

Introdução à Física Computacional

Introdução à programação: tutorial

Esse tutorial apresenta comandos básicos do Linux e do Fortran. Ele não é de forma alguma exaustivo, mas serve aos propósitos básicos do curso. Sugerimos fortemente que consulte os tutoriais de comandos na página do LEF (<http://www.lef.ifsc.usp.br/index.php/computacional>). Caso deseje aprofundar seus conhecimentos em Linux e Fortran, existem inúmeros manuais disponíveis na internet.

I. COMANDOS BÁSICOS DO LINUX

Todos os comandos do Linux podem ser rodados diretamente no terminal. Apresentaremos aqui apenas comandos básicos a serem utilizados ao longo do curso.

- `mkdir` – esse comando cria uma nova pasta dentro do diretório atual. Por exemplo, `mkdir fiscomp` cria uma pasta chamada `fiscomp`;
- `ls` – lista os arquivos e pastas no seu diretório atual;
- `pwd` – comando que indica qual é o seu diretório atual;
- `cd` – comando utilizado para entrarmos em um diretório. Por exemplo `cd fiscomp` acessa o diretório `fiscomp`. Se você digitar `cd ..` você vai para o diretório imediatamente acima.
- `↑` – Ao teclar a seta para cima você acessa os comandos de linha previamente utilizados.

II. ESCRREVENDO SEU CÓDIGO FORTRAN

O próximo passo é escrever o código Fortran. Nesse curso, utilizaremos o editor de texto `gedit`.¹ Para acessá-lo, basta digitar `gedit &` no terminal. O símbolo `&` é importante aqui para que você possa continuar a trabalhar na mesma janela do terminal (caso contrário, precisará abrir outra janela caso deseje usar o terminal enquanto usa o `gedit`):

- `gedit &` – abre o programa `gedit` no terminal.

Uma vez dentro do `gedit`, podemos agora criar nosso primeiro programa. Nossa sintaxe seguirá aquela do Fortran-77. Como exemplo inicial, crie o arquivo `celsius.f` dentro da pasta `fiscomp`. Esse arquivo conterá um programa que transforma uma dada temperatura em Celsius para Fahrenheit. O programa segue abaixo:

```
1 ! Use ! para escrever comentários no programa
2 program celsius2fahrenheit !Nome do programa. Não é necessário iniciar um programa Fortran com o comando program
3 implicit none !Nenhuma variável declarada implicitamente
4 real*8 tc,tf !Declaração de variáveis reais com dupla precisão. Padrão ao longo do curso.
5 !Temperatura em C e em F
6 write(*,*)"Valor da temperatura em graus Celsius:" !Escreva no terminal.
7 !Tudo que estiver entre aspas sai exatamente como no código
8 read(*,*)tc !Leia o valor de tc do terminal
9 if (tc .ge. -273.15d0) then !Teste condicional
10 tf = 9.0d0 * tc / 5.0d0 + 32.0d0!Transforma C -> para F. A adição do d0 força precisão dupla
11 write(*,*)"Valor da temperatura em graus Farenheit:"
12 write(*,*) tf
13 else !Do contrário, faça
14 write(*,*)"Temperatura inválida abaixo do zero absoluto."!Pode-se usar " ou '
15 endif !Fim da condição
16 STOP
17 end program celsius2fahrenheit !Todo programa termina assim, com o comando end.
18 !Se o comando program foi usado no início, então precisa-se terminar com end program <nome do programa>
```

Comentários:

¹ Você pode usar outro editor como o “vi” que é nativo no Linux e roda diretamente no terminal (vide <http://www.lef.ifsc.usp.br/downloadFile/CartaoRefLinux.pdf>).

- Todo código Fortran se inicia com `program` e termina com `end program`, ou apenas termina com `end`;
- Não é necessário dentear um código Fortran. Mas ele fica mais fácil de ler com essa prática;
- Não há diferença entre letras maiúsculas e minúsculas;
- Ao longo do curso, sempre iremos declarar todas as variáveis explicitamente. Por isso, os códigos começam com `implicit none`;
- Utilizaremos precisão dupla sempre. Assim, todas as variáveis reais serão declaradas como `real*8`;
- As funções `read` e `write` são bem parecidas. Uma lê (*read*) alguma coisa e salva numa variável e a outra escreve (*write*) o conteúdo de uma variável em algum lugar. Esse lugar pode ser um arquivo ou na tela (terminal);
- Em alguns casos, o programa precisa tomar uma decisão baseada em algum resultado. Para isso existe a estrutura de condicionais `if`;

Recomendações:

- Tente sempre usar a lógica mais efetiva possível, i.e., aquela que utilize o menor número de operações e/ou que utilize menos tempo de processamento.
- Tente sempre utilizar menos memória.
- Seja eficiente na declaração de variáveis. Utilize a sintaxe do FORTRAN de que variáveis iniciando com (i,j,k,l,m,n) são inteiras e as demais reais.
- Sempre que possível, utilize `!` para comentar seu programa de forma razoável e sem exageros. Isso é uma boa prática de programação.
- Evite comandos “if” desnecessários.
- Em variáveis ou constantes reais, utilize a terminação correta `eX`, `dX`, ou `qX` ao final, onde `X` é o expoente correspondente. Por exemplo, `1.d0`, `0.23e0` ou `2.3e-1`;

Uma vez que terminarmos o código, temos agora que o compilar. Ao contrário de, por exemplo, Python não podemos simplesmente escrever o código e rodar. Em Fortran, temos que compilar o código antes. Isso significa rodar um programa (compilador) que cria um executável. Em nosso exemplo acima fazemos, devemos abrir o terminal no qual o programa `celsius.f` está localizado e digitar a seguinte linha de comando

```
gfortran -O celsius.f -o celsius.exe
```

Cada coisa depois do `gfortran` (compilador) significa algo:

- `-O` – faz com que o compilador otimize o código gerado, para ficar mais rápido. Observação: essa é a letra `O` maiúscula e não o número zero. Maiores detalhes sobre a otimização podem ser encontrados em <https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>;
- `celsius.f` – o nome do código fonte a ser compilado;
- `-o celsius.exe` – gera o programa executável `celsius.exe`.

Para finalmente executar, ou rodar, o código, digite no terminal

```
./celsius.exe
```

Temos assim a estrutura básica de um código Fortran. Vamos agora discutir mais exemplos de códigos Fortran.

A. Laços

Quando queremos executar uma ação repetida vezes, utilizamos uma estrutura de *loop* ou laço. Em Fortran, isso é feito pelo comando **do**:

```
do i = início, fim, passo
    comandos a serem executados
enddo
```

Aqui, i é um inteiro que é aumentado de passo em passo enquanto i for menor ou igual a fim. Caso o passo seja omitido (do $i =$ início, fim) subentende-se que o passo é igual a 1. Uma outra opção comum de laço é obtida se utilizamos uma condicional onde executamos uma dada operação até que uma certa condição de saída do laço seja alcançada. Isso é implementado por meio do comando **do while**:

```
do while (condição)
    comandos a serem executados
enddo
```

Para ilustrarmos a aplicação de um laço, calcularemos a série de Taylor da função exponencial, ao redor de $x_0 = 0$, em um ponto x qualquer

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

O program é esse que se segue

```
1  implicit none !Nenhuma variável declarada implicitamente
2  integer i,n !Declaração de variáveis inteiras
3  real*8 x,esum,ei,fac !Declaração de variáveis reais com dupla precisão
4  open(10,file='exp_series.in')!Abrimos um arquivo existente com as entradas para esse programa
5  read(10,*)x,n !Lemos o valor de x e a ordem da expansão
6  close(10) !Fechamos o arquivo de entrada
7  open(20,file='exp_series.dat') !Abrimos o arquivo de saída
8  esum = 1.0d0 !Iniciamos a soma com o seu valor para o termo de ordem zero !
9  fac = 1.0d0 !Valor inicial do fatorial
10 do i=1,n !Inicio laço
11     fac = fac*real(i,8) !real(i,8) transforma a variável inteira i em real*8
12     ei = (x**i)/fac !termo da série de Taylor correspondente. Dá a precisão da série!
13     esum = esum + ei !fazemos a soma de fato
14     write(20,*)i,esum,ei !Escreve a saída em um arquivo
15 enddo !Fim do laço
16 write(20,*)n+1,exp(x) !Valor exato
17 close(20) !Feche o arquivo de saída
18 end program expserie
```

Alguns comentários sobre o código:

- Lemos e escrevemos as variáveis em arquivos. Note que o arquivo de entrada já deve existir antes de rodarmos o programa. O arquivo de saída é gerado durante a execução. Associamos a cada um dos arquivos um número inteiro (10 e 20 nesse exemplo). Nunca utilize os números 5 e 6 para trabalhar com arquivos, pois eles são especiais: 5 é o teclado e 6 é a tela (do terminal);
- O ***** nos comandos para ler e escrever quer dizer que utilizaremos uma formatação livre;
- Nesse programa, utilizamos variáveis inteiras e reais de precisão duplas. Elas devem ser declaradas separadamente;
- Por uma questão de precisão numérica, calculamos o fatorial como um número real. Para transformarmos um inteiro **int** em um número real de precisão dupla utilizamos a função **real(int,8)**. Aqui o **8** quer dizer precisão dupla. Para precisão simples (quádrupla) utilizamos **4 (16)**;

- Podemos comparar o valor da série com o valor exato. Para isso utilizamos a função intrínseca do Fortran **EXP**.

Um exemplo similar é o dado pelo cálculo da soma da seguinte progressão geométrica (PG)

$$1 + x + x^2 + x^3 + \dots = \frac{1}{1-x}, \quad |x| < 1. \quad (1)$$

Claramente, podemos entender a PG como a série de Taylor da função $1/(1-x)$, expandida ao redor da origem. Para esse programa, utilizaremos a condicional **do while** para verificar a convergência da série até a precisão desejada

```

      program pg
      !
      integer i,n,nmax
      real*8 x,eps,xi,xsum
      !
      open(10,file='pg.in')
      !
      read(10,*)x,eps,nmax
      !
      close(10)
      !
      xsum=0.0d0
      xi=1.0d0
      i=0
      !
      open(20,file='pg.dat')
      !
      do while ((abs(xi).gt.eps).and.(i <= nmax))
         xi=x**i
         xsum=xsum + xi
         i=i+1
         write(20,*)i,xsum,xi
      enddo
      !
      write(20,*)xsum,1.0d0/(1.0d0-x)
      !
      close (20)
      !
      end program

```

Alguns comentários sobre o código:

- Diferentemente do **do**, aqui temos que incrementar o valor do contador i manualmente;
- Temos que tomar cuidado com os valores de entrada das variáveis contidas no **do while**. Tenha certeza de o laço será executado;
- Veja que enquanto o i -ésimo termo da série, x^i , possuir um módulo menor que a tolerância **eps** continuamos a executar o laço. A outra opção de saída é se somarmos um número de termos maior que um número máximo preestabelecido, **nmax**. Note que nesse último caso a tolerância pode não ter sido ainda alcançada. Essa segunda opção é importante para evitar que o código fique rodando indefinidamente no caso de alguma patologia;
- Como no caso da exponencial, podemos verificar nossa resposta comparando-a diretamente com a resposta exata.

B. Matrizes, vetores e sub-rotinas

O Fortran também apresenta uma maneira conveniente para trabalharmos com vetores e matrizes. Sua definição ocorre da seguinte forma

```
integer n
parameter (n=4)
real*8 mat(n,n), vec(n)
```

Alternativamente, pode-se usar

```
integer,parameter :: n=4
real*8, dimension (n,n) :: mat
real*8, dimension (n) :: vec
```

Definimos o inteiro n , dimensão da matriz, como um parâmetro. Isso é conveniente, pois para mudarmos a dimensão da matriz basta mudarmos o valor do parâmetro n e recompilarmos o código. Veja também que a definição de matrizes e vetores é direta: $\text{mat}(n, n)$ é uma matriz $n \times n$ e $\text{vec}(n)$ é um vetor n -dimensional. Como exemplo do uso de matrizes e vetores, considere a operação na qual multiplicamos uma matriz \mathbf{M} , $n \times n$, por um vetor n -dimensional \vec{x} :

$$\vec{y} = \mathbf{M}\vec{x},$$

que em termos de componentes fica

$$y_i = \sum_{j=1}^n M_{ij}x_j.$$

Essa equação pode ser diretamente implementada no Fortran como

```
!
! Loop que faz a multiplicação de uma matriz por um vetor
!
do i=1,n
  msum=0.0d0
  do j=1,n
    msum=msum + mat(i,j)*x(j)
  enddo
  y(i)=msum
enddo
```

Temos assim um algoritmo muito simples para realizar essa multiplicação. Como essa é uma operação muito importante e que queremos realizar várias vezes, pode ser conveniente definirmos uma sub-rotina. Sub-rotinas são programas dentro do programa. Geralmente, a utilizamos quando temos que implementar uma tarefa a ser repetida várias vezes durante a execução do código. É uma boa tática lançarmos mão de sub-rotinas na medida em que o número de linhas do código aumenta porque aumentamos a clareza do código diminuimos a chance de erros simples de programação. Sem mais rodeios, mostro o exemplo de um código que realiza a multiplicação de matrizes por meio de uma sub-rotina

