

Demography 213  
Simulating data with proportional hazard  
assumption  
starting with a life table

Carl Mason  
carlm@demog.berkeley.edu

October 31, 2011

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>A life table from the Human Mortality Database</b>	<b>2</b>
<b>3</b>	<b>Generating random numbers to fit a distribution</b>	<b>3</b>
3.1	Profound truth number 1 . . . . .	3
3.2	Profound inconvenience number 1 . . . . .	5
3.3	Profound truth number 2 (more true than profound) . . . . .	7
3.4	Almost there...but not quite . . . . .	10
<b>4</b>	<b>Assignment</b>	<b>15</b>
4.1	Part I . . . . .	15
4.2	Part II . . . . .	16

### Abstract

This week we will begin exploring proportion hazard models by generating a data set based on the underlying assumptions of the model. Because we are demographers, our starting point will be the life table.

## 1 Introduction

In this exercise we are going to use R to generate a simulated set of death dates based on the  $l_x$  values from a real life table. We will notice, after doing this, that life tables come from a universe in which time is discrete. Events occur in this universe only in 5 year age intervals. We will find this to be inadequate for our high minded scientific pursuits. But rather than wallow in self pity,

we will bravely make an assumption. We will assume that events times within an interval (generally ages at death) are exponentially distributed. Or in other words have a constant *hazard rate*—within the age intervals of the life table. This will allow us to generate simulated data that suitable for a Cox Regression analysis.

## 2 A life table from the Human Mortality Database

In order to accomplish the lofty goal that we have set for ourselves, we shall need a life table. Here is one from the Human Mortality Database. The HMD is the place to go for human mortality data – and thanks to John Wilmoth, it is right here: [www.mortality.org](http://www.mortality.org)

To save time, I have downloaded the 1X5 US life tables. The “1” in this case refers to calendar years and the “5” to the width of the age intervals. The file includes life tables for many calendar years; we will only use one of them today.

To save even more time, I have written a some R code that extracts the lower age bound from the unfortunately formatted “age” column. You can simply `source()` it.

```
> ### 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)
> ## not an ideal format for our purposes -- that age ranges are
> ## particularly pernicious
>
> head(Ltab,n=24)
```

	Year	Age	mx	qx	ax	lx	dx	Lx	Tx	ex
1	1933	0	0.06094	0.05818	0.22	100000	5818	95456	6095053	60.95
2	1933	1-4	0.00481	0.01901	1.41	94182	1790	372098	5999597	63.70
3	1933	5-9	0.00163	0.00811	2.31	92392	750	459943	5627499	60.91
4	1933	10-14	0.00135	0.00673	2.62	91643	617	456743	5167556	56.39
5	1933	15-19	0.00229	0.01139	2.70	91026	1037	452743	4710813	51.75
6	1933	20-24	0.00330	0.01637	2.58	89989	1473	446383	4258070	47.32
7	1933	25-29	0.00375	0.01857	2.53	88516	1644	438527	3811687	43.06
8	1933	30-34	0.00440	0.02176	2.59	86873	1890	429803	3373159	38.83
9	1933	35-39	0.00561	0.02769	2.56	84982	2353	419175	2943356	34.63
10	1933	40-44	0.00701	0.03448	2.61	82629	2849	406344	2524182	30.55
11	1933	45-49	0.00986	0.04816	2.62	79780	3842	389742	2117838	26.55
12	1933	50-54	0.01358	0.06576	2.59	75938	4994	367633	1728096	22.76
13	1933	55-59	0.01850	0.08860	2.62	70944	6286	339777	1360463	19.18
14	1933	60-64	0.02803	0.13128	2.58	64658	8488	302791	1020687	15.79
15	1933	65-69	0.03989	0.18199	2.60	56170	10222	256287	717896	12.78
16	1933	70-74	0.06319	0.27315	2.52	45948	12551	198633	461608	10.05

```

17 1933 75-79 0.08867 0.36277 2.50 33397 12116 136643 262976 7.87
18 1933 80-84 0.13649 0.50299 2.39 21282 10704 78427 126332 5.94
19 1933 85-89 0.19737 0.63922 2.24 10577 6761 34256 47906 4.53
20 1933 90-94 0.26053 0.73566 2.04 3816 2807 10775 13650 3.58
21 1933 95-99 0.33725 0.82862 1.93 1009 836 2479 2875 2.85
22 1933 100-104 0.42760 0.89523 1.75 173 155 362 396 2.29
23 1933 105-109 0.52331 0.93814 1.58 18 17 32 34 1.90
24 1933 110+ 0.60115 1.00000 1.66 1 1 2 2 1.66

```

```

> ### There's some code in this file that adds two columns age.lower and
> ### age.upper
> source(file="/hdir/0/carlm/213/PropHazI/snorkHMD.r")
> ltab<-cleanHMD(Ltab)
> ## ltab still has gobs of data on lots of years
> sort(unique(ltab$Year))

```

```

[1] 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947
[16] 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962
[31] 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977
[46] 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992
[61] 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005

```

```

> ## isolate a single year life table
> us05<-ltab[ltab$Year=="2005",]
> names(ltab)

```

```

[1] "Year"      "Age"      "mx"      "qx"      "ax"      "lx"
[7] "dx"      "Lx"      "Tx"      "ex"      "age.lower" "age.upper"

```

```

> ## add a width column
> us05$width<-us05$age.upper - us05$age.lower +1

```

### 3 Generating random numbers to fit a distribution

So now we have a life table in usable form. How are we going to generate random lifetimes that reflect it ?

#### 3.1 Profound truth number 1

Since the Cumulative Distribution Function (CDF) maps a value (say for example an age at death) onto its *percentile* (aka the percent of other ages at death in the sample that are *smaller* than the given age at death)– it follows that the *inverse* of the CDF maps a *percentile* onto a value (that is an age at death) that is at least as large as 'percentile' of other ages at death in the sample.

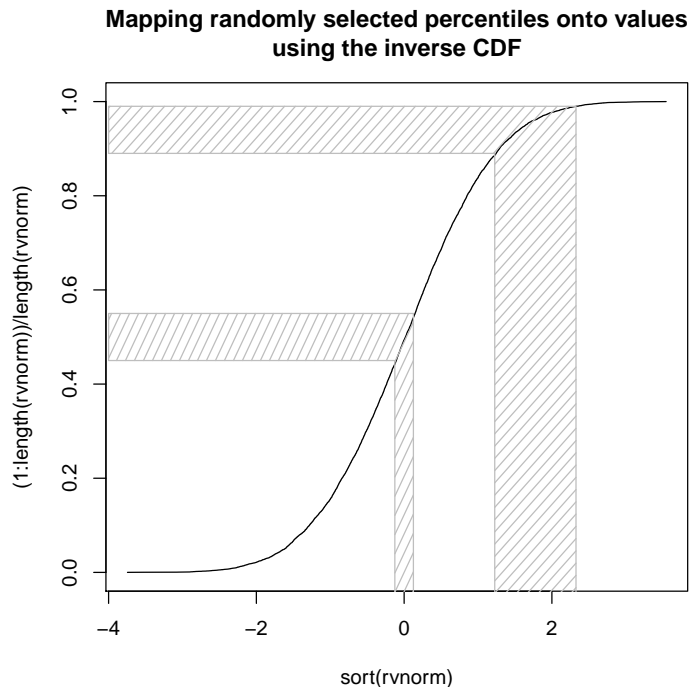
The profound bit is that if we draw random uniforms and call them percentiles – and feed those percentiles to the inverse of the CDF– we get numbers back with the distribution that we looking for.

It is worth pausing to reflect on why this is so. First note that a uniformly distributed random variable takes on values between 0 and 1 – just like percentiles. Consider this plot of the CDF of the normal.

```

> rvnorm<- rnorm(10000)
> plot(sort(rvnorm), (1:length(rvnorm))/length(rvnorm), type='l', lty=1,
+      main="Mapping randomly selected percentiles onto values\nusing the inverse CDF")
> draw.bands<-function(prob0=.05,prob1=.1){
+   ### draws gray bands that illustrate how a range of percentile
+   ### values is projected onto a range of realized values from the
+   ### underlying distribution
+
+   polygon(x=c(-4,qnorm(prob0),qnorm(prob0),qnorm(prob1),qnorm(prob1),-4),
+           y=c(prob0,prob0,-1,-1,prob1,prob1),
+           density=15,col='gray',angle=dnorm(mean(prob0,prob1))*180)
+ }
> draw.bands(.89,.99)
> draw.bands(.45,.55)

```



### 3.2 Profound inconvenience number 1

It turns out, that the inverse CDFs of many useful probability functions are diabolically difficult to find. You might wish to convince yourself of this by trying to find a closed form inverse CDF for the good old Normal distribution. Good luck.

When you need a distribution function with an easy to find inverse CDF, The Exponential Distribution is your friend. In Equation 1,  $f(t)$  is the **the probability density function**;  $F(t)$  is the CDF,  $\lambda$  is the “rate” parameter of the exponential distribution, and  $F^{-1}(u)$  is the inverse of the CDF. Note that the argument of  $F^{-1}(u)$  is  $u$  and it’s result is  $t$ . The clever trick in finding  $F^{-1}(u)$  is to substitute  $u$  for  $F(t)$  and then to substitute  $u$  for  $1 - u$ . The latter step is OK because  $u$  is going to be a random uniform.

$$f(t) = \lambda * e^{-\lambda t} \tag{1}$$

$$F(t) = 1 - e^{-\lambda t} \tag{2}$$

$$1 - F(t) = e^{-\lambda t} \tag{3}$$

$$\frac{\log(1 - F(t))}{-\lambda} = t \tag{4}$$

$$F^{-1}(u) = t = \frac{-\log(u)}{\lambda} \tag{5}$$

What this all means is that we can generate random exponentials by simply logging a random uniform and dividing by  $-\lambda$

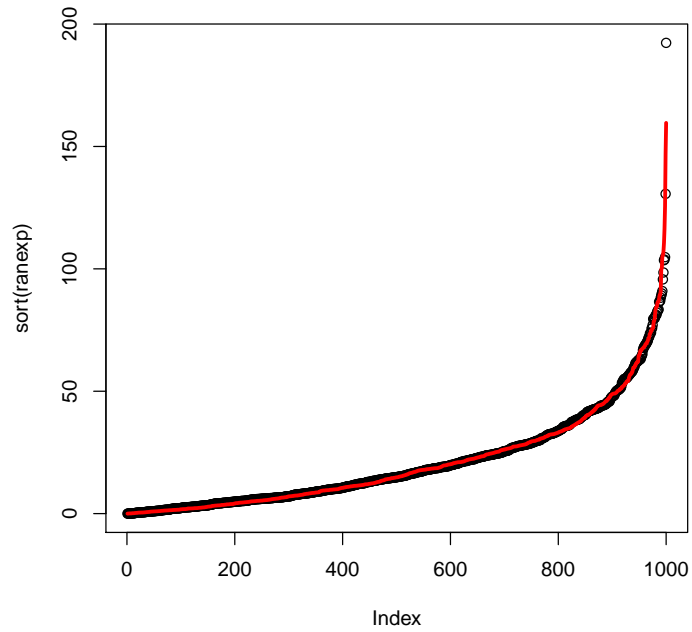
```
> ##pick a "rate" parameter
> lambda<- .05
> ## insert random uniforms into inverse CDF
> ranexp<- -log(runif(n=1000))/lambda
> ### the mean of the exponential(lambda) is 1/lambda
> mean(ranexp)

[1] 19.11148

> 1/lambda

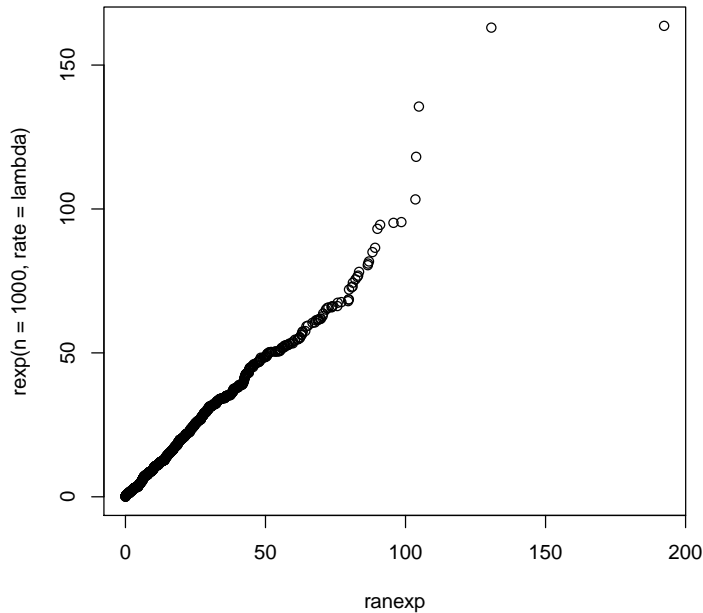
[1] 20

> ## plot result
> plot(sort(ranexp))
> ## overlay red line of empirical truth by using R's inbuilt rexp()
> ## function for generating random exponentials
> lines(sort(rexp(n=1000,rate=lambda)),col='red',lwd=3)
```



Although overlaying the plot above is pretty compelling, another way of comparing two random variables is with `qqplot()`. `Qqplot()` plots quantile against quantile

```
> qqplot(ranexp, rexp(n=1000, rate=lambda))
```



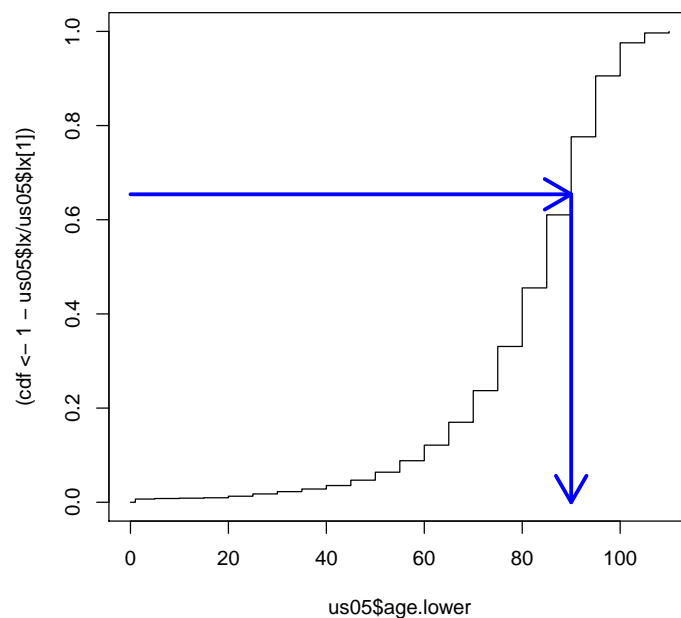
### 3.3 Profound truth number 2 (more true than profound)

Since our immediate goal is to generate random numbers from an inverse CDF, it is probably useful to notice that if you divide the  $l_x$  column of the life table by its radix, what you get the *Survival Function* which is 1-*Cumulative Distribution Function* of life times.

```
> plot(us05$age.lower, (cdf<-1-us05$lx/us05$lx[1]), type='s')
> ### Generating a single age at death value from the distribution
> ###implied by lx
>
> ##### (1)generate a random uniform to bounce off of 1-lx/lx[1]
> u<- runif(n=1)
> u<-.654 ## to make a better picture.
> ## Graphically, we want to "bounce" a horizontal ray at the level of
> ## "u" off of the CDF at 90 degrees.
>
> ## mathematically this translates to finding the highest age category
> ## X for which 1-lx[X]/lx[1] < u and then adding 1.
>
> LifeTime<- us05$age.lower[sum(cdf < u)+1]
> LifeTime
```

[1] 90

```
> ## Graphing ...
> arrows(x0=0, x1=LifeTime,,
+        y1=u,y0=u,col='blue',lwd=3)
> arrows(x0=LifeTime, x1=LifeTime,
+        y1=0,y0=u,col='blue',lwd=3)
```



If we want to generate 10,000 random life times, we can use the familiar `cut()` function which does what the above picture says to do.

```
> ### (1) generate 100,000 ages at death; and initialize a variable to
> ### (1) hold all those numbers
>
> N<-100000
> Us<-runif(n=N)
> lives<-cut(Us, breaks=c(1-(us05$lx/us05$lx[1]),1),
+           labels=us05$age.lower)
> ## calculate the lx implied by the lives we just generated
> ## so that we might compare it to us05$lx
>
> lx.hat<- c(N,N-cumsum(table(lives)))
> ##### compare the lx.hat that we just generated with the lx values that
```

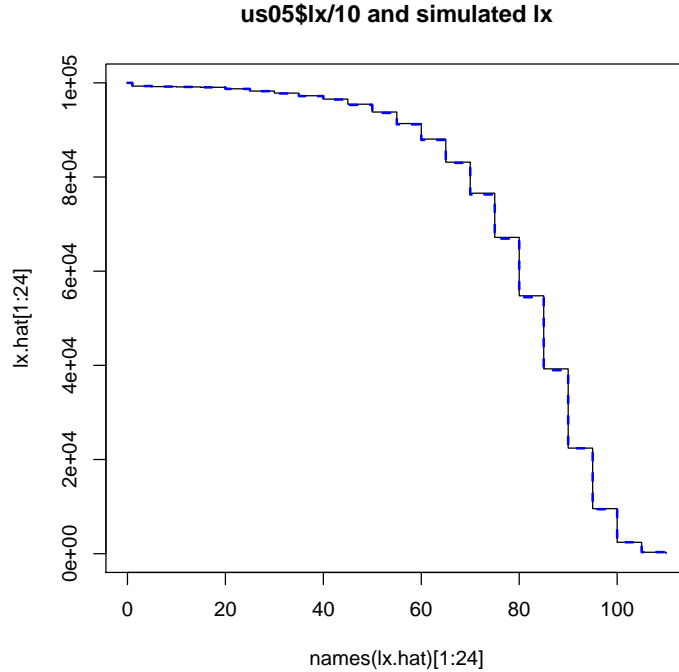
```

> ##### are supposed to define the distribution of lives
> cbind(us05$lx,lx.hat)

      lx.hat
100000 100000
0    99320 99337
1    99205 99214
5    99134 99129
10   99045 99047
15   98719 98720
20   98231 98251
25   97743 97755
30   97189 97234
35   96451 96562
40   95312 95397
45   93617 93695
50   91180 91331
55   87875 87960
60   83014 83170
65   76293 76526
70   66916 67150
75   54478 54601
80   38966 38960
85   22385 22362
90    9450  9520
95    2428  2398
100    330   335
105     21   18
110 100000    0

> ## clean up and plot
> lx.hat<-lx.hat[-25] ## the us05 life table doesn't have the last row
> names(lx.hat)<-us05$age.lower
> plot( lx.hat[1:24]~names(lx.hat)[1:24],      type='s')
> lines(us05$age.lower, us05$lx, type='s',col='blue',lty=2,lwd=2)
> title("us05$lx/10 and simulated lx")

```



### 3.4 Almost there...but not quite

So now we know how to generate some random lifetimes that reflect a particular  $l_x$  curve and we can use those randomly generated lifetimes to recover the underlying  $l_x$ . Not bad, but it is too soon to declare victory and go home.

The problem is that we have generated the lifetimes as if time were discrete. All death dates within an age category are indistinguishable from one another. But in real life, or real death as it were, time is continuous and  $l_x$  looks discrete only because of the limitations of data collection. For simulation (and other) purposes we need to do better. BUT how can we? the  $l_x$  values that we start out with are categorical?

Once again, the intrepid scientist must MAKE AN ASSUMPTION.

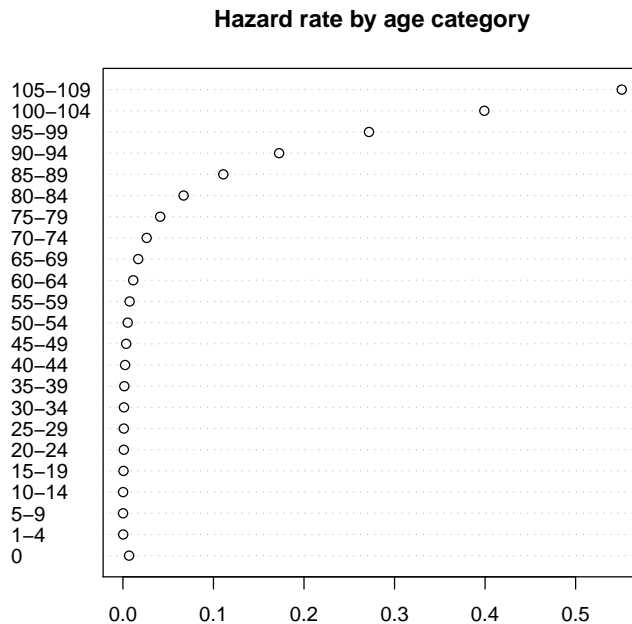
Since the  $l_x$  values are all that we have – we really cannot say with scientific sincerity that the process which produced those  $l_x$  values did not come from a universe where time is discrete. Nonetheless, since our immediate goal is to produce a vector of lifetimes that make sense in our universe AND which are consistent with the  $l_x$  values that we have... it is proper and fitting to assume that the true process underlying  $l_x$  is continuous. For centuries, demographers have chosen a piece-wise exponential distribution as the assumed underlying true model. Although one might question the truthiness of the piece-wise exponential assumption, it sure is mathematically convenient.

The defining characteristic of the exponential distribution is that the hazard rate is constant (which is just another way of saying that the distribution is "memoryless")<sup>1</sup>.

For a piece-wise exponential distribution, the hazard is piece-wise constant. That is, it changes across age categories, but for the duration of each age category it is constant.

As you no doubt recall from Ken's book the constant hazard associated with each age category is the negative slope of the log of lx. or:

```
> Nages<-length(us05$Age)
> rise<- log(us05$lx[-1]) - log(us05$lx[-Nages])
> run<- us05$width[-Nages]
> hazards <- -rise/run
> dotchart(hazards,labels=us05$Age,main="Hazard rate by age category")
```



With these hazard rates we can improve our simulated lifetime data by interpolating across the age intervals and thereby convert the step function of lx values into a continuous function.

The most straight forward approach (i.e. the only one I can think of) is to first generate a discrete lifetime by "bouncing" a random uniform u off of the survival curve (step function) to find the age interval that is appropriate

<sup>1</sup>Since our next goal will be to use our simulated data in a Cox Regression, this constant hazard thing could turn out to be convenient.

for each value of  $u$ . Higher values of  $u$  imply longer survival times. Since  $u$  is continuous it will never fall exactly on age interval boundary. We can use the "left over" bit of  $u$  to determine where *within* the age interval the lifetime (corresponding to the random  $u$ ) should fall. Since mortality within the age interval is piecewise exponential – and we know how to find the hazard rate – we just apply the inverse CDF (of the exponential) trick using the left over bit of  $u$  and the hazard rate for that age interval. What could be simpler?

Let's write a function that does that.

```
> lxCinverse<-function(u,LifeTable=us05){
+   ## This converts random uniforms into age at death values
+   ## according to the lx in LifeTable
+
+   ## u is a presumably random uniform -- but it needs to be between 0
+   ## and 1
+
+   stopifnot(length(u)>0)
+   ##
+   survival<-with(LifeTable,lx/lx[1])
+
+   ## First the discrete part: find the age category the u implies
+   ## based on the step function representation of lx
+
+   if (length(u)==1){
+     ## if x is a single scalar then so is ac.index.
+     ac.index<-sum( u <= survival )
+   }else{
+     ## if u is a vector then we need to compare each value of u
+     ## to each value of cdf which is trickier.
+
+     ac.index <- apply( outer (u,survival, FUN="<="),MARGIN=1,FUN=sum)
+   }
+
+   ## survivors to the lower age bound of the category in which x lives
+   age.atleast<-LifeTable$age.lower[ac.index]
+
+   ## hazard rate to which survivors shall be exposed until next age category
+   Lambda<-with(LifeTable,
+     {rise<-(log(lx[-1])-log(lx[-length(lx)]))
+       run<- age.lower[-1]-age.lower[-length(lx)]
+       -rise/run})
+   ## Unlikely to matter but...
+   Lambda<-c(Lambda,1) ## a very large u could land in 110+
+   ## select the lambda(s) that apply to each u
+   lambda<-Lambda[ac.index]
+ }
```

```

+
+   ## take the u that's left over after subtracting the portion of
+   ## log(u) that was required to get to the lower bound of the age
+   ## category, and apply the inverse CDF trick on the remaining bit.
+
+   ## if u= zero we can't log it.
+   uLeftOver<-ifelse(u>0, log(u) - log(survival[ac.index]),0)
+   extraLife<- uLeftOver/-lambda
+
+   return(age.atleast +extraLife)
+
+ }
> ## generate some age-at-deaths for random us. They ought to be between
> ## 0 and 110
> lxCinverse(runif(5))

[1] 53.65875 70.99169 50.54586 72.76985 74.20982

```

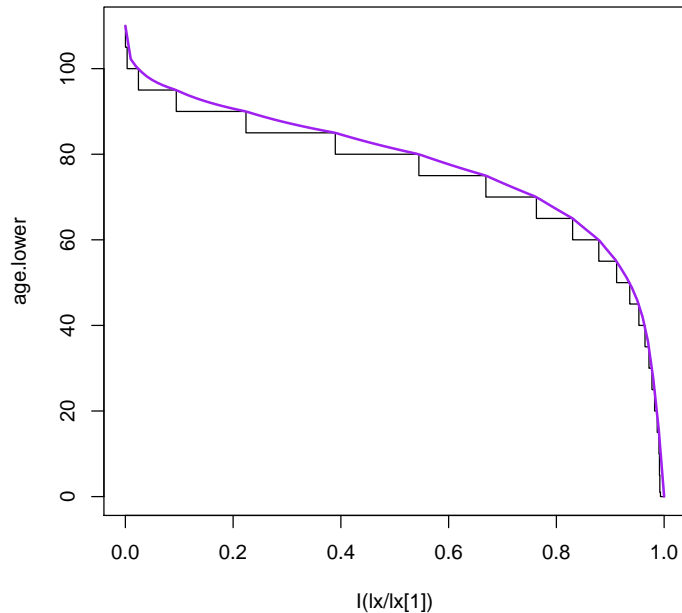
Using a couple of nifty R tricks, we can plot

```

> ## with() temporarily makes the columns of a data.frame act like
> ## vectors in the work space
> with(us05,plot(age.lower~I(lx/lx[1]),type='s'))
> ## curve() can plot the values of function
> curve(lxCinverse,from=0, to=1,lwd=2,add=TRUE,col='purple')
> title(main="Survival function: discrete and continuous from lx values")

```

### Survival function: discrete and continuous from lx values



Above, I asked the facetious question... "what could be simpler?" ... Well what if we dispensed entirely with the hazard rate bit and just "bounced" random uniforms off of a survival curve constructed by drawing straight lines (aka interpolating linearly) between the points  $(age, lx/lx[1])$  in other words just take the discrete survival curve and connect the dots.

It's fairly easy to do this with another cool R trick, the `approx()` function.

```
> #To take care of the open interval, we
> #assume that no one lives past 124
>
> survival<-c(with(us05,lx/lx[1]),0)
> survival.age<-c(us05$age.lower,110)
> ## the approx() function performs linear interpolation. It finds y
> ## values associated with what you give it as "xout" based on ##
> ## interpolation between the y and x values that you give it
>
> lxCinverse.cheap<-approx(y=survival.age,
+                           x=survival,
+                           xout=seq(0,1,length=1000))
> ## the output of approx is a list with components x and y
> names(lxCinverse.cheap)

[1] "x" "y"
```

```

> ## lots of graphics functions will take a list with x and y components
> plot(y=survival.age,x=survival,col='red',type='p')
> curve(lxCinverse,from=0, to=1,n=200,lwd=2,add=TRUE,col='purple')
> lines(lxCinverse.cheap,col='red',lwd=1)

```

It looks like the quick and dirty approach produces 99% of the value for about 1% of the effort. Where the slow and clean approach matters is only at the extreme ages – which of course is where Demographers get tenure. Sigh.

## 4 Assignment

All of the code in this document is in `~carlm/213/PropHazI/demonstration.r`. Create the usual `213/WeekN` directory; launch emacs so as to create a buffer/file called `exN.r` and import the demonstration file.

### 4.1 Part I

Run a Cox Proportional Hazard regression on a data set consisting of two slightly different randomly generated lifetimes. (don't panic below are some pretty good instructions on how to do this). Then answer the question below.

```

> ## 1) generate two sets of lifetimes with a slight difference
> lives0<- lxCinverse(u=runif(n=10000),LifeTable=us05) ### all the default values
> ## a slightly modified life table
> us05.vegetables<-us05
> us05.vegetables$lx<-us05.vegetables$lx^(exp(-.25))
> lives1<- lxCinverse(u=runif(n=10000),LifeTable=us05.vegetables)
> ## 2) create a data.frame consisting of the two vectors of lives and an
> ## indicator variable, "veggies" that tells which vector the lifetime
> ## is from slightly modified set of lifetimes
> Dset<-data.frame(lives=c(lives0,lives1),
+                 veggies=c(rep(0,length(lives0)),
+                 rep(1,length(lives1))))
> ### 3) load the survival library
> library(survival)
> ### 4) create a special type of variable that is used for survival analysis
> Dset$devar<-Surv(time=rep(0,dim(Dset)[1]),
+                 time2=Dset$lives,
+                 event=rep(1,dim(Dset)[1]))
> ## 5) run a cox proportional hazard regression
> cox.res<-coxph(devar ~ veggies,data=Dset)
> ## look at the results:
> summary(cox.res)

```

## 4.2 Part II

Run the same experiment as you did in Part I – but using the quick and dirty method of generating lifetimes. Note the huge difference in the results. If necessary, see the frustration management section below