UNIVERSIDADE DO MINHO
ESCOLA DE ENGENHARIA

DEPARTAMENTO DE PRODUÇÃO E SISTEMAS

MATLAB

Uma ferramenta para Engenharia

Noções Gerais

Celina Pinto Leão

Novembro 1999

Introdução

O nome MatLab deriva de MATrix LABoratory. É um sistema interactivo, baseado na representação matricial para a resolução de problemas, no âmbito científico e de engenharia, e na sua visualização gráfica. Possui um conjunto de "Toolboxes" (Sistemas de Controlo, Optimização, Redes Neuronais Artificiais) que permitem resolver classes particulares de problemas. Tem a possibilidade de chamar rotinas desenvolvidas em C ou em Fortran.

Esta sebenta serve como ajuda a quem contacta com o MatLab pela primeira vez. Não tem a intenção de substituir o 'User's Guide' e o 'Reference Guide dot MatLab'.

O aluno deverá utilizar o comando `help` para obter informação detalhada. Por exemplo, o comando `help eig` dá informação acerca da função no cálculo de valores e vectores próprios. O comando `help` pode também ser usado sozinho, disponibilizando uma lista dos tópicos disponíveis.

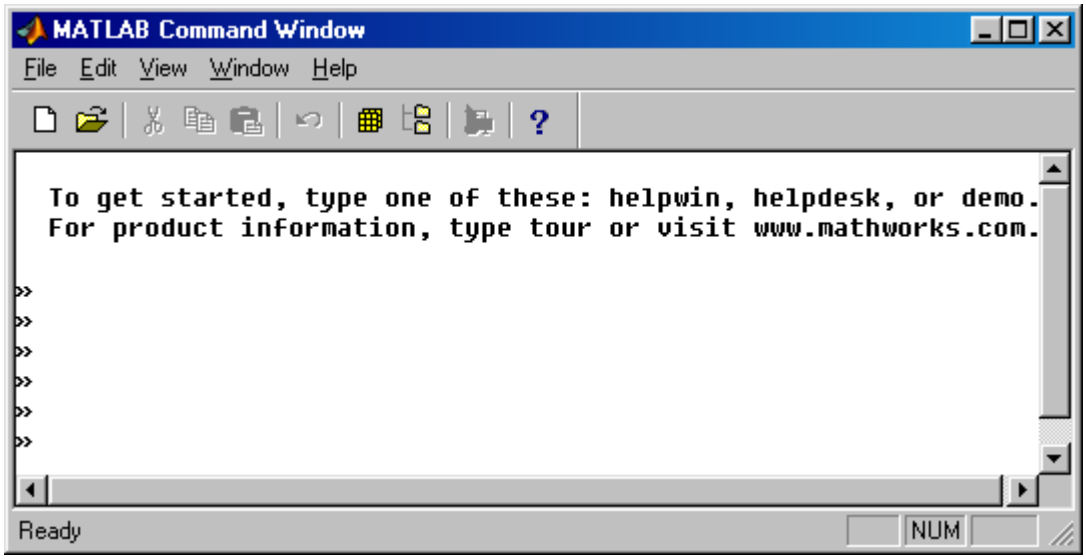
Alguns exercícios são apresentados ao leitor, ao longo do texto. Estes estão assinalados por ➢.

MatLab, licença de The MathWorks, Inc., 24 Prime Park Way, Natick, MA 01760 – 1500 USA, (508)653-1415, Fax: (508)653-2997, Email: info@mathworks.com.

1. Acesso ao MatLab

O acesso ao MatLab, no nosso laboratório, é feito através do menu ou 'clickando' no icon respectivo.

O ambiente do MatLab tem o seguinte aspecto:



É a seguir ao "prompt", **»**, que cada instrução deve ser introduzida. As instruções devem ser inseridas em letras minúsculas. As variáveis introduzidas em minúsculas e em maiúsculas correspondem a variáveis diferentes.

Os comandos mais vulgares dos sistemas operativos Unix e DOS podem ser utilizados. Os comandos do DOS ficam acessíveis através do símbolo `!` seguido da instrução pretendida;

```
» !rename aaa.txt bbb.m;
```

A seguir, uma pequena lista dos comandos do MatLab mais vulgares;

`who`	lista o nome das variáveis em memória;
`pwd`	apresenta o caminho do directório corrente;
`cd xpto`	desloca-se para o directório xpto quando o directório corrente contém o directório xpto;
`cd ..`	desloca-se do directório presente para o directório precedente;
`dir`	lista os ficheiros do corrente directório;
`clear`	apaga todas as variáveis que estão na memória.

2. Ficheiros do MatLab

Os ficheiros com extensão `mat`, armazenam as variáveis de sessões em MatLab;

```
» save sessao
```

Como após a saída do MatLab, todas as variáveis são perdidas, o comando `save` pode ser utilizado para gravar os valores de todas essas variáveis num ficheiro (de preferência numa disquete) com o nome `sessao.mat`. Para aceder mais tarde, às variáveis dessa sessão, fazer;

```
» load sessao
```

Os ficheiros com extensão `m`, são ficheiros que podem conter instruções em MatLab e funções. São interpretados pelo MatLab.

3. Elementos Básicos

As expressões que se escreve na linha de comando do MatLab são interpretadas e calculadas. Os comandos no MatLab são da forma

`variável = expressão`, ou simplesmente
`expressão`

As expressões são usualmente compostas por operadores, funções ou/e nome de variáveis. O cálculo de uma expressão produz uma matriz, que é escrita no ecran e atribuída a uma variável para uso futuro. Se o nome da variável e o sinal = são omitidos, a resposta é guardada numa variável com o nome `ans` que é automaticamente criada. O sinal de igualdade, =, é usado para atribuir valores a variáveis.

Cada comando é normalmente terminado com a tecla ↵ (return). No entanto, um comando pode continuar nas linhas seguintes, sempre seguindo da tecla ↵. Também, mais do que um comando pode ser introduzido numa única linha, se forem separados por vírgulas ou ponto e vírgulas.

Se o último caracter numa linha de comando for um ponto e vírgula, a sua impressão é omitida, no entanto a sua execução é efectuada. O símbolo `;` é utilizado quando a impressão de resultados intermédios não é desejável.

Usa-se uma notação convencional para realizar operações aritméticas. As potências são executadas antes das divisões e das multiplicações, as quais são realizadas antes das adições e das subtracções.

No MatLab, tal como em C ou em Fortran, todas as variáveis devem ter um valor (numérico ou uma 'string' de caracteres). Todos as operações aritméticas são realizadas em dupla precisão (cerca de 16 dígitos decimais), embora os resultados sejam normalmente apresentados numa forma reduzida.

- **Experimente os seguintes exemplos**:

```
» 2+3
» 3*4,4^2
```

```
» 2+3*4^2
» x=3,y=x^2
» y/x
» z=2*ans,ans
» a=sqrt(2)
» format long,b=sqrt(2)
» a-b
» format short
```

Por defeito, a apresentação dos resultados é com 4 casas decimais. Outros possíveis comandos ,

`format short e`	notação científica com 4 casas decimais
`format long e`	notação científica com 15 casas decimais
`format hex`	formato hexadecimal
`format bank`	dollars e cents

4. Matrizes

MatLab trabalha essencialmente com matrizes rectangulares com a possibilidade de entrada de valores complexos. Todas as variáveis representam matrizes sem que estas tenham que ser dimensionadas. Em particular, matrizes 1 por 1 são interpretadas como escalares e matrizes com uma só linha ou coluna interpretadas como vectores.

Matrizes podem ser introduzidas de várias maneiras:

- entrada por uma lista explicita de elementos,
- gerada por construção de funções ou comandos,
- criada a partir de um editor (`m` – files),
- editada a partir de ficheiros ou aplicações externas (ver 'User's Guide').

Por exemplo, ambos os comandos

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

criam uma matriz 3 por 3 que é guardada na variável `A`.

➢ **Experimente**.

As matrizes podem vir de ficheiros que tenham a extensão "`.m`", por exemplo se tivermos o ficheiro `exemplo1.m` com a seguinte informação:

```
A=[1 2 3
   4 5 6
   7 8 9]
```

Quando estivermos na janela de comandos do MatLab, se executarmos o seguinte comando:

```
» exemplo1;
```

O resultado é:

```
A =

     1     2     3
     4     5     6
     7     8     9
```

Os elementos numa linha da matriz podem ser separados por vírgula como por espaço. Quando se escreve um número na forma exponencial (por exemplo: 2.3e-9), espaços em branco não devem ser utilizados.

MatLab permite números complexos em todas as operações e funções. Duas maneiras de entrada de matrizes complexas são:

```
» A=[1 2;3 4] + i*[5 6;7 8]
» A=[1+5i 2+6i;3+7i 4+8i]
```

Quando escrever números complexos (por exemplo: 2+6i) numa matriz, não utilizar espaços em branco. Tanto *i* como *j* podem ser utilizados como unidade imaginária. Pode-se gerar novas unidades imaginárias, por exemplo `ii = sqrt(-1)`.

Existem funções já definidas, por exemplo `rand`, `magic`, e `hilb`, oferecendo uma maneira fácil de criar matrizes. O comando `rand(n)` cria uma matriz $n \times n$ com valores aleatoriamente distribuídos uniformemente entre 0 e 1, enquanto `rand(m,n)` cria uma matriz $m \times n$. `magic(n)` cria uma matriz quadrada $n \times n$ mágica (linhas, colunas e diagonais têm soma iguais); `hilb(n)` cria a matriz $n \times n$ de Hilbert. (m e n são números inteiros positivos).

As entradas individuais de matrizes e de vectores podem ser referenciadas usando índices dentro de parêntesis. Por exemplo, `A(2,3)` faz referencia ao valor da segunda linha terceira coluna da matriz `A` e `x(3)` representa a terceira coordenada do vector `x`.

- **Experimente**.

Uma matriz ou um vector só aceita números inteiros positivos para índice.

Caso se pretenda criar um vector com elementos entre 0 e 20 com espaçamento constante e igual a 2, fazer

```
» t=0:2:20
     t=
         0   2   4   6   8   10   12   14   16 18   20
```

4.1. Operações Matriciais

- transposição → B=A';
- adição e subtracção → C=A+B;, D=A-B;
- multiplicação → C=A*B;

- divisão matricial → x=A\b; (solução de A*x=b)
  → x=A/b; (solução de x*A=b)
- cálculo da inversa → B=**inv**(A);
- cálculo do determinante → D=**det**(A);

Estas operações matriciais podem também serem aplicadas a escalares.

As operações matriciais só são possíveis para matrizes de igual dimensão. Caso isso não acontecer, uma mensagem de erro aparecera, excepto para o caso de operações escalar-matriz.

O ponto antes do sinal da operação a efectuar, é utilizado quando a operação é para ser efectuada elemento a elemento. Por exemplo, `[1,2,3,4].*[1,2,3,4]` dá o mesmo resultado que `[1,2,3,4].^2`.

- **Experimente**.
- **Considere** $A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix}$ e $x=\begin{bmatrix}-1\\ 0\\ 2\end{bmatrix}$**. Calcule:**
  - matriz B, a transposta de A.
  - soma de A+B.
  - o vector y=x-1.
  - x'*y
  - x*y'
  - y*x'
- **Qual a solução do sistema** $\begin{cases}x_1-2x_2+3x_3=1\\ 4x_1-5x_2-6x_3=-2\\ 4x_1+2x_2-x_3=-1\end{cases}$**?**

4.2. Funções Elementares

4.2.1. Trignométricas

`sin`	seno
`cos`	cosseno
`tan`	tangente
`asin`	arcosseno
`acos`	arcocosseno
`atan`	arcotangente
`sinh`	seno hiperbólico
`cosh`	cosseno hiperbólico
`tanh`	tangente hiperbólica
`asinh`	arcosseno hiperbólico
`acosh`	arcocosseno hiperbólico
`atanh`	arcotangente hiperbólica

4.2.2. Matemáticas elementares

`abs`	valor absoluto ou amplitude do complexo
`angle`	ângulo de fase
`sqrt`	raiz quadrada
`real`	parte real de um complexo
`imag`	parte imaginária de um complexo
`sign`	função sinal
`exp`	exponencial base e
`log`	logaritmo natural
`log10`	logaritmo base 10

4.2.3. Matrizes especiais

As funções mais comuns para a construção de matrizes especiais são

`eye`	matriz identidade
`zeros`	matriz nula
`ones`	matriz de uns
`diag`	cria ou extrai diagonais
`triu`	parte triangular superior de uma matriz
`tril`	parte triangular inferior de uma matriz
`rand`	matriz gerada aleatóriamente
`hilb`	matriz de Hilbert
`magic`	matriz quadrada mágica
`toeplitz`	ver `help toeplitz`
`bessel`	função de bessel
`gama`	função gama completa e incompleta
`rat`	aproximação racional
`erf`	função erro
`erfinv`	função erro inversa
`ellipk`	integral elíptico do 1º tipo
`ellipj`	função elíptica jacobiana

`compan` função geradora de uma matriz cujo polinómio característico apresenta como coeficientes o vector p.

Por exemplo, considere o polinómio x^3-7x+6. O vector com os coeficientes deste polinómio é:

```
» p=[1 0 -7 6]
```

A matriz gerada associada a este polinómio é obtida usando a seguinte função:

```
» A=compan(p)
A =
     0     7    -6
     1     0     0
     0     1     0
```

Os valores próprios da matriz `A` são as raízes do polinómio

```
» eig(A)
ans=
     -3.000
     2.000
     1.000
```

Se `x` é um vector, `diag(x)` é uma matriz diagonal com os elementos de `x` na diagonal; se `A` é uma matriz quadrada, então `diag(A)` é um vector cujos elementos são a diagonal de `A`.

- **Qual será o valor de diag(diag(A))? Experimente**.

Matrizes podem ser construídas em blocos. Por exemplo, se `A` for uma matriz 3x3, então com o comando `B=[A, zeros(3,2); zeros(2,3), eye(2)]` será construída uma matriz `B` de dimensão 5x5.

- **Experimente**.

4.2.4. outras funções

`eig`	valores próprios de uma matriz
`chol`	factorização Cholesky
`inv`	inversa de uma matriz
`lu`	decomposição LU
`qr`	decomposição QR
`det`	determinante
`norm`	norma de uma matriz
`lu`	decomposição LU
`polyfit`	ajuste de um polinómio de grau n a partir de dados (x,y)
`polyval`	valor calculado pelo polinómio de ajuste
`interp1`	interpolação a uma dimensão
`interp2`	interpolação a 2 dimensões
`spline`	interpolação por splines cúbicas

- **Calcule o vector y com os valores próprios da matriz A**. **O que acontece se usarmos o comando**

» [U,D]=eig(A)

5. Análise de dados em coluna

Esta secção apresenta uma introdução à análise de dados usando o MatLab e descreve algumas ferramentas elementares de estatística.

`max`	valor máximo
`min`	valor mínimo

`mean`	valor médio
`median`	valor mediano
`std`	desvio padrão
`sort`	ordenação
`sum`	somatório
`prod`	produto
`cumsum`	somatório dos elementos cumulativos
`cumprod`	produto dos elementos cumulativos
`diff`	derivadas aproximadas
`hist`	histograma
`corrcoef`	coeficiente de correlação
`cov`	matriz covariância

- **Exemplo: supondo a matriz A,**

```
» A=[9 8 4;1 6 5;3 2 7]
» m=max(A);
» mv=mean(A);
» s=sort(A)
```

6. Polinómios

Os polinómios são representados no MatLab como vectores contendo os coeficientes ordenados por ordem decrescente de potência. Por exemplo, polinómio característico da matriz `A`:

```
» A=[1 2 3;4 5 6;7 8 0];
```

é obtido por

```
»p=poly(a)

p=
   1 -6 -72 -27
```

Que é a representação no MatLab do polinómio $x^3-6x^2-72x-27$. As raízes do polinómio, que são também os valores próprios de `A`:

```
»r=roots(p)
r=
   12.1229
   -5.7345
   -0.3884
```

Os valores próprios de `A` podem também ser obtidos por:

```
»p=eig(a);
```

- **Experimente**.

7. Integração Numérica

As funções matemáticas são introduzidas no MatLab como funções matemáticas e não como matrizes. Por exemplo a função $humps(x) = \frac{1}{(x-0.3)^2+0.01} + \frac{1}{(x-0.9)^2+0.04} - 6$ pode estar disponível no MatLab criando um ficheiro com o nome `humps.m` com o seguinte conteúdo

```
function y=humps(x)
y=1/((x-0.3).^2+0.01)+1//x-0.9).^2+0.04)-6;
```

As funções de integração são as funções `quad` e `quad8`.
O integral entre 0 e 1, da função `humps` é obtido com o seguinte comando:

```
»q=quad('humps',0,1)

q=
   29.8534
```

Que foi obtido utilizando a regra de Simpson adaptativa.

8. Resolução de equações diferenciais

As aplicações no MatLab para a resolução de equações diferenciais ordinárias são:

`ode23`	método de Runge-Kutta de 2ª/3ª ordem
`ode45`	método de Runge-Kutta de 4ª/5ª ordem

Considerando a equação diferencial de Segunda ordem $\frac{d^2x}{dt^2} + (x^2 - 1)\frac{dx}{dt} + x = 0$, equação de Van der Pol.

Podemos escrever esta equação como um sistema de equações diferenciais de 1ª ordem:

$$\begin{cases} \frac{dx_1}{dt} = x_1(1 - x_1^2) - x_2 \\ \frac{dx_2}{dt} = x_2 \end{cases}$$

- **Resolução:** criação do ficheiro `vdpol.m` cujo conteúdo é o sistema das equações diferenciais.

```
function xdot=vdpol(t,x)
xdot(1)=x(1)*(1-x(2).^2)-x(2);
xdot(2)=x(1);
xdot=xdot';
```

Para a resolução do sistema de equações diferenciais definida em `vdpol.m`, entre o intervalo `0≤t≤50`, na linha de comandos do MatLab, fazer

```
»x0=[0 0.25];
»tspan=[0 50];
»[t,x]=ode23('vdpol',tspan,x0);
```

Para a visualização gráfica, fazer

```
»plot(t,x);
```

No gráfico aparecem os valores de `x1` e de `x2` que equivalem ao `x` em função de `t`. Ou,

```
»options=odeset('RelTol',1e-4)
»[t,x]=ode23('vdplo',tspan,x0,options);
```

Alguns dos parâmetros da função odeset mais utilizados,

`AbsTol`	Tolerância absoluta
`InitialStep`	Passo inicial
`MaxStep`	Passo máximo
`RelTol`	Tolerância relativa

9. Programação em MatLab

9.1. Procedimento de utilização de um ficheiro `m`

Criação do ficheiro `m` com um editor de texto ou utilizando o editor do MatLab

```
function c=myfile(a,b)
c=sqrt((a.^2)+b.^2));
```

Chamar o ficheiro `m` na linha de comandos do MatLab ou a partir de outro ficheiro `m`

```
»a=7.5;
»b=3.342;
»c=myfile(a,b)
»c=
     8.2109
```

- **Experimente.**

Existem dois tipos de ficheiros `m`:

1. ficheiros `m` – funções;

2. ficheiros `m` – scripts.

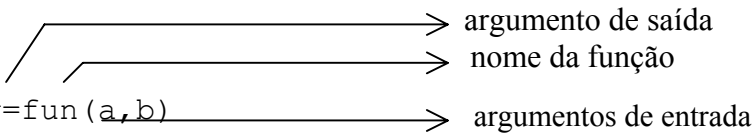
Os ficheiros do primeiro tipo aceitam argumentos de entrada e retornam argumentos de saída. Por omissão, as variáveis internas são locais para a função e são utilizadas como uma extensão ao MatLab em aplicações específicas.
Os ficheiros do segundo tipo não aceitam argumentos de entrada e saída pois operam com dados do espaço de trabalho. São importantes para automatizar uma série de passos que são necessários realizar algumas vezes.

9.2. Anatomia de um ficheiro `m` - função

Cada linha de um ficheiro `m` – função tem o seu significado, a saber:

- linha de definição da função;
- linha de *help* da função;
- linhas de corpo da função;
- outras linhas de comentário.

➢ Exemplo:




```
function y=fun(a,b)
%fun(a,b) - Soma de a+b
y=a+b;
%fim da funcao
```

Existe a possibilidade de existirem num ficheiro `m`, várias funções, sendo cada uma delas subfunções da principal. A função principal deve ser a primeira a ser apresentada,

```
function [avg,med]=newstats(u)
%encontra a media e a mediana com funcoes internas
n=length(u);
avg=mean(u,n);
med=median(u,n);

function a=mean(v,n)
a=sum(v)/n;

function m=median(v,n)
w=sort(v);
if rem(n,2)==1
m=(w(n/2)+w(n/2+1))/2
end
```

9.3. Operadores

- Operadores aritméticos

- adição (+) e subtracção (-)
- transposta ('), potência (^)
- multiplicação (*); divisão à esquerda(\); divisão à direita (/)
- operadores : e .

- Operadores relacionais
  - < menor que
  - <= menor ou igual a
  - > maior que
  - >= maior ou igual a
  - ==igual a
  - ~= diferente de

9.4. Estruturas de controlo de fluxo

Existem quatro tipo de estruturas dependendo do tipo de instruções a efectuar;

1. `If` com `else` e `elseif` – executa um grupo de instruções baseado em alguma condição lógica;
2. `swich` com `case` e `otherwise` – executa diferentes grupos de instruções dependendo do valor de alguma condição lógica;
3. `while` – executa um grupo de instruções num número infinito de vezes, baseado numa condição lógica;
4. `for` – executa um conjunto de instruções num número finito de vezes.

9.4.1. Estrutura `if, else` e `elseif`

Expressão geral do comando:

```
if expressão_lógica
  instruções
end
```

- **Exemplo:**

```
if n<0
disp('A entrada deve ter um valor positivo')
elseif rem(n,2)==0
a=n/2;
else
a=(n+1)/2;
end
```

9.4.2. Estrutura `switch`

Expressão geral do comando:

```
switch expressão (escalar ou string)
case valor1
instruções
```

```
   case valor2
   instruções
   ...
   otherwise
   instruções
   end
```

➢ **Exemplo:**

```
switch input_num
case -1
disp('valor negativo um')
case 0
disp('valor zero')
case 1
disp('valor positivo um')
otherwise
disp('outro valor')
end
```

9.4.3. Estrutura `while`

Expressão geral do comando:

```
   while expressão
   instruções
   end
```

As instruções serão repetidamente executadas enquanto a expressão for verdadeira.

➢ **Exemplo:**

```
n=1;
while prod(1:n)<1e100
n=n+1;
end
```

9.4.4. Estrutura `for`

Expressão geral do comando:

```
   for índice=início:incremento:fim
   instruções
   end
```

➢ **Exemplo:**

```
for i=1:m
   for j=1:n
   H(i,j)=1/(i+j-1);
   end
end
H
```

9.5. Avaliação de variáveis string

A função `eval` avalia uma *string* que contém uma expressão em MatLab.

```
eval('string')
```

- **Exemplos:**

```
t='1/(i+j-1)'
for i=1:n
   for j=1:n
   a(i,j)=eval(t)
   end
end

eval('t=clock')
```

A função `feval` difere da função `eval` no sentido em que esta executa a função cujo nome é uma *string*,

- **Exemplo:**

```
fun=['sin';'cos';'log'];
k=input('Escolha o número da função:');
x=input('Introduza um valor:');
feval(fun(k,:),x)
```

9.6. Introdução de entradas a partir do teclado

```
n=input('palavra')
```

A função apresenta a palavra e espera que seja introduzido um valor.

`name=input('palavra','s')` – entrada de uma variável *string*.

- **Exemplo:**

```
i=1;
j=2;
funcao=input('Funcao:','s');
eval(funcao)
```

10. Tratamento Gráfico

MatLab pode produzir gráficos a 2 dimensões, gráficos de curvas a 3 dimensões, gráficos de superfície a 3 dimensões, etc. Uma introdução para cada um deles é feita a seguir.

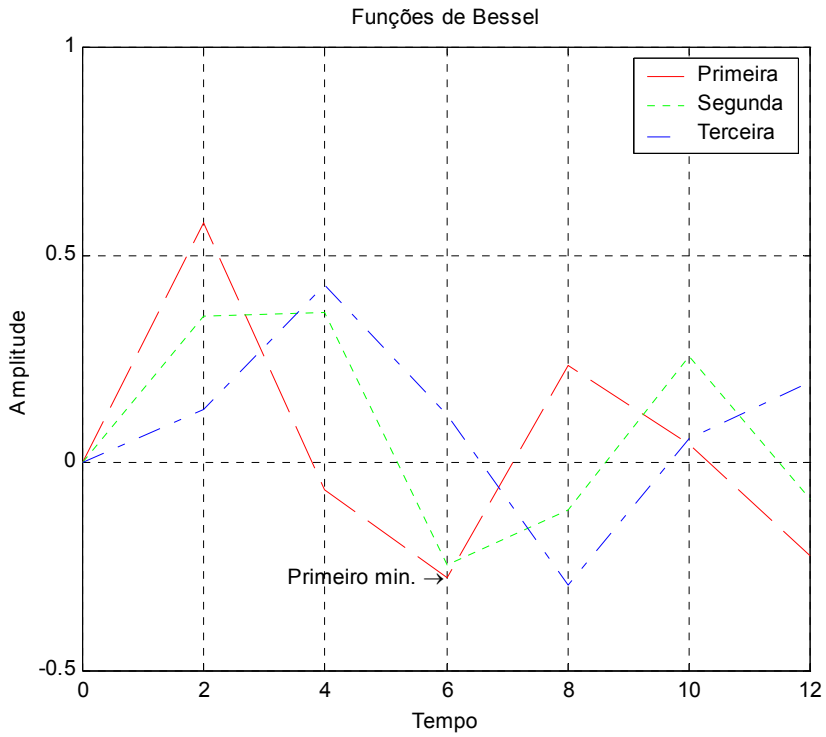
Passos para a criação de gráfico a 2D:

- preparar os dados;
- seleccionar a janela e a posição do gráfico dentro da janela;
- chamar a função que faz o gráfico;
- seleccionar as características das linhas e dos símbolos;
- seleccionar os limites dos eixos, tamanho dos símbolos, e grelhas;
- introduzir o título, legendas dos eixos e texto no gráfico;
- imprimir o gráfico.

➢ **Experimente o seguinte exemplo:**

```
» x=0:2:12;
» y1=bessel(1,x);
» y2=bessel(2,x);
» y3=bessel(3,x);
» figure(1)
» h=plot(x,y1,x,y2,x,y3);
» set(h,'LineWidth',2,{'LineStyle'},{'--';':';'-.'})
» set(h,{'Color'},{'r';'g';'b'})
» axis([0 12 -0.5 1])
» grid on
» xlabel('Tempo')
» ylabel('Amplitude')
» legend(h,'Primeira','Segunda','Terceira')
» title('Funções de Bessel')
» [y,ix]=min(y1);
» text(x(ix),y,'Primeiro min. \rightarrow',...
'HorizontalAlignment','right')
```

No final deverá obter um gráfico semelhante ao apresentado a seguir:



Aos gráficos pode ser dado um título, legendado, e colocado um texto num determinado ponto no gráfico com um *string* como argumento, com os seguintes comandos,

`title('Título')`	adiciona um título ao gráfico,
`xlabel('xx')`	adiciona uma legenda ao eixo do xx',
`ylabel('yy')`	adiciona uma legenda ao eixo do yy',
`zlabel('zz')`	adiciona uma legenda ao eixo zz',
`legend(h,'Primeira','Segunda','Terceira')`	adiciona uma legenda ao gráfico existente,
`text(xvalor,yvalor,'texto')`	apresenta texto numa posição especificada,
`gtext('texto 1')`	coloca texto no gráfico usando o rato

Por defeito, a escala dos eixos é feita automaticamente. No entanto, esta pode ser alterada através de,

`axis([`x_{min}`,`x_{max}`,`y_{min}`,`y_{max}`])`	eixos nos limites especificados
`axis auto`	volta ao escalonamento automatico
`v=axis`	escala no vector v
`axis square`	mesma escala para ambos os eixos
`axis equal`	mesma escala e marcas pequenas para ambos os eixos
`axis off`	sem escala
`axis on`	com escala

Duas maneiras de representar várias funções num só gráfico,

```
» x=0:.01:2*pi;y1=sin(x);y2=sin(2*x);y(3)=sin(4*x);
» plot(x,y1,x,y2,x,y3)
```

e, formando uma matriz Y cujas colunas são as funções a representar graficamente,

```
» x=0:.01:2*pi;Y=[sin(x)',sin(2*x)',sin(4*x)'];
» plot(x,Y)
```

- **Experimente o seguinte exemplo:**

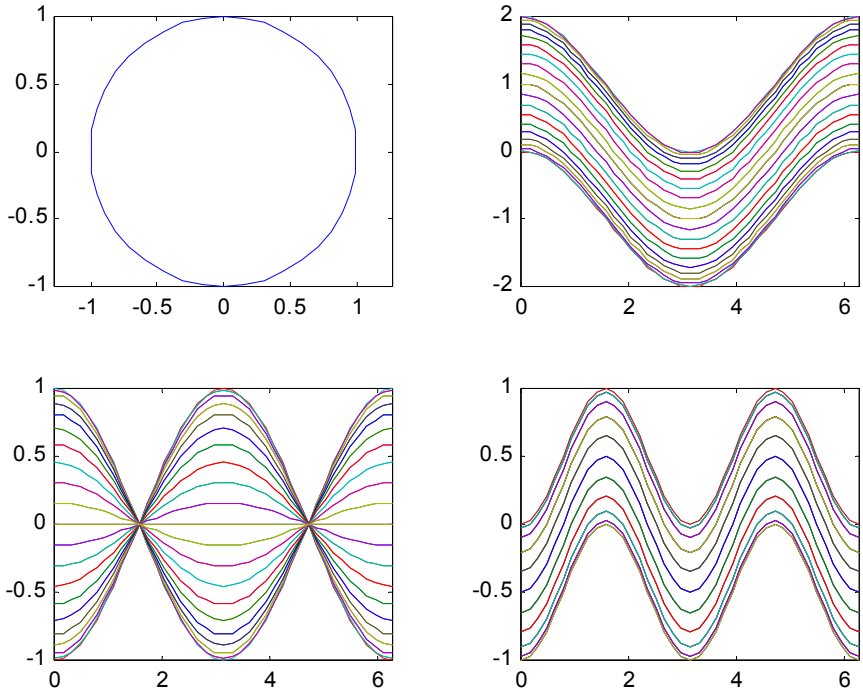
```
» t=0:.01:2*pi;y=sin(t); plot(x,y);
» xlabel('t=0 até 2\pi','FontSize',10)
» ylabel('sen(t)','FontSize',10)
» title('\it{Valores de sen(t) de zero até 2{\pi}}')
```

10.1. Vários gráficos numa única janela

- **Experimente o seguinte exemplo:**

```
» t=0:pi/20:2*pi;
» [x,y]=meshgrid(t);
» subplot(2,2,1),plot(sin(t),cos(t))
» axis equal
» z=sin(x)+cos(y);
» subplot(2,2,2),plot(t,z)
» axis([0 2*pi,-2 2])
» z=sin(x).*cos(y);
» subplot(2,2,3),plot(t,z)
» axis([0 2*pi -1 1])
» z=sin(x).^2-cos(y).^2;
» subplot(2,2,4),plot(t,z)
» axis([0 2*pi -1 1])
```

Seguindo estas instruções obtêm-se o seguinte gráfico:



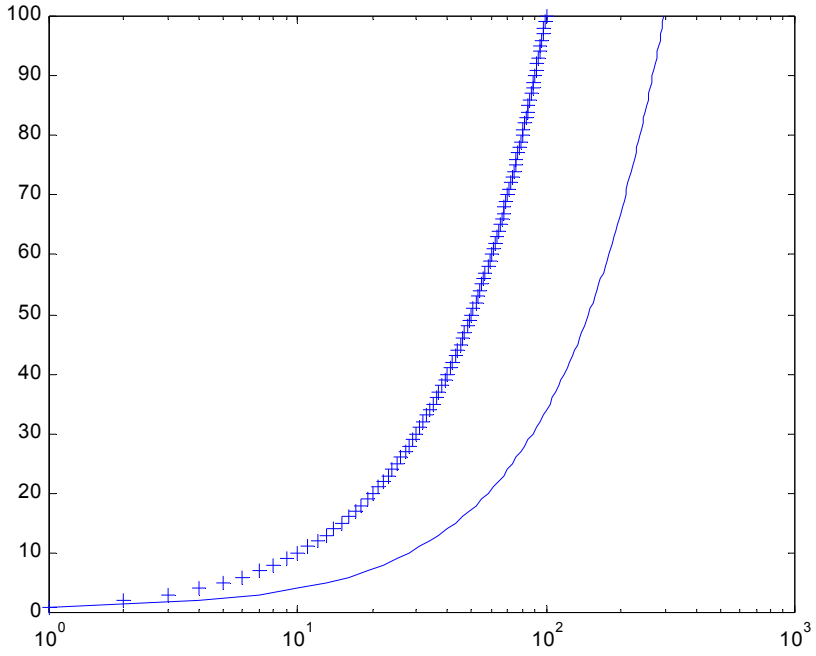
10.2. Adição de gráficos a gráficos já criados

O comando `hold`, fixa o gráfico inicialmente apresentado de modo a que gráficos subsequentes possam ficar sobrepostos.

➢ **Experimente o seguinte exemplo:**

```
» semilogx(1:100,'+')
» hold on
» plot(1:3:300,1:100,'—')
» hold off
```

Resultado final:



10.3. Funções Gráficas Elementares

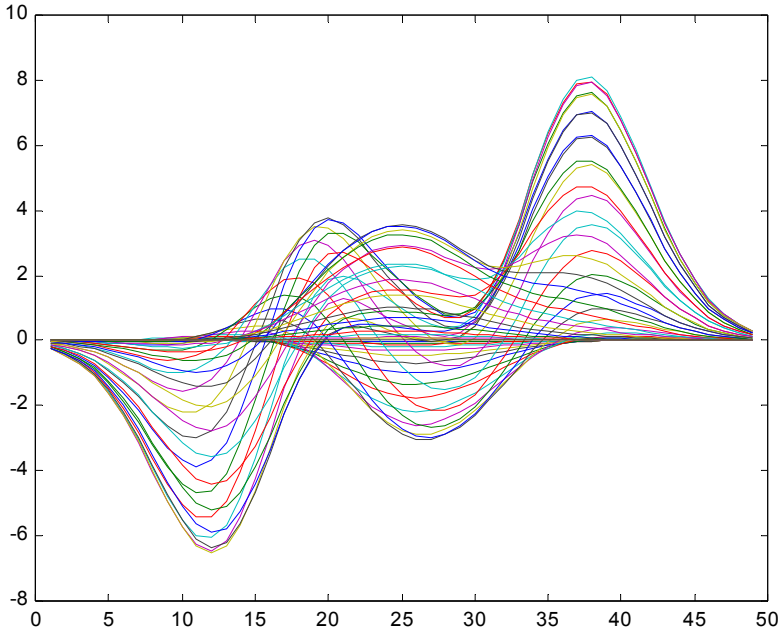
`plot`	gráfico com escalas lineares em ambos os eixos
`loglog`	gráfico com escalas logarítmicas em ambos os eixos
`semilogx`	gráfico com escala logarítmica no eixo dos xx' e escala linear no eixo do yy'
`semilogy`	gráfico com escala logarítmica no eixo yy' e escala linear no eixo xx'
`plotyy`	gráfico com escalas lineares e com dois eixos yy', um do lado esquerdo e outro do lado direito.

10.4. Gráficos de matrizes

`plot(y)`	desenha um gráfico com o eixo yy' correspondente ao vector y e o eixo xx' com os pontos 1:m, sendo m o número de elementos do vector y;
`plot(z)`	(matriz z) desenha um gráfico com as várias colunas da matriz z. O eixo xx' corresponde aos pontos 1:m, sendo m o número de linhas da matriz z.

➢ **Exemplo:**

```
» z=peaks;
» plot(z)
```

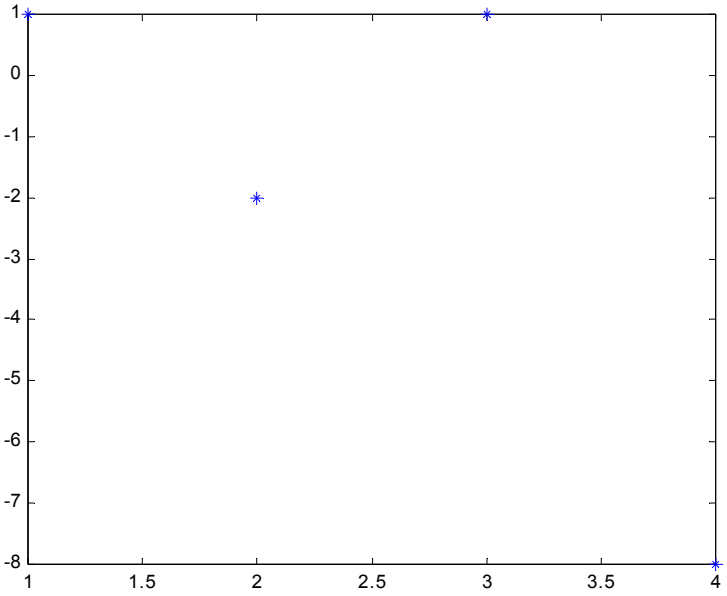


10.5. Gráficos de números complexos

`plot(z)` (z complexo) é equivalente a fazer
`plot(real(z),imag(z))`

➢ **Exemplo:**

```
» z=[1+i 2-2i 3+i 4-8i];
» plot(z,'*')
```



10.6. Controlo dos Gráficos

```
plot(x,y,'cor_estilodelinha_marca')
```

Cor

`c` cyan, `m` magenta, `y` amarelo, `r` vermelho, `g` verde, `b` azul, `w` branco, `k` preto

Estilo de linhas

`-` sólida, `--` tracejada, `:` pontos, `-.` traço-ponto, `none` nenhuma

Marcas

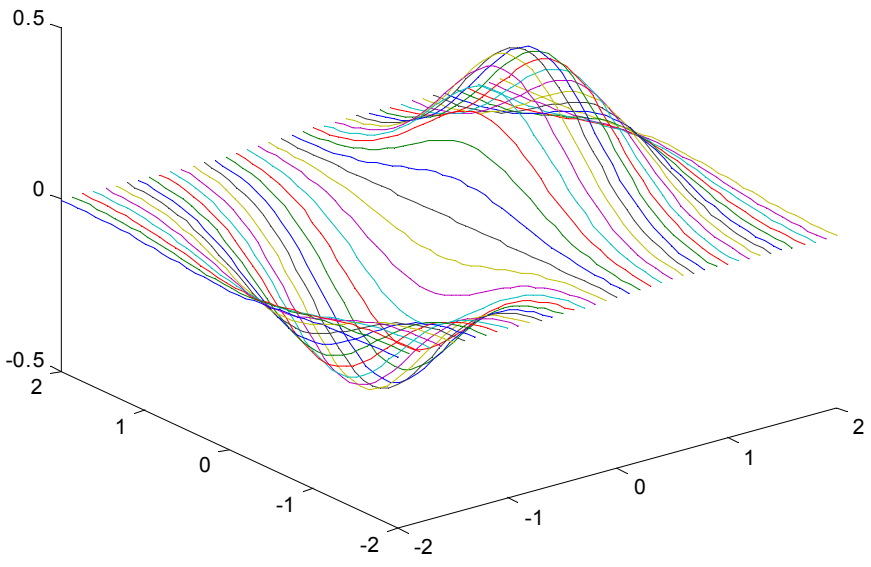
`+` sinal +, `o` círculo, `*` asterisco, `.` ponto, `x` cruz, `square` quadrado, `diamond` diamante, `v` triângulo invertido

10.7. Gráficos a 3-D – Funções Elementares

```
plot3(x,y,z)
```

➢ **Exemplo:**

```
» [x,y]=meshgrid([-2:.1:2]);
» z=x.*exp(-x.^2-y.^2);
» plot3(x,y,z)
```



10.8. Representação de uma matriz como uma superfície

Funções

`mesh, surf`	gráfico de superfície `surf(x,y,z)`
`meshc, surfc`	gráfico de superfície com contornos `surfc(x,y,z)`
`meshz`	gráfico de superfície.

10.9. Gráficos especiais – gráfico de barras

`bar(y)`	gráfico de barras normal
`bar3(y)`	gráfico de barras a 3 dimensões
`bar3(y,'group')`	gráfico de barras a 3 dimensões agrupado
`bar(y,'stack')`	gráfico de barras sobreposto

- **Exemplo:**

Especificação dos dados do eixo do xx'

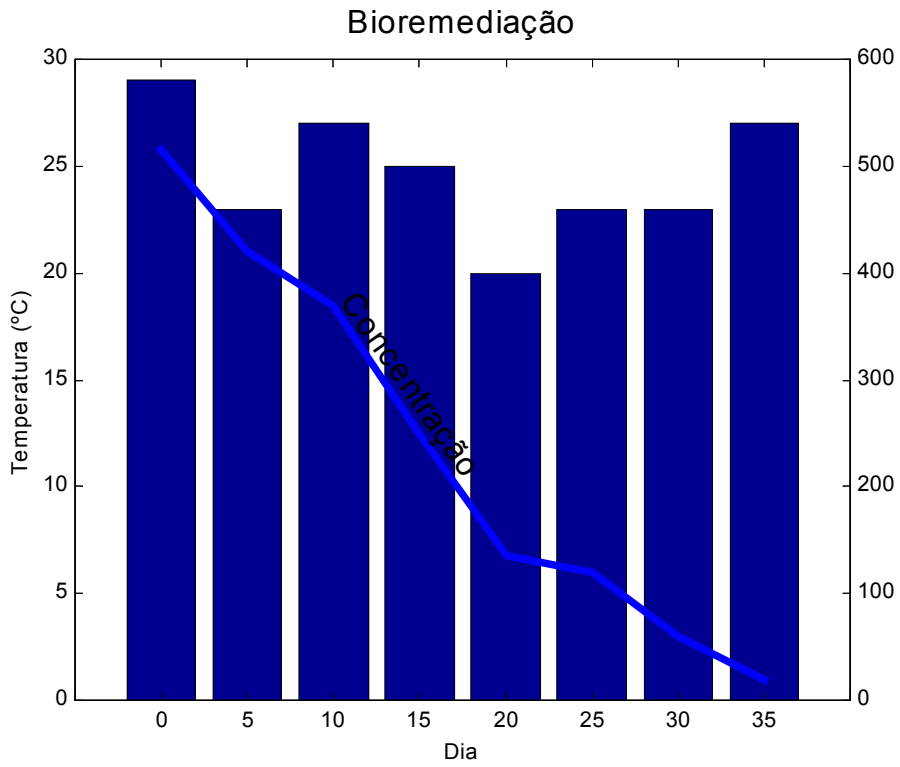
```
» dias=0:5:35;
» temp=[29 23 27 25 20 23 23 27];
» bar(dias,temp)

» xlabel('Dia')
» ylabel('Temperatura')
```

Por defeito, os limites do yy' são 0 e 30 – especificar outros limites

```
» set(gca,'Ylim',[15 30],'Layer','top')
```

10.9.1. Sobrepor um gráfico a um gráfico de barras

```
» dias=0:5:35;
» temp=[29 23 27 25 20 23 23 27];
» bar(dias,temp)
» xlabel('Dia')
» ylabel('Temperatura (°C)')
» tce=[515 420 370 250 135 120 60 20];
» h1=gca;
» h2=axes('Position',get(h1,'Position'));
» plot(dias,tce,'LineWidth',3)
» set(h2,'YaxisLocation','right','Color','none',
'XTickLabel',[])
» set(h2,'Xlim',get(h1,'Xlim'),'Layer','top')
» text(11,380,'Concentração','Rotation',-55',
'FontSize',14)
» title('Bioremediação','FontSize',14)
```



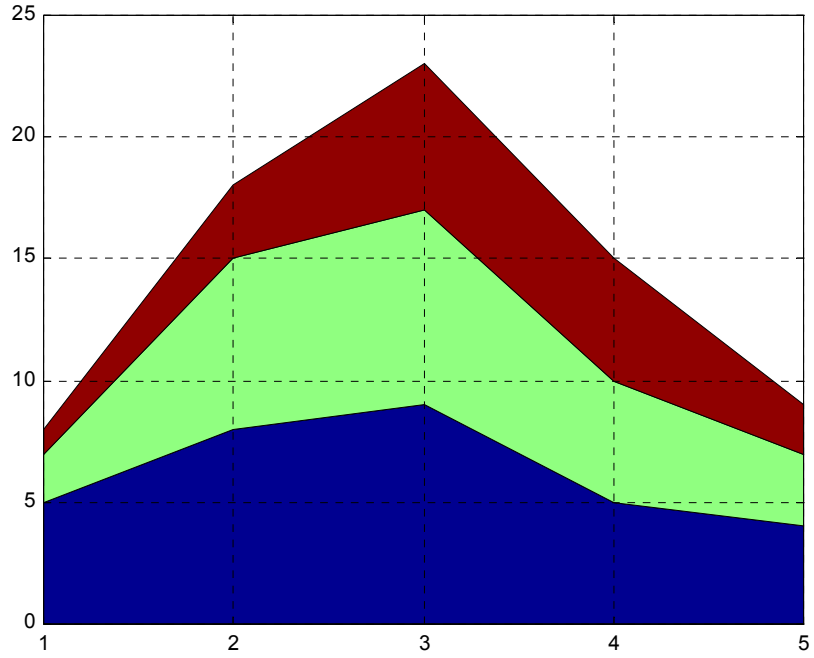
10.10. Gráficos especiais

10.10.1. Gráfico de areas

```
y=[5 2 1;8 7 3;9 8 6;5 5 5;4 3 2];

area(y)
```

Apresentação das grelhas

```
grid on
set(gca,'Layer','top')
set(gca,'Xtick',1:5)
```



10.10.2. Histogramas

➢ **Experimente os seguintes exemplos:**

Histogramas de um vector

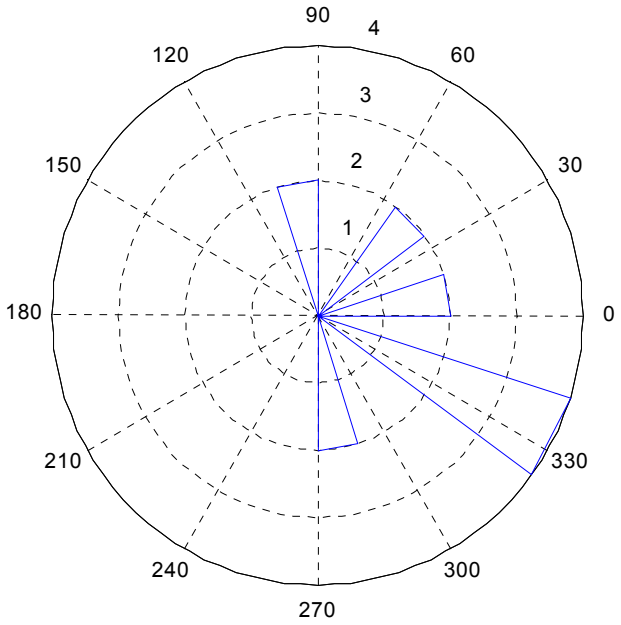
```
» yn=randn(10000,1);
» hist(yn)
```

Histogramas de matrizes

```
» y=randn(10000,3);
» hist(y)
```

Histogramas em coordenadas polares

```
» wdir=[45 90 90 45 360 335 360 270 335 270 335 335];
%direcção do vento num período de 12 horas
» wdir=wdir*pi/180; %transformaçãoem radianos
» rose(wdir)
```



A figura mostra que a direcção do vento é primariamente de 335º no período analisado.

10.11. Gráficos de valores discretos

2D

```
» t=0:0.1:2*pi;
» y=sin(t);
» stem(t,y)
```

Outras funções

```
» stem(t,y,'fill')
```

3D

```
» t=0:0.1:10;
» s=0.1+i;
» y=exp(-s*t);
» stem3(real(y),imag(y),t);
» hold on
» plot3(real(y),imag(y),t,'k')
» hold off
» rotate3d on
```

10.12. Gráficos em degraus

```
» t=0:0.2:10;
» y=sin(t);
» stairs(t,y);
» hold on
» plot(t,y)
```

10.13. Gráficos interactivos

```
» clf
» axis([0 10 0 10])
» hold on
» x=[];,y=[];,n=0;
» disp('O botão esquerdo do rato introduz os pontos')
» disp('O botão direito do rato introduz o último
ponto')
» but=1;
» while but==1
» [xi,yi,but]=ginput(1);,plot(xi,yi,'go')
» n=n+1;,x(n,1)=xi;,y(n,1)=yi;
» end
» t=1:n;,ts=1:0.1:n;
» xs=spline(t,x,ts);,ys=spline(t,y,ts);
» plot(xs,ys,'c-')
» hold off
```

11. O que é a Optimization Toolbox?

A toolbox de Optimização é uma colecção de funções que estendem as capacidades do Matlab. Inclui rotinas para vários tipos de de optimização:

- minimização não linear, sem restrições
- minimização não linear, com restrições
- programação linear e quadrática
- mínimos quadrados não lineares
- resolução de sistemas de equações não lineares
- mínimos quadrados lineares, com restrições

A toolbox de Optimização contém algoritmos específicos para problemas de grande dimensão.

Todas as funções da toolbox são ficheiros (M-files), feitos em Matlab, que implementam algoritmos de optimização específicos. As funções podem ser visualizadas através da instrução:

```
type function_name
```

Se quiser uma demosntração da aplicação da toolbox, deve executar o comando `optdemo`.

11.1. Aspectos gerais

11.1.1. Como usar a toolbox de Optimização?

- é necessário definir um ficheiro (M-file) que contenha a função objectivo a minimizar;
- as derivadas das funções a minimizar são calculadas através do método das diferenças finitas a não ser que sejam fornecidas no ficheiro da função objectivo;
- a passagem de parâmetros pode ser feita directamente pelas funções.

A toolbox de Optimização inclui rotinas que permitem a escolha de uma variedade de algoritmos de optimizaçãoe estratégias de procura unidimensional.

Para problemas de minimização, sem restrições, os algoritmos implementados são:

- método de Nelder-Mead
- método Quasi-Newton (fórmula BFGS)

Para problemas de minimização, com restrições, os algoritmos implementados são:

- programação quadrática sequencial

Para problemas de mínimos quadrados não lineares os algoritmos implementados são:

- método de Gauss-Newton
- método Levenberg-Marquardt

As estratégias de procura unidimensional para problemas de minimização sem restrições e de mínimos quadrados não lineares são:

- método de interpolação quadrática e cúbica
- extrapolação

Para problemas de grande dimensão, com excepção dos de programação linear, os métodos são baseados na estratégia de regiões de confiança. Os problemas com restrições de limites simples são resolvidos através do método de Newton. Os problemas com restrições de igualdade são resolvidos usando o método dos gradientes conjugados com uma matriz de precondicionamento.

11.1.2. Notação

A definição da função objectivo e das restrições, assim como, dos seus gradientes são estruturas que têm de ser definidas conforme a notação utilizada na toolbox de Optimização. Assim, para um problema com 3 variáveis (n = 3) e 2 restrições (m = 2) as funções e gradientes ficarão da seguinte forma:

- O gradiente da função objectivo é um vector que tanto pode ser definido como vector-linha como vector-coluna, isto é, nx1 ou 1xn.

$$\nabla f\left(\underset{\sim}{x}\right)=\begin{pmatrix}\dfrac{\partial f}{\partial x_1}\\ \vdots \\ \dfrac{\partial f}{\partial x_n}\end{pmatrix}\in IR^n \quad \text{ou} \quad \nabla f\left(\underset{\sim}{x}\right)=\begin{pmatrix}\dfrac{\partial f}{\partial x_1} & \cdots & \dfrac{\partial f}{\partial x_n}\end{pmatrix}$$

- O vector das restrições é um vector que tanto pode ser definido como vector-linha como vector-coluna, isto é, mx1 ou 1xm.

$$c\left(\underset{\sim}{x}\right)=\begin{pmatrix}c_1\left(\underset{\sim}{x}\right)\\ \vdots \\ c_m\left(\underset{\sim}{x}\right)\end{pmatrix}\in IR^m \quad \text{ou} \quad c\left(\underset{\sim}{x}\right)=\begin{pmatrix}c_1\left(\underset{\sim}{x}\right) & \cdots & c_m\left(\underset{\sim}{x}\right)\end{pmatrix}$$

- A matriz do gradiente das restrições (de dimensão nxm) tem de ser da seguinte forma:

$$\nabla c\left(\underset{\sim}{x}\right)=\begin{pmatrix} \frac{\partial c_1}{\partial x_1^2} & \cdots & \frac{\partial c_m}{\partial x_1^2} \\ \vdots & \ddots & \vdots \\ \frac{\partial c_1}{\partial x_n^2} & \cdots & \frac{\partial c_m}{\partial x_n^2} \end{pmatrix} \in IR^{n\times m}$$

11.2. Exemplo de optimização sem restrições

Considere o seguinte problema de minimização sem restrições:

$$f(x) = 3x_1^2 + 2x_1x_2 + x_2^2$$

1. Inicialmente deve definir-se a função objectivo através da criação de um ficheiro (M-file) de nome `myfun.m`:

```
function f = myfun(x)
f = 3*x(1)^2 + 2*x(1)*x(2) + x(2)^2;            %
função objectivo
```

2. De seguida, invocar-se a função `fminunc` que calcula o mínimo de `'myfun'` começando com o valor inicial de `[1,1]`:

```
 x0 = [1,1];
 % indicação de que o problema não é de grande dimensão
 options=optimset('LargeScale','off');
 [x,fval,exitflag,output]                                  =
fminunc('myfun',x0,options);
```

3. Depois de uma série de iterações, a solução, `x`, o valor da função em `x`, `fval`, e o `output` são apresentados:

```
x =
    -8.35607957827555e-009     2.89908724804278e-009
fval =
     1.69426897129167e-016
exitflag =
     1   %(se>0) indica que convergiu
output =
      iterations: 3
       funcCount: 16
        stepsize: 1.23529415048065
   firstorderopt: 1.71838286496623e-007
         algorithm:   'medium-scale:Quasi-Newton   line
search'
```

11.3. Visualização de resultados

Se se desejar que seja feito o *display* em cada iteração, em vez de apenas no fim, deve configurar-se o `optimset` com o seguinte:

```
options=optimset('Display','iter');
```

11.4. Introdução do valor gradiente e da matriz Hessiana

Para minimizar um função em que é dado o seu gradiente, deve-se modificar a M-file `myfun.m` para que o gradiente apareça como segundo argumento:

```
function [f,g] = myfun(x)
f = 3*x(1)^2 + 2*x(1)*x(2) + x(2)^2;      % função
objectivo
if nargout > 1
   g(1) = 6*x(1)+2*x(2);
   g(2) = 2*x(1)+2*x(2);
end
```

e indicar, antes da invocação da função `fminunc`, que o valor do gradiente está disponível, através de:

```
options = optimset('GradObj','on');
x0 = [1,1];
[x,fval] = fminunc('myfun',x0,options)
```

Depois de várias iterações é apresentado o resultado:

```
x =
  1.0e-015 *
   -0.6661         0
fval2 =
  1.3312e-030
```

Se a matriz Hessiana da função objectivo também for fornecida, então também devem ser definidas as opções de existência de Hessiana nesse sentido, i.e., `options=optimset('Hessian','on')`, e a função `myfun` deve retornar a Hessiana em `H`.

```
function [f,g,H] = myfun(x)
f = ...     % Compute the objective function value at x
if nargout > 1   % myfun called with two output
arguments
   g = ...    % gradient of the function evaluated at x
   if nargout > 2
```

```
   H = ...    % Hessian evaluated at x
end
```

11.5. Exemplo de optimização com restrições

Pretende-se determinar o valor de *x* que minimiza a função:

$$f(x) = -x_1 x_2 x_3$$
$$s.a \;\; x_1 - 2x_2 - x_3 \geq 0$$
$$x_1 x_2 + x_3 = 12$$
$$x_2^2 + x_1 * x_3 = 0$$

1. Inicialmente deve definir-se a função objectivo através da criação de um ficheiro (M-file) de nome `objfun.m`:

```
function f = objfun(x)
f = -x(1)*x(2)*x(3);            % função
objectivo
```

2. De seguida, escrever-se um ficheiro (M-file) para definição das restrições:

```
function [c, ceq]=confun(x)
% restrições de desigualdade não lineares
c = -x(1)+2*x(2)+x(3);    %equação reescrita
% restrições de igualdade não lineares
ceq = [x(1)*x(2)+x(3)-12; x(2)^2+x(1)*x(3)];
```

3. Por fim, invocar-se a função `fmincon` para calcular o mínimo de `'objfun'` próximo de `[-1,1,1]`:

```
x0 = [-1,1,1];
options=optimset('LargeScale','off');
[x,fval] = …
fmincon('objfun',x0,[],[],[],[],[],[],'confun',options)
;
```

4. E depois de uma série de iterações, a solução `x` e o valor da função `fval` é:

```
x =
    -9.3560e-009     1.8990e-009
fval =
     1.6942e-016
```

5. Pode-se, ainda, saber os valores das restrições na solução:

```
[c,ceq] = confun(x)
c =
```

```
    -9.3560e-009
ceq =
     1.8990e-009
       3.7990e-009
```

11.6. Exemplo de optimização com restrições, com gradientes

Pretende-se determinar o valor de *x* que minimiza a função:

$$f(x) = -x_1 x_2 x_3$$
$$s.a \;\; x_1 x_2 + x_3 = 12$$
$$x_2^2 + x_1 * x_3 = 0$$

1. Inicialmente deve definir-se a função objectivo e o seu gradiente através da criação de um ficheiro (M-file) de nome `objfun.m`:

```
function [f,G] = objfun(x)
  f=-x(1)*x(2)*x(3);                    %função
  objectivo
  %gradiente da função objectivo
  G=[-x(2)*x(3); -x(1)*x(3); -x(1)*x(2)];
```

2. Depois, compor-se um ficheiro (M-file) para definição das restrições e seus gradientes:

```
function [c,ceq,Dc,Dceq]=confun(x)
  % restrições de desigualdade não lineares
   c=[];
   Dc=[];
   % restrições de igualdade não lineares
   ceq=[x(1)*x(2)+x(3); x(2)^2+x(1)*x(3)];
   Dceq=[x(2), x(3);x(1), 2*x(2); 1, x(1)];
```

3. Uma vez que se fornece o gradiente da função objectivo e das restrições, é necessário informar a função `fmincon`. Então deve-se usar o `optimset` desta forma:

```
options=optimset('LargeScale','off');
options=optimset(options,'GradObj','on','GradConstr','o
n');
```

4. Por fim,invocar-se a rotina de optimização:

```
x0 = [-1,1,1];
[x,fval,exitflag,output] = ...
fmincon('objfun',x0,[],[],[],[],[],[],'confun',options)
```

```
[c,ceq,Dc,Dceq]=confun(x)

[f,g]=objfun(x)
```

5. Como resultado temos que:

```
x =
   -0.6564
   -0.0000
   -0.0000
fval =
  2.9007e-015
exitflag =
     1
output =
       iterations: 4
        funcCount: 9
         stepsize: 1
        algorithm: 'medium-scale: SQP, Quasi-Newton,
line-search'
    firstorderopt: []
     cgiterations: []

c =
     []
ceq =
  1.0e-007 *
   -0.4133
    0.5143
Dc =
     []
Dceq =
   -0.0000   -0.0000
   -0.6564   -0.0000
    1.0000   -0.6564

 f =
   2.9007e-015
 g =
   1.0e-007 *
    -0.0000
    -0.5143
    -0.3702
```

11.7. Funções da toolbox de Optimização

11.7.1. Funções de Minimização

Function	Purpose
`fgoalattain`	Multiobjective goal attainment
`fminbnd`	Scalar nonlinear minimization with bounds
`fmincon`	Constrained nonlinear minimization
`fminimax`	Minimax optimization
`fminsearch, fminunc`	Unconstrained nonlinear minimization
`fseminf`	Semi-infinite minimization
`linprog`	Linear programming
`quadprog`	Quadratic programming

11.7.2. Resolução de equações

Function	Purpose
`\`	Linear equation solving
`fsolve`	Nonlinear equation solving
`fzero`	Scalar nonlinear equation solving

11.7.3. Mínimos quadrados

Function	Purpose
`\`	Linear least squares (see the online *MATLAB Function Reference* guide)
`lsqlin`	Constrained linear least squares
`lsqcurvefit`	Nonlinear curve fitting
`lsqnonlin`	Nonlinear least squares
`lsqnonneg`	Nonnegative linear least squares

11.7.4. Definição de opções

Function	Purpose
`optimset, optimget`	Parameter setting

11.7.5. Demonstrações de métodos de grande dimensão

Function	Purpose
`circustent`	Quadratic programming to find shape of a circus tent
`molecule`	Molecule conformation solution using unconstrained nonlinear minimization
`optdeblur`	Image deblurring using bounded linear least-squares

11.7.6. Demonstrações de métodos de média dimensão

Function	Purpose
`bandemo`	Minimization of the `banana` function
`dfildemo`	Finite-precision filter design (requires *Signal Processing Toolbox*)
`goaldemo`	Goal attainment example
`optdemo`	Menu of demonstration routines
`tutdemo`	Tutorial walk-through