



Introdução ao MATLAB

Marcelo A. Trindade e Rubens Sampaio

Departamento de Engenharia Mecânica, Laboratório de Dinâmica e Vibrações,
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio),
rua Marquês de São Vicente, 225, 22453-900 Rio de Janeiro, Brasil
trindade@mec.puc-rio.br e rsampaio@mec.puc-rio.br

Rio de Janeiro, 25 de Março de 2002

Conteúdo

1	Introdução	4
2	Comandos Básicos do MATLAB	6
2.1	Entrando no MATLAB	6
2.2	Primeiros passos	6
2.3	Operações básicas com matrizes	11
2.4	Gravação e recuperação de dados	19
2.5	Zeros de polinômios	20
2.6	Criação e edição de gráficos	21
2.7	Programando em MATLAB	28
2.7.1	Arquivos .m	28
2.7.2	O comando global	30
2.7.3	O comando if	31
2.7.4	O comando for	33
3	Aplicações	35
3.1	Programas interativos	35
3.2	Zeros de funções	36
3.3	Calculando Integrais	36
3.3.1	Regra trapezoidal	37
3.3.2	Regra de Simpson	39
3.4	Equações diferenciais	39
3.4.1	Sistema massa-mola-amortecedor	40
3.4.2	Pêndulo simples	42

Capítulo 1

Introdução

O MATLAB[®] pode ser usado como uma linguagem de programação ou como uma ferramenta de cálculo interativa. Em ambos os casos, o ambiente MATLAB permite realização de cálculos, visualização de resultados e desenvolvimento de algoritmos usando uma sintaxe muito próxima da notação matemática standard.

O nome MATLAB vem de MATrix LABoratory e representa bem o seu modo de funcionamento. Todas as variáveis são definidas de forma matricial, o que faz com que este ambiente seja ideal para resolver problemas de cálculo envolvendo matrizes e vetores. Ele é desenvolvido pela The MathWorks, Inc. (<http://www.mathworks.com>).

Um arquivo de programa MATLAB recebe a extensão `.m` e, por isso, também é chamado de `m-file` ou `arquivo .m`. Ele pode ser criado e/ou modificado no editor próprio do MATLAB, MATLAB Editor/Debugger, que possui características que auxiliam bastante a edição de programas como auto-colorização de palavras de acordo com a sintaxe e interação com o ambiente de cálculo do MATLAB. Uma das grandes virtudes do MATLAB é que todas suas funções são escritas em `m-files`, sendo que a maioria delas podem ser editadas/modificadas. Este fato possibilitou a criação de diversos conjuntos de funções específicas a determinadas áreas, como controle, análise de sinais, finanças, entre outras. Estes conjuntos de funções, ou de `m-files`, são chamados de `toolbox`. `Toolboxes` direcionados para diversas áreas são distribuídos pela própria MathWorks e inúmeros outros são desenvolvidos em todo o mundo, com e sem reconhecimento da MathWorks, e podem geralmente ser encontrados na internet.

Enquanto que o programa principal do MATLAB depende da plataforma (Unix, Windows, Macintosh) os arquivos `.m`, que constituem as funções e programas, são independentes da plataforma e podem portanto ser utilizados em todas elas. Esta é outra razão para o estabelecimento do MATLAB como ferramenta de cálculo matricial de predileção entre os profissionais do meio acadêmico e industrial. De fato, não é incomum encontrar um `toolbox` na internet bem adaptado ao problema específico que se quer resolver. E mesmo que ele tenha sido desenvolvido em ambiente Unix, pode ser utilizado em ambiente Windows.

A linguagem de programação do MATLAB é muito intuitiva. Por esta razão, o objetivo deste trabalho é iniciar a aprendizagem da utilização do ambiente MATLAB, dos comandos básicos e de construção de programa. A partir daí, esperamos que o usuário seja capaz de desenvolver e direcionar os conhecimentos adquiridos para sua área específica.

Marcelo A. Trindade
Rubens Sampaio

Capítulo 2

Comandos Básicos do MATLAB

2.1 Entrando no MATLAB

O MATLAB pode ser executado pressionando o ícone correspondente, criado na instalação do programa, ou através da barra de tarefas. Quando executado, a janela principal do MATLAB, chamada [MATLAB Command Window](#), é aberta. O símbolo `>>` na tela indica que O MATLAB está pronto para receber um comando. Assim, esta janela funciona como um ambiente interativo no qual comandos são fornecidos pelo usuário e executados instantaneamente pelo MATLAB. Uma linha de comandos é finalizada pressionando a tecla [Enter]. Os comandos fornecidos são memorizados e podem ser listados usando as setas para cima e para baixo do teclado.

2.2 Primeiros passos

Uma variável pode ser definida dando-se valores a ela. O tipo da variável é automaticamente definido pelo MATLAB em função do valor dado a ela. A sintaxe é a seguinte

```
>> t=1

t =

    1
```

Neste caso, `t` é definido como um escalar pelo usuário. No entanto, o MATLAB interpreta a variável `t` como uma matriz de uma coluna e uma linha. Assim, vetores e matrizes também são definidos da mesma maneira.

```
>> T=[0 1 2 3 4 5 6 7 8 9 10]

T =

     0     1     2     3     4     5     6     7     8     9    10
```

Esta última definição também pode ser realizada de uma maneira mais simples, fornecendo o valor inicial, o incremento e o valor final,

```
>> T=0:1:10

T =

     0     1     2     3     4     5     6     7     8     9    10
```

As variáveis `t` e `T` foram armazenadas na memória do MATLAB. Isto pode ser verificado através da função `whos` que mostra o nome, o tamanho e classe de todas as variáveis presentes na memória.

```
>> whos

Name      Size      Bytes  Class

T         1x11      88    double array
t         1x1       8     double array

Grand total is 12 elements using 88 bytes
```

Cabe ressaltar que o MATLAB faz diferença entre variáveis maiúsculas e minúsculas. Assim, `'t'` e `'T'` são variáveis distintas para o MATLAB. Pode-se observar que a dimensão das duas variáveis é dada em linhas x colunas. Portanto, a variável escalar `t` é definida como uma matriz de uma linha e uma coluna. Já a variável `T` tem uma linha e dez colunas. O elemento $(i \times j)$ da matriz `T`, `T(i, j)`, pode ser recuperado através do seguinte comando

```
>> T(1,5)

ans =

     4
```

A variável `ans` armazena o valor do último cálculo realizado quando este não é associado a uma variável. Neste caso, como a variável `T` possui apenas uma linha, o comando `T(5)` também forneceria o mesmo resultado. É possível também recuperar vários elementos de `T` ao mesmo tempo usando vetores dos índices correspondentes,

```
>> T(1,[1 3:6 10])  
  
ans =  
  
    0    2    3    4    5    9
```

Note que o vetor de índices é definido da mesma maneira que uma variável.

Todas as operações realizadas pelo MATLAB consideram variáveis matriciais, assim sendo a sintaxe do comando independe da dimensão da variável. Por exemplo, podemos calcular o seno de t e T usando a mesma sintaxe,

```
>> x = sin(t)  
  
x =  
  
    0.8415
```

```
>> X = sin(T)  
  
X =  
  
Columns 1 through 7  
    0    0.8415    0.9093    0.1411   -0.7568   -0.9589   -0.2794  
  
Columns 8 through 11  
0.6570    0.9894    0.4121   -0.5440
```

Desta maneira, as novas variáveis x e X , relativas a t e T , são matrizes com dimensão de t e T respectivamente, cujos elementos são os senos dos elementos t e T .

Outras operações podem ser realizadas da mesma maneira

```
>> a = 2*T  
  
a =  
  
    0    2    4    6    8   10   12   14   16   18   20
```

O MATLAB opera com números complexos da mesma forma que com números reais. A variável i representa $\sqrt{-1}$ desde que não tenha sido definida anteriormente pelo usuário.

```
>> a=1+i, b=5+2i,  
  
a =  
  
    1.0000 + 1.0000i  
  
b =  
  
    5.0000 + 2.0000i
```

As operações soma, produto e divisão são realizadas da seguinte maneira

```
>> e=a+b, g=a*b, f=a/b,  
  
e =  
  
    6.0000 + 3.0000i  
  
g =  
  
    3.0000 + 7.0000i  
  
f =  
  
    0.2414 + 0.1034i
```

O módulo e o argumento de um número complexo podem ser encontrados com os comandos `abs` e `angle`. Estes comandos são de especial interesse para o estudo de vibrações.

```
>> h=abs(b)  
  
h =  
  
    5.3852
```

```
>> d=angle(b)  
  
d =  
  
    0.3805
```

Note que o argumento é dado em radianos. A passagem para graus é feita da seguinte maneira

```
>> d/pi*180

ans =

    21.8014
```

As variáveis armazenadas na memória do MATLAB podem ser removidas da memória através do comando `clear`. Este comando é bastante interessante quando se deseja liberar uma quantidade de memória do MATLAB e/ou para manter na memória apenas as variáveis de maior interesse para facilitar o controle das mesmas. Quanto usado sem parâmetros, remove todas as variáveis da memória.

```
>> clear

>>
```

```
>> whos

>>
```

Este comando pode também ser usado para remover apenas algumas variáveis selecionadas. Primeiro, definimos algumas variáveis

```
>> a=1; b=[2 3]; A=b'*b;

>>
```

Através do comando `whos`, pode-se ver que estas variáveis foram armazenadas na memória do MATLAB

```
>> whos

Name      Size      Bytes  Class

A         2x2         32  double array
a         1x1          8  double array
b         1x2         16  double array

Grand total is 7 elements using 56 bytes
```

Vamos agora remover a variável `a` da memória usando o comando `clear` e listar novamente as variáveis presentes na memória

```
>> clear a, whos,

Name      Size      Bytes  Class

A         2x2         32  double array
b         1x2         16  double array

Grand total is 6 elements using 48 bytes
```

Os comandos básicos gerais do MATLAB estão resumidos na tabela abaixo a seguir.

Nome	Função
cd	Muda diretório corrente
clc	Limpa a janela de comandos
clear	Apaga todas as variáveis da memória do MATLAB
delete	Apaga arquivos
demo	Executa programas de demonstração das capacidades do MATLAB
dir	Lista arquivos no diretório corrente
help	Lista os tópicos de ajuda disponíveis
helpwin	Abre janela a navegação nos tópicos de ajuda
lookfor	Procura arquivos .m por palavra chave
quit	Encerra a sessão do MATLAB
whos	Mostra informações sobre as variáveis armazenadas na memória

2.3 Operações básicas com matrizes

As matrizes são tratadas da mesma maneira que escalares e vetores no MATLAB, já que estes últimos são tratados como casos particulares de matrizes. Por ser o elemento fundamental do MATLAB, nesta seção trataremos com mais detalhe de definição, inserção e extração de linhas e colunas, rearranjo e operações com matrizes.

A definição de uma matriz pode ser feita da seguinte maneira

```
>> A=[1 2 3; 4 5 6]

A =

     1     2     3
     4     5     6
```

Elementos podem ser extraídos de uma matriz usando o mesmo procedimento utilizado anteriormente para vetores, lembrando que a sintaxe do MATLAB é $A(i, j)$ com i e j denotando linha e coluna respectivamente.

```
>> A(2,3)

ans =

     6
```

Linhas e colunas inteiras podem também ser extraídas de uma matriz através da utilização o símbolo de dois pontos : para denotar todos os elementos.

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

ans =

     1     2     3

ans =

     2
     5
```

Pode-se também remover linhas e colunas de uma matriz usando a matriz vazia []

```
>> A(:,2)=[ ]

A =

     1     3
     4     6
```

Neste caso, a segunda coluna da matriz A foi removida e o resultado armazenado na própria matriz A. Poderíamos também criar uma nova matriz B eliminando a segunda coluna de A, com o comando $B=A(:, [1 \ 3])$ ¹. Desta forma, a matriz B é formada apenas pelas colunas 1 e 3 da matriz A, ou seja, a coluna 2 é eliminada.

Da mesma forma, pode-se inserir novas linhas e colunas em uma matriz. Isto pode ser feito simplesmente associando valores aos novos elementos da matriz.

```
>> A(3,:)= [7 8 9]

A =

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

Este mesmo procedimento pode ser realizado para modificar valores de elementos específicos de uma matriz

```
>> A(3,2)=10

A =

     1     2     3
     4     5     6
     7    10     9
```

Podemos também concatenar matrizes no MATLAB desde que as dimensões se-

¹Lembre-se de redefinir A, pois pelo último comando a segunda coluna já foi eliminada.

jam apropriadas,

```
>> [A A(:,1); A(1:2,:) A(2:3,2)]

ans =

     1     2     3     1
     4     5     6     4
     7    10     9     7
     1     2     3     5
     4     5     6    10
```

Operações básicas de matrizes, como transposição, determinante e inversão, são bastante simples no MATLAB. O operador de transposição é representado por ' ,

```
>> A'

ans =

     1     4     7
     2     5    10
     3     6     9
```

Lembrando que o operador ' fornece na verdade o complexo conjugado transposto. No caso de variáveis complexas, o operador .' é uma opção para transposição simples.

Os comandos para calcular o determinante e a inversa de uma matriz são os seguintes

```
>> det(A), inv(A)

ans =

    12

ans =

   -1.2500    1.0000   -0.2500
    0.5000   -1.0000    0.5000
    0.4167    0.3333   -0.2500
```

Note que estes comandos só funcionam para matrizes quadradas evidentemente. Para se conhecer as dimensões de uma variável armazenada na memória do MATLAB, pode-se usar o comando size

```
>> [l,c]=size(A)

l =

     3

c =

     3
```

As duas variáveis *l* e *c* fornecidas pelo MATLAB representam o número de linhas e colunas da matriz *A*. Outras operações com matrizes são listadas digitando-se `help matfun`.

Algumas funções do MATLAB se aplicam individualmente à cada coluna da matriz fornecida produzindo um vetor linha cujos elementos correspondem ao resultado de cada coluna. Estas funções podem ser aplicadas as linhas de uma matriz, fornecendo-se a transposta da matriz como entrada. Algumas destas funções são apresentadas na tabela abaixo.

<code>max</code>	maior componente de um vetor
<code>mean</code>	média dos componentes de um vetor
<code>min</code>	menor componente de um vetor
<code>prod</code>	produto dos componentes de um vetor
<code>sort</code>	ordena o vetor em ordem crescente
<code>std</code>	desvio padrão dos componentes de um vetor
<code>sum</code>	soma dos componentes de um vetor

Cabe observar que para um vetor '*a*' complexo, a função `sort` ordena seus componentes por ordem crescente de módulo. Os valores máximos, mínimos e médios globais de uma matriz podem ser obtidos usando a sintaxe `max(max(A))`.

O MATLAB possui também algumas matrizes especiais muito úteis, cujos elementos de entrada são número de linhas e colunas. Quando apenas um elemento é fornecido, a matriz produzida será quadrada com a dimensão requerida. Algumas destas funções são apresentadas na tabela abaixo, outras podem ser encontradas digitando `help elmat`.

<code>eye</code>	matriz identidade
<code>ones</code>	matriz com todos elementos iguais à 1
<code>rand</code>	matriz com elementos aleatórios distribuídos entre 0 e 1
<code>zeros</code>	matriz nula

As operações de adição/subtração e multiplicação de matrizes podem ser realizadas com os mesmos operadores utilizados para escalares e vetores. Os principais operadores aritméticos do MATLAB são listados digitando-se `help ops`. Sejam as matrizes

```
>> A=[1 2 3; 4 5 6], B=[1 3 ; 4 6; 3 4],  
  
A =  
  
    1    2    3  
    4    5    6  
  
B =  
  
    1    3  
    4    6  
    3    4
```

Uma nova matriz C pode ser definida como resultado da multiplicação das matrizes A e B da seguinte maneira

```
>> C=A*B  
  
C =  
  
    18    27  
    42    66
```

É claro que a multiplicação de matrizes só é definida quando o número de colunas da primeira matriz é igual ao número de linhas da segunda. Assim, se tentarmos efetuar a operação $A*A$, o MATLAB produzirá uma mensagem de erro. Pode-se também multiplicar matrizes elemento por elemento através da operação $A.*A$

```
>> A.*A  
  
ans =  
  
    1    4    9  
   16   25   36
```

Já neste caso, as dimensões das matrizes devem ser as mesmas. As operações elemento por elemento podem ser muito úteis. Assim, o último cálculo também pode ser realizado usando o comando $A.^2$. Note a diferença entre as duas operações de multiplicação

```
>> D=C*C, E=C.*C

D =

    1458    2268
    3528    5490

E =

    324    729
   1764   4356
```

Os autovalores e autovetores de uma matriz podem ser calculados através do comando `eig`. Este comando produz uma matriz diagonal D de autovalores e uma matriz V cujas colunas são os autovetores correspondentes, tal que $C*V=V*D$.

```
>> [V,D]=eig(C)

V =

   -0.8412   -0.3818
    0.5406   -0.9242

D =

    0.6479    0
    0   83.3521
```

O problema de autovalores generalizados $C*V=E*V*D$ também pode ser resolvido com a mesma função fornecendo adicionalmente a matriz E .

```
>> [V,D]=eig(C,E)

V =

    0.9341   -0.8258
   -0.3570    0.5640

D =

    0.1693    0
    0    0.0025
```

Pode-se também utilizar o comando `svd` para produzir uma decomposição em valores singulares

```
>> [U,S,V] = svd(C)

U =

    0.3831   -0.9237
    0.9237    0.3831

S =

   84.6912    0
    0    0.6376

V =

    0.5395   -0.8420
    0.8420    0.5395
```

S é uma matriz diagonal com a mesma dimensão de C e elementos não negativos em ordem decrescente na diagonal, e U e V são matrizes unitárias, tal que $C = U*S*V'$.

O comando usado para produzir a decomposição LU é

```
>> [L,U]=lu(C)

L =

    0.4286    1.0000
    1.0000     0

U =

   42.0000   66.0000
    0   -1.2857
```

U é uma matriz triangular superior e L é uma matriz “quase triangular inferior” isto é, seria uma matriz triangular inferior se A fosse substituída por PA onde P é uma matriz de permutação. A razão disto tem a ver com a escolha do pivô, e não será discutido aqui. Outra forma deste comando é

```

>> [L,U,P]=lu(C)

L =

    1.0000    0
    0.4286    1.0000

U =

   42.0000   66.0000
         0   -1.2857

P =

    0    1
    1    0

```

U é uma matriz triangular superior, L, uma matriz triangular inferior e P uma matriz de permutação, tal que $PA=LU$. Note a diferença entre L no primeiro e no segundo caso.

Outra operação de simples realização no MATLAB e de muita utilidade na prática é a resolução de sistemas lineares do tipo $A*x=b$. Além da maneira natural, porém definitivamente não recomendada, de resolver o problema $x=inv(A)*b$, o MATLAB possui uma operação de divisão à esquerda, representada pelo operador \backslash . Assim, a solução $x=A\backslash b$ é calculada fazendo eliminação de Gauss de A e retrosubstituição. Isto é feito fatorando A na forma LU. O sistema fica $LUx=b$ e o sistema é resolvido calculando y do sistema $Ly=Pb$ e então x do sistema $Ux=y$. Se a matriz A não for quadrada, o MATLAB utiliza a ortogonalização de Householder com pivotamento de colunas. O valor de x é calculado pelo método dos mínimos quadrados.

```

>> A=[1 2; 3 4]; b=[5; 6]; x=A\b

x =

   -4.0000
    4.5000

```

Esta é a solução do sistema linear

$$\begin{aligned}x + 2y &= 5 \\ 3x + 4y &= 6\end{aligned}$$

2.4 Gravação e recuperação de dados

O MATLAB permite ao usuário de gravar dados no disco e recuperar dados do disco. Estes dados são geralmente variáveis resultantes de um cálculo realizado no MATLAB que se pretende arquivar por diversos motivos. Os dados podem ser gravados num arquivo em formato texto (ASCII) ou em formato binário próprio ao MATLAB usando o comando `save`. O formato padrão é o binário, que origina um arquivo com extensão `.mat`. A sintaxe do comando `save` é mostrada no exemplo a seguir

```
>> A=[1 2 3; -4 5 -6; 7 9 11]; b=[1; 3; 5]; x=A\b;
>> save dados A b x
>> dir dados*

dados.mat
```

Vemos então que basta fornecer o nome do arquivo, sem extensão, no qual queremos gravar as variáveis. O MATLAB executa o comando criando um arquivo chamado `dados.mat` que contém as variáveis `A`, `b` e `x` e que é gravado no diretório corrente. Verifique o diretório em que está trabalhando com `cd` antes de gravar o arquivo para poder encontrá-lo depois! É importante ressaltar que o arquivo criado é um arquivo binário, isto é, não é possível editá-lo num editor de texto. Como o arquivo foi gravado em disco, podemos fechar o MATLAB que os dados não serão perdidos. Para recuperar os dados, utilizamos o comando `load`, com a seguinte sintaxe

```
>> clear
>> load dados
>> whos

Name          Size          Bytes  Class

A             3x3             72  double array
b             3x1             24  double array
x             3x1             24  double array

Grand total is 15 elements using 120 bytes
```

Se não especificássemos as variáveis a gravar, todas as variáveis armazenadas na memória do MATLAB seriam gravadas no arquivo `dados.mat`.

Algumas vezes precisamos exportar os dados do MATLAB para um outro aplicativo. Nestes casos, geralmente precisamos gravar os dados em formato ASCII. Isto pode ser feito usando o parâmetro `-ascii` no comando `save`. No entanto, precisa-se tomar alguns cuidados ao gravar várias variáveis no mesmo arquivo. Façamos, por exemplo,

```
>> save dados.txt A b x -ascii
>> type dados.txt

 1.0000000e+000  2.0000000e+000  3.0000000e+000
-4.0000000e+000  5.0000000e+000 -6.0000000e+000
 7.0000000e+000  9.0000000e+000  1.1000000e+001
 1.0000000e+000
 3.0000000e+000
 5.0000000e+000
 1.1000000e-001
 5.8000000e-001
-9.0000000e-002
```

Podemos observar que os elementos das variáveis *A*, *b* e *x* foram gravados linha-a-linha no arquivo texto *dados.txt*. Este formato só deve realmente ser utilizado para exportar dados para outros aplicativos, pois ele impõe dois problemas para recuperação dos dados no próprio MATLAB. O primeiro deles é que se perdeu a informação de a que variáveis os elementos pertencem. Assim, se tentarmos recuperar os dados através do comando `load dados.txt`, este só poderá criar uma variável de nome *dados*. O segundo problema é que ao tentar criar uma única variável com os elementos do arquivo *dados.txt*, o MATLAB será incapaz pois as linhas têm diferentes números de colunas. A opção seria gravar um arquivo por variável,

```
>> save A.txt A -ascii
>> save b.txt b -ascii
>> save x.txt x -ascii
>> clear
>> load A.txt, load b.txt, load x.txt,
```

Evidentemente, este formato só faria sentido no caso de importação de dados de outro aplicativo para o MATLAB ou exportação de dados do MATLAB para outro aplicativo.

2.5 Zeros de polinômios

Algumas vezes precisamos calcular as raízes de um polinômio. Isto pode ser feito no MATLAB através do comando `roots`. Como exemplo, vamos calcular os zeros do seguinte polinômio $p(x) = 5/8x^5 - 5x^4 - 3x^3 + 2x - 10$. Um polinômio é definido no MATLAB através de um vetor contendo os seus coeficientes. Neste caso, temos

```
>> p=[5/8 -5 -3 0 2 -10];
>> x=roots(p)

x =
    8.5587
   -0.9890 + 0.8316i
   -0.9890 - 0.8316i
    0.7097 + 0.7848i
    0.7097 - 0.7848i
```

Como podemos notar, este comando fornece todas as raízes do polinômio $p(x)$. O processo inverso também pode ser realizado. Assim, para achar os coeficientes do polinômio que tem as raízes x usamos o comando `poly`,

```
>> p=poly(x)

p =
    1.0000   -8.0000   -4.8000   -0.0000    3.2000  -16.0000
```

Notamos que não obtemos os mesmos coeficientes iniciais. No entanto, podemos obtê-los fazendo `>> 5/8*p` mostrando que o polinômio fornecido $p(x)^* = x^5 - 8x^4 - 4.8x^3 + 3.2x - 16$ é tal que $p(x) = 5/8p(x)^*$. Isto é verdadeiro já que $p(x) = 0$ implica $\alpha p(x) = 0$ e a razão do $\alpha = 8/5$ escolhido pelo MATLAB vem do fato que o comando `poly` encontra o polinômio cujo primeiro coeficiente é unitário.

2.6 Criação e edição de gráficos

O MATLAB, além de realizar cálculos, possui diversas funções permitindo também criar gráficos em duas e três dimensões a partir de dados fornecidos. O procedimento usual para a construção de um gráfico no MATLAB é apresentado a seguir

1. Preparação dos dados

```
>> x = 0:0.3:30;
>> y1 = exp(-.01*x).*sin(x); y2 = exp(-.05*x).*sin(.7*x); y3 =
exp(-.1*x).*sin(.5*x);
```

2. Escolha de uma janela para traçar o gráfico e utilização de uma das funções gráficas do MATLAB

```
>> figure(1)
>> h = plot(x,y1,x,y2,x,y3);
```

3. Seleção de tipos de linha e marcadores para as diferentes curvas

```
>> set(h,'LineWidth',2,'LineStyle','--';':':'-.')
>> set(h,'Color','r';'g';'b')
```

4. Definição de limites dos eixos e inclusão de grid

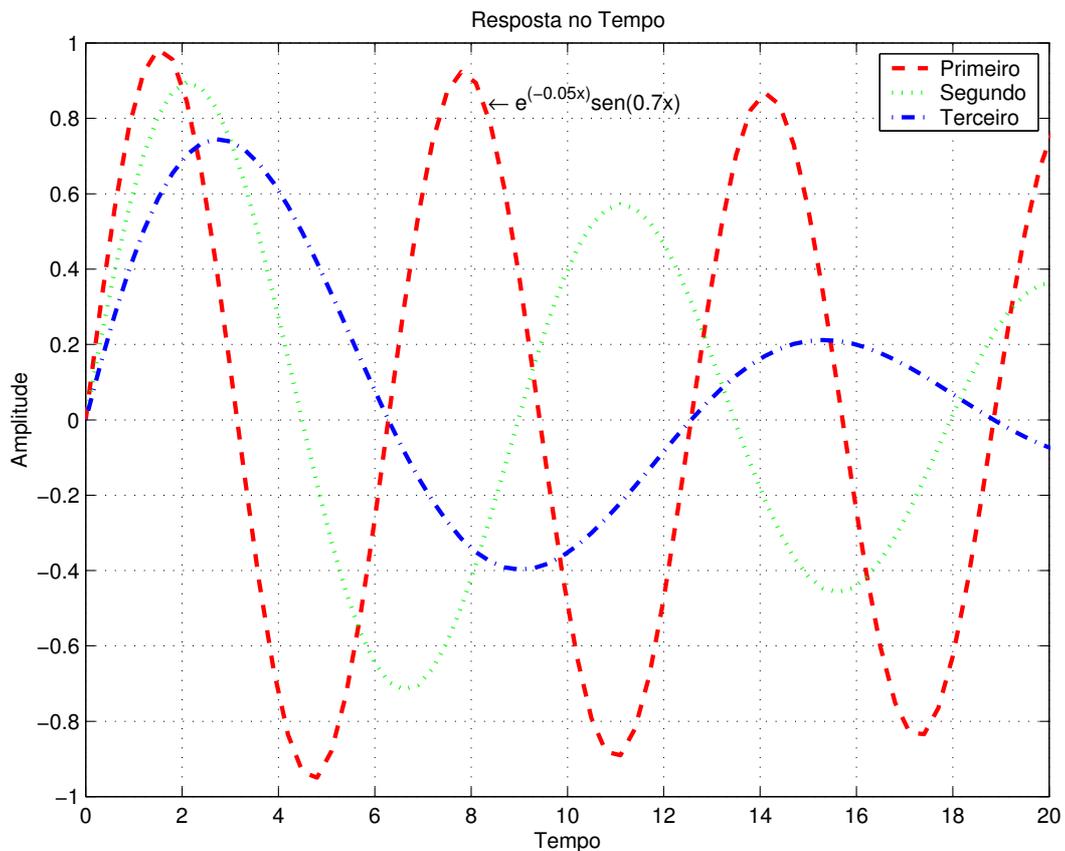
```
>> axis([0 20 -1 1]);
>> grid on
```

5. Adição de anotações no gráfico como identificação dos eixos, título, legendas e texto

```
>> xlabel('Tempo'); ylabel('Amplitude');
>> legend(h,'Primeiro','Segundo','Terceiro'); title('Resposta no
Tempo');
>> text(8.35,0.85,'\leftarrow e^{(-0.05x)}sen(0.7x)',...
'HorizontalAlignment','left')
```

6. Exportação do gráfico para algum aplicativo (de processamento de texto, por exemplo)

```
>> print -depsc -r600 myplot
```



Convém ressaltar que a seleção de tipos e cores das linhas pode ser realizada diretamente no comando gráfico plot. Assim, o mesmo resultado poderia ser obtido fazendo

```
>> h = plot(x,y1,'r--',x,y2,'g:',x,y3,'b-.');
```

As espessuras das linhas no entanto só podem ser definidas a posteriori neste caso com o comando set, como mostrado anteriormente.

Podemos também adicionar uma outra curva no gráfico já existente usando o comando hold. Quando ligado (hold on), todos os próximos gráficos serão colocados sobre aqueles já existentes até que o hold seja desligado (hold off) ou que a janela seja fechada.

```
>> hold on
>> plot(x,-y3,'m>--');
>> hold off
```

A legenda pode ser atualizada re-executando o comando legend, agora com mais um parâmetro (legend('Primeiro','Segundo','Terceiro','Quarto')).

O comando axis permite, entre outras coisas, mostrar apenas uma certa região do gráfico. Para isso, basta fornecer os intervalos em x e y (e em z para gráficos 3D) que se deseja mostrar.

```
>> axis([0 2 -1 1])
```

Neste caso, apenas os limites do eixo x foram modificados. Note que o restante do gráfico não é perdido, só não está sendo mostrado. A visualização original pode ser recuperada com o comando `axis auto`. Em alguns casos, também é interessante modificar os valores usados na marcação dos eixos. Isto pode ser feito com o comando `set`.

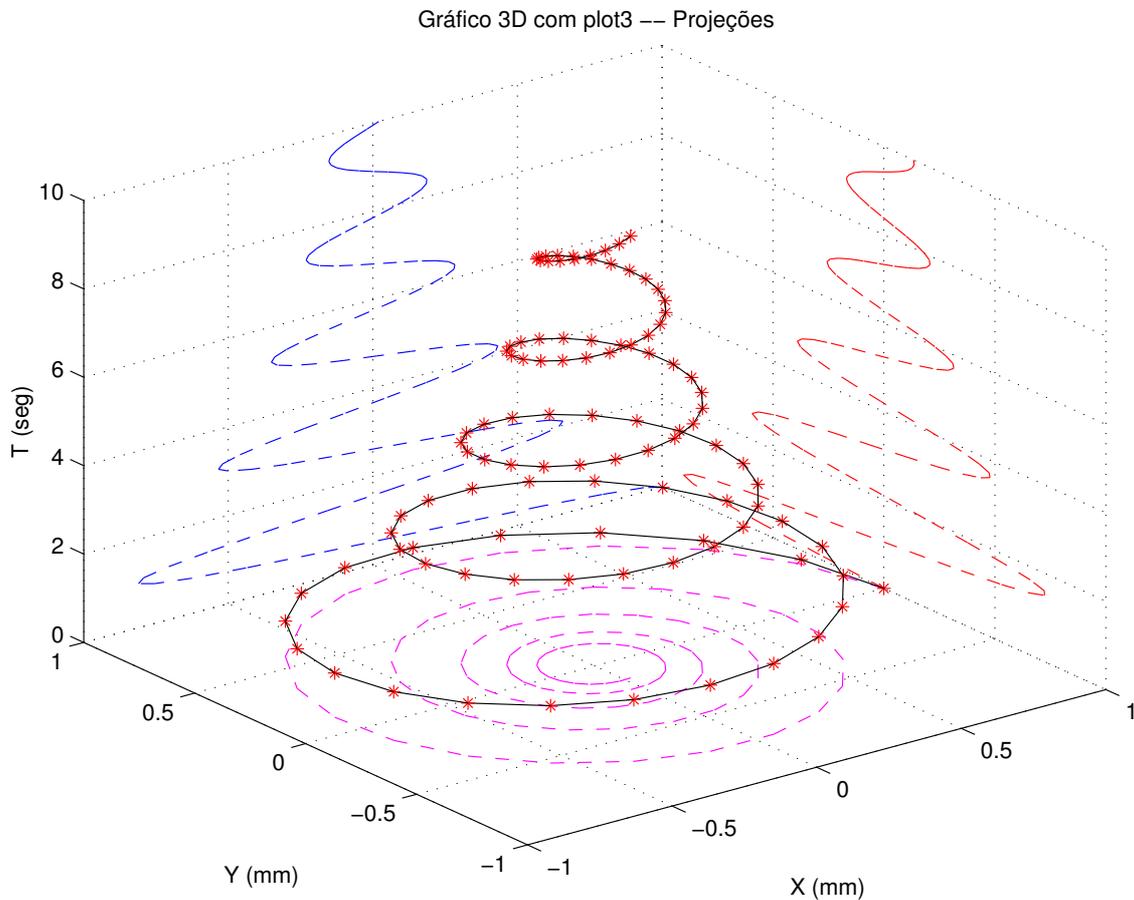
```
>> set(gca,'XTick',[0:0.25:2],'YTick',[-1:0.25:1])
```

O comando `set` pode ser utilizado para ajustar quaisquer propriedades de gráficos no MATLAB. Aqui, usamos `gca` para denotar o eixo corrente. Digitando-se `set(gca)`, obtemos todas as propriedades do eixo que podem ser modificadas e seus valores possíveis.

Existem diversos comandos para traçar gráficos também em três dimensões no MATLAB. Seguem alguns exemplos de como traçar gráficos 3D. O primeiro exemplo consiste em traçar uma curva em 3D com o comando `plot3`.

```
>> t=0:0.1:10; x=exp(-t/5).*cos(3*t); y=exp(-t/5).*sin(3*t);  
>> plot3(x,y,t,'k-',x,y,t,'r*'),  
>> hold on, plot3(x,y,zeros(size(t)),'m--',ones(size(x)),y,t,'r--',  
x,ones(size(y)),t,'b--'), hold off  
>> grid  
>> title('Gráfico 3D com plot3 -- Projeções')  
>> xlabel('X (mm)'), ylabel('Y (mm)'), zlabel('T (seg)'),  
>> print -depsc plot3
```

Na verdade, neste exemplo, cinco curvas são traçadas pelo MATLAB no mesmo gráfico. A primeira chamada do comando `plot3` imprime no gráfico os pontos (x,y,t) duas vezes, primeiro conectados por uma linha sólida de cor preta e depois com estrelas de cor vermelha. A segunda chamada de `plot3` traça projeções da curva (x,y,t) no mesmo gráfico 3D, graças ao comando `hold on`. As projeções foram feitas, traçando os seguintes conjuntos pontos: $(x,y,0)$, $(1,y,t)$ e $(x,1,t)$. Para isso foram utilizadas as matrizes especiais zeros e ones que dão origem neste caso a vetores compostos de elementos nulos e unitários respectivamente, do mesmo tamanho dos vetores x , y e t .

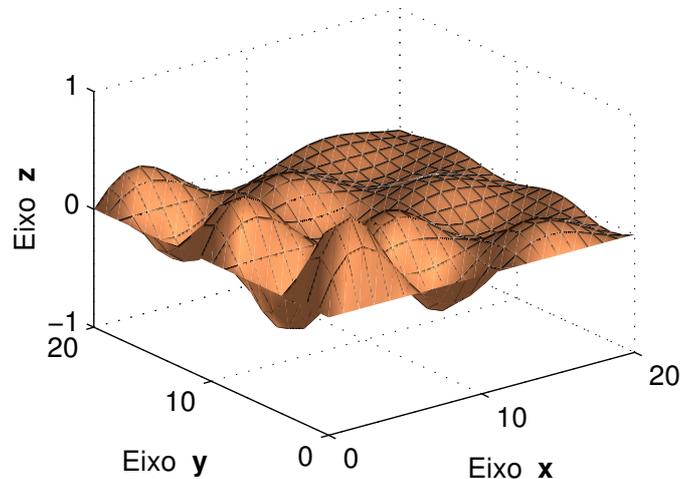


É interessante realizar o mesmo gráfico com o comando `comet3`. Este comando traça a trajetória animada “tipo cometa” dos pontos (x,y,t) e pode ser muito útil para facilitar a visualização de curva complexas no espaço tridimensional.

```
>> comet3(x,y,t)
```

Seguem agora dois exemplos de gráficos de superfície 3D usando variações do comando `mesh`.

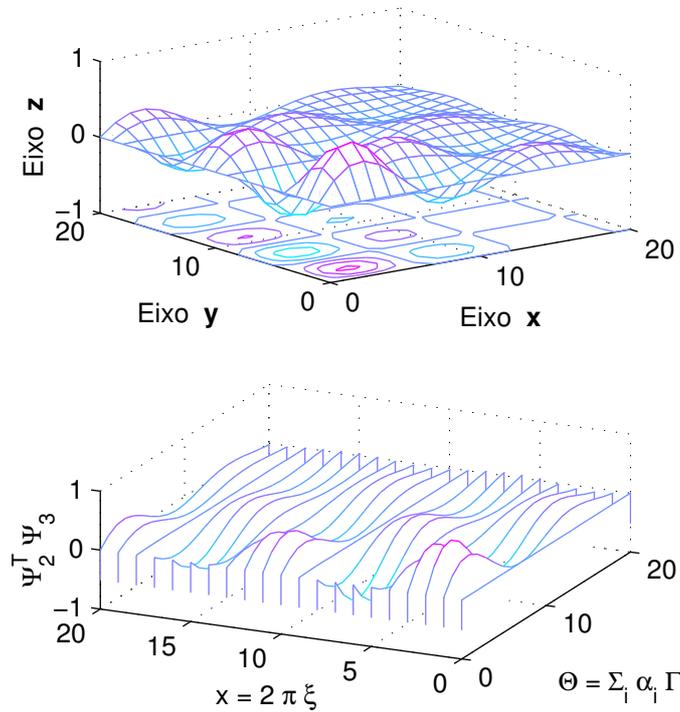
```
>> figure('Position',[232 20 330 250],'PaperPositionMode','auto')
>> h=surfl(x,x,y2'*y3);
>> shading interp
>> colormap(copper)
>> xlabel('Eixo {\bf x}'), ylabel('Eixo {\bf y}'), zlabel('Eixo
{\bf z}')
>> print -depsc surfl
```



Neste exemplo, posição e tamanho da figura são definidos na propriedade 'Position'. A propriedade seguinte 'PaperPositionMode' definida como 'auto' permite que o tamanho da figura seja mantido na impressão. Como pode ser observado, o comando `surf1` produz uma superfície iluminada a partir de uma direção pré-determinada, sendo que a aparência sólida da superfície é dada pela interpolação do mapa de cores (comando `shading`). O mapa de cores a ser utilizado é selecionado através do comando `colormap`. As opções de mapas de cores pré-definidos no MATLAB é listada com `help graph3d`.

No exemplo a seguir, utilizamos o comando `subplot` para traçar dois gráficos na mesma figura. Fazendo `subplot(n,m,j)` ou `subplot(nmj)`, divide-se a figura corrente em $n \times m$ eixos dispostos em n linhas e m colunas, sendo que o eixo j será utilizado para traçar o próximo gráfico (a contagem é feita da esquerda para direita de linha em linha).

```
>> figure('Position',[232 20 340 400],'PaperPositionMode','auto')
>> subplot(211), meshc(x,x,y2'*y3),
>> xlabel('Eixo {\bf x}'), ylabel('Eixo {\bf y}'), zlabel('Eixo
{\bf z}')
>> subplot(212), waterfall(x,x,y2'*y3), view(-65,45)
>> xlabel('\Theta = \Sigma_i \alpha_i \Gamma_i'), ylabel('x = 2 \pi
\xi'), zlabel('\Psi_2^T \Psi_3')
>> colormap(cool)
```



Os comandos gráficos 3D utilizados neste exemplo são `meshc` e `waterfall`. O primeiro é uma combinação dos comandos `mesh`, que traça uma malha colorida de acordo com o `colormap`, e `contour`, que traça as curvas de contorno. O comando `waterfall` também é uma variação do comando `mesh` mas sem traçar as linhas correspondentes às colunas (ou eixo y). Pode-se observar também que o segundo gráfico tem uma rotação diferente do primeiro. A posição do observador de gráficos em 3D pode ser modificado com o comando `view`, com o qual define-se o ângulo de observação e a altura do observador. Enfim, pode-se colocar textos com formatação mais complexa nos títulos, como aqueles apresentados no segundo gráfico, utilizando comandos `TEX`.

Os principais comandos para traçar, editar e modificar gráficos no MATLAB são apresentados nas tabelas seguintes.

Comandos para traçar gráficos	
plot	Gráfico 2D em escala linear para ambos eixos
loglog	Gráfico 2D em escala logarítmica para ambos eixos
semilogx	Gráfico 2D em escala logarítmica para o eixo x e linear para y
semilogy	Gráfico 2D em escala linear para o eixo x e logarítmica para y
plotyy	Gráfico 2D com eixos y diferentes à direita e à esquerda
comet	Traça trajetória animada 2D dos pontos (x,y)
bar	Gráfico de barras
hist	Histograma
polar	Gráfico usando coordenadas polares com ângulo em radianos
stairs	Gráfico em forma escada
stem	Gráfico 2D com seqüência discreta de y
plot3	Traça curvas em 3D
comet3	Traça trajetória animada 3D dos pontos (x,y,z)
mesh	Traça uma malha colorida em 3D, meshc (+ contorno)
surf	Traça uma superfície colorida em 3D, surfc (+ contorno)
Comandos para editar e modificar gráficos	
axis	Controla a escala e a visualização dos eixos
colormap	Especifica o mapa de cores utilizado
grid	Adiciona ou remove linhas de grelha do gráfico corrente
hold	Mantém o gráfico corrente (futuros gráficos são sobrepostos a este)
legend	Adiciona e define uma legenda no gráfico corrente
subplot	Divide uma figura em várias sub-figuras e seleciona uma delas
text	Coloca um texto no gráfico corrente
title	Título do gráfico corrente
xlabel	Título do eixo x corrente
ylabel	Título do eixo y corrente

2.7 Programando em MATLAB

2.7.1 Arquivos .m

No primeiro capítulo, os comandos básicos do MATLAB foram apresentados. Neste caso, o ambiente MATLAB foi utilizado como uma ferramenta de cálculo interativa, no sentido que várias linhas de comandos foram digitadas e executadas, uma por vez, com o objetivo de gerar um ou mais resultados numéricos ou gráficos. No entanto, é fácil imaginar que a medida que a série de comandos aumenta em complexidade, e/ou em quantidade, este procedimento se torna cansativo e pouco prático. Felizmente, o MATLAB também permite que se construa e execute programas na linguagem MATLAB. Neste caso, a seqüência de comandos digitadas e executadas, uma por vez, anteriormente pode ser concentrada em um arquivo “programa” .m que ao ser executado no MATLAB, executa cada linha do programa, composta pelas linhas de comandos usadas anterior-

mente, também uma por vez. Desta forma, o método de execução das linhas de comandos é o mesmo. No entanto, se de várias linhas de comandos queremos alterar apenas uma delas e repetir a seqüência de comandos, neste caso basta modificar a linha correspondente no arquivo e re-executar o programa. Este arquivo programa recebe o nome de **roteiro (script)** já que ele informa ao MATLAB o roteiro a ser seguido na execução da seqüência de linhas de comandos contidas no arquivo programa.

Um arquivo .m pode ser criado usando-se qualquer editor de texto. Antes de criar um novo arquivo, assegure-se que o diretório corrente é de seu uso pessoal. É altamente recomendado que cada usuário trabalhe em um diretório de uso pessoal para evitar conflito com arquivos de outros usuários. O procedimento para mudar de diretório e verificar o diretório corrente é o seguinte:

```
>> cd c:\users\myname
>> cd

c:\users\myname
```

O usuário pode ainda listar os programas contidos no diretório corrente de duas maneiras. O comando `dir` lista todos os arquivos no diretório corrente enquanto que o comando `what` lista somente os arquivos específicos do MATLAB

```
>> dir

.          ap31.m          mm2.m          p2.m          teste.eps
..         ap4.m          odecomp.m      p_eqlin.m     teste.m
ap2.m      ap41.m           p1.m          pend.m
ap2result.mat d1_bode.m      p11.m         pend_eq.m
ap3.m      mm1.m           p12.m         sample.tex
```

```
>> what

M-files in the current directory d:\trindade\cursos\Vib

ap2      ap4      mm1      p1      p2      pend_eq
ap3      ap41    mm2      p11     p_eqlin teste
ap31     d1_bode odecomp  p12     pend

MAT-files in the current directory d:\trindade\cursos\Vib

ap2result
```

As versões mais atuais do MATLAB (5.x e 6.x) possuem um editor de texto integrado. Para criar um novo arquivo .m, selecione no menu principal do MATLAB "File/New/M-file" ou pressione o botão "Folha em branco". Para fixar o procedimento, digite o programa seguinte no arquivo recém-criado:

```
clear all
t=0:0.1:10;
y=sin(t).*cos(t);
plot(t,y,'bo-')
```

Grave o arquivo com um nome qualquer (por exemplo, exemplo1.m) selecionando “File/Save” no menu. O programa pode agora ser executado na janela de comandos do MATLAB fazendo `>> exemplo1`. Certifique-se que o diretório corrente é o mesmo utilizado para gravar o arquivo .m. Este programa traça $t \in [0, 10]$ contra $y(t) = \sin t \cos t$ com linha sólida e círculos azuis.

Assim como os comandos do MATLAB, o usuário também pode criar programas que atuam como funções, isto é fornecem um ou mais resultados quando um ou mais parâmetros são fornecidos. Funções são também arquivos .m e podem ser criadas da mesma maneira que os programas roteiro. Para exemplificar, criemos uma função $sc(x) = \sin x \cos x$ num novo arquivo .m:

```
function f = sc(x)

f = sin(x).*cos(x);
```

É preciso gravar este arquivo com o mesmo nome da função, isto é `sc.m`. Note que agora a recém-criada função `sc` pode ser utilizada no MATLAB da mesma maneira que as funções (comandos) do MATLAB.

```
>> a = sc(2)

a =

-0.3784
```

É interessante observar que as variáveis utilizadas na função, `f` e `x`, são internas à função. De fato, executando-se o comando `whos` veremos que estas variáveis não foram armazenadas na memória do MATLAB. A variável `a` assume o resultado da função e o parâmetro `x` utilizado pela função é igual ao valor fornecido pelo usuário, neste caso 2.

Podemos também utilizar esta nova função para reescrever o programa roteiro `exemplo1` da seguinte forma:

```
clear all
t=0:0.1:10;
y=sc(t);
plot(t,y,'bo-')
```

2.7.2 O comando `global`

Apesar de as funções criadas no MATLAB só utilizarem variáveis internas à função em questão ou locais, pode-se também utilizar variáveis armazenadas

na memória do MATLAB através do comando `global <var1> <var2>`. Este comando determina que as variáveis `<var1> <var2> ...` são globais, ou seja podem ser utilizadas em outros programas.

```
function f = sc(x) % Arquivo sc.m
global p

f = sin(p*x).*cos(x);
```

```
clear all % Arquivo principal.m (chama arquivo função sc.m)
global p
p=3;
t=0:0.1:10;
y=sc(t);
plot(t,y,'bo-')
```

2.7.3 O comando if

O comando `if` permite que se utilize comandos condicionais no MATLAB, tanto no Command Window como no interior de roteiros ou funções. A melhor maneira de entender a sintaxe deste comando é através de um exemplo. Considere a função `sc` definida anteriormente. Suponha que a função agora tenha um valor limite para o parâmetro p , de forma que se $p < 1$ a função retorna $sc(x) = \sin x \cos x$ e não $sc(x) = \sin px \cos x$. Isto pode ser realizado modificando novamente o arquivo `sc.m` e incluindo o comando `if`.

```
function f = sc(x) % Arquivo sc.m
global p

if p<1
    f = sin(1*x).*cos(x);
else
    f = sin(p*x).*cos(x);
end
```

O MATLAB interpreta esta função da seguinte maneira: 1) o valor de p definido na memória do MATLAB é recuperado, 2) se o valor de p for menor que 1, a função executa a linha `f = sin(1*x).*cos(x);` e retorna o valor de `f` como resposta, 3) se o valor de p for maior que ou igual a 1, a função executa a linha `f = sin(p*x).*cos(x);` e retorna o valor de `f` como resposta.

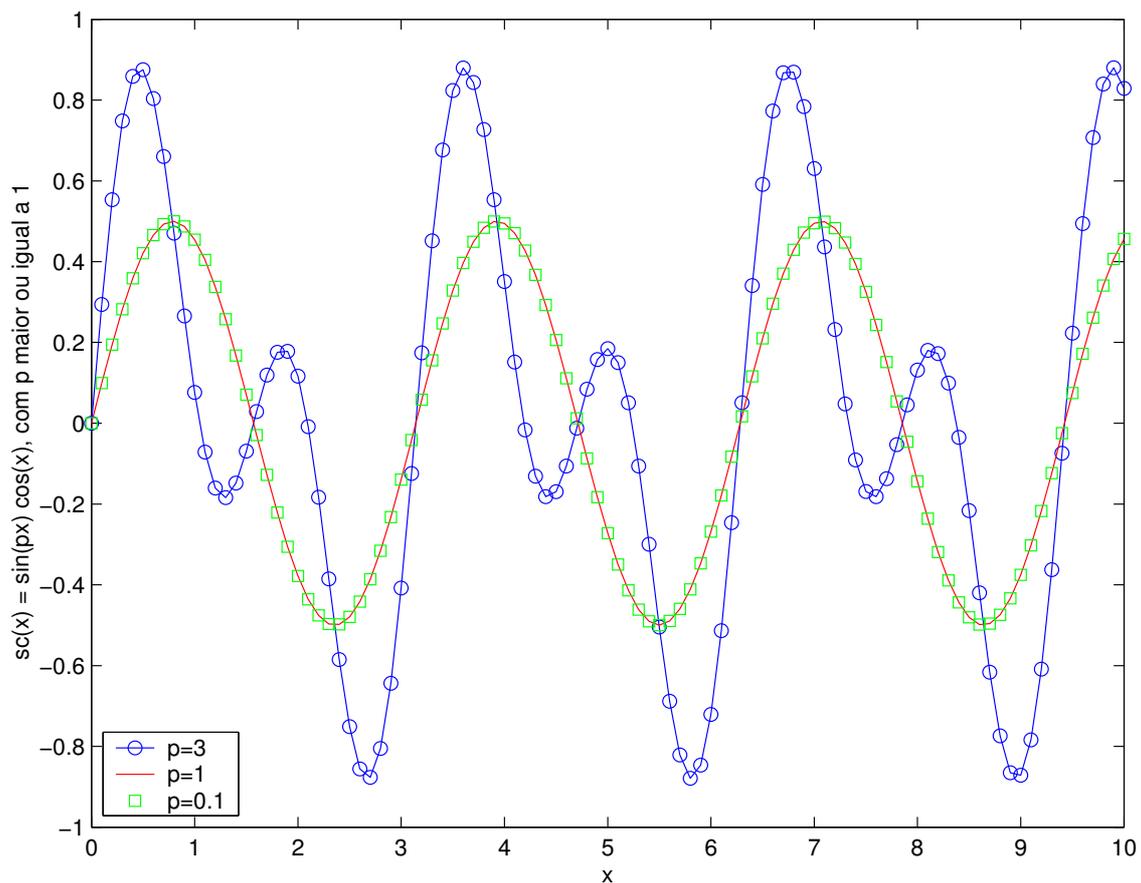
Desta forma, mesmo que p seja definido menor que 1 no programa `principal.m`, o mínimo valor de p considerado na função `sc` é 1. O programa seguinte ilustra a utilização da nova função `sc` com o comando `if` incluído.

```

clear all % Novo arquivo principal.m (chama arquivo função sc.m)
global p
t=0:0.1:10;
p= 3; y1=sc(t);
p= 1; y2=sc(t);
p=0.1; y3=sc(t); % p<1
plot(t,y1,'bo-',t,y2,'r-',t,y3,'gs')
xlabel('x'),
ylabel('sc(x) = sin(px) cos(x), com p maior ou igual a 1'),
legend('p=3', 'p=1', 'p=0.1',3)

```

Executando o arquivo principal.m no MATLAB, a seguinte figura é produzida.



Alternativamente, várias condições podem ser consideradas utilizando o comando `if` em conjunção com `elseif`. O exemplo seguinte ilustra esta utilização do comando `if`.

```
function f = sc(x) % Arquivo sc.m
global p

if p<1
    f = sin(1*x).*cos(x);
elseif p<2
    f = sin(2*x).*cos(x);
elseif p<3|p==7
    f = sin(3*x).*cos(x);
else
    f = sin(p*x).*cos(x);
end
```

Os operadores relacionais e os conectivos lógicos que podem ser usados com o comando `if` são listados na tabela a seguir.

Operadores relacionais	
<	menor que
>	maior que
<=	menor que ou igual a
>=	maior que ou igual a
==	igual a
~=	diferente de
Conectivos lógicos	
&	e
	ou
~	não
xor	ou excludente

2.7.4 O comando `for`

O comando `for` permite que se repita uma série de comandos para diferentes valores de uma ou mais variáveis. A sintaxe do comando `for` é a seguinte:

```
for k=ki:dk:kf
    comandos
end
```

onde a linha `comandos` é executada para cada um dos seguintes valores da variável controladora `k`: `ki`, `ki+dk`, `ki+2dk`, ..., `kf`. Note que se $(kf - ki)$ não for múltiplo de `dk`, o último valor assumido por `k` é `ki+N dk`, onde `N` é o máximo valor inteiro para o qual `ki+N dk` é menor que `kf`.

O valor do incremento `dk` pode também ser negativo desde que `kf` seja menor que `ki`. Sempre que o incremento `dk` não é fornecido, utilizando por exemplo `for k=ki:kf`, o MATLAB supõe que `dk` é igual a 1. Tente reproduzir os exemplos

abaixo²:

```
for k=1:0.5:3
    disp(k)
end
```

```
1
1.5000
2
2.5000
3
```

Aqui k começa em 1 e é incrementado de 0.5 até chegar em 3.

```
for k=1:-0.4:-1
    disp(k)
end
```

```
1
0.6000
0.2000
-0.2000
-0.6000
-1
```

Aqui k começa em 1 e é decrementado de 0.4 até chegar em -1.

```
for k=1:3.5
    disp(k)
end
```

```
1
2
3
```

Aqui k começa em 1 e é incrementado de 1 até chegar em 3. Ele não chega a 3.5 porque o próximo valor incrementado seria 4.

²Lembre-se que você pode usar o comando for no command window ou em roteiros ou funções (.m). Reproduza estes exemplos das duas maneiras.

Capítulo 3

Aplicações

3.1 Programas interativos

Muitas vezes é interessante fazer um programa interativo de forma que o usuário possa executá-lo entrando os dados necessários ao programa via teclado sem ter que editar o arquivo .m do programa. Isto pode ser feito através do comando input. Considere por exemplo que você quer fazer um programa que calcula valores para a função $f(x) = ae^{-2x+b}$, e plota x versus $f(x)$, a partir dos valores inicial e final de x (x_i e x_f) e dos parâmetros a e b . No entanto, você também quer que o usuário entre com estes valores sem ter que alterar o seu programa. Isto pode ser feito da seguinte maneira:

```
function [y,x] = interat
clear
disp('Programa Interat - Versão 1.0 ')
a = input('Entre com o valor de a = ');
b = input('Entre com o valor de b = ');
xi= input('Entre com o valor inicial de x = ');
xf= input('Entre com o valor final de x = ');
x = xi:(xf-xi)/99:xf; % define 100 valores de x entre xi e xf
y = a*exp(-2*x+b);
plot(x,y,'ro-')
```

Quando o usuário executar o programa interat, os valores de a , b , x_i e x_f são pedidos ao usuário pelo programa. O usuário pode fornecer os valores simplesmente digitando o valor pedido seguido de $\langle \text{Enter} \rangle$. Assim que os quatro valores são fornecidos o programa calcula os valores de $f(x)$ e plota o gráfico. Note que a função interat também fornece como saída os valores de x e $f(x)$. Se o usuário executar o programa fazendo \gg interat os valores de x e $f(x)$ são imprimidos no comand window após execução do programa. Se o usuário executar o programa fazendo \gg f=interat; os valores de $f(x)$ são armazenados na memória do MATLAB na variável f. Os valores não são impressos na

tela e os valores de x são perdidos. Se o usuário executar o programa fazendo `[f,x]=interat`; os valores de x e $f(x)$ são armazenados na memória do MATLAB nas variáveis x e f respectivamente.

3.2 Zeros de funções

O MATLAB acha zeros de funções usando o comando `fzero`. A função, da qual deseja-se encontrar os zeros, deve ser definida em um arquivo `.m` como definido previamente. Considere a seguinte função $f(x) = \sin x - \cos x$. A função `apl1` é então escrita

```
function f=apl1(x)
f=sin(x)-cos(x);
```

A raiz pode ser determinada usando o comando `fzero` no comand window do MATLAB ou no interior de um outro programa.

```
>> fzero('apl1',1)

ans =
    0.7854
```

Note que o segundo argumento 1 é um chute inicial para o cálculo da raiz. Note também que o valor do ângulo x que satisfaz $f(x) = 0$ está em radianos (0.7854 radianos = 45 graus). No entanto, existem outros valores de x para os quais $f(x) = 0$, isto é, a função $f(x)$ tem outras raízes. Dando o chute inicial 3, por exemplo, tem-se

```
>> fzero('apl1',3)

ans =
    3.9270
```

ou, em graus, 225. Este comando pode ser utilizado com qualquer outra função escrita no MATLAB.

3.3 Calculando Integrais

Pode-se também calcular integrais de uma função no MATLAB. Existem vários métodos de aproximação numérica de integrais definidas implementados no MATLAB. Considere a seguinte integral

$$I = \int_a^b f(x) dx$$

Esta integral pode ser aproximada usando a regra de Simpson com o comando `quad` ou usando a regra trapezoidal com o comando `trapz`. Os dois comandos trabalham de maneira bem diferente.

3.3.1 Regra trapezoidal

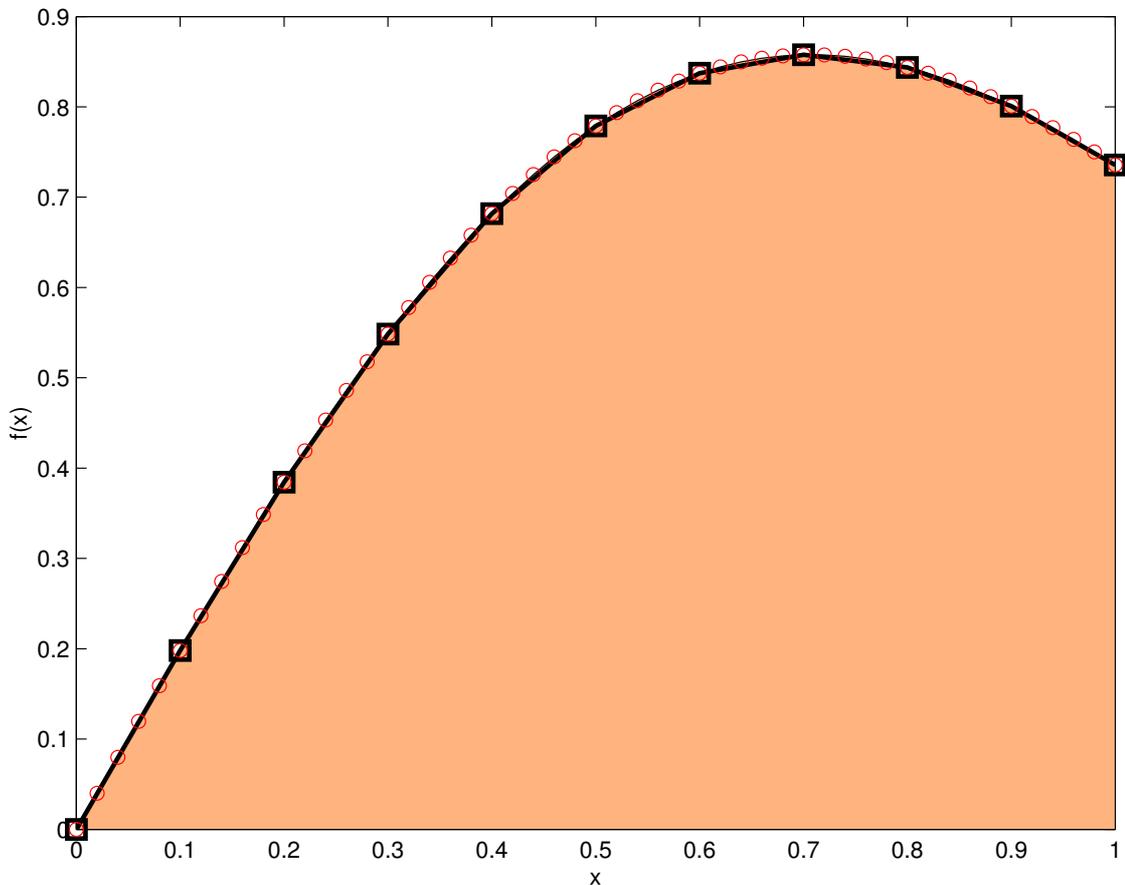
O comando `trapz` aproxima a integral usando os pontos da função definidos pelo usuário. Assim, para usar este comando primeiro deve-se definir os vetores x , no intervalo $[a, b]$, e f , os valores da função correspondentes aos valores definidos no vetor x . Considere, por exemplo, a seguinte integral

$$I = \int_0^1 2xe^{-x^2} dx$$

A aproximação para esta integral pode ser calculada através do seguinte programa:

```
function I=integral(dx)
x=0:dx:1;
f=2*x.*exp(-x.^2);
I=trapz(x,f);
```

Este programa cria um vetor x de $a = 0$ até $b = 1$, com incremento dx definido pelo usuário, calcula os valores de $f(x)$ nestes pontos e usa o comando `trapz` para aproximar a integral de $f(x)$. A figura seguinte mostra graficamente a função para $dx = 0.1$ e para $dx = 0.02$. A área sob a curva é a integral definida da função $f(x)$.



O cálculo da integral é feito no MATLAB, usando a recém-definida função `integral`, da seguinte maneira

```
>> integral(.1)
```

```
ans =  
0.6298
```

```
>> integral(.02)
```

```
ans =  
0.6320
```

Observe que as aproximações da integral I para $dx = 0.1$ e $dx = 0.02$ são diferentes. De fato, usando o incremento menor $dx = 0.02$ aproxima-se melhor a área sob a curva $f(x)$. O valor exato desta integral é $I_{exato} = 1 - e^{-1}$ ou calculando no MATLAB $I = 0.63212055882856$. Observando o gráfico da função, é fácil imaginar porque a diminuição do incremento dx melhora a aproximação da integral. No entanto, na próxima seção, um método mais eficiente para o cálculo da integral é apresentado.

3.3.2 Regra de Simpson

O comando `quad` permite que se calcule a integral I usando a regra de Simpson. A sintaxe deste comando é a seguinte:

```
quad('fc',a,b,tol)
```

sendo que 'fc' é o arquivo .m que define a função da qual se quer calcular a integral. a e b são os limites de integração e tol é a tolerância de erro exigida para a aproximação da integral. Mostra-se mais adiante que, quanto menor a tolerância de erro exigida, mais preciso é o cálculo da integral.

Primeiro, cria-se o programa `apl2.m` que define a função $f(x)$.

```
function y=apl2(x) % função apl2.m
y=2*x.*exp(-x.^2);
```

O cálculo da integral I pode ser efetuado usando a tolerância usual que é de 10^{-3} . Neste caso, é desnecessário definir `tol` no comand window.

```
>> quad('apl2',0,1)
```

```
ans =
    0.63212053454568
```

Calculando agora com uma tolerância de 10^{-8}

```
>> quad('apl2',0,1,1e-8)
```

```
ans =
    0.63212055882677
```

Note que, comparando-se com o valor exato apresentado na seção anterior, quando diminui-se a tolerância melhora-se a aproximação da integral.

3.4 Equações diferenciais

No MATLAB existem diversos algoritmos para resolver sistemas de equações diferenciais ordinárias de primeira ordem. Estes são mostrados na tabela abaixo.

Integradores de equações diferenciais ordinárias	
ode45	Runge-Kutta explícito de ordem 4/5 (Dormand-Prince)
ode23	Runge-Kutta explícito de ordem 2/3 (Bogacki-Shampine)
ode113	Preditor-corretor de passo variável (Adams-Bashforth-Moulton)
ode15s	BDF de passo quase constante para EDOs rígidas (Klopfenstein-Shampine)
ode23t	Implementação da regra trapezoidal
ode23s	Implementação de um par Rosenbrock (2,3) modificado para EDOs rígidas
ode23tb	Runge-Kutta implícito trapezoidal/BDF (Bank-Rose-Hosea-Shampine)

A seguir, alguns exemplos de solução de equações diferenciais ordinárias são apresentados.

3.4.1 Sistema massa-mola-amortecedor

Considere um sistema massa-mola-amortecedor descrito pela equação diferencial ordinária seguinte

$$m\ddot{x} + c\dot{x} + kx = f \sin t$$

Note que esta equação diferencial ordinária possui ordem 2 (pois \ddot{x} é derivada segunda de x no tempo, d^2x/dt^2). No entanto, esta equação pode ser escrita como um sistema de duas equações de primeira ordem da seguinte forma

$$\begin{aligned} y_1 = x &\Rightarrow \dot{y}_1 = \dot{x} \\ y_2 = \dot{y}_1 = \dot{x} &\Rightarrow \dot{y}_2 = \ddot{x} = \frac{f}{m} \sin t - \frac{c}{m}\dot{x} - \frac{k}{m}x = \frac{f}{m} \sin t - \frac{c}{m}y_2 - \frac{k}{m}y_1 \end{aligned}$$

Em resumo, tem-se duas equações de primeira ordem

$$\begin{aligned} \dot{y}_1 &= y_2 \\ \dot{y}_2 &= \frac{f}{m} \sin t - \frac{c}{m}y_2 - \frac{k}{m}y_1 \end{aligned}$$

Primeiro, escreve-se uma função que calcula \dot{y}_1 e \dot{y}_2 a partir dos valores de t , y_1 e y_2 . Esta função é chamada diversas vezes no processo de integração do sistema de equações diferenciais.

```
function dydt=apl3(t,y,opt,P)
m = P(1); c = P(2);
k = P(3); f = P(4);
dydt = [y(2); (f*sin(t) - c*y(2) - k*y(1))/m];
```

Note que as variáveis y_1 , o deslocamento da massa, e y_2 , a velocidade da massa, foram agrupadas no vetor y . Assim, a saída da função `apl3` tem como elementos \dot{y}_1 e \dot{y}_2 :

$$y = \begin{Bmatrix} y_1 \\ y_2 \end{Bmatrix}; \quad dydt = \begin{Bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{Bmatrix}$$

A sintaxe para a utilização dos comandos 'ode*' na integração da equação diferencial do massa-mola-amortecedor é a seguinte

```
>> [t,Y]=ode45('apl3',[ti tf],Yi);
```

sendo que t_i e t_f são os instantes de tempo inicial e final e Y_i é o vetor de condições iniciais. Note que Y_i deve ser um vetor coluna (neste caso, com duas linhas) tal que

$$Y_i = \begin{Bmatrix} y_1|_{t=0} \\ y_2|_{t=0} \end{Bmatrix} = \begin{Bmatrix} x(0) \\ \dot{x}(0) \end{Bmatrix}$$

A função `ode45`, assim como as outras funções de integração de EDOs, retorna um vetor coluna t contendo os instantes de tempo em que a solução foi calculada

e uma matriz Y com mesmo número de linhas que t . Cada coluna de Y corresponde ao comportamento de uma variável no tempo. No caso deste exemplo, a primeira coluna representa os deslocamentos da massa assumidos em cada instante de tempo e a segunda coluna representa as velocidades da massa assumidas em cada instante de tempo.

A linha de comando utilizada para executar a integração das equações pode também ser parte de um outro programa. Considere, por exemplo, o seguinte programa

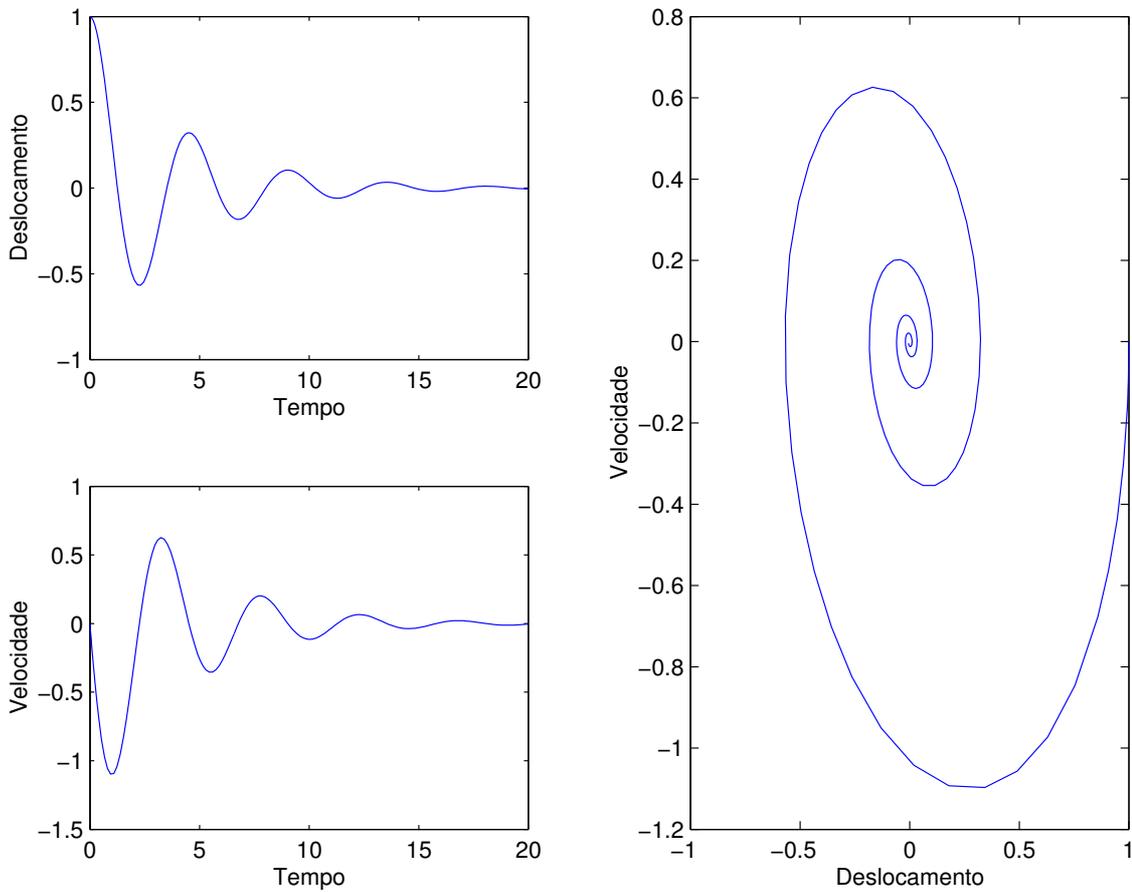
```
function mma(m,c,k,f,xi,vi,ti,tf)
parametros = [m c k f];
Yi = [xi; vi];
[t,Y] = ode45('apl3',[ti tf],Yi,[],parametros);
subplot(221)
plot(t,Y(:,1));
xlabel('Tempo'); ylabel('Deslocamento');
subplot(223)
plot(t,Y(:,2));
xlabel('Tempo'); ylabel('Velocidade');
subplot(122)
plot(Y(:,1),Y(:,2))
xlabel('Deslocamento'); ylabel('Velocidade');
```

Note que os parâmetros m , c , k e f são passados do programa principal `mma` para a integração das equações através do vetor `parametros`. O recebimento dos parâmetros pela função que calcula $dydt$ é feita através do argumento extra `P`. O parâmetro `opt` não tem utilidade mas é necessário devido à sintaxe de chamada do integrador. É possível também fazer o mesmo programa utilizando outros integradores de EDOs. Para isto, basta substituir `ode45` por outro integrador (`ode15s` por exemplo).

Assim, pode-se executar o programa principal no comand window do MATLAB

```
>> mma(1, .5, 2, .5, 1, 0, 0, 10)
```

sendo que o primeiro valor corresponde à massa m , o segundo ao amortecedor c , o terceiro à rigidez k , o quarto à amplitude da força f , o quinto ao deslocamento inicial $x(0)$, o sexto à velocidade inicial $\dot{x}(0)$, o sétimo ao instante inicial t_i e o oitavo ao instante final t_f . A próxima figura apresenta o resultado.



3.4.2 Pêndulo simples

A dinâmica do pêndulo simples é representada pela seguinte equação diferencial ordinária

$$\ddot{\theta} + \frac{g}{l} \sin \theta = 0$$

Seguindo a metodologia apresentada na seção anterior, esta equação pode ser integrada usando um programa principal que explicita o método de integração e de um programa auxiliar que defina a equação a resolver. Primeiramente, deve-se transformar a equação anterior em um sistema de equações de primeira ordem.

$$\begin{aligned} \dot{y}_1 &= y_2 \\ \dot{y}_2 &= -\frac{g}{l} \sin y_1 \end{aligned}$$

O programa principal `pendulo.m` é então muito similar àquele utilizado para o problema do sistema massa-mola-amortecedor.

```
function pendulo(g,l,xi,vi,ti,tf)
parametros = [g l];
Yi = [xi; vi];
[t,Y] = ode45('apl4',[ti tf],Yi,[],parametros);

subplot(221)
plot(t,Y(:,1));
xlabel('Tempo'); ylabel('Deslocamento');
subplot(223)
plot(t,Y(:,2));
xlabel('Tempo'); ylabel('Velocidade');
subplot(122)
plot(Y(:,1),Y(:,2))
xlabel('Deslocamento'); ylabel('Velocidade');
```

e o programa auxiliar que define a equação do pêndulo é

```
function dydt=apl4(t,y,opt,P)

g = P(1); l = P(2);

dydt = [y(2); -g/l*sin(y(1))];
```

Pode-se então executar o programa principal pendulo no comand window do MATLAB:

```
>> pendulo(9.81,10,1,0,0,20)
```

Experimente aumentar o valor de xi de 1 para 2 e 3, para ver o que acontece com os gráficos. Você consegue distinguir o comportamento deste sistema não-linear daquele do sistema linear massa-mola-amortecedor.

Procure na biblioteca de arquivos .m os programas mmola.m e pduplo.m. Execute estes arquivos no MATLAB e, em seguida, veja como eles foram feitos (>> type mmola) e (>> type pduplo). Apesar de serem mais extensos e usarem alguns comandos novos, estes dois programas utilizam a mesma metodologia apresentada nas últimas duas seções para resolver as equações diferenciais de um sistema massa-mola-amortecedor com 3 graus de liberdade e de um pêndulo duplo.