Examples of Constructing Confidence Intervals by Likelihood Test

Mai Zhou

Department of Statistics
University of Kentucky
Lexington, KY 40536

SUMMARY

These notes details several examples that help understand the technique of profiling the (either regular or empirical) likelihood. The likelihood is then used to produce (Wilks) confidence intervals. This provides more details to a technique used in the book *Empirical Likelihood Methods in Survival Analysis* (Zhou, 2016), and show case the capability of some R functions inside packages emplik and ELYP.

AMS 2000 Subject Classification: Primary 60E15; secondary 60G30.
Key Words and Phrases: Empirical likelihood, Confidence region and interval.

1 Confidence Intervals Based on the Likelihood Ratio Test[1]

1. Let's consider the binomial case. Let X be a random variable distributed as a Binomial(n, p). Then, we know that the maximum likelihood estimator of p will be $\hat{p} = \frac{x}{n}$. And, from our introductory statistics courses, we know that we can calculate a confidence interval for p by the following formula for a Wald confidence interval:

$$\hat{p} \pm z\sqrt{\frac{\hat{p}\left(1-\hat{p}\right)}{n}} \tag{1}$$

One issue with this type of confidence interval is that it's possible for it to contain values outside of the interval $(0, 1)$, which doesn't make any sense. There are some known transformations which eliminate (or reduce the likelihood) of this possibility. So, let's consider an arbitrary (monotone) transformation $g\left(p\right)$ and construct a confidence interval for it. From the formula for a Wald confidence interval, we have:

$$g\left(\hat{p}\right) \pm z\sqrt{Var\left(g\left(\hat{p}\right)\right)} \tag{2}$$

where we used the fact that the MLE is invariant, so we know $\widehat{g\left(p\right)} = g\left(\hat{p}\right)$. And, by the delta method, we have:

$$Var\left(g\left(\hat{p}\right)\right) \approx \left[g'(p)\right]^2 Var\left(\hat{p}\right) \tag{3}$$

[1]These notes were obtained using ELPY version 0.7-3 and emplik version 1.0-1

This gives us the confidence interval for $g(p)$: $g(p) \in [L, U]$. Which yields the following confidence interval for p: $p \in [g^{-1}(L),\ g^{-1}(U)]$.

So, why do we use transformations and why are they useful?
If you use a "good" transformation, then you can guarantee that the confidence interval for p will be between $(0, 1)$.

But, there exist many different "good" transformations. Which one is the best? (kind of arbitrary, in terms of accurate, speed, etc.).
The selection of the best "good" transformation is somewhat arbitrary. Your decision could be based on accuracy, speed, or any other criteria that you deem important. An example of this arbitrariness is the discrepancy between the transformation used by SAS and that used by R (in their confidence intervals based on the Kaplan-Meier for survival probability). In SAS, they use $\log(-\log(1-p))$, while R uses $\log(1-p)$.

2. Hypothesis Testing
Let's consider testing the following hypotheses:

$$H_0 : p = p_0 \tag{4}$$
$$H_1 : p \neq p_0 \tag{5}$$

Using the likelihood ratio test, we would reject the null when:

$$-2 \times \log\left[\frac{likelihood(x, p_0)}{likelihood(x, \hat{p})}\right] > C = \chi^2_{1,0.95} \tag{6}$$

We can simplify the left-hand side (without the -2 and took p for p_0) of this inequality to:

$$x\log(p) + (n-x)\log(1-p) - x\log(\hat{p}) - (n-x)\log(1-\hat{p}) \tag{7}$$
$$= x\log(\frac{p}{\hat{p}}) + (n-x)\log(\frac{1-p}{1-\hat{p}}) \tag{8}$$

Now, there are two conditions the we need to take into consideration:
a. $p \neq 0$ and $p \neq 1$
b. $\hat{p} \neq 0$ and $\hat{p} \neq 1$
We can write a function in R which calculates this value for us:

```
BinLLR = function(n,x,p){

if(x >= n) stop("100 percent success?")
if(p <= 0) stop("P must between 0 and 1")
if(x <= 0) stop("no seccess?")
if(p >= 1) stop("P must < 1")

phat = x/n
```

```
-2*(x*log(p/phat)+(n-x)*log((1-p)/(1-phat)))
}

ts = BinLLR(n=100,x=30,p=0.5)
1 - pchisq(ts,df=1)

## [1] 4.977722e-05
```

This show the P-value is near zero in testing p=0.5. When we test various values of success probabilities, those that lead to a P-value larger than 0.05 form a 95% confidence interval for p. This may be accomplished by the function findUL( ) from package emplik. Which gives us a confidence interval of [0.2160426, 0.3940769].

3.
```
myfun <- function(theta, n, x){
temp <- BinLLR(n=n, x=x, p=theta)
list("-2LLR"=temp)
}

library(emplik)
## Warning: package 'emplik' was built under R version 3.2.2
## Loading required package: quantreg
## Warning: package 'quantreg' was built under R version 3.2.2
## Loading required package: SparseM
## Warning: package 'SparseM' was built under R version 3.2.2
## Loading required package: methods
##
## Attaching package: 'SparseM'
## The following object is masked from 'package:base':
##
##    backsolve

findUL(fun=myfun, MLE=0.3, n=100, x=30)

## $Low
## [1] 0.2160426
##
## $Up
## [1] 0.3940769
##
## $FstepL
## [1] 1e-10
##
```

```
## $FstepU
## [1] 1e-10
##
## $Lvalue
## [1] 3.84
##
## $Uvalue
## [1] 3.84
```

4. Good Properties of the Likelihood Ratio Test Approach to Confidence Intervals

   (a) The LRT method is transformation invariant

      i. Monotone Transformations
      If you have a confidence interval for p, ($lower$, $upper$), and you want a confidence interval for $\log(p)$ (or any monotone function), you can find it by taking the log of the lower and upper bounds for p, ($\log(lower)$, $\log(upper)$). This is much easier than the Wald method which involves calculating derivatives, using the delta method, etc.

      ii. Transformations Which Aren't Monotone
      This property even holds for transformations which aren't monotone. If you have a transformation $g(p)$ which isn't monotone and a confidence interval for p, (l, u), then you can find your transformed confidence interval by the following steps:

$$upper_{new} = \max_{l<p<u} g(p)$$
$$lower_{new} = \min_{l<p<u} g(p)$$

   (b) The LRT method is range preserving
   This means that the confidence intervals it produces will always be inside of the parameter space. So, in the binomial case, the confidence intervals produced by the LRT method will always be between $(0, 1)$.

   (c) The LRT method doesn't necessarily give symmetric intervals
   The asymmetry of the intervals is more similar to the actual distributions under consideration. So, in the binomial case when we have a small value of p, we know that there are more values above p than below it, so we'd expect our interval to reflect this. The Wald interval, on the other hand, will always be symmetric which we know is not always true of the distributions under consideration.

5. Possible Draw Backs of the Likelihood Ratio Test Approach to Confidence Intervals

   (a) You may not have to use the delta method or take any derivatives, but you still have to find maximums and roots. These calculations should be easily done using computers, so this isn't too much of an issue.

2 The Ratio of Two Success Probabilities

Let's consider the case where we have two samples, each containing a parameter: success probability. From each sample we will have an n_i and an x_i, where n_i is the total number of units observed in the i^{th} sample and x_i is the total number of successes observed in the i^{th} sample. We can model this situation by two binomial random variables, X_1 and X_2, where $X_1 \sim Binomial\,(n_1, p_1)$ and $X_2 \sim Binomial\,(n_2, p_2)$, where the MLE for p_1 is $\widehat{p_1} = \frac{x_1}{n_1}$ and the MLE for p_2 is $\widehat{p_2} = \frac{x_2}{n_2}$.

Now, let's suppose that we want to use the likelihood ratio test method of creating a confidence interval for the ratio of $\frac{p_1}{p_2}$. Since maximum likelihood estimators are invariant, we know that the MLE for $\frac{p_1}{p_2}$ will simply be $\frac{\hat{p_1}}{\hat{p_2}} = \frac{x_1 n_2}{x_2 n_1}$. We also know that since independent assumptions made the log-likelihood ratio additive, it follows that $LLR = LLR\,(\text{sample}_1) + LLR\,(\text{sample}_2)$. And since we want to use the likelihood ratio test method, we'll need to consider the following hypotheses:

$$H_0 : \frac{p_1}{p_2} = \theta$$
$$H_1 : \frac{p_1}{p_2} \neq \theta$$

Under the likelihood ratio test, our initial parameters of interest are p_1 and p_2. But this is equivalent to considering the parameters p_1 and θ, since $\theta = \frac{p_1}{p_2}$. If we take p_1 and θ to be our parameters of interest, then it follows that p_1 can be viewed as our nuisance parameter since θ is what we're truly interested in. From our previous coursework, we know that we can get rid of a nuisance parameter in the log-likelihood ratio by simply "maximizing it out." In this case, we'll have

$$LLR(\theta) = \max_{\left\{p_1, p_2 : \frac{p_1}{p_2} = \theta\right\}} \{LLR\,(\text{sample}_1) + LLR\,(\text{sample}_2)\}$$

And then we can multiply this by -2 to obtain our likelihood ratio test statistic:

$$\begin{aligned} -2\,[LLR\,(\theta)] &= -2\left[\max_{\left\{p_1, p_2 : \frac{p_1}{p_2} = \theta\right\}} \{LLR\,(\text{sample}_1) + LLR\,(\text{sample}_2)\}\right] \\ &= \min_{\left\{p_1, p_2 : \frac{p_1}{p_2} = \theta\right\}} \{-2 \cdot LLR\,(\text{sample}_1) - 2 \cdot LLR\,(\text{sample}_2)\} \end{aligned}$$

So, once we have our likelihood ratio test statistic, we can compare it to 3.84 and determine whether or not we reject H_0. Then, we can create a confidence interval for $\theta = \frac{p_1}{p_2}$ by finding the values of θ which give us a likelihood ratio test statistic less than 3.84.
Below is the R code for the computing a confidence interval for the ratio of two success probabilities using the likelihood ratio test method. Note that the code is for $\theta = \frac{p_2}{p_1}$ and not $\theta = \frac{p_1}{p_2}$. But, using the invariance property of the likelihood ratio confidence intervals, we can simple take the inverse of the confidence limits to get those for $\theta = \frac{p_1}{p_2}$.

```
BinRatio <- function(n1, x1, n2, x2, step=0.1, initStep=0.1, level=3.84){

if(x1 >= n1) stop("all success(sample 1)?")
if(x2 >= n2) stop("all success(sample 2)?")

MLE <- (x2*n1)/(x1*n2)   #### recall we are to find CI for p2/p1, not p1/p2.
EPS <- .Machine$double.eps

Theta <- function(theta, n1=n1, x1=x1, n2=n2, x2=x2)
  {
       llr <- function(const, n1, x1, n2, x2, theta) {
              npllik1 <- -2*(x1*log((const*n1)/x1) +
                            (n1-x1)*log(((1-const)*n1)/(n1-x1)))
              npllik2 <- -2*(x2*log((const*theta*n2)/x2) +
                            (n2-x2)*log(((1-const*theta)*n2)/(n2-x2)))
        return(npllik1 + npllik2)
        }
       upBD <- min( 1-EPS, 1/theta - EPS)
       temp <- optimize(f = llr,
                        lower = EPS,
                        upper = upBD,
                        n1 = n1,
                        x1 = x1,
                        n2 = n2,
                        x2 = x2,
                        theta = theta)

    cstar <- temp$minimum
    val <- temp$objective
    pvalue <- 1 - pchisq( val, df=1)
    list(`-2LLR` = val, cstar = cstar, Pval=pvalue)
  }

value <- 0
step1 <- step
Lbeta <- MLE - initStep
for( i in 1:8 ) {
while(value < level) {
Lbeta <- Lbeta - step1
value <- Theta(theta = Lbeta, n1=n1, x1=x1, n2=n2, x2=x2)$'-2LLR'
}
Lbeta <- Lbeta + step1
step1 <- step1/10
value <- Theta( theta =Lbeta, n1=n1, x1=x1, n2=n2, x2=x2)$'-2LLR'
}
```

```
value1 <- value
value <- 0

Ubeta <- MLE + initStep
for( i in 1:8 ) {
   while(value < level) {
    Ubeta <- Ubeta + step
    value <- Theta(theta=Ubeta, n1=n1, x1=x1, n2=n2, x2=x2 )$'-2LLR'
   }
   Ubeta <- Ubeta - step
   step <- step/10
   value <- Theta(theta=Ubeta, n1=n1, x1=x1, n2=n2, x2=x2)$'-2LLR'
}
return( list(Low=Lbeta, Up=Ubeta, FstepL=step1, FstepU=step,
Lvalue=value1, Uvalue=value) )
}

BinRatio(n1=100, x1=30, n2=90, x2=33)

## $Low
## [1] 0.8149795
##
## $Up
## [1] 1.844754
##
## $FstepL
## [1] 1e-09
##
## $FstepU
## [1] 1e-09
##
## $Lvalue
## [1] 3.84
##
## $Uvalue
## [1] 3.84
```

Exercise: Similar to the above BinRatio( ) function, write an R function BinDiff( ), that will produce the Wilks confidence interval for the difference of two binomial success probabilities.

3 The Weibull Distribution

We know from survival analysis that a random variable, X, following a Weibull distribution will satisfy the following probability:

$$P(X > t) = e^{-(\lambda t)^{\beta}} \quad \text{for} \quad t \geq 0 \quad \text{and} \quad \lambda, \beta > 0$$

If we have a sample of size n from a Weibull distribution, then we can derive the log-likelihood function for λ and β:

$$\log\left(L\left(\lambda, \beta | X_1, \ldots, X_n\right)\right) = \sum_{i=1}^{n} \left[\log(\beta) + (\beta - 1)\log(\lambda x_i) + \log(\lambda) - (\lambda x_i)^{\beta}\right]$$

But in survival analysis, it is common to have censored data. This means that we have some X values that we don't know the exact value of, only that they're greater than some specific value. As an example, consider the case where you track ants for a 12 month period and you record the time at which they die. If, at the end of that 12 month period, an ant is still alive, then all you know is that it survived for longer than 12 months. So, if ant$_9 = X_9$ is still alive, then you know that it survived for some time greater than 12 months. So, the survival value of ant$_9$ is $X_9 > 12$, which we denote by $X_9 = 12^{+}$. When dealing with survival data, it's common to represent it with two vectors. The first vector contains the survival values, and the second vector is the corresponding "status" vector, which contains a 1 if the corresponding survival value is a true value and a 0 if the corresponding survival value is a censored value. Using the survival package in R, you can convert a a pair of vectors into survival data using the Surv function. Here's an example:

```
require(survival)

x = c(5,3,7,2,8,9,5,4,6,2)
d = c(0,1,1,1,1,1,0,0,1,1)

Surv(x,d)

##  [1] 5+ 3  7  2  8  9  5+ 4+ 6  2
```

How does censored data change our likelihood function?
For an observation with a status of 1, its contribution to the likelihood function is simply $f(x_i|\lambda, \beta)$. For an observation with a status of 0, its contribution to the likelihood function is the probability $P(X > x_i) = e^{-(\lambda x_i)^{\beta}}$. So, if we have some censored data, then our likelihood function becomes (this in fact assumes independent censorship etc. For more detailed discussion of likelihood formation please my other note):

$$L\left(\lambda, \beta | X_1, \ldots, X_n\right) = \prod_{i=1}^{n} \left[f\left(x_i|\lambda, \beta\right)\right]^{\delta_i} \left[e^{-(\lambda x_i)^{\beta}}\right]^{1-\delta_i}$$

where δ_i is the status of the i^{th} observation. Below is some R code for the computing the log-likelihood for a Weibull distribution with censored data:

```r
WeibLL <- function(x, d, lam, beta){
  ###  lam  can be a vector of length(x), but use with care.
  ###  beta must be a scalar.
  ##############################
  ### if you use survreg of Survival package, then beta=1/scale
  ### and  lam=exp(-location)
  ############################################

  if (any((d != 0) & (d != 1) ))
    stop("d must be 0(right-censored) or 1(uncensored)")
  if(any(lam <= 0)) stop("lam must >0")
  if(beta <= 0) stop("beta must > 0")

  sum(d*(log(beta)+(beta-1)*log(x)+ beta*log(lam))) - sum((x*lam)^beta)
}
```

Finding the MLE

Calculating the MLE of the Weibull distribution can be difficult, especially with censored data, so we'll use a pre-made function in R. Under the survival package, we can use the survreg function to calculate the MLE by just fitting the intercept. Here's some sample code:

```r
t = c(5,4,6,8,11,12,16,14,21,26,33,44)
d = c(1,0,1,1,1,1,1,1,1,0,1,1)

survreg( Surv(t,d) ~ 1, dist = "weibull")
```

```
## Call:
## survreg(formula = Surv(t, d) ~ 1, dist = "weibull")
##
## Coefficients:
## (Intercept)
##    3.052159
##
## Scale= 0.6375651
##
## Loglik(model)= -38.5   Loglik(intercept only)= -38.5
## n= 12
```

One issue with this function is that it doesn't return the MLE's for the Weibull distribution, but the MLE's for the extreme value distribution which is just the log of a Weibull distribution. Since the MLE is/are invariant, we know that we can convert the MLE of the extreme value distribution to that of the Weibull distribution by a transformation. By

using the following transformations, we can find the MLE for the Weibull distribution:

$$\lambda = e^{-location}$$
$$\beta = \frac{1}{scale}$$

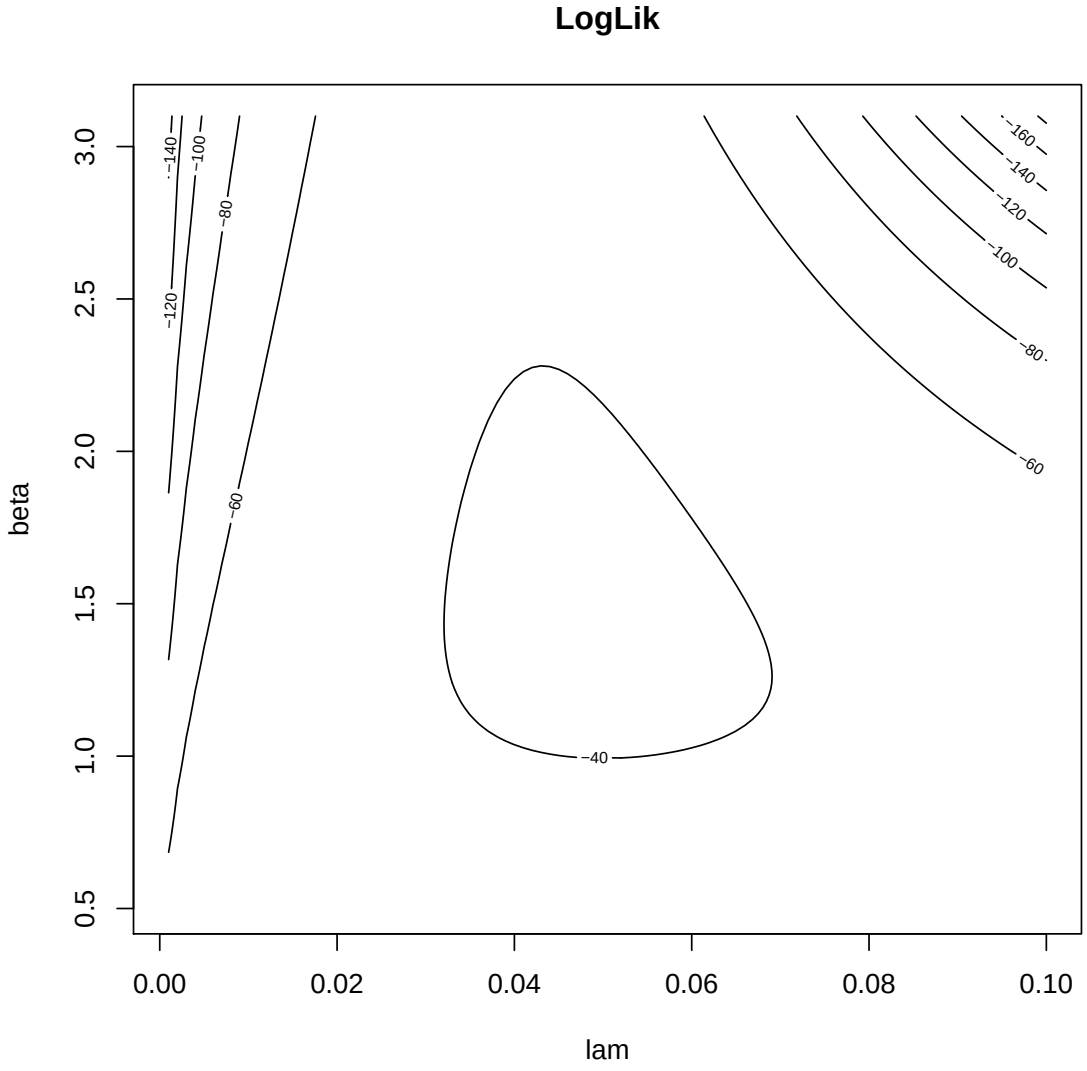
where *location* and *scale* are the MLE/parameters of the extreme value distribution. In R, we can find the MLE of a Weibull distribution in the following way:

```
t = c(5,4,6,8,11,12,16,14,21,26,33,44)
d = c(1,0,1,1,1,1,1,1,1,0,1,1)

reg = survreg( Surv(t,d) ~ 1, dist = "weibull")

location = as.vector(reg$coefficients)
scale = as.vector(reg$scale)

exp(-location)

## [1] 0.04725676

1/scale

## [1] 1.568467

reg$loglik

## [1] -38.53396 -38.53396

Maxi <- reg$loglik[1]
print(Maxi)

## [1] -38.53396
```

Now that we have the MLE, the next step is to turn this into a confidence interval for the survival probability, $P(X > t) = e^{-(\lambda t)^{\beta}}$ where t is some fixed time point. So, let's first look at a plot of the likelihood function with respect to λ and β when we take $t = 15$:

```
WeibLL <- function(x, d, lam, beta){
  if (any((d != 0) & (d != 1) ))
    stop("d must be 0(right-censored) or 1(uncensored)")
  if(any(lam <= 0)) stop("lam must >0")
  if(beta <= 0) stop("beta must > 0")

  sum(d*(log(beta)+(beta-1)*log(x)+ beta*log(lam))) - sum((x*lam)^beta)
```

```
}

lamvec <- 1:100/1000
betavec <- 0.5 + 1:130/50

ymat <- xmat <- matrix(NA, 100, 130)
for(i in 1:100) for(j in 1:130) xmat[i,j] <- WeibLL(x=t, d=d,
                                                   lam=lamvec[i], beta=betavec[j])
for(i in 1:100) for(j in 1:130) {
ymat[i,j] <- exp(-(lamvec[i]*15)^betavec[j] )
}

contour(lamvec, betavec, xmat, xlab="lam", ylab="beta", main="LogLik")
```



We can see that the log-likelihood function is its highest around $(0.05, 1.5)$, which agrees with the MLE values that we found above. Now, we want to find a 95% confidence interval for the survival probability given a value of t:

$$P\left(X > t\right) = g\left(\lambda, \beta\right) = e^{-(\lambda t)^{\beta}}$$

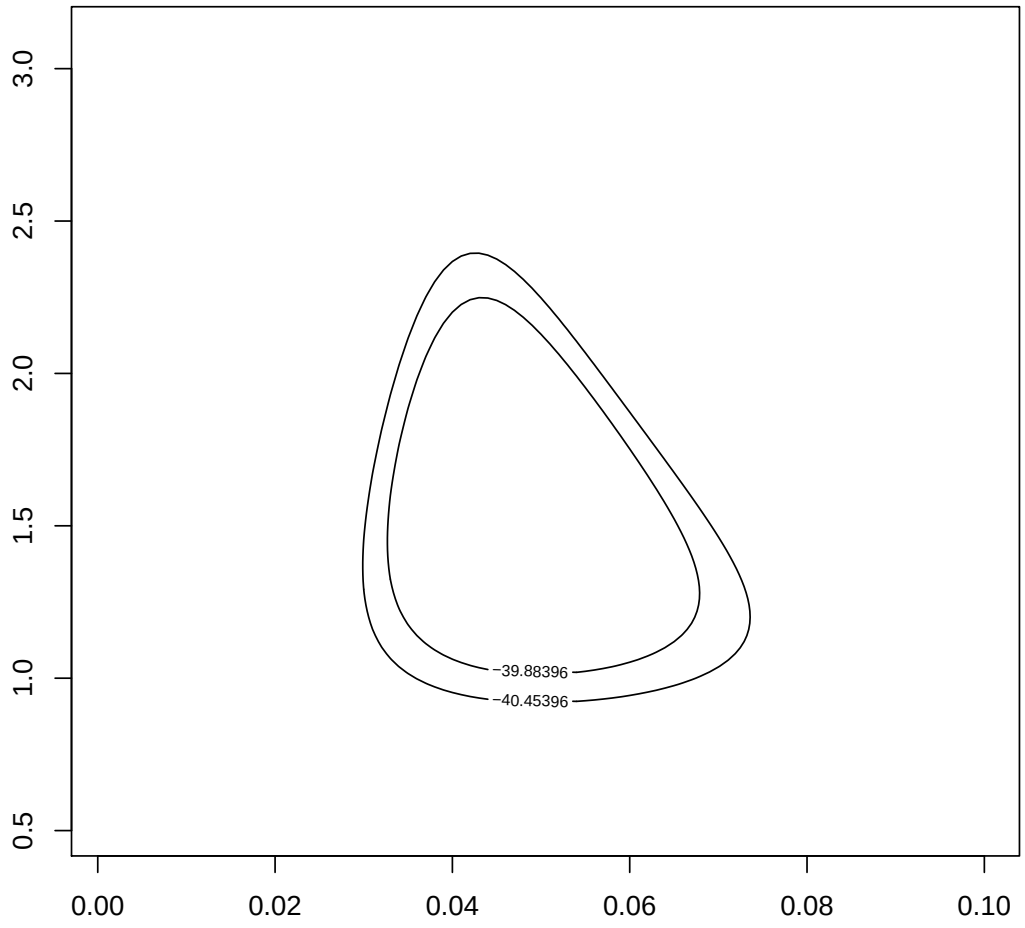
We need to find the region of our -2 log likelihood function which is within 3.84 of our MLE. Let's define this region to be U. Once we have the region U, we can find the upper and lower bounds of our confidence interval in the following way:

$$\text{upper} = \max_{\lambda,\beta \in U} \left\{\exp[-\left(\lambda t\right)]\right\}$$

$$\text{lower} = \min_{\lambda,\beta \in U} \left\{\exp[-\left(\lambda t\right)]\right\}$$

We can use R to find the region U by using the following code: (95% and 90%)

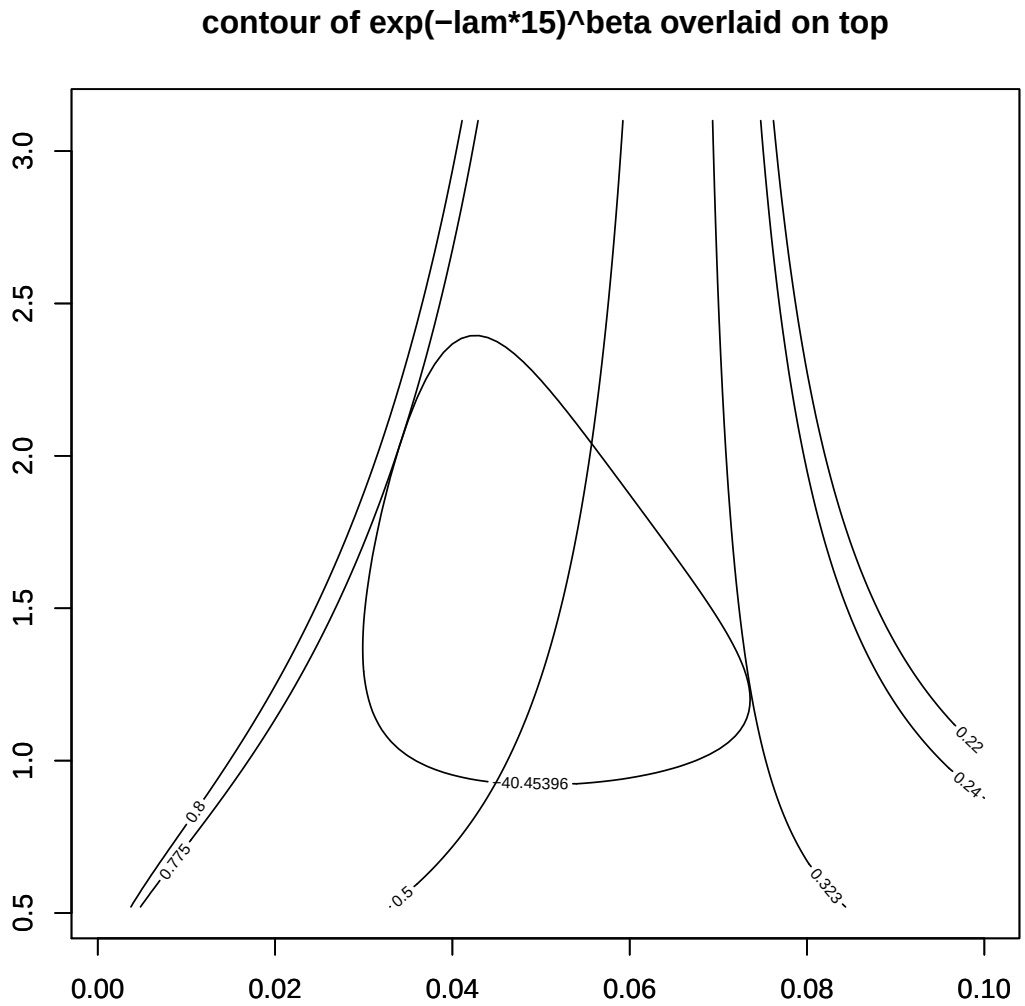
```r
lamvec <- 1:100/1000
betavec <- 0.5 + 1:130/50

ymat <- xmat <- matrix(NA, 100, 130)
for(i in 1:100) for(j in 1:130) xmat[i,j] <- WeibLL(x=t, d=d,
                                                   lam=lamvec[i], beta=betavec[j])
for(i in 1:100) for(j in 1:130) ymat[i,j] <- exp(-(lamvec[i]*15)^betavec[j])

contour(lamvec, betavec, xmat, level=c(Maxi - 3.84/2, Maxi - 2.70/2))
```



A note of caution: we used the percentile of Chi-square with df=1, not df=2 (why?) Since we don't have an explicit formula or function for this region, we will have to find the confidence interval for it algorithmically. So, we'll have to search the region U for a

maximum and minimum value of the survival probability. An example of this process is shown below:

```
contour(lamvec, betavec, xmat, level=c(Maxi - 3.84/2))
par(new=TRUE)
contour(lamvec, betavec, ymat, level=c(0.8, 0.775, 0.5, 0.323, 0.24, 0.22),
            main="contour of exp(-lam*15)^beta overlaid on top")
```



We can see that the smallest probability value which falls in our region U is somewhere around 0.323. We could do the same for the upper bound, and give an interval of approximately [0.323, 0.775] but it would also imprecise. Instead, we should create a more general search method for finding maximum and minimum values which would work in higher dimensions as well. One possible method is the "Blind Man Climbing a Mounting" approach.

The steps for this algorithm are:

1. Choose a radius value.
2. Starting with the MLE, calculate the likelihood values at every (or a select number) point on that radius.
3. Find the largest value and take that point as your new center.
4. From your new center point, calculate the likelihood values at every (or a select number) point on that radius.
5. Continue this process until you have a point which lies outside your region U.
6. Once you find one point outside of U, return to your previous point and reduce your radius value and recalculate you likelihood values.
7. Repeat until the radius value gets sufficiently small.

Likelihood Ratio Test Method vs. the Wald Method in the Weibull Case:
Both require work to get a confidence interval, but the Wald requires lots of calculations by hand while the LRT method only requires computation time once your search algorithm is written. So, we obviously prefer the LRT method.

Here is the Log likelihood ratio function that we will use for optimization in this case we have the maximum for this problem set as a magic number (find this number from previous computer output) .

```
WLLRfn <- function(para, dataMat, Maxi=-38.53396) {
  mytemp <- 2*( Maxi-WeibLL(x=dataMat[1,], d=dataMat[2,],
                lam=para[1], beta=para[2]))
  list("-2LLR"=mytemp)
}
```

This is a function created that will uses the steps above to find the upper value of the confidence interval.

```
FindU2 <- function(mle, ConfInt, LogLikfn, Pfun, dataMat, level = 3.84) {
    temp0 <- LogLikfn(para = mle, dataMat)
    PfunUold <- Pfun(mle[1], mle[2])
    stepsize <- ConfInt/3
    b1vec <- mle[1] + stepsize[1] * (1:10 - 5.5)
    b2vec <- mle[2] + stepsize[2] * (1:10 - 5.5)
    temp <- matrix(NA, nrow = 4, ncol = 100)
    temp[1, ] <- rep(b1vec, each = 10)
    temp[2, ] <- rep(b2vec, times = 10)
    for (i in 1:10) for (j in 1:10) {
        ind <- (i - 1) * 10 + j
        temp[4, ind] <- LogLikfn(para = c(b1vec[i], b2vec[j]),
```

```
                                    dataMat)$"-2LLR"
    }
    subsetInd <- (temp[4, ] < level)
    ParaTrials <- temp[, subsetInd]
    TrialValues <- Pfun(ParaTrials[1, ], ParaTrials[2, ])
    maxInd <- which.max(TrialValues)
    PfunU <- TrialValues[maxInd]
    maxParaOLD <- maxPara <- ParaTrials[, maxInd]
    print(c(PfunU, maxPara))
    for (N in 1:25) {
        if (sum(subsetInd) < 20)
             stepsize <- stepsize/2
        if (PfunU <= PfunUold) {
            maxpara <- maxParaOLD
            stepsize <- stepsize/4
        }
        b1vec <- maxPara[1] + stepsize[1] * (1:10 - 5.5)
        b2vec <- maxPara[2] + stepsize[2] * (1:10 - 5.5)
        temp[1, ] <- rep(b1vec, each = 10)
        temp[2, ] <- rep(b2vec, times = 10)
        for (i in 1:10) for (j in 1:10) {
            ind <- (i - 1) * 10 + j
            temp[4, ind] <- LogLikfn(para = c(b1vec[i], b2vec[j]),
                                             dataMat)$"-2LLR"
        }
        subsetInd <- (temp[4, ] < level)
        ParaTrials <- temp[, subsetInd]
        TrialValues <- Pfun(ParaTrials[1, ], ParaTrials[2, ])
        maxInd <- which.max(TrialValues)
        PfunUold <- PfunU
        PfunU <- TrialValues[maxInd]
        maxPara <- ParaTrials[, maxInd]
        print(c(PfunU, maxPara))
    }
    list(Upper = PfunU, maxParameterNloglik = maxPara)
}

FindU2(mle=c(0.04725676,1.568467), ConfInt=c(0.005, 0.1), LogLikfn=WLLRfn,
       Pfun=function(b1, b2){exp(-(b1*15)^b2)}, dataMat=rbind(t, d))

## [1] 0.66275712 0.03975676 1.71846700         NA 0.79339007
## [1] 0.76806668 0.03225676 1.83513367         NA 3.79568527
## [1] 0.77040069 0.03309009 1.91846700         NA 3.73881799
## [1] 0.77212023 0.03392343 2.00180033         NA 3.73192277
## [1] 0.77325549 0.03475676 2.08513367         NA 3.77981888
```

```
## [1] 0.77212023 0.03392343 2.00180033         NA 3.73192277
## [1] 0.77467381 0.03413176 2.03930033         NA 3.82877426
## [1] 0.77435928 0.03392343 2.01846700         NA 3.82042130
## [1] 0.77471912 0.03397551 2.02575867         NA 3.83344927
## [1] 0.77479815 0.03402759 2.03096700         NA 3.83548205
## [1] 0.77487495 0.03407968 2.03617533         NA 3.83773170
## [1] 0.77479815 0.03402759 2.03096700         NA 3.83548205
## [1] 0.77488676 0.03404061 2.03278992         NA 3.83886198
## [1] 0.77490598 0.03405364 2.03409200         NA 3.83942093
## [1] 0.77492507 0.03406666 2.03539408         NA 3.83999349
## [1] 0.77490598 0.03405364 2.03409200         NA 3.83942093
## [1] 0.77491848 0.03405038 2.03389669         NA 3.83999034
## [1] 0.77491368 0.03404712 2.03357117         NA 3.83984997
## [1] 0.77491680 0.03404631 2.03352234         NA 3.83999257
## [1] 0.7749156 0.0340455 2.0334410        NA 3.8399576
## [1] 0.77491638 0.03404529 2.03342875         NA 3.83999331
## [1] 0.77491608 0.03404509 2.03340841         NA 3.83998459
## [1] 0.77491642 0.03404514 2.03341553         NA 3.83999787
## [1] 0.77491635 0.03404509 2.03341044         NA 3.83999569
## [1] 0.7749164 0.0340451 2.0334122        NA 3.8399990
## [1] 0.77491645 0.03404512 2.03341349         NA 3.83999955
## $Upper
## [1] 0.7749165
##
## $maxParameterNloglik
## [1] 0.03404512 2.03341349         NA 3.83999955
```

We see the upper bound of the 95% confidence interval is 0.7749165, this confirms our previous result read from the contour plot. Similarly we may get the lower bound (not shown here).

4 The Cox model and the partial likelihood

4.1 Introduction of the Partial Likelihood

When you have a lot of nuisance parameters and the normal likelihood equation does not work, you can use Cox's Partial Likelihood. The example we discuss below has each observation consisting of the elements (T_i, x_i, δ_i) for $i = 1, 2, \ldots, n$. Where T_i is the observed survival time, x_i is the covariate, and δ_i is the censor status, all related to the i^{th} patient. In this example the partial likelihood, denoted PL is

$$PL = \prod_{i=1}^{n} \left(\frac{e^{\beta x_i}}{\sum_{j=1}^{n} e^{\beta x_i} I[T_j \geq T_i]} \right)^{\delta_i}$$

This is in a sense related to the probability of the rank of the T_i's, not the T_i's themselves.

We have a model assumption that T_i has a hazard function given by

$$h_i(t) = h_0(t)e^{\beta x_i}$$

where $h_i(t)$ represents the hazard function of the i^{th} patient and $h_0(t)$ is the baseline hazard function (common to all patients). Here β is the parameter of interest and $h_0(t)$ is a nuisance parameter of infinite dimension.

The question we need to ask ourselves is if this baseline hazard function is going to destroy the use of the likelihood.

4.1.1 Revisiting the Hazard Function

The hazard function is of the form

$$h(t) = \frac{f(t)}{1 - F(t)}$$

The cumulative hazard function, sometimes denoted $\Lambda(t)$ is simply

$$\Lambda(t) = \int h(s)ds$$

4.1.2 Steps to Deriving the Partial Likelihood Equation

The steps to deriving the Partial Likelihood of the Cox Model are quite simple.

1. Write down the likelihood for the Cox Model. Make sure to put it in terms of the baseline hazard function, $h_0(\cdot)$, and β.
2. Profile out the baseline hazard function, $h_0(\cdot)$.
3. Produce the Partial Likelihood equation.

We will work through the above steps for the Cox Model.

1. The likelihood of the Cox Model is as follows (when there is no censoring):

$$\begin{aligned} Lik &= \prod_{i=1}^{n} f(x_i) \\ &= \prod_{i=1}^{n} \frac{f(x_i)}{1 - F(x_i)} \left[1 - F(x_i)\right] \\ &= \prod_{i=1}^{n} h(x_i)e^{-\Lambda(x_i)} \end{aligned}$$

We can move from lines 2 to 3, because we know that $e^{-\Lambda(x_i)} = 1 - F(x_i)$.

2. For observations WITH right censoring,

$$\begin{aligned} Lik &= \prod_{i=1}^{n} \left\{ f_i(t_i)dt * I\left[\delta_i = 1\right] + \left[1 - F_i(t_i)\right] I\left[\delta_i = 0\right] \right\} \\ &= \prod_{i=1}^{n} \left\{ \frac{f_i(t_i)dt * e^{-\Lambda_i(t_i)}}{1 - F_i(t_i)} I\left[\delta_i = 1\right] + e^{-\Lambda_i(t_i)} I\left[\delta_i = 0\right] \right\} \\ &= \prod_{i=1}^{n} \left[h_i(t_i)dt\right]^{\delta_i} e^{-\int_0^{t_i} h_i(s)ds} \end{aligned}$$

By using the Cox model assumption mentioned earlier, we can manipulate the likelihood equation even more.

$$Lik = \prod_{i=1}^{n} \left[h_0(t_i)e^{\beta x_i}dt\right]^{\delta_i} e^{-\int_0^{t_i} h_0(s)e^{\beta x_i}ds}$$

By taking the log of each side of the equation and letting $w_i = h_0(t_i)dt$ for $i = 1, 2, \ldots, n$

$$\log Lik(\beta, \vec{w}) = \sum_{i=1}^{n} \left\{ \delta_i \left[\beta x_i + \log(w_i)\right] - e^{\beta x_i} \int_0^{t_i} w_s \right\}$$

We know that w_i must be discrete. If it was continuous, there would be no maximum. Therefore we can change the integral to a summation.

$$\log Lik(\beta, \vec{w}) = \sum_{i=1}^{n} \left\{ \delta_i \left[\beta x_i + \log(w_i)\right] - e^{\beta x_i} \sum_{s=0}^{t_i} w_s \right\}$$

In this example w_i is considered to be a nuisance parameter (of infinite dimension) because it continues to grow as n increases. In this case the logLik still works, unlike the Neymann-Scott example. Here is a reminder of what the logPL looks like, and that it does not involve $\vec{w}$.

$$\log PL = \sum_{i=1}^{n} \delta_i \left[\beta x_i - \log \left(\sum_{j=1}^{n} e^{\beta x_i} I\left[T_j \geq T_i\right] \right) \right]$$

3. Profiling out the w_i's is based on a theorem. For any given and fixed $\beta = \beta^*$, we can find the solution to $max_{\vec{w}} \left\{\log Lik(\beta^*, \vec{w})\right\}$. The solution is

$$w_i(\beta^*) = \frac{\delta_i}{\sum_{j=1}^{n} e^{\beta x_i} I[T_j \geq T_i]}$$

The values of these $w_i(\beta^*)$'s will be the MLEs if the β^* is the MLE of β.

4. Now we can see the relationship between logLik and logPL when the solution $w_i(\beta^*)$ is plugged in.

$$\log Lik(\beta^*, w_i(\beta^*)) = \log PL(\beta^*) - \sum_{i=1}^{n} \delta_i$$

In this example we can use the partial likelihood, logPL, if we would like to do any inference such as confidence intervals and hypothesis testing just involving the β's as this equation is much easier to work with. However if we would like to perform any inferences involving the w_i's, then we would have to use the original likelihood: loglik.

4.1.3 Using R to Compute Empirical and Partial Likelihoods

The package ELYP, contains a function CoxEL. This function computes the empirical likelihood as well as the partial likelihood for the purpose of testing the parameter β and the baseline hazard feature, which we defined via lam and fun.

This function has 6 inputs: y, d, Z, beta, lam, and fun. In this case, y is equivalent to the t_i used above, d is equivalent to δ_i, Z is equivalent to x_i, and beta is equivalent to β^*. The new inputs that we have not discussed yet are lam, which is a scalar used in the constraint of baseline, and fun, a function which together with lam defines the feature of the baseline hazard. When lam=0 there is no restriction on the baseline w_i's which means that they will follow the formula $w_i(\beta^*) = \frac{\delta_i}{\sum_{j=1}^{n} e^{\beta x_i} I[T_j \geq T_i]}$. Here is an example where y, d, and Z are defined below. We will set beta equal to 1, just to run the function, and lam equal to 0. This means that it does not matter what function for fun we put in, since it will not affect the w_i's at all.

```
library(ELYP)

## Warning:  package 'ELYP' was built under R version 3.2.2
##
## Attaching package:  'ELYP'
## The following object is masked from 'package:emplik':
##
##     cumsumsurv

y <- c(3, 5.3, 6.4, 9.1, 14.1, 15.4, 18.1, 15.3, 14, 5.8, 7.3, 14.4)
x <- c(1, 1.5, 2, 3, 4, 5, 6, 5, 4, 1, 2, 4.5)
d <- c(1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0)

CoxEL(y=y, d=d, Z=x, beta=1, lam=0, fun=function(t){t})

## $d
##  [1] 1 1 1 1 1 1 1 1 0 1 1 0
##
## $Hazw
##  [1] 0.001059041 0.001062099 0.001067178 0.001070283 0.001078815
##  [6] 0.001087484 0.001111768 0.001183613 0.000000000 0.001428051
## [11] 0.001812113 0.000000000
##
## $mu
## [1] 0.1216266
##
```

```
## $logPlik
## [1] -38.94194
##
## $logEmpLik
## [1] -48.94194
```

A brief explanation of the output of this function.

- d - This is a list of the input d values, sorted by the y values.
- Hazw - These are your w_i's. Remember that $w_i = h_0(t_i)dt$ where $h_0(t_i)$ is your baseline hazard function.
- mu - This is equal to $\int f(t)h_0(t)dt$. This is the value that is pulled either up or down by your lam input, where f(t) is the fun in the input of CoxEL( ).
- logPlik - This is the value of log partial likelihood.
- logEmpLik - This is the value of the log empirical likelihood.

You will notice that $\log EmpLik = \log Plik - \sum_{i=1}^{12} d$, which matches our derivation of the partial likelihood.

Since the beta input is not the MLE of β, the Hazw output are not the MLEs of the w_i's. This is how to find the MLE of β.

```
beta<-coef(coxph(Surv(y,d)~x))
beta

##         x
## -3.729459
```

This gives the true MLE of β. This is what happens when we input this value of beta into the CoxEL function.

```
CoxEL(y=y, d=d, Z=x, beta=beta, lam=0, fun=function(t){t})

## $d
##  [1] 1 1 1 1 1 1 1 1 0 1 1 0
##
## $Hazw
##  [1] 1.890424e+01 3.461123e+01 3.972519e+01 8.568066e+02 1.692491e+03
##  [6] 6.865377e+04 1.366519e+06 2.501949e+06 0.000000e+00 6.197345e+07
## [11] 1.224941e+08 0.000000e+00
##
## $mu
## [1] 2889654017
##
## $logPlik
```

```
## [1] -5.373885
##
## $logEmpLik
## [1] -15.37389
```

You will see that the outputs changes, with the exception of the d values. These Hazw's are now the MLEs of the w_i's. It is also important to note that this function retrieves the same value for the log partial log likelihood as the unconditional likelihood given by the coxph function.

```
coxph(Surv(y,d)~x)$loglik

## [1] -18.600920  -5.373885
```

The Cox Partial Likelihood is a a good way to make inferences on the non-nuisance parameters. However if you would like to find the MLEs of the nuisance parameters, CoxEL is a good way to find those estimates.

4.2 Construct confidence interval related to β

Let us now try an example for the Cox model. This is similar to the example on page58 from the book *Modeling Survival Data* by Therneau and Grambsch. The data set is pbc.

```
library(survival)
library(emplik)
myfit <- coxph(Surv(time,status==2) ~ age+edema+log(bili)+
                                      log(albumin)+log(protime), data=pbc)
myfit

## Call:
## coxph(formula = Surv(time, status == 2) ~ age + edema + log(bili) +
##     log(albumin) + log(protime), data = pbc)
##
##
##                  coef exp(coef) se(coef)     z       p
## age           0.0396    1.0404  0.00767  5.16 2.4e-07
## edema         0.8963    2.4505  0.27141  3.30 9.6e-04
## log(bili)     0.8636    2.3716  0.08294 10.41 0.0e+00
## log(albumin) -2.5069    0.0815  0.65292 -3.84 1.2e-04
## log(protime)  2.3868   10.8791  0.76851  3.11 1.9e-03
##
## Likelihood ratio test=231  on 5 df, p=0  n= 416, number of events= 160
##    (2 observations deleted due to missingness)

myfit$loglik

## [1] -866.9573 -751.4697
```

We see the full model loglik value is -751.4697 (the max value). We then define a function that will offset the slope for log(protime) and return the -2LLR.

```
pbcloglik <- function(beta, max, pbc) {
temp <- coxph(Surv(time,status==2) ~
                age+edema+log(bili)+log(albumin)+offset(beta*log(protime)), data=pbc)
loglikratio <- 2*( max - temp$loglik[2])
list("-2LLR"=loglikratio)
}
```

Now we are ready to find the confidence interval. You find the MLE of beta for log(protime) if you read the output of coxph( ) above, instead of looking only at loglik.

```
findUL(fun=pbcloglik, MLE=2.387, max=-751.4697, pbc=pbc)
```

```
## $Low
## [1] 0.8106858
##
## $Up
## [1] 3.825281
##
## $FstepL
## [1] 1e-10
##
## $FstepU
## [1] 1e-10
##
## $Lvalue
## [1] 3.84
##
## $Uvalue
## [1] 3.84
```

So, the 95% Wilks confidence interval for beta(protime) is $[0.8106858, 3.825281]$.

4.3 Construct confidence interval related to (β, μ) by inverting empirical Likelihood

4.3.1 Construct confidence interval

This example is similar to (but more complicated than) the Weibull survival probability example of section 3. It is strongly advised that you first go through that example first. The two added twist here are (1). we need to turn the infinite dimensional parameter $\Lambda_0(t)$ into a finite dimensional parameter via the function fun: $\mu = \int f(t)\mathrm{d}\Lambda_0(t)$. (2). Produce the contour plot of log lik for (β, μ) is hard, so we instead produce a contour plot of (β, λ) where λ is a monotone, one-to-one function of μ: $\mu = K(\lambda)$; where $K(0) = \mu_{mle}$ In other words, we setup the grid points in terms of β and λ before create the contour plot using R.

First, let us look at the null hypothesis below:

$$H_0: \begin{array}{c} \text{300 days survival probability for a patient with covariate } Z = c; \\ i.e. \qquad e^{-\Lambda_0(300)e^{\beta Z}} = c \end{array}.$$

Note that even though we have two unknown things, $\Lambda_0(300)$ and β in the hypothesis, the number of parameter under testing is only equal to one. And we could build up a confidence interval for the term $e^{-\Lambda_0(300)e^{\beta Z}}$ by inverting Likelihood Ratio Test (LRT).
Consider the easier 2-parameter hypothesis $H'_{00}s$, where

$$H_{00}: \begin{array}{c} \mu = \Lambda_0(300) = \mu^* \\ \beta = \beta^* \end{array}$$

But here we have two parameters. And we could do the same thing, that is to invert LRT to get a joint confidence set (a region) for (β, μ). Once we have the confidence set (a region), then the confidence interval for $e^{-\Lambda_0(300)e^{\beta Z}}$ is just the range that $e^{-\Lambda_0(300)e^{\beta Z}}$ can take on that region.

Let us focus on one specific H_{00}

$$H_{00}: \begin{array}{c} \mu = \Lambda_0(300) = 0.2 \\ \beta = 0.5 \end{array}$$

For the data set "smallcell" in R, we can try to compute the true likelihood of survival for the above H_{00} using CoxEL function in "ELYP" package.

```
library("ELYP")
data(smallcell)
myy <- smallcell$survival
myd <- smallcell$indicator
myZ <- smallcell$arm
```

But we need to define $f(x)$ such that $\int f(t)d\Lambda_0(t) = \Lambda_0(300)$. And we could easily achieve this by defining $f(t) = I[t \leq 300]$.

```
myfun <- function(t){as.numeric(t<=300)}
```

Now let's compute the likelihood

```
CoxEL(y=myy,d=myd,Z=myZ,beta=0.5,lam=0,fun=myfun)

## $d
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
##  [71] 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0 0 0 0 0
## [106] 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0
##
## $Hazw
##   [1] 0.006278467 0.006344138 0.006384643 0.006425668 0.006494472
##   [6] 0.006564764 0.006636595 0.006710016 0.006785079 0.006861840
##  [11] 0.006909250 0.006988863 0.007070333 0.007120678 0.007205268
##  [16] 0.007291892 0.007380624 0.007435503 0.007527786 0.007622389
##  [21] 0.007680937 0.007740390 0.007840448 0.007943126 0.008006725
##  [26] 0.008113834 0.008223848 0.008292041 0.008361374 0.008478251
##  [31] 0.008598443 0.008722091 0.008849347 0.008980372 0.009115335
##  [36] 0.009199188 0.009340860 0.009486964 0.009637711 0.009793326
##  [41] 0.009954048 0.010120134 0.010291857 0.010469508 0.010653399
##  [46] 0.010843866 0.010962745 0.011164538 0.011373900 0.011504754
##  [51] 0.011638654 0.011775707 0.011916027 0.012059731 0.012206943
##  [56] 0.012357794 0.012512420 0.012670965 0.012941320 0.013110994
##  [61] 0.013400668 0.013703431 0.013893824 0.014089582 0.014290935
##  [66] 0.014498127 0.014711415 0.014931072 0.015307910 0.015545885
##  [71] 0.015791376 0.016213504 0.016480714 0.016756879 0.017042458
##  [76] 0.017337939 0.017848135 0.018389269 0.018964242 0.019330836
##  [81] 0.000000000 0.020374027 0.020797761 0.021239495 0.022010251
##  [86] 0.022839052 0.000000000 0.024309646 0.024915329 0.000000000
##  [91] 0.000000000 0.027902960 0.028703884 0.029552145 0.000000000
##  [96] 0.031408523 0.000000000 0.034258577 0.035473861 0.000000000
## [101] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [106] 0.050270638 0.000000000 0.055889878 0.059198473 0.065601274
## [111] 0.073557076 0.000000000 0.000000000 0.000000000 0.000000000
## [116] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [121] 0.000000000
##
## $mu
## [1] 0.1772499
##
## $logPlik
## [1] -402.907
##
## $logEmpLik
## [1] -500.907
```

Notice that in the output, log likelihood is -500.907, but μ is not equal to 0.2, which is the value of $\Lambda_0(300)$ in H_{00}. Hence, we need to find a lam value to make $\mu = \Lambda_0(300) = 0.2$. After several trials, we find out that with $lam = -15.95$, we could have μ approximately 0.2, as shown below.

```
CoxEL(y=myy,d=myd,Z=myZ,beta=0.5,lam=-15.95,fun=myfun)

## $d
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0 0 0 0 0
## [106] 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0
##
## $Hazw
##   [1] 0.006977171 0.007058367 0.007108541 0.007159435 0.007244953
##   [6] 0.007332540 0.007422270 0.007514224 0.007608484 0.007705139
##  [11] 0.007764970 0.007865668 0.007969012 0.008033028 0.008140847
##  [16] 0.008251600 0.008365408 0.008435978 0.008554966 0.008677357
##  [21] 0.008753313 0.008830610 0.008961076 0.009095456 0.009178942
##  [26] 0.008113834 0.008223848 0.008292041 0.008361374 0.008478251
##  [31] 0.008598443 0.008722091 0.008849347 0.008980372 0.009115335
##  [36] 0.009199188 0.009340860 0.009486964 0.009637711 0.009793326
##  [41] 0.009954048 0.010120134 0.010291857 0.010469508 0.010653399
##  [46] 0.010843866 0.010962745 0.011164538 0.011373900 0.011504754
##  [51] 0.011638654 0.011775707 0.011916027 0.012059731 0.012206943
##  [56] 0.012357794 0.012512420 0.012670965 0.012941320 0.013110994
##  [61] 0.013400668 0.013703431 0.013893824 0.014089582 0.014290935
##  [66] 0.014498127 0.014711415 0.014931072 0.015307910 0.015545885
##  [71] 0.015791376 0.016213504 0.016480714 0.016756879 0.017042458
##  [76] 0.017337939 0.017848135 0.018389269 0.018964242 0.019330836
##  [81] 0.000000000 0.020374027 0.020797761 0.021239495 0.022010251
##  [86] 0.022839052 0.000000000 0.024309646 0.024915329 0.000000000
##  [91] 0.000000000 0.027902960 0.028703884 0.029552145 0.000000000
##  [96] 0.031408523 0.000000000 0.034258577 0.035473861 0.000000000
## [101] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [106] 0.050270638 0.000000000 0.055889878 0.059198473 0.065601274
## [111] 0.073557076 0.000000000 0.000000000 0.000000000 0.000000000
## [116] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [121] 0.000000000
##
## $mu
## [1] 0.2000094
##
## $logPlik
## [1] -402.907
##
## $logEmpLik
## [1] -501.0959
```

Hence the $\log Lik_0 = \log Lik(\beta = 0.5, \mu = 0.2) = -501.0959$.

The relation between lam and μ is such that lam=0 corresponds to $\mu = \mu_{mle}$, and it is a monotone decreasing function.

Recall the process of inverting LRT to get confidence region/interval for the parameters. We reject H_{00} if $\frac{Lik_0}{Lik_A}$ is smaller than some number. That is, $-2[\log Lik_0 - \log Lik_A] >$ some number. And since $\log Lik_A = \log Lik(MLE)$, all we need to do is to plug in $\beta = \hat{\beta}_{Cox}$ and set $lam = 0$, as shown below.

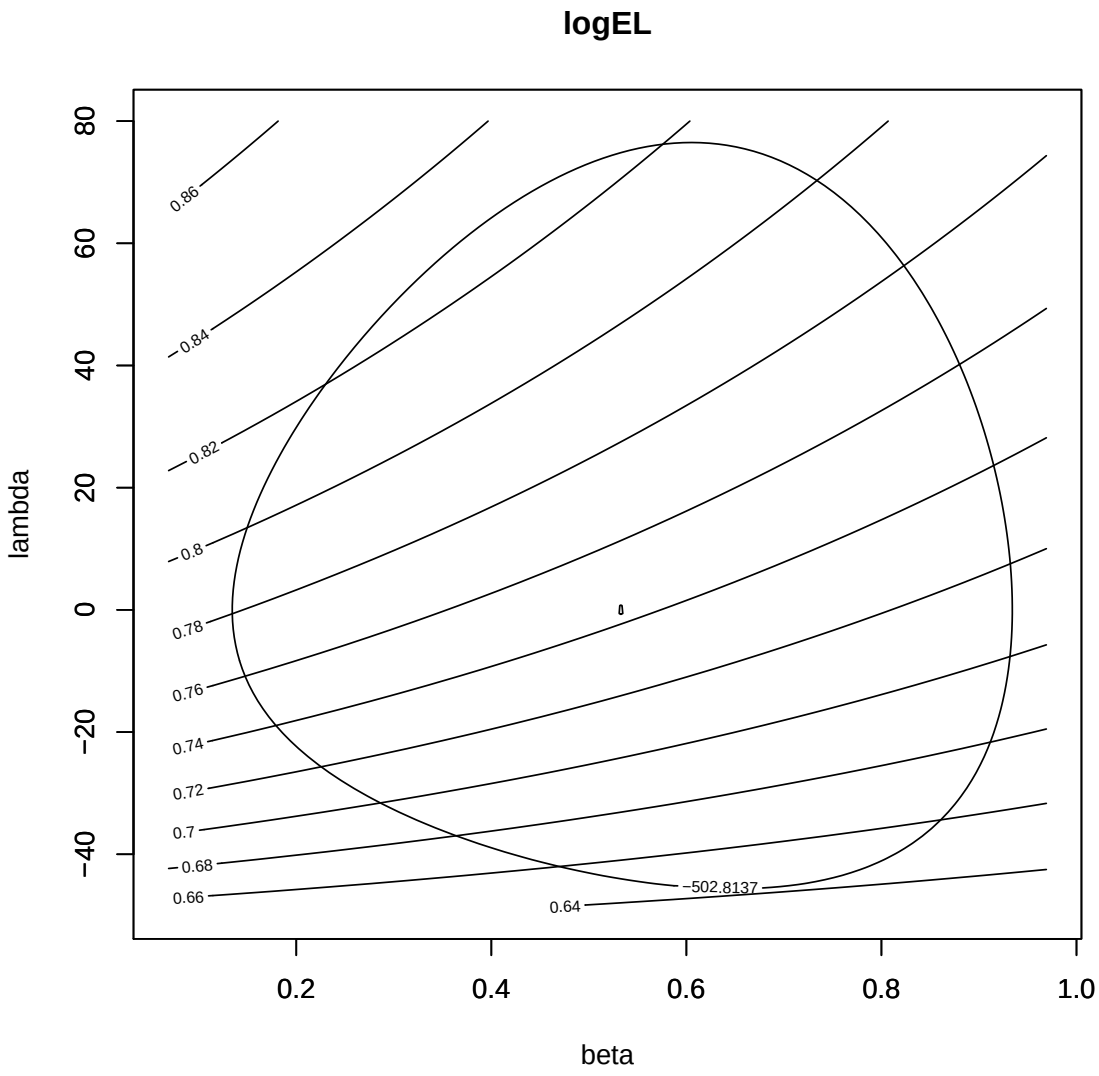
```
library(survival)
coxph(Surv(myy,myd)~myZ)$coef

##       myZ
## 0.5337653

#from here we read the beta(MLE)=0.5337653
CoxEL(y=myy,d=myd,Z=myZ,beta=0.5337653,lam=0,fun=myfun)

## $d
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0 0 0 0 0
## [106] 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0
##
## $Hazw
##   [1] 0.006149489 0.006214662 0.006253525 0.006292878 0.006361143
##   [6] 0.006430905 0.006502214 0.006575122 0.006649684 0.006725956
##  [11] 0.006771501 0.006850610 0.006931589 0.006979971 0.007064056
##  [16] 0.007150192 0.007238454 0.007291231 0.007383032 0.007477174
##  [21] 0.007533504 0.007590688 0.007690236 0.007792430 0.007853629
##  [26] 0.007960241 0.008069788 0.008135439 0.008202167 0.008318523
##  [31] 0.008438227 0.008561426 0.008688276 0.008818942 0.008953598
##  [36] 0.009034489 0.009175860 0.009321726 0.009472305 0.009627828
##  [41] 0.009788544 0.009954716 0.010126628 0.010304581 0.010488901
##  [46] 0.010679934 0.010795227 0.010997689 0.011207891 0.011334932
##  [51] 0.011464885 0.011597854 0.011733942 0.011873262 0.012015931
##  [56] 0.012162069 0.012311806 0.012465276 0.012736013 0.012900311
##  [61] 0.013190495 0.013494033 0.013678613 0.013868312 0.014063347
##  [66] 0.014263946 0.014470350 0.014682816 0.015059904 0.015290173
##  [71] 0.015527592 0.015949944 0.016208468 0.016475511 0.016751501
##  [76] 0.017036894 0.017546691 0.018087938 0.018663639 0.019018595
##  [81] 0.000000000 0.020050214 0.020460451 0.020887825 0.021659350
##  [86] 0.022490056 0.000000000 0.023947074 0.024534606 0.000000000
##  [91] 0.000000000 0.027511775 0.028290086 0.029113715 0.000000000
##  [96] 0.030913743 0.000000000 0.033735083 0.034912872 0.000000000
## [101] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [106] 0.049426533 0.000000000 0.054848473 0.058031407 0.064405158
```

```
## [111] 0.072351752 0.000000000 0.000000000 0.000000000 0.000000000
## [116] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [121] 0.000000000
##
## $mu
## [1] 0.1737539
##
## $logPlik
## [1] -402.8937
##
## $logEmpLik
## [1] -500.8937
```

Then we have $\log Lik_A = -500.8937$.

Then, $-2[\log Lik_0 - \log Lik_A] = 2[\log Lik_A - \log Lik_0] = 2[-500.8937 + 501.0959] = 0.4044$. And we've learned the that the approximate distribution of $-2[\log Lik_0 - \log Lik_A]$ is χ^2 with $df = 2$ under H_{00}. However, our final use of the confidence region is to produce a confidence interval for one parameter. Therefore, we use 3.84 as a cut off point to result a 95% confidence interval. We can calculate the p-value and decide whether to reject based on some confidence. In any case 0.4044 (not larger than 3.84) will not lead to rejection. If reject, we will include this (β^*, μ^*) is outside the confidence set. We build a contour plot for (β,λ) instead, since it is a lot easier than those for (β,μ).

```
bvec <- 1:100/110 + 0.06
lvec <- (1:100)*1.3 - 50

myEtafun <- function(beta,theta){exp(- theta*exp(beta))}
LLout1 <- LLout2 <- LLout3 <- matrix(NA, 100,100)

for(i in 1:100) for(j in 1:100) {
temp <- CoxEL(y=myy,d=myd,Z=myZ,beta=bvec[i],lam=lvec[j],fun=myfun)
LLout1[i,j] <- temp$logEmpLik
LLout2[i,j] <- temp$mu
LLout3[i,j] <- myEtafun(beta=bvec[i], theta=temp$mu)  }

contour(bvec, lvec, z=LLout1, level=c(-500.8937 - 3.84/2, -500.8941),
        xlab="beta", ylab="lambda", main="logEL")
par(new=TRUE)
contour(bvec, lvec, z=LLout3)
```



Then we need to find the range of the values that $e^{-\mu e^{\beta Z}} = e^{-\Lambda_0(300)e^{\beta Z}}$ can take (here $Z = 1$) on that region. And that will be our confidence interval for $e^{-\Lambda_0(300)e^{\beta}}$.

Finally search for $e^{-\Lambda_0(300)e^{\beta}}$ over the region of (β,λ) for maximum and minimum to form the confidence interval. We see the approximate MLE of $e^{-\Lambda_0(300)e^{\beta}}$ is 0.742, the 95% confidence interval for $e^{-\Lambda_0(300)e^{\beta}}$ is approximately [0.64, 0.83].

Without plot all the contour plots, we may search for the lower and upper confidence limit of the confidence interval using the function CoxFindU2( ) and CoxFindL2( ). From the output we can read the 95% confidence interval is [0.6422, 0.8278].

```
CoxFindL2(BetaMLE=0.5337653, StepSize=c(0.04,1.6),
          Hfun=myfun, Efun=myEtafun, y=myy, d=myd, Z=myZ)

## [1] -502.8137
```

```
## [1]    0.7171024    0.7137653   -7.2000000    0.1628753 -501.3128952
## [1]    0.6926673    0.8937653  -14.4000000    0.1502283 -502.5377573
## [1]    0.6823801    0.8737653  -21.6000000    0.1595083 -502.5011890
## [1]    0.6708913    0.8537653  -28.8000000    0.1699607 -502.5523584
## [1]    0.6579964    0.8337653  -36.0000000    0.1818250 -502.7137826
## [1]    0.6461763    0.7337653  -43.2000000    0.2096505 -502.7154663
## [1]    0.6435193    0.6337653  -45.6000000    0.2338859 -502.8116300
## [1]    0.6425724    0.6787653  -45.4000000    0.2243412 -502.8026834
## [1]    0.6422423    0.6850153  -45.4500000    0.2232025 -502.8132423
## [1]    0.6422502    0.6887653  -45.4000000    0.2223609 -502.8127701
## [1]    0.6422423    0.6850153  -45.4500000    0.2232025 -502.8132423
## [1]    0.6422502    0.6887653  -45.4000000    0.2223609 -502.8127701
## [1]    0.6422423    0.6850153  -45.4500000    0.2232025 -502.8132423
## [1]    0.6422502    0.6887653  -45.4000000    0.2223609 -502.8127701
## [1]    0.6422423    0.6850153  -45.4500000    0.2232025 -502.8132423
## [1]    0.6422502    0.6887653  -45.4000000    0.2223609 -502.8127701
## [1]    0.6422423    0.6850153  -45.4500000    0.2232025 -502.8132423
## [1]    0.6422502    0.6887653  -45.4000000    0.2223609 -502.8127701
## [1]    0.6422423    0.6850153  -45.4500000    0.2232025 -502.8132423
## [1]    0.6422502    0.6887653  -45.4000000    0.2223609 -502.8127701
## [1]    0.6422423    0.6850153  -45.4500000    0.2232025 -502.8132423
## [1]    0.6422502    0.6887653  -45.4000000    0.2223609 -502.8127701
## [1]    0.6422423    0.6850153  -45.4500000    0.2232025 -502.8132423
## [1]    0.6422502    0.6887653  -45.4000000    0.2223609 -502.8127701
## [1]    0.6422423    0.6850153  -45.4500000    0.2232025 -502.8132423
## [1]    0.6422502    0.6887653  -45.4000000    0.2223609 -502.8127701
## [1]    0.6422423    0.6850153  -45.4500000    0.2232025 -502.8132423
## [1]    0.6422502    0.6887653  -45.4000000    0.2223609 -502.8127701
## [1]    0.6422423    0.6850153  -45.4500000    0.2232025 -502.8132423
## [1]    0.6422502    0.6887653  -45.4000000    0.2223609 -502.8127701
## [1]    0.6422423    0.6850153  -45.4500000    0.2232025 -502.8132423
## $Lower
## [1] 0.6422423
##
## $minParameterNloglik
## [1]    0.6850153  -45.4500000    0.2232025 -502.8132423
```

```
CoxFindU2(BetaMLE=0.5337653, StepSize=c(0.04,1.6),
         Hfun=myfun, Efun=myEtafun, y=myy, d=myd, Z=myZ)
```

```
## [1] -502.8137
## [1]    0.7711578    0.3537653    7.2000000    0.1824336 -501.3189740
## [1]    0.7989250    0.1737653   14.4000000    0.1886811 -502.6135875
## [1]    0.8062829    0.1937653   21.6000000    0.1773921 -502.6142243
## [1]    0.8128260    0.2137653   28.8000000    0.1673527 -502.6632129
## [1]    0.8186729    0.2337653   36.0000000    0.1583655 -502.7499426
```

```
## [1]    0.8223675    0.2537653   41.6000000    0.1517360 -502.8043126
## [1]    0.8242512    0.2737653   45.6000000    0.1469914 -502.8122076
## [1]    0.8249406    0.3337653   52.8000000    0.1378325 -502.7036415
## [1]    0.8264397    0.3537653   56.8000000    0.1338287 -502.7498178
## [1]    0.8278185    0.3737653   60.8000000    0.1300316 -502.8053010
## [1]    0.8278155    0.3937653   63.2000000    0.1274592 -502.8034289
## [1]    0.8278214    0.3787653   61.4000000    0.1293807 -502.8040613
## [1]    0.8278218    0.3837653   62.0000000    0.1287350 -502.8033368
## [1]    0.8278214    0.3787653   61.4000000    0.1293807 -502.8040613
## [1]    0.8278218    0.3837653   62.0000000    0.1287350 -502.8033368
## [1]    0.8278214    0.3787653   61.4000000    0.1293807 -502.8040613
## [1]    0.8278218    0.3837653   62.0000000    0.1287350 -502.8033368
## [1]    0.8278214    0.3787653   61.4000000    0.1293807 -502.8040613
## [1]    0.8278218    0.3837653   62.0000000    0.1287350 -502.8033368
## [1]    0.8278214    0.3787653   61.4000000    0.1293807 -502.8040613
## [1]    0.8278218    0.3837653   62.0000000    0.1287350 -502.8033368
## [1]    0.8278214    0.3787653   61.4000000    0.1293807 -502.8040613
## [1]    0.8278218    0.3837653   62.0000000    0.1287350 -502.8033368
## [1]    0.8278214    0.3787653   61.4000000    0.1293807 -502.8040613
## [1]    0.8278218    0.3837653   62.0000000    0.1287350 -502.8033368
## [1]    0.8278214    0.3787653   61.4000000    0.1293807 -502.8040613
## [1]    0.8278218    0.3837653   62.0000000    0.1287350 -502.8033368
## [1]    0.8278214    0.3787653   61.4000000    0.1293807 -502.8040613
## [1]    0.8278218    0.3837653   62.0000000    0.1287350 -502.8033368
## [1]    0.8278214    0.3787653   61.4000000    0.1293807 -502.8040613
## [1]    0.8278218    0.3837653   62.0000000    0.1287350 -502.8033368
## $Upper
## [1] 0.8278218
##
## $maxParameterNloglik
## [1]    0.3837653   62.0000000    0.1287350 -502.8033368
```


Reference:

Therneau, T. and Grambsch, P. (2000). *Modeling Survival Data.* Springer

Zhou, M. (2016) *Empirical Likelihood Methods in Survival Analysis* CRC Press