Broyden

Agustin Huczok

2/10/2021

#2 variables

```
options(scipen=999)
```

Vector de aproximaciones iniciales —-

```
x <- matrix( c(0, 0),
               nrow = 2 ,
               ncol = 1,
               byrow = TRUE)
```

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

```
fa <- function(x1,x2){

}

fae=expression()
D(fae,"x1")
```

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

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

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

```
dfa <- function(x1,x2){
  dfa1 <-
  dfa2 <-


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

```
fb <- function(x1,x2){

}
  fbe=expression()
  D(fbe,"x1")
```

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

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

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

```
  dfb <- function(x1,x2){
    dfb1 <-
    dfb2 <-


    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
```

Corroboro

```
#Asigno los rdos del algoritmo a las variables x1,x2
#x1 <- BroydenSEnoL(x, 10^-5, 1000)[1] #posicion, osea mult por posicion 1
#x2 <- BroydenSEnoL(x, 10^-5, 1000)[2]
```

##Resultados

```
#fa(x1, x2)
#fb(x1, x2)
```

#3 variables

```
options(scipen=999)
```

Vector de aproximaciones iniciales —-

```
x <- matrix( c(0, 0,0),
               nrow = 3 ,
               ncol = 1,
               byrow = TRUE)
```

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

```
fa <- function(x1,x2,x3){

}

fae=expression()
D(fae,"x1")
```

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

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

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

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

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

```
dfa <- function(x1,x2,x3){
  dfa1 <-
  dfa2 <-
  dfa3 <-


  return(matrix(c(dfa1,dfa2,dfa3),
                nrow = 1, ncol = 3))
}

fb <- function(x1,x2,x3){

}
  fbe=expression()
  D(fbe,"x1")
```

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

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

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

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

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

```
  dfb <- function(x1,x2,x3){
    dfb1 <-
    dfb2 <-
    dfb3 <-

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

  fc <- function(x1,x2,x3){

}
  fce=expression()
  D(fce,"x1")
```

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

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

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

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

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

```
  dfc <- function(x1,x2,x3){
    dfc1 <-
    dfc2 <-
    dfc3 <-

    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^-5,1000)
#Resultado
```

Corroboro

```
#Asigno los rdos del algoritmo a las variables x1,x2
#x1 <- BroydenSEnoL(x, 10^-5, 1000)[1] #posicion, osea mult por posicion 1
#x2 <- BroydenSEnoL(x, 10^-5, 1000)[2]
#x3 <- BroydenSEnoL(x, 10^-5, 1000)[2]
```

##Resultados

```
#fa(x1, x2, x3)
#fb(x1, x2, x3)
#fc(x1, x2, x3)
```