

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-Raphson é este abaixo

```
1 program NewtonProgramMain
2   !Seção declaração
3   implicit none
4   real :: fun, x, h, tol, x0, x1
5   integer :: step
6   fun(x) = (x*x)-4.0      !definimos a função
7   h = 0.001              !h da derivada de fun
8   tol = 1.E-06           !tolerância para a busca
9   x0 = 3                  !chute inicial
10  step = 0                !contagem inicial das iterações
11
12  !Seção execução
13  do while(abs(fun(x0)) >= tol)
14      x1 = x0 - fun(x0) / ( (fun(x0+h) - fun(x0-h)) / (2*h) )      !
15      !no denominador está a derivada central
16      write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x1)
17      x0 = x1              !Novo chute baseado na reta tangente
18      step = step + 1      !incremento a cada iteração
19  end do
20 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,

```
1 program NewtonFunctionExternal
2   !Seção declaração
3   implicit none
4   real :: tol, x0, x1, Fprime
5   real, external :: fun    !Devemos declarar aqui como EXTERNAL.
6   integer :: step
7
8   tol = 1.E-06
9   x0 = 3
10  step = 0
11
12  !Seção execução
13  do while(abs(fun(x0)) >= tol)
14    x1 = x0 - fun(x0) / Fprime(fun, x0)
15    write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x1)
16    x0 = x1
17    step = step + 1
18  end do
19 end program
20
21 !Procedimento externo:
22 !Função a ser definida pelo usuário.
23 real function fun(x)
24   !Seção declaração
25   implicit none
26   real, intent(in) :: x    !A inteção é de utilizar apenas como
27   entrada.
28
29   !Seção de execução/saída
30   fun = x*x-4
31 end function
32
33 !Fprime usa fun como uma função externa a ser definida pelo usuário,
34 !e que está na lista de argumentos (fun, x)
35 real function Fprime(fun, x)
36   !Seção declaração
37   implicit none
38   real, external :: fun    !E aqui também declaramos como external
39   real, intent(in) :: x    !Estes são os inputs!
40   real :: h
41
42   !Seção execução
43   h = 0.001
44   Fprime = (fun(x+h) - fun(x-h)) / (2*h)
45 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

```
1 program NewtonFunctionInternal
2   !Seção declaração
3   implicit none
4   real :: tol, x0, x1
5   integer :: step
6
7   tol = 1.E-06
8   x0 = 3
9   step = 0
10
11  !Seção execução
12  do while(abs(fun(x0)) >= tol)
13      x1 = x0 - fun(x0) / Fprime(x0)
14      write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x1)
15      x0 = x1
16      step = step + 1
17  end do
18
19  !Procedimento interno:
20  contains
21      real function fun(x)
22          implicit none
23          real, intent(in) :: x
24          fun = x*x-4
25      end function
26
27      real function Fprime(x)
28          implicit none
29          real, intent(in) :: x
30          real :: h
31          h = 0.001
32          Fprime = (fun(x+h) - fun(x-h)) / (2*h)
33      end function
34 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:

```
1 program NewtonSubroutine
2   !Seção declaração
3   implicit none
4   real :: x0, root
5   real, external :: fun, Fprime    !Agora entramos com duas funções
6   x0 = 3                           como argumento da subrotina
7   !Seção execução
8   call newton(fun, Fprime, x0, root)
9   write(*,*) 'A raiz é', root
10 end program
11
12 !Função a ser definida pelo usuário.
13 real function fun(x)
14   implicit none
15   real, intent(in) :: x
16   fun = x*x-4
17 end function
18
19 real function Fprime(fun, x)
20   implicit none
21   real, external :: fun
22   real, intent(in) :: x
23   real :: h
24   h = 0.001
25   Fprime = (fun(x+h) - fun(x-h)) / (2*h)
26 end function
27
28 subroutine newton(fun, Fprime, guess, result)
29   !Seção declaração
30   implicit none
31   real, external :: fun, Fprime
32   real, intent(in) :: guess
33   real, intent(out) :: result
34   real :: tol, x1, x0
35   integer :: step
36   tol = 1.E-06
37   step = 0
38   x0 = guess
39
40   !Seção execução
41   do while(abs(fun(x0)) >= tol)
42     x1 = x0 - fun(x0) / Fprime(fun, x0)
43     write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x1)
44     x0 = x1
45     step = step + 1
46   end do
47   result = x0
48 end subroutine
```

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

```
1 program NewtonSubroutineWithFunction
2   !Seção declaração
3   implicit none
4   real :: x0, root
5   real, external :: fun    !Agora entramos com duas funções como
6                             argumento da subrotina
7   x0 = 3                    !chute inicial.
8   !Seção execução
9   call newton(fun, x0, root)
10  write(*,*) 'A raiz é', root
11 end program
12 !Função a ser definida pelo usuário.
13 real function fun(x)
14   implicit none
15   real, intent(in) :: x
16   fun = x*x-4
17 end function
18 subroutine newton(fun, guess, result)
19   !Seção declaração
20   implicit none
21   real, external :: fun
22   real, intent(in) :: guess
23   real, intent(out) :: result
24   real :: tol, x1, x0
25   integer :: step
26   tol = 1.E-06
27   step = 0
28   x0 = guess
29   !Seção execução
30   do while(abs(fun(x0)) >= tol)
31     x1 = x0 - fun(x0) / Fprime(x0)
32     write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x0)
33     x0 = x1
34     step = step + 1
35   end do
36   result = x0
37 !Procedimento interno
38 contains
39   real function Fprime(x)
40     implicit none
41     real, intent(in) :: x
42     real :: h
43     h = 0.001
44     Fprime = (fun(x+h) - fun(x-h)) / (2*h)
45   end function
46 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

```
1 program NewtonSubroutineInternal
2   !Seção declaração
3   implicit none
4   real :: guess, root
5   guess = 3
6
7   !Seção execução
8   call newton(guess, root)
9   write(*,*) 'A raiz é', root
10  !Procedimento interno
11  contains
12    real function fun(x)
13      implicit none
14      real, intent(in) :: x
15      fun = x*x-4
16    end function
17
18    real function Fprime(x)
19      implicit none
20      real, intent(in) :: x
21      real :: h
22      h = 0.001
23      Fprime = (fun(x+h) - fun(x-h)) / (2*h)
24    end function
25
26    subroutine newton(x, result)
27      implicit none
28      real, intent(in) :: x
29      real, intent(out) :: result
30      real :: tol, x1, x0
31      integer :: step
32      tol = 1.E-06
33      step = 0
34      x0 = x
35
36      do while(abs(fun(x0)) >= tol)
37        x1 = x0 - fun(x0) / Fprime(x0)
38        write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x0)
39      )
40      x0 = x1
41      step = step + 1
42    end do
43    result = x0
44  end subroutine
45 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:

```
1 subroutine newton(fun, guess, result)
2   !Seção declaração
3   implicit none
4   real, external :: fun
5   real, intent(in) :: guess
6   real, intent(out) :: result
7   real :: tol, x1, x0
8   integer :: step
9   tol = 1.E-06
10  step = 0
11  x0 = guess
12
13  !Seção execução
14  do while(abs(fun(x0)) >= tol)
15     x1 = x0 - fun(x0) / Fprime(x0)
16     write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x1)
17     x0 = x1
18     step = step + 1
19  end do
20  result = x0
21
22  !Procedimento interno
23  contains
24     real function Fprime(x)
25        implicit none
26        real, intent(in) :: x
27        real :: h
28        h = 0.001
29        Fprime = (fun(x+h) - fun(x-h)) / (2*h)
30    end function
31 end subroutine
```

newtonSubroutineFile2.f90:

```
1 program NewtonSubRoutine2files
2   !Seção declaração
3   implicit none
4   real :: guess, root
5   real, external :: fun
6   guess = 3
7
8   !Seção execução
9   call newton(fun, guess, root)
10  write(*,*) 'A raiz é', root
11 end program
12
13 real function fun(x)
14   implicit none
15   real, intent(in) :: x
16   fun = x*x-4
17 end function
```

E então digitamos no terminal:

```
1 $ 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 `contains`. Já no programa principal, para utilizar o módulo usamos o comando `USE name-module`. Segue abaixo o resultado:

newton08Module1.f90

```
1 module rootNewton
2 implicit none
3 contains
4 subroutine newton(fun, guess, result)
5     !Seção declaração
6     implicit none
7     real, external :: fun
8     real, intent(in) :: guess
9     real, intent(out) :: result
10    real :: tol, x1, x0
11    integer :: step
12    tol = 1.E-06
13    step = 0
14    x0 = guess
15
16    !Seção execução
17    do while(abs(fun(x0)) >= tol)
18        x1 = x0 - fun(x0) / Fprime(x0)
19        write(*,*) 'Iteração=>', step, 'x0=', x0, 'Fun(x0)=', fun(x0)
20        x0 = x1
21        step = step + 1
22    end do
23    result = x0
24
25    !Procedimento interno
26    contains
27    real function Fprime(x)
28        implicit none
29        real, intent(in) :: x
30        real :: h
31        h = 0.001
32        Fprime = (fun(x+h) - fun(x-h)) / (2*h)
33    end function
34 end subroutine
35 end module
```

newtonSubroutineFile2.f90: newton08Module2.f90

```
1 program NewtonModule
2   !Seção declaração
3   use rootNewton
4   implicit none
5   real :: guess, root
6   real, external :: fun
7   guess = 3
8
9   !Seção execução
10  call newton(fun, guess, root)
11  write(*,*) 'A raiz é', root
12 end program
13
14 real function fun(x)
15   implicit none
16   real, intent(in) :: x
17   fun = x*x-4
18 end function
```

e então compilamos no terminal

```
1 $ gfortran newton08Module1.f90 newton08Module2.f90
```

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