Actividades en Finanzas 2: Valuación de Opciones y Simulaciones de Monte Carlo en R

Semana 3: R Script I

Ricardo Huamán

2022-03-10

Contents

1 R File Manipulation Commands

Limpiar el espacio de trabajo

```
rm(list=ls())
```

Para ubicar el espacio de trabajo actual, use el comando **getwd()**

```
getwd()
```

```
## [1] "C:/Users/sandr/Dropbox/PUCP/2022-0/Actividades en Finanzas/Dictado/1FIN17/clase3"
```

Para hacer un listado de los directorios y subdirectorios dentro del espacio de trabjo, use el comando **list.files()**

```
list.files()
```

```
## [1] "clase3.pdf" "clase3.Rmd"
```

Si se desea cambiar el espacio de trabajo, se puede settear usando el comando **setwd()**, y dentro de los paréntesis, puede ubicar el path del nuevo espacio de trabajo.

```
# setwd()
```

2 Instalar paquetes

```
install.packages("dplyr")
```

Luego de haber instalado el paquete, se debe llamar a la librería para poder ser usado en el presente script.

```
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

3 R como un espacio de calculadora

R usa los símbolos usuales de aritmética como + para suma, - para restas, * para multiplicaciónes, / para división, y ^ para exponente. Asimismo, se pueden usar paréntesis () para especificar el orden de las operaciones. R también usa %% para poder operar módulos y %/% para divisiones enteras.

```
(1 + 1/100)^100
```

```
## [1] 2.704814
```

```
17 %% 5
```

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

```
17 %/% 5
```

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

R tiene integrado cierto número de funciones: `sin(x)`, `cos(x)`, `tan(x)`, (todo en radiones), `exp(x)`, `log(x)`, y `sqrt(x)`. Algunas constantes especiales como `pi` también ya están predefinidas.

```
exp(1)
```

```
## [1] 2.718282
```

```
options(digits = 16)
exp(1)
```

```
## [1] 2.718281828459045
```

```
 pi
```

```
## [1] 3.141592653589793
```

```
sin(pi/6)
```

```
## [1] 0.4999999999999999
```

Las funciiones `floor(x)`y `ceiling(x)`redondean hacia arriba y hacia abajo, respectivamente, hasta el entero más cercano.

```
floor(sin(pi/6))
```

```
## [1] 0
```

```
ceiling(sin(pi/6))
```

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

4 Variables

4.1 Tipos de datos

Para asignar un valor a una variable, se utiliza el comando `<-`. Las variables se crean en la primera vez en que se les asigna un valor. Estas pueden ser llamadas por letras, números, y `.` o `_`, siempre que el nombre comience con una letra o número. Es importante mencionar que los nombres de las variables se distinguen entre mayúsculas y minúsculas.

Para mostrar el valor de una variable `x` en la pantalla, basta con escribir `x`. Esto es similar al conocido `print(x)`. No obstante, en algunas situaciones, tenemos que utilizar el formato más largo, o su equivalente cercano `show(x)`, por ejemplo, cuando para ver los esultados dentro de un loop.

- Definiendo objetos

```
x <- 100  # entero
```

```
(1 + 1/x)^x
```

```
## [1] 2.704813829421528
```

Actualizando el valor de la variable `x`

```
x <- 200
```

```
(1 + 1/x)^x
```

```
## [1] 2.711517122929317
```

Podemos almacenar el resultado de estas operaciones entre `()`para ver el output inmediatamente.

```
(y <- (1 + 1/x)^x)
```

```
## [1] 2.711517122929317
```

4.2 Funciones

En matemática, una función necesita de uno o más argumentos (inputs) para producir uno o más outputs.

Para llamar a una función en R, se necesita escribir el nombre de la función, seguida de sus argumentos dentro de paréntesis y separados por comas.

La función `seq`genera una secuencia aritmética.

```
seq(from = 1, to = 9, by = 2)
```

```
## [1] 1 3 5 7 9
```

Algunos argumentos son opcionales, y tienen valores predeterminador. Por ejemplo, si se omite el argumento `by`, R asume que `by = 1`.

```
seq(from = 1, to = 9)
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

Para obtener más información sobre cualquier función o paquete inslatado, puede tipear `help(name)`o `?name`

Cada funciòn tiene un orden determinado para sus argumentos. En este sentido, si se ingresan argumentos en ese orden, no necesitan ser llamados. Por otro lado, se puede elegir el orden en el que se ingresan los argumentos; para esto, se le debe agregar el nombre de dicho argumento.

```
seq(1, 9, 2)
```

```
## [1] 1 3 5 7 9
```

```
seq(to = 9, from = 1)
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

```
seq(by = -2, 9, 1)
```

```
## [1] 9 7 5 3 1
```

Cada argumento es una expresiòn que puede ser definida como una constante, una variable, otra funciòn o alguna combinaciòn algebraica de estas.

```
x <- 9
seq(1, x, x/3)
```

```
## [1] 1 4 7
```

4.3 Vectores

Un vector es una lista indexada de variables.

```
(x <- seq(1, 20, by = 2))
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19
```

```
(y <- rep(3, 4))
```

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

```
(z <- c(y, x))
```

```
##  [1]  3  3  3  3  1  3  5  7  9 11 13 15 17 19
```

Para referirnos al elemento i de un vector $\mathbf{x}$, usamos `x[i]`.

```
(x <- 100:110)
```

```
##  [1] 100 101 102 103 104 105 106 107 108 109 110
```

```
i <- c(1, 3, 2)
x[i]
```

```
## [1] 100 102 101
```

```
j <- c(-1, -2, -3)
x[j]
```

```
## [1] 103 104 105 106 107 108 109 110
```

```
x[1] <- 1000
x
```

```
##  [1] 1000  101  102  103  104  105  106  107  108  109  110
```

La función `length(x)`devuelve el número de elementos de `x` . Es posible tener un vector vacío.

```
x <- c()
length(x)
```

```
## [1] 0
```

Se puede realizar operaciones algebraicas entre vectores. Estas operaciones se aplican en cada uno de los elementos de un vector de forma separada.

```
x <- c(1, 2, 3)
y <- c(4, 5, 6)
x * y
```

```
## [1]  4 10 18
```

```
x + y
```

```
## [1] 5 7 9
```

```
y^x
```

```
## [1]   4  25 216
```

Cuando se realizan operaciones algebraicas entre vectores de diferentes dimensiones, R repite el vector más corto hasta obtener un resultado del mismo tamaño que el vector más largo.

```
c(1, 2, 3, 4) + c(1, 2)
```

```
## [1] 2 4 4 6
```

```
(1:10) ^ c(1, 2)
```

```
##  [1]   1   4   3  16   5  36   7  64   9 100
```

Esto sucede si incluso el vector más corto es de tamaño 1.

```
2 + c(1, 2, 3)
```

```
## [1] 3 4 5
```

```
2 * c(1, 2, 3)
```

```
## [1] 2 4 6
```

```
(1:10)^2
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

Algunas funciones útiles son:

- `sum(...)`
- `prod(...)`
- `max(...)`
- `min(...)`
- `sqrt(...)`
- `sort(...)`
- `mean(...)`
- `var(...)`

```
sqrt(1:6)
```

```
## [1] 1.000000000000000 1.414213562373095 1.732050807568877 2.000000000000000
## [5] 2.236067977499790 2.449489742783178
```

```
mean(1:6)
```

```
## [1] 3.5
```

```
sort(c(5,1,3,4,2))
```

```
## [1] 1 2 3 4 5
```

4.3.1 EJEMPLO: MEDIA Y VARIANZA

```
x <- c(1.2, 0.9, 0.8, 1.0, 1.2)
x.mean <- sum(x)/length(x)
x.mean - mean(x)
```

```
## [1] 0
```

```
x.var <- sum((x - x.mean)^2)/(length(x) - 1)
x.var - var(x)
```

```
## [1] 0
```

4.3.2 EJEMPLO: INTEGRACIÓN NUMÉRICA SIMPLE

```
dt <- 0.005
t <- seq(0, pi/6, by = dt)
ft <- cos(t)
(I <- sum(ft)*dt)
```

```
## [1] 0.5015486506255458
```

*Nota: `t` es un vector, entonces `ft` también es un vector, en donde `ft[i]` es igual a `cos(t[i])`.

```
I - sin(pi/6)
```

```
## [1] 0.001548650625545822
```

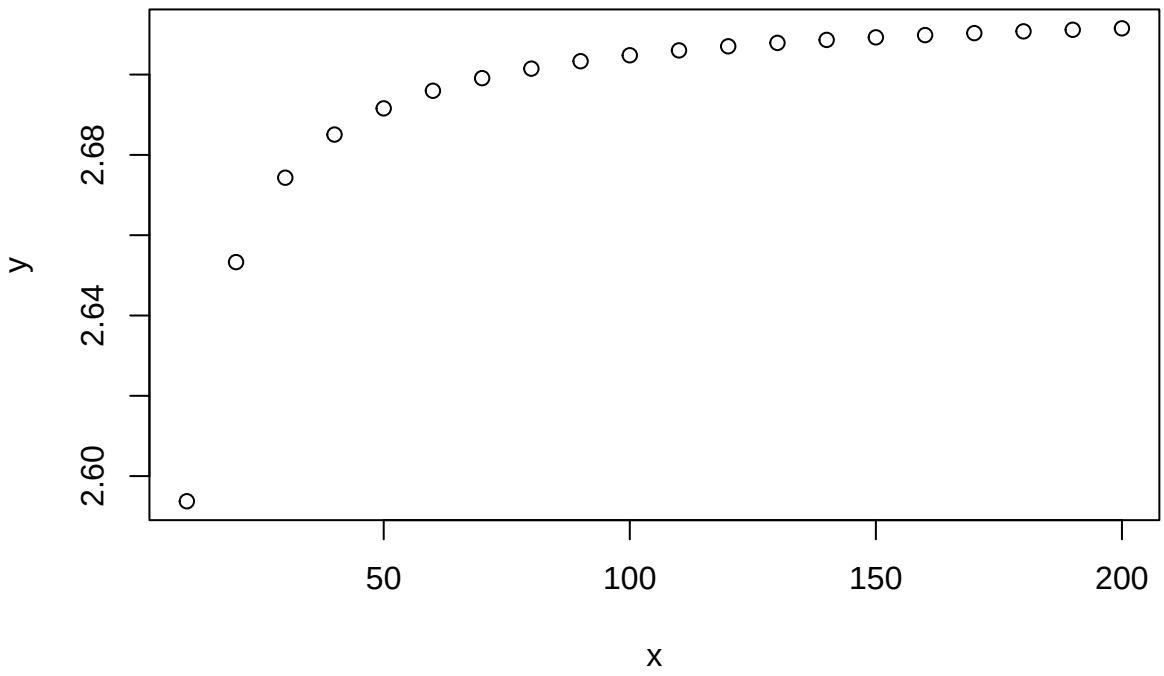
4.3.3 EJEMPLO: LÍMITE EXPONENCIAL

Para dibujar un vector contra otro, se usa la función `plot(x, y, type)`. Dentro de los argumentos de la función `plot`, `x`e `y` deben ser vectores de la misma longitud. El argumento `type`es opcional, utilizado para controlar la apariencia del gráfico: "`p`" para puntos (el valor predeterminado); "`l`" para las líneas; "`o`" para puntos sobre líneas; etc.

```
x <- seq(10, 200, by = 10)
y <- (1 + 1/x)^x
exp(1) - y
```

```
##  [1] 0.124539368359042779 0.064984123314622888 0.043963052588742446
##  [4] 0.033217990069081882 0.026693799385437256 0.022311689128828860
##  [7] 0.019165457482859694 0.016796887705717634 0.014949367400859170
## [10] 0.013467999037516609 0.012253746954290712 0.011240337596801542
## [13] 0.010381746740967479 0.009645014537900565 0.009005917124194074
## [16] 0.008446252151229849 0.007952077235180433 0.007512532619638357
## [19] 0.007119033847887923 0.006764705529727966
```

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



4.4 Datos missing: NA

En experimentos reales, sucede muy a menudo que ciertas observaciones registran datos missing. Dependiendo del análisis estadístico, los datos missing pueden ser ignorados o imputados.

R representa los datos missing a través del valor NA. Estos pueden ser parte de datos con otro tipo de registro. Se puede detectar si una variable contiene missing values usando `is.na(...)`.

```
a <- NA   # Asignando NA a una variable
is.na(a)  # ¿Esta variable es missing?
```

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

```
a <- c(11,NA,13)  # Asignando NA a uno de los elementos de un vector
is.na(a)          # Identificando los valores missings
```

```
## [1] FALSE  TRUE FALSE
```

```
any(is.na(a))   # ¿Hay algún dato missing en el vector a?
```

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

```
mean(a)
```

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

```
 mean(a, na.rm = TRUE)  # Los valores missings pueden ser removidos
```

```
## [1] 12
```

4.5 Expresiones lógicas

El valor que devuelve una expresión lógica es `TRUE` o `FALSE`. De la misma forma, los valores `1` y `0`, pueden ser usador para representar los valores `TRUE` o `FALSE`, respectivamente.

- `<` o `>`
- `<=` o `>=`
- `==` igual a
- `!=` no igual a
- `&` y
- `|` o
- `!` no
- Note que `A|B` es `TRUE` si `A` o `B` o ambos son `TRUE`. Si se busca una disyuntiva exclusiva, entre `A` o `B`, pero no ambos, usar `xor(A,B)`.

```
c(0,0,1,1)|c(0,1,0,1)
```

```
## [1] FALSE  TRUE  TRUE  TRUE
```

```
xor(c(0,0,1,1),c(0,1,0,1))
```

```
## [1] FALSE  TRUE  TRUE FALSE
```

4.5.1 EJEMPLO: Encontrar los números de 1 a 20 que son divisibles entre 4

```
x <- 1:20
x %% 4 == 0
```

```
##  [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
## [13] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
```

```
(y <- x[x %% 4 == 0])
```

```
## [1]  4  8 12 16 20
```

R provee la función `subset(x)` para elegir un vector o subvector de una variable x. La diferencia entre usar `subset` y usar el operador de indexación `x[i]`es que `subset` ignora los missing values, mientras que `x[i]` los preserva.

```
x <- c(1, NA, 3, 4)
x > 2
```

```
## [1] FALSE    NA  TRUE  TRUE
```

```
x[x > 2]
```

```
## [1] NA  3  4
```

```
subset(x, subset = x > 2)
```

```
## [1] 3 4
```

Si se busca conocer el índex de aquellos elementos que devuelven `TRUE` después de una expresión lógica, se usa `which(x)`.

```
x <- c(1, 1, 2, 3, 5, 8, 13)
which(x %% 2 == 0)
```

```
## [1] 3 6
```

4.5.2 EJEMPLO: ROUNDING ERROR

Solo los enteros y fracciones cuyo denominador es una potencia de 2 pueden representarse exactamente con sin decimales. Todos los demás números están sujetos a error de redondeo.

```
2 * 2 == 4
```

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

```
sqrt(2) * sqrt(2) == 2
```

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

El problema aquí es que `sqrt(2)` tiene error de redondeo que se magnifica cuando se le eleva al cuadrado. La solución es utilizar la función `all.equal(x, y)`, que devuelve `TRUE` si la diferencia entre `x` e `y` es menor que algún conjunto de tolerancia, basado en el nivel operativo de precisión de R.

```
all.equal(sqrt(2) * sqrt(2), 2)
```

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

4.6 Matrices

Una matriz es creada con vectores. Se usa la función `matrix(data, nrow = 1, ncol = 1, byrow = FALSE)`.

Aquí, `data` es un vector de longitud a más de dimensión `nrow*ncol`. `nrow` y `ncol` son los números de filas y columnas, respectivamente (con valores predeterminados de 1). `byrow` puede ser `TRUE` o `FALSE` (el valor predeterminado es FALSE) e indica si se desea rellenar la matriz fila por fila, o columna por columna, utilizando los elementos contenidos en `data`. Si `length(data)` es menor que `nrow*ncol`, los valores de `data` se reutilizarán tantas veces como sean necesarios.

```
(A <- matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE))
```

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

Para conocer la dimensión de una matriz, se usa `dim`:

```
dim(A)
```

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

Para crear una matriz diagonal, se usa `diag(x)`. Para unir matrices con la misma cantidad de filas, se usa `rbind(x)`. Para unir matrices con columnas de la misma extensión (unión horizontal), se usa `cbind()`.

Para referirse a los elementos de una matriz se usan dos índices.

```
A[1, 3] <- 0
A
```

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

```
A[, 2:3]
```

```
##      [,1] [,2]
## [1,]    2    0
## [2,]    5    6
```

```
(B <- diag(c(1, 2, 3)))
```

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

Las operaciones algebraicas habituales, incluyendo `*`, actúan elementalmente sobre matrices. Para realizar una multiplicación de matrices, se usa el operador `%*%`. Por otro lado, otras funciones que pueden ser usadas con matrices son '`nrow(x)`, `ncol(x)`, `det(x)`(el determinante), `t(x)` (la transpuesta) y `solve(A, B)`, que devuelve `x` tal que `A %*% x == B`. Si `A` es invertible, entonces `solve(A)` devuelve la matriz inversa de A.

```
(A <- matrix(c(3, 5, 2, 3), nrow = 2, ncol = 2))
```

```
##      [,1] [,2]
## [1,]    3    2
## [2,]    5    3
```

```
(B <- matrix(c(1, 1, 0, 1), nrow = 2, ncol = 2))
```

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

```
 A %*% B
```

```
##      [,1] [,2]
## [1,]    5    2
## [2,]    8    3
```

```
 A * B
```

```
##      [,1] [,2]
## [1,]    3    0
## [2,]    5    3
```

```
 (A.inv <- solve(A))
```

```
##                    [,1]               [,2]
## [1,] -3.000000000000004  2.000000000000003
## [2,]  5.000000000000007 -3.000000000000004
```

```
A %*% A.inv
```

```
##      [,1]                   [,2]
## [1,]    1 -8.881784197001252e-16
## [2,]    0  1.000000000000000e+00
```

```
 A^(-1)
```

```
##                    [,1]               [,2]
## [1,] 0.3333333333333333 0.5000000000000000
## [2,] 0.2000000000000000 0.3333333333333333
```

Mientras que R imprime un vector `x` (que no tiene atributos de dimensión), en operaciones con matrices, tratará a `x` como un vector fila o columna en su intento de hacer que los componentes sean operables.

```
A <- matrix(c(3, 5, 2, 3), nrow = 2, ncol = 2)
A
```

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

```
## [2,]    5    3
```

```
(x <- c(1,2))
```

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

```
x %*% A
```

```
##      [,1] [,2]
## [1,]   13    8
```

```
A %*% x
```

```
##      [,1]
## [1,]    7
## [2,]   11
```

4.7 Espacio de trabajo

Para hacer un listado de todos los objetos, use `ls()` o `objects()`.

Para remover un objeto `x`, use `rm(x)`.

Para remover todo los objetos, use `rm(list=ls())`.

Para guardar todos los objetos en un archivo llamado `fname` en el espacio de trabajo actual, use `save.image(file = "fname")`.

Para guardar objetos específicos, use `save(x, y, file = "fname")`.

Para abrir un set de objetos guardados, use `load(file = "fname")`.