Statistics 1 Unit 2

Group 8

December 15, 2017

Contents

1 Task 15

1.1 a)

From Sylvester's Criterion it is clear that α being the first leading principal minor must be positive.

Since matrix B is a positive definite square matrix, $x'Bx > 0$ for all x. Let's take $x = (0, x_1 ... x_n)$. In this case, $x'Bx = y'Ay$, where $y = (x_1 ... x_n)$ and it must be > 0. Since $y'Ay > 0$ for arbitrary $y = (x_1 ... x_n)$, A is positive definite by definition.

1.2 b)

$$C = \begin{pmatrix} D & C^T \\ C & E \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ CD^{-1} & 1 \end{pmatrix} \begin{pmatrix} D & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} 1 & D^{-1}C^T \\ 0 & 1 \end{pmatrix}$$

Substitution $D = L_d L_d^T$ and $S = L_s L_s^t$ gives us the Cholesky factorization:

$$B = \begin{pmatrix} D & C^T \\ C & E \end{pmatrix} = \begin{pmatrix} L_d & 0 \\ CL_d^{-1} & L_s \end{pmatrix} \begin{pmatrix} L_d^t & L_d^{-1}C^T \\ 0 & L_s^t \end{pmatrix} = L_b L_b^t$$

In our case: D=α, C=a, E=A, $C^t = a^t$ then $L_d = \sqrt{\alpha}$, $L_d^{-1} = \frac{1}{\sqrt{\alpha}}$, L_d^t $=\sqrt{\alpha}$, $D^{-1} = \frac{1}{\alpha}$

$$S = A - \frac{1}{\alpha} a a^t$$

$$S = L_s L_s^t - Choleski - Factorsation$$

Hence,

$$B = \begin{pmatrix} \alpha & a^T \\ a & A \end{pmatrix} = \begin{pmatrix} \sqrt{\alpha} & 0 \\ \frac{1}{\alpha} a & L_s \end{pmatrix} \begin{pmatrix} \sqrt{\alpha} & \frac{1}{\alpha} a^T \\ 0 & L_s^T \end{pmatrix}$$

2 Task 16

2.1 a)

Here we use the same idea. Since matrix B is a positive definite square matrix, $x'Bx > 0$ for all x. Let's take $x = (x_1 ... x_n, 0)$. In this case, $x'Bx = y'Ay$, where $y = (x_1 ... x_n)$ and it must be > 0. Since $y'Ay > 0$ for arbitrary $y = (x_1 ... x_n)$, A is positive definite by definition. For α we can use $x = (0...0, 1)$. Applying the same idea, fact that $\alpha > 0$ is easily proved.

2.2 b)

The idea of Choleski decomposition is absolutelly the same as in 15(b). In this case: D=A, $C = a^t$, E=α, $C^t = a$

This gives us:

$$B = \begin{pmatrix} A & a \\ a^T & \alpha \end{pmatrix} = \begin{pmatrix} L_A & 0 \\ a^t L_A^{-1} & \sqrt{\alpha - a^T A^{-1} a} \end{pmatrix} \begin{pmatrix} L_A^T & L_A^{-1} \\ 0 & \sqrt{\alpha - a^T A^{-1} a} \end{pmatrix}$$

3 Task 17

```
> f17 <- function(k) {
+ mat_A <- matrix(c(10^(-2*k),1,1,1), 2, 2)
+ mat_M <- matrix(c(1, -1/(10^(-2*k)), 0,1), 2, 2)
+ vec_B <- matrix(c(1+(10^(-2*k)), 2), 2, 1)
+ S <- backsolve((mat_M %*% mat_A), (mat_M %*% vec_B))
+ S
```

```
+ }
> a<-seq(1:10)
> sapply(a, f17)
```

```
     [,1] [,2] [,3] [,4] [,5]      [,6]      [,7]     [,8] [,9] [,10]
[1,]    1    1    1    1    1 0.9998669 0.9992007 2.220446    0     0
[2,]    1    1    1    1    1 1.0000000 1.0000000 1.000000    1     1
```

When, ϵ decreases ($k > 6$) we are getting greater error.

4 Task 18

```
> matrix_2norm <- function(x){
+   norm(x, type="2")
+ }
> x <- matrix(c(1,2,3,4),2,2)
> max(svd(x)$d)
```

```
[1] 5.464986
```

```
> matrix_2norm(x)
```

```
[1] 5.464986
```

It is clear that 2-norm is just the maximum value of the singular value decomposition of the matrix.

5 Task 19

```
> cond_num <- function(p){
+   r <- matrix_2norm(p)
+   s <- matrix_2norm(solve(p))
+   s * r
+ }
> max(svd(x)$d) / min(svd(x)$d)
```

```
[1] 14.93303
```

```
> cond_num(x)
```

```
[1] 14.93303
```

We observe that the condition number of a matrix is the quotient of the lowest and the highest value of the SVD.
This follows from: $M = U\Sigma V'$ and $M^{-1} = V\Sigma^{-1}U'$. The values of the diagonal matrix Σ^{-1} are just the reciprocals of the diagonal values of Σ. Since the 2-norm is the maximum value of the SVD, the 2-norm of Σ^{-1} has to be 1 over the smallest singular value of the matrix.

6 Task 20

Ax=b
$\Delta x = A^{-1}\Delta b$
$\|b\| = \|Ax\|\|x\|$
Using norm's property:
$\|Ax\| \leq \|A\|\|x\|$
$\Rightarrow \|\Delta x\| = \|A^{-1}\Delta b\| \leq \|A^{-1}\|\|\Delta b\| \wedge \|b\| = \|Ax\| \leq \|A\|\|x\|$
Multiply these two inequalities (the norm>0)
$\|\Delta x\|\|b\| \leq \|A^{-1}\|\|\Delta b\|\|A\|\|x\|$
Divide by:$\|b\|\|x\|$and get
$\frac{||\Delta x||}{||x||} \leq k(A)\frac{||\Delta b||}{||b||}$

7 Task 21

```
> # Matrix is filled by columns by default
> A <- function(epsilon) {
+   matrix(data = c(1, 1-epsilon, 1+epsilon, 1),nrow =  2, ncol = 2)}
```

Now, given a small ϵ we will trying to solve the system. "normal" solve command, qr.solve, qr.decomposition "by hand" and the SVD method.
As suggested we will use sqare root of the machine precision as our ϵ.

```
> epsilon <- unname(sqrt(unlist(.Machine[1])))
> epsilon
```

```
[1] 1.490116e-08
```

```
> b <- c(1+epsilon +epsilon^2, 1)
```

First, let's check condition number using task 19 approach:

```
> svdA <- svd(A(epsilon))
> cond.nr <- max(svdA$d) / min(svdA$d)
> cond.nr
```

```
[1] 2.547621e+16
```

As we can see the condition number is very big. This means that small changes of a parameter will dramatically change the solution.

7.1 Solve

```
> # solve(A(epsilon), b)
> # Error in solve.default(A(epsilon), b) :
> #  system is computationally singular: reciprocal condition number = 5.55112e
```

7.2 QR.Solve

```
> # qr.solve(A(epsilon), b)
> # Error in qr.solve(A(epsilon), b) : singuläre Matrix 'a' in 'solve'
```

7.3 SVD

```
> # svdA <- svd(A(epsilon))
> # D <- diag(x=svdA$d)
> # D
> # a <- (tcrossprod(svdA$u %*% D, svdA$v))
> #solve(a,b)
> # Error in solve.default(a, b) :
> #  system is computationally singular: reciprocal condition number = 2.77556e
```

7.4 QR + rearrangement

QR decomposition using rearrangement $Rx = Q'b$ gives result, but with really big error:

```
> AQR <- qr(A(epsilon))
> qr.Q(AQR)
```

```
           [,1]       [,2]
[1,] -0.7071068 -0.7071068
[2,] -0.7071068  0.7071068
```

```
> backsolve(qr.R(AQR), crossprod(qr.Q(AQR),b))
```

```
     [,1]
[1,]  0.5
[2,]  0.5
```

Since the relative error is bounded by $\mathcal{K}(A)\frac{||\Delta b||}{||b||}$,larger ϵ leads to smaller condition numbers $\mathcal{K}(A)$ and hence more accurate solution.

8 Task 23

8.1 a

To prove the task just use the formula below:

$$Au = (uv^T)u = u(v^Tu) = (v^Tu)u.$$

8.2 b

The other eigenvalues of A. Since rank(A) = 1 all other eigenvalues equals are zeros.

8.3 c

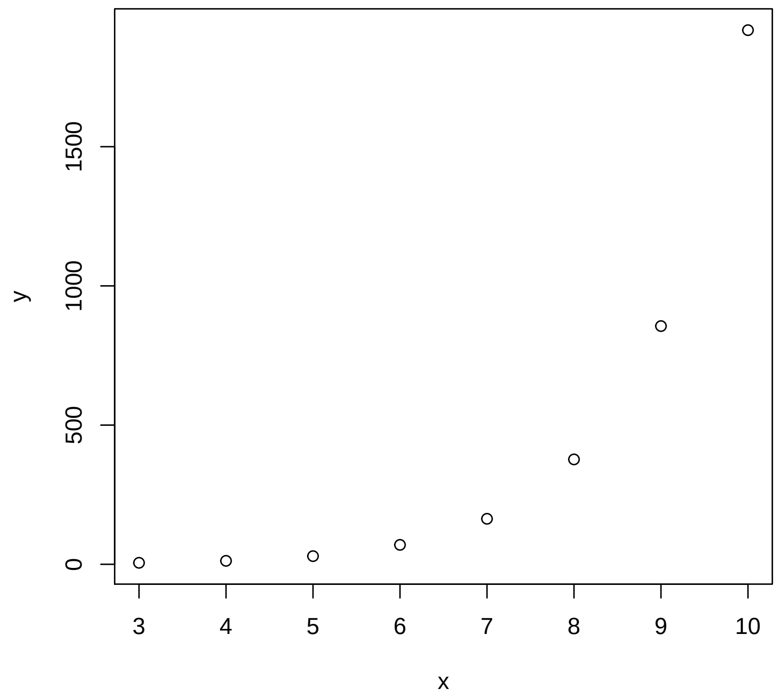
Let U $\in C_{m\times m}$ be a unitary matrix so that $Uu = \|u\|_2 e1$, and V $\in C_{n\times n}$ be a unitary matrix so that $Vv = \|v\|_2 e1$. We can substitute it into uv* shows SVD. $D = \|u\|_2 \|v\|_2 e1e1*$

9 Task 26

From task 19 follows that the ratio is a condition number of out matix. To show the dinamic it is convenient to use a plot: x-axis the dimention, y-axis the condition number.

```
> f26 <- function(n) {
+   M <- mat.or.vec(n, n)
+   M[upper.tri(M)] <- -1
+   diag(M) <- 1
+   condition_number <- max(svd(M)$d) / min(svd(M)$d)
+   return (condition_number)
+ }
> x <- 3:10
> y <- c(rep(0, 8))
> for (i in (1:8)) {
+   y[i] <- f26(x[i])
+ }
>
```

```
> plot(x,y)
```



As we can see, condition number exponentially increases with increasing the dimension of matrix.

10 Task 28

Singular Valued Decomposition constructs orthonormal bases for the range and null space of a matrix The columns of U which correspond to non-zero singular values of A are an orthonormal set of basis vectors for the range of A The columns of V which correspond to zero singular values form an orthonormal basis for the null space of A

```
> # The columns of output matrix contain orthonormal bases for the range of A
> get_bases_for_the_range <- function (A)
+ {
+   A_svd = svd(A)
+
+   d = A_svd$d
```

```
+   u = A_svd$u
+   v = A_svd$v
+
+   column_indices <- which(d >= .Machine$double.eps)
+
+   B = u[,column_indices]
+
+   return(B)
+ }
> # The columns of output matrix contain orthonormal bases for the null space c
> get_bases_for_the_null_space <- function (A)
+ {
+   A_svd = svd(A)
+
+   d = A_svd$d
+   u = A_svd$u
+   v = A_svd$v
+
+   column_indices <- which(d < .Machine$double.eps)
+
+   B = v[,column_indices]
+
+   return(B)
+ }
> # Test
> A <- matrix (c(3,7,8,7,10,9,8,9,12, 15) , nrow = 5, ncol = 2)
> get_bases_for_the_range(A)
```

```
           [,1]          [,2]
[1,] -0.3131156  0.7326549483
[2,] -0.3605707 -0.4066120946
[3,] -0.4080145 -0.4879538258
[4,] -0.4744268  0.2445079999
[5,] -0.6167582  0.0004828064
```

```
> get_bases_for_the_null_space(A)
```

```
[1,]
[2,]
```

11 Task 31

Required function is represented below:

```
> library(matrixcalc)
> Duplication <- function (n) {
+   # Arbitrary matrix
+   A <- matrix((1:n ^ 2), n, n)
+   # Make it symmetric
+   A[lower.tri(A)] = t(A)[lower.tri(A)]
+   vec(A)
+   vech(A)
+   D <- mat.or.vec(length(vec(A)), length(vech(A)))
+   for (i in 1:n ^ 2) {
+     col_num <- match(vec(A), vech(A))
+     row_num <- 1:n ^ 2
+     D[row_num[i], col_num[i]] <- 1
+   }
+   return(D)
+ }
> Duplication(2)
```

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

12 Task 32

Let's compute Singular Value Decomposition of the duplication matrix D_n with $n = 2$ and $n = 3$:

```
> svd(Duplication(2))
```

```
$d
[1] 1.414214 1.000000 1.000000

$u
           [,1] [,2] [,3]
[1,]  0.0000000    0    1
```

```
[2,] -0.7071068    0    0
[3,] -0.7071068    0    0
[4,]  0.0000000   -1    0

$v
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]   -1    0    0
[3,]    0   -1    0

> svd(Duplication(3))

$d
[1] 1.414214 1.414214 1.414214 1.000000 1.000000 1.000000

$u
            [,1]       [,2]       [,3] [,4] [,5] [,6]
 [1,]  0.0000000  0.0000000  0.0000000    1    0    0
 [2,]  0.0000000 -0.7071068  0.0000000    0    0    0
 [3,]  0.0000000  0.0000000 -0.7071068    0    0    0
 [4,]  0.0000000 -0.7071068  0.0000000    0    0    0
 [5,]  0.0000000  0.0000000  0.0000000    0   -1    0
 [6,] -0.7071068  0.0000000  0.0000000    0    0    0
 [7,]  0.0000000  0.0000000 -0.7071068    0    0    0
 [8,] -0.7071068  0.0000000  0.0000000    0    0    0
 [9,]  0.0000000  0.0000000  0.0000000    0    0   -1

$v
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    0    0    1    0    0
[2,]    0   -1    0    0    0    0
[3,]    0    0   -1    0    0    0
[4,]    0    0    0    0   -1    0
[5,]   -1    0    0    0    0    0
[6,]    0    0    0    0    0   -1
```

It can be seen that if $n = 2$, then last 2 diagonal elements of matrix D, which are singular values, are equal to 1. If $n = 3$, then the last 3 diagonal elements of matrix D are equal to 1. Other elements are 1.414214 what is just the square root of 2.

13 Task 33

Function returning the elimination matrix L_n for given n and example of its application are as follows:

```
> library(matrixcalc)
> Elimination <- function (n) {
+ A <- matrix((1:n^2),n,n)
+ vec(A)
+ vech(A)
+ D <- mat.or.vec(length(vech(A)),length(vec(A)))
+ for (i in 1:n^2) {
+ col_num <-match(vech(A), vec(A))
+ row_num <- 1:(n*(n+1)/2)
+ D[row_num[i], col_num[i]] <- 1}
+ return(D)}
> Elimination(2)
```

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

14 Task 34

Let's compute Singular Value Decomposition of the elimination matrix L_n for $n = 2$:

```
> svd(Elimination(2))
```

```
$d
[1] 1 1 1

$u
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1

$v
     [,1] [,2] [,3]
```

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

It can be seen that matrix U is an identity matrix and diagonal matrix D is an identity matrix too. In addition, the elimination matrix itself equals to V^t matrix.

15 Task 35

Function returning the commutation matrix K_{mn} for given m and n and example of its application are as follows:

```
> library(matrixcalc)
> commutation_matrix <- function (r, c = r)
+ {
+   H <- H.matrices(r, c)
+   p <- r * c
+   K <- matrix(0, nrow = p, ncol = p)
+   for (i in 1:r) {
+     for (j in 1:c) {
+       Hij <- H[[i]][[j]]
+       K <- K + (Hij %x% t(Hij))
+     }
+   }
+   return(K)
+ }
> k <- commutation_matrix(3,2)
> k
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    0    0    0    0
[2,]    0    0    0    1    0    0
[3,]    0    1    0    0    0    0
[4,]    0    0    0    0    1    0
[5,]    0    0    1    0    0    0
[6,]    0    0    0    0    0    1
```

16 Task 36

The singular values of the commutation matrix are:

```
> svd(k)$d
```

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

17 Task 37

The Moore-Penrose inverse can be expressed in terms of SVD, $A = UDV'$, since $A^{+} = VD^{-1}U'$
where each element in D^{-1} is taken as a reciprocal of corresponding element in matrix D, if it is greater, than given tolerance, or 0 otherwise.

```
> f37 <- function(A, tol = 1e-10){
+   SVD <- svd(A)
+   D <- NULL
+   for(i in 1:length(SVD$d)){
+     if(SVD$d[i] < tol){
+       SVD$d[i] <- 0
+       D <- c(D,as.numeric(SVD$d[i]))
+     }
+     else{
+       D <- c(D,as.numeric(1/(SVD$d[i])))
+     }
+   }
+   SVD$v %*% diag(D)
+   A_plus <- crossprod(t(crossprod(t(SVD$v), diag(D))), t(SVD$u))
+   return(A_plus)
+ }
```

18 Task 38

Let's use the property that the trace is invariant under cyclic permutations when inner matrix is square.

```
> wcptrace <- function(A, w) {
+   if(dim(A)[1] == dim(A)[2]) {
+     a <- numeric(dim(A)[1])
+     for(i in 1:dim(A)[1]){
+       a[i] <- sum(A[i, ]^2)
+     }
```

```
+     return(sum(a * w))
+   } else {
+     cat("The input matrix is not square, therefore there is not much to impro
+     sum(diag(t(A) %*% diag(w) %*% A))
+   }
+ }
> mat <- matrix(data = rexp(200, rate = 10), nrow = 100, ncol = 100)
> w <- sample(1:100)
> system.time( replicate(10000, wcptrace(mat, w)))

   user  system elapsed
   2.36    0.00    2.38

> system.time( replicate(10000, sum(diag(t(mat) %*% diag(w) %*% mat))))

   user  system elapsed
  15.16    0.00   15.24
```

19 Task 39

```
> #Using Cholesky Decomposition is more efficient for our task
>
> dmvnorm2 <- function (x, m, V) {
+   mat_chol <- chol(V)
+   delta <- x - m
+   y <-det(mat_chol)^2 * (2*pi)^nrow(mat_chol)
+   e <- exp(-t(delta) %*% solve(mat_chol) %*% t(solve(mat_chol)) %*% delta / 2
+   diag(y^(-1/2) * e)
+ }
> #TEST
> library(mvtnorm)
> M <- matrix(c(1,0.5,0.5,0.5,1,0.5,0.5,0.5,1),nrow=3)
> dmvnorm(1:3, 1:3, M)

[1] 0.08979356

> #Our code test
> dmvnorm2(1:3, 1:3, M)

[1] 0.08979356
```