INTRODUÇÃO À COMPUTAÇÃO CIENTÍFICA

IIº 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 $F = (f_1(X_1), f_2(X_2))$ e um vetor $X^{(0)} = (X_1^{(0)}, X_2^{(0)})$ para chute inicial;
2) Cálculo da matriz Jacobiana $J^{(0)}$de forma manual;
3) Processo iterativo ($k = 0, 1, 2..., n$):
    - Aplica-se os valores iniciais na matriz Jacobiana $J^{(k)}(X^{(0)})$;
    - Aplica-se os valores iniciais nas funções $F^{(0)}(X^{(0)})$;
    - Resolve-se o sistema $J^{(0)} \cdot S^{(k)} =- F^{(k)}$, onde $S^{(k)}$ é o passo de Newton;
    - Encontra-se o novo vetor solução $X^{(k+1)} = X^{(k)} + S^{(k)}$;
    - Verifica-se se o erro está dentro da tolerância, $\left|\left|X^{(k+1)} - X^{(K)}\right|\right| < e$, sendo e o erro máximo permitido.

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

$$A_{i,j} = J_{i,j} \cdot S_i - F_i$$

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 $X_{anterior}$ que armazena o vetor solução da iteração anterior $X^{(k-1)}$ 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 B.

$$B^{(k)} \cdot S^{(k)} =- F(x^{(k)})$$

Onde $S^{(k)}$ é a solução do sistema linear e $F(x^{(k)})$ 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 $B^{(k+1)}$ para a iteração seguinte.

$$B^{(k+1)} = B^{(k)} + u^{(k)} \cdot (S^{(k)})^T$$

Onde $u^{(k)}$ é dado por:

$$u^{(k)} = \frac{y^{(k)} - B^{(k)} \cdot S^{(k)}}{(S^{(k)})^T \cdot S^{(k)}}$$

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 B e na última coluna recebe os valores de $F(x^{(k)})$. A solução do sistema linear foi pelo método de Gauss Jacobi que retorna o vetor solução $S^{(x)}$.

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.

$$f'(x) \approx \frac{f(x+h)-f(x)}{h} \text{ em que } h \approx 0$$

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 u pelo vetor s 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.

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"
!Instituto 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"

```
!Instituto 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
```