Ejercicios

Uriel Paluch

1/10/2021

Ejercicio 1

Utilice el metodo de Newton para resolver el siguiente sistema de ecuaciones no lineales tomando como valores iniciales X(0)=(0.1;0.1) o X(0)=(0.1;0.1;-0.1) segun corresponda. Verificar si la solucion hallada (x1, x2) o (x1, x2, x3) satisface las ecuaciones.

```
# Metodo de Newton ------------------------------------------------------
newton <- function(x, TOL, N = 100, ecuaciones){
  # x: aproximacion inicial
  # TOL: tolerancia
  # N: cantidad maxima de iteraciones

  # Instancio las variables -----------------------------------------------
  # n: numero de ecuaciones e incognitas
  n <- length(ecuaciones)
  # funciones valuadas en cero
  f0 <- rep(NA, n)
  jacobiano = matrix(rep(NA, n*n), nrow = n, ncol = n)

  # Comienza el metodo ----------------------------------------------------
  for (max_reps in 1:N) {
    # Recorro las filas
    for (i in 1:n) {
      #Evaluo las ecuaciones
      f0[i] <- eval(ecuaciones[i], x)
      # Recorro las columnas
      for (j in 1:n){
        # Derivo en cada variable y evaluo
        jacobiano[i,j] <- eval((D(ecuaciones[i], glue::glue("x",j))), x)
      }
    }

    y0 <- solve(jacobiano) %*% (-f0)
    x <- y0 + unlist(x, use.names=FALSE)

    norma <- norm(y0,type = 'M')
    if (norma < TOL) {
      return(x)
    }

    x0 <- list()
    for (i in 1:n) {
      x0[glue::glue("x",i)] <- x[i]
```

```
    }
    x <- x0

  }
  return("Numero de iteraciones maximo excedido")
}
```

```
# Metodo de Broyden ------------------------------------------------------
Broyden <- function(x, TOL, N = 100, ecuaciones){
  # x: aproximación inicial
  # TOL: tolerancia
  # N: cantidad maxima de iteraciones

  # Instancio las variables ------------------------------------------------
  # n: número de ecuaciones e incognitas
  n <- length(ecuaciones)
  # funciones valuadas en cero
  f0 <- rep(NA, n)
  jacobiano <- matrix(rep(NA, n*n), nrow = n, ncol = n)

  # Recorro las filas
  for (i in 1:n) {
    #Evaluo las ecuaciones
    f0[i] <- eval(ecuaciones[i], x)
    # Recorro las columnas
    for (j in 1:n){
      # Derivo en cada variable y evaluo
      jacobiano[i,j] <- eval((D(ecuaciones[i], glue::glue("x",j))), x)
    }
  }

  # Calcula la inversa de una matriz
  A <- solve(jacobiano)
  v <- f0
  s <- -A %*% v

  x <- unlist(x, use.names = FALSE) + s

  for (max in 1:N) {
    w <- v

    # listo las x
    x0 <- list()
    for (i in 1:n) {
      x0[glue::glue("x",i)] <- x[i]
    }
    x <- x0
    for (i in 1:n) {
      #Evaluo las ecuaciones
      f0[i] <- eval(ecuaciones[i], x)
    }
    v <- f0

    y <- v - w
```

```
    z <- -A%*%y

    p <- -t(s) %*% z

    u <- t(s) %*% A
    u <- t(u)

    A <- A + (1/p)[1,1] * (s+z) %*% t(u)

    s <- -A%*%v
    x <- unlist(x, use.names = FALSE) + s

    norma <- norm(s,type = 'M')
    if (norma < TOL) {
      return(x)
    }
  }
  return("Numero de iteraciones maximo excedido")
}
```

a.

```
# IMPORTANTE: declarar las funciones con x1, x2, ..., xn
print(newton(x = list(x1 = 0.1, x2 = 0.1), TOL = 0.000000001,
       ecuaciones = c(
                      expression(3*x1^2-x2^2),
                      expression(3*x1*x2^2-x1^3-1)
                      )
))
```

```
##           [,1]
## [1,] 0.5000000
## [2,] 0.8660254
```

b.

```
# IMPORTANTE: declarar las funciones con x1, x2, ..., xn
print(newton(x = list(x1 = 0.1, x2 = 0.1, x3 = -0.1), TOL = 0.000000001,
       ecuaciones = c(
                      expression(3*x1-cos(x2*x3)-0.5),
                      expression(4*x1^2-625*x2^2+2*x2-1),
                      expression(20*x3+(10*pi-3)/3+exp(-x1*x2))
                      )
))
```

```
##              [,1]
## [1,]  0.499999533
## [2,]  0.003199065
## [3,] -0.523518863
```

c.

```
# IMPORTANTE: declarar las funciones con x1, x2, ..., xn
print(newton(x = list(x1 = 0.1, x2 = 0.1, x3 = -0.1), TOL = 0.000000001,
       ecuaciones = c(
                       expression(x1^3+x1^2*x2-x1*x3+6),
                       expression(exp(x1)+exp(x2)-x3),
                       expression(x2^2-2*x1*x3)
                       )
))
```

```
##             [,1]
## [1,]  1.167123
## [2,] -2.765193
## [3,]  3.275701
```

d.

```
# IMPORTANTE: declarar las funciones con x1, x2, ..., xn
print(newton(x = list(x1 = 0.1, x2 = 0.1), TOL = 0.000000001,
       ecuaciones = c(
                       expression(5*x1^2-x2^2),
                       expression(x2-0.25* ( sin(x1) + cos(x2) ) )
                       )
))
```

```
##             [,1]
## [1,] 0.1212419
## [2,] 0.2711052
```

e.

```
# IMPORTANTE: declarar las funciones con x1, x2, ..., xn
print(newton(x = list(x1 = 0.1, x2 = 0.1), TOL = 0.000000001,
       ecuaciones = c(
                       expression(log(x1^2+x2^2)-sin(x1*x2)-log(2)),
                       expression(exp(x1-x2)+cos(x1*x2))
                       )
))
```

```
##              [,1]
## [1,] -2.0938850
## [2,] -0.8967253
```

Ejercicio 2

Utilice el método de Broydenpara resolver el siguiente sistema de ecuaciones no lineales. Realice supuestos razonables respecto de la tolerancia, verificando que la solución hallada (x1, x2, x3) satisface el par de ecuaciones.

a.

```
# Llamo a la funcion ------------------------------------------------------
# IMPORTANTE: declarar las funciones con x1, x2, ..., xn
print(Broyden(x = list(x1 = 0.1, x2 = 0.1, x3 = -0.1), TOL = 0.000001,
              ecuaciones = c(
```

```
              expression(3*x1-cos(x2*x3)-0.5),
              expression(x1^2-81*(x2+0.1)^2+sin(x3)+1.06),
              expression(exp(-x1*x2)+20*x3+(10*pi-3)/3)
            )
))
```

```
##                         [,1]
## [1,]  0.5000000000003339551
## [2,]  0.0000000000005346354
## [3,] -0.5235987755991023951
```

b.

```
# Llamo a la funcion ----------------------------------------------------
# IMPORTANTE: declarar las funciones con x1, x2, ..., xn
print(Broyden(x = list(x1 = -1, x2 = -2, x3 = 1), TOL = 0.000001,
              ecuaciones = c(
                expression(x1^3+x1^2*x2-x1*x3+6),
                expression(exp(x1)+exp(x2)-x3),
                expression(x2^2-2*x1*x3-4)
              )
))
```

```
##            [,1]
## [1,] -1.4560428
## [2,] -1.6642305
## [3,]  0.4224934
```

c.

```
# Llamo a la funcion ----------------------------------------------------
# IMPORTANTE: declarar las funciones con x1, x2, ..., xn
print(Broyden(x = list(x1 = 0, x2 = 0, x3 = 0), TOL = 0.00001,
              ecuaciones = c(
                expression(6*x1-2*cos(x2*x3)-1),
                expression(9*x2+sqrt(x1^2+sin(x3)+1.06)+0.9),
                expression(60*x3+3*exp(-x1*x2)+10*pi-3)
              )
))
```

```
##            [,1]
## [1,]  0.4981447
## [2,] -0.1996059
## [3,] -0.5288260
```

d.

```
# Llamo a la funcion ----------------------------------------------------
# IMPORTANTE: declarar las funciones con x1, x2, ..., xn
print(Broyden(x = list(x1 = 2, x2 = 2), TOL = 0.000001,
              ecuaciones = c(
                expression(log(x1^2+x2^2) - sin(x1*x2)-log(2)-log(pi)),
                expression(exp(x1-x2)+cos(x1*x2))
```

```
            )
))
```

```
##          [,1]
## [1,] 1.772454
## [2,] 1.772454
```

e.

```
# Llamo a la funcion ------------------------------------------------------
# IMPORTANTE: declarar las funciones con x1, x2, ..., xn
print(Broyden(x = list(x1 = 0, x2 = 0), TOL = 0.0001,
              ecuaciones = c(
                expression(4*x1^2-20*x1+0.25*x2^2+8),
                expression(0.5*x1*x2^2+2*x1-5*x2+8)
              )
))
```

```
##      [,1]
## [1,]  0.5
## [2,]  2.0
```