

# 3701\_\_HW7\_\_YOU

Tae Uk You

12/4/2018

## Problem 1.

```
gen_X = function(n, p, rho, sigma_x, mu){  
  # Generate A1,..,An realizations  
  a.list = sigma_x * sqrt(12*rho)*(runif(n=n)-0.5)  
  
  # Using the inversion method to generate the Z_ij realization through V ~ Exp(1). I am using the inversion method  
  z.list = sigma_x * sqrt(1-rho) * (-1*log(1-runif(n=n*(p-1))))-1  
  
  # Without the 1st row, let's generate the matrix X[,2:p]  
  X.mat2 = mu + matrix(rep(a.list, p-1), nrow=n, ncol=p-1) + matrix(z.list, nrow=n, ncol=p-1)  
  
  # Combine the 1st row and the matrix I just created  
  X.mat=cbind(rep(1,n), X.mat2)  
  return(X.mat)  
}
```

**Problem 2.** Consider testing the hypothesis that  $\beta_3 = \beta_4 = \beta_5 = 0$  in a linear regression model with normal errors.

(a) Find the sample size  $n$  such that the power of the 5% level test is about 90%.

First, let's write a function that generates p-values

```
get.pvals.test=function(n, beta, sigma, rho, which.zero, reps, version=1){  
  p=length(beta)  
  d=length(which.zero)  
  
  pval.list=numeric(reps)  
  aic.mat = matrix(NA, nrow=reps, ncol=2)  
  bic.mat = matrix(NA, nrow=reps, ncol=2)  
  
  for(r in 1:reps){  
    # let's use previous gen_X function to generate X matrix  
    X.mat = gen_X(n=n, p=5, rho=rho, sigma_x=1.5, mu=10)  
    y = X.mat %*% beta + rnorm(n=n, mean=0, sd=sigma)  
    beta.hat = qr.solve(crossprod(X.mat), crossprod(X.mat, y))  
  
    # Get RSSf and RSS0  
    rssf = sum((y-X.mat %*% beta.hat)^2)  
    X0 = X.mat[, -which.zero]  
    beta.hat.0 = lm.fit(x=X0, y=y)$coefficients  
    rss0 = sum((y-X0 %*% beta.hat.0)^2)
```

```

# Compute AIC and BIC full model and null model
aic.mat[r,1]=n*log(2*pi) + n*log(rssf/n) + n + 2*(p+1)
bic.mat[r,1]=n*log(2*pi) + n*log(rssf/n) + n + log(n)*(p+1)

aic.mat[r,2]=n*log(2*pi) + n*log(rss0/n) + n + 2*(p+1-d)
bic.mat[r,2]=n*log(2*pi) + n*log(rss0/n) + n + log(n)*(p+1-d)

# Get Statistic
f = ((rss0 - rssf)/d) / (rssf/(n-p))

pval.list[r]= 1-pf(f, d, n-p)
}
if(version==1){
  return(pval.list)
}else{
  cat("Est. prob that AIC prefers full over null is ",
      mean(apply(aic.mat, 1, which.min) ==1), "\n")
  cat("Est. prob that BIC prefers full over null is ",
      mean(apply(bic.mat, 1, which.min) ==1), "\n")
  cat("The F-test (at 1%) prefers full over null is ",
      mean(pval.list < 0.01), "\n")
}
}

```

Then, let's write a function that simulate estimate of the power of the test.

```

generate_power_curve = function(min_n, max_n, by, rho, reps=2500){
  set.seed(3701)
  n_seq = seq(from=min_n, to=max_n, by=by)

  power.seq=numeric(length(n_seq))
  LB_seq=numeric(length(n_seq))
  UB_seq=numeric(length(n_seq))

  for(i in 1:length(n_seq)){
    pvals = get.pvals.test(n=n_seq[i], beta=c(0.5,0.5,0,0.25,0), rho=rho, sigma=sqrt(2), which.zero=c(3
    power.seq[i] = mean(pvals < 0.05)

    bounds=binom.test(x=sum(pvals<0.05), n=reps, conf.level=0.95)$conf.int[1:2]
    LB_seq[i]=bounds[1]
    UB_seq[i]=bounds[2]
  }
  plot(n_seq, power.seq, t="l", xlab=expression(n), ylab="Power")
  lines(n_seq, LB_seq, lty=2)
  lines(n_seq, UB_seq, lty=2)
}

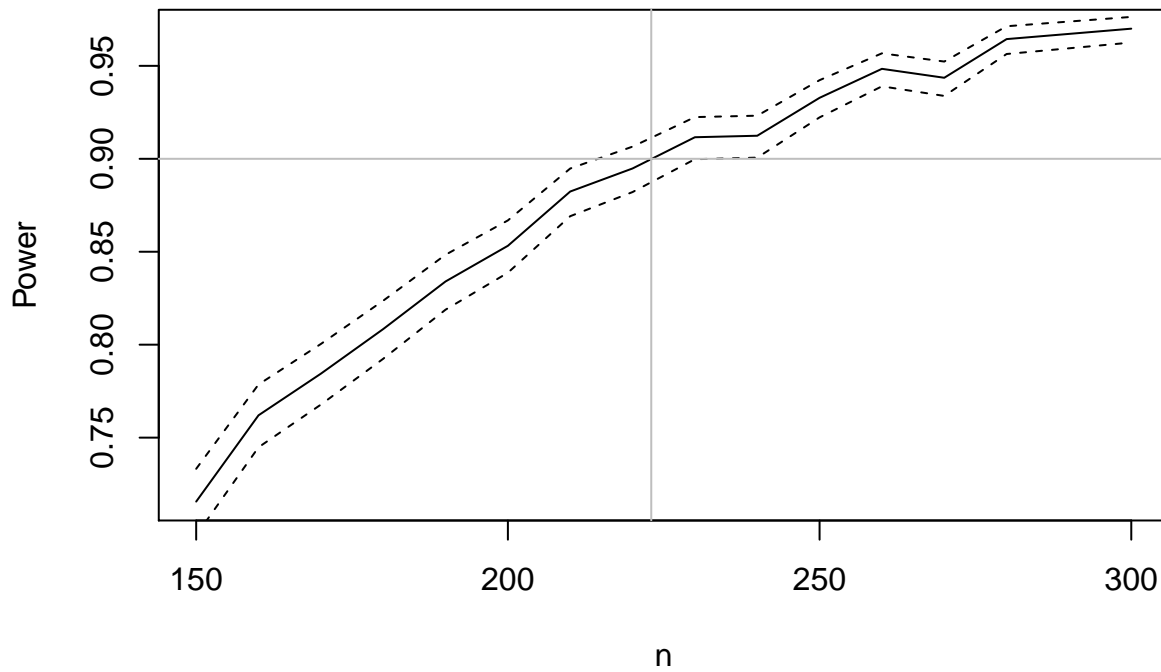
```

Now, let's compute the power curve based on the functions I made.

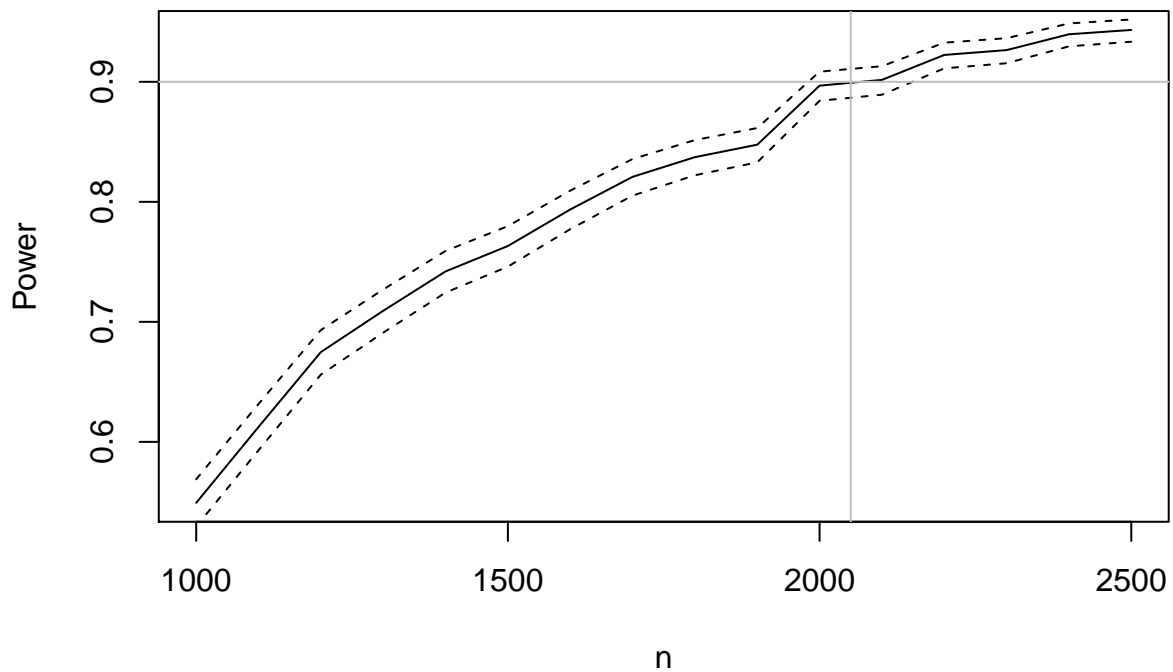
```

# when rho = 0.2
generate_power_curve(min_n=150, max_n=300, by=10, rho=0.2)
abline(h=0.90, col="grey")
abline(v=223, col="grey")

```



```
# when rho = 0.95
generate_power_curve(min_n=1000, max_n=2500, by=100, rho=0.95)
abline(h=0.90, col="grey")
abline(v=2050, col="grey")
```



Comment: The sample sizes of 223 and 2050 are needed to have power about 90% each when  $\rho = 0.3$  and  $\rho = 0.95$ .

(b) Report the following when  $(n,p) = \{(100,0.4), (400,0.95)\}$

```
set.seed(3701)
reps=2500
get.pvals.test(n=100, beta=c(0.5,0.5,0,0.25,0), rho=0.4, sigma=1.5,
               which.zero=c(3,4,5), reps=reps, version=2)
```

```
## Est. prob that AIC prefers full over null is 0.6076
## Est. prob that BIC prefers full over null is 0.1532
## The F-test (at 1%) prefers full over null is 0.2312
```

```
get.pvals.test(n=400, beta=c(0.5,0.5,0,0.25,0), rho=0.95, sigma=1.5,
               which.zero=c(3,4,5), reps=reps, version=2)
```

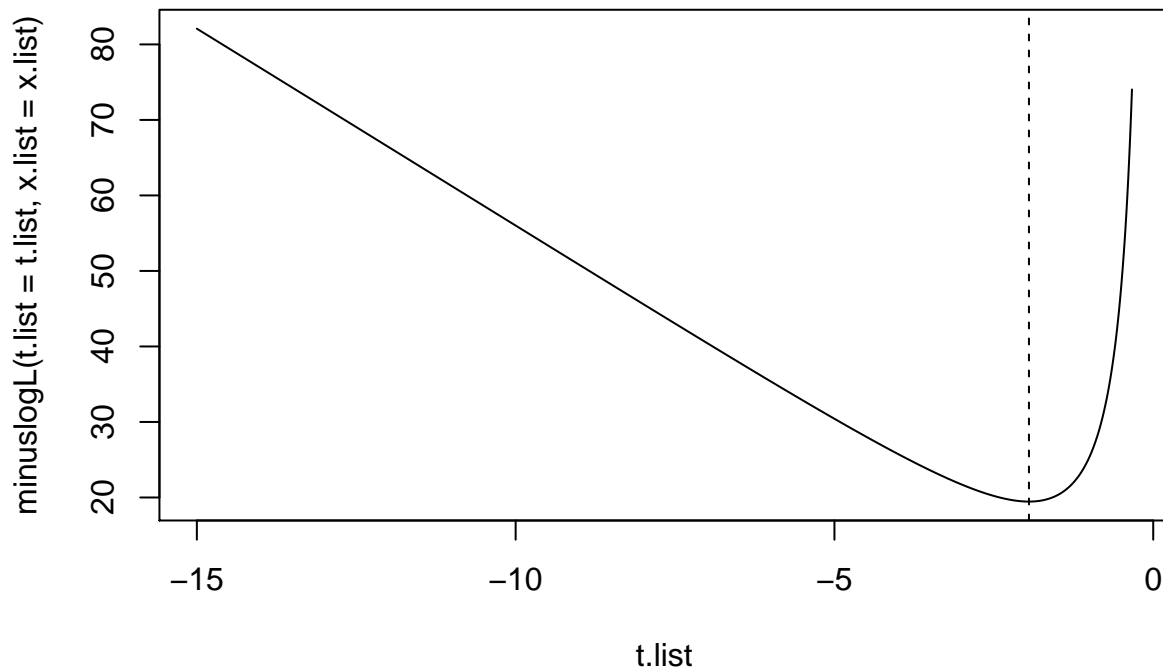
```
## Est. prob that AIC prefers full over null is 0.3548
## Est. prob that BIC prefers full over null is 0.0108
## The F-test (at 1%) prefers full over null is 0.0852
```

Comment: In this simulation, we computed the AIC, BIC, and F-test to see if the Full model performs better than null-model. For size=100 and rho=0.4, performance was better in order of AIC > F-test > BIC, and likewise for size=400 and rho=0.95, AIC performs better than F-test, and the F-test performed better than BIC.

### Problem 3.

#### Part (a)

```
set.seed(3701); n=10; theta=-2; mu=theta
x.list = rnorm(n=n, mean=theta, sd= sqrt(abs(theta)))
minuslogL = function(t.list, x.list){
  val.list = numeric(length(t.list))
  for(k in 1:length(t.list)){
    val = -sum(log(exp(-(x.list-t.list[k])^2)/(-2*t.list[k]))/sqrt(-2*pi*t.list[k]))
    val.list[k] = val
  }
  return(val.list)
}
t.list=seq(from=-1/3, to=-15, length.out=1e3)
plot(t.list, minuslogL(t.list=t.list, x.list=x.list), t="l")
abline(v=-1.949327, lty=2)
```



Comment: Yes, it seems to have a unique global maximizer since it looks like the negative log-likelihood function has a global minimizer.

## Part (b)

```
set.seed(3701)
bsearch =function(df, a, b, L=1e-7, quiet=FALSE,...){
  k=0
  while((b-a)>L){
    k=k+1
    m=(a+b)/2
    df.at.m=df(m,...)

    if(df.at.m<0){
      a=m
    }else if(df.at.m>0){
      b=m
    }else{
      a=m
      b=m
    }
    if(!quiet){
      cat("after iteration k =",k,
          "the interval is", a, b, "\n")
    }
  }
  return((a+b)/2)
}

df=function(t.list, x.list){
```

```

n=length(x.list)
val.list =sum(x.list^2)/(2*(t.list)^2) - n/2 + n/(2*(t.list))
return(val.list)
}

# Run the algorithm
bsearch(df=df, a=-15, b=-1/3, quiet=TRUE, x.list=x.list)

## [1] -1.949327

```

Comment: We employ the bisection search to compute the minimum of the negative log-likelihood function. Since the plot shows that the minimum of the function was when  $t$  is close to -1.9 with the dotted line, this dsearch algorithm confirms the result in the interval of  $t(-1/3, -15)$ .

## Problem 4

### Part (a)

From the information we learned from the lab and the class, the Probability Mass Function would be  $p_i^{y_i}(1-p_i)^{1-y_i} = e^{y_i x_i \theta - C(x_i \theta)}$ , where  $C(t) = \log(1+e^t)$ . And we plug  $p_i = \frac{1}{1+e^{-x_i \theta}}$  since  $Y_i = \text{Bern}(\frac{1}{1+e^{-x_i \theta}})$ .

Now, we know  $f_\theta(y_1, \dots, y_n) = \prod_{i=1}^n e^{y_i x_i \theta - \log(1+e^{x_i \theta})} = e^{\sum_{i=1}^n y_i x_i \theta - \log(1+e^{x_i \theta})}$

Now, when we compute the log-likelihood function from the previous term,  $l(\theta) = \sum_{i=1}^n y_i x_i \theta - \log(1+e^{x_i \theta})$ .

Then, the first derivative is as follows:  $l'(\theta) = \sum_{i=1}^n y_i x_i - \frac{x_i e^{x_i \theta}}{1+e^{x_i \theta}}$

Lastly, the second derivative is as follows:  $l''(\theta) = \sum_{i=1}^n \frac{-x_i^2 e^{x_i \theta}}{(1+e^{x_i \theta})^2}$

### Part (b)

```

# Create the function to get the vals for the negative log-likelihood function

minuslogL2 = function(x.list,y.list,theta){
  val=-sum(y.list*x.list*theta - log(1+exp(x.list*theta)))
  return(val)
}

# Create the variables and plot the likelihood
set.seed(3701); n=100; theta=1
theta = seq(from=-2, to=2, length.out=100)

x.list = runif(n=100)

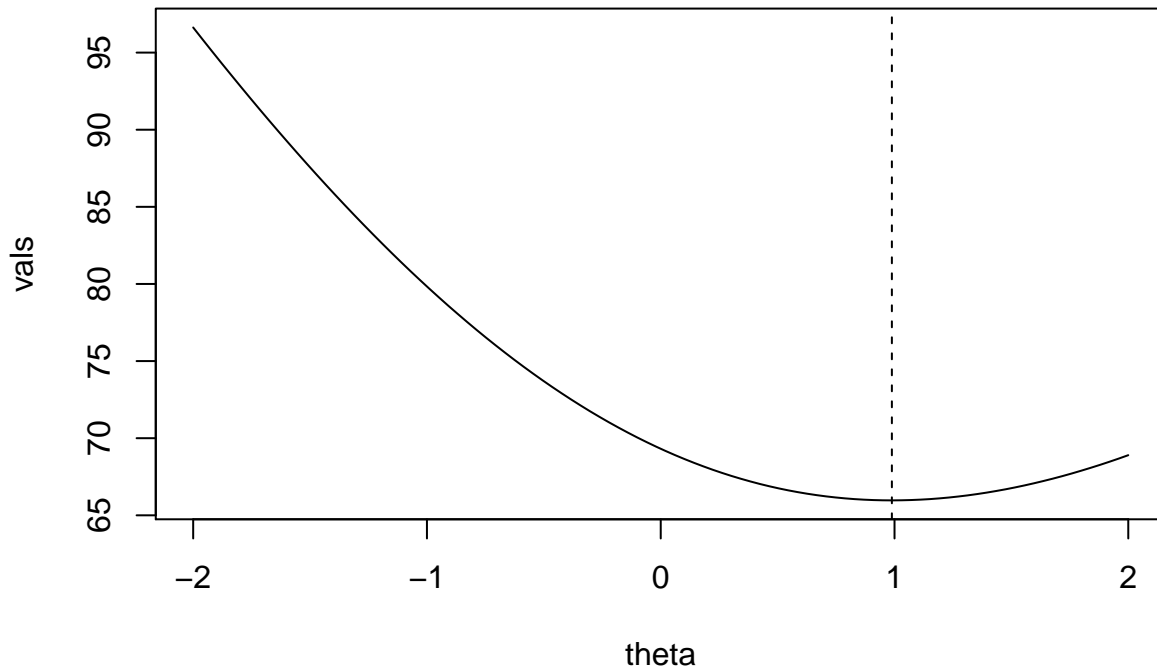
y.list = numeric(n)
for(k in 1:n){
  y.list[k] = 1* ( runif(n=1) < 1/(1+exp(-x.list[k]*1)) )
}

vals= numeric(n)
for(j in 1:n)

```

```
{
  vals[j]= minuslogL2(x.list, y.list, theta[j])
}
```

```
# Plot the negative log-likelihood
plot(theta, vals, type="l")
abline(v=0.9886896, lty=2)
```



Comment: It seems to have a unique global maximizer since it looks like the negative log-likelihood function has a global minimizer.

## Part (c)

```
set.seed(3701)
dsearch = function(f, a, b, L=1e-7, eps=(L/2.1), quiet=FALSE, ...){
  k = 0
  while( (b-a) > L){
    k = k+1
    m = (a + b)/2
    m1 = m - eps
    m2 = m + eps

    f.at.m1 = f(x.list, y.list, m1)
    f.at.m2 = f(x.list, y.list, m2)

    if(f.at.m1 < f.at.m2){
      b = m2
    } else if (f.at.m1 > f.at.m2){
      a = m1
    } else{

```

```

    a = m1
    b = m2
  }
  if(!quiet){
    cat("after iteration k = ", k, "the interval is", a, b, "\n")
  }
}
return( (a + b)/2 )
}

# Run the algorithm
set.seed(3701)
dsearch(f=minuslogL2, a=min(theta), b=max(theta), quiet=TRUE, x.list=x.list, y.list=y.list)

```

```
## [1] 0.9886896
```

Comment: We computed dichotomous search algorithm to find the minimum of the negative log-likelihood function. Since the plot shows that the minimum of the log-likelihood function was close to 1 with the dotted line, this dsearch algorithm confirms the result.