Introdução ao Matlab

O Matlab é um ambiente de programação orientado ao cálculo numérico científico e visualização de dados. É composto por várias toolboxes que enriquecem o modelo de programação elementar que tem por base o cálculo matricial.

É uma excelente ferramenta para a prototipagem de algoritmos de processamento de dados. Para maior eficiência, oferece bindings para C/C++.

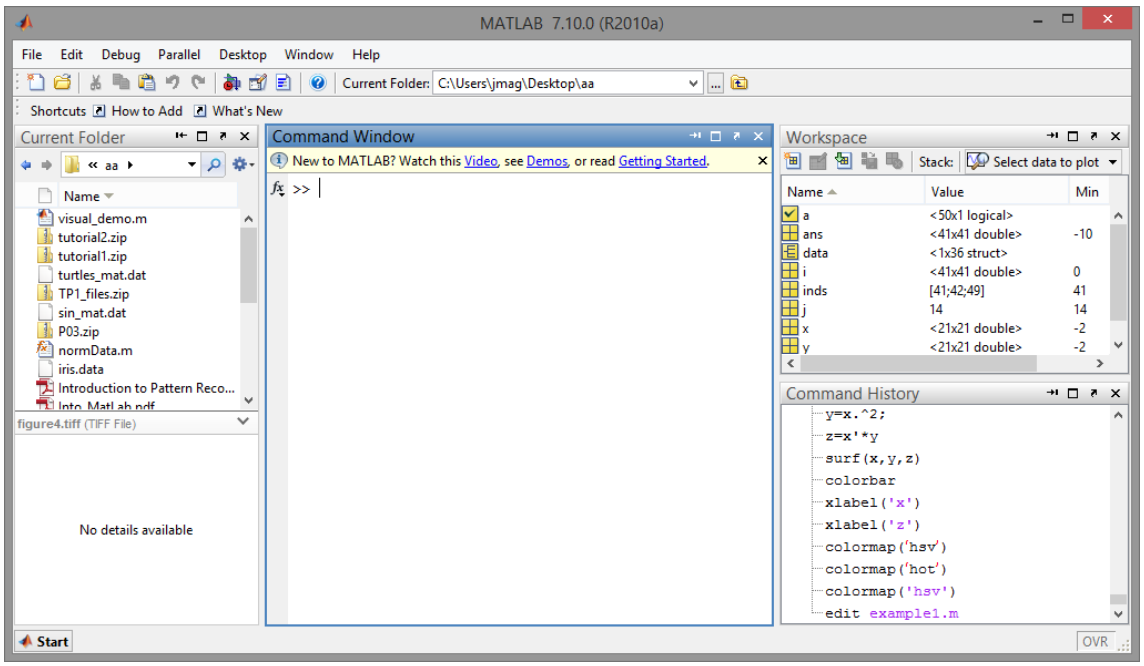
O ambiente de programação Matlab é composto pelas seguintes partes:

- Visualização da pasta actual e dos ficheiros aí presentes;
- Workspace, onde são listadas todas as variáveis globais;
- Janela de comandos interactiva;
- Histórico de comandos com todos os comandos anteriores;

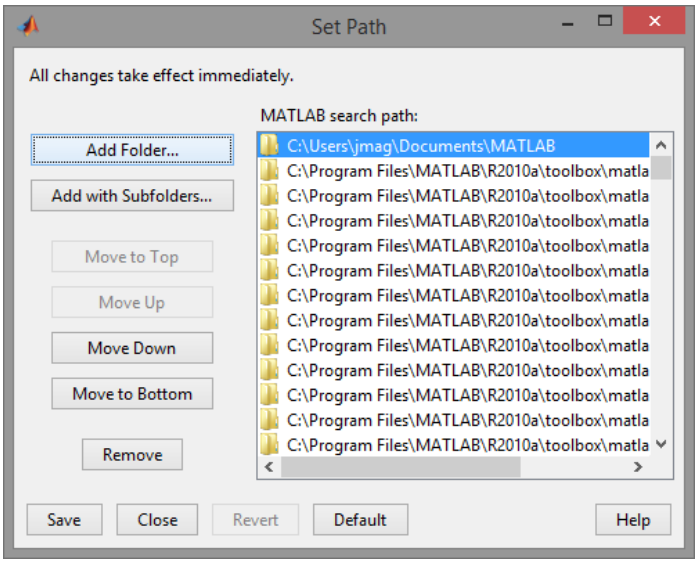
Quando se executa o Matlab num servidor sem X, é possível executar o Matlab apenas em modo consola: `matlab -nodesktop`. Este modo, é particularmente útil quando a simulação leva muito tempo e pretendemos que fique a ser executada em *background*.

Alguns comandos úteis são:

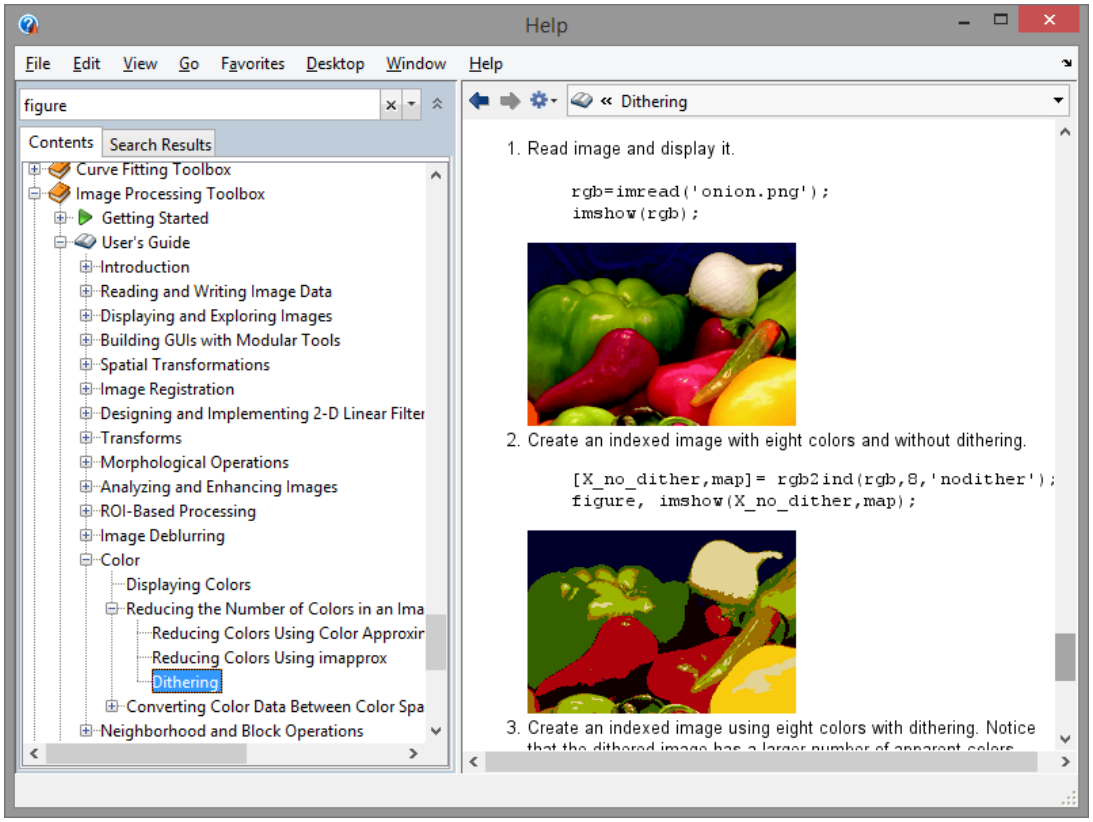
`version`	`verifica versão do Matlab`
`help`	`abre sistema de ajuda`
`cd / ls`	`comandos idênticos aos do SO`
`edit example.m`	`edits or creates a matlab function file or script file`



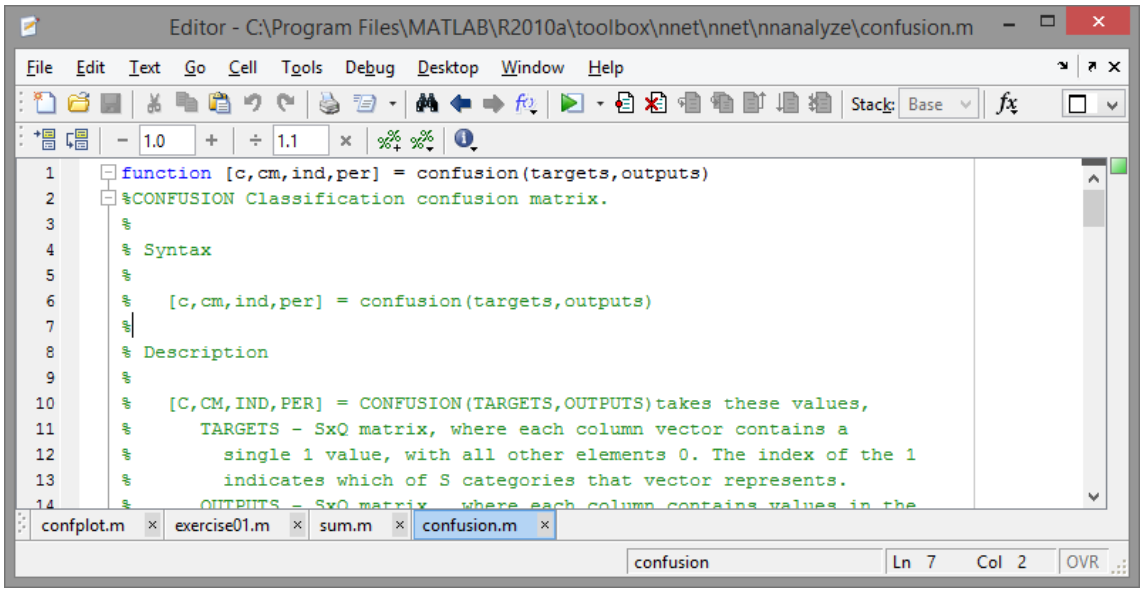
Os *paths* alternativos onde o Matlab procura funções é definido no através do menu *File*, conforme ilustrado na figura seguinte.



O *help* do Matlab oferece uma vasta documentação sobre as diferentes toolboxes e funções que estão disponíveis. Em muitos casos também contêm tutoriais bastante completos sobre algumas matérias. Por fim, existem vários exemplos e demonstradores de algumas toolboxes.



O componente final do ambiente Matlab é o editor de funções e scripts. O editor permite implementar, executar e testar as aplicações/algoritmos. Está disponível as operações habituais de debug como *breakpoints*, *step-by-step* e *watch variables*.



Funções e scripts

Funções e scripts são os elementos de programação em matlab que permitem modularizar um programa com maiores dimensões. Também pode dividir o seu código em módulos em pastas diferentes (o comando `addpath` permite adicionar novas pastas ao *search path* do Matlab).

Uma função é definida como:

```
function [out1,out2,out3] = myfunction(arg1,arg2)
```

O ficheiro deve ter o mesmo nome da função. A passagem de parâmetros é feita por lazy-copy o que pode provocar um consumo de memória muito elevado se houver muitas chamadas a funções (evitar funções recursivas). Em alternativa, as variáveis podem ser definidas como globais.

Em geral as operações com matrizes são bastante eficientes e operações como ciclos for e while não são eficientes. Habitualmente estes ciclos podem ser convertidos para operações matriciais.

Tipos de dados

Matrizes, escalares e strings são os tipos mais comuns no Matlab. Também existem tipos estruturados onde os dados podem ser organizados por campos de diferentes tipos.

Funções e operações úteis

`whos/who`	lista variáveis globais
`clear`	apaga variáveis globais
`a=[1 2 3; 4 5 6]`	define matriz com duas linhas e três colunas
`size(a)`	verifica número de linhas/colunas
`ones(n,m)`	define matriz preenchica com 1s
`zeros(n,m)`	define matriz preenchida com 0s
`eye(n,m)`	define matriz diagonal
`rand(n,m)`	define matriz com números aleatórios
`sum(a)`	soma todas as linhas/colunas de uma matriz
`mean`	calcula a média das colunas de uma matriz
`std`	calcula o desvio padrão das colunas de uma matriz
`plot(x,y)`	constrói gráfico de linhas
`bar`	constrói gráfico de barras
`figure`	cria uma nova figura para um gráfico
`hold on/off`	mantém ativo o último gráfico par se sobrepor mais elementos ao gráfico

Operações matriciais:

`a*b`	multiplicação de matrizes
`a+b`	soma de matrizes
`a-b`	subtração de matrizes
`a.*b`	multiplicação elemento a elemento de matrizes
`a.^b`	todos os elementos de a são elevados à potência do elemento de b correspondente

Replicação de vectores e matrizes:

```
a=[1 2 3]
y=ones(1,10)
b= y'*a          % vector y é repetido por 10 linhas
c= a'*y          % vector y é repetido por 10 colunas
b=repmat(A,2,3)  % replica a matriz A num xadrez de 2-por-3
```

Parte 1: Visualização de Dados

Exercicio 1

Analise cada um dos métodos utilizados nos exemplos abaixo, e documente os métodos convenientemente.

Exemplo 1

```
x=rand(50,1)
a = x<0.7
inds=find(a)
y=x(inds)
x(inds)=0
stem(x)
bar(x)
```

Exemplo 2

```
x=-10:0.5:10;
y=x.^2;
plot(x,y)
xlabel('x')
ylabel('y')
```

Exemplo 3

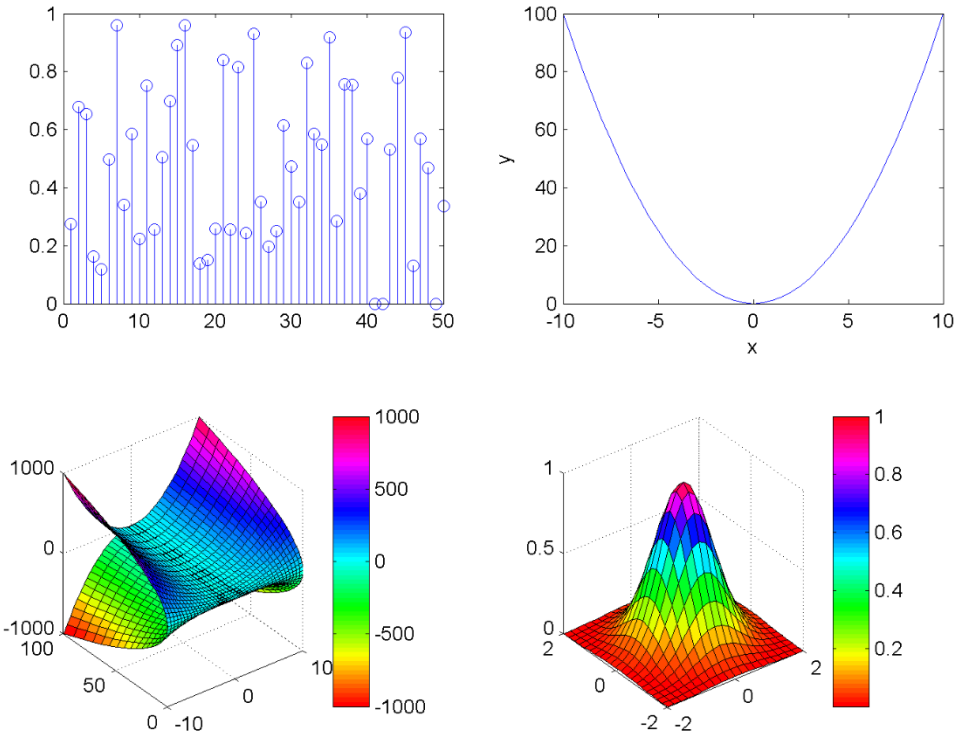
```
x=-10:0.5:10;
y=x.^2;
z=x'*y
surf(x,y,z)
colorbar
colormap('hsv')
```

Exemplo 4

```
[x,y] = meshgrid([-2:0.2:2]);
z = exp(-x.^2-y.^2);
surf(x,y,z)
colorbar
print -dtiff -r200 figure.tiff
```

Exercicio 2

Recorrendo aos métodos apresentados anteriormente e à função `subplot(n,m,i)`, faça um script que crie a seguinte figura:



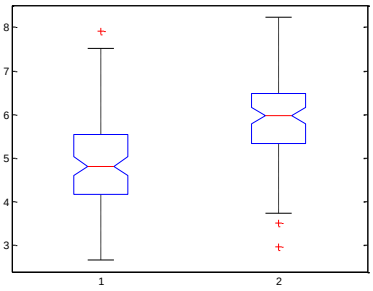
Parte 2: Processamento de Dados

Consider the file “carcinoma.txt” containing gene expression data on tumor and normal tissue samples. Each sample is in one column, with the first line containing the sample label, which indicates if the sample was from a tumor or normal tissue. The first column contains the gene labels.

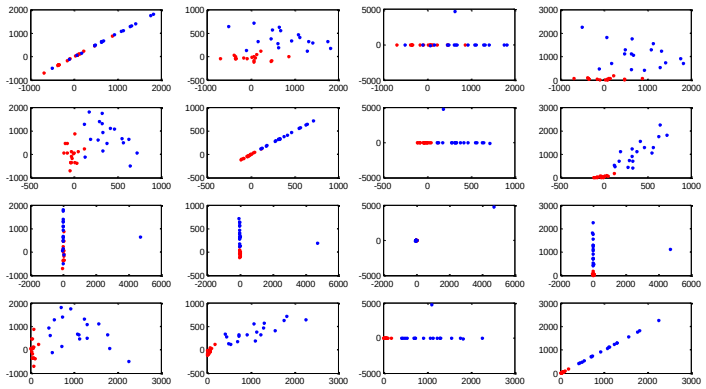
In this exercise, you should:

- Load the data into an appropriate data structure (e.g. a structure vector). The function *loadcarcinoma* (in annex), should be used to load this data file.
- Normalize attributes (subtract the mean, divide by the standard deviation, functions mean and std) and store the standardized values conveniently (e.g. in matrices), separating the gene expression values of the normal tissue from those of tumor tissue. The function *normData* (in annex), should be completed and used to normalize the data.
- Create box plots compare the different (standardized) gene expression ranges (tumor and normal) and choose two genes where tumor and normal tissue differ significantly.

```
x1 = normrnd(5,1,100,1);
x2 = normrnd(6,1,100,1);
boxplot([x1,x2],'notch','on')
```



- Create correlation graphs to compare the different (standardized) genes and choose two genes where tumor and normal tissue differ significantly.



- Compute the t test p-values for the comparison of the expression of each gene across tumor and normal tissue.
- Use the previous plots and p-values to choose the two attributes (gene expression values) that are more promising for separating tumors and normal tissue. Then plot these two attributes in a scatter plot using different colors for normal and tumor samples.

Data from http://genomics-pubs.princeton.edu/oncology/. U. Alon, N. Barkai, D. A. Notterman, K. Gish, S. Ybarra, D. Mack, and A. J. Levine, Broad patterns of gene expression revealed by clustering of tumor and normal colon tissues probed by oligonucleotide arrays Proc. Natl. Acad. Sci. USA, Vol. 96, Issue 12, 6745-6750, June 8, 1999

```
function data=loadcarcinoma(filename)
  fid=fopen(filename,'r');
  %load first line, headers
  lin=fgetl(fid);
  tabixs=[findstr(lin,sprintf('\t')),length(lin)+1]
  data=[];

  %read data labels
  %first is gene/sample
  for f=1:length(tabixs)-1
    label=lin(tabixs(f)+1:tabixs(f+1)-1);
    data(f).label=label;
    data(f).isTumor=length(findstr('T',label))>0;
    data(f).x=[];
  end

  %read attributes
  while ~feof(fid)
    lin=fgetl(fid);
    if ischar(lin)
      tabixs=[findstr(lin,sprintf('\t')),length(lin)+1];
      %first one is the attribute label
      attribIx=length(data(1).x)+1;
      for f=1:length(tabixs)-1
        num=lin(tabixs(f)+1:tabixs(f+1)-1);
        data(f).x(attribIx)=str2num(num);
      end
    end
  end
  fclose(fid);
end
```

```
function [normdata] = normData(data,type);
%
% function to normalize using different rules
%

if nargin == 0
    fprintf('\n ================================================ \n');
    fprintf('  Usage of ''normalize'' is: \n');
    fprintf('>> normalize(data,type)\n');
    fprintf('\n');
    fprintf('>> data:    data array \n');
    fprintf('>> type:       normalization type (''zscore'',''range'', ''abs'')\n');
    fprintf(' ================================================ \n\n');
    return;
end

if nargin<2
    type=0;
end

av=mean(data);
[N V]=size(data);
%%N=dim(1);
%%%col=dim(2);

st= std(data,1);  %argument 1 indicates that divisor at sigmas is N and not N-1;

ra= max(data)-min(data);

%%% To properly standardize the data, the V-dimensional rows must be
%%% converted to the format N x V matrices, which can be efficiently done
%%% with the operation repmat(x, m, n)

avr= repmat(av, N, 1);
str= repmat(st, N, 1);
rar= repmat(ra, N, 1);

%%% We now use the Matlab operation of the entry-wise division of arrays

if strcmp(type,'range')

    normdata= (data-avr)./rar;

elseif strcmp(type(1,:) , 'zscore' )

    normdata= (data-avr)./str;

    diff = max(data) - min(data);
    for i=1:columns
        if diff(i) > 0
            tmp = (data(:,i) - m(i)) ./ diff(i);
            normdata = [normdata,tmp];
        else
            normdata = [normdata,zeros(lines,1)];
        end
    end



%%%%%% TO DO by students %%%%%
%%%% elseif strcmp(type(1,:) , 'abs')


else
    fprintf(type); fprintf(' it is not a valid rule for normalization;\n');
end
```