Problem Set 7

Natalia Sarabia Vásquez

November 17, 2021

1 Problem 1

I solved this problem using my SCF account, following these steps:

1. I wrote an R script called ScriptP1.R
2. While in my local machine, I sent my script to my arwen account using scp.
3. I executed my script in SCF and saved the output.
4. I sent back my output to my computer. I will present the output.

Here are the instructions I followed on the command line:

```
# In my computer, send my script to SCF:
scp ScriptP1.R natalia_sarabia10@arwen.berkeley.edu:~/

# In my arwen account...
# Execute my script and save the output:
OMP_NUM_THREADS=1 R CMD BATCH ScriptP1.R ScriptP1.out

# See most recent files
ls -lt

# While in my local machine, sent back to my computer:
scp natalia_sarabia10@arwen.berkeley.edu:~/ScriptP1.out ~/Documents/STAT243/PS7/.
scp natalia_sarabia10@arwen.berkeley.edu:~/differences.rda ~/Documents/STAT243/PS7/.
```

Before I proceed, note that I ensured that A is invertible. I built A as $A = W^T W$ where the elements of the $n \times n$ matrix W are generated independently using rnorm(). I took n = 5000.

```
n <- 5000
# By doing this, we ensure that A is positive definite (the square):
W <- matrix(rnorm(n^2), n)

# By the documentation of the function:
# Given matrices x and y as arguments, return a matrix cross-product. This is
# formally equivalent to (but usually slightly faster than) the call t(x) %*% y
# (crossprod). Y = NULL is taken to be the same matrix as x.

# Therefore, it will be enough to do:
A <- crossprod(W)
```

```
include_graphics('SCF.png')
```

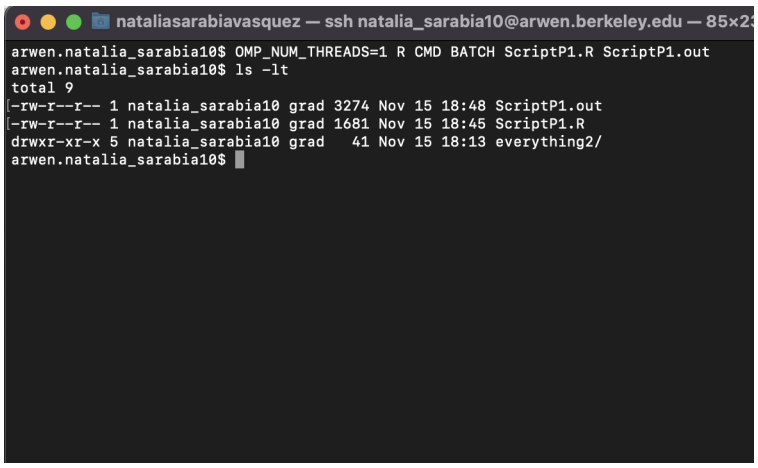


Figure 1: Execution in SCF

I will show the output and omit printing my script here. This because basically my ScriptP1.R is a subset of ScriptP1.out. Here the output from the SCF:

```
R version 4.1.2 (2021-11-01) -- "Bird Hippie"
Copyright (C) 2021 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> library(numbers)
>
> # Define the variables I will need:
> # Number of columns and rows of my matrix
> n <- 5000
>
> # Auxiliar matrix to construct A
> W <- matrix(rnorm(n^2), n)
>
> # I construct A as WtW. This assures me that A is invertible:
> A <- crossprod(W)
>
> # The construction of b is more general
```

```
> b <- rnorm(n)
>
> # a) Using a single thread, I will time each approach:
> # To compare the speed of $x = A^{-1}b $ using:
>
> # (i) solve(A)%*%b,
> system.time(
+   out1 <- solve(A)%*%b
+ )
   user  system elapsed
 35.370   0.901  36.275
>
> # (ii) solve(A,b), and
> system.time(
+   out2 <- solve(A,b)
+ )
   user  system elapsed
  9.361   0.352   9.715
>
> # (iii) Cholesky decomposition followed by solving triangular systems.
> # First, we compute the Cholesky decomposition:
> # U is upper-triangular
>
> system.time({
+   U <- chol(A)
+   out3 <- backsolve(U, backsolve(U, b, transpose = TRUE))
+ })
   user  system elapsed
  4.557   0.188   4.744
>
> # See how the results look for each approach:
> head(out1, n=4)
           [,1]
[1,] -157.26830
[2,]  -86.62308
[3,] -313.89732
[4,] -217.96256
> head(out2, n=4)
[1] -157.26830  -86.62308 -313.89732 -217.96256
> head(out3, n=4)
[1] -157.26830  -86.62308 -313.89732 -217.96256
>
> # b) Are the results the same numerically for methods ii) and iii) (up to machine epsilon)?
> # First, I compare that the results are the same for each approach:
> all.equal(as.vector(t(out1)), out2)
[1] TRUE
> all.equal(out2, out3)
[1] TRUE
>
> # I will use the function defined by the professor in the notes:
> dg <- function(x, digits = 20) formatC(x, digits, format = 'f')
>
> # Then, I can substract them:
```

```
> differences <- abs(out2 - out3)
>
> head(dg(differences), n=10)
 [1] "0.00000000502930674884" "0.00000011265015587O1" "0.00000000311410985887"
 [4] "0.00000000088405727183" "0.00000000333462857327" "0.00000000333614025294"
 [7] "0.00000000019705481691" "0.00000011174979047391" "0.00000000066762595452"
[10] "0.00000000382340203942"
>
> # Comparing to Machine.epsilon
> table(differences < .Machine$double.eps)

FALSE
 5000
>
> # I will compute the condition number. I need the biggest and the smallest eigenvalues of A.
> # First, eigenvalue decomposition for A:
> e <- eigen(A)
>
> # Norm2 for A is:
> norm2A <- max(e$val)
>
> # Norm2 for A^-1 is:
> norm2Ainv <- min(e$val)
>
> # Finally, the condition number is:
> condition <- norm2A / norm2Ainv
> condition
[1] 527693392
>
> # Save differences in rda format to export and analyze in local machine:
> save(differences, file = "differences.rda")
>
> proc.time()
   user  system elapsed
167.096   5.439 172.975
```

1.1 a) Using a single thread, how do the timing and relative ordering amongst methods compare to the order of computations we discussed in class and the notes? Note that if one works out the complexity of the full inversion using the LU decomposition, it is $\frac{4n^3}{3}$.

In class, we have discussed that for the three discussed methods, the order of computations are:
1. $\frac{4n^3}{3} + O(n^2)$, this is given in the problem description.
2. $\frac{n^3}{3} + O(n^2)$, reviewed in class.
3. $\frac{n^3}{6} + O(n^2)$, obtained in question 3 of this problem set.

If I order them, I will have: $\frac{4n^3}{3} > \frac{n^3}{3} > \frac{n^3}{6}$

Therefore, if I compare this ordering, I expect to see that the timing of each one will follow the same order. For convenience in the calculations, I will use as benchmark the time for method 2, because it will imply that I will just need to multiply it by 4 and divide it by 2, to compare to methods 1 and 3, respectively:

```
time_method1 <- 36.275
time_method2 <- 9.715
```

```
time_method3 <- 4.744

# I set time for method 2 as a benchmark:
time_method2/2
```

```
## [1] 4.8575
```

```
4 * time_method2
```

```
## [1] 38.86
```

And the resulting comparison is what I was expecting (roughly speaking):

$36.275 \approx 9.715 * 4 = 38.86$

$4.744 \approx 9.715/2 = 4.8575$

1.2 b) Are the results for x the same numerically for methods (ii) and (iii) (up to machine precision)? Comment on how many digits in the elements of x agree, and relate this to the condition number of the calculation.

First, if I compare the difference of my results to Machine.double.epsilon, I get that the magnitude of all my differences are actually bigger than this number.

Now, in class, we reviewed that the precision we would expect to have in our computer is up to 16 digits in general. If I take the results from ii) and iii) and then substract them, I see that the differences start around the ninth digit, in general, which corresponds to what I was expecting because of the condition number of the matrix. As my condition number is approximately 10^7, I have accuracy of order $10^{7-16} = 10^{-9}$ instead of 10^{-16}. I can think of the condition number as giving me the number of digits of accuracy lost during a computation relative to the precision of number on the computer:

```
# We know that in this case, the condition number is approx 10^7
527693392
```

```
## [1] 527693392
```

```
load(file.path('..','PS7/differences.rda'))

# Therefore, we would expect to have an accuracy of 10^(order of the condition-16) digits:
expected_accuracy <- 7-16
# Expected digits of accuracy:
abs(expected_accuracy)
```

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

```
# This goes in line with what I saw in the actual differences.
# The majority of them have 9 digits of accuracy:
dg <- function(x, digits = 20) formatC(x, digits, format = 'f')
head(dg(differences), n = 10)
```

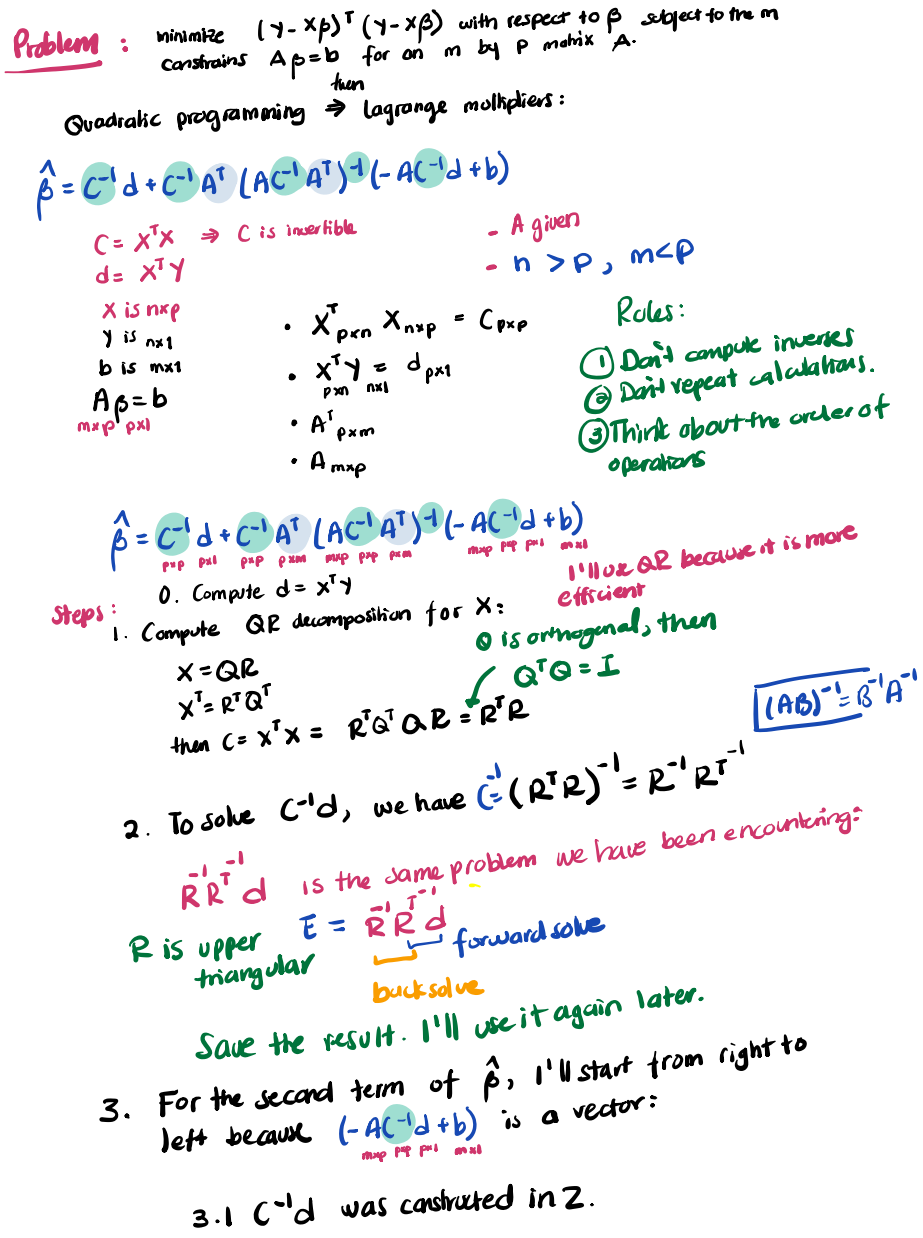
```
##  [1] "0.000000005029306748 84" "0.000000011265015587 01" "0.000000003114109858 87"
##  [4] "0.000000000884057271 83" "0.000000003334628573 27" "0.000000003336140252 94"
##  [7] "0.000000000197054816 91" "0.000000011749790473 91" "0.000000000667625954 52"
## [10] "0.000000003823402039 42"
```

```
# I counted the number of zeroes before the first digit different from zero and in fact in general,
# the digit different from zero starts after the 9th digit.
```

2 Problem 2

2.1 Part a)

I include my thinking process as part of the answer for this problem. When required, I will only use QR decomposition because, as we reviewed in class, is the most efficient given my constrains (I don't know the characteristics of A, which can be non-square or no positive definite, etc.):



3.2 Perform $-A$ (Result from 2) $+ b$ = vector g

4. For $(AC^{-1}A^T)^{-1}$ (mxp pxp pxm), first $C^{-1}A^T$ is $R^{-1}R^{-T}A^T$ (forward solve); save this into D F

Here, we can compute backward now, but note this:

$$AR^{-1}R^{-T}A^T$$
$$D^T \quad D$$

$$(R^{-T}A^T)^T = (A^{T}R^{-1T}) \text{ by properties} = AR^T = AR^{-1}$$

- $(AB)^T = B^TA^T$
- $(A^T)^T = A$ and
- $R^T = R^{-1}$ for orthogonal matrices.

Finally, $(AC^{-1}A^T)^{-1} = (D^TD)^{-1}$. We can decompose this one and

Chol(D^TD) or QR(D)

We have $(AC^{-1}A^T)^{-1}$ (Result of 3) $= (D^TD)^{-1}$ (Result of 3) (vector)

$= (R^TQ^TQR)^{-1}$ (Result of 3) (vector)

$= R^{-1}R^{-T}$ (vector)

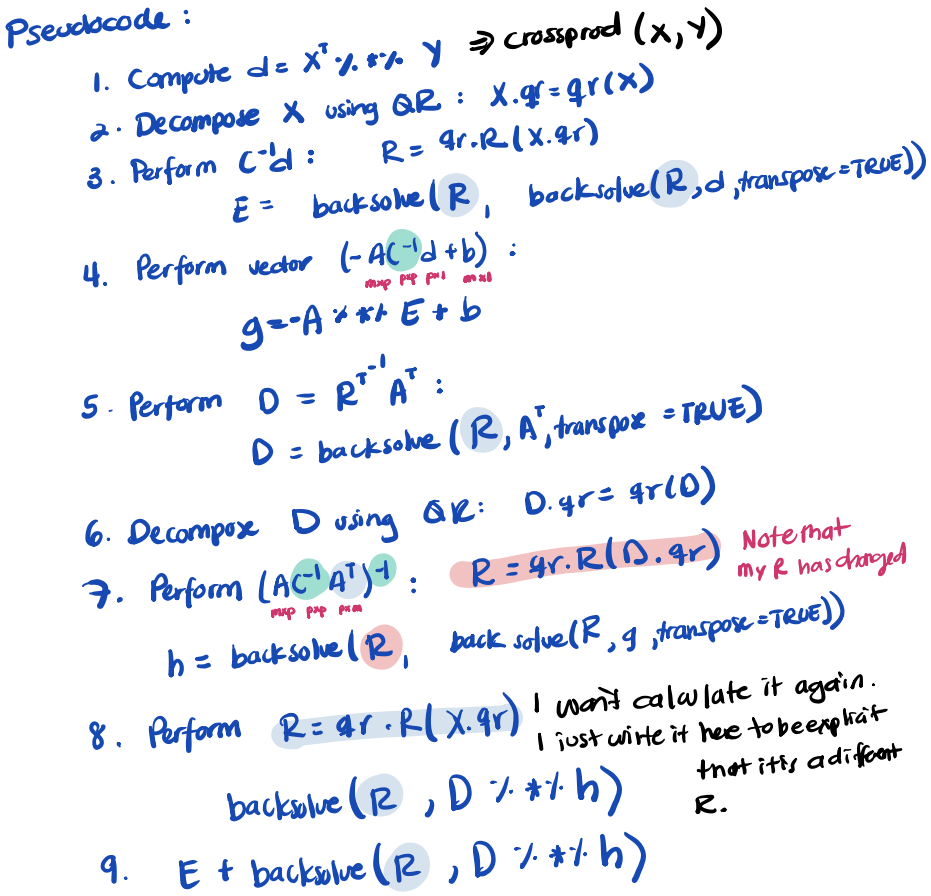
forwardsolve

backsolve

5. For $C^{-1}A^T$ (Result of 4) (vector), $R^{-1}D$ (Result of 4) (vector)

D from 4; product, compute

backsolve



2.2 Part b)

After setting up my code, I was having some issues to produce my output. I was advised by the professor to be careful with the dimensions and observe that: n needs to be bigger than p and m less than p. Knowing

this, I set up my numbers and ran my code:

```
set.seed(1)

# Define the dimensions of the matrices according to my talk with the professor:
n <- 25
m <- 15
p <- 20

# Create my matrices and vectors:
A <- matrix(rnorm(m*p),ncol = p, nrow = m)
X <- matrix(rnorm(n*p),ncol = p, nrow = n)
Y <- rnorm(n)
b <- rnorm(m)

# Define my function:
constrained_ls <- function(A,X,Y,b){
  # 1. Compute Xt Y
  d <- crossprod(X,Y)

  # 2. QR decomposition of X
  X.qr <- qr(X)

  # 3. Perform C-1 d
  R.x <- qr.R(X.qr)
  E <- backsolve(R.x, backsolve(R.x, d, transpose = TRUE))

  # 4. Perform -A C-1 d + b
  g <- -A %*% E + b

  # 5. Perform D
  D <- backsolve(R.x,t(A),transpose = TRUE)

  # 6. QR decomposition of D
  D.qr <- qr(D)

  # 7. Perform (A C-1 At)-1
  R.d <- qr.R(D.qr)
  h <- backsolve(R.d, backsolve(R.d,g,transpose = TRUE))

  # 8. Perform the sum
  result <- E + backsolve(R.x, D %*% h)

  return(result)
}

# Test the function:
yes243result <- constrained_ls(A,X,Y,b)

# See how it looks like
head(yes243result, n = 10)
```

```
##              [,1]
##  [1,] -0.15308129
##  [2,] -0.05638649
```

```
##  [3,]  0.16869113
##  [4,]  0.64318520
##  [5,]  0.79025015
##  [6,]  0.34221304
##  [7,]  0.11621405
##  [8,] -0.07814198
##  [9,]  0.19399982
## [10,]  0.97645559
```

```
# Performing the results by "hand":
# Save some recurrently used operations:
C_inv <- solve(crossprod(X))
d <- crossprod(X,Y)

# Perform the calculation
not243result <- C_inv %*% d + (C_inv %*% t(A) %*%
                                 solve(A %*% C_inv %*% t(A)) %*% ((-A %*% C_inv %*% d)+ b))

head(not243result, n = 10)
```

```
##              [,1]
##  [1,] -0.15308129
##  [2,] -0.05638649
##  [3,]  0.16869113
##  [4,]  0.64318520
##  [5,]  0.79025015
##  [6,]  0.34221304
##  [7,]  0.11621405
##  [8,] -0.07814198
##  [9,]  0.19399982
## [10,]  0.97645559
```

```
# Compare that I have the result under both approaches:
all.equal(yes243result,not243result)
```

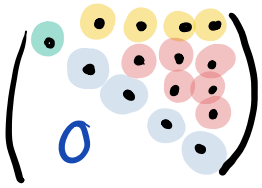
```
## [1] TRUE
```

In the last chunk, I made the calculations by "hand" just to verify that my function is working properly. When comparing my results, I actually obtained the same results under both approaches. I called 'yes243result' to the one using matrix decomposition and 'not243results' to the one computing inverses.

3 Problem 3

Work out the operation count (total number of multiplications plus divisions) for the Cholesky decomposition:

Cholesky Algorithm:



1. $U_{11} = \sqrt{A_{11}}$
2. For $j = 2, \dots, n$ $\quad U_{1j} = A_{1j} / U_{11}$
3. For $i = 2, \dots, n$
   a) • $U_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} U_{ki}^2}$
   b) • for $j = i+1, \dots, n$ $\quad U_{ij} = \dfrac{\left(A_{ij} - \sum_{k=1}^{i-1} U_{ki} U_{kj}\right)}{U_{ii}}$

First step: 0

Second step: $(n - 2 + 1) = (n-1)$ divisions.

Third step:

a) For the diagonal:

$$\sum_{i=2}^{n} (i - 1 - 1 + 1)(1)$$

$$= \sum_{i=2}^{n} (i-1) = \left[\frac{n(n+1)}{2} - 1\right] - (n-2+1)$$

$$= \frac{n(n+1)}{2} - \frac{2}{2} - \frac{2(n-1)}{2}$$

$$= \frac{n(n+1) - 2(1 + n - 1)}{2}$$

$$= \frac{n(n+1-2)}{2} = \frac{n(n-1)}{2}.$$

b) First, note that i cannot be n, I am working with values that are not in the diagonal., then:

$$i = 2, \dots, n-1$$
$$j = i+1, \dots, n$$

$$U_{ij} = \frac{\left(A_{ij} - \sum_{k=1}^{i-1} U_{ki} U_{kj}\right)}{U_{ii}}$$

The numbers of calculations here depend on i and j.

Basically, I'll have as many products as:

$$\sum_{k=1}^{i-1} (1) = (i-1-1+1) = i-1$$

(1 product; $i-1$ ↑ this many times)

But for each term, I'll also have 1 division.

Therefore, in total, for each combination of i and j, I have $i-1+1 = i$ operations.

Now, I have to sum across all the combinations of i and j:

$$\sum_{i=2}^{n-1} \sum_{j=i+1}^{n} i = \sum_{i=2}^{n-1} i\,(n-i-1+1)$$

$$= \sum_{i=2}^{n-1} i\,(n-i) = \sum_{i=2}^{n-1} in - \sum_{i=2}^{n-1} i^2 \quad (1)$$

Facts:

- $\sum_{i=2}^{n-1} i + 1 - 1 = \sum_{i=1}^{n-1} i - 1 = \frac{(n-1)(n)}{2} - 1$
- $\sum_{i=2}^{n-1} i^2 + 1 - 1 = \sum_{i=1}^{n-1} i^2 - 1$

$2n-2+1 = 2n-1$

$$= \frac{(n-1)(n)(2(n-1)+1)}{6} - 1 = \frac{(n-1)(n)(2n-1)}{6} - 1$$

$$= \frac{(n-1)(n)(2n-1) - 6}{6}$$

$$= \sum_{i=2}^{n-1} in - \sum_{i=2}^{n-1} i^2 \quad (1)$$

$$= n\left[\frac{(n-1)(n)}{2} - 1\right] - \left[\frac{(n-1)(n)(2n-1)-6}{6}\right]$$

$$= \frac{3n^2(n-1) - 6n - (n-1)(n)(2n-1) + 6}{6}$$

$$= \frac{3n^2(n-1) - 6(n-1) - (n-1)(n)(2n-1)}{6}$$

$$= \frac{(n-1)(3n^2 - 6 - (n)(2n-1))}{6}$$

$$= \frac{(n-1)(3n^2 - 6 - 2n^2 + n)}{6}$$

$$= \frac{(n-1)(n^2 - 6 + n)}{6}$$

$$= \frac{(n^3 - 6n + n^2 - n^2 + 6 - n)}{6}$$

$$= \frac{n^3 - 7n + 6}{6} = \frac{n^3}{6} - \frac{7n}{6} + 1$$

In total, for the three steps, I have:

$$(n-1) + \frac{n(n-1)}{2} + \frac{n^3}{6} - \frac{7n}{6} + 1$$

$$= n - 1 + \frac{n^2}{2} - \frac{n}{2} + \frac{n^3}{6} - \frac{7n}{6} + 1$$

$$= n\left(\frac{6-3-7}{6}\right) + \frac{n^2}{2} + \frac{n^3}{6}$$

$$= n\left(-\frac{4}{6}\right) + \frac{n^2}{2} + \frac{n^3}{6}$$

$$= \frac{1}{6}\left(n^3 + 3n^2 - 4n\right)$$ multiplications and divisions.

Compare your result to that given in the notes.

In the notes, we are told that the $\frac{n^3}{6} + O(n^2)$. My results are consistent with the computational complexity (order) discussed in class (I obtained the same order of operations $O(n^3)$, although more detailed).

4 Extra credit

Extra -

Consider A symmetric. By definition, I have:

$$\|A\|_2 := \sup_{z:\|z\|_2=1} \|Az\|.$$

Now, taking the singular value decomposition of A.

$$= \sup_{z:\|z\|_2=1} \|USV^T z\| \quad \cdots (1)$$

Recall that U, V are orthogonal

Consider $\|USV^T z\|$, U is orthogonal so that $\|U\| = 1$ then $\|USV^T z\| = \|SV^T z\|$. As V is orthogonal, $\|V^T z\| = \|z\| = 1$. Let $w = V^T z$, I have:

$$(1) := \sup_{z:\|z\|_2=1} \|Sw\|$$

By definition, S is a diagonal matrix that has the singular values of A in the diagonal, ordered in decreasing order. Remember that by definition also, the singular values of A are the roots of the eigenvalues of A^TA.

Therefore, the product Sw will have the form $\begin{pmatrix} \lambda_1 w_1 \\ \vdots \\ \lambda_n w_n \end{pmatrix}$

And then, I am trying to find the supreme of $\sqrt{\sum_{i=1}^{n} \lambda_i^2 w_i^2}$ under the constrain that $\sum w_i^2 = 1$.

Because of this, the maximum is taking $w = e_1$, the first canonical vector (in other words, taking the max singular value, and making the rest of them zero, thus I'll also have $\sum w_i^2 = 1$).

$\|A\|_2^2 = \sup_{z : \|z\|_2 = 1} \sum_{i=1}^{n} \lambda_i^2 w_i^2 = \lambda_{max}^2$. Taking the square: *square root

$$\|A\|_2 = \sqrt{\lambda_{max}^2} = |\lambda_{max}| \quad \blacksquare$$

That is, the matrix norm induced by the usual l_2 vector norm is the largest of the eigenvalues of A symmetric.