STAT34800 HW3

Sarah Adilijiang

Problem A

Note: The original codes are not shown here.

(1) Method accuracy: Trend Filtering vs Wavelet Smoothing

(a) Trend Filtering

Considerations:

Because the x axis here is cyclical, the value of $E(Y|x)$ near $x = 0$ should be similar to the value near $x = 2\pi$. But the trend filtering method using package "genlasso" does not know this. Thus we can encourage this periodic behaviour by duplicating the data using a translation.

```
library(genlasso)
```

```
## Loading required package: MASS
```

```
## Loading required package: Matrix
```

```
## Loading required package: igraph
```

```
## Warning: package 'igraph' was built under R version 3.5.3
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```
# read data & order by "theta"
d = readRDS("cyclegenes.rds")
d = d[order(d[,1]), ]

# Trend Filtering
n = nrow(d)
MSE_trend = rep(NA,10)
pred_trend = matrix(NA,n,10)

for (i in 1:10) {
    yy = c(d[,i+1], d[,i+1], d[,i+1])            # duplicated gene expression data
    xx = c(d$theta-2*pi, d$theta, d$theta+2*pi)  # shifted/translated x coordinates

    yy.tf2 = trendfilter(yy, ord=2)     # order=2
    yy.tf2.cv = cv.trendfilter(yy.tf2)  # 5-fold CV

    include = c(rep(FALSE,n), rep(TRUE,n), rep(FALSE,n))
    pred_trend[,i] = predict(yy.tf2, yy.tf2.cv$lambda.min)$fit[include]
```

```r
    MSE_trend[i] = sum((yy[include]-pred_trend[,i])^2)/n
}
```

```
## Warning: 'rBind' is deprecated.
##  Since R version 3.2.0, base's rbind() should work fine with S4 objects

## Fold 1 ... Fold 2 ... Fold 3 ... Fold 4 ... Fold 5 ...
## Fold 1 ... Fold 2 ... Fold 3 ... Fold 4 ... Fold 5 ...
## Fold 1 ... Fold 2 ... Fold 3 ... Fold 4 ... Fold 5 ...
## Fold 1 ... Fold 2 ... Fold 3 ... Fold 4 ... Fold 5 ...
## Fold 1 ... Fold 2 ... Fold 3 ... Fold 4 ... Fold 5 ...
## Fold 1 ... Fold 2 ... Fold 3 ... Fold 4 ... Fold 5 ...
## Fold 1 ... Fold 2 ... Fold 3 ... Fold 4 ... Fold 5 ...
## Fold 1 ... Fold 2 ... Fold 3 ... Fold 4 ... Fold 5 ...
## Fold 1 ... Fold 2 ... Fold 3 ... Fold 4 ... Fold 5 ...
## Fold 1 ... Fold 2 ... Fold 3 ... Fold 4 ... Fold 5 ...
```

(b) Wavelet Smoothing

Considerations:

(i) Because the default in "wavethresh" package is to assume the data are periodic on their region of definition - that is, circular. So we don't have to do anything special here to deal with the circularity of the data.

(ii) For cross-validation, to find the prediction on the validation set, we can first find the corresponding nearest x coordinate in the training set, then use the fitted value of this data point as the prediction on the validation set.

(iii) The **wd** function in "wavethresh" package requires the lenghth of the data vector to be a power of 2. Since we have 990 elements, we can subset the data to 512 elements here, and get the fitted values. For the left 478 elements, we can use the method in (ii) to find the predictions of these points. Besides, since we want to perform 5-fold cross-validation, we should subset the data to 640 elements here for 5-fold CV: 512 in training set and 128 in validation set.

First, we calculate the average MSE of 5-fold cross-validation for wavelet smoothing method to compare with trend filtering method.

```r
set.seed(1)
library(wavethresh)
```

```
## Warning: package 'wavethresh' was built under R version 3.5.3

## WaveThresh: R wavelet software, release 4.6.8, installed

## Copyright Guy Nason and others 1993-2016

## Note: nlevels has been renamed to nlevelsWT
```

```r
# read data & order by "theta"
d = readRDS("cyclegenes.rds")
d = d[order(d[,1]), ]

# function for finding data index of prediction on test set
pred_index = function(train.X, test.X){
    index = rep(NA,length(test.X))
    for(i in 1:length(test.X)){
        index[i] = which.min(abs(test.X[i]-train.X))
        }
```

```
    return(index)
}

# Wavelet Smoothing: average MSE of 5-fold CV
index = sample(1:nrow(d),640,replace=FALSE)
d.sub = d[index, ]
MSE_wavelet = matrix(NA,10,5)

for (i in 1:10) {
    for (j in 1:5) {
        valid = d.sub[(1:640)%%5==(j-1), ]
        train = d.sub[(1:640)%%5!=(j-1), ]
        wds = wd(train[,i+1], family="DaubLeAsymm", filter.number=8)
        wtd = threshold(wds, levels=4:8, policy="manual", value=99999)
        fd = wr(wtd)    # reconstruct
        index = pred_index(train$theta, valid$theta)
        MSE_wavelet[i,j] = sum((valid[,i+1]-fd[index])^2)/128
    }
}
MSE_wavelet = rowMeans(MSE_wavelet)
```

Then, we find the fitted values and predictions for all the data points.

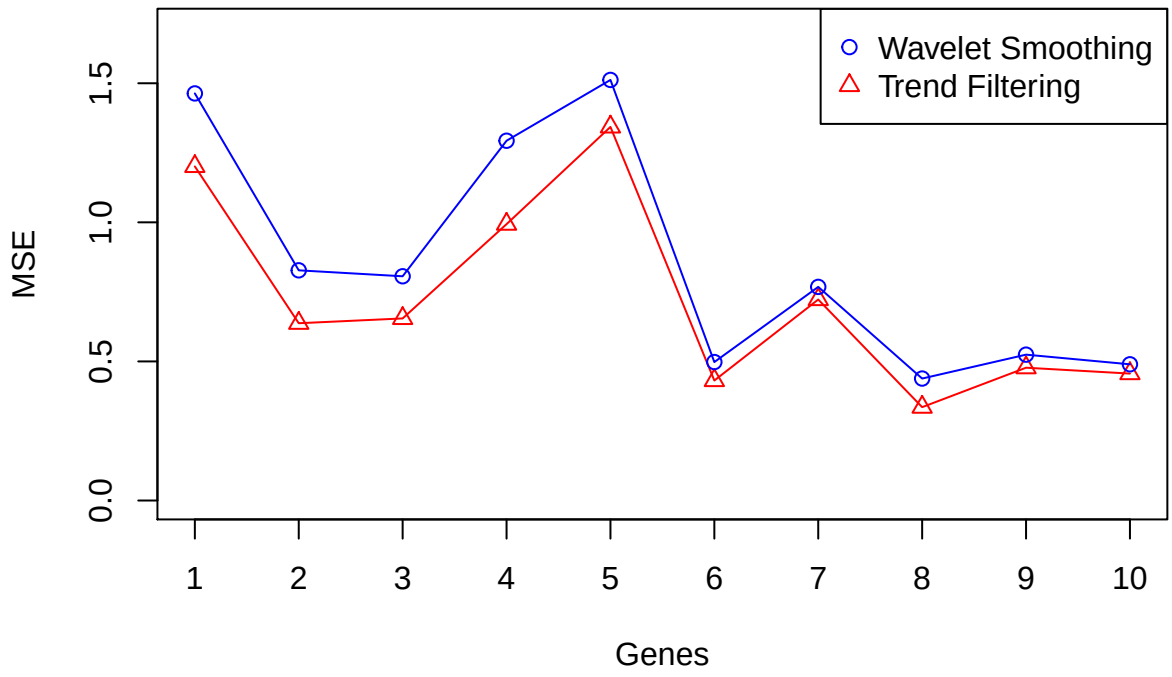
```
# subset data (512 elements)
sub_inx = sort(sample(1:nrow(d),512,replace=FALSE))
d.sub = d[sub_inx, ]
d.other = d[-sub_inx, ]

# Wavelet Smoothing
pred_wavelet = matrix(NA,nrow(d),10)
for (i in 1:10) {
    wds = wd(d.sub[,i+1], family="DaubLeAsymm", filter.number=8)
    wtd = threshold(wds, levels=4:8, policy="manual", value=99999)
    fd = wr(wtd)    # reconstruct
    pred_wavelet[sub_inx,i] = fd

    index = pred_index(d.sub$theta, d.other$theta)
    pred_wavelet[-sub_inx,i] = fd[index]
}
```

(c) Comparison between two methods

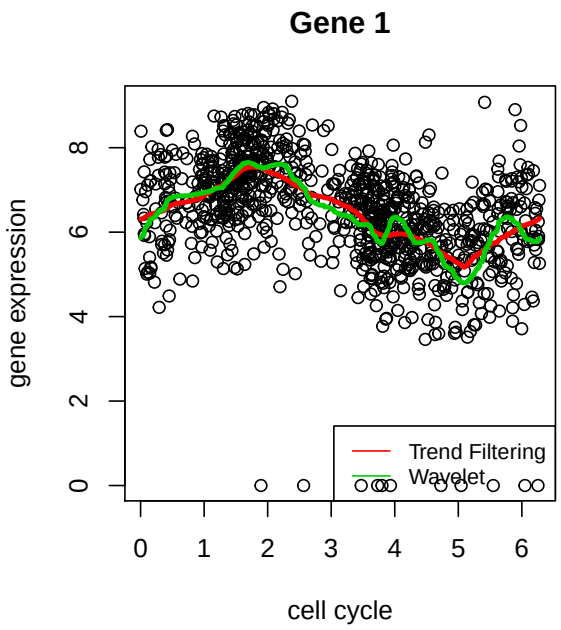
```
# MSE values
compare = rbind(MSE_trend, MSE_wavelet)
rownames(compare) = c("MSE_trend", "MSE_wavelet")
compare
```

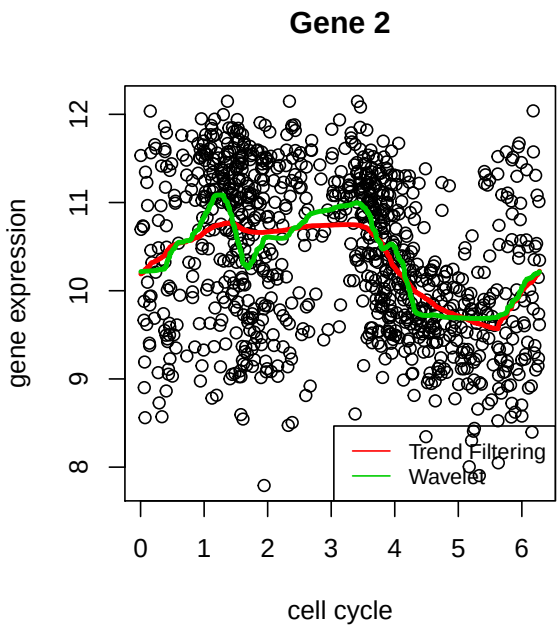
```
##                 [,1]      [,2]      [,3]      [,4]     [,5]      [,6]
## MSE_trend   1.200954 0.6372830 0.6545507 0.9935207 1.343053 0.4313456
## MSE_wavelet 1.463420 0.8274853 0.8061333 1.2933917 1.511992 0.4977332
##                 [,7]      [,8]      [,9]     [,10]
## MSE_trend   0.7225216 0.3355743 0.4773332 0.4562608
## MSE_wavelet 0.7679599 0.4384518 0.5243480 0.4896261
```

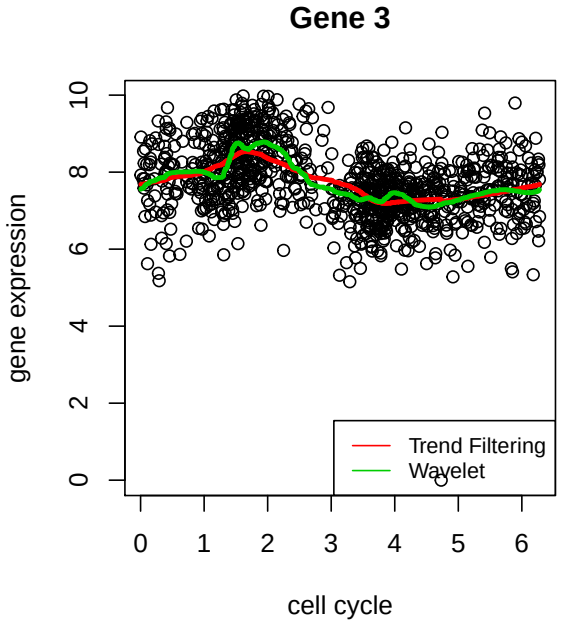
```r
# plot of MSE values
plot(MSE_trend~c(1:10), pch=2, col=2, xlab="Genes", ylab="MSE", ylim=c(0,1.7), xaxt='n')
axis(side=1, at=1:10)
lines(MSE_trend~c(1:10), col=2)
points(MSE_wavelet~c(1:10), pch=1, col=4)
lines(MSE_wavelet~c(1:10), col=4)
legend("topright", legend=c("Wavelet Smoothing","Trend Filtering"), pch=c(1,2), col=c(4,2))
```

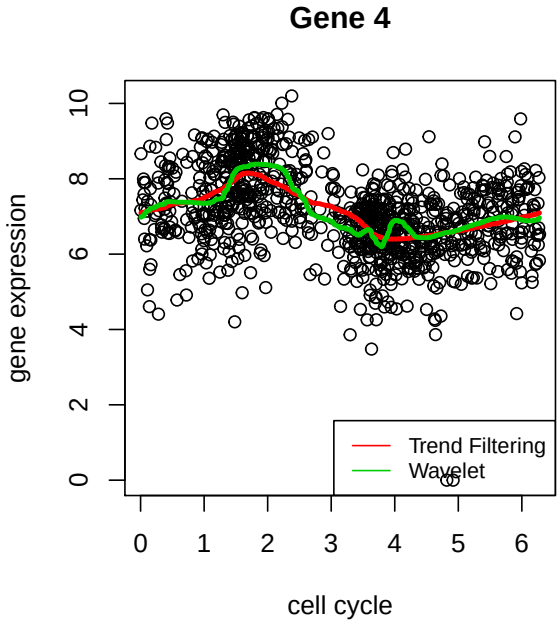


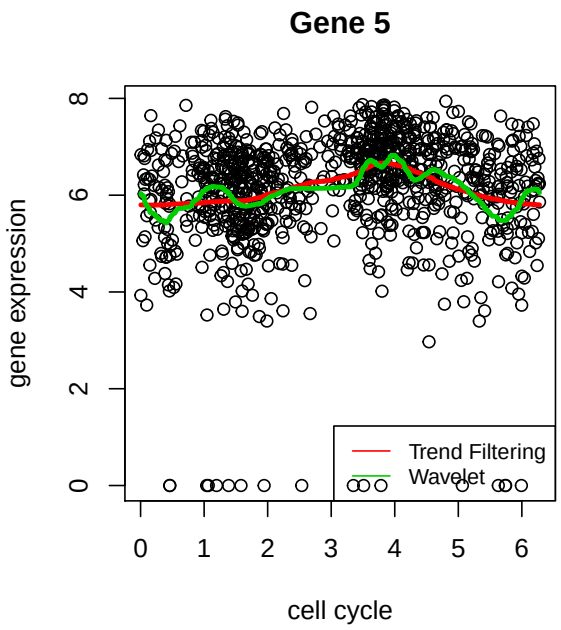

```r
# plots of predictions of all 10 genes
par(mfrow=c(1,2))
for (i in 1:10) {
    plot(d$theta, d[,i+1], xlab="cell cycle", ylab="gene expression", main=paste0("Gene ",i))
    lines(d$theta, pred_trend[,i], col=2, lwd=3)
    lines(d$theta, pred_wavelet[,i], col=3, lwd=3)
    legend("bottomright", legend=c("Trend Filtering", "Wavelet"), col=c(2,3), lty=1, cex=0.8)
}
```

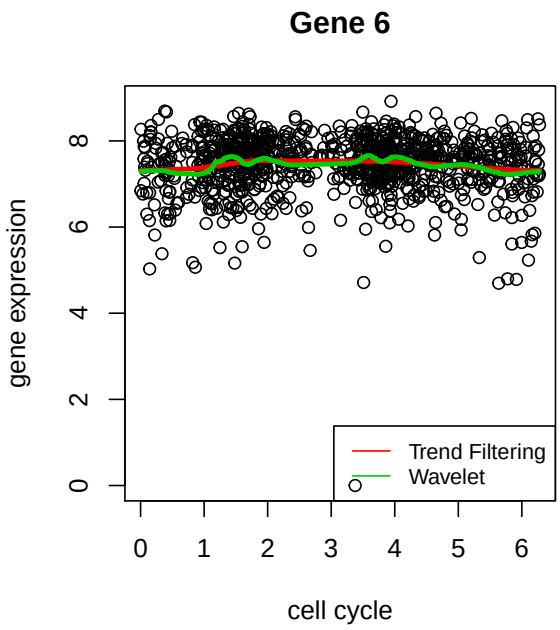

Gene 1
gene expression
cell cycle
Trend Filtering
Wavelet

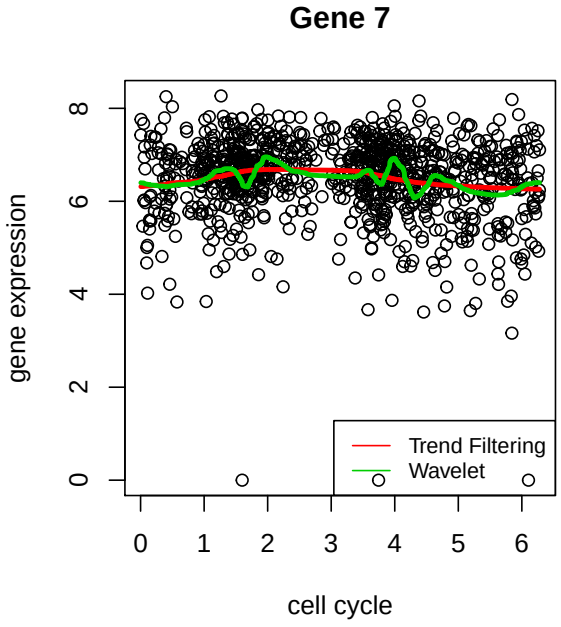

Gene 2
gene expression
cell cycle
Trend Filtering
Wavelet

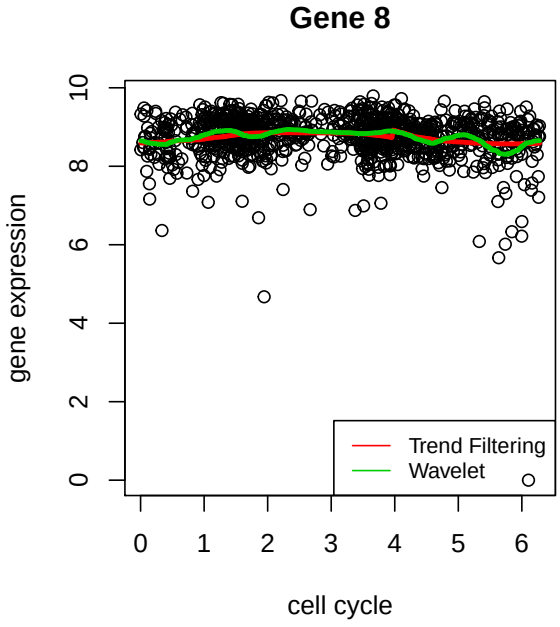

Gene 3
gene expression
cell cycle
Trend Filtering
Wavelet

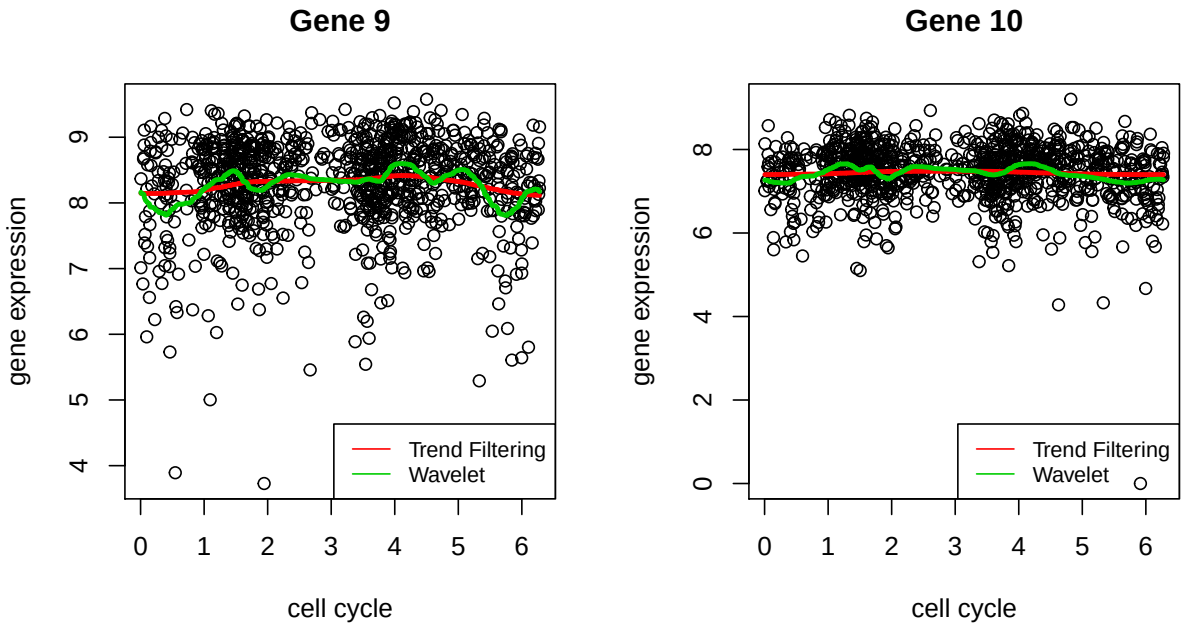

Gene 4
gene expression
cell cycle
Trend Filtering
Wavelet


Gene 5
gene expression
cell cycle
Trend Filtering
Wavelet


Gene 6
gene expression
cell cycle
Trend Filtering
Wavelet


Gene 7
gene expression
cell cycle
Trend Filtering
Wavelet


Gene 8
gene expression
cell cycle
Trend Filtering
Wavelet



Discussion:

(1) Selected version of each method:

For trend filtering, I choose the model of **order 2**, which provides a smoother fit, being less "spiky" in places compared with the model of **order 1**.

For wavelet smoothing, I choose the model that uses Daubechies Least-Asymmetric Wavelets (**family="DaubLeAsymm"**) with more smoothness of wavelet (**filter.number=8**). This method is less "jumpy" by using a "less step-wise" wavelet basis, while using **family="DaubExPhase"** with **filter.number=1** is a bit "jumpy", due to the use of the Haar wavelet and the rather naive hard thresholding.

(2) Measure of accuracy:

Here I use MSE (mean squared error) as the measure of accuracy for the cross-validation methods.

For the trend filtering, the 5-fold cross-validation is performed using the "genlasso" package function. For wavelet smoothing, the 5-fold cross-validation is performed using my own codes.

(3) Results:

The MSE results show that Trend Filtering method has better accuracy than the Wavelet Smoothing for all the 10 genes.

In the plots of all the 10 genes, we can see that the fit of Trend Filtering is more smooth, while the fit of Wavelet smoothing is more "jumpy". This is consistent with the result that the Trend Filtering method fit the data better than the Wavelet Smoothing.

(2) Rank the genes

In question (1), we have found that the Trend Filtering method fit the data better thus being more accurate than the Wavelet Smoothing method. Therefore, in this question, we use the Trend Filtering method to rank the 10 genes.

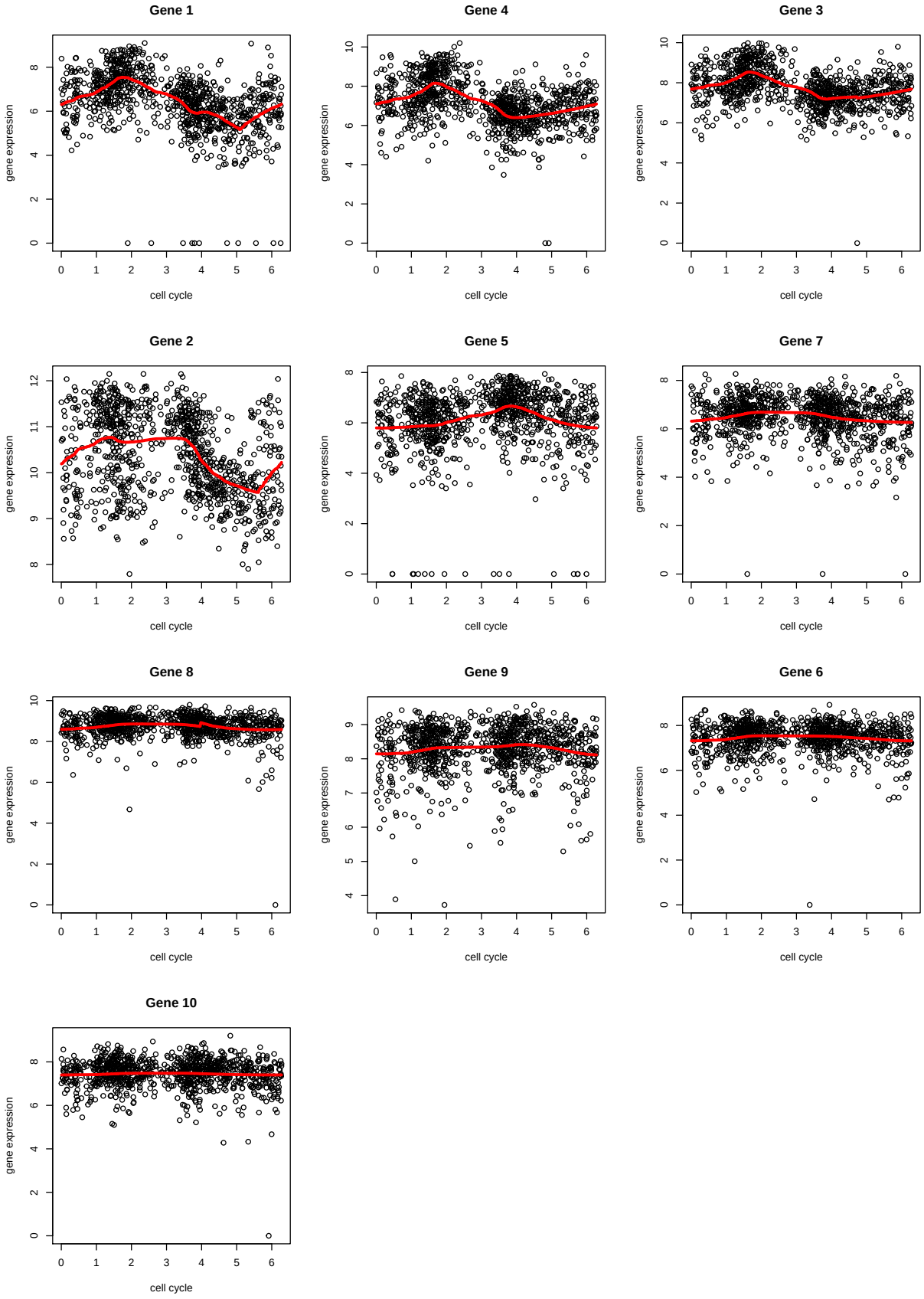
The original question is: which genes show greatest evidence for varying in their expression through the cell cycle? So here we need to pick out genes where the change in the mean over theta is most variable (in some

sense). Therefore, I choose the **variance of the fitted values** as the numeric criteria to rank the genes.

```
# rank the genes by variance of Trend Filtering fitted values
variance = rep(NA, 10)
for (i in 1:10) {
    variance[i] = var(pred_trend[,i])
}
rank = order(variance, decreasing=TRUE); rank
```

```
##  [1]  1  4  3  2  5  7  8  9  6 10
```

```
# check whether the ranking is reasonable
par(mfrow=c(4,3))
for (i in 1:10) {
    plot(d$theta, d[,rank[i]+1], xlab="cell cycle", ylab="gene expression",
         main=paste0("Gene ",rank[i]))
    lines(d$theta, pred_trend[,rank[i]], col=2, lwd=3)
}
```


Gene 1
Gene 4
Gene 3
Gene 2
Gene 5
Gene 7
Gene 8
Gene 9
Gene 6
Gene 10
gene expression
cell cycle

Comments:

The ranking of genes from the highest variance (strongest evidence) to the smallest variance (weakest evidence) is: 1, 4, 3, 2, 5, 7, 8, 9, 6, 10.

The plots of ranked genes is consistent with their visual variability over "theta". Thus this ranking method is reasonable.

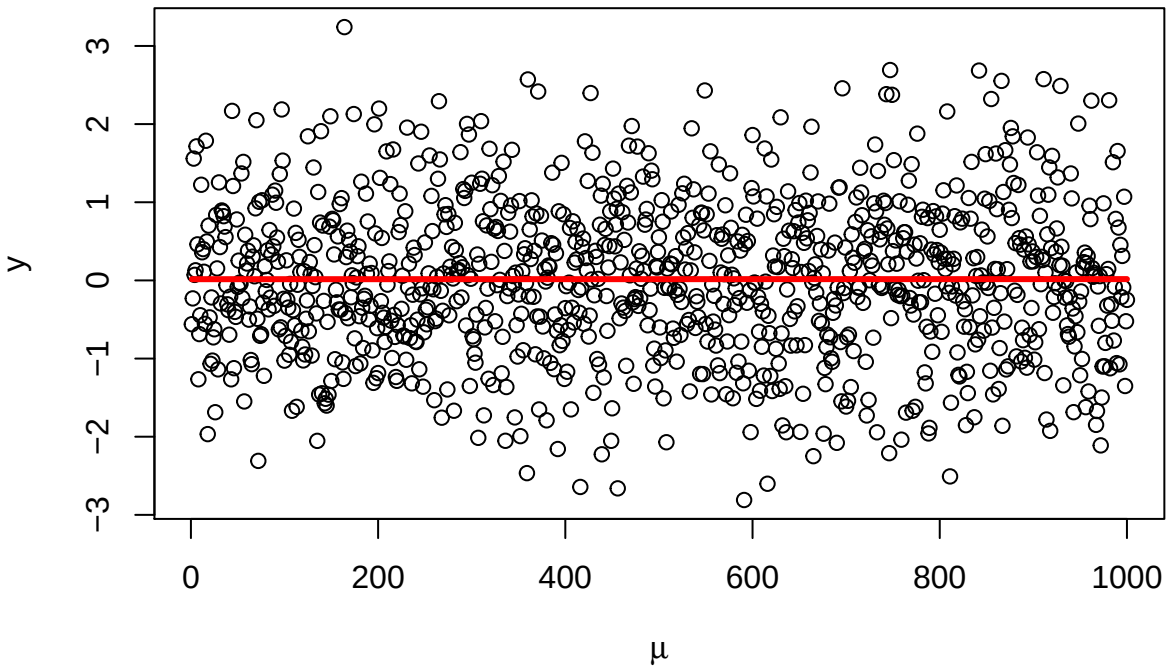
Problem B

(1) Constant mean case

```
# simulate data
set.seed(123)
mu = rep(0,1000)      # constant mean
e = rnorm(1000,0,1)
y = mu + e

# Trend Filtering
y.tf = trendfilter(y, ord=1)      # order=1
y.tf.cv = cv.trendfilter(y.tf)    # 5-fold CV
```

```
## Fold 1 ... Fold 2 ... Fold 3 ... Fold 4 ... Fold 5 ...
```

```
plot(1:1000, y, xlab=expression(mu), ylab=expression(y))
lines(1:1000, predict(y.tf, y.tf.cv$lambda.min)$fit, col=2, lwd=3)
```

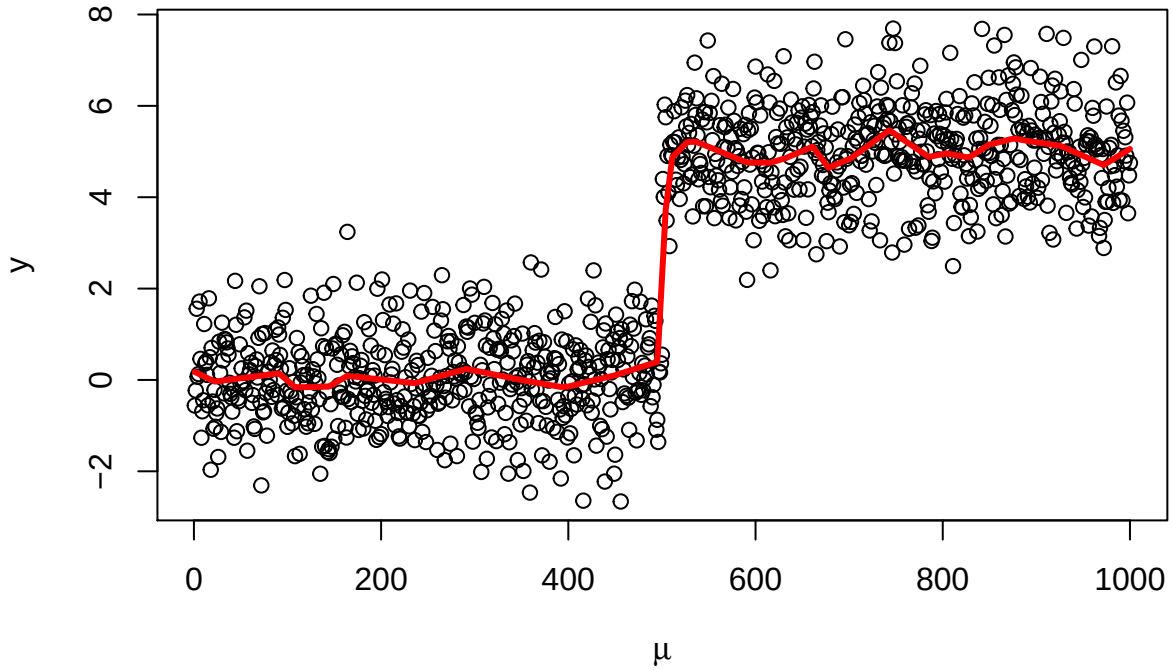


(2) Step function case

```
# simulate data
mu2 = c(rep(0,500), rep(5,500))    # the mean vector is a step function
y2 = mu2 + e        # use same error as in constant mean case

# Trend Filtering
y2.tf = trendfilter(y2, ord=1)      # order=1
y2.tf.cv = cv.trendfilter(y2.tf)    # 5-fold CV
```

```
## Fold 1 ... Fold 2 ... Fold 3 ... Fold 4 ... Fold 5 ...
```

```
plot(1:1000, y2, xlab=expression(mu), ylab=expression(y))
lines(1:1000, predict(y2.tf, y2.tf.cv$lambda.min)$fit, col=2, lwd=3)
```



(3) Comparison

```
# optimal lambda chosen by CV
y.tf.cv$lambda.min
```

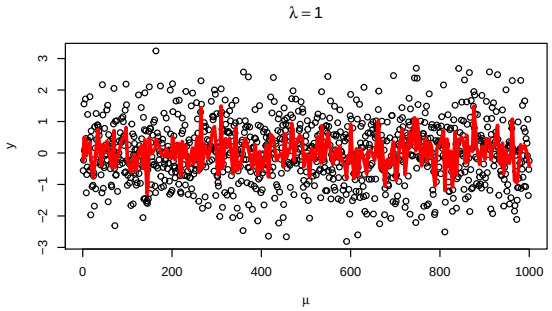
```
## [1] 1057.34
```

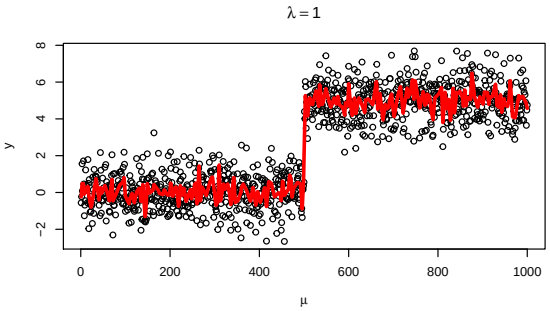
```
y2.tf.cv$lambda.min
```

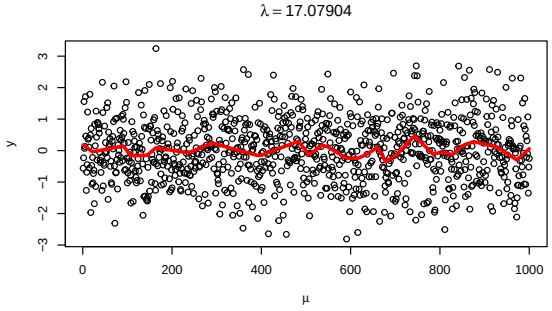
```
## [1] 17.07904
```

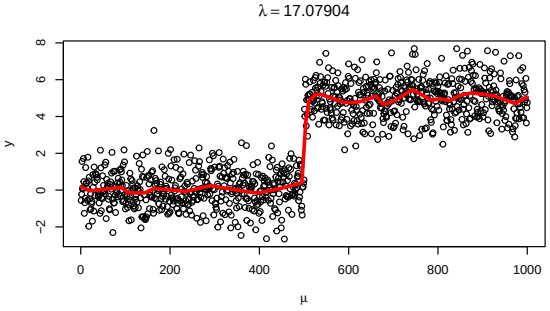
```
# try different lambda values
lambdas = c(1, y2.tf.cv$lambda.min, 100, y.tf.cv$lambda.min)
```

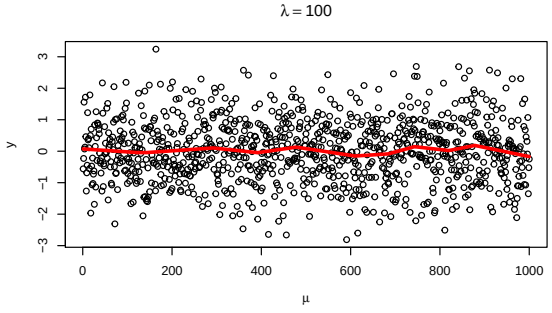
```r
par(mfrow=c(5,2))
for (i in 1:length(lambdas)) {
    plot(1:1000, y, xlab=expression(mu), ylab=expression(y), main=bquote(lambda == .(lambdas[i])))
    lines(1:1000, predict(y.tf, lambda=lambdas[i])$fit, col=2, lwd=3)

    plot(1:1000, y2, xlab=expression(mu), ylab=expression(y), main=bquote(lambda == .(lambdas[i])))
    lines(1:1000, predict(y2.tf, lambda=lambdas[i])$fit, col=2, lwd=3)
}
```

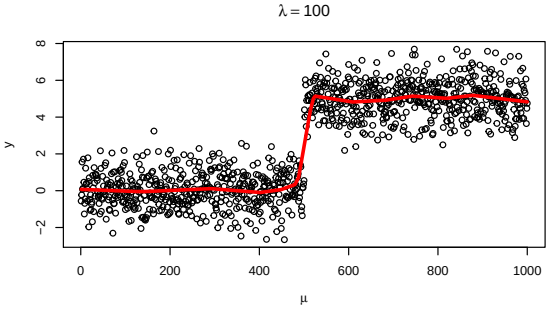

λ = 1
y
μ

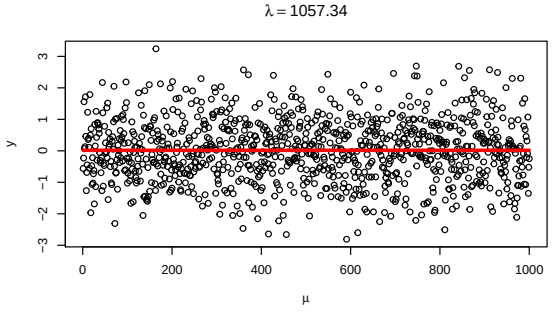

λ = 1
y
μ

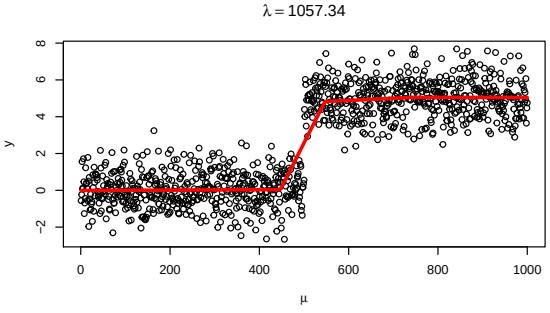

λ = 17.07904
y
μ


λ = 17.07904
y
μ


λ = 100
y
μ


λ = 100
y
μ


λ = 1057.34
y
μ


λ = 1057.34
y
μ

Discussion:

We can see that in the step function case, the Trend Filtering method does find the big change in mean accurately, but the estimate of μ in the regions where it is not changing becomes less smooth.

Trend Filtering method is implemented via L1-penalized regression. By exploring different *lambda* values of L1-penalty, we see that when *lambda* increases, the changes in means $|\mu_k - \mu_{k+1}|$ is more sparse (near zero), thus the fit becomes more smooth.

Here in the constant mean case, the optimal $\lambda = 1057.34$, while in the step function case, the optimal $\lambda = 17.07904$. The optimal λ in the step function is smaller beacuse it needs to capture the big change in the mean. As a result, this smaller optimal λ value makes the estimate of μ in the regions where it is not changing becomes less smooth.

Problem C

(1) Conjugate prior for Multinomial probability vector

Prior distribution of p is: $p = (p_1, ..., p_k) \sim Dirichlet(\alpha_1, ..., \alpha_k)$, so:

$$P(p|\alpha) = \frac{\Gamma(\alpha_1 + \cdots + \alpha_k)}{\Gamma(\alpha_1) \cdots \Gamma(\alpha_k)} \prod_{i=1}^{k} p_i^{\alpha_i - 1} \qquad (p_i \geq 0;\ \sum_{i=1}^{k} p_i = 1)$$

Since $X = (X_1, ..., X_k) \sim Multinomial(n, p)$, so the likelihood function is:

$$P(X|p) \ = \frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \cdots p_k^{x_k} \ = \ \frac{\Gamma(\sum_{i=1}^{k} x_i + 1)}{\prod_{i=1}^{k} \Gamma(x_i + 1)} \prod_{i=1}^{k} p_i^{x_i}$$

Thus the posterior distribution of p is:

$$P(p|X) \propto \ P(X|p)\ P(p|\alpha) \propto \ \prod_{i=1}^{k} p_i^{x_i + \alpha_i - 1} \qquad (p_i \geq 0;\ \sum_{i=1}^{k} p_i = 1)$$

Therefore, the posterior distribution of p is: $p|X \sim Dirichlet(\alpha_1 + x_1, ..., \alpha_k + x_k)$

And the posterior mean of p_i is:

$$E(p_i|X) = \frac{\alpha_i + x_i}{\sum_{j=1}^{k} (\alpha_j + x_j)} \qquad (i = 1, ..., k)$$

(2) EB estimation function

Assume $\alpha_1 = \cdots = \alpha_k = \alpha$, so $\sum_{i=1}^{k} \alpha_i = k\alpha$

The marginal distribution of $X|n, \alpha$ (integrating out the vector p) is known as the "Dirichlet-multinomial" distribution, where $n = \sum_{i=1}^{k} x_i$ is known. The probability mass function is:

$$P(X|\alpha) = \int_p P(X|p)\ P(p|\alpha)\ dp = \frac{(n!)\ \Gamma(\sum_{i=1}^{k} \alpha_i)}{\Gamma(n + \sum_{i=1}^{k} \alpha_i)} \prod_{i=1}^{k} \frac{\Gamma(x_i + \alpha_i)}{(x_i!)\ \Gamma(\alpha_i)} = \frac{(n!)\ \Gamma(k\alpha)}{\Gamma(n + k\alpha)} \prod_{i=1}^{k} \frac{\Gamma(x_i + \alpha)}{(x_i!)\ \Gamma(\alpha)}$$

Thus the log-likelihood is:

$$l(\alpha) = \log P(X|\alpha) = \log(n!) + \log\Gamma(k\alpha) - \log\Gamma(n+k\alpha) + \sum_{i=1}^{k}(\log\Gamma(x_i+\alpha) - \log(x_i!) - \log\Gamma(\alpha))$$

Now we write a function that first estimate α by maximizing this log-likelihood, then use estimated $\hat{\alpha}$ to return the posterior mean of p, which is:

$$E(p_i|X,\hat{\alpha}) = \frac{\hat{\alpha}+x_i}{k\hat{\alpha}+\sum_{j=1}^{k}x_j} \quad (i=1,...,k)$$

EB function

```
## Empirical Bayes function for Multinomial probability vector p
## input: x=(x_1, ..., x_k)
## output: MLE prior parameter (alpha_hat); posterior mean of p


eb_multinomial_prob = function(x) {
    k = length(x)
    n = sum(x)

    # log-likelihood function
    log_Lik = function(alpha) {
        log(factorial(n)) + lgamma(k*alpha) - lgamma(n+k*alpha) +
        sum( lgamma(x+alpha) - log(factorial(x)) - lgamma(alpha) )

    }

    # find MLE parameter that maximize log-likelihood
    # constraint: alpha>0
    alpha_hat = optimize(log_Lik, interval=c(0,100), maximum=TRUE)$maximum

    # compute posterior mean
    pm = (alpha_hat + x)/(k*alpha_hat + sum(x))

    # output
    results = list(alpha.hat=alpha_hat, posterior.mean=pm)
    return(results)
}
```

Demonstration by simulation

```
set.seed(1)
library(MCMCpack)
```

```
## Warning: package 'MCMCpack' was built under R version 3.5.3
```

```
## Loading required package: coda
```

```
## Warning: package 'coda' was built under R version 3.5.3
```

```
## ##
## ## Markov Chain Monte Carlo Package (MCMCpack)
```

```
## ## Copyright (C) 2003-2019 Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park
## ##
## ## Support provided by the U.S. National Science Foundation
## ## (Grants SES-0350646 and SES-0350613)
## ##
```

```r
# data simulation
n = 100   #number of total trials
k = 10     #number of classes
alpha = 1
p = rdirichlet(1, alpha=rep(alpha,k))
x = as.vector(rmultinom(1, size=n, prob=p))

# Empirical Bayes
results = eb_multinomial_prob(x)
results
```

```
## $alpha.hat
## [1] 0.9885622
##
## $posterior.mean
##  [1] 0.01809666 0.25470632 0.17280297 0.09089963 0.14550186 0.12730111
##  [7] 0.12730111 0.01809666 0.01809666 0.02719703
```

```r
# accuary of estimates
data.frame( mle_MSE = sum((p-x/n)^2)/length(p),
            pm_MSE = sum((p-results$posterior.mean)^2)/length(p) )
```

```
##       mle_MSE       pm_MSE
## 1 0.000460156 0.0003349576
```

Comments:

The Empirical Bayes estimate of α (i.e. $\hat{\alpha} = 0.9885622$) is sensible compared with the true value of $\alpha = 1$.

And the MSE results show that the Empirical Bayes posterior means provide better accuracy (i.e. lower MSE errors) than the maximum likelihood estimates.

Problem D

(1) CV function for choosing opitmal bin number

```r
# data x on [0,1]
# bins are of equal width, spanning 0 to 1
# range of bin numbers: (lower, upper)

cv.hist.opt.bn = function(x, cv, lower, upper) {
    ave_cv_KL = rep(0, upper-lower+1)
    for (bn in lower:upper) {
        for (k in 1:cv) {
            valid = x[(1:length(x))%%cv==(k-1)]
            train = x[(1:length(x))%%cv!=(k-1)]

            estimate = rep(0, bn)
            for (i in 1:bn) {
```

```
                estimate[i] = sum(train>=(i-1)/bn & train<i/bn)/length(train) * bn
            }

            for (j in 1:length(valid)) {
                bin_inx = floor(valid[j] * bn) + 1
                if (estimate[bin_inx] == 0) {        # if no data, estimate is near 0
                      ave_cv_KL[bn-lower+1] = ave_cv_KL[bn-lower+1] -log(10^(-5))}
                else {ave_cv_KL[bn-lower+1] = ave_cv_KL[bn-lower+1] -log(estimate[bin_inx])}
            }
        }
        ave_cv_KL[bn-lower+1] = ave_cv_KL[bn-lower+1]/cv
    }
    opt_bin = which.min(ave_cv_KL) + lower -1
    min_KL = min(ave_cv_KL)
    results = list(opt.bin.num = opt_bin, KL.min=min_KL)
    return(results)
}
```

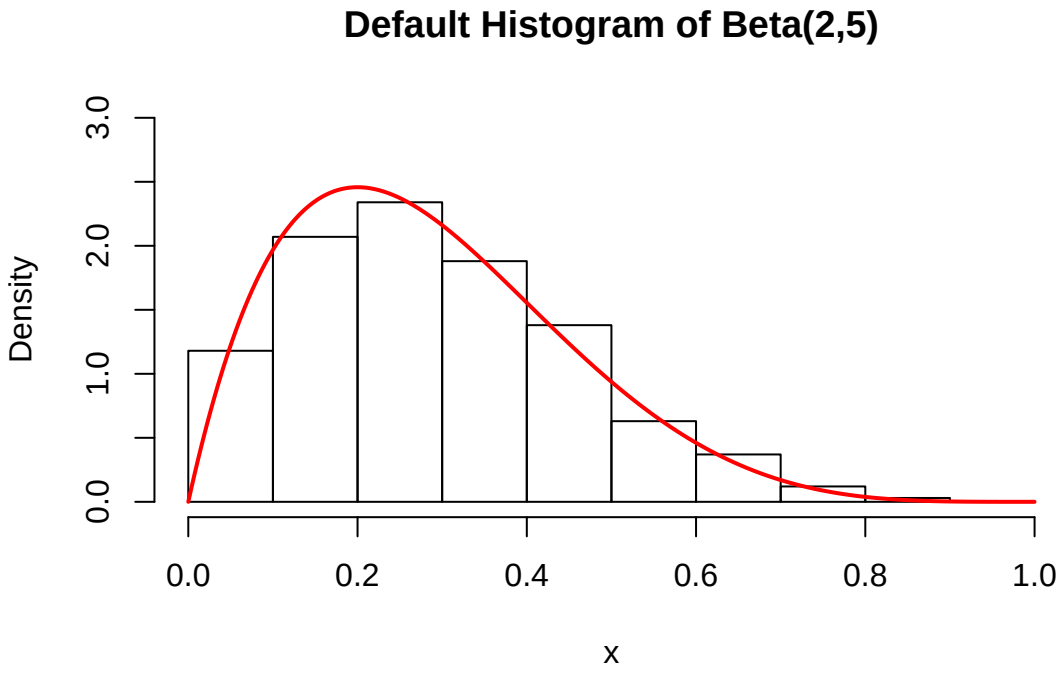
(2) Demonstration by simulation

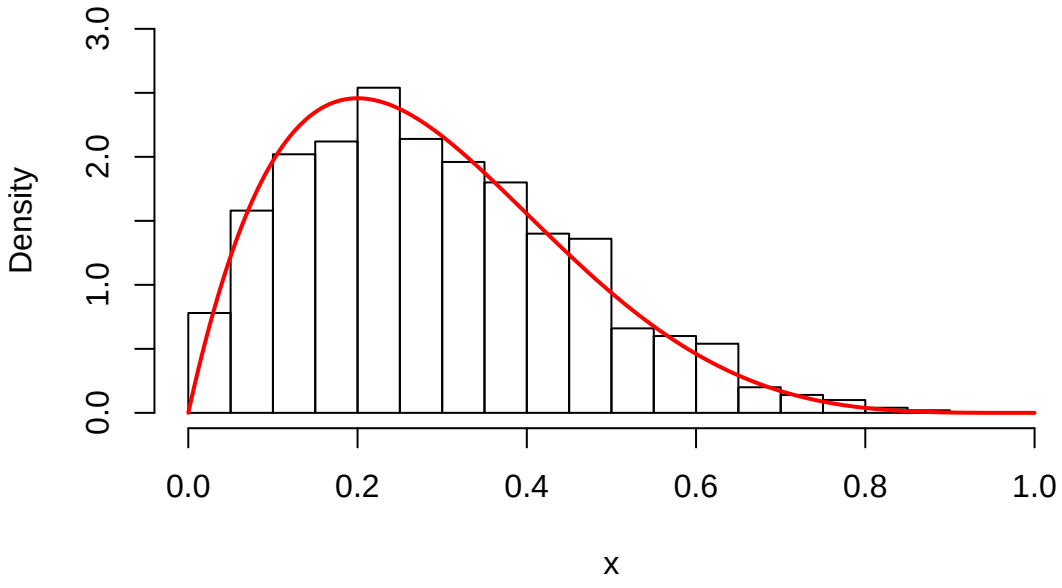
Dramatically changing density: Beta(2,5)

```
# simulation: dramatically changing density - Beta(2,5)
set.seed(100)
x = rbeta(1000,2,5)
beta.cv = cv.hist.opt.bn(x, cv=5, lower=1, upper=50)   # 5-fold CV
opt_bin = beta.cv$opt.bin.num
opt_bin
```

```
## [1] 28
```

```
# histogram plots: default vs optimal bin numer
par(mfrow=c(2,1))
hist(x, probability=TRUE, xlim=c(0,1), ylim=c(0,3), main="Default Histogram of Beta(2,5)")
lines(dbeta(seq(0,1,0.001),2,5)~seq(0,1,0.001), col=2, lwd=2)
hist(x, nclass=opt_bin, probability=TRUE, xlim=c(0,1), ylim=c(0,3),
     main=paste0("Histogram of Beta(2,5) with optimal bin number = ", opt_bin))
lines(dbeta(seq(0,1,0.001),2,5)~seq(0,1,0.001), col=2, lwd=2)
```





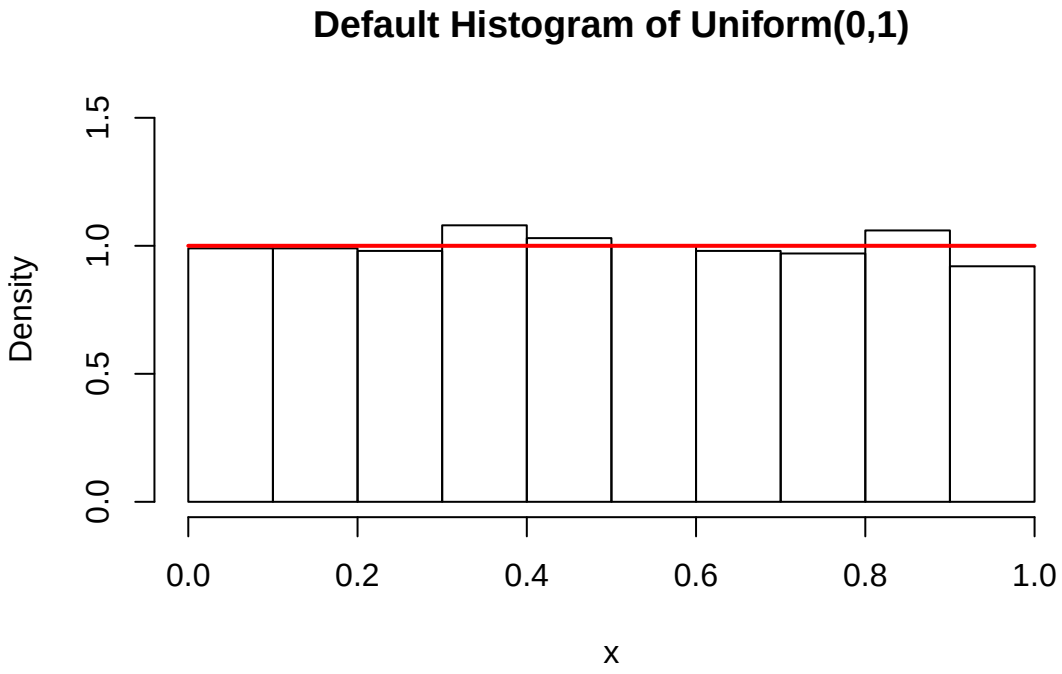
Uniform density: Uniform(0,1)

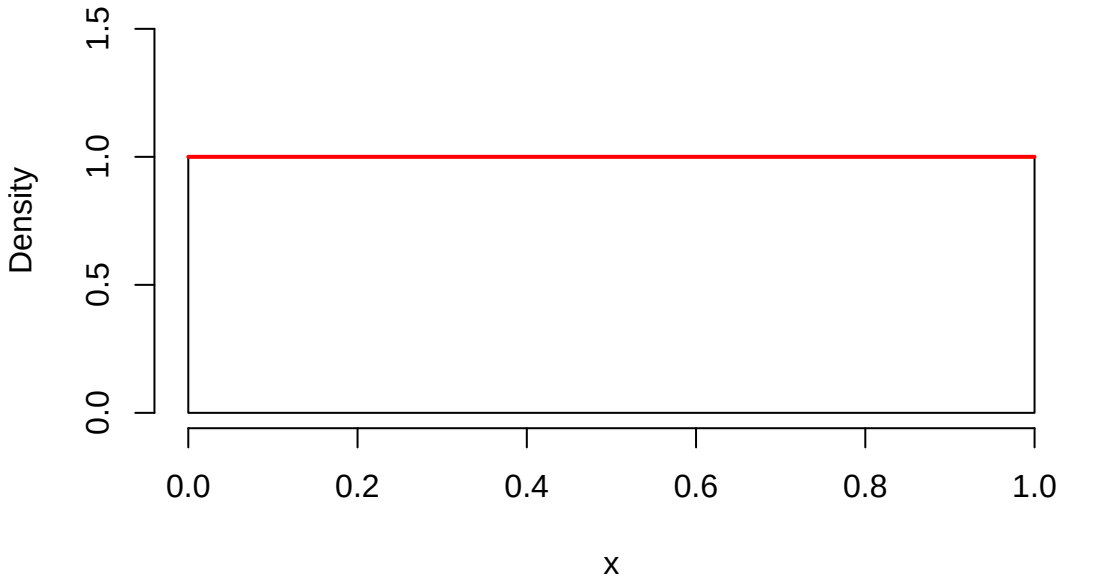
```
# simulation: uniform density - Uniform(0,1)
set.seed(123)
```

```r
x = runif(1000,0,1)
unif.cv = cv.hist.opt.bn(x, cv=5, lower=1, upper=50)   # 5-fold CV
opt_bin = unif.cv$opt.bin.num
opt_bin
```

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

```r
# histogram plots: default vs optimal bin numer
par(mfrow=c(2,1))
hist(x, probability=TRUE, ylim=c(0,1.5), main="Default Histogram of Uniform(0,1)")
lines(dunif(seq(0,1,0.001),0,1)~seq(0,1,0.001), col=2, lwd=2)
hist(x, nclass=opt_bin, probability=TRUE, ylim=c(0,1.5),
     main=paste0("Histogram of Uniform(0,1) with optimal bin number = ", opt_bin))
lines(dunif(seq(0,1,0.001),0,1)~seq(0,1,0.001), col=2, lwd=2)
```





Comments:

(1) Dramatically changing density: Beta(2,5)

In this simulation, the optimal bin number provided by the 5-fold cross-validation is larger than that of the default bin number provided by the hist() function in R. And the default function seems to provide a better

fit of data.

(2) Uniform density: Uniform(0,1)

In this simulation, the optimal bin number provided by the 5-fold cross-validation is smaller than that of the default bin number provided by the hist() function in R. Here the CV method provides a better fit of data.

(3) Empirical Bayes estimate

(i) EB-CV function for choosing opitmal bin number

```
# data x on [0,1]
# bins are of equal width, spanning 0 to 1
# range of bin numbers: (lower, upper)

# NOTE: here I directly used the function from Problem C

eb.cv.hist.opt.bn = function(x, cv, lower, upper) {
    ave_cv_KL = rep(0, upper-lower+1)
    for (bn in lower:upper) {
        for (k in 1:cv) {
            valid = x[(1:length(x))%%cv==(k-1)]
            train = x[(1:length(x))%%cv!=(k-1)]

            data = rep(0, bn)
            for (i in 1:bn) {
                data[i] = sum(train>=(i-1)/bn & train<i/bn)
            }
            estimate = eb_multinomial_prob(data)$posterior.mean * bn

            for (j in 1:length(valid)) {
                bin_inx = floor(valid[j] * bn) + 1
                if (estimate[bin_inx] == 0) {        # if no data, estimate is near 0
                      ave_cv_KL[bn-lower+1] = ave_cv_KL[bn-lower+1] -log(10^(-5))}
                else {ave_cv_KL[bn-lower+1] = ave_cv_KL[bn-lower+1] -log(estimate[bin_inx])}
            }
        }
        ave_cv_KL[bn-lower+1] = ave_cv_KL[bn-lower+1]/cv
    }
    opt_bin = which.min(ave_cv_KL) + lower -1
    min_KL = min(ave_cv_KL)
    results = list(opt.bin.num = opt_bin, KL.min=min_KL)
    return(results)
}
```

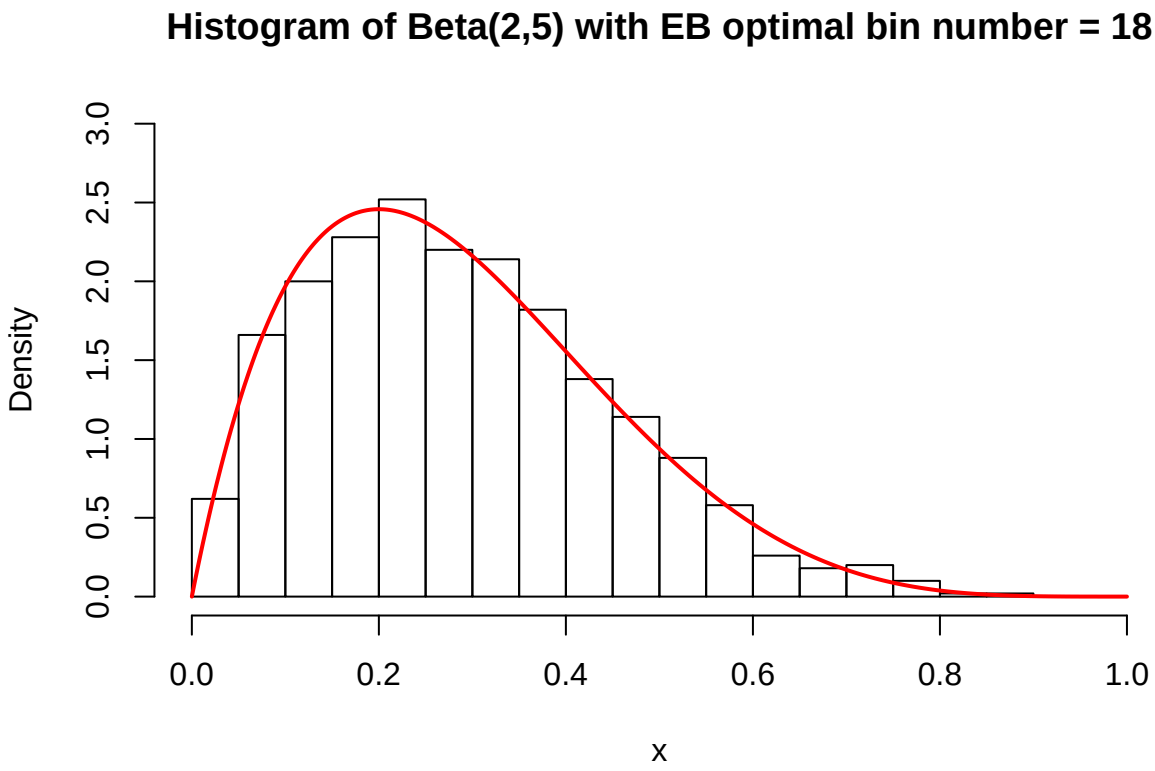
(ii) Demonstration by simulation

Dramatically changing density: Beta(2,5)

```
# simulation: dramatically changing density - Beta(2,5)
set.seed(100)
x = rbeta(1000,2,5)
beta.cv_EB = eb.cv.hist.opt.bn(x, cv=5, lower=1, upper=50)   # 5-fold CV
opt_bin = beta.cv_EB$opt.bin.num
opt_bin
```

```
## [1] 18
```

```
# histogram plot: optimal bin numer
hist(x, nclass=opt_bin, probability=TRUE, xlim=c(0,1), ylim=c(0,3),
     main=paste0("Histogram of Beta(2,5) with EB optimal bin number = ", opt_bin))
lines(dbeta(seq(0,1,0.001),2,5)~seq(0,1,0.001), col=2, lwd=2)
```

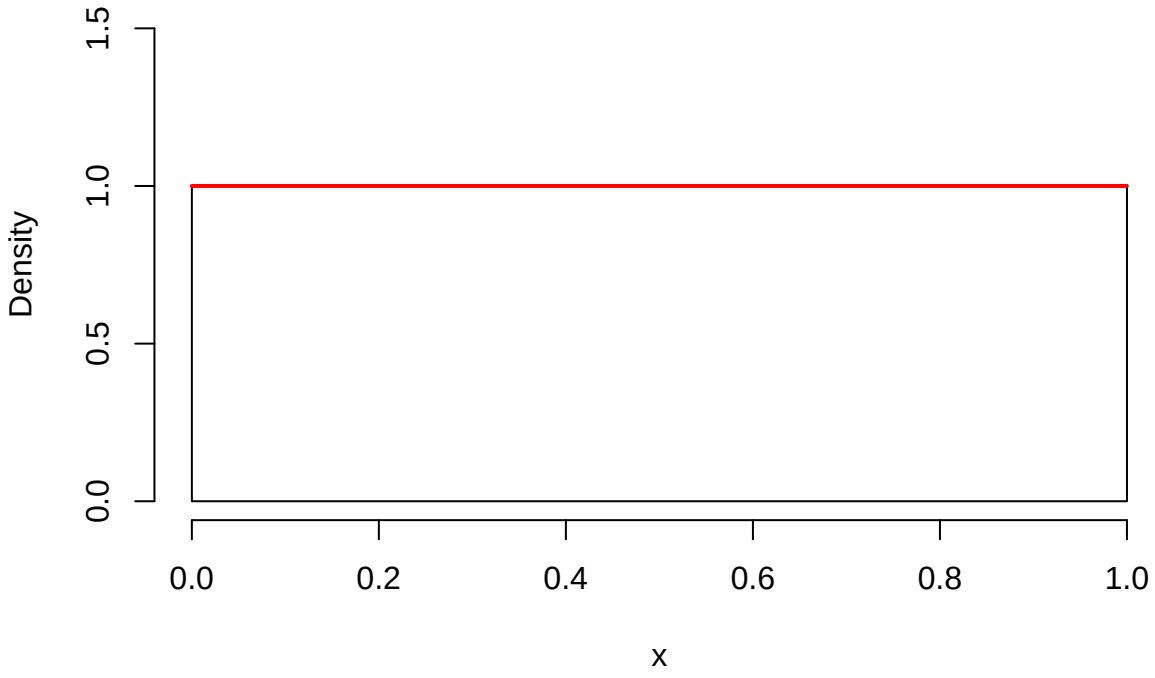


Uniform density: Uniform(0,1)

```
# simulation: uniform density - Uniform(0,1)
set.seed(123)
x = runif(1000,0,1)
unif.cv_EB = eb.cv.hist.opt.bn(x, cv=5, lower=1, upper=50)   # 5-fold CV
opt_bin = unif.cv_EB$opt.bin.num
opt_bin
```

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

```
# histogram plot: optimal bin numer
hist(x, nclass=opt_bin, probability=TRUE, ylim=c(0,1.5),
     main=paste0("Histogram of Uniform(0,1) with EB optimal bin number = ", opt_bin))
lines(dunif(seq(0,1,0.001),0,1)~seq(0,1,0.001), col=2, lwd=2)
```



Compre KL-loss

```
# compare the min KL loss of cv experiments
c(beta.cv$KL.min, beta.cv_EB$KL.min)
```

```
## [1] -87.00602 -62.91891
```

```
c(unif.cv$KL.min, unif.cv_EB$KL.min)
```

```
## [1] 0 0
```

Comments:

(1) Dramatically changing density: Beta(2,5)

In this simulation, the optimal bin number provided by the 5-fold cross-validation using Epirical Bayes estimation is smaller than that of the CV method in question (2). And the Epirical Bayes estimation seems to provide a better fit of data than the CV method here. Also, the minimal average KL-loss of 5-fold CV agrees with this conclusion.

(2) Uniform density: Uniform(0,1)

In this simulation, the optimal bin number provided by the 5-fold cross-validation using Epirical Bayes estimation is the same as that of the CV method in question (2). Therefore, the two methods have the same performance here. Again, the minimal average KL-loss of 5-fold CV agrees with this conclusion.