

Ejercicios Broyden

Agustin Huczok

2/10/2021

#a

Vector de aproximaciones iniciales —

```
x <- matrix( c(0.1, 0.1,-0.1),  
             nrow = 3 ,  
             ncol = 1,  
             byrow = TRUE)
```

Elementos de la matriz Jacobiana $J(X)$, derivadas de las funciones

```
fa <- function(x1,x2,x3){  
  3*x1-cos(x2*x3)-0.5  
}  
  
fae=expression(3*x1-cos(x2*x3)-0.5)  
D(fae,"x1")
```

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

```
D(fae,"x2")
```

```
## sin(x2 * x3) * x3
```

```
D(fae,"x3")
```

```
## sin(x2 * x3) * x2
```

```
dfa <- function(x1,x2,x3){  
  dfa1 <- 3  
  dfa2 <- sin(x2 * x3) * x3  
  dfa3 <- sin(x2 * x3) * x2  
  
  return(matrix(c(dfa1,dfa2,dfa3),
```

```

      nrow = 1, ncol = 3))
}

fb <- function(x1,x2,x3){
  x1^2-81*(x2+0.1)^2+sin(x3)+1.06
}
fbe=expression(x1^2-81*(x2+0.1)^2+sin(x3)+1.06)
D(fbe,"x1")

```

```
## 2 * x1
```

```
D(fbe,"x2")
```

```
## -(81 * (2 * (x2 + 0.1)))
```

```
D(fbe,"x3")
```

```
## cos(x3)
```

```

dfb <- function(x1,x2,x3){
  dfb1 <- 2 * x1
  dfb2 <- -(81 * (2 * (x2 + 0.1)))
  dfb3 <- cos(x3)

  return(matrix(c(dfb1,dfb2,dfb3),
                  nrow = 1, ncol = 3))
}

fc <- function(x1,x2,x3){
  exp(-x1*x2)+20*x3+(10*pi-3)/3
}
fce=expression(exp(-x1*x2)+20*x3+(10*pi-3)/3)
D(fce,"x1")

```

```
## -(exp(-x1 * x2) * x2)
```

```
D(fce,"x2")
```

```
## -(exp(-x1 * x2) * x1)
```

```
D(fce,"x3")
```

```
## [1] 20
```

```

dfc <- function(x1,x2,x3){
  dfc1 <- -(exp(-x1 * x2) * x2)
  dfc2 <- -(exp(-x1 * x2) * x1)
  dfc3 <- 20

  return(matrix(c(dfc1,dfc2,dfc3),
                  nrow = 1, ncol = 3))
}

```

##Algoritmo

```
BroydenSEnol <- function(x , TOL, N){  
  
  # defino el vector FX adentro para que tome los valores de x  
  FX <- matrix(c(fa(x[1],x[2],x[3]),  
                 fb(x[1],x[2],x[3]),  
                 fc(x[1],x[2],x[3])),ncol = 1,nrow = 3,byrow=TRUE)  
  
  # defino la matriz Jacobiana J(X) que tiene las derivadas de las funciones de F respecto a cada varia  
  # ordenada por fila  
  Jacobiano <- matrix(data = c(dfa(x[1],x[2],x[3]),  
                                dfb(x[1],x[2],x[3]),  
                                dfc(x[1],x[2],x[3])),ncol = 3, nrow = 3,byrow = TRUE)  
  
  #Paso 1 asigno FX a v  
  v <- FX  
  
  # Paso 2 invierto la matriz A0 y la llamo A  
  A <- solve(Jacobiano)  
  # Paso 3 creo el vector s resultado de Av y x  
  s <- -A%*%v  
  x <- x + s  
  k <- 2  
  # Paso 4 empieza el bucle y va iterando y reasignando los vectores  
  while(k <= N){  
  
    # Paso 5  
    w <- v  
    v <- matrix(c(fa(x[1],x[2],x[3]),  
                  fb(x[1],x[2],x[3]),  
                  fc(x[1],x[2],x[3])),ncol = 1,nrow = 3,byrow=TRUE)  
  
    y <- v-w  
  
    # Paso 6  
    z <- -A%*%y  
  
    # Paso 7 uso vector s traspuesto para p  
    p <- -t(s)%*%z  
  
    # Paso 8 (u) traspuesta (el resultado ya es traspuesto)  
    u <- t(s)%*%A  
  
    # paso 9 Reasigno la matriz A, el vect s y el vect x para la nueva vuelta  
    A <- A + (s+z)%*%(u/p[1])  
  
    # Paso 10  
    s <- -A%*%v  
  
    # Paso 11  
    x <- x + s  
  
    # Paso 12 condicion para salir del while  
    norma <- norm(s,type = 'F')  
    if (norma <= TOL){
```

```

    return(x)
    break
  }
  k <- k + 1
}
return(paste('El procedimiento fallo luego de superar el maximo de', N,'iteraciones'))
}

```

##Aplico logaritmo

```

Resultado=BroydenSEnoL(x,10^-10,1000)
Resultado

```

```

##           [,1]
## [1,]  5.000000e-01
## [2,]  1.664988e-13
## [3,] -5.235988e-01

```

Corroboro

```

#Asigno los rdos del algoritmo a las variables x1,x2
x1 <- BroydenSEnoL(x, 10^-10, 1000)[1] #posicion, osea mult por posicion 1
x2 <- BroydenSEnoL(x, 10^-10, 1000)[2]
x3 <- BroydenSEnoL(x, 10^-10, 1000)[3]

```

##Resultados

```
fa(x1, x2, x3)
```

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

```
fb(x1, x2, x3)
```

```
## [1] -2.691181e-12
```

```
fc(x1, x2, x3)
```

```
## [1] 6.039613e-14
```

```
#b
```

```
#c
```

```
#d
```

```
#e
```

Vector de aproximaciones iniciales —

```
x <- matrix( c(0.1, 0.1),
             nrow = 2 ,
             ncol = 1,
             byrow = TRUE)
```

Elementos de la matriz Jacobiana J(X), derivadas de las funciones

```
fa <- function(x1,x2){
  4*x1^2-20*x1+0.25*x2^2+8
}

fae=expression(4*x1^2-20*x1+0.25*x2^2+8)
D(fae,"x1")
```

```
## 4 * (2 * x1) - 20
```

```
D(fae,"x2")
```

```
## 0.25 * (2 * x2)
```

```
dfa <- function(x1,x2){
  dfa1 <- 4 * (2 * x1) - 20
  dfa2 <- 0.25 * (2 * x2)

  return(matrix(c(dfa1,dfa2),
                 nrow = 1, ncol = 2))
}

fb <- function(x1,x2){
  0.5*x1*x2^2+2*x1-5*x2+8
}

fbe=expression(0.5*x1*x2^2+2*x1-5*x2+8)
D(fbe,"x1")
```

```
## 0.5 * x2^2 + 2
```

```
D(fbe,"x2")
```

```
## 0.5 * x1 * (2 * x2) - 5
```

```
dfb <- function(x1,x2){
  dfb1 <- 0.5 * x2^2 + 2
  dfb2 <- 0.5 * x1 * (2 * x2) - 5

  return(matrix(c(dfb1,dfb2),
                 nrow = 1, ncol = 2))
}
```

##Algoritmo

```
BroydenSEnol <- function(x , TOL, N){

  # defino el vector FX adentro para que tome los valores de x
  FX <- matrix(c(fa(x[1],x[2]),
                 fb(x[1],x[2])
                 ),ncol = 1,nrow = 2,byrow=TRUE)

  # defino la matriz Jacobiana J(X) que tiene las derivadas de las funciones de F respecto a cada varia
  # ordenada por fila
  Jacobiano <- matrix(data = c(dfa(x[1],x[2]),
                               dfb(x[1],x[2])),ncol = 2, nrow = 2,byrow = TRUE)

  #Paso 1 asigno FX a v
  v <- FX

  # Paso 2 invierto la matriz A0 y la llamo A
  A <- solve(Jacobiano)
  # Paso 3 creo el vector s resultado de Av y x
  s <- -A%*%v
  x <- x + s
  k <- 2
  # Paso 4 empieza el bucle y va iterando y reasignando los vectores
  while(k <= N){

    # Paso 5
    w <- v
    v <- matrix(c(fa(x[1],x[2]),
                  fb(x[1],x[2])),ncol = 1,nrow = 2,byrow=TRUE)
    y <- v-w

    # Paso 6
    z <- -A%*%y

    # Paso 7 uso vector s traspuesto para p
    p <- -t(s)%*%z

    # Paso 8 (u) traspuesta (el resultado ya es traspuesto)
    u <- t(s)%*%A

    # paso 9 Reasigno la matriz A, el vect s y el vect x para la nueva vuelta
    A <- A + (s+z)%*%(u/p[1])

    # Paso 10
    s <- -A%*%v

    # Paso 11
    x <- x + s

    # Paso 12 condicion para salir del while
    norma <- norm(s,type = 'F')
    if (norma <= TOL){
      return(x)
    }
  }
}
```

```

        break
    }
    k <- k + 1
}
return(paste('El procedimiento fallo luego de superar el maximo de', N, 'iteraciones'))
}

```

##Defino Fx

```

Fx <- function(x){
  Fx <- rbind(fa(x[1],x[2]), fb(x[1],x[2]))
  return(Fx)
}

```

##Aplico logaritmo

```

Resultado=BroydenSEnol(x,10^-5,1000)
Resultado

```

```

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

```

Corroboro

```

#Asigno los rdos del algoritmo a las variables x1,x2
x1 <- BroydenSEnol(x, 10^-5, 100)[1] #posicion, osea mult por posicion 1
x2 <- BroydenSEnol(x, 10^-5, 100)[2]

```

##Resultados

```

fa(x1, x2)

```

```

## [1] -1.178261e-09

```

```

fb(x1, x2)

```

```

## [1] -6.917489e-10

```