Homework 4

Jimmy Hickey

Due @ 5pm on March 6, 2020

1

Part 1. In this homework, you will practice writing equivalent optimization problems.

1

1. Inpainting

In the inpainting problem we seek to fill in missing pixels in an image. Think reconstructing lost or deteriorated parts of a painting. Images tend to be smooth in the sense that a pixel's value tends to be similar to its neighboring pixel values. Thus, a natural strategy to filling in missing pixels is to perform interpolation. We can formalize an interpolation strategy as an LP. We will focus on square gray-scale images but rectangular color images can be delt with similarly. Let $\mathbf{X} \in \{1, \dots, 255\}^{m \times n}$ denote our estimate of an image with mn pixels. Suppose we only get to observe a subset $\Omega \subset \{1, \dots, m\} \times \{1, \dots, n\}$ of the total number of pixels. Let $\mathbf{x}$ denote vec($\mathbf{X}$), the mn length vector obtained by stacking up all n columns of $\mathbf{X}$. Let $\mathbf{y} \in \mathbb{R}^{|\Omega|}$ denote the values of the image over the subset of observed pixels, and let $\mathbf{D}_m$ denote the first-order differencing matrix:

$$\mathbf{D}_m = \begin{pmatrix} 1 & -1 & 0 & & \\ 0 & 1 & -1 & 0 & \\ & & \ddots & \ddots & \\ & & & 1 & -1 \end{pmatrix} \in \mathbb{R}^{(m-1)\times m}.$$

The matrix $\mathbf{D}_m\mathbf{X}$ is the $(m-1) \times n$ matrix obtained by differencing adjacent rows. The matrix $\mathbf{X}\mathbf{D}_n^{\mathsf{T}}$ is the $m \times (n-1)$ matrix obtained by differencing adjacent columns. We can enforce smoothness in adjacent horizontal and vertical pixels by seeking an $\mathbf{X}$ that minimizes the sum of the absolute values of the entries of the matrices $\mathbf{D}_m\mathbf{X}$ and $\mathbf{D}_n\mathbf{X}^{\mathsf{T}}$, namely

$$f(\mathbf{X}) = \left[\sum_{j=1}^{n}\sum_{i=1}^{m-1}|x_{i,j} - x_{i+1,j}|\right] + \left[\sum_{i=1}^{m}\sum_{j=1}^{n-1}|x_{i,j} - x_{i,j+1}|\right]$$

(a) Find the matrix $\hat{\mathbf{D}}$ such that $f(\mathbf{X}) = \|\hat{\mathbf{D}}\mathbf{x}\|_1$. Hint: Use the identity vec($\mathbf{ABC}$) $= [\mathbf{C}^{\mathsf{T}} \otimes \mathbf{A}]$vec($\mathbf{B}$).

Inpainting can be solved by solving the following optimization problem.

$$\begin{aligned} &\text{minimize } \|\hat{\mathbf{D}}\mathbf{x}\|_1 \\ &\text{subject to } P_\Omega\mathbf{x} = \mathbf{y}, \mathbf{x} \geq 0, \end{aligned}$$

where $P_\Omega \in \{0,1\}^{|\Omega| \times mn}$ is a matrix that maps $\mathbf{x}$ to a smaller vector $P_\Omega\mathbf{x}$ corresponding to the estimate $\mathbf{x}$ at the observed indices.

$$\begin{aligned}
f(X) &= ||vec(D_m X)||_1 + ||vec(X D_n^T)||_1 \\
&= ||vec(D_m X I_n)||_1 + ||vec(I_m X D_n^T)||_1 \\
&= (I_n \otimes D_m) vec(X) + (D_n \otimes I_m) vec(X) \\
&= \begin{bmatrix} I_n \otimes D_m \\ D_n \otimes I_m \end{bmatrix} vec(X)
\end{aligned}$$

$$\widehat{D} = \begin{bmatrix} I_n \otimes D_m \\ D_n \otimes I_m \end{bmatrix}$$

(b) Formulate the above optimization problem as an LP.

Take $w = \widehat{D}x = w_+ - w_-$. Then $\widehat{D}x - w_+ + w_- = 0$. Then we get our constraints to be

$$\underbrace{\begin{bmatrix} -I & 0 & 0 \\ 0 & -I & 0 \\ 0 & 0 & -I \end{bmatrix}}_{G} \underbrace{\begin{bmatrix} x \\ w_+ \\ w_- \end{bmatrix}}_{x} \leq \underbrace{0}_{h}$$

$$\underbrace{\begin{bmatrix} P_\Omega & 0 & 0 \\ \widehat{D} & -I & I \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x \\ w_+ \\ w_- \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} y \\ 0 \end{bmatrix}}_{b}$$

And we are minimizing,

$$\min_{x, w_+, w_-} \underbrace{\begin{bmatrix} 0 & 1 & 1 \end{bmatrix}}_{c^T} \underbrace{\begin{bmatrix} x \\ w_+ \\ w_- \end{bmatrix}}_{x}.$$

2

2. ℓ_1-Trend Filtering

Let $\mathbf{y} \in \mathbb{R}^n$ be a noisy time series signal that we wish to smooth. Let $\mathbf{D}_n^{(1)}$ denote the $(n-1) \times n$ first-order differencing matrix. Note that the second-order differencing matrix $\mathbf{D}_{n-1}^{(2)}$ can be written as the product of two first-order differencing matrices: $\mathbf{D}_{n-1}^{(2)} = \mathbf{D}_{n-1}^{(1)}\mathbf{D}_n^{(1)}$. More generally, the kth order differencing matrix $\mathbf{D}_{n-k+1}^{(k)}$ can be writen as the product of a first-order differencing matrix and a $k-1$th order differencing matrix: $\mathbf{D}_{n-k+1}^{(k)} = \mathbf{D}_{n-k+1}^{(1)}\mathbf{D}_{n-k+2}^{(k-1)}$. In ℓ_1 trend filtering, we obtain a smoothed estimate $\boldsymbol{\theta}$ of $\mathbf{y}$ by solving the following quadratic program

$$\underset{\boldsymbol{\theta} \in \mathbb{R}^n}{\text{minimize}} \ \frac{1}{2}\|\mathbf{y} - \boldsymbol{\theta}\|_2^2 + \lambda\|\mathbf{D}_{n-k+1}^{(k)}\boldsymbol{\theta}\|_1,$$

where $\lambda \geq 0$ is a regularization parameter that trades off data fit with smoothness.

(a) Formulate the ℓ_1 trend filtering problem as a quadratic program.

Take $\Delta = D_{n-k+1}^{(k)}\theta = \Delta_+ - \Delta_-$. Then we get the following constraints.

$$\underbrace{\begin{bmatrix} 0 & -I & 0 \\ 0 & 0 & -I \end{bmatrix}}_{G} \underbrace{\begin{bmatrix} \theta \\ \Delta_+ \\ \Delta_- \end{bmatrix}}_{x} \leq \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{h}$$

$$\underbrace{\begin{bmatrix} D_{n-k+1}^{(k)} & -I & I \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \theta \\ \Delta_+ \\ \Delta_- \end{bmatrix}}_{x} = \underbrace{0}_{b}$$

Now we can expand our objective function to

$$\frac{1}{2}\|\mathbf{y} - \boldsymbol{\theta}\|_2^2 + \lambda\|\mathbf{D}_{n-k+1}^{(k)}\boldsymbol{\theta}\|_1 = \frac{1}{2}||y||_2^2 - y^t + \frac{1}{2}\theta^T\theta + \lambda 1^T\Delta_+ + \lambda 1^T\Delta_-$$

$$\underbrace{\frac{1}{2}||y||_2^2}_{r} + \underbrace{\begin{bmatrix} -y^T & \lambda 1^T & \lambda 1^T \end{bmatrix}}_{q^T} \underbrace{\begin{bmatrix} \theta \\ \Delta_+ \\ \Delta_- \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} \theta & \Delta_+ & \Delta_- \end{bmatrix}}_{x^T} \underbrace{\begin{bmatrix} \frac{1}{2}I & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} \theta \\ \Delta_+ \\ \Delta_- \end{bmatrix}}_{x}$$

You can find my code for part inpainting in `inpainting.R`. You can find the driver code in `/homework4/inpainting_driver.R`. I have pasted the code below, but used my driver to actually run it.

2

Part 2. Use R Gurobi to solve the convex programs you formulated in Part 1.

1. Inpainting:

Write a function "myGetTV2d" that computes the matrix $\hat{\mathbf{D}}$ from question 1 in Part 1.

Note that I adapted my Dkn code from homework 2 to fit this problem and to avoid having a repeated function name in my package.

```r
#' Compute kth order differencing matrix
#'
#' @param k order of the differencing matrix
#' @param n Number of time points
#' @param negative Flip signs, row one will look lie 1 -1 0 ... 0 (default: FALSE)
#' @return kth order differencing matrix
myGetDkn <- function(k, n, negative=FALSE)
{
  library(Matrix)
  neg = negative

  ii = 1 + (!negative) * (-2)
  iiplusone = -ii

   # create D1n
  if (k==1)
  {
    D1 = matrix(nrow = n-1, ncol = n)
    zeros_matrix = rep(0, n)

    for(i in 1:n-1)
    {
      D1[i,] = zeros_matrix
      D1[i,i] = ii
      D1[i, i+1] = iiplusone
    } # for
    return(Matrix(D1, sparse = TRUE))
  } # endif

  return(myGetDkn(k=1, n=n - k + 1, neg) %*% myGetDkn(k=k-1, n=n, neg))

}
```

```r
#' Compute the inpainting differencing matrix
#'
#' @param m number of rows
#' @param n number of columns
#' @export
myGetTV2d <- function(m,n) {

 In = sparseDiag(n, 1)
 Im = sparseDiag(m, 1)

 return( rbind(kronecker(In, myGetDkn(1, m, negative=TRUE)),
               kronecker(myGetDkn(1, n, negative=TRUE),Im)))
}
```

Your function should return a **sparse** matrix (Please use the Matrix package). It is most convenient to use matrix Kronecker products to do this.

Step 2: Write a function "myInpaint" to compute a reconstructed image (matrix) X, from the available data matrix Y, by solving the standard form LP obtained in question 1 in Part 1 using gurobi. The input matrix Y should be the same size as the "smooth" matrix X that you are aiming to recover. The missing entries of Y should be expected to be non-numeric, e.g. NA or NaN, so the function is.numeric should be useful.

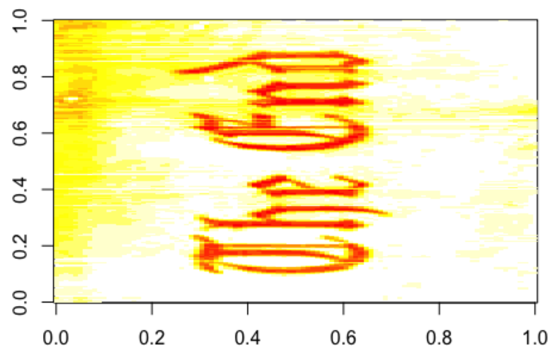
```
#' Create an n x n sparse diagonal matrix
#'
#' @param n size of matrix
#' @param diag number on diag
#' @export
sparseDiag = function(n, diag)
{
 return(bandSparse(n, n, 0, as.matrix(rep(diag,n))))
}

#' Remove missing data to calculate y and P_Omega
#'
#' @param inMatrix input data matrix
#' @export
removeMissing = function(inMatrix)
{
 in_vec = c(inMatrix)
 mn = length(in_vec)

 not_na_indices = which(!is.na(inMatrix) & in_vec != -1 )
 num_not_na = length(not_na_indices)

 y = in_vec[not_na_indices]
 P_Omega = sparseMatrix(i = 1:num_not_na, j = not_na_indices, x = 1, dims=c(num_not_na, mn))

 out_list = list(
                "p_omega" = P_Omega,
                "y" = y
              )

 return(out_list)
}


#' Compute the inpainting completion
#'
#' @param Y data matrix
#' @export
myInpaint <- function(Y) {
 # x vector = (x, w+, w-)

 library(gurobi)

 m = nrow(Y)
 n = ncol(Y)
```

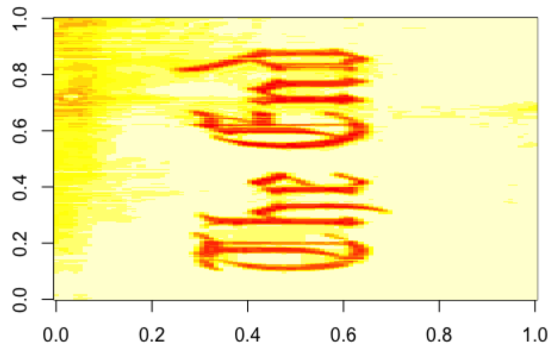
```r
mn = m * n


Dhat = myGetTV2d(m, n)
Dhat_nrow = nrow(Dhat)

no_missing = removeMissing(Y)
p_omega = no_missing$p_omega
y = no_missing$y

neg_Imn = sparseDiag(mn, -1)

neg_Idrow = sparseDiag(Dhat_nrow, -1)
zero_drow = sparseDiag(Dhat_nrow, 0)
one_vec_drow =  rep(1, Dhat_nrow)


zero_p_drow = Matrix(0, nrow=nrow(p_omega), ncol = Dhat_nrow, sparse=TRUE)
zero_mn_drow = Matrix(0, nrow=mn, ncol = Dhat_nrow, sparse=TRUE)
zero_drow_mn =  Matrix(0, nrow=Dhat_nrow, ncol = mn, sparse=TRUE)
zero_drow_drow = Matrix(0, nrow=Dhat_nrow, ncol = Dhat_nrow, sparse=TRUE)


zero_vec_mn = rep(0, mn)
zero_vec_drow = rep(0, Dhat_nrow)

equality_constraints = rbind(
                          cbind(no_missing$p_omega, zero_p_drow, zero_p_drow),
                          cbind(Dhat, neg_Idrow, -1 * neg_Idrow)
                         )


inequality_constraints = rbind(
                          cbind(neg_Imn, zero_mn_drow, zero_mn_drow),
                          cbind(zero_drow_mn, neg_Idrow, zero_drow_drow),
                          cbind(zero_drow_mn, zero_drow_drow, neg_Idrow)
                          )

gurobi_model = list()

gurobi_model$A = rbind(equality_constraints, inequality_constraints)
gurobi_model$obj = c(zero_vec_mn, one_vec_drow, one_vec_drow)
gurobi_model$modelsense = 'min'
gurobi_model$rhs = c(y, zero_vec_drow,
                     zero_vec_mn, zero_vec_drow, zero_vec_drow)
gurobi_model$sense = c(
                       rep('=', length(y)),
                       rep('=', length(zero_vec_drow)),
                       rep('<=', length(zero_vec_mn)),
                       rep('<=', length(zero_vec_drow)),
                       rep('<=', length(zero_vec_drow)))

params <- list(OutputFlag=0)
```

```
  result <- gurobi(gurobi_model, params)

  return(result)
}
```

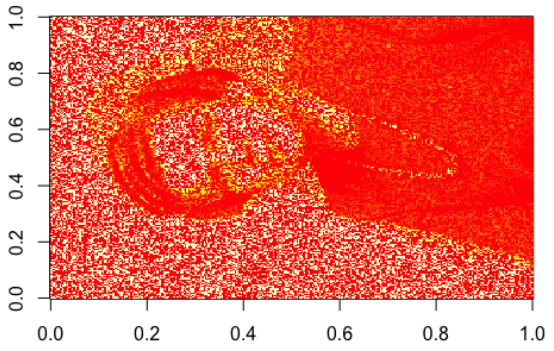
Your function should return a regular R matrix that is the restored image. Make sure that you use sparse matrices to construct the LP coefficient matrix. Your code will be painfully slow if you do not use sparse matrices.

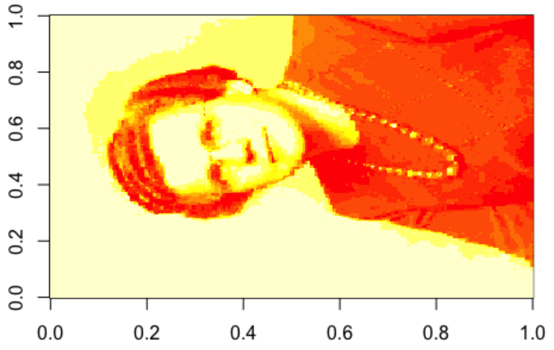
Step 3: Use your inpainting function to "restore" Ed Ruscha's painting. Include a before and after image.





Step 4: Use your inpainting function to "restore" the photo of Gertrude Cox. Include a before and after image.


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

You can find my code for part inpainting in `l1_trend_filtering.R`. You can find the driver code in `/homework4/trend_filtering_driver.R`. I have pasted the code below, but used my driver to actually run it.

2. ℓ_1-Trend Filtering:

Step 1: Write a function "myGetDkn" to compute the kth order differencing matrix $D_n^{(k)}$

Note that I adapted my Dkn code from homework 2 to fit this problem and to avoid having a repeated function name in my package.

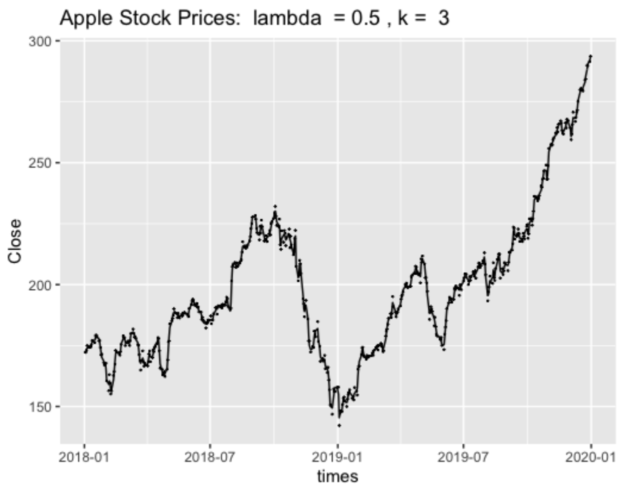
```r
#' Compute kth order differencing matrix
#'
#' @param k order of the differencing matrix
#' @param n Number of time points
#' @param negative Flip signs, row one will look lie 1 -1 0 ... 0 (default: FALSE)
#' @return kth order differencing matrix
myGetDkn <- function(k, n, negative=FALSE)
{
  library(Matrix)
  neg = negative

  ii = 1 + (!negative) * (-2)
  iiplusone = -ii

   # create D1n
  if (k==1)
  {
    D1 = matrix(nrow = n-1, ncol = n)
    zeros_matrix = rep(0, n)

    for(i in 1:n-1)
    {
      D1[i,] = zeros_matrix
      D1[i,i] = ii
      D1[i, i+1] = iiplusone
    } # for
    return(Matrix(D1, sparse = TRUE))
  } # endif

  return(myGetDkn(k=1, n=n - k + 1, neg) %*% myGetDkn(k=k-1, n=n, neg))

}
```

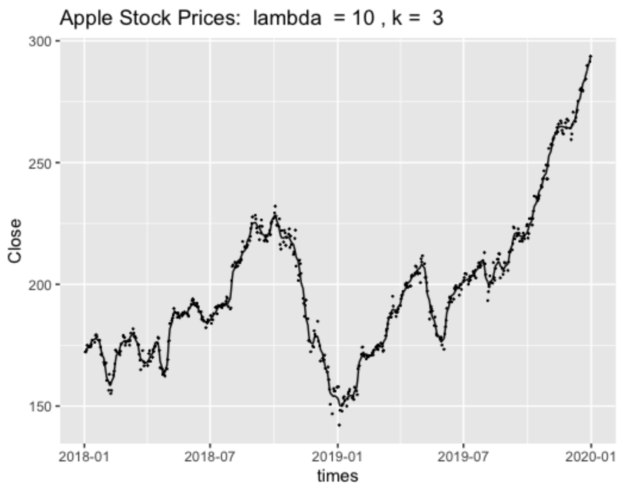
where the output matrix should be a **sparse** matrix.

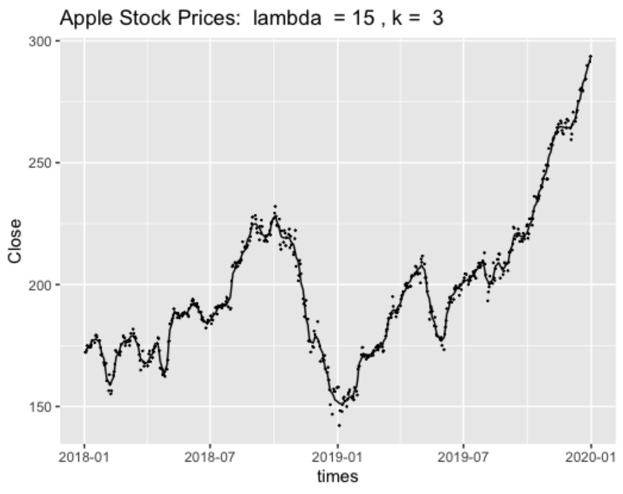
Step 2: Write a function "myTrendFilter" to compute a smoothed estimate θ of noisy time series data y by solving the QP obtained in question 2 in Part 1.

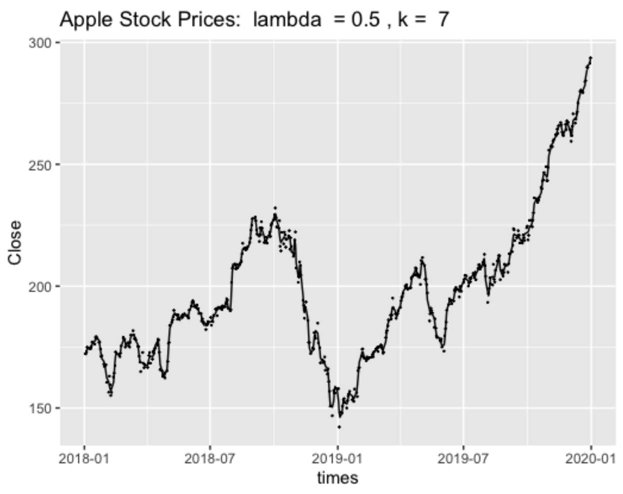
```r
#' Compute the kth order trend filtering estimator
#'
#' @param y noisy signal
#' @param k order of differencing operator
#' @param lambda regularization parameter
#' @export
myTrendFilter <- function(y, k, lambda) {
 library(gurobi)

 n = length(y)

 Dk = myGetDkn(k = k, n = n)
 nrow_d = nrow(Dk)

 neg_id_d = -1 * Diagonal(nrow_d)
 pos_id_d = Diagonal(nrow_d)
 half_id_n = 0.5 * Diagonal(n)

 zero_d_n = Matrix(0, nrow=nrow_d, ncol = n, sparse=TRUE)
 zero_d_d = Matrix(0, nrow=nrow_d, ncol = nrow_d, sparse=TRUE)
 zero_n_d = Matrix(0, nrow=n, ncol = nrow_d, sparse=TRUE)
 zero_n_n = Matrix(0, nrow=n, ncol = n, sparse=TRUE)

 lambda_d = rep(lambda, nrow_d)

 linear = c(-1 * t(y), lambda_d, lambda_d)

 quadratic = rbind(
                   cbind(half_id_n, zero_n_d, zero_n_d),
                   cbind(zero_d_n, zero_d_d, zero_d_d),
                   cbind(zero_d_n, zero_d_d, zero_d_d)
                  )

 equality_constraints = cbind(Dk, neg_id_d, pos_id_d)
 inequality_constraints = rbind(
                               cbind(zero_d_n, neg_id_d, zero_d_d),
                               cbind(zero_d_n, zero_d_d, neg_id_d)
                               )

 model = list()


 model$A = rbind(equality_constraints, inequality_constraints)
 model$Q = quadratic
 model$obj = linear
 model$rhs = c(
               rep(0, nrow_d),
               rep(0, nrow_d),
               rep(0, nrow_d)
              )
 model$sense = c(
                 rep("=", nrow_d),
                 rep("<=", nrow_d),
                 rep("<=", nrow_d)
                )

 cat(dim( model$A))
 cat(" ")
 cat(dim( model$Q))

 result = gurobi(model)

 return(result)

}
```

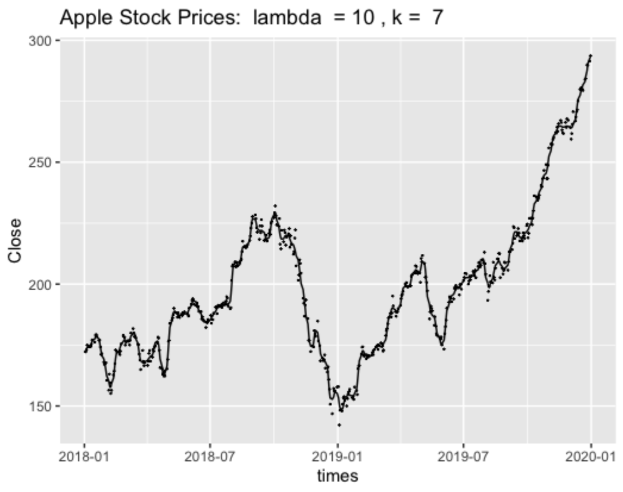
Step 3: Use your trend filtering function to smooth some interesting time series data. For example, you might use the tseries R package on CRAN (see the function **get.hist.quote**) to download historical financial data for the daily closing prices of Apple stock over the past two years. Try several λ values - different enough to generate noticably different smoothed estimates - and at least two differencing matrix orders, e.g. $\mathbf{D}_n^{(2)}$ and $\mathbf{D}_n^{(3)}$. Provide plots of the noisy and smoothed estimates generated by your code.

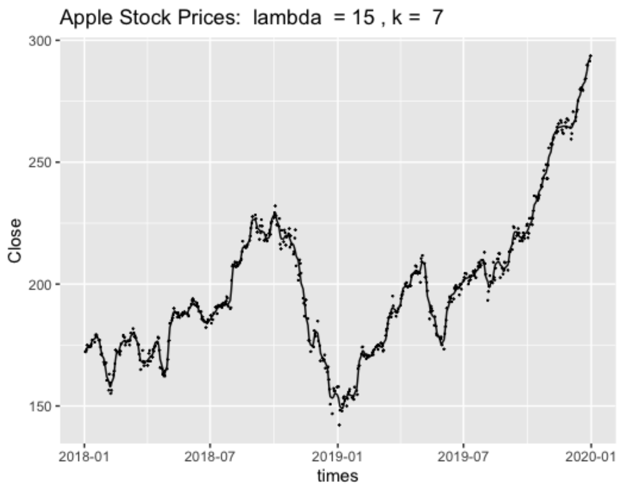







Apple Stock Prices: lambda = 0.5 , k = 7
Close
times
300
250
200
150
2018-01
2018-07
2019-01
2019-07
2020-01


Apple Stock Prices: lambda = 10 , k = 7
Close
times
300
250
200
150
2018-01
2018-07
2019-01
2019-07
2020-01


Apple Stock Prices: lambda = 15 , k = 7
Close
times
300
250
200
150
2018-01
2018-07
2019-01
2019-07
2020-01