses for this problem,

- $H_0 : h_1(t) = h_2(t) = \cdots = h_n(t)$ for all t.
- $H_A : h_i(t_0) \neq h_j(t_0)$ for at least one pair i, j and time t_0.

Let

- t_i be times where events are observed (assume these are ordered and there are D such times),
- d_{ik} be the number of observed events from group k at time t_i,
- Y_{ik} be the number of subjects in group k that are at risk at time t_i,
- $d_i = \sum_{j=1}^{n} d_{ij}$,
- $Y_i = \sum_{j=1}^{n} Y_{ij}$, and
- $W(t_i)$ be the weight of the observations at time t_i.

Then to test the hypothesis above, a vector Z is computed, where

$$Z_k = \sum_{i=1}^{D} W(t_i) \left[d_{ik} - Y_{ik} \frac{d_i}{Y_i} \right]$$

The covariance matrix $\widehat{\Sigma}$ is also computed from the data (the formulas to compute this are found on page 207 of Klein and Moeschberger's book). Then the test statistic is given by $X^2 = Z' \hat{\Sigma}^{-1} Z$, which, under the null hypothesis, is distributed as a χ^2 distribution with n degrees of freedom.

R code :: `survdiff(formula, rho=0)` **::** To check the null hypothesis, use `survdiff()`. The first argument is a survival object against a categorical covariate variable that is typically a variable designating which groups correspond to which survival times. The output directly from `survdiff()` is of most use (`summary()` of a `survdiff()` object does not provide much information).

```
> data(btrial); attach(btrial)
> survdiff(Surv(time, death) ~ im)   # output omitted
```

The second argument shown, `rho`, designates the weights according to $\hat{S}(t)^\rho$ and may be any numeric value. The default is `rho=0`, which corresponds to the log-rank test. When `rho=1`, this is the "Peto & Peto modification of the Gehan-Wilcoxon test":

```
> survdiff(Surv(time, death) ~ im, rho=1)   # output omitted
```

To give greater weight to the first part of the survival curves, use `rho` larger than `0`. To give weight to the later part of the survival curves, use `rho` smaller than `0`. The output of `survdiff` is relatively self-explanatory. A χ^2 statistic is computed along with a p-value.

```
> detach(btrial)
```

Cox proportional hazards model, constant covariates :: The basic Cox PH model attempts to fit survival data with covariates z to a hazard function of the form

$$h(t|z) = h_0(t)\exp\left\{\beta' z\right\}$$

where β is an unknown vector. $h_0(t)$ is the *baseline hazard*, which is non-parametric. Primary interest lies in finding the parameter β, which is found by solving the partial likelihood:

$$L(\beta) = \prod_{i=1}^{D} \frac{\exp\left[\beta' z_{(i)}\right]}{\sum_{j \in R(t_i)} \exp\left\{\beta' z_j\right\}}, \ R(t_i) \text{ is the 'risk set' at time } t_i$$

Given the estimate of β, $\hat{\beta}$ (a vector), along with the covariance matrix of the estimates, $\hat{I}^{-1}$, $\hat{\beta} \sim AN(\beta, \hat{I}^{-1})$ holds approximately since $\hat{I} \to I$ as $n \to \infty$. This approximation makes doing "local tests" possible (a local test checks a null hypothesis that is not the global null). A local null hypothesis can usually be put into matrix form, $C\beta = d$, where C is a $q \times p$ matrix of full rank and d is a vector of length q. Under this setup, the test statistic is

$$X_W^2 = \left(C\hat{\beta} - d\right)' \left[C\hat{I}^{-1}C'\right]^{-1} \left(C\hat{\beta} - d\right),$$

which under the null hypothesis follows χ_q^2 (this is the Wald test).

Beyond obtaining test p-values, there may be interest in the survival function for particular covariates. If the estimate of the baseline survival function, $\hat{S}_0(t)$, is provided, then the estimate of the survival function for an individual with covariates z_k may be obtained via

$$\hat{S}(t|z_k) = \left[\hat{S}_0(t)\right]^{\exp\left(\hat{\beta}' z_k\right)}$$

R code :: `coxph(formula, method)` **::** The function `coxph()` fits a Cox PH model to the supplied data. The two arguments of particular interest are `formula` and `method`. `formula` will be almost identical to fitting a linear model (via `lm()`) except that the response variable will be a survival object instead of a vector. An example of a simple setup is the following:

```
> data(burn); attach(burn)
> my.surv <- Surv(T1, D1)
> coxph(my.surv ~ Z1 + as.factor(Z11), method='breslow')   # output omitted
```

Two covariates have been used in this example. The second argument listed, `method`, is used to specify how to handle ties. The default is `'efron'`. Other options are `'breslow'` and `'exact'`.

The bulk of useful information from `coxph()` comes from the summary, which includes

- estimates of the β_k, including standard errors and p-values for each test $H_0 : \beta_k = 0$ with the other $\beta_j = \hat{\beta}_j$,
- estimate of the risk ratio with confidence bounds, and
- p-values for likelihood ratio, Wald and score tests for the global null, $H_0 : \beta_i = 0$ for all i.

More complex hypotheses may be checked using other output from the model fit:

```
> coxph.fit <- coxph(my.surv ~ Z1 + as.factor(Z11), method='breslow')
> coxph.fit$coefficients  # may use my.fit$coeff instead
> coxph.fit$var           # I^(-1), estimated cov matrix of the estimates
> coxph.fit$loglik        # log-likelihood for alt and null MLEs, resp.
```

A rudimentary function for doing local checks (using C and d) has been written and placed online. It may be loaded using the following command:

```
> source('http://www.stat.ucla.edu/~david/teac/surv/local-coxph-test.R')
```

The function loaded is called `local.coxph.test` and its format is

```
    local.coxph.test(coxph.fit, pos, C=NA, b=NA, sign.digits=3)
```

Here `coxph.fit` is an output from `coxph()`, `pos` is a vector of the coefficients (their positions) to include in the local test, `C` and `d` are the matrices described on the previous page. The last parameter is the number of significant figures to include in the output with the default set to 3. This function may be convenient for determining whether a factor variable should be included in a model. For example,

```
> coxph.fit
...
                  coef exp(coef) se(coef)     z     p
Z1               0.497     1.644    0.208  2.38 0.017
as.factor(Z11)2 -0.877     0.416    0.498 -1.76 0.078
as.factor(Z11)3 -1.650     0.192    0.802 -2.06 0.040
as.factor(Z11)4 -0.407     0.666    0.395 -1.03 0.300
...
> local.coxph.test(coxph.fit, 2:4)
[1] 0.103
```

In this example, `Z11` was a factor variable. To check the p-value of whether it should be included, a test on the second through fourth parameters that were fit in `my.fit` had to be checked (namely, a global test on `as.factor(Z11)2`, `as.factor(Z11)3`, and `as.factor(Z11)4`). The p-value of whether to include `Z11` is then 0.103.

To obtain the baseline survival function from a Cox PH model, apply `survfit()` to `coxph()`:

```
> my.survfit.object <- survfit(coxph.fit)
```

The output from `survfit()` are all the same as when it was applied to a survival object (and it may be plotted, just like the previous `survfit()` objects).

```
> detach(burn)
```

Cox proportional hazards model, time-dependent covariates :: Time-independent covariates are easy to work with in R, however, working with time-dependent covariates is an exercise in organization. There is a way to work around the fact that there aren't functions in R that will directly accept time-dependent covariates: use left-truncation liberally. This works on the basic principle that if there is one right-censored observation, say 45+, it is the same as having two observations that are left-truncated right-censored, $(0, 12+]$ and $(12, 45+]$, where the choice of splitting the observation at 12 was arbitrary. Furthermore, these intervals could be broken down further. This is the workaround used in R to make time-dependent variables.

The documentation by John Fox listed in the references goes into greater depth than I will go into here and I suggest reading his document (specifically, pages 7-11). Essentially, the reader is taking each observation, cutting it up into many left-truncated observations, and then making the covariates, both those that are time-dependent and time-independent, be appropriate for each interval.

Writing up code for this in R will likely consist of the following steps:

- For each individual, break their observation up at appropriate time points into k_i intervals. For each interval, note the start and stop times, whether the event was observed or not, and the covariates (both time-independent and time-dependent). Ideally, each different variable will be stored in a vector and position `i` in each vector will correspond to the same individual and truncated/censored interval.

- Now the left-truncated data may be constructed by using the start, stop, and observation vectors plus the model may be constructed using `coxph()`.

This is a very brief discussion of how this may be done and a review of Fox's description is highly recommended. Fox is more thorough and his examples are certainly useful in working out how one may consider approaching this type of problem.

One case of interest is creating a new time-dependent covariate, Z_i^t, from another covariate, Z_i. This may be done by using a transformation on the time and multiply the result by Z_i:

$$Z_i^t = Z_i * transformation(t)$$

A common transformation is log but others may be used. A function to create a type of variable of this form and also to run a Cox PH model with it is the following:

```
time.dep.coxph(d.f, col.time, col.delta, col.cov, td.cov, transform=log,
   method='efron', output.model=TRUE, output.data.frame=FALSE, verbose=TRUE)
```

The function may be loaded using

```
> source('http://www.stat.ucla.edu/~david/teac/surv/time-dep-coxph.R')
```

Here,

- `d.f` is a data frame with all of the original data,

- `col.time` is the column number (or name) of the event/censoring times,

- `col.delta` is the column number/name indicating which events were observed,
- `col.cov` is a vector of the column numbers/names of the covariates to be considered in the model, and
- `td.cov` is the column number/name of the covariate Z_i to be used to create the time-dependent covariate. Z_i must be a vector with only 2 unique numerical values. (ex, Z_i is a vector of `0`'s and `1`'s).

The following arguments have defaults, as are specified above:

- `transform` is the type of transformation to be used on time,
- `method` is the argument to be given for `method` in `coxph()`,
- `output.model` indicates whether the Cox PH model should be included in the output,
- `output.data.frame` indicates whether the time-dependent data frame that was created should be output (this may be useful if many closely related models are going to be checked), and
- `verbose` indicates whether the user should be notified at how far along the function is at computing the data frame to be used in the Cox PH model (this is useful to see how fast the function is running).

Below is an example of how to use this function:

```
> data(burn); attach(burn)
> source('http://www.stat.ucla.edu/~david/teac/surv/time-dep-coxph.R')
> td.coxph <- time.dep.coxph(burn, 'T1', 'D1', 2:4, 'Z1', verbose=F)
```

The covariates in the output will be of the same order as specified in 'col.cov'.

```
> td.coxph   # some model output is omitted for brevity
...
                coef exp(coef) se(coef)      z      p
Z1            1.3569     3.884    0.719  1.886 0.0590
Z2            0.6899     1.994    0.231  2.981 0.0029
Z3            0.0496     1.051    0.302  0.164 0.8700
time.dep.cov -0.3383     0.713    0.312 -1.085 0.2800
```

The default output of `time.dep.coxph()` is the output of `coxph()`. It has all the usual outputs of `coxph()`. For example, log-likelihood:

```
> td.coxph$loglik
[1] -420.4825 -411.8814
> detach(burn)
```

Accelerated failure-time models :: An accelerated failure-time (AFT) model is a parametric model with covariates that takes the form $S(t|z) = S_0(t\exp(\gamma' z))$, where γ is a vector of parameters and z is the vector of covariates. This model essentially puts individuals with different covariates on different time scales. The model assumes the log of failure time, $\log X$, is in a linear relationship with a mean μ, the covariates and parameters $\gamma' z$, and an error term σW, where W takes a particular distribution.

$$\log X = \mu + \gamma' z + \sigma W$$

The main choice to be made is which distribution to use. The options discussed in Klein and Moeschberger are shown in the table below:

distribution	df	included in `survival`?
exponential	1	yes
Weibull	2	yes
lognormal	2	yes
log logistic	2	yes
generalized gamma	3	no

R code :: `survreg(formula, dist='weibull')` :: The function `survreg()` is used for AFT modeling. The first argument is `formula`, which is a typical formula argument. The argument `dist` has several options (`'weibull'`, `'exponential'`, `'gaussian'`, `'logistic'`, `'lognormal'`, and `'loglogistic'`) and is the parametric model used. The example code below follows Example 12.2 in Klein and Moeschberger:

```
> data(larynx); attach(larynx)   # output omitted
> sr.fit <- survreg(Surv(time, delta) ~ as.factor(stage) + age, dist='weibull')
> summary(sr.fit)
...
                    Value Std. Error      z        p
(Intercept)        3.5288     0.9041  3.903 9.50e-05
as.factor(stage)2 -0.1477     0.4076 -0.362 7.17e-01
as.factor(stage)3 -0.5866     0.3199 -1.833 6.68e-02
as.factor(stage)4 -1.5441     0.3633 -4.251 2.13e-05
age               -0.0175     0.0128 -1.367 1.72e-01
Log(scale)        -0.1223     0.1225 -0.999 3.18e-01

Scale= 0.885

Weibull distribution
Loglik(model)= -141.4   Loglik(intercept only)= -151.1
Chisq= 19.37 on 4 degrees of freedom, p= 0.00066
Number of Newton-Raphson Iterations: 5
```

Here `(Intercept)` corresponds to the estimate of μ, `Log(scale)` corresponds to the estimate of $\log\sigma$, and the other estimates correspond to covariate coefficient estimates. Also of significant interest is the log-likelihood, which may be used to find the AIC:

$$AIC = -2\log L + 2p$$

In this case, the AIC is equal to 286.8 ($p = 2$). The AIC may be used for model selection (fit several models and determine which has the lowest AIC). For example, we may fit an exponential model on the same data to find the AIC of this model and compare:

```
> sr.fit.exp <- survreg(Surv(time, delta) ~ as.factor(stage) + age, dist=’exponential’)
> summary(sr.fit.exp)
...
                    Value Std. Error      z        p
(Intercept)        3.7550     0.9902  3.792 1.49e-04
as.factor(stage)2 -0.1456     0.4602 -0.316 7.52e-01
as.factor(stage)3 -0.6483     0.3552 -1.825 6.80e-02
as.factor(stage)4 -1.6350     0.3985 -4.103 4.08e-05
age               -0.0197     0.0142 -1.388 1.65e-01

Scale fixed at 1

Exponential distribution
Loglik(model)= -141.9   Loglik(intercept only)= -151.1
Chisq= 18.44 on 4 degrees of freedom, p= 0.001
Number of Newton-Raphson Iterations: 4
```

The AIC for the exponential is 285.8 ($p = 1$), which is lower than the AIC for the Weibull, which suggests the exponential model may be more appropriate.

Alternatively, it could have been noted that the exponential model is a special case of the Weibull model; when $\sigma = 1$, the Weibull model is an exponential. This suggests two alternatives for model selection:

- a likelihood ratio test could be run to check if the parameter σ is necessary, or
- it could have been noted that `Log(scale)` (ie, $\log\hat{\sigma}$) is not statistically significant relative to 0, again suggesting the exponential model may be more appropriate.

Some of the hidden output from a `survreg()` object:

```
> # the output is omitted from each command below
> sr.fit.exp$coeff    # covariate coefficients
> sr.fit.exp$icoef    # intercept and scale coefficients
> sr.fit.exp$var      # variance-covariance matrix for coeff and icoef
> sr.fit.exp$loglik   # log-likelihood
> sr.fit$scale        # not using sr.fit.exp since that is, by default, 1
```

Detaching the data:

```
> detach(larynx)
```

References ::

[1] Lumley, Thomas, 2007. The Survival Package (R help guide).

[2] Fox, John 2002. Cox Proportional-Hazards Regression for Survival Data. Appendix to An R and S-PLUS Companion to Applied Regression.

[3] Klein, John P., and Melvin L. Moeschberger. Survival Analysis: Techniques for Censored and Truncated Data. New York: Springer, 2003.