Math 365 / Comp 365: Exam 2

Instructions:

- This portion of the exam should be added to the back of the written part, and you should submit a single, stapled copy.
- You must also submit this file in Moodle.
- Please fill in your name in the appropriate place below.
- Please leave intact the leading blocks which load the Matrix package, the `365FUnctions.r` file, and the data for problem 5.
- Please read carefully the instructions for each problem.
- Please preserve all of the section headings for solutions, and insert your solutions in the appropriate places. If you used R to solve any of the written problems, create careful section headings like I have done here.
- You are certainly allowed to pull code from things we've used/developed in this course (e.g., any code I gave you as part of in-class activities).

Lawson Busch

```
require(Matrix)
require(mosaicData)
source("https://drive.google.com/uc?export=download&id=10dNH3VbvxS8Z3OHjP4i9gRbtsf91VVBb") #365functions.R
source("https://drive.google.com/uc?export=download&id=1et61nwz2dQzJ6TiJlNpGNx5vtVrqHMmE") #facebook.prices
```

Problem 1 Computations

```
u = c(1/3, 2/3, 2/3)
v = c(2/3, -1/3, 2/3)
A = cbind(u, v)
P1 = u%*%t(u) + v%*%t(v)
P1
```

```
##           [,1]      [,2]      [,3]
## [1,] 0.5555556 0.0000000 0.6666667
## [2,] 0.0000000 0.5555556 0.2222222
## [3,] 0.6666667 0.2222222 0.8888889
```

```
ata = solve(t(A)%*%A)
P2 = A%*%ata%*%t(A)
P2
```

```
##             [,1]       [,2]      [,3]
## [1,]  0.4461538 -0.1846154 0.4615385
## [2,] -0.1846154  0.9384615 0.1538462
## [3,]  0.4615385  0.1538462 0.6153846
```

Problem 2 Computations

```
A = cbind(c(1, -3, 7, 2), c(2, -8, 9, 3))
ata = solve(t(A)%*%A)
P = A%*%ata%*%t(A)
P
```

```
##             [,1]        [,2]       [,3]        [,4]
## [1,]  0.03229279 -0.16361679 0.05920344  0.03121636
## [2,] -0.16361679  0.96232508 0.03336921 -0.09149623
## [3,]  0.05920344  0.03336921 0.94187298  0.22389666
## [4,]  0.03121636 -0.09149623 0.22389666  0.06350915
```

```
w1 = c(-29/2, -5/2, 1, 0)
w2 = c(-7/2, -1/2, 0, 1)
t(A)%*%w1
```

```
##      [,1]
## [1,]    0
## [2,]    0
```

```
t(A)%*%w2
```

```
##      [,1]
## [1,]    0
## [2,]    0
```

```
wq1 = (1/vnorm(w1))*w1
wq2 = (1/vnorm(w2))*(w2-wq1%*%t(wq1)%*%w2)
wq1
```

```
## [1] -0.98319208 -0.16951588  0.06780635  0.00000000
```

```
wq2
```

```
##              [,1]
## [1,] -0.009072184
## [2,]  0.026590885
## [3,] -0.065069459
## [4,]  0.272165527
```

```
A0 = cbind(wq1, wq2)
ata0 = solve(t(A0)%*%A0)
P0 = A0%*%ata0%*%t(A0)
P0
```

```
##             [,1]        [,2]        [,3]        [,4]
## [1,]  0.96770721  0.16361679 -0.05920344 -0.03121636
## [2,]  0.16361679  0.03767492 -0.03336921  0.09149623
## [3,] -0.05920344 -0.03336921  0.05812702 -0.22389666
## [4,] -0.03121636  0.09149623 -0.22389666  0.93649085
```

```
b0 = c(2,0,1,5)
P0b = P0 %*% b0
A[,1]%*%P0b
```

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

```
A[,2]%*%P0b
```

```
##              [,1]
## [1,] 1.776357e-15
```

```
P+P0
```

```
##               [,1]          [,2]          [,3]          [,4]
## [1,]  1.000000e+00 -8.326673e-17  1.387779e-17  1.734723e-17
## [2,] -1.665335e-16  1.000000e+00 -2.081668e-17 -1.249001e-16
## [3,]  1.040834e-16 -2.081668e-17  1.000000e+00  3.053113e-16
## [4,]  6.938894e-17 -2.498002e-16  4.163336e-16  1.000000e+00
```

Problem 3 Computations

```
t = c(1, 3, 5, 7, 9)
ones = rep(1, 5)
v = c(2, 5, 7, 4, 1)
w = c(3, 7, 10, 6, 3)
A = cbind(ones, t, t^2)
wstar = solve(t(A)%*%A, t(A)%*%w)
vstar = solve(t(A)%*%A, t(A)%*%v)

wstar
```

```
##        [,1]
## ones -0.325
## t     3.700
##      -0.375
```

```
vstar
```

```
##            [,1]
## ones -0.6107143
## t     2.8857143
##      -0.3035714
```

```
wres = vnorm(w-A%*%wstar,p=2)^2
vres = vnorm(v-A%*%vstar,p=2)^2

wres
```

```
## [1] 3.2
```

```
vres
```

```
## [1] 1.257143
```

```
xStar = c(-0.5,5,-0.5)
uHat = A%*%xStar
uHat
```

```
##      [,1]
## [1,]    4
## [2,]   10
## [3,]   12
## [4,]   10
## [5,]    4
```

```
out=qr(A)
(Q.bar=qr.Q(out,complete=TRUE))
```

```
##             [,1]          [,2]       [,3]        [,4]       [,5]
## [1,] -0.4472136 -6.324555e-01  0.5345225 -0.02576091 -0.3370788
## [2,] -0.4472136 -3.162278e-01 -0.2672612  0.28091869  0.7413880
## [3,] -0.4472136  5.551115e-17 -0.5345225 -0.68819061 -0.2016914
## [4,] -0.4472136  3.162278e-01 -0.2672612  0.63666879 -0.4724662
## [5,] -0.4472136  6.324555e-01  0.5345225 -0.20363596  0.2698483
```

```
qhat = Q.bar[,4:5]
r=sqrt(3/4)*qhat[,1]+0.5*qhat[,2]
r
```

```
## [1] -0.19084898  0.61397670 -0.69683622  0.31513826 -0.04142976
```

```
vnorm(r) #checks out
```

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

```
u = r+uHat
u
```

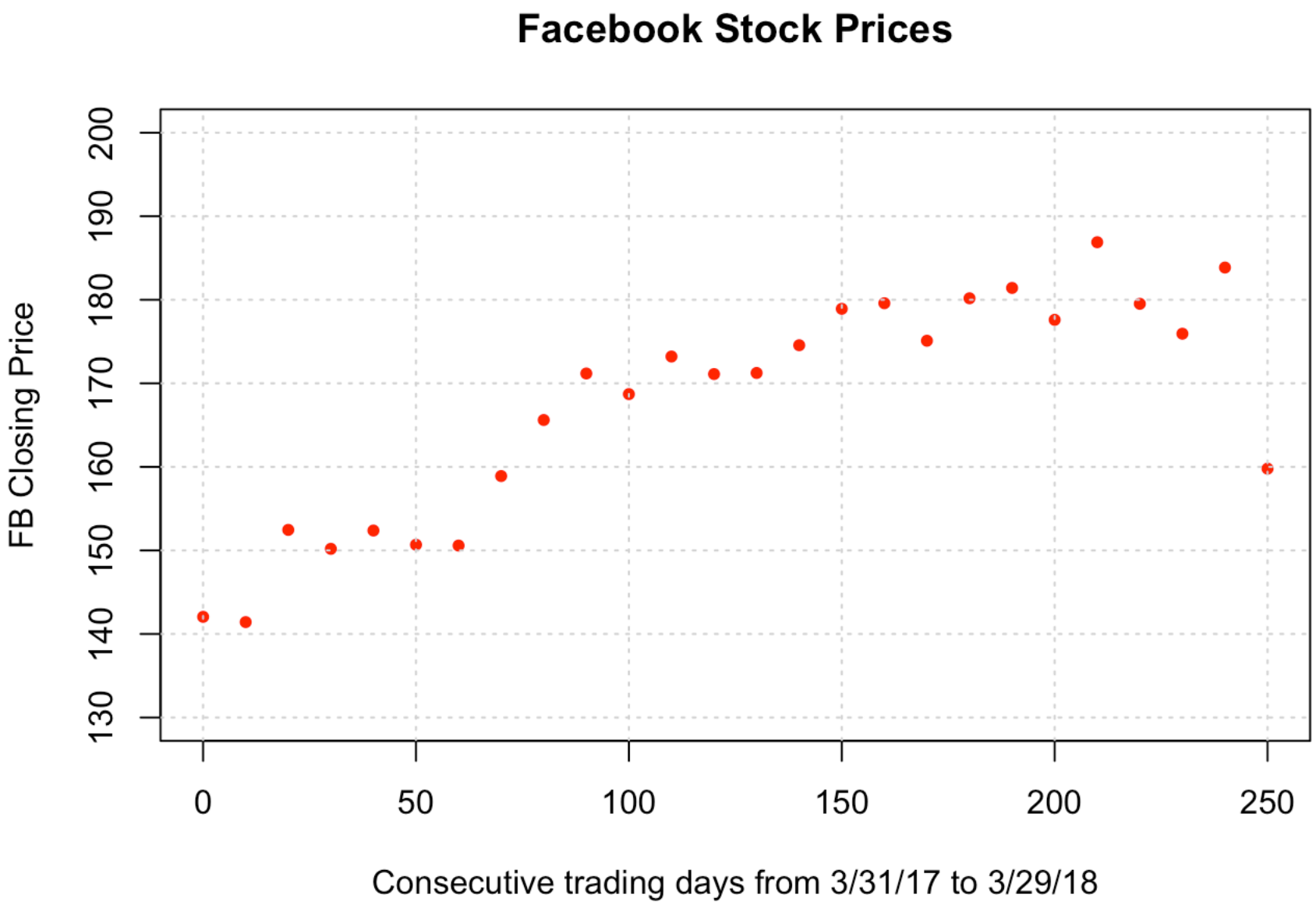
```
##           [,1]
## [1,]  3.809151
## [2,] 10.613977
## [3,] 11.303164
## [4,] 10.315138
## [5,]  3.958570
```

Problem 5

The stock price of Facebook has been in the news a lot lately, so I was interested to see how it has fluctuated throughout the last year. I accidentally only downloaded the closing prices at every 10^{th} trading day (roughly every two weeks) for the past 251 trading days (one trading year from 3/31/2017 to 3/29/2018). Let's examine three different ways to fill in the missing data.

For now, here is a plot of the data we have. Note: the prices are stored in `samp.prices`, which I have loaded for you.

```
days = 0:250
samp.days = seq(0,250,by=10) # we only have data on these days
plot(samp.days,samp.prices,pch=20,cex=.5,ylim=c(130,200),
     ylab = "FB Closing Price",
     xlab = "Consecutive trading days from 3/31/17 to 3/29/18",
     main = "Facebook Stock Prices")
points(samp.days,samp.prices,pch=20,cex=1,col='red')
grid()
```



Here is the function we are going to use to plot the different approximation models:

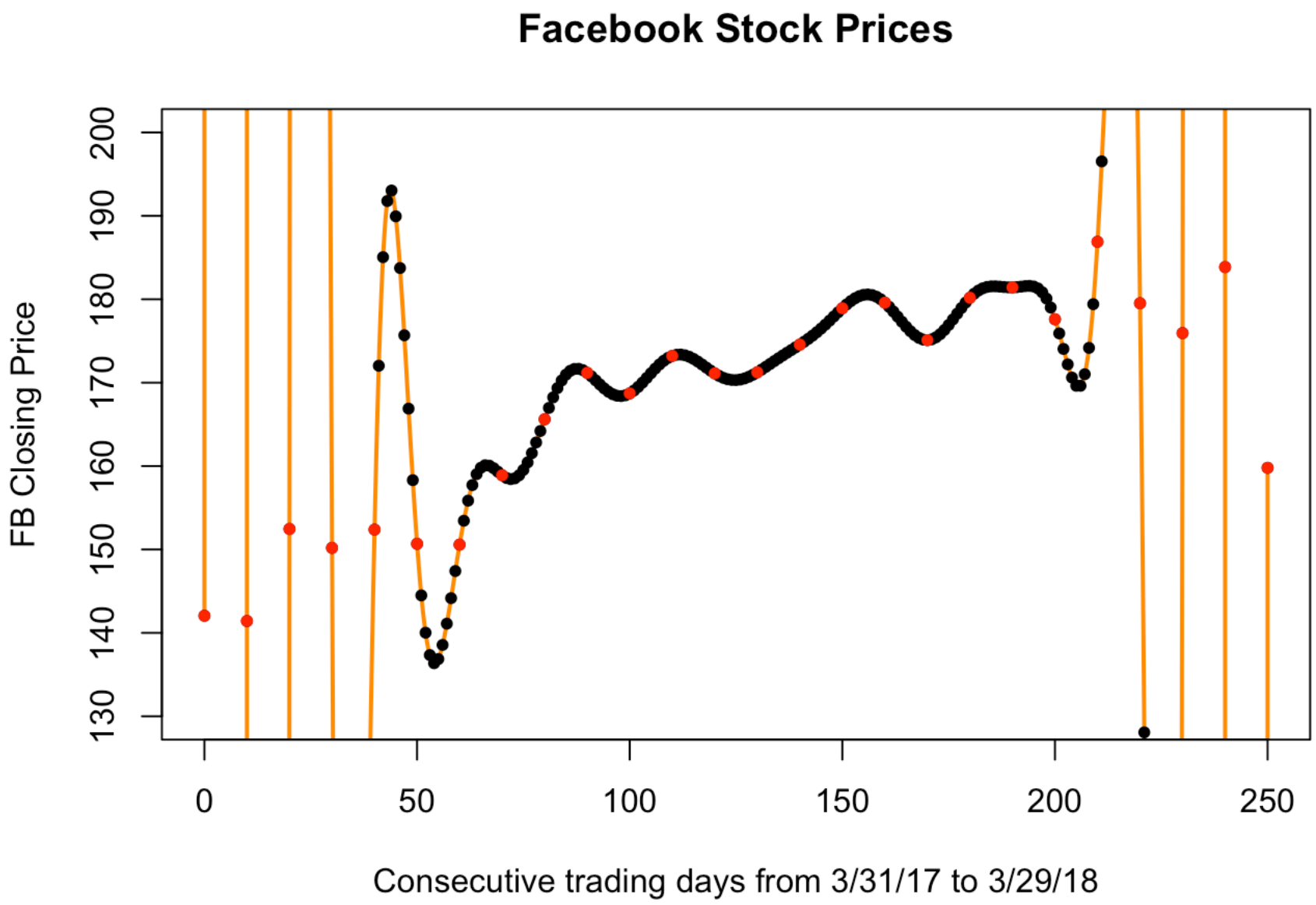
```
plotApproximations=function(f){
  tt = seq(0,250,length=1001)
  plot(tt,f(tt),col="DarkOrange",type='l',lwd=2,ylim=c(130,200),
       ylab = "FB Closing Price",
       xlab = "Consecutive trading days from 3/31/17 to 3/29/18",
       main = "Facebook Stock Prices")
  points(days,f(days),pch=20,cex=1,col='black') # The points guessed by the interpola
ting/regressing functions
  points(samp.days,samp.prices,pch=20,cex=1,col='red') # The sample points we have fr
om every 10th day
}
```

Part (a):

Fit a degree 25 interpolating polynomial to this data. Call the function that evaluates the polynomial `p`. Write your code in the block below and uncomment the last line there to display your answer.

Part (a) solution

```
xx = seq(0, 250, length=1001)
p = function(x) {
  Interpolator(samp.days, samp.prices, x)
}
plotApproximations(p)
```

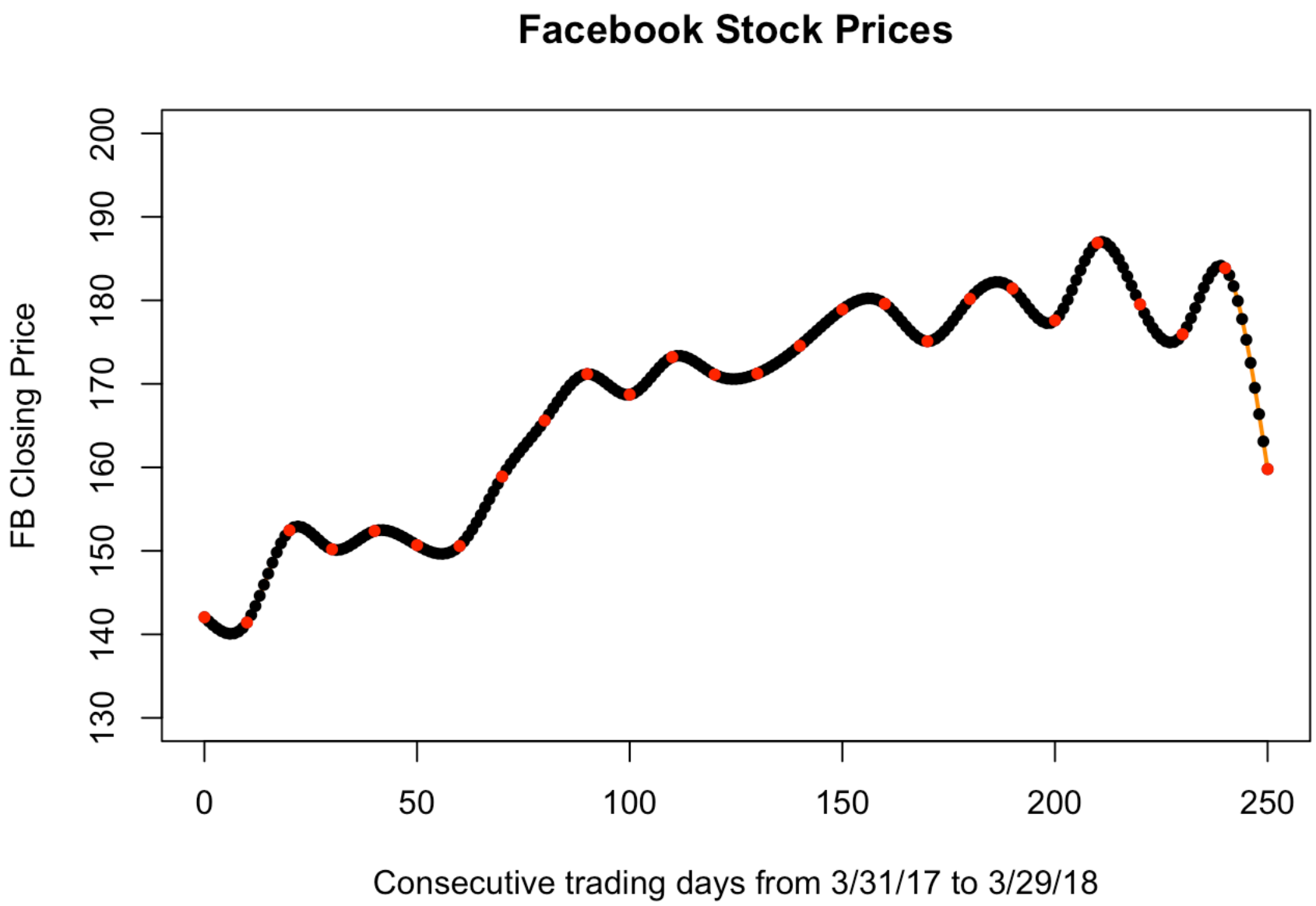


Part (b):

Compute a natural cubic spline passing through the 26 stock prices. Call the function that evaluates the spline `s`. Write your code in the block below and uncomment the last line there to display your answer.

Part (b) solution

```
s = splinefun(samp.days, samp.prices, method='natural')
plotApproximations(s)
```



Part (c):

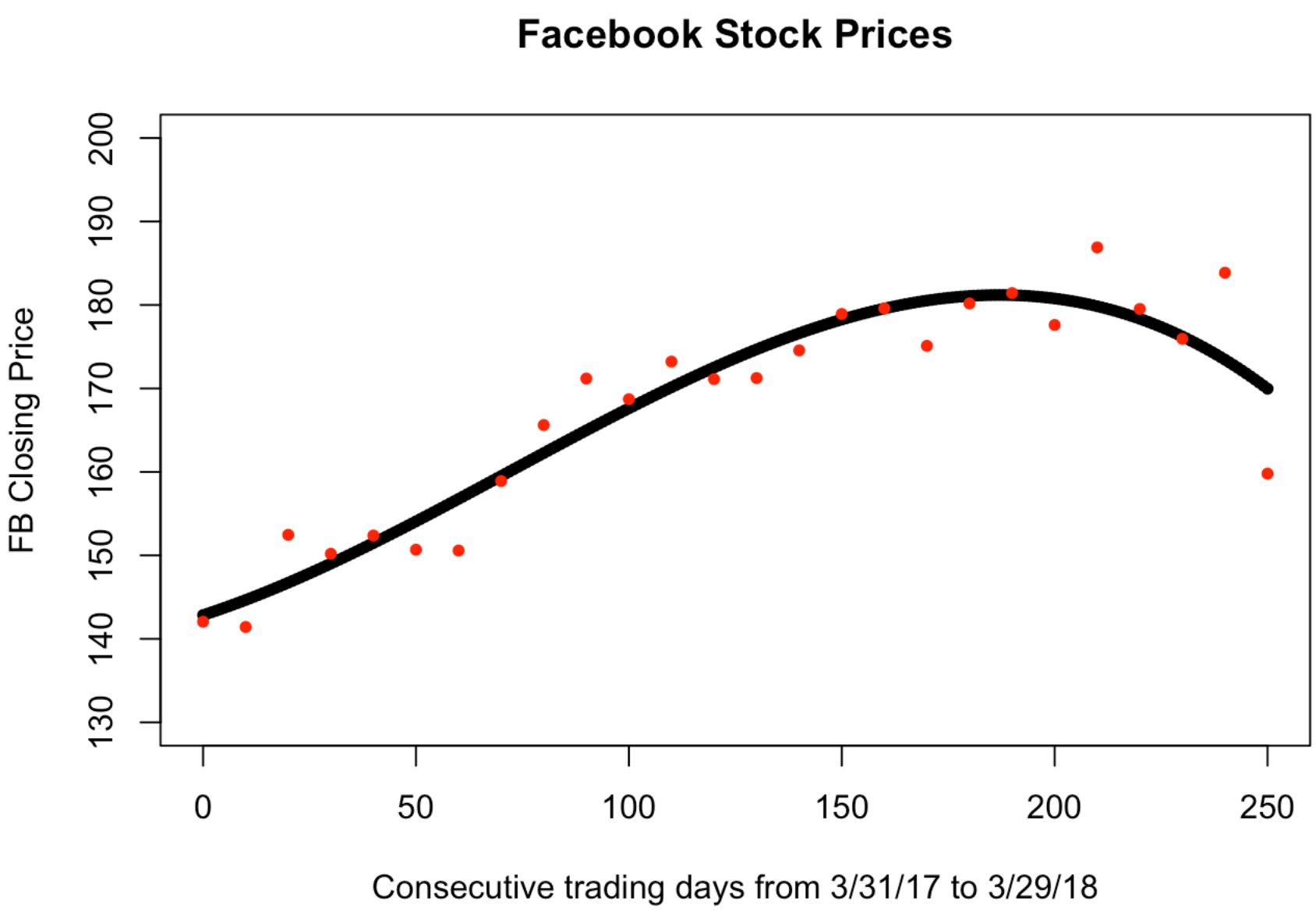
Solve the least squares problem to fit a regression line through the data. Do not use any statistical modeling commands like `lm` for this part of the problem; rather, you may want to (though you do not have to) use the function `qr.solve()` to solve the least squares problem. Write your code in the block below and uncomment the last line there to display your answer.

Part (c) solution

```
ones = rep(1, 26)

A = cbind(ones)
for(i in 1:3) {
  A = cbind(A, samp.days^i)
}

res = solve(t(A)%*%A, t(A)%*%samp.prices)
res
```

```
##                [,1]
## ones  1.428510e+02
##       1.666626e-01
##       1.505151e-03
##      -6.951815e-06
```

```
r = function(x) {
  res[1] + res[2]*x + res[3]*x^2 + res[4]*x^3
}

plotApproximations(r)
```



Part (d):

Aha! I found the full data after all. Turns out it was also right here this entire time, and it is stored as `all.prices`. All the days are stored in `days`. For each of the three methods above, compute the 2-norm of the vector of errors between the true values of the stock price and your estimated error. Do this by completing the first three lines in the code block below, and leave the following three lines to print the answers. Finally, on the last line, fill in the proper information to order the three methods from best to worst in terms of how they performed with the 2-norm.

Part (d) solution

```
x = seq(0, 250, length = 251)
poly.err <- vnorm(all.prices - p(x)) #REPLACE 0 WITH YOUR CODE
spline.err <- vnorm(all.prices - s(x)) #REPLACE 0 WITH YOUR CODE
reg.err <- vnorm(all.prices - r(x)) #REPLACE 0 WITH YOUR CODE
print(paste("2-norm of error for polynomial interpolation =",poly.err))
```

```
## [1] "2-norm of error for polynomial interpolation = 340926.674499865"
```

```
print(paste("2-norm of error for spline interpolation =",spline.err))
```

```
## [1] "2-norm of error for spline interpolation = 58.3603463531276"
```

```
print(paste("2-norm of error for linear regression =",reg.err))
```

```
## [1] "2-norm of error for linear regression = 76.1218093673002"
```

```
print(c("BEST METHOD: Natural Cubic Spline","NEXT BEST METHOD: Least Squares Fit","WO
RST METHOD: Interpolating Polynomial"))
```

```
## [1] "BEST METHOD: Natural Cubic Spline"
## [2] "NEXT BEST METHOD: Least Squares Fit"
## [3] "WORST METHOD: Interpolating Polynomial"
```

Problem 6

This question reveals the math behind an important machine learning algorithm called `Ridge Regression'.

Part a:

We will consider a dataset recording the average total SAT score of each state.

```
SAT = mosaicData::SAT[,c(2,3,4,5,8)]
SAT.S = scale(SAT)
```

I have extracted columns from the data to contain 4 input variables (expenditure per pupil in a state `expend`, pupil/teacher ratio `ratio`, teacher's average salary `salary`, and percent of eligible students taking SAT `frac`), and the 5th column is the predictor variable (total SAT score `sat`). I have also standardized the data, and called this scaled data `SAT.S`. (Check out both `SAT` and `SAT.S`. We would like to model the (scaled) SAT score as a linear function of each of the (scaled) inputs:

$$\texttt{sat} = x_1\,\texttt{expend} + x_2\,\texttt{ratio} + x_3\,\texttt{salary} + x_4\,\texttt{frac}$$

Part a(i) Set up a system of equations $Ax = b$ for this model, where the x represents the unknown coefficients of your model. Explain why your system is as you have determined. Note: this will be an overdetermined system.

Part a(i): Solution

```
A=cbind(SAT.S[,1], SAT.S[,2], SAT.S[,3], SAT.S[,4]) #Input the appropriate matrix her
e
b=SAT.S[,5] #Input the appropriate vector here
```

Part a(ii)

Use the normal equations to solve for the least squares solution of your system.

Part a(ii): Solution

```
#Insert code to solve for the least squares solution, call it sol
sol = solve(t(A)%*%A, t(A)%*%b)
#Uncomment below
print(sol)
```

```
##             [,1]
## [1,]  0.08128318
## [2,] -0.10977993
## [3,]  0.13006185
## [4,] -1.03889785
```

Part b:

To set up the Tikhonov regularization, we form the optimization problem

$$\min_{x\in\mathbb{R}^n}\left\{\frac{1}{2}||Ax-b||_2^2 + \frac{\lambda}{2}||x||_2^2\right\},$$

to solve for x, where λ is a positive constant that penalizes large magnitudes of x, the coefficients of the model. Notice if $\lambda = 0$, the solution to the ridge regression problem and the least squares solution will be the same.

Part b(i)

Using the same 3 parts as in HW 6 problem 3, explain why solving the above optimization problem can be reduced to solving the equation $(A^TA + \lambda I)x = A^Tb$ for x.

Part b(i): Solution

I encourage you to hand write this part out on paper.

Part b(ii)

Write a function for the ridge regression `ridge.reg` that takes in the parameters `A, b, lambda` . Find the ridge regression result when lambda is 10. Check your example by solving the ridge regression with the (scaled) SAT data and $\lambda = 50$. ### Part b(ii): Solution

```
ridge.reg = function(A,b,lambda){
  # Solve the system in part b(i) and store the solution in x
  # Insert your code here
  ata = t(A)%*%A
  n = nrow(ata)
  M = ata - lambda*diag(n)
  x = solve(M, t(A)%*%b)
  return(x)
}
test = ridge.reg(A, b, 10)
test
```

```
##            [,1]
## [1,]  0.4966513
## [2,] -0.1119375
## [3,]  0.1950540
## [4,] -1.6654998
```

```
r = ridge.reg(A,b,50) #Uncomment and run once function is written
r
```

```
##            [,1]
## [1,] -0.9828353
## [2,]  6.0885039
## [3,]  1.6336126
## [4,]  0.7381972
```

Part b(iii)

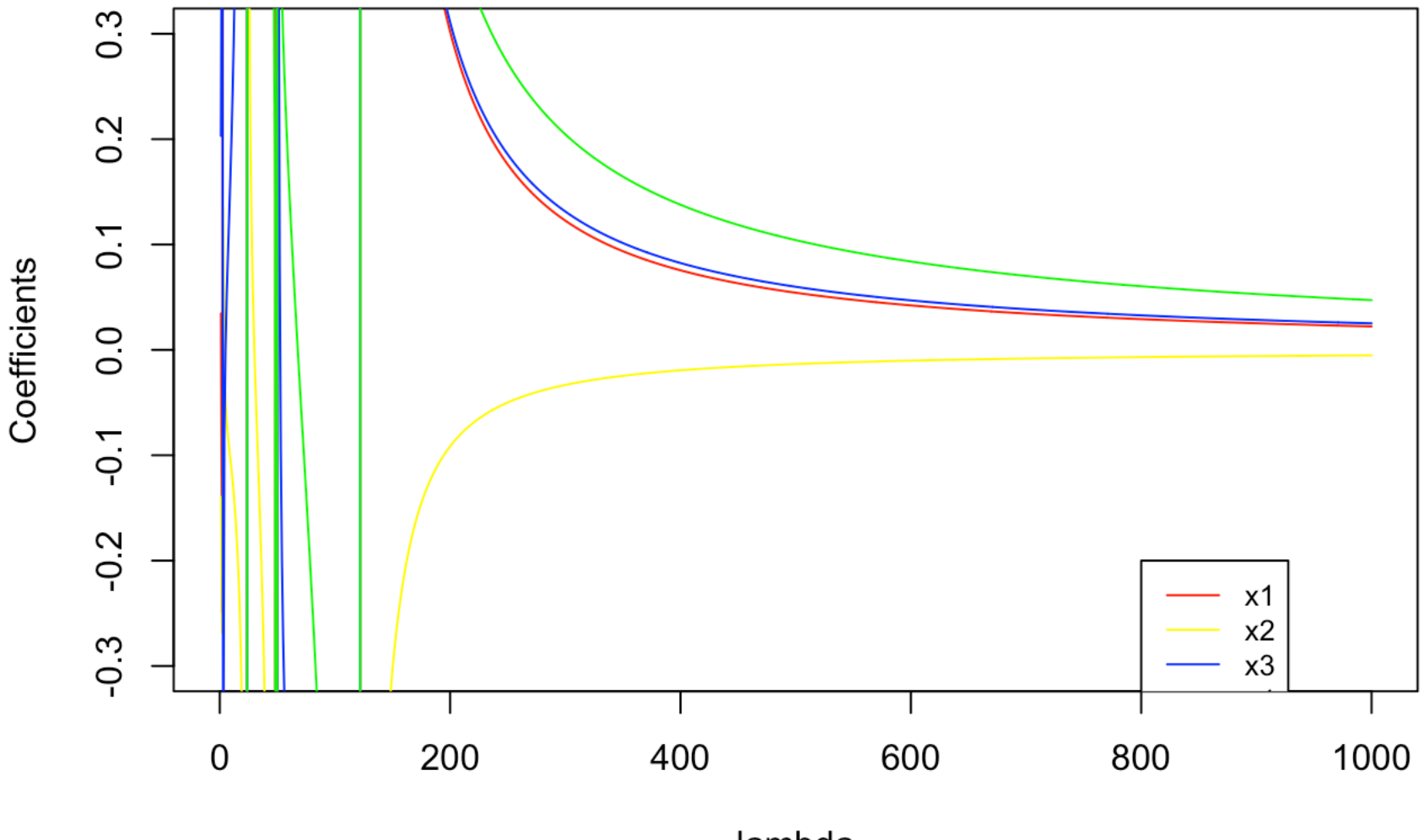
Plot the trend of each coefficient for λ ranging from 0 to 1000 (integer values). What do you observe (i.e. how does λ affect each coefficient)?

Part b(iii) Solution

```
result = c()
#Insert code here to compute result, which should be a 4 x 1001 matrix for each of th
e coefficients

for(i in 1:1000) {
  res = ridge.reg(A, b, i)
  result = cbind(result, res)
}

dim(result)
```

```
## [1]    4 1000
```

```
##Uncomment code below to plot
plot(result[1,],type = 'l',xlim=c(0,1000),ylim = c(-0.3,0.3),col = "red",xlab="lambda
",ylab="Coefficients" )
lines(result[2,],col = "yellow")
lines(result[3,],col = "blue")
lines(result[4,],col = "green")
legend(800,-0.2,legend=c("x1","x2","x3","x4"),col=c("red","yellow","blue","green"),lt
y=1,cex=0.8)
```



Insert your observation here

The lambda value affects each coefficient significantly, as it causes all the coefficients to converge for large values of lamda.

Part b(iv)

There is a balancing act in trying to solve this optimization problem. The $\frac{1}{2}||Ax-b||_2^2$ term wants to minimize the squared residual error, while the $\frac{\lambda}{2}||x||_2^2$ term wants to minimize the magnitude of x. We would like to determine the λ that leads to the lowest minimum of the sum of both terms. One way to do this is a common method called ``K-fold cross validation“. Below is an outline of how to perform a K-fold cross validation:

- Choose a value K and a λ. We will fix $K = 10$, and vary λ from the integers 0 to 100.
- Divide the data into K equal parts. For our state SAT data, we separate the 50 states into 10 groups g_n where $n = 1, 2, 3, \ldots, 10$, each having 5 data points.
- For each $\ell = 1, 2, 3, \ldots, 10$, `leave out' group g_ℓ in your data and fit the ridge regression model to all the other groups g_j where $j \neq \ell$ using parameter λ. Create the model using the coefficients that you find from the ridge regression, and use this model to predict a value for all the data points you left out in g_ℓ. Calculate the residual sum of squares $\sum_{i=1}^{5}(predict_i - b_i)^2$ where b_i is the actual scaled SAT score of the data points in g_ℓ.
- Now, you will have K residual sum of squares values, one for each `left out' group. Compute the means

of these K values. This is called the cross validation value $CV(\lambda)$.

- Repeat the process with a different λ.
- The `best' λ is the one with the lowest $CV(\lambda)$.

Implement $K = 10$-fold cross validation to find the best (integer) λ between 1 and 100.

Part b(iv) Solution

```
K = 10
BestLambda = 0 #Update this to be the optimal choice
bestError = 1000000 #Update this to the optimal choice (want it as close to 0 as poss
ible)
for (lambda in 1:100){
  error = 0
  for (i in 1:K){
    if(i>1 && i < K) {
      startVal = (i-1)*5
      endVal = i*5+1
      A1 = rbind(SAT.S[1:startVal,1:4])
      A2 = rbind(SAT.S[endVal:50, 1:4])
      AF = rbind(A1, A2)
      b1 = c(SAT.S[1:startVal, 5])
      b2 = c(SAT.S[endVal:50, 5])
      bf = c(b1, b2)
      x1 = ridge.reg(AF, bf, lambda)
    resSum = 0
    for(j in startVal:endVal) {
      coeffs = c(SAT.S[j, 1], SAT.S[j, 2], SAT.S[j, 3], SAT.S[j, 4])
      resSum = resSum + (coeffs%*%x1 - SAT.S[j, 5])^2
    }
    error = error + resSum
  }

  else if(i==1) {
    AF = SAT.S[6:50, 1:4]
    bf = SAT.S[6:50, 5]
    x1 = ridge.reg(AF, bf, lambda)
    resSum = 0
    for(j in 1:5) {
      coeffs = c(SAT.S[j, 1], SAT.S[j, 2], SAT.S[j, 3], SAT.S[j, 4])
      resSum = resSum + (coeffs%*%x1 - SAT.S[j, 5])^2
    }
    error = error + resSum
  }

  else {        #i = k loop
    AF = SAT.S[1:45, 1:4]
    bf = SAT.S[1:45, 5]
    x1 = ridge.reg(AF, bf, lambda)
    resSum = 0
```

```
      for(j in 46:50) {
        coeffs = c(SAT.S[j, 1], SAT.S[j, 2], SAT.S[j, 3], SAT.S[j, 4])
        resSum = resSum + (coeffs%*%x1 - SAT.S[j, 5])^2
      }
      error = error + resSum
    }
  }


  error = error/10      #10 residual sum of squares error
  if(error < bestError) {
    bestError = error
    BestLambda = lambda
  }
}
print(list(lambda = BestLambda,error = bestError))
```

```
## $lambda
## [1] 1
##
## $error
##          [,1]
## [1,] 1.294298
```

Part c

Observe two items

- $||x_a^*||_2^2 \geq ||x_b^*||_2^2$ where x_a^* corresponds to the optimal least squares solution in part (a) and x_b^* corresponds to the optimal ridge regression solution in part (b) using the best λ.
- $||Ax_a^* - b||_2^2$ $||Ax_b^* - b||_2^2$

Compute each of these quantities, and explain why this makes sense.

Part c Solution

```
#Update each of these four values
norm_xa= vnorm(sol)^2 #Solution for least squares found earlier
norm_xb= vnorm(ridge.reg(A, b, 1))^2
residual_xa= vnorm(A%*%sol - b)
residual_xb= vnorm(A%*%ridge.reg(A, b, 1))^2

print(paste("2-norm squared of least squares solution=",norm_xa))
```

```
## [1] "2-norm squared of least squares solution= 1.1148834202045"
```

```
print(paste("2-norm squared of ridge regression solution =",norm_xb))
```

```
## [1] "2-norm squared of ridge regression solution = 1.23867941803151"
```

```
print(paste("2-norm squared of residual for least squares solution =",residual_xa))
```

```
## [1] "2-norm squared of residual for least squares solution = 2.93196949148946"
```

```
print(paste("2-norm squared of residual for ridge regression solution =",residual_xb)
)
```

```
## [1] "2-norm squared of residual for ridge regression solution = 42.8137515379195"
```

Insert your explanation here

This makes sense because the optimal least squares solution is the closest solution to b in the column space of A, while the optimal ridge regression solution is the minimized x value that is close to b in the column space of A. So the x from the ridge regression should have a smaller magnitude (since it is minimized) than the least squares solution, but not be as close to b as the least squares solution, and thus have a larger residual.