Lab: Week 7

36-350 – Statistical Computing

Week 7 – Fall 2020

Name: Kimberly Zhang

Andrew ID: kyz

You must submit **your own** lab as a PDF file on Gradescope.

Numerical Tools

Question 1

(5 points)

Notes 7A (3)

Display the value of the one real root of

$$f(x) = -2x^5 - 4x^3 + 3x - 7$$

Note that this will require you to extract one element of the output. You can hard-code this, i.e., you can simply specify by hand which vector element is to be extracted from the output. (But do use `Re()` to simplify the answer, i.e., to eliminate the imaginary part.)

```
roots= polyroot(c(-7, 3, 0, -4, 0, -2))
Re(roots[which(abs(Im(roots))<1.e-6)])
```

```
## [1] -1.161141
```

Question 2

(5 points)

Notes 7A (3)

Determine the root of the equation

$$f(x) = -2x^3 + 4x^2 - 5$$

that lies between $x = -2$ and $x = 0$. Use `uniroot()` here, and just display the `root` element of `uniroot()`'s output list.

```
f= function(x){-2*x^3 + 4*x^2-5}
uniroot(f, interval = c(-2, 0))$root
```

```
## [1] -0.9245579
```

Question 3

(5 points)

Notes 7A (3)

Note that for the "root" that you found in Q2 $f(x) = -0.0001336624$ and not zero. (At least, this is what I get; this is the value of the list element `f.root` in the output of `uniroot()`.) Identify an argument to `uniroot()` that allows you to get a more accurate estimate of the root. Set that argument to `1.e-8`. Display your new value of `f.root`... it should be $\sim 10^{-13}$.

```
uniroot(f, interval = c(-2, 0), tol = 1.e-8)$f.root
```

```
## [1] 8.881784e-16
```

Question 4

(5 points)

Notes 7A (4)

You are given a tabulated function (i.e., a sequence of (x, y) pairs) below that represents $f(x) = x^3$. Use `approx()` to estimate $f(x)$ at $x = 1.75$. Display the percentage error in your estimate. (This is calculated as 100 times the absolute value of the difference between your estimate and 1.75^3, divided by the true value, which is 1.75^3.)

```
x = seq(-3,3,by=0.5)
y = x^3
res= approx(x, y, xout = 1.75)$y
cat("Percent Error: ", abs(res - 1.75^3)*100)
```

```
## Percent Error:  32.8125
```

Question 5

(5 points)

Notes 7A (5)

Repeat Q4 using a natural spline instead. (See the documentation for `spline()`: a natural spline is not the default!) Show the percentage error. What if you use the `fmm` spline instead?

```
res.natural = spline(x, y, xout= 1.75, method = "natural")$y
res.fmm = spline(x, y, xout= 1.75, method = "fmm")$y
cat(cat("Natural Spline Percent Error: ", abs(res.natural - 1.75^3)*100), "\n")
```

```
## Natural Spline Percent Error:  1.478365
```

```
cat(cat("FMM Percent Error: ", abs(res.fmm - 1.75^3)*100), "\n")
```

```
## FMM Percent Error:  0
```

Question 6

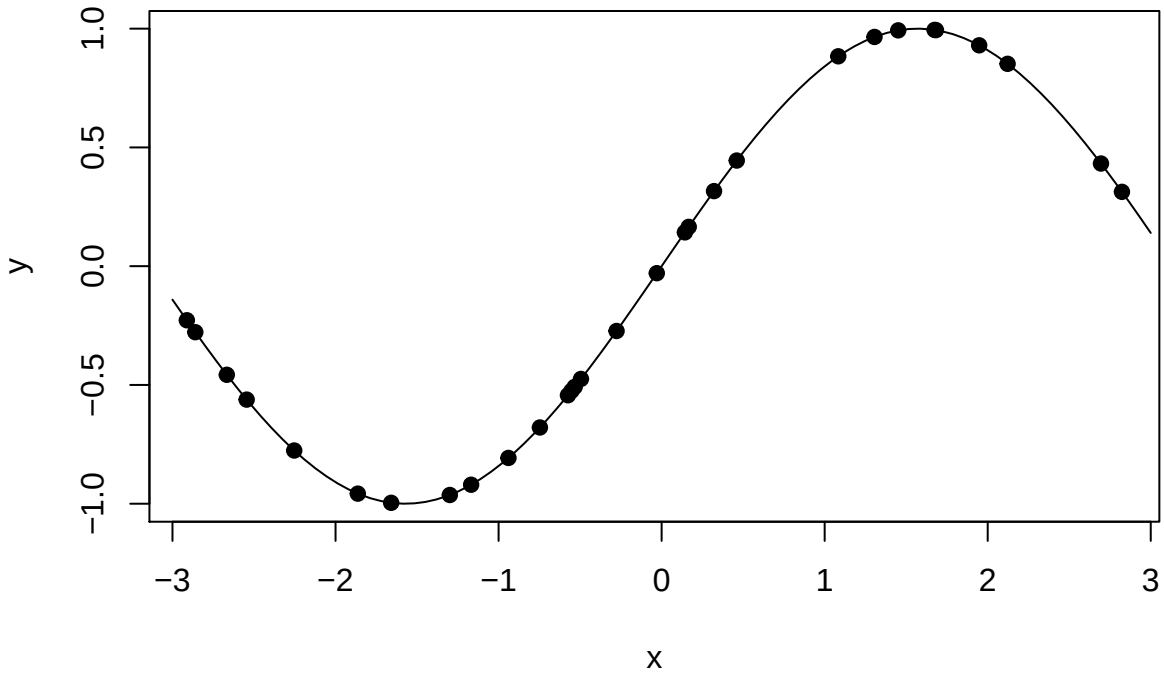
(5 points)

Notes 7A (5)

See the data below. Assume that you do not know that $f(x) = \sin(x)$. Code a spline fit to the data. Note: the extent of the spline function should be from the minimum value of x to the maximum value of x; don't extrapolate!! Use the `lines()` function to draw your spline function over the plotted points.

As a side note, splines work best when applied to *deterministic* data, i.e., to noiseless data. If you were to uncomment the noise generator in the third line below, you would find that your spline function would instantly become an unholy mess. When there is noise, you would apply, e.g., lowess smoothing or regression splines, and not `spline()` itself!

```
set.seed(404)
x = runif(30,min=-3,max=3)
y = sin(x)
plot(x,y,pch=19)
lines(spline(x, y, xmin= -3, xmax= 3))
```



Question 7

(5 points)

Notes 7A (6)

An important integral in cosmology is (with details left unspoken and proportionality constants left out):

$$f(x) = \int_0^x \frac{dz}{\sqrt{0.3(1+z)^3 + 0.7}}$$

Compute this integral for $x = 1$ and for $x = \infty$. (There's a particular reserved word in `R` for infinity... don't just use a really big number. Also, the output values are effectively meaningless here-this is just an academic exercise.) Try to adjust the output to show just the number, without the bit about absolute error. Look under "Value" in the documentation for `integrate()`.

```r
f= function(z){1/sqrt(0.3*(1+z)^3 + 0.7)}
cat(cat("When x=1: ", integrate(f, 0, 1)$value), "\n")
```

```
## When x=1:  0.7714271
```

```r
cat(cat("When x=inf: ", integrate(f, 0, Inf)$value), "\n")
```

```
## When x=inf:  3.305076
```

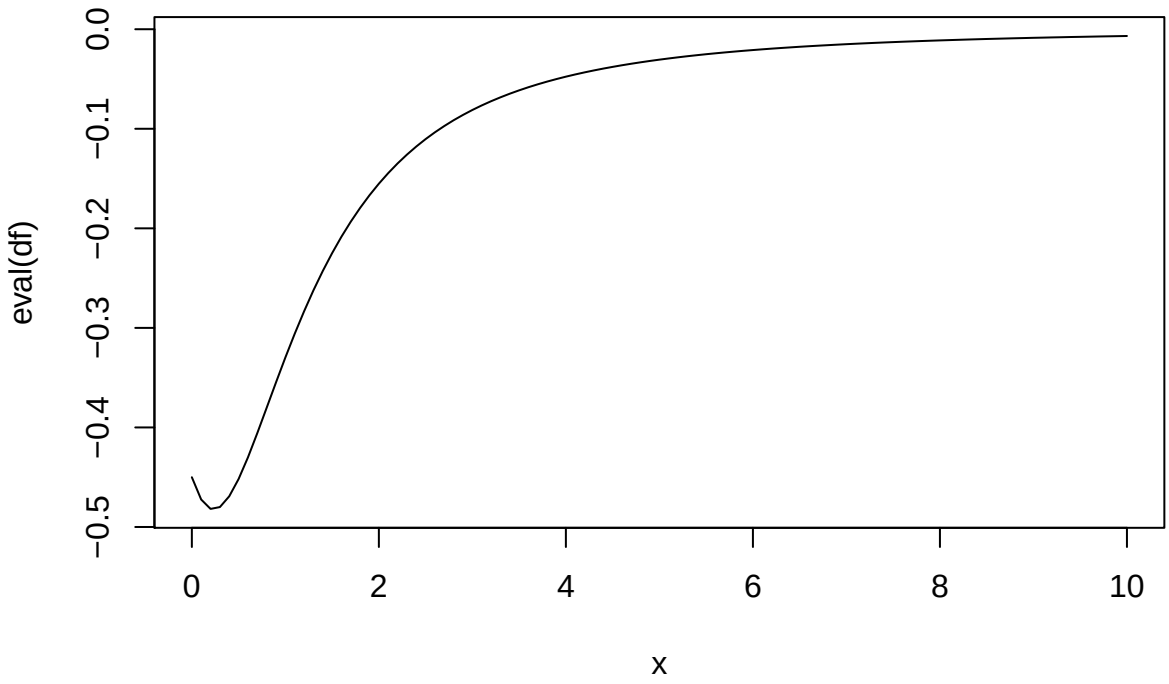
Question 8

(5 points)

Notes 7A (7)

Compute the derivative of the integrand of $f(x)$ as given in Q7 at the sequence of points $x = 0, 0.1, 0.2, \ldots, 10$. Then plot the derivative versus x, with the plot showing a line, not individual points. Use either base `R` plotting or `ggplot()`.

```r
x = seq(0, 10, by= 0.1)
f= expression(1/sqrt(0.3*(1+x)^3 + 0.7))
df = D(f, "x")
plot(x, eval(df), type= "l")
```



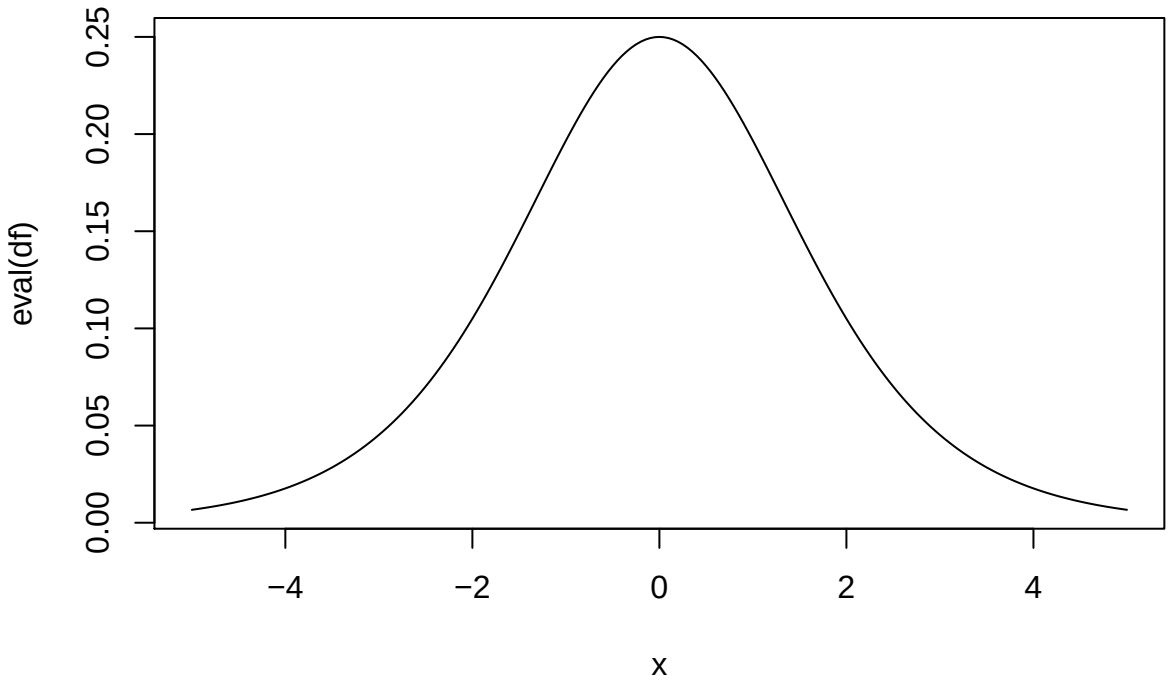
Question 9

(5 points)

Notes 7A (7)

In Q10 of Lab 2, you defined a logistic function. Here, you are to determine the derivative of your logistic function using the `D()` function, and plot the result via either base `R` plotting or `ggplot()`. For simplicity, assume $L = 1$, $x_o = 0$, and $k = 1$, and assume that you step from $x_{\min} = -5$ to $x_{\max} = 5$ in steps of size 0.05. (Hint/note: you can hardwire the numbers given above, so that, e.g., you don't need to figure out how to pass a value of k to `expression`.)

```
x= seq(-5, 5, by= 0.05)
logistic= expression(1/(1+exp(-1*(x))))
df = D(logistic, "x")
plot(x, eval(df), type= "l")
```



Optimization and Bootstrapping

Given a set of data x sampled from a probability density function f with parameters θ, the likelihood is

$$\mathcal{L} = \prod_{i=1}^{n} f(x_i|\theta)$$

and thus the log-likelihood is

$$L = \sum_{i=1}^{n} \log\left[f(x_i|\theta)\right] .$$

(This derivation holds for probability mass functions p as well.) Because we choose the convention that to optimize requires minimizing a fit metric, when we program the function passed as, e.g., the argument `fn` to `optim()`, we return $-L$ as opposed to L.

Question 10

(10 points)

Notes 7B (7)

Below we load in 40 data sampled from a gamma distribution. (The data are contained in the variable `my.data`.) Following the steps shown in Notes_7B, determine (and display!) the optimal estimates for α and β. Also display the value of the minimized negative log-likelihood. If you call `dgamma()` (which is your function $f()$ in this context), the argument name for β is `scale`.

```
load(url("http://www.stat.cmu.edu/~pfreeman/Lab_07_Q10.Rdata"))
my.fit.fun = function(my.par,my.data)
{
  -sum(log(dgamma(my.data,shape=my.par[1],scale=my.par[2])))
}
my.par = c(1,1)
optim.out = optim(my.par,my.fit.fun,my.data=my.data,method="Nelder-Mead")
optim.out$par
```

```
## [1] 1.244689 2.614646
```

```
optim.out$value
```

```
## [1] 86.63177
```

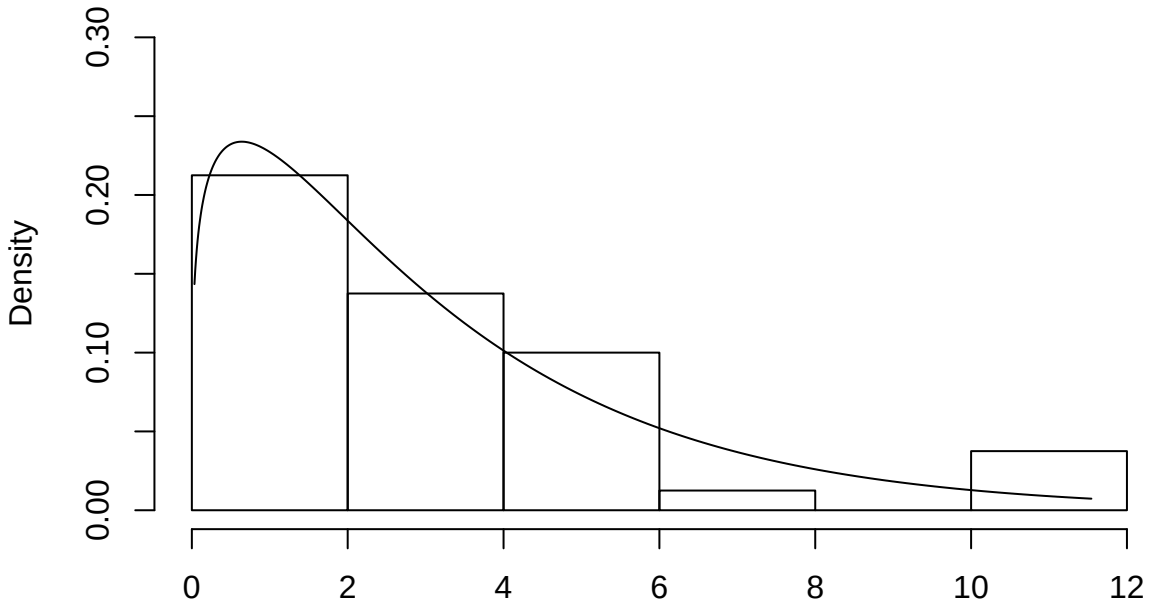
Question 11

(5 points)

Review of Plotting

Histogram the data in `my.data` and overlay a line showing the best-fit gamma function. You can do this using base `R` plot tools or `ggplot()`, whichever you prefer. If the overlaid line goes off the top edge of the plotting window, then adjust the y limits for your histogram.

```
hist(my.data,prob=TRUE,main=NULL,xlab=NULL, ylim = c(0, 0.3))
x = seq(min(my.data),max(my.data),by=0.01)
lines(x,dgamma(x,shape=optim.out$par[1],scale=optim.out$par[2]))
```



Question 12

(10 points)

Notes 7B (10)

As detailed in the documentation for `constrOptim`, the linear constraints on the region of possible solutions in parameter space have to be defined such that

$$U\theta - C \geq 0\,,$$

where U is a matrix and C a column vector. If we have one constraint equation and two parameters, the equation above simplifies to

$$U_{11}\theta_1 + U_{12}\theta_2 - C \geq 0\,,$$

where U is a matrix with one row and two columns. Repeat the fit of Q10 while incorporating the constraint $\alpha + \beta \leq 2$. Display the best-fit parameter values. The best-fit point will lie along the parameter-space constraint; show this is the case by displaying the sum of the estimated parameters $\hat{\alpha}$ and $\hat{\beta}$.

```
my.fit.fun = function(my.par,my.data)
{
  -sum(log(dgamma(my.data,shape=my.par[1],scale=my.par[2])))
}
my.par = c(0.5,0.5)
ui = matrix(c(-1,-1),nrow=1)
ci = -2
optim.out = constrOptim(my.par,my.fit.fun,grad=NULL,ui=ui,ci=ci,my.data=my.data)
optim.out$par
```

```
## [1] 0.7782065 1.2217935
```

```
sum(optim.out$par)
```

```
## [1] 2
```

Question 13

(10 points)

Notes 7C (6)

Utilize the bootstrap, as detailed in Notes_7C, to estimate the uncertainties in $\hat{\alpha}$ and $\hat{\beta}$ for the unconstrained optimization of Q10. Basically, re-implement the example from the last page of Notes_7C, showing histograms for both $\hat{\alpha}$ and $\hat{\beta}$. Set $B = 1000$. In addition to implementing the code from the notes, add in calls to `quantile()` that output the 2.5th percentile and the 97.5th percentile for both $\hat{\alpha}$ and $\hat{\beta}$. In the end, you should see that looking at the marginals for the estimated parameter values does not tell the whole story: the scatter plot shows how $\hat{\beta}$ is very much negatively correlated with $\hat{\alpha}$.

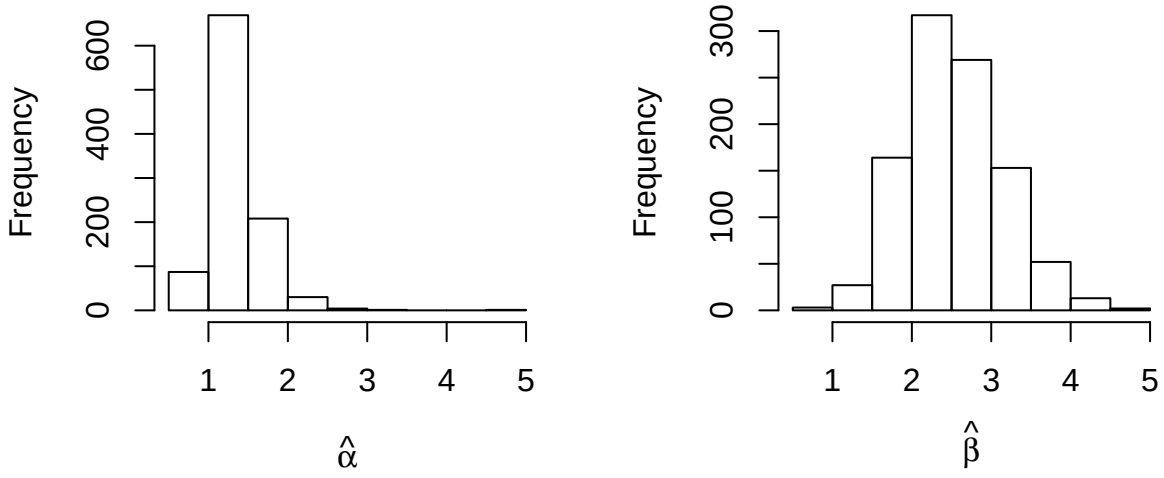
```
B = 1000
load(url("http://www.stat.cmu.edu/~pfreeman/Lab_07_Q10.Rdata"))
my.fit.fun = function(my.par,my.data)
{
  -sum(log(dgamma(my.data,shape=my.par[1],scale=my.par[2])))
}
my.fun = function(x)
{
  optim.out = optim(c(1,1),my.fit.fun,my.data=x)
  return(optim.out$par)
}

indices = sample(length(my.data),B*length(my.data),replace=TRUE)
data.array = matrix(my.data[indices],nrow=B)
apply.out = apply(data.array,1,my.fun)
alpha.hat = apply.out[1,]
beta.hat = apply.out[2,]
quantile(alpha.hat, probs = c(0.025, 0.975))
```

```
##      2.5%     97.5%
## 0.9020125 2.0961622
```

```
quantile(beta.hat, probs = c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 1.465970 3.841489
```

```
par(mfrow=c(1,2))
hist(alpha.hat,xlab=expression(hat(alpha)),main=NULL)
hist(beta.hat,xlab=expression(hat(beta)),main=NULL)
```



Question 14

(10 points)

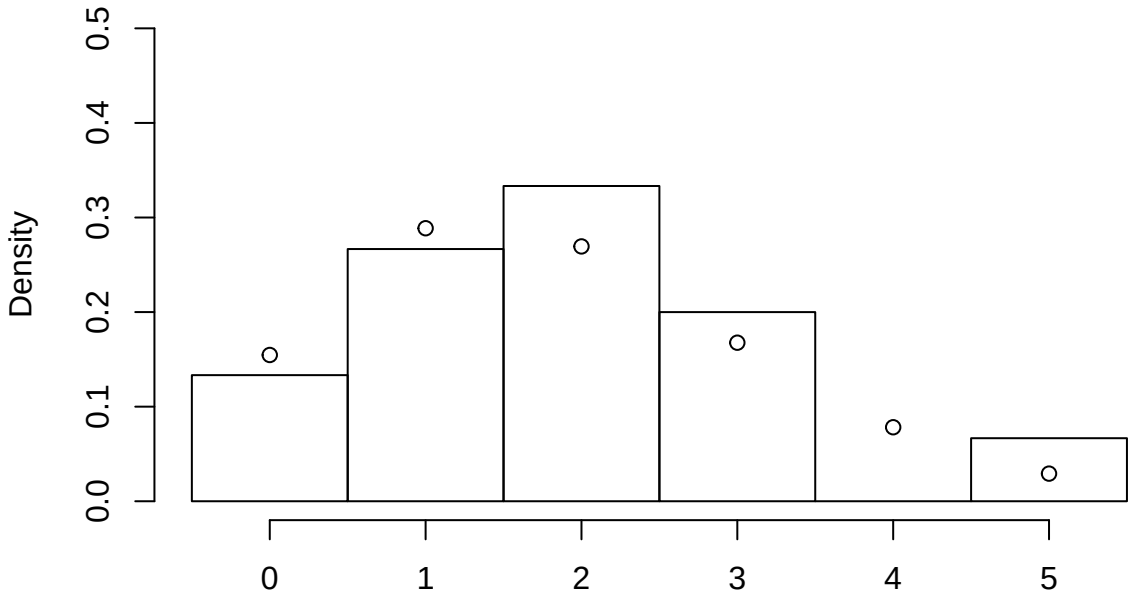
Notes 7B (7-10)

Below we load in 15 data sampled from a Poisson distribution. As above, these data are dubbed `my.data`. Determine (and display) the optimal estimate of λ, using one of the functions shown in Notes_7B. Histogram the data, setting breaks every unit step from -0.5 upwards until all data are included in the histogram, and overlay points showing the probability mass function (i.e., the output from `dpois()`, given the estimate $\hat{\lambda}$) at every possible value of x from the minimum observed value of x to the maximum observed value of x.

```
load(url("http://www.stat.cmu.edu/~pfreeman/Lab_07_Q14.Rdata"))

my.fit.fun = function(my.par,my.data)
{
  -sum(log(dpois(my.data,lambda = my.par)))
}
optim.out = optimize(my.fit.fun,interval = c(0,2), my.data=my.data)
optim.out$minimum
```

```
## [1] 1.866681
```

```
x = seq(-0.5,max(my.data)+1,by=1)
hist(my.data,prob=TRUE,main=NULL,xlab=NULL, ylim = c(0, 0.5),
     breaks= x)
input.x= seq(min(my.data),max(my.data),by=1)
points(input.x,dpois(input.x,lambda=optim.out$minimum))
```



Question 15

(10 points)

Notes 7C (4-6)

Utilize the bootstrap to estimate the uncertainty in $\hat{\lambda}$ in Q14. Assume $B = 1000$ again, and provide a central 95% bootstrap interval again by utilizing the `quantile()` function with appropriate arguments. Histogram your output, and add a vertical line at the true value ($\lambda = 1.75$).

```
load(url("http://www.stat.cmu.edu/~pfreeman/Lab_07_Q14.Rdata"))
B= 1000
my.fit.fun = function(my.par,my.data)
{
  -sum(log(dpois(my.data,lambda = my.par)))
}
my.fun = function(x)
{
  optim.out = optimize(my.fit.fun,interval = c(0,2), my.data=x)
  return(optim.out$minimum)
}

indices = sample(length(my.data),B*length(my.data),replace=TRUE)
data.array = matrix(my.data[indices],nrow=B)
apply.out = apply(data.array,1,my.fun)
sd(apply.out)
```

```
## [1] 0.232565
```

```
quantile(apply.out, probs = c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 1.266670 1.999956
```

```
hist(apply.out,xlab=expression(hat(lambda)),main=NULL)
abline(v=1.75)
```