

Demography 213

Messing with the assumptions underlying the Cox Proportional Hazard Model

Carl Mason
carlm@demog.berkeley.edu

November 7, 2011

Contents

1	Introduction	1
2	Toward a general continuous-time-lx data simulating function	2
2.1	Plotting coxph results	8
2.2	Random Error	9
2.3	(Un)informative censoring	12
2.4	Non-proportionality	14
3	Assignment	21

Abstract

This week we continue our investigation of the Cox Proportional Hazard Model by simulating data with violations to the basic assumptions of the model and hopefully gaining some insight into the sensitivity of the estimates to those sorts of violations.

1 Introduction

Last week we succeeded in creating simulated random death-dates corresponding to the *continuous* time model that underlies the lx column of a life table. We then applied an arbitrary proportional shift in the underlying hazard and then—as if all that were not enough—we recovered that arbitrary shift parameter by running a Cox regression on our simulated data.

This week we will continue simulating data for proportional hazard estimations but we'll get considerably more elaborate. First we will create a more general `lxCinverse()` function that will generate simulated data based on an lx column plus as many covariates as we care to make up. Then we will systematically damage the data by imposing random measurement error, different

sorts of censoring, and violations of proportionality. Hopefully the result will be an intuitive sense for how robust Cox Regression estimates might be when data are –as is all too often the case– disappointing.

All of the code in this document is in `~carlm/213/PropHazII/demonstration.r` and on the class web site. The assignment this week is to run the code and experiment to see what breaks a Cox regression.

2 Toward a general continuous-time-lx data simulating function

The three lesson from Part II of last week's exercise are:

1. That it is essential to understand that the hazard rate *within* each age interval of the life table is constant and thus ages at death *within* an age interval are *exponentially* distributed.
2. That it is scientifically compelling to realize that we can think of the life table as having come from a piece-wise exponential distribution of life times.
3. But that from a practical stand point, the quick and dirty procedure that used `approx()` to simply linearly interpolate the inverse survival function without even calculating the hazard rates – worked about as well. In fact for purposes of recovering the arbitrary parameter with a Cox regression, it worked even better than you might think – more on this later.

Consequently, we're going to write our more general continuous-time-lx simulating function with proportional shifts using the quick and dirty approach.

First let's read and clean up the HMD life table just as we did last week.

```
> ## This is code from last week. It just reads a life table from an HMD
> ## file and adds a couple of useful columns.
> ### Get an lx curve from the Human Mortality Database
> Ltab<-read.table(file='/hdir/0/carlm/213/PropHazI/US1X5.lifeTable',
+                 skip=2,header=T,
+                 as.is=T)
> ## load the cleanHMD function
> source(file="/hdir/0/carlm/213/PropHazI/snorkHMD.r")
> ## run the cleanHMD to add the age bound columns
> ltab<-cleanHMD(Ltab)
> ## isolate a single year life table
> us05<-ltab[ltab$Year == "2005",]
```

Now we can get on with the business of writing a function for generating simulated ages at death in continuous time based on a discrete time lx column of a life table—with *linear proportional shift* of the hazard rates. It will be useful to first consider Equation 1

$$h_i(t) = h_0(t)e^{x_i^1\beta^1 + \dots + x_k^i\beta_k} \quad (1)$$

Equation 1 shows the Cox Hazard model. It is the equation whose partial likelihood is maximized when we run the `coxph()` function. $h_i(t)$ is the hazard at time t of individual i . The point of the Cox Model is that this *individual* hazard function is composed of a *baseline hazard*, $h_0(t)$ which is common to all individuals *and* an individual bit which is unique to each individual, or at least to each unique set of covariate values. The individual bit is $x_i^1\beta^1 + \dots + x_k^i\beta_k$ where $\beta^{1..k}$ are a set of parameters to be estimated (or in our case arbitrarily selected and then recovered by `coxph()`). And $x_i^{1..k}$ is a vector of covariate values corresponding to individual i . The inner product of $X_i^{1..k}$ and $\beta^{1..k}$ is a scalar value for each individual. When exponentiated, It it is the amount by which an individual's hazard rates differ from the baseline hazard at *every* time t .

In the code below, `Xbeta` will be a vector with one value for each simulated lifetime that we care to create. We may also refer to it as the “linear part” of the hazard function.

Now let's write a function which will take a life table and a vector, `Xbeta` and generate the kind of simulated lifetimes that we have come to appreciate: random, continuous based on a piece-wise exponential and an `lx`—but using the quick and dirty `approx()` function instead of the clean and slow technique that we mastered last week.

```
> ## We'll give our function the catchy name lxicqdx for lx inverse
> ## continuous quick and dirty with Xbeta.
> lxicqdx<-function(Xbeta,LifeTable=us05){
+   ## This will generate quick and dirty lifetimes based on the
+   ## provided lx values with hazards proportionally shifted for each
+   ## value if Xbeta. Xbeta is the linear part of the proportional
+   ## hazard model. The tricky bit is that each individual in the
+   ## sample with a unique value of Xbeta (covariatesXparameters) has a
+   ## unique hazard function -- the baseline hazard shifted
+   ## proportionally per their very own personal Xbeta.
+
+   if(missing(Xbeta)){stop("Xbeta is the whole point of this function")}
+
+   ## If Xbeta is produced by %*% w then it will be a 1XN matrix rather
+   ## than a vector. We could work with it either way but we need to
+   ## pick one.
+   Xbeta<-as.vector(Xbeta)
+   ### just in case we draw a u very close to 1..
+   ## We'll add a zero onto the end to make sure everyone behaves
+
+   Survival<-c(LifeTable$lx,0)/LifeTable$lx[1]
+   ## We'll add a corresponding maximum age beyond which no one misbehaves
+   ages<-c(LifeTable$age.lower,115)
+ }
```

```

+
+
+   ## this is where
+   ## the proportional shift happens -- NOTE that we are effectively
+   ## raising lx to
+   ## the power of the exp(Xbeta). To work out why, recall that the
+   ## hazard is the negative slope of the log of lx.
+
+   survivals<-outer(Survival,exp(Xbeta),"^")
+
+   u<-runif(n=length(Xbeta))
+   res<-NULL
+   ## we need to run approx() separately for each individual, luckily
+   ## it's a pretty quick (and dirty) function so we can just loop it.
+   ## notice that we are interpolating the inverse of the survival function
+   for(i in 1:length(u)){
+     res<-c(res,approx(x=survivals[,i],y=ages,xout=u[i])$y)
+   }
+   return(res)
+ }
> ## sanity check: if Xbeta=0 then it's all based on us05$lx
> ltimes.baseline<-lxicqdx(Xbeta=rep(0,10000))
> mean(ltimes.baseline)

[1] 77.98753

> us05$ex[1]

[1] 77.89

> mean(ltimes.baseline[ltimes.baseline> 50]) - 50

[1] 30.97085

> us05$ex[us05$age.lower==50]

[1] 30.98

> mean(ltimes.baseline[ltimes.baseline> 90]) - 90

[1] 5.277552

> us05$ex[us05$age.lower==90]

[1] 4.97

>

```

Now let's test the function with `coxph()`

```

> library(survival)
> ## N observations
> N<-2500
> ## X is a matrix of covariates we'll just use standard normals for
> ## charm and angst, virtue will be a dummy/categorical variable.
> X<- data.frame(charm=rnorm(N),
+               virtue=runif(N)<.5,
+               angst=rnorm(N))
> ## beta is a vector of parameters that determine the effect of
> ## the covariates on the hazard function
> Xbeta<-as.matrix(X)%*(beta<-c(.1, -.2, .3))
> names(beta)<-c("charm","virtue","angst")
> ## Just like last week, we'll create a data frame with a special
> ## dependent variable for coxph()
> Dset<-data.frame(X,
+                 depvar<-Surv(time=rep(0,N),
+                               time2=1xicqdx(LifeTable=us05,Xbeta=Xbeta),
+                               event=rep(1,N))
+                 )
> mod1<-coxph(depvar~charm+virtue +angst,data=Dset)
> summary(mod1)

```

Call:

```
coxph(formula = depvar ~ charm + virtue + angst, data = Dset)
```

n= 2500, number of events= 2500

	coef	exp(coef)	se(coef)	z	Pr(> z)	
charm	0.08424	1.08789	0.02018	4.174	3.00e-05	***
virtueTRUE	-0.17480	0.83963	0.04022	-4.347	1.38e-05	***
angst	0.29896	1.34845	0.02070	14.441	< 2e-16	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

	exp(coef)	exp(-coef)	lower .95	upper .95
charm	1.0879	0.9192	1.046	1.1318
virtueTRUE	0.8396	1.1910	0.776	0.9085
angst	1.3485	0.7416	1.295	1.4043

Concordance= 0.589 (se = 0.007)

Rsquare= 0.09 (max possible= 1)

Likelihood ratio test= 234.8 on 3 df, p=0

Wald test = 234.6 on 3 df, p=0

Score (logrank) test = 234.7 on 3 df, p=0

```
> ## did we recover beta or what?
```

```
> beta
```

```
charm virtue angst
0.1 -0.2 0.3
```

Before going on, let's take a moment to justify our decision to forgo the scientific purity/tedium of explicitly calculating the hazard of each age interval in favor of the quick and dirty `approx()` based method.

Consider the following:

```
> ## Using the same Xbeta from last time
> ## the covariates on the hazard function
> Xbeta<-as.matrix(X)%*(beta)
> ## we'll set up two coxph estimations -- the only difference is that
> ## in one, we will make some very drastic changes to the dependent variable.
>
>
> Time2.original<-lxicqdx(LifeTable=us05,Xbeta=Xbeta)
> ##Time2.modified<- log(rank(Time2.original)^.75) +rank(Time2.original)/25
> Time2.modified<- log((rank(Time2.original)+1)^.75) +trunc(Time2.original/25)
> plot(Time2.modified~Time2.original,main="Modified Simulated Lifetime")
> Dset<-data.frame(X,
+                 depvar.original<-Surv(time=rep(0,N),
+                                     time2=Time2.original,
+                                     event=rep(1,N)),
+                 depvar.modified<-Surv(time=rep(0,N),
+                                       time2=Time2.modified,
+                                       event=rep(1,N))
+                 )
> ## estimate the cox model with original and modified
> ## dependent variable
> summary(coxph(depvar.original~charm+virtue +angst,data=Dset))
```

Call:

```
coxph(formula = depvar.original ~ charm + virtue + angst, data = Dset)
```

n= 2500, number of events= 2500

	coef	exp(coef)	se(coef)	z	Pr(> z)
charm	0.11520	1.12210	0.02020	5.704	1.17e-08 ***
virtueTRUE	-0.24862	0.77988	0.04031	-6.168	6.92e-10 ***
angst	0.30081	1.35096	0.02114	14.229	< 2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

	exp(coef)	exp(-coef)	lower .95	upper .95
charm	1.1221	0.8912	1.0785	1.167
virtueTRUE	0.7799	1.2823	0.7206	0.844

```
angst          1.3510    0.7402    1.2961    1.408
```

```
Concordance= 0.591 (se = 0.007 )  
Rsquare= 0.098 (max possible= 1 )  
Likelihood ratio test= 258.3 on 3 df, p=0  
Wald test = 256.4 on 3 df, p=0  
Score (logrank) test = 256.8 on 3 df, p=0
```

```
> summary(coxph(depvar.modified~charm+virtue +angst,data=Dset))
```

```
Call:
```

```
coxph(formula = depvar.modified ~ charm + virtue + angst, data = Dset)
```

```
n= 2500, number of events= 2500
```

	coef	exp(coef)	se(coef)	z	Pr(> z)	
charm	0.11520	1.12210	0.02020	5.704	1.17e-08	***
virtueTRUE	-0.24862	0.77988	0.04031	-6.168	6.92e-10	***
angst	0.30081	1.35096	0.02114	14.229	< 2e-16	***

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

	exp(coef)	exp(-coef)	lower .95	upper .95
charm	1.1221	0.8912	1.0785	1.167
virtueTRUE	0.7799	1.2823	0.7206	0.844
angst	1.3510	0.7402	1.2961	1.408

```
Concordance= 0.591 (se = 0.007 )  
Rsquare= 0.098 (max possible= 1 )  
Likelihood ratio test= 258.3 on 3 df, p=0  
Wald test = 256.4 on 3 df, p=0  
Score (logrank) test = 256.8 on 3 df, p=0
```

```
>
```

```
> #####
```

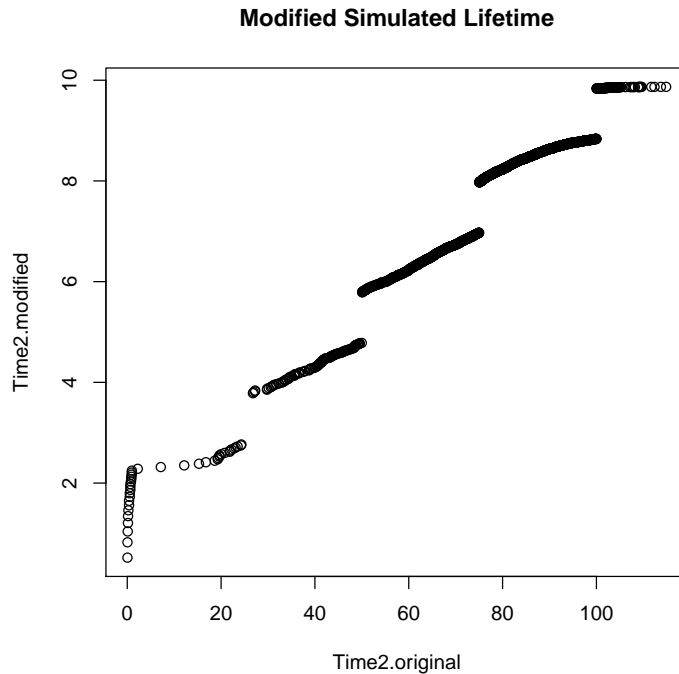
```
> ## QUESTION: WTF?
```

```
> ## Or, more scientifically, how does this result justify our use of the
```

```
> ## quick and dirty approach?
```

```
> #####
```

```
>
```



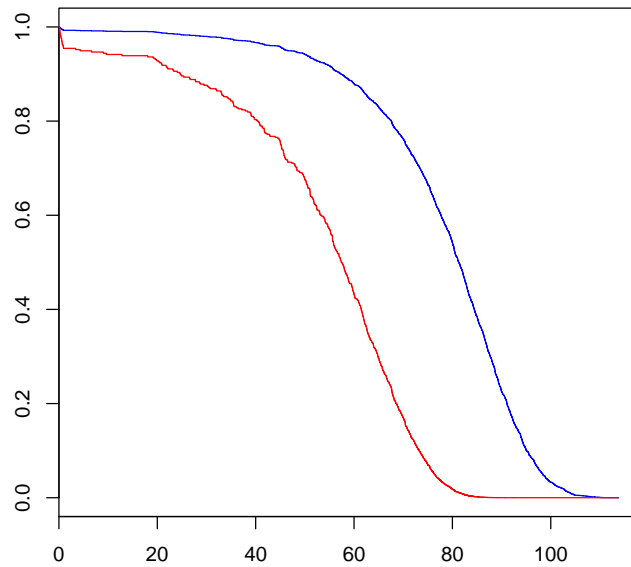
2.1 Plotting coxph results

Let's plot the survival function for a few values of Xbeta so we can see what proportion hazard shifts do to survival functions.

```
> ## the survfit function is used to calculate hazard, survival and
> ## other functions from hazard models. In it's simplest form it takes
> ## a coxph object (the output of a coxph()) estimation, and a matrix
> ## of covariates. Each row of the matrix represents and individual's
> ## covariates.
>
>
> ndat<-data.frame(rbind(c(0,0,0),c(9,9,9)))
> names(ndat)<-names(X)
> ## individual=FALSE means that each row of newdata represents a distinct
> ## individual entitled to her own survival curve. The TRUE option is used
> ## when there are time-varying covariates.
>
> mod1.fit<-survfit(mod1,newdata=ndat,se.fit=FALSE,individual=FALSE)
> ## the output of survfit can be fed to the plot() function
> plot(mod1.fit,col=c("blue","red"),
+      main="Survival functions from baseline and a Herman Cain supporter")
```

>
>

Survival functions from baseline and a Herman Cain supporter



2.2 Random Error

Strikingly absent from the Equation 1 is the random error term that we are accustomed to seeing in regression models. Randomness is inherent in the model as the very thing being modeled is a probability distribution. Still, one could imagine that there might be measurement error in the covariates as well. Let's add that and see what happens.

```
> Xbeta.original<-as.matrix(X)%*(beta)
> error.size<-5
> Xbeta.deviated<- Xbeta.original+
+   rnorm(n=N,
+         mean=0,
+         sd=error.size)
> summary(Xbeta.original)
```

```
      V1
Min.   :-1.2518
1st Qu.:-0.3204
Median :-0.1079
```

```

Mean      :-0.1062
3rd Qu.:  0.1184
Max.      :  1.0008

> summary(Xbeta.deviated)

      V1
Min.    :-18.9797
1st Qu.: -3.6812
Median  : -0.1880
Mean    : -0.1947
3rd Qu.:  3.3491
Max.    : 16.5562

> plot(Xbeta.deviated~Xbeta.original,main="Xbeta vs Xbeta+random error",
+       col=rainbow(100),pch=1)
> Time2.original<-lxicqdx(LifeTable=us05,Xbeta=Xbeta.original)
> Time2.deviated<-lxicqdx(LifeTable=us05,Xbeta=Xbeta.deviated)
> Dset<-data.frame(X,
+                  depvar.original<-Surv(time=rep(0,N),
+                                       time2=Time2.original,
+                                       event=rep(1,N)),
+                  depvar.deviated<-Surv(time=rep(0,N),
+                                       time2=Time2.deviated,
+                                       event=rep(1,N))
+                  )
> summary(coxph(depvar.original~charm+virtue +angst,data=Dset))

Call:
coxph(formula = depvar.original ~ charm + virtue + angst, data = Dset)

n= 2500, number of events= 2500

      coef exp(coef) se(coef)      z Pr(>|z|)
charm      0.12566  1.13390  0.02016  6.234 4.53e-10 ***
virtueTRUE -0.14243  0.86725  0.04009 -3.553 0.000381 ***
angst      0.29913  1.34869  0.02088 14.326 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

      exp(coef) exp(-coef) lower .95 upper .95
charm      1.1339      0.8819      1.0900      1.1796
virtueTRUE  0.8672      1.1531      0.8017      0.9381
angst      1.3487      0.7415      1.2946      1.4050

Concordance= 0.591 (se = 0.007 )

```

```

Rsquare= 0.093 (max possible= 1 )
Likelihood ratio test= 243.2 on 3 df, p=0
Wald test = 243.2 on 3 df, p=0
Score (logrank) test = 243.2 on 3 df, p=0

> summary(coxph(depvar.deviated~charm+virtue +angst,data=Dset))

Call:
coxph(formula = depvar.deviated ~ charm + virtue + angst, data = Dset)

n= 2500, number of events= 2500

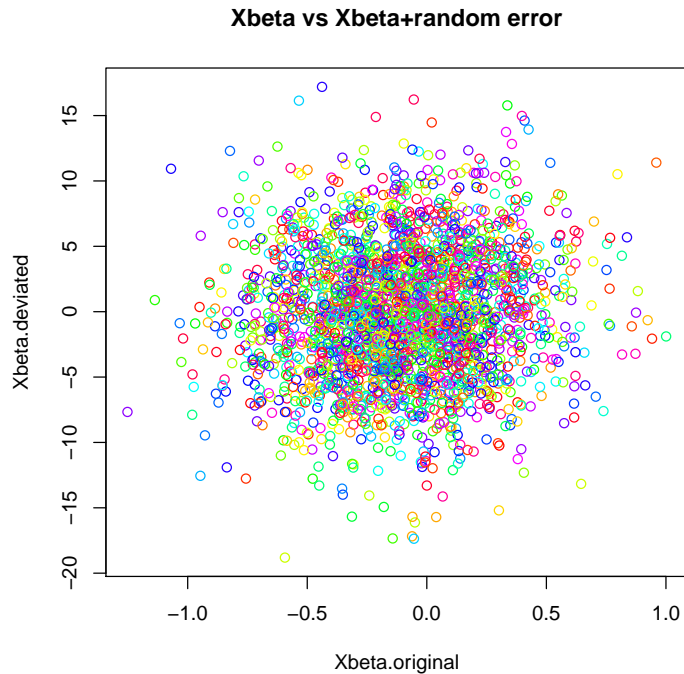
              coef exp(coef) se(coef)      z Pr(>|z|)
charm          0.01092  1.01098  0.01978  0.552  0.58092
virtueTRUE    -0.09512  0.90927  0.04009 -2.373  0.01766 *
angst          0.05546  1.05702  0.02056  2.697  0.00699 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

              exp(coef) exp(-coef) lower .95 upper .95
charm          1.0110    0.9891    0.9725    1.0509
virtueTRUE     0.9093    1.0998    0.8406    0.9836
angst          1.0570    0.9461    1.0153    1.1005

Concordance= 0.532 (se = 0.007 )
Rsquare= 0.005 (max possible= 1 )
Likelihood ratio test= 13.55 on 3 df, p=0.003591
Wald test = 13.55 on 3 df, p=0.003581
Score (logrank) test = 13.56 on 3 df, p=0.003573

>
> #####
> ## QUESTION: Experiment enough to form an opinion as to how much error
> ## in Xbeta is too much. Note the Pr(>|z|) column and the se(coef)
> ## column as well as whether or not the coefficients are recovered.
> ## For the curious: try adding random error to just one variable.
> #####

```



2.3 (Un)informative censoring

One of the problems inherent in any analysis in which the dependent variable is time is that not all objects of study will complete the task that you have set out for them. Particularly if the task in question is death, you and your subjects are likely to disagree about the desirability of it. In this case your study subjects may actively work to thwart science. This is understandable but nonetheless unfortunate.

The Cox Model mitigates this problem by allowing for *uninformative* Censoring. By *uninformative*, we mean that when or whether an observation is censored does **not** depend on the values of the covariates X .

Let's add uninformative censoring to our sandbox and see what (doesn't) happen.

```
> ## start with the usual Xbeta
> Xbeta<-as.matrix(X) %*% (beta)
> ## the usual simulated lifetimes
>
> Time2.original<-lxicqdx(LifeTable=us05,Xbeta=Xbeta)
> fraction.censored<-0.25
> Dset<-data.frame(X,
+                 depvar.original<-Surv(time=rep(0,N),
```

```

+                                     time2=Time2.original,
+                                     event=rep(1,N)),
+     depvar.censored<-Surv(time=rep(0,N),
+                             time2=Time2.original,
+                             ## here is where the censoring happens
+                             ##event=ifelse(runif(N)<fraction.censored,
+                             event=ifelse(X$charm> -1,
+                                     0,1))
+     )
> ## estimate the cox model with original and modified
> ## dependent variable
> summary(coxph(depvar.original~charm+virtue +angst,data=Dset))

```

Call:

```
coxph(formula = depvar.original ~ charm + virtue + angst, data = Dset)
```

n= 2500, number of events= 2500

	coef	exp(coef)	se(coef)	z	Pr(> z)
charm	0.08812	1.09212	0.01972	4.468	7.88e-06 ***
virtueTRUE	-0.22486	0.79863	0.04032	-5.578	2.44e-08 ***
angst	0.30548	1.35727	0.02102	14.533	< 2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

	exp(coef)	exp(-coef)	lower .95	upper .95
charm	1.0921	0.9156	1.051	1.1352
virtueTRUE	0.7986	1.2522	0.738	0.8643
angst	1.3573	0.7368	1.302	1.4144

Concordance= 0.591 (se = 0.007)

Rsquare= 0.096 (max possible= 1)

Likelihood ratio test= 251.8 on 3 df, p=0

Wald test = 250.3 on 3 df, p=0

Score (logrank) test = 250.5 on 3 df, p=0

```
> summary(coxph(depvar.censored~charm+virtue +angst,data=Dset))
```

Call:

```
coxph(formula = depvar.censored ~ charm + virtue + angst, data = Dset)
```

n= 2500, number of events= 388

	coef	exp(coef)	se(coef)	z	Pr(> z)
charm	-1.54900	0.21246	0.05664	-27.348	< 2e-16 ***
virtueTRUE	-0.21498	0.80656	0.10366	-2.074	0.03809 *
angst	0.21429	1.23898	0.05570	3.847	0.00012 ***

 Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

	exp(coef)	exp(-coef)	lower .95	upper .95
charm	0.2125	4.7068	0.1901	0.2374
virtueTRUE	0.8066	1.2398	0.6583	0.9883
angst	1.2390	0.8071	1.1108	1.3819

Concordance= 0.913 (se = 0.017)
 Rsquare= 0.289 (max possible= 0.875)
 Likelihood ratio test= 853.7 on 3 df, p=0
 Wald test = 776.2 on 3 df, p=0
 Score (logrank) test = 805.9 on 3 df, p=0

```
> #####
> ## QUESTION: experiment with the fraction.censored variable to see how
> ## much censoring it takes to break the coxph() Notice the se(coef)
> ## and Pr(<|z|) columns as well as the coef estimates. Notice if
> ## virtue (the categorical variables) behaves differently from the
> ## continuous variables.
> #####
>
> #####
> ## QUESTION: Modify the code to make censoring dependent on the value
> ## of Xbeta and then on the level of one parameter notice what happens
> ## to the parameters BUT also to the estimated standard errors. An
> ## easy way to do this is to modify the "event" argument in the Surv()
> ## function. 0= censored, 1=event observed. Recall that charm
> ## and angst are all standard normals.
>
> ##Frustration Management:
> ## fiddling with the event argument of Surv is the way to modify censoring
> ## here is an example:
> ## depvar.censored<-Surv(time=rep(0,N),
> ##                          time2=Time2.original, ## about
> ##                          32% will have charm > -1
> ##                          event=ifelse(X$charm> -1, 0,1))
> #####
```

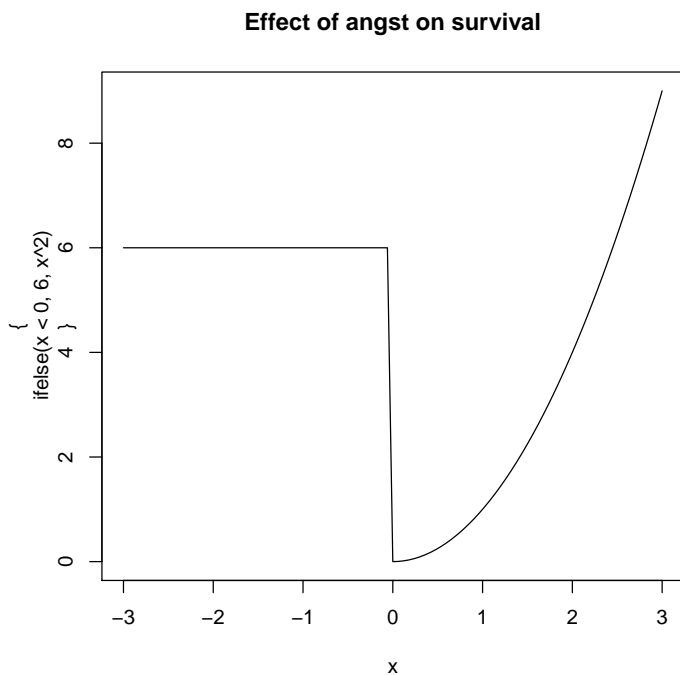
2.4 Non-proportionality

Proportionality is a key assumption of the Cox Model. Since it is an assumption of the model, it can mess with you. In the example below, you might notice that even though one of the variables, *angst*, has a pretty dramatic non-proportionality to it, the Cox regression is going to return a statistically significant (and incorrect) result.

```

> ##
> ##N<-1000
> ## X is a matrix of covariates we'll just use standard normals
> X<- data.frame(charm=rnorm(N),
+               virtue=rnorm(N),
+               angst=rnorm(N))
> Xbeta<-as.matrix(X)%*%(beta)
> ## Start with Xbeta based on the charm and virtue -- we'll make angst
> ## be nonproportional
>
> Xbeta.nonp<-as.matrix(X[,1:2])%*%(beta[1:2])
> curve({ifelse(x < 0,6,x^2) }, from= -3, to=3, main="Effect of angst on survival")
> Xbeta.nonp<- Xbeta.nonp + ifelse(X$angst < 0,6,X$angst^2)
> Time2.original<-lxicqdx(LifeTable=us05,Xbeta=Xbeta)
> Time2.nonp<-lxicqdx(LifeTable=us05,Xbeta=Xbeta.nonp)

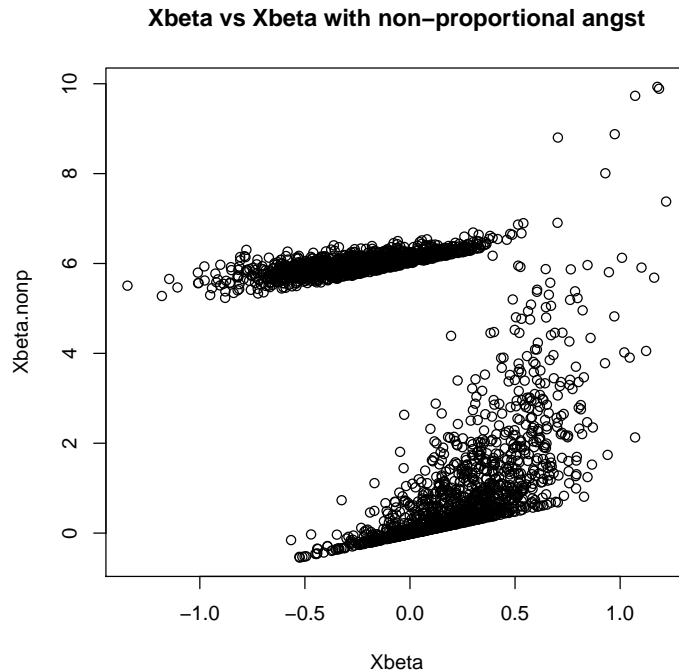
```



```

> plot(Xbeta.nonp~Xbeta,main="Xbeta vs Xbeta with non-proportional angst")

```



```

> Dset<-data.frame(X,
+                 depvar.original<-Surv(time=rep(0,N),
+                                     time2=Time2.original,
+                                     event=rep(1,N)),
+                 depvar.nonp<-Surv(time=rep(0,N),
+                                   time2=Time2.nonp,
+                                   event=rep(1,N))
+                 )
> ## estimate the cox model with original and modified
> ## dependent variable
> summary(mod1<-coxph(depvar.original~charm+virtue +angst,data=Dset))

```

Call:
 coxph(formula = depvar.original ~ charm + virtue + angst, data = Dset)

n= 2500, number of events= 2500

	coef	exp(coef)	se(coef)	z	Pr(> z)	
charm	0.08289	1.08643	0.01983	4.179	2.92e-05	***
virtue	-0.18138	0.83412	0.02056	-8.821	< 2e-16	***
angst	0.26949	1.30929	0.02142	12.580	< 2e-16	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

	exp(coef)	exp(-coef)	lower .95	upper .95
charm	1.0864	0.9205	1.0450	1.1295
virtue	0.8341	1.1989	0.8012	0.8684
angst	1.3093	0.7638	1.2555	1.3654

Concordance= 0.587 (se = 0.007)
Rsquare= 0.09 (max possible= 1)
Likelihood ratio test= 237.1 on 3 df, p=0
Wald test = 236.4 on 3 df, p=0
Score (logrank) test = 237 on 3 df, p=0

```
> ### Hmm what do you think of that P value on angst?  
> summary(mod1.np<-coxph(depvar.nonp~charm+virtue +angst,data=Dset))
```

Call:
coxph(formula = depvar.nonp ~ charm + virtue + angst, data = Dset)

n= 2500, number of events= 2500

	coef	exp(coef)	se(coef)	z	Pr(> z)
charm	0.05426	1.05576	0.01967	2.759	0.005797 **
virtue	-0.07293	0.92967	0.02013	-3.623	0.000292 ***
angst	-0.72879	0.48249	0.02690	-27.091	< 2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

	exp(coef)	exp(-coef)	lower .95	upper .95
charm	1.0558	0.9472	1.0158	1.0972
virtue	0.9297	1.0757	0.8937	0.9671
angst	0.4825	2.0726	0.4577	0.5086

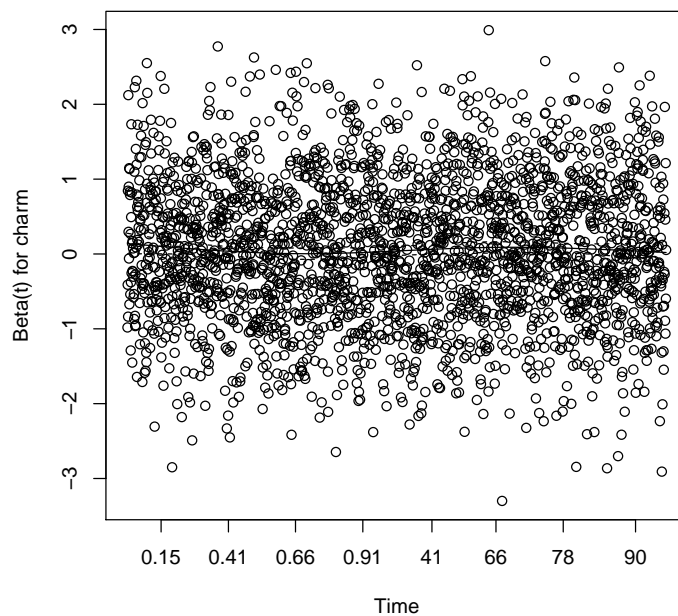
Concordance= 0.685 (se = 0.007)
Rsquare= 0.247 (max possible= 1)
Likelihood ratio test= 708.6 on 3 df, p=0
Wald test = 755.7 on 3 df, p=0
Score (logrank) test = 750.5 on 3 df, p=0

The `cox.zph()` function calculates the *schoenfeld residuals* and checks for correlation with time. *Schoenfeld* residuals are difference between observed event times and expected event times based on each estimated coefficient. There are as many Schoenfeld residuals per observation as there are covariates.

Roughly speaking, the Cox estimation procedure should remove the effect of time leaving the Schoenfeld residuals *uncorrelated* with time. Any sort non-random-looking relationship between the Schoenfeld residual and time indicates non-proportionality.

Graphically, the Schoenfeld residuals can be plotted against time. Numerically, we can look at the correlation coefficient between the residual and time. If it is statistically significantly different from zero, then we have a non-proportionality problem.

```
> mod1.np.z<-cox.zph(mod1.np)
> ## the Schoenfeld residuals are calculated for each covariate -- they
> ## are the difference between the failure time and expected failure
> ## time calculated with the estimated coefficients
>
> ## here is a plot of the Schoenfeld residuals against time for "charm"
> ## -- charm is proportional so there should be no pattern
> plot(mod1.np.z[1,])
```



```
> ## Now draw the plot for angst
> plot(mod1.np.z[3,])
> ## print the output of cox.zph() for a numerical test of proportionality
> ## If the proportionality assumption holds, then the correlation
> ## between the Schoenfeld residuals and time should be zero. The chisq
> ## column is a whether or not the correlation is significantly
> ## different from zero. If it IS then we have a non-proportionality
> ## problem.
```

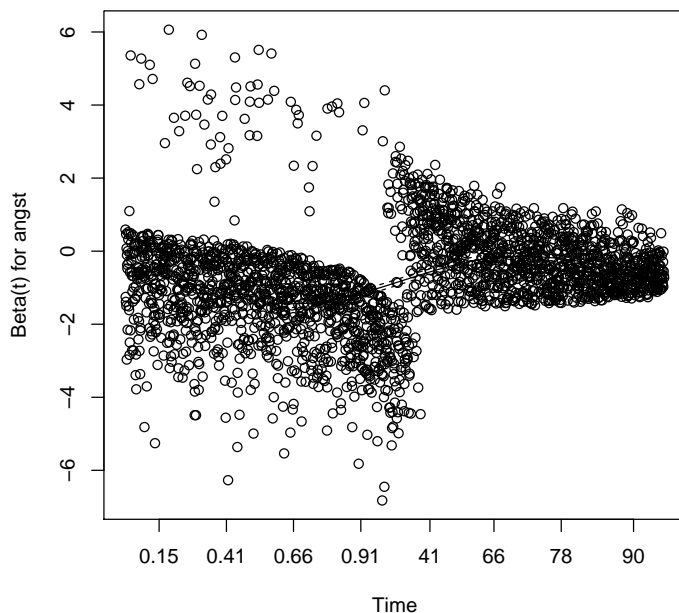
```

>
> print(mod1.np.z)

      rho   chisq      p
charm  0.00113  0.00303 9.56e-01
virtue -0.03412  2.92235 8.74e-02
angst   0.16230 75.36332 0.00e+00
GLOBAL      NA 78.22529 1.11e-16

>
> #####
> ## GOTCHA: The Schoenfeld residuals are not defined for censored
> ## observations. Consequently, a dataset that suffers from both a lot
> ## of censoring and non-proportionality... could mean trouble.
> #####

```



Dealing with non-proportional effects is pretty easy when the data are simulated by your own bad self. It's much harder when the data are captured in the wild. In some cases it is possible to mitigate the problem by adding squared covariates or interacting covariates. In the present case, we can do ourselves a lot of good by "stratifying" the sample. That is by telling `coxph()` to estimate a single set of parameter but with two (or more) baseline hazards. We'll estimate one baseline hazard for observations where `angst` is less than zero and another where `angst` is greater than zero.

```

> ## cooking up the same angst filled example data
> Xbeta<-as.matrix(X[,1:3])%*(beta<-c(.1, -.2, .3))
> ## Start with Xbeta based on the charm and virtue -- we'll make angst
> ## be non-proportional
>
> Xbeta.nonp<-as.matrix(X[,1:2])%*(beta[1:2])
> ##curve({ifelse(x < 0,6,x^2) }, from= -3, to=3, main="Effect of angst on survival")
>
> Xbeta.nonp<- Xbeta.nonp + ifelse(X$angst < 0,6,X$angst^2)
> ## create a new variable on which to stratify 0 is where angst changes
> ## it's tune
> X$broody<-X$angst < 0
> Dset<-data.frame(X,
+                 depvar.nonp<-Surv(time=rep(0,N),
+                                   time2=lxicqdx(Xbeta.nonp),
+                                   event=rep(1,N))
+                 )
> summary(mod2.np<-coxph(depvar.nonp~charm+virtue +angst +strata(broody),
+                        data=Dset))

```

Call:

```
coxph(formula = depvar.nonp ~ charm + virtue + angst + strata(broody),
      data = Dset)
```

n= 2500, number of events= 2500

	coef	exp(coef)	se(coef)	z	Pr(> z)	
charm	0.09138	1.09569	0.02098	4.355	1.33e-05	***
virtue	-0.13846	0.87069	0.02003	-6.914	4.73e-12	***
angst	0.67103	1.95625	0.04486	14.957	< 2e-16	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

	exp(coef)	exp(-coef)	lower .95	upper .95
charm	1.0957	0.9127	1.0515	1.1417
virtue	0.8707	1.1485	0.8372	0.9056
angst	1.9562	0.5112	1.7916	2.1360

Concordance= 0.614 (se = 0.009)

Rsquare= 0.111 (max possible= 1)

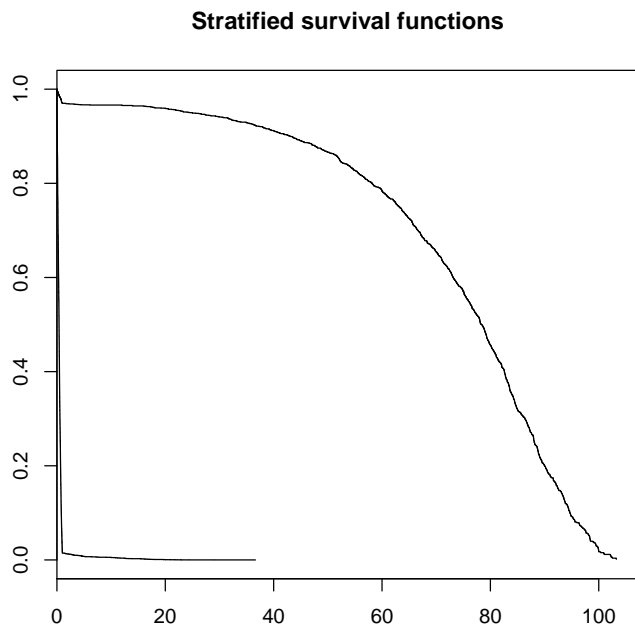
Likelihood ratio test= 293.7 on 3 df, p=0

Wald test = 276.9 on 3 df, p=0

Score (logrank) test = 276.5 on 3 df, p=0

A plot of the survival functions reveals the effect of stratifying on broody.

```
> ## Plot the survival function, revealing the effect of stratifying.
> plot(survfit(mod2.np),main="Stratified survival functions")
> ## In case you want to rerun earlier code, we'll remove broody from X
> X$broody<-NULL
```



3 Assignment

Create a new `WeekN` directory in `~/213`; `cd` to it; and launch Emacs so as to create a buffer called `exN.r` as usual. Then import the `~/carlm/213/PropHazII/demonstration.r` file and work through it. There are a few questions, worth considering, but go where your curiosity leads you. There are lots of ways to damage data.