8 formas de escrever um programa em Fortran

Matheus Roos

2 de julho de 2022

Resumo

A linguagem Fortran nos oferece diversas formas de escrevermos um programa conforme a nossa necessidade. Estes procedimentos poderão se dar na forma externa ou interna, podendo ser um subprograma de função (FUNCTION) ou uma sub-rotina (SUBROUTINE) e estes podendo serem organizados através de um módulo (MODULE). Vamos ver como proceder com estas formas aplicando para o caso do cálculo de uma raiz através do método de Newton.

Introdução

O script mais simples que podemos fazer para encontrar a raiz de uma função através do método de Newton-Raphsode é este abaixo

```
program NewtonProgramMain
   !Seção declaração
   implicit none
   real :: fun, x, h, tol, x0, x1
   integer :: step
   fun(x) = (x*x)-4.0      !definimos a função
   h = 0.001           !h da derivada de fun
   tol = 1.E-06        !tolerância para a busca
   x0 = 3          !chute inicial
   step = 0            !contagem inicial das iterações

   !Seção execução
   do while(abs(fun(x0)) >= tol)
      x1 = x0 - fun(x0) / ( (fun(x0+h) - fun(x0-h)) / (2*h) )      !
    no denominador está a drivada central
      write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x1)
      x0 = x1         !Novo chute baseado na reta tangente
      step = step + 1 !incremento a cada iteração
   end do
end program
```

onde concentramos todo o método dentro do programa principal, este código até faz o trabalho, mas vamos explorar outras formas de escrevê-lo através dos procedimentos externos (função e sub-rotina).

1 Funções

Podemos passar para um procedimento externo tanto a função que será definida pelo usuário como o cálculo da derivada desta função. Assim,

```
program NewtonFunctionExternal
   !Seção declaração
   implicit none
   real :: tol, x0, x1, Fprime
   real, external :: fun   !Devemos declarar aqui como EXTERNAL.
   integer :: step

   tol = 1.E-06
   x0 = 3
   step = 0

   !Seção execução
   do while(abs(fun(x0)) >= tol)
      x1 = x0 - fun(x0) / Fprime(fun, x0)
      write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x1)
      x0 = x1
      step = step + 1
   end do
end program

!Procedimento externo:
!Função a ser definida pelo usuário.
real function fun(x)
   !Seção declaração
   implicit none
   real, intent(in) :: x   !A inteção é de utilizar apenas como
    entrada.

   !Seção de execução/saída
   fun = x*x-4
end function

!Fprime usa fun como uma função externa a ser definida pelo usuário,
!e que está na lista de argumentos (fun, x)
real function Fprime(fun, x)
   !Seção declaração
   implicit none
   real, external :: fun      !E aqui também declaramos como external
   real, intent(in) :: x    !Estes são os inputs!
   real :: h

   !Seção execução
   h = 0.001
   Fprime = (fun(x+h) - fun(x-h)) / (2*h)
end function
```

onde a Function Fun atuará como uma função externa na Function Fprime.

Analogamente, poderíamos ter passado estas funções como um procedimento interno, escrevendo

```
program NewtonFunctionInternal
   !Seção declaração
   implicit none
   real :: tol, x0, x1
   integer :: step

   tol = 1.E-06
   x0 = 3
   step = 0

   !Seção execução
   do while(abs(fun(x0)) >= tol)
      x1 = x0 - fun(x0) / Fprime(x0)
      write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x1)
      x0 = x1
      step = step + 1
   end do

!Procedimento interno:
contains
   real function fun(x)
      implicit none
      real, intent(in) :: x
      fun = x*x-4
   end function

   real function Fprime(x)
      implicit none
      real, intent(in) :: x
      real :: h
      h = 0.001
      Fprime = (fun(x+h) - fun(x-h)) / (2*h)
   end function
end program
```

2 Sub-rotina

Agora nós vamos introduzir uma sub-rotina, onde todo o método estará dentro dela, restando ao programa principal apenas invocá-la através do comando `Call name_subroutine(input, output)`. Como podemos ver:

```
program NewtonSubroutine
   !Seção declaração
   implicit none
   real :: x0, root
   real, external :: fun, Fprime    !Agora entramos com duas funções como argumento da subrotina
   x0 = 3
   !Seção execução
   call newton(fun, Fprime, x0, root)
   write(*,*) 'A raiz é', root
end program

!Função a ser definida pelo usuário.
real function fun(x)
   implicit none
   real, intent(in) :: x
   fun = x*x-4
end function

real function Fprime(fun, x)
   implicit none
   real, external :: fun
   real, intent(in) :: x
   real :: h
   h = 0.001
   Fprime = (fun(x+h) - fun(x-h)) / (2*h)
end function

subroutine newton(fun, Fprime ,guess, result)
   !Seção declaração
   implicit none
   real, external :: fun, Fprime
   real, intent(in) :: guess
   real, intent(out) :: result
   real :: tol, x1, x0
   integer :: step
   tol = 1.E-06
   step = 0
   x0 = guess

   !Seção execução
   do while(abs(fun(x0)) >= tol)
      x1 = x0 - fun(x0) / Fprime(fun, x0)
      write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x1)
      x0 = x1
      step = step + 1
   end do
   result = x0
end subroutine
```

Mas nós também podemos ter procedimentos internos, dentro de um procedimento externo. Segue abaixo a adaptação:

```
program NewtonSubroutineWithFunction
   !Seção declaração
   implicit none
   real :: x0, root
   real, external :: fun   !Agora entramos com duas funções como
    argumento da subrotina
   x0 = 3                  !chute inicial.
   !Seção execução
   call newton(fun, x0, root)
   write(*,*) 'A raiz é', root
end program
!Função a ser definida pelo usuário.
real function fun(x)
   implicit none
   real, intent(in) :: x
   fun = x*x-4
end function
subroutine newton(fun, guess, result)
   !Seção declaração
   implicit none
   real, external :: fun
   real, intent(in) :: guess
   real, intent(out) :: result
   real :: tol, x1, x0
   integer :: step
   tol = 1.E-06
   step = 0
   x0 = guess
   !Seção execução
   do while(abs(fun(x0)) >= tol)
      x1 = x0 - fun(x0) / Fprime(x0)
      write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x1)
      x0 = x1
      step = step + 1
   end do
   result = x0
!Procedimento interno
contains
   real function Fprime(x)
      implicit none
      real, intent(in) :: x
      real :: h
      h = 0.001
      Fprime = (fun(x+h) - fun(x-h)) / (2*h)
   end function
end subroutine
```

Da mesma forma que a gente pode passar as funções como um procedimento interno, podemos fazê-lo com a sub-rotina, como segue

```
program NewtonSubroutineInternal
   !Seção declaração
   implicit none
   real :: guess, root
   guess = 3

   !Seção execução
   call newton(guess, root)
   write(*,*) 'A raiz é', root
!Procedimento interno
contains
   real function fun(x)
      implicit none
      real, intent(in) :: x
      fun = x*x-4
   end function

   real function Fprime(x)
      implicit none
      real, intent(in) :: x
      real :: h
      h = 0.001
      Fprime = (fun(x+h) - fun(x-h)) / (2*h)
   end function

   subroutine newton(x, result)
      implicit none
      real, intent(in) :: x
      real, intent(out) :: result
      real :: tol, x1, x0
      integer :: step
      tol = 1.E-06
      step = 0
      x0 = x

      do while(abs(fun(x0)) >= tol)
         x1 = x0 - fun(x0) / Fprime(x0)
         write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x1
   )
         x0 = x1
         step = step + 1
      end do
      result = x0
   end subroutine
end program
```

Uma grande vantagem do uso da sub-rotina é que podemos escrevê-la num arquivo separado do programa principal e então ao compilar informamos ao gFortran o nome do arquivo que contém o programa principal, bem como o da sub-rotina. O resultado será este:

newtonSubroutineFile1.f90:

```
subroutine newton(fun, guess, result)
   !Seção declaração
   implicit none
   real, external :: fun
   real, intent(in) :: guess
   real, intent(out) :: result
   real :: tol, x1, x0
   integer :: step
   tol = 1.E-06
   step = 0
   x0 = guess

   !Seção execução
   do while(abs(fun(x0)) >= tol)
      x1 = x0 - fun(x0) / Fprime(x0)
      write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x1)
      x0 = x1
      step = step + 1
   end do
   result = x0

!Procedimento interno
contains
   real function Fprime(x)
      implicit none
      real, intent(in) :: x
      real :: h
      h = 0.001
      Fprime = (fun(x+h) - fun(x-h)) / (2*h)
   end function
end subroutine
```

newtonSubroutineFile2.f90:

```
program NewtonSubRoutine2files
   !Seção declaração
   implicit none
   real :: guess, root
   real, external :: fun
   guess = 3

   !Seção execução
   call newton(fun, guess, root)
   write(*,*) 'A raiz é', root
end program

real function fun(x)
   implicit none
   real, intent(in) :: x
   fun = x*x-4
end function
```

E então digitamos no terminal:

```
$ gfortran newtonSubroutineFiles1.f90 newtonSubroutineFile2.f90
```

3 Módulo

Há ainda a possibilidade de acomodarmos a subrotina dentro de um módulo. Então criamos um arquivo que conterá apenas o módulo e incluímos a subrotina através da declaração `conteins`. Já no programa principal, para utilizar o módulo usamos o comando `USE name-module`. Segue abaixo o resultado:

newton08Module1.f90

```
module rootNewton
implicit none
contains
subroutine newton(fun, guess, result)
   !Seção declaração
   implicit none
   real, external :: fun
   real, intent(in) :: guess
   real, intent(out) :: result
   real :: tol, x1, x0
   integer :: step
   tol = 1.E-06
   step = 0
   x0 = guess

   !Seção execução
   do while(abs(fun(x0)) >= tol)
      x1 = x0 - fun(x0) / Fprime(x0)
      write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x1)
      x0 = x1
      step = step + 1
   end do
   result = x0

!Procedimento interno
contains
   real function Fprime(x)
      implicit none
      real, intent(in) :: x
      real :: h
      h = 0.001
      Fprime = (fun(x+h) - fun(x-h)) / (2*h)
   end function
end subroutine
end module
```

newtonSubroutineFile2.f90: newton08Module2.f90

```
program NewtonModule
   !Seção declaração
   use rootNewton
   implicit none
   real :: guess, root
   real, external :: fun
   guess = 3

   !Seção execução
   call newton(fun, guess, root)
   write(*,*) 'A raiz é', root
end program

real function fun(x)
   implicit none
   real, intent(in) :: x
   fun = x*x-4
end function
```

e então compilamos no terminal

```
$ gfortran newton08Module1.f90 newton08Module2.f90
```

Será gerado além é do arquivo `.out`, também um `.mod`,