UQ Winter School - The Basics, session 1

Kathryn Kemper

13/06/2022

Objective

This practical session will review the definitions and matrix algebra operations outlined in the lecture, and be a basic introduction to using R. Part 1, will give the answers for the matrix algebra problems and Part 2 will conduct a simple PCA (Principal Component Analysis, i.e. an eigenvalue decomposition).

Important skills in R include writing loops, defining matrices and matrix manipulations; and making basic plots of data.

Part 1: Matrix Algebra

Defining matrices in R is relatively straight-forward. Just need to remember it fills rows then across columns when inputting the data, and to define either the number of rows or columns. We will input each of the matrices in the worksheet.

```
A = matrix(c(1,3,2,1,3,-1,-2,0,1),ncol=3) ; A
```

```
##      [,1] [,2] [,3]
## [1,]    1    1   -2
## [2,]    3    3    0
## [3,]    2   -1    1
```

```
B = matrix(c(1,0,1,0,1,1),nrow=2) ; B
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    0    0    1
```

```
C = diag(2) ; C
```

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

```r
D = matrix(c(2,0,1,1,4,2),ncol=2) ; D
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    0    4
## [3,]    1    2
```

```r
E = t(D) ; E
```

```
##      [,1] [,2] [,3]
## [1,]    2    0    1
## [2,]    1    4    2
```

```r
F = matrix(c(3,1,0,2),nrow=2) ; F
```

```
##      [,1] [,2]
## [1,]    3    0
## [2,]    1    2
```

```r
G = matrix(c(1,3),nrow=2) ; G
```

```
##      [,1]
## [1,]    1
## [2,]    3
```

```r
H = diag(3) ; H
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```r
I = matrix(c(1,3,0,2,6,0),nrow=3) ; I
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    6
## [3,]    0    0
```

Note above how to transpose a matrix, `t()`, and the short-hand for defining an identity matrix. Note that the `diag()` function in R can have two uses, depending on the argument provided to the function. By providing a single number, as above, the function returns an identity matrix with the given dimension. If the argument is a square matrix, it will return the diagonal elements of the matrix.

Next we need the dimensions of the matrices. The dimensions can be found in R using the `dim()` function.

```r
#matrix dimensions
 dim(A) ; dim(B) ; dim(C)
```

```
## [1] 3 3
```

```
## [1] 2 3
```

```
## [1] 2 2
```

```
  dim(D) ; dim(E) ; dim(F)
```

```
## [1] 3 2
```

```
## [1] 2 3
```

```
## [1] 2 2
```

```
  dim(G) ; dim(H) ; dim(I)
```

```
## [1] 2 1
```

```
## [1] 3 3
```

```
## [1] 3 2
```

There are a few special matrices in the examples, including 2 identity matrices (C, H) and 2 square matrices (A, F).

The rank of a matrix is the number of independent rows. If a matrix is not 'full rank' (meaning that the rank of the matrix is less than the largest dimension of the matrix) then it does not have a unique inverse. Can you identify the matrix that is not full rank? Calculating the inverse of a matrix is important when solving systems of equations, e.g. when estimating regression slopes. The rank of a matrix can be found using `qr()$rank` in R.

If the matrix is not full rank alternative methods to calculate the inverse (i.e. a generalized inverse) need to be used.

```
  # the matrix that is not full rank is matrix 'I', can you see why?
   qr(I)$rank
```

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

Next check the answers for the matrix multiplications, note that matrix multiplication has a special symbol, %*%.

```
  2*B
```

```
##      [,1] [,2] [,3]
## [1,]    2    2    2
## [2,]    0    0    2
```

```
  # B%*%C, non-conformable
  B%*%D
```

```
##      [,1] [,2]
## [1,]    3    7
## [2,]    1    2
```

```
# C%*%D, non-conformable
B-E
```

```
##      [,1] [,2] [,3]
## [1,]   -1    1    0
## [2,]   -1   -4   -1
```

```
t(C)
```

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

```
A%*%H
```

```
##      [,1] [,2] [,3]
## [1,]    1    1   -2
## [2,]    3    3    0
## [3,]    2   -1    1
```

Which matrices are invertible? Inverses are only calculable for square matrices, therefore only A and F potentially have inverses. We can use `solve()` in R to calculate the inverse and show its properties.

```
#calculate the inverse
 Ainv=solve(A)
 Finv=solve(F)
#show that a matrix multiplied by its inverse gives an identity matrix.
 A%*%Ainv
```

```
##      [,1]         [,2] [,3]
## [1,]    1 0.000000e+00    0
## [2,]    0 1.000000e+00    0
## [3,]    0 8.326673e-17    1
```

```
 Finv%*%F
```

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

The final useful function to know for matrix manipulate in R is using `cbind()` and `rbind()` to combine matrices. Try out these functions to join some of your matrices together.

Part 2: PCA Analysis

PCA analysis is an approach often used to identify population structure in genomic data. You will be provide with a pre-computed genomic relationship matrix for a small subset of the 1000 Genomes project data. Further information on the 1000 Genomes Project can be found here (https://www.internationalgenome.org/).

The genomic relationship matrix in this example has been computed for 50 individuals identified from one of 5 ancestry groups. It was computed from about 1.3M SNP spread evenly throughout the human genome using standard methods in GCTA (https://yanglab.westlake.edu.cn/software/gcta/#Overview).

First we will read in the data, and have a quick look at the format of the file. Note here that you will need to modify the directory to be read it to:
`dir=/data/module1/basicsSession1/`

```
  # read in the data
  x=read.table(paste0(dir,"1kg_winterSchool.grm.gz"))
  labels=read.table(paste0(dir,"1kg_winterSchool_superPop.txt"))[,1]

  # columns are row number, column number, number of SNP, genomic relationship
  head(x)
```

```
##   V1 V2      V3         V4
## 1  1  1 1365790 0.95116320
## 2  2  1 1365790 0.06590147
## 3  2  2 1365790 0.92498110
## 4  3  1 1365790 0.05928111
## 5  3  2 1365790 0.07948754
## 6  3  3 1365790 0.92168420
```

```
  tail(x)
```

```
##      V1 V2      V3          V4
## 1270 50 45 1365790 0.001001333
## 1271 50 46 1365790 0.046047610
## 1272 50 47 1365790 0.034500270
## 1273 50 48 1365790 0.049708950
## 1274 50 49 1365790 0.048485100
## 1275 50 50 1365790 0.878313700
```

The GRM is a square symmetrical matrix. The data read in are the lower triangular and diagonal elements, as indicated by the row and column number. How many elements do you expect for a 50x50 GRM?

In the first column there will be n elements, n-1 in the 2nd column, n-2 in the 3rd, etc. until 1 in the 50th column. Check this calculation against the dimensions of the GRM provided.

```
  # n = number of individuals
  n=50
  # expected number of elements in lower-triangular matrix, inc. diagonal elements
  N=sum(n:1) ; N
```

```
## [1] 1275
```

```r
# dim() gives the dimensions of the data read into R, i.e. the number of elements in lower-triangle +
dim(x)
```

```
## [1] 1275    4
```

Each of the 1000G participants has a ancestry label assigned to it. Let's use the `table()` function to count the labels, note that AFR = African, AMR = admixed American, EAS = east Asian, SAS = south Asian, EUR = European.

```r
table(labels)
```

```
## labels
## AFR AMR EAS EUR SAS
##   7   8  12  11  12
```

Now construct the GRM as a square matrix so that we can use the `eigen()` function in R to do the eigenvalue decomposition. The relationship values are in the 4th column of x. We can access the 4th column by using the notation object[rows,columns], where x[,4] will access all rows of the 4th column or x[1,4] will access a single element from the 1st row and 4th column.

This example also uses a `for(i in array) {action}` loop in R. If you are not familiar with loops in R, try a basic loop such as for `(i in 1:10) {print(i)}` to test it out. Make sure you understand the code!

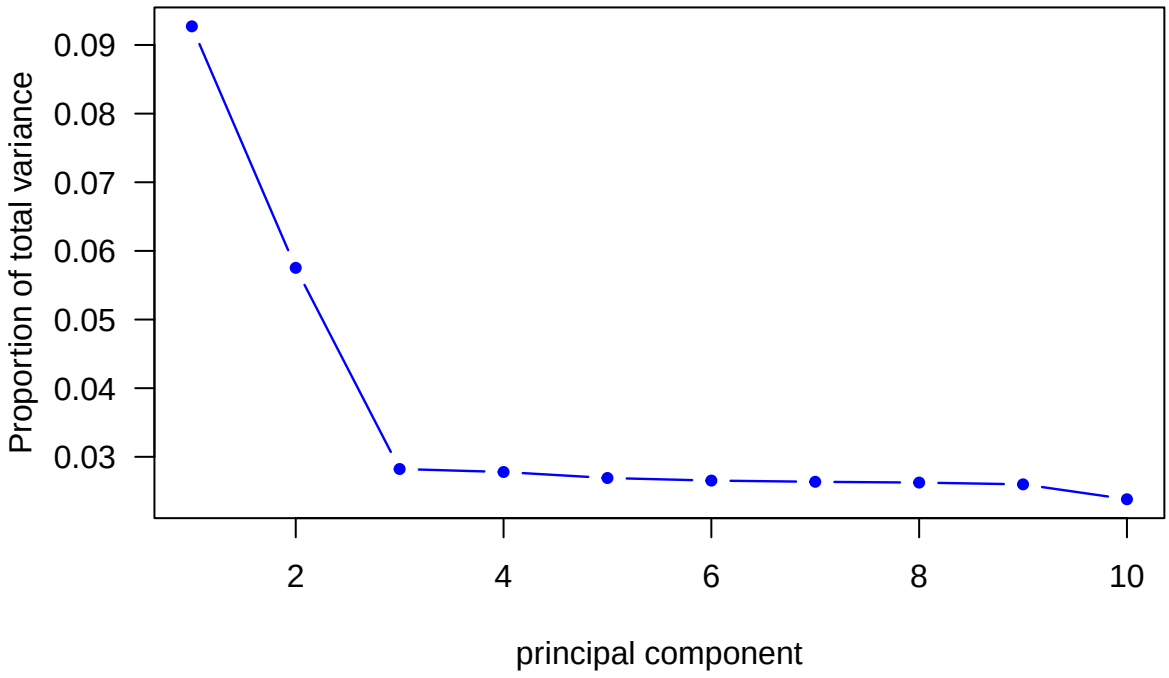
```r
# define the matrix using NA, then loop to fill the matrix
GRM=matrix(NA,nrow=n,ncol=n)
for (k in 1:N) {
  i=x[k,1] ; j=x[k,2]
  GRM[i,j]=x[k,4]
  GRM[j,i]=x[k,4]
}
```

Now conduct the PCA analysis using the `eigen()` function. Save the output and check the structure of the returned object using `str()`. If the matrix is full-rank, we expect 50 eigenvalues with values > 0, and 50 corresponding vectors of length 50 (i.e. a 50x50 matrix). We'll plot the first 2 PCs with the labels provided and detailing how much variation the PCs capture.

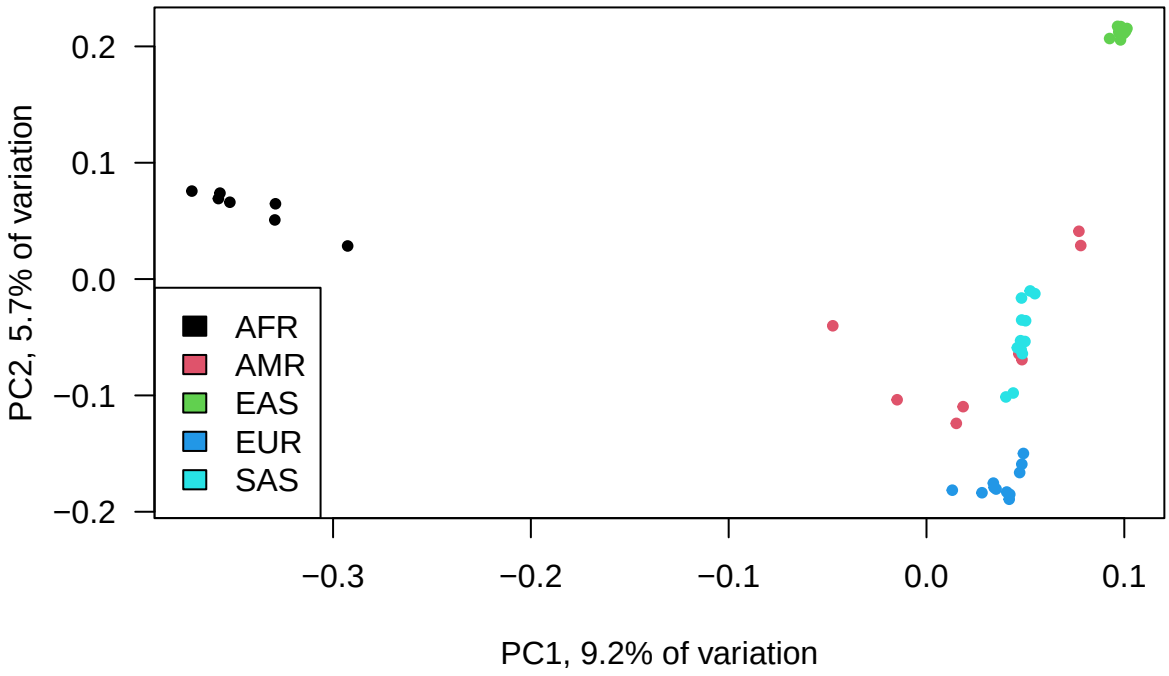
```r
#eigen value decomposition
 PCA=eigen(GRM)

#eigenvalues (vector), eigenvectors (50x50 matrix, 1 column per eigenvector)
str(PCA)
```

```
## List of 2
##  $ values : num [1:50] 4.78 2.97 1.46 1.43 1.39 ...
##  $ vectors: num [1:50, 1:50] 0.0406 0.0471 0.049 0.0482 0.1006 ...
##  - attr(*, "class")= chr "eigen"
```

```r
# proportion of variation explained by PCs
 total=sum(PCA$values)
 plot(PCA$values[1:10]/total,
      las=1, type="b", lwd=1.2, col="blue", xlab="principal component",
      ylab="Proportion of total variance", pch=20)
```




```
# plot
labels=as.factor(labels)
PC1=PCA$vectors[,1] ; PC2=PCA$vectors[,2]
plot(PC2~PC1, col=labels, pch=20, las=1,
     xlab="PC1, 9.2% of variation", ylab="PC2, 5.7% of variation")
legend("bottomleft",legend=levels(labels),fill=1:5)
```


0.2
0.1
0.0
−0.1
−0.2
PC2, 5.7% of variation
AFR
AMR
EAS
EUR
SAS
−0.3
−0.2
−0.1
0.0
0.1
PC1, 9.2% of variation