Temas selectos de econometría y finanzas (modulo de matrices aleatorias)

J. Antonio García Ramirez, Tarea 5, la resolvente

25 de Octobre, 2018

Ejercicio 1

Insertar la fórmula del semicírculo

$$p(x) = \frac{1}{pi}\sqrt{2 - x^2}$$

En la definición del resolvente

$$G_{\infty}^{av} = \int dx' \frac{p(x')}{z - x}$$

Y realice la integración numérica para $z = x - i\epsilon$, con $\epsilon > 0$, separando los casos:

i. $0 < X < \sqrt{2}$
ii. $-\sqrt{2} < x < 0$

Compare los resultados con la expresión dada por:

$$G_{\infty}^{av} z \pm \sqrt{z^2 - 2}$$

Para 10 elecciones distintas de z. El ejercicio se evalúa como correcto si el error relativo porcentual promedio es menor al 1%.

Después de digerir el significado del ejercicio se implementaron las siguientes funciones para realizar el cálculo numérico (de integrales complejas) y el resultado teórico.

```
p <- function(x, complejo)
{
    # Closure con la densidad
    # ojo: esta funcion regresa una funion, solo sirve para fijar el parametro 'z'
    z <- complejo
    function(x)  (1/(pi*(z-x)))*( 2- x**2 )**.5 #densidad con 'z' fijo
}
G.numerica <- function(x, complejo)
{
    library(elliptic)   #package para integral numerica
    densidad <- p(complejo = complejo) # fijamos la funcion a integrar con parametro
                                       # z y domminio [-2**.5, 2**.5]
    myintegrate(densidad, lower= -2**.5, upper = 2**.5) #integral numerica
}
G.teorica <- function(complejo)
{
    # Resultado analitico de la revolvente solo hay que checar la parte
    # real del parametro 'z'
    teoria <- ifelse(Re(complejo) > 0, complejo - (complejo**2-2)**.5,
                     complejo + (complejo**2-2)**.5)
```

```
    return(teoria)
}
```

Con lo que pudimos realizar la elección de diez números complejos, después de fijar una semilla, eligiendo uniformemente en $[-\sqrt{2}, \sqrt{2}]$ para la parte real y fijando $-i\epsilon$ con $\epsilon = 1e-05$ para la parte imaginaria.

```
set.seed(0)
epsilon <- 10e-6
n <- 10
x <- runif(n, -2**.5, 2**.5) #fijamos diez numeros en el plano complejo
z.s <- complex(real = x, imaginary = -rep(epsilon, n ))
z.s
```

```
##  [1]  1.1220291-0.00001i -0.6632417-0.00001i -0.3616882-0.00001i
##  [4]  0.2060604-0.00001i  1.1545860-0.00001i -0.8437709-0.00001i
##  [7]  1.1268162-0.00001i  1.2577316-0.00001i  0.4548048-0.00001i
## [10]  0.3651897-0.00001i
```

Con lo que al realizar el cálculo con estos diez números complejos se tienen los siguiente errores relativos (los cuales se pueden calcular porque disponemos del resultado analitico)

```
numericos <- vector(mode='complex', length = n)
teoricos <- vector(mode='complex', length = n)
for(i in 1:n)
{
    numericos[i] <- G.numerica(x, complejo = z.s[i])
    teoricos[i] <- G.teorica(z.s[i])
}
errores <- abs(numericos-teoricos)/abs(teoricos)
errores
```

```
##  [1] 1.581997e-09 6.555535e-09 1.081126e-08 1.002927e-08 4.284720e-09
##  [6] 8.416001e-09 3.228410e-08 2.021764e-08 7.931086e-09 1.074399e-08
```

Con media de `1.1285559\times 10^{-6}` porcentaje de error

Lo curioso es la pérdida de estabilidad numérico con un ϵ de dos órdenes de magnitud menor.

```
set.seed(0)
epsilon <- 10e-8
n <- 10
x <- runif(n, -2**.5, 2**.5) #fijamos diez numeros en el plano complejo
z.s <- complex(real = x, imaginary = -rep(epsilon, n ))
numericos <- vector(mode='complex', length = n)
teoricos <- vector(mode='complex', length = n)
for(i in 1:n)
{
    numericos[i] <- G.numerica(x, complejo = z.s[i])
    teoricos[i] <- G.teorica(z.s[i])
}
errores <- abs(numericos-teoricos)/abs(teoricos)
mean(errores)*100
```

```
## [1] 78.0237
```

Pero consideramos que con $\epsilon = 1e^{-05}$ se tiene una buena aproximación.

Ejercicio 2

Graficar la distribución empírica de probabilidad para el valor propio más grande de un ensemble de $m = 1000$ matrices simétricas de dimensión $n = 1e09$. Comparar el resultado con la distribución de Tracy-Widom (teórica). Se sugiere seguir la aproximación propuesta en: ***A. Edelman, Random Matrix Theory and its Innovative Applications (2013)****. Si se basa en este artículo para realizar la simulación, explicar los elementos claves del algoritmo ¿Cuál fue el tiempo de cómputo? ¿De que manera escala con m y n?*

Después de revisar el interesante artículo sugerido. Se implementó el código matlab que proveen en el lenguaje R para poder simular la distribución de los valores propios más grandes de matrices de dimensiones altas como billones ingleses, es decir $n = 1e09$.

Lo interesante de la metodología empleada reside en dos hechos estadísticos importantes. Primero que la distribución χ^2 es la suma de dos v.a. normales con misma media y varianza estandarizadas y que los ensembles GOE, GUE y GSE poseen invarianza rotacional.

Con lo anterior primero se realiza la observación de que una matriz simétrica de los ensembles citados, puede llevarse a una forma tridiagonal utilizando rotaciones y reflexiones (como lo hacen los métodos numéricos de Householder) donde los elementos de la diagonal superior e inferior se corresponden con la norma de un vector con m entradas normales, con $n - 1 \geq m \geq 1$, así estas entradas tienen distribución dada por la raíz cuadrada de v.a. χ^2 con m grados de libertad. Mientras que los elementos de la diagonal principal se distribuyen como normales con media cero y desviación estándar de 2.

Y lo interesante del método del articulo, y que desde mi perspectiva es igual de interesante que su resultado es que existen y se han implementado métodos para encontrar valores propios de interés en matrices esparcidas (los mayores en magnitud o menores o los que se encuentren alrededor de un punto de interés) de matrices tridiagonales y bidiagonales desde 1970 conocidos como los métodos de Arnoldi y de Lanczos (en nuestro caso cuando las matrices son simétricas podemos usar Lanczos en particular en su implementación de LAPACK, Implicited Restarted Lanczos, en el caso general se utiliza el método de Arnoldi) los cuales en lugar de ser cúbicos en complejidad son lineales en lo referente al número de entradas por fila diferentes de cero.

Finalmente una observación curiosa del artículo es que la distribución de los valores propios de estas matrices tridiagonales están fuertemente dominados por las primeras $10n^{1/3}$ filas de la matriz por lo que basta solo simular matrices de esta dimensión cuando estamos interesados en las de orden n cuidando los grados de libertad de las respectivas entradas χ^2.

Para concluir muestro un ejemplo de cómo escala el método propuesto en m y n, como lo mencionamos anteriormente el método es lineal en el número de repeticiones (tanto en tiempo de ejecución como en en memoria), por otra parte en el parámetro n el método es de orden $n^{1/3}$. Para comprobarlo realizamos a continuación la simulación para $m = 1$ y $n = 1e^{09}$ desplegamos el tiempo de ejecución, el cual tomamos como referencia y es de menos de 6 segundos.

```
library(Matrix) # packages con la implementacion de matrices 'sparse'
library(irlba)  # package con la implementacion de Lanczos para matrices esparcidas
beta <- 1       # la beta que señala el tipo de ensemble
set.seed(0)     # fijamos una semilla
n <- 1e9        # dimension de la matriz a simular
alpha <- 10     # el numero magico
m <- 1         # numero de repeticiones
k <- round(alpha*n**(1/3)) #reduciendo dimensionalidad
                           # resultado de que se conoce la distro de las tridiagonales
d.f <- beta*(n: (n - k+ 2)) # la convergencia de los val. prop. se ve dominada por las primeras 'k' fil
maxi.vals.prop <- vector(mode='numeric', length = m)
time1 <- Sys.time()
for (x in 1:m)
{
    i <- 1:k
```

```
    j <- 1:k
    H <- sparseMatrix(i,j,x=rep(0,k)) #inicializacion de matriz sparse
    index.diagonal.sup <- as.matrix(cbind( 1:(k-1),2:k))
    index.diagonal <- as.matrix(cbind(1:k, 1:k))
    colnames(index.diagonal.sup) <- colnames(index.diagonal)  <- c('x', 'y')
    H[index.diagonal] <-  rnorm(k)
    H[index.diagonal.sup] <- sapply(d.f ,
                                    function(x)
                                    {
                                        rchisq(n = 1, df=x )**.5
                                    }) # rellenamos la diagonal superior
    H <- (H + t(H))/sqrt(4*n*beta) # hacemos simetrica la matriz y escalamos
    valor <- partial_eigen(H, n=1, symmetric = T,
                           maxit=5300, tol=1e-5) #valor propio más grande
    maxi.vals.prop[x] <- valor$values
}
time2 <- Sys.time()
(time <- time2 - time1)
```

```
## Time difference of 5.014528 secs
```

```
##################
```

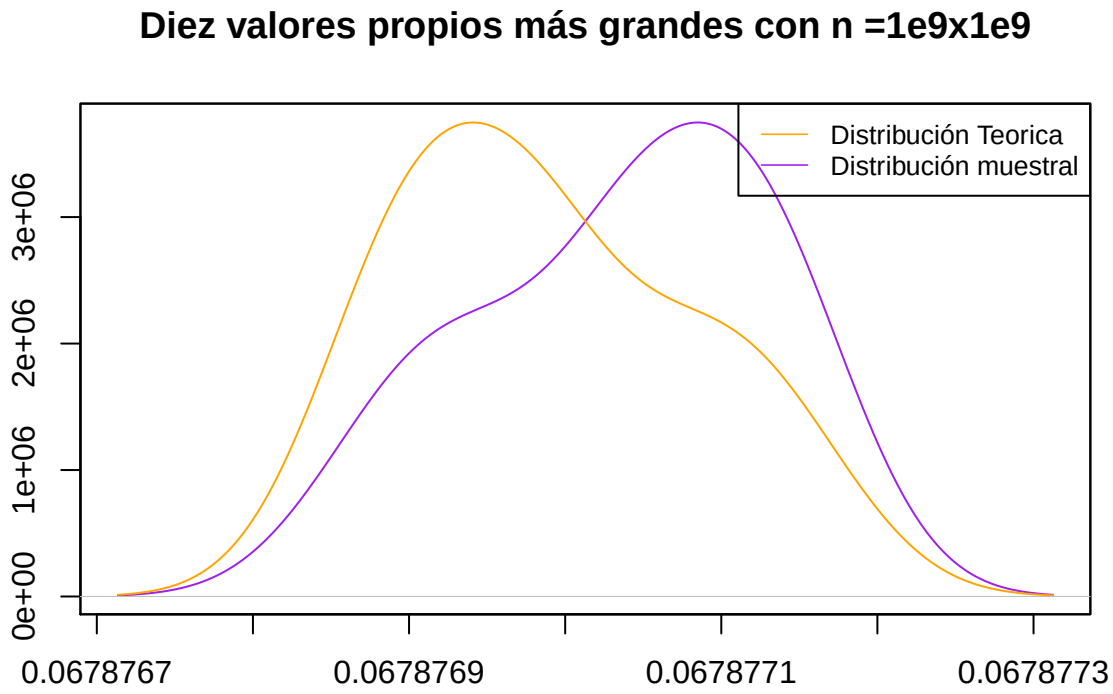
Ahora movemos $m = 10$ y lo comparamos con el resultado base, que requiere de aprox. 50 segundos, es decir casi 10 veces más tardado que el caso con $m = 1$ ($n = 1e^{09}$).

```
set.seed(0)      # fijamos una semilla
n <- 1e9         # dimension de la matriz a simular
alpha <- 10      # el numero magico
m <- 10          # numero de repeticiones
k <- round(alpha*n**(1/3)) #reduciendo dimensionalidad
                           # resultado de que se conoce la distro de las tridiagonales
d.f <- beta*(n: (n - k+ 2)) # la convergencia de los val. prop. se ve dominada por las primeras 'k' fil
maxi.vals.prop <- vector(mode='numeric', length = m)
time1 <- Sys.time()
for (x in 1:m)
{
    i <- 1:k
    j <- 1:k
    H <- sparseMatrix(i,j,x=rep(0,k)) #inicializacion de matriz sparse
    index.diagonal.sup <- as.matrix(cbind( 1:(k-1),2:k))
    index.diagonal <- as.matrix(cbind(1:k, 1:k))
    colnames(index.diagonal.sup) <- colnames(index.diagonal)  <- c('x', 'y')
    H[index.diagonal] <-  rnorm(k)
    H[index.diagonal.sup] <- sapply(d.f ,
                                    function(x)
                                    {
                                        rchisq(n = 1, df=x )**.5
                                    }) # rellenamos la diagonal superior
    H <- (H + t(H))/sqrt(4*n*beta) # hacemos simetrica la matriz y escalamos
    valor <- partial_eigen(H, n=1, symmetric = T,
                           maxit=5300, tol=1e-5) #valor propio más grande
    maxi.vals.prop[x] <- valor$values
}
time2 <- Sys.time()
```

```
(time <- time2 - time1)
```

```
## Time difference of 49.46005 secs
```

```
##################
##################
library(RMTstat)
plot(density(dtw(maxi.vals.prop, beta=1)),
             main = "Diez valores propios más grandes con n =1e9x1e9",
             xlab='', ylab='', col='purple')
legend("topright", legend=c("Distribución Teorica", "Distribución muestral"),
       col=c("orange", "purple"),cex = 0.8, lwd = 0.8)
par(new=TRUE)
plot(density(maxi.vals.prop), col="orange", xaxt='n',
     yaxt='n', main="", sub="",xlab = "",ylab = "")
```



Para terminar de explorar realizamos la simulación pero con $m = 1$ y $n = 1e^{08}$ que tarda aprox. 3 segundos, es decir 2/3 partes menos tardado que el caso base.

```
set.seed(0)      # fijamos una semilla
n <- 1e8         # dimension de la matriz a simular
alpha <- 10      # el numero magico
m <- 1          # numero de repeticiones
k <- round(alpha*n**(1/3)) #reduciendo dimensionalidad
                           # resultado de que se conoce la distro de las tridiagonales
d.f <- beta*(n: (n - k+ 2)) # la convergencia de los val. prop. se ve dominada por las primeras 'k' fil
maxi.vals.prop <- vector(mode='numeric', length = m)
```

```
time1 <- Sys.time()
for (x in 1:m)
{
    i <- 1:k
    j <- 1:k
    H <- sparseMatrix(i,j,x=rep(0,k)) #inicializacion de matriz sparse
    index.diagonal.sup <- as.matrix(cbind( 1:(k-1),2:k))
    index.diagonal <- as.matrix(cbind(1:k, 1:k))
    colnames(index.diagonal.sup) <- colnames(index.diagonal)  <- c('x', 'y')
    H[index.diagonal] <-  rnorm(k)
    H[index.diagonal.sup] <- sapply(d.f ,
                                    function(x)
                                    {
                                        rchisq(n = 1, df=x )**.5
                                    }) # rellenamos la diagonal superior
    H <- (H + t(H))/sqrt(4*n*beta) # hacemos simetrica la matriz y escalamos
    valor <- partial_eigen(H, n=1, symmetric = T,
                           maxit=5300, tol=1e-5) #valor propio más grande
    maxi.vals.prop[x] <- valor$values
}
time2 <- Sys.time()
(time <- time2 - time1)
```

```
## Time difference of 2.67419 secs
```

```
##################
```

Finalmente tenemos el caso requerido con $m = 1,000$ y $n = 1e^{09}$, que tardó 1.5 hrs.

```
set.seed(0)     # fijamos una semilla
n <- 1e9        # dimension de la matriz a simular
alpha <- 10     # el numero magico
m <- 1000         # numero de repeticiones
k <- round(alpha*n**(1/3)) #reduciendo dimensionalidad
                           # resultado de que se conoce la distro de las tridiagonales
d.f <- beta*(n: (n - k+ 2)) # la convergencia de los val. prop. se ve dominada por las primeras 'k' fil
maxi.vals.prop <- vector(mode='numeric', length = m)
time1 <- Sys.time()
for (x in 1:m)
{
    i <- 1:k
    j <- 1:k
    H <- sparseMatrix(i,j,x=rep(0,k)) #inicializacion de matriz sparse
    index.diagonal.sup <- as.matrix(cbind( 1:(k-1),2:k))
    index.diagonal <- as.matrix(cbind(1:k, 1:k))
    colnames(index.diagonal.sup) <- colnames(index.diagonal)  <- c('x', 'y')
    H[index.diagonal] <-  rnorm(k)
    H[index.diagonal.sup] <- sapply(d.f ,
                                    function(x)
                                    {
                                        rchisq(n = 1, df=x )**.5
                                    }) # rellenamos la diagonal superior
    H <- (H + t(H))/sqrt(4*n*beta) # hacemos simetrica la matriz y escalamos
    valor <- partial_eigen(H, n=1, symmetric = T,
                           maxit=5300, tol=1e-5) #valor propio más grande
```

```
    maxi.vals.prop[x] <- valor$values
}
time2 <- Sys.time()
(time <- time2 - time1)
```

```
## Time difference of 1.403651 hours
```

```
##################
plot(density(dtw(maxi.vals.prop, beta=1)),
             main = "Mil valores propios más grandes con n =1e9x1e9",
             xlab='', ylab='', col='purple')
legend("topright", legend=c("Distribución Teorica", "Distribución muestral"),
       col=c("orange", "purple"),cex = 0.8, lwd = 0.8)
par(new=TRUE)
plot(density(maxi.vals.prop), col="orange", xaxt='n', yaxt='n',
     main="", sub="",xlab = "",ylab = "")
```