Métodos Numéricos en Ciencia de Datos

PEC 2

Alan Coila Bustinza

3/3/2022

1.1 Alternativa al cálculo de los coeficientes del polinomio característico

Definición de la matriz L

Definimos la matriz L como en la PEC anterior

```
L<-matrix(c(2,35,-36,-180,1,0,0,0,0,1,0,0,0,0,1,0),nrow=4,ncol=4,byrow=TRUE)
L
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2   35  -36 -180
## [2,]    1    0    0    0
## [3,]    0    1    0    0
## [4,]    0    0    1    0
```

Orden de la matriz L

El orden de la matriz es el número de filas y número de columnas que la componen. Podemos calcularlo mediante la funcion ***dim()***, esta retorna dos valores, al tratarse de un matriz cuadrada, solo necesitaremos el primer valor, lo almacenamos en la variable N

```
N<-dim(L)[1]
```

Para obtener los coeficientes del polinomio característico: $P(\lambda) = \lambda^4 + c_1\lambda^3 + c_2\lambda^2 + c_3\lambda + c_4$, haremos uso de la expresión $c_n = -\frac{1}{n}Tr(B_n)$ donde $B_N = L(B_{n-1} + c_{n-1}I)$.

Necesitamos definir la matriz identidad.

```
I<-diag(1,N) #Matriz identidad de dimension igual que L
I
```

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

Ahora calcularemos los coeficientes del polinomio característico reemplazado los valores que tenemos en las expresiones correspondientes. Usaremos como en la PEC anterior las funciones anidadas ***sum(diag())*** para obtener la traza, teniendo cuidado en la multiplicación entre matrices y con un escalar.

```
B1<-L
c1<--sum(diag(B1))
c1
```

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

```
B2<-L%*%(B1 + c1*I)
c2<-(-0.5)*sum(diag(B2))
c2
```

```
## [1] -35
```

```
B3<-L%*%(B2+c2*I)
c3<-(-1/3)*sum(diag(B3))
c3
```

```
## [1] 36
```

```
B4<-L%*%(B3+c3*I)
c4<-(-0.25)*sum(diag(B4))
c4
```

```
## [1] 180
```

1.2 Tercera parte del algoritmo de Leverrier: Cálculo de los valores propios de L(I)

Primero obtedremos los autovalores de mayor módulo, como se piden los valores de a_n y b_n para $3 <= n <= 20$, debemos calcular $s1, s2, ..., s20$. Si recordamos que $s_n = Tr(L^n)$, mediante una iteración podremos obtener las trazas.

```
n=20 #definimos el numero iteraciones.
Ln = diag(N) #creamos la matriz Ln , matriz identidad de orden igual al de L
sn = 0 #definimos la variable que almacenara los valores de la iteración
for(i in 1:n){
  Ln =  Ln%*%L # Ln multiplicada por L
  sn[i] = sum(diag(Ln)) #Traza de la matriz Ln
}
print(sn)
```

```
##  [1] 2.000000e+00 7.400000e+01 1.100000e+02 2.018000e+03 4.862000e+03
##  [6] 6.307400e+04 2.038700e+05 2.077058e+06 8.143742e+06 7.029187e+07
## [11] 3.141440e+08 2.421458e+09 1.184158e+10 8.447248e+10 4.396817e+11
## [16] 2.973741e+12 1.616385e+13 1.053750e+14 5.902874e+14 3.751529e+15
```

Utilizaremos las relaciones de Leverrier para calcular los valores propios, estan definidos mediante las siguientes fomulas:

- $\sigma_n = \frac{s_n}{s_{n-1}}, n \geq 2$ con $\sigma = 0$ y
- $\delta_n = \frac{\sigma_n - \sigma_{n-1}}{\sigma_{n-1} - \sigma_{n-2}}, n \geq 3.$

Por lo que calcularemos los valores de σ de la sigueitne forma:

```
sigma = c(0, 0) #inicializamos a cero el vector

for(i in 2:length(sn)){ # iteramos dentro de la variable que almacena la
                        #suma de los valores propios de la matriz Ln
```

```
  sigma[i] <- sn[i]/sn[i-1] #aplicamos la primera identidad y la almacenamos en la variable sigma
}
print(sigma)
```

```
##  [1]  0.000000 37.000000  1.486486 18.345455  2.409316 12.972851  3.232235
##  [8] 10.188149  3.920806  8.631397  4.469137  7.708116  4.890266  7.133550
## [15]  5.205029  6.763394  5.435527  6.519180  5.601776  6.355428
```

Ahora haremos un cálculo similar para δ

```
delta = c(0, 0, 0)   #inicializamos a cero el vector
for(i in 3:length(sigma)){    #iteramos de la misma forma que el bucle anterior
  delta[i] = (sigma[i]-sigma[i-1])/(sigma[i-1]-sigma[i-2])
}
```

A continuacion calculamos las sucesiones a_n y b_n, que convergen a los valores propios de mayor móulo. Para ellos debemos resolver el siguiente sistema de ecuaciones:

- $a_n + b_n = \sigma_n + \delta_n \sigma_{n-2}, n \geq 3$
- $a_n b_n = \sigma_{n-1} \sigma_{n-2} \delta_n, n \geq 3$

Creamos las sucesiones auxiliares $y_n = a_n + b_n$, y $z_n = a_n b_n$.

```
yn = c(0, 0, 0)
zn = c(0, 0, 0)

#para yn
for (i in 3:n){
  yn[i] = sigma[i] + (delta[i]*sigma[i-2])
}

#para zn
for (i in 3:n){
  zn[i] = sigma[i-1]*sigma[i-2]*delta[i]
}
```

Los valores de a_n y b_n se obtienen al resolver la ecuacion: $x^2 - y_n[i]x + z_n[i] = 0$ para cada uno de los valores de i. Usamos la funcion ***polyroot()***, esta devuelve valores complejos y nosotros nos quedamos con la parte real mediante al funcion ***Re()***.

```
for (i in 3:n){
  lambda = Re(polyroot(c(zn[i], -yn[i], 1)))
  print(lambda)
}
```

```
## [1] 0.000000 1.486486
## [1]  5.515058 -4.734236
## [1]  5.604025 -4.599827
## [1]  5.834173 -5.021912
## [1]  5.897536 -4.886926
## [1]  5.953583 -5.029534
## [1]  5.974638 -4.966105
## [1]  5.987920 -5.014027
## [1]  5.993727 -4.988999
## [1]  5.996934 -5.005580
## [1]  5.998441 -4.996239
## [1]  5.999229 -5.002092
```

```
## [1]  5.999612 -4.998678
## [1]  5.999807 -5.000767
## [1]  5.999903 -4.999529
## [1]  5.999952 -5.000278
## [1]  5.999976 -4.999831
## [1]  5.999988 -5.000100
```

```r
lambda4 = lambda[1]
lambda3 = lambda[2]
```

1.3 Cuarta parte del algoritmo de Leverrier: Cálculo de los valores propios de L(II)

Ahora calcularemos los autovalores lambda1 y lambda2 de la matriz L. Utilizaremos la identidad:

$$\frac{P(\lambda)}{(\lambda - a_20)(\lambda - b_20)} = 0$$
$$\frac{\lambda^4 + c_1\lambda^3 + c_2\lambda^2 + c_3\lambda + c_4}{\lambda^2 - (a_{20} + b_{20}) + a_{20}b_{20}} = 0$$

Necesitaremos instalar la libreria "pracma"

```r
install.packages("pracma")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.1'
## (as 'lib' is unspecified)
```

```r
library(pracma)
```

```r
#Creamos los vectores correspondientes a los coeficientes del numerador y denominador.
num = c(1, c1, c2, c3, c4)
den = c(1, -(lambda[1] + lambda[2]), lambda[1]*lambda[2])

div = deconv(num, den) #la función deconv() de la libreria "pracma" nos devuelve un polinomio
                       #cociente "q" y uno "r"
print(div[[1]]) #imprimimos el polinomio cociente "q"
```

```
## [1]  1.000000 -1.000113 -5.999458
```

Calculamos las raices del polinomio obtenido

```r
lambda = Re(polyroot(c(rev(div[[1]]))))
lambda2 = lambda[1]
lambda1 = lambda[2]

print(c(lambda4, lambda3, lambda2, lambda1))
```

```
## [1]  5.999988 -5.000100  2.999959 -1.999847
```

```r
print(eigen(L)$values) #comparamos con las raices el ejercicio anterior
```

```
## [1]  6 -5  3 -2
```