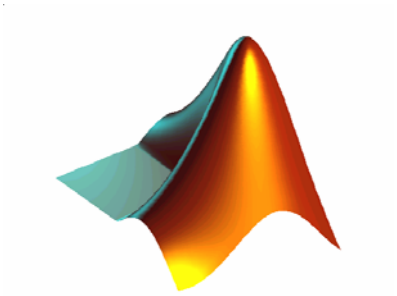

MATLAB

Dicas iniciais de utilização

M.Sc. – Programa de Engenharia Química/COPPE/UFRJ – Janeiro de 2003

Andréa Oliveira Souza da Costa
costa@peq.coppe.ufrj.br

CONTEÚDO

O objetivo desta apostila é apresentar ao usuário iniciante do MATLAB 6 noções básicas de utilização deste programa. Assim, é realizada uma rápida descrição do *software*, onde são apresentadas as principais características e potencialidades do programa. Posteriormente, separados em itens, são descritos alguns comandos importantes para a utilização do MATLAB. Noções de programação em MATLAB também são apresentadas nesta apostila, bem como os comandos empregados na resolução de sistemas de equações não lineares e de equações diferenciais ordinárias. Sugestões para consultas futuras e alguns exercícios de fixação são apresentadas ao final da apostila.

MATLAB é uma abreviação para MATrix LABoratory. Trata-se de um ambiente de alto nível que possui ferramentas avançadas de análise e visualização de dados. Mais do que um aplicativo, o MATLAB também possui características de linguagem de programação.

A programação em ambiente MATLAB dispensa tarefas como declaração de variáveis, alocação de memória, utilização de ponteiros, necessárias durante a utilização de linguagens de programação como C ou Fortran.

As funções matemáticas já existentes no MATLAB são otimizadas, programadas em linguagem MATLAB e estão agrupadas de acordo com a área de interesse em *toolboxes*. Assim, o usuário tem acesso aos arquivos das funções matemáticas o que possibilita a realização de alterações nas rotinas já existentes. Todavia, vale ressaltar que estas alterações são perigosas e só devem ser realizadas como última alternativa.

Antes de abordar a programação em ambiente MATLAB, faz-se necessária uma análise da execução de comandos de linha)diretamente digitados na área de trabalho (*workspace*)). Inicia-se esta análise discutindo como se trabalha com variáveis em ambiente MATLAB.

1. Trabalhando com variáveis

No MATLAB os nomes das variáveis devem ser palavras únicas, sem a inclusão de espaços e não devem conter acentos. As regras básicas para nomes de variáveis são apresentadas na Tabela 1.1.

Tabela1.1: Regras básicas para nomes de variáveis no MATLAB.

As variáveis são sensíveis a letras maiúsculas e minúsculas	`Itens`, `itens` e `ITENS` São entendidas como diferentes variáveis.
As variáveis podem possuir até 31 caracteres. Os caracteres além do 31^{o} são ignorados.	`Oquevoceachadestenomedevariavel` Pode ser usado como nome de variável.
O nome da variável deve começar com uma letra, seguida de qualquer número, letra ou sublinhado.	`O_que_voce_acha_deste_nome` e `X51` podem ser utilizados como nome de variáveis.

Existem algumas variáveis especiais que o MATLAB utiliza que são apresentadas na Tabela 1.2. Se o usuário redefine estas variáveis, o MATLAB passa a atribuir a nova função às mesmas.

Tabela 1.2: Variáveis especiais utilizadas pelo MATLAB.

Variável	*Significado*
`ans`	Variável padrão usada para resultados.
`pi`	Razão entre o perímetro da circunferência e seu diâmetro.
`eps`	Precisão relativa da máquina.
`inf`	Infinito
`NaN nan`	Não numérico
`i j`	i = j = $\sqrt{-1}$
`nargin`	Número de argumentos de entrada de uma função.
`nargout`	Número de argumentos de saída de uma função.
`realmin`	Menor número real positivo utilizável pela máquina.
`realmax`	Maior número real positivo utilizável pela máquina.

Caso o usuário necessite apagar alguma variável da memória do MATLAB, isto pode ser realizado utilizando-se o comando "`clear`". Por exemplo:

```
» a=10;
» a
a =
    10
» clear a
» a
??? Undefined function or variable 'a'.
```

Se a necessidade do usuário for de apagar todas as variáveis que estão sendo utilizadas deve-se utilizar o comando "`clear all`". Se por outro lado, o usuário deseja a listagem de todas as variáveis que estão sendo utilizadas, basta utilizar o comando "`who`". Por exemplo:

```
>> a=10; b=a;
>> who
Your variables are:
a  b
>>
```

Todo o texto depois do sinal de percentagem (%) é considerado comentário. Além disso, pode-se colocar mais de um comando em uma linha, separando-os por vírgula ou ponto e vírgula. A vírgula diz ao MATLAB para mostrar o resultado após executar o comando. Já o ponto e vírgula dispensa a visualização. Por exemplo:

```
» a=10; b=20, % isto é um comentario
b =
    20
» a
a =
    10
»
```

Quando se deseja continuar o comando na próxima linha, o sinal utilizado pelo MATLAB é representado por 3 pontos (`...`). Isso só funcionará se os pontos estiver entre nomes de variáveis e operações. Este comando não funciona para comentários. Ou seja:

```
>> a=10; b=20;
>> c=a+...
b
c =
    30
>>
```

O usuário pode interromper a execução do MATLAB, a qualquer momento, pressionando o **Crtl-c**. Para limpar o *workspace* o usuário deve utilizar o comando "`clc`".

1.1. Utilizando strings

O MATLAB entende como *strings* o conjunto de caracteres (vetor de caracteres) colocados entre aspas simples. Assim, para acessar uma parte da variável é necessário listar a localização dos caracteres. Ou seja:

```
>> s='esta variavel e uma string'
s =
esta variavel e uma string
>>
>> s(6:13), % retirando a palavra variavel da string s
ans =
variavel
>>
```

2. Utilizando funções matemáticas elementares

A Tabela 2.1 apresenta uma listagem das principais funções matemáticas que o MATLAB possui. Vale ressaltar que o MATLAB trabalha apenas com radianos (2π radianos = 360^o).

Tabela 2.1: Principais funções matemáticas utilizadas pelo MATLAB.

Função	*Significado*
`acos(x)`	Arco coseno.
`acosh(x)`	Arco coseno hiperbólico.
`asin(x)`	Arco seno.
`asinh(x)`	Arco seno hiperbólico.
`atan(x)`	Arco tangente.
`atanh(x)`	Arco tangente hiperbólico.
`cos(x)`	Coseno
`cosh(x)`	Coseno hiperbólico.
`exp(x)`	Exponencial: e^x
`gcd(x,y)`	Máximo divisor comum entre os inteiros x e y.
`log(x)`	Logaritmo natural.
`log10(x)`	Logaritmo na base 10.
`rem(x,y)`	Resto da divisão de x por y.
`round(x)`	Arredondamento para o número inteiro mais próximo.
`sign(x)`	Função sinal. Retorna o sinal do argumento x.
`sin(x)`	Seno
`sinh(x)`	Seno hiperbólico.
`sqrt(x)`	Raiz quadrada.
`tan(x)`	Tangente
`tanh(x)`	Tangente hiperbólica.

3. Trabalhando com vetores e matrizes

O MATLAB foi desenvolvido especialmente para trabalhar com representações matriciais. Desta forma, o usuário deve dar preferência para este tipo de representação quanto estiver utilizando o MATLAB, já que isto significa a realização de cálculos com

maior eficiência.

O MATLAB manipula vetores de uma maneira simples e intuitiva. Considere que se deseja calcular a função y = sen(x) em $0 \leq x \leq \pi$. O primeiro passo é criar um vetor com todos os valores de x para os quais se deseja calcular y. Uma vez definido o vetor, calcula-se os valores correspondentes de y. Ou seja:

```
» x=[0 0.1*pi 0.2*pi 0.3*pi 0.4*pi 0.5*pi 0.6*pi 0.7*pi 0.8*pi 0.9*pi pi]
x =
  Columns 1 through 7
         0    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850
  Columns 8 through 11
    2.1991    2.5133    2.8274    3.1416
» y=sin(x)
y =
  Columns 1 through 7
         0    0.3090    0.5878    0.8090    0.9511    1.0000    0.9511
  Columns 8 through 11
    0.8090    0.5878    0.3090    0.0000
»
```

Para se resgatar um determinado elemento do vetor, basta indicar entre parênteses a localização do mesmo. Ou seja:

```
» x=[0 0.1*pi 0.2*pi 0.3*pi 0.4*pi 0.5*pi 0.6*pi 0.7*pi 0.8*pi 0.9*pi pi];
» x(1)
ans =
     0
» x(11)
ans =
    3.1416
»
```

Para ter acesso a blocos de componentes ao mesmo tempo, o MATLAB utiliza a notação de dois pontos. Ou seja:

- componentes de x, do primeiro ao quinto elemento:

```
» x(1:5)
ans =
         0    0.3142    0.6283    0.9425    1.2566
»
```

- componentes de x, iniciando do sétimo e indo até o final:

```
» x(7:end)
ans =
    1.8850    2.1991    2.5133    2.8274    3.1416
»
```

- componentes de x, iniciando do terceiro, contanto regressivamente de um em um e

parando no primeiro:

```
» x(3:-1:1)
ans =
    0.6283    0.3142         0
»
```

3.1. Construindo vetores

Existem formas para construção de vetores que dispensam a tarefa de digitar termo a termo. São elas:

- cria um vetor que começa em zero e vai até o valor π, incrementado 0,1*π:

```
» x=(0:0.1:1)*pi
x =
  Columns 1 through 7
         0    0.3142    0.6283    0.9425    1.2566    1.5708
1.8850
  Columns 8 through 11
    2.1991    2.5133    2.8274    3.1416
»
```

- cria um vetor que começa em zero e vai até o valor π com 11 elementos utilizando a função `linspace`. Os argumentos desta função são:

  `linspace(primeiro_valor, ultimo_valor, numero_de_valores)`.

```
» linspace(0,pi,11)
ans =
  Columns 1 through 7
         0    0.3142    0.6283    0.9425    1.2566    1.5708
1.8850
  Columns 8 through 11
    2.1991    2.5133    2.8274    3.1416
»
```

- cria um vetor em escala logarítmica que começa em 10^0 e vai até o valor 10^2 com 11 elementos utilizando a função `logspace`. Os argumentos desta função são:

  `logspace(primeiro_expoente, ultimo_expoente, numero_de_valores)`.

```
» logspace(0,2,11)
ans =
  Columns 1 through 7
    1.0000    1.5849    2.5119    3.9811    6.3096   10.0000
15.8489
  Columns 8 through 11
   25.1189   39.8107   63.0957  100.0000
»
```

Até agora foram construídos apenas vetores linhas. Todavia, muitas vezes se faz necessária a utilização de vetores colunas.

A maneira mais direta de construção de um vetor coluna é especificando elemento por elemento e separando os valores com ponto e vírgula:

```
» a=[1; 2; 3]
a =
     1
     2
     3
»
```

Uma alternativa a esta proposta é transpor um vetor linha, já especificado, transformando-o em um vetor coluna:

```
» a=1:3
a =
     1     2     3
» b=a'
b =
     1
     2
     3
»
```

Além da forma de transposição vista anteriormente (‘), o MATLAB tem o recurso de transposição pontuada (.’). Este comando é interpretado como a transposição sem a operação de conjugação complexa. Isto porque quando um vetor é complexo, o operador de transposição (‘) nos dá a transposição do complexo conjugado, isto é, o sinal da parte imaginária é mudado como parte da operação de transposição. Já o operador de transposição pontuada (.’) transpõe o vetor mas não o conjuga. Vale ressaltar que para vetores reais estes operadores são equivalentes. Ou seja:

```
>> a=(-4)^.5; b=[a a a]
b =
  0.00 + 2.00i  0.00 + 2.00i  0.00 + 2.00i
>> b=[a a a]'
b =
  0.00 - 2.00i
  0.00 - 2.00i
  0.00 - 2.00i
>> b=[a a a].'
b =
  0.00 + 2.00i
  0.00 + 2.00i
  0.00 + 2.00i
>>
```

3.3. Construindo matrizes

Para a construção de matrizes no MATLAB utilizam-se ponto e vírgula para separar os elementos de uma linha da outra:

```
» g=[1 2 3 4; 5 6 7 8]
g =
     1     2     3     4
     5     6     7     8
»
```

3.4. Manipulando vetores e matrizes

Considerando-se vetores ou matrizes, a adição, a subtração, a multiplicação e a divisão por um escalar simplesmente aplica a operação a todos os elementos do vetor. Ou seja:

```
» g=[1 2 3 4; 5 6 7 8];
» 2*g-1
ans =
     1     3     5     7
     9    11    13    15
» g/2+1
ans =

    1.5000    2.0000    2.5000    3.0000
    3.5000    4.0000    4.5000    5.0000
»
```

Já as operações entre vetores e/ou matrizes não são tão simples. Quando dois vetores ou matrizes possuem a mesma dimensão, a adição e a subtração são realizadas elemento a elemento pelo MATLAB. Ou seja:

```
» g=[1 2 3 4; 5 6 7 8];
» h=[1 1 1 1; 2 2 2 2];
» g+h
ans =
     2     3     4     5
     7     8     9    10
» g-h
ans =
     0     1     2     3
     3     4     5     6
»
```

Quando se deseja multiplicar duas matrizes, elemento por elemento, deve-se utilizar o símbolo de multiplicação escalar pontuada (`.*`). O ponto que precede o

asterisco, símbolo padrão de multiplicação, diz ao MATLAB para fazer a multiplicação elemento por elemento. A multiplicação sem o ponto significa multiplicação matricial. Ou seja:

```
» g=[1 2; 5 6];
» h=[1 1; 2 2];
» g.*h
ans =
     1     2
    10    12
» g*h
ans =
     5     5
    17    17
»
```

Para a divisão de matrizes, elemento por elemento, deve-se utilizar o símbolo de divisão escalar pontuada `./`. Novamente o ponto que precede o símbolo padrão de divisão diz ao MATLAB para fazer a divisão elemento por elemento. A divisão sem o ponto significa divisão matricial. Ou seja:

```
» g=[1 2; 5 6];
» h=[1 8; 2 4];
» g./h
ans =
    1.0000    0.2500
    2.5000    1.5000
» g/h
ans =

         0    0.5000
   -0.6667    2.8333
»
```

É possível elevar cada elemento de uma matriz a uma dada potência. Para isto aplica-se o operador `.^n`, onde n é potência que se deseja aplicar a cada elemento da matriz. Ou seja:

```
» g=[1 2; 5 6];
» g.^2
ans =
     1     4
    25    36
»
```

A Tabela 3.4.1 fornece ainda a listagem de alguns comandos úteis de manipulação matricial.

Tabela 3.4.1: Comandos úteis de manipulação matricial utilizados pelo MATLAB.

Função	*Significado*	*Exemplo*
	Considere em todos os exemplos: $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$; r =1.	

`A(r,:)`	Fornece a submatriz de A cujas linhas são definidas pelo vetor r e que inclui todas as colunas.	`» A(r,:)`
		`ans =`
		`1 2`

`A(:,r)`	Fornece a submatriz de A cujas colunas são definidas pelo vetor r e que inclui todas as linhas.	`» A(:,r)`
		`ans =`
		`1`
		`3`

`A(:)`	Fornece todos os elementos de A em um vetor coluna, percorrendo as colunas de A pela ordem crescente de seus índices	`» A(:)`
		`ans =`
		`1`
		`3`
		`2`
		`4`

3.5. Comparando vetores e matrizes

Os comandos mais utilizados para comparação entre vetores e matrizes são listados na Tabela 3.5.1.

Tabela 3.5.1: Comandos úteis de comparação entre vetores e matrizes.

Função	*Significado*	*Exemplo*
	Considere em todos os exemplos: $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$; $B = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix}$; $C = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	

`isequal(A,B)`	Variável lógica: verdadeira se A e B são idênticos.	`» isequal(A,B)`
		`ans =`
		`0`
		`» isequal(A,C)`
		`ans =`
		`1`
		`» `

`ismember(A,B)`	Variável lógica: verdadeira quando os elementos de A são também elementos de B.	`» ismember(A,B)`
		`ans =`
		`1 1`
		`0 0`
		`» ismember(A,C)`
		`ans =`
		`1 1`
		`1 1`
		`» `

3.6. Realizando operações matriciais

A Tabela 3.6.1 fornece as principais funções matriciais existentes no MATLAB.

Tabela 3.6.1: Principais funções matriciais existentes no MATLAB.

Função	*Significado*	*Exemplo*
	Considere em todos os exemplos: $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	

`det(A)`	Calcula o determinante da matriz A.	`» det(A)`
		`ans =`
		`-2`

`d=eig(A)`
`[V,D]=eig(A)`
	Determina os autovalores e autovetores de A.	`» d=eig(A)`
		`d =`
		`-0.3723`
		`5.3723`
		`» [V,D]=eig(A)`
		`V =`
		`-0.8246 -0.4160`
		`0.5658 -0.9094`
		`D =`
		`-0.3723 0`
		`0 5.3723`
		»

`inv(A)`	Calcula a matriz inversa da matriz A.	`» inv(A)`
		`ans =`
		`-2.0000 1.0000`
		`1.5000 -0.5000`
		»

`poly(A)`	Calcula a equação característica de A.	`» poly(A)`
		`ans =`
		`1.0000 -5.0000 -2.0000`
		»

`rank(A)`	Determina o número de linhas e colunas linearmente independentes de A.	`» rank(A)`
		`ans =`
		`2`
		»

`svd(A)`	Calcula a decomposição em valores singulares.	`» svd(A)`
		`ans =`
		`5.4650`
		`0.3660`
		»

3.7. Utilizando matrizes especiais

A Tabela 3.7.1 fornece algumas matrizes especiais existentes no MATLAB.

Tabela 3.7.1: Matrizes especiais existentes no MATLAB.

Função	*Significado*	*Exemplo*

`eye`	Matriz identidade.	`» eye(3)`
		`ans =`
		`1 0 0`
		`0 1 0`
		`0 0 1`
		`»`

`ones`	Matriz onde todos os elementos são iguais a 1.	`» ones(2)`
		`ans =`
		`1 1`
		`1 1`
		`»`

`rand`	Matriz com elementos aleatórios distribuídos entre 0 e 1.	`» rand(2)`
		`ans =`
		`0.9501 0.6068`
		`0.2311 0.4860`
		`»`

`randn`	Matriz com elementos aleatórios distribuídos que seguem a distribuição normal e têm média zero e variância igual a 1.	`» rand(3)`
		`ans =`
		`0.8913 0.0185 0.6154`
		`0.7621 0.8214 0.7919`
		`0.4565 0.4447 0.9218`
		`»`

`zeros`	Matriz onde todos os elementos são iguais a 0.	`» zeros(2)`
		`ans =`
		`0 0`
		`0 0`
		`»`

3.8. Ordenando matrizes

A ordenação dos elementos de um vetor ou de uma matriz podem ser realizados utilizando o comando "`sort`". A utilização deste comando possibilita ainda o armazenamento da localização original dos dados. Assim:

```
>> a=rand(1,3)
a =
   0.04389532534714   0.02718512299667   0.31268504808015
>> sort(a)
ans =
   0.02718512299667   0.04389532534714   0.31268504808015

>> a=rand(4,3)
a =
   0.01286257467300   0.03533832396916   0.01635493355000
   0.38396728849430   0.61239548137302   0.19007458907973
   0.68311596780460   0.60854036122399   0.58691847188467
   0.09284246174092   0.01575981791975   0.05758108987829

>> [a_ordenado_l,ord]=sort(a(1,:)), % ordena a linha 1 de "a"
a_ordenado_l =
   0.01286257467300   0.01635493355000   0.03533832396916
ord =
     1     3     2

>> [a_ordenado_c,ord]=sort(a(:,2)), % ordena a coluna 2 de "a"
a_ordenado_c =
   0.01575981791975
   0.03533832396916
   0.60854036122399
   0.61239548137302
ord =

     4
     1
     3
     2
```

3.9. Utilizando matrizes multidimensionais

Para a utilização de matrizes multidimensionais são necessários 3 índices ao invés dos 2 adotados durante a utilização de matrizes bidimensionais. A Figura 3.8.1 mostra a forma visual de interpretar matrizes multidimensionais.

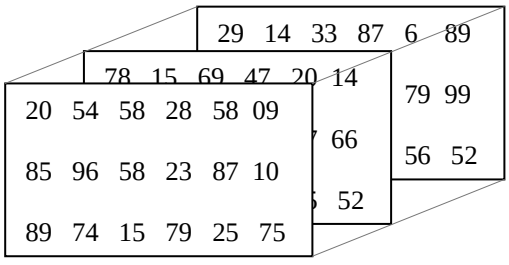

Figura 3.8.1: Forma visual de interpretar matrizes multidimensionais.

As matrizes multidimensionais podem ser construídas e manipuladas utilizando os mesmos comandos apresentados para as matrizes bidimensionais. Desta forma:

```
>> M = rand(2,4,3)
M(:,:,1) =
   0.49655244970310   0.82162916073534   0.81797434083925   0.34197061827022
   0.89976917516961   0.64491038419384   0.66022755644160   0.28972589585624
M(:,:,2) =
   0.34119356941488   0.72711321692968   0.83849604493808   0.37041355663212
   0.53407901762660   0.30929015979096   0.56807246100778   0.70273991324038
M(:,:,3) =
   0.54657115182911   0.69456724042555   0.79482108020093   0.52259034908071
   0.44488020467291   0.62131013079541   0.95684344844488   0.88014220741133
>> M(1,1,1)
ans =
   0.49655244970310
```

3.10. Utilizando listas

As listas (ou disposição em células, *cell arrays*) são formas especiais de representar de matrizes. Neste tipo de representação, cada elemento da matriz pode conter matrizes com diferentes dimensões. O exemplo abaixo ilustra a criação de uma lista.

```
>> a=1:3; b=rand(2,2); c=1; d={a c};
>> lista={a b c d}
lista =
    [1x3 double]    [2x2 double]    [1]    {1x2 cell}
>>
```

Para a manipulação dos elementos de uma lista também são utilizadas chaves e parênteses como exemplificado abaixo.

```
>> a=1:3; b=rand(2,2); c=1; d={a c};
>> lista={a b c d};
>> lista{1}
ans =
     1     2     3
>> lista{1}(1,2)
ans =
     2
>>
```

3.11. Utilizando estruturas

As estruturas são matrizes especiais utilizadas pelo MATLAB para armazenar dados de naturezas diferentes. As estruturas diferem das listas por possuírem nomes que identificam a localização dos dados. O exemplo abaixo evidencia a utilização de

estruturas.

```
>> resultado.alunos='maria';
>> resultado.notas=[9 8.5];
>> resultado
resultado =
    alunos: 'maria'
     notas: [9 8.50000000000000]
>> resultado(2).alunos='joao';
>> resultado(2).notas=[10 7];
>> resultado
resultado =
1x2 struct array with fields:
    alunos
    notas
>> resultado.alunos
ans =
maria
ans =
joao
>>
```

Uma outra forma de gerar estruturas é utilizando o comando ‘`struct`’.

```
>> resultado=struct('alunos','maria','notas',[9 8.5]);
>> resultado
resultado =
    alunos: 'maria'
     notas: [9 8.50000000000000]
>> resultado.notas
ans =
   9.00000000000000   8.50000000000000
>>
```

3.12. Utilizando matrizes esparsas

As matrizes esparsas são aquelas onde apenas alguns de seus elementos possuem valores diferentes de zero. Neste caso, o armazenamento de toda a matriz representa um desperdício de espaço de armazenagem e de poder computacional em operações aritméticas com zeros. No MATLAB é possível considerar a esparticidade de uma matriz utilizando os comandos “`sparse`” e “`full`”.

O comando “`sparse`” armazena os elementos não nulos da matriz original, desconsiderando os elementos iguais a zero. Já o comando “`full`” rescreve a matriz esparsa original. Ou seja:

```
>> clear all
>> X=[1 0 5 6 0 0 0; 0 0 0 0 1 0 0]
X =
     1     0     5     6     0     0     0
     0     0     0     0     1     0     0
>> S = sparse(X)
S =
   (1,1)        1
   (1,3)        5
   (1,4)        6
   (2,5)        1

>> whos
  Name      Size         Bytes  Class
  S         2x7             80  sparse array
  X         2x7            112  double array
Grand total is 18 elements using 192 bytes

>> X2=full(S)
X2 =
     1     0     5     6     0     0     0
     0     0     0     0     1     0     0

>> whos
  Name      Size         Bytes  Class
  S         2x7             80  sparse array
  X         2x7            112  double array
  X2        2x7            112  double array
Grand total is 32 elements using 304 bytes
>>
```

4. Analisando dados

A análise de dados no MATLAB é feita utilizando-se matrizes orientadas por coluna. As diversas variáveis são armazenadas em diferentes colunas e cada linha representa uma observação diferente de cada variável. As principais funções de análise de dados são apresentadas na Tabela 3.11.1

Tabela 3.11.1: Principais funções de análise de dados.

Função	*Significado*	*Exemplo*
	Considere em todos os exemplos: $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	

`corrcoef(A)`	Coeficientes de correlação	`>> corrcoef(A)`
		`ans =`
		`1 1 1`
		`1 1 1`
		`1 1 1`
		`»`

`cumprod(A)`	Produto acumulativo das colunas de A.	`» cumprod(A)`
		`ans =`
		`1 2 3`
		`4 10 18`
		`28 80 162`
		`»`

`cumsum(A)`	Soma acumulativa ao longo das colunas de A.	`» cumsum(A)`
		`ans =`
		`1 2 3`
		`5 7 9`
		`12 15 18`
		`»`

`max(A)`	Máximo ao longo das colunas de A.	`» max(A)`
		`ans =`
		`7 8 9`
		`»`

`mean(A)`	Média ao longo das colunas de A.	`» mean(A)`
		`ans =`
		`4 5 6`
		`»`

`min(A)`	Mínimo ao longo das colunas de A.	`» min(A)`
		`ans =`
		`1 2 3`
		`»`

`prod(A)`	Produto ao longo das colunas de A.	`» prod(A)`
		`ans =`
		`28 80 162`
		`»`

`std(A)`	Calcula o desvio padrão ao longo das colunas de A.	`» std(A)`
		`ans =`
		`3 3 3`
		`»`

`sum(A)`	Soma os elementos ao longo das colunas de A.	`» sum(A)`
		`ans =`
		`12 15 18`
		`»`

5. Trabalhando com polinômios

No MATLAB, um polinômio é representado por um vetor linha contendo seus coeficientes em ordem decrescente. Por exemplo, o polinômio $x^4 – 12.x^3 + 25.x + 116$ é representado da seguinte forma:

```
» p=[1 -12 0 25 116]
p =
     1   -12     0    25   116
»
```

Vale ressaltar que os termos com coeficientes iguais a zero devem ser incluídos.

Através do uso do comando `roots` é possível encontrar as raízes de um dado polinômio. Ou seja:

```
» p=[1 -12 0 25 116];
» r=roots(p)
r =
  11.7473
   2.7028
  -1.2251 + 1.4672i
  -1.2251 - 1.4672i
»
```

Dadas as raízes do polinômio, também é possível construir o polinômio associado. Para isto, é utilizado o comando `poly`:

```
» pp=poly(r)
pp =
    1.0000  -12.0000         0   25.0000  116.0000
»
```

A multiplicação de polinômios é realizada pelo comando "`conv`". Desta forma, considerando o produto de dois polinômios $f_1(x) = x^3 + 2.x^2 + 3.x + 4$ e $f_2(x) = x^3 + 4.x^2 + 9.x + 16$ temos:

```
» f1=[1 2 3 4]; f2=[1 4 9 16];
» f3=conv(f1,f2)
f3 =
     1     6    20    50    75    84    64
»
```

O resultado desta operação é o polinômio $f_3(x) = x^6 + 6.x^5 + 20.x^4 + 50.x^3 + 75.x^2 + 84.x + 64$.

A função "`deconv`" é utilizada para dividir um polinômio por outro. Considerando-se as funções $f_3(x)$ e $f_2(x)$ anteriores temos:

```
» [q,r]=deconv(f3,f2)
q =
     1     2     3     4
r =
     0     0     0     0     0     0     0
»
```

Como resultado, o comando retorna o polinômio resultante q, que neste caso é a função $f_1(x)$, e o resto da divisão r.

O MATLAB possui a função "`polyder`" que é utilizada na obtenção de derivadas de polinômios. Ou seja:

```
» f1=[1 2 3 4];
» d=polyder(f1)
d =
     3     4     3
»
```

É possível que seja necessário utilizar razões entre polinômios (funções de transferência por exemplo). Nestes casos, o MATLAB considera a existência de 2 polinômios distintos.

6. Confeccionando gráficos

6.1. Gráficos bidimensionais

Uma das funções que o MATLAB possui para elaboração de gráficos é a função "`fplot`". Este comando calcula a função a ser representada e certifica-se de que suas propriedades estejam bem representadas. Como entrada, o "`fplot`" precisa saber a função a representada (como variável *string*) e o domínio do gráfico. Ou seja:

```
» f='2*exp(-x).*sin(x)';
» fplot(f,[0 8])
»
```

Outra função utilizada para confeccionar gráficos bidimensionais, onde f = f(x), é a função "`ezplot`". Esta função também tem como argumentos de entrada uma função *string* e um intervalo de variação. Se f = f(x,y), o comando "`ezplot`" representa a função considerando f(x,y)=0.

```
>> ezplot('x^3 + y^3 - 5*x*y + 1/5',[-3,3])
```

O comando mais comum utilizado para elaboração de gráficos bidimensionais no MATLAB é o comando "`plot`". Esse comando cria gráficos de vetores de dados em eixos adequados e conecta os pontos a linhas retas. Por exemplo:

```
» x=linspace(0, 2*pi, 30);
» y=sin(x);
» plot(x,y)
»
```

É possível utilizar o comando "`plot`" para traçar mais de um gráfico no mesmo sistema de eixos. Ou seja:

```
» x=linspace(0, 2*pi, 30);
» y=sin(x);
» z=cos(x);
» plot(x,y,x,z)
»
```

Durante a confecção de um gráfico no MATLAB , o usuário pode escolher a cor e o estilo das linhas bem como o marcador utilizado. A Tabela 6.1 mostra os códigos utilizados e o exemplo abixo evidencia a utilização dos mesmos.

```
» x=linspace(0, 2*pi, 30);
» y=sin(x);
» z=cos(x);
» plot(x,y,'b:p',x,z,'m+--')
»
```

O usuário pode ainda modificar a cor de fundo do gráfico utilizando o comando "`colordef`". Neste caso, sugere-se ao usuário consultar o *help* na área de trabalho do MATLAB (» `help colordef`).

O comando "`grid on`" adiciona linhas de grade ao gráfico nas posições dos eixos que há marcadores. O comando "`grid off`" remove as linhas de grade. Vale ressaltar que o MATLAB começa sempre com "`grid off`" Para gráficos bidimensionais.

Tabela 6.1: Códigos para marcadores, cores e tipos de linha na confecção de gráficos no MATLAB.

Cores de linhas		*Marcadores*		*Tipo de linha*	
símbolo	cor	Símbolo	Marcador	Símbolo	Tipo de linha
b	azul	.	ponto	-	linha contínua
g	verde	o	círculo	:	linha pontilhada
r	vermelho	x	x	-.	traços e pontos
c	ciano	+	+	--	linha tracejada
m	magenta	*	estrela		
y	amarelo	s	quadrado		
k	preto	d	losango		
w	branco	<	triângulo para a esquerda		
		>	triângulo para a direita		
		p	pentagrama		
		h	hexagrama		

Para atribuir nomes aos eixos, pode ser utilizado os comandos comando "`xlabel`" e "`ylabel`". O comando "`title`" adiciona um título ao gráfico. Ou seja:

```
» x=linspace(0, 2*pi, 30);
» y=sin(x);
» plot(x,y,'r-.>')
» grid on
» xlabel('x')
» ylabel('seno(x)')
» title('gráfico')
» grid off
»
```

Para criar legendas no gráfico, o usuário pode utilizar os comandos "`legend`" ou "`gtext`". Ou seja:

```
» x=linspace(0, 2*pi, 30);
» y=sin(x);
» z=cos(x);
» plot(x,y,x,z)
» legend('seno(x)','coseno(x)')
» legend off % retira a legenda
» gtext('seno(x)')
» gtext('coseno(x)')
»
```

É possível criar uma janela com mais de um sistema de eixos. Para isto, é utilizado o comando "`subplot(m,n,p)` ". Este comando subdivide a janela de gráficos em uma matriz com `m` por `n` regiões. A variável `p` indica a localização do gráfico. Ou seja:

```
» x=linspace(0, 2*pi, 30);
» y=sin(x);
» z=cos(x);
» a=2*sin(x).*cos(x);
» b=sin(x)./(cos(x)+eps);
» subplot(2,2,1)
» plot(x,y)
» axis([0 2*pi -1 1]) % define os valores: mínimos e máximos para x e y
» title('seno(x)')
» subplot(2,2,2)
» plot(x,z), axis([0 2*pi -1 1]), title('coseno(x)')
» subplot(2,2,3)
» plot(x,a), axis([0 2*pi -1 1]), title('2.seno(x).coseno(x)')
» subplot(2,2,4)
» plot(x,b), axis([0 2*pi -20 20]), title('seno(x)/coseno(x)')
»
```

Além dos gráficos já elaborados, outros recursos gráficos bidimensionais são disponíveis no MATLAB. São eles:

- gráfico tipo torta:

```
» a=[.05 .15 .5 .05 .05 .2];  pie(a)
» pie(a, a==max(a)) % traça o gráfico separando a maior fatia
»
```

- gráficos com escalas diferentes:

```
» x=-2*pi:pi/10:2*pi;
» y=sin(x); z=2*cos(x);
» subplot(2,1,1),plot(x,y,x,z), title('escalas iguais'),
» subplot(2,1,2),plotyy(x,y,x,z), title('escalas diferentes'),
»
```

- gráficos de barras:

```
» x=-2.9:0.2:2.9;
» y=exp(-x.*x);
» subplot(2,2,1), bar(x,y),
» subplot(2,2,2), bar3(x,y),
» subplot(2,2,3), stairs(x,y),
» subplot(2,2,4), barh(x,y),
»
```

- gráfico com barras de erros:

```
» x=linspace(0,2,21);
» y=erf(x); % y é a função erro de x
» e=rand(size(x))/10; % contém  valores de erros aleatórios
» errorbar(x,y,e)
»
```

- gráficos com pontos selecionados manualmente:

```
» x=linspace(-2*pi, 2*pi,60);
» y=sin(x).^2./(x+eps);
» plot(x,y)
» [a,b]=ginput(5) % toma 5 pontos
» hold on % Para escrever por cima do gráfico anterior
» plot(a,b,'mo')
» hold off % Libera o gráfico anterior para ser eliminado
»
```

6.2. Gráficos tridimensionais

O comando "plot" para gráficos bidimensionais é estendido para os tridimensionais com o comando "plot3". Este comando cria gráficos de linhas tridimensionais.

```
» t=linspace(0,10*pi,1000);
» plot3(sin(t),cos(t),t)
»
```

O comando "meshgrid" é utilizado para gerar matrizes x e y, contendo linhas e colunas repetidas. Esse par de matrizes , x e y, pode ser então usado para calcular funções de duas variáveis usando os recursos do MATLAB de matemática vetorial.

Uma vez estabelecidos os valores dos pontos a serem representados em um gráfico tridimensional, o comando "mesh" pode ser utilizado para gerar um gráfico de rede e o comando "surf" pode ser utilizado para gerar uma superfície. Já os comandos "contour" e "contour3" geram as curvas de nível. Uma forma similar do comando "contour" é o comando "pcolor" onde são utilizadas cores distintas para delimitar regiões de alturas diferentes. Ou seja:

```
>> x=-3:0.1:3; y=x; [X,Y]=meshgrid(x,y); Z=X.^3 + Y.^3 - 5*X.*Y + 1/5;
>> mesh(X,Y,Z)
>> surf(X,Y,Z)
>> contour(X,Y,Z,40)
>> contour3(X,Y,Z,40)
>> pcolor(X,Y,Z)
>>
```

É possível para o usuário trocar as cores padrão utilizadas pelo MATLAB na confecção de gráficos. Para isto, utiliza-se dos mapas de cores que são matrizes com três colunas. Cada linha define uma cor particular, usando os números 0 a 1.

```
>> x=-3:0.1:3; y=x; [X,Y]=meshgrid(x,y); Z=X.^3 + Y.^3 - 5*X.*Y + 1/5;
>> surf(X,Y,Z), colormap([1 1 1]) % branco
>> surf(X,Y,Z), colormap([.5 .5 .5]) % cinza
>>
```

O usuário pode ainda escolher um dos mapas de cores já existente no MATLAB (Tabela 6.2.1).

Tabela 6.2.1: Mapas de cores utilizadas pelo MATLAB.

Função	*Descrição do mapa de cores*
`Hsv`	Escalar com cores saturadas.
`Hot`	Preto-vermelho-amarelo-branco
`Gray`	Escalar linear de tons de cinza.
`Bone`	Escala de tons de cinza levemente azulados.
`Copper`	Escala linear de tons acobreados.
`Pink`	Tons pastéis de rosa.
`White`	Mapa de cores totalmente branco.
`Flag`	Vermelho, branco, azul e preto alternados.
`Jet`	Uma variante do mapa `hsv`
`Prism`	Mapa de cores denominado “prisma”.
`Cool`	Tons de ciano e magenta.
`lines`	Mapa de cores que usa as mesmas cores do comando `plot`.
`colorcube`	Mapa de cores denominado “cubo colorido”.
`summer`	Tons de amarelo e verde.
`autumn`	Tons de vermelho e amarelo.
`winter`	Tons de azul e verde.
`spring`	Tons de magenta e amarelo.

```
>> x=-3:0.1:3; y=x; [X,Y]=meshgrid(x,y); Z=X.^3 + Y.^3 - 5*X.*Y + 1/5;
>> surf(X,Y,Z), colormap([summer])
>> surf(X,Y,Z), colormap([hot])
>>
```

Para acrescentar uma legenda no gráfico tridimensional, o usuário deve utilizar o comando `colorbar`.

```
>> x=-3:0.1:3; y=x; [X,Y]=meshgrid(x,y); Z=X.^3 + Y.^3 - 5*X.*Y + 1/5;
>> surf(X,Y,Z), colormap([winter]), colorbar
>>
```

7. Trabalhando com tempo

O MATLAB possui várias funções para manipular datas e horas. A função “`clock`“ , por exemplo, fornece a data e a hora atuais em um vetor.

```
>> T=clock
T =
  1.0e+003 *
   2.003   0.001   0.009   0.010   0.031   0.057
>>
```

Os elementos que retornam no vetor T ao se aplicar o comando “`clock`“ são: ano,

mês, dia do mês, hora, minutos e segundos. Já a função "`date`" fornece a data atual como um texto no formado dia-mês-ano. Ou seja:

```
>> date
ans =
09-Jan-2003
```

Os comandos "`tic`" e "`toc`" podem ser utilizados para cronometrar uma operação. Ou seja:

```
>> tic; plot(rand(5)); toc
elapsed_time =
   0.11000000000000
```

A função "`cputime`" fornece o tempo, em segundos, da Unidade Central de Processamento (CPU) usado pelo MATLAB desde o início da sessão corrente. Ou seja:

```
>> to=cputime; plot(rand(5)); cputime-to
ans =
   0.11000000000058
```

8. Obtendo modelos empíricos

O MATLAB possui funções que possibilitam a obtenção de diferentes modelos empíricos tais como: modelos ARX, ARMAX, redes neuronais e ainda modelos obtidos via regressão linear e não linear. Nesta apostila, a utilização da regressão linear é exemplificada.

8.1. Regressão linear

O comando utilizado pelo MATLAB para a realização de regressão linear é denominado "`regress`". Para utilizar este comando o usuário deve fornecer a matriz com as variáveis independes e um vetor com a variável dependente. Cada linha da matriz ou do vetor deve corresponder a uma observação. Caso o modelo linear possua uma constante, a matriz com as variáveis independentes deve possuir uma última coluna com valores iguais a 1. O exemplo abaixo exemplifica a utilização deste comando:

```
>> clear all
>> x=[(100:200)'+randn(101,1) (300:400)'+randn(101,1) ones(101,1)];
>> % matriz com variaveis independentes
>> P1=2; P2=1; P3=100; y=P1*x(:,1)+P2*x(:,2)+P3; y=y+randn(101,1);
>> % vetor com variaveis dependentes
>> [P,Pin,R,Rin,stat]=regress(y,x,0.05);
>> P % parametros do modelo
P =
   1.93344230551157
   1.06793100944515
  86.20341131968449
>> Pin % intervalo de confiança para os parametros
Pin =
  1.0e+002 *
   0.01813296312069   0.02053588298954
   0.00947622221814   0.01188239797076
   0.62069772941691   1.10337049697678
>> stat
stat =
  1.0e+005 *
   0.00000999899004   4.85119348155784                  0
>>
```

Os intervalos de confiança para os parâmetros do modelo foram calculados considerando um nível de confiança igual a 95%. Isto porque usamos o valor 0,05 como terceiro argumento de entrada para a função "`regress`".

O 5º argumento de saída, neste caso denominado "`stat`", fornece respectivamente: o valor de R^2, o resultado do teste de hipótese F (testa-se se todos os coeficientes do modelo obtido são iguais a zero) e o p-valor associado a este teste. Durante a realização do teste F assume-se a existência de uma constante no modelo.

Neste caso, o modelo obtido consegue descrever cerca de 99,9899% da variabilidade experimental e podemos considerar, com grande segurança, que todos os coeficientes do modelo não são iguais a zero.

Os erros verificados entre os valores preditos pelo modelo e os reais, bem como o intervalo de confiança para estes valores são fornecidos através do 3º e do 4º argumento de saída da função "`regress`". Neste caso: "R" e "`Rin`". Uma melhor visualização desta informação pode ser obtida através do comando "`rcoplot`" como exemplificado abaixo.

```
>> rcoplot(R, Rin)
```

Neste caso, todos os resíduos que não atingirem o valor zero são considerados *outliers*.

O MATLAB também possui funções destinadas à realização de regressões não lineares. Neste caso, a soma dos quadrados dos resíduos, verificados entre os valores reais e preditos pelo modelo, é minimizada utilizando um método de otimização Quasi Newton (Levenberg-Marquardt). Para maiores detalhes sugere-se a consulta ao comando "`nlinfit`". ("`help nlinfit`").

9. Iniciando um programa

Todos os comandos descritos anteriormente podem ser utilizados durante a elaboração de um programa em MATLAB.

Antes de começar a programar, você deve escolher qual o diretório de trabalho.

A programação em MATLAB é realizada através da elaboração de arquivos tipo "m". Sendo assim, uma vez escolhido o diretório de trabalho, você deve abrir um arquivo "m" seguindo o seguinte caminho: *File*→ *New* → *M-file*. Neste arquivo deve ser escrito o programa.

Uma vez escrita a rotina a ser executada, deve-se salvar o arquivo. Para "rodar" a rotina, deve-se digitar o nome do arquivo tipo "m" no *workspace* do MATLAB.

O MATLAB possui diversas funções que são particularmente apropriadas para o uso em arquivos tipo "m". Estas funções são apresentadas na Tabela 9.1.

Tabela 9.1: Principais funções utilizadas em arquivos tipo "m".

Variável	*Significado*
`Disp('texto')`	Mostra o texto escrito entre as aspas.
`input`	Solicita ao usuário que forneça algum dado de entrada.
`pause`	Suspende a execução até que o usuário pressione alguma tecla.
`Pause(n)`	Suspende a execução por n segundos.

Exemplo:

1. Abra um arquivo novo tipo "m";

2. Digite:

```
% Primeiro programa
clear all  %limpa toda a memória
disp('Rodando programa ...');
n=input('Qual o valor final de x desejado?');
x=0:n;
y=x.^2;
plot(x,y);
pause(2)
xlabel('x'); ylabel ('y');
```

3. Salve o programa com o nome “prog1”;
4. Vá à área de trabalho do MATLAB e digite “prog1”.

A(s) primeira(s) linha(s) comentada(s) existente(s) no arquivo “.m” é(são) exibida(s) caso o usuário utilize o comando *help + nome do arquivo*. Ou seja:

```
>> help prog1
  Primeiro programa
>>
```

10. Utilizando comandos de fluxo e operadores lógicos

10.1. Utilizando a função `for`

Os *loops* `for` possibilitam que uma série de comandos seja repetida por um número de vezes fixo e predefinido. Vale ressaltar que o comando `for` não pode ser encerrado atribuindo-se valores ao contador (no exemplo ‘n’) dentro do *loop*.

Exemplo:

1. Abra um arquivo novo tipo “m”;
2. Digite:

```
clear all
for n=1:10
   x(n)=n/2;
   n=10; % ao final do primeiro cálculo de x n = valor maximo
end
x
```

3. Salve o programa com o nome “prog2”;
4. Vá à área de trabalho do MATLAB e digite “prog2”.
5. A resposta obtida será:

```
» prog2
x =
  Columns 1 through 7
    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000
  Columns 8 through 10
    4.0000    4.5000    5.0000
»
```

Naturalmente, é possível a utilização de mais de uma estrutura `for`.

Exemplo:

1. Abra um arquivo novo tipo “m”;
2. Digite:

```
clear all
for n=1:5
   for m=5:-1:1
      A(n,m)=n^2+m^2;
   end
   disp(n)
end
A
```

3. Salve o programa com o nome “prog3”;
4. Vá à área de trabalho do MATLAB e digite “prog3”.
5. A resposta obtida será:

```
» prog3
     1
     2
     3
     4
     5
A =
     2     5    10    17    26
     5     8    13    20    29
    10    13    18    25    34
    17    20    25    32    41
    26    29    34    41    50
»
```

10.2. Utilizando a função `while`

Os *loops* `while` executam um grupo de comandos um número indefinido de vezes.

Exemplo:

1. Abra um arquivo novo tipo “m”;
2. Digite:

```
clear all
n=1;
while n<10
   x(n)=1;
   n=n+1;
end
x
```

3. Salve o programa com o nome “prog4”;
4. Vá à área de trabalho do MATLAB e digite “prog4”.
5. A resposta obtida será:

```
» prog4
x =
     1     1     1     1     1     1     1     1     1
»
```

10.3. Utilizando a função `if-else-end`

Quando ações devem ser executados condicionalmente, com base em um teste relacional, são utilizados comandos `if-else-end`. Como no comando `for`, é possível utilizar vários comandos `if-else-end` simultaneamente.

Exemplo:

1. Abra um arquivo novo tipo “m”;
2. Digite:

```
clear all
macas=10; custo=macas*10;
if macas>5
   custo=0.8*custo; % desconto de 20%
end
custo
```

3. Salve o programa com o nome “prog5”;
4. Vá à área de trabalho do MATLAB e digite “prog5”.
5. A resposta obtida será:

```
» prog5
custo =
   200
»
```

Exemplo:

1. Abra um arquivo novo tipo “m”;
2. Digite:

```
clear all
m=5
if m==10;
   disp('m é igual a 10');
elseif m<10;
   disp('m é menor que 10');
else m~=10;
   disp('m é diferente de 10');
end
```

3. Salve o programa com o nome “prog6”;
4. Vá à área de trabalho do MATLAB e digite “prog6”.
5. A resposta obtida será:

```
>> prog6
m =
     5
m é menor que 10
>>
```

Observação importante: Como já mencionado anteriormente, o MATLAB é um programa desenvolvido para trabalhar com matrizes. Sendo assim, quando se deseja um *loop* onde o objetivo é calcular uma expressão diversas vezes, é mais eficiente trabalhar utilizando notação matricial.

11. Resolvendo um sistema de equações algébricas

No MATLAB a função utilizada para resolver sistemas de equações algébricas não lineares é denominada “`fsolve`” e se localiza no *toolbox* de otimização. Para maiores detalhes a respeito desta função, sugere-se consultar o *help* digitando “*help fsolve*” na página principal do MATLAB.

O método padrão, utilizado para resolução do sistema pelo comando “fsolve”, é o Gauss-Newton com um método misto (quadrático e cúbico) de busca em linha. Caso o usuário prefira, pode ser utilizado o método de Levenberg-Marquardt em alternativa ao

Gauss-Newton.

Exemplo:

Considere o sistema de equações algébricas não lineares apresentado pela equação 1.11:

$$\begin{aligned} 2 \cdot x_1 - x_2 - e^{-x1} &= 0 \\ -x_1 + 2 \cdot x_2 - e^{-x2} &= 0 \end{aligned} \qquad (11.1)$$

Para resolver este sistema no MATLAB, realize as seguintes tarefas:

1. abra um novo arquivo “m” e escreva:

```
function F=fun1(x)
F(1)=2*x(1)-x(2)-exp(-x(1));
F(2)=-x(1)+2*x(2)-exp(-x(2));
```

2. salve o arquivo como “fun1”;
3. abra um segundo arquivo “m” e escreva:

```
clear all
x0=[-5 -5]; % estimativas iniciais para x(1) e x(2)
options(1)=1; % para mostrar detalhes sobre o cálculo que será realizado
x=fsolve('fun1',x0,options)  % chamada da rotina que resolve o sistema
```

4. salve este segundo arquivo como “teste1”;
5. vá à área de trabalho do MATLAB e digite “teste1”. O programa retornará o seguinte resultado:

```
» teste1
f-COUNT      RESID    STEP-SIZE      GRAD/SD
    3      47071.2            1   -9.41e+004
    8      966.828            1   -1.81e+003
   15      1.99465         3.85          5.6
   20  0.000632051        0.895      -0.0867
   25 1.39647e-015        0.998   -1.89e-009
Optimization Terminated Successfully

x =

    0.5671    0.5671
```

12. Resolvendo um sistema de equações diferenciais

O MATLAB possui diversas funções destinadas à resolução de sistemas de equações diferenciais ordinárias. Nesta apostila utilizaremos a função “ode23”. Para maiores detalhes a respeito destas funções, sugere-se consultar o *help* digitando “*help*

ode23" na área de trabalho do MATLAB.

O método padrão utilizado para resolução do sistema de equações diferenciais pelo comando "ode23" é o Runge Kutta.

Exemplo:

Considere o sistema de equações diferenciais apresentado por Simmons (1988) e descrito pela equação 12.1:

$$\frac{dx}{dt} = x + 2 \cdot y$$
$$\frac{dy}{dt} = 3 \cdot x + 2 \cdot y \qquad (12.1)$$

Assuma as condições iniciais apresentadas pela equação (12.2):

$$t = 0: \qquad x = 2 \qquad y = 3 \qquad (12.2)$$

Para resolver este sistema no MATLAB, realize as seguintes tarefas:

1. abra um novo arquivo "m" e escreva:

```
function [dy]=fun2(t,y)
dy(1)=y(1)+2*y(2);
dy(2)=3*y(1)+2*y(2);
dy=[dy(1);dy(2)];
```

2. salve o arquivo como "fun2";
3. abra um segundo arquivo "m" e escreva:

```
clear all
yo=[2 3];      % condição inicial
tspan=[0 1]; % intervalo no qual será realizada a integração [de 0 à 1]
[t,y]=ode23('fun2',tspan,yo);
plot(t,y);
xlabel('tempo');
ylabel('variáveis x e y');
legend(['x(t)';'y(t)'])
```

4. salve este segundo arquivo como "teste2";
5. vá à área de trabalho do MATLAB e digite "teste2". O programa retornará como resultado um gráfico apresentando a evolução das funções x(t) e y(t) de 0 a 1.

De acordo com Simmons (1988) o sistema apresentado pela equação 2 possui a solução analítica descrita pela equação 12.3. Esta solução confere com a solução numérica apresentada pelo programa.

$$x = 2 \cdot e^{4 \cdot t}$$
$$y = 3 \cdot e^{4 \cdot t} \qquad (12.3)$$

SIMMONS, G. F., *Cálculo com Geometria Analítica*, Vol. 2, 1ª ed. São Paulo,McGraw-Hill, 1988

13. Como saber mais sobre o MATLAB?

O MATLAB possui uma biblioteca com vários arquivos tipo pdf. Cada um destes arquivos diz respeito a um *toolbox* do MATLAB. A seguir é apresentada uma listagem com a indicação de qual arquivo deve ser consultado pelo usuário, se o mesmo necessitar aprofundar seus conhecimentos sobre os tópicos apresentados nesta apostila. Além disso, buscou-se adicionar alguns assuntos de interesse para engenheiros químicos que não são tratados neste material. Parte da literatura citada foi utilizada para a confecção desta apostila.

Tabela 13.1: Arquivos pdf recomendados.

Assunto	*Arquivo pdf*	
Comandos básicos de utilização, descrição do programa	getstart.pdf	The Language of Technical Computing, Getting Started with MATLAB, 136 páginas, 6ª versão, 2000
Tratamento estatístico de dados (regressão linear, ANOVA, regressão não linear, matriz de correlação, planejamento fatorial, PCA)	stats_tb.pdf	Statistics Toolbox, 560 páginas, 3ª versão, 2000
Controle de processos	usingcontrol.pdf	Control System Toolbox, 591 páginas, 1ª versão, 2000
Controle preditivo de processos	mpc.pdf	Model Predictive Control Toolbox, 250 páginas, 1ª versão, 1998
Resolução de sistemas de equações diferenciais parciais	pde.pdf	Partial Differential Equation Toolbox, 284 páginas, 1997
Redes neuronais	nnet.pdf	Neural Networks Toolbox, 846 páginas, 4ª vesão, 2000
Identificação de processos (modelos ARX, ARMAX,)	ident.pdf	System Identification Toolbox, 368 páginas,5ª versão, 2000
Otimização	optim_tb.pdf	Optimization Toolbox, 330 páginas, 2ª versão, 2000
Utilização simultânea do Excel e do MATLAB	exlink.pdf	Excel Link , 74 páginas, versão 1.1.2, 1999

Exercícios

Série A

1) Considere os resultados experimentais apresentados na Tabela A:

Tabela A: Resultados experimentais.

Experimento	*Concentrações: Ca e Cb*
1	*0,5 0,8*
2	*0,4 0,75*

1.1) armazene os dados da Tabela A em uma matriz bidimensional;

1.2) armazene os dados da Tabela A em uma matriz multidimensional.

2) Crie uma matriz de dimensão 4x4 e a chame de A. (Sugestão: crie uma matriz com números aleatórios.)

1.1) apague a 2ª linha de A;

1.2) apague a 3ª coluna de A.

Série B

Represente graficamente as seguintes funções:

1. $z = -x^2 + 2.y^2$ onde $-10 < x < 10$, $-10 < y < 10$
2. $z = -x^2 + 2.y^2 + 5.x.\text{sen}(y)$ onde $-10 < x < 10$, $-10 < y < 10$

Série C

Considere o seguinte problema:

Uma indústria produtora de alumínio deseja conhecer a influência das variáveis razão Al_2O_3/NaOH e temperatura de reação sobre o teor de Na_2O presente na alumina. Utilizando os dados operacionais disponíveis na Tabela C proponha um modelo linear que descreva o processo.

Tabela C: Dados operacionais.

Teor de Na_2O (p/p)	*Al_2O_3/NaOH*	*Temperatura de reação (^oC)*
0,43	0,647	77,1
0,39	0,638	78,3
0,44	0,651	76,0
0,42	0,648	77,9
0,43	0,640	74,1
0,42	0,643	74,6
0,41	0,643	76,0
0,46	0,651	73,3
0,42	0,650	78,6
0,40	0,639	78,7

Este problema é originalmente apresentado por: WERKEMA, M. C. C., AGUIAR, S., *Análise de Regressão: Como entender o relacionamento entre as variáveis de um processo*, 1ª ed. Belo Horizonte, Fundação Christiano Ottoni, 1996. Nesta apostila é utilizada uma versão simplificada do problema original.

Série D

Considere o reator bioquímico descrito na Figura D, onde x_1 representa a biomassa presente no reator e x_2 o substrato. O reator possui um volume constante (V_r) uma vez que as vazões de alimentação e de saída são iguais (F). A corrente de alimentação possui concentração de substrato igual a x_{2i} e uma concentração de biomassa igual a zero.

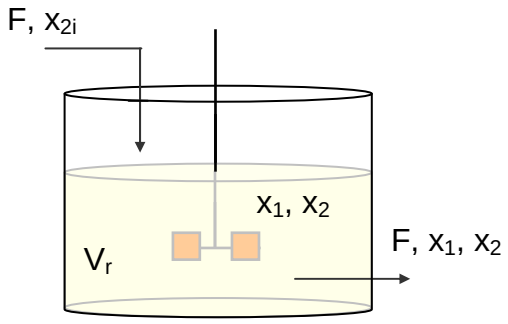

Figura D: Representação esquemática do reator bioquímico.

Bequette (1998) propõe o modelo descrito pela equação (D.1) para o processo,

$$\frac{dx_1}{dt} = (\mu - D) \cdot x_1$$
$$\frac{dx_2}{dt} = D \cdot (x_{2i} - x_2) - \frac{\mu \cdot x_1}{Y} \qquad \text{(D.1)}$$
$$D = \frac{F}{V_r}$$

onde Y é a relação constante entre as taxas de geração de biomassa e o consumo de substrato e μ é o coeficiente de taxa de crescimento descrito pela equação (D.2).

$$\mu = \frac{\mu_{max} \cdot x_2}{k_m + x_2 + k_1 \cdot x_2^2} \qquad \text{(D.2)}$$

Descreva o comportamento das concentrações de biomassa e de substrato considerando um tempo de integração de 30 horas (0<t<30), as condições iniciais descritas pela equação (D.3) e os valores para os parâmetros descritos na Tabela D.

$t = 0: \qquad x_1 = 1 \qquad x_2 = 1$

Tabela D: Valores para os parâmetros do modelo (Bequette, 1998)

Parâmetro	*Valor*
μ_{max}	0,53 h^{-1}
k_m	0,12 g/L
k_1	0,4545 L/g
Y	0,4
x_{2i}	4 g/L
D	0,3 h^{-1}

BEQUETTE, B. W., *Process Dynamics: Modeling, Analysis and Simulation*, 1ª ed. USA, Prentice-Hall PTR, 1998.