Log-normal Regression with R*

```
> rm(list=ls());  options(scipen=999)
> # install.packages("survival",dependencies=TRUE) # Only need to do this once
> library(survival) # Do this every time
> # install.packages("asaur",dependencies=TRUE) # Only need to do this once
> library(asaur)
> # summary(pharmacoSmoking)
> # Make fixed-up data frame called quit
> quit = within(pharmacoSmoking,{
+ DayOfRelapse = Surv(ttr+1,relapse)
+ contrasts(grp) = contr.treatment(2,base=2) # Patch only is reference category
+ colnames(contrasts(grp)) = c('Combo') # Names of dummy vars -- just one
+ # Collapse race categories
+ Race = as.character(race) # Small r race is a factor. This is easier to modify.
+ Race[Race!='white'] = 'blackOther'; Race=factor(Race)
+ }) # Finished making data frame quit
> # with(quit, table(race,Race) )
>
> wmod =  survreg(DayOfRelapse ~ grp + age + employment , dist='weibull',
data=quit)
> summary(wmod) # This was model full2 in an earlier analysis
Call:
survreg(formula = DayOfRelapse ~ grp + age + employment, data = quit,
    dist = "weibull")
                  Value Std. Error     z           p
(Intercept)      1.4957     0.8414  1.78     0.07545
grpCombo         1.1023     0.3793  2.91     0.00366
age              0.0643     0.0186  3.45     0.00055
employmentother -1.2880     0.4617 -2.79     0.00528
employmentpt    -1.2123     0.5616 -2.16     0.03088
Log(scale)       0.5454     0.0894  6.10 0.000000001

Scale= 1.73

Weibull distribution
Loglik(model)= -464.3   Loglik(intercept only)= -476.5
      Chisq= 24.31 on 4 degrees of freedom, p= 0.000069
Number of Newton-Raphson Iterations: 5
n= 125
> lognorm = survreg(DayOfRelapse ~ grp + age + employment,dist='lognormal',
data=quit)
> summary(lognorm)
Call:
survreg(formula = DayOfRelapse ~ grp + age + employment, data = quit,
    dist = "lognormal")
                  Value Std. Error     z                   p
(Intercept)      0.8923     0.8719  1.02              0.3061
grpCombo         1.1248     0.4055  2.77              0.0055
age              0.0569     0.0182  3.13              0.0017
employmentother -1.0495     0.4766 -2.20              0.0277
employmentpt    -0.8757     0.6456 -1.36              0.1749
Log(scale)       0.7671     0.0798  9.61 <0.0000000000000002

Scale= 2.15

Log Normal distribution
Loglik(model)= -460.4   Loglik(intercept only)= -470.4
      Chisq= 20 on 4 degrees of freedom, p= 0.0005
Number of Newton-Raphson Iterations: 3
n= 125
```

```
> # Now predict the day of relapse for a 50-year-old in the patch-only condition
> # who is working part-time. Use the estimated median exp(muhat) as a prediction.
>
> # First do it the hard way
```

Prediction interval for t_{n+1} is from

$$\exp\left(\mathbf{x}_{n+1}^\top \widehat{\boldsymbol{\beta}} - 1.96\sqrt{\widehat{\sigma}^2 + \mathbf{x}_{n+1}^\top \widehat{\mathbf{C}}_n \mathbf{x}_{n+1}}\right)$$

to

$$\exp\left(\mathbf{x}_{n+1}^\top \widehat{\boldsymbol{\beta}} + 1.96\sqrt{\widehat{\sigma}^2 + \mathbf{x}_{n+1}^\top \widehat{\mathbf{C}}_n \mathbf{x}_{n+1}}\right)$$

where $\mathbf{C}_n$ is the estimated asymptotic covariance matrix of $\widehat{\boldsymbol{\beta}}$, obtained from the inverse of the Hessian.

```
> betahat = lognorm$coefficients; betahat

    (Intercept)         grpCombo              age employmentother    employmentpt
     0.89234850       1.12482022       0.05688783     -1.04950269     -0.87571454

> xnplus1 = cbind(1,0,50,0,1); xnplus1
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0   50    0    1
> sigmahat = lognorm$scale
> Vn = vcov(lognorm); round(Vn,4)

                (Intercept) grpCombo     age employmentother employmentpt
(Intercept)          0.7602  -0.0532 -0.0147          0.0632      -0.0404
grpCombo            -0.0532   0.1644 -0.0004         -0.0008      -0.0044
age                 -0.0147  -0.0004  0.0003         -0.0030      -0.0007
employmentother      0.0632  -0.0008 -0.0030          0.2272       0.0773
employmentpt        -0.0404  -0.0044 -0.0007          0.0773       0.4168
Log(scale)          -0.0031   0.0027  0.0001         -0.0017      -0.0025
                Log(scale)
(Intercept)        -0.0031
grpCombo            0.0027
age                 0.0001
employmentother    -0.0017
employmentpt       -0.0025
Log(scale)          0.0064

> Cn = Vn[(1:5),(1:5)]; round(Cn,4)
                (Intercept) grpCombo     age employmentother employmentpt
(Intercept)          0.7602  -0.0532 -0.0147          0.0632      -0.0404
grpCombo            -0.0532   0.1644 -0.0004         -0.0008      -0.0044
age                 -0.0147  -0.0004  0.0003         -0.0030      -0.0007
employmentother      0.0632  -0.0008 -0.0030          0.2272       0.0773
employmentpt        -0.0404  -0.0044 -0.0007          0.0773       0.4168
```

Prediction interval for t_{n+1} is from

$$\exp\left(\mathbf{x}_{n+1}^\top\widehat{\boldsymbol{\beta}} - 1.96\sqrt{\widehat{\sigma}^2 + \mathbf{x}_{n+1}^\top\widehat{\mathbf{C}}_n\mathbf{x}_{n+1}}\right)$$

to

$$\exp\left(\mathbf{x}_{n+1}^\top\widehat{\boldsymbol{\beta}} + 1.96\sqrt{\widehat{\sigma}^2 + \mathbf{x}_{n+1}^\top\widehat{\mathbf{C}}_n\mathbf{x}_{n+1}}\right)$$

where $\mathbf{C}_n$ is the estimated asymptotic covariance matrix of $\widehat{\boldsymbol{\beta}}$, obtained from the inverse of the Hessian.

```
> yhat = sum(xnplus1*betahat); yhat # This is estimated x'beta
[1] 2.861025
> exp(yhat) # This is predicted number of days
[1] 17.47944
> se = sqrt( sigmahat^2 + as.numeric(xnplus1 %*% Cn %*% t(xnplus1)) ); se
[1] 2.24123
> A = yhat-1.96*se; B = yhat+1.96*se; c(A,B)
[1] -1.531785  7.253836
> c( exp(A), exp(B)) # Prediction interval
[1]    0.2161495 1413.5165098
> exp(A)*24  # Lower limit in hours
[1] 5.187587
> exp(B)/365 # Upper limit in years
[1] 3.872648
```

So with 95% confidence, the old guy will be able to hold out between 5.2 hours and 3.9 years. Thank you very much.

```
> # Now the easy way
> oldguy = data.frame(grp='patchOnly',age=50,employment='pt')
> pred1 = predict(lognorm,newdata=oldguy,type='linear',se=TRUE) ; pred1
$fit
       1
2.861025

$se.fit
        1
0.6206626
```

se of $\widehat{y}_{n+1}$ is $\sqrt{\mathbf{x}_{n+1}^\top\widehat{\mathbf{C}}_n\mathbf{x}_{n+1}}$. Want se of $(y_{n+1} - \widehat{y}_{n+1}) = \sqrt{\widehat{\sigma}^2 + \mathbf{x}_{n+1}^\top\widehat{\mathbf{C}}_n\mathbf{x}_{n+1}}$.

```
> # Construct prediction interval
> se = sqrt(sigmahat^2+pred1$se^2)
> L = yhat - 1.96*se; U = yhat + 1.96*se
> t_hat= exp(yhat)
> lower95 = exp(L); upper95 = exp(U)
> pi = c(t_hat,lower95,upper95)
> names(pi) = c('t-hat','lower95','upper95')

> pi
      t-hat      lower95      upper95
  17.4794407    0.2161495 1413.5165098
```

Wide prediction intervals are a fact of life for many data sets. “Predicting” university calculus score from a bunch of good variables, using a normal model with no censoring ...

```
> round( predict(good,interval='prediction'), 1)
      fit  lwr   upr
1    44.6 17.6  71.6
2    30.4  3.2  57.7
3    59.5 32.6  86.4
4    62.5 35.5  89.5
5    69.2 42.3  96.1
6    43.3 16.2  70.5
7    68.8 41.7  96.0
8    44.6 17.6  71.6
9    46.9 19.8  74.0
10   78.2 51.2 105.2
14   72.2 45.2  99.3
17   61.5 34.3  88.7
18   80.0 53.0 107.0
19   52.4 25.5  79.3
21   59.4 32.2  86.5
22   68.5 41.6  95.5
23   59.8 32.6  86.9
25   64.8 37.8  91.8
26   48.8 21.9  75.7
28   63.3 36.3  90.3
29   59.8 32.9  86.7
32   46.9 20.0  73.9
33   58.9 31.5  86.3
34   69.6 42.6  96.6
36   50.5 23.2  77.8
43   71.0 43.7  98.3
45   65.5 38.3  92.7
46   39.4 12.4  66.3
49   55.1 28.2  81.9
50  100.4 72.5 128.3
51   62.1 35.1  89.0
52   47.9 21.0  74.8
53   85.1 58.0 112.2
55   80.7 53.5 107.9
57   59.3 32.1  86.6
65   43.9 16.7  71.1
67   57.1 30.3  84.0
71   56.4 29.5  83.2


567  74.3 47.1 101.5
568  65.1 38.2  92.1
570  70.8 43.8  97.8
571  52.6 25.6  79.6
572  68.2 41.3  95.2
574  56.7 29.7  83.7
576  64.7 37.6  91.7
577  50.5 23.4  77.6
579  57.6 30.7  84.5

Warning message:
In predict.lm(good, interval = "prediction") :
  predictions on current data refer to _future_ responses
```

```
> # First, truly lognormal data
>
>
> rm(list=ls());  # options(scipen=999)
> Ex = 10; SDx = 1 # Parameters of (normal) explanatory variable X
> beta0 = -10; beta1 = 1; sigma = 2 # Regression parameters
> # install.packages("survival",dependencies=TRUE) # Only need to do this once
> library(survival)
>
> # Simulate: simulate a data set of size n, generate prediction intervals for a
second set of data of size n2, and record the proportion in the interval. Do this
nsim times. The mean proportion in the interval is a MC estimate of the coverage
probability.
>
> nsim = 10000; propin = numeric(nsim) # Proportion in the interval.
> set.seed(9999)
>
> for(sim in 1:nsim)
+ {
+ n = 200
+ delta = numeric(n) # Indicator for uncensored, initially zero
+ x = round(rnorm(n,Ex,SDx),1)
+ mu = beta0 + beta1*x
+ y = rnorm(n,mu,sigma); lifetime = exp(y)
+ # sort(lifetime)
+ # hist(sort(lifetime)[1:(n-2)],breaks=20)
+ censortime = abs(rcauchy(n)) # Absolute Cauchy censoring time
+ # censortime = 1/runif(n) - 1 # Shifted Pareto censoring time
+ # If censoring time is greater than lifetime, then it's NOT censored.
+ delta[censortime>lifetime] = 1; # table(delta)
+ # Minumum of censortime and lifetime is what we can observe.
+ Time = pmin(censortime,lifetime) # pmin is parallel minimum.
+ # Time = round(Time,3); # summary(Time)
+
+ # round(cbind(x,lifetime,censortime,Time,delta)[1:10,],3) # Take a look
+ # lndata = cbind(x,Time,delta); # lndata # This is all you can see in practice.
+ # lndata = data.frame(lndata) # Make it a data frame
+ # head(lndata)
+
+ # complete = lm(y~x) # Normal regression on complete (log) data -- no censoring
+ # summary(complete)
+ # predict(complete, interval='prediction')
+
+ # Now lognormal regression on censored data
+ stime = Surv(Time,delta)
+
+ lognorm1 =  survreg(stime ~ x , dist='lognormal')
+ # summary(lognorm1)
+
+ betahat = lognorm1$coefficients
+ betahat0 = betahat[1]; betahat1 = betahat[2]
+ sigmahat = lognorm1$scale
+
+ # Now take a large sample from that same population, and see how the prediction
+ # interval works. Only check the uncensored observations.
+
```

```
+ # Now take a large sample from that same population, and see how the prediction
+ # interval works. Only check the uncensored observations.
+
+ n2 = 100
+ delta2 = numeric(n2) # Indicator for uncensored, initially zero
+ x2 = round(rnorm(n2,Ex,SDx),1)
+ mu2 = beta0 + beta1*x2
+ y2 = rnorm(n2,mu2,sigma); lifetime2 = exp(y2)
+ # censortime2 = abs(rcauchy(n2)) # Absolute Cauchy censoring time
+ censortime2 = 1/runif(n2) - 1 # Shifted Pareto censoring time
+ # If censoring time is greater than lifetime, then it's NOT censored.
+ delta2[censortime2>lifetime2] = 1;  # table(delta2)
+ m = sum(delta2) # Number of uncensored observations
+ # Minumum of censortime and lifetime is what we can observe.
+ Time2 = pmin(censortime2,lifetime2) # pmin is parallel minimum.
+ # Time2 = round(Time2,4)
+ stime2 = Surv(Time2,delta2)
+ data2 = data.frame(x2,stime2)
+ colnames(data2) = c('x','stime')
+ # Variables must have the same names as in original data
+
+ data2 = subset(data2,delta2 == 1) # Select uncensored observations
+ dim(data2)
+
+ pred = predict(lognorm1,newdata=data2,type='linear',se=TRUE)
+ # yhat = betahat0 + betahat1*x2 # By "hand"
+ # cbind(yhat,pred$fit) # Same
+
+ yhat = pred$fit
+ vhat = pred$se^2 # Estimated variance of x_i'betahat: Verified
+ se = sqrt(sigmahat^2+vhat) # Denominator of Z stat for prediction interval
+ # Lower and upper prediction limits for y2 = log(Time2)
+ L = yhat - 1.96*se; U = yhat + 1.96*se
+ lower95 = exp(L); upper95 = exp(U)
+ time2 = subset(Time2,delta2==1) # Just uncensored times
+
+ ininterval = logical(m) # Logical vector of length m, initialized to FALSE
+ ininterval[(lower95 < time2) & (time2 < upper95)] = TRUE
+ propin[sim] = mean(ininterval) # Proportion in the interval
+ } # Next Simulation
>
> mean(propin,na.rm=TRUE) # Estimated coverage probability

[1] 0.9467085
```

```
> # ------------------------------------------------------------------- #
>
> # What if the data were really Weibull?
> # Sigma = 1 makes it exponential
>
> rm(list=ls()); # options(scipen=999)
> Ex = 5; SDx = 1 # Parameters of (normal) explanatory variable X
> beta0 = -10; beta1 = 2; sigma = 1 # Regression parameters. sigma = 1 makes it
exponential
>
> # Simulate
> set.seed(9999)
> n = 200; delta = numeric(n) # Indicator for uncensored, initially zero
> x = round(rnorm(n,Ex,SDx),1)
> mu = beta0 + beta1*x
> epsilon = rexp(n)
> lifetime = exp(mu)*epsilon^sigma
> # sort(lifetime)
> censortime = 1/runif(n) - 1 # Shifted Pareto censoring time
> # If censoring time is greater than lifetime, then it's NOT censored.
> delta[censortime>lifetime] = 1; # table(delta)
> # Minumum of censortime and lifetime is what we can observe.
> Time = pmin(censortime,lifetime) # pmin is parallel minimum.
> # Time = round(Time,3)
> # round(cbind(x,lifetime,censortime,Time,delta)[1:10,],3) # Take a look
> # wdata = cbind(x,Time,delta); # wdata # This is all you can see in practice.
> # head(wdata)
>
> ########################
>
> rm(list=ls());  # options(scipen=999)
> Ex = 10; SDx = 1 # Parameters of (normal) explanatory variable X
> beta0 = -10; beta1 = 1; sigma = 2 # Regression parameters
> # install.packages("survival",dependencies=TRUE) # Only need to do this once
> library(survival)
>
> # Simulate: simulate a data set of size n, generate prediction intervals for a
second set of data of size n2, and record the proportion in the interval. Do this
nsim times. The mean proportion in the interval is a MC estimate of the coverage
probability.
>
> nsim = 10000; propin = numeric(nsim) # Proportion in the interval.
> set.seed(9999)
>
> for(sim in 1:nsim)
+ {
+ n = 200; delta = numeric(n) # Indicator for uncensored, initially zero
+ x = round(rnorm(n,Ex,SDx),1)
+ mu = beta0 + beta1*x
+ epsilon = rexp(n)
+ lifetime = exp(mu)*epsilon^sigma
+ # sort(lifetime)
+ censortime = 1/runif(n) - 1 # Shifted Pareto censoring time
+ # If censoring time is greater than lifetime, then it's NOT censored.
+ delta[censortime>lifetime] = 1; # table(delta)
+ # Minumum of censortime and lifetime is what we can observe.
+ Time = pmin(censortime,lifetime) # pmin is parallel minimum.
+
+ # Now lognormal regression on censored data
+ stime = Surv(Time,delta)
+
+ lognorm2 =  survreg(stime ~ x , dist='lognormal')
+
```

```
+ betahat = lognorm2$coefficients
+ betahat0 = betahat[1]; betahat1 = betahat[2]
+ sigmahat = lognorm2$scale
+
+ # Now take a sample from that same population, and see how the prediction
+ # interval works. Only check the uncensored observations.
+
+ n2 = 100
+ delta2 = numeric(n2) # Indicator for uncensored, initially zero
+ x2 = round(rnorm(n2,Ex,SDx),1)
+ mu2 = beta0 + beta1*x2
+
+ epsilon2 = rexp(n2)
+ lifetime2 = exp(mu2)*epsilon2^sigma
+ censortime2 = 1/runif(n2) - 1 # Shifted Pareto censoring time
+ # If censoring time is greater than lifetime, then it's NOT censored.
+ delta2[censortime2>lifetime2] = 1; # table(delta2)
+ m2 = sum(delta2)
+ # Minumum of censortime2 and lifetime2 is what we can observe.
+ Time2 = pmin(censortime2,lifetime2) # pmin is parallel minimum.
+ # Time2 = round(Time2,3)
+ stime2 = Surv(Time2,delta2)
+
+ data2 = data.frame(x2,stime2)
+ colnames(data2) = c('x','stime')
+ # Variables must have the same names as in original data
+
+ data2 = subset(data2,delta2 == 1) # Select uncensored observations
+ pred = predict(lognorm2,newdata=data2,type='linear',se=TRUE)
+
+ yhat = pred$fit
+ vhat = pred$se^2 # Estimated variance of x_i'betahat: Verified
+ se = sqrt(sigmahat^2+vhat) # Denominator of Z stat for prediction interval
+ # Lower and upper prediction limits for y2 = log(Time2)
+ L = yhat - 1.96*se; U = yhat + 1.96*se
+ lower95 = exp(L); upper95 = exp(U)
+ time2 = subset(Time2,delta2==1) # Just uncensored times
+
+ ininterval = logical(m2) # Logical vector of length m2, initialized to FALSE
+ ininterval[(lower95 < time2) & (time2 < upper95)] = TRUE
+ propin[sim] = mean(ininterval) # Proportion in the interval
+ } # Next Simulation
>
> mean(propin,na.rm=TRUE) # Estimated coverage probability

[1] 0.9391845

# So it works in this example even if the distribution is exponential and not log-
normal. That's encouraging but it's not a general proof. How about other, nameless
distributions? Here is a way to check the predictive utility for a particular data
set. The idea is to predict each uncensored value in turn, based on the other n-1
data vectors. Illustrate with the smoking data.
```

Illustrate with the smoking data

```
> rm(list=ls());  options(scipen=999)
> # install.packages("survival",dependencies=TRUE) # Only need to do this once
> library(survival) # Do this every time
> # install.packages("asaur",dependencies=TRUE) # Only need to do this once
> library(asaur)
> # summary(pharmacoSmoking)
> # Make fixed-up data frame called quit
> quit = within(pharmacoSmoking,{
+ DayOfRelapse = Surv(ttr+1,relapse)
+ contrasts(grp) = contr.treatment(2,base=2) # Patch only is reference category
+ colnames(contrasts(grp)) = c('Combo') # Names of dummy vars -- just one
+ # Collapse race categories
+ Race = as.character(race) # Small r race is a factor. This is easier to modify.
+ Race[Race!='white'] = 'blackOther'; Race=factor(Race)
+ }) # Finished making data frame quit
> # with(quit, table(race,Race) )
>
> head(quit)
   id ttr relapse         grp age gender     race employment yearsSmoking
1  21 182       0   patchOnly  36   Male    white         ft           26
2 113  14       1   patchOnly  41   Male    white      other           27
3  39   5       1 combination  25 Female    white      other           12
4  80  16       1 combination  54   Male    white         ft           39
5  87   0       1 combination  45   Male    white      other           30
6  29 182       0 combination  43   Male hispanic         ft           30
  levelSmoking ageGroup2 ageGroup4 priorAttempts longestNoSmoke       Race
1        heavy     21-49     35-49             0              0      white
2        heavy     21-49     35-49             3             90      white
3        heavy     21-49     21-34             3             21      white
4        heavy       50+     50-64             0              0      white
5        heavy     21-49     35-49             0              0      white
6        heavy     21-49     35-49             2           1825 blackOther
  DayOfRelapse
1         183+
2           15
3            6
4           17
5            1
6         183+


> n = dim(quit)[1]; n
[1] 125


> uncen = subset(1:n,quit[,3]==1) # Row numbers of the uncensored data
> m = length(uncen); m
[1] 89


> # Fill these vectors in the loop
> y_i = y_hat = t_hat = st_er = lower95 = upper95 = Z = numeric(m)
>
> # For each uncensored observation, get a prediction interval for y = log(t),
> # with that observation left out, so that beta-hat and Vn are based on the other
> # n-1 observations, censored as well as uncensored.
```

```
> # For each uncensored observation, get a prediction interval for y = log(t),
> # with that observation left out, so that beta-hat and Vn are based on the other
> # n-1 observations, censored as well as uncensored.
>
> for(j in 1:m)
+     {
+     # leftout is a data frame, with uncensored observation j left out
+     leftout = quit[-uncen[j],]
+     leftout$grp = factor(leftout$grp)
+     d = quit[uncen[j],] # The left-out case
+     y = log(d[[2]]+1) # Log day of relapse for omitted case.
+                       # Double bracket makes d[2] it numeric.
+     y_i[j] = y
+     model = survreg(DayOfRelapse ~ grp + age + employment,
+                     dist='lognormal', data=leftout)
+     sigmahat = model$scale
+     pred = predict(model,newdata=d,type='linear',se=TRUE)
+     yhat = pred$fit; vhat = pred$se^2 # That's vhat of x_j'betahat
+     se = sqrt(sigmahat^2+vhat) # Denominator of Z stat for prediction interval
+     st_er[j] = se
+     # Lower and upper prediction limits for log(Time_j)
+     L = yhat - 1.96*se; U = yhat + 1.96*se
+     y_hat[j] = yhat; t_hat[j] = exp(yhat)
+     lower95[j] = exp(L); upper95[j] = exp(U)
+     Z[j] = (y-yhat)/se # Reject H0: mu=0 if y is very large (pos or neg).
+
+     } # Next uncensored observation>
>
> mean(abs(Z)<1.96)
[1] 0.988764


> cbind(uncen[abs(Z)>1.96], Z[abs(Z)>1.96]) # Locates the "significant" Z stats
     [,1]      [,2]
[1,]   60 -2.385464
> # Bonferroni correcting for 89 tests might be a good idea.
>
> ftime = quit$ttr[uncen]+1 # Uncensored dy of relapse
> ininterval = logical(m) # Logical vector of length m, initialized to FALSE
> ininterval[(lower95 < ftime) & (ftime < upper95)] = TRUE
> mean(ininterval) # Proportion in the interval
[1] 0.988764
> 88/89
[1] 0.988764

> # Is 98% SIGNIFICANTLY greater than 95%?
> phat = mean(ininterval) # = 88/89
> z1 = sqrt(89)*(phat-0.95) / sqrt(0.95*(1-0.95)); z1
[1] 1.677943
>
> z2 = sqrt(89)*(phat-0.95) / sqrt(phat*(1-phat)); z2
[1] 3.469547
```

Final note: The Z statistics could be called “standardized deleted residuals.”
