Lista 2, aula 2 em 23/9

Henrique Ap. Laureano
laureano@ufpr.br ∧ www.leg.ufpr.br/~henrique/

October 2, 2019

Contents

Exercise 1

Considere o seguinte sistema linear $Ax = b$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ -1 \end{bmatrix}$$

(a) Mostre que a matriz dos coeficientes é positiva definida

Para A ser positiva definida, seus autovalores devem ser positivos. Os autovalores de A são as soluções da seguinte equação de ordem 3

$$\lambda^3 - 6\lambda^2 + 5\lambda - 1 = 0$$

Para encontrar tais soluções, autovalores, empregamos o método de Newton

<r code>

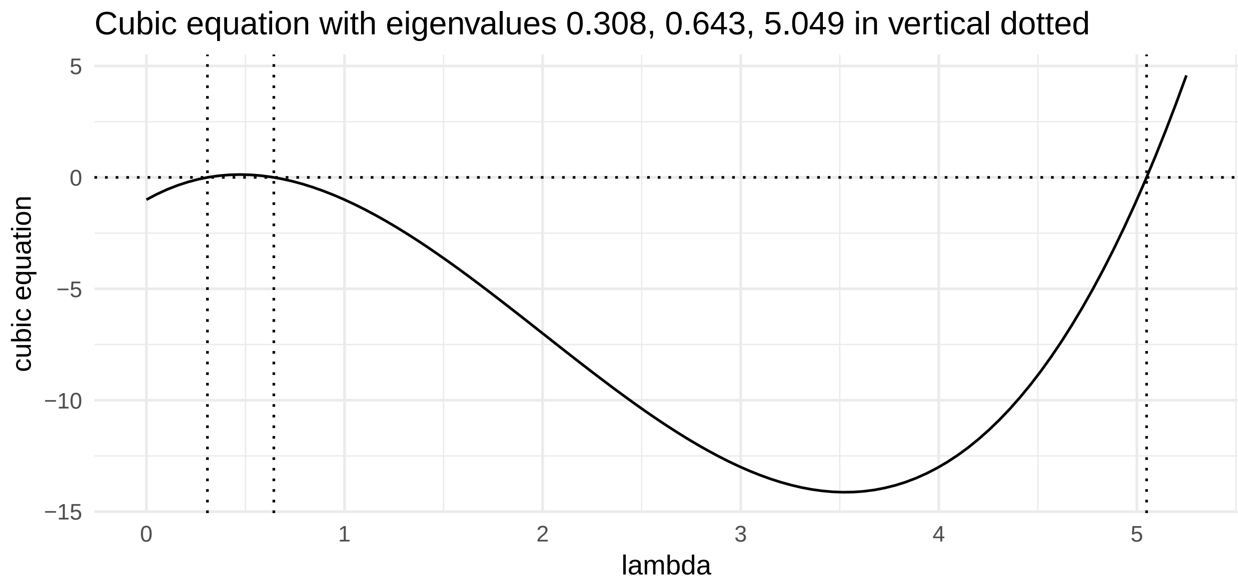
```
f1a <- function(lambda) { lambda^3 -  6 * lambda^2 + 5 * lambda - 1 }

f1a_prime <- function(lambda) { 3 * lambda^2 - 12 * lambda + 5 }

newton <- function(f, fprime, init, kmax = 100, tol = 1e-3) {
    xs <- numeric(kmax)
    xs[1] <- init - f(init)/fprime(init)
    xs[2] <- xs[1] - f(xs[1])/fprime(xs[1])
    k <- 2
    while(abs(diff(xs[(k - 1):k]))/abs(xs[k]) > tol & k < kmax) {
        k <- k + 1
        xs[k] <- xs[k - 1] - f(xs[k - 1])/fprime(xs[k - 1])
    }
    return(xs[seq(k)])
}
lambdas <- c(tail(newton(f1a, f1a_prime, init = 0), n = 1),
             tail(newton(f1a, f1a_prime, init = .5), n = 1),
             tail(newton(f1a, f1a_prime, init = 4), n = 1))

library(ggplot2)

ggplot(data.frame(x = seq(0, 5.25, .25)), aes(x = x)) +
    theme_minimal() +
    stat_function(fun = f1a) +
    geom_hline(yintercept = 0, linetype = "dotted") +
    geom_vline(xintercept = lambdas, linetype = "dotted") +
    labs(x = "lambda", y = "cubic equation",
         title = paste("Cubic equation with eigenvalues",
                       paste(round(lambdas, 3), collapse = ", "),
                       "in vertical dotted"))
```



(b) Implemente a decomposição de Cholesky

<r code>

```r
library(Matrix)

A <- Matrix(c(1, 1, 1, 1, 2, 2, 1, 2, 3), 3)

cholesky <- function(A) {
    n <- nrow(A)
    if (n != ncol(A)) {
        message("A isn't a square matrix(?), check it.")
        stop
    }
    U <- Matrix(0, nrow = n, ncol = n)
    P <- Diagonal(n)
    for (p in seq(n)) {
        index <- which.max(A[ , p])
        newlineA <- A[index, ] ; newlineP <- P[index, ]
        oldlineA <- A[p, ] ; oldlineP <- P[p, ]
        A[p, ] <- newlineA ; P[p, ] <- newlineP
        A[index, ] <- oldlineA ; P[index, ] <- oldlineP
    }
    for (k in seq(n)) {
        U[k, k] <- sqrt(diag(A)[k] - U[seq(k - 1), k] %*% U[seq(k - 1), k])
        if (k == n) stop
        else
            for (i in (k + 1):n) {
                U[k, i] <- (
                    A[k, i] - U[seq(k - 1), k] %*% U[seq(k - 1), i]
```

```r
                )/U[k, k]
        }
    }
    U <- U %*% P
    return(list(U = U, tU = t(U), A = t(U) %*% U))
}
cholesky(A)

$U
3 x 3 sparse Matrix of class "dgCMatrix"

[1,] 1 1 1
[2,] . 1 1
[3,] . . 1

$tU
3 x 3 sparse Matrix of class "dgCMatrix"

[1,] 1 . .
[2,] 1 1 .
[3,] 1 1 1

$A
3 x 3 sparse Matrix of class "dgCMatrix"

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

(c) Agora, implemente a solução do sistema baseada na decomposição de Cholesky

<r code>

```r
b <- Matrix(c(5, 3, -1))

backsub <- function(A, b) {
    n <- nrow(A)
    if (n != ncol(A)) {
        message("A isn't a square matrix(?), check it.")
        stop
    }
    x <- Matrix(numeric(n))
    for (i in rev(seq(n))) {
        if (i == n) x[i] <- b[i]/A[i, i]
        else x[i] <- (
```

```
            b[i] - A[i, seq(i + 1, n)] %*% x[seq(i + 1, n)]
        )/A[i, i]
    }
    S <- cbind(A, x, b)
    colnames(S) <- c(paste0("a", seq(n)), "x", "b")
    return(S)
}

solver <- function(A, b, method = c("Cholesky", "LU")) {
    n <- nrow(A)
    if (n != ncol(A)) {
        message("A isn't a square matrix(?), check it.")
        stop
    }
    L <- switch(method, "Cholesky" = cholesky(A)$tU, "LU" = LU(A)$L)
    y <- numeric(n)
    for (i in seq(n)) {
        y[i] <- (b[i] - L[i, seq(i - 1)] %*% y[seq(i - 1)])/L[i, i]
    }
    x <- backsub(t(L), y)[ , "x"]
    return(Matrix(x))
}
solver(A, b, method = "Cholesky")
```

```
3 x 1 Matrix of class "dgeMatrix"
     [,1]
[1,]    7
[2,]    2
[3,]   -4
```

(d) Calcule o determinante de A, baseado na decomposição de Cholesky

<r code>

```
detchol <- function(A) {
    n <- nrow(A)
    ifelse(n == ncol(A),
           L <- cholesky(A)$tU,
           message("A isn't a square matrix(?), check it."))
    det <- prod(diag(L))^2
    return(det)
}
detchol(A)
```

```
[1] 1
```

Exercise 2

Implemente o método de Eliminação de Gauss no sistema do Exercício 1.

<r code>

```
gauss <- function(A, b) {
    n <- nrow(A)
    if (n != ncol(A)) {
        message("A isn't a square matrix(?), check it.")
        stop
    }
    L <- P <- Diagonal(n)
    for (p in seq(n)) {
        index <- which.max(A[ , p])
        newlineA <- A[index, ] ; newlineP <- P[index, ]
        oldlineA <- A[p, ] ; oldlineP <- P[p, ]
        A[p, ] <- newlineA ; P[p, ] <- newlineP
        A[index, ] <- oldlineA ; P[index, ] <- oldlineP
    }
    S <- cbind(A, b)
    colnames(S) <- c(paste0("a", seq(n)), "b")
    for (k in seq(n - 1)) {
        for (i in seq(k + 1, n)) {
            L[i, k] <- S[i, k]/S[k, k]
            S[i, ] <- S[i, ] - L[i, k] * S[k, ]
        }
    }
    return(S)
}
(gaussAb <- gauss(A, b))
```

```
3 x 4 Matrix of class "dgeMatrix"
     a1 a2 a3  b
[1,]  1  1  1  5
[2,]  0  1  1 -2
[3,]  0  0  1 -4
```

```
backsub(gaussAb[ , seq(ncol(A))], gaussAb[ , "b"])
```

```
3 x 5 Matrix of class "dgeMatrix"
     a1 a2 a3  x  b
[1,]  1  1  1  7  5
[2,]  0  1  1  2 -2
[3,]  0  0  1 -4 -4
```

Exercise 3

Implemente a decomposição LU da matriz dos coeficientes do sistema dado no exercício 1.

<r code>

```
LU <- function(A) {
    n <- nrow(A)
    if (n != ncol(A)) {
        message("A isn't a square matrix(?), check it.")
        stop
    }
    L <- P <- Diagonal(n)
    for (p in seq(n)) {
        index <- which.max(A[ , p])
        newlineA <- A[index, ] ; newlineP <- P[index, ]
        oldlineA <- A[p, ] ; oldlineP <- P[p, ]
        A[p, ] <- newlineA ; P[p, ] <- newlineP
        A[index, ] <- oldlineA ; P[index, ] <- oldlineP
    }
    for (k in seq(n - 1)) {
        for (i in seq(k + 1, n)) {
            L[i, k] <- A[i, k]/A[k, k]
            A[i, ] <- A[i, ] - L[i, k] * A[k, ]
        }
    }
    U <- A
    return(list(L = L, U = U, P = P, A = P %*% L %*% U))
}
LU(A)
```

```
$L
3 x 3 sparse Matrix of class "dtCMatrix"

[1,] 1 . .
[2,] 1 1 .
[3,] 1 1 1

$U
3 x 3 Matrix of class "dgeMatrix"
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    0    1    1
[3,]    0    0    1

$P
3 x 3 diagonal matrix of class "ddiMatrix"
     [,1] [,2] [,3]
```

```
[1,]    1    .    .
[2,]    .    1    .
[3,]    .    .    1

$A
3 x 3 Matrix of class "dgeMatrix"
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    2    2
[3,]    1    2    3
```

Exercise 5

Utilize a decomposição LU da matriz dos coeficientes do exercício 1 para resolver o sistema.

<r code>

```
solver(A, b, method = "LU")

3 x 1 Matrix of class "dgeMatrix"
     [,1]
[1,]    7
[2,]    2
[3,]   -4
```

Exercise 6

Calcule a inversa da matriz dos coeficientes do Exercício 1 usando de Eliminação de Gauss.

<r code>

```
(gaussb1 <- gauss(A, Matrix(c(1, 0, 0))))

3 x 4 Matrix of class "dgeMatrix"
     a1 a2 a3  b
[1,]  1  1  1  1
[2,]  0  1  1 -1
[3,]  0  0  1  0

(gaussb2 <- gauss(A, Matrix(c(0, 1, 0))))

3 x 4 Matrix of class "dgeMatrix"
     a1 a2 a3  b
[1,]  1  1  1  0
[2,]  0  1  1  1
[3,]  0  0  1 -1
```

```
(gaussb3 <- gauss(A, Matrix(c(0, 0, 1))))
```

```
3 x 4 Matrix of class "dgeMatrix"
     a1 a2 a3 b
[1,]  1  1  1 0
[2,]  0  1  1 0
[3,]  0  0  1 1
```

```
(inversaA <- cbind(
     backsub(gaussb1[ , seq(ncol(A))], gaussb1[ , "b"])[ , "x"],
     backsub(gaussb2[ , seq(ncol(A))], gaussb2[ , "b"])[ , "x"],
     backsub(gaussb3[ , seq(ncol(A))], gaussb3[ , "b"])[ , "x"]))
```

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

Exercise 7

Calcule a inversa da matriz dos coeficientes do Exercício 1 usando a decomposição de Cholesky.

<r code>

```
(meiainversa <- cbind(backsub(cholesky(A)$U, Matrix(c(1, 0, 0)))[ , "x"],
                      backsub(cholesky(A)$U, Matrix(c(0, 1, 0)))[ , "x"],
                      backsub(cholesky(A)$U, Matrix(c(0, 0, 1)))[ , "x"]))
```

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

```
(inversa <- meiainversa %*% t(meiainversa))
```

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

Exercise 8

Uma matriz H é Hessenberg superior se os elementos $h_{ij} = 0$ para $i > j+1$.

<r code>

```
createH <- function(order) {
    H <- toeplitz(seq(order))
    for (i in seq(order)) {
        for (j in seq(order)) {
            H[i, j] <- ifelse(i > j + 1, 0, H[i, j])
        }
    }
    return(Matrix(H))
}
(H <- createH(5))
```

```
5 x 5 Matrix of class "dgeMatrix"
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    2    1    2    3    4
[3,]    0    2    1    2    3
[4,]    0    0    2    1    2
[5,]    0    0    0    2    1
```

(a) Implemente a triangulação superior da matriz de Hessenberg $H_{n \times n}$

<r code>

```
upper_hessenberg <- function(H) {
    n <- nrow(H)
    if (n != ncol(H)) {
        message("H isn't a square matrix(?), check it.")
        stop
    }
    L <- Diagonal(n)
    for (k in seq(n - 1)) {
        L[k + 1, k] <- H[k + 1, k]/H[k, k]
        H[k + 1, k:n] <- H[k + 1, k:n] - L[k + 1, k] * H[k, k:n]
    }
    return(list(L = L, upperH = H, H = L %*% H))
}
upper_hessenberg(H)
```

```
$L
5 x 5 sparse Matrix of class "dtCMatrix"

[1,] 1  .          .    .        .
[2,] 2  1.0000000  .    .        .
[3,] . -0.6666667  1.0  .        .
[4,] .  .         -1.2  1.000000 .
[5,] .  .          .   -3.333333 1
```

```
$upperH
5 x 5 Matrix of class "dgeMatrix"
     [,1] [,2]      [,3]      [,4]      [,5]
[1,]    1    2  3.000000  4.000000  5.000000
[2,]    0   -3 -4.000000 -5.000000 -6.000000
[3,]    0    0 -1.666667 -1.333333 -1.000000
[4,]    0    0  0.000000 -0.600000  0.800000
[5,]    0    0  0.000000  0.000000  3.666667

$H
5 x 5 Matrix of class "dgeMatrix"
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    2    1    2    3    4
[3,]    0    2    1    2    3
[4,]    0    0    2    1    2
[5,]    0    0    0    2    1
```

Last modification on ...

```
[1] "2019-10-02 16:28:18 -03"
```