Comandos de Matlab de apoio às aulas de Métodos Numéricos

Ana Maria A. C. Rocha

Departamento de Produção e Sistemas
Escola de Engenharia
Universidade do Minho
2021

Conteúdo

1 Introdução ao MATLAB

O MATLAB (MATrix LABoratory) é um sistema interativo para a execução de cálculos científicos e de engenharia, suportado por software sofisticado baseado em cálculo matricial. O Matlab é um software de computação numérica que integra:

- computação
- visualização
- programação

Para além de ter um ambiente de execução, define uma linguagem de programação. No entanto é possível efetuar cálculos complexos sem ser necessário escrever um programa.

1.1 O sistema Matlab

O sistema Matlab é constituído pelas seguintes partes:

- A linguagem
- O ambiente de trabalho
- Gráficos
- Toolboxes

A linguagem

Permite a manipulação e criação de matrizes de forma rápida e intuitiva. Diferentes soluções para um problema podem ser testadas numa fração do tempo que levaria com outras linguagens (C ou Fortran por ex.).

Possui um conjunto muito vasto de funções que permitem resolver problemas complexos de forma eficiente.

O ambiente de trabalho

O Matlab proporciona um ambiente de trabalho que permite a gestão e visualização das variáveis, ler e gravar variáveis em disco e gerar programas em linguagem Matlab, possibilitando assim a automatização de cálculos complexos.

Gráficos

As funções de criação, visualização e manipulação de gráficos são muito fáceis de usar e permitem a criação de gráficos 2D e 3D.

O ajuste de escala é automático e o utilizador pode começar a utilizar as funções de geração de gráficos pouco tempo depois do primeiro contacto com o ambiente do Matlab.

Toolboxes

O Matlab disponibiliza um conjunto de pacotes de funções para as mais variadas áreas de cálculo científico, sendo estes denominados "toolboxes".

Existem toolboxes para otimização, estatística, processamento de sinal, processamento de imagem, controlo, cálculo simbólico, etc.

1.2 O comando HELP

Para esclarecer a maior parte das dúvidas acerca da utilização de uma dada função do Matlab o comando help é de grande utilidade.

Se pretender, informação sobre a função `plot`, basta fazer

```
>> help plot
```

obtendo-se a seguinte descrição

```
PLOT Linear plot.
PLOT(X,Y) plots vector Y versus vector X...
PLOT(Y) plots the columns of Y versus their index...
```

O Matlab possui todas as funções organizadas em grupos e a própria estrutura de directórios onde o Matlab é armazenado em disco reflete esse facto. Por exemplo, todas as funções de álgebra linear estão armazenadas na pasta matfun. Para obter uma lista completa deste tipo de funções basta fazer

```
>> help matfun
```

Como não é fácil decorar os nomes de todas as categorias de funções, existe uma janela de ajuda mais organizada, bastando para tal escrever o comando

```
>> helpwin
```

1.3 Expressões matemáticas

O Matlab permite a construção de expressões matemáticas sem qualquer declaração do formato numérico ou dimensão das matrizes.

Existem quatro constituintes básicos nas expressões do Matlab:

- Variáveis
- Números
- Operadores
- Funções

Variáveis

Todas as variáveis do Matlab são do tipo matriz e a sua criação é automática. Por exemplo, o comando

```
>> Custo = 1000
```

resulta na criação em memória de uma matriz de 1×1 (guardada na variável 'Custo') com o valor 1000.

Declaração de uma variáveis:

- variáveis são alocadas na memória ao serem declaradas;
- nomes de variáveis são sensíveis a letras maiúsculas e minúsculas;
- vetores e matrizes devem ser declarados entre [ ];
- elementos de uma mesma linha numa matriz são separados por espaço(s) ou vírgula;
- ponto-e-vírgula(;) indica o final de uma linha de uma matriz ou expressão.

Para visualizar o valor de uma variável basta escrever o seu nome.

```
>> Custo
```

Constantes e variáveis especiais

Símbolo	**Operação**
ans	resposta mais recente
eps	precisão da máquina
realmax	maior número de ponto flutuante
realmin	menor número de ponto flutuante
pi	3.1415926535897....
i, j	unidades Imaginárias
inf	Infinito
NaN	Not-a-Number

Números

O Matlab utiliza uma notação standard para a representação dos números, admitindo notação científica e números complexos.

Exemplos:

3	-99	0.0001
9.6397238	1.60210e-20	6.02252e23
$12i - 2j$	-3.14159j	3e5i

Formato Numérico

O comando `format` define o modo como os valores numéricos são exibidos.

O comando `format` controla apenas a forma com os valores são apresentados e não a forma como estes são usados internamente.

Todos os cálculos em Matlab são executados em dupla precisão. `short`.

FORMAT	(Default) Format igual a SHORT
FORMAT SHORT	Formato de ponto fixo com 4 dígitos após o ponto decimal
FORMAT LONG	Formato de ponto fixo com 14 dígitos após o ponto decimal
FORMAT SHORTE	Formato de notação científica com 4 dígitos após o ponto decimal
FORMAT LONGE	Formato de notação científica com 15 dígitos após o ponto decimal
FORMAT SHORTG	Melhor formato entre ponto fixo ou notação científica com 4 dígitos após o ponto decimal
FORMAT LONGG	Melhor formato entre ponto fixo ou notação científica com 14 dígitos após o ponto decimal
FORMAT SHORTENG	format de engenharia com 4 dígitos após o ponto decimal
FORMAT LONGENG	format de engenharia com 14 dígitos após o ponto decimal
FORMAT HEX	Formato hexadecimal
FORMAT +	Escreve os símbolos +, - e espaço para positivos, negativos e zeros, respetivamente
FORMAT BANK	Formato fixo para dólares e cêntimos
FORMAT RAT	Formato racional

Exemplo 1

```
>> pi
```

ans = 3.1416

```
>> format long

>> pi
```

ans = 3.141592653589793

Exemplo 2

```
>> x = 5/2
```

De seguida, apresentam-se as possibilidades de saída:

Comando	Formato	Saída
5/2	format short	2.5000
	format long	2.50000000000000
	format shorte	2.5000e+000
	format longe	2.500000000000000e+000
	format shortg	2.5
	format longg	2.5
	format shorteng	2.5000e+000
	format hex	4004000000000000
	format rat	5/2
	format bank	2.50

Operadores

As operações nas expressões do Matlab seguem as regras habituais de precedência e podem ser aplicadas quer a matrizes quer a números.

Operadores matemáticos

Símbolo	Operação
+	Adição
-	Subtração
*	Multiplicação
/	Divisão à esquerda
^	Potenciação

Funções

O Matlab possui um conjunto muito grande de funções matemáticas que permitem resolver grande parte dos problemas de cálculo.

Quando as funções são aplicadas sobre matrizes a função é aplicada a cada um dos elementos.

Funções matemáticas

2*x	multiplicação	$2x$
x^2	potenciação	x^2
exp(x)	exponencial de x	e^x
log(x)	logaritmo natural de x na base e	$\ln(x)$
log10(x)	Logaritmo de x na base 10	$\log(x)$
sqrt(x)	raiz quadrada de x	$\sqrt{(}x)$
abs(x)	Valor absoluto de x	$\|x\|$
real	parte real de número complexo	
imag	parte imag. de núm. complexo	
conj	conj. dos núm. complexos	
std	desvio padrão	

Funções trigonométricas

sin(x)	seno	cos(x)	cosseno
tan(x)	tangente	csc(x)	co-secante
sec(x)	secante	cot(x)	co-tangente
acos(x)	arco-cosseno	asin(x)	arco-seno
atan(x)	arco-tangente	asec(x)	arco-secante
acsc(x)	arco-cossecante	acot(x)	arco-cotangente

Funções

round(x)	Converte x para o inteiro mais perto
fix(x)	Converte x para o inteiro mais perto em direção a zero
floor(x)	Converte x para o inteiro mais perto em direção a $-\infty$
ceil(x)	Converte x para o inteiro mais perto em direção a $+\infty$
sign(x)	Devolve -1 se x<0 ou 1 se x≥0
rem(x,y)	Resto da divisão de x/y

Note-se que `x` pode ser um escalar, vetor ou matriz. Caso seja uma matriz, por exemplo, a operação é aplicada a cada um dos elementos da matriz. Além disso, para qualquer uma das funções assume-se que o valor de `x` se encontra na gama permitida, caso contrário dá erro.

2 Vetores e Matrizes

2.1 Introdução de dados

A forma de introdução de um vetor pode ser:

- pela separação dos elementos por um espaço ou vírgula, resultando num vetor linha
- pela separação dos elementos por ponto e vírgula, resultando num vetor coluna

Exemplos:

```
>> u = [1 2 3]
```

Solução:

```
u = 1 2 3
```

```
>> v = [1; 2; 3]
```

Solução:

```
v =
1
2
3
```

```
>> x = [1
2
3]
```

Solução:

```
x =
1
2
3
```

Outra forma de criar um vetor é através do comando:
`a=val_inicial:incremento:valor_final`

- o `val_inicial` é o primeiro valor do vetor
- `incremento` é o valor do espaçamento entre os valores

- `val_final` é o último valor do vetor

De seguida, apresentam-se dois exemplos relativos à geração (automática) de vetores ordenados:

```
>> a=1:1:4
a =
     1     2     3     4
```

Se o incremento for de 1 valor, pode-se omitir, ou seja:

```
>> a=1:4
a =
     1     2     3     4
```

Com um incremento de 0.2, resulta em:

```
>> b=1:0.2:2
b =
    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
```

A forma de introdução de uma matriz envolve:

- a separação dos elementos de uma linha por espaço ou , (vírgula)
- a indicação do fim de uma linha usando o ; (ponto e vírgula)
- a inserção da lista dos elementos da matriz entre [ ] (parênteses retos)

Exemplos:

```
>> A=[2 2 3; 4 5 6; 7 8 9]
```
Solução:
```
A =
2 2 3
4 5 6
7 8 9
```

```
>> A = [2 2 3
4 5 6
7 8 9]
```
Solução:
```
A =
2 2 3
4 5 6
7 8 9
```

Manipulação de matrizes

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

>> A(2,3)
     6

>> A(2:3,1)
     4
     7
```

Funções de geração de matrizes especiais

Comando	Operação
rand(m,n)	Gera uma matriz (m x n) de elementos aleatórios entre [0,1]
rand(n)	Gera uma matriz quadrada de elementos aleatórios [0,1]
eye(m,n)	Gera uma matriz com elementos na diagonal iguais a 1
eye(n)	Gera a matriz identidade de dimensão n
zeros (m,n)	Gera matriz com todos elementos iguais a 0
zeros(n)	Gera matriz quadrada com todos elementos iguais a 0
ones(m,n)	Gera matriz com todos elementos iguais a 1
ones(n)	Gera matriz quadrada com todos elementos iguais a 1
magic(n)	Gera uma matriz de números inteiros (n x n) cuja soma dos elementos de cada linha é igual à soma dos elementos de cada coluna

2.2 Operações básicas com vetores/matrizes

Adição e subtração de matrizes

(Nota: as matrizes têm de ter a mesma dimensão)

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

```
>> B=[ 2 2 2; 3 3 3; 4 4 4]
```

```
A =
2 2 3
4 5 6
7 8 9
```

```
B =
2 2 2
3 3 3
4 4 4
```

```
>> X=A+B
X =
4  4  5
7  8  9
11 12 13
```

```
>> X=A-B
X =
0 0 1
1 2 3
3 4 5
```

Produto de vetores

(Nota: deve ter em atenção as suas dimensões)

```
>> u = [1 2 3]
>> v = [1; 2; 3]
```

```
u =
1 2 3
```

```
v =
1
2
3
```

```
>> u*v
ans =
14
```

```
>> y = v*u
y =
1 2 3
2 4 6
3 6 9
```

Multiplicação de matrizes

(Nota: as matrizes têm de ter a mesma dimensão)

```
A =                B =
2 2 3              2 2 2
4 5 6              3 3 3
7 8 9              4 4 4
```

```
>> A*B             >> B*A             >> A.*B
ans =              ans =              ans =
22 22 22           26 30 36           4   4  6
47 47 47           39 45 54           12 15 18
74 74 74           52 60 72           28 32 36
```

Operações elemento a elemento

Símbolo	Operação
.*	Multiplicação elemento a elemento
./	Divisão à esquerda elemento a elemento
.^	Potenciação elemento a elemento

Exemplos:

```
A =                B =                u = 2 3 5
2 2 3              2 2 2
4 5 6              3 3 3              v = 4 2 1
7 8 9              4 4 4
```

```
>> u.*v            >> u.^3            >> u.\v
ans =              ans =              ans =
8 6 5              8 27 125           0.5000 1.5000 5.0000
```

```
>> A.*B            >> B.^2            >> A.\B
ans =              ans =              ans =
4 4 6              4 4 4              1.0000 1.0000 1.5000
12 15 18           9 9 9              1.3333 1.6667 2.0000
28 32 36           16 16 16           1.7500 2.0000 2.2500
```

Transposta de um vetor ou matriz

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

```
A =                u =                >> x=u'            >> x=A'
2 2 3              1 2 3              x =                x =
4 5 6                                 1                  2 4 7
7 8 9                                 2                  2 5 8
                                      3                  3 6 9
```

Potência de matrizes

- `A^2 é equivalente a A*A`
- `A.^2 é equivalente ao quadrado de cada elemento de A`

Exemplos:

```
A =                 A.^2                A^2
     2   2   3        4   4   9          33  38  45
     4   5   6       16  25  36          70  81  96
     7   8   9       49  64  81         109 126 150
```

2.3 Outros comandos

Comando	Operação
max	máximo de um vetor/matriz ou entre 2 valores
min	mínimo de um vetor/matriz ou entre 2 valores
mean	média aritmética
median	mediana
std	desvio padrão
var	variância
sort	ordenar de forma ascendente
sortrows	ordenar linhas de forma ascendente

Comando	Operação
size	Dimensões da matriz.
length	Tamanho de um vetor.
ndims	Número de dimensões.
disp	apresenta a matriz ou texto.
isempty	Verdadeiro se array vazio.
isequal	Verdadeiro se arrays idênticos.
isnumeric	Verdadeiro se array de valores numéricos.
islogical	Verdadeiro se array de valores lógicos.
logical	Converte valores numéricos em lógicos

3 Ficheiros .m

3.1 O que são?

As instruções em MATLAB são geralmente dadas e executadas linha a linha. No entanto, é possível executar uma sequência de comandos que está guardada num ficheiro. Ficheiros que contêm código em linguagem MATLAB são chamados M-files e são do tipo nome ficheiro.m. Uma M-file consiste numa sequência de comandos MATLAB que pode inclusivamente fazer referência a outras M-files. As M-files podem criar-se usando um qualquer editor de texto. O

Matlab tem um editor de texto associado, que pode ser acedido através da opção New do menu FILE para criar uma nova M-file ou Open para editar uma M-file já existente.

- podemos criar novas funções ou scripts
- MATLAB possui um editor próprio e um debugger
- comentários começam por %
- ao se criar uma função ou script ela dever ser definida no path

Há dois tipos de M-files que podem ser usadas: scripts e funções.

scripts : executam os argumentos diretamente, automatizando uma série de comandos

função : argumentos podem ser passados para a função, havendo uma manipulação de variáveis.

3.2 Criação de scripts

Uma script é um ficheiro de texto onde são armazenados os comandos a serem executados pelo Matlab, um em cada linha e pela sequência desejada.

Quando uma script é invocada, o Matlab executa os comandos encontrados no ficheiro e as variáveis que aparecem na script são globais. Este tipo de ficheiro é muito usado quando há necessidade de executar um grande número de operações.

O ficheiro poder ter qualquer nome apenas tendo como restrição a extensão que tem de ser .m (daí o nome de ficheiros *.m).

Por exemplo, pretende-se determinar a soma de x (com o valor de 2) , y (com o valor de 3) e z (com o valor de 1). Assim cria-se um ficheiro .m, que se poderia chamar soma.m, e escrevia-se os seguintes comandos:

```
x = 2
y = 3
z = 1
c = x+y+z
```

Uma vez criado este ficheiro soma.m bastaria agora digitar o seguinte comando na linha de comando do Matlab (não é necessário a extensão).

```
>> soma
```

Pode também fazer F5 para executar um ficheiro .m (se for o caso de uma script, apenas). Quando é uma função não se pode executar, apenas é possível fazer na linha de comandos, não esquecendo os argumentos de entrada.

Automaticamente o Matlab executa cada uma das linhas do ficheiro soma.m. A este tipo de ficheiros em que os comandos são executados sequencialmente dá-se o nome de Scripts para os diferenciar das funções, como se verá mais adiante.

Assim sempre que se executa uma sessão em Matlab deve-se de preferência criar um ficheiro de texto e escrever aí todos os comandos. Sempre que se queira executar todas as instruções basta digitar um comando no Matlab. Doutra forma seria necessária digitar todas as instruções como comandos de Matlab.

Além disso, se for necessário modificar algum dado (por exemplo alterar o valor do número x de 2 para 4) basta modificar o ficheiro de texto. No modo de comando seria necessário, além de modificar o valor de x, digitar todas as restantes instruções.

3.3 Criação de funções

Uma M-file que contém a palavra "function" no início da primeira linha é chamada uma Função. As funções diferem das scripts na medida em que podem ser usados argumentos e as variáveis definidas e manipuladas dentro de uma função são locais, i.e não operam globalmente no espaço de trabalho.

Enquanto que um ficheiro *.m executa sequencialmente todas as instruções uma função é um ficheiro *.m especial que devolve um ou mais valores. Distingue-se de um script principalmente pela primeira linha, que é da forma:

Sintaxe de definição de uma função:

```
function [argumentos saida] = nome_funcao (argumentos entrada)

ou

function[res1, res2, ...] = nome_da_funcao (arg1,arg2, ...)
\% comentario para help

lista de procedimentos da funcao

return
```

Nota: gravar o ficheiro com o nome `nome_da_funcao`.

Suponha, por exemplo, que se implementa uma função capaz de somar três números x, y e z. Para resolver o problema começa-se por criar um ficheiro com o nome de, por exemplo, soma.m. Uma possível solução para o conteúdo deste ficheiro será:

```
function r = soma( x, y, z)
r = x + y + z
end
```

Ou seja, como parâmetros de entrada tem-se os valores de x, y e z. Como parâmetro de saída temos o valor r que é a soma dos três valores. Em Matlab aceder-se-ia a esta função simplesmente fazendo:

```
>> r = soma( 1, 2 ,3)
```

e o resultado seria r = 6. Suponhamos que se deseja uma função que calcule a soma e o produto dos três elementos x, y e z. Da mesma forma cria-se um ficheiro de nome somaprod.m com o seguinte conteúdo:

```
function [ soma, produto] = soma_produto( x, y, z)
soma = x + y + z ;
produto = x * y * z ;
end
```

Para usar esta função em Matlab executa-se,

```
>> [s,p] = somaprod( 2, 2, 3)
```

resultando s= 7 e p= 12.

Nota importante: O Matlab identifica a função pelo nome do ficheiro (neste caso somaprod) e não pelo nome que lhe atribui dentro do ficheiro de texto (neste caso soma_produto).

Uma função pode ser criada num ficheiro .m e deve ser definida da seguinte forma.

Exemplo:

```
function f=fact(n)
% função factorial
% retorna o factorial de n
f= prod(1:n);
```

n - é argumento de entrada
fact - nome da função
f - argumento de saída
function - palavra reservada
% -indica comentário

Representação de funções

O MATLAB representa funções matemáticas exprimindo-as em M-files ou diretamente na linha de comandos sob a forma de string.

Exemplo:

Para a função

$$\frac{1}{(x-3)^2+0.01}+\frac{1}{(x-0.9)^2+0.04}-6$$

1. Pode criar-se um ficheiro (M-file) de 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;
```

Para saber o valor da função quando x=2.5 faz-se:

```
humps(2.5);
```

2. Pode inserir-se na linha de comando

```
f=inline('1./((x-0.3).^2+0.01)+1./((x-0.9).^2+0.04)-6')
```

Para saber o valor de f(2.5) faz-se:

```
f(2.5);
```

3.4 Argumentos de funções

Uma função pode ser definida com argumentos de entrada e/ou argumentos de saída.

Exemplo:

function [x,y,z] = sphere (theta,phi,rho)

é uma função que tem 3 argumentos de entrada (theta,phi,rho) e 3 argumentos de saída (x,y,z).

Se uma função não tem argumentos de saída e tem, por exemplo, um argumento de entrada x:

function printresults(x)

ou

function []=printresults(x)

As variáveis nargin e nargout indicam quantos argumentos de entrada e de saída, respectivamente, tem uma função.

Exemplo:

```
function c=testarg(a,b)
if (nargin==1)
c=a.^2;
elseif (nargin==2)
c=a+b;
end
```

Em alguns casos de definição de funções devem considerar-se os argumentos (tanto de entrada como de saída) como vectores.

Exemplo:

```
function [p,j]=exerc(v)
a=v(1)
b=v(2)
c=v(3)
p=[(a+b)*(c-b), (a-c)*b]
j=(a-b).^2+(b-2)-(c+5).^3;
```

Neste exemplo, v é um vetor de dimensão 3, p é um vector de dimensão 2 e j um valor numérico.

4 Gráficos

A geração de gráficos no Matlab representa um dos seus aspetos mais úteis.

O MatLab oferece uma vasta biblioteca de funções voltadas à construção de gráficos.

Através de comandos simples, o Matlab pode produzir gráficos bidimensionais ou tridimensionais (ex: gráficos de contornos, de superfície, etc).

4.1 Comando `plot`

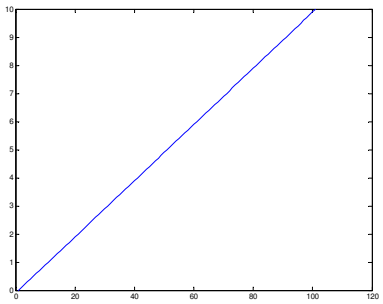
A função `plot` é a mais utilizada no Matlab para gerar gráficos variando o seu comportamento consoante os parâmetros de entrada.

A sintaxe de utilização é:

plot(X,Y)	desenha a função Y em função do vetor X
plot(X,Y,S)	desenha X em função de Y, segundo os parâmetros S

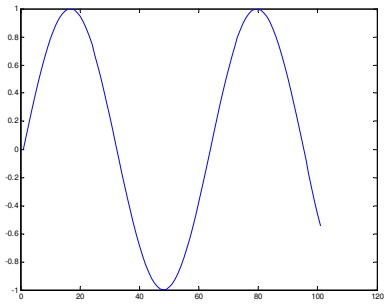
A sua forma mais simples consiste em passar como entrada apenas um vetor:

```
>> x=0:1.0:100;
>> y=0:0.1:10;
>> plot(x,y);
```



Passando uma função é:

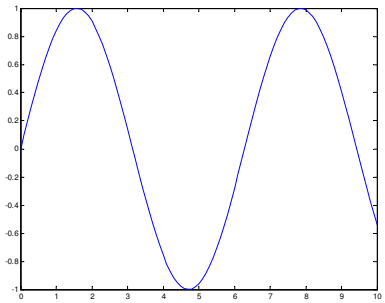
```
>> x=0:0.1:10;
>> plot(sin(x));
```



O gráfico gerado apresenta em abcissas os índices i dos elementos do vetor e em ordenadas o valor de cada um dos elementos do vetor.

Também é possível utilizar um segundo vetor para o eixo das abcissas tal como no exemplo seguinte:

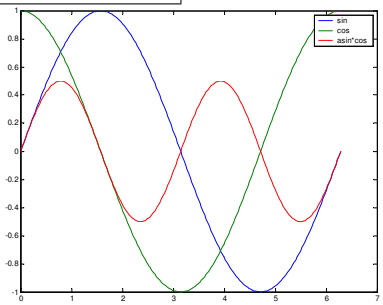
```
>> x=0:0.1:10;
>> y=sin(x);
>> plot(x,y);
```



A função `plot` admite a representação simultânea de várias curvas, acrescentando mais argumentos de entrada, devendo os vetores possuir o mesmo número de amostras.

Vejamos um exemplo:

```
>> w=0:pi/100:2*pi;
>> x1= sin(w);
>> x2= sin(w+pi/2);
>> x3= x1.*x2;
>> plot(w,x1,w,x2,w,x3);
>>legend('sin','cos','sin*cos');
```



A função `plot` permite escolher o tipo de linha, a cor, etc, e existem ainda funções para, por exemplo, acrescentar etiquetas aos eixos, criar uma grelha, etc.

Destas funções destacam-se as seguintes:

xlabel	Etiqueta do eixo das abcissas
ylabel	Etiqueta do eixo das ordenadas
title	Nome da figura
legend	legenda com o significado de cada linha
figure	Cria uma janela nova
figure(gcf)	Coloca a janela de gráfica corrente à frente
zoom	Para aumentar zonas de um gráfico
grid	Grelha
axis	Define os limites dos eixos do gráfico

O Matlab possui uma função que permite gerar um certo número de valores num dado intervalo. Se pretender, por exemplo, gerar 100 pontos no intervalo de $-\pi$ a π:

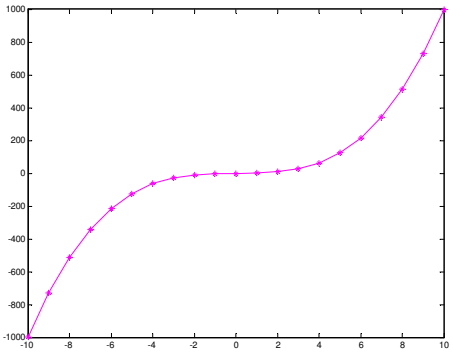
```
>> x = linspace(-pi,pi,100);
```

Pode-se definir o estilo de linha, símbolo ou cor de um gráfico através de um conjunto de caracteres formado por um elemento de cada uma das seguintes 3 colunas:

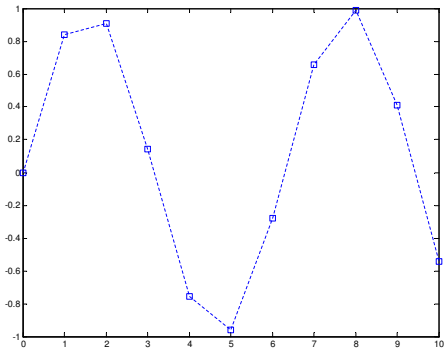
Cor		Tipo de símbolo		Estilo de linha	
c	azul claro	.	ponto	-	sólida
m	magenta	o	círculo	–	tracejada
y	amarelo	x	cruz	.	pontos
g	verde	*	asterisco	-.	traço-ponto
b	azul	s	quadrado	none	nenhuma
w	branco	d	diamante	:	ponteada
k	preto	v	triângulo invertido		
		^	triângulo para cima		
		<	triângulo para esquerda		
		>	triângulo para direita		
		+	mais		
		p	pentagrama		
		h	hexagrama		

Exemplos

```
>> x=-10:1:10;
>> y=sin(x)+x.^3;
>> plot(x,y ,'m*-');
```




```
>> x=-10:1:10;
>> plot(x,sin(x),'bs:');
```



4.2 Comando `fplot`

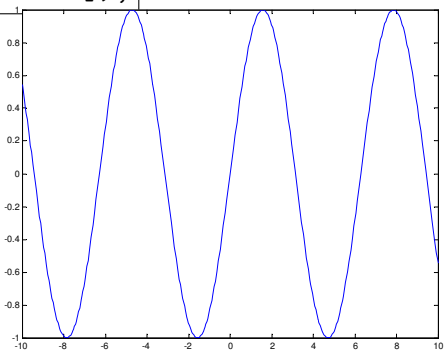
O comando `fplot` faz o gráfico de uma função.

A sintaxe de utilização é:

```
fplot(@funcao,[intervalo pretendido]);
```

Exemplo:

```
>> fplot(@(x)sin(x),[-10 10]);
```



A função deve estar especificada num ficheiro e depois invocada através do comando `fplot`.

Exemplo:

```
>> fplot(@(x)sin(x),[-10 10]);
```

4.3 Comando `ezplot`

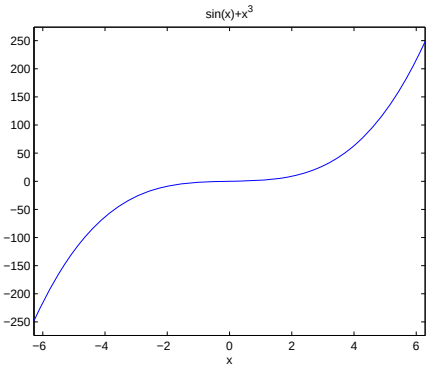
O comando `ezplot` permite desenhar o gráfico de uma função sem especificar o intervalo de domínio.

A sintaxe de utilização é:

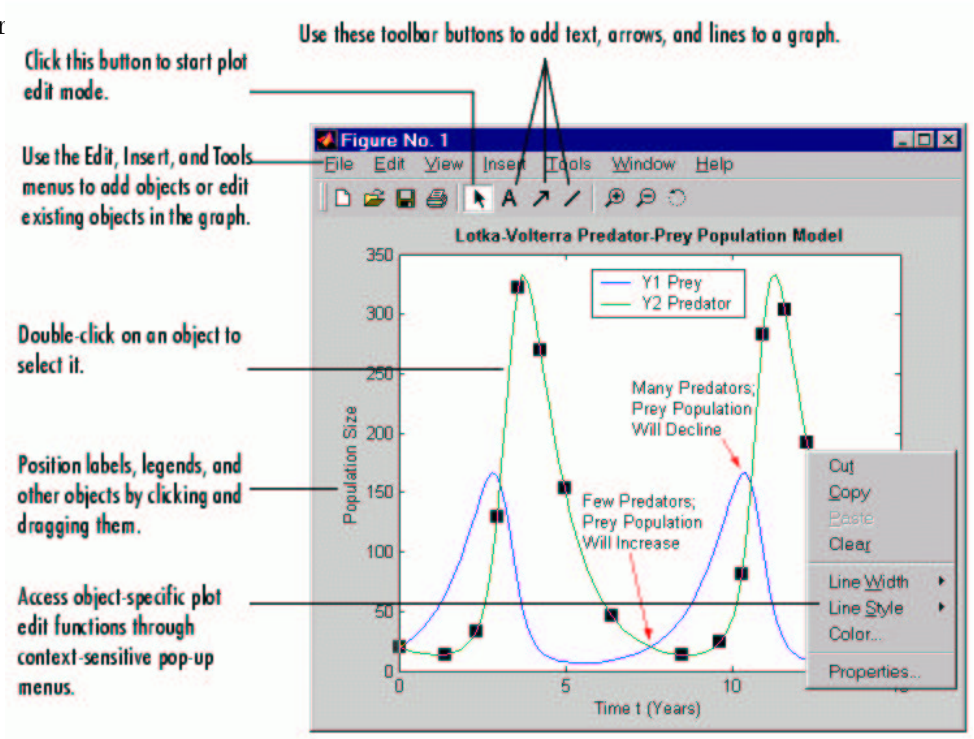
```
ezplot(@funcao);
```

Exemplo:

```
>> ezplot(@(x)sin(x))
```



Para alterar



5 Cálculo de raízes de equações

O MATLAB possui duas funções para o cálculo da raiz de uma equação:

- **Comando `fzero`**

  Determina uma raiz de uma equação algébrica ou transcendente

- **Comando `roots`**

  Obtém todas as raízes de uma equação algébrica

5.1 Comando `fzero`

O comando `fzero` permite encontrar o zero (raiz) de uma função de uma variável.

5.1.1 Sintaxe de utilização

A sintaxe de utilização é:

```
x = fzero(fun,x0)
x = fzero(fun,x0,options)
x = fzero(fun,x0,options,P1,P2,...)
[x,fval] = fzero (fun,x0,options,P1,P2,...)
[x,fval,exitflag,output] = fzero(fun,x0,options,...)
```

O algoritmo implementado usa uma combinação do método da bissecção e de interpolação.

Parâmetros de entrada

- `x0` é uma aproximação inicial à solução (ou então um intervalo cujos valores da função, cujos valores da função nos limites têm sinais opostos).
- `fun` é a função que se pretende encontrar a sua raíz
-

`Options` - definição de parâmetros	
`Display`	Nível de apresentação
`off`	não apresenta nada
`iter`	apresenta resultados em cada iteração
`final`	apresenta apenas o resultado final
`notify`	(default) apresenta apenas o resultado final
`TolX`	Tolerância de paragem relativamente a X

Para ver as opções disponíveis para o `fzero` fazer:

```
optimset('fzero')
```

Parâmetros de saída

- `x` contém o valor da raiz
- `fval` contém o valor da função na raiz
-

`exitflag` - descreve valores de saída do `fzero`	
> 0	indica que a função encontrou a raiz em X
< 0	indica que a função não convergiu

-

`output` - estrutura que contém informação acerca da otimização	
`iterations`	indica o número de iterações realizadas
`algorithm`	indica o algoritmo usado
`funcCount`	indica o número de avaliações da função

5.1.2 Exercícios resolvidos

1. Determine o zero da função

$$x^3 - 2x - 5$$

que está próximo de 2.

Resolução

```
[x,fval,exitflag,output] = fzero('x^3-2*x-5',2)
```

Solução:

```
       x =   2.0946
    fval =  -8.8818e-16
exitflag =     1
  output =
    intervaliterations: 3
            iterations: 6
             funcCount: 13
             algorithm: 'bisection, interpolation'
               message: [1x45 char]
```

Esta solução mostra que:

$x^* = 2.0946$ e $f(x^*) = -8.8818 \times 10^{-16}$ e que o método convergiu (pois, exitflag=1).

Outra forma de resolução... (criando uma script)

```
[x,fval,exitflag,output]= fzero(@fun, 2)
%funcao
function y = fun(x)
y = x^3-2*x-5;
end
```

2. Determine o zero da função cosseno entre 1 e 2.

   Resolução

   ```
   [x,fval,exitflag] = fzero(@cos,[1,2])
   ```

   Solução:

   ```
   x =   1.5708
   fval=     6.1232e-017
   exitflag =     1
   ```

3. Detremine o zero do exercício anterior (função cosseno entre 1 e 2), mas com uma tolerância em `x` de 1×10^{-3}.

   Resolução

   ```
   op = optimset('TolX',1e-3)
   [x,fval,exitflag,output] = fzero('cos(x)',[1,2],op)
   ```

   Solução:

   ```
   x =   1.5710
   fval=     -1.8225e-004
   exitflag =     1
   ```

```
output =
     iterations: 4
     funcCount: 4
     cgiterations: 3
     algorithm: 'bisection, interpolation'
```

5.2 Comando `roots`

A função `roots` permite encontrar as raízes (os zeros) de um polinómio.

5.2.1 Sintaxe de utilização

A sintaxe de utilização é:

`roots(c)`

em que `c` é um vetor linha com os coeficientes de um polinómio.

Se o vetor linha `c` contém os coeficientes do polinómio

$$P(x) = c_1 x^n + c_2 x^{n-1} + \ldots + c_n x + c_{n+1}$$

então a função `roots(c)` fornece um vetor coluna contendo as n raízes de $P(x) = 0$.

5.2.2 Exercícios resolvidos

1. Calcular as 4 raízes de

$$P(x) = x^4 + 2x^3 - 13x^2 - 14x + 24 = 0$$

Resolução

```
c = [1 2 -13 -14 24];
r = roots(c)
```

```
r =
   -4.0000
    3.0000
   -2.0000
    1.0000
```

2. Calcular as raízes de

$$P(x) = x^5 + 3x^3 + 2x + 4 = 0$$

Resolução

```
c = [1 0 3 0 2 4];
r = roots(c)
```

```
r =
    -0.2804 + 1.6770i
    -0.2804 - 1.6770i
     0.7060 + 1.0616i
     0.7060 - 1.0616i
    -0.8513 + 0.0000i
```

6 Sistemas de equações lineares

6.1 Comandos auxiliares

Comando	Operação
diag	elementos da diagonal principal de uma matriz
triu	matriz triangular superior de uma matriz
tril	matriz triangular inferior de uma matriz
sort	ordenar de forma ascendente
sortrows	ordenar linhas de forma ascendente
sum	soma dos elementos de um vetor ou matriz
prod	produto dos elementos de um vetor ou matriz
norm(x)	norma 2 de uma matriz ou vetor
norm(x,1)	norma 1 de uma matriz ou vetor
norm(x,2)	norma 2 de uma matriz ou vetor
norm(x,inf)	norma infinita de uma matriz ou vetor

Diagonal de uma matriz e comandos triu e tril

```
>> A                    diag(A)        >> triu(A)           >> tril(A)
  2     2     3          2            2     2     3        2     0     0
  4     5     6          5            0     5     6        4     5     0
  7     8     9          9            0     0     9        7     8     9
```

Soma das colunas ou linhas de uma matriz

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

 A =                 >> sum(A)          >> sum(A,2)        >> sum(A')
 2 2 3               ans =              ans =              ans =
 4 5 6               13 15 18           7 15 24            7 15 24
 7 8 9
```

Norma de vetores ou matrizes

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

norma 2

```
>> norm(u,2)
ans = 3.7417

>> norm(A)
ans = 16.9587
```

norma 1

```
>> norm(u,1)
ans = 6

>> norm(A,1)
ans = 18
```

norma infinita

```
>> norm(u,inf)
ans = 3

>> norm(A,inf)
ans = 24
```

Determinante de uma matriz

```
A =
2 2 3
4 5 6
7 8 9
>> det(A)
ans = -3
```

```
B =
2 2 2
3 3 3
4 4 4
>> det(B)
ans = 0
```

Inversa de uma matriz

```
A =
2 2 3
4 5 6
7 8 9
>> inv(A)
ans =
1.0000 -2.0000 1.0000
-2.0000 1.0000 0
1.0000 0.6667 -0.6667
```

```
B =
2 2 2
3 3 3
4 4 4
>> inv(B)
Warning: Matrix is singular.
ans =
Inf Inf Inf
Inf Inf Inf
Inf Inf Inf
```

6.2 Resolução de sistemas de equações lineares

Resolução de $\boxed{\text{Ax} = \text{B}}$ sendo A e B matrizes.

```
A =
2 2 3
4 5 6
7 8 9
```

```
B =
2 2 2
3 3 3
4 4 4
```

```
>> A\B
    0.0000    0.0000    0.0000
   -1.0000   -1.0000   -1.0000
    1.3333    1.3333    1.3333
```

Método de Eliminação de Gauss

Resolução de $\boxed{\text{Ax} = \text{v}}$, sendo `A` uma matriz e `v` um vetor coluna.

```
A =
2 2 3
4 5 6
7 8 9
```

```
v =
1
2
3
```

```
>> A\v
    0.0000
         0
    0.3333
```

7 Sistemas de equações não lineares

O comando utilizado para resolver sistemas de equações não lineares é denominado `fsolve` e localiza-se na **toolbox de otimização** (esta toolbox tem de estar instalada para o comando funcionar).

7.1 Sintaxe de utilização

A sua sintaxe de utilização é:

```
[X,FVAL]=FSOLVE(FUN,X0,...)
[X,FVAL,EXITFLAG]=FSOLVE(FUN,X0,...)
[X,FVAL,EXITFLAG,OUTPUT]=FSOLVE(FUN,X0,options,...)
[X,FVAL,EXITFLAG,OUTPUT,JACOB]=FSOLVE(FUN,X0,options,...)
```

Tem os seguintes algoritmos implementados:'trust-region-dogleg' (default), 'trust-region', e 'levenberg-marquardt'.

Parâmetros de entrada

- `x0` é um vetor próximo da solução
- `fun` é o nome função com o sistema não linear $F(x) = 0$
-

`Options` - definição de parâmetros	
`DerivativeCheck`	compara o Jacobiano com diferenciação finita
`Display`	Nível de apresentação
`off`	não apresenta nada
`iter`	apresenta resultados em cada iteração
`final`	(default) apresenta apenas o resultado final
`Jacobian`	Definição do Jacobiano pelo utilizador ou não
`MaxFunEvals`	Número máximo de avaliações da função
`MaxIter`	Número máximo de iterações
`TolFun`	Tolerância de paragem relativamente a FUN
`TolX`	Tolerância de paragem relativamente a X

Para ver as opções disponíveis fazer:

`optimset('fsolve')` ou `optimoptions('fsolve')`

Parâmetros de saída

- `x` contém o de soluções
- `fval` contém o vetor das funções na solução
-

`exitflag` - descreve valores de saída do `fsolve`	
> 0	indica que convergiu para uma solução X
$= 0$	indica que atingiu o `'MaxFunEvals'` ou `'MaxIter'`
< 0	indica que não convergiu para uma solução

-

`output` - estrutura que contém informação acerca da otimização	
`iterations`	indica o número de iterações realizadas
`funcCount`	indica o número de avaliações da função
`algorithm`	indica o algoritmo usado
`cgiiterations`	indica o número de iterações GC realizadas (se "large-scale")
`stepsize`	indica o comprimento de passo final (se "medium-scale")
`firstorderopt`	condição de otimalidade de 1^a ordem (se "large-scale")

7.2 Exercícios resolvidos

1. Resolva o seguinte sistema de equações não lineares, apresentando os resultados em cada iteração

$$\begin{cases} 2x_1 - x_2 - e^{-x_1} = 0 \\ -x_1 + 2x_2 - e^{-x_2} = 0 \end{cases}$$

Resolução

Para resolver este sistema no MATLAB, criar uma script, por exemplo, denominada 'teste1.m', com os seguintes comandos:

```
x0=[-1 ; -1]; % estimativas iniciais para x(1) e x(2)
options=optimset('Display','iter'); % mostra a solução em cada it.
[x,fval,exitflag,output]=fsolve(@fun1,x0,options) % resolve
%funcao
function F=fun1(x)
  F(1)=2*x(1)-x(2)-exp(-x(1));
  F(2)=-x(1)+2*x(2)-exp(-x(2));
end
```

Depois, fazer 'Run' ou na janela de comando do MATLAB, digitar `>> teste1`

Ou (em substituição de criar uma script), na janela de comandos do MATLAB digitar os comandos apresentados.

Em primeiro lugar, a solução apresenta alguns resultados por iteração, uma vez que se colocou a opção de "Display" por "iter".

Iter.	Func count	f(x)	Norm of step	First-order optimality	Trust-region radius
0	3	27.6512		13.8	1
1	6	5.33464	1	3.82	1
2	9	0.137328	0.986918	0.437	1
3	12	0.000122467	0.222302	0.0123	2.47
4	15	9.99189e-011	0.00704881	1.11e-005	2.47
5	18	6.63847e-023	6.37844e-006	9.03e-012	2.47

Depois, o programa retorna os seguintes resultados:

```
x =
    0.5671
    0.5671

fval =
   1.0e-11 *
   -0.5761   -0.5761

exitflag =     1

output =
       iterations: 5
        funcCount: 18
        algorithm: 'trust-region-dogleg'
    firstorderopt: 9.0288e-12
```

Esta solução mostra que o método convergiu (exitflag=1) e

$$x^* = \begin{pmatrix} 0.5671 \\ 0.5671 \end{pmatrix} \quad f(x^*) = \begin{pmatrix} -0.5671 \times 10^{-11} \\ -0.5671 \times 10^{-11} \end{pmatrix}$$

2. Resolver o o sistema de equações não lineares anterior

$$\begin{cases} 2x_1 - x_2 - e^{-x_1} = 0 \\ -x_1 + 2x_2 - e^{-x_2} = 0 \end{cases}$$

com as seguintes opções $TolX = 10^{-3}$ e $TolFun = 10^{-2}$.

Resolução

```
options=optimset('TolFun',1e-2,'TolX',1e-3);
[x,fval,exitflag,output]=fsolve(@fun1,[-1 ; -1],options)
%funcao
function F=fun1(x)
  F(1)=2*x(1)-x(2)-exp(-x(1));
  F(2)=-x(1)+2*x(2)-exp(-x(2));
end
```

Solução:

```
x =
    0.5671
    0.5671

fval =
   1.0e-05 *
   -0.7068   -0.7068

exitflag =     1

output =
       iterations: 4
        funcCount: 15
        algorithm: 'trust-region-dogleg'
    firstorderopt: 1.1077e-05
          message: 'Equation solved.'
```

Note-se que, apesar da solução x ser amesma, o valor da função é maior e fez menos iterações.

3. Resolver o sistema de equações não lineares

$$\begin{cases} 2x_1 - x_2 - e^{-x_1} = 0 \\ -x_1 + 2x_2 - e^{-x_2} = 0 \end{cases}$$

fornecendo a matriz do Jacobiano.

Resolução

Se quisermos dar a matriz do jacobiano temos que redefinir a função.

```
options=optimset('Jacobian','on');
x0=[-1 ; -1];
[x,fval,exitflag,output]=fsolve(@fun1,x0,options)
%funcao
function [F,G]=fun1(x)
     F(1)=2*x(1)-x(2)-exp(-x(1));
     F(2)=-x(1)+2*x(2)-exp(-x(2));
     if (nargout>1)
            G=[2+exp(-x(1)) -1; -1 2+exp(x(2))];
     end
end
```

8 Equações Diferenciais

Os comandos do MATLAB para resolver uma equação diferencial do tipo

$$\frac{dy}{dx} = f(x, y)$$

ou um sistema de equações diferenciais ordinárias

$$\begin{cases} \frac{dy_1}{dx} &= f_1(x, y_1, y_2, \ldots, y_n) \\ \frac{dy_2}{dx} &= f_2(x, y_1, y_2, \ldots, y_n) \\ & \ldots \\ \frac{dy_n}{dx} &= f_n(x, y_1, y_2, \ldots, y_n) \end{cases}$$

são:

- `ode23` - Método de Runge-Kutta $2^{\underline{a}}/3^{\underline{a}}$ ordem.
- `ode45` - Método de Runge-Kutta $4^{\underline{a}}/5^{\underline{a}}$ ordem.

8.1 Sintaxe de utilização

A sintaxe de utilização é:

```
[T,Y] = ode23(ODEFUN,TSPAN,Y0,OPTIONS)
[T,Y] = ode45(ODEFUN,TSPAN,Y0,OPTIONS)
```

- `T` é um vetor com os valores da variável independente
- `Y` é um vetor com os valores da solução
- `ODEFUN` é o nome da função que contém a equação diferencial deve ser definida numa M-file ou através do comando "inline".

- `TSPAN` é um vetor com os valores da variável independente, i.e., `TSPAN = [T0 TFINAL]` ou `TSPAN = [T0 T1 ... TFINAL]`
- `Y0` é o vetor das condições iniciais para a variável dependente
- `OPTIONS` são opções de resolução (opcional)

Para ver as opções disponíveis fazer:

```
odeset
```

Alguns dos parâmetros da função `odeset` mais utilizados são:

`Options` - definição de parâmetros	
`OutputFcn`	Permite a visualização gráfica dos resultados
`AbsTol`	Tolerância do erro absoluto (por defeito é 1e-6)
`InitialStep`	Passo inicial
`MaxStep`	Passo máximo
`RelTol`	Tolerância do erro relativo (por defeito é 1e-3)

8.2 Exercícios resolvidos

1. A taxa de escoamento de um líquido que está dentro de um cilindro vertical através de um buraco na base é dada por:
$$\frac{dy}{dt} = -kt\sqrt{y}$$
onde $y(t)$ é a altura da água no tanque ao fim de t minutos. Supondo que $k = 0.1$ e que a altura inicial da água é de 2 metros determine a altura da água no tanque ao fim de 2 minutos.

   Use um passo de 0.5 e um método de $4^{\underline{a}}/5^{\underline{a}}$ ordem.

   Resolução

```
[t,y]=ode45(@odefun,[0 0.5 1 1.5 2],[2])
function ydot=odefun(t,y)
ydot=-0.1.*t.*sqrt(y);
end
```

   Solução

```
t =
         0
    0.5000
    1.0000
    1.5000
    2.0000
```

```
y =
    2.0000
    1.9824
    1.9299
    1.8441
    1.7272
```

2. Consideremos a equação diferencial de 2ª ordem denominada "Equação de Van der Pol":

$$\frac{d^2y}{dt^2} + (y^2 - 1)\frac{dy}{dt} + y = 0$$

Resolva o sistema de equações diferenciais

- através do Método de Runge-Kutta 4ª/3ª ordem,
- no intervalo $0 \leq t \leq 20$ (i.e, com escolha adaptativa do comprimento do passo)
- com as seguintes condições iniciais $y(0) = 0$ e $\frac{dy(0)}{dt} = 0.25$

Resolução

Primeiro deve-se rescrever a equação diferencial de segunda ordem como um sistema de equações diferenciais de primeira ordem

$$\begin{cases} \frac{dy_1}{dt} &= y_2 \\ \frac{dy_2}{dt} &= y_2\left(1 - y_1^2\right) - y_1 \end{cases}$$

De seguida, construa a script de resolução, da seguinte forma:

```
tspan = [0 20];
y0 = [0 0.25];
[t,y] = ode23(@volpol, tspan, y0)
[t,y] %para ver a solucao em formato tabela
%funcao
function ydot=volpol(t,y)
      ydot(1)=y(2);
      ydot(2)=y(2)*(1- y(1)^2) - y(1);
      ydot=ydot';
end
```

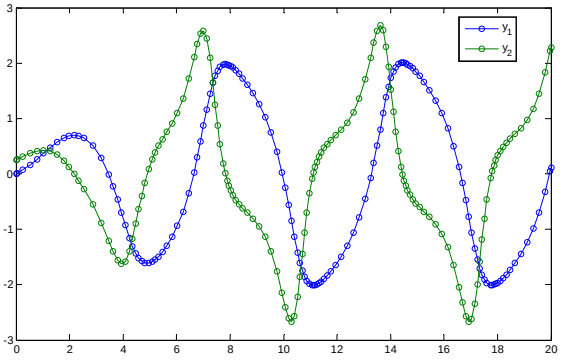
Solução:

t	y_1	y_2
0	0	0.2500
0.0003	0.0001	0.2501
0.0019	0.0005	0.2505
0.0099	0.0025	0.2525
0.0499	0.0128	0.2625
0.2499	0.0703	0.3117
0.5207	0.1631	0.3720
0.7772	0.2645	0.4139
1.0230	0.3688	0.4299
...	...	...
18.6575	-1.6053	0.7192
18.8696	-1.4417	0.8263
19.1018	-1.2338	0.9725
19.3339	-0.9862	1.1723
19.5565	-0.6970	1.4429
19.7832	-0.3288	1.8282
19.9708	0.0505	2.2231
20.0000	0.1164	2.2860

Se executar o comando `ode23` sem argumentos de saída

```
tspan = [0 20];
y0 = [0 0.25];
ode23(@volpol, tspan, y0)
%funcao
function ydot=volpol(t,y)
      ydot(1)=y(2);
      ydot(2)=y(2)*(1- y(1)^2) - y(1);
      ydot=ydot';
end
```

então obtém-se como solução, apenas o gráfico seguinte:



3. Suponha agora que pretende resolver o mesmo sistema de equações diferenciais

   - através do Método de Runge-Kutta $4^{\text{a}}/5^{\text{a}}$ ordem,
   - no intervalo $0 \leq t \leq 20$ com comprimento do passo $h = 2$,
   - com as seguintes condições iniciais $y(0) = 0$ e $\dfrac{dy(0)}{dt} = 0.25$

   Resolução

```
tspan = 0:2:20;
y0 = [0 0.25];
[t,y] = ode45(@volpol, tspan, y0)
[t,y] %para ver a solucao em formato tabela
%funcao
function ydot=volpol(t,y)
      ydot(1)=y(2);
      ydot(2)=y(2)*(1- y(1)^2) - y(1);
      ydot=ydot';
end
```

   Solução

t	y_1	y_2
0	0	0.2500
2.0000	0.6876	0.1172
4.0000	-0.7872	-1.6217
6.0000	-0.9784	1.0624
8.0000	1.9535	-0.2744
10.0000	-0.0948	-2.2627
12.0000	-1.6047	0.7194
14.0000	1.6993	1.6143
16.0000	0.9948	-1.1648
18.0000	-1.9766	0.3223
20.0000	0.0813	2.2548

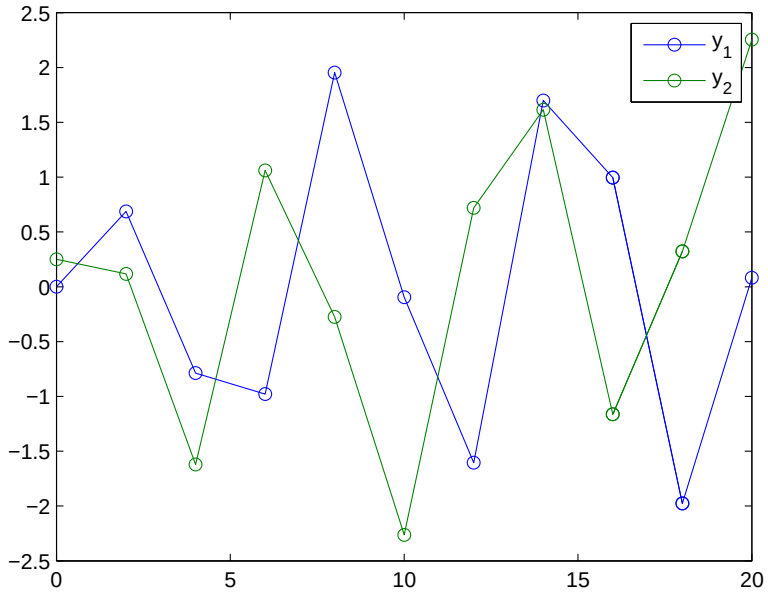
4. Suponha agora que pretende resolver o mesmo sistema de equações diferenciais e com os valores iniciais definidos em 3. No entanto, pede-se ainda a visualização gráfica da solução.

   Resolução

   Para a resolução com os parâmetros anteriores e visualização gráfica, executar

```
op = odeset('OutputFcn',@odeplot);
tspan = 0:2:20;
y0 = [0 0.25];
[t,y] = ode45(@volpol, tspan, y0, op)
legend('y_1','y_2') %coloca legenda no grafico
%funcao
function ydot=volpol(t,y)
      ydot(1)=y(2);
      ydot(2)=y(2)*(1- y(1)^2) - y(1);
      ydot=ydot';
end
```

cujo esboço se apresenta na figura seguinte:



Note-se que este gráfico não tem boa caracterização das duas curvas, uma vez que o número de pontos que está a ser usado é pequeno (ou seja, tem apenas os pontos em `tspan = 0:2:20`)

5. Suponha agora que pretende resolver o mesmo sistema de equações diferenciais, usando os valores iniciais definidos em 3. No entanto, pretende-se resolver o sistema de equações com uma tolerância relativa de 1×10^{-4}.

Resolução

```
options=odeset('RelTol',1e-4)
tspan = 0:2:20;
y0 = [0 0.25];
[t,y] = ode45(@volpol, tspan, y0, options);
%funcao
function ydot=volpol(t,y)
      ydot(1)=y(2);
      ydot(2)=y(2)*(1- y(1)^2) - y(1);
      ydot=ydot';
end
```

Solução:

t	y_1	y_2
0	0	0.2500
2.0000	0.6876	0.1163
4.0000	-0.7899	-1.6223
6.0000	-0.9759	1.0663
8.0000	1.9531	-0.2734
10.0000	-0.0942	-2.2618
12.0000	-1.6041	0.7196
14.0000	1.7002	1.6102
16.0000	0.9936	-1.1657
18.0000	-1.9688	0.3242
20.0000	0.1075	2.2772

9 Aproximação dos mínimos quadrados

O método dos mínimos quadrados permite fazer o ajuste de curvas, para um conjunto de pontos, por exemplo, adquiridos numa experiência.

O método analítico dos mínimos quadrados, permite encontrar o melhor modelo que se ajusta ao conjunto de pontos, podendo acontecer que nenhum dos pontos pertença ao mesmo.

De notar que este método é muito distinto da interpolação, dado que na interpolação a curva ajustada passa por todos os pontos.

9.1 Comando `polyval`

Este comando permite determinar os valores de um polinómio, para um dado conjunto de pontos.

A sua sintaxe de utilização é:

```
y = polyval(coef,x)
```

- `coef` é um vetor com os coeficientes do polinómio

- `x` contém os valores para os quais desejamos avaliar o polinómio
- `y` contém os valores do polinómio

9.2 Exemplos de utilização

1. Considere os seguinte polinómio

$$p(x) = 3x^2 + 2x + 5.$$

Pretende-se determinar os valores do polinómio para $x = 4$.

```
coef = [3, 2, 5];
f = polyval(coef,4)
```

Solução: `f = 61`

Estes comandos também podem ser combinados num só:

```
f = polyval([3, 2, 5],4);
```

2. Suponha que quer saber o valor do polinómio

$$g(x) = -x^5 + 3x^3 - 2.5x^2 - 2$$

para todos os valores de `x` no intervalo de [0,3], com espaçamento de 0.5.

```
x = 0:0.5:2;
coef = [-1,0,3,-2.5,0,-2];
g = polyval(coef,x)
```

Solução:

```
g =   -2.0000   -2.2813   -2.5000   -5.0938  -20.0000
```

em que

```
g(0) = -2.0000
g(0.5) = -2.2813
g(1) = -2.5
g(1.5) = -5.0938
g(2) = -20.0000
```

9.3 Comando `polyfit`

Permite calcular analiticamente os coeficientes de um polinómio a ajustar ao conjunto de pontos. Calcula também o resíduo.

A sua sintaxe de utilização é:

```
[p,S] = polyfit(x,y,N)
```

- `x` é um vetor com os pontos
- `y` é um vetor com os valores da função nos pontos
- `N` representa o grau do polinómio
- `p` é um vetor com os coeficientes do polinómio $P(x) = c_1x^n + c_2x^{n-1} + \ldots + c_nx + c_{n+1}$
- `S` é uma estrutura para usar com `polyval` para obter uma estimativa dos erros. A estrutura contém os seguintes campos:
  - `R`, `df`
  - `normr` - é a norma dos resíduos, i.e., a raiz quadrada da soma do quadrado dos erros.

9.4 Exemplo de utilização

Determine o polinómio de grau 3 que melhor se ajusta aos dados da tabela, no sentido dos mínimos quadrados.

x_i	0	1	2	3	4	5
f_i	0	20	60	68	77	110

Comandos a executar:

```
x = [0,1,2,3,4,5];
y = [0,20,60,68,77,110];
[p,S] = polyfit(x,y,3)
```

Solução:
```
p = 1.1019 -9.3175 41.1918 -3.0556
S =
    R: [4x4 double]
    df:  2
    normr:  15.3853
```

Polinómio: $p_3(x) = 1.1019x^3 - 9.3175x^2 + 41.1918x - 3.0556$

A soma do quadrado dos resíduos calcula-se através de:

```
S.normrˆ2
```

Solução: `236.7063`

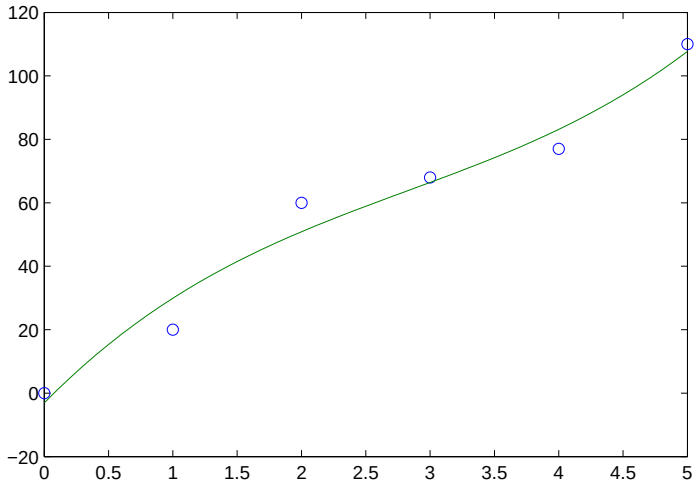
Para calcular o valor do polinómio no ponto $x = 2.5$, ou seja para calcular $p_3(2.5)$, deve-se fazer o seguinte comando

```
polyval(p,2.5)
```

9.5 Representação gráfica do polinómio

Para fazer o gráfico com os vários pontos (criando um novo vetor de pontos`newx`) e o polinómio encontrado (que é obtido em `newy`), basta fazer:

```
newx = 0:0.1:5;
newy = polyval(p,newx);
plot(x, y, 'o', newx, newy, 'g');
```



9.6 Comando `lsqcurvefit`

Este comando permite determinar os coeficientes de um modelo linear para ajustar a um dado conjunto de pontos.

A sua sintaxe de utilização é:

```
[m,resnorm] = lsqcurvefit(fun,c0,xdata,ydata)
```

- `fun` é uma função com o modelo
- `c0` é um vetor com os valores iniciais ods coeficientes
- `xdata` é um vetor com os pontos
- `ydata` é um vetor com os valores da função nos pontos
- `m` é um vetor com os coeficientes do modelo
- `resnorm` contém o valor da soma do quadrado dos resíduos

9.7 Exemplo de utilização

Determine os coeficientes do modelo que melhor se ajusta aos dados da tabela, no sentido dos mínimos quadrados, e calcule a soma dos quadrados dos resíduos. Use o vetor $(1, 1)$ para aproximação inicial aos parâmetros.

$$m(x) = c_1 e^{(1-0.1x)} + c_2 e^{(0.1x-1)}$$

x	0	1	2	3	4	5
y	0	20	60	68	77	110

Começar por criar um ficheiro com o modelo:

```
xdata = [0,1,2,3,4,5];
ydata = [0,20,60,68,77,110];
x0=[1,1];
[m,S] = lsqcurvefit(@fun,x0,xdata,ydata)
%funcao
function f=fun(c,x)
     f=c(1)*exp(1-0.1*x)+c(2)*exp(0.1*x-1);
end
```

Solução:

```
m =  -35.8101  276.8977


S =  376.5176
```

em que os coeficientes do modelo são $c_1 = -35.8101$ e $c_2 = 276.8977$, pelo que o modelo fica

$$m(x) = -35.8101 e^{(1-0.1x)} + 276.8977 e^{(0.1x-1)}.$$

A soma do quadrado dos resíduos, que nos permite avaliar o modelo, é dada por S e tem o valor de 376.5176, pelo que não nos parece um modelo que ajuste bem os dados.

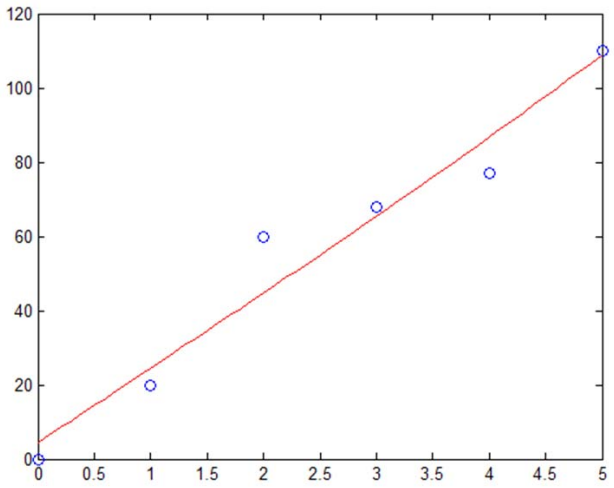
Para calcular o valor do modelo no ponto $x = 2.5$, ou seja para calcular $m(2.5)$, deve fazer-se o seguinte comando

```
fun(m,2.5)
```

9.8 Representação gráfica do modelo

Para fazer o gráfico com os vários pontos (criando um novo vetor de pontos`newx`) e o modelo encontrado (que é obtido em `newy`), basta fazer:

```
newx = 0:0.1:5;
newy = fun(m,newx);
plot(x, y, 'o', newx, newy, 'r');
```



9.9 Exercícios resolvidos

1. Determine o polinómio de grau 2 que melhor se ajusta aos dados da tabela, no sentido dos mínimos quadrados, e calcule a soma dos quadrados dos resíduos.

x_i	2.5	3.5	4	6	7
f_i	0	25	45	65	75

Resolução

Comandos a executar:

```
x = [2.5, 3.5, 4, 6, 7];
y = [0, 25, 45, 65, 75];
[p,S] = polyfit(x,y,2)
S.normr^2
```

O polinómio é: $p_2(x) = -3.3464x^2 + 47.9971x - 98.8069$

A soma dos quadrados dos resíduos é: 55.9580.

2. Considere a seguinte tabela de valores

x_i	-1	1	2	5.5	7	10	15
f_i	0	20	60	88	120	150	100

   a. Determine um polinómio de grau 2, um polinómio de grau 3 e um polinómio de grau 4 para ajustar aos dados da tabela, no sentido dos mínimos quadrados.

   b. Determine os coeficientes do modelo que melhor se ajusta aos dados da tabela, no sentido dos mínimos quadrados, usando o vetor $(1, 1, 1)$ para aproximação inicial.

$$m(x) = c_1 x^2 - c_2 \sin(x) + c_3$$

c. Calcule uma aproximação de cada polinómio e do modelo para $x = 3$.

d. Calcule a soma dos quadrados dos resíduos, para cada um dos polinómios determinados. Indique qual deles aproxima melhor os dados da tabela. Justifique

e. Represente graficamente os pontos dados e os vários polinómios calculados (a cores diferentes). Não se esqueça de legendar as curvas de forma a identificar cada um dos polinómios.

Resolução

a. Para determinar os três polinómios executam-se os seguintes comandos:

```
x = [-1,1,2,5.5,7,10,15];
y = [0,20,60,88,120,150,100];
[p2,S2] = polyfit(x,y,2)
[p3,S3] = polyfit(x,y,3)
[p4,S4] = polyfit(x,y,4)
```

Solução

```
p2 =   -1.0932   22.6613   13.0522
S2 =
        R: [3x3 double]
       df: 4
    normr: 32.8540

p3 =   -0.0956    0.9314   13.3587   13.7053
S3 =
        R: [4x4 double]
       df: 3
    normr: 22.3419

p4 =   -0.0111    0.1950   -1.1758   16.7218   16.3203
S4 =
        R: [5x5 double]
       df: 2
    normr: 19.9306
```

Os polinómios determinados são:

- $p_2(x) = -1.0932x^2 + 22.6613x + 13.0522$
- $p_3(x) = -0.0956x^3 + 0.9314x^2 + 13.3587x + 13.7053$
- $p_4(x) = -0.0111x^4 + 0.1950x^3 - 1.1758x^2 + 16.7218x + 16.3203$

b. Para determinar os coeficientes do modelo, executam-se os seguintes comandos, tendo em consideração que já foram definidos os vetores x e y:

```
x0=[1,1,1];
[m,S] = lsqcurvefit(@fun,x0,x,y)
function f=fun(c,x)
f=c(1)*x.^2-c(2).*sin(x)+c(3);
end
```

Solução:

```
m =    0.3772      -5.7255       55.542
S =    11707
```

em que os coeficientes do modelo são $c_1 = 0.3772$, $c_2 = -5.7255$ e $c_3 = 55.542$, pelo que o modelo é dado por

$$m(x) = 0.3772x^2 - -5.7255\sin(x) + 55.542$$

c. Para determinar o valor de cada polinómio em $x = 3$ executam-se os seguintes comandos:

```
polyval(p2,3)
polyfit(p3,3)
polyfit(p4,3)
```

d. Para determinar o valor do modelo em $x = 3$ executa-se o comando:

```
fun(m,3)
```

e. A soma dos quadrados dos resíduos, para cada um dos polinómios determina-se usando:

```
S2.normr^2
S3.normr^2
S4.normr^2
S
```

Solução

Logo, a soma dos quadrados dos resíduos para

- o polinómio de grau 2 é 1.0794e+03
- o polinómio de grau 3 é 499.1589
- o polinómio de grau 4 é 397.2293
- o modelo linear nos parâmetros é 11707 (saiu em S)

Concluímos que o modelo que melhor ajusta os dados da tabela é o polinómio de grau 4, pois é o que apresenta menor resíduo.

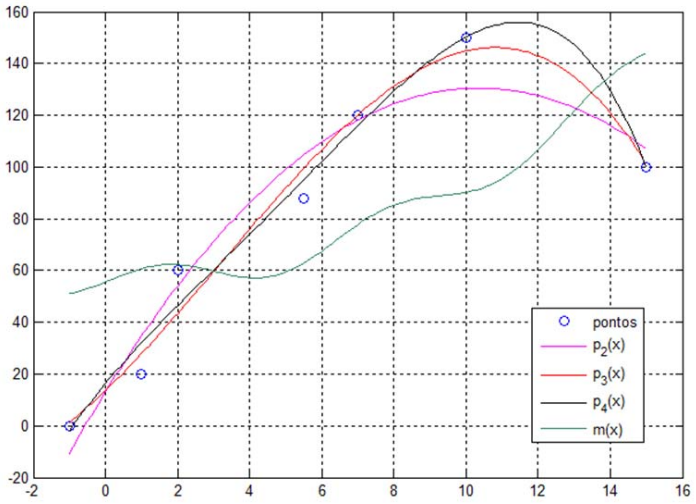
f. Para a representação gráfica dos pontos dados e dos polinómios calculados executam-se os seguintes comandos:

```
newx = -1:0.1:15;
newy2 = polyval(p2,newx);
newy3 = polyval(p3,newx);
newy4 = polyval(p4,newx);
newy5 = fun(m,newx);
plot(x,y,'o',newx,newy2,'m',newx,newy3,'r',newx,newy4,'k',newx,newy5,'g');
grid;
legend('pontos','p_2(x)','p_3(x)','p_4(x)','m(x)')
```

Solução

A azul encontram-se os pontos da tabela; a cor de rosa o polinómio de grau 2; a vermelho o polinómio de grau 3; a preto o polinómio de grau 4 e a verde o modelo $m(x)$.

A representação gráfica dos 3 polinómios e dos pontos é:



10 Interpolação spline

Uma das técnicas mais utilizadas para estimar o comportamento de uma determinada função entre dois pontos considerados, é a interpolação.

Uma spline cúbica é uma curva suave ajustada ao conjunto de pontos considerados que, entre cada par de pontos, é determinada por um polinómio do 3^o grau.

Esta técnica permite obter uma curva suave, ao invés da interpolação linear, onde os pontos são unidos com segmentos de reta.

O comando `spline` permite determinar no MATLAB uma spline cúbica (not-a-knot).

10.1 Spline cúbica (not-a-knot)

10.1.1 Sintaxe de utilização

A sintaxe de utilização do comando `spline` (not-a-knot) é:

```
YY = spline(X,Y)
YY = spline(X,Y,XX)
```

- X é um vetor que contém os pontos para a interpolação
- Y contém a função ou os respetivos valores da função
- XX é o valor do ponto (ou vetor de pontos) interpolador

10.1.2 Exemplo de utilização

Considere a tabela que representa a evolução da temperatura de um sólido ao longo do tempo:

tempo	0	2	3	4	5
temperatura	0	20	35	60	110

Para determinar a spline cúbica (not-a-knot) deve primeiro deve definir os vetores `x` (tempo) e `y` (temperatura) com os dados da tabela.

```
x = [0,2,3,4,5];
y = [0,20,35,60,110];
```

Se pretende determinar a temperatura no instante 4.2 segundos, através de

- **Spline cúbica (not-a-knot)**

  ```
  temp = spline(x, y, 4.2)
  ```

  Solução: `temp = 67.4447`

Determinação de cada segmento da spline (not-a-knot)

1. O primeiro comando a fazer é:

   ```
   s = spline(x,y)
   ```

   E aparece a seguinte estrutura:

   ```
   cs_comp =
         form: 'pp'
       breaks: [0 2 3 4 5]
        coefs: [3x4 double]
   ```

```
    pieces: 3
     order: 4
       dim: 1
```

- `pp` - indica que é da forma polinomial por partes
- `breaks` - representa os vários pontos da interpolação
- `coefs` - é a matriz dos coeficientes para construir os vários segmentos

2. Seguidamente, devem ser visualizados os coeficientes das splines, através do comando

```
s.coefs
```

Cuja solução é:

```
0.5392 -1.0294 9.9020 0
0.5392 2.2059 12.2549 20.0000
2.8922 3.8235 18.2843 35.0000
2.8922 12.5000 34.6078 60.0000
```

No subintervalo $[x_i, x_{i+1}]$ a spline é representada por

$$s_i(x) = c_{l1} * (x - x_i)^3 + c_{l2} * (x - x_i)^2 + c_{l3} * (x - x_i) + c_{l4}$$

em que $[c_{l1},\ c_{l2},\ c_{l3},\ c_{l4}]$ corresponde à linha l da matriz dos coeficientes.

3. Depois, é só construir os vários segmentos

- $s_3^{(1)} = 0.5392(x-0)^3 - 1.0294(x-0)^2 + 9.9020(x-0) + 0,\ x \in [0,2]$
- $s_3^{(2)} = 0.5392(x-2)^3 + 2.2059(x-2)^2 + 12.2549(x-2) + 20,\ x \in [2,3]$
- $s_3^{(3)} = 2.8922(x-3)^3 + 3.8235(x-3)^2 + 18.2843(x-3) + 35,\ x \in [3,4]$
- $s_3^{(4)} = 2.8922(x-4)^3 + 12.5000(x-4)^2 + 34.6078(x-4) + 60,\ x \in [4,5]$

Como determinar uma estimativa da spline (not-a-knot)

Para estimar o valor da spline no ponto 4.2 podemos fazer:

```
cs = spline(x,y)
a=ppval(cs,4.2)
```

ou

```
a=spline(x,y,4.2)
```

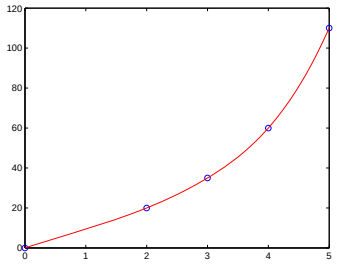
Se pretende determinar o valor da spline num vetor de pontos, deve fazer:

```
b = spline(x,y,[1 1.5 2.5 3.5 4.5])
```

10.1.3 Representação gráfica de spline (not-a-knot

Se pretender determinar e representar graficamente uma spline noutro intervalo de tempo, deve gerar um novo vetor `newx` e usá-lo como 3^o argumento do comando spline.

```
newx = [0:.1:10];
newy = spline(x, y, newx);
plot(x,y,'o',newx,newy,'r');
grid;
```



10.2 Spline cúbica completa

Para construir uma spline cúbica completa

- devem ser indicados dois valores extra para y.
- esses valores são as derivadas nos extremos.

A sintaxe de utilização da spline cúbica completa é:

```
spline(xx,[f'0 yy f'n])
```

em que f'_0 e f'_n são os valores das derivadas nos extremos.

Cálculo dos valores das derivadas nos extremos

- Caso exista a expressão da função $f(x)$:

  Calcular $f'_0 = f'(x_0)$ e $f'_n = f'(x_n)$

- Caso não exista a expressão da função $f(x)$:

  1. Calcular as dd1 para aproximar f'_0 e f'_n (usando o 2^o e penúltimos pontos).
  2. Omitir o 2^o e penúltimos pontos usados para calcular as dd1, da tabela de pontos dados.

10.3 Exercícios resolvidos

1. Considere os dados apresentados para o exemplo anterior, que representam a evolução da temperatura de um sólido ao longo do tempo:

tempo	0	2	3	4	5
temperatura	0	20	35	60	110

Estime o valor da temperatura para o tempo de 4.2, através de uma s**pline cúbica completa**, e apresente o corresponde segmento da spline.

Resolução

1º Calcular $f'_0 = \dfrac{0-20}{0-2} = 10$

2º Calcular $f'_n = \dfrac{60-110}{4-5} = 0.3333$

3º Definir os vetores `xx` e `yy`, omitindo o 2^o e penúltimos pontos.

```
xx = [0,3,5];
yy = [0,35,110];
s_completa = spline(xx, [10 yy 0.3333])
```

Cuja solução é:

```
s_completa =
      form: 'pp'
    breaks: [0 3 5]
     coefs: [2x4 double]
    pieces: 2
     order: 4
       dim: 1
```

Para determinar o correspondente segmento da spline, fazer:

```
s_completa.coefs
```

resultando em

```
   2.8130   -7.8833   10.0000         0
  -9.0042   17.4333   38.6500   35.0000
```

A primeira linha corresponde aos coeficientes do segmento da spline para `tempo`$\in [0,3]$ e a segunda linha para `tempo`$\in [3,5]$. Assim, o segmento da spline que permite estimar a temperatura para um tempo de 4.2 é o segundo segmento, que tem a seguinte expressão:

$$s_3^{(2)} = -9.0042(x-3)^3 + 17.4333(x-3)^2 + 38.6500(x-3) + 35.0000, \;\; x \in [3,5]$$

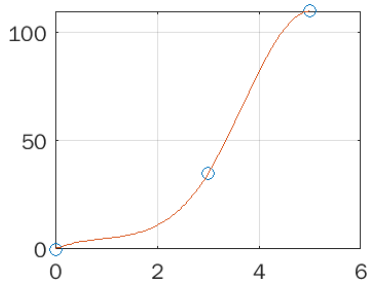
Para determinar a temperatura no instante 4.2 segundos, através de uma **spline cúbica completa**

```
temp = spline(xx, [10 yy 0.3333], 4.2)
```

Solução: `temp = 71.9525`

Para representar graficamente a **spline cúbica completa** no intervalo [0,5].

```
xx = [0,3,5];
yy = [0,35,110];
newx = 0:0.05:5;
newy = spline(xx,[10 yy 0.3333],newxx);
plot(xx,yy,'o',newx,newy);
grid;
```



2. Considere a seguinte tabela de valores

tempo	0	1	2	3	4	5
temperatura	0	20	60	68	77	110

   a. Pretende-se usar uma **spline cúbica (not-a-knot)** para estimar o valor da temperatura quando `x=4.6` segundos:

   ```
   x = [0,1,2,3,4,5];
   y = [0,20,60,68,77,110];
   temp1 = spline(x,y,4.6)
   ```

   Solução: `temp1 = 92.9813`

   b. Determine, simultaneamente, a temperatura em `xx=2.6` e `xx=4.9` segundos através de uma **spline cúbica** (not-a-knot)

   ```
   x = [0,1,2,3,4,5];
   y = [0,20,60,68,77,110];
   temp3 = spline(x,y,[2.6,4.9])
   ```

   Solução: `temp3 = [67.3013,105.2020]`

   c. Estime o valor da temperatura quando `xx=4.6` usando uma **spline cúbica completa**.

```
xx = [0,2,3,5];
yy = [0,60,68,110];
temp2 = spline(xx,[20 yy 33],4.6)
```

Solução: `temp2 = 97.6480`

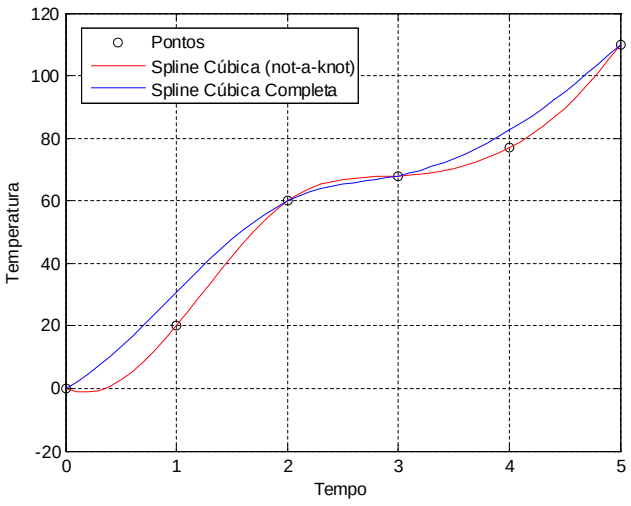
d. Determine, simultaneamente, a temperatura em $x = 2.6$ e $x = 4.9$ segundos usando uma **spline cúbica completa**

```
xx = [0,2,3,5];
yy = [0,60,68,110];
temp4 = spline(xx,[20 yy 33],[2.6,4.9])
```

Solução: `temp4 = 65.7960 106.7517`

e. Represente graficamente a curva da **spline cúbica (not-a-knot)** e a da **spline cúbica completa** abrangendo um conjunto alargado de valores (gerando um novo vetor `newx` e colocando-o como terceiro argumento do comando spline).

```
x = [0,1,2,3,4,5];
y = [0,20,60,68,77,110];
newx = 0:  0.1 :5;
newy = spline(x,y,newx);
xx = [0,2,3,5];
yy = [0,60,68,110];
newyy = spline(xx,[20 yy 33],newx);
plot (x,y,'ok',newx,newy,'r',newx,newyy,'b');
legend('Pontos','Spline Cúbica (not-a-knot)','Spline Cúbica Completa')
xlabel('Tempo');
ylabel('Temperatura');
grid;
```



11 Integração numérica

O MATLAB dispõe de algumas funções para calcular numericamente o integral de uma função:

- Comando `trapz`

  Baseado no método trapezoidal

- Comando `integral`

  Baseado no método da quadratura adaptativa global

11.1 Comando `trapz`

O comando `trapz(x,y)` calcula uma aproximação ao integral de $y = f(x)$.

11.1.1 Sintaxe de utilização

A sintaxe de utilização do comando `trapz` é a seguinte:

```
Z = trapz(Y)
Z = trapz(X,Y)
```

Se os valores de `X` tiverem um espaçamento unitário, então `X` pode ser omitido.

- `Z` retorna uma aproximação ao integral
- `X` é um vetor com os pontos de integração
- `Y` é um vetor com os valores da função nos pontos `X`.

Este comando é usado quando se tem uma tabela de pontos.

Os valores de `Y` podem ser especificados a partir de:

- Diretamente de um vetor de pontos

```
x = [0 1 2 3 4];
y = [1 3 5 2 8];
Z = trapz(x,y)
```

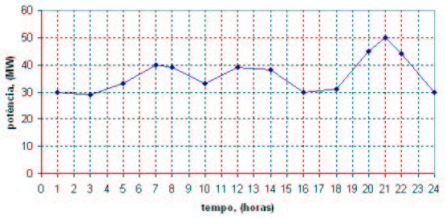
- Primeiro construir o vetor de pontos

```
x = 0:0.1:5;
y = x.^2+2.*x;
Z = trapz(x,y)
```

Nota: como `X` é um vetor, a função deve fazer operações elemento a elemento.

11.1.2 Exercício resolvido

1. A curva de carga típica de uma determinada cidade (MW) está representada na figura



ou pela correspondente tabela

tempo	1	3	5	7	8	10	12	14	16	18	20	21	22	24
potência	30	29	33	40	39	33	39	38	30	31	45	50	44	30

Estime o consumo de energia diário desta cidade.

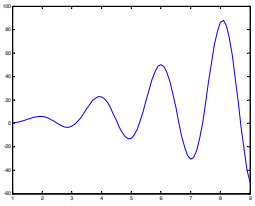
Resolução

```
x = [1 3 5 7 8 10 12 14 16 18 20 21 22 24]
y = [30 29 33 40 39 33 39 38 30 31 45 50 44 30]
consumo = trapz(x,y)
```

Solução: `consumo = 828`

2. Considere a função

$$f(x) = \cos(3x+1)x^2 + x^{1.5}$$



Calcular

$$\int_1^9 \left(\cos(3x+1)x^2 + x^{1.5}\right) dx$$

a) usando um conjunto de pontos com espaçamento 1.

```
x = 1:1:9;
y = cos(3.*x+1).*x.^2+x.^1.5;
area = trapz(x,y)
```

Solução: `area = 93.8111`

b) usando um conjunto de pontos com espaçamento 0.1.

```
x = 1:0.1:9;
y = cos(3.*x+1).*x.^2+x.^1.5;
area = trapz(x,y)
```

Solução: `area = 102.4429`

c) usando um conjunto de pontos com espaçamento 0.01.

Outra forma de resolver, em que a função a ser integrada pode ser definida através do comando `inline`.

Assim, para calcular o integral acima, basta fazer:

```
x = 1:0.01:9;
g = inline('cos(3.*x+1).*x.^2+x.^1.5');
area = trapz(x,g(x))
```

Solução: `area = 102.5100`

Sabendo que o valor do integral é ≈ 102.5106, o resultado da integração numérica torna-se mais exato, quando se diminui o espaçamento.

d) usando 21 pontos (equivale a 20 subintervaloes, originando $h = (9-1)/20 = 0.4$).

```
x = 1:(9-1)/20:9;
y = cos(3.*x+1).*x.^2+x.^1.5;
area = trapz(x,y)
```

Solução: `area = 101.3909`

11.2 Comando `integral`

Para calcular numericamente um integral de uma função dentro dos limites inferior e superior é através do comando`integral`.

11.2.1 Sintaxe de utilização

A sua sintaxe de utilização é:

```
q = integral(fun,xmin,xmax)
q = integral(fun,xmin,xmax,Name,Value)
```

- `q` retorna o valor da aproximação ao integral
- `fun` é o nome da função (ou a função) a ser integrada
- `xmin` é o limite inferior de integração
- `xmax` é o limite superior de integração
- `Name-Value` especifica argumentos que podem ser usados. Por exemplo, especifica a tolerância (valor predefinido de 10^{-6}).
  - AbsTol - Tolerância do erro absoluto (default=1e-10)
  - RelTol - Tolerância do erro relativo (default=1e-6)

Exemplo:
Calcular:

$$\int_0^2 \frac{1}{x^3 - 2x - 5} dx$$

- a função `fun` pode ser especificada diretamente no comando `integral`

```
I = integral(@(x)1./(x.^3-2.*x-5),0,2)
```

ou

- a função `fun` pode ser especificada através de uma função

```
I = integral(@myfun,0,2)
function y = myfun(x)
     y = 1./(x.^3-2.*x-5);
end
```

Note-se que `x` é um vetor, logo a função deve fazer operações elemento a elemento.

11.2.2 Exercícios resolvidos

1. Calcular

$$\int_0^1 \left(x^2 + \frac{1}{x+1} \right) dx$$

```
Q=integral(@(x)x.^2+1./(x+1),0,1)
```

Solução: `Q = 1.0265`

2. Calcular

$$\int_{1.5}^5 \left(x^3 - \frac{4\cos(x-1)}{e^{(x+5)}} \right) dx$$

considerando $RelTol = 10^{-3}$.

```
Q=integral(@(x)x.^3-(4.*cos(x-1))./exp(x+5),1.5,5,'RelTol',1e-3)
```

Solução: `Q = 154.9832`

3. Calcular

$$\int_{1.5}^5 \left(x^3 - \frac{4\cos(x-1)}{e^{(x+5)}} \right) dx$$

considerando $AbsTol = 10^{-2}$.

```
Q=integral(@(x)x.^3-(4.*cos(x-1))./exp(x+5),1.5,5,'AbsTol',1e-2)
```

Solução: `Q = 154.9832`