Método de Newton

Allan Victor Almeida Faria

190127180

04/12/2019

Método de Newton para sistemas não lineares:

Consiste em um sistema de **n** váriaveis onde $F(x) \in (f_1, ..., f_n)$ para $F(x) = 0$:

$$F(x) = \begin{cases} f_1(x_1, ..., x_n) = 0 \\ ... \\ f_n(x_1, ..., x_n) = 0 \end{cases}$$

Um sistema onde admite solução única para $x_1, ..., x_n$, o cálculo é feito a partir da linearização das funções $(f_1, ..., f_n)$, onde o **Método de Newton** para uma função $f_i(x) = 0$, $i = 1, ..., n$, é linearizado em torno de um $x^{(k)}$ onde é dado implicitamente por $f_i(x) = f_i(x^{(k)}) + \nabla f_i(x^{(k)})(x - x^{(k)})$, no qual $x^{(0)} \in D : F(X)$, gera uma sequência $\{x^{(k)}\}$ de vetores onde há uma covergência quando $\lim\limits_{k \to \infty} x^{(k)} = x^*$ para um vetor solução x^* .

Assim a linearização do sistema $F(x) = 0$ é dada como $F(x) = F(x^{(k)}) + J(x^{(k)})(x - x^{(k)})$ onde $J(x)$ é o Jacobiano de $F(x)$. Como todo método de um algorítimo interativo, precisa de um critério de parada, interpretando $(x - x^{(k)}) = S^{(k)}$, temos $x^{(k+1)} = x^{(k)} + S^{(k)}$, e admitindo $F(x) = 0$, então $J(x^{(k)})S = -F(x^{(k)})$ onde S é a solução do sistema linear:

$$S^{(k)} = \begin{cases} \dfrac{\partial f_1(x^{(k)})}{\partial x_1} s_1 + ... + \dfrac{\partial f_1(x^{(k)})}{\partial x_n} s_n = -f_1(x^{(k)}) \\ ... \\ \dfrac{\partial f_n(x^{(k)})}{\partial x_1} s_1 + ... + \dfrac{\partial f_n(x^{(k)})}{\partial x_n} s_n = -f_n(x^{(k)}) \end{cases} , x^{(k)} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ ... \\ x_n^{(k)} \end{bmatrix}, S^{(k)} = \begin{bmatrix} s_1^{(k)} \\ s_2^{(k)} \\ ... \\ s_n^{(k)} \end{bmatrix}$$

$$x^{(k+1)} = \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ ... \\ x_n^{(k+1)} \end{bmatrix} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ ... \\ x_n^{(k)} \end{bmatrix} + \begin{bmatrix} s_1^{(k)} \\ s_2^{(k)} \\ ... \\ s_n^{(k)} \end{bmatrix}$$

No critério de parada, analizamos se ${}_{max}\{|F(x^{(k)})|\} < \epsilon_1$ ou ${}_{máx}\{|x^{(k+1)} - x^{(k)}|\} < \epsilon_2$.

Algorítimo:

Especificações:

A função criada no *Software R* é dada como **MtNewton(expressions, name, n, e, test)** onde:

- **expressions:** É as expressões do $F(x)$ dada em uma **lista** como intepretação de um sistema.
- **name:** São as variáveis de F(x) dado como vetor.
- **n:** É o número de variáveis existentes.
- **e:** É o critério do erro para a parada do algorítimo, onde se admite $\epsilon_1 = \epsilon_2 = e$.
- **test:** É a primeira interação $x^{(0)}$, dado como vetor, podendo começar em qualquer número que pertença ao domínio de F(x).

Algorítimo no R:

```
MtNewton <- function(expressions,name,n,e,test) {
############## Criando as Matrizes ##############
  Mj<-matrix(0, ncol = n, nrow = n)  # Matriz do Jacobiano = Mj
  Mf<-matrix(0,nrow = n)             # Matriz da funçao F = Mf
  Mx<-matrix(0,nrow = n)             # Matriz do valor das váriaveis
  cnt = 1
  for (v in name) {  # Atribuir os valores para as variáveis
    Mx[cnt,] <- test[cnt]         # Atribuindo valor do Xo na matriz Mx
    assign(v,test[cnt])
    cnt = cnt +1
  }
  for (i in 1:n) {    # Atribuindo o valor do Xo nas matrizes da Função F (Mf)
                      # e do Jacobiano (Mj)
    jacob <- deriv(expressions[[i]],name)

    fx <- eval(jacob)
    Mf[i] <- fx[1]

    j <- attributes(eval(jacob))
    Mj[i,]<-j$gradient
  }

####### Testes para ver se a interação continua ou para com base no erro ############

  if (max(abs(Mf)) >= e ) { # Teste do maior valor da matriz F com base no erro

    Ms <- solve(Mj,-Mf)  #  Montando a matriz com suas soluções. Criando um sistema
                         #  da forma (Mj)S = -Mf
    Mx1 <-  Mx + Ms   #  Achando o novo valor para as váriaveis

  }
  else {
    row.names(Mx) <- name
    return(print(Mx))
  }
  não <- 1
  if (max(abs(Mx1-Mx)) >= e ) {  # Teste da maior diferença entre X1 e Xo

    while (não <= 500 & ((max(abs(Mx1-Mx)) >= e) | (max(abs(Mf)) >= e) )) {
                                             # Laço para m interações do Teste
      não <- não +1                          # para F e a diferença
      Mx <- Mx1
      cnt = 1

      for (v1 in name) {
        assign(v1,Mx[cnt])  # Atribuindo novos valores para as variáveis do Xo
        cnt = cnt + 1

      }
      for (k in 1:n) {     # Atribuindo novos valores para as matrizes F (Mf),
                           #  Jacobino (Mj) e S com base no novo valor de Xo
```

```
        jacob <- deriv(expressions[[k]],name)
        fx <- eval(jacob)

        Mf[k] <- fx[1]
        j <- attributes(eval(jacob))

        Mj[k,]<-j$gradient

      }
      Ms <- solve(Mj,-Mf)
      Mx1 <-  Mx + Ms

    }
  }
  else {
    row.names(Mx1) <- name
    return(print(Mx1))
  }
  if (max(abs(Mx1-Mx)) <= e ) {
    row.names(Mx1) <- name
    return(print(Mx1))
  }
  else if (max(abs(Mf)) <= e) {
    row.names(Mx) <- name
    return(print(Mx))
  }
  else {
   return(print("Não existe solução"))
  }
}
```

Exemplos:

Exemplo 1:

$$H(x,y) = \begin{cases} x^2 + y^2 - 16 = 0 \\ x^2 + (y-4)^2 - 1 = 0 \end{cases} \quad , \epsilon_1 = \epsilon_2 = 10^{-7}, x^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

```
{
  h1 = expression(x^2 + y^2 - 16)
  h2 = expression((x)^2 + (y-4)^2 - 1 )
sistema <- list(h1,h2)

MtNewton(sistema,c("x","y"),2,10^-7,c(1,1))}
```

```
##         [,1]
## x 0.9921567
## y 3.8750000
```

Exemplo 2:

$$F(x,y,z) = \begin{cases} xy^2 + 2z - 20 = 0 \\ log(x^3) - 4 = 0 \\ 3zx + e^z + 3 = 0 \end{cases} \quad , \epsilon_1 = \epsilon_2 = 10^{-4}, x^{(0)} = \begin{bmatrix} 2 \\ 3 \\ 3 \end{bmatrix}$$

```r
{
  f1 = expression(x*y^2 + 2*z - 20)
  f2 = expression(log(x^3) -4)
  f3 = expression(z*x*3+exp(z)+3)
sistema = list(f1,f2,f3)

MtNewton(sistema,c("x","y","z"),3, 10^-4, c(2,3,3))
}
```

```
##          [,1]
## x  3.7936679
## y  2.3333052
## z -0.3269583
```

Exemplo 3:

$$G(x,y,z,t) = \begin{cases} t - xlog(x^2) + 25 = 0 \\ xz + (t-4)^{-2} - 3 = 0 \\ x^2 + y^{-3} - 7 = 0 \\ zt^{-2} + y^{e^y} - 4 = 0 \end{cases} , \epsilon_1 = \epsilon_2 = 10^{-6}, x^{(0)} = \begin{bmatrix} 3 \\ 2 \\ 3 \\ 2 \end{bmatrix}$$

```r
{
  g1 = expression(t - x*log(x^2) + 25)
  g2 = expression(x*z +(t-4)^-2 - 3)
  g3 = expression(x^2 + y^-3 - 7)
  g4 = expression(z*(t^-2) + y^exp(y) - 4)
sistema <- list(g1,g2,g3,g4)

MtNewton(sistema,c("x","y","z","t"),4,10^-6,c(3,2,3,2))
}
```

```
##          [,1]
## x   2.576703
## y   1.404940
## z   1.163611
## t -20.122245
```

Exemplo 4:

$$L(x,y,z) = \begin{cases} x^2 + y^2 + z^2 - 16 = 0 \\ z + x - 6 = 0 \\ y - z - 5 = 0 \end{cases} , \epsilon_1 = \epsilon_2 = 10^{-5}, x^{(0)} = \begin{bmatrix} 1 \\ 1 \\ 4 \end{bmatrix}$$

```r
{
  l1 = expression(x^2 + y^2 + z^2 - 16)
  l2 = expression( z + x - 6)
  l3 = expression( y - z - 5)
sistema <- list(l1,l2,l3)

MtNewton(sistema,c("x","y","z"),3,10^-5,c(1,1,4))
}
```

```
## [1] "Não existe solução"
```