Tutorial Octave 1 - Criação e Indexação de Matrizes

Introdução

Esta unidade curricular utiliza como software de programação o ambiente de computação científica Octave (www.octave.org), um poderoso ambiente de programação que permite facilmente lidar com vectores, matrizes e gráficos. Este software é gratuito, é directamente instalável num sistema Windows ou Linux e possui um manual em formato pdf e online. Recomenda-se que os alunos instalem a versão mais recente disponível.

Consulte o manual do software e faça algumas experiências com a criação de matrizes, indexação dos seus elementos, operações elementares, criação de um gráfico x-y, definição de funções em ficheiros separados, instruções de ciclo e decisão. Apresenta-se a seguir uma série de pequenos tutoriais do ambiente de programação Octave de modo a facilitar essa aprendizagem. Estes conhecimentos serão essenciais para o desenvovimento de programas que implementam métodos de resolução numérica de problemas matemáticos.

O objetivo destes tutoriais é introduzir de um modo rápido e eficiente as principais características (do ponto de vista da Computação Numérica) do ambiente de programação Octave. Estes tutoriais não dispensam a consulta e leitura do manual para descrições mais detalhadas e aprendizagem de novos comandos e potencialidades do software.

O ambiente Octave funciona através de uma linha de comandos, de uma forma interactiva, tal como uma shell ou consola, apresentando uma prompt e esperando que o utilizador digite um comando. Após a digitação do comando, o utilizador deverá premir a tecla de Return (ou Enter) para que o Octave execute o comando. A execução do comando poderá eventualmente gerar a impressão de dados de saída ou de gráficos, após o que a prompt é novamente imprimida na linha seguinte, ficando o Octave à espera de um novo comando. Neste tutorial é assumido como exempo que a prompt é constituída por ">> ". Esta prompt pode ser obtida executando o comando PS1(">> ").

Também é possível agrupar vários comandos num ficheiro ("scripts" ou ficheiros "batch", de extensão .m) , definir novas funções (também em ficheiros .m) e dar o nome do ficheiro como comando.

I - Criação de Matrizes

O principal objecto no ambiente octave é a matriz (ou array bidimensional) de dimensões genéricas m*n, ou seja, m linhas e n colunas. Dado o valor particular de m,n, a matriz pode ser particularizada para um vector coluna m*1, vector linha 1*n, ou escalar 1*1.

Por defeito, o tipo de dados é o de vírgula flutuante de precisão dupla (double), embora em casos específicos seja possível utilizar precisão simples ou inteiros, tipicamente em casos de armazenamento de grandes quantidades de dados, para rentabilizar a memória. No entanto, para processamento, os dados são convertidos para vírgula flutuante de precisão dupla, processados e novamente convertidos para o tipo original para armazenamento.

Para criar uma variável, à partida não é necessário definir nem o seu tipo nem dimensões. Por exemplo, para efectuar a multiplicação de 23 por 47 e guardar o resultado na variável x, ao ser dado o comando,

```
>> x=23*47
x =  1081
>>
```

é criada a variável x de dimensões 1*1, do tipo vírgula flutuante de precisão dupla (apesar de ser um produto de inteiros com resultado inteiro), tudo de forma automática sem que o utilizador tenha esse trabalho adicional. Como resultado da execução do comando "x=23*47", é impresso o actual valor da variável x (que é o valor resultado do produto) e reimpressa a prompt. Se não for desejado que sejam impressos os dados de saída resultantes da execução de um comando, basta terminar o comando com o caracter ';',

```
>> x=23*47;
>>
```

O actual valor de x continua a ser o resultado de 23*47. Para consultar o valor de uma variável, basta dar um comando com apenas o seu nome,

```
>> x
x =  1081
>>
```

Também é possível dar vários comandos de uma vez, separados por ';' para não impressão de dados de saída, ou por ',' com impressão de dados de saída, para os respectivos comandos,

```
>> x=23*47; y=19.20, z=x+y
y =  19.200
z =  1100.2
>>
```

Se os vários comandos não couberem numa linha (ou por qualquer outra razão), pode-se colocar "..." no fim da linha e continuar a lista de comandos na próxima. Na continuação da digitação de comandos numa nova linha, o Octave apresenta a prompt reduzida "> ".

```
>> x=23*47; y=19.20, ...
> z=x+y
y =  19.200
z =  1100.2
>>
```

Para criar uma matriz, as regras de construção são as seguintes: - é utilizado o caracter '[' para dar início à definição da matriz - é utilizado o espaço ' ' ou vírgula ',' para separar elementos de uma mesma linha (ou horizontalmente), como por exemplo 1 2 3 4 5 ou 1,2,3,4,5 - é utilizado o ponto e vírgula ';' para separar linhas (ou verticalmente), como por exemplo 1 2 3 4 5; 6 7 8 9 10 - é utilizado o caracter ']' para terminar a definição da matriz.

Por exemplo, para criar a matriz

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \end{bmatrix}$$

um possível comando seria,

```
>> A=[1 2 3 4 5; 6,7,8,9,10]
A =

    1    2    3    4    5
    6    7    8    9   10

>>
```

Consequentemente, um vector linha ou coluna pode ser criado de forma análoga,

```
>> vl=[1 2 3 4 5], vc=[6;7;8;9;10]
vl =

   1   2   3   4   5

vc =

    6
    7
    8
    9
   10

>>
```

A transposição de uma matriz ou vector é obtida através do operador transposição, representado pelo caracter apóstrofe ' . Assim a matriz transposta de A e o vector transposto de vc podem ser obtidos por,

```
>> A2=A', vl2=vc'
A2 =

    1    6
    2    7
    3    8
    4    9
    5   10

vl2 =

    6    7    8    9   10

>>
```

Matrizes e vectores podem também ser criados através da "concatenação" de sub-matrizes e/ou sub-vectores, desde que tenham dimensões apropriadas. Por exemplo, a matriz A pode neste momento ser criada através da concatenação dos vectores linha vl e vl2 com o comando,

```
>> A=[vl; vl2]
A =

    1    2    3    4    5
    6    7    8    9   10

>>
```

Quando um comando não faz uma atribuição a uma variável, o (último) resultado, seja escalar, vector ou matriz, é guardado sempre na variável "ans", que pode ser usada no comando seguinte,

```
>> vl+vl2
ans =

    7    9   11   13   15

>> a= 2*ans
a =
```

```
   14   18   22   26   30

>>
```

Pela sua frequência de utilização, para algumas matrizes particulares o Octave disponibiliza funções próprias para a sua criação, como é o caso da matriz identidade, de zeros, de uns e de elementos aleatórios com distribuição uniforme (entre 0 e 1) ou gaussiana (normal). Estas funções recebem como argumentos as dimensões desejadas para as matrizes,

```
>> I=eye(3), Z=zeros(2,3), U=ones(3,2), Uni=rand(2,2),
Gau=randn(1,4)
I =

   1   0   0
   0   1   0
   0   0   1

Z =

   0   0   0
   0   0   0

U =

   1   1
   1   1
   1   1

Uni =

   0.80140   0.61215
   0.14824   0.82639

Gau =

  -0.091613   0.034901   2.301904   0.768301

>>
```

Por outro lado, existe também a função size() que permite obter as dimensões de uma matriz,

```
>> size(U),size(Gau)
ans =
```

```
   3   2

ans =

   1   4

>>
```

retornando um vector (linha) de dois elementos, o primeiro indicando o número de linhas e o segundo o número de colunas da matriz argumento. Se à partida se souber tratar-se de um vector, a função length() permite obter o seu "comprimento",

```
>> length(Gau)
ans =  4
>>
```

Outro caso importante é a criação de um vector "linear", ou seja, em que cada elemento é igual ao anterior mais uma constante ou incremento. Este tipo de vectores é criado através do operador ':' que tem a sintaxe

valor inicial : incremento : valor final

Por exemplo, para gerar o vector u=[2 2.1 2.2 2.3 2.4 2.5], pode ser utilizado o comando,

```
>> u= 2:0.1:2.5
u =

   2.0000   2.1000   2.2000   2.3000   2.4000   2.5000

>>
```

Se o valor incremento for omitido, fica por defeito com o valor 1,

```
>> b= 1:10
b =

   1   2   3   4   5   6   7   8   9  10

>>
```

II - Indexação de Matrizes

Os elementos de matrizes e vectores são indexados com os índices separados por vírgulas e colocados entre parentesis curvos a seguir ao nome da variável,

```
>> A=[1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15]; vl2=[6 7 8 9
10];
>> A(1,1), A(2,4), vl2(3)
ans =  1
ans =  9
ans =  8
>>
```

Com o auxílio do operador ':' é possível indexar subvectores e submatrizes dos vectores e matrizes originais. Por exemplo, para obter um vector linha constituído pelos 3 primeiros elementos da segunda linha da matriz A (esses elementos correspondem aos da linha 2, colunas 1, 2 e 3), pode fazer-se,

```
>> A(2,1:3)
ans =

   6   7   8

>>
```

onde se relembra que 1:3 representa ou gera o vector [1 2 3], cujos elementos vão servir de índices para indexar elementos da matriz A. De facto o exemplo anterior é equivalente a A(2,[1 2 3]). Utilizando a mesma estratégia quer para as linhas, quer para as colunas, é possível indexar uma submatriz. Por exemplo, para a submatriz constituída pela gama de elementos correspondentes às linhas 1 a 3 e colunas 3 a 4, vem

```
>> A(1:3,3:4)
ans =

    3    4
    8    9
   13   14

>>
```

No caso especial de uma submatriz incluir todas as linhas ou todas as colunas, a linguagem Octave permite uma forma rápida de o fazer, em que apenas ':' significa todas as linhas ou todas as colunas conforme o caso. O exemplo anterior poderia ser reescrito com idênticos resultados como,

```
>> A(:,3:4)
ans =
```

```
     3     4
     8     9
    13    14

>>
```

uma vez que se estão a indexar todas as linhas. O operador ':' é muito útil quando se indexam linhas ou colunas consecutivas. Se não forem consecutivas, é necessário utilizar vectores explicitos para indexar as linhas e colunas desejadas. Por exemplo, para obter a submatriz correspondente às linhas 1 e 3 e colunas 1, 3, 4 e 5, vem

```
>> A([1 3],[1 3 4 5])
ans =

     1     3     4     5
    11    13    14    15

>>
```

ou

```
>> A([1 3],[1 3:5])
ans =

     1     3     4     5
    11    13    14    15

>>
```

uma vez que 3:5 expande para [3 4 5], obtendo-se [1 [3 4 5]], que por concatenação origina [1 3 4 5]. Para vectores, a filosofia é a mesma, mas apenas se opera com uma dimensão. Para se obter o subvector de vl2 constituído pelos elementos de valores 7,8 e 9, vem

```
>> vl2(2:4)
ans =

    7    8    9

>>
```

Como curiosidade, verifica-se que representam a mesma matriz A e A(:,:), e que representam o mesmo vctor vl2 e vl2(:).

A indexação de elementos pode ocorrer de qualquer dos lados da atribuição. Por exemplo,

```
>> a=A(2,1:4)
a =

   6   7   8   9

>> A(2,1:3)=[50 50 50 50]
A =

    1    2    3    4    5
   50   50   50    9   10
   11   12   13   14   15

>> A(:,1)=[99 99 99]'
A =

   99    2    3    4    5
   99   50   50    9   10
   99   12   13   14   15

>>
```

onde foi utilizado o operador transposição como forma mais pratica de obter o vector coluna [99; 99; 99]. Note-se que mesmo quando se efectua uma atribuição a apenas uma parte da matriz, obtém-se como dados de saída para impressão a matriz inteira.

FIM

Última alteração: Sexta, 20 Setembro 2019, 16:28

Tutorial Octave 2 - Operações com Matrizes

III - Operações com Matrizes

O ambiente Octave permite realizar com simplicidade as operações aritméticas vulgares com matrizes. Há no entanto que ter em atenção que estas operações requerem "compatibilidade" de dimensões entre as matrizes envolvidas para que a operação faça sentido. Por exemplo, não se pode somar matrizes de dimensões diferentes.

As operações disponíveis são: adição ou soma '+', subtracção ou negação '-', multiplicação '*', divisão à direita '/', divisão à esquerda '\', potência '^',

```
>> A=[1 2;0 3], B=[1 1;0 1]
A =

   1   2
   0   3

B =

   1   1
   0   1

>> soma= A+B, subtraccao= A-B, negacao= -A, multiplicacao=
A*B, potencia2= A^2
soma =

   2   3
   0   4

subtraccao =

   0   1
   0   2

negacao =

  -1  -2
  -0  -3

multiplicacao =

   1   3
   0   3
```

```
potencia2 =

   1   8
   0   9

>>
```

As divisões são "equivalentes" à multiplicação pela matriz inversa e só fazem sentido para matrizes quadradas. Assim,

A divisão à direita A/B é equivalente a A*inv(B)

A divisão à esquerda A\B é equivalente a inv(A)*B

onde inv() é a função em Octave que permite obter a inversa de uma matriz. Veja-se o exemplo seguinte, onde no segundo cálculo se obtém a matriz identidade como seria de esperar,

```
>> inversaA=inv(A), inversaA*A, inversaA*B, A\B
inversaA =

   1.00000  -0.66667
   0.00000   0.33333

ans =

   1.00000   0.00000
   0.00000   1.00000

ans =

   1.00000   0.33333
   0.00000   0.33333

ans =

   1.00000   0.33333
   0.00000   0.33333

>>
```

Além das operações aritméticas regulares, também é possível efectuar as chamadas operações aritméticas elemento a elemento. Por exemplo, a soma de matrizes é naturalmente uma operação elemento a elemento, pois de C=A+B, obtém-se que cada elemento da matriz resultado é dado por C(i,j)=A(i,j)+B(i,j).

Para se poder efectuar uma operação elemento a elemento as matrizes intervenientes devem ter as mesmas dimensões. O operador elemento a elemento define-se colocando um ponto '.' antes do operador regular. As operações (extra) disponíveis são: multiplicação '.*', divisão á direita './', divisão à esquerda '.\' e potência '.^',

```
>> D=[2 2; 3 3], A.*A, A.^2, 2.^A, D.\A, A./D
D =

   2   2
   3   3

ans =

   1   4
   0   9

ans =

   1   4
   0   9

ans =

   2   4
   1   8

ans =

   0.50000   1.00000
   0.00000   1.00000

ans =

   0.50000   1.00000
   0.00000   1.00000

>>
```

O Octave disponibiliza um conjunto de funções matemáticas e outras, que também operam numa base de elemento a elemento. São funções em que C=funcao(A) tem como resultado C(i,j)=funcao(A(i,j)), ou seja, operam individualmente em cada elemento da matriz argumento. Apresentam-se a seguir algumas dessas funções disponibilizadas pelo Octave.

cos() coseno
sin() seno

tan() tangente
acos() arco-coseno
asin() arco-seno
atan() arco-tangente
exp() exponencial
log() logaritmo natural ou neperiano
log10() logaritmo base dez
sqrt() raiz quadrada
sign() sinal
round() arredondamento para o inteiro mais próximo
floor() arredondamento para o inteiro mais próximo, no sentido de $-\infty$
ceil() arredondamento para o inteiro mais próximo, no sentido de $+\infty$
fix() arredondamento para o inteiro mais próximo, no sentido de zero
abs() valor absoluto ou módulo de um número complexo
real() parte real de um número complexo
imag() parte imaginária de um número complexo
angle() ângulo de um número complexo

Por exemplo, para se obter os valores da função exponencial $f(x) = e^x$ para x entre 0 e 1 com intervalos de .2, vem,

```
>> x=0:0.2:1, f=exp(x)
x =

   0.00000   0.20000   0.40000   0.60000   0.80000   1.00000

f =

   1.0000   1.2214   1.4918   1.8221   2.2255   2.7183

>>
```

Note-se que o vector (matriz) de saída tem as mesmas dimensões do vector (matriz) de entrada. Para se obter vectores coluna utiliza-se o operador de transposição,

```
>> x=(0:0.2:1)', f=exp(x)
x =

   0.00000
   0.20000
   0.40000
   0.60000
   0.80000
   1.00000
```

```
f =

   1.0000
   1.2214
   1.4918
   1.8221
   2.2255
   2.7183

>>
```

onde foi necessário utilizar parentesis porque 0:0.2:1'=0:0.2:(1'), não se obtendo o vector coluna desejado. Esta questão está relacionada com a precedência de operadores.

Quando o Octave arranca, existem algumas variáveis pré-definidas com valores matemáticos úteis, que no entanto podem posteriormente ser alteradas se desejado ou se esses valores não forem considerados necessários. Algumas dessas variáveis e os valores com que são inicializadas são:

pi=3.1415...
e=2.7183...
i=sqrt(-1)
j=sqrt(-1)
eps= precisão do tipo vírgula flutuante precisão dupla
realmin= menor número real
realmax= maior número real

No Octave pode-se trabalhar com números complexos com a mesma facilidade com que se trabalha com números reais. As variáveis i ou j podem ser utilizadas para definir números complexos, no entanto são as mais vulgarmente alteradas porque tradicionalmente são muito usadas em ciclos. O utilizador é livre de definir outra variável para representar o número complexo puro unitário. Veja-se a sequência seguinte de comandos,

```
>> pi
ans =  3.1416
>> e
ans =  2.7183
>> i
ans =  0 + 1i
>> j
ans =  0 + 1i
>> z=3+2*i
z =  3 + 2i
>> z=3+2*j
```

```
z =  3 + 2i
>> z=3+2*sqrt(-1)
z =  3 + 2i
>> icpx=sqrt(-1)
icpx =  0 + 1i
>> z=3+2*icpx
z =  3 + 2i
>> y=5+5*icpx
y =  5 + 5i
>> w=z+y
w =  8 + 7i
>> eps
ans =  2.2204e-16
>> realmin
ans =  2.2251e-308
>> realmax
ans =  1.7977e+308
>>
```

FIM

Última alteração: Quinta, 19 Setembro 2019, 17:32

Tutorial Octave 3 - Gráficos 2D

A criação de gráficos é de importância fundamental pois permite de uma forma compacta através da visualização mostrar ou realçar características dos dados.

O ambiente Octave permite elaborar uma grande variedade de gráficos desde os bidimensionais (2D) muito simples a gráficos tridimensionais (3D) extremamente complexos construídos a partir de uma hierarquia de objectos.

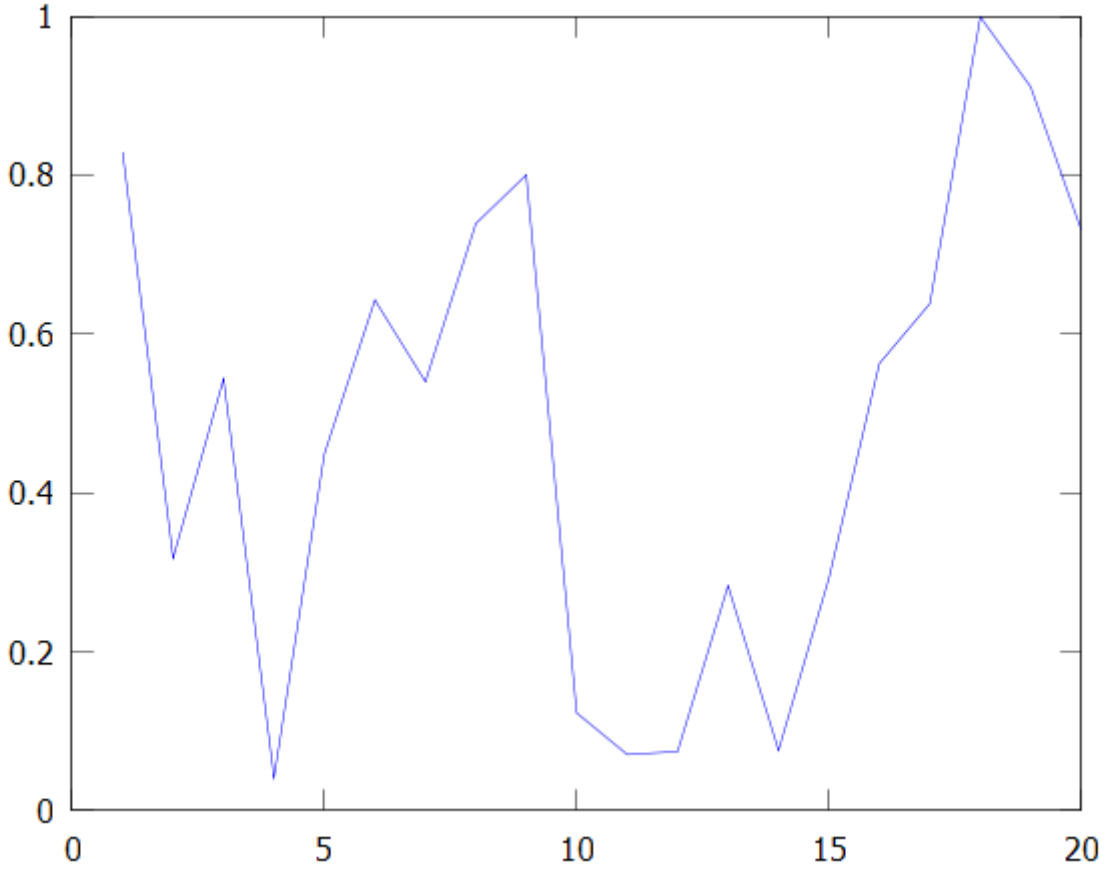
Para efeitos da UC de Computação Numérica, são aqui abordados os comandos utilizados para construir gráficos 2D que permitam visualizar em simultâneo uma ou mais funções reais, com traços e cores diferentes, marcas para assinalar pontos importantes, textos para o título do gráfico e etiquetas dos eixos, grelhas e legendas das várias curvas constituintes do gráfico.

Entende-se por um gráfico 2D um gráfico com dois eixos (duas dimensões), onde o eixo horizontal é designado por eixo das abcissas e vulgarmente atribuído a x ou t (tempo) e o eixo vertical é o eixo das ordenadas, vulgarmente atribuído a y. Dado um conjunto de pontos (x,y) que define uma curva, é possível representá-los num plano x,y e unir os pontos com rectas ou outro método de interpolação e obter um gráfico da curva.

O comando principal para se obter um gráfico de um curva 2D é a função plot(), que na sua forma mais simples plot(y) aceita como único argumento um vector de pontos para as ordenadas y e assume para x o vector x=1:n, onde n é o comprimentos de y. O comando,

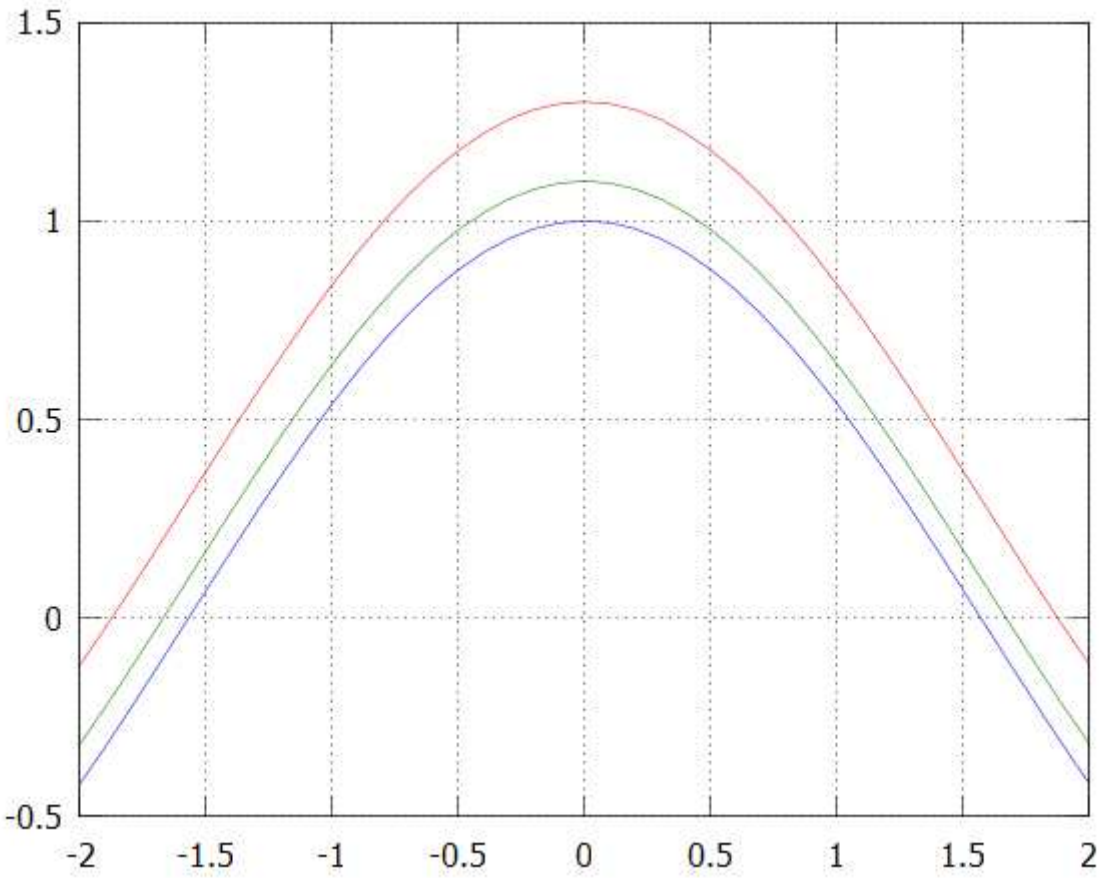
```
>> y=rand(20,1); plot(y);
```

cria um vector coluna y com 20 números aleatórios e cria numa janela separada um gráfico com os pontos por defeito ligados (interpolados) por rectas,



Também é possível especificar os valores de abcissa x e representar várias curvas de uma vez. Para melhor visualização pode-se desenhar uma grelha com o comando grid,

```
>> x=(-2:.1:2)'; y1=cos(x); y2=y1+0.1; y3=y1+0.3;
>> plot(x,[y1 y2 y3]); grid;
```

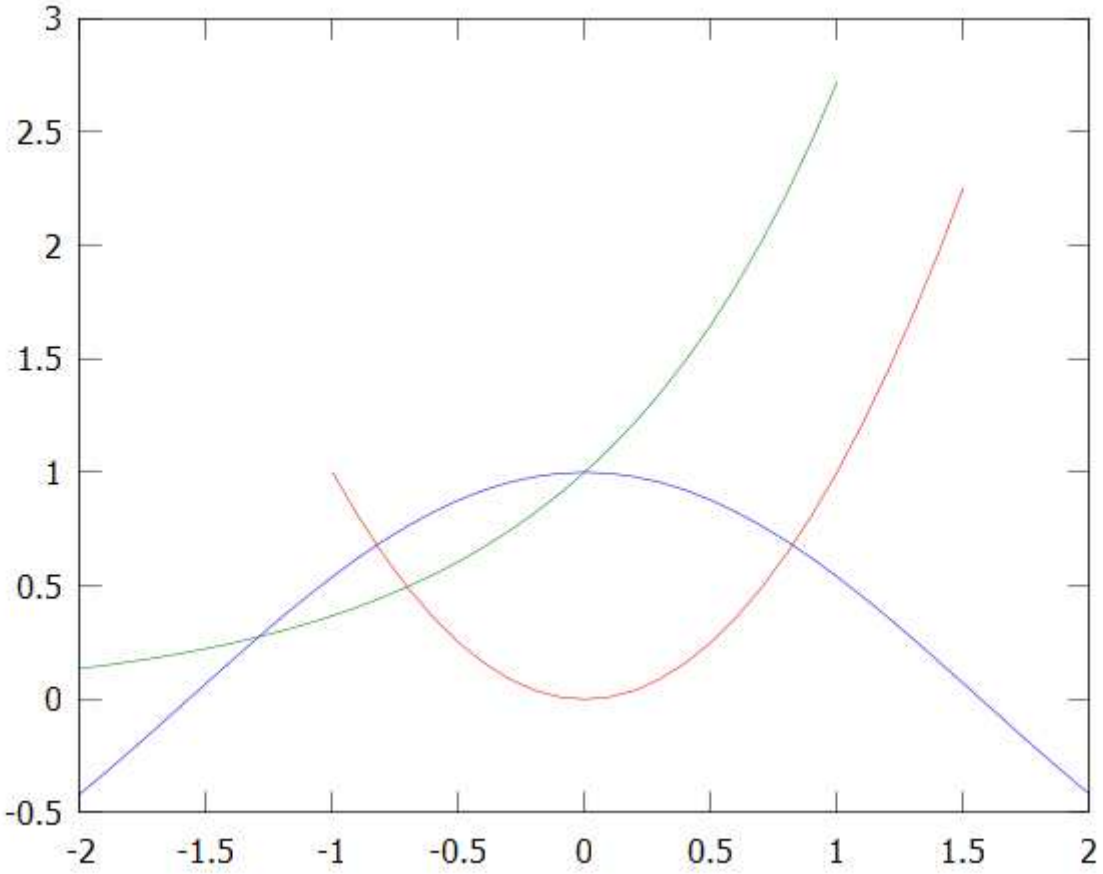


Neste caso os vectores de dados são vectores coluna, mas o mesmo gráfico podia ter sido obtido a partir de vectores linha, conforme a conveniência, com os comandos,

```
>> x=-2:.1:2; y1=cos(x); y2=y1+0.1; y3=y1+0.3;
>> plot(x,[y1; y2; y3]); grid;
```

sendo a função plot() suficientemente "inteligente" para determinar qual é o caso. Se o vector de abcissas x não for o mesmo para cada curva, as curvas do gráfico podem ser especificadas em pares individuais,

```
>> x1=-2:.1:2; y1=cos(x1);
>> x2=-2:.1:1; y2=exp(x2);
>> x3=-1:.1:1.5; y3=x3.^2;
>> plot(x1,y1,x2,y2,x3,y3);
```



É possível através da função plot() especificar certas características e propriedades para os gráficos criados. A função plot() em termos genéricos aceita os seguintes argumentos, pela ordem indicada,

```
plot(x, y, "LCP;legenda;")
```

onde x contem os pontos correspondentes ao eixo das abcissas (eixo horizontal), y contém os pontos correspondentes ao eixo das ordenadas (eixo vertical), a string "LCP;legenda;" constitui uma palavra de controlo onde se pode especificar propriedades das curvas do gráfico, como tipo de linha e cor. Este trio de argumentos pode ser repetido várias vezes para se obterem curvas diferentes com propriedades diferentes. De facto, para cada trio, apenas y é obrigatório, sendo assumido para x os valores dos índices 1,2,3,... e tipo de linha contínua e cores por defeito, como já visto anteriormente.

A palavra de controlo "LCP;legenda;" pode conter até quatro ítens, todos opcionais mas se usados devem sê-lo pela ordem indicada, com o significado que a seguir se descreve:

- L simboliza um caracter que indica o tipo de linha. Só existem duas possibilidades:
L='-' para linha contínua;
L='.' para linha ponteada.

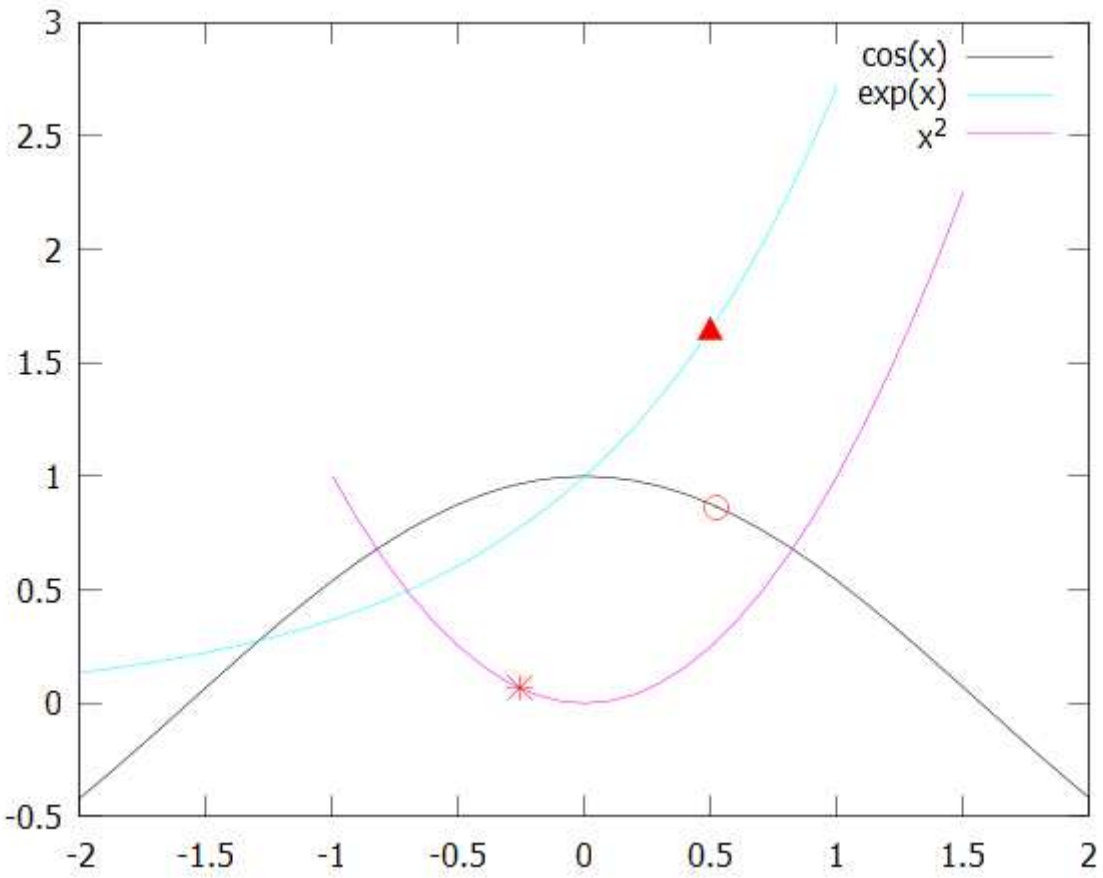
- C simboliza um caracter que indica a cor. As possibilidades são:
C='k' para preto (black);
C='r' para vermelho (red);
C='g' para verde (green);
C='b' para azul (blue);
C='m' para roxo (magenta);
C='c' para azul turquesa (cyan);
C='w' para branco (white);
C='y' para amarelo (yellow).

- P simboliza um caracter que indica um marcador a ser colocado em cada ponto da curva. As possibilidades são:
P='+' para um sinal mais;
P='*' para um asterisco;
P='o' para uma bola (circunferência);
P='x' para uma cruz;
P='^' para um triângulo.

- Em `;legenda;`, legenda simboliza o texto que identifica cada tipo de linha no gráfico, como por exemplo cos(x). Os caracteres ';' são usados apenas como delimitadores.

Assim, se no gráfico anterior se pretendesse usar cores diferentes das de defeito para as curvas, legendar o gráfico e assinalar com marcas alguns pontos notáveis como por exemplo cos(pi/6), exp(0.5) e (-0.26)^2,

```
>> plot(x1,y1,"k;cos(x);", x2,y2,"c;exp(x);", ...
> x3,y3,"m;x^2;", pi/6,cos(pi/6),"ro", ...
> 0.5,exp(0.5),"r^", -0.26,(-0.26)^2,"r*");
```



De facto, ainda é possível especificar mais propriedades para o gráfico, colocando a seguir da string de controlo pares "propriedade",valor, obtendo-se ainda de um modo mais genérico,

```
plot(x, y, "LCP;legenda;", "propriedade", valor)
```

onde entre outras possibilidades, "propriedade" pode ser:

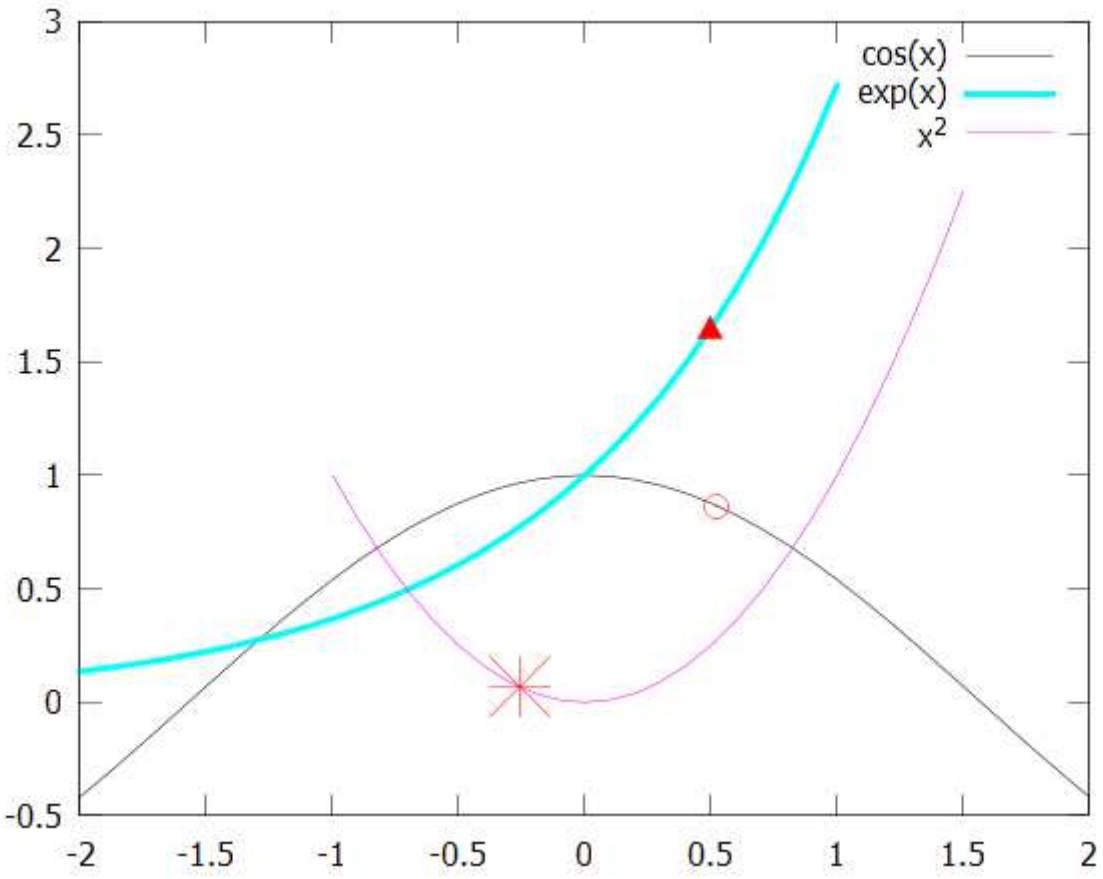
- "`markersize`" para especificar a dimensão do caracter utilizado como marcador;
- "`linewidth`" para especificar a expessura da linha utilizada para as curvas.

e o valor é um inteiro maior ou igual a 1, que actua como um factor de escala proporcional. Por exemplo, valor=2 significa o dobro da dimensão ou expessura de valor=1.

Suponha-mos que no gráfico anterior se pretendia que a curva associada a exp(x) tivesse maior expessura e que o marcador '*' fosse maior,

```
>> plot(x1,y1,"k;cos(x);", ...
> x2,y2,"c;exp(x);", "linewidth",3, ...
```

```
> x3,y3,"m;x^2;", pi/6,cos(pi/6),"ro", ...
> 0.5,exp(0.5),"r^", ...
> -0.26,(-0.26)^2,"r*","markersize",15);
```



Nota: enquanto que para a expessura de linha, valor=3 parece expecificar a expessura em pixeis, já para a dimensão do marcador a melhor estratégia paraece ser tentativa e erro para obter a dimensão desejada, já que valor=1 corresponde a um marcador minúsculo!

Como se viu nos exemplos anteriores, os comandos para criar gráficos podem tornar-se extensos ao ponto de serem necessárias várias linhas para completar um comando. Quando se pretende criar um gráfico a partir de vários comandos, o comando "hold on" permite congelar ou manter o gráfico actual, de modo que os próximos comandos não redesenham o gráfico mas acrescentam-lhe os novos elementos. Uma vez terminado o gráfico, deve executar-se "hold off". O gráfico do exemplo anterior poderia ter sido criado por vários comandos como segue,

```
>> plot(x1,y1,"k;cos(x);");
>> hold on;
>> plot(x2,y2,"c;exp(x);", "linewidth",3);
```

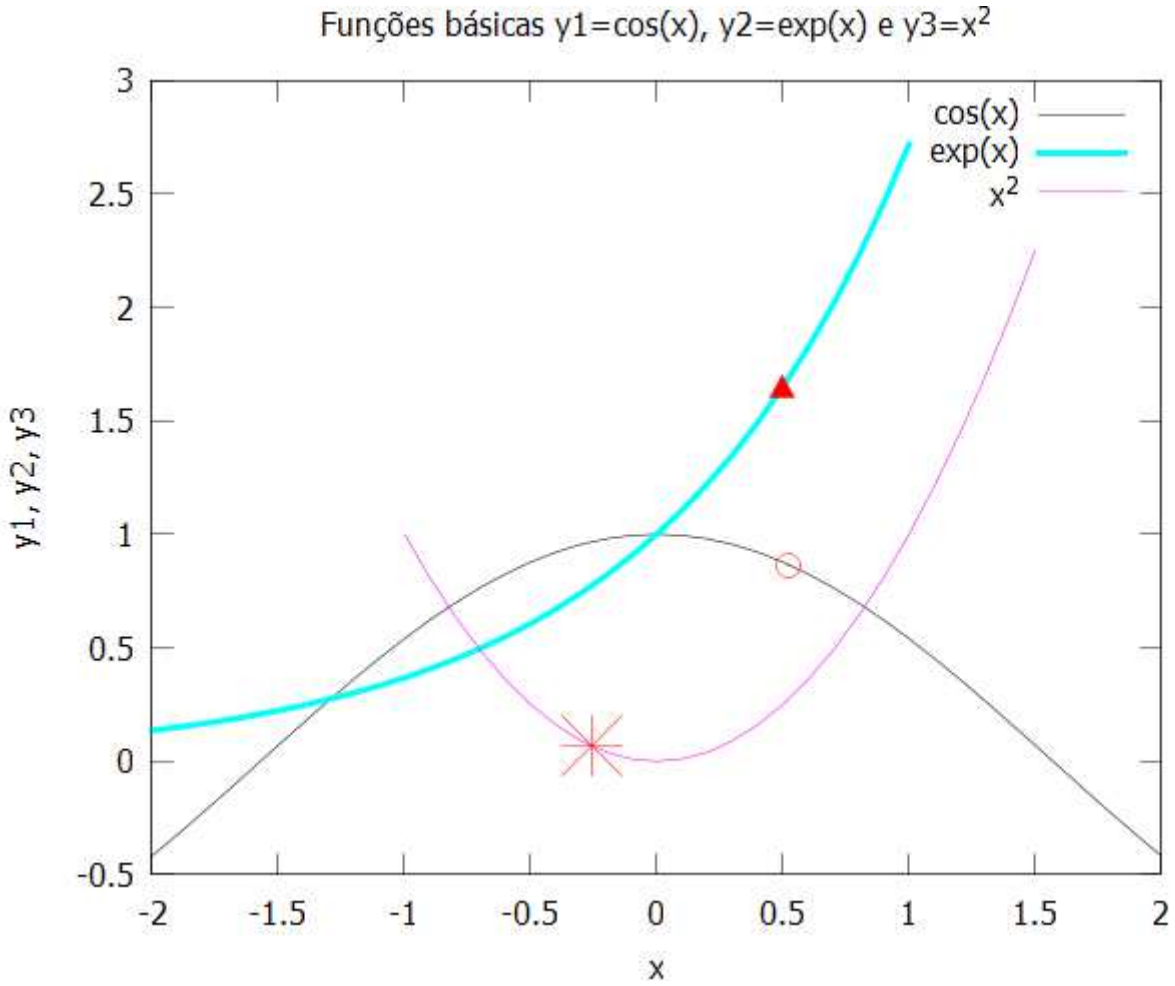
```
>> plot(x3,y3,"m;x^2;");
>> plot(pi/6,cos(pi/6),"ro", 0.5,exp(0.5),"r^");
>> plot(-0.26,(-0.26)^2,"r*","markersize",15);
>> hold off;
```

Existem mais uns comandos que permitem refinar ainda mais o gráfico,

```
xlabel("texto da etiqueta do eixo das abcissas");
ylabel("texto da etiqueta do eixo das ordenadas");
title("título do gráfico");
```

pelo que na sequência do exemplo anterior, para acrescentar etiquetas aos eixos e um título ao gráfico,

```
>> xlabel("x");
>> ylabel("y1, y2, y3");
>> title("Funções básicas y1=cos(x), y2=exp(x) e y3=x^2");
```



onde para estes comandos não é necessário executar "hold on" e "hold off" porque estes comandos não alteram o gráfico propriamente dito.

Finalmente, no ambiente Octave é possível criar muitos gráficos diferentes em muitas janelas diferentes que são numeradas ou indexadas sequencialmente (1,2,3,...). Para ajudar à gestão de várias janelas gráficas existem os comandos gcf, figure() e clf. Existe também o conceito de janela activa, que é aquela onde serão desenhados os gráficos se for executado um comando como plot() ou title().

clf; limpa ou apaga a janela gráfica activa (clear figure).
gcf; retorna o número ou índice da janela activa (get current figure).
figure; sem argumentos, cria uma nova janela e torna-a a janela activa.
figure(n); cria e/ou torna a janela número n a janela activa.
close(n); fecha a janela número n.
close("all"); fecha todas as janelas gráficas.

É possível por exemplo criar a janela número 5 sem existirem a 1,2,3 e 4. Antes de efectuar um novo gráfico, é usual executar o comando clf para limpar a janela. Por exemplo, para a figura número 3,

```
>> figure(3); clf; plot(...);
```

e para a janela activa,

```
>> figure(gcf); clf; plot(...);
```

FIM

Última alteração: Quinta, 19 Setembro 2019, 17:36

Tutorial Octave 4 - Scripts, Definição de Funções, Controlo de Fluxo e Entrada/Saída de Dados

V - Scripts

No ambiente Octave, scripts ou ficheiros batch têm a extensão .m permitindo agrupar vários comandos (programa) num ficheiro. Para executar os comandos contidos num ficheiro nome.m, trata-se o nome do ficheiro como outro comando qualquer do Octave,

```
>> nome
>>
```

executa os comandos contidos no ficheiro nome.m como se estes fossem dados um a um à prompt do Octave. Isto implica que o script pode usar variáveis definidas antes da sua execução e que variáveis criadas pelo script estarão disponíveis através da prompt após a execução do script.

VI - Controlo de fluxo

Ciclos

O ambiente Octave disponibiliza 3 meios de implementar ciclos: for, while e do-until. O ciclo for itera uma variável pelos elementos de um vector ou as colunas de uma matriz,

```
for i= v
   % i e' um escalar, elemento do vector v
   ...
end

for i= A
   % i e' um vector coluna da matriz A
   ...
end
```

Exemplo: somar todos os números ímpares até N>0

```
s=0;
for i= 1:2:N
```

```
   s=s+i;
end
```

O ciclo while permite colocar uma condição no início do ciclo, onde o corpo do ciclo é executado se a condição for verdadeira,

```
while (condicao)
   ...
end
```

Para o exemplo anterior vem,

```
s=0;
i=1;
while ( i<=N )
   s=s+i;
   i=i+2;
end
```

O ciclo do-until permite colocar uma condição no fim do ciclo, onde a execução do ciclo é terminada se a condição for verdadeira,

```
do
   ...
until (condicao)
```

Para o exemplo anterior vem,

```
s=0;
i=1;
do
   s=s+i;
   i=i+2;
until ( i>N )
```

Execução condicional

O ambiente Octave implementa o tradicional if-else, onde o corpo do if é executado se a condição for verdadeira, ou o corpo do else é executado se a condição for falsa,

```
if (condicao)
   ...
else
```

```
   ...
end
```

No caso de ser necessário testar várias condições, pode ser utilizada a forma if-elseif-...-elseif-else onde um número arbitrário n de elseif's pode ser usado,onde o corpo do else só é executado se todas as condições forem falsas,

```
if (condicao 1)
   ...
elseif (condicao 2)
   ...
elseif (condicao 3)
   ...
elseif (condicao ...)
   ...
elseif (condicao n)
   ...
else
   ...
end
```

Exemplo: somar todos os números ímpares e pares até N>0

```
s1=0; % somatorio impares
s2=0; % somatorio pares
aux=1; % aux==1 <==> i impar, aux==2 <==> i par
for i= 1:N
   if ( aux==1 )
      s1=s1+i; % i impar
      aux=2; % proximo e' par
   else
      s2=s2+i; % i par
      aux=1; % proximo e' impar
   end
end
```

VII - Funções

No ambiente Octave as funções são definidas em ficheiros .m, utilizando-se regra geral um ficheiro por função. Os ficheiros que definem funções distinguem-se dos scripts pelo facto que devem começar pela palavra chave function. Por exemplo, um ficheiro de nome soma.m que contem uma função que calcula a soma de dois escalares seria,

```
function s=soma(a,b)
   s=a+b;
end
```

onde a,b são as variáveis (argumentos) de entrada (separados por vírgulas) e s a variável de saída. Os nomes são arbitrários. A partir do momento que este ficheiro é definido, a função soma não se distingue de qualquer outra função interna do Octave, como a função cos para o cálculo do coseno, podendo ser utilizada na definição de outras funções ou invocada na linha de comandos,

```
>> soma(2,3)
ans = 5
>>
```

Também é possível definir mais do que uma variável de saída. Por exemplo, para definir uma função que calcula a soma e a multiplicação de dois escalares,

```
function [s,m]=somamult(a,b)
   s=a+b;
   m=a*b;
end
```

onde s,m são as variáveis de saída, definidas separadas por vírgulas e entre parêntesis rectos. De igual modo se invoca a função,

```
>> [x,y]=somamult(2,3)
x = 5
y = 6
>>
```

Não é obrigatório "recolher" todas as variáveis retornadas pela invocação de uma função, mas é preciso atribuir as n primeiras (ordenadas da esquerda para a direita). Por exemplo, se apenas interessar a soma, dado que é a primeira variável de saída,

```
>> z=somamult(2,3)
z = 5
>>
```

No caso de na definição de uma função ser necessário utilizar outras funções que não têm interesse em estarem acessíveis globalmente, podem-se definir no mesmo ficheiro funções privadas auxiliares da função principal. Por exemplo, se a função somamult fosse definida através de duas funções privadas soma e mult, o ficheiro somamult.m teria o seguinte conteúdo,

```
function [s,m]=somamult(a,b)
   s=soma(a,b);
   m=mult(a,b);
end

function s=soma(a,b)
   s=a+b;
end

function m=mult(a,b)
   m=a*b;
end
```

Todas as variáveis de entrada, saída e internas a uma função são locais. Quando uma função é invocada são feitas cópias das variáveis de e para as do "utilizador".

Referências para funções (Function Handlers)

Em Octave pode defenir-se uma variável que contém uma referência para uma função. Esta referência pode ser usada quer para invocar a função original quer para passá-la como argumento a outra função.

As referências são definidas precedendo-se o nome da função pelo carater '@', como em variável=@funcao. Suponha-se que já se encontra definida a função s=soma(a,b) como definida anteriormente. No exemplo seguinte, atribui-se à variável sh uma referência para a função soma(),

```
>> sh= @soma;
>> soma(1,4)
ans =  5
>> sh(1,4)
ans =  5
>>
```

que posteriormente é usada para invocar a função soma() como se da função original se tratasse.

A utilidade das referências para funções vai mais além permitindo que funções sejam elas próprias argumentos de outras funções. Seja a função

```
function r=operacao(op,a,b)
   r= op(a,b);
end
```

onde o argumento op é uma referência para uma função (deduz-se da utilização que lhe é dada no corpo da função). Baseado nas funções s=soma(a,b) e m=mult(a,b) como definidas anteriormente, poderia-se fazer,

```
>> ref= @soma;
>> operacao(ref,4,6)
ans =  10
>> ref= @mult;
>> operacao(ref,4,6)
ans =  24
>>
```

Funções anónimas (Anonymous Functions)

Uma função anónima é uma função que não tem nome, só existe uma referência para a função.

Numa função anónima define-se a expressão da função e uma referência ao mesmo tempo, nunca se chegando a definir um nome formal para essa função. A definição segue uma determinada sintaxe, como por exemplo

```
g= @(x,y) x+y*y;
```

define g como uma referência para uma hipotética função h(x,y)=x+y*y, só que a função h (podia ser outro nome) não existe. Recorrendo à função operacao() definida anteriormente, poderia-se fazer,

```
>> g= @(x,y) x+y*y;
>> operacao(g,4,6)
ans =  40
>>
```

As funções anónimas são úteis quando se pretende definir rapidamente funções relativamente simples.

VIII - Entrada e Saída de dados

O formato por defeito de apresentação dos números pode ser definido pelo comando format. De um modo geral pode-se escolher entre notação virgula fixa com poucos dígitos (short) ou muitos dígitos (long), notação exponencial (short ou long e,E), notação que resulta na mais curta entre vírgula fixa ou exponencial (short ou long g,G) e notação engenharia (short ou long eng) com 10^n onde n é sempre múltiplo de 3). Normalmente o formato por defeito é short e a notação em virgula fixa só é mantida se o nº for suficientemente pequeno.

Exemplos:

```
>> format short
>> x=12.34
x =  12.340
>> x=12345.67890123
x =  1.2346e+04
>> format long
>> x
x =  12345.6789012300
>> format short e
>> x
x =  1.2346e+04
>> format short E
>> x
x =  1.2346E+04
>> format long e
>> x
x =  1.23456789012300e+04
>> format long g
>> x
x =       12345.67890123
>> format short eng
>> x
x =   12.3457e+03
>>
```

É possível também escolher entre uma apresentação mais espaçada (loose) com mais linhas em branco ou mais compacta (compact) para a apresentação de matrizes.

```
>> format compact
>> x=rand(2,2)
```

```
x =
   0.10067   0.93581
   0.74862   0.60815
>> format loose
>> x=rand(2,2)
x =

   0.76929   0.83783
   0.98342   0.60729

>>
```

Saída de dados

Para saída simples de dados existe a função disp(x) que efetua a impressão do conteúdo da variável x seguido de uma mudança de linha.

```
>> x=123;
>> disp('Valor de x:'), disp(x)
Valor de x:
 123
>>
```

O octave possui um conjunto de funções muito similar às funções do C para ler/escrever em ficheiros, incluindo a possibilidade de ler/escrever na consola a partir de stdin/stdout. Para a saída de dados formatados existe a função

```
fprintf(fid,template,...)
```

onde a variável fid é uma handler para uma stream, que se for omitido é assumido stdout. A variável template é muito similar à da função printf() do C, com a possibilidade de indicar especificadores de conversão com o carater %, como por exemplo %d, %i, %s, %c, %o, %x, %f, %e, %g e %% para o próprio carater '%'.

```
>> x=67.56289;
>> fprintf('Valores de x: %d, %i, %o, %x, %.2f, %.3e, %g, %s,
%c\n',x,x,x,x,x,x,x,'abcde','A')
Valores de x: 67, 67, 103, 43, 67.56, 6.756e+01, 67.5629,
abcde, A
>>
```

Entrada de dados

Análogamente existe a função fscanf(fid,template,...) também muito similar à função scanf() do C, onde se fid for omitido é assumido stdin. Para entrada de dados muito simples na consola pode ser utilizada a função input(string), sendo impressa a string e retornado numa variável os dados introduzidos pelo utilizador e terminados com return.

```
>> x=input('Introduza o valor de x: ');
Introduza o valor de x: 3
>> x
x =  3
>> v=input('Introduza o vetor v: ');
Introduza o vetor v: [1 4 2 6 3 7]
>> v
v =
   1   4   2   6   3   7
>>
```

FIM

Última alteração: Quinta, 19 Setembro 2019, 18:08