Problem Set 7

Alexander Brandt
SID: 24092167

November 16 2015 (Extension Request E-mail Included)

Friendly Collaborators: Milos Atz, Alex Ojala and Guillame Baquiast.

1

1. What are the goals of their simulation study and what are the metrics do they consider in assessing their method?

   The purpose of the simulation study is to assess the accuracy of the proposed asymptotic approximation in finite samples and to examine the power of the EM test, in determining the number of constituient normal distributions in a data set, by using SLC activity in RBCs, adulteration in wine production, and differential gene expression in cancerous and healthy patients. Type I error and the power are the metrics considered.

2. What choices did the authors have to make in designing their simulation study? What are the key aspects of the data generating mechanism that likely affect the statistical power of the test?

   The authors had a choice of mixing fractions, number of normal distributions to inclue (as well as their means and variances), and the number of data points. Sample size has obvious implications on statstical power, but "distinct-ness" (based one mean and variance), as well as the mixing fraction, have implications as well.

3. Suggest some alternatives to how the authors designed their study. Are there data-generating scenarios that they did not consider that would be useful to consider?

   Perhaps the ability to fit various skewed normal distributions. Or very overlapped distributions.

4. Give some thoughts on how to set up a simulation study for their problem that uses principles of basic experimental design, or if you think it would be difficult, say why.

   How does the model perform under different distributions? How robust is the model w/ respect to breaking assumption. Be more ambitious with respect to the mixing fraction.

5. Do their figures/tables do a good job of presenting the simulation results and do you have any alterantive suggestions for how to do this? Do the authors address the issue of simulation uncertainty/simuilation statndard errors and/or do they convince the reader they've done enough simulation replications?

   I think the presentation is well done, but label your axes, please. The tables showing the various p-values for different m = 2 or 3 are especially effective. The box plots are a nice touch as well.

6. Interpret their tables on power (Tables 4 and 6) - do the results make sense in terms of how the power varies as a function of the data generating mechanism?

The EM values increase with iteration, which is pretty much assured by the algorithm. It'd be nice to know how the mixing fraction differences. I'm not sure if the dimensionality of the tests lends itself especially well to table formats...

7. Discuss the extent to which they follow JASA's guidelines on simulation studies (see the end of the Unit 10 class notes for the JASA guidelines).

   I would like to see more discussion about the accuracy (perhaps using a real dataset?). There is no discussion of pseudorandom-number generation, numerical algorithms, or computers. Programming languages and major software components are discussed. P.S. Dude, where's my code? That should always be a criterea for publishing...

2

2.1

The number of multiplications is $\sum_{n=2}^{N}(-n^2 + n(N+1) - 1)$, which simplifies, for the first two terms, to:

$$Ops \approx \frac{1}{6}n^3 + \frac{1}{2}n^2$$

This fits with Chris' notes.

Guidence/Confirmation Citation: Floating Point Operations in Matrix-Vector Calculus (v 1.3), Raphael Hunger. 2007.

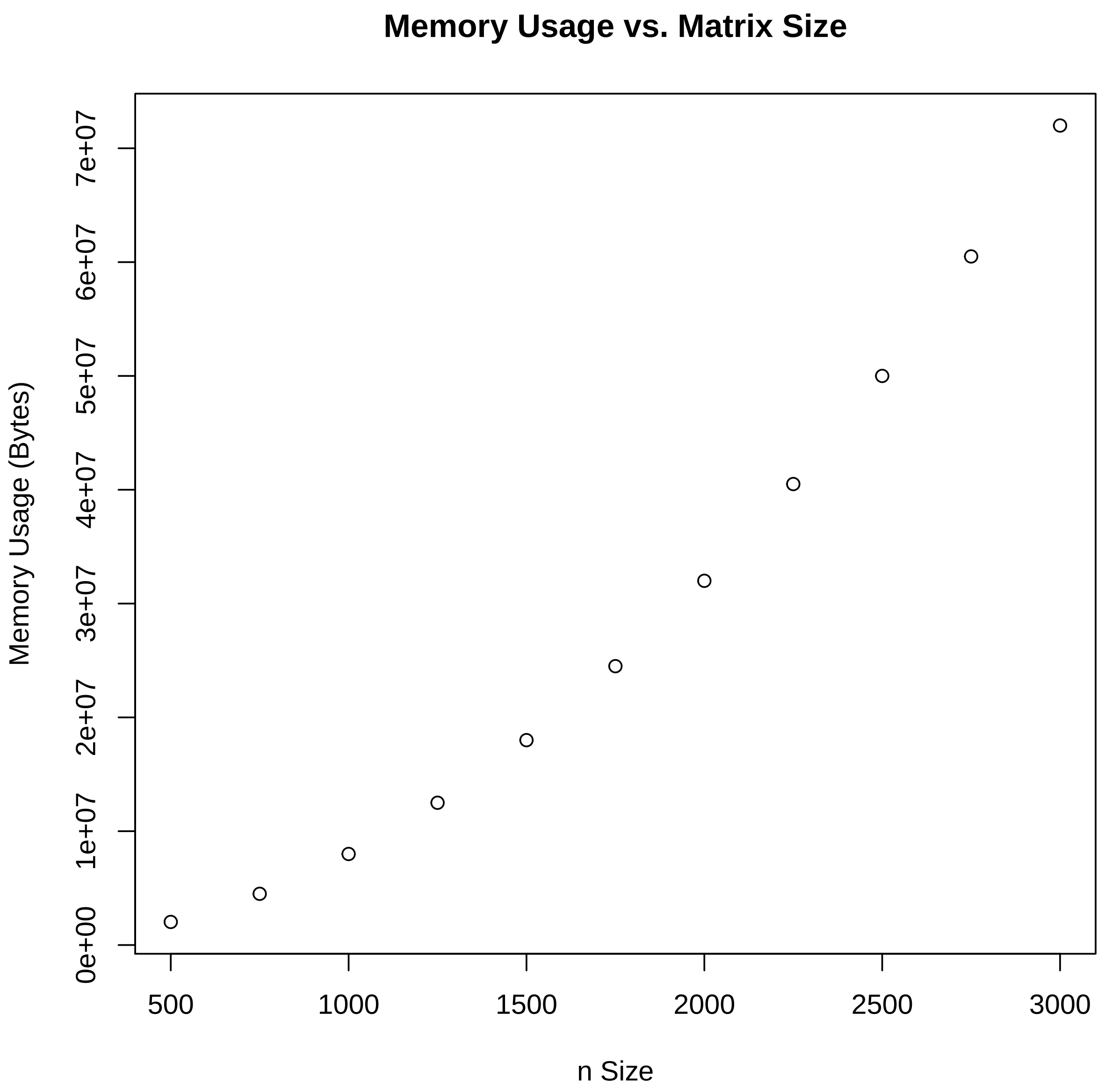
2.2

You can overwrite as you go along. This is because the Cholesky decomposition works by moving "down" the matrix (i.e. never refers to the preceeding rows). The animation Chris showed in class (or maybe I just saw it on The YouTube trying to figure out part a)?) provides a graphical intution about this.

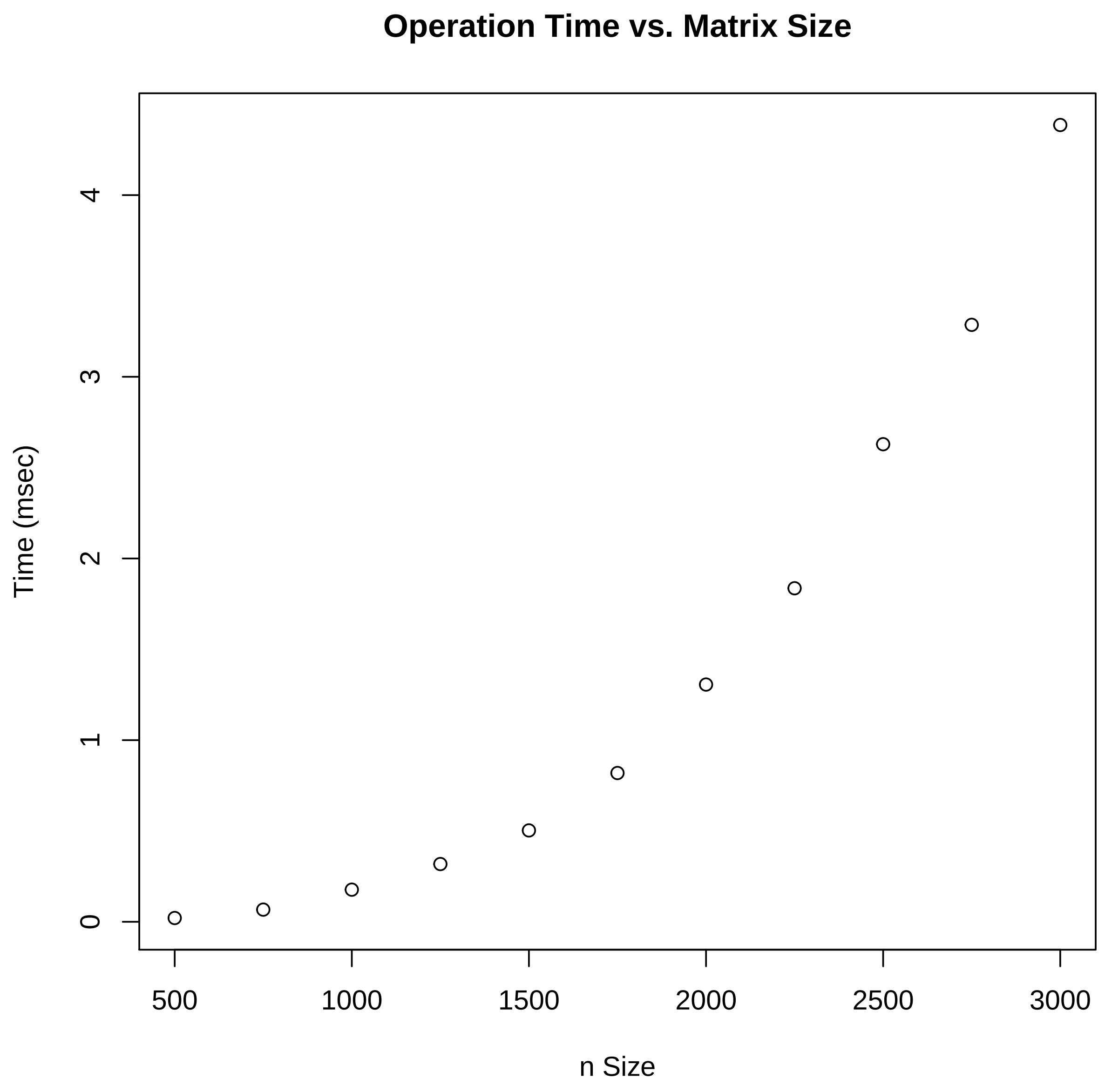
2.3

```r
library(pryr)
# Initialize our arrays for later plotting
sizes <- seq(500,3000,by=250)
memories <- rep(0,length(sizes))
times <- rep(0,length(sizes))

for (i in seq(length(sizes)))
{
    # Construct the matrix
    X <- crossprod(matrix(rnorm(sizes[i]^2), sizes[i]))
    # Find out memory use before
    before <- mem_used()
    # Microbenchmark our cholesky decomposition
    times[i] <- system.time(cX <- chol(X))[3]
    # Find out memory use after
    memories[i] <- mem_used() - before
    rm(cX,X) # To clear before the next usage!
}
plot(x=sizes,y=memories,xlab="n Size",ylab="Memory Usage (Bytes)",main="Memory Usage vs. Matrix Size")
```



```
plot(x=sizes,y=times,xlab="n Size",ylab="Time (msec)",main="Operation Time vs. Matrix Size")
```



The values are scaling about $O(n^3)$ which is roughly what we expect given the algorithm.

3

```r
set.seed(0)
options(digits=22)
N <- 5000
# Initialize the pos-def matrix
X <- crossprod(matrix(rnorm(N^2), N))
y <- rnorm(N)

print("For the solve(X), then %*% method...")
```

```
## [1] "For the solve(X), then %*% method..."
```

```r
system.time(v1 <- solve(X) %*% y)
```

```
##                        user                    system
## 250.449999999999886313162   0.956000000000001829648
##                     elapsed
## 252.209000000000316049409
```

```r
print("For the solve(X,y)...")
```

```
## [1] "For the solve(X,y)..."
```

```r
system.time(v2 <- solve(X,y))
```

```
##                       user                    system
## 28.146999999999913598003  0.112999999999999893419
##                    elapsed
## 28.325000000000454747351
```

```r
print("For the Cholesky decomposition...")
```

```
## [1] "For the Cholesky decomposition..."
```

```r
system.time(
  {
    # Cholesky decompose, and then backsolve
    U <- chol(X);
    v3 <- backsolve(U, backsolve(U, y, transpose=TRUE));
  }
)
```

```
##                       user                    system
## 19.690999999999740794010  0.121999999999998863132
##                    elapsed
## 20.088000000000222826202
```

```r
print("Checking that our values are reasonable (i.e., the same)...")
```

```
## [1] "Checking that our values are reasonable (i.e., the same)..."
```

```r
v1[1:10]
```

```
##  [1]  -4.803833033478003500250  17.758187412524087278598
##  [3] -19.910537805559485491358 -14.628034395977977055736
##  [5]  32.546734117480198733574  32.251918358480054394022
##  [7]  34.282461021149927660190 -73.309490570436381062791
##  [9] -20.014343424005812011046  -1.749447230489489468752
```

```r
v2[1:10]
```

```
##  [1]  -4.803833033477777902931  17.758187412523792403363
##  [3] -19.910537805559318513815 -14.628034395978076531719
##  [5]  32.546734117480127679300  32.251918358480033077740
##  [7]  34.282461021149750024506 -73.309490570436068423987
##  [9] -20.014343424005993199444  -1.749447230489707516554
```

```r
v3[1:10]
```

```
##  [1]  -4.803833043393967550116  17.758187421434300290457
##  [3] -19.910537817897626666763 -14.628034398574007113325
##  [5]  32.546734139602186530738  32.251918375158908247613
##  [7]  34.282461045482996553346 -73.309490601046292113097
##  [9] -20.014343430242931276553  -1.749447229275409743110
```

...Cholesky is defintely the fastest! a) $O(n^3)$ b) $1/3n^3 + O(n^2)$ c) $1/6n^3 + O(n^2)$ (source: the notes). The timings generally agree.

3.1

```
library(MASS)
print("Numerical differences between our answers")

## [1] "Numerical differences between our answers"

# Max of differences between methods 1, 2 and 2, 3 and 3, 1 respectively
max(abs(v1-v2))

## [1] 7.105427357601001858711e-13

max(abs(v2-v3))

## [1] 6.342163771932973759249e-08

max(abs(v3-v1))

## [1] 6.342166614103916799965e-08

a <- norm(X,type="2") * norm(ginv(X),type="2")
print("The Condition Number")

## [1] "The Condition Number"

print(a * 1e-16)

## [1] 4.029048851270799130557e-09
```

The differences are really not that pronounced. This agrees with the relative numerical stability assured by the condition number.

4

The general scheme and pseudocode for the matrix manipulations used was as follows:

Given:

$$\hat{\beta} = \left(X^T \Sigma^{-1} X\right)^{-1} X^T \Sigma^{-1} Y$$

We use eigenvectors and eigenvalues to rewrite sigma inverse as:

$$\Sigma^{-1} = C \Lambda C^{-1}$$

Which we can further simply to:

$$\Sigma^{-1} = P P^T$$

Where:

$$P = C \Lambda^{1/2}$$

This allows us to transform X and Y, so the QR decomposition can be performed, so that our calculation is of an acceptable speed:

$$X_* = P^T X$$

and

$$Y_* = P^T Y$$

Now we follow a QR decomposition, much like in the notes with OLE:

$$X_*^T X_* \hat{\beta} = X_*^T Y$$

$$R^T Q^T Q R \hat{\beta} = R^T Q^T Y_*$$

And we can use back solving to find:

$$R\hat{\beta} = Q^T Y_*$$

```
library(MASS)
n <- 50
p <- 10

X <- matrix(runif(n*p), ncol=p)
Sigma <- crossprod(matrix(rnorm(n^2), n))
Y <- matrix(runif(n), ncol=1)

my_gls <- function(X, Sigma, Y, n, p) {
    # Generate sigma inverse
    Sigma_inv <- ginv(Sigma)
    # Eigendecompose sigma inverse
    Sigma_inv_eigen_object <- eigen(Sigma_inv)
    P <- Sigma_inv_eigen_object$vectors %*%
      diag(sqrt(Sigma_inv_eigen_object$values))
    Pt <- t(P)
    # Generate our Y/X "stars" form the pseudocode
    Y_mod <- Pt %*% Y
    X_mod <- Pt %*% X
    # Perform a QR decomposition
    X_mod.qr <- qr(X_mod)
    Q <- qr.Q(X_mod.qr)
    R <- qr.R(X_mod.qr)
    # Backsolve, and return the values
    return(backsolve(R, t(Q) %*% Y_mod))
}

print(my_gls(X, Sigma, Y, n, p))

##                             [,1]
##  [1,] -0.38178770437887171729230
##  [2,]  0.43199236710490312995958
##  [3,] -0.06137662709808201083606
##  [4,]  0.60279399525459143038830
##  [5,] -0.47983751521344436552496
##  [6,] -0.01672323669622955907843
##  [7,] -0.14536684054820322997692
##  [8,] -0.03198516007207349159946
##  [9,]  0.03981707405928997661748
## [10,]  1.07242103894418416842882

# Checking our answer against the "long" way...
# This was the problem statement (but too slow for our solution)

Sigma_inv <- ginv(Sigma)
ginv(t(X) %*% Sigma_inv %*% X) %*% t(X) %*% Sigma_inv %*% Y

##                            [,1]
##  [1,] -0.38178770437844999907639
##  [2,]  0.43199236710481431211761
##  [3,] -0.06137662709890697593229
##  [4,]  0.60279399525442456386770
##  [5,] -0.47983751521576090137700
##  [6,] -0.01672323669648601018878
##  [7,] -0.14536684054862153425702
##  [8,] -0.03198516006718693516220
##  [9,]  0.03981707406077009125944
## [10,]  1.07242103894115814455290
```

5

5.1

Suppose we write X as:

$$X = U_{nxn} S_{nxp} V^T_{pxp}$$

Manipulate further:

$$\begin{aligned} X^T &= (USV^T)^T \\ &= VS^T U^T \\ X^T X &= VS^T U^T USV^T \\ &= VS^T ISV^T \end{aligned}$$

S is diagonalizable, so $S^T = S$:

$$X^T X = VS^2 V^T$$

So our eigenvectors are V and V^T – the right singular vecotrs of X. We can also see that S^2 are the eigenvalues – the square of the singular values of X.

We now show that XX^T is positive semidefinite. Take v s.t. $v_i \neq 0 \forall i$:

$$\begin{aligned} v^t X^T X v &= (vX)^T X v \\ &= s^T s \\ &= \sum s_i^2 \geq 0 \end{aligned}$$

Sources: Dug through some econometrics notes as well as a linear algebra textbook my roommate had lying around.

```r
n <- 1000
p <- 500

X <- matrix(runif(n*p), ncol=p)
eigen_object <- eigen(t(X) %*% X)
X_svd <- svd(X)

# Are the right singular vectors of X equal to
# the eigenvectors of X^t * X?

print("Right Singular Vectors of X...")

## [1] "Right Singular Vectors of X..."

head(X_svd$v[,1:10])

##             [,1]          [,2]         [,3]        [,4]         [,5]
## [1,] -0.04431472 -0.0713039531  0.015694327 -0.09116894  0.039093598
## [2,] -0.04550727  0.0987448093 -0.049431463  0.06035542 -0.013809752
## [3,] -0.04475009  0.0550687752 -0.006895094 -0.04915989  0.019327870
## [4,] -0.04464549  0.0653306270  0.025773791  0.02762509 -0.029110710
## [5,] -0.04390919 -0.0084620402  0.092150351 -0.07410252 -0.052462206
## [6,] -0.04438632  0.0009634196 -0.098575350  0.04329162 -0.008783357
##              [,6]         [,7]        [,8]          [,9]        [,10]
## [1,] -0.008111073  0.023962732 -0.10558649  2.613512e-02 -0.033087289
## [2,]  0.025553020  0.001906576  0.02446195 -4.220050e-03 -0.053928820
## [3,]  0.048226365 -0.094035323 -0.02009505  7.209434e-06  0.032340415
## [4,]  0.020982967 -0.035142588  0.02920922 -6.258991e-02 -0.009316251
## [5,]  0.040685745 -0.034828816  0.04830455 -5.505051e-02 -0.027224582
## [6,]  0.051545496 -0.024424176  0.02181362  8.102247e-02 -0.025520552

print("Eigenvectors of X^t * X")

## [1] "Eigenvectors of X^t * X"

head(eigen_object$vectors[,1:10])

##             [,1]          [,2]         [,3]        [,4]         [,5]
## [1,] -0.04431472  0.0713039531  0.015694327 -0.09116894 -0.039093598
## [2,] -0.04550727 -0.0987448093 -0.049431463  0.06035542  0.013809752
## [3,] -0.04475009 -0.0550687752 -0.006895094 -0.04915989 -0.019327870
## [4,] -0.04464549 -0.0653306270  0.025773791  0.02762509  0.029110710
## [5,] -0.04390919  0.0084620402  0.092150351 -0.07410252  0.052462206
## [6,] -0.04438632 -0.0009634196 -0.098575350  0.04329162  0.008783357
##              [,6]         [,7]        [,8]          [,9]        [,10]
## [1,] -0.008111073  0.023962732  0.10558649 -2.613512e-02 -0.033087289
## [2,]  0.025553020  0.001906576 -0.02446195  4.220050e-03 -0.053928820
## [3,]  0.048226365 -0.094035323  0.02009505 -7.209434e-06  0.032340415
## [4,]  0.020982967 -0.035142588 -0.02920922  6.258991e-02 -0.009316251
## [5,]  0.040685745 -0.034828816 -0.04830455  5.505051e-02 -0.027224582
## [6,]  0.051545496 -0.024424176 -0.02181362 -8.102247e-02 -0.025520552

# Are the eigenvalues of X^t * X equal to the
# squares of the singular values of X?

print("Eigenvalues of X^t * X")
```

```
## [1] "Eigenvalues of X^t * X"

head(eigen_object$values)

## [1] 125215.5298    239.2660    237.1677    235.4653    231.5683    228.7211

print("Squared singular values of X")

## [1] "Squared singular values of X"

head(X_svd$d**2)

## [1] 125215.5298    239.2660    237.1677    235.4653    231.5683    228.7211

# Is X' * X a positive semidefinite matrix?

library("matrixcalc")
print("Is X' * X a positive semidefinite matrix?")

## [1] "Is X' * X a positive semidefinite matrix?"

is.positive.semi.definite(t(X) %*% X)

## [1] TRUE
```

5.2

Now we consider an n x n positive semi-definite matrix X and assume we have already computed the eigendecomposition.

$$\begin{aligned} XDX^T + Ic &= XDX^T + XcX^T \\ &= X(D + cI)X^T \end{aligned}$$

Some code illustrating our proof(s):

```
N <- 10
X <- crossprod(matrix(rnorm(N^2), N))
# Store our eigenvalues for later
eigen_objects <- eigen(X)

# Create Z from X and cI
Z <- X + diag(N) * rep(2,10)
Z_eigen_objects <- eigen(Z)

# So slow!  Our "truth" to compare against
Z_eigen_objects$values

##  [1] 25.575519 23.419765 17.188487 12.155543 10.797895  9.377336  4.723900
##  [8]  3.225932  2.450137  2.038489

# Order O(n)... much faster!
eigen_objects$values + rep(2,10)

##  [1] 25.575519 23.419765 17.188487 12.155543 10.797895  9.377336  4.723900
##  [8]  3.225932  2.450137  2.038489
```