INTRODUÇÃO À COMPUTAÇÃO CIENTÍFICA

IIIº Trabalho: Solução de sistemas não lineares

Tobias J. D. E. Rosa

2022

1º Período

# 

# Resumo

O presente trabalho tem como objetivo encontrar as soluções dos sistemas não lineares de Rosenbrock e tridiagonal de Broyden pelos métodos de Newton, quase-Newton de Broyden e Newton discreto. Para encontrar as respectivas soluções foi desenvolvido um programa em linguagem fortran.

# Solução do Problema

**Método de Newton:** O método de Newton para solução de sistemas não lineares tem como principais obstáculos a solução de um sistema linear a cada iteração e o cálculo da matriz jacobiana antes do início do processo iterativo.

A estratégia para a criação de um programa que resolve sistemas não lineares pelo método de Newton segue as seguintes etapas:

1. As entradas são um vetor de funções e um vetor para chute inicial;
2. Cálculo da matriz Jacobiana de forma manual;
3. Processo iterativo ():

* Aplica-se os valores iniciais na matriz Jacobiana ;
* Aplica-se os valores iniciais nas funções ;
* Resolve-se o sistema , onde é o passo de Newton;
* Encontra-se o novo vetor solução ;
* Verifica-se se o erro está dentro da tolerância, , sendo o erro máximo permitido.

Uma estratégia particular para a solução do sistema foi criar uma matriz , um sistema linear da seguinte forma:

Para a solução desse sistema será utilizado o método de Gauss Jacobi. Neste caso, deve ser dada a devida atenção para que os elementos da diagonal principal não sejam zero. No exercício proposto bastou a permutação das linhas e o sistema foi resolvido.

Na etapa do critério de parada, foi criado um vetor que armazena o vetor solução da iteração anterior para que possa ser verificado o erro através do módulo da maior diferença entre os elementos dos vetores das duas últimas iterações, ou entre o vetor do chute inicial e a primeira iteração.

**Método de Broyden:** O método de Broyden é um dos métodos de quase-Newton. Esses métodos se diferenciam principalmente pela eliminação do cálculo da matriz Jacobiana, que é substituída por uma matriz estimada .

Onde é a solução do sistema linear e ) é o vetor estimado inicial aplicado no vetor de funções.

Entre os métodos de quase-Newton, a particularidade do método de Broyden é a forma que é definida a matriz para a iteração seguinte.

Onde é dado por:

A estratégia para solução de sistema linear utilizada para resolver o método de Broyden foi similar à utilizada no método de Newton, em que foi criada uma matriz que recebe valores da matriz e na última coluna recebe os valores de . A solução do sistema linear foi pelo método de Gauss Jacobi que retorna o vetor solução .

**Método de Newton discreto:** O método de Newton discreto basicamente faz a substituição do cálculo da derivada na matriz jacobiana pelo uso de uma aproximação para a derivada por uma função muito próxima da derivada pela definição.

em que

**Tabela 1** - Resultados obtidos para o sistema de Rosenbrock.

| **MÉTODO** | **VETOR SOLUÇÃO** | **NÚMERO DE ITERAÇÕES** |
| --- | --- | --- |
| Método de Newton | x=(1.000,1.000) | 10 |
| Método de Broyden | x=(1.000,1.000) | 10 |

**Tabela 1** - Resultados obtidos para o sistema tridiagonal de Broyden.

| **MÉTODO** | **VETOR SOLUÇÃO** | **NÚMERO DE ITERAÇÕES** |
| --- | --- | --- |
| Método de Newton | x=( -0.570722103  -0.681806922  -0.702210069  -0.705510616  -0.704906225  -0.701496661  -0.691889346  -0.665796518  -0.596035123  -0.416412264 ) | 10 |

# Apresentação e Discussão de Resultados

Uma dificuldade encontrada no método de Newton foi utilizar funções que retornassem uma matriz jacobiana a cada iteração, a princípio haviam sido criadas 10 funções que eram chamadas separadamente. Posteriormente com o uso de uma interface foi possível compactar o código.

Outro problema foi no método de Broyden, em que a multiplicação do vetor pelo vetor transposto apresentava erro ao utilizar a função “transpose”. Posteriormente com a recomendação do professor, foi solucionado utilizando um comando de repetição que multiplica elemento por elemento.

Os resultados obtidos para o sistema de Rosenbrock, tanto pelo método de Broyden quanto para o método de Newton foram idênticos. O sistema da tridiagonal de broyden foi solucionado apenas pelo método de Newton e teve solução similar à apresentada por Ruggiero (2008). Os próximos passos seriam trabalhar em cima do programa do método de broyden utilizado para Rosenbrock de forma que resolvesse também o sistema da tridiagonal de broyden e na sequência implementar o método de Newton discreto a partir de modificação no programa utilizado para o método de Newton já validado.

**Conclusões**

Os métodos de solução de sistemas não lineares podem ser escolhidos conforme a particularidade do problema. Nos casos em que as derivadas parciais são complicadas de serem calculadas, o método de Newton tende a ser evitado, e nesse caso é melhor fazer o uso dos métodos de Quase-Newton, como o de Broyden, por exemplo, ou utilizar o método de Newton discreto. Apesar de existir uma complexidade matemática em ambos os métodos, os entraves da solução do problema proposto foi maior com relação à programação e contribuiu de forma significativa para o aprendizado.

# Referências

[RUGGIERO, Marcia A. Gomes; LOPES, Vera Lucia da Rocha. *Cálculo numérico: aspectos teóricos e computacionais*. São Paulo: Pearson Makron Books, 2008.](https://www.zotero.org/google-docs/?0BCUCt)

BROYDEN, Charles G. *A class of methods for solving nonlinear simultaneous equations*. Mathematics of computation, 1965.

CÓDIGOS-FONTE

# newton\_d.f90

!gfortran newton\_d.f90; ./a.out entrada\_newton\_d

!esse programa encontra a solução do sistema e grava em um arquivo de nome “Sol\_Sist\_Rosenbrock\_Met\_Newton.txt”

!Institudo Militar de Engenharia - IME

!Programa utilizado na disciplina de introdução a computação científica

!Aluno: Tobias José Degli Esposte Rosa

program newton

implicit none

interface

function J\_x(x,N) result (M)

real, dimension(:), intent(in) :: x

real, dimension( N,N):: M

end function

end interface

interface

function F\_x(x,N) result (f)

real, dimension(:), intent(in) ::x

real, dimension( SIZE(x) ) :: f

end function

end interface

!=======================================================================

real,external::f1,f2

real::s

real, dimension (:,:), allocatable :: Mat\_sist,JAC

real, dimension (:), allocatable:: x,xx,vetor\_de\_funcoes,passo\_de\_newton,dif

character(20)::argument

integer :: n\_lin, n\_col,i,j,ctd1,ctd2

call get\_command\_argument(1, argument)

open(1,file=trim(argument)//".txt",status='old')

open(2,file="Sol\_Sist\_Rosenbrock\_Met\_Newton.txt",status='unknown')

read(1,\*)n\_lin,n\_col

allocate ( Mat\_sist(n\_lin,n\_col),JAC(n\_lin,n\_lin),x(n\_lin),xx(n\_lin),vetor\_de\_funcoes(n\_lin),passo\_de\_newton(n\_lin),dif(n\_lin))

read(1,\*)x(:)

ctd1=0

! INÍCIO DO PROCESSO ITERATIVO DO MÉTODO DE NEWTON

do while (ctd1<10)

ctd1=ctd1+1

vetor\_de\_funcoes=F\_x(x,n\_lin)

JAC=J\_x(x,n\_lin)

do i=1,n\_lin

Mat\_sist(i,n\_col)=-vetor\_de\_funcoes(i)

do j=1,n\_lin

Mat\_sist(i,j)=JAC(i,j)

end do

end do

! ENCONTRA O PASSO DE NEWTON RESOLVEDO O SISTEMA PELO MÉTODO DE GAUSS JACOBI

ctd2=0

do while (ctd2<10)

do i=1,n\_lin

s=0

do j=1,n\_lin

if (i/=j) then

s=s+Mat\_sist(i,j)\*passo\_de\_newton(j)

end if

end do

xx(i)=(Mat\_sist(i,n\_col)-s)/(Mat\_sist(i,i))

end do

ctd2=ctd2+1

passo\_de\_newton=xx

end do

x=x+passo\_de\_newton

!print\*,"sol sist"

end do

call writevetor(x,n\_lin)

deallocate (Mat\_sist,JAC,x,xx,vetor\_de\_funcoes,passo\_de\_newton)

end program newton

!FUNÇÕES UTILIZADAS

function J\_x(x,N) result (M)

implicit none

real, dimension(:), intent(in) :: x

real, dimension( N,N ):: M !! Define result using input param

integer :: i,j,N

do i=1,N

do j=0,N

if (i==1 .and. j==1) then

M(i,j) = -1

elseif (i==1 .and. j==2) then

M(i,j) = 0

elseif (i==2 .and. j==1) then

M(i,j) =-20\*x(1)

elseif (i==2 .and. j==2) then

M(i,j) = 10

end if

end do

end do

end function

function F\_x(x,N) result (f)

implicit none

real, dimension(:), intent(in) :: x

real, dimension(N) :: f

integer::N

f(1)=-x(1)+1

f(2)=10\*x(2) -10\*x(1)\*\*2

end function

! SUBROUTINE

subroutine writevetor(array, n)

implicit none

real, intent(in) :: array(n)

integer, intent(in) :: n

integer :: i

write(2,\*) "O vetor solução é:"

do i = 1,n

write(2,\*) array(i) !IMPRIME NO ARQUIVO DE SAÍDA

write(\*,\*) array(i) !IMPRIME NA TELA

end do

end subroutine writevetor

# broyden\_d.f90

!gfortran broyden\_d.f90; ./a.out entrada\_broyden\_d

!esse programa encontra a solução do sistema e grava em um arquivo de nome “Sol\_Sist\_Rosenbrock\_Met\_Broyden.txt”

!Institudo Militar de Engenharia - IME

!Programa utilizado na disciplina de introdução a computação científica

!Aluno: Tobias José Degli Esposte Rosa

program broyden

implicit none

!===================INTERFACES======================================

!FUNÇÃO DE VETORES

interface

function F\_x(x) result (v)

real, dimension(:), intent(in) ::x

real, dimension( SIZE(x) ) :: v

end function

end interface

!==================END INTERFACES===================================

real, dimension( :, : ),allocatable :: B,Mat\_sist,us

real, dimension( : ),allocatable ::x0,xx,F,sx,x1,y,u

integer :: i,j,n\_lin,n\_col,ctd1,ctd2

real:: soma

character(20)::argument

call get\_command\_argument(1, argument)

open(1,file=trim(argument)//".txt",status='old')

open(2,file="Sol\_Sist\_Rosenbrock\_Met\_Broyden.txt",status='unknown')

read(1,\*)

read(1,\*)n\_lin,n\_col

allocate(B(n\_lin,n\_col),Mat\_sist(n\_lin,n\_col+1),us(n\_lin,1),y(n\_lin),&

&x0(n\_lin),F(n\_lin),xx(n\_lin),x1(n\_lin),sx(n\_lin))

!------------------------------MATRIZ-----------------------------

read(1,\*)

do i=1,n\_lin

read(1,\*)B(i,:)

end do

!-------------------------------VETOR--------------------------------

read(1,\*)

read(1,\*)x0(:)

!---------------------------INÍCIO PROCESSO ITERATIVO MÉTODO DE BROYDEN ------

ctd1=0

do while (ctd1<10)

F=F\_x(x0) ! CALCULA A F E INSERE NA ÚLTIMA COLUNA DA MATRIZ PARA RESOLVER O SISTEMA LINEAR

do i=1,n\_lin

Mat\_sist(i,n\_lin+1)=-F(i)

do j=1,n\_col

Mat\_sist(i,j)=B(i,j)

end do

end do

!-------------------- SOLUÇÃO DO SISTEMA POR GAUSS JACOBI----------------

ctd2=0

do while (ctd2<10)

do i=1,n\_lin

soma=0

do j=1,n\_lin

if (i/=j) then

soma=soma+Mat\_sist(i,j)\*sx(j)

end if

end do

xx(i)=(Mat\_sist(i,n\_col+1)-soma)/(Mat\_sist(i,i))

end do

ctd2=ctd2+1

sx=xx

end do

!print\*,"sol\_jacobi",sx

!------------------------------------END SISTEMA------------------

x1=x0+sx

y=F\_x(x1)-F\_x(x0)

u=(y-matmul(B, sx))/dot\_product(sx,sx)

do i=1,n\_lin

us(i,j)=u(i)\*sx(i) !MATRIZ MULTIPLICAÇÃO DE u POR S TRANSPOSTO

end do

B=B+us

x0=x1

ctd1=ctd1+1

end do

call writevetor(x0,n\_lin)

deallocate (B,Mat\_sist,us,F,x0,x1,xx,y,u,sx)

end program broyden

!==========================================FUNÇÕES==================

function F\_x(x) result (f)

implicit none

real, dimension(:), intent(in) :: x

real, dimension( SIZE(x)) :: f

f(2)=-10\*x(1)\*\*2+10\*x(2)

f(1)=1-x(1)

end function

subroutine writevetor(array, n)

implicit none

real, intent(in) :: array(n)

integer, intent(in) :: n

integer :: i

write(2,\*) "O vetor solução é:"

do i = 1,n

write(2,\*) array(i) !IMPRIME NO ARQUIVO DE SAÍDA

write(\*,\*) array(i) !IMPRIME NA TELA

end do

end subroutine writevetor

# newton\_e.f90

!gfortran newton\_e.f90; ./a.out entrada\_newton\_e

!esse programa encontra a solução do sistema e grava em um arquivo de nome “Sol\_Tridag\_Broyden\_Met\_Newton.txt”

!Institudo Militar de Engenharia - IME

!Programa utilizado na disciplina de introdução a computação científica

!Aluno: Tobias José Degli Esposte Rosa

program newton

implicit none

interface

function J\_x(x,N) result (M)

real, dimension(:), intent(in) :: x

real, dimension( N,N ):: M

end function

end interface

interface

function F\_x(x,N) result (f)

real, dimension(:), intent(in) ::x

real, dimension( N ) :: f

end function

end interface

!=======================================================================

real::s

real, dimension (:,:), allocatable :: Mat\_sist,JAC

real, dimension (:), allocatable:: x,xx,vetor\_de\_funcoes,passo\_de\_newton

character(20)::argument

integer :: n\_lin, n\_col,i,j,ctd1,ctd2

call get\_command\_argument(1, argument)

open(1,file=trim(argument)//".txt",status='old')

open(2,file="Sol\_Tridag\_Broyden\_Met\_Newton.txt",status='unknown')

read(1,\*)n\_lin,n\_col

allocate ( Mat\_sist(n\_lin,n\_col),JAC(n\_lin,n\_lin),x(n\_lin),xx(n\_lin),vetor\_de\_funcoes(n\_lin),passo\_de\_newton(n\_lin))

read(1,\*)x(:)

ctd1=0

! INÍCIO DO PROCESSO ITERATIVO DO MÉTODO DE NEWTON

do while (ctd1<10)

ctd1=ctd1+1

vetor\_de\_funcoes=F\_x(x,n\_lin)

JAC=J\_x(x,n\_lin)

do i=1,n\_lin

Mat\_sist(i,n\_col)=-vetor\_de\_funcoes(i)

do j=1,n\_lin

Mat\_sist(i,j)=JAC(i,j)

end do

end do

! ENCONTRA O PASSO DE NEWTON RESOLVEDO O SISTEMA PELO MÉTODO DE GAUSS JACOBI

ctd2=0

do while (ctd2<10)

do i=1,n\_lin

s=0

do j=1,n\_lin

if (i/=j) then

s=s+Mat\_sist(i,j)\*passo\_de\_newton(j)

end if

end do

xx(i)=(Mat\_sist(i,n\_col)-s)/Mat\_sist(i,i)

end do

ctd2=ctd2+1

passo\_de\_newton=xx

end do

x=x+passo\_de\_newton

end do

call writevetor(x,n\_lin)

deallocate (Mat\_sist,JAC,x,xx,vetor\_de\_funcoes,passo\_de\_newton)

end program newton

!-----------------------------FUNÇÕES UTILIZADAS----------------------

!TRIDIAGONAL DE BROYDEN

function F\_x(x,N) result (f)

implicit none

real, dimension(:), intent(in) :: x

real, dimension(N) :: f

integer:: i,N

do i=1,N

if (i==1) then

f(i)=(3-2\*x(i))\*x(i)-2\*x(i+1)+1

elseif (i==10) then

f(i)=(3-2\*x(i))\*x(i)-x(i-1)+1

else

f(i)=(3-2\*x(i))\*x(i)-x(i-1)-2\*x(i+1)+1

end if

end do

end function

!JACOBIANA DA TRIDIAGONAL DE BROYDEN

function J\_x(x,N) result (M)

implicit none

real, dimension(:), intent(in) :: x

real, dimension( N,N ):: M

integer :: i,j,N

do i=1,N

do j=1,N

if (i==j) then

M(i,j) = -4\*x(i)+3

elseif (i-j==1) then

M(i,j) = -1

elseif (j-i==1) then

M(i,j) = -2

else

M(i,j)=0

end if

end do

end do

end function

! SUBROUTINE

subroutine writevetor(array, n)

implicit none

real, intent(in) :: array(n)

integer, intent(in) :: n

integer :: i

write(2,\*) "O vetor solução é:"

do i = 1,n

write(2,\*) array(i) !IMPRIME NO ARQUIVO DE SAÍDA

write(\*,\*) array(i) !IMPRIME NA TELA

end do

end subroutine writevetor