Stat 516 Homework 2 Solutions

1. See handwritten notes at the end of these solutions.

2. Cut and paste the matrix of numbers to a text file. I saved that file as affypixel.txt. Open R and set the working directory to the directory containing the file. For example

   ```
   setwd("c:/microarray/HW2")
   ```

   Read the file (which will be space delimeted) into an R matrix with a name of your choosing. For example,

   ```
   d=read.table("affypixels.txt")
   ```

   Keep only the middle 36 pixels because Affy discards the first row, first column, last row, and last column.

   ```
   middle=d[2:7,2:7]
   ```

   Now we must find the 75th percentile of the pixel intensities because that is what Affymetrix uses as the intensity for a probe cell. The 75th percentile is the same as the 0.75 quantile of the 36 pixel intensities. According to our class notes, the 0.75 quantile is any number that is greater than or equal to at least 75% of the data points and less than or equal to at least 25% of the data points. Any number between the 27th and 28th pixel intensities (including the 27th and 28th values themselves) ordered from smallest to largest will satisfy this definition. Thus we need to put the pixel intensities in order and select any number between the 27th and 28th values. This can be done in R as follows.

   The object middle that I created above contains the 36 pixel intensities that we want to deal with. It is an object of class data.frame. To determine this, issue the following command.

   ```
   attributes(middle)
   ```

   The command will report the row names associated with middle, the column names for middle, and the class of the object middle. We would like to convert middle to a vector. To do so, issue the following commands.

   ```
   middle=as.vector(as.matrix(middle))
   ```

   Now middle is simply a vector of 36 values. We can sort it as follows.

   ```
   sort(middle)
   ```

   We can look specifically at the 27th and 28th values by issuing the command

   ```
   sort(middle)[27:28]
   ```

   From this command we can easily see that the 27th and 28th values are 32175 and 32228. Thus the 0.75 quantile is any number between 32175 and 32228. I do not know which number Affymetrix uses, but one strategy would be to use the average of these two numbers. This is obtained in R as follows.

   ```
   mean(sort(middle)[27:28])
   ```

   This gives 32201.5 as the 0.75 quantile. Any answer in the interval [32175, 32228] is acceptable for this homework problem.

   We could alternatively use the quantile function in R to get a 0.75 quantile.

   ```
   quantile(middle,0.75)
   ```

   This command gives 32188.25 as a 0.75 quantile. This number is 25% of the way between 32175 and 32228.

3.
```
affy.tbw=
function (x)
{
  m=median(x)
  mad=median(abs(x-m))
  u=(x-m)/(5*mad+.0001)
  tbw=weighted.mean(x,w=bisquare(u))
  return(tbw)
}

bisquare=
function (u)
{
  b=rep(0,length(u))
  index=abs(u)<1
  b[index]=(1-u[index]^2)^2
  return(b)
}
```

4. (a)
```
ideal.mismatch=
function (x)
{
  d=x[1,]-x[2,]
  if(sum(x[1,]-x[2,]<=0)>0){
    M=affy.tbw(log(x[1,]/x[2,],2))
    if(M>.03){
      im=x[1,]/2^M
    }
    else{
      im=x[1,]/2^(0.03/(1+((0.03-M)/10)))
    }
    im[d>0]=x[2,d>0]
  }
  else{
    im=x[2,]
  }
  return(im)
}
```

```
810.0751 2238.0000 1887.0000  300.0000  647.9875  154.0000  156.0000
```

(b)
```
probe.value=
function (x)
{
  pm=x[1,]
  im=ideal.mismatch(x)
  v=pmax(pm-im,2^(-20))
```

```
  return(log(v,2))
}

10.475657 10.737247 11.195372 11.195372 10.153568  9.885696 11.096056
```

(c)
```
affy.tbw=
function (x)
{
  m=median(x)
  mad=median(abs(x-m))
  u=(x-m)/(5*mad+0.0001)
  tbw=weighted.mean(x,w=bisquare(u))
  return(tbw)
}

signal.log.value=
function (x)
{
  pv=probe.value(x)
  return(affy.tbw(pv))
}

10.70408
```

5. I cut and pasted the data from the pdf document to a file called rmadata.txt. This is simply a space-delimited file that can be read in R as follows:

```
rmadata=read.table("rmadata.txt")

mp=medpolish(rmadata)
1 : 15.17
2 : 13.785
Final: 13.7025

fitted.values=rmadata-mp$residuals
apply(fitted.values,1,mean)

       1        2        3        4        5        6        7
 9.68216 11.60466 10.62466 10.68716 10.96966  9.56966 10.48216
       8        9       10
10.05716 10.64716 11.13966
```

You will have a slightly different answer if you iterated for more steps than the default in the R function medpolish.

For example `mp=medpolish(rmadata,eps=0.000000000000000001,maxiter=100)` will cause R to iterate 45 times for this dataset. The answer will change as follows

```
fitted.values=rmadata-mp$residuals
apply(fitted.values,1,mean)
        1         2         3         4         5         6         7
 9.685227 11.603977 10.610227 10.687727 10.970227  9.570227 10.483977
        8         9        10
10.055227 10.647727 11.147727
```

6. (a) $f(x,m) = (x-m)^2$ because the sample mean minimizes $g(m) = \sum_{i=1}^{n}(x_i - m)^2$.

   (b) $f(x,m) = |x-m|$ because the sample median minimizes $g(m) = \sum_{i=1}^{n}|x_i - m|$.

   (c)
```
MAD=
function (x)
{
  return(median(abs(x-median(x))))
}

three.d=
function (m,x,k)
{
  u=(x-m)/(k*MAD(x))
  out=(1-(1-u^2)^3)/6
  out[abs(u)>1]=1/6
  return(sum(out))
}
```

   (d) The key to getting optimize to work well is to carefully select the interval over which optimize will search for a solution. If that interval is too wide, optimize will not find the proper minimizer. I originally tried to use the maximum and minimum of the x vector as the interval. This fails if there are extreme outlying points. I ultimately decided to use the median of the sample plus or minus 3 times the MAD value. That seemed to work well for the examples that I tried.

```
tukey.optimize=
function (x,k)
{
  med=median(x)
  mad=MAD(x)
  return(optimize(three.d,lower=med-3*mad,
         upper=med+3*mad,tol=0.000000000001,x=x,k=k)$minimum)
}
```

   (e)
```
one.step.tbw=
function (x)
{
  m=median(x)
  mad=median(abs(x-m))
  u=(x-m)/(5*mad)
  tbw=weighted.mean(x,w=bisquare(u))
  return(tbw)
}
```

```
one.step.tbw(ds1)
[1] 8.509166
one.step.tbw(ds2)
[1] 0.8572995
tukey.optimize(ds1,k=5)
[1] 8.072215
tukey.optimize(ds2,k=5)
[1] 0.8688253
```

(f)
```
multi.step.tbw=
function (x,it)
{
  m=median(x)
  mad=median(abs(x-m))
  u=(x-m)/(5*mad)
  tbw=weighted.mean(x,w=bisquare(u))
  for(i in 2:it){
    cat(paste("Iteration",i-1,"TBW=",tbw,"\n"))
    u=(x-tbw)/(5*mad)
    tbw=weighted.mean(x,w=bisquare(u))
  }
  cat(paste("Iteration",i,"TBW=",tbw,"\n"))
  return(tbw)
}

multi.step.tbw(ds1,10)
Iteration 1 TBW= 8.50916568679602
Iteration 2 TBW= 8.15035165520759
Iteration 3 TBW= 8.08618719219727
Iteration 4 TBW= 8.07471363553245
Iteration 5 TBW= 8.07266192154818
Iteration 6 TBW= 8.07229502898253
Iteration 7 TBW= 8.07222942025397
Iteration 8 TBW= 8.0722176879193
Iteration 9 TBW= 8.07221558991062
Iteration 10 TBW= 8.07221521473887
[1] 8.072215
```

If we insist on agreement to 5 decimal places, agreement occurs after 8 iterations.

```
multi.step.tbw(ds2,10)
Iteration 1 TBW= 0.857299535076844
Iteration 2 TBW= 0.867707880396175
Iteration 3 TBW= 0.868716985246538
Iteration 4 TBW= 0.868814816742477
Iteration 5 TBW= 0.868824301364025
Iteration 6 TBW= 0.868825220884098
Iteration 7 TBW= 0.86882531003021
Iteration 8 TBW= 0.868825318672795
Iteration 9 TBW= 0.868825319510681
```

```
Iteration 10 TBW= 0.868825319591913
[1] 0.8688253
```

If we truncate digits beyond the fifth decimal place, we have agreement after 5 iterations.

A solution to problem 1 is on the following pages.

The joint density of X_i and Y_i is

$$f(x,y) = \begin{cases} \lambda e^{-\lambda x} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y-\mu)^2} & \text{for } x>0,\ y\in\mathbb{R} \\ 0 & \text{otherwise.} \end{cases}$$

Note that

$$\begin{aligned} X_i &= X_i \\ D_i &= X_i + Y_i \end{aligned} \quad \Rightarrow \quad \begin{aligned} X_i &= X_i \\ Y_i &= D_i - X_i \end{aligned}$$

Thus, the joint density of X_i and D_i, say $g(x,d)$, is

$$g(x,d) = f(x, d-x) = \lambda e^{-\lambda x} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d-x-\mu)^2}$$

for $x>0$, $d\in\mathbb{R}$ and 0 otherwise.

(Note that the Jacobian matrix is $\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$, which has determinant 1.)

$$g(x,d) = \lambda \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}\{(d-\mu)^2 - 2x(d-\mu) + x^2 + 2\lambda\sigma^2 x\}}$$

$$= \lambda e^{\frac{-1}{2\sigma^2}(d-\mu)^2} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2\sigma^2}\{x^2 - 2x(d-\mu-\lambda\sigma^2)\}}$$

$$= \lambda e^{\frac{-1}{2\sigma^2}\{(d-\mu)^2 - (d-\mu-\lambda\sigma^2)^2\}} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2\sigma^2}\{x-(d-\mu-\lambda\sigma^2)\}^2}$$

The conditional distribution of X_i given D_i, say $h(x|d)$, is

$$h(x|d) = \frac{g(x,d)}{\int_0^\infty g(x,d)\,dx} = \frac{\phi(x; d-\mu-\lambda\sigma^2, \sigma^2)}{1 - \underline{\Phi}(0; d-\mu-\lambda\sigma^2, \sigma^2)} \quad \text{for } x > 0.$$

Thus, $$E(X_i|D_i) = \frac{\int_0^\infty x\,\phi(x; d-\mu-\lambda\sigma^2, \sigma^2)\,dx}{1 - \underline{\Phi}(0; d-\mu-\lambda\sigma^2, \sigma^2)}$$

Now $\int_0^\infty x\,\phi(x;\alpha,\kappa^2)\,dx = \int_0^\infty x \frac{1}{\sqrt{2\pi\kappa^2}} e^{-\frac{1}{2\kappa^2}(x-\alpha)^2} dx$

$= \int_0^\infty (x-\alpha) \frac{1}{\sqrt{2\pi\kappa^2}} e^{-\frac{1}{2\kappa^2}(x-\alpha)^2} dx + \int_0^\infty \alpha\,\phi(x;\alpha,\kappa^2)\,dx$

$= \kappa^2 \left[-\phi(x;\alpha,\kappa^2)\right]_0^\infty + \alpha\left\{1 - \underline{\Phi}(0;\alpha,\kappa^2)\right\}$

$= \kappa^2\,\phi(0;\alpha,\kappa^2) + \alpha\left\{1 - \underline{\Phi}(0;\alpha,\kappa^2)\right\}.$

Thus, $E(X_i \mid D_i = d) = \dfrac{\sigma^2\,\phi(0;d-\mu-\lambda\sigma^2,\sigma^2) + (d-\mu-\lambda\sigma^2)\left\{1-\underline{\Phi}(0;d-\mu-\lambda\sigma^2,\sigma^2)\right\}}{1-\underline{\Phi}(0;d-\mu-\lambda\sigma^2,\sigma^2)}$

$= d-\mu-\lambda\sigma^2 + \dfrac{\sigma^2\,\phi(0;d-\mu-\lambda\sigma^2,\sigma^2)}{1-\underline{\Phi}(0;d-\mu-\lambda\sigma^2,\sigma^2)}$

The expression in the notes is correct.