Octave: Guia de Estudo

Nuno Cavalheiro Marques e Carmen Morgado

1 Breve Introdução

O Octave é uma linguagem de programação de alto nível, destinada ao tratamento de problemas para computação numérica. O interface com o programador é efectuado através de uma linha de comando.
O Octave pode efectuar cálculos aritméticos com reais, escalares complexos e matrizes; resolver sistemas de equações algébricas; integrar funções sobre intervalos finitos e infinitos e integrar sistemas de equações diferenciais ordinárias e diferenciais algébricas.
Permite gerar para o ecrã e para a impressora gráficos 2D e 3D, utilizando o Gnuplot.
O Octave é em grande parte compatível com o MatLab.
Este documento constitui um breve resumo do manual do Octave que pode ser consultado em http://www.octave.org/doc/octave_toc.html ou localmente em http://www.di.fct.unl.pt/cursos/icp/downloads/octave.pdf.

2 Utilização do sistema

Nesta secção apresentar-se-á o Octave como uma ferramenta de cálculo para problemas de computação numérica. O seu objectivo é descrever o funcionamento geral da aplicação, enquanto linguagem de linha de comando.

O Octave (http://www.octave.org) é uma aplicação desenvolvida utilizando uma filosofia *OpenSource*. A principal vantagem deste tipo de aplicações relativamente aos seus equivalentes comerciais (nomeadamente do MatLab, http://www.mathworks.com/) é o custo: as ferramentas *OpenSource* são de utilização livre, estando inclusive o seu código fonte disponível para o público em geral. Assim aplicações com este tipo de licença, podem ser livremente distribuídas e instaladas em qualquer computador. O código fonte da aplicação está igualmente disponível, o que fornece uma garantia de qualidade e robustez, sendo ainda possível adaptar ou estender a própria aplicação para a solução de determinados problemas. Podem ser obtidos mais detalhes sobre as ferramentas desenvolvidas com este tipo de licença, consultando http://www.gnu.org/philosophy/free-sw.pt.html.

O Octave é uma ferramenta desenvolvida para o Sistema Operativo Linux

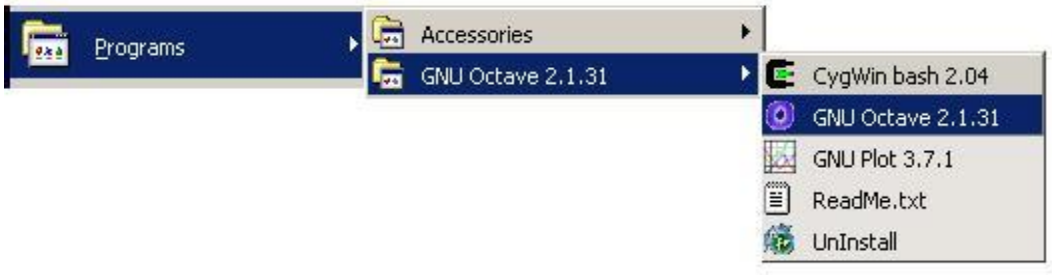
(http://www.linux.org), sendo actualmente distribuída com as principais versões deste sistema operativo. A sua utilização no ambiente Windows é igualmente possível através da utilização da ferramenta Cygwin (http://www.cygwin.com), a qual fornece um ambiente de emulação da plataforma Linux sobre Microsoft Windows.

Pode efectuar-se o download do Octave para MS-Windows quer da página da cadeira (http://di.fct.unl.pt/cursos/icp), quer directamente, do site http://sourceforge.net/projects/matlinks. A instalação do Octave num computador equipado com sistema operativo Windows XP, NT ou 2000, deverá ser relativamente simples. A instalação é igualmente possível nos sistemas MS-Windows 95/98 e Me, no entanto nestes últimos por vezes ocorrem problemas.

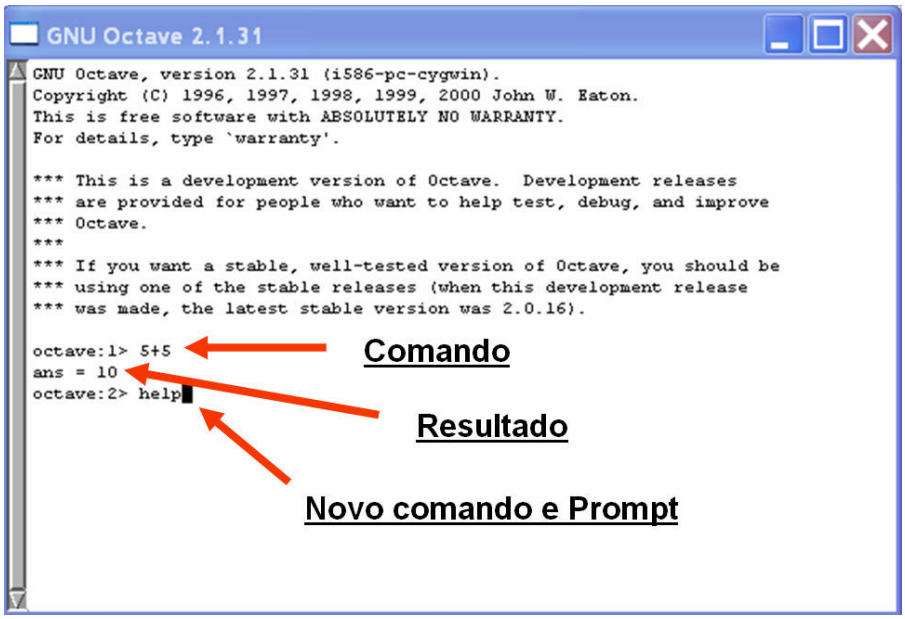
Após efectuar o Download do programa de instalação, bastará executar este programa e o Octave será instalado no seu computador. Em caso de dificuldades, o site indicado para obter mais informação para resolução de problemas relativamente à instalação do Octave sobre plataformas Windows é http://octave.sourceforge.net/Octave_Windows.htm.

2.1 Linha de Comandos

O primeiro passo para trabalhar com o Octave é obviamente o inicio desta aplicação. Para tal bastará seleccionar o programa no menu iniciar, conforme a figura abaixo.



De seguida deverá ser apresentada a linha de comandos do Octave com uma aparência semelhante à seguinte figura:



Contrariamente aos ambientes gráficos normalmente disponíveis no Windows, o Octave caracteriza-se por receber os seus comandos directamente da linha de comandos (ou *Prompt*). Na linha de comando, o sistema indica que aguarda um comando apresentando (por omissão) o texto `"octave:1>"` seguido de um cursor a piscar.

Após este *prompt* é possível escrever os comandos que controlam a aplicação. Deverá introduzir cada comando separadamente e seguido pela tecla [*Return*]. Num ambiente gráfico os comandos são especificados ao computador através de acções do rato sobre símbolos (ou ícones) visuais. Numa linha de comandos os comandos devem ser especificados através do teclado.

No exemplo anterior foi introduzido (pelo utilizador) um comando simples: **`5+5`**

Tendo o computador dado a resposta (resultado): **`ans = 10`**

Seguida do *prompt* pedindo novo comando: **`octave:2>`**

É fácil utilizar o Octave como calculadora. Eis alguns exemplos auto-explicativos:

```
octave:9> 2+3
ans = 5

octave:10> 2-2
ans = 0

octave:11> 2*2
ans = 4
```

```
octave:12> 2/3
ans = 0.66667

octave:13> 5*5*5
ans = 125

octave:14> 5^3
ans = 125

octave:15> 5^2.5
ans = 55.902

octave:16> 8\3
ans = 0.37500

octave:17> 2\4
ans = 2

octave:18> 3*(23+14.7-4/6)/3.5
ans = 31.743

octave:19> 5^-3
ans = 0.0080000
```

O único comando menos usual é a divisão à esquerda, a qual pode ser facilmente compreendida com a igualdade x/y ≡ y\x.

De igual forma podem ser utilizados comandos um pouco mais complexos, envolvendo funções. As seguintes funções elementares de cálculo poderão ser utilizadas:

`octave:20> sqrt(5)` `ans = 2.2361`	raíz quadrada
`octave:21> log10(1000)` `ans = 3`	logaritmo base 10
`octave:22> log(e^10)` `ans = 10`	logaritmo neperiano
`octave:23> pi` `pi = 3.1416`	constante pi
`octave:24> sin(pi/6)` `ans = 0.50000`	seno
`octave:25> cos(0)` `ans = 1`	coseno
`octave:26> tan(pi/4)` `ans = 1.0000`	tangente
`octave:27> sin(pi)` `ans = 1.2246e-16`	seno de pi não é 0, por causa de erros de aproximação

octave:28> 30*pi/180 ans = 0.52360 octave:29> sin(ans)^2+cos(ans)^2 ans = 1	utilização de valor anterior em cálculo

Repare-se que `sin(pi)` não é calculado como zero! Tal deve-se a um erro de cálculo do próprio computador. O número apresentado é no entanto muito próximo de zero: 1.2245×10^{-16}. Trata-se de um erro de aproximação do algoritmo de cálculo do seno. Normalmente este número varia de computador para computador, mas está sempre presente quando se efectuam cálculos reais. Note-se aliás que é possível calcular a precisão de um resultado, para tal utiliza-se o comando `format`:

octave:61> pi pi = 3.1416	contante **pi**
octave:62> format long octave:63> pi pi = 3.14159265358979	formatação **long**, a partir deste ponto a precisão dos resultados passa a ser de 15 algarismos significativos (máx de 24 caracteres)
octave:64> format short octave:65> pi pi = 3.14 octave:66> sin(pi) ans = 1.22e-16	formatação **short**, precisão passa a ser de 3 algarismos significativos
octave:67> format bank octave:68> sin(pi) ans = 0.00	formatação **bank**, precisão passa a ser de 2 casas decimais
octave:69> format octave:70> pi pi = 3.1416	formatação standard, precisão de 5 algarismos significativos (é o default)

Em geral os interfaces por linha de comandos são menos intuitivos que os interfaces gráficos, pois exigem do seu utilizador a memorização de um conjunto de palavras chave. No entanto, após o período inicial de adaptação os interfaces de linha de comandos revelam-se bastante mais fiáveis e fáceis de descrever: o número de formas distintas de entrar um mesmo comando é muito menor, sendo o número de acções de gestão de interface reduzidas ao mínimo. Compreende-se pois que seja muito mais fácil descrever a interacção com uma ferramenta deste tipo que a interacção com uma ferramenta visual. Por ser igualmente mais difícil entrar comandos não intencionais, é igualmente muito mais claro qual foi a acção que deu origem a um determinado erro.

A maior vantagem desta forma de interacção é no entanto a de permitir ao utilizador especificar sequências não elementares de comandos simples. É assim possível a definição de comandos extremamente complexos. Este tipo de utilização será descrito no próximo capítulo.

Neste capítulo vamos centrar-nos na aprendizagem dos comandos elementares. Durante o estudo deste texto, é muito importante testar cada um dos comandos isoladamente, para melhor se compreender a que se deve cada erro. Basta escrever mal uma letra, ou esquecer um dos espaços necessários num comando para que este não seja entendido pelo computador e dê origem a um erro.

Tal como na generalidade dos interfaces com *prompt* de comando, é possível navegar pelo histórico de comandos, bem como pesquisar comandos antigos. De seguida são apresentadas algumas das teclas utilizadas no Octave para facilitar (e tornar mais rápida) a entrada e alteração de comandos:

- Teclas cursoras (cima e baixo): comando anterior/seguinte no histórico de comandos.
- Teclas cursoras (esquerda e direita): letra anterior/seguinte no comando corrente. Em alguns terminais onde as teclas cursoras não funcionem, pode ser necessário utilizar as teclas **CTRL-b** (de *backward*) e **CTRL-f** (de *forward*).
- **CTRL-a** : o cursor desloca-se para o inicio do texto.
- **CTRL-e** : o cursor desloca-se para o fim do texto.
- **CTRL-r/CTRL-s** : pesquisa incremental de um comando anterior/seguinte no histórico de comandos.
- **CTRL-_** : desfazer o último comando.
- **TAB** : Terminar o comando.
- **CTRL-k** : Mover todo o texto até ao fim da linha na área de transferência do Octave.
- **CTRL-y** : Mover o texto na área de transferencia do Octave para a linha de comandos.

Convém igualmente referir a possibilidade de o computador começar a executar cálculos muito longos ou mesmo que não terminem. Nesse caso há um conjunto de teclas destinado a indicar ao computador que deve interromper os seus cálculos e voltar a apresentar a linha de entrada de comandos. Para interromper qualquer calculo no Octave, bastará manter pressionadas em simultâneo a tecla **Ctrl** e a tecla **c** (i.e. **CTRL-c**).

Há igualmente alguns comandos de controle do próprio Octave que pode ser útil conhecer: o comando `help`, o comando `history`, o comando `run_history` e o

comando `exit`.

O comando `help`, apresenta um manual on-line referindo todos os possíveis comandos do Octave. Pode ser seguido do texto de outro comando, como por exemplo: `octave:1> help exit`

O comando `history` possibilita ao utilizador visualizar os comandos que entrou até ao momento.

O comando `run_history <numero_inicial> <numero_final>` possibilita ao utilizador repetir comandos anteriores. Veja-se o exemplo:

```
octave:100> history

...

22 5*5
23 3+2;
24 5+5

...

octave:101> run_history 22 24
ans = 10
ans = 5
ans = 25
```

O comando `exit` termina a execução do Octave e retorna ao MS-Windows.

Resumo
• Como iniciar o Octave. • Linha de comandos. • Operações básicas de cálculo. • Formatação. • Vantagens da linha de comando face aos interfaces gráficos. • Teclas de edição. • Comandos de controlo do Octave.

2.2 Variáveis e Matrizes

A principal vocação do Octave é, como tem sido referido, o calculo numérico e matricial. Comecemos pois por definir a matriz ***a***:

```
octave:3> a=[ 1,1,2 ; 3,5,8 ; 13,21,34 ]
a =
    1   1   2
    3   5   8
   13  21  34
```

variável *a* passa a representar a matriz:

$$\begin{bmatrix} 1 & 1 & 2 \\ 3 & 5 & 8 \\ 13 & 21 & 34 \end{bmatrix}$$

Como se pode observar , representando novas colunas e os ; representando novas linhas. Podemos igualmente definir uma matriz como o retorno de uma função, neste caso uma função que retorna uma matriz com 3 linhas e 2 colunas com valores aleatórios entre 0 e 1: `octave:4> b=rand(3,2) ;`

Desta vez, no entanto, o Octave não imprime o resultado após o comando. Tal deve-se ao *Ponto e Vírgula* no final da linha de comando. Este ponto e vírgula indica que não estamos para já interessados na avaliação do valor definido (que pode ser bastante pesada). Assim que pretendermos ver um valor anteriormente definido, bastará pedir o seu valor, p.ex:

```
octave:4> b
b =
   0.88406 0.90013
   0.73682 0.15829
   0.68952 0.74250


octave:5> a*b
ans =
      2.9999 2.5434
     11.8525 9.4318
     50.4098 40.2707
```

o valor de **a** e **b** utilizado no produto de matrizes, é o da matriz **a** e **b** já definidas

Os exemplos anteriores introduzem igualmente o conceito de variável no Octave.

Em Octave uma variável é um nome que se atribui (através da utilização do operador **=**) a um valor (seja ele escalar ou matriz), por forma a que este valor possa ser repetidamente consultado mais tarde. Os nomes das variáveis podem conter qualquer sequência de letras, números ou *underscore* (i.e. o carácter _) que não seja iniciada por número. Não é aconselhável utilizar nomes de variáveis com mais de 30 caracteres.

A utilização de letras minúsculas ou maiúsculas é relevante e tem significados distintos. Assim a variável *a* pode conter um valor e a variável *A* outro, i.e. *a* e *A* são distintos.

Há dois comandos essenciais para gerir as variáveis definidas até determinado momento: *who* e *clear*. *who* lista as variáveis já definidas e *clear* remove variáveis.

```
octave:41> i = 10;
octave:42> id = [1,0;0,1]
id=
   1 0
   0 1
octave:43> who

*** local user variables:

i id

octave:44>clear i
octave:45>who

*** local user variables:

id
```

As variáveis iniciadas por dois sinais de *underscore* seguidos estão reservadas para utilização do sistema. Não devem pois ser utilizadas, excepto se se souber exactamente o que se quer fazer.

Há igualmente um conjunto de variáveis de sistema que disponibilizam informações relevantes sobre o sistema. Por agora apenas necessitaremos de utilizar a variável `ans` - último valor calculado.

2.3 Matrizes e Séries

A principal vantagem do Octave é a sua capacidade para tratamento de Matrizes.

Vamos utilizar a matriz ***a*** da secção anterior:

```
octave:3> a = [ 1,1,2; 3,5,8; 13,21,34 ]
```

Podemos igualmente criar novas matrizes contendo esta. O único cuidado a ter é manter o número de linhas e colunas constante. Será fácil definir as matrizes:

Correcto	Errado
octave:4> [a, a] ans= 1 1 2 1 1 2 3 5 8 3 5 8 13 21 34 13 21 34 octave:5> [a; a] ans= 1 1 2 3 5 8 13 21 34 1 1 2 3 5 8 13 21 34	octave:4> [a, a; a] error: number of columns must match (3 != 6)

Podemos consultar uma dada posição ou submatriz numa matriz especificando os índices das respectivas dimensões num tuplo. Como pode observar nos seguintes exemplos (assumindo a matriz *a* anteriormente definida)

```
octave:14> a(1,2)
ans = 1

octave:15> a(1,[1,2,3])
ans =
     1  1  2

octave:16> a([1,2,3],2)
ans =
      1
      5
     21
```

Repare-se, nos 2º e 3º exemplos, na utilização dum vector para indicar, respectivamente, as colunas e linhas a seleccionar.

A necessidade de especificar séries enquanto vectores é tão grande, que existe um operador especifico para o fazer:

```
octave:1> 1:5
ans =
      1  2  3  4  5

octave:2> 1:3:11
ans =
     1  4  7  10
```

A sintaxe geral de uma série é pois:

LIMITE_INFERIOR:PASSO:LIMITE_SUPERIOR

O valor de PASSO pode ser omitido (será considerado a 1 por omissão). Há ainda outra simplificação às séries, quando utilizados como índices de matrizes. Neste caso, como sabemos o limite inferior e superior podemos utilizar apenas o sinal : para especificar toda uma linha ou toda uma coluna da matriz original:

```
octave:23> [a(1:2,2:3); a(:,2:3)]
ans =
     1  2
     5  8
     1  2
     5  8
    21 34
```

O tamanho da matriz pode igualmente ser obtido com a função `size` (retorna o número de linhas e colunas):

```
octave:24> size(a)
ans =
     3  3

octave:25> size(ans)
ans =
    1  2

octave:26> ans(1)
ans = 1
```

Por fim resta referir que o operador de atribuição permite alterar posições numa matriz, podendo ser utilizada a matriz vazia (`[]`) para apagar linhas ou colunas:

```
octave:51> a(:,1) = [ 1; 2; 0 ]
a =
     1  1  2
     2  5  8
     0 21 34
octave:52> a(1,:) = 1:10:30
a =
    1 11 21
    2  5  8
    0 21 34
octave:53> size(a)
ans =
    3  3
octave:54> a(1,:) = []
a =
```

```
     2  5  8
     0 21 34
octave:55> size(a)
ans =
     2  3
octave:56> a(:,1)= []
a =
     5  8
    21 34
octave:57> size(a)
ans =
     2  2
```

Com pares de valores, podemos igualmente atribuir variáveis dentro de uma matriz:

```
octave:58> [la,ca] = size(a)
la = 2
ca = 2
```

Resumo

- Definição de matrizes.
- Variáveis.
- Comandos `who` e `clear` sobre variáveis.
- Construção de matrizes compostas.
- Selecção de submatrizes.
- Tamanho de uma matriz.
- Séries.
- Atribuição de submatrizes
- Remoção de linhas e colunas

2.4 Strings

Uma constante do tipo **String** é uma sequência de caracteres entre `"` ou `'`. Em Octave uma string pode ter qualquer tamanho. Eis um exemplo:

```
octave:1> a= "uma cadeia de caracteres"
a = uma cadeia de caracteres
```

Se tentarmos definir uma matriz com várias **strings** numa mesma linha acabamos por obter uma **string** com a junção das várias **strings** base:

```
octave:2> ["Uma string e",a]
ans = Uma string euma cadeia de caracteres
```

De facto as **strings** são vistas internamente como vectores linha contendo caracteres. As linhas de uma matriz de **strings** têm forçosamente a mesma dimensão. É assim possível utilizar os operadores:

```
octave:3> ["123";"3"]
ans =
     123
     3
```

```
octave:4> a(1:7)
ans = uma cad
```

Vejamos algumas funções pré-definidas para o tratamento de **strings**:

Descrição	*Exemplo*

`findstr(s,t)` : Encontra todas as posições de `t` em `s`	`octave:5> findstr("abcabcabdad","ab")`
	`ans =`
	`1 4 7`

`split(s,t)` : Divide uma string num vector (coluna) de strings separados por `t`	`octave:6> split("abcabcabdad","ab")`
	`ans =`
	`c`
	`c`
	`dad`

`strrep(s, x, y)` : substitui todas as ocorrências de `x` por `y` na string `s`	`octave:6> strrep("abcabcabdad","ab","AB")` `ans = ABcABcABdad`
`str2num(s)` : converte um número representado numa string para um número	`octave:7> str2num("555") + 5` `ans = 560`
`tolower(s)` / `toupper(s)` : converte uma string para minúsculas/maiúsculas	`octave:8> toupper("Atencao - aviso")` `ans = ATENCAO - AVISO`

`strcmp(s1,s2)` : compara as strings `s1` com `s2`, se forem iguais retorna 1, se diferentes retorna 0	`octave:9> strcmp("teste 1", "teste 1")`
	`ans = 1`
	`octave:10> strcmp("teste 1", " test")`
	`ans = 0`

Na página 40 (capítulo 5) do manual de Octave poderá obter mais informação sobre outras funções para a manipulação de **Strings**.

2.5 Principais Operadores e Funções

Até agora vimos as principais características essenciais ao funcionamento do Octave. Resta no entanto enumerar as principais funções e operadores. A lista que se segue é complementada com exemplos ilustrativos do tipo de cálculos que podem ser obtidos com os diversos operadores.

Eye, Ones e Zeros

As matrizes Identidade, ou contendo apenas zeros ou uns são tão frequentes que foram criados três operadores específicos para as gerar:

```
octave:37> eye(3)
ans =
      1  0  0
      0  1  0
      0  0  1
```

```
octave:38> eye(3,2)
ans =
      1 0
      0 1
      0 0
```

```
octave:39> ones(3,2)
ans =
      1  1
      1  1
      1  1
```

```
octave:40> zeros(2,4)
ans =
      0 0 0 0
      0 0 0 0
```

Matriz transposta

Para converter linhas em colunas bastará aplicar o operador ' a uma matriz:

```
octave:41> ones(3,2)'
ans =
     1  1  1
     1  1  1
```

Operadores Soma e subtracção

Para além de efectuarem a adição básica, efectuam também a adição e subtracção de matrizes. O número de linhas/colunas tem de ser igual.

```
octave:1> analitico = [0.0314, 0.1257, 0.9998];
octave:2> numerico = [0.0389, 0.1530, 1.0082];
octave:3> erro = analitico - numerico
erro =
     -0.0075000  -0.0273000  -0.0084000
```

Operadores para Produto e Potenciação em Matrizes

O operador produto é dos mais poderosos no Octave/Matlab. De facto pode ser aplicado a pares de números, a um número e matriz ou a duas matrizes n-por-m e m-por-p, em todos os casos o significado é o usual da álgebra. Considere-se o exemplo:

```
octave:4> tudo = [1,0,0;0,1,0;-1,1,0]*[analitico;numerico;zeros(1,3)]
tudo =
       0.0314000 0.1257000 0.9998000
       0.0389000 0.1530000 1.0082000
       0.0075000 0.0273000 0.0084000
octave:5> erroTotal=(([0,0,0; 0,0,0; 0,0,1]* tudo)'*ones(3,1))'*ones(3,1)
erroTotal = 0.043200
octave:6> [1,0,0;0,1,0;0,0,1/erroTotal*100]*tudo
ans =
      0.031400  0.125700  0.999800
      0.038900  0.153000  1.008200
     17.361111 63.194444 19.444444
```

Neste exemplo, na linha 4 (2º termo), construímos uma matriz contendo na primeira linha um conjunto de resultados analíticos, na segunda dados observados (numéricos) e na terceira zeros. Devido à primeira matriz,

construiu-se uma matriz `tudo`, contendo as primeiras 2 linhas da matriz do 2º termo e no fim a subtracção da 1ª linha pela 2ª linha.

A linha 5 é um pouco mais sofisticada. A ideia base é obter apenas a última linha da variável `tudo` num vector linha. Para tal, primeiro anulam-se todos os elementos das duas primeiras linhas, e depois com uma matriz de uns, retira-se apenas a última linha. Depois multiplica-se o resultado por um vector linha contendo uns, para obter o somatório de todos os elementos. Obviamente trata-se de um simples exercício para ilustrar as potencialidades do produto de matrizes. A solução Octave seria bem mais simples: `tudo(3,:)`

A linha 6 limita-se a transformar a última linha da matriz `tudo` num valor percentual.

Tal como o produto, a potenciação de matrizes (que só pode ser aplicada a matrizes quadradas), limita-se a efectuar o produto de uma matriz por ela própria n vezes.

Extensão das funções simples a matrizes: produto e potenciação pontual

Podemos aplicar qualquer das funções elementares a uma matriz: a função é simplesmente aplicada elemento a elemento. Veja-se o seguinte exemplo:

```
octave:60> A=[1,2,3;4,5,6;7,8,9]
A =
   1 2 3
   4 5 6
   7 8 9
octave:61> L=log10(A)
L =
    0.00000 0.30103 0.47712
    0.60206 0.69897 0.77815
    0.84510 0.90309 0.95424

octave:62> (10*ones(size(A))).^L-A
ans =
     0.0000e+00  0.0000e+00  0.0000e+00
     0.0000e+00  8.8818e-16  0.0000e+00
     0.0000e+00 -8.8818e-16  0.0000e+00
```

Note-se o operador `.^` que eleva todos os elementos da matriz (individualmente) a `L`. Comandos semelhantes são o produto pontual e a divisão pontual:

```
octave:66> A.*[zeros(1,3) ; ones(2,3)]
ans =
     0 0 0
     4 5 6
     7 8 9

octave:67> A./(ones(size(A))/2)
ans =
      2  4  6
      8 10 12
     14 16 18
```

Solução de sistemas de equações lineares e determinantes

O cálculo de um determinante em Octave é realizado com o auxílio da função `det(A)`. Vejamos um exemplo para solução de um sistema de equações:

$$x_1 + 2 \cdot x_2 + 3 \cdot x_3 = 4$$
$$2 \cdot x_1 + 3 \cdot x_2 + 4 \cdot x_3 = 5$$
$$4 \cdot x_1 + 2 \cdot x_2 + 5 \cdot x_3 = 1$$

Pode ser representado por duas matrizes `A` e `B`, se considerarmos `AX=B`:

```
octave:71> A=[1, 2, 3; 2, 3, 4; 4, 2, 5]
A =
    1 2 3
    2 3 4
    4 2 5
octave:72> B=[4;5;1]
B =
    4
    5
    1
```

A aplicação da regra de Cramer (onde se substitui as colunas de `A`, correspondentes a cada x_i, por B de modo a obter 3 matrizes, dividindo-se o determinante de cada matriz obtida pelo determinante de `A`), pode ser feita da seguinte forma:

```
octave:84> D1=A; D1(:,1) = B
D1 =
     4 2 3
     5 3 4
```

```
     1 2 5
octave:85> D2=A; D2(:,2) = B
D2 =
     1 4 3
     2 5 4
     4 1 5

octave:86> D3=A; D3(:,3) = B
D3 =
     1 2 4
     2 3 5
     4 2 1
octave:87> X=[det(D1); det(D2); det(D3)] / det(A)
X =
    -1.40000
     1.80000
     0.60000

octave:88> A*X-B
ans =
     0.0000e+00
     0.0000e+00
    -9.9920e-16
```

Eliminação Gaussiana: Podemos obter o mesmo resultado utilizando o operador `\` (eliminação Gaussiana):

```
octave:89> X =A \ B
X =
    -1.40000
     1.80000
     0.60000
```

Inversão de Matrizes

A inversão de uma matriz é feita com o comando `inv(M)`.

```
octave:90> inv( A )
ans =
     -1.40000   0.80000   0.20000
     -1.20000   1.40000  -0.40000
      1.60000  -1.20000   0.20000
octave:90> A * inv( A )
ans =
      1.00000   0.00000   0.00000
```

```
    0.00000   1.00000   0.00000
    0.00000  -0.00000   1.00000
```

2.5 Gráficos

Os gráficos em Octave utilizam uma ferramenta OpenSource especialmente dedicada à geração de gráficos: o GnuPlot. O Octave comunica com o gnuplot com duas funções base (sobre as quais as restantes são implementadas), estas funções são descritas nas páginas 125 (gplot) e 131 (gsplot). Na prática iremos apenas utilizar funções de mais alto nível (desenvolvidas com recurso às funções elementares apresentadas).

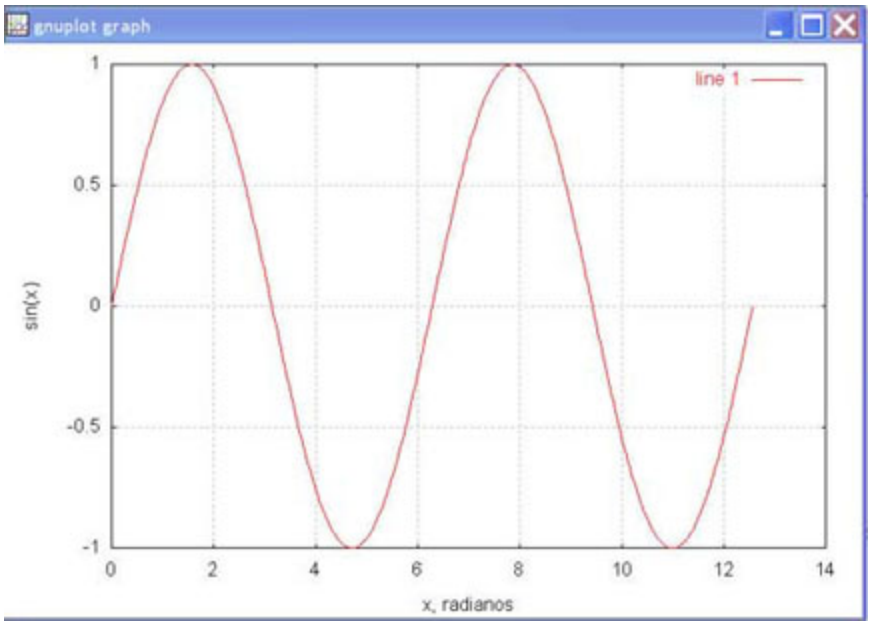
Gráficos simples

Na sua forma mais simples, a função `plot` apresenta um gráfico dos valores de dois vectores linha, correspondendo ao eixo dos `X` e `Y`:

```
octave:19> x = 0:pi/90:4*pi;
octave:20> y = sin(x);
octave:21> plot(x,y)
```

Neste momento deverá ser aberta uma nova janela contendo o gráfico. Poderá ter que utilizar o sistema operativo para ver esta janela (bastará carregar conjuntamente em **ALT-TAB**). Poderá alterar os parâmetros do gráfico com funções adicionais. Para visualizar as alterações terá que executar o comando replot:

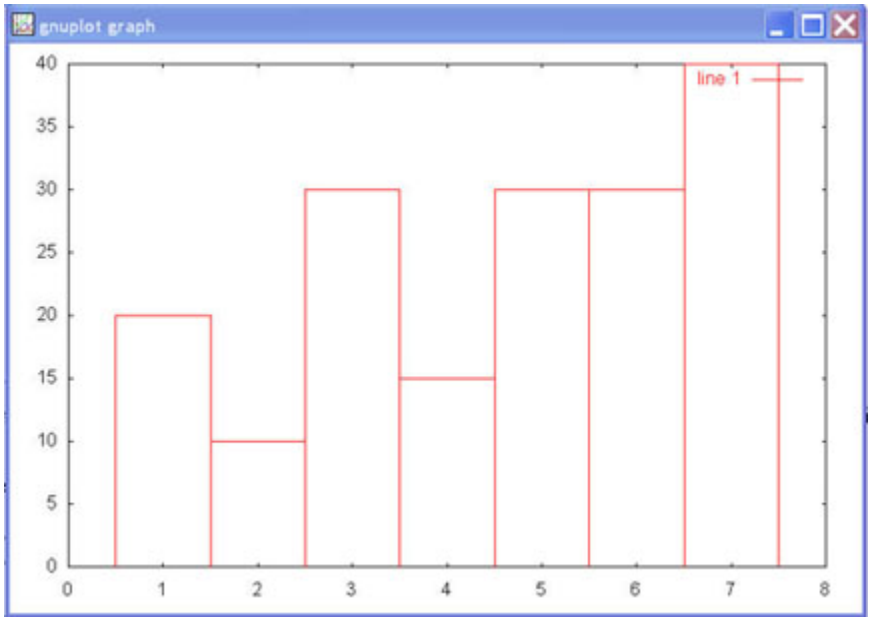
```
octave:29> xlabel('x,radianos');
octave:30> ylabel('sin(x)');
octave:31> grid;
octave:32> replot
```



De facto, a função plot pode ter diversos comportamentos distintos, consoante o tipo de argumentos utilizados. Para mais detalhes consultar o manual do Octave página 127.

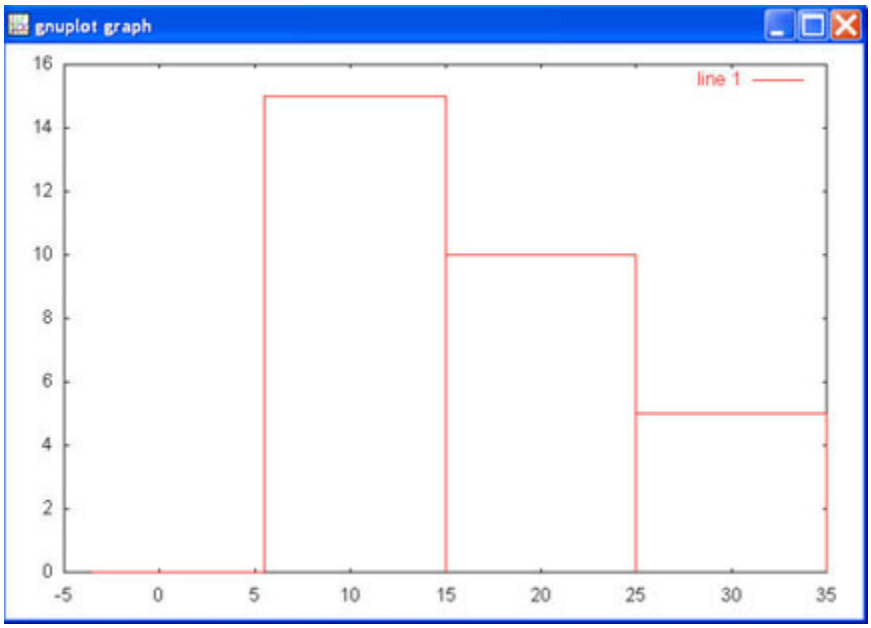
Duas funções gráficas também bastante utilizadas são: `bar` e `hist` :

```
octave:34> clearplot;
octave:35> bar([20,10,30,15,30,30,40])
```



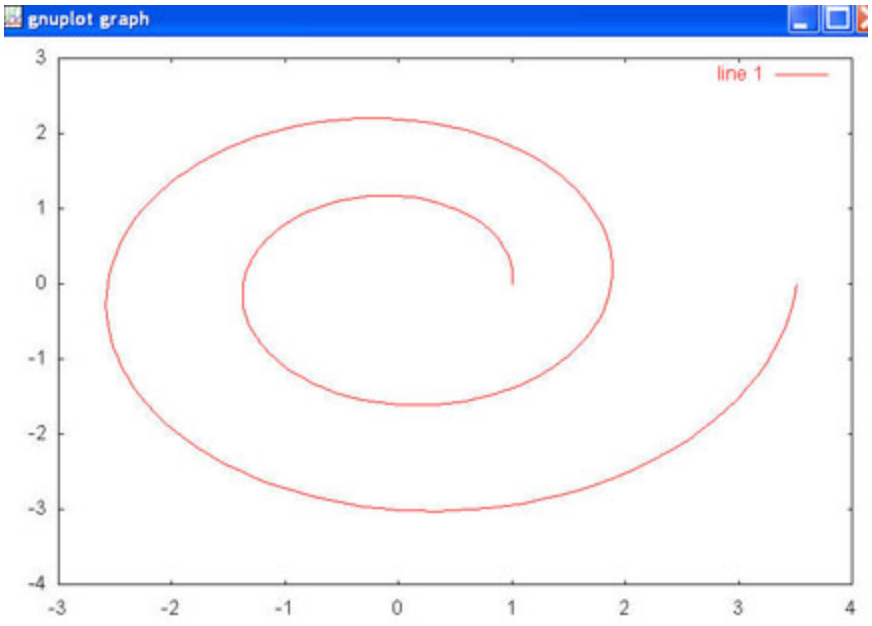
```
octave:36> clearplot
octave:37> a = [10,10,10,20,20,30];
octave:38> hist(a,[1,10,20,30])
```

O primeiro argumento da função contém a amostra de valores.
O segundo, o centro de distribuição.



Pode igualmente utilizar gráficos em coordenadas polares:

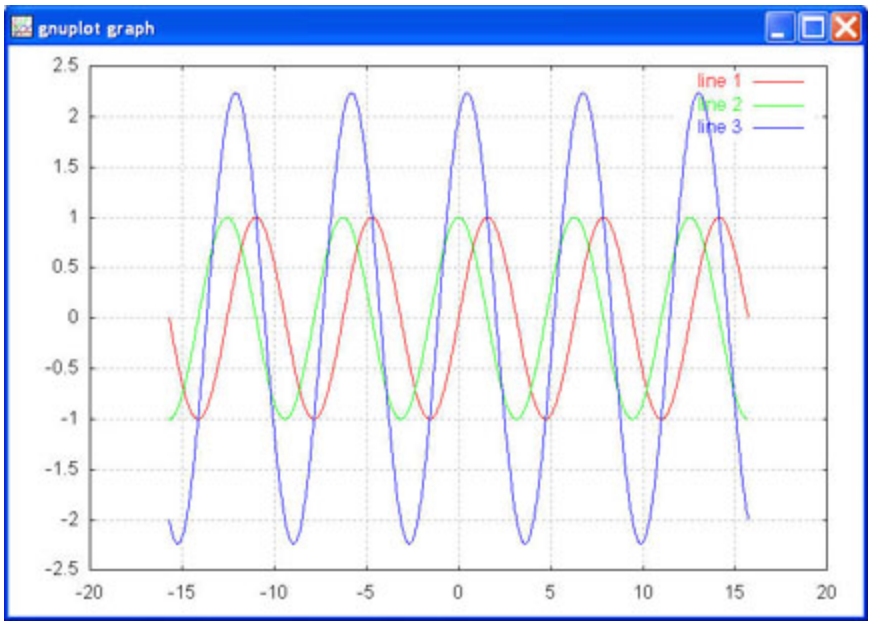
```
octave:61> clearplot
octave:62> m = 0.1;
octave:63> phi = 0:pi/60:4*pi;
octave:64> r = exp(m*phi)
octave:65> polar(phi,r)
```



Gráficos Sobrepostos

Antes de cada comando, o Octave, por omissão, limpa a janela de gráfico. Caso se pretendam comparar diversos gráficos pode ser utilizado o comando `hold on` para sobrepor gráficos:

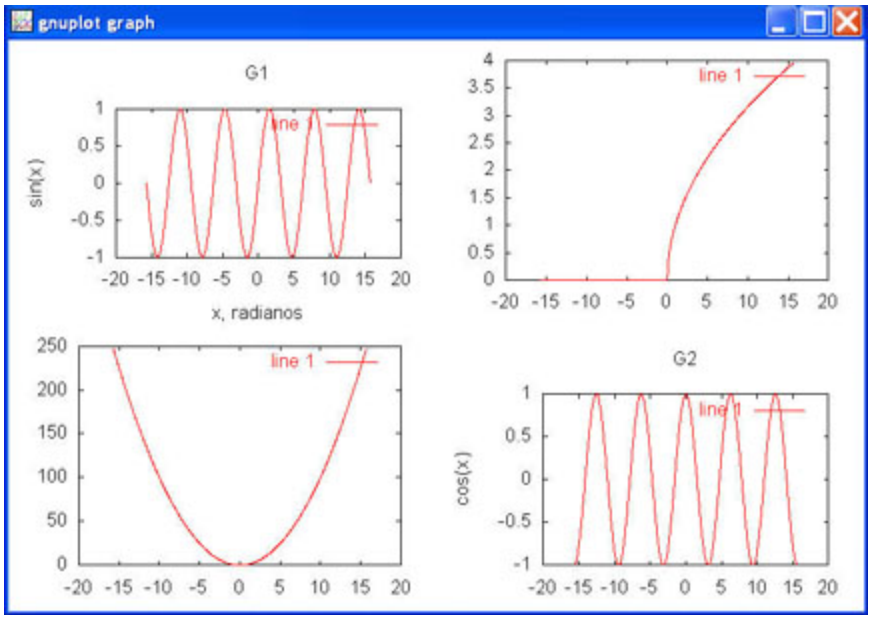
```
octave:71> clearplot
octave:72> hold on
octave:73> x = -5*pi:pi/90:5*pi;
octave:74> y1 = sin(x);
octave:75> y2 = cos(x);
octave:76> y3 = sin(x)+2*cos(x);
octave:77> plot(x,y1)
octave:78> plot(x,y2)
octave:79> plot(x,y3)
```



Note igualmente que pode utilizar directamente os menus na janela do GnuPlot para alterar o seu gráfico.

Se quiser apresentar vários gráficos numa mesma janela poderá utilizar as funções `multiplot` e `subwindow`:

```
octave:93> multiplot(2,2)
octave:94> subwindow(1,1)
octave:95> xlabel("x, radianos")
octave:96> ylabel("sin(x)")
octave:97> title("G1")
octave:98> plot(x,y1)
octave:99> subwindow(2,2)
octave:100> ylabel("cos(x)")
octave:101> title("G2")
octave:102> plot(x,y2)
octave:103> subwindow(1,2)
octave:104> plot(x,x.*x)
octave:105> subwindow(2,1)
octave:106> plot(x,sqrt(x))
```



Para voltar ao modo `normal`:

```
octave:108> oneplot
octave:109> clearplot
```

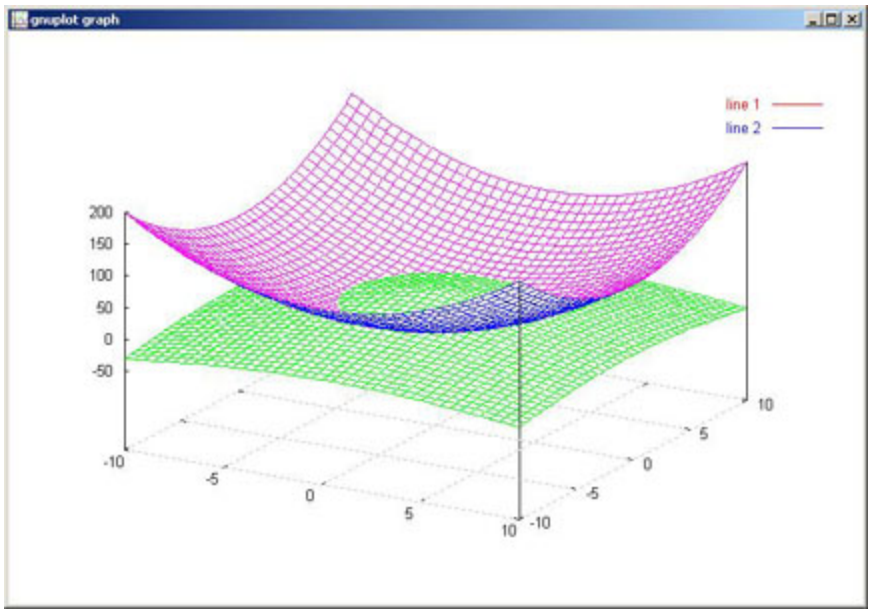
Gráficos 3D

A criação de um gráfico 3D necessita da definição de uma matriz de valores sobre os quais a função vai ser avaliada. Tal pode ser conseguido com a função `meshdom`:

```
octave:30> clearplot
octave:31> x = -10:0.5:10;
octave:32> [X,Y]=meshdom(x,x);
```

Resta apresentar o gráfico:

```
octave:33> hold on
octave:34> grid
octave:35> mesh(X,Y,-sqrt(X.^2+Y.^2)*5+40)
octave:36> mesh(X,Y,X.^2+Y.^2)
```



2.6 Exercícios I

1- Escrever numa matriz de 15 colunas e 1 linha, os quadrados dos primeiros 15 naturais (1, 4, 9 ...).

```
octave:1> (1:15).^2
```

2- Escrever numa matriz de 15 linhas e 1 coluna, os cubos dos primeiros 15 naturais (1, 8, 27 ...)

```
 octave:2> [(1:15).^3]'
```

3- Escrever numa tabela de 15 linhas por 3 colunas, os primeiros 15 naturais na 1ª coluna; os quadrados dos primeiros 15 naturais na 2ª e os cubos dos primeiros 15 naturais na 3ª coluna.

```
octave:3> seq=(1:15)
octave:4> [seq; seq.^2; seq. ^3]'
```

4- Dado que a conversão de graus Fahrenheit para graus Célcios se faz com a seguinte fórmula C= (5/9) . (F - 32). Coloque em tabela as conversões para as temperaturas de 0ºF, 25ºF,..., 250ºF.

```
octave:5> faixa = 0:25:250
octave:6> [faixa;(5/9)*(faixa-32)]'
```

5- Faça uma tabela que traduza graus Célcios em graus Fahrenheit. Coloque em tabela as conversões para as temperaturas de -200ºC, -175ºC,..., 200ºC.

```
octave:7> faixa= -200:25:200
octave:8> [faixa; faixa * (9/5) + 32]'
```

6- Considere o problema "Quem quer ser milionário ?" já conhecido. Resolva o seguinte utilizando Octave:

6.1- Qual o valor do saldo ao longo dos anos?

```
octave:9> limite=10
octave:10> cap_inicial=5000000
octave:11> periodo=0:10
octave:12> taxa=1.05
octave:13> [periodo; cap_inicial * (taxa.^periodo)]'
```

6.2- Junte uma coluna que indique o ganho (relativo ao ano aterior) ao longo dos anos.

```
octave:14>acumulado = cap_inicial * taxa .^ periodo
octave:15>ant_acumulado = [cap_inicial,acumulado(1:limite)]
octave:16>evolução=[periodo; acumulado; acumulado - ant_acumulado]'
```

6.3- Indique o ganho médio e ao longo dos anos.

```
octave:17>ganho_médio= sum(evolução(3,1:11))/limite
```

6.4- Indique o ganho total

```
octave:18>ganho_total=sum(evolução(3,:))
```

Ou

```
octave:18>ganho_total = evolução(2,limite+1) - cap_inicial
```

3 Programação

Como já foi visto anteriormente, o Octave permite ao utilizador especificar instruções através de uma linha de comando. Nesta secção vamos elaborar um pouco mais o tipo de instruções que podem ser executadas e introduzir algumas noções básicas de programação. Assim, os objectivos desta secção são os seguintes:

1. **Compreender** os conceitos básicos de programação, e saber quais os comandos Matlab/Octave que os implementam;
2. Dado um problema de engenharia concreto, conseguir determinar qual a **sequência de comandos** da linguagem MatLab/Octave a utilizar;

3. **Implementar** essa sequência de comandos sobre a forma de um Script;
4. Conseguir **depurar** os erros desse Script, e adaptar o código desenvolvido a novos problemas de Engenharia.

3.1 Alguns conceitos básicos

Ao programar está-se de um modo geral a expressar acções ou comportamentos que simulem uma determinada actividade do mundo real. Assim, será primeiro necessário compreender a actividade que se quer simular. Depois, saber quais são os dados ou parâmetros que condicionam a actividade no mundo real (i.e. quais os elementos que contêm a informação necessária para o problema). Por fim é necessário identificar quais as acções e comportamentos que permitem simular esse processo. No fundo é necessário modelar o problema ou, se tal já tiver sido feito, utilizar um modelo (normalmente matemático) que já alguém elaborou.

Como já foi estudado nas aulas teóricas, um computador é uma máquina que funciona com base num conjunto de acções elementares bastante simples e pode ser descrito matematicamente por um máquina de Turing Universal. Tal como na máquina de Turing, as acções e comportamentos estão associados ao controlo da sequência de instruções que constituem o programa. Normalmente este processo é designado como controlo de fluxo de execução do programa.

No desenvolvimento de um programa é igualmente usual especificar primeiro acções mais complexas. Cada acção complexa deverá ser decomposta em acções mais simples, as quais por sua vez serão mais uma vez decompostas, até termos por fim todas as acções expressas através de pequenas acções elementares. No caso de comandos matlab este paradigma pode ser chamado de programação de Cima para Baixo. Alguns autores propõe também a abordagem inversa: começar por conjugar acções elementares até se resolver o problema final (esta abordagem pode também ser designada por da programação de Baixo para Cima). Na prática é usual conjugar ambos os métodos: constroem-se de baixo para cima acções relativamente simples que sabemos serem necessárias e depois tenta-se decompor o problema de acordo com essas e outras instruções. No meio de tudo o que é necessário reter é que qualquer sequência de acções, por mais complexa que seja, pode ser expressa numa linguagem de programação desde que essa linguagem possua: um mecanismo sequencial; um mecanismo condicional; e um mecanismo de repetição. Fica como exercício compreender de que forma estes mecanismos são implementados e implementam a máquina de Turing (i.e. na prática são igualmente expressivos!).

Mecanismo sequencial

Entende-se como mecanismo sequencial, uma forma de expressar através de uma linguagem que a acção 1 deve ser executada antes da acção 2. Na maior parte das linguagens de programação o mecanismo sequencial surge como uma simples enumeração das acções a executar, da esquerda para a direita, de cima para baixo.

Mecanismo condicional

O mecanismo condicional permite expressar uma situação em que se pretende que: caso uma condição se verifique, seja executada uma acção (acção 1), caso essa mesma condição não se verifique seja executada uma outra acção (acção2). Só uma das acções é executada!

Mecanismo de repetição

O mecanismo de repetição da linguagem permite expressar a necessidade de uma determinada acção, ou conjunto (bloco) de acções ser repetida um certo número de vezes. Tipicamente essa acção (ou conjunto de acções) é repetida até que uma condição de paragem se verifique.

A maior parte das linguagens de programação, são linguagens que possuem os mecanismos anteriormente referidos, e que permitem que as sequências de acções expressas através da linguagem possam ser transformadas em sequências de comandos passíveis de serem executadas por um computador. No entanto, há linguagens adaptadas às mais diversas funções: um Professor pode gostar de ensinar futuros programadores a programar utilizando a linguagem Pascal, pois esta é uma linguagem desenhada explicitamente para ensinar a programar. De igual forma, um técnico de bases de dados pode gostar de especificar o seu problema utilizando SQL, uma linguagem explicitamente virada para pesquisas em grandes volumes de informação (e que correntemente é utilizada nas principais bases de dados como por exemplo ORACLE, DB2 da IBM ou o SQL Server da Microsoft).

O Octave permite a elaboração de programas, expressos na linguagem matlab. Linguagem essa que possui os mecanismos de programação anteriormente expressos.

3.2 Instruções

Em Octave, instruções (ou comandos) podem ser uma simples expressão constante (`x= 2;`) ou uma lista de comandos que por sua vez podem conter outros comandos. Nesta secção são descritos os comandos que irão permitir a elaboração de programas em Octave, em particular os mecanismos de controlo de fluxo (mecanismos de condição e de repetição). Os mecanismos de controlo de fluxo permitem controlar o fluxo de execução de um programa. Todos eles começam e terminam com uma palavra chave.

Condição

Uma condição é uma expressão Booleana - pode por isso tomar um de dois valores Verdade (*true*) ou Falso (*false*). Essa expressão pode ser uma expressão simples ou agregar outras expressões através de operadores Booleanos.

Operadores de comparação

Como exemplo de expressões simples tem-se as operações envolvendo os operadores de comparação:

`x < y`	verdade se x menor que y
`x <= y`	verdade se x menor ou igual a y
`x == y`	verdade se x igual a y
`x >= y`	verdade se x maior ou igual a y
`x > y`	verdade se x maior que y

`x != y`
`x ~= y`
`x <> y`
	verdade se x diferente de y

Os operadores de comparação comparam dois valores numéricos - se a relação for verdadeira o resultado é 1 caso contrário é 0.

Alguns exemplos:

`octave:1> x=2`
`x = 2`
`octave:2> x > 4`
`ans = 0`
	comparação de dois valores numéricos

`octave:3> [1, 2; 3,4] == [1, 2; 2, 4]`
`ans =`
`1 1`
`0 1`
	comparação de matrizes é efectuada elemento a elemento

`octave:10> [1, 2; 2,4] == 2`
`ans =`
`0 1`
`1 0`
	neste caso o valor escalar é comparado com cada um dos elementos da matriz

`octave:7> index("file.txt",".")!=5` `ans = 0`	comparação em que um dos valores é retornado por uma função

Operadores Booleanos

Uma expressão Booleana é uma combinação de expressões de comparação utilizando os operadores Booleanos "ou" ('||'), "e" ('&&') e a "negação" ('!'). Descrição sumária dos operadores:

`Booleano1 || Booleano2`	operador **ou** (or): o resultado é verdade se pelo menos uma das expressões for verdade (true)
`Booleano1 && Booleano2`	operador **e** (and): o resultado é verdade se e só se todas as expressões forem verdade

`! Booleano1` `~ Booleano1`	operador **negação** (not): o resultado é verdade se a expressão for falsa (false)

A instrução `if`

O comando `if` é um comando de condição. Em Octave a sintaxe deste comando é a seguinte:

Sintaxe	*Exemplo*	*Descrição*
if *(condição)* *bloco* ***endif***	***if*** *(x > y)* *y = y+1* *x = 1* ***endif***	o bloco de instruções (y=y+1 e x=1) só é cumprido se a condição (x>y) for verdadeira (=1)

Em que o bloco pode ser um comando ou uma lista de comandos (inclusive outro if).

Na condição (x>y) é feita uma comparação entre o valor de x e de y, pelo que se pressupõe que estas variáveis foram previamente definidas, ou seja que lhes foi anteriormente atribuído um valor. O mesmo se passa com instruções do bloco, por exemplo em y=y+1.

Outra forma do comando `if`:

Sintaxe	*Exemplo*	*Descrição*
if *(condição)* *bloco1* ***else*** *bloco2* ***endif***	***if*** *(x > y)* *maior =x* ***else*** *maior =y* ***endif***	se a condição for verdadeira, o bloco1 é executado; senão, é cumprido o bloco2

A instrução `while`

O comando `while` é um comando de repetição (*loop*), ou seja, leva a que uma determinada zona de código seja executada enquanto determinada condição se verificar.

Em Octave a sintaxe deste comando é a seguinte:

Sintaxe	*Exemplo*	*Descrição*

`while (condição)`
`  bloco1`
`endwhile`
	`while ( x > y)`
	`  y = y+1`
	`endwhile`
		enquanto a condição for verdadeira o bloco de instruções vai ser executado, até que a condição se torne falsa.

A condição vai controlar o número de vezes que o ciclo vai ser executado. Se a condição for inicialmente falsa, o bloco de instruções nunca será executado. Tenha em atenção que se a condição não convergir para falso o ciclo nunca terminará.

Analise o seguinte exemplo, que cria uma variável `fib` que contém os elementos da sequência de Fibonacci.

Exemplo	*Descrição*

`fib = ones(1,10);`
`i = 3;`
`while ( i <= 10)`
`  fib(i)=fib(i-1)+fib(i-2)`
`  i = i+1;`
`endwhile`
	A variável *i* é inicializada com o valor 3.
	O `while` testa se o valor de *i* é menor do que 10 - se for verdade, o i-ésimo elemento de *fib* fica com o valor da soma dos dois anteriores elementos e o valor de *i* é incrementado.
	Este ciclo vai ser repetido enquanto a condição se verificar, termina quando *i* atinge o valor 11.

A instrução `for`

O comando `for` é também um comando de repetição, tal como o `while`. Este comando é utilizado quando se pretende contar o número de iterações de um ciclo. A sua sintaxe é a seguinte:

Sintaxe	*Exemplo*	*Descrição*

`for var = expr`
`  bloco`
`endfor`
	`for i=1:10`
	`  y = y+1`
	`endfor`
		O bloco de instruções (y=y+1) vai ser executado 10 vezes (início do ciclo em 1 e fim em 10).
		o valor da variável *i* vai sendo incrementado automaticamente em cada passagem (o valor do passo neste caso é 1).

Outros exemplos utilizando como domínio para a variável de iteração um vector e uma matriz:

Exemplo 1	*Exemplo 2*

`octave:1> x=[2,4,10,14];`
`octave:2> for i=x`
`> v = i`
`> endfor`
`v = 2`
`v = 4`
`v = 10`
`v = 14`
`octave:3>`
	`octave:3> y=[2,4;10,12]`
	`y =`
	`  2  4`
	`  10  12`
	`octave:4> for i=y`
	`> m = i`
	`> endfor`
	`m =`
	`  2`
	`  10`
	`m =`
	`  4`
	`  12`
	`octave:5>`

Analise o exemplo da geração da série de Fibonacci, mas recorrendo agora à utilização do comando `for`.

Exemplo	*Descrição*

`fib = ones(1,10);`
`for i = 3:10`
`  fib(i)=fib(i-1)+fib(i-2)`
`endfor`
	Inicialmente é avaliada a expressão '3:10', para gerar um intervalo de valores de 3 a 10 inclusive. Seguidamente, a variável *i* toma o valor inicial do intervalo (3) e o bloco de instruções é executado a primeira vez. No final da execução do bloco, a variável *i* toma o próximo valor do intervalo e o bloco será novamente executado. Este processo repete-se até que não existam mais valores no intervalo.

A instrução `break`

O comando `break` permite parar uma instrução de repetição (`while` ou `for`) e saltar para a próxima linha de código do programa. Este comando só pode ser utilizado dentro de um ciclo.

Exemplo	*Descrição*

`for i=1:10`
`  y = y + i`
`  if ( y == 6 )`
`  break`
`  endif`
`endfor`
`x = y`
	Neste exemplo, temos um ciclo que irá ser executado 10 vezes, se a condição `y==6` não ocorrer. Se durante a execução do ciclo essa condição se verificar, a instrução `break` será executada e o ciclo de `for` quebrado, passando a execução directamente para a instrução seguinte ao ciclo (`x=y`).

3.3 Leitura/Escrita de valores

Para além dos comandos de controlo de fluxo de dados existe a necessidade de em algum ponto do programa ler valores e fazer sair para o exterior (quer seja para o ecrã ou para um ficheiro) os dados resultantes do processo de execução do programa.

O Octave tem disponíveis várias instruções que permitem efectuar a entrada de dados para o programa (ler do exterior, ficheiro ou utilizador na linha de comando) e fazer sair resultados (escrita para ficheiro ou para o ecrã).

De seguida são descritas algumas dessas instruções, existem no entanto mais e para tomar conhecimento pode consultar o manual no capítulo "13. Input and Output".

Saída para o Terminal (instrução `disp`)

Normalmente o Octave faz sair para o terminal (ecrã) o valor da expressão que acabou de avaliar (tal só não acontece se colocar no final do comando o carácter ;).

No entanto, existe muitas vezes a necessidade de visualizar o resultado de um cálculo não quando este é avaliado mas, por exemplo, no final de uma determinada sequência de acções.

O comando `disp(x)`, escreve no terminal o valor de `x`.

Compare os dois exemplos a seguir apresentados:

Exemplo1	*Exemplo2*

```
octave:1> y=0;
octave:2> for i=1:4
> y=y+10
> endfor
y = 10
y = 20
y = 30
y = 40
octave:3>
```

```
octave:3> y=0;
octave:4> for i=1:4
> y = y+10;
> endfor
octave:5> disp("O valor de y:"), disp(y)
O valor de y:
40
octave:6> disp("O valor de y:"), y
O valor de y:
y = 40
octave:7>
```

Como se pode verificar no exemplo1, em cada iteração do ciclo é enviado para o terminal o valor da variável, enquanto que no exemplo 2, o valor da variável é visualizado após o final do ciclo utilizando o comando `disp`. Compare a diferença entre a utilização do comando `disp(y)` e a chamada `y`.

Entrada a partir do terminal (instrução `input`)

O Octave tem disponíveis três funções que possibilitam a interacção com o utilizador ao nível da entrada de dados, vamos aqui referir somente uma delas (`input`), poderá obter informação sobre as restantes consultando o manual.

A instrução `input(prompt)` emite para o terminal o prompt e aguarda que o utilizador introduza um valor. Analise o seu comportamento através do seguinte exemplo:

```
octave:1> input("Qual o valor de x? ")
Qual o valor de x? 10
ans = 10
octave:2> x=ans;
octave:3> y = x^2;
octave:4> disp("quadrado de x :"), disp(y)
quadrado de x :
100
octave:5>
```

Para que não seja visível no terminal `ans=10` deverá colocar ; no final do comando de `input`.

Ficheiros de entrada/saída

Para além da interacção directa com o utilizador através do terminal, existe muitas vezes necessidade de ler dados do exterior e armazenar os resultados que estão a ser produzidos pelo nosso programa. Este problema surge normalmente quando o volume de informação envolvida possui uma dimensão considerável. O mais apropriado nestas situações é armazenar a informação em ficheiros, para que esta possa ser facilmente manipulada.

Abertura e fecho de ficheiros (`fopen` e `fclose`)

Antes de poder efectuar qualquer acção de leitura ou escrita sobre um ficheiro deve proceder à sua abertura. No final da manipulação deverá sempre fechá-lo.

Assim duas funções essenciais na manipulação de ficheiros são; **`fopen`** e **`fclose`**.

<table>
<tr><th>Sintaxe</th><th>Descrição</th></tr>
<tr>
<td>
<code>fid = fopen(nome_fich,modo)</code>

Possíveis valores para <code>modo</code>:
<table>
<tr><td><code>'r'</code></td><td>abre ficheiro existente para leitura</td></tr>
<tr><td><code>'w'</code></td><td>abre ficheiro para escrita, o anterior conteúdo do ficheiro é eliminado</td></tr>
<tr><td><code>'a'</code></td><td>abre ficheiro para escrita no final do ficheiro (append)</td></tr>
<tr><td><code>'r+'</code></td><td>abre existente para leitura e escrita</td></tr>
<tr><td><code>'w+'</code></td><td>abre ficheiro para leitura e escrita, o anterior conteúdo do ficheiro é eliminado</td></tr>
<tr><td><code>'a+'</code></td><td>abre ou cria ficheiro para leitura e escrita no final do ficheiro</td></tr>
</table>
</td>
<td>
abre o ficheiro com o nome <code>nome_fich</code>, no modo especificado no parâmetro <code>modo</code>, e devolve uma variável <code>fid</code>, com o seguinte formato:

<pre>fid =
{
  id =
  name =
  mode =
  arch =
  status =
}</pre>
Em que:
<code>id</code> é um valor inteiro que, a partir do momento da abertura, vai identificar o ficheiro;
<code>name</code> é o nome do ficheiro, deve ser igual ao parâmetro <code>nome_fich</code>;
<code>mode</code> é o modo como o ficheiro foi aberto;
<code>arch</code> é o tipo de interpretação pela arquitectura;
<code>status</code> indica o estado em que se encontra o ficheiro;
</td>
</tr>
</table>

	No caso de problemas na abertura, `fid` toma o valor -1
fclose(fid)	fecha o ficheiro com o identificador `fid`.

A partir do momento da abertura, o ficheiro passa a ser reconhecido no nosso programa pelo seu **`fid`**.

Exemplo de utilização:

Exemplo	*Descrição*

`octave:4> f1=fopen("dados.txt","r")`
`f1 =`
`{`
`  id = 3`
`  name = dados.txt`
`  mode = r`
`  arch = native`
`  status = open`
`}`
	Abertura do ficheiro já existente para leitura, com o nome "dados.txt", que se encontra na directoria corrente.

`octave:10> f2=fopen("tmp/dados2.txt","r")`
`f2 =`
`{`
`  id = 4`
`  name = tmp/dados2.txt`
`  mode = r`
`  arch = native`
`  status = open`
`}`
	Abertura de um ficheiro já existente para leitura, com o nome "dados2.txt", que se encontra na subdirectoria `tmp` da directoria corrente.

`octave:7> f3=fopen("teste.txt","r")` `f3 = -1`	Tentativa de abertura de um ficheiro para leitura, mas em que ocorreu um problema (por exemplo, ficheiro não existente).

`octave:8> f4=fopen("teste.txt","w")`
`f3 =`
`{`
`  id = 5`
`  name = teste.txt`
`  mode = w`
`  arch = native`
`  status = open`
`}`
	Abertura do ficheiro "teste.txt" para escrita. Neste caso o ficheiro ainda não existia, mas como foi aberto para escrita, vai ser automaticamente criado na directoria corrente.

Mais uma vez não se esqueça que antes de poder escrever ou ler de um

ficheiro há que garantir que este está aberto. Quando terminar a manipulação de um ficheiro deverá sempre fechá-lo.

Leitura e escrita simples de ficheiro

São de seguida apresentadas algumas das funções disponíveis em Octave para efectuar a leitura e escrita simples de ficheiros.

Para escrever uma string num ficheiro pode utilizar a função **`fputs`**, que tem como argumentos a identificação do ficheiro e a string que se pretende escrever. Existe uma função semelhante (**`puts`**), mas que em vez de escrever para o ecrã escreve para o terminal, e que tem como único argumento a string.

Para efectuar a leitura de dados de um ficheiro de texto, podem ser utilizadas duas funções, a **`fgetl`** ou a **`fgets`**, tendo ambas dois argumentos: o identificador de ficheiro e a dimensão máxima dos dados a ler. Ambas lêem caracteres de um ficheiro até encontrar o carácter de fim de linha ou até ter atingido a máxima dimensão (o que ocorrer primeiro). A única diferença é que a função `fgetl` não coloca no valor devolvido o carácter fim de linha.

Exemplo	Descrição

`octave:1> fid=fopen("teste.txt","w");`
`octave:2> str="Para teste a escrita em ficheiros\n";`
`octave:3> fputs(fid,str)`
`ans = 0`
`octave:4> fputs(fid,"segunda linha")`
`ans = 0`
`octave:5> fputs(fid,"\n")`
`ans = 0`
`octave:6> fputs(fid,"outra linha")`
`ans = 0`
`octave:7> fclose(fid)`
`ans = 0`
	Este exemplo abre ficheiro para escrita (se ainda não existia, cria-o) e escreve em "teste.txt", após o que fecha o ficheiro.
	O carácter '`\n`' sinaliza o fim de linha.
	
	Analise o resultado desta acção, abrindo o ficheiro "teste.txt" com um editor de texto (por exemplo o Notepad) e observe o conteúdo do ficheiro.

`octave:14> fid=fopen("teste.txt","r");`
`octave:15> str1=fgetl(fid,100);`
`octave:16> str2=fgets(fid,100);`
`octave:17> str3=fgets(fid,4);`
`octave:18> fclose(fid);`
`octave:19> str1`
`str1 = Para teste a escrita em ficheiros`
`octave:20> str2`
`str2 = segunda linha`
	Exemplo de leitura de ficheiros de texto utilizando as funções `fgetl` e `fgets`.

```
octave:21> str3
str3 = outr
octave:22>
```

Escrita e leitura formatada

As funções aqui apresentadas permitem a escrita e leitura, em ficheiro e no terminal, de qualquer tipo de elementos e não só de strings. Este tipo de funções de leitura e escrita são algo semelhantes às existentes na linguagem de programação C.

Para Escrita	
printf(template, ...)	escreve para o terminal de acordo com a formatação especificada em `template`
fprintf(fid, template, ...)	escreve para o ficheiro `fid`, de acordo com a formatação especificada em `template`

Para a leitura de ficheiros pode ser utilizada a função **`fscanf`**. Tenha em atenção que a leitura em Octave está pensada para a manipulação de matrizes de valores e não para a manipulação de ficheiros de texto.

Para Leitura	
[val,count] = **fscanf**(fid, template, size)	lê do ficheiro `fid`, de acordo com a formatação especificada em `template`. `size` indica o número de elementos a ler.

Tem disponíveis várias conversões de variáveis, as mais usuais são: **`%d`**, inteiro com sinal; **`%f`**, real com sinal; **`%s`**, string; **`%c`**, carácter.

Observe os seguintes exemplos de utilização destas funções:

```
octave:1> nome="Antonio Silva";
octave:2> num=12753;
octave:3> nota=15.5;
octave:4> printf("Aluno %s num: %d nota %f \n",nome,num,nota);
Aluno Antonio Silva num: 12753 nota 15.500000
octave:5> m=[1 2 3; 4 5 6; 7 8 9];
```

Escrita formatada para ecrã

```
octave:6> printf("%d ",m);
1 4 7 2 5 8 3 6 9
```

```
octave:7> fid=fopen("teste.dat","w");
octave:8> fprintf(fid,"%d ",m);
octave:9> fprintf(fid,"%d ",m*2);
octave:10> fclose(fid)
ans = 0
octave:11> fid2=fopen("teste.dat","r");
octave:12> x1=fscanf(fid2,"%d ",[3,3])
x1 =
  1  2  3
  4  5  6
  7  8  9
octave:13> [x2,dim]=fscanf(fid2,"%d ",[3,3])
x2 =
   2   4   6
   8  10  12
  14  16  18
dim = 9
octave:14> fclose(fid2);
```

Escrita de duas matrizes de valores num ficheiro e respectiva leitura.
O terceiro argumento da função indica a dimensão da matriz.
Como pode observar, no segundo exemplo de leitura, a função também retorna o número de elementos lidos.

```
octave:15> fid=fopen("teste.txt","r");
octave:16> x3=fscanf(fid,"%d ",10)'
x3 =

   1   4   7   2   5   8   3   6   9   2

octave:17> x4=fscanf(fid,"%d ",10)'
x4 =

   8  14   4  10  16   6  12  18

octave:18> x5=fscanf(fid,"%d ",10)'
x5 = []
octave:19> fclose(fid);
```

Outro exemplo de leitura do mesmo ficheiro "`teste.dat`", mas agora lendo os valores para uma variável vector.
Agora o terceiro argumento indica o numero de elementos a ler. Ao chegar ao fim do ficheiro (não tendo mais elementos para ler) afecta a variável com o vector vazio `[]`.
Se não colocar nenhum valor no terceiro parâmetro da função, esta vai ler para um vector todos os elementos do ficheiro.

```
octave:21> m2=log(m)
m2 =

  0.00000  0.69315  1.09861
```

Escreve no ecrã os valores formatados como reais.

```
  1.38629  1.60944  1.79176
  1.94591  2.07944  2.19722

octave:22> printf("%.5f %.4f %.6f \n",m2);
0.00000 1.3863 1.945910
0.69315 1.6094 2.079442
1.09861 1.7918 2.197225
octave:23>printf("%f %d %.2f \n",[1,2],[3,4]);
1.000000 2 3.00
4.000000
```

O `printf` efectua um ciclo sobre o template de formatação, o que pode levar a alguma confusão (linha 23) (O valor antes de `f` indica o número de casas decimais).

3.4 Funções

A organização do código de modo a torná-lo mais versátil e estruturado deve ser um dos objectivos a ter em mente aquando da elaboração de um programa.

Em Octave é possível ao utilizador/programador definir as suas próprias funções. Uma função é uma parte de um programa identificada por um nome, possivelmente parametrizada, que corresponde a uma acção (ou conjunto de acções) que pode ser executada mediante a sua chamada.

Na sua forma mais simples, uma função pode ser:

Exemplo	*Descrição*
function *msg_entrada* *printf("Hoje está um lindo dia\n");* ***endfunction***	A partir do momento da sua declaração é possível utilizar a função. Para tal, basta executar o seu nome (`msg_entrada`) na linha de comando.

O exemplo anterior é um exemplo muito simples, não possui parâmetros de entrada nem retorna nenhum valor de saída.

Exemplo	*Descrição*
function *ret_val = area_circ(raio)* *ret_val = pi*raio^2;* ***endfunction***	Esta função calcula a área de um círculo, tem um parâmetro de entrada (*raio*) e retorna um valor.

Tenha em atenção que no caso da função retornar um valor (caso típico), este deve ser sempre (em qualquer situação) efectuado.

Errado	*Correcto*
function *retval = avg(v)* ***if*** *(is_vector(v))* *retval = sum(v)/length(v);* ***endif*** ***endfunction***	***function*** *retval = avg(v)* *retval = 0;* ***if*** *(is_vector(v))* *retval = sum(v)/length(v);* ***else*** a = 78.540

Exemplo de utilização (chamada):

```
octave:10> area_circ(5)
```

```
    printf("erro, o argumento deve ser um vector\n")
  endif
endfunction
```

O exemplo da esquerda não é correcto porque no caso da condição (`if`) não se verificar, o valor de retorno não é afectado.

Uma função pode retornar um ou mais valores. No caso de retornar múltiplos valores estes são especificados segundo um vector, como pode ver no exemplo:

Exemplo	*Descrição*

`function [max,idx] = vmax(v)`
`  idx = 1;`
`  max = v(idx);`
`  for i=2:length(v)`
`  if ( v(i)> max )`
`  max = v(i);`
`  idx = i;`
`  endif`
`  endfor`
`endfunction`
	Esta função devolve o valor máximo de um vector e a posição que ocupa.
	
	Exemplo de utilização (chamada):
	
	`octave:30> [valor,pos] = vmax([2 5 3 7 8 1 4])`
	`valor = 8`
	`pos = 5`
	`octave:31> maximo = vmax([2 4 6 7 3 6])`
	`maximo = 7`
	
	Repare que na segunda chamada, como não foi especificado um vector vai devolver somente o primeiro valor (`max`)

3.5 Ficheiros de script e de funções

Um ficheiro de script pode conter qualquer sequência de comandos Octave. Os comandos descritos no ficheiro são executados um a um e em sequência, como se estivessem a ser introduzidos na linha de comando.

O Octave (no seu ambiente de linha de comando) executa os ficheiros script que possuam a extensão ".m". Por exemplo: "`octave: 2 > teste`" irá executar os comandos que se encontram escritos no ficheiro *teste.m*, ficheiro esse que deverá estar na directoria corrente, ou então a directoria, onde se encontra o ficheiro, deverá estar definida na variável `LOADPATH` do Octave (para mais informações deve consultar o manual).

Para determinar qual a directoria corrente pode executar o comando `pwd`, se obtiver como resposta "`/cygdrive/c`", quer dizer que está na directoria raiz de execução. Se pretender mudar de directoria pode utilizar o comando `cd` seguido do caminho para onde pretende ir. Por exemplo se pretender deslocar-se para

a directoria *tmp/test_files* (que é uma subdirectoria da directoria raiz), deve escrever na linha de comando "`octave: 3 > cd tmp/test_files`". Para listar os ficheiros que se encontram na directoria corrente execute o comando `ls`.

Pode atribuir outra extensão ao ficheiro, no entanto para que este seja interpretado como um ficheiro de *script* deverá escrever na linha de comando: `source nome_do_ficheiro`.

Para além de comandos pode escrever comentários nos ficheiros. Um comentário é um troço de texto que é incluido num programa e que tem como objectivo facilitar a compreensão durante a leitura, do código do programa. Um comentário serve por exemplo, para explicar o que uma determinada função faz, como funciona, quais os parâmetros de entrada e o que retorna.

Uma linha de comentário deverá ser iniciada por '**#**' ou '%' (só é válido linha a linha). O Octave ao detectar um destes símbolos ignora toda a linha.

Ficheiros de funções

Na maior parte das vezes não é prático ter que definir todas as funções de cada vez que são necessárias. O que deve fazer é guardá-las num ficheiro de forma a que possam ser facilmente consultadas e alteradas, caso seja necessário.

Para que o Octave reconheça um ficheiro como sendo um ficheiro de função, deverá logo no início do ficheiro definir a função. Ou seja, o primeiro comando a ser executado deve ser a declaração de função. Apresenta-se de seguida um ficheiro exemplo:

```
# função que calcula a area de uma circunferencia
# dado o valor do seu raio.
function ret_val = area_circ(raio)
     ret_val = pi*raio^2
endfunction
```

O Octave não necessita que o ficheiro de função seja "chamado" antes de poder utilizar a função, no entanto há que garantir que o ficheiro é colocado numa directoria onde o Octave o consiga encontrar. **Porquê?** O Octave ao encontrar um identificador como sendo um nome de função (e se não for uma que já se encontre "carregada"), vai efectuar uma busca na lista de directorias especificadas na variável LOADPATH de ficheiros com extensão '.m' que possuam o mesmo nome da função que pretende.

Deverá portanto ter cuidado na atribuição do nome a um ficheiro de função

(i.e. se definir a função *area_circ* o ficheiro deverá chamar-se '*area_circ.m*').

Muitas das funções standard do Octave, como por exemplo funções estatísticas, de manipulação de polinómios, de tratamento de sinais, etc..., estão definidas em ficheiros '.m'. Esses ficheiros encontram-se numa subdirectoria do Octave \usr\local\share\octave\2.1.31\m.

Exemplo da utilização de funções de polinómios definidas no Octave

`octave:1> mypoly=[1,3,2,-1]`
`mypoly =`
`1 3 2 -1`
	$x^3+3\cdot x^2+2\cdot x-1=0$
	polinónio deve ser definido como um vector

`octave:2> polyderiv(mypoly)`
`ans =`
`3 6 2`
	cálculo da derivada

`octave:3> polyinteg(mypoly)`
`ans =`
`0.25000 1.00000 1.00000 -1.00000 0.00000`
	cálculo do integral

`octave:4> polyval(mypoly,2)` `ans = 23`	cálculo do valor do polinómio (para x=2)

`octave:5> roots(mypoly)`
`ans =`
`-1.66236 + 0.56228i`
`-1.66236 - 0.56228i`
`0.32472 + 0.00000i`
	determinação das raízes do polinómio (neste caso são raízes imaginárias)

Exemplo da utilização de funções estatísticas definidas no Octave

`octave:6> a=[1,1,2 ; 3,5,8 ; 13,21,34 ]`
`a =`
`1 1 2`
`3 5 8`
`13 21 34`
	utilizando como exemplo a matriz *a*.

`octave:7> mean(a)`
`ans =`
`5.6667 9.0000 14.6667`
	calcula a média dos valores. No caso de uma matriz calcula a média de cada coluna.

`octave:8> median(a)`
`ans =`
`  3  5  8`
	calcula a mediana. No caso de uma matriz calcula a mediana de cada coluna

`octave:9> std(a)`
`ans =`
`   6.4291  10.5830  17.0098`
	calcula o desvio padrão dos valores de *a*

`octave:10> cov(a,a)`
`ans =`
`   41.333   68.000  109.333`
`   68.000  112.000  180.000`
`  109.333  180.000  289.333`
	calcula a covariância dos valores entre *x* e *y* (*cov(x,y)*), neste caso entre *a* e *a*.

Para mais informação sobre funções oferecidas pelo Octave deve consultar o manual e/ou analisar a(s) directoria(s) de ficheiros '.m'

3.6 Exemplos de funções

```
function hello
    disp("Hello world!")
endfunction
```

```
# função que grava uma matriz num ficheiro (ler com load_mat)

function ret_mat=save_mat(file_n, mat)

  fid=fopen(file_n, "w");
  sz=size(mat);
  fprintf(fid, "%d %d\n", sz(1), sz(2));

  for i=1:sz(1)
     fprintf(fid, "%f ", mat(i,:));
     fprintf(fid, "\n");
  endfor

  ret_v=fclose(fid);

endfunction
```

```
# função que le uma matriz gravada com o comando save_mat

function ret_mat=load_mat(file_n)

   fid=fopen(file_n, "r");
   size_m=fscanf(fid, "%d ", [1,2]);
   ret_mat=fscanf(fid, "%f ", size_m);
   fclose(fid);
```

```
endfunction
```