2. Aplicação do Matlab à Resolução de Problemas

Neste capítulo mostram-se as potencialidades do Matlab para resolver alguns problemas concretos.

Destacam-se sobretudo as suas capacidades de cálculo numérico e gráficos.

2.1. Estudo de Polinómios

Nesta parte aborda-se o uso do Matlab ao estudo de polinómios. A saber referem-se as seguinte funções do Matlab:

• *conv*	Produto de dois polinómios
• *deconv*	Divisão de dois polinómios
• *polyval*	Calcula o valor de um polinómio y=f(x) dado o valor de x
• *roots*	Raízes de um polinómio
• *poly*	Calcula um polinómio dadas as suas raízes
• *polyder*	Calcula a derivada de um polinómio

Na introdução teórica referiram-se como aspectos interessantes sobre este assunto algumas operações polinomiais, determinação de raízes e cálculo de polinómios conhecidas as suas raízes. Aborda-se agora como o uso do Matlab pode ser útil nestes casos.

2.1.1 Definição de um polinómio

Um polinómio é definido em Matlab à custa de um vector, cujos valores são os coeficientes do polinómio ordenado por ordem decrescente das suas potências. Por exemplo o polinómio

$$f(x) = x^2 + x - 6$$

é definido como,

```
>> polinomio = [ 1 1 -6]
```

Mais exemplos

$$f(x) = x^2 – 6$$

```
>> polinomio = [ 1 0 -6]
```

$$f(x) = x^3 - x$$

```
>> polinomio = [ 1 0 -1 0 ]
```

$$f(x) = x^4 - x^3 - 2\,x^2 + 3$$

```
>> polinomio = [ 1 -1 -2 0 3 ]
```

2.1.2 Cálculo de valores de polinómios

Existem duas formas de calcular o valor de um polinómio: ‘escrevendo’ directamente o polinómio ou usando a função ***polyval***. Da primeira forma procede-se como se indica. Seja o polinómio

$$f(x) = x^4 - 3x^3 - 2\,x^2 + 3$$

O valor de f(x=4) pode-se calcular como

```
>> x = 4;
>> f4 = x^4 - 3*x^3 - 2*x^2 + 3
>> f4 = 35.0
```

Pode-se calcular de uma só vez o valor para um conjunto de valores (vector neste caso).

```
>> w = 0:1:10;
>> f = w.^4 - 3*w^3 - 2*w.^2 + 3
```

Usando a função ***polyval***

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

é possível calcular o valor de polinómio (pol) no ponto x. Para o exemplo em questão

```
>> x = 4
>> pol = [ 1 -3 -2 0 3 ]
>> f4 = polyval( pol, x)
>> f4 = 35.0
```

```
>> w = 0:1:10;
>> f = polyval( pol, w)
```

Portanto quando x é um escalar ***polyval*** devolve um escalar, quando w é um vector devolve um vector.

1.1.3 - Operações aritméticas

A adição consiste em somar simplesmente dois vectores. Por exemplo pretende-se somar os dois seguintes vectores

$$f(x) = x^4 - 3x^2 - x + 2 \qquad g(x) = 4x^3 - 2x^2 + 5x - 16$$

A sua soma s(x) = f(x) + g(x) é calculada como se indica,

```
>> f = [ 1  0 -3 -1   2 ]
>> g = [ 0  4 -2  5 -16 ]
>> s = f + g
>> s = [ 1  4 -5  4 -14 ]
```

O polinómio calculado é $s(x) = x^4 + 4x^4 - 5x^2 + 4x - 14$

Multiplicação por um escalar

A multiplicação de um polinómio por um escalar obtém-se simplesmente pela multiplicação de um vector (que define o polinómio) por um escalar. Seja o polinómio s(x) = 3*f(x), em que f(x) é o polinómio definido acima então,

```
>> f = [ 1  0 -3  -1  2 ]
>> s = 3*f
>> s = [ 3  0 -9  -3  6 ]
```

Portanto s(x) = 3 x^4 - 9 x^2 - 3 x + 6

Multiplicação de dois polinómios

A multiplicação de dois polinómios é conseguida usando a função ***conv***. Por exemplo pretende-se multiplicar os polinómios f(x) e g(x)

$$f(x) = x^2 - x + 2 \quad g(x) = 5x - 16$$

```
>> f = [ 1 -1   2 ]
>> g = [ 5  -16 ]
>> s = conv( f, g)
>> s = [ 5  -17 -6 -32  ]
```

Seria o mesmo que s(x) = 5 x^3 - 17 x^2 - 6 x – 32

Divisão de dois Polinómios

A divisão de dois polinómios é conseguida em Matlab usando a função ***deconv***. Para dividir os polinómios

$$f(x) = x^2 - x + 2 \quad g(x) = 5x - 16$$

faz-se

```
>> [q,r] = deconv(f,g)
>> q = [ 0.20 0.44]
>> r = [ 0.0  0.0  9.04]
```

A função ***deconv*** calcula o quociente (q) e o resto (r) da divisão dos polinómios. Neste caso,

$$q(x) = 0.2\,x + 0.44 \qquad r = 9.04$$

Note-se que é válida a relação

$$f(x) = q(x) * g(x) + r$$
$$x^2 - x + 2 = (0.2\,x + 0.44)(5\,x - 16) + 9.04$$

1.1.4 - Raízes de um Polinómio

Existe frequentemente a necessidade de calcular as raízes de uma polinómio. Recorrendo ao Matlab é fácil realizar esta tarefa, mesmo que o polinómio seja de ordem elevada. A função ***roots*** é para esse fim utilizada.
Considere-se, por exemplo, o seguinte polinómio de segunda ordem

$$f(x) = x^2 + x - 6$$

```
>> f = [ 1  1 -6 ]
>> r = roots(f)
>> r= -3
       2
```

Existem portanto duas raízes, em x=2 e x=-3. Outro exemplo, agora com raízes complexas,

$$f(x) = x^2 - 2x + 5$$

```
>> f = [ 1  -2 5 ]
>> r = roots(f)
>> 1.0000 + 2.0000 j
   1.0000 - 2.0000 j
```

Existem portanto duas raízes complexas, x=1.0 + 2.0 i e x= 1.0 - 2.0 i.

Especificação de um Polinómio conhecendo os suas raízes

É possível em Matlab determinar automaticamente um polinómio que tem determinadas raízes conhecidas à priori. A função que o permite é a ***poly***.
Por exemplo sabe-se que as raízes de um polinómio são x=-3 e x=2. É possível determinar um polinómio que tem essas raízes fazendo.

```
>> f = poly ([ -3   2 ])
```

```
>> f = [ 1  1  -6 ]
```

Por outras palavras o polinómio resultando é

$$f(x) = x^2 + x - 6$$

Mais um exemplo. As raízes de um polinómio são -2, 0, 3 e -3±2 i, portanto três raízes reais e duas complexas. Eis como se pode calcular um polinómio que tenha tais raízes.

```
>> raizes = [ -2  0  3  -3+2*i  -3-2*i ]
```

```
>> f = poly ([ raizes ])
```

```
>> f = poly [ 1   5   1  -49  -78   0 ]
```

Será então o polinómio $f(x) = x^5 + 5x^4 + x^3 - 49x^2 - 78x$

2.2 - Estudo de Funções

Na secção anterior estudou-se um caso particular de funções, os polinómios. Nesta abordam-se os problemas de cálculo de mínimos e raízes de funções genéricas.
O Matlab distingue dois casos: funções de uma variável e funções de várias variáveis. Estudam-se as seguintes as funções:

• fmin	Mínimo de uma função de uma variável
• fzero	Zeros de uma função de uma variável
• fmins	Mínimo de uma função de várias variáveis

Considere-se a função

$$y = f(x) = \frac{1}{(x-1)^2 + 0.1} + \frac{1}{(x-2)^2 + 0.05} - 1$$

Pretende-se determinar o seu mínimo e determinar a seus zeros, isto é, os pontos x* tais que y(x*)=0.

Usando um ficheiro ASCII, começa por se definir a função em questão, por exemplo com o nome de **funcao1.m**

```
function y = funcao1(x)

   y= 1./( (x-1).^2 + 0.1) + 1./( (x-2).^2 + 0.05) - 1;

end
```

Pode-se depois visualizá-la na gama x∈[-2..5]. Para isso faz-se

```
>> x=-2:0.1:5;
>> y=funcao1(x);
>> plot(x,y);
>> grid
```

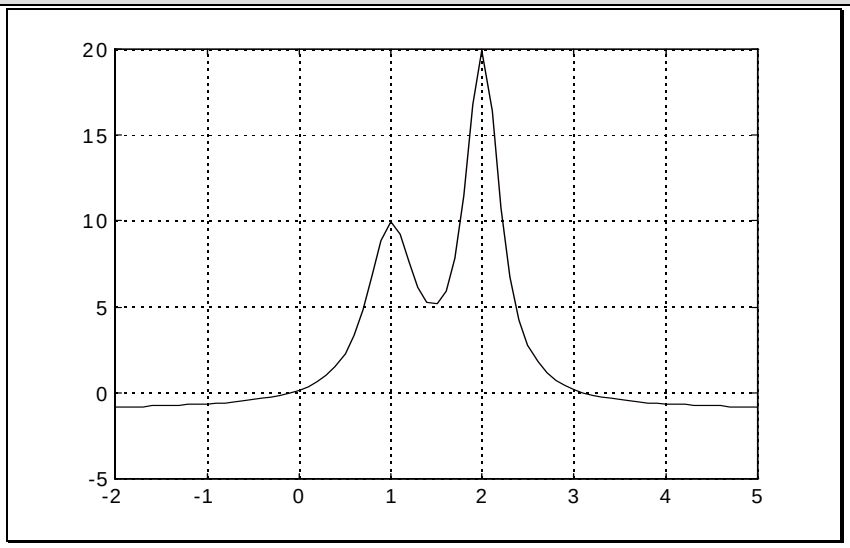

Figura 2.1 - Função de uma variável

Mínimos

Para determinar de uma função definida como acima se explicou, existe a função ***fmin***

```
>> min=fmin('funcao',xini,xfin)
```

É assim possível determinar o mínimo da função 'funcao.m' no intervalo compreendido entre x∈[xini..xfin].

Considerando os dois seguintes intervalos: [-2, 5] e [1, 2] faz-se

```
>> min1=fmin('funcao1',-2,5)
>> min2=fmin('funcao1', 1,2)
```

```
>> ymin1=funcao1(min1)
>> ymin2=funcao1(min2)
```

As duas primeiras linhas permitem determinar os x mínimos em questão e as duas últimas o valores de y correspondentes. Os valores obtidos foram

```
xmin1 = -1.9999
ymin1= -0.8278
xmin2 = 1.4629
ymin2 = 5.1363
```

Zeros

Para determinar os zeros de uma função existe a função fzero

```
>> zer=fzero('funcao',xini)
```

O valor de xini permite especificar uma solução inicial para o zero da função. Recorrendo ao Matlab calcula-se a solução nas proximidades de x=0.5 e x=2.5

```
>> zer1=fzero('funcao1',0.5)
>> zer2=fzero('funcao1',2.5)
```

Os valores obtidos foram

```
zer1 = -0.0918
zer2 = 3.1098
```

Na figura que se segue mostra-se os resultados obtidos nos dois exemplos anteriores: mínimos e zeros.

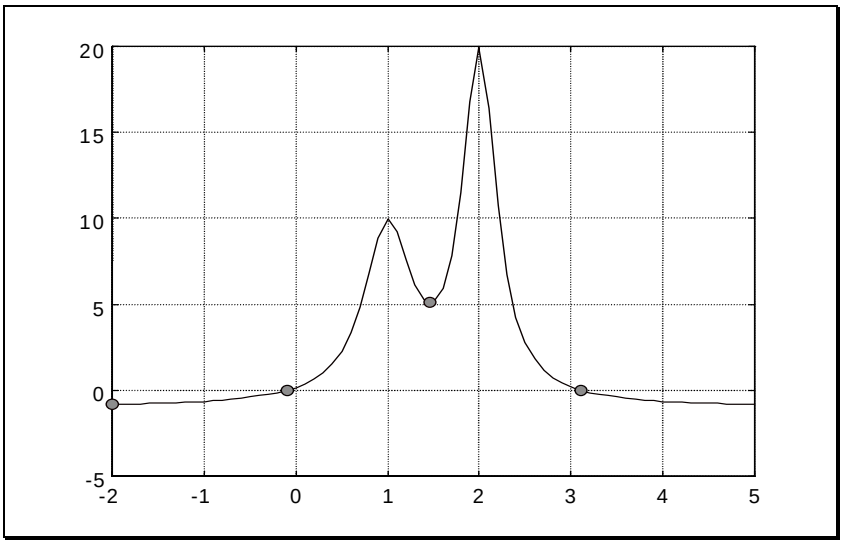

Figura 2.2 - Mínimos e zeros de uma função

2.3 - Sistemas de Equações Lineares

Nesta parte estuda-se a aplicação do Matlab à resolução de sistemas de equações lineares. A saber referem-se as seguinte funções/caracteres especiais no Matlab

• \	Divisão à esquerda
• |	Divisão à direita
• inv(A)	Inversa da matriz A
• rank(A)	Característica de uma matriz

2.3.1 - Divisão à Esquerda / Inversa

Considere-se o seguinte sistema de equações lineares

$$\begin{aligned} 3x &+ 2y - z = 10 \\ -x &+ 3y + 2z = 5 \\ x &- y - z = -1 \end{aligned}$$

Usando a notação matricial:

$$\begin{bmatrix} 3 & 2 & -1 \\ -1 & 3 & 2 \\ 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}$$

$$A\,X = B$$

Existem duas formas de solucionar o problema

- Divisão à esquerda X = A\B
- Inversa $X = A^{-1} B$

Divisão à esquerda

```
>> A = [ 3, 2, -1; -1, 3, 2; 1, -1, -1];
>> B = [ 10, 5, -1]';
>> X = A\B
```

Inversa

```
>> A = [ 3, 2, -1; -1, 3, 2; 1, -1, -1];
>> B = [ 10, 5, -1]';
>> X = inv(A)*B
```

Nos dois exemplos anteriores a solução será

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -2 \\ 5 \\ -1 \end{bmatrix}$$

2.3.2 - Divisão à Direita

Se o sistema for definido como

$$X A = B$$

então a sua solução pode ser dada por uma divisão à direita

$$X = B/A$$

Seja o seguinte exemplo

$$\begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} 3 & 2 & -1 \\ -1 & 3 & 2 \\ 1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 10 & 5 & -1 \end{bmatrix}$$

Divisão à direita

```
>> A = [ 3, 2, -1; -1, 3, 2; 1, -1, -1];
>> B = [ 10, 5, -1];
>> X = B/A
```

X = [x y z] = [-3.0 15.0 34.0]

2.3.3 - Exemplos

Eis alguns exemplos, e respectiva solução, de sistemas de equações lineares:

$$\begin{aligned} x &+ 2y - z = 1 \\ x &- 2y + z = 2 \\ x &+ y - z = -1 \end{aligned}$$

Solução: (x,y,z) = (1.5, 2, 4.5)

$$\begin{aligned} x &+ 2y + z = 1 \\ 2x &- y - z = 5 \end{aligned}$$

Solução: (x,y,z) = ..

Não é possível determinar a solução uma vez que existe um número de incógnitas (3) superior ao de equações (2). Existe uma infinidade de soluções …

$$\begin{aligned} x &+ y + z = 10 \\ -x &- y - z = 5 \\ x &+ 2y - z = -1 \end{aligned}$$

Solução: (x,y,z) = (Nan, Nan, Nan)

Note-se que rank(A)=2, ou seja, não existe um número de equações linearmente independentes igual ao número de incógnitas e portanto o sistema não tem solução (a primeira linha é o simétrico da segunda).

$$\begin{aligned} x &+ 2y + z = 1 \\ 2x &-+3y - z = 2 \\ 2x &- 2y - z = 5 \\ 2x &- y + z = 1 \end{aligned}$$

Solução: (x,y,z) = ..

Não é possível determinar a solução uma vez que existe um número de incógnitas (3) inferior ao de número de equações (4). Não existe portanto solução.

2.4 - Interpolação a Ajuste de Curvas

Nesta parte estuda-se a aplicação do Matlab à interpolação e ajuste de curvas. A saber referem-se as seguinte funções no Matlab

• polyfit	Calcula um polinómio de ordem n usando a técnica dos mínimos quadrados
• polyval	Calcula o valor de um polinómio y=f(x) dado o valor de x
• spline	Determine uma interpolação cúbica

• tabl1	Determina uma interpolação linear a uma dimensão
• table2	Determina uma interpolação linear a duas dimensão

2.4.1 - Interpolação

Basicamente a interpolação tem por objectivo estimar o valor entre dois pontos para os quais se conhecem os valores. Estudam-se duas formas de aproximação: a linear e a cúbica

Exemplo

Considere o tanque de água que se mostra na figura seguinte.

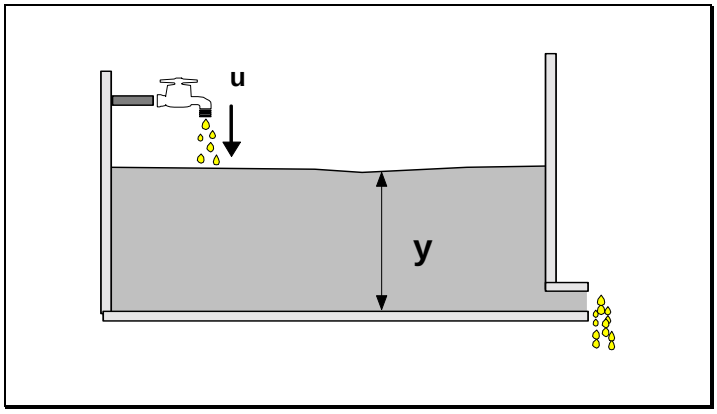

Figura 2.3 - Tanque de água

Para estudar o sistema abriu-se a torneira e recolheram-se valores da áltura da água (y) durante os primeiros dez minutos, que se mostram na tabela seguinte

Tempo (s)	Áltura (y)
0	0
1	0.7
2	2.4
3	3.1
4	4.2
5	4.8
6	5.7
7	5.9
8	6.2
9	6.4
10	6.4

Tabela 2.1 - Valores da evolução da altura da água

Graficamente pode-se ver a variação da áltura da água

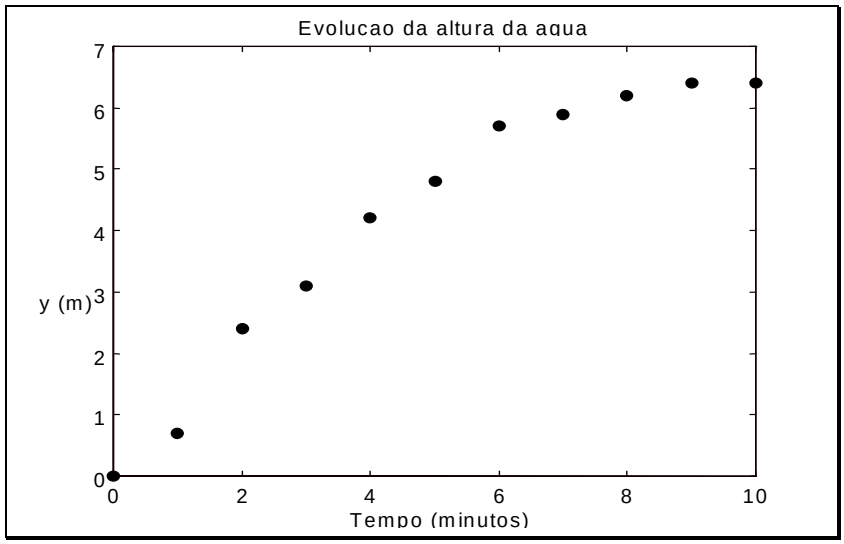

Figura 2.4 - Evolução da altura água

O objectivo da interpolação é estimar o valor da áltura da água em pontos intermédios daqueles que se conhece o valor correcto da áltura da água. Pretende-se, por exemplo, estimar o valor da áltura da água no instante 3.5 minutos.

Na análise teórica do problema, estudada no capítulo 2, foi explicado como a interpolação linear poderia ser usada para resolver o problema. Porque a interpolação é linear aproxima-se esse valor por uma função linear, ou seja, uma recta.

Assim é fácil de entender graficamente o resultado da interpolação de todos os pontos: rectas que unem cada um dos pontos adjacentes, como se mostra na figura seguinte:

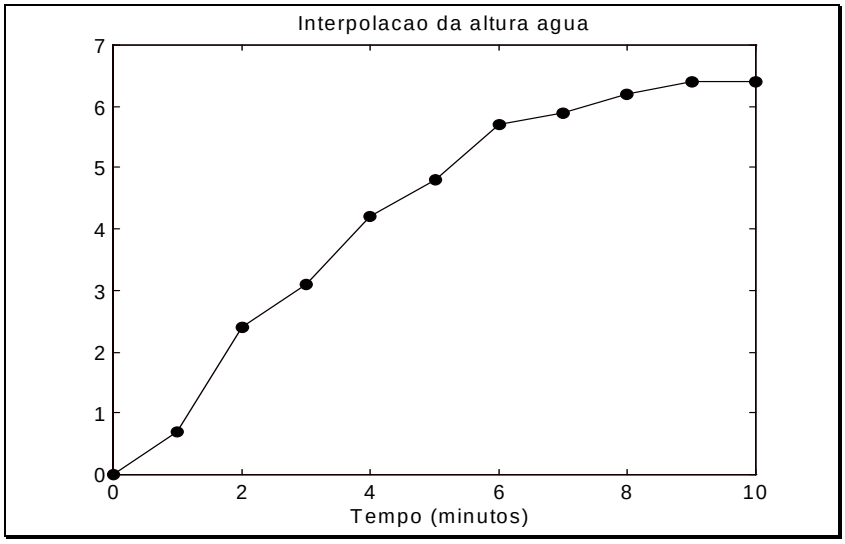

Figura 2.5 - Interpolação linear da altura da água

O Matlab, além da interpolação linear, permite solucionar tal problema recorrendo a um método mais preciso, a cúbica. Mais tarde aborda-se novamente este último método.

Interpolação Linear

Começa-se por representar os dados segundo uma matriz em que a primeira linha é constituída pelos minutos (t) e a segunda pelos valores da altura da água (y) de vectores:

```
t = 0:1:10;

y = [0  0.7  2.4  3.1  4.2  4.8  5.7  5.9  6.2   6.4  6.4   ];

dados(:,1)= t';

dados(:,2)= y';
```

A função ***table1*** permite interpolar a altura da água num valor desejado.

```
>> y=table1( dados, x )
```

Como argumentos de entrada a tabela de dados e o valor de x a a interpolar, como variável de saída y=f(x). Neste caso particular pretende-se o valor no instante 3.5, então

```
>> y1=table1( dados, 3.5 )
>> y1 =3.65
```

Para obter a interpolação não para um ponto (escalar) mas para um conjunto de pontos (vector) procede-se da mesma forma:

```
>> x= 0:0.1:10;
>> y=table1( dados, x )
```

Exemplo 2

No exemplo anterior os dados são constituídos por duas colunas: o tempo e a altura da água. Suponha-se agora que é adicionalmente são recolhidos mais dois dados relativos ao tanque de água, a temperatura da água na base e nível superior do tanque. A tabela seguinte contém esses valores:

Tempo (s)	Altura (y)	Temp. base	Temp. Sup.
0	0	10	10
1	0.7	10.6	10.7
2	2.4	10.9	10.4
3	3.1	11.1	11.1
4	4.2	11.7	12.2

5	4.8	11.6	12.8
6	5.7	11.5	12.7
7	5.9	11.2	12.9
8	6.2	11.6	12.6
9	6.4	11.4	12.7
10	6.4	11.4	12.7

Tabela 2.2 - Valores da altura e temperatura da água

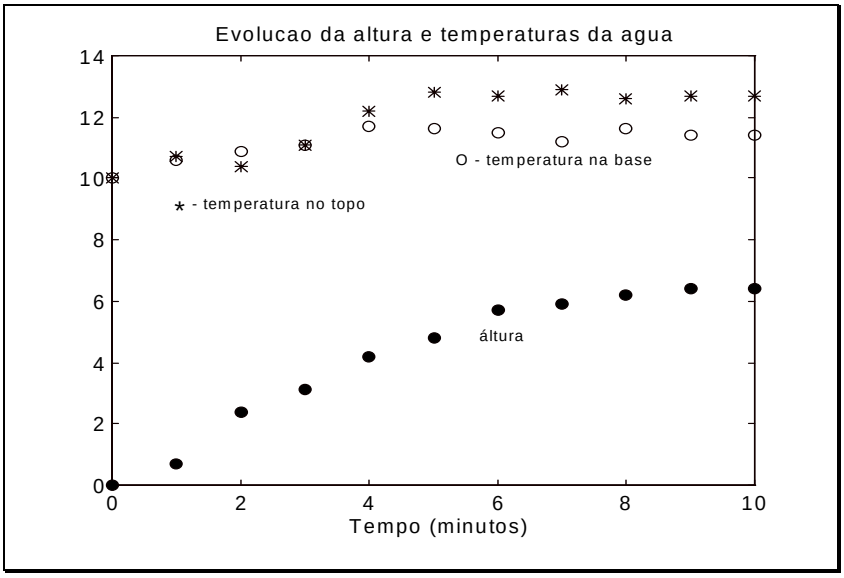

Figura 2.6 - Evolução da altura e temperaturas da água

Começa por se definir o conjunto de dados, notando que existem agora quatro colunas

```
t = 0:1:10;

y = [0  0.7  2.4  3.1  4.2  4.8  5.7  5.9  6.2   6.4  6.4   ];

temp1=[10 10.6 10.9 11.1 11.7 11.6 11.5 11.2 11.6 11.4 11.4 ];

temp2=[10 10.7 10.4 11.1 12.2 12.8 12.7 12.9 12.6 12.7 12.7 ];

dados(:,1)=t';

dados(:,2)=y';

dados(:,3)=temp1';

dados(:,4)=temp2';
```

Para interpolar valores num dado instante usa-se novamente a função ***table1***

```
>> y1 = table1(dados,3.5);

>> y1 = [ 3.65  11.40   11.65 ]
```

ou seja são interpolados os valores para a áltura da água e temperaturas na base e topo.

Interpolação a duas dimensões

Novamente para o exemplo do tanque de níveis, foram recolhidos valores para a áltura do nível da água considerando variável o diâmetro da torneira (de 1 a 1.4 centímetros). Esses resultados mostram-se na tabela e figura seguintes

Tempo (s)	1.0	1.1	1.2	1.3	1.4
0	0	0	0	0	0
1	0.7	4.0	3.3	3.7	3.8
2	2.4	4.2	2.4	5.2	5.0
3	3.1	4.1	3.5	6.1	5.8
4	4.2	4.8	4.5	6.2	6.3
5	4.8	5.2	5.2	7.2	7.4
6	5.7	5.4	5.5	7.7	7.7
7	5.9	6.1	6.5	7.9	8.0
8	6.2	6.5	6.9	8.2	8.3
9	6.4	6.7	7.4	8.2	8.4
10	6.4	6.8	7.5	8.1	8.3

Tabela 2.3 - Valores da altura em função do diâmetro da torneira

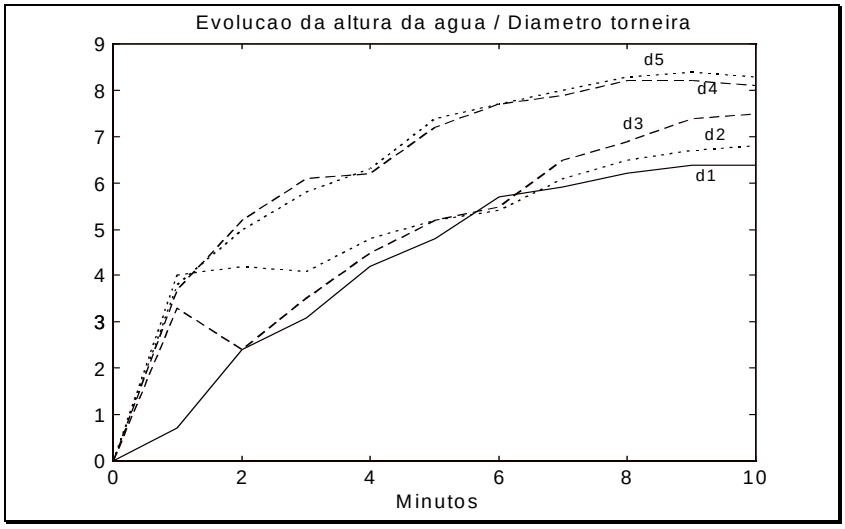

Figura 2.7 - Evolução da altura em função do diâmetro da torneira

Com este conjunto de dados pretende-se agora conhecer o valor da áltura da água num instante qualquer, tendo em atenção também o diâmetro da torneira. Para esse efeito existe a função ***table2***. Para começar constrói-se uma matriz de dados em que cada linha contém os dados relativos a cada instante:

```
dados( 1,:)  = [ 0 1.0 1.1 1.2 1.3 1.4 ] ;

dados( 2,:)  = [ 0 0   0   0   0   0   ] ;

dados( 3,:)  = [ 1 0.7 4.0 3.3 3.7 3.8 ] ;
```

```
dados( 4,:)  = [ 2 2.4 4.2 2.4 5.2 5.0 ] ;
dados( 5,:)  = [ 3 3.1 4.1 3.5 6.1 5.8 ] ;
dados( 6,:)  = [ 4 4.2 4.8 4.5 6.2 6.3 ] ;
dados( 7,:)  = [ 5 4.8 5.2 5.2 7.2 7.4 ] ;
dados( 8,:)  = [ 6 5.7 5.4 5.5 7.7 7.7 ] ;
dados( 9,:)  = [ 7 5.9 6.1 6.5 7.9 8.0 ] ;
dados(10,:)  = [ 8 6.2 6.5 6.9 8.2 8.3 ] ;
dados(11,:)  = [ 9 6.4 6.7 7.4 8.2 8.4 ] ;
dados(12,:)  = [10 6.4 6.8 7.5 8.1 8.3 ] ;
```

Note-se que os dados são organizados por linhas e não por colunas, como no exemplo anterior. Para interpolar o valor da água no instante 3.5 e para um diâmetro 1.25 faz-se

```
>> y = table2( dados, 3.5, 1.25);
>> y = 5.0750
```

Pode-se entender este processo de interpolação como uma interpolação a duas dimensões, isto é, o valor da variável a determinar depende de duas variáveis, neste caso o tempo e a capacidade da torneira (diâmetro).

Interpolação Cúbica

Por vezes a aproximação linear pode não constituir uma solução satisfatória. O Matlab implementa um outro tipo de interpolação, em que cada curva entre dois pontos adjacentes consiste num polinómio de terceira ordem (dai a designação de interpolação cúbica). A função que permite tal é a ***spline***.

Retomando o exemplo da áltura da água dado no início o aspecto geral deste último tipo de aproximação, comparada com a linear seria:

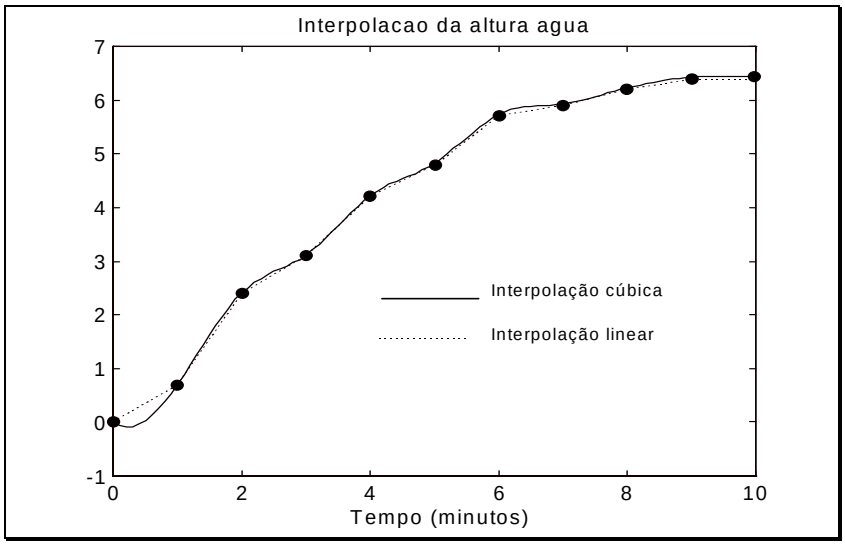

Figura 2.8 - Interpolação linear / cúbica

Admitindo que se quer, tal como anteriormente, estimar o valor da áltura no instante 3.5. Usando a função ***spline*** tem-se

```
>> t = 0:1:10;
>> y = [0 0.7 2.4 3.1 4.2 4.8 5.7 5.9 6.2  6.4 6.4   ];
>> y1 = spline( t, y, 3.5 )
>> y1 =3.6457
```

Note-se que não é necessário definir uma matriz de dados como o era no uso da table1. São usados directamente os valores de t e y.

2.4.2 - Ajuste de Curvas

Assume-se agora que se dispõe de um conjunto de dados, por exemplo recolhidos do processo dos tanque de água, e que se pretende determinar uma função que aproxime os dados reais existentes. Na análise teórica do problema foram apresentados métodos que utilizam regressões lineares para essa aproximação, o mesmo é dizer, usam rectas. Foi introduzido o método dos mínimos quadráticos como aquele que de entre todas as rectas possíveis determina a que garante o menor desvio do quadrado das distâncias.

Foram depois definidos métodos de regressão polinomial, isto é, a aproximação é feita não por um polinómio de primeira ordem (recta), mas por um polinómio de qualquer ordem.

Aproximação linear: Método dos Mínimos Quadráticos

O Matlab dispõe da função ***polyfit*** que permite aproximar um dado conjunto de pontos por uma recta utilizando o método dos mínimos quadráticos. Considerando novamente o problema do tanque de água, conhece-se o seguinte conjunto de dados:

```
>> t = 0:1:10;
```

```
>> y = [0 0.7  2.4  3.1  4.2  4.8  5.7  5.9  6.2   6.4
6.4 ];
```

A função ***polyfit*** permite obter os coeficiente do polinómio (dois, os coeficientes da recta):

```
>> coef = polyfit(t,y,1);
```

```
>> coef = [ 0.6664  0.8318  ]
```

Ou seja a recta resultante é

$$y = 0.6664\,x + 0.8318$$

Graficamente o resultado pode ser obtido como se segue

```
>> yap = coef(1).*t+coef(2);
```

```
>> plot(t,yap,t,y,'o')
```

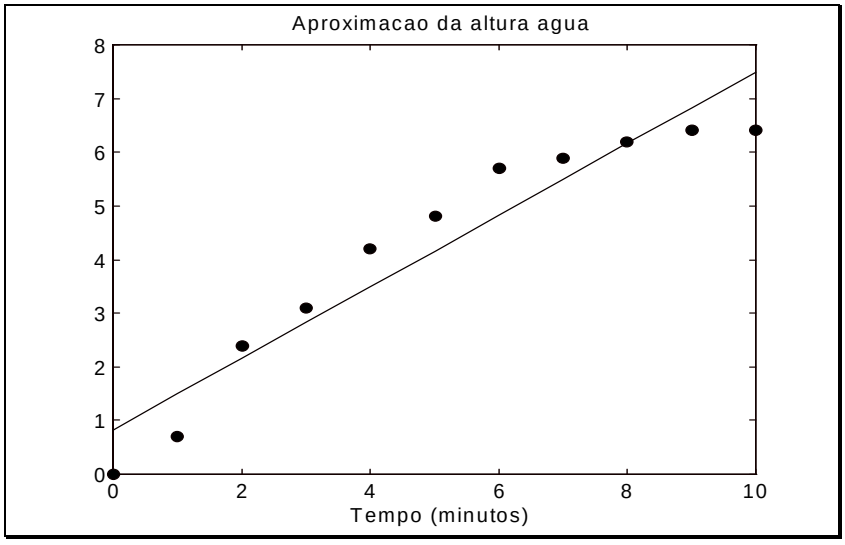

Figura 2.9 - Recta calculada pelos mínimos quadráticos

Aproximação Polinomial

Usando a função ***polyfit*** é possível aproximar o conjunto de pontos por um polinómio de qualquer ordem. A sintaxe é

```
>> coef = polyfit(t,y,n);
```

Em que n é a ordem do polinómio. Note-se que a função é válida para aproximações lineares, sendo neste caso n=1.

Para o mesmo exemplo apresenta-se a solução para polinómios de segunda, terceira, quarta e quinta ordem.

Segunda ordem: **n=2**

```
>> coef = polyfit(t,y,2);
```

```
>> coef = [ -0.0712  1.3785 -0.2364 ]
```

$$f(x) = -0.0712\ x^2 + 1.3785\ x - 0.2364$$

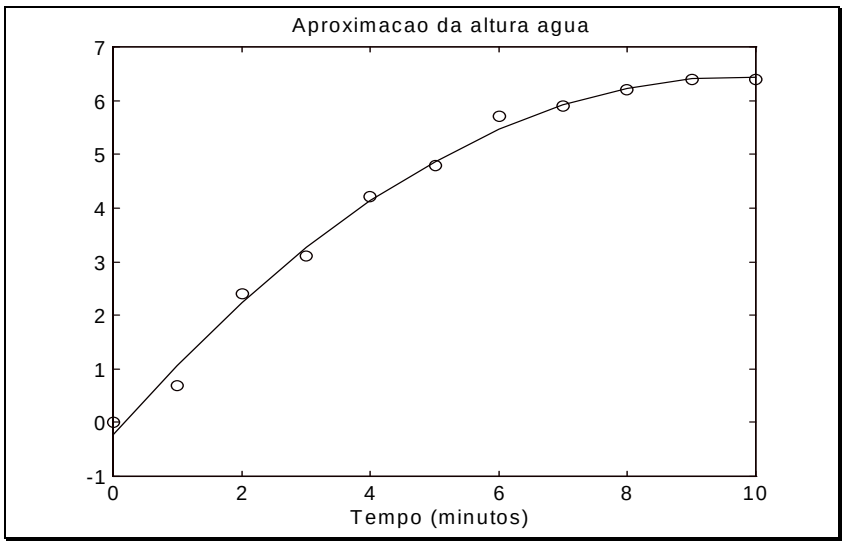

Figura 2.10 - Aproximação de segunda ordem

Terceira ordem: **n=3**

```
>> coef = polyfit(t,y,3);
```

```
>> coef = [ -0.0022 -0.0386  1.2540 -0.1580 ]
```

$$f(x) = -0.0022\ x^3 - 0.0386\ x^2 + 1.2540\ x - 0.1580$$

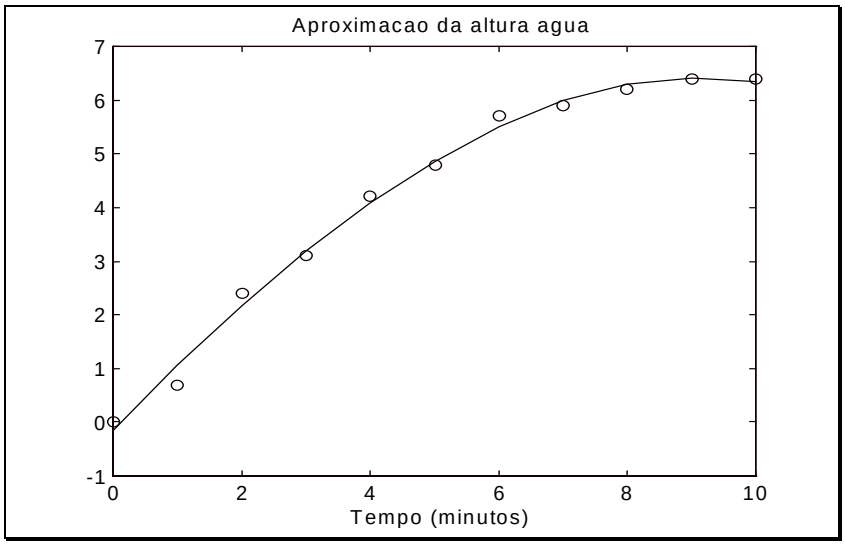

Figura 2.11 - Aproximação de terceira ordem

Quarta ordem: **n=4**

```
>> coef = polyfit(t,y,4);
```

```
>> coef = [ 0.0010 -0.0232  0.0925  0.9918 -0.0825 ]
```

$$f(x) = 0.0010\,x^4 - 0.0232\,x^3 + 0.0925\,x^2 + 0.9918\,x - 0.0825$$

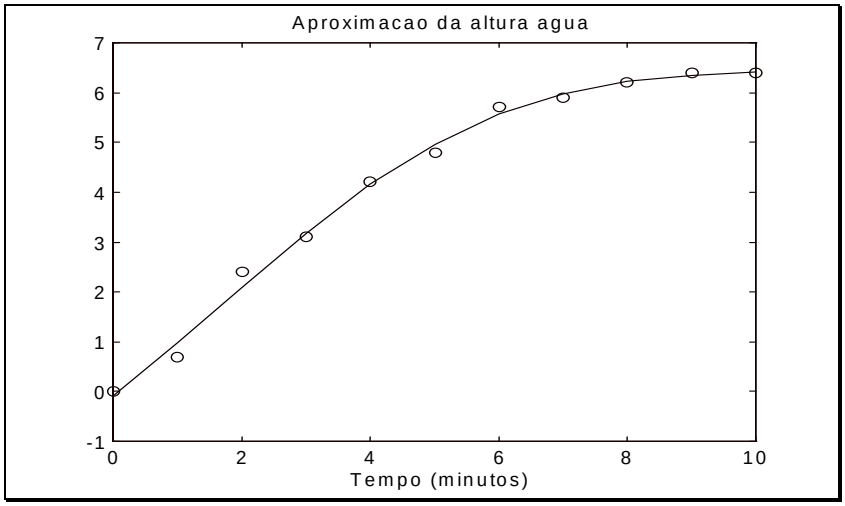

Figura 2.12 - Aproximação de quarta ordem

Quinta ordem: n=5

```
>> coef= polyfit(t,y,5);
```

```
>> coef= [-0.0003 0.0075 -0.0791 0.2913 0.7506 -0.0517 ]
```

$$f(x) = -0.0003\,x^5 + 0.0075x^4 - 0.0791\,x^3 + 0.2913\,x^2 + 0.7506\,x - 0.0517$$

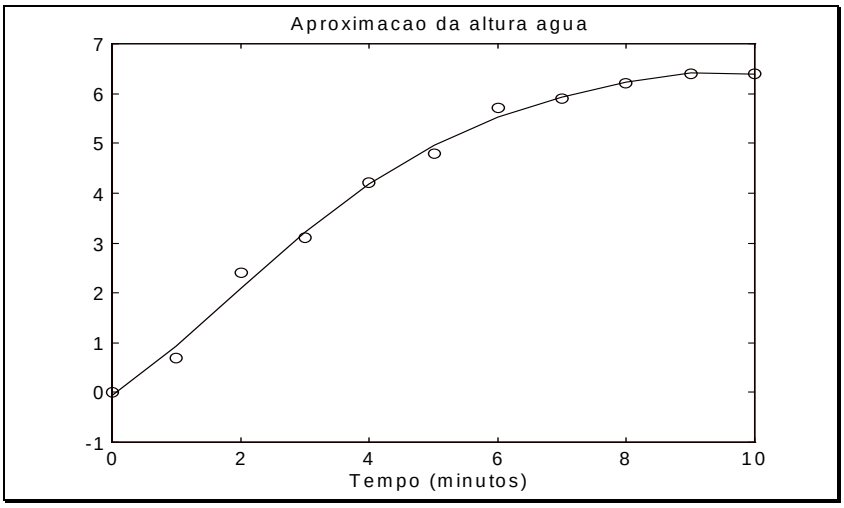

Figura 2.13- Aproximação de quinta ordem

Interpolação Usando a Função Calculada na Aproximação

A determinação de uma função que aproxime um conjunto de pontos pode ser usada para o cálculo de valores intermédios, tal como a interpolação. Ora, uma vez calculado o polinómio que aproxima o conjunto de pontos inicial, é depois possível interpolar o valor para um outro ponto qualquer recorrendo à função determinada.

Por exemplo, pretende-se interpolar um valor para a altura da água no instante 3.5. Uma possível solução é usar esta último aproximação (quinta ordem) e calcular o valor da função em 3.5, fazendo

```
>> coef =[ -0.0003   0.0075 -0.0791   0.2913   0.7506 -
0.0517 ]
```

```
>> y1 = polyval(coef, 3.5)
```

```
>> y1  = 3.7349
```

2.5 - Integração Numérica e Diferenciação

Nesta parte estuda-se a aplicação do Matlab à integração numérica e diferenciação de funções. A saber referem-se as seguinte funções no Matlab

• diff	Cálculo das diferenças entre dois valores adjacentes
• quad	Calcula o integral sobre uma curva (Método de Simpson)
• find	Índice dos valores não nulos de um vector

2.5.1 - Integração numérica

Referiu-se no estudo teórico do problema duas técnicas de calcular numericamente o integral de uma função: a regra trapezoidal e a de Simpson. Na primeira é usando uma aproximação linear para os valores da função na segunda uma aproximação quadrática. Em termos de cálculo os métodos distinguem-se pela fórmula que permite calcular o integral (a primeira diz respeito à regra trapezoidal e a segunda à de Simpson):

1. $K_{ap} = \frac{b - a}{2n} (f(x_o) + 2 f(x_1) + 2 f(x_2) + \ . \ . \ . + 2 f(x_{n-1}) + f(x_n))$

2. $K_{ap} = \frac{h}{3} (\ f(x_o) + 4 f(x_1) + 2 f(x_2) + 4 f(x_3) + . \ . \ . + 2 f(x_{n-2}) + 4 f(x_{n-1}) + f(x_n))$

em que h é o passo de integração.

O Matlab implementa a regra de Simpson disponibilizando a função ***quad*** para esse efeito.

Funções pré definidas

Considere-se por exemplo a seguinte função, já referida na parte teórica,

$$y = \sqrt{x}$$

Pretende-se calcular o integral, o mesmo é dizer a área abaixo da função, entre dois valores positivos a e b (b>a). Na figura seguinte mostra-se a função no intervalo [0, 5]

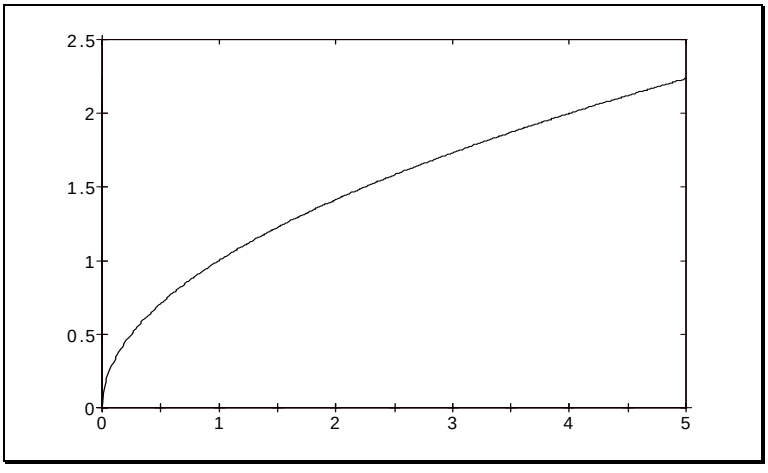

Figura 2.14 - Função raiz quadrada

Para calcular a solução (valor da área) para este problema faz-se

```
>> area =quad('sqrt', 0, 5)
```

```
>> area =  7.4535
```

Os parâmetros são:
- 'sqrt' especifica o nome da função,
- 0 - valor inicial
- 5 - valor final de integração.

Funções definidas pelo utilizador

Acontece que na prática é desejado calcular o integral de funções definidas pelo próprio utilizador (no exemplo anterior a função existe disponível no Matlab). Por considere a seguinte função

$$y = f(x) = x^2 + 5*\sin(x) - \cos(2*x) + 1$$

A seguir mostra-se o seu gráfico no intervalo [-1 .. 5]

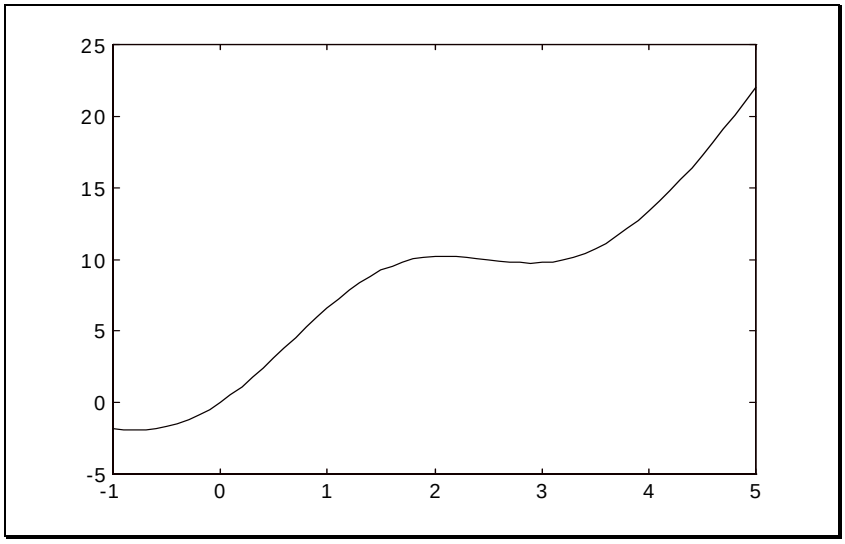

Figura 2.15 - Exemplo de uma função

Pretende-se calcular a área abaixo da figura no intervalo [1 .. 4], dada pela área da figura abaixo

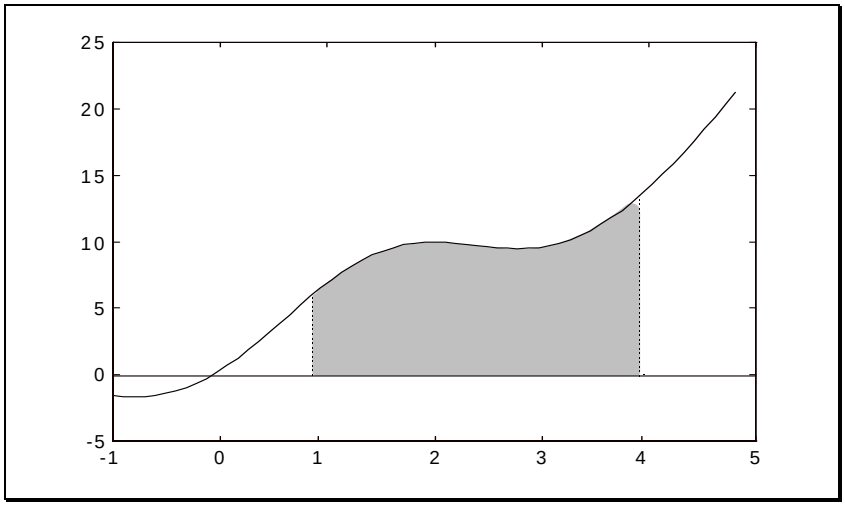

Figura 2.16 - Integral de uma função

Primeiro deve-se definir a função desejada criando uma função num ficheiro *.m (assunto abordado na parte de Matlab). O seguinte código é então criado e armazenado num ficheiro de nome funcao.m

```
function y = funcao(x)

   y= x.^2 + 5*sin(x) - cos(2*x)+ 1;

end
```

Para calcular o valor do integral basta executar o seguinte comando

```
>> area = quad('funcao',1,4)

>> area  = 29.9297
```

2.5.2 - Diferenciação numérica

Tal como a integração o cálculo da derivada de uma função pode ser facilmente implementada em Matlab. A seguir mostra-se como a função ***diff*** pode ser utilizada nessa tarefa.

Derivada

Seja a função definida pelo seguinte polinómio (rever parte teórica)

$$f(x) = x^5 - 3x^4 - 11x^3 + 27x^2 + 10x - 24$$

Graficamente

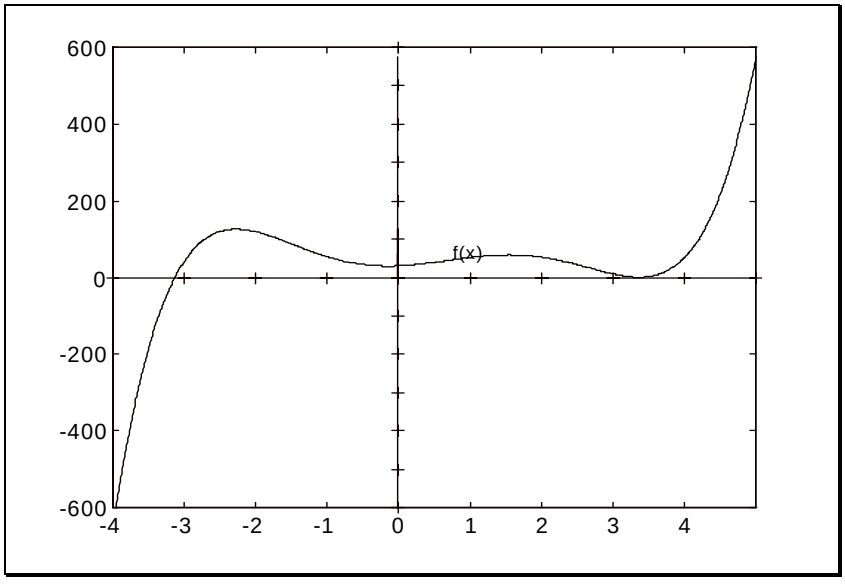

Figura 2.17 - Exemplo de uma função

Para calcular a sua derivada deve-se fazer

```
>> x = -4:0.1:5;
>> y = x.^5 - 3*x.^4 - 11*x.^3 + 27*x.^2 + 10*x -24;
>> dy = diff(y)./diff(x);
```

Refira-se que a função ***diff*** não implementa directamente o cálculo da derivada mas sim do cálculo das diferenças entre dois valores adjacentes de um vector. Daí o cálculo da derivada ser a divisão entre as diferenças de y e as diferenças de x.

$$y' \approx \frac{dy}{dx}$$

Na figura seguinte mostra-se a função (y) e respectiva derivada (dy)

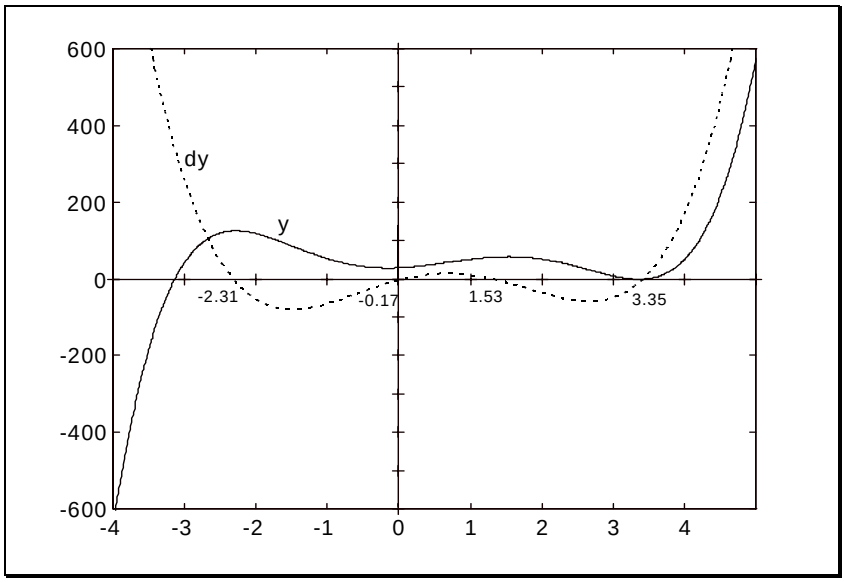

Figura 2.18 - Função e respectiva derivada

Pontos Críticos

Sabe-se que o conhecimento da derivada permite identificar os valores máximos ou mínimos de uma função. Viu-se também no capítulo 2 como a evolução da derivada permite concluir da existência de pontos críticos.

1. Calcula-se a derivada de uma função em pontos adjacentes num dado intervalo
2. Se o produto da deriva num ponto x(k) pelo valor no ponto x(k+1) for:
   - > 0 - não houve variação de sinal
   - = 0 - a derivada é nula num (ou nos dois) dos pontos
   - < 0 - houve uma variação de sinal
3. Os valores <0 permitem portanto identificar pontos críticos

Em Matlab este raciocínio é possível de implementar usando a função ***find*** . A função find permite determinar, para um determinado vector, quais os índices dos seus elementos não nulos.

Por exemplo

```
>> indices = find( X > 10);
```

devolve os índices do vector X cujos elementos são superiores a 10.

Assim os pontos críticos no exemplo em estudo são obtidos como se indica a seguir

```
>> xd = x(2:length(x));
>> produto = dy(1:length(dy)-1).*dy(2:length(dy));
>> pcriticos = xd( find (produto<0))
>> pcriticos = [ -2.31   -0.17   1.53    3.35 ]
```

Os pontos críticos obtidos podem ser confirmados pela observação da figura anterior.

2.6 - Equações diferenciais de primeira ordem

Nesta parte estuda-se a aplicação do Matlab à resolução de equações diferenciais. A saber referem-se as seguintes funções do Matlab

• ode23	Método de Runge-Kutta de segunda e terceira ordem
• ode45	Método de Runge-Kutta de terceira e quarta ordem

2.6.1 - Introdução

Em Matlab existem duas funções, ***ode23*** e ***ode45***, capazes de calcular numericamente a solução de uma equação diferencial do tipo:

Equação diferencial y'= g(x,y)

Solução y=f(x)

A primeira implementa o método de Runge-Kutta de segunda-terceira ordem e a segunda o método de Runge-Kutta de quarta-quinta ordem.

Uma vez que as duas funções possuem o mesmo número de argumentos, explica-se apenas o funcionamento de uma delas. A outra é equivalente diferindo apenas no método implementado. A sintaxe da função ***ode23*** é a seguinte

```
[ x, y ] = ode23( 'derivada', inicio, fim, y0)
```

Saídas

- x - coordenadas
- y - valores correspondente para y=f(x)

Entradas

- 'derivada' - nome do ficheiro *.m onde deve ser definida a equação diferencial
- início - ponto inicial a partir do qual se calcula a solução
- fim - ponto final para o qual se calcula a solução
- y0 - condição inicial para a y=f(xo)

2.6.2 - Exemplo 1

Considere-se a seguinte equação diferencial

$$y' = 2x + 2$$

Sabe-se que analiticamente a solução desta equação é

$$y(x) = x^2 + 2x + yo$$

sendo y0 a chamada condição inicial, isto é y(0)=f(x(0)). Em Matlab este problema pode-se resolver da seguinte forma:

1. Define-se a função derivada, construindo um ficheiro *.m, por exemplo, de nome deriva.m com o conteúdo:

```
function  dy=deriva(x,y)
```

```
dy = 2*x + 2;
```

2. Uma vez definida a condição inicial, por exemplo y0=1, e o intervalo desejado [0..2] faz-se

```
>> yo= 1;
```

```
>> [x,y]= ode23('deriva',0,2,yo);
```

Na figura que se segue mostra-se o valor calculado usando a função ***ode23*** (método de Runge-Kutta) e o valor analítico exacto (ya) calculado usando a solução

$$y(x) = x^2 + 2x + 1$$

```
>> ya = x.^2 + 2*x + 1;
```

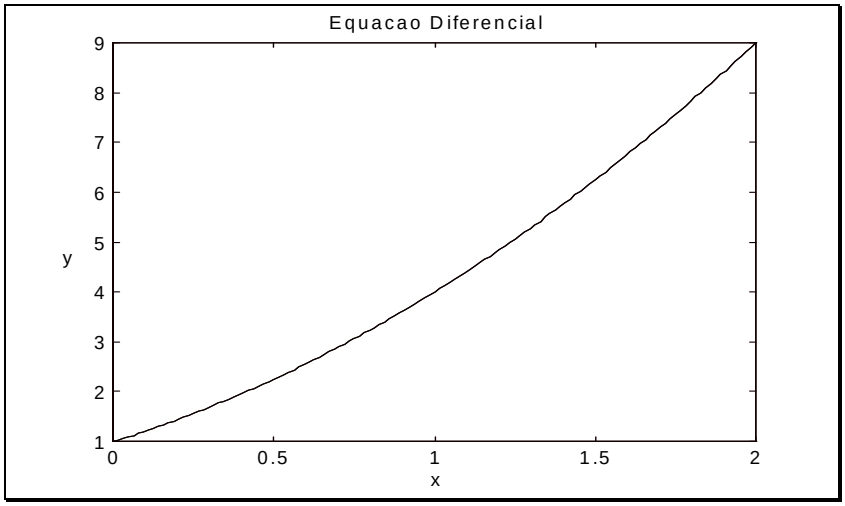

Figura 2.19 - Derivada analítica e calculada numericamente

Note-se a proximidade das duas soluções. Na verdade não é possível distingui-las gráficamente.

2.6.3 - Exemplo 2

Considere-se agora a equação diferencial

$$y' = 4y + e^{2x}$$

Usando o Matlab procede-se como se referiu no exemplo anterior. Constrói-se uma função

```
function  dy=deriva(x,y)
   dy = 3*y + exp(2*x);
```

Define-se a condição inicial e usa-se a função ode23.

```
>> xo=3;
>>[x,y]=ode23('deriva',0,1,xo);
```

Sabe-se que analiticamente a solução é dada por

$$y = f(x) = 3e^{3x} - e^{2x}$$

```
>> ya=4*exp(3*x)-exp(2*x);   %--  y(0) = 3
```

A seguir mostra-se um gráfico comparativo das duas soluções: analítica e numérica

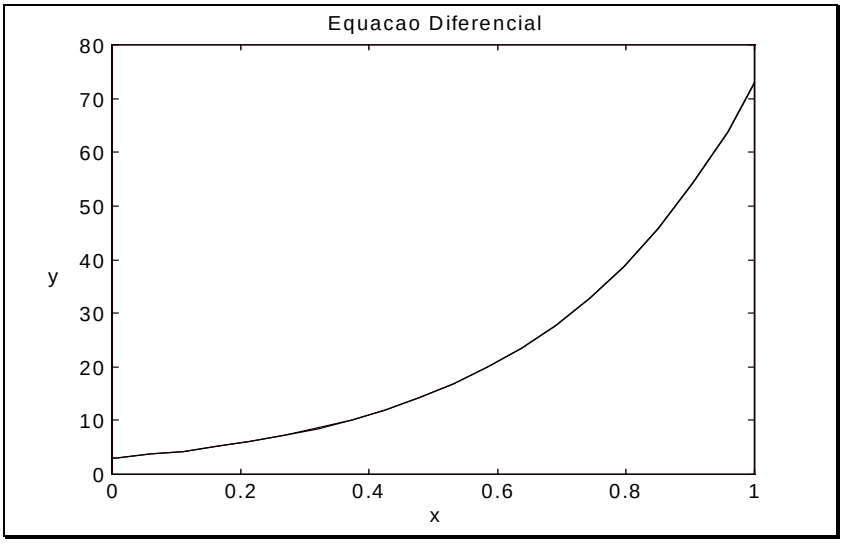

Figura 2.20 - Solução analítica / numérica de uma equação diferencial