Programação estruturada no Fortran 90 - 5

Alexandre Diehl

Departamento de Física – UFPel

Argumentos de Sub-Programas com palavras-chave:

- Usados para os casos em que os Sub-Programas têm vários argumentos.
- Os Sub-Programas são chamados usando como opção

<nome-do-argumento> = valor do argumento

- Não é necessário chamar o Sub-Programa mantendo a ordem dos argumentos.
- Apenas os nomes dos argumentos do Sub-Programa devem ser conhecidos.

Exemplo: Função interna

```
[...]
CONTAINS
  real function AREA(inicio,final,tol)
    real,intent(in) :: inicio, final, tol
    [...]
  end function AREA
```

Como podemos usar esta Função?

Argumentos de Sub-Programas com palavras-chave:

- Usados para os casos em que os Sub-Programas têm várias argumentos.
- Os Sub-Programas são chamados usando como opção

<nome-do-argumento> = valor do argumento

- Não é necessário chamar o Sub-Programa mantendo a ordem dos argumentos.
- Apenas os nomes dos argumentos do Sub-Programa devem ser conhecidos.
- **Depois que uma palavra-chave é utilizada pela primeira vez, todos os argumentos restantes devem ser transferidos usando palavras-chave.**

Exemplo: Função interna

```
[...]
  A = AREA(0.0,100.0,1.0E-5)
  B = AREA(inicio= 0.0,tol= 1.0E-5,final= 100.0)
  C = AREA(0.0,tol= 1.0E-5,final= 100.0)
  VALOR = 100.0
  erro = 1.0E-5
  D = AREA(0.0,tol= erro,final= VALOR)
```

Argumentos de Sub-Programas opcionais:

- Usados quando nem todos os argumentos de um Sub-Programa necessitam ser transferidos durante uma chamada do mesmo.
- Os Sub-Programas com argumentos opcionais são chamados usando o atributo OPTIONAL na declaração do tipo de variáveis:

<tipo>, OPTIONAL :: <lista de variáveis>

Exemplo: Função interna

```
[...]
CONTAINS
  real function AREA(inicio,final,tol)
    real,intent(in), OPTIONAL :: inicio, final, tol
    [...]
  end function AREA
```

Como podemos usar esta Função?

Argumentos de Sub-Programas opcionais:

- Usados quando nem todos os argumentos de um Sub-Programa necessitam ser transferidos durante uma chamada do mesmo.
- Os Sub-Programas com argumentos opcionais são chamados usando o atributo OPTIONAL na declaração do tipo de variáveis:

<tipo>, OPTIONAL :: <lista de variáveis>

- **Um argumento obrigatório (não declarado como opcional) deve aparecer uma vez na lista de argumentos.**

Exemplo: Função interna

```
[...]
  A = AREA(0.0,100.0,1.0E-5)
  B = AREA(inicio = 0.0,final = 100.0,tol = 1.0E-5)
  C = AREA(0.0)
  D = AREA(0.0,tol = 1.0E-5)
```

Matrizes como argumentos de Sub-Programas

- Matrizes podem ser usadas como argumentos mudos de Sub-Programas:

  → para transferir a matriz inteira;

  → para transferir um elemento da matriz.

- Para permitir o uso de matrizes de qualquer tamanho nos Sub-Programas, as matrizes como como argumentos mudos são classificadas como:

  → matrizes ajustáveis (herança do Fortran 77);

  → matrizes de forma assumida;

  → objetos automáticos (não tratados nesta disciplina).

- A alocação dentro dos Sub-Programas será feita na etapa de execução do código, a partir da alocação das matrizes feitas dentro do Programa Principal.

Matrizes como argumentos de Sub-Programas

Matrizes ajustáveis:

- As expressões inteiras que definem os limites em cada dimensão das matrizes mudas devem ser transferidas para a rotina (Função ou Subrotina).
- A transferência é feita usando variáveis inteiras para os limites, junto com as matrizes, nos argumentos da rotina ou através do uso de um bloco COMMON (herança do Fortran 77).
- Para matrizes multi-dimensionais a forma de transferência é a mesma.

```
[...]

  CALL prod(100,A,B,C)
[...]
end

subroutine prod(N,x,y,z)
  implicit none
  real :: x(N),y(N),z(N)
end subroutine prod
```

```
[...]

  CALL multi(A,0,10,0,20,B)
[...]
end

subroutine multi(MAP,K1,L1,K2,L2,TRACO)
  implicit none
  real :: MAP(K1:L1,K2:L2)
  real :: TRACO(L1-K1+1)
end subroutine multi
```

Matrizes como argumentos de Sub-Programas

Matrizes de forma assumida:

- Generalização das **matrizes de tamanho assumido** do Fortran 77, onde o tamanho real da matriz é desconhecido quando a rotina é inicialmente chamada.
- A matriz é dita de forma assumida porque a matriz muda pode assumir a forma (não somente o tamanho) da matriz real usada no argumento da rotina, ou seja, é transferida a forma da matriz, não seus limites.
- A sintaxe é feita usando a especificação DIMENSION, onde cada dimensão da matriz é especificada como

  [<limite inferior>] :

- Se <limite inferior> é omitido, o valor padrão 1 é usado.
- O limite superior é tomado como desconhecido, podendo assumir qualquer valor.

  Exemplo: Matriz real A, com uma dimensão, começando no elemento A(0) e tendo limite superior desconhecido.

  REAL, DIMENSION(0 :), :: A

Matrizes como argumentos de Sub-Programas

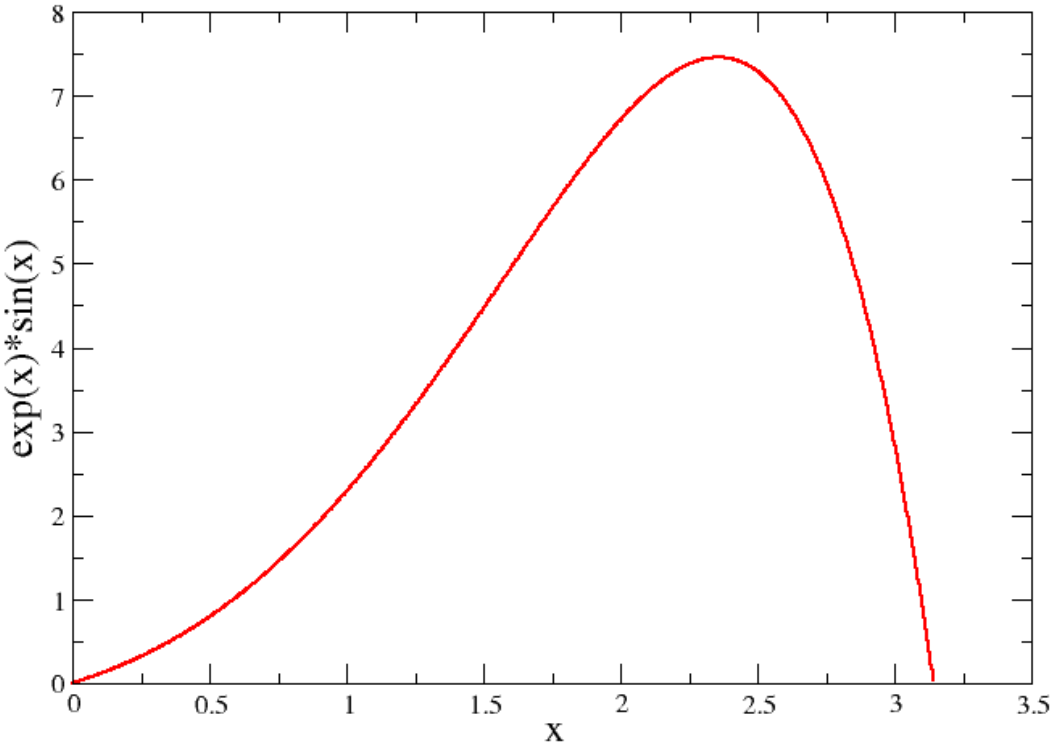
Matrizes de forma assumida:

```
program testa_matriz
  implicit none
  real, dimension(0:10,0:20) :: A
  [...]
  CALL sub_matriz(A)
  [...]

CONTAINS

  subroutine sub_matriz(DA)
    real, dimension(0:,0:), intent(inout) :: DA
    [...]
  end subroutine sub_matriz

end program testa_matriz
```

Sub-Programas como argumentos de rotinas

- Usado nos casos em que os nomes de rotinas são usados como argumentos mudos de outras rotinas.
- No caso de **rotinas EXTERNAS** (sem CONTAINS), definidas em arquivos .f90 em separado:
  → No padrão do Fortran 77 devemos usar a declaração EXTERNAL.

Exemplo: Cálculo do máximo da função $y = e^x \sin(x)$, entre 0 e π.



Sub-Programas como argumentos de rotinas

Exemplo: Cálculo do máximo da função $y = e^x \sin(x)$, entre 0 e π.

```
subroutine calc_maximo(x,f,FUNC)
  implicit none
  real, intent(in) :: x
  real, intent(out) :: f
  real :: FUNC

  f = FUNC(x)

end subroutine calc_maximo
```

```
real function FUNC(x)
  real, intent(in) :: x

  FUNC = exp(x)*sin(x)

end function FUNC
```

```
program maximo
   [...]
   real, EXTERNAL :: FUNC
   [...]
end program maximo
```

- A declaração EXTERNAL não pode ser omitida no Programa Principal.
- Cada unidade de programa deve ser escrita num arquivo .f90 em separado.
- Nenhum mecanismo de controle existe para auxiliar o compilador e o linkador na tarefa de determinar a coerência entre as diferentes unidades de programa.

Sub-Programas como argumentos de rotinas

Exemplo: Cálculo do máximo da função $y = e^x \sin(x)$, entre 0 e π.

```
program maximo
  implicit none
  real :: a, b, x, f
  real :: x_maximo, f_maximo
  real, EXTERNAL :: FUNC

  print*," limites para a funcao?" ;     read*,a,b
  x = a ;     f_maximo = TINY(f_maximo)
  do
     if (x > b) exit
     call calc_maximo(x,f,FUNC)
     if (f >= f_maximo) then
        f_maximo = f ;          x_maximo = x
     end if
     x = x + 0.01
  end do
  print*," x onde y e maximo = ",x_maximo
  print*," maximo de y entre a e b = ",f_maximo

end program maximo
```