Universidade Estadual de Montes Claros
Departamento de Ciências da Computação
Bacharelado em Engenharia de Sistemas

Minicurso

Uma Introdução ao MatLab

Minicurso oferecido na I Escola de Verão em Computação UNIMONTES/IFNMG a ser realizada no período 28 de janeiro à 01 de fevereiro de 2013 nas instalações da UNIMONTES.

por

Camila Katheryne Santos Cangussu
Guilherme Augusto Duque e Silva Costa
Lucas Almeida Aguiar
Orientador: Prof. Rosivaldo Antônio Gonçalves

Janeiro – 2013

Universidade Estadual de Montes Claros
Departamento de Ciências da Computação
Bacharelado em Engenharia de Sistemas

Uma Introdução ao MatLab

Camila Katheryne Santos Cangussu
Guilherme Augusto Duque e Silva Costa
Lucas Almeida Aguiar

Orientador: Prof. Rosivaldo Antônio Gonçalves

Minicurso oferecido na I Escola de Verão em Computação UNIMONTES/IFNMG a ser realizada no período 28 de janeiro à 01 de fevereiro de 2013 nas instalações da UNIMONTES.

Janeiro/2013

Sumário

Lista de Figuras **v**

1 Introdução **1**
1.1 O que é **MatLab** ? . . . 1
1.2 Sintaxe . . . 1
1.3 Toolboxes . . . 2

2 Ajuda no MatLab **4**
2.1 Comandos . . . 4
2.1.1 Comando `help` . . . 4
2.1.2 Comando `demo` . . . 4
2.2 Editor/Depurador de Programas . . . 4
2.3 Limitação de Memória . . . 5
2.4 Arquivos .m . . . 5

3 Manipulação de Matrizes, Vetores e Escalares **6**
3.1 Atribuição . . . 7
3.1.1 Comando `who` . . . 8
3.1.2 Comando `whos` . . . 8
3.1.3 Comando `clear` . . . 8
3.1.4 Comando `clc` . . . 8
3.2 Operações Básicas: +, -, *, / . . . 9
3.3 Operador dois pontos ':' . . . 12
3.3.1 Outros Usos do Operador dois pontos . . . 12
3.4 Cálculos Fundamentais e Matrizes Especiais . . . 13
3.4.1 Constantes Predefinidas . . . 14

4 Funções Elementares **16**
4.1 Funções Básicas . . . 16
4.1.1 Exemplos simples . . . 17
4.1.2 Números Complexos . . . 19
4.1.3 Comandos de Conversão . . . 21
4.2 Funções Trigonométricas . . . 21
4.3 Funções Hiperbólicas : Nomenclatura . . . 22

5 Controle de Fluxo **23**
5.1 Regras para escrever uma function . . . 25
5.2 Operadores Relacionais . . . 25
5.3 Operadores Lógicos . . . 27
5.4 Estruturas de Controle . . . 28
5.4.1 Estruturas Condicionais . . . 28
5.4.2 Estruturas de Repetição . . . 31

6 Operações Sobre Matrizes **33**
6.1 Outras Funções Úteis . . . 33

7 Medidas Estatísticas **35**

8 Gráficos **37**
8.1 Exemplo 1 - Comando `hold on` . . . 37
8.2 Exemplo 2 - Gráficos superpostos com opções distintas de pontos e traçado . . . 37
8.3 Exemplo 3 - Opções de representação gráfica simultâneas . . . 38
8.4 Exemplo 4 - Comando `subplot` . . . 39
8.5 Exemplo 5 - Comando `pie` . . . 40
8.6 Exemplo 6 - Comando `pie3` . . . 40
8.7 Exemplo 7 - Comando `bar` . . . 41
8.8 Exemplo 8 - Comando `plot3` . . . 42
8.9 Exemplo 9 - Gráfico de Superfície . . . 42
8.10 Exemplo 10 - Animação Gráfica . . . 43
8.10.1 Animação bidimensional . . . 43
8.10.2 Animação tridimensional . . . 44

9 Solução de Sistemas de Equações Lineares **45**
9.1 Métodos Diretos . . . 46
9.1.1 Eliminação de Gauss . . . 46
9.1.2 Decomposição LU . . . 46
9.2 Métodos Iterativos . . . 47
9.2.1 Método PCG . . . 48
9.2.2 Método Minres . . . 48
9.2.3 Comando `spy` . . . 49

10 Ajuste de Curvas e Interpolação **50**
10.1 Interpolação Linear por Partes . . . 50
10.2 Interpolação Superficial . . . 51
10.3 Comando `spline` . . . 52
10.4 Comando `polyfit` . . . 53

11 Leitura e Escrita de Arquivos de Dados **55**

12 Análise Polinomial **57**

13 Integração e Diferenciação **59**
13.1 Integração 59
13.2 Diferenciação 61

14 Equações Diferenciais Ordinárias **64**

15 Decomposição e Fatoração de Matrizes **66**
15.1 Fatoração Triangular - LU 66
15.2 Decomposição - QR 66
15.3 Decomposição em Valores Singulares - SVD 67
15.4 Autovalores e Autovetores 67

16 Considerações Finais **69**

Referências bibliográficas **70**

Lista de Figuras

5.1 Estruturas Condicionais: If-else-end 28
5.2 Controle de Fluxo: If-else-end 29
5.3 Estruturas Condicionais: Switch-case-otherwise-end 30

7.1 Gráfico de barras: Histograma 36

8.1 Comando hold on : Permite plotar várias figuras numa mesma janela 38
8.2 Gráficos superpostos com opções distintas de pontos e traçado 38
8.3 Opções de representação gráfica simultâneas 39
8.4 Comando subplot 40
8.5 Comando pie 41
8.6 Comando pie3 41
8.7 Comando bar 41
8.8 Comando plot3 42
8.9 Gráfico de Superfície 43

9.1 Gráfico mostrando os valores não nulos de uma matriz 49

10.1 Interpolação linear por partes 51
10.2 Superfície interpolante da matriz temperatura 52
10.3 Interpolação utilizando spline 53
10.4 Ajuste polinomial de grau 1 54

12.1 Curvas contínuas de um polinômio e a sua derivada 58

13.1 Derivada numérica: 30 divisões do intervalo [-1,1] 63
13.2 Derivada numérica: 100 divisões do intervalo [-1,1] 63

14.1 Gráfico das soluções exata e numérica da equação $y' = 3x^2$ 65

Capítulo 1

Introdução

1.1 O que é MatLab ?

Matlab(MATrix LABoratory) é uma ferramenta computacional(software) que possui diversas funcionalidades numéricas, algébricas e gráficas. O Matlab permite a resolução de vários problemas matemáticos, sem requerer muito conhecimento de programação, como seria necessário para resolver os mesmos problemas em linguagens de programação de baixo nível, como por exemplo C.

Os usos típicos para o Matlab incluem:

- Cálculos matemáticos;
- Desenvolvimento de algoritmos;
- Modelagem, simulação e confecção de protótipos;
- Análise, exploração e visualização de dados;
- Gráficos científicos e da engenharia;
- Desenvolvimento de aplicações, incluindo a elaboração de interfaces gráficas com o usuário.

1.2 Sintaxe

A sintaxe do Matlab é diferente da sintaxe usada por nós no dia-a-dia, por motivos óbvios, como podemos ver nos exemplos abaixo.

- Lucas Thundercats mudou sua foto do facebook.

Essa frase pode ter dois entendimentos:

- A foto pertencente a Lucas Thundercats foi mudada.
- A foto de outra pessoa foi mudada por Lucas Thundercats.

Essa frase é ambígua, e para um computador, que é uma maquina, não é possível destinguir o sentindo analisando o contexto.

- Se você não comer seu almoço, não comerá a sobremesa.
- Se você não alcançar o total na primeira prova, não alcançará total no final do trimestre.

Na primeira frase, é esperado que se a pessoa comer o almoço, ela comerá sobremesa, porém na segunda frase, se não satisfeita a primeira condição, não se espera que aconteça o contrário. Para a linguagem computacional, esse tipo de ambiguidade não pode ocorrer, portanto o "se"do Matlab é semelhante ao do segundo caso, que, por sua vez, é semelhante ao "se"matemático.

- André atirou em João Pessoa.

A ambiguidade dessa frase é causada pelo fato de uma cidade possuir um nome próprio, pois não sabemos se André atirou estando em João Pessoa, ou em alguém com esse nome. Por isso, no Matlab não se pode repetir nomes nem usar palavras reservadas.

Dessa forma, abordaremos adiante como nomear variáveis e utilizar comandos.

1.3 Toolboxes

O Matlab é tanto um ambiente quanto uma linguagem de programação, e um dos aspectos mais poderosos é o fato de que a linguagem Matlab permite construir suas próprias ferramentas reutilizáveis. O usuário pode facilmente criar suas próprias funções e programas especiais em linguagem Matlab. A medida que se escreve mais e mais funções para lidar com certos problemas naturalmente se é levado a agrupar por conveniência, funções relacionadas entre si em diretórios especiais. Isso nos introduz o conceito de Toolbox: uma coleção de arquivos para tratar classes especiais de problemas. As toolboxes são mais do que uma simples coleção de funções úteis, elas representam os esforços de alguns dos maiores pesquisadores do mundo em campos como controle, processamento de sinais e identificação de sistemas, dentre outros. Novas toolboxes são criadas a cada ano, dentre alguns exemplos no Matlab tem-se:

- Toolbox de Processamento de Sinais;
- Toolbox de Identificação de Sistemas;

- Toolbox de Otimização;
- Toolbox de Sistemas de Controle;
- Toolbox de Controle Robusto;
- Toolbox de Redes Neurais;
- Toolbox Spline.

Capítulo 2

Ajuda no MatLab

2.1 Comandos

2.1.1 Comando `help`

Uma vez inicializado, o Matlab, aparecerá na janela de comandos um prompt ». O prompt significa que o Matlab está esperando um comando. Todo comando deve ser finalizado teclando-se Enter. No Matlab, pode-se obter ajuda sobre qualquer comando ou função. O comando » help (sem o prompt ») mostra uma listagem de todos os pacotes disponíveis. Ajuda sobre um pacote específico ou sobre um comando ou função específica é obtida com o comando » help nome,(sem a vírgula) onde nome pode ser o nome de um pacote ou o nome de um comando ou função.

2.1.2 Comando `demo`

A visualização de um exemplo de código e sua execução pode, em alguns casos, ser mais explicativa do que a própria definição do comando ou função. Para isso, usa-se o comando demo, que permite o acesso a uma janela indexada com várias ferramentas que o Matlab implementa, as quais apresentam rápidas demonstrações, como vídeo-aulas e tutoriais, contendo informações úteis para uso eficiente dos comandos disponibilizados pelo software.

2.2 Editor/Depurador de Programas

A atual versão do Matlab apresenta um editor específico para manipular arquivos de dados e executáveis, além de permitir a depuração de programas na linguagem do Matlab. Este editor conta com comandos do tipo break line, step into, etc. Também existem no Matlab comandos

específicos para depurar um programa. Para acessar pela ajuda e conhecer estes comandos digite: » help debug e em seguida pode-se fazer o help de qualquer um dos comandos mostrados.

2.3 Limitação de Memória

É recomendável, quando se trabalha com programas ou algoritmos que utilizam grande quantidade de memória, verificar a priori se a memória alocada é suficiente para evitar que em determinado momento da execução o programa aborte por insuficiência de memória. Isto pode às vezes ser testado pelo dimensionamento dos arranjos que em determinado momento serão gerados. Também isto pode ser monitorado testando ou executando isoladamente aquelas partes do algoritmo que apresentaram maior consumo de memória.

Sabe-se que não existe limitação computacional para a utilização das ferramentas do Matlab a não ser àquelas impostas pela máquina em que esta sendo executado. A este respeito existe o comando bench, que testa a priori a performance da máquina do usuário, comparando certos algoritmos, como por exemplo, a decomposição matricial LU dentre outros, e retornando o tempo de CPU esperado. Este comando sem argumentos retorna, depois de um tempo, uma lista de máquinas com diferentes estruturas, e os tempos de CPU que elas utilizam incluindo a máquina do usuário, para desta forma efetuar a comparação.

2.4 Arquivos .m

Os comandos do MatLab são normalmente digitados na janela de comando, onde uma única linha de comando é introduzida e processada imediatamente. O MatLab é também capaz de executar sequências de comandos armazenadas em arquivos.

Os arquivos que contêm as declarações do MatLab são chamadas arquivos ".m", e consistem de uma sequência de comandos normais do MatLab, possibilitando incluir outros arquivos ".m"escritos no formato texto (ASCII).

Para editar um arquivo texto na janela de comando do MatLab, selecione **New M-File** para criar um novo arquivo ou **Open M-File** para editar um arquivo já existente, a partir do menu **File**. Os arquivos podem, também, ser editados fora do MatLab utilizando qualquer editor de texto.

Capítulo 3

Manipulação de Matrizes, Vetores e Escalares

Uma das praticidades do MatLab é que não se faz necessária a declaração do tipo da variável se esta não for uma variável global. Isso ocorre porque, para o MatLab, todas as variáveis constituem-se matrizes.

Por exemplo, ao definirmos a seguinte frase:

» text = 'I Escola de Verão';

Automaticamente, a variável *text* é tratada como um vetor de 17 posições ou como uma matriz linha de dimensões 1x17. De fato, para manipularmos o nosso vetor ou matriz, ao utilizarmos o comando

» text

Obteremos como saída:

text =

I Escola de Verão

Da mesma forma, se utilizarmos o comando

» text(3)

Obteremos como saída:

ans =

E

Se quisermos tratar a variável como uma matriz, ao utilizarmos o comando

» text(1,3)

Obtemos como saída:

ans =

E

Podemos notar que, para acessar apenas uma posição do vetor, ou matriz, basta que esta seja identificada entre parênteses. Repare que a variável ans (de answer) é padrão para o MatLab e é utilizada sempre que um resultado, que é retornado por alguma função, não foi designado a nenhuma outra variável.

3.1 Atribuição

Para atribuir um valor de retorno a uma variável basta digitar o nome da variável seguida do sinal = à esquerda da função que retorna o valor ou cadeia de caracteres (string). Para melhor demonstrar, temos os seguintes exemplos:

```
>> laranjas = 4;

>> bananas = 10;

>> preco.laranja = 1.75;

>> preco.banana = 2.3;

>> frutas = laranjas + bananas;

>> preco.medio = (laranjas*preco.laranja + bananas*preco.banana)/frutas;
```

3.1.1 Comando `who`

Quando se faz necessário o conhecimento de quais variáveis estão ativas no ambiente (atualmente definidas), utiliza-se o comando who. Dessa forma, tendo sido declaradas todas as variaveis supracitadas, temos:

» who

Your variables are:
bananas frutas laranjas preco

3.1.2 Comando `whos`

Além de querer saber quais são as variáveis ativas, se quisermos ter conhecimento também sobre o tamanho e tipo, podemos utilizar o comando whos:

» whos

Name	Size	Bytes	Class	Attributes
bananas	1x1	8	double	
frutas	1x1	8	double	
laranjas	1x1	8	double	
preco	1x1	552	struct	

3.1.3 Comando `clear`

Como já foi falado, o MatLab dispõe de vários comandos simples e práticos. Algumas vezes, utilizamos variáveis em apenas uma parte do programa, dessa forma, podemos limpar a memória exigida por essa variável, através do comando

» clear + <nome da variavel>

Temos também a possibilidade de apagar todas as variáveis ativas no ambiente, utilizando apenas os comandos

» clear ou » clear all

3.1.4 Comando `clc`

Se, além de apagar todas as variáveis, deseja-se, também, limpar a tela, o MatLab dispõe o comando » clc para tal.

3.2 Operações Básicas: +, -, *, /

Por exemplo, para definir uma matriz não se requer dimensionamento prévio, embora isto seja possível. Existem várias formas de se definir uma matriz. Por exemplo, os valores são inseridos por linhas, separadas estas por ";"ou "Enter"da seguinte maneira:

» A = [2 1 1; 1 2 1; 1 1 2]

A =

2	1	1
1	2	1
1	1	2

» B = [1 2 3 4 + <ENTER>
5 6 7 8 + <ENTER>
9 10 11 12]

B =

1	2	3	4
5	6	7	8
9	10	11	12

» C = [1 0 0; 0 1 0; 0 0 1]

C =

1	0	0
0	1	0
0	0	1

Já que qualquer dado numérico é considerado um arranjo, os símbolos +, -, *, / são designados como operações matriciais ao passo que para operações com escalares podem ser utilizados os símbolos .- , .+ , .* , ./ . Quando o arranjo é um número pode-se usar um ou outro símbolo indistintamente. Usando as matrizes A, B e C definidas acima observem os seguintes exemplos:

» A*C

ans =

2 1 1
1 2 1
1 1 2

Produtos entre matrizes de dimensões distintas são possíveis desde que respeitadas as dimensões para o produto, por exemplo, A*B é bem definida, já B*A não é permitido dado que o número de colunas de B é 4 e o número de linhas de A é 3.

» A*B

ans =

16 20 24 28
20 24 28 32
24 28 32 36

» B*A

??? Error using ==> mtimes
Inner matrix dimensions must agree.

» MAT = A.*C

MAT =

2 0 0
0 2 0
0 0 2

Isto corresponde a um produto componente a componente, isto é, a matriz MAT cujas entradas são $MAT(i,j) = A(i,j) * C(i,j)$.

» A+1

ans =

3 2 2
2 3 2
2 2 3

As formas A.+1, A(1,1).+1 não são validas ao passo que a forma 5.+1 é aceita.

```
» A*5 %(ou A*.5)

ans =

  10   5   5
   5  10   5
   5   5  10

» C/A %ou C * inv(A)

ans =

  0.7500  -0.2500  -0.2500
 -0.2500   0.7500  -0.2500
 -0.2500  -0.2500   0.7500

» C./A %é a matriz definida por C(i,j)/A(i,j)

ans =

  0.5000        0        0
       0   0.5000        0
       0        0   0.5000
```

Outras operações ainda são permitidas:

```
» X = [20 21 22];
» Y = [A ; X]

Y =

  2   1   1
  1   2   1
  1   1   2
 20  21  22
```

Esta forma é útil para se trabalhar com sub matrizes ou matrizes em blocos. Para atribuir um valor a uma determinada entrada de uma matriz:

```
» A(2,3) = 0

A =
```

```
2   1   1
1   2   0
1   1   2
```

3.3 Operador dois pontos ':'

É útil considerar vetores construídos com valores contidos em intervalos. Duas formas podem ser usadas:
vet = valor inicial : valor final; ou

vet = valor inicial : incremento : valor final;

Exemplos:

» v = 3:7

v =

```
 3   4   5   6   7
```

» v = 3: 1.5: 6

v =

```
 3.0000   4.5000   6.0000
```

3.3.1 Outros Usos do Operador dois pontos

Pode-se selecionar sub matrizes de uma matriz utilizando o operador dois pontos. Considerando a matriz B definida acima, sub1 é formada por todas as linhas das colunas 2 e 3 de B.

» sub1 = B(:,2:3)

sub1 =

```
 2     3
 6     7
 10   11
```

A matriz sub2 é definida pelas entradas de B das linhas 2 e 3 interceptadas com as colunas 1 até 3.

```
» sub2 = B(2:3,1:3)

sub2 =

 5    6     7
 9   10   11
```

3.4 Cálculos Fundamentais e Matrizes Especiais

Pode-se transformar uma matriz em vetor coluna da seguinte maneira:

```
» a = [1 2;3 4]; % matriz 2x2

» b = a(:)

b =

1
2
3
4
```

Para saber a dimensão de uma matriz:

```
» size(a)

ans =

 2   2
```

```
» f = 1:3; % vetor dos inteiros de 1 até 3
» b = f'; % transposta de f
» a = b*f % matriz 3x3
```

a =

1 2 3
2 4 6
3 6 9

» b = 3*a; % redefine b com componentes $b(i,j) = 3 * a(i,j)$
» c = a/5; % c é definida por : $c(i,j) = a(i,j)/5$
» d = a. ^ 2; % d é definida por : $d(i,j) = a(i,j)^2$
» e = 3. ^ a; % e é definida como: $e(i,j) = 3^{a(i,j)}$

Potências escalares usando matrizes:

» a = [1 2 3];
» b = [2 2 2];
» a. ^ b

ans =

1 4 9

O comando *zeros* aloca uma matriz de zeros de um dado tamanho.

» A = zeros(2); % aloca uma matriz 2x2 de zeros
» B = zeros(3,2); % a matriz de zeros é de tamanho 3x2
» C = ones(2,3); % matriz de uns de dimensões 2x3
» D = eye(4); % matriz identidade de ordem 4

3.4.1 Constantes Predefinidas

Existem no Matlab certos nomes de variáveis que assumem valores "*default*". Algumas destas "constantes"são:

- *pi* : constante de proporção entre o perímetro da circunferência e seu diâmetro;
- *i,j* : número imaginário igual a raiz quadrada de -1;
- *inf* : denota o infinito na reta;
- *NaN* : quando o resultado de uma operação errada não é um número (**N**ot **a** **N**umber);

- *date* : retorna a data atual;
- *clock* : retorna a hora no formato de vetor : ano, mês, dia, hora, minuto, segundo;
- *ans* : variável de saída "*default*".

Apesar das características especiais destas variáveis elas não são proibidas no sentido de poderem ser definidas pelo usuário. Se isto acontecer, o valor atual destas variáveis será aquele definido pelo usuário. Depois de aplicado o comando *clear*, sobre a "constante"redefinida pelo usuário, ela assumira o seu valor "*default*"como constante predefinida pelo Matlab.

Por exemplo:

» date

ans =

08-Jan-2013

» date = 10

date =

10

» clear date

» date

ans =

08-Jan-2013

Capítulo 4

Funções Elementares

Nesta seção, serão apresentados comandos para efetuar chamadas das funções matemáticas elementares, como funções analíticas trigonométricas, exponenciais, logarítmicas, que podem ser usadas tanto em escalares como em vetores e matrizes. No Matlab trabalha-se com listas numéricas, portanto para se calcular o valor de uma função conhecida ou definida, em uma variável x, deve-se conhecer o valor x ou a lista x. Se x é um número a função retornará um número. Se x é uma lista, a função também retornara uma lista de valores, os correspondentes valores da função para cada valor da lista. Por exemplo:

» x = [.1 .2 .3];

» $sin(x)$ % (função trigonométrica seno)

ans =

0.0998 0.1987 0.2955

Isto é, $[sin(.1)sin(.2)sin(.3)]$. Ainda se x é uma matriz qualquer o resultado de $b = sin(x)$ também é uma matriz cujas componentes são $b(i,j) = sin(x(i,j))$.

4.1 Funções Básicas

Apresentam-se a seguir algumas funcoes simples que podem ser usadas tanto em escalares como em matrizes ou vetores.

» abs(x); % valor absoluto da variavel x
» sqrt(x); % raiz quadrada de x

```
» round(x); % arredonda x para o inteiro mais proximo
» fix(x); % arredonda x para o inteiro mais proximo de zero
» floor(x); % arredonda x para o inteiro mais proximo de −∞
» ceil(x); % arredonda x para o inteiro mais proximo de +∞
» sign(x); % sinal de x, +1 ou -1
» rem(x); % resto de x:y
» exp(x); % exponencial de x
» log(x); % funcao logaritmo com base e=2.7182818284590...
» exp(1); % Neperiano
» log10(5); % logaritmo em base 10 de 5
```

4.1.1 Exemplos simples

Defina as seguintes matrizes:

```
» a = -2.6;
» A = [1 2 3];
» B = [2 2 2];
» C = [2.6 1.3 -3.2 3.5];
```

entao os seguintes calculos sao obtidos:

```
» x = round(a) % inteiro mais proximo

x =

-3

» x=fix(a) % inteiro mais proximo de zero

x =

-2

» x=floor(C) % inteiro mais proximo de −∞

x =
```

2 1 -4 3

» x = ceil(C) % inteiro mais proximo de $+\infty$

x =

3 2 -3 4

» x = sign(C) % sinal

x =

1 1 -1 1

» abs(a) % valor absoluto

x =

2.6

» x = sqrt(C) % raiz quadrada

x =

1.6125 1.140 0 + 1.7889i 1.8708

No caso do numero negativo o terceiro valor resulta num complexo.

» x = rem(7.5,2) % resto da divisao 7.5/2

x =

1.5

» x = rem(A,B) % resto da divisao entre as componentes

x =

```
1   0   1
```

No caso do argumento de entrada ser uma lista/matriz, cada elemento da lista (ou matriz) resposta e a solucao dada pela funcao que correspondente ao elemento na mesma posicao na lista de entrada. Por exemplo:

```
» rem( [3 3 3;4 4 4;5 5 5] , [1 2 3;1 2 3;1 2 3] )

ans =

 0   1   0
 0   0   1
 0   1   2
```

4.1.2 Números Complexos

Como citado, o valor padrão de i e j é a raiz quadrada de -1. Logo, um número complexo, que se define da seguinte maneira:

```
» c1 = 3-2*i;
» c2 = -1/2+j;
```

As operações aritméticas podem ser usadas livremente.

```
» c1+c2;
» c1*c2;
» c1/c2;
```

O produto *c1*c2* corresponde ao produto de dois binômios com a restrição de que *i*i = j*j = -1*. O número complexo *1/c2* é definido tal que *c2*(1/c2) = 1*. As partes real e imaginária de c1 podem ser obtidas com:

```
» real(c1);
» imag(c1);
```

Outras funções envolvendo complexos são exemplificadas a seguir. O valor absoluto de um complexo *a+b*i* é a norma de (a,b).

```
» abs(c1)
```

ans =

3.6056

» conj(2+3*j);% é o complexo conjugado de 2+3*j, isto é, 2-3*i

A forma polar de um complexo também pode ser obtida.

» a = 1;
» b 2;
» x = a+b*i;
» r = abs(x);
» theta = angle(x); % retorna o ângulo polar do vetor (a,b) ou atan(b/a)
» y = r*exp(theta*i); % forma polar
» z = r*(cos(theta)+i*sin(theta)); % é igual a y e igual a x (formas equivalentes de um complexo)

Exemplo simples de cálculo com utilização da função tangente.
» A = [-pi/5 , 0;1+2*i , 3*pi/2+0.001];
» tan(A)
ans =

1.0e+02 *
 -0.0073   0
 0.0003 + 0.0101*i -10.0000

O primeiro valor (1.0e+02) indica que deverá multiplicar-se o número 100 a matriz 2x2 que vem em seguida. Para aumentar a precisão na resposta numérica aplique-se o comando:

» format long; % precisão da ordem 10-15
» 1/pi
ans =

0.31830988618379

Por padrão a precisão mostrada é de 5 dígitos que é recuperada com o comando:

» format short; % ou simplesmente format designa a precisão default
» ans % variável de saída padrão que guarda o último resultado

ans =

0.3183

4.1.3 Comandos de Conversão

Existe um número grande de funções que permite a conversão entre diferentes tipos de dados. Algumas destas funções são exemplificadas a seguir.

» letra = num2str(2); % permite manipular um número como um caractere

Isto é uma abreviação das palavras "numeric to string". De tal forma que a variável letra não é mais um número e sim um caractere ao passo que o resultado da operação:

» letra+1

ans =

51

não é 3 que seria a soma 2+1.

» str2num(letra)+1 % converte o conteúdo da variável letra, isto é, o caractere 2 para o número 2 e soma-lhe 1

ans =

3

» dec2hex(11); % converte o decimal 11 para o sistema hexadecimal (B)
» hex2dec('F'); % converte o valor hexadecimal para o decimal (15)

4.2 Funções Trigonométricas

Os números contidos na variável argumento destas funções podem assumir qualquer valor real ou complexo, considerado este em radianos.

» sin(x); % função trigonométrica seno de um ângulo

```
» cos(x); % função coseno
» tan(x); % tangente de x
» asin(x); % arcoseno ou função inversa do seno do argumento x
» acos(x); % arco coseno ou inversa do coseno de x
» atan(x); % inversa da tangente de x
```

As funções csc(x), sec(x), cot(x), acsc(x), asec(x), acot(x) definem as funções cossecante, secante e cotangente de x e suas inversas respectivamente.

4.3 Funções Hiperbólicas : Nomenclatura

De forma semelhante às funções trigonométricas, definem-se os comandos para calcular funções hiperbólicas.

```
» sinh(x); % seno hiperbólico de x
» cosh(x); % coseno hiperbólico de x
» tanh(x); % tangente hiperbólica de x
» asinh(x); % inversa do seno hiperbólico de x
» acosh(x); % inversa do coseno hiperbólico de x
» atanh(x); % inversa da tangente hiperbólica de x
```

Com definições equivalentes respectivamente para csch(x), sech(x), coth(x), acsch(x), asech(x) e acoth(x).

Capítulo 5

Controle de Fluxo

As funções executáveis escritas na linguagem utilizada pelo Matlab estão implementadas em arquivos com extensão ".m". Suponha que o arquivo "exemplo.m"contenha uma série de linhas de comandos ou funções do Matlab, então quando este arquivo é chamado sem extensão do prompt do Matlab:

» exemplo + <ENTER>

o Matlab entenderá que este arquivo é executável e assim cada linha será interpretada na sequência de acordo com a sua função.

Da mesma forma, um método ou sub-rotina pode ser definido dentro de um arquivo com extensão ".m"contendo argumentos de entrada e de saída. Por exemplo, cria-se o arquivo "circum.m"contendo as seguintes linhas:

% Esta função calcula o perímetro de uma circunferência de
% raio R. Si R é uma matriz, circum(R) retornará uma matriz
% contendo os perímetros das circunferências de raios iguais
% aos respectivos valores da matriz R
$function C = circum(R)$
$C = 2 * pi * R;$

As linhas acima correspondem ao arquivo "circum.m". Para utilizar está função basta digitar:

» circum(2)

ans =

12.5664

Em caso de ser R um vetor:

» R = [1 2 3];

» circum(R)

ans =

6.2832 12.5664 18.8496

As primeiras linhas iniciadas com o símbolo % constituem uma valiosa fonte de informações a respeito da funcionalidade do arquivo. Para ter acesso a elas não é necessário abrir o arquivo com um editor, basta digitar:

» help circum

então as primeiras linhas juntas de comentários serão mostradas. Usando lookfor como exemplo, obtém-se a saída:

» lookfor perímetro

circum - Esta função calcula o perímetro de uma circunferência de

Este comando mostra só a primeira linha do arquivo, caso esta contenha a palavra perímetro, alem do nome do arquivo. Por isto é importante documentar de forma precisa esta primeira linha, caso o número de funções venha a aumentar consideravelmente. Não é necessário que o nome do arquivo seja igual ao nome da "function"dentro do mesmo.

Quando a função tem mais de um argumento de saída, digamos área, volume, então estes devem vir entre colchetes, por exemplo, a seguinte função definida no arquivo "cilindro.m"calcula a área e o volume de um cilindro.

```
% Calcula-se a área e volume de um cilindro
% de altura h e raio r
function[area, volume] = cilindro(h, r)
area = 2 * pi * r * h + 2 * pi * r ^ 2;
```

$volume = pi * r \hat{} 2 * h;$

Para chamar esta função seja do "prompt"ou de outro executável ".m", basta digitar uma chamada na forma exemplificada a seguir:

$[a, v] = cilindro(0.5, 1)$

então, as variáveis a e v conterão as informações da área e do volume do cilindro de h = 0.5 e r = 1.

5.1 Regras para escrever uma function

É conveniente observar as seguintes características na definição de uma função: Escrever comentários a respeito da função, tendo especial ênfase na primeira linha de comentário.

A primeira linha executável deve conter a palavra "function"seguida dos argumentos de saída entre colchetes, se forem mais de um, o sinal "="e o nome da função com argumentos de entrada entre parênteses.

Todos os argumentos de retorno devem estar definidos dentro do corpo da função.

Qualquer erro cometido dentro do arquivo ".m"será acusado quando este for executado, retornando o formato de um erro de compilação.

5.2 Operadores Relacionais

Estes operadores são similares aqueles usados em uma linguagem de programação tal como C ou Fortran.

Operador	Descrição
$<$	menor que
$<=$	menor ou igual
$>$	maior que
$>=$	maior ou igual
$==$	igual no sentido de condição/comparação
$\sim =$	não igual, distinto

Exemplos:

» 1<2

ans =

1

Esta resposta indica que a condição 1<2 é verdadeira. Se o retorno é zero, isto indica que a condição é falsa. » 1>2

ans =

0

No caso do uso de arranjos: » a = [1 2;3 4];
» b = [0 2;4 -5];
» a = b

ans =

1 0
1 1

Isto quer dizer que: 1 distinto de 0 é verdadeiro, 2 distinto de 2 é falso, 3 distinto de 4 é verdadeiro e 4 distinto de -5 é verdadeiro. Ao se comparar caracteres também ocorre algo semelhante, por exemplo:

» texto1 = 'teste';
» texto2 = 'texte';
texto1 == texto2

ans =

1 1 0 1 1

Esta forma efetua uma comparação caractere por caractere, posição a posição. Se as letras são iguais retorna 1, se não retorna 0. Um comando mais direto para saber se dois textos são iguais na sua totalidade é:

» strcmp(texto1,'teste')

ans =

1

Isto é, o conteúdo da variável texto1 é exatamente o conjunto dos caracteres da palavra teste.

5.3 Operadores Lógicos

São amplamente utilizados em expressões condicionais do tipo "if-else", as quais serão abordadas detalhadamente na próxima seção.

Operador	Descrição
&	e
|	ou
~	não

Por exemplo:

» a = 2; b = 0; c = 1; % é possível definir variáveis em uma mesma linha
» a < b & b < c % equivale a questionar: é $a < b < c$?

ans =

0

A resposta é que a afirmação é falsa. Para isso basta que seja $a \geq b$ ou que $b \geq c$.
Outro exemplo:

» $\sim (b == c \mid c < a)$ % pergunta: é falso que b é igual a c ou c é menor que a?

ans =

0

5.4 Estruturas de Controle

O Matlab, como toda linguagem de programação, possui estruturas que permitem o controle do fluxo de execução de comandos baseadas em tomadas de decisão.

5.4.1 Estruturas Condicionais

São estruturas que, em uma sequência de comandos, são executadas dependendo do resultado de uma condição.

- **If-else-end**

Existem três variantes no Matlab de estruturas if-else-end:

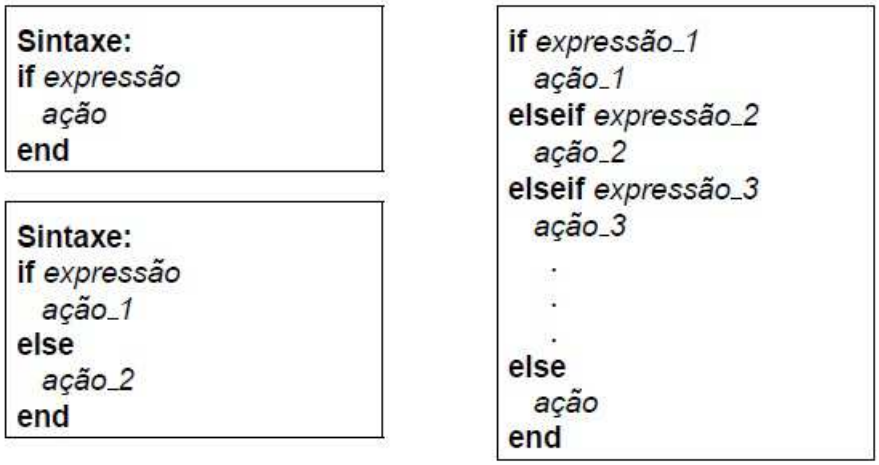

Figura 5.1: Estruturas Condicionais: If-else-end

Exercício Resolvido:

Um problema muito comum é escolher o material adequado para cercar um terreno. Se o preço total da cerca for menor que R$ 100,00, construímos com um material, e se for maior, construímos com outro. Como fazer um programa que fale qual material usar se tivermos o valor do metro de cerca e o perímetro do terreno em metros?

Solução:

```
function M = decidematerial(precoMetro, perimetro)
    if((precoMetro*perimetro) < 100)
        M = 1;
    else
        M = 2;
    end
```

A estrutura de condição if-else-end funciona como uma bifurcação do fluxo de execução do programa, abrindo duas possibilidades de caminhos(comandos), como pode ser visualizado a seguir:

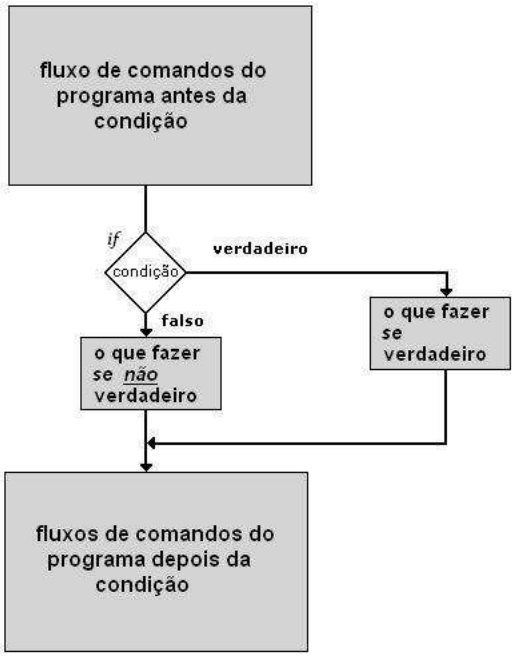

Figura 5.2: Controle de Fluxo: If-else-end

- **Switch-case-otherwise-end**

O switch case é um bloco de controle que funciona como um agrupamento de vários ifs e elses, que tenham relação entre si.

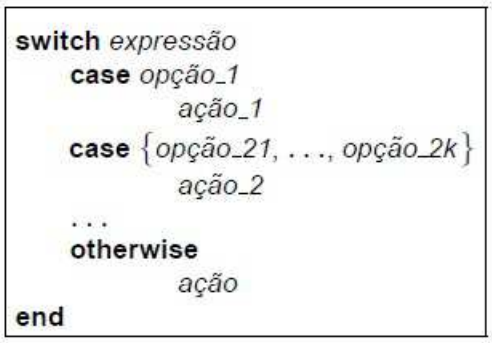

Figura 5.3: Estruturas Condicionais: Switch-case-otherwise-end

Temos o exemplo de instruções que associem números aos dias da semana, no qual teríamos muito trabalho fazendo um if para cada caso. Para facilitar esse caso usamos o switch-case:

```
mynumber = input('Enter a number:');

switch mynumber
case 1
    dia = 'Domingo'
case 2
    dia = 'Segunda-feira'
case 3
    dia = 'Terca-feira'
case 4
    dia = 'Quarta-feira'
case 5
    dia = 'Quinta-feira'
case 6
    dia = 'Sexta-feira'
case 7
    dia= 'Sabado'
otherwise
    dia= 'nenhum'

end
```

5.4.2 Estruturas de Repetição

As estruturas de repetição são a razão para hoje os computadores serem tão úteis como são. É a partir de repetições de operações que se calcula numericamente os valores de ln(2) ou arcsin(0.43) por exemplo, e é partir de repetições também que se integra, deriva, encontra a raiz de equações transcendentais, e se acha os termos de uma série ou sequência. As estruturas de repetição são praticamente maneiras de "mandar o computador fazer algo que você queira". Por isso são tão importantes de serem estudadas. Por exemplo se computadores tivessem mecanismos de cortar batatas, e quiséssemos mandá-lo cortar batas usaríamos os comandos a seguir em sintaxe computacional.

Enquanto tiver batatas com casca na cesta
Descasque uma batata
Fim

Esse exemplo seria para o caso de quisermos cortar todas as batatas, sem nos preocuparmos com a quantidade de batata a ser cortada. Porém e se fosse de nosso interesse descascar 100 batatas? Pediríamos assim:

Para 100 batatas da cesta
Descasque uma batata
Fim

Como abstração, enxergamos que em uma estrutura de repetição precisamos avisar para o computador quando parar, seja quando atingir uma condição preestabelecida ou quando atingir uma quantidade predeterminada. Para o primeiro caso temos a estrutura *while*, e para o segundo temos a estrutura *for*.

- **While**

A estrutura de repetição *while* é usada quando não se sabe ao certo o número de repetições, como é mostrado no exemplo a seguir:

```
% aproximação inteira de log de n na base dois
n = 1024;

log = 0;

while(n > 1)
        n = n/2;
```

```
        log = log+1;
end

log =

    10
```

- **For**

Diferente do *while*, a estrutura de repetição *for* é usada quando se sabe ao certo o número de repetições, como é mostrado no exemplo a seguir:

```
% prencher a matriz A NxN
N = 5;

for linha = 1:N
    for coluna = 1:N
        A(linha,coluna) = linha+coluna;
    end
end

A =

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

Capítulo 6

Operações Sobre Matrizes

O Matlab conta com algumas operações especiais sobre matrizes que se tornam necessárias quando se trabalha matematicamente com elas.

```
» A = rand(5); % gera uma matriz 5x5 randômica
» A'; % transposta de A
» det(A); % determinante de uma matriz
» inv(A); % inversa de A, quando esta existe
» diag(A);% gera um vetor coluna com a diagonal de A
» C = [1 0 2 1 1];
» diag(C); % gera uma matriz 5x5 cuja diagonal é C
» v1 = [1 2 3 4 5];
» v2 = [6 7 8 9 0];
» sum(v1.*v2); % produto escalar de dois vetores
» B = randn(3,5); % matriz aleatoria 3x5
```

6.1 Outras Funções Úteis

Existem outras operações especiais sobre matrizes. Algumas delas são dadas a seguir. Considerando as definições dadas anteriormente para as matrizes A, B e C.

```
» rot90(A); % rotação de 90 graus da matriz A
» fliplr(A); % permuta as colunas primeira e última de A
» flipud(B); % permuta as linhas primeira e última de B
» D = reshape(C,5,1); % reescreve C com diferentes dimensões
» E = reshape(B,5,3); % reescreve B com diferentes dimensões
```

```
» F = triu(A); % reescreve A preenchendo com zeros a parte triangular superior
» G = tril(A); % G é a parte triangular inferior de A
```

Capítulo 7

Medidas Estatísticas

No próximo exemplo constrói-se uma matriz aleatoriamente onde as colunas correspondem as notas de uma turma de 13 alunos, cada coluna representa uma matéria. As linhas representam as notas de cada aluno.

```
% ex7
% Este exemplo calcula medidas estatísticas relativas as notas
% de 6 matérias de uma turma de 13 alunos

for i=1:13,
    for j=1:6,
   TURMA(i,j)=100*rand;
    end
end

media_turma = mean(TURMA); % media por matéria (turma toda)

media_portugues = mean(TURMA(:,3)); % média das notas da coluna

[nota_minima,numero_alunos] = min(TURMA); % nota mínima de cada matéria

mat = TURMA(:,1); % primeira coluna da matriz TURMA

[ordem_ascendente,num] = sort(mat); % ordem ascendente do vetor mat
                                    % num é o número do aluno na lista

[nota_maxima,numero_alunos] = max(TURMA); % nota máxima de cada matéria e
                                  % o número do aluno (o primeiro que achar)
```

```
std(mat); % desvio padrão da primeira matéria
hist(nota_minima); % histograma (10 barras por default) figura.
[n,m] =hist(nota_minima); % n=altura, m = centro da barra
hist(nota_minima,3); % histograma com 3 barras
[n,m] = hist(nota_minima,3);
```

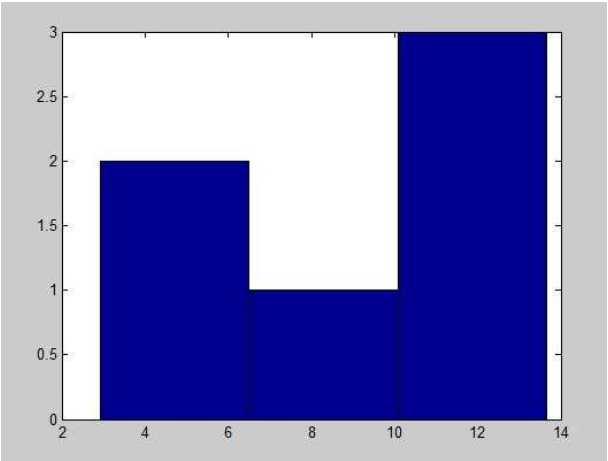

Figura 7.1: Gráfico de barras: Histograma

Capítulo 8

Gráficos

Para obter gráficos em 2D usa-se o comando plot. Este comando permite uma variedade de argumentos de entrada. Estes argumentos permitem adicionar uma série de opções à saída gráfica como título para os eixos, título para o gráfico, opções para o tipo de letra, tamanho, cores. É possível a superposição de gráficos ou obter vários gráficos em uma mesma janela. A seguir serão exemplificados estas opções, com vários gráficos.

8.1 Exemplo 1 - Comando `hold on`

```
% ex8.1
x = 0:0.02*pi:2*pi;
y = sin(x);
z = cos(x);
plot(x,y); % gráfico do seno de x
hold on; % permite a saída do próximo gráfico na mesma janela que o anterior
plot(x,z); % gráfico do coseno de x
plot(x,y,x,z); % grafica simultaneamente seno e coseno na mesma janela
```

8.2 Exemplo 2 - Gráficos superpostos com opções distintas de pontos e traçado

```
% ex8.2
x = (0:0.21:2*pi); % 30 divisões do intervalo [0,2*pi]
y = sin(x);
plot(x,y,'-'); % o traço da gráfica sin(x) é pontilhado
```

```
z = cos(x);
plot(x,y,'b:p',x,z,'c-',x,z,'m+');
```

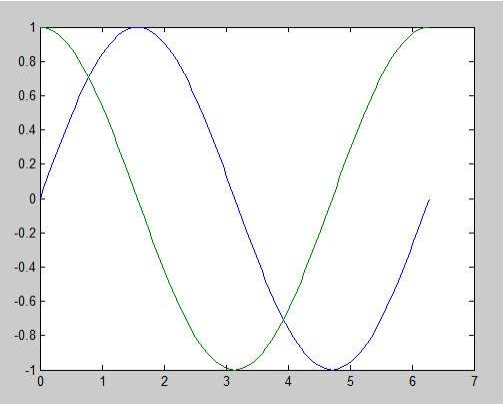

Figura 8.1: Comando hold on : Permite plotar várias figuras numa mesma janela

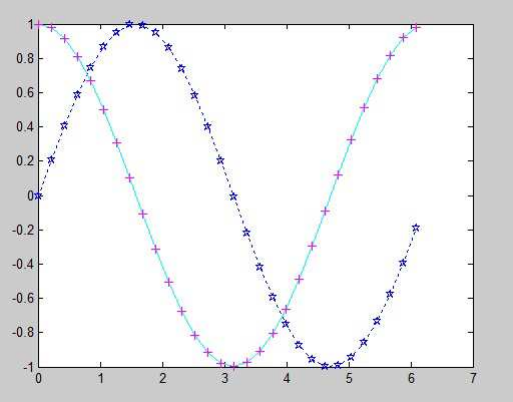

Figura 8.2: Gráficos superpostos com opções distintas de pontos e traçado

8.3 Exemplo 3 - Opções de representação gráfica simultâneas

```
% ex8.3
x = (0:0.21:2*pi); % 30 divisões do intervalo [0,2*pi]
y = sin(x);
z = cos(x);
t = tan(x); % tangente de x
ct = cot(x); % cotangente de x
set(gcf,'Color',[1 1 1]); % fundo branco
plot(x,y,'r*',x,z,'mx',x,t,'bo-',x,ct,'kp-');
axis([0 6 -3.5 3.5]); % estabelece os limites para os eixos atuais
grid; % cria uma grade superposta ao gráfico atual

title('Funções Trigonométricas','FontSize',14,'Fontweight','Bold');

legend('sin(\theta)','cos(\theta)','tan(\theta)','cot(\theta)');
```

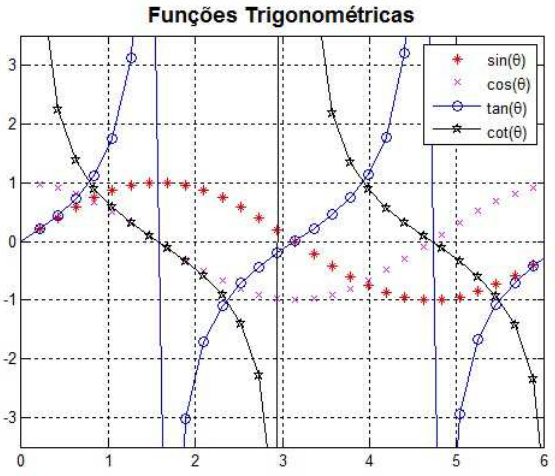

Figura 8.3: Opções de representação gráfica simultâneas

8.4 Exemplo 4 - Comando `subplot`

```
% ex8.4
x = (0:0.21:2*pi);
y = sin(x);
z = cos(x);
a = 2*sin(x).*(cos(x));
b = sin(x)./(cos(x)+eps);
subplot(2,2,1); % reserva uma janela para 4 gráficos
plot(x,y); % sin(x) será visualizada na parte superior esquerda da janela
axis([0 2*pi -1 1]); % intervalos para ambos os eixos x e y, respectivamente
title('sin(x)'); % o titulo épara o último gráfico: y = sin(x)
subplot(2,2,2); % o próximo gráfico sairá na posição acima a direita
plot(x,z);
axis([0 2*pi -1 1]); % intervalos para x e y no segundo gráfico
title('cos(x)');
subplot(2,2,3); % posição do próximo gráfico, abaixo a esquerda
plot(x,a);
axis([0 2*pi -1 1]); % intervalo para os eixos do último gráfico
title('2 sin(x) cos(x)'); % título para o último gráfico
subplot(2,2,4); % posição do próximo gráfico
plot(x,b); % gráfico atual
axis([0 2*pi -20 20]); % eixos para o atual gráfico
```

```
title('sin(x)/cos(x)'); % titulo para o gráfico atual
```

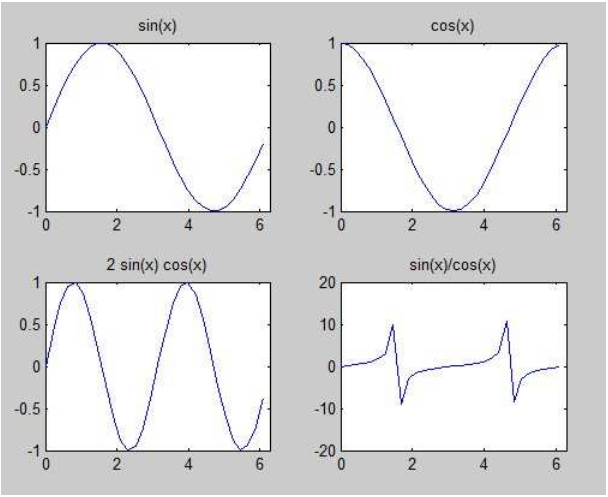

Figura 8.4: Comando subplot

8.5 Exemplo 5 - Comando `pie`

Gráficos em formato de torta ou pizza são criados usando este comando pie.

```
% ex8.5
a = [0.5 1 1.5 2.0 1 0.3];
pie(a,a == max(a));
title('Produção de grãos');
legend('Arroz','Feijão','Soja','Trigo','Milho','Cevada');
```

8.6 Exemplo 6 - Comando `pie3`

A versão tridimensional do gráfico de torta, gerado anteriormente, pode ser obtido usando o comando pie3.

```
% ex8.6
a = [0.5 1 1.5 2.0 1 0.3];
destaque = [0 1 0 1 0 1]; % pedaços em destaque
pie3(a,destaque);% formato pizza tridimensional
colormap hsv; % opção de tonalidade de cor
```

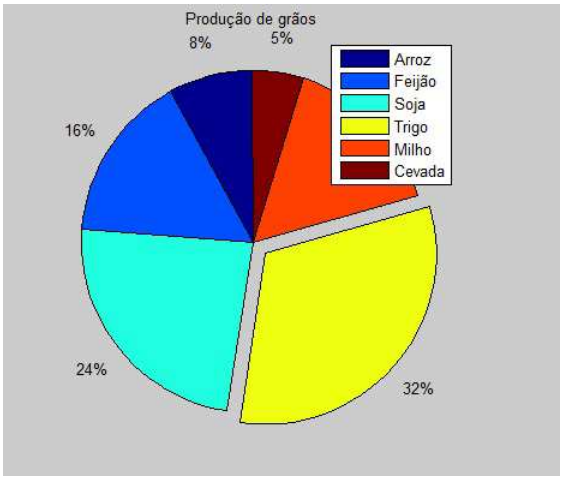

Figura 8.5: Comando pie

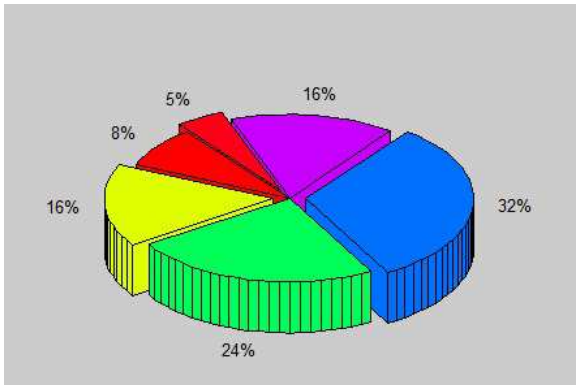

Figura 8.6: Comando pie3

8.7 Exemplo 7 - Comando `bar`

```
% ex8.7
a = [0.5 1 1.5 0.3 1 2];
b = [1960 1970 1980 1990 2000 2010];
bar(b,a); %barra vertical, barh(barra horizontal)
xlabel('Anos de produção');
ylabel('Produção de grãos (em milhões de toneladas)');
axis xy; % ordena os valores do eixo Y em forma ascendente
```

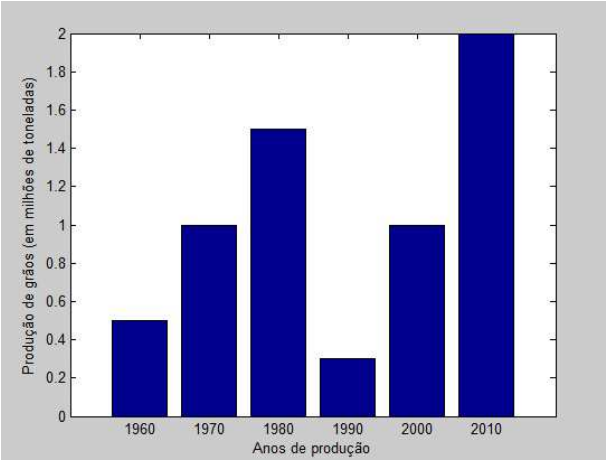

Figura 8.7: Comando bar

8.8 Exemplo 8 - Comando `plot3`

Para traçar gráficos curvilíneos 3D utiliza-se o comando ou função plot3. No exemplo a seguir, a variável t aumenta com o par $(cos(t), sin(t))$ que está dentro de uma circunferência de raio 1. Ao mesmo tempo o par $(cos(t), sin(t))$ está girando em sentido antihorário no plano X-Y e a terceira componente de t está aumentando e se afastando deste plano, logo o gráfico corresponde a uma linha que vai girando e se afastando do plano X-Y, é uma hélice, cujo ponto de partida é o ponto $(0, 1, 0) = (cos(0), sin(0), 0)$.

```
% ex8.8
t = (0:0.1571:10*pi); % [0,10pi] dividido em 200 partes
plot3(sin(t),cos(t),t);
title('H é l i c e : curva paramétrica');
xlabel('X = Sin(t)');
ylabel('Y = cos(t)');
zlabel('Z = t');
```

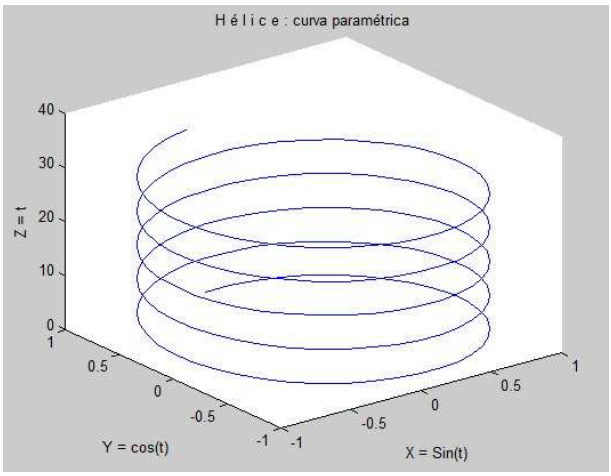

Figura 8.8: Comando plot3

8.9 Exemplo 9 - Gráfico de Superfície

A seguir é apresentado um exemplo com explicações de cada comando gráfico a fim de se obter um gráfico de superfície.

```
% ex8.9
x = -7.5:0.5:7.5; % vetor ou lista x
```

```
y = -7.5:0.5:7.5; % vetor ou lista y
[X,Y] = meshgrid(x,y); % matrizes coordenadas (x,y)
R = sqrt(X.^ 2+Y.^ 2)+eps; % soma-se eps para evitar divisão por zero
Z = sin(R)./R+1;
mesh(X,Y,Z); % gráfico de rede
hold on; % segura as saídas gráficas posteriores na janela atual
pcolor(X,Y,Z); % provoca a aparição de uma malha colorida no domínio
shading interp; % a cor é definida de acordo com a altura
contour(X,Y,Z,20,'k'); % 20 curvas de nível em preto, escolha automática
colorbar; % apresenta uma escala colorida de valores
hold off; % cada comando gráfico será apresentado em uma janela distinta
```

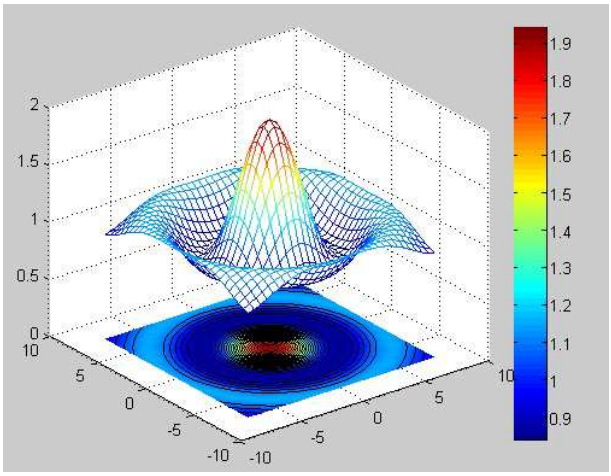

Figura 8.9: Gráfico de Superfície

Cada linha da matriz X é corresponde à lista x e cada coluna da matriz Y corresponde à lista y. Assim a matriz R é o quadrado de cada elemento de X mais o quadrado de cada elemento de Y, componente a componente respectivamente. Semelhantemente segue a definição da matriz Z. O comando mesh grafica os pontos definidos pelas ternas, componentes respectivas de X, Y e Z e em seguida os liga com linhas retas formando uma rede.

8.10 Exemplo 10 - Animação Gráfica

8.10.1 Animação bidimensional

Este exemplo mostra como se constrói uma sequência de gráficos. Estes gráficos são armazenados em uma matriz M, em seguida o comando movie permite mostrar a seqüência armazenada

em M a uma certa taxa de repetição.

```
% ex8.10.1
x = -pi/2:.1:pi/2;
for c=1:20,
    y = sin(2*x+c*pi/10);
    plot(x,y,'LineWidth',18);
    axis([-pi/2 pi/2 -1 1]);
   M(c) = getframe;
end;
movie(M,20,10);
```

A última linha de comando significa que os gráficos armazenados em M são mostrados 10 vezes a uma taxa de repetição de 20 figuras por segundo.

8.10.2 Animação tridimensional

Analogamente à animação bidimensional, temos:

```
% ex8.10.2
x = -pi/2:.1:pi/2;
y = -pi/2:.1:pi/2;
[X,Y] = meshgrid(x,y);
for c=1:20,
    Z = sin(2*X+c*pi/10)+1.5*cos(2*Y+c*pi/10);
    surf(X,Y,Z);
   M(c) = getframe;
end;
movie(M,20);
```

Capítulo 9

Solução de Sistemas de Equações Lineares

Um sistema de equacoes lineares se escreve da forma $Ax = b$ onde A e uma matriz conhecida de dimensoes nxm, geralmente quadrada (m=n), x e o vetor coluna de incognitas e b e um vetor coluna de valores numericos. Caso A (quadrada) seja nao singular, isto e, existe uma matriz denotada A^{-1} tal que $A * A^{-1} = A^{-1} * A = I$, onde I e a matriz identidade. Entao pela teoria o sistema apresenta uma unica solucao x. A matriz identidade e caracterizada por $I_{ii} = 1 e I_{ij} = 0$, se $i \neq j$, isto é, os valores da diagonal sao todos 1 e fora da diagonal os valores sao zero.

Uma boa medida da "qualidade"de uma matriz é dado pelo número de condição. O comando *cond(A)* retorna este número, se o número retornado é grande, indicará que a matriz A é aproximadamente singular, o que é ruim do ponto de vista numérico. A precisão da solução numérica utilizando A será afetada.

Caso $m < n$, o sistema tem mais incógnitas que equações e existem infinitas soluções. Neste caso, procura-se, às vezes, uma solução que satisfaça a condição de minimização da soma das distâncias aos hiperplanos definidos pelas equações (mínimos quadrados). Caso $m > n$, o sistema tem mais equações que incógnitas, então o sistema é sobre determinado e a solução geralmente não existe. O problema de mínimos quadrados não tem solução única (norma Euclidiana):

$$\text{minimize Norm}(A * x = b)$$

Se a solução existe, esta se acha diretamente da seguinte maneira:

$$A * x = b \Longrightarrow x = A^{-1} * b$$

A inversão da matriz A do sistema nem sempre é uma operação barata. Para matrizes grandes ou cheias, prefere-se a utilização de métodos alternativos, como os métodos de decomposição de matrizes tais como LU, QR, *Cholesky* ou métodos iterativos como minres, gmres

e outros (implementados no Matlab). Outro cuidado que deve ser tomado é que os métodos de decomposição não se aplicam a qualquer tipo de matriz. Por exemplo, para o método de *Cholesky*, a matriz do sistema deve ser definida positiva. Uma operação simples como:

```
» A = [3 1 2;0 4 0;-1 -2 -3]; % matriz não definida positiva
» R = chol(A); % método de decomposição de Cholesky
```

Retornará a seguinte mensagem de erro:

```
??? Error using ==> chol
Matrix must be positive definite.
```

Para resolver o sistema utilizando métodos de decomposição, deve-se contar com o algoritmo que calcula a solução baseado nas matrizes decompostas. Já o método iterativo, consiste no cálculo ou atualização, a cada passo, de matrizes e vetores envolvidos no algoritmo que aproxima a solução. Geralmente utilizam-se produtos vetor-vetor, matriz-vetor e calcula-se uma norma no erro da aproximação, tentando atingir uma tolerância preestabelecida.

9.1 Métodos Diretos

A forma mais simples para resolver um sistema, porém a mais cara, consiste em calcular a inversa da matriz do sistema. Outros métodos diretos usam decomposição de matrizes. Algumas destas formas são exemplificadas a seguir.

```
» A = [1 2 1;2 1 2;1 2 2];
» b = [1;0;-1]; %vetor coluna
» B = inv(A); % inverte uma única vez
» C = A^ -1; % forma alternativa para inverter A
» x = C*b; % caso b mude C continua o mesmo
```

9.1.1 Eliminação de Gauss

Utiliza-se a barra invertida para resolver o sistema com uso de eliminação Gaussiana.

```
» x = A\b; % eliminação de Gauss
```

9.1.2 Decomposição LU

```
>> A = [1  2  2;2  1  2;2  2  1];

>> [L,U,P] = lu(A);
```

Esta simples função retornará 3 matrizes 3x3. Uma matriz triangular inferior **L**, uma triangular superior **U** e uma matriz de permutação **P**, de forma que L*U = P*A. Para resolver o sistema, foi utilizado o seguinte método:

```
function x  = solve_LU(L,U,P,b)
    c = P*b;%permutação lado b: P*A*x = L*U*x = P*b
    % substituição triangular inferior: L y = c
    y = zeros(n,1);
    y(1) = c(1,1)/L(1,1);
    for i=1:n,
        soma = 0.0;
        for j=1:(i-1),
            soma = soma+L(i,j)*y(j);
            y(i) = c(i)-soma;
        end
    end
    % substituição triangular inferior U x = y
    x = zeros(n,1);
    x(n,1) = y(n,1)/U(n,n);
    for i=n:-1:1,
        soma = 0.0;
        for j=(i+1):n,
            soma = soma+U(i,j)*x(j);
            x(i) = (y(i)-soma)/U(i,i);
        end
    end
end
```

9.2 Métodos Iterativos

Estes métodos utilizam vários argumentos de entrada como a matriz A, lado b, tolerância (o "default"é $1 * 10^{-06}$), número máximo de iterações, o tamanho da matriz, o chute inicial (o "default"é zero), etc. Os valores de retorno podem ser a solução x, o número efetivo de iterações para atingir a precisão, um número de diagnóstico (flag), o resíduo relativo etc.

9.2.1 Método PCG

O seguinte exemplo usa o método PCG ou gradiente conjugado pré-condicionado que se aplica a uma matriz simétrica e definida positiva.

```
size = 10;
A = rand(size); % matriz aleatória com valores entre 0 e 1
tol = 1.e-03;
for i=1:size,
    b(i,1) = 1; % como exemplo b=1
    for j=i:size,
        A(j,i) = A(i,j); % matriz simétrica
        if j == i
            A(i,i) = 100*A(i,i); % diagonalmente dominante o que
        end % garante que A seja definida positiva
    end
end
[x,flag,res,iter] = pcg(A,b,tol,size);
```

O primeiro teste convergiu em 10 iterações com um resíduo de 2.7609e-015. O número flag = 0 indica que o método convergiu com a desejada tolerância num número de iterações menor ou igual ao "default".

9.2.2 Método Minres

O próximo exemplo utiliza o método *minres* que procura uma solução que minimiza o resíduo. Neste caso a matriz não precisa ser definida positiva, basta ser simétrica.

```
size = 10;
A=rand(size);
tol = 1.e-04;
for i=1:size,
    b(i,1) = 1;
    for j=i:size,
        A(j,i) = A(i,j); % matriz simétrica para o método minres
    end
end
x=zeros(size,1);
[x,flag,res,iter]=minres(A,b,tol,size)
```

O primeiro teste retornou os seguintes resultados. A variável flag = 0 indica que o método convergiu dentro do número de iterações desejado atingindo a tolerância desejada, no caso, em 10 iterações.A norma do resíduo relativo foi:

$$\frac{\|b - A * x\|}{\|b\|} = 3.5881e - 009$$

9.2.3 Comando spy

O comando spy é útil para verificar visualmente o grau de esparsidade de uma matriz. O seguinte algoritmo gera uma matriz ligeiramente esparsa e simétrica que então é visualizada com o comando spy.

```
size = 20;% tamanho da matriz
s = rand(size);% matriz aleatória 20x20 (valores entre 0 e 1)
for i=1:size,
    for j=1:size,
        if j>i
            s(i,j) = s(j,i);% provoca simetria em s
        end
        if s(i,j) < 0.6 & abs(i-j) > 4
            s(i,j) = 0.0;% introdução de esparsidade em s
        end
    end
end
spy(s,'*r');% mostra a matriz em formato gráfico
```

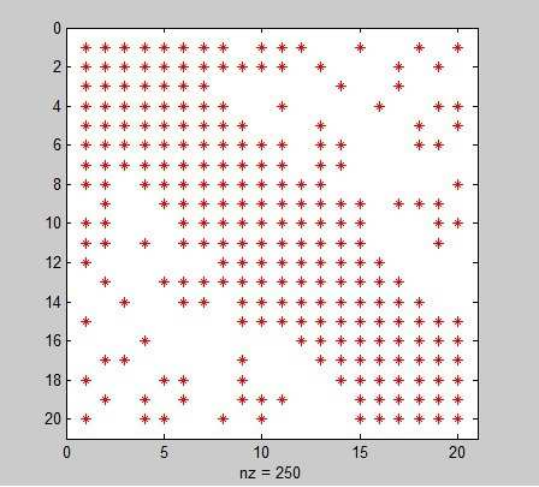

Figura 9.1: Gráfico mostrando os valores não nulos de uma matriz

A última linha preenche os valores não nulos com asterisco e em cor vermelha. O gráfico mostra também o número de zeros *nz* da matriz A.

Capítulo 10

Ajuste de Curvas e Interpolação

Este tipo de ferramentas é útil quando se dispõe de um conjunto descontínuo de dados (valores numéricos pontuais), e se procura traçar uma curva ou superfície (função contínua) que contenha estes pontos.

Existem diferentes curvas que podem ser utilizadas para interpolar estes pontos. Podem ser usados distintos graus para o polinômio que interpola, ou ainda podem ser usados polinômios por partes. Quando se usam polinômios por partes pode acontecer que a curva que interpola os pontos tenha derivada contínua ou não, naqueles pontos conhecidos. Naturalmente surge a questão de que modelo é mais apropriado para interpolar os dados. O analista numérico deve saber decidir sobre esta questão, já que o uso de um modelo ou de outro acarretará uma resposta diferente. Nem sempre de um polinômio de grau maior será obtida uma resposta mais precisa.

Caso o analista conheça a priori o comportamento do seu modelo ele poderá usar esta informação para a escolha do grau de interpolação. Caso contrário ele deverá procurar obter maior quantidade de medições experimentais para obter um comportamento mais apurado na zona de interesse no domínio do problema.

10.1 Interpolação Linear por Partes

O primeiro exemplo interliga pontos discretos no plano por meio de linhas retas, obtendo-se uma curva linear ou de grau um por partes:

```
% ex 10.1
dados1=[ 0 0;1 20;2 60;3 68;4 77;5 110]; % Conjunto de pontos a ser interpolado
disp('Cálculo que interpola este conjunto de pontos em x=2.6 e x=4.9, respectivamente');
y1 = interp1(dados1(:,1), dados1(:,2), 2.6) % valor interpolado no ponto 2.6
y2 = interp1(dados1(:,1), dados1(:,2), 4.9) % valor interpolado no ponto 4.9
```

```
xi=0:.01:5; % lista no intervalo [0,5]
yi = interp1(dados1(:,1), dados1(:,2),xi); % lista yi, interpola a lista xi
plot(dados1(:,1), dados1(:,2),'o',xi,yi); % gráfico da curva interpolada
```

A resposta para os dois valores pontuais é: y1 = 64.8000, y2 = 106.7000. Os valores conhecidos para x estão no intervalo [0,5], e os de y em [0,110], a figura mostra a curva de interpolação.

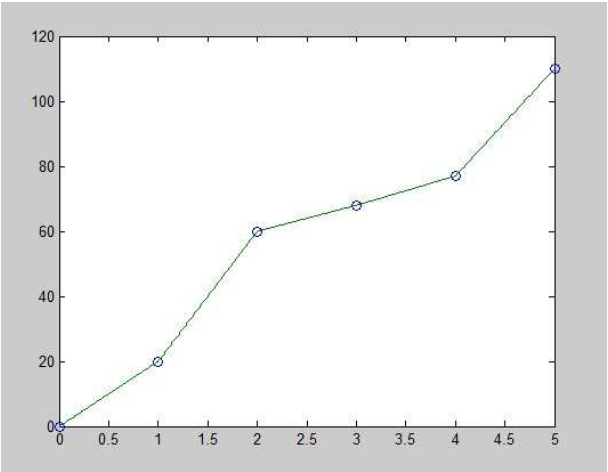

Figura 10.1: Interpolação linear por partes

10.2 Interpolação Superficial

O próximo exemplo corresponde a uma interpolação superficial:

```
% ex 10.2
velocidade = [2000,3000,4000,5000,6000];
tempo = [0 1 2 3 4 5];
temperatura = [ 0, 0, 0, 0, 0; 20, 110, 176, 190, 240; 60, 180, 220, 285, 327; 68, 240, 349,
380, 428; 77, 310, 450, 510, 620; 110, 405, 503, 623, 785];
temp = interp2(velocidade, tempo,temperatura,3800,3.1)
% os próximos passos permitem graficar a curva de interpolação vel = 2000:100:6000; % 40
pontos
t = 0:0.125:5; % 40 pontos
[X,Y] = meshgrid(vel,t);
Z = interp2(velocidade,tempo,temperatura,X,Y);
```

```
mesh(X,Y,Z);
xlabel('E I X O X');
ylabel('E I X O Y');
zlabel('E I X O Z');
```

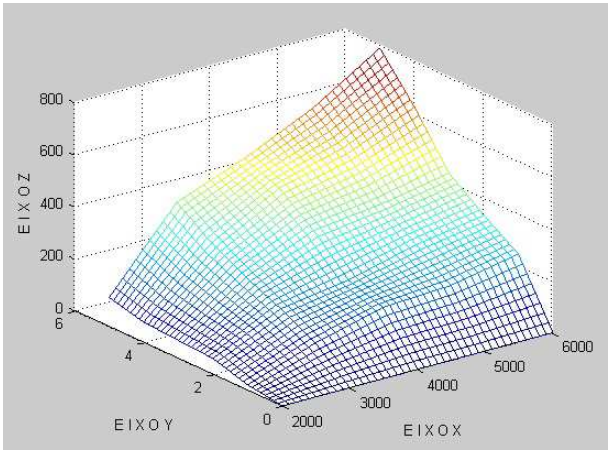

Figura 10.2: Superfície interpolante da matriz temperatura

10.3 Comando `spline`

Este tipo de interpolação utiliza polinômios cúbicos para, no caso plano, unir cada dois pontos com um polinômio, e cada 4 pontos no espaço com uma superfície polinomial. Ainda a primeira e segunda derivada nestes pontos são contínuas:

```
% ex 10.3
x = [0 1 2 3 4 5];
y = [0 20 60 68 77 110];
temp1 = spline(x,y,2.6);
temp2 = spline(x,y,[2.6 4.9]);
z = [.5 1.5 2.5 3.5 4.5];
temps = spline(x,y,z);
temp1 % resultado
temp2 % resultado
temps % vetor interpolante da lista z
xi = 0:0.2:5; %25 pontos
yi = spline(x,y,xi); % interpolação da lista xi
```

```
plot(x,y,'o-',xi,yi); % curva interpolante
xlabel('Tempo[s]');
ylabel('Graus Farenheit');
```

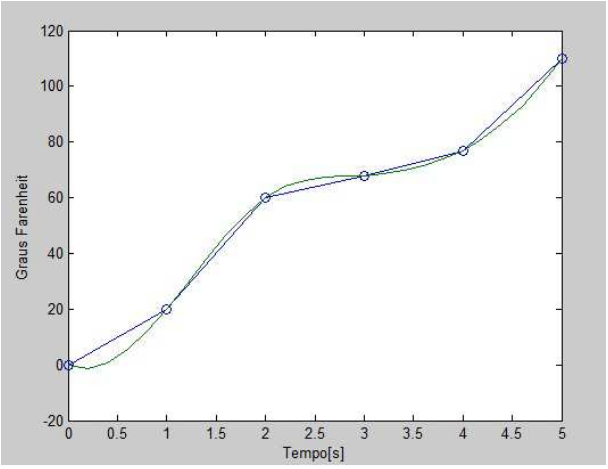

Figura 10.3: Interpolação utilizando spline

10.4 Comando `polyfit`

Este comando acha os coeficientes de um único polinômio que interpola os dados no sentido dos mínimos quadrados, isto é, não necessariamente o polinômio passa pelos pontos, mas a soma das distâncias dos pontos ao polinômio é minimizada. O grau do polinômio deve ser especificado. O exercício seguinte utiliza este comando. O resultado deste comando é um vetor de comprimento grau+1 contendo os coeficientes em ordem descendente de potências xn , n = grau -1,...,2,1,0 . O aumento do grau do polinômio nem sempre garante uma melhor aproximação à resposta procurada.

```
% ex 10.4
x = [0 1 2 3 4 5];
y = [0 20 60 68 77 110];
grau = 1; % grau do polinômio a ajustar
coef = polyfit(x,y,grau);
ybest = polyval(coef,x); % para verificar a qualidade da aproximação
axis([-1,6,-20,120]);
plot(x,ybest,x,y, 'o' );% compara graficamente os valores dados com os obtidos por ajuste
title ('Ajuste polinomial: grau 1');
```

```
xlabel('X');
ylabel('Y');
grid;
```

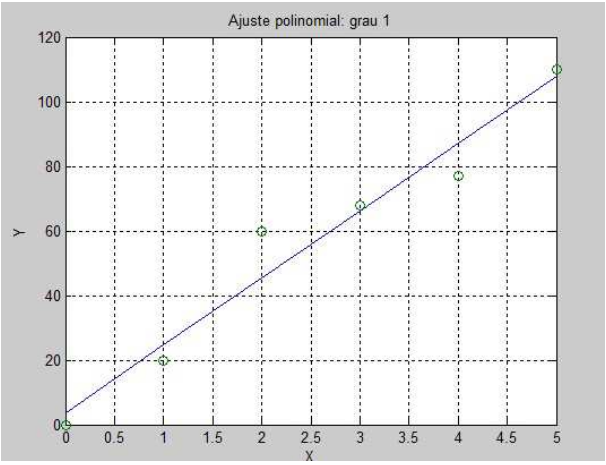

Figura 10.4: Ajuste polinomial de grau 1

Capítulo 11

Leitura e Escrita de Arquivos de Dados

Pode-se ler um arquivo de dados que contenha apenas números usando o comando *load*. Um arquivo com qualquer extensão, por exemplo, "matriz.dat", que contenha números em formato de vetor ou matriz pode ser lido com o comando *load*. O Matlab cria uma variável de nome *matriz* como sendo vetor ou matriz da mesma dimensão daquela que foi lida.

Por exemplo, se o conteúdo do arquivo "matriz.dat"é:

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

Depois de lido o arquivo com o comando *load*, cria-se a variável matriz como sendo uma matriz 2x4:

```
» load matriz.dat;
» matriz
matriz =

 1   2   3   4
 5   6   7   8

» size(matriz) % dimensões da variável matriz
ans =

 2   4
```

O arquivo não deve conter letras ou caracteres entre aspas. Os números devem estar dispostos em forma de matriz. Para salvar as variáveis ou dados em geral usa-se o comando *save*. Exemplo:

```
» clear;
» x = [0 1 2 3 4 5];
» y = sqrt(2)*eye(3);
» z = rand(2);
» save dados x y z; % salvará os dados x,y e z dentro do arquivo 'dados.mat'
» save dados2.txt x y z -ascii; % salva em formato txt, 8 digitos
» save dados3.txt x y z -ascii -double; % salva em formato txt com 16 digitos
```

Para se trabalhar com leitura e escrita de dados existe a possibilidade de se utilizar comandos semelantes aos da linguagem C, tais como *fopen*, *fscanf*, *fclose*, cujo entendimento é mais complexo, e que necessitam mais experiência em programação para serem entendidos. Mais informações são encontradas através dos mecanismos de ajuda fornecidos pelo Matlab.

Capítulo 12

Análise Polinomial

Para definir um polinômio basta escrever a lista referente aos coeficientes deste polinômio. Por exemplo para trabalhar com o polinômio $f(x) = 3x^4 - 0.5x^3 + x - 5.2$ no Matlab defina-se a lista:

» f = [3,-0.5, 0, 1,-5.2];

Cada coeficiente deve aparecer em ordem decrescente a partir da maior potencia não nula deste polinômio, e quando a potencia não existe deve-se preencher com zero, como é o caso do termo x^2 . Para avaliar este polinômio f usa-se o comando *polyval* tanto para valor simples como para listas.

```
% ex 12.1
f = [3,-0.5,0,1,-5.2]; x = 1; polyval(f,x) % valor f(1)
x = [0 1 1.1 1.2]; % redefinição de x
y = polyval(f,x) % lista de valores f(xi)correspondente a cada valor da lista x
g = [1 0 -3 -1 2.4]; % polinômio g(x) = x^4 - 3x^2 - x + 2.4
m = conv(f,g) % produto dos polinômios f e g
[q,r] = deconv(f,g); % divisão polinomial f/g
rts = roots(f) % raízes ou zeros do polinômio f
f = poly(r) % coeficientes do polinômio cujas raízes são os elementos de r
f = [1 -2 3];
df = polyder(f) % coeficientes do polinômio 2x-2 derivada de f(x) = x^2 - 2x + 3
x = -5:0.1:7;
yf = polyval(f,x) % valores de f relativos a x
ydf = polyval(df,x) % valores da derivada de f em x
plot(x,yf,x,ydf) % gráfico de f e da sua derivada
```

legend('f','df');
title('Função e a sua derivada: f = x2-2x+3 ; df = 2x-2');

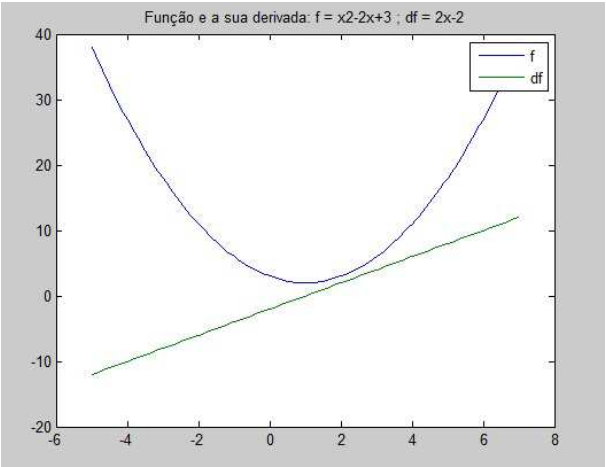

Figura 12.1: Curvas contínuas de um polinômio e a sua derivada

A variável *m* é uma lista de coeficientes do polinômio produto dos polinômios *f* e *g*, *q* e *r* são, respectivamente, a lista de coeficientes do polinômio quociente *f/g* e resto desta divisão. A lista *rts* contém as raízes do polinômio *f*. O comando *poly(r)* acha os coeficientes do polinômio cujas raízes são os valores de r. O comando *polyder(f)* retorna os coeficientes do polinômio derivado de *f*.

Existe uma grande variedade de comandos para trabalhar com polinômios, mas eles são para aplicações mais específicas (execute *lookfor polyn*). Mais duas aplicações como exemplos são dadas a seguir.

- `polyint` : integra polinômios analiticamente, retorna a lista de coeficientes da antiderivada de um polinômio tal que f(0) = 0. Por exemplo, polyint(f,3) calcula antiderivada de f tal que f(0) = 3.
- `poly2str` : converte uma lista em uma seqüência de caracteres em formato de polinômio. Por exemplo: poly2str([1 0 2],'s') retorna o texto (string) 's2 + 2'.

Capítulo 13

Integração e Diferenciação

Existem métodos que trabalham com aproximação numérica e outros com funções analíticas. É possível no Matlab trabalhar com algoritmos numéricos e algumas vezes também com métodos simbólicos. No caso de integração e diferenciação existem comandos para trabalhar com ambos os métodos.

13.1 Integração

No Matlab existe o comando *int* que calcula a integral indefinida de uma função analítica com respeito a sua variável simbólica. Primeiro deve-se definir a variável independente como sendo simbólica, veja exemplo a seguir.

```
>> syms x x1 alpha u t;%define estas variáveis como simbólicas

>> int(1/(1+x^2)) %antiderivada de 1/(1+x2)

ans =

atan(x)

>> int(sin(alpha*u),alpha) % integra com respeito a alpha e não u

ans =

-cos(alpha*u)/u
```

```
>> int(x1*log(1+x1),0,1) % integra no intervalo [0,1]

ans =

1/4

>> int(4*x*t,x,2,sin(t)) % integra no intervalo [2, sin(t)]

ans =

-2*t*(cos(t)^2 + 3)

>> int([exp(t),exp(alpha*t)]) % integra o par c/r a variável comum t

ans =

[ exp(t), exp(alpha*t)/alpha]

>> A = [cos(x*t),sin(x*t); -sin(x*t),cos(x*t)]%dois pares de integrais

A =

[  cos(t*x), sin(t*x)]
[ -sin(t*x), cos(t*x)]
```

A integração numérica é efetuada com os comandos *quad*, *quadl*, sendo precisão 10^{-6} a precisão "default". Utilizam respectivamente quadratura de Simpson e Lobatto (adaptativa).Para especificar a precisão utiliza-se o argumento tolerância.

$$\int f(x)dx$$

Exemplo:

```
» a = 0;
» b = 1;
```

O arquivo func.m define a função a ser integrada, como dado a seguir.

```
% arquivo func.m
```

function f = func(x)

f = x.^ 2;

As formas a seguir são válidas e levam ao mesmo resultado

» quad('x.*x',0,1); % integra x2 no intervalo [0,1]
» quad('func',a,b); % integra x2 no intervalo [0,1]
» quad(@func,a,b); % integra x2 no intervalo [0,1]

O resultado é 0.3333.

» quad(@func,a,b,1.e-14); %exige-se uma precisão 10^{-14} na integração

A integração de Lobatto implementa um método de alta ordem e uma tolerância absoluta do erro. Para integração dupla utilize-se o comando *dblquad*, que integra numericamente, por exemplo:

» f = 'x.*y';

» dblquad(f,0,1,0,1)

ans =

0.2500

Outra forma válida é:

» dblquad(inline('x.*y'),0,1,0,1);

Em qualquer caso o resultado é o mesmo, isto é, 0.2500.

13.2 Diferenciação

A derivada de uma função f num ponto x_0 é definida como o seguinte limite:

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

ou equivalentemente quando $x \to x_0, onde \Delta x = x_0 - x$:

$$\frac{df}{dx} = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

Para trabalhar tanto simbólica como numericamente utiliza-se o comando *diff*.
Exemplos:

```
>> syms x; % declara x simbólico
>> df = diff('x^ 2',x); % deriva simbolicamente df
>> df =2*x;
>> diff('cos(a*x)',x)

ans =

    -a*sin(a*x)
```

Observe que a foi tratada como constante. Para permitir que cos(a*x) seja derivável com respeito a a, deve-se primeiramente declarar a como sendo simbólico.

```
>> syms a;
>> h = 'cos(a*x)';
>> dfa = diff(h,a)

ans =

    -x*sin(a*x)
```

Numericamente a diferenciação calcula-se como a diferença entre dois pontos adjacentes quando estes se aproximam. Para tal utiliza-se o comando diff calculando diferenças entre valores adjacentes. O próximo exercício exemplifica o cálculo da derivada numérica e compara-se com a derivada analítica.

```
x = (-1:1/15:1); % 30 divisões do intervalo [-1,1]
f = x.^5 - 3*x.^4 - 11*x.^3 + 27*x.^2 + 10*x - 24;
df_an = 5*x.^4 - 12*x.^3 - 33*x.^2 + 54*x + 10; % derivada analítica
df = diff(f)./diff(x); % derivada numérica de f como aproximação à
                       % tangente a curva f em pontos adjacentes
xd = x(2:length(x));  % Dado que o vetor diferença df tem um elemento a
```

```
                                % menos e as dimensões de x e df devem coincidir
plot(x,f,'r-',xd,df,'b-',x,df_an,'k-')
legend('f(x)','{df/dx} numérico','{df/dx} analítico');
xlabel('x');
ylabel('f(x) e {df/dx}');
title('Polinômio e sua derivada');
```

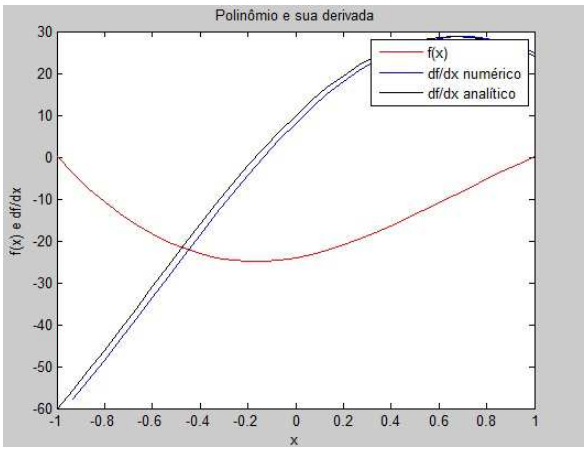

Figura 13.1: Derivada numérica: 30 divisões do intervalo [-1,1]

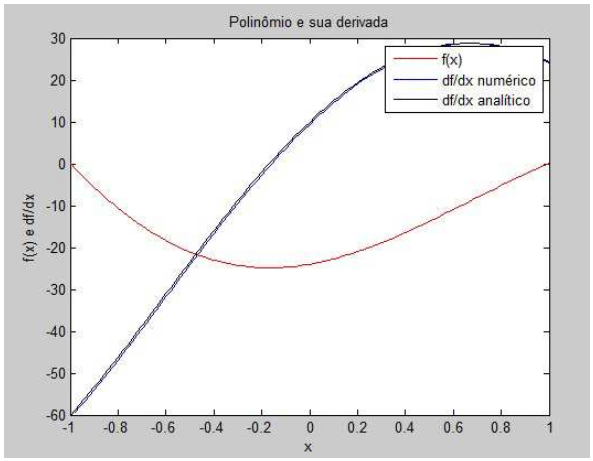

Figura 13.2: Derivada numérica: 100 divisões do intervalo [-1,1]

A quarta linha do arquivo acima mostra como usar o comando *diff* para calcular numericamente a derivada. O gráfico mostra as curvas do polinômio f e as derivadas analítica e numérica. A precisão do resultado numérico depende do número de divisões considerados ou de quão próximo estão os pontos, no caso da primeira figura, foram tomados 30 divisões do intervalo [-1,1]. Já na outra figura, é mostrada a mesma situação anterior, mas agora foram consideradas 100 divisões no intervalo. Percebe-se claramente a diferença de precisão entre os casos considerados.

Capítulo 14

Equações Diferenciais Ordinárias

As funções *ode45, ode23, ode113, ode15s, ode23s, ode23t, ode23tb* permitem resolver problemas a valores iniciais para equações diferenciais ordinárias (uma variável independente) por vários métodos.

Os comandos *ode23* e *ode45* são baseados no método de Runge-Kuta. Ambos possuem os mesmos tipos de argumentos, *ode23* é usado para equações de segunda e terceira ordem, *ode45* é utilizado para equações de quarta e quinta ordem.

O exemplo a seguir apresenta uma equação diferencial de primeira ordem cuja solução analítica é conhecida. A equação diferencial é uma de primeira ordem (a derivada de maior ordem é 1):

$$y' = g_1(x, y) = 3x^2, [2, 4]$$
Condição Inicial: $y(2) = 0.5$
Solução Analítica: $y = x^3 - 7.5$

Solução usando o Matlab: primeiro defina a function $d = y' = 3x^2$:

```
function dy = g1(x,y)
dy = 3*x^ 2
```

O arquivo contendo a resolução pelo Matlab é dado por:

```
% Este exemplo resolve uma equação diferencial com condições de contorno

[x,num_y] = ode23('g1',[2,4],0.5);
```

```
anl_y = x.^3 - 7.5; % solução analítica
plot(x,num_y,'r:+',x,anl_y,'b-'); % gráfico das soluções analítica pelo métod
title('Solução ODE');
xlabel('X');
ylabel('y = f(x)');
legend( 'Runge Kuta','Solução analítica');
grid;
anl_y-num_y % erro absoluto cometido
```

No comando *ode23*, [2,4] é o intervalo onde procura-se a solução, 0.5 é a condição inicial no ponto $x = 2$. A saída do comando corresponde a um conjunto de coordenadas, os quais representam os pontos, no exemplo atual (x, num_y), da função $y = f(x)$ solução numérica da equação diferencial. O número de pontos a determinar é calculado pelo método do Matlab. Pode ser usado mais um parâmetro referente à tolerância que está relacionada com o tamanho do passo. A tolerância "default"é 0.001 para *ode23* e 0.000001 para ode45. O erro ponto a ponto cometido pelas soluções analítica e numérica, como é mostrado pela diferença $anl_y - num_y$, é da ordem 10^{-13} , conforme verificado pela superposição dos pontos à curva na figura a seguir.

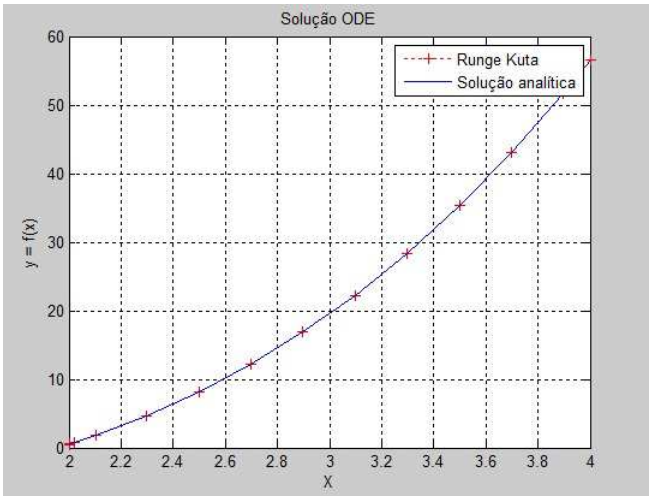

Figura 14.1: Gráfico das soluções exata e numérica da equação $y' = 3x^2$

O exemplo anterior também pode ser facilmente resolvido através do comando *dsolve*, cuja resolução é analítica e simbólica:

```
>> y = dsolve('Dy = 3*x^2','y(2) = 0.5','x')
y =
    x^3 - 15/2
```

Capítulo 15

Decomposição e Fatoração de Matrizes

Existem muitos métodos e técnicas numéricas que utilizam decomposição e fatoração de matrizes, como por exemplo, a resolução de sistemas lineares. A decomposição LU serve, por exemplo, para determinar o determinante de uma matriz grande, ou a matriz inversa.

15.1 Fatoração Triangular - LU

Como já vimos, este método decompõe uma matriz quadrada em duas matrizes triangulares, uma inferior (L:lower) e uma superior (U:upper), mas uma matriz de permutação que corresponde à matriz identidade onde foram permutadas linhas ou colunas. Assim o comando *lu* efetua esta decomposição.

» [L,U,P] = lu(A); % A é uma matriz quadrada

De tal forma que: L * U = P * A

15.2 Decomposição - QR

Nesta decomposição Q é uma matriz ortonormal e R uma matriz triangular superior. A propriedade de Q ser ortonormal indica que:

$$Q^T * Q = I$$

Isto quer dizer que Q e a sua transposta Q^T sao inversas. As linhas da matriz Q possuem a seguinte propriedade: denote por q_i a linha i de Q, entao $q_i * q_j^T = \delta_{ij}$, isto é, o produto escalar de duas linhas distintas é nulo(vetores ortogonais) e o produto escalar de uma linha por si mesma é 1 (vetor de norma 1). O mesmo aplica-se as colunas de Q.

Este método é usado para resolver um sistema indeterminado $Ax = B$ resolvendo-se $Rx = Q^T B$, onde $A = QR$. No Matlab esta decomposição é obtida assim:

» [Q,R] = qr(A); % fatoração QR

15.3 Decomposição em Valores Singulares - SVD

Permite a fatorização de uma matriz A em matrizes ortogonais S e V e uma matriz diagonal D. O comando é:

» [S,D,V] = svd(A); % fatoração SVD

De forma que $A = S * D * V^T$. Os valores da matriz diagonal S são chamados de valores singulares de onde vem o nome. O número de valores singulares não nulos corresponde ao "rank"da matriz, isto é o número de linhas ou colunas linearmente independentes da matriz A. Este mesmo valor pode ser pesquisado com o comando *rank*.

15.4 Autovalores e Autovetores

Entre outras aplicacoes o problema de autovalores encontra utilidade na transmissao de sinais de comunicacao. Um autovalor λ e um autovetor x estao associados a uma matriz A da seguinte maneira:

$$Ax = \lambda x$$

Logo a determinacao destes esta associada a resolucao de um sistema linear de equacoes. A primeira vista uma solucao trivial deste sistema e $x = 0, \lambda = 0$. Mas podem existir valores nao nulos que tambem resolvem o sistema. No Matlab os autovalores e autovetoers sao encontrados com os comandos:

» lambda = eig(A);% lambda: vetor contendo os autovalores de A

» [V,D] = eig(A); % autovalores e autovetores de A

Neste último comando D é uma matriz diagonal contendo os autovalores de A e V uma matriz cujas colunas são os autovetores de A tal que:

$$A * V = V * D$$

Ou em termos de vetores:

$$A * v_i = v_i * d_{ii}, i = 1, 2, ..., n$$

De outra forma no Matlab $A * V(:, i)$ deve ser igual a $V(:, i) * D(i, i)$,para $i = 1, 2, ..., n$.

Capítulo 16

Considerações Finais

Além do abordado nessa apostila, o Matlab apresenta ainda muitas outras funções, além de que para problemas mais específicos existem vários toolboxes com outras ferramentas.

As funções e os métodos abordados nessa apostila foram escolhidos para que houvesse a melhor compreensão das ferramentas gráficas, numéricas, analíticas e de implementação de algoritmos que o Matlab possui, no tempo e espaço disposto.

Quem entender o conteúdo presente neste material já possui capacidade para trabalhar com as principais ferramentas do Matlab e autonomia suficiente para continuar o estudo acerca deste ambiente.

Para concluir, vale a pena lembrar que este estudo pretende apresentar o Matlab ao pesquisador como uma ferramenta de ajuda científica, e que um estudo apurado dependerá necessariamente da dedicação à pesquisa na área de atuação de cada usuário.

Referências Bibliográficas

[1] C. M. A. A Bravo and E. L. de Albuquerque, *Intrudução ao MATLAB* , Centro Nacional de Processamento de Alto Desempenho em São Paulo, Universidade Estadual de Campinas.

[2] L. A. Farina and M. S. Posser, *MATLAB: Ferramenta matemética para Engenharia*, Laboratório de Controle e Integração de Processos, Universidade Federal do Rio Grando do Sul.

[3] D. Hanselman and B. Littlefield, *Versão de Estudante, Curso de Matlab 5*, Makron Books do Brasil, São Paulo - Brasil, 1999.