HW3

Jinhong Du - 12243476

2020/04/29

Contents

A: Non-Parametric Smoothing (1)

- Start by downloading and running the code for the cell cycle example in https://github.com/stephens999/stat34800/blob/master/analysis/cell_cycle.Rmd. You may need to install some packages etc…
- From the examples in the code (and, if desired, using further investigation of your own) settle on one version of the wavelet smoothing, and one version of the trend filtering. Try to select what you think will be the best, or at least a satisfactory, version of each.
- Then, using some kind of cross-validation based approach, compare the accuracy of these two methods on the data from 10 genes that are provided in the repository. You are free to define your own measure of accuracy, but you should provide a report of what you did, and explain which method did best. Include plots that compare the results for both methods for all 10 genes.
- Return now to our original goal. The question we want to answer is this: which genes show greatest evidence for varying in their expression through the cell cycle? For now we are not worried about “statistical significance” - it is enough to rank the genes. Using the results from the previous part, and some reasonable numeric criteria which you should explain, provide a ranking of the genes from the strongest evidence to weakest evidence. Check that your ranking by a numeric criteria seems reasonable (or, at least, not unreasonable) based on a visual assessment of the plots you made in the previous part.

(1) Examples

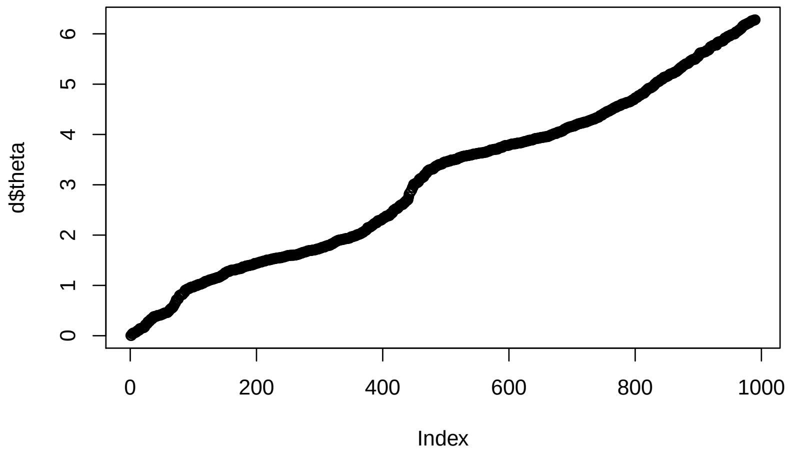
The data come from a recent experiment performed in the Gilad lab by Po Tung, in collaboration with Joyce Hsiao and others. The data are measuring the activity of 10 genes that may or may not be involved in the “cell cycle”, which is the process cells go through as they divide. (We have data on a large number of genes, but Joyce has picked out 10 of them for us to look at.) Each gene is measured in many single cells, and we have some independent (but noisy) measurement of where each cell is in the cell cycle.

```r
d = readRDS("cyclegenes.rds")
dim(d)
```

```
## [1] 990  11
```

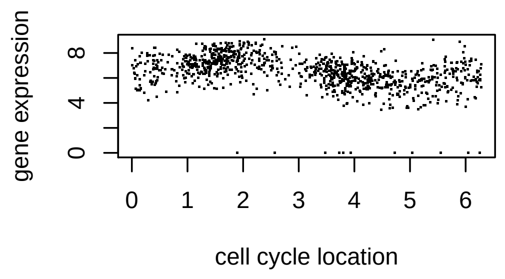
Here each row is a single cell. The first column (“theta”) is an estimate of where that cell is in the cell cycle, from 0 to 2pi. (Note that we don’t know what stage of the cell cycle each point in the interval corresponds to - so there is no guarantee that 0 is the “start” of the cell cycle. Also, because of the way these data were created we don’t know which direction the cell cycle is going - it could be forward or backward.) Then there are 10 columns corresponding to 10 different genes.

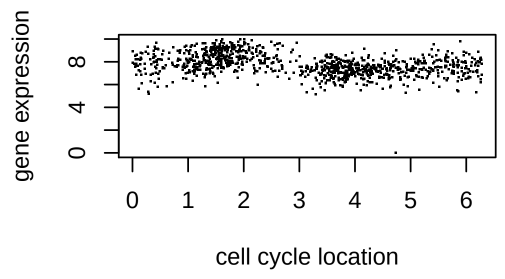
```r
# order the data by cell cycle
o = order(d[,1])
d = d[o,]
plot(d$theta)
```

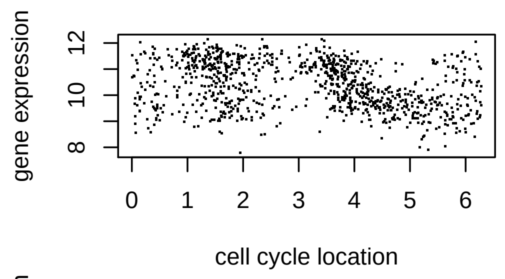


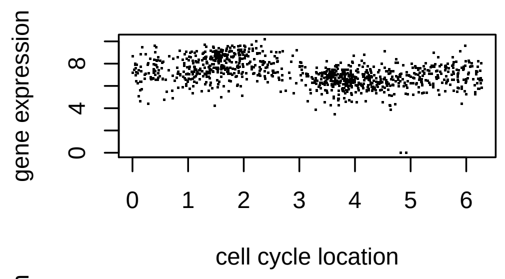
Here we just plot 8 genes to get a sense for the data:

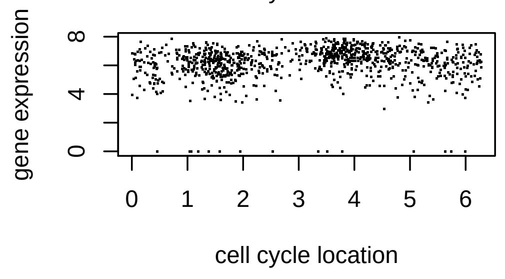
```r
par(mfcol=c(2,2))
for(i in 1:4){
  plot(d$theta, d[,(i+1)],pch=".",ylab="gene expression",xlab="cell cycle location")
}
par(mfcol=c(2,2))
for(i in 1:4){
  plot(d$theta, d[,(i+5)],pch=".",ylab="gene expression",xlab="cell cycle location")
}
```

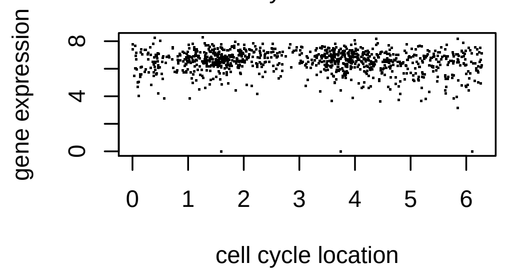


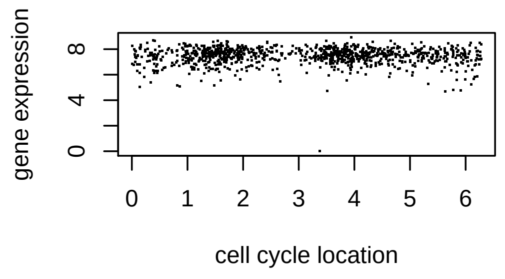


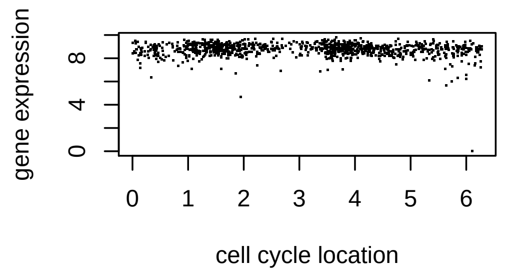










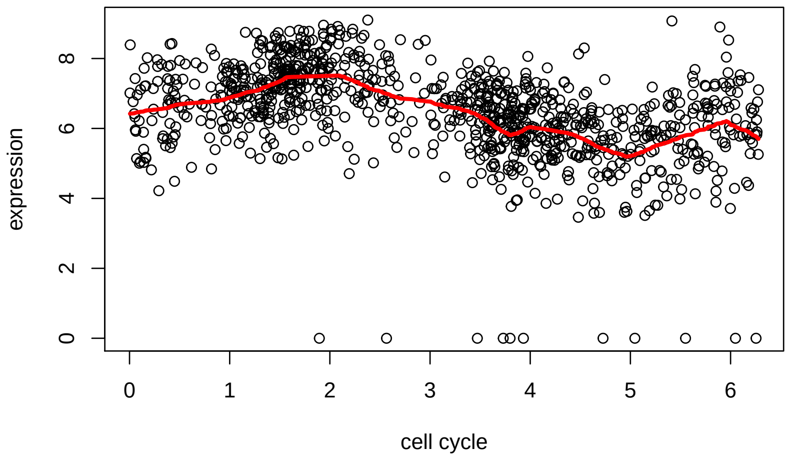


The question we want to answer is this: which genes show greatest evidence for varying in their expression through the cell cycle? For now what we really want is a filter we can apply to a large number of genes to pick out the ones that look most “interesting”. Later we might want a more formal statistical measure of evidence.

Our current idea is to try “smoothing” these data, and pick out genes where the change in the mean over theta is most variable (in some sense). The extreme would be if the smoother fits a horizontal line - that indicates no variability with theta, so those genes are not interesting to us.

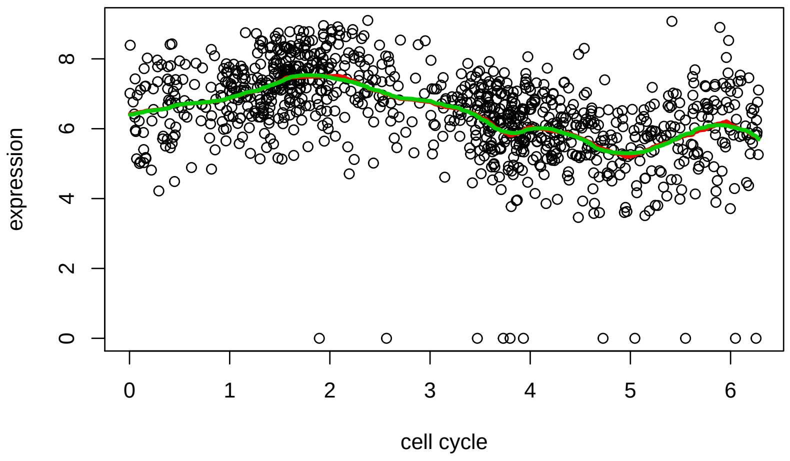
(a) Trend filtering. Here we will apply trend filtering to smooth these data. Trend filtering, at its simplest, applies L_1 regularization to the changes in mean from one observation to the next. (The extreme would be no changes in any of these means, so a flat line.)

```r
library(genlasso)
d2.tf = trendfilter(d[,2],ord = 1)
d2.tf.cv = cv.trendfilter(d2.tf) # performs 5-fold CV
plot(d[,1],d[,2],xlab="cell cycle",ylab="expression")
lines(d[,1],predict(d2.tf, d2.tf.cv$lambda.min)$fit,col=2,lwd=3)
```



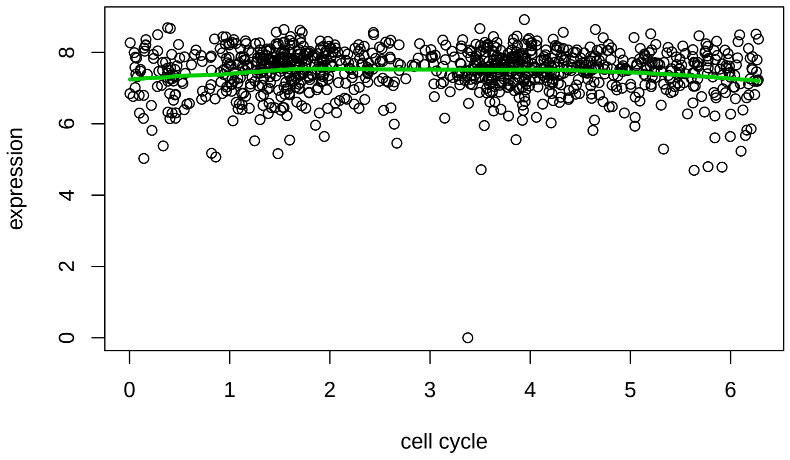
This fit is a bit on the “spiky” side in places. We can get a smoother fit by using a higher order filter. Basically instead of shrinking first order differences it shrinks things that also measure differences with neighbors a bit further apart (2nd order).

```r
d2.tf2 = trendfilter(d[,2],ord = 2)
d2.tf2.cv = cv.trendfilter(d2.tf2) # performs 5-fold CV
plot(d[,1],d[,2],xlab="cell cycle",ylab="expression")
lines(d[,1],predict(d2.tf, d2.tf.cv$lambda.min)$fit,col=2,lwd=3)
lines(d[,1],predict(d2.tf2, d2.tf2.cv$lambda.min)$fit,col=3,lwd=3)
```



And here we try another gene that maybe shows less evidence for variability.

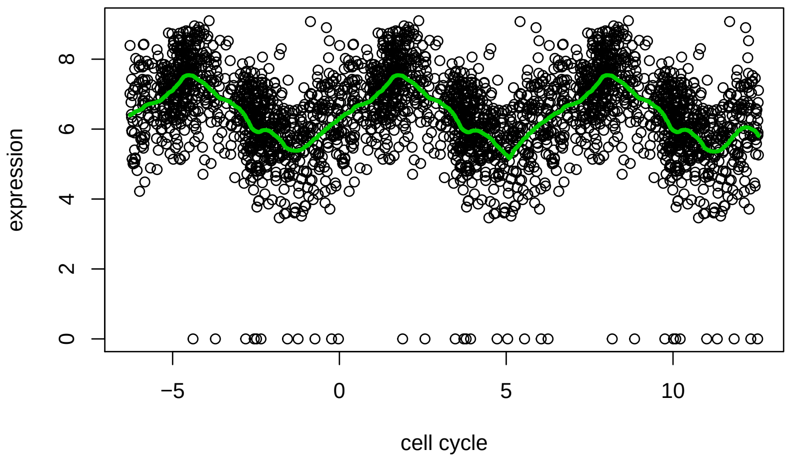
```r
d7.tf2 = trendfilter(d[,7],ord = 2)
d7.tf2.cv = cv.trendfilter(d7.tf2) # performs 5-fold CV
plot(d[,1],d[,7],xlab="cell cycle",ylab="expression")
lines(d[,1],predict(d7.tf2, d7.tf2.cv$lambda.min)$fit,col=3,lwd=3)
```



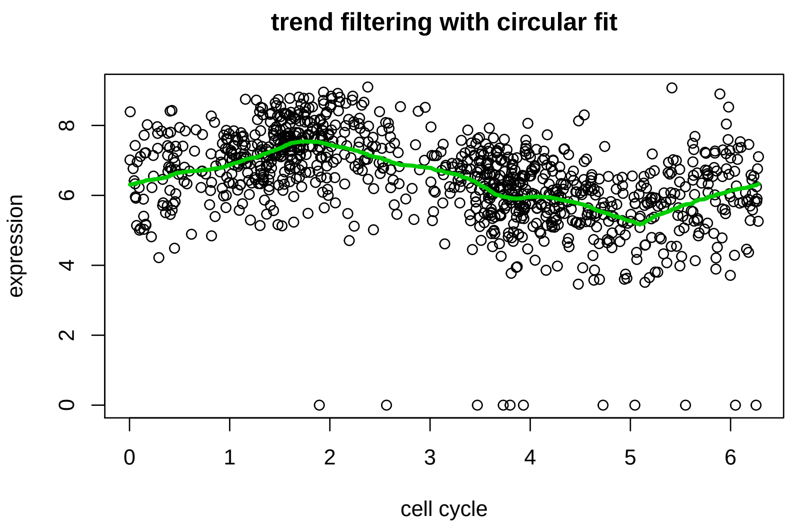
Because the x axis here is cyclical, the value of $\mathbb{E}(Y|x)$ near $x = 0$ should be similar to the value near $x = 2\pi$. But trend filtering does not know this. We can encourage this behaviour by duplicating the data using a translation. (Note this is different than reflecting it about the boundaries).

Here is an example:

```r
yy = c(d[,2],d[,2],d[,2]) ## duplicated data
xx = c(d[,1]-2*pi, d[,1], d[,1]+2*pi) # shifted/translated x coordinates

yy.tf2 = trendfilter(yy,ord = 2)
yy.tf2.cv = cv.trendfilter(yy.tf2) # performs 5-fold CV
plot(xx,yy,xlab="cell cycle",ylab="expression")
lines(xx,predict(yy.tf2, yy.tf2.cv$lambda.min)$fit,col=3,lwd=3)
```



```r
# plot only a single version of data
include = c(rep(FALSE,length(d[,2])), rep(TRUE, length(d[,2])), rep(FALSE, length(d[,2])))
plot(xx[include],yy[include],xlab="cell cycle",ylab="expression",
     main="trend filtering with circular fit")
lines(xx[include],predict(yy.tf2, yy.tf2.cv$lambda.min)$fit[include],col=3,lwd=3)
```

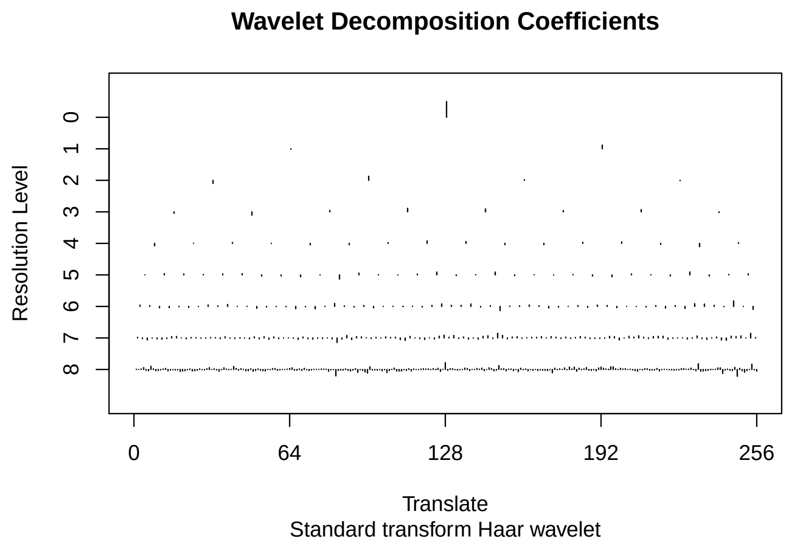


(b) Wavelets. Here we will apply wavelets to smooth these data. To apply wavelets we need the data to be a power of 2. Also we need the data to be ordered in terms of theta. We'll subset the data to 512 elements here, and order it:

```r
# subset the data
set.seed(1)
subset = sort(sample(1:nrow(d),512,replace=FALSE))
d.sub = d[subset,]
```

Here we do the Haar wavelet by specifying `family="DaubExPhase",filter.number = 1` to the discrete wavelet transform function `wd`. The plot shows the wavelet transformed values, separately at each resolution.

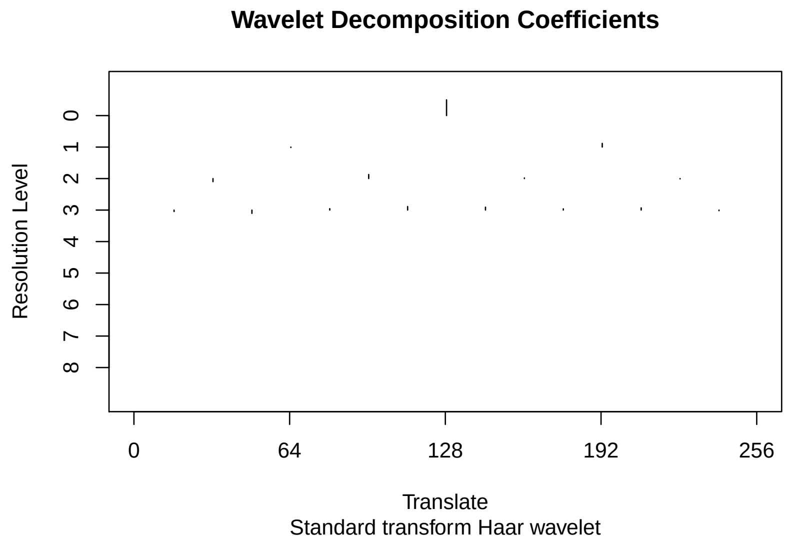
```r
library("wavethresh")
wds <- wd(d.sub[,2],family="DaubExPhase",filter.number = 1)
plot(wds)
```



```
## [1] 12.31629 12.31629 12.31629 12.31629 12.31629 12.31629 12.31629 12.31629
## [9] 12.31629
```

To illustrate the idea behind wavelet shrinkage we use the policy "manual" to shrink all the high-resolution coefficients (levels 4-8) to 0.

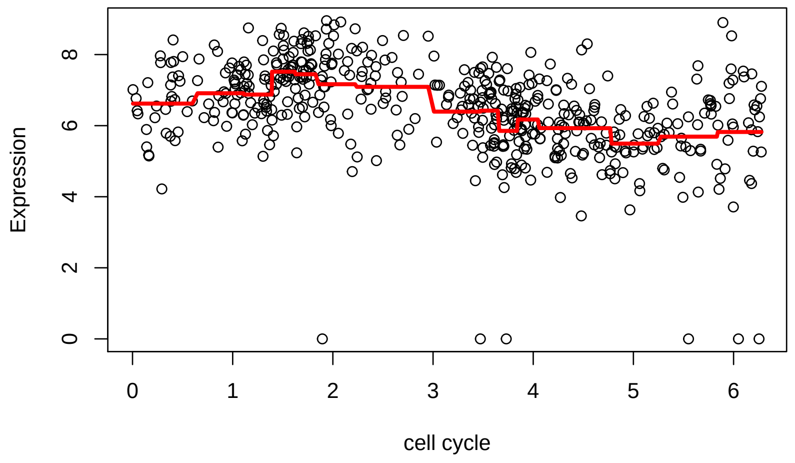
```r
wtd <- threshold(wds, levels = 4:8,  policy="manual",value = 99999)
plot(wtd)
```



```
## [1] 12.31629 12.31629 12.31629 12.31629 12.31629 12.31629 12.31629 12.31629
## [9] 12.31629
```

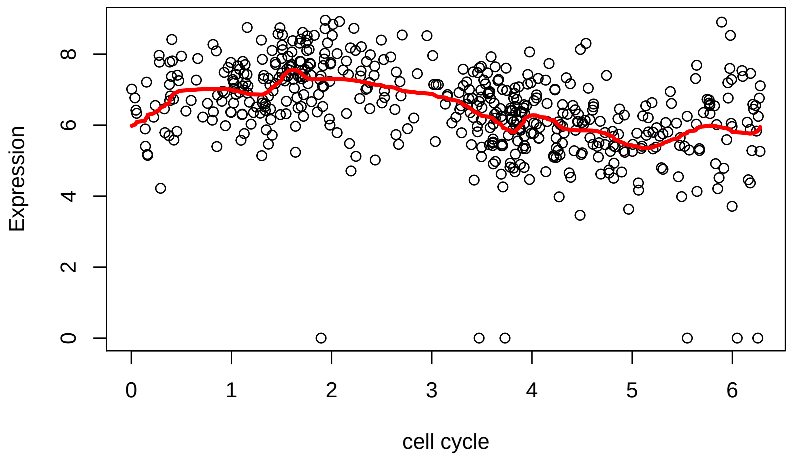
Now undo the wavelet transform on the shrunken coefficients

```r
fd <- wr(wtd) #reconstruct
plot(d.sub$theta,d.sub[,2],xlab="cell cycle", ylab = "Expression")
lines(d.sub$theta,fd,col=2,lwd=3)
```



The estimate here is a bit “jumpy”, due to the use of the Haar wavelet and the rather naive hard thresholding. We can make it less “jumpy” by using a “less step-wise” wavelet basis

```r
wds <- wd(d.sub[,2],family="DaubLeAsymm",filter.number = 8)
wtd <- threshold(wds, levels = 4:8,  policy="manual",value = 99999)
fd <- wr(wtd) #reconstruct
plot(d.sub$theta,d.sub[,2],xlab="cell cycle", ylab = "Expression")
lines(d.sub$theta,fd,col=2,lwd=3)
```



Note that the default in this particular package (`wavethresh`) 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. It is taken care of.

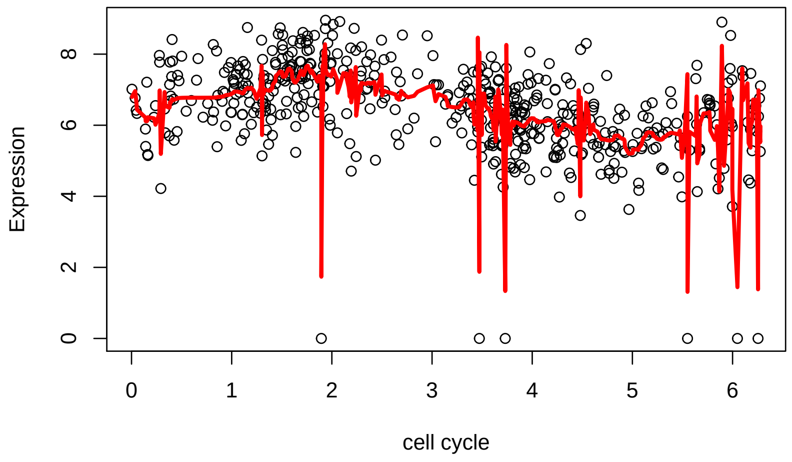
(c) Empirical Bayes thresholding rule. What happens if we use an Empirical Bayes thresholding rule?

The EbayesThresh package essentially solves the EBNM problem, using a prior distribution that is a mixture of a point mass at 0 and a Laplace distribution with rate parameter a:

$$\pi_0\delta_0 + (1-\pi_0)DExp(a).$$

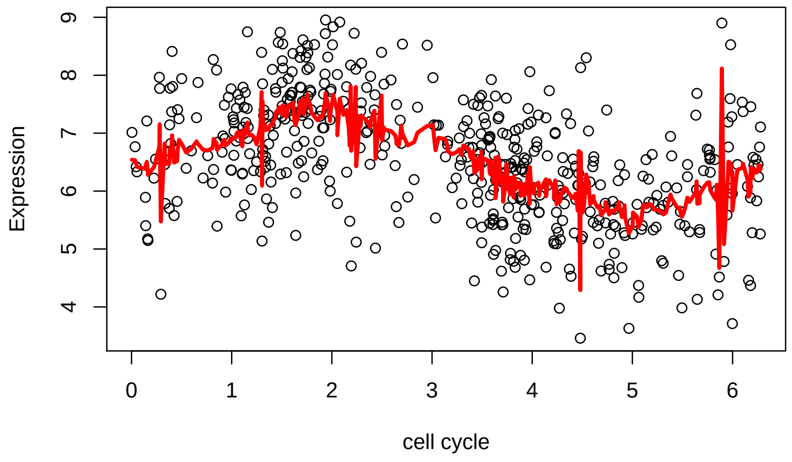
Here `a=NA` tells Ebayesthresh ot estimate a. Notice that the outliers cause "problems"

```
library("EbayesThresh")
wds <- wd(d.sub[,2],family="DaubLeAsymm",filter.number = 8)
wtd <- ebayesthresh.wavelet(wds,a=NA)
fd <- wr(wtd) #reconstruct
plot(d.sub$theta,d.sub[,2],xlab="cell cycle", ylab = "Expression")
lines(d.sub$theta,fd,col=2,lwd=3)
```



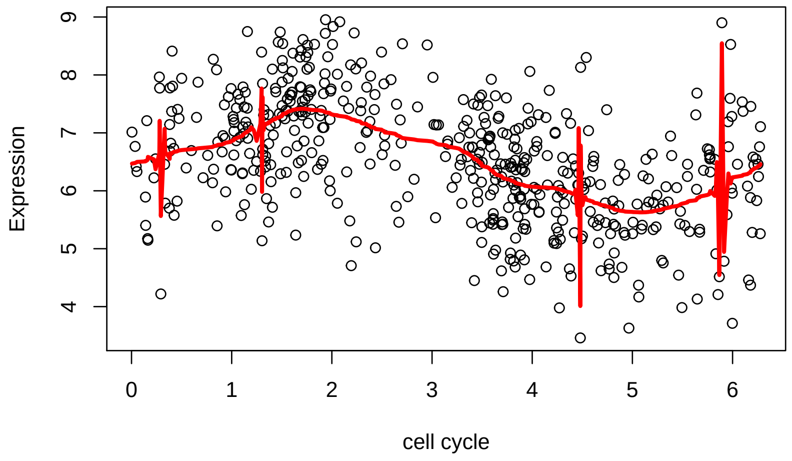
Try removing the outliers (actually setting them to the mean here), just to see what happens.

```
xx = ifelse(d.sub[,2]<2,mean(d.sub[,2]),d.sub[,2])
wds <- wd(xx,family="DaubLeAsymm",filter.number = 8)
wtd <- ebayesthresh.wavelet(wds,a=NA)
fd <- wr(wtd) #reconstruct
plot(d.sub$theta,xx,xlab="cell cycle", ylab = "Expression")
lines(d.sub$theta,fd,col=2,lwd=3)
```



Notice that estimating a is really important (the default is to set $a = 0.5$):

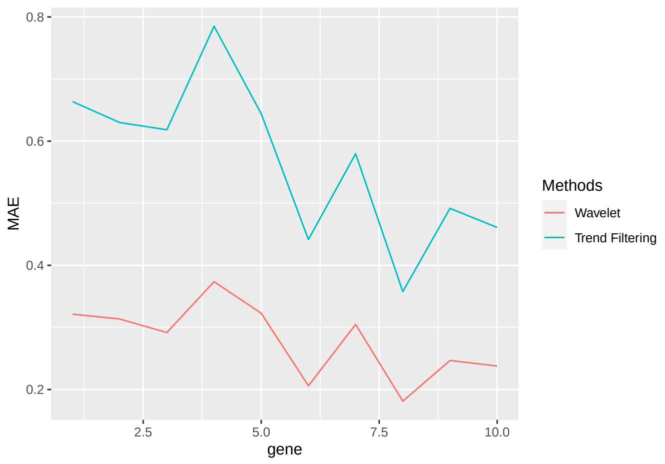
```r
xx = ifelse(d.sub[,2]<2,mean(d.sub[,2]),d.sub[,2])
wds <- wd(xx,family="DaubLeAsymm",filter.number = 8)
wtd <- ebayesthresh.wavelet(wds)
fd <- wr(wtd) #reconstruct
plot(d.sub$theta,xx,xlab="cell cycle", ylab = "Expression")
lines(d.sub$theta,fd,col=2,lwd=3)
```



(2) Comparison of Two Methods

I choose the second-order trend filtering model and the Wavelets with `filter_number=8`.

```r
set.seed(1)
d <- d[!apply(d[,2:11]==0,1,any),]
subset <- sort(sample(1:nrow(d),512,replace=FALSE))
d.sub <- d[subset,]
MAE <- array(0, c(2,10))
for(i in c(1:10)){
    for(fold in 1:2){
        tf = trendfilter(d.sub[seq(fold,512,2),i+1],ord = 2)
        tf.cv = cv.trendfilter(tf)
        wds <- wd(d.sub[seq(fold,512,2),i+1],family="DaubLeAsymm",filter.number = 8)
        wtd <- threshold(wds, levels = 4:7, policy="manual",value = 99999)
        MAE[1,i] <- MAE[1,i]+mean(abs(d.sub[seq(3-fold,512,2),i+1]-
                predict(tf, tf.cv$lambda.min)$fit))
        MAE[2,i] <- mean(abs(d.sub[seq(3-fold,512,2),i+1] - wr(wtd)))
    }
}
MAE <- MAE/2
```



```r
cat(mean(MAE[1,]), mean(MAE[2,]))
```

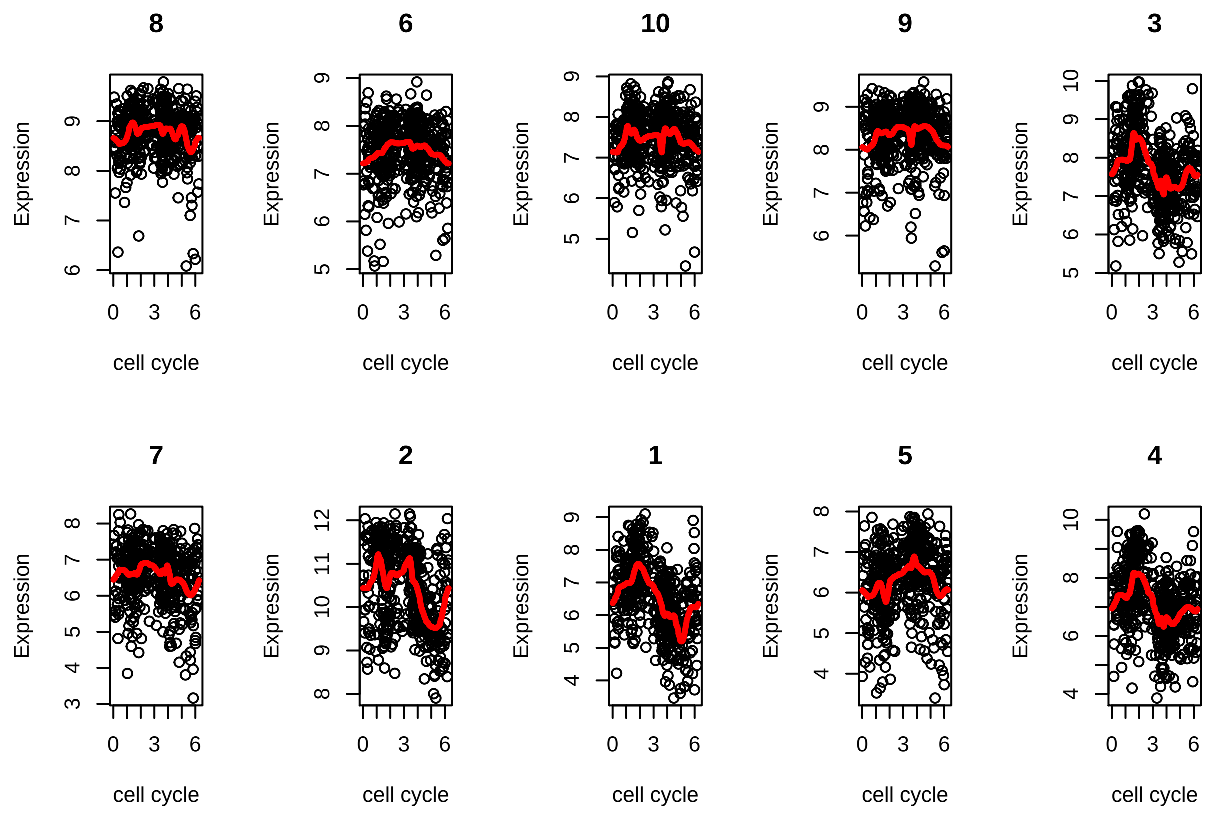
```
## 0.554885 0.5498338
```

I choose MAE as the accuracy measure as I don't want to add to much loss to outliers. The cross-validation is done by split the data $\{x_1,\ldots,x_{512}\}$ into two parts $\{x_1,x_3,\ldots,x_{511}\}$ and $\{x_2,x_4,\ldots,x_{512}\}$. Then we use each part to get a estimated line and compare it with the other part. A good model should have similar behaviors for these two folds.

As we can see, the performances of the wavelet model performs better than the trend filtering model at each gene.

(3) Gene Ranking

```r
idx <- sort(MAE[2,],index.return = TRUE)$ix
par(mfrow=c(2,5))
for(i in idx){
    wds <- wd(d.sub[,i+1],family="DaubLeAsymm",filter.number = 8)
    wtd <- threshold(wds, levels = 4:8, policy="manual",value = 99999)
    fd <- wr(wtd) #reconstruct
    plot(d.sub$theta,d.sub[,i+1],xlab="cell cycle", ylab = "Expression", main=paste(i))
    lines(d.sub$theta,fd,col=2,lwd=3)
}
```

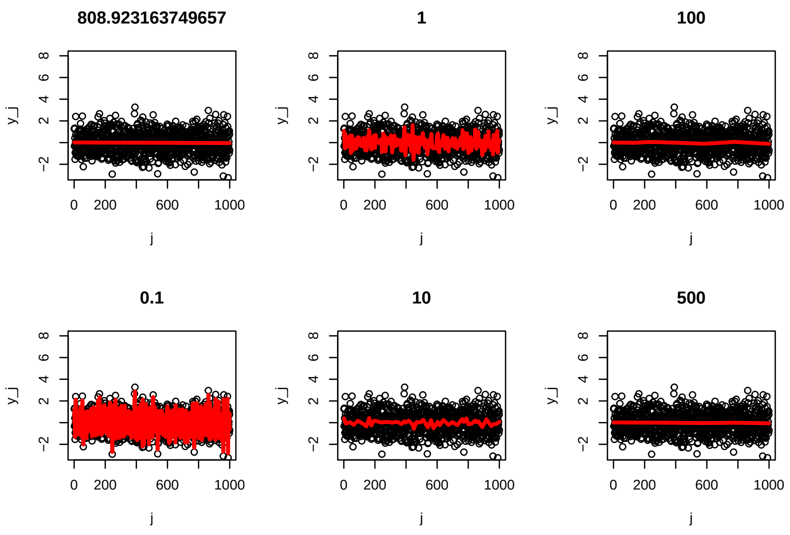


We use the MAE calculated in part (2) to measure how much a gene varies in different cell cycles. As we can see, the first few plots with lower MAEs look like smoother than the last few plots with higher MAEs. This means that this numeric criteria seems reasonable based on a visual assessment.

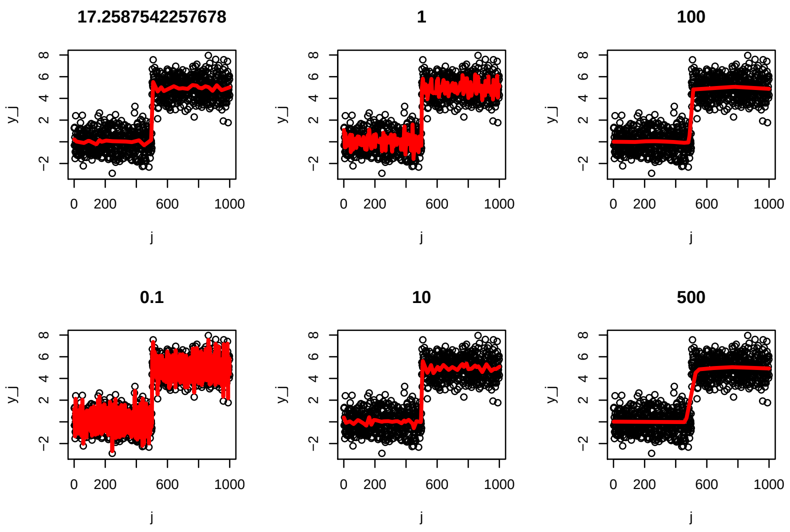
B: Non-Parametric Smoothing (2)

- This question aims to get some intuition into trend filtering through simple simulations. It arises from me thinking "what circumstances might reduce the performance of trend filtering?". This is usually a good question to ask before applying a method!
- 1. Constant mean case
    - Simulate some data with $y = \mu + e$ where: `mu = rep(0,1000)`, $e_j \sim N(0,1)$ independently $j = 1, \dots, 1000$.
    - Apply trend filtering to estimate the mean vector μ. Plot the data with the estimate of μ overlaid.
- 2. Step function
    - repeat 1, but with `mu = c(rep(0,500), rep(5,500))`. That is, the mean vector is a step function.
- You should find that in the second case (2) trend filtering finds the big change in mean accurately, but the estimate of μ in the regions where it is not changing becomes less smooth. (If this does not happen, try another random number seed.) Can you explain why?
    - Hint: you might get more insight by examining the results for multiple values of λ in each case, not only the "optimal" λ chosen by CV.

```r
set.seed(0)
mu <- rep(0,1000)
e <- rnorm(1000)
y <- mu + e

library(genlasso)
tf1 = trendfilter(y,ord = 1)
tf1.cv = cv.trendfilter(tf1) # performs 5-fold CV

lam <- c(tf1.cv$lambda.min, 0.1, 1, 10, 100, 500)
par(mfcol=c(2,3))
for(i in 1:6){
    plot(c(1:1000),y,xlab="j",ylab="y_j",main=paste(lam[i]),ylim=c(-3,8))
    lines(c(1:1000),predict(tf1, lam[i])$fit,col=2,lwd=3)
}
```



```r
set.seed(0)
mu <- c(rep(0,500), rep(5,500))
e <- rnorm(1000)
y <- mu + e

tf2 = trendfilter(y,ord = 1)
tf2.cv = cv.trendfilter(tf2) # performs 5-fold CV

lam <- c(tf2.cv$lambda.min, 0.1, 1, 10, 100, 500)
par(mfcol=c(2,3))
for(i in 1:6){
    plot(c(1:1000),y,xlab="j",ylab="y_j",main=paste(lam[i]),ylim=c(-3,8))
    lines(c(1:1000),predict(tf2, lam[i])$fit,col=2,lwd=3)
}
```



As we can see,

- As λ increases, the fitted line is much smoother for each case.
- The trend filtering captures the jump for the step function, while the estimate of μ in the regions where it is not changing becomes less smooth. For example, for the optimal value of λ chosen by CV, the fitted line for the constant function is almost constant, while the one for the step function has some fluctuations.

C: Bayesian Calculations: Dirichlet multinomial

- This goal here is to introduce you to the conjugate analysis for multinomial data. It might help to begin by refreshing your memory about the beta-binomial result, and then read an introduction to the Dirichlet distribution, which is a multivariate generalization of the beta distribution. I've also included a link to the multinomial distribution if you need a refresher on that.
  - https://stephens999.github.io/fiveMinuteStats/bayes_beta_binomial.html
  - https://stephens999.github.io/fiveMinuteStats/dirichlet.html
  - https://en.wikipedia.org/wiki/Multinomial_distribution
- Let $X = (X_1, \dots, X_k) \sim \text{Mult}(n, p)$ be a vector of multinomially-distributed count data on k classes, where n is a fixed and known sample size, and $p = (p_1, \dots, p_k)$ is to be estimated. (So $X_1 + \cdots + X_k = n$ and $p_1 + \cdots + p_k = 1$).
- Assume that the prior distribution for p is Dirichlet$(\alpha_1, \dots, \alpha_k)$. Show that the posterior distribution for p is also Dirichlet, and find its posterior mean.
- Now assume the special case $\alpha_1 = \alpha_2 = \cdots = \alpha_k$, and all are equal to α say. Write a function that, given data $X_1, \dots, X_k$, first estimates α by maximum likelihood (you will need to use a numerical optimizer for this), and then returns the posterior mean of p for that value of α. (Essentially this is an Empirical Bayes approach to estimating p.)
  - Hint: the marginal distribution of X given (n, α) (integrating out the vector p) is known as the "Dirichlet-multinomial" distribution. You can read about it here: https://en.wikipedia.org/wiki/Dirichlet-multinomial_distribution, where a formula for the probability mass function $p(X|\alpha)$ is given.

(1) Posterior Distribution of p

Since $\boldsymbol{X}|\boldsymbol{p} \sim \text{Multi}(n, \boldsymbol{p})$ and $\boldsymbol{p} \sim \text{Dirichlet}(\alpha_1, \dots, \alpha_k)$, we have

$$f(\boldsymbol{X}|n, \boldsymbol{p}) = \frac{n!}{X_1! \cdots X_k!} \prod_{j=1}^k p_j^{X_j}, \quad \sum_{j=1}^k X_j = n$$
$$f(\boldsymbol{p}|\boldsymbol{\alpha}) = \frac{\Gamma(\alpha_1 + \cdots + \alpha_k)}{\Gamma(\alpha_1) \cdots \Gamma(\alpha_k)} \prod_{j=1}^k p_j^{\alpha_j - 1}, \quad p_j \geq 0, \sum_{j=1}^k p_j = 1$$

So

$$\begin{aligned} f(\boldsymbol{p}|\boldsymbol{X}, n, \boldsymbol{\alpha}) &\propto f(\boldsymbol{X}|n, \boldsymbol{p}) f(\boldsymbol{p}|\boldsymbol{\alpha}) \\ &\propto \prod_{j=1}^k p_j^{X_j + \alpha_j - 1}, \end{aligned}$$

i.e., $\boldsymbol{p}|\boldsymbol{X}, n, \boldsymbol{\alpha} \sim \text{Dirichlet}(X_1 + \alpha_1, \dots, X_k + \alpha_k)$.

(2) Empirical Bayes Estimation of p

The marginal distribution of $\boldsymbol{X}|n, \boldsymbol{\alpha}$ is given by

$$\begin{aligned} f(\boldsymbol{X}|n, \boldsymbol{\alpha}) &= \int_{\{p: p_j \geq 0, \sum_{j=1}^k p_j = 1\}} f(\boldsymbol{X}|n, \boldsymbol{p}) f(\boldsymbol{p}|\boldsymbol{\alpha}) \mathrm{d}p \\ &= \frac{n! \Gamma(\sum_{j=1}^k \alpha_j)}{\Gamma(n + \sum_{j=1}^k \alpha_j)} \prod_{j=1}^k \frac{\Gamma(X_k + \alpha_k)}{X_k! \Gamma(\alpha_k)}. \end{aligned}$$

So the log-likelihood for $\boldsymbol{X}$ is

$$
\begin{aligned}
l(\boldsymbol{\alpha}) &= \log(n!) + \log(\Gamma(k\alpha)) - \log(\Gamma(n(+k\alpha))) + \sum_{j=1}^{k}[\log(\Gamma(X_k+\alpha)) - \log(\Gamma(\alpha)) - \log(X_k!)] \\
&= \log(\Gamma(k\alpha)) - \log(\Gamma(n+k\alpha))) - k\log(\Gamma(\alpha)) + \sum_{j=1}^{k}\log(\Gamma(X_k+\alpha)) + \text{Constant}.
\end{aligned}
$$

Since $\boldsymbol{p}|\boldsymbol{X}, n, \boldsymbol{\alpha} \sim \text{Dirichlet}(X_1+\alpha_1, \ldots, X_k+\alpha_k)$, the posterior mean of $\boldsymbol{p}$ is $\mathbb{E}(\boldsymbol{p}|\boldsymbol{X}, n, \boldsymbol{\alpha}) = \frac{\boldsymbol{X}+\boldsymbol{\alpha}}{k\alpha+\sum_{j=1}^{k}X_j}$.

```r
library(gtools)
library(combinat)

log_likelihood <- function(par,x){
    alpha <- par[1]
    n_samples <- dim(x)[1]
    n <- sum(x[1,])
    k <- dim(x)[2]
    n_samples*(lgamma(k*alpha) - lgamma(n+k*alpha) - k*lgamma(alpha)) + sum(lgamma(x+alpha))
}

eb <- function(x){
    par_init <- c(1)
    res <- optim(par=par_init,fn=log_likelihood, method="L-BFGS-B",
                 control=list(fnscale=-1),lower=1e-3,x=x)
    return(res$par)
}

posterior_mean <- function(x, alpha_hat){
    k <- dim(x)[2]
    (x+alpha_hat)/ (k*alpha_hat+apply(x, 1, sum))
}

set.seed(1)
k <- 5
alpha <- 2 * rep(1,k)
n_samples <- 100
n = 10
p <- rdirichlet(n_samples, alpha)
X <- rmultinomial(n,p)
alpha_hat <- eb(X)
p_alpha <- posterior_mean(X, alpha_hat)
cat('Estimated alpha = ',alpha_hat, '\nPosterior mean = ',p_alpha[1,])
```

```
## Estimated alpha =  2.083745
## Posterior mean =  0.1510253 0.1510253 0.2 0.2489747 0.2489747
```

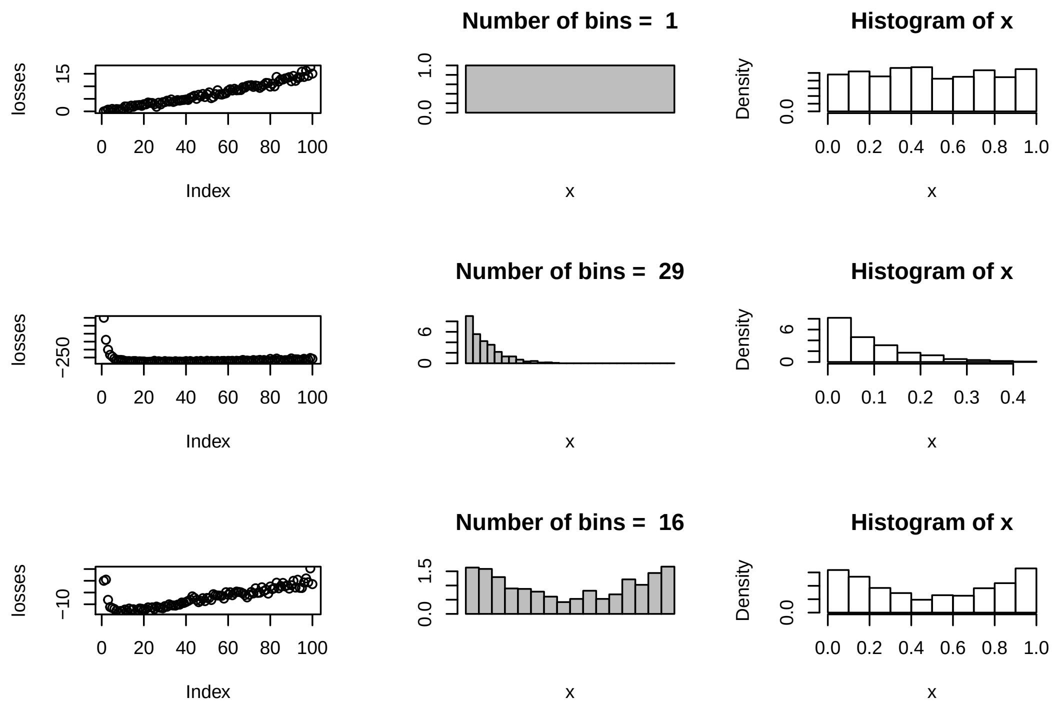
D: Density Estimation

- Following the ideas on cross validation marked #cv in the lecture notes above, write a function to automatically select an optimal bin-width for a histogram (where here optimal is measured in terms of its accuracy as a density estimator). You can assume the data are on $[0, 1]$ and that the bins are to be of equal width, spanning 0 to 1.
- Illustrate your function by plotting the resulting histogram for several simulated data sets. Use at least one dataset where the true density varies dramatically and one where the data have uniform density. Compare your results with the default of the `hist()` function in R. Comment on any good or bad features of your histograms compared with the default.
- One problem with a histogram as a density estimate is that if a bin j contains no data then the density estimate is 0. (Note that this is not necessarily a problem for visualizing the data: if a bin contains no data, then it may be fine to show that! But as a density estimate it can be unsatisfactory to estimate the density as 0.) One possible solution is to replace the maximum likelihood estimate of p_j used in constructing a histogram with the Empirical Bayes estimate (posterior mean) implemented in part C. Write a function to implement this modified density estimate, and see whether it improves the performance of the density estimate in your cross-validation experiment.

(1) Density Estimation by Histograms

```r
kl_loss <- function(p, test){
    nbin <- length(p)
    breaks <- c(0:nbin)/nbin
    breaks[1] <- -1
    p_test <- rep(0,length(test))
    for(i in 1:nbin){
        p_test[(test>breaks[i])&(test<=breaks[i+1])] <- p[i]
    }
    return(-sum(log(p_test+1e-6)))
}
com_hist <- function(x,nbin){
    breaks <- c(0:nbin)/nbin
    breaks[1] <- -1 # deal with the first bin
    counts <- c()
    for(i in c(1:nbin)){
        counts[i] <- sum((x>breaks[i])&(x<=breaks[i+1]))
    }
    p_train <- counts/sum(counts)*nbin
    return(p_train)
}
nbin_cv <- function(x){
    losses <- rep(0, 100)
    folds <- cut(seq(1,n), breaks=5, labels=FALSE)
    for(fold in 1:5){
        testIndexes <- which(folds==fold,arr.ind=TRUE)
        test <- x[testIndexes]
        train <- x[-testIndexes]
        for(nbin in c(1:100)){
            p <- com_hist(train, nbin)
            losses[nbin] <- losses[nbin] + kl_loss(p,test)
        }
    }
    losses <- losses/5
    return(losses)
}
```

```r
plot_hist <- function(x, nbin){
    p <- com_hist(x, nbin)
    barplot(p, space=0, main = paste('Number of bins = ',nbin), xlab='x')
}
set.seed(0)
n <- 1000
par(mfrow=c(3,3))
# Uniform
x <- runif(n)
losses <- nbin_cv(x)
plot(losses)
plot_hist(x, which.min(losses))
hist(x, freq=FALSE)
# Beta
x <- rbeta(n, 1, 10)
losses <- nbin_cv(x)
plot(losses)
plot_hist(x, which.min(losses))
hist(x, freq=FALSE)
# Mixture of Beta
x1 <- rbeta(n, 1, 4)
x2 <- rbeta(n, 4, 1)
b <- rbinom(n,1,0.5)
x <- x1
x[b==1] <- x2[b==1]
losses <- nbin_cv(x)
plot(losses)
plot_hist(x, which.min(losses))
hist(x, freq=FALSE)
```

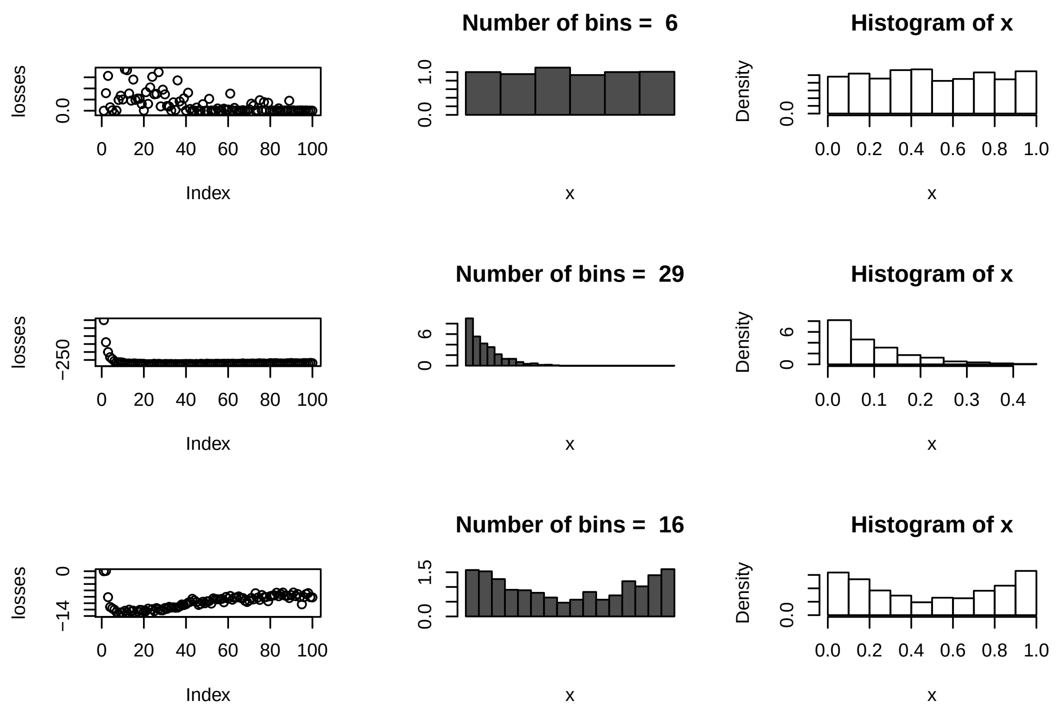


From the plots above, we can see that our density estimates are similar to the ones produced by the `hist` function. However, for uniform distribution, the best number of bins chosen by cross-validation is 1, so the plot looks better than the plot produced by `hist` function.

(2) Density Estimation by Empirical Bayes

```r
com_hist_nb <- function(x,nbin){
    breaks <- c(0:nbin)/nbin
    breaks[1] <- -1 # deal with the first bin
    counts <- matrix(0,1,nbin)
    for(i in c(1:nbin)){
        counts[1,i] <- sum((x>breaks[i])&(x<=breaks[i+1]))
    }
    alpha_hat <- eb(counts)
    p_alpha <- posterior_mean(counts, alpha_hat)*nbin
    return(p_alpha)
}
nbin_cv_nb <- function(x){
    losses <- rep(0, 100)
    folds <- cut(seq(1,n), breaks=5, labels=FALSE)
    for(fold in 1:5){
        testIndexes <- which(folds==fold,arr.ind=TRUE)
        test <- x[testIndexes]
        train <- x[-testIndexes]
        for(nbin in c(1:100)){
            p <- com_hist_nb(train, nbin)
            losses[nbin] <- losses[nbin] + kl_loss(p,test)
        }
    }
    losses <- losses/5
    return(losses)
}
plot_hist_nb <- function(x, nbin){
    p <- com_hist_nb(x, nbin)
    barplot(p, space=0, main = paste('Number of bins = ',nbin), xlab='x')
}
set.seed(0)
n <- 1000
par(mfrow=c(3,3))
# Uniform
x <- runif(n)
losses <- nbin_cv_nb(x)
plot(losses)
plot_hist_nb(x, which.min(losses))
hist(x, freq=FALSE)
# Beta
x <- rbeta(n, 1, 10)
losses <- nbin_cv_nb(x)
plot(losses)
plot_hist_nb(x, which.min(losses))
hist(x, freq=FALSE)
# Mixture of Beta
x1 <- rbeta(n, 1, 4)
x2 <- rbeta(n, 4, 1)
b <- rbinom(n,1,0.5)
x <- x1
x[b==1] <- x2[b==1]
losses <- nbin_cv_nb(x)
plot(losses)
```

```r
plot_hist_nb(x, which.min(losses))
hist(x, freq=FALSE)
```



For the uniform distribution and the mixed beta distribution, the modified histogram algorithm performs similar to the original histogram algorithm. While for the Beta$(1, 10)$ distribution, its right tail is very thin. The original histogram algorithm will give zero density estimation:

```r
x <- rbeta(n, 1, 10)
com_hist(x, 29)
```

```
##  [1] 8.584 6.815 4.553 2.784 2.581 0.899 1.102 0.609 0.464 0.174 0.232 0.087
## [13] 0.000 0.029 0.029 0.029 0.000 0.029 0.000 0.000 0.000 0.000 0.000 0.000
## [25] 0.000 0.000 0.000 0.000 0.000
```

While the modified histogram algorithm gives nonzero density estimation:

```r
com_hist_nb(x, 29)
```

```
##          [,1]    [,2]     [,3]     [,4]     [,5]     [,6]     [,7]      [,8]
## [1,] 8.554042 6.79203 4.538965 2.776953 2.574755 0.899399 1.101597 0.6105445
##           [,9]     [,10]     [,11]      [,12]       [,13]      [,14]      [,15]
## [1,] 0.4661173 0.1772629 0.2350338 0.09060655 0.003950215 0.03283566 0.03283566
##           [,16]       [,17]      [,18]       [,19]       [,20]       [,21]
## [1,] 0.03283566 0.003950215 0.03283566 0.003950215 0.003950215 0.003950215
##            [,22]       [,23]       [,24]       [,25]       [,26]       [,27]
## [1,] 0.003950215 0.003950215 0.003950215 0.003950215 0.003950215 0.003950215
##            [,28]       [,29]
## [1,] 0.003950215 0.003950215
```