
CODIGO R

A PREPRINT

GRUPO MESSI *
Facultad de Ingenieria
Tecnicas y Herramientas Modernas

April 10, 2024

Abstract

En este documento se vera la utilizacion del lenguaje de programacion R, sus comandos y sus utilidades al momento de necesitar vectores y matrices.

1 Introduction

El lenguaje de programacion R es un lenguaje de programación gratuito y de código abierto que se utiliza principalmente para análisis de datos, estadística y visualización de datos. Es muy popular entre científicos, investigadores y profesionales de datos por su flexibilidad y potencia.

2 Comandos

Hay distintos comandos que se utilizan en este lenguaje, algunos de ellos se nombran a continuacion:

- Si se desea saber que hace cada comando se coloca un signo de preguntas antes del comando el cual se cunsluta y se obtiene una explicacion del mismo y un ejemplo. Ejemplo: ?summary
- Si se tiene una grafica en la cual queremos obtener exactamente los valores e la grafica, lo que se hace es poner el nombre de la variable directamente y se obtendra cada punto caracteristico de la misma. Ejemplo: pressure
- Colocando el comando hist y las variables anteriormente analizadas, se logra un histograma de los valores que se obtienen en la grafica.Ejemplo: hist(pressure(signo pesos)temperature)
- Si se quiere cambiar el titulo del mismo se coloca luego del nombre de las variables, separando con una coma, main="nombre del titulo" Ejemplo: hist(pressure(signo pesos)temperature, main="Temperatura del reactor")
- Para cambiar el nombre de los ejes se coloca como en el caso anterior, separado por comas, xlab o ylab igual al nombre que se desea en comillas. Ejemplo: hist(pressure(signo pesos)temperature, main="temperatura del reactor", ylab="valores multiplicados por 100", xlab="hPa")

2.1 Vectores

Un vector es una estructura de datos que almacena numeros de doble presicion

```
mi_vector_a <-c(12,34,12,54,23,12,65,34,12,56,66)
mi_vector_b <- seq(1:16)

mi_vector_a
```

*Ing. Palma

```
## [1] 12 34 12 54 23 12 65 34 12 56 66
```

```
mi_vector_b
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

2.2 Matrices

Las matrices se parecen a los vectores pero tienen filas y columnas. Se alimentan de vectores

```
mi_matriz_c <- matrix(mi_vector_b,nrow=4,byrow=TRUE)
```

Para mostrar la matriz se coloca el nombre de la matriz.

```
mi_matriz_c
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
```

Para acceder a un elemento de la matriz ponemos la fila y la columna entre corchetes nombrando anteriormente a la matriz

```
mi_matriz_c[4,2]
```

```
## [1] 14
```

Para traer una fila completa de la matriz claco el nombre de la matriz y entre corchetes coloco el numero de fila y luego de la coma un espacio para que venga todas las columnas

```
mi_matriz_c[4, ]
```

```
## [1] 13 14 15 16
```

Para suprimir una linea y obtener la matriz resultante se coloca el nombre de la matriz y luego entre corchetes con un menos la linea que se quiere suprimir

```
mi_matriz_c[-2, ]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    9   10   11   12
## [3,]   13   14   15   16
```

2.3 Ejercicio TICTOC

```
library(tictoc)
tic("sleeping")
A<-20
print("dormire una siestita...")
```

```
## [1] "dormire una siestita..."
```

```
## [1] "dormire una siestita..."
Sys.sleep(2)
print("...suena el despertador")
```

```
## [1] "...suena el despertador"
```

```
## [1] "...suena el despertador"
toc()
```

```
## sleeping: 2.005 sec elapsed
```

2.4 Ejercicio Matriz de 100x100

Primero generamos la matriz de 100x100 con numeros aleatorios utilizando el siguiente comando

```
matriz_aleatoria <- matrix(rnorm(10000), nrow = 100, ncol = 100)
```

Luego iniciamos el cronómetro con la siguiente linea

```
tiempo_inicio <- Sys.time()
```

Operación que deseas medir (en este caso, calcular la inversa de la matriz)

```
inversa <- solve(matriz_aleatoria)
```

Se detiene el cronómetro

```
tiempo_fin <- Sys.time()
```

Calcula el tiempo de ejecución

```
tiempo_ejecucion <- tiempo_fin - tiempo_inicio
```

Imprime el tiempo de ejecución

```
print(paste("Tiempo de ejecución:", tiempo_ejecucion))
```

```
## [1] "Tiempo de ejecución: 0.00770258903503418"
```

2.5 Ejercicio Rbenchmark

```
library(microbenchmark)
set.seed(2017)
n <- 10000
p <- 100
X <- matrix(rnorm(n*p), n, p)
y <- X %>% rnorm(p) + rnorm(100)
check_for_equal_coefs <- function(values) {
  tol <- 1e-12
  max_error <- max(c(abs(values[[1]] - values[[2]]),
                     abs(values[[2]] - values[[3]]),
                     abs(values[[1]] - values[[3]])))
  max_error < tol
}
mbm <- microbenchmark("lm" = { b <- lm(y ~ X + 0)$coef },
                      "pseudoinverse" = {
    b <- solve(t(X) %*% X) %*% t(X) %*% y
  },
  "linear system" = {
    b <- solve(t(X) %*% X, t(X) %*% y)
  },
  check = check_for_equal_coefs)
```

Este código da por resultado gráficas en forma de violin en donde se puede analizar cuanto tiempo demora el código en tomar la información y dependiendo del ancho que tenga el violin implica el consumo de memoria que tiene el mismo.

2.6 Castigo de Taylor

Este algoritmo define una función llamada `castigo_de_taylor` que calcula la sumatoria según el método del castigo de Taylor. El método consiste en sumar cada número entero junto con su inmediato superior, comenzando desde un valor dado y terminando en otro valor dado. La función toma dos argumentos `a` y `b`, que representan los dos números entre los cuales se realizará la sumatoria. Luego, utiliza un bucle `for` para iterar desde `a` hasta `b`, sumando cada número `i` junto con su inmediato superior `i + 1`. Finalmente, devuelve la suma total. Además, el algoritmo solicita al usuario que ingrese dos números, llama a la función `castigo_de_taylor` con estos números como argumentos, y muestra el resultado de la sumatoria.

Lo primero que hacemos es definir las variables y luego correr el código en base a ellas.

```
a<- 5
b<- 10

taylor_sum <- function(a,b) {
  sum <- 0
  for (i in a:b) {
    sum <- sum + i + (i+1)
  }
  return(sum)
}

resultado <- taylor_sum(a, b)

cat("La sumatoria usando el método de Taylor es:", resultado, "\n")
```

```
## La sumatoria usando el método de Taylor es: 96
```

2.7 Fibonacci

La serie de Fibonacci es una famosa secuencia de números infinita que aparece a menudo en la naturaleza y las matemáticas.

Aquí te cuento algunos datos interesantes sobre la serie de Fibonacci:

Su definición: La serie de Fibonacci se define por la regla siguiente: Los dos primeros números son 0 y 1. A partir del tercer término, cada número es la suma de los dos anteriores. **Sucesión inicial:** Así que la serie de Fibonacci comienza de la siguiente manera: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144... y sigue infinitamente. **Origen del nombre:** Aunque la secuencia se conocía en la India antes del siglo VII, la introdujo en Europa el matemático italiano Leonardo de Pisa, más conocido como Fibonacci, en su libro *Liber abbaci* ("Libro de cálculo") en el año 1202. De ahí su nombre. La serie de Fibonacci tiene muchas propiedades interesantes y aplicaciones en diversos campos, incluyendo:

Naturaleza: La serie de Fibonacci aparece en patrones de crecimiento en plantas (como el número de pétalos en una margarita), la disposición de las hojas en un tallo, o la forma de una espiral en una concha. **Matemáticas:** La serie de Fibonacci está relacionada con el número áureo, una proporción geométrica que se considera estéticamente agradable y se encuentra en la naturaleza y el arte. **Informática:** La serie de Fibonacci se utiliza en algoritmos de búsqueda y optimización.

La serie fibonacci es muy utilizada debido a su poder de predicción. Se utiliza en los mercados de criptomonedas para saber que puede pasar, y en caso que no se cumpla se dice que hubo una manipulación humana.