Centro Nacional de Processamento de Alto Desempenho em São Paulo
Universidade Estadual de Campinas

Introdução ao MATLAB

Sumário

1 Introdução

A primeira versão do Matlab escrita no final da década de 70 nas Universidades de Stanford e do Novo México era destinada a cursos de teoria matricial, álgebra linear e análise numérica.

No desenvolvimento dos pacotes EISPAC e LINPACK de sub-rotinas em código FORTRAN para manipulação de matrizes, pretendia-se que os alunos pudessem utilizar esses pacotes sem a necessidade de escrever programas em Fortran.

Atualmente a capacidade do Matlab se estende além do "Laboratório de Matrizes"' original. O Matlab é um sistema interativo e uma linguagem de programação para computação técnica e científica em geral.

Ele integra a capacidade de fazer cálculos, visualização gráfica e programação em um ambiente fácil de usar, em que problemas e soluções são expressos em uma linguagem matemática familiar.

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.

O Matlab é um sistema interativo cujo elemento de dados básico é uma matriz que não requer dimensionamento. Isso permite solucionar muitos problemas computacionais, como aqueles que envolvem formulações matriciais ou vetoriais em uma fração de tempo bem menor daquele ocupado para escrever um programa em uma linguagem como C ou Fortran, veja [1] e [2].

1.1 Toolboxes

O Matlab é tanto um ambiente quanto uma linguagem de programação [3], 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.

Extraída e adaptada da Introdução do texto ”Guia do Usuário” , [2].

2 Ajuda no Matlab

O Matlab pode ser utilizado tanto no ambiente Unix como no Windows. Após chamar o *matlab* aparecem três opções de ajuda a serem acessadas a partir do prompt (>>) ou linha de comandos do Matlab, a saber, *helpwin, helpdesk* e *demo*.

O comando *demo* permite o acesso a uma janela indexada com várias ferramentas que o Matlab implementa. Escolhe-se um tópico e sub tópico e clica-se sobre o botão *Run*

o qual chama uma outra janela que mostra uma série de comandos usados pelo Matlab com a correspondente explicação acompanhada por uma visualização gráfica e várias opções/botões de execução como *start*, *reset, play, next, prev, info*, etc.

Com o comando *helpdesk* obtém-se acesso a uma ajuda online com informações completas sobre o Matlab em todos seus aspectos. Dentre estes também há itens onde são dados exemplos com explicações do potencial do Matlab e dos comandos que são utilizados, apresenta-se também uma visualização gráfica quando esta compete. Alguns tópicos são apresentados no formato conteúdo-índice. Não é preciso estar conectado a Internet para usar esse sistema. O comando *help docopt* ensina a configurar o Web Browser para ter acesso a esta ajuda online com o comando *helpdesk.*

O comando *Helpwin* apresenta um conjunto de diretórios e um título que reflete o tipo de ferramentas nele contidas. Clicando sobre qualquer um destes itens aparece um glossário de comandos com a correspondente definição da função que lhe é designada. Clicando ainda sobre uma destas definições tem-se acesso a uma explicação sucinta dos usos desta função assim como os argumentos de entrada e saída e de funções afins com ela. Caso existam outras funções que este comando possa executar, serão mostrados uma série de arquivos com o mesmo nome, mas com funções distintas (funções sobrecarregadas). Para ter acesso a uma ajuda sobre estes comandos basta digitar qualquer das linhas apresentadas na linha de comandos do Matlab, com ou sem a extensão '.m'.

Como será visto depois a extensão '.m' é reservada para designar um arquivo executável dentro do Matlab. Assim, as funções ou métodos usados pelo Matlab, estão implementados dentro de arquivos com extensão '.m', e são fáceis de acessar e entender, pois estão escritos em uma linguagem matemática familiar.

No Matlab os comandos e variáveis são caso sensitivo, isto é, as variáveis cujos nomes são *var* e *vaR* são distintas dentro do ambiente Matlab.

Para ter acesso direto a explicação de uma determinada função do Matlab cujo nome é conhecido basta digitar *help* e o nome do comando, por exemplo, para acessar a explicação da função 'power', elevação à potencia de um número, digite (todos os comandos ou funções próprias do matlab devem ser digitadas em letra minúscula) :

```
>> help power
```

A saída resposta do Matlab será:

```
.^ Array power.
Z = X.Y denotes element-by-element powers. X and Y
must  have  the  same dimensions unless one is a scalar.
A scalar can operate into anything.
C = POWER(A,B) is called for the syntax 'A . B' when A or B
is an object.
See also MPOWER.
Overloaded methods
help demtseries/power.m
help fints/power.m
help sym/power.m
```

A função *mpower* é afim com a função *power***,** pois ela calcula potências numéricas. A função *mpower* permite elevar uma matriz a uma potência escalar, isto representa o produto de uma matriz por si mesma, coisa que não é possível calcular diretamente com *power*. Por exemplo, para A definida por:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \tag{1}$$

```
>> B = power(A,2)
```

retorna o resultado

$$B = \begin{bmatrix} 1 & 4 \\ 9 & 16 \end{bmatrix} \tag{2}$$

Isto representa o quadrado de cada entrada de A. Ao passo que o comando

```
>> C = mpower(A,2)
```

devolve o resultado

$$C = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix} \tag{3}$$

que equivale ao produto A vezes A.

O arquivo *power.m* no diretório local matlab6/toolbox/matlab/ops é acionado quando o comando *help power* é executado. O mencionado arquivo, no caso específico, só contém linhas de comentário sem argumentos de entrada. Já o arquivo sobrecarregado *power.m*, no diretório local matlab6/toolbox/finance/ findemos/@demtseries, calcula a potência de um número elemento a elemento e cujos argumentos de entrada são uma matriz e um número. Qual destes arquivos será acionado dependerá, obviamente, do número e do tipo de argumentos.

Outro comando de ajuda do Matlab é o *lookfor* que pesquisa dentre as funções do Matlab aquelas que contém uma certa palavra chave. Por exemplo

```
>> lookfor sound
```

apresenta a seguinte saída:

```
BEEP Produce beep sound.
AUREAD Read NeXT/SUN (".au") sound file.
AUWRITE Write NeXT/SUN (".au") sound file.
SAXIS Sound axis scaling.
SOUND Play vector as sound.
SOUNDSC Autoscale and play vector as sound.
WAVPLAY Play sound using Windows audio output device.
WAVREAD Read Microsoft WAVE (".wav") sound file.
WAVRECORD Record sound using Windows audio input device.
WAVWRITE Write Microsoft WAVE (".wav") sound file.
SOUNDVIEW View and play sound with replay button.
XPSOUND Demonstrate MATLAB's sound capability.
NNSOUND Turn Neural Network Design sounds on and off.
PLAYSND Implementation for SOUND.
```

Sendo que cada palavra inicial, ressaltada em letra maiúscula, corresponde a uma função definida dentro do Matlab.

O comando de ajuda *which* permite visualizar o caminho atual de uma função qualquer com extensão '.m'. Por exemplo:

```
>> which power -all
```

Apresenta a seguinte saída:

```
power is a built-in function.
```

```
/usr/matlab6/toolbox/finance/findemos/@demtseries/power.m %
demtseries method
/usr/matlab6/toolbox/ftseries/ftseries/@fints/power.m % fints
method
/usr/matlab6/toolbox/symbolic/@sym/power.m % sym method
/usr/matlab6/toolbox/matlab/ops/power.m % Shadowed
```

Estes são os caminhos dos atuais métodos sobrecarregados com o nome *power*. O símbolo % significa que o que está a esquerda nessa linha é comentário, equivale a ! em Fortran e a // em C++.

Informações adicionais do Matlab na Web, tais como fone/fax/e-mails, podem ser obtidas com o comando *info.*

2.1 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.

2.2 Comandos do UNIX/DOS

É possível executar comandos do UNIX ou do DOS dentro do Matlab. Quando o comando é simples basta digitá-lo como se estivesse numa console do UNIX ou uma janela DOS no Windows. Por exemplo, ls, dir, cd, etc. Quando o comando é mais complexo ou com um maior número de parâmetros proceda como é dado a seguir.

Por exemplo, quer-se utilizar o editor *pico* no ambiente Linux:

```
>> unix 'pico meu_programa.m' + <ENTER>
```

quando sair deste editor retorna-se ao prompt '>>' do Matlab.

Por exemplo, quer-se apagar um arquivo dentro do ambiente Windows:

```
>> dos 'del arquivo.dat' + <ENTER>
```

2.3 Limitações 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.

3 Manipulação de matrizes, vetores e escalares

Para o Matlab todas as variáveis constituem-se em matrizes. Por exemplo, para definir texto procede-se da seguinte forma:

```
>> texto='Esta é uma frase exemplo';
```

Este procedimento por si só define a variável “texto” como um vetor de 24 componentes ou equivalentemente uma matriz linha de tamanho 1x24. O comando:

```
>> texto + <ENTER>
```

provoca a saída:

```
texto =

Esta é uma frase exemplo
```

ao passo do que o comando

```
>> texto(24) + <ENTER>
```

retorna a saída

```
ans =
o
```

que corresponde a entrada ou letra na posição número 24, isto é a última letra da frase. A mesma saída é obtida com a variável:

```
>> texto(1,24) + <ENTER>
ans =
o
```

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. 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). O símbolo ";" inibe a resposta de saída, exceto quando o comando utilizado seja para mostrar por exemplo gráficos ou mensagens.

- Cálculos elementares: exemplo

```
>> bananas = 4;
>> laranjas = 5;
>> frutas = laranjas + bananas;
>> preco_laranja = 2.5;
>> preco_banana = 2;
>>
preco_medio=(laranja*preco_laranja+banana*preco_banana)/...
frutas;
```

- Comando *who*

Mostra as variáveis ativas do ambiente (atualmente definidas). Caso foram digitados todos os exemplos desde o inicio da seção 3 o comando *who* daria a seguinte saída:

```
>> who
Your variables are:
ans          frutas           preco_banana      preco_medio
bananas      laranjas          preco_laranja       texto
```

Para ter acesso ao valor de uma determinada variável basta digitar o seu nome no prompt e dar <*ENTER*>. Caso a variável digitada não tenha sido definida ou não exista dentro do ambiente será emitido uma mensagem de erro. Sempre que uma operação ou comando não seja válido dentro do Matlab será visualizada uma mensagem de erro, da mesma forma se não existe memória suficiente para executar uma determinada operação.

- Comando *whos*

Mostra as variáveis do ambiente mais o tamanho e tipo:

```
>> whos
```

```
Name                Size        Bytes Class
ans                  1x1        2 char array
bananas              1x1        8 double array
frutas               1x1        8 double array
laranjas             1x1        8 double array
preco_banana         1x1        8 double array
preco_laranja        1x1        8 double array
preco_medio          1x1        8 double array
texto               1x24        48 char array
```

Repare que uma variável simples como bananas é considerada como um arranjo de dimensões 1x1 (double).

- Comando *clear*

Permite apagar variáveis do ambiente.

```
>> clear bananas
```

Apaga a variável bananas do ambiente, o comando who acusará a ausência desta variável.

```
>> clear prec*
```

Apagara todas as variáveis que se iniciam com as letras prec

```
>> clear all
```

ou simplesmente *clear* apaga todas as variáveis do ambiente.

3.1 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 ";" da seguinte maneira:

```
>> A = [ 2 1 1;1 2 1;1 1 2] + <ENTER>
A =
2   1   1
1   2   1
1   1   2
```

É válido separar as entradas de cada linha por vírgula ou com um espaço. Se uma linha é longa pode-se continuar a linha de uma matriz na linha seguinte do prompt, usando reticências,

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

Um outro exemplo onde cada <ENTER> determina uma linha da matriz:

```
C = [7 1 2 + <ENTER>
1 7 2 + <ENTER>
1 2 7] + <ENTER>
C =
3   1   2
2   3   1
1   2   3
```

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 + <ENTER>
ans =
    9  7  8
    8  9  7
    7  8  9
```

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.

```
>> MAT = A.*C+ <ENTER>
ans =
    6  1  2
    2  6  1
    1  2  6
```

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 + <ENTER>
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 % o mesmo que A.*5 + <ENTER>
ans =
    10  5  5
    5  10  5
    5   5 10
>> C/A % é igual C vezes inversa de A ou A*inv(C) + <ENTER>
```

```
ans =
     1.5  -0.5   0.5
     0.5   1.5  -0.5
    -0.5   0.5   1.5
>> C./A % é a matriz definida por C(i,j)/A(i,j)
ans =
    1.5   1   2
    2   1.5   1
    1    2   1.5
```

Outras operações ainda são permitidas:

```
>> z=[20 21 22];
>> y=[A;z] % agrega uma quarta linha + <ENTER>
ans =
    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 + <ENTER>
ans =
    2    1    1
    1    2    0
    1    1    2
```

Deve-se ter um cuidado especial quando se trabalha com indexação matricial, pois uma atribuição como:

```
>> A(4,4) = 100
ans =
    2    1   1    0
    1    2   1    0
    1    1   2    0
    0    0   0  100
```

pode resultar em um redimensionamento indesejado da matriz *A*, por padrão os valores restantes são preenchidos com zeros. Outras formas de manipulação de matrizes serão observadas no capítulo 5.

3.2 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 + <ENTER>
v =
   3  4  5  6  7
>> v = 3:1.5:7 + <ENTER>
v =
   3.0  4.5  7.0
```

3.2.1 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)
ans =
    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)
ans =
    5   6   7
    9  10  11
```

Matrizes vazias podem ser definidas da seguinte maneira:

```
>> V = [];
>> V = 4:-1:5;
```

3.3 Cálculos fundamentais e matrizes especiais

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

```
>> clear % apaga-se todas as variáveis definidas até agora
>> a=[1 2;3 4];%matriz 2x2
>> b=a(:) + <ENTER>
b =
   1
   2
   3
   4
```

Para saber a dimensão de uma matriz (de agora em diante supõe-se que na linha do prompt >> digitou-se a tecla <*ENTER*>)

```
>> 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 define-se como : 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
```

3.3.1 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
- *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.

Assim por exemplo, quando se trabalha com números complexos, deve-se evitar o uso das "constantes" *i*, *j* para efetuar laços tipo for, pois isso ocasionará erros no cálculo com números complexos do tipo *a+b*i*, dado que o imaginário *i* assumira valores inteiros e deixaria de valer a raiz quadrada de -1. Um outro exemplo:

```
>> date
ans =
   21-Nov-2001
>> date = 1;
>> date
date =
    1
>> clear date
>> date
ans =
   21-Nov-2001
```

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 funções simples que podem ser usadas tanto em escalares como em matrizes ou vetores.

```
>> abs(x);  % valor absoluto da variável x
>> sqrt(x); % raiz quadrada de x
>> round(x);% arredonda x para o inteiro mais próximo
>> fix(x);  % arredonda x para o inteiro mais próximo de zero
>> floor(x);% arredonda x para o inteiro mais próximo de −∞
>> ceil(x); % arredonda x para o inteiro mais próximo de +∞
>> sign(x); % sinal de x, +1 ou -1
>> rem(x);  % resto de x:y
>> exp(x);  % exponencial de x
>> log(x);  % função 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];
```

então os seguintes cálculos são obtidos:

```
>> x = round(a) % inteiro mais próximo
x =
   -3
>> x=fix(a) % inteiro mais próximo de zero
x =
  -2
>> x=floor(C) % inteiro mais próximo de +∞
x =
  2  1  -4  3
>> x = ceil(C) % inteiro mais próximo de +∞
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 número negativo o terceiro valor resulta num complexo.

```
>> x = rem(7.5,2) % resto da divisão 7.5/2
x =
  1.5
>> x = rem(A,B) % resto da divisão entre as componentes
x =
  1  0  1
```

No caso do argumento de entrada ser uma lista/matriz, cada elemento da lista (ou matriz) resposta é a solução dada pela função que correspondente ao elemento na mesma posição 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
```

Outros exemplos, divisão a esquerda e a direita.

```
>> u=[1 2 3];
>> v=[2 2 2];
>> u./v; % vetor contendo as divisões ui/vi
>> v.\u; % o mesmo resultado anterior
```

A hierarquia entre operações aritméticas tais como operações entre parênteses, potenciação, radicalização, soma, produto, etc é a usual.

4.1.2 Números complexos

Na próxima seção será preciso o conceito de número complexo. Como citado o valor padrão de *i* e *j* é a raiz quadrada de -1. Logo, naturalmente, um número complexo define-se da seguinte maneira (por exemplo):

```
>> 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* são 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
```

Para maiores informações do comando *format* digite *help format*.

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 é 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.

Para as funções sin(*x*) e cos(*x*), se o argumento é real este não é limitado, isto é $x_{ij} \in [-\infty,+\infty]$, se *x* é complexo a funções sin(*x*), cos(*x*), tan(*x*) não estão definidas para qualquer complexo. A função tan(*x*) é indefinida para valores da forma (2**n*+1)**pi*/2, já que tan(*x*)=sin(*x*)/cos(*x*) e cos((2**n*+1)**pi*/2) é nulo para *n* inteiro. Por exemplo:

```
>> A = [pi -pi;pi/2 -pi/2];
>> sin(A)
ans =
   0.0000  -0.0000
   1.0000  -1.0000
>> sin(1000+i)
ans =
  1.2759 + 0.6609i
>> sin(1000*i)
ans =
   NaN + Inf i
```

Os símbolos *NaN* e *Inf* foram mencionados na subseção 3.3.1. As funções asin(*x*), acos(*x*) e atan(*x*) estão definidas para qualquer valor *x* real. Se *x* está no intervalo [-1,1] o resultado é real, caso contrário é complexo.

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).

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 seqüência 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
circun.m;%Esta função calcula o perímetro de uma
        %circunferência de raio R
```

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^ 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.

Observações

Para que o arquivo ".m" seja localizado pelo Matlab, este deve se encontrar no diretório atual. O comando *pwd* do Unix fornecerá o caminho atual, por exemplo:

```
>> pwd;
/a/home/cenapad/cedric/MATLAB
```

Se o arquivo ".m", a ser interpretado pelo Matlab, estiver em outro diretório diferente do atual, digamos /u/cedric/programas, então o Matlab terá acesso a ele declarando o caminho com o comando *path* da seguinte maneira.

```
>> path(path,'/u/cedric/programas');%no Unix
>> path(path,'c:\cedric\programas');%no Windows
```

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 do que
<=	menor ou igual
>	maior que
>=	maior ou igual
= =	igual no sentido de condição/comparação
~ =	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';
text01==text2
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”.

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>=b* ou que *b>=c*. Outro exemplo:

```
>> ~ (b= =c | c<a) % pergunta: é falso que b é igual a c ou
                   %c é menor que a?
ans =
   0
```

A última expressão é uma negação que equivale a questionar ou afirmar**:** *b* é distinto de *c* e *c* é maior ou igual que *a*? Isto é falso devido a que c<a.

```
>> a ~ = c | b < c
ans =
   1
```

A resposta é que a expressão é verdadeira. Para isto basta que uma das comparações seja verdadeira, isto é, que *a* seja distinto de *c* ou que *b* seja menor do que *c*.

5.4 Laço estrutural *if-else-end*

Com os operadores anteriores estamos prontos para criar programas com a linguagem fornecida pelo Matlab. Os exemplos do uso destas estruturas em nada diferem daquelas utilizadas em linguagens como C ou Fortran.

```
>> a=rand; % rand retorna um número aleatório entre 0 e 1
>> if a >= .5
   disp('O valor aleatório é maior ou igual do que
1/2');%comando display
else
   disp('O valor aleatório é menor que 1/2');%permite
%visualizar uma mensagem durante a execução
end
```

O valor aleatório é maior ou igual do que ½.

Observações:

O laço foi iniciado em modo interativo.

Enquanto o laço não foi fechado, com o *end* final, este não foi executado nem o prompt >> foi liberado.

No exemplo o valor de *a* foi *a*=0.6068, assim o fluxo do programa passou pela primeira bifurcação e a mensagem correspondente foi mostrada. Repare que a variável "default" *ans* não foi visualizada.

A estrutura de *if* 's pode ser aumentada da seguinte maneira. Num arquivo ".m" ficaria.

```
%ex1
val1 = input('Entre um número : ');%permite a entrada de va-
                  %lores via teclado em modo interactivo
val2 = input('Entre outro número : ');
if val1<val2
   disp('O primeiro valor é menor que o segundo');
elseif val1>val2
   disp('O segundo valor é maior que o primeiro');
else
   disp('Os valores são iguais');
end
```

É possível fazer uso da estrutura *if-else-end* de modo aninhado, como mostra o próximo exemplo.

```
%ex2
%exemplo de if-else aninhado
disp('Este programa testa si dois números são divisíveis');
num=input('Entre numerador');
den=input('Entre denominador');
if(num>=den)
   if(rem(num,den)==0) % o resto é nulo?
      if(num==den) % numerador e denominador iguais?
         disp('Os números são iguais');
      else
         disp('Eles são divisíveis');
      end
   else % o resto é diferente de zero
      disp('Eles não são divisiveis');
   end
else % caso num<den
   disp('O denominador é maior que o numerador');
end
```

Um outro exemplo é de um programa que transforma unidades, centímetros, polegadas e pés.

```
%ex3
x=input('Entre com o valor da medida: ');
unidade1=input('Unidade de entrada (cen/pol/pes): ','s');
unidade2=input('Unidade de saída (cen/pol/pes): ','s');
if (strcmp(unidade1,'cen') & strcmp(unidade2,'pol'))
   y=0.393701*x;
elseif (strcmp(unidade1,'pol') & strcmp(unidade2,'cen'))
```

```
   y=2.54*x;
elseif (strcmp(unidade1,'cen') & strcmp(unidade2,'pes'))
   y=0.0328084*x;
elseif (strcmp(unidade1,'pes') & strcmp(unidade2,'cen'))
   y=30.84*x;
elseif (strcmp(unidade1,'pol') & strcmp(unidade2,'pes'))
   y=x/12;
elseif (strcmp(unidade1,'pes') & strcmp(unidade2,'pol'))
   y=12*x;
else
   disp('Símbolo da unidade não definido');
end
```

O caso acima pode também ser refeito utilizando a estrutura *switch-case-otherwise-end*, como é dado na próxima subseção.

5.5 Estrutura *switch-case-otherwise-end*

Outra forma padrão de resolver uma tomada de decisões é exemplificada a seguir.

```
%ex4
% Conversão entre unidades centímetros, polegas e pés
fprintf('\n\n');%pula duas linhas
disp('Conversão entre unidades: centímetros, polegadas e ...
pés');
fprintf('\n');
x=input('Entre valor numérico a converter : ');
fprintf('\n\n');
disp('Os seguintes são sistemas válidos de conversão:');
disp('cen-pol ; pol-cen ; cen-pes ; pes-cen ; pol-pes ;...
pes-pol');
fprintf('\n');
sistema = input('entre opção entre aspas> Entre sistema de...
conversao : ');
switch sistema
   case 'cen-pol'
      y=0.393701*x;
      disp([num2str(x),' centímetros = ',num2str(y),'
polegadas']);
   case 'pol-cen'
      y=2.54*x;
      disp([num2str(x),' polegadas = ',num2str(y),'
centímetros']);
   case 'cen-pes'
      y=0.0328084*x;
      disp([num2str(x),' centímetros = ',num2str(y),' pés']);
```

```
   case 'pes-cen'
      y=30.48*x;
      disp([num2str(x),' pés = ',num2str(y),' centimetros']);
   case 'pol-pes'
      y=x/12;
      disp([num2str(x),' polegadas = ',num2str(y),' pés']);
   case 'pes-pol'
      y=12*x;
      disp([num2str(x),' pés = ',num2str(y),' polegadas']);
   otherwise disp('Unidade desconhecida');
end
```

5.6 Estrutura *while-end*

O seguinte exemplo usa o comando *while-end* e permite achar um número pequeno que a maquina considera não nulo. Este número é uma constante do Matlab denotada por eps.

```
%ex5
%Este programa calcula a constante eps
fprintf('\n\n');
disp('O laço while-end acha um número pequeno EPS2 tal que...
1+EPS2/2 = 1');
EPS2 = 1;
num = 1;% contador do número de laços
while (1+EPS2) > 1
   EPS2 = EPS2/2;
   num = num+1;
end
EPS2 = 2*EPS2;
disp(['EPS2 = ',num2str(EPS2)]);
disp(['número de lacos = ',num2str(num)]);
disp('Faça os testes lógicos : EPS2+1>1 e EPS2/2+1>1');
```

5.7 Estrutura *for-end*

Para o valor da variável indicial do laço deve ser dado um valor inicial, um incremento e um valor final para o termino do laço.

```
%ex6
seg = input('Contagem regressiva: tempo -> ');
fprintf('\n');
disp('* Aguardando o inicio da contagem *');
fprintf('\n');
tic;%marca o inicio do tempo
```

```
cont = 0;
for k=seg:-1:0,
   cont = cont+1;
   while toc < cont,
   end
   disp(['Contagem regressiva : ',num2str(k)]);
end
```

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); % 5x5
>> A';%transposta de A
>> det(A);%determinante de uma matriz
>> inv(A);%inversa de A, quando esta existe
>> A-1;%inversa de A
>> 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)=1*6+2*7+3*8+4*9+5*0;%somatória de produtos
>> %ou produto escalar de dois vetores
>> B = randn(3,5);%matriz aleatoria 3x5
>> B.A;%produto de matrizes, 3x5-5x5
>> v3=[1 3 5 7 9]';%vetor coluna
>> v4=[0 2 4 6 8]';%vetor coluna
>> sum(v1.*v2);%somatória de produtos (vetores linhas)
>> sum(v3.*v4);%somatória de produtos (vetores colunas)
>> sum(v1'.*v3);%somatória de produtos
>> sum(v1.*v4');%somatória de produtos
>> C=randn(5,3);%matriz aleatoria 5x3
```

As seguintes operações são validas:

```
>> 2*A+A.A;
>> C'*A;%(3x5)-(5x5)
>> B*C; C*B; % 3x5-5x3, 5x3-3x5
>> B*B'; % 3x5-5x3
>> A-1*A;%si existe a inversa o resultado é a identidade
>> (C'*A)-1; % caso a inversa existe (3x5-5x5)
>> (B'*C')-1*(A'*A); % 5x3, 3x5, 5x5, 5x5
```

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,4,2);%reescreve C com diferentes dimensões
>> E = reshape(B,6,7);% reescreve B preenchendo com zeros
                          % as novas posições
>> F = triu(A);% reescreve A preenchendo com zeros
           % a parte triangular superior
>> G = tril(A);%G é a parte triangular inferior de A
```

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; % valores entre 0 e 100
   end
end
media_turma = mean(TURMA); %media por matéria (turma toda)
media_portugues = mean(TURMA(:,3));%média das notas da coluna
3
[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);
```

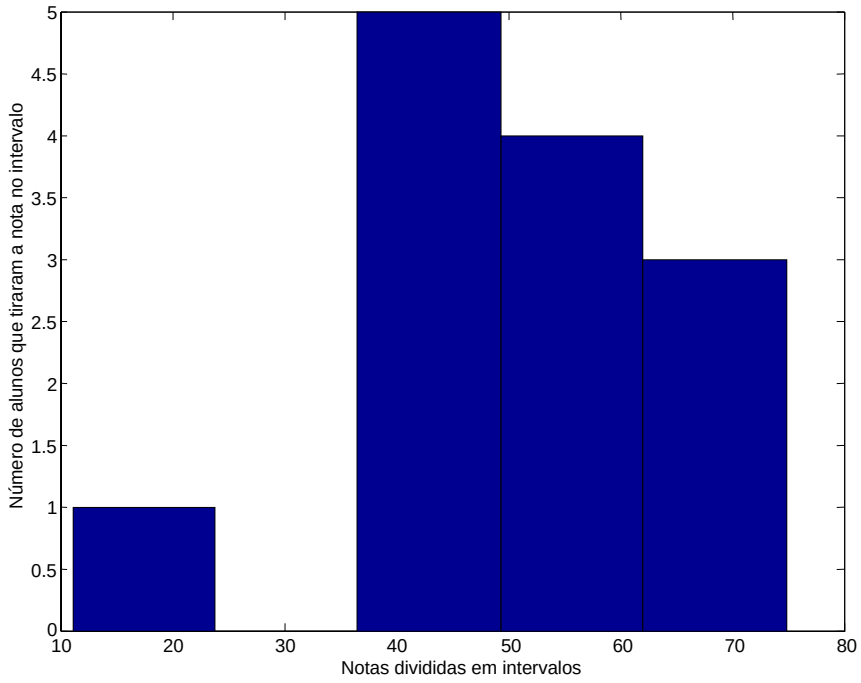

Figura 1: Gráfico de barras: histograma

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 letras, 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. Exemplo simples:

```
%ex8
>> 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, figura 2
>> plot(x,z);%gráfico do coseno de x
>> plot(x,y,x,z);%grafica simultaneamente seno e coseno na
mesma janela
```

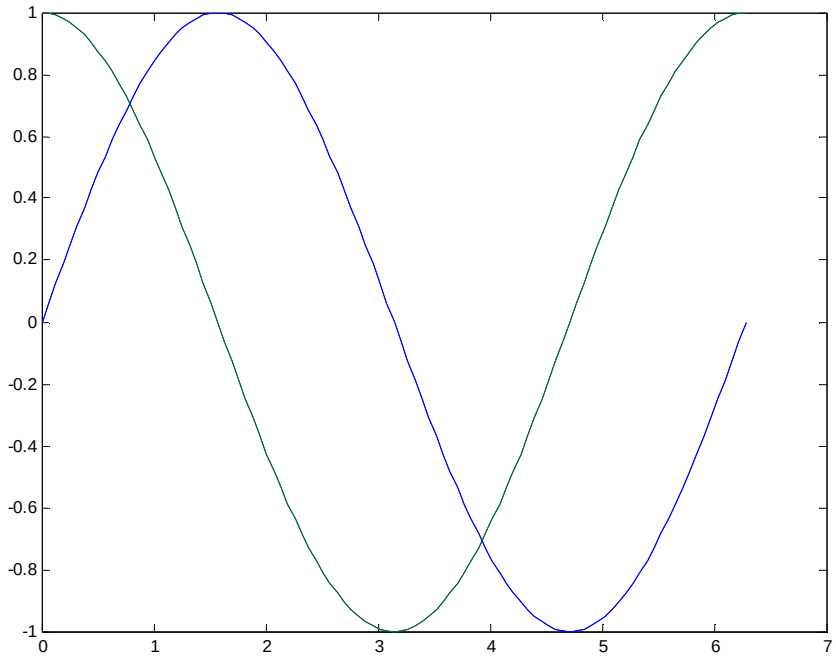

Figura 2: Comando hold on : permite plotar várias figuras numa mesma janela

Com o comando *hold on* todas as figuras posteriores serão desenhadas na última janela (em justaposição). A diferença do último comando acima com o comando

As opções *b:p, c-, m+* estão indicadas na tabela a seguir. A combinação *b:p* indica que os pontos do gráfico de *sin(x)* são identificados com um pentagrama (letra *p*), veja figura 4, os pontos serão unidos por uma linha pontilhada, indicado por dois pontos ":", tanto os pontos como a linha serão mostradas em azul, indicado pela letra *b*. A opção *c-* indica que o gráfico de *cos(x)* será mostrado com uma linha contínua, indicado com o traço -, de cor ciano, indicado com a letra *c*. O próximo exemplo permite modificar o fundo da janela do gráfico, controlar o tamanho do intervalo, tanto nos eixos *x* como *y*. Este mesmo exemplo trabalha com títulos e o tamanho da fonte e tipo de letra, como negrito, itálico, normal.

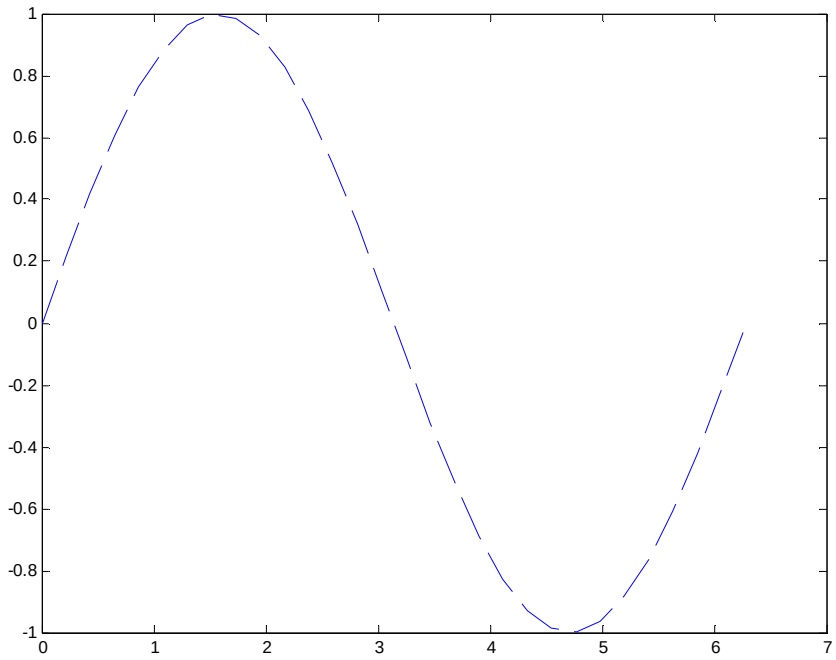

Figura 3: Função sin(x) : traçado pontilhado

Considerando o mesmo intervalo anterior assim como as funções seno e co-seno, ainda define-se:

```
%ex10
>> 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-');
```

Na última linha os pontos *(x,sin(x))* são plotados com asteriscos em vermelho. Os pontos *(x,cos(x))* são plotados com círculos brancos e estes unidos por uma linha contínua de cor branco. Os pontos da curva *cot(x)* são desenhados como pentagramas e unidos com uma linha tracejada de cor preta.

```
>> axis([0 2.5 -3.5 3.5]);
```

A última linha limita o comprimento dos intervalos do eixo *x* e do eixo *y*. O eixo *x* é visualizado entre 0 e 2.5 e o eixo *y* entre -3.5 e 3.5.

```
>> grid;%cria uma grade superposta ao gráfico atual
>> title('Funções Trigonométricas','FontSize',14,...
'Fontweight','Bold');
```

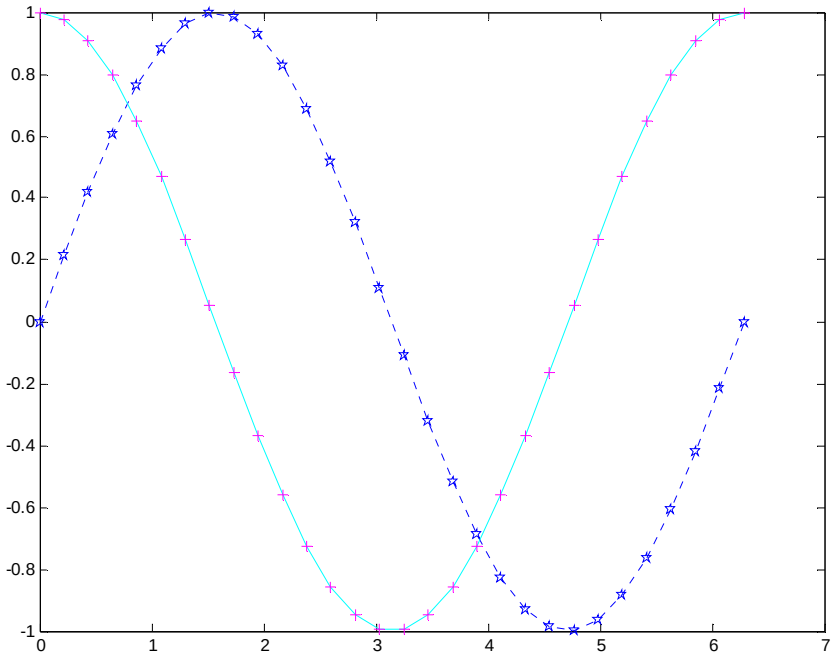

Figura 4: Gráficos superpostos com opções distintas de pontos e traçado

Esta última linha adiciona o título Funções "trigonométricas" na parte superior da janela do gráfico, com um tamanho 14 de fonte tipo negrito.

```
>> xlabel('\Tetha','FontSize',14,'Fontweight','Bold',...
'FontAngle','normal');
>> ylabel('Funções','FontSize',14,'Fontweight','Bold',...
'FontAngle','italic');
```

Estas linhas agregam títulos a ambos os eixos *x* e *y*. A saída para *\tetha* será a letra grega que leva este nome. Escolhem-se letras em negrito de tamanho 14 pontos, normal e itálico respectivamente. Define-se:

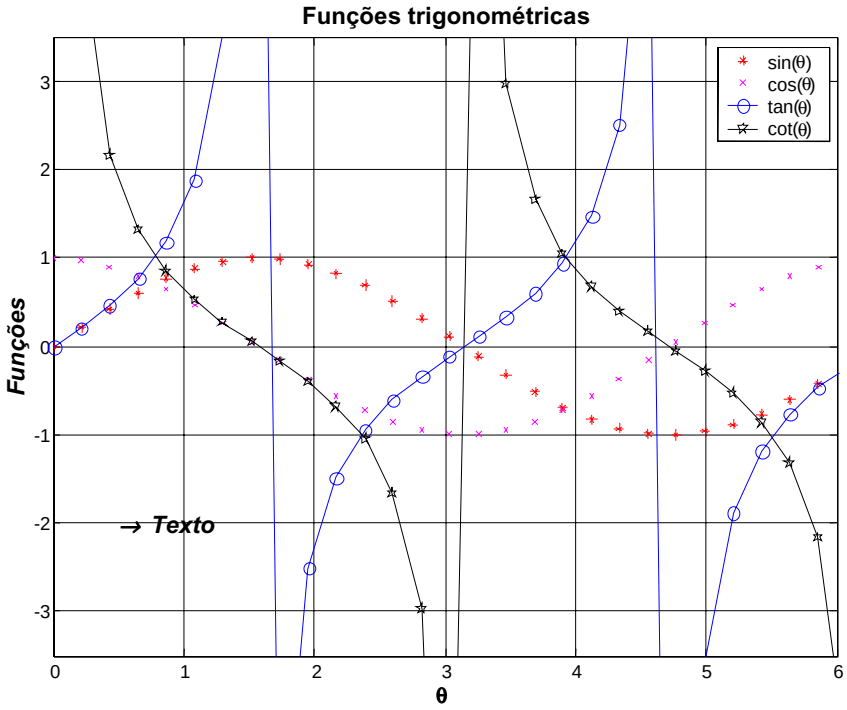

Figura 5: Opções de representação gráfica simultâneas

```
>> legend('sin(\tetha)','cos(\tetha)','tan(\tetha)',...
'cot(\tetha)');
```

Esta última opção provoca a aparição de um pequeno retângulo na parte superior direita da janela do gráfico, que identifica os gráficos e a forma de representação dos mesmos, segundo a cor, tipo de traço e ponto adotado pela ordem de aparição no comando

plot, da esquerda para direita respectivamente, figura 5. A seqüência *\Tetha* é visualizada como sendo a letra grega correspondente, figura 5.

Um outro comando usado para introduzir texto dentro do gráfico no ponto coordenado, digamos (x,y), é *text*. Por exemplo, a linha:

```
>> text(0.5,1,'\rightarrow Texto');
```

Coloca uma seta apontando a direita para a palavra Texto (-> Texto), este texto será colocado no ponto (1/2,1). Este tipo de comando permite também utilizar letras gregas e símbolos matemáticos no formato latex como por exemplo *\delta*, *\nabla* etc.

8.1 Comando *subplot*

O comando subplot permite visualizar várias gráficas separadas dentro de uma mesma janela. Por exemplo:

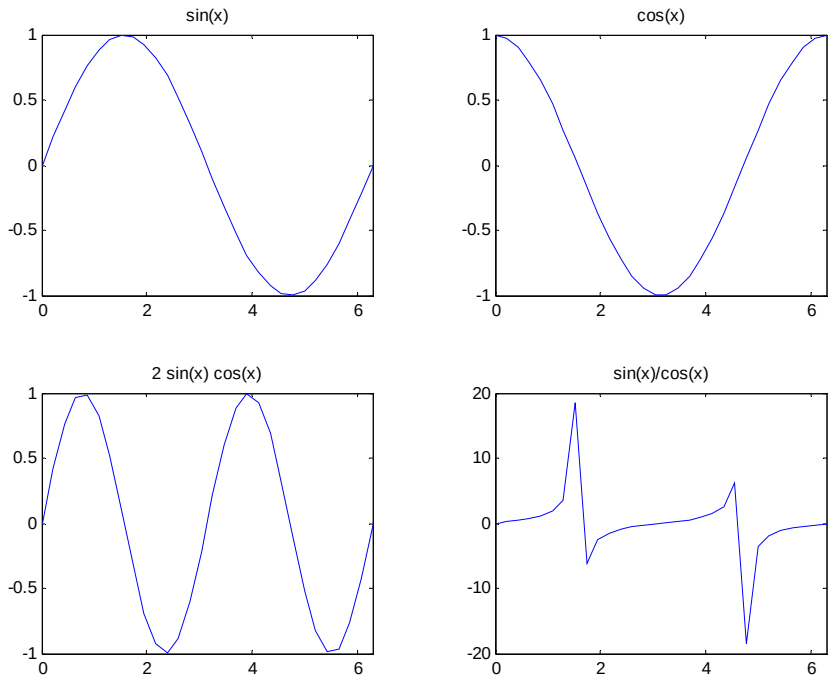

Figura 6: Subgráficos: arranjo gráfico de dimensões 2 x 2

```
%ex11
>> 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);%reserva uma janela para 4 gráficos
```

Os argumentos estão designando uma mesma janela para plotar os seguintes 4 gráficos. Os primeiros dois argumentos indicam o número de gráficos que podem ser alocados numa mesma janela. Neste caso 4 gráficos dispostos em uma ordenação 2x2. Então os valores 2,2,1 indicam que o próximo gráfico a ser plotado será visualizado na parte superior esquerda da janela, o que é indicado pelo argumento 1.

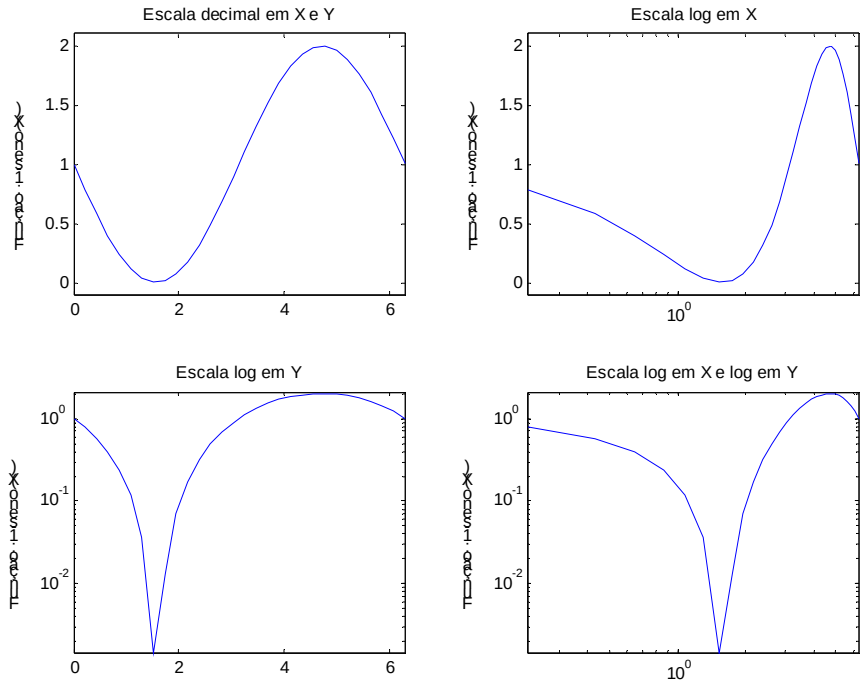

Figura 7: Gráficos utilizando escalas logarítmicas

```
>> 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 é para o último gráfico: y =
sin(x)
```

```
>> subplot(2,2,2);%o 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
```

O resultado destas operações é mostrado na figura 6.

8.2 Outros recursos para gráficos bidimensionais

Existem comandos que permitem plotar em escalas logarítmicas, gráficos de barras ou pizzas. Os seguintes comandos podem ser usados no lugar de *plot.*

```
>> x=(-pi,pi,30);
>> y=sin(x);
>> loglog(x,y);
```

Ao invés dos valores de x e y serão plotados os valores dos logaritmos desses valores.

```
>> semilog(x,y);
```

Neste último caso escala logarítmica será usada somente para os valores do eixo x.

```
>> semilogy(x,y)
```

Neste caso escala logarítmica será utilizada só no eixo y. O gráfico apresenta exemplificações destes comandos.

```
>> area(x,y);%a área baixo a curva será preenchida
% com uma cor
>> pie(a,b);%gráfico em formato pizza
```

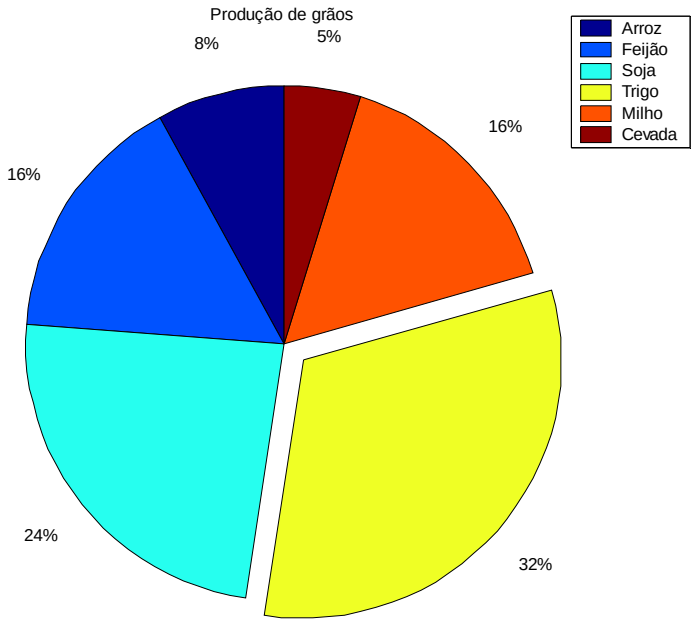

Figura 8: Gráfico em formato pizza ou torta

Gráficos em formato de torta ou pizza são criados usando este comando. O vetor *a* contém valores a serem plotados e *b* é um vetor lógico opcional que descreve as fatias a serem separadas.

Exemplo: gráfico de torta, figura 8.

```
%ex13
>> 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');
```

A versão tridimensional do gráfico de torta pode ser obtida da usando o mesmo vetor *a* anterior, figura 9.

```
>> destaque = [0 1 0 1 0 1];%pedaços em destaque
>> pie3(a,destaque);%formato pizza tridimensional
>> colormap hsv;%opção de tonalidade de cor
```

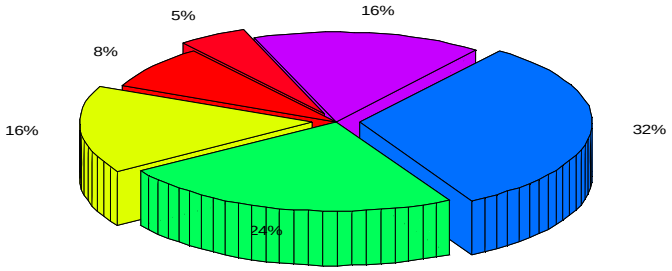

Figura 9: Gráfico de torta tridimensional

Exemplo: gráfico de barras.

```
%ex14
>> b = [1960 1970 1980 1990 2000 2001];
>> bar(b,a);%barra vertical
```

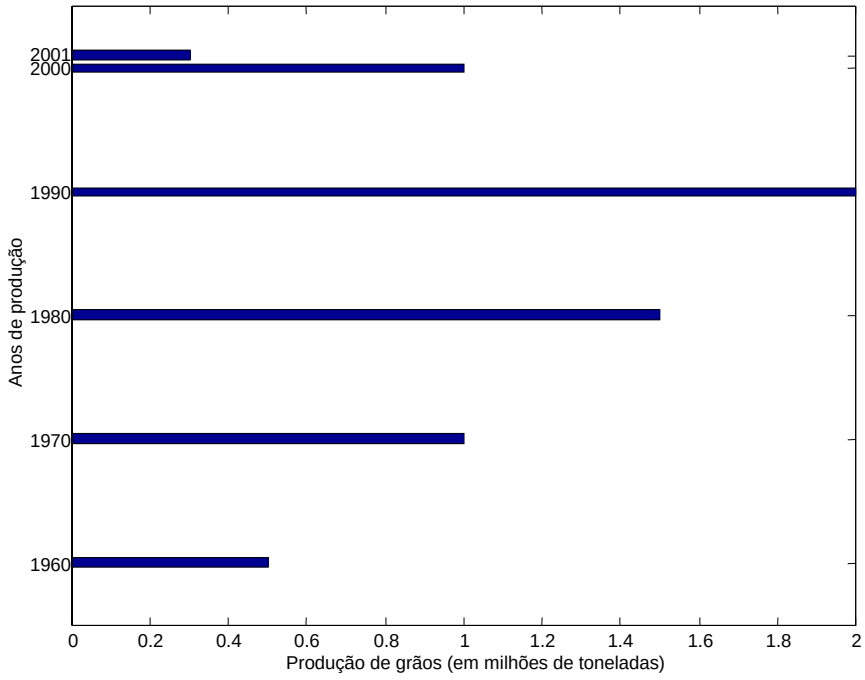

Figura 10: Gráfico de barras horizontal

Os valores de *b* são as abscissas os de *a* são as ordenadas correspondentes. A escala dos valores do vetor *b* não é adequada para um gráfico de barras vertical, os primeiros valores são a cada 10 anos enquanto que nos últimos dois a diferença é de um ano, assim uma forma mais adequada é utilizar um gráfico de barras horizontal como exemplificado a seguir.

```
>> barh(b,a);%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 11
```

Exemplo: gráfico de escada, figura 11.

```
>> stairs(b,a);
>> xlabel(' Anos de produção ');
>> ylabel(' Produção em milhões de toneladas ');
>> title('Gráfico em escada','FontSize',18);
```

Figura 11: Gráfico em formato de escada

Exemplo: gráfico de área, figura 12.

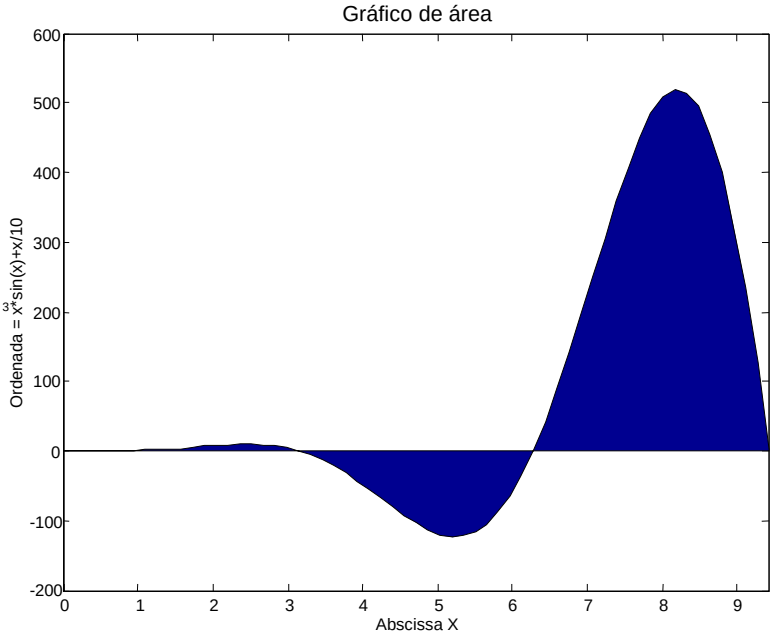

Figura12: Gráfico de área: a área entre a curva e o eixo X

```
%ex15
>> x=0:0.05*pi:3*pi;
>> y = x.3.*sin(x)+x/10;
>> xlabel(' Abcisa X ');
>> title('Gráfico de área','FontSize',18);
>> ylabel(' Ordenada = x.3 * Sin(x) + x/10 ');
>> area(x,y);
```

8.2.1 Gráfico em coordenadas polares

Neste caso cada ponto do plano é identificado a um raio *r* (distância da origem) e um ângulo *tetha* (aquele que forma o raio com o eixo *X*). Cada ponto de uma curva no plano é identificado com um par *(r,tetha).* Exemplo:

```
%ex16
>> tetha=0:2*pi/100:2*pi;%divisão do intervalo [0,2pi] em 100
partes
>> r = tetha/(2*pi);
>> subplot(1,2,1);
>> polar(tetha,r);%gráfico em coordenadas polares
>> title('* E S P I R A L *');
```

Neste caso o raio é inversamente proporcional à constante 2*pi. Quando o ângulo tetha aumenta de 0 a 2pi e o raio também aumenta proporcionalmente. Então a curva é circular afastando-se do centro que é a própria origem, isto é, uma espiral. Este e o próximo exemplo são mostrados na figura 13.

```
>> subplot(1,2,2);
>> polar(tetha,sin(2*tetha).*cos(2*tetha));
>> title('Rosa de 8 pétalas');
```

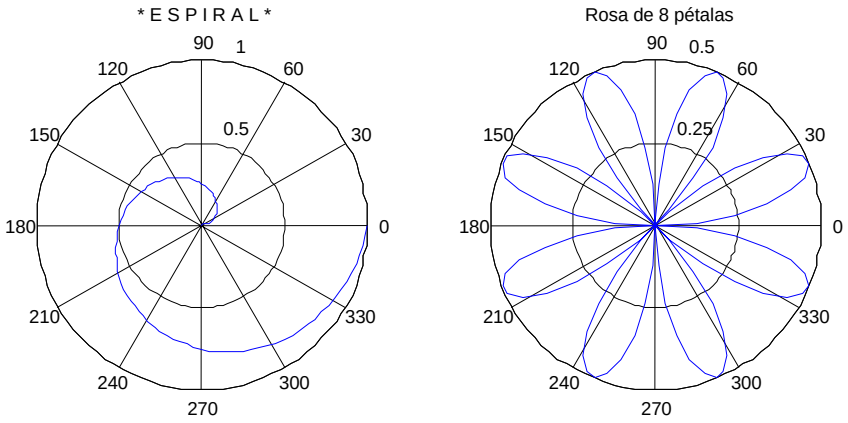

Figura 13: Gráficos em coordenadas polares

8.3 Gráficos tridimensionais

Para traçar gráficos curvilíneos 3D utiliza-se o comando ou função *plot3*. No próximo exemplo 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 anti-horário 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). O gráfico está desenhado na figura 14.

```
%ex17
>> t = linspace(0,10*pi,200);%[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');
```

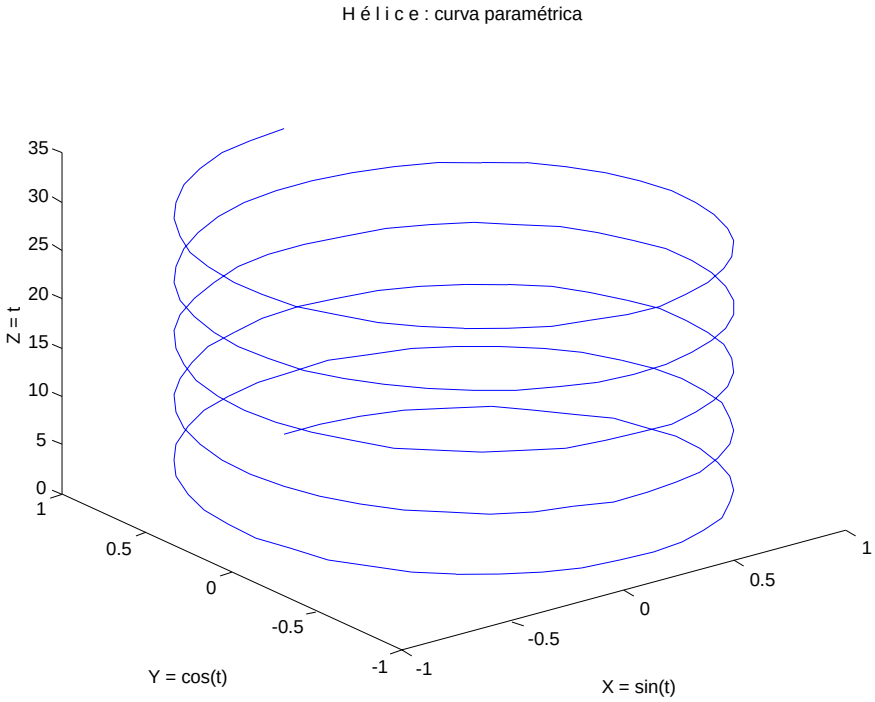

Figura 14: Gráfico de linhas tridimensional (paramétrico)

8.3.1 Gráficos de superfície

A seguir é apresentado um exemplo com explicações de cada comando gráfico.

```
%ex18
>> 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
```

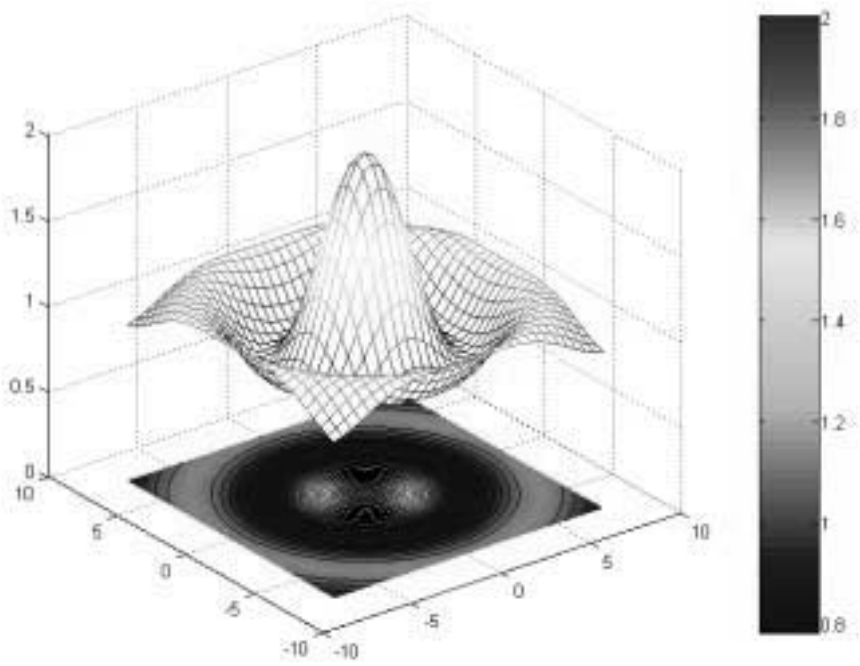

Figura 15: Produto final da seqüência de comandos do exemplo

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. A seqüência de comandos escritos acima é visualizada na figura 15. Pode-se salvar o gráfico em vários formatos.

8.3.2 Animação gráfica

- Animação bidimensional

Este exemplo mostra como se constrói uma seqüência 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.

```
%ex19
>> 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.

- Animação tridimensional

O laço que se apresenta a seguir não difere muito daquele apresentado anteriormente. Exemplo:

```
%ex20
>> 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(c0 = getframe;
  end;
>> movie(M,20);%Esta animação pode ser salva
```

9 Solução de sistemas de equações lineares

Um sistema de equações lineares se escreve da forma $Ax = b$ onde A é uma matriz conhecida de dimensões *nxm*, geralmente quadrada (*m=n*), x é o vetor coluna de incógnitas e b é um vetor coluna de valores numéricos. Caso A (quadrada) seja não singular, isto é, existe uma matriz denotada A^{-1} tal que $A * A^{-1} = A^{-1} * A = I$, onde I é a matriz identidade. Então pela teoria o sistema apresenta uma única solução x. A matriz identidade é caracterizada por $I_{ii} = 1$ e $I_{ij} = 0$ se $i \neq j$, isto é, os valores da diagonal são todos 1 e fora da diagonal os valores são 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):

$$\text{minimize Norm}(A * x - b) \tag{4}$$

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

$$A * x = b \Rightarrow x = A^{-1} * b \tag{5}$$

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
               %retornara a seguinte
               %mensagem de erro:
G??? 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
```

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

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

Decomposição LU

O seguinte exemplo apresenta os conceitos citados anteriormente a respeito dos métodos de decomposição.

```
>> 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 implementado o seguinte arquivo "resolucao_LU.m". A chamada resolucao_LU(L,U,P,b) retorna a solução x.

```
%solve_LU
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
```

Os algoritmos de resolução associados a decomposição de matrizes podem ser achados em textos de Álgebra Linear e a maioria são fáceis de implementar.

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.

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

```
%ex21
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 6 iterações com um resíduo de 7.4264e-05. O número flag = 0 indica que o método convergiu com a desejada tolerância num número de iterações menor ou igual ao "default".

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.

```
%ex22
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 8 iterações. A norma do resíduo relativo foi:

$$\frac{\|b - A * x\|}{\|b\|} = 2.1228e - 07 \tag{6}$$

Um segundo teste onde foi usada uma tolerância de 1.e-08, convergiu em 10 iterações, atingindo um erro relativo de 1.1667e-13, com *flag* = 0.

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*, figura 16.

```
%ex22a
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
```

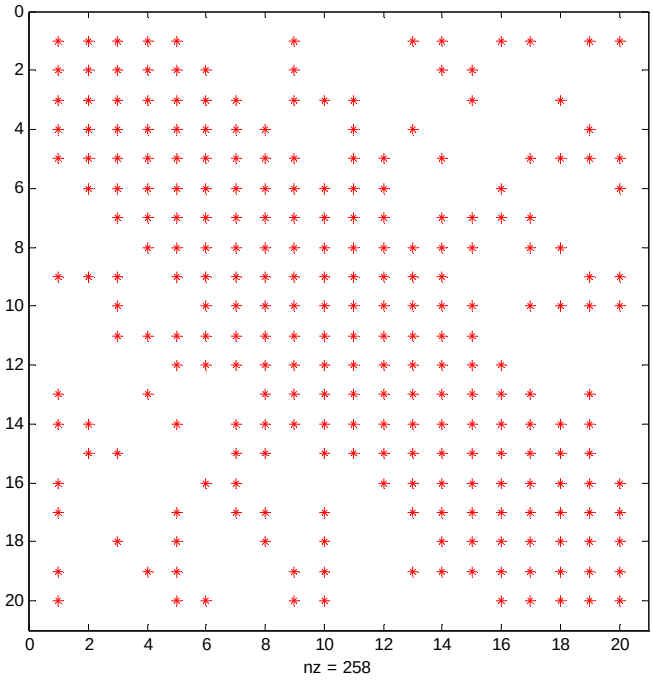

Figura 16: Gráfico mostrando os valores não nulos de uma matriz

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*.

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.

O primeiro exemplo interliga pontos discretos no plano por meio de linhas retas, obtendo-se uma curva linear ou de grau um por partes, figura 17.

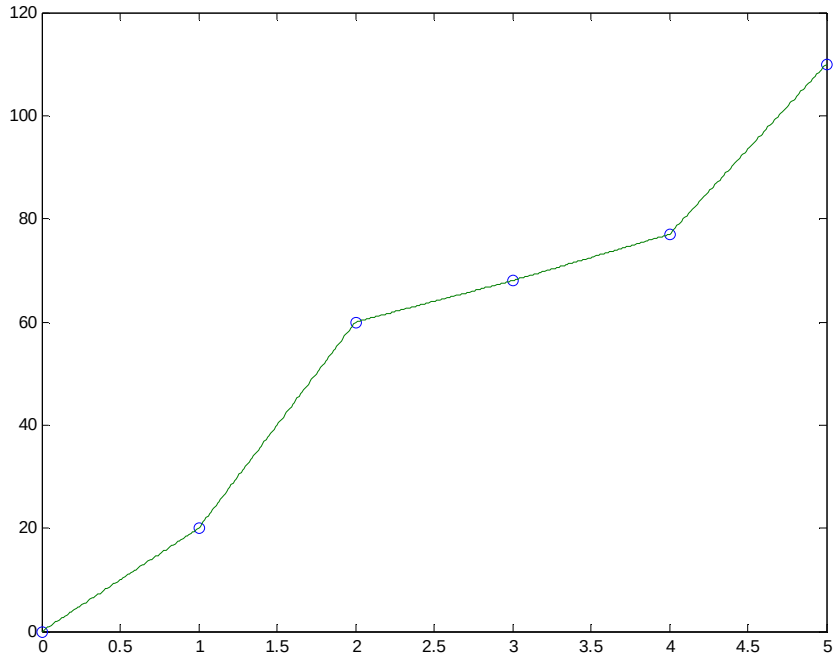

Figura 17: Interpolação linear por partes

```
%ex23
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.

O próximo exemplo corresponde a uma interpolação superficial, figura 18.

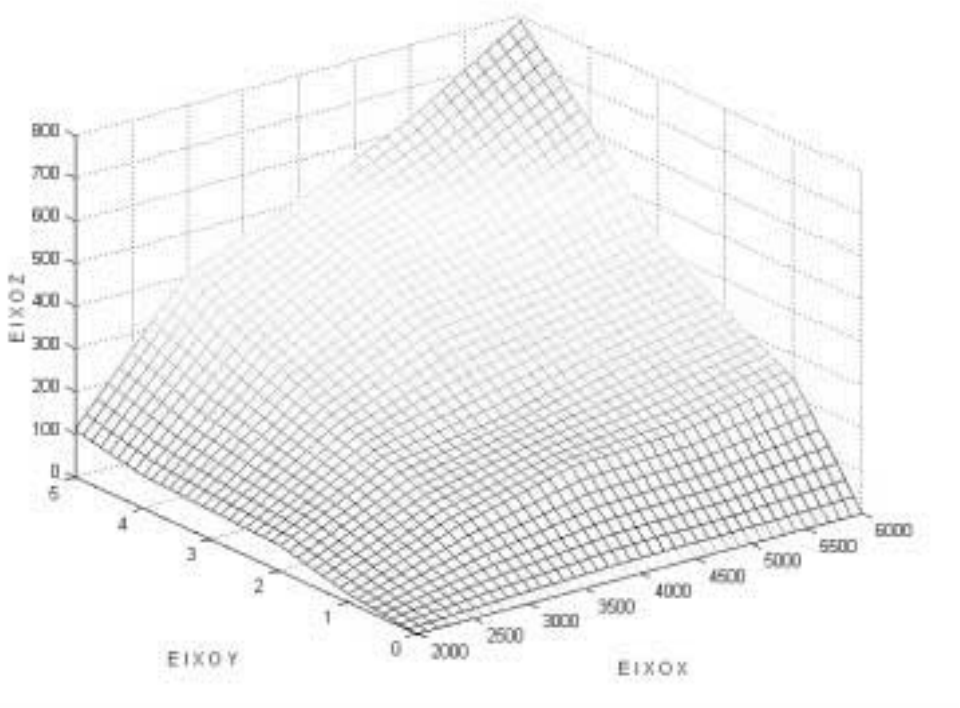

Figura 18: Superfície interpolante da matriz temperatura

```
%ex24
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');
```

- 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. A figura 19 mostra a curva *spline* interpolante do exercício a seguir.

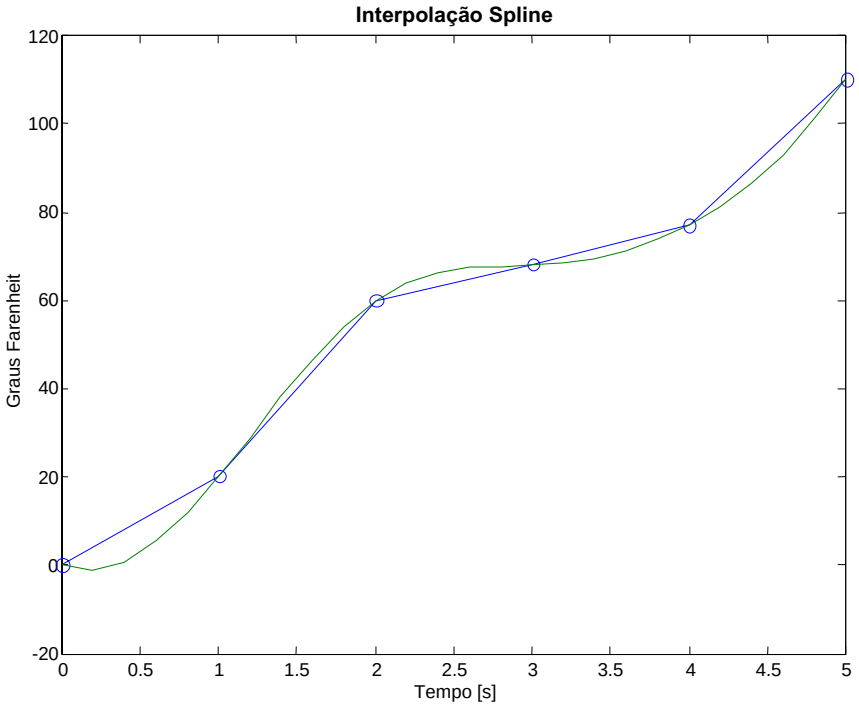

Figura 19: Interpolação utilizando spline

```
%ex25
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
```

A saída numérica foi: *temp1* = 67.3013, *temp2* = 67.3013, 105.2020, *temps* = 2.7917, 42.2083, 66.8750, 70.4167, 89.5833. Observe na figura como a curva é suave nos pontos conhecidos.

- 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 x^n, $n = grau - 1,...,2,1,0$. O aumento do grau do polinômio nem sempre garante uma melhor aproximação à resposta procurada. Veja figura 20 para a saída gráfica deste exemplo.

```
%ex26
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;
```

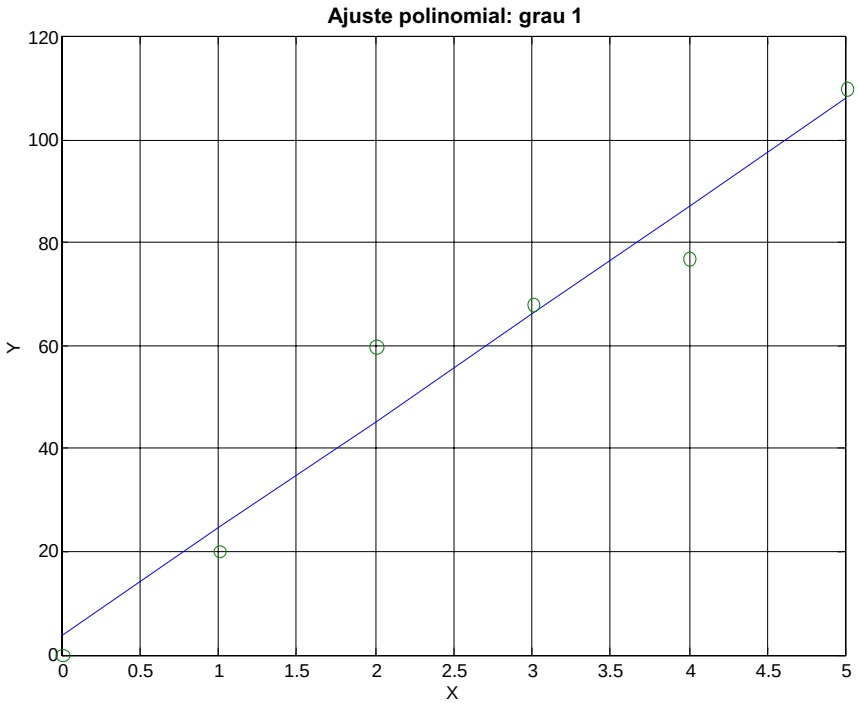

Figura: Ajuste polinomial de grau 1

O resultado *coef* = 20.8286, 3.7619 significa que o polinômio considerado na aproximação é:

$$P1 = 20.8286 * x + 3.7619 \tag{7}$$

A variável *ybest* = 3.7619, 24.5905, 45.4190, 66.2476, 87.0762 ,107.9048 apresenta a avaliação deste polinômio para os valores do vetor *x*. O próximo exemplo mostra como o aumento do grau do polinômio não necessariamente aumenta a precisão dos resultados procurados. Ajustam-se aos dados polinômios de graus 2 e 10.

```
%ex27
x = 0:0.1:1;
y = [-0.447 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 9.30
11.2];
grau = 2; % n é o grau do polinômio de interpolação
p2=polyfit(x,y,grau);
grau = 10;
p10=polyfit(x,y,grau);
```

```
xi = 0:0.01:1;
y2 = polyval(p2,xi);
y10 = polyval(p10,xi);
plot(x,y,'r+',xi,y2,'b-',xi,y10,'g-');
xlabel('x');
ylabel('y');
legend( 'Resultados experimentais','P2', 'P10');
```

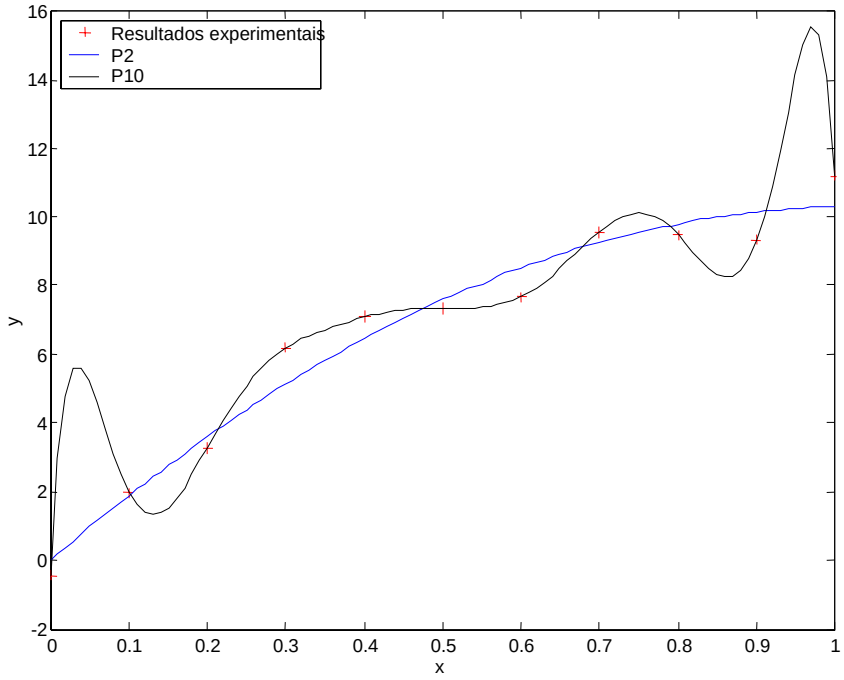

Figura 21: Ajuste polinomial: graus 2 e 10

Em geral, se existem *n* pontos a serem interpolados, então um polinômio de grau acima de *n*-1 deve conter todos os pontos conhecidos. Como pode ser observado na figura 21. Na mesma figura o polinômio de grau 10 apresenta uma maior oscilação em relação ao polinômio de grau 2.

- Mais sobre os comandos *interp1* e *interp2*

A outras opções que podem ser passadas para estes comandos como interpolação tipo *spline* que já foi mencionada; tipo *cubic* (o mesmo que Hermite cúbico por partes) onde fora dos valores da função são também requisitadas condições na primeira derivada nos

pontos conhecidos; tipo *linear* que é "default". A figura 22 mostra o gráfico do exemplo onde foram usados os três tipos de interpolação citados.

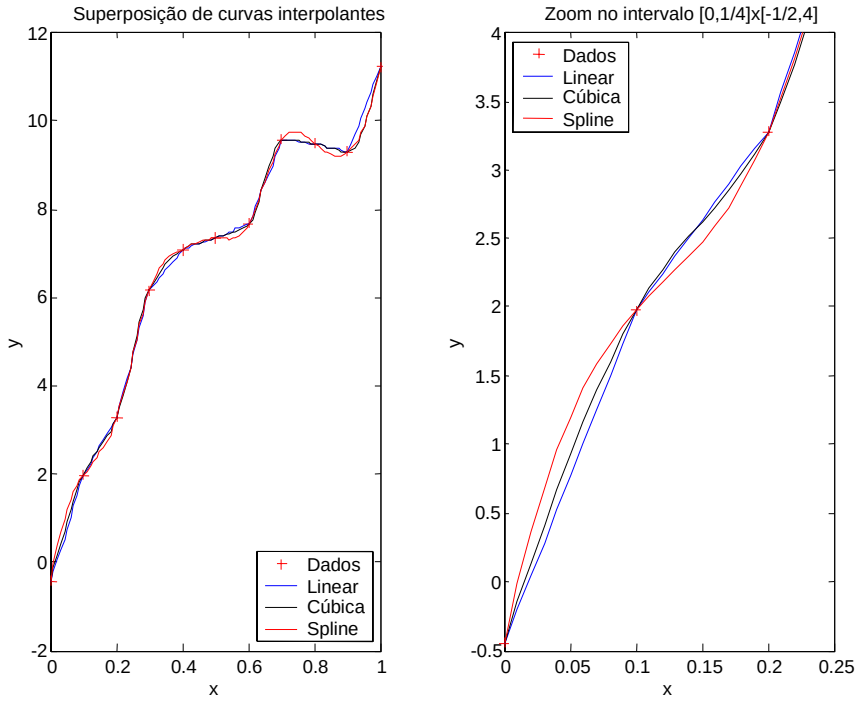

Figura 22: Comando interp1: opções linear, cúbica e spline

```
%ex28
x = 0:0.1:1;
y = [-0.447 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 9.30
11.2];
xi = 0:0.01:1;
intlin=interp1(x,y,xi,'linear');
intcubic=interp1(x,y,xi,'cubic');
intspline=interp1(x,y,xi,'spline');
plot(x,y,'r+',xi,intlin,'b-xi,intcubic,'k:',xi,intspline,...
'r-.');
xlabel('x');
ylabel('y');
legend( Dados,'Linear', 'Cúbica','Spline');
```

O seguinte exemplo corresponde a uma interpolação bidimensional. Dada uma malha discreta de valores que medem a profundidade do solo marinho, matriz *z*, de uma

área retangular de 4x6 quilômetros quadrados, utiliza-se interpolação tipo *cubic* e *nearest*, figura 23.

```
%ex29
x = 0:0.5:4;
y = 0:0.5:6;
z=-[100 99 100 99 100 99 99 99 100; 100 99 99 100 99 99...
100 99 99;
99 99 98 98 100 99 100 100 100; 100 98 97 97 99 100 100...
100 99;
101 100 98 98 100 102 103 100 100; 102 103 101 100 102 106...
104 101 100;
99 102 100 100 103 108 106 101 99; 97 99 100 100 102 105...
103 102 100;
100 102 103 101 101 102 103 100 99; 100 102 103 102 101...
101 100 99 99;
100 100 101 101 100 100 100 99 99; 100 101 101 100 100 99...
99 99 99;
100 100 100 99 99 100 99 100 99];
mesh(x,y,z);%visualiza malha grossa com nós (x,y,z)
xlabel('Eixo x (km)');
ylabel('Eixo y (km)');
zlabel('Profundidade do oceano - metros');
title('Medidas da profundidade do oceano');
%alguns valores particulares
zi = interp2(x,y,z,2.2,3.3) % linear (default)
zi= interp2(x,y,z,2.2,3.3, 'cubic')
zi= interp2(x,y,z,2.2,3.3, 'spline')
zi= interp2(x,y,z,2.2,3.3, 'nearest')
xi=linspace(0,4,30); % eixo x refinado
yi=linspace(0,6,40); % eixo y refinado
[xxi,yyi]=meshgrid(xi,yi); % malha de todas as possíveis
                          %combinações (xi ,yi)
subplot(2,1,1);% figura da esquerda
zzi=interp2(x,y,z,xxi,yyi,'cubic'); % interpolação tipo cubic
mesh(xxi,yyi,zzi);
xlabel('Eixo x - Quilômetros');
ylabel('Eixo y - Quilômetros');
zlabel('Profundidade do oceano - metros');
title('Interpolação tipo cubic : Malha refinada');
axis([0 4 0 6 -110 -95]);
subplot(2,1,2);
zzi=interp2(x,y,z,xxi,yyi,'nearest'); % interpolação tipo
%nearest
mesh(xxi,yyi,zzi);
xlabel('Eixo x - Quilômetros');
```

```
ylabel('Eixo y - Quilômetros');
zlabel('Profundidade do oceano - metros');
title('Interpolação tipo nearest : Malha refinada');
axis([0 4 0 6 -110 -95]);
```

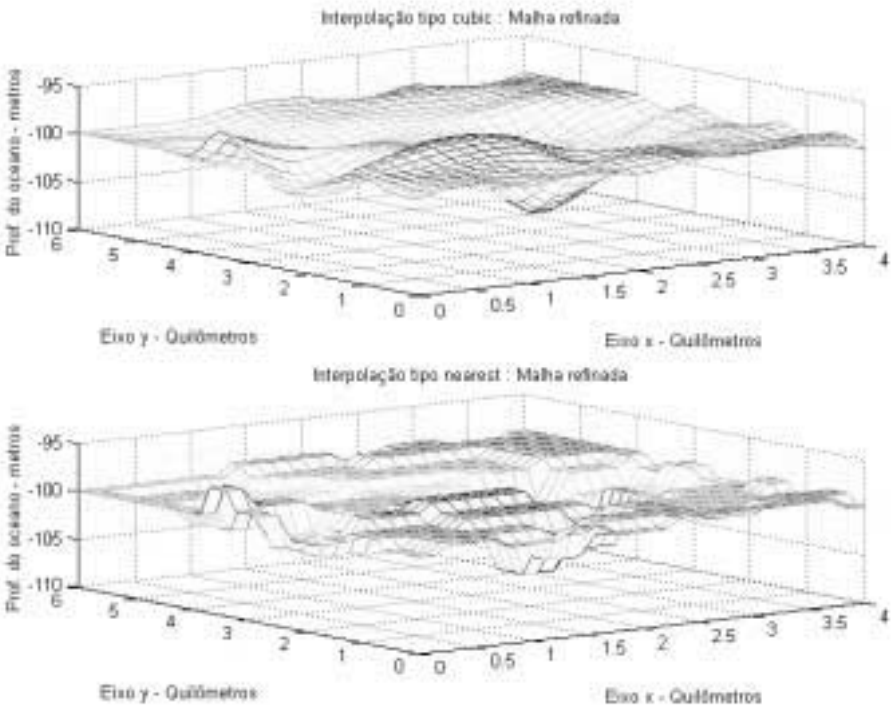

Figura 23: Interpolação bidimensional: cubic e nearest

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 da linguagem C tais como *fopen, fscanf, fclose*. O seguinte exemplo abre o arquivo existente vetor.dat com o atributo 'somente leitura' para leitura de dados. O primeiro valor lido é o comprimento do vetor. Depois o vetor é lido e armazenado no vetor coluna vet e finalmente o arquivo é fechado.

```
%ex30
clear;
fid=fopen('vetor.dat','r');% abre um arquivo existente para
                        %permitir só leitura
nval = fscanf(fid,'%2d',1);%número de valores a ler e´ 1
vet = zeros(nval,1);%cria o vetor coluna vet cheio de zeros
for k=1:nval,
   vet(k) = fscanf(fid,'%5d',1);%le valores em formato
end        %inteiro com no maximo 5 digitos decimais
```

```
fclose(fid);%fecha arquivo
disp(' F I M ');
who % mostra as variáveis globais do ambiente
disp('Vetor lido');
vet % mostra no prompt o vetor lido do arquivo vetor.dat
```

Para abrir ou criar um arquivo usa-se o comando *fopen* **(**'nome de arquivo','opção'**)**. O atributo "r" no campo "opção" do comando *fopen* indica que o arquivo é somente leitura, isto é, não será possível escrever nenhum dado nele. O valor 1 no comando *fscanf* indica que será lido um valor por vez. O formato *%5d* significa que o valor a ser lido é inteiro, indicado pela letra *d*, com no máximo 5 dígitos decimais.

O comando *fopen* retornará um número inteiro positivo que identifica este arquivo para leitura e escrita. Caso o número retornado for -1, isto indicará que o arquivo não existe ou não foi criado.

Opções de abertura ou geração de arquivos:

- "a" : cria ou abre um arquivo existente para escrita, agrega os dados ao final do arquivo
- "a+" : cria ou abre um arquivo existente para leitura ou escrita, agrega os dados ao final do arquivo
- "w" : cria ou abre um arquivo existente só para escrita, descartando o seu conteúdo
- "w+" : cria ou abre um arquivo existente para leitura ou escrita, descartando o seu conteúdo
- "r" : abre arquivo existente só para leitura nao cria o arquivo nem permite escrita nele
- "r+" : abre arquivo existente para leitura e escrita

O seguinte exemplo cria um arquivo cujo nome é entrado por teclado, gera uma matriz 8x8 e copia os valores dela dentro do arquivo em formato ponto fixo.

```
Ex30a
%a seguinte linha pede o nome de um novo arquivo por teclado
arquivo_novo = input('Entre nome de um arquivo a ser criado -
> ','s');
ordem = 8;
mat=randn(ordem);% cria matriz 8x8
mat % mostra a matriz gerada na tela
fid=fopen(arquivo_novo,'w');%cria arquivo novo, caso este
                        %existe o seu conteúdo será perdido
% a seguir escreve-se a matriz no novo arquivo cujo nome é
% guardado na variável arquivo_novo
```

```
for k=1:ordem,
   for l=1:ordem,
      fprintf(fid,'%3.2f\t',mat(k,l));% escreve valores em
%formato ponto fixo com 3 digitos dos quais dois sao decimais
   end
   fprintf(fid,'\n');% pula a linha seguinte
end
fclose(fid);% e´ importante não esquecer de fechar o arquivo
         % caso contrário dados podem ser perdidos
disp('F I M');
who
```

O formato *%3.2f* indica que o valor *mat(k,1)* será escrito com 5 dígitos dos quais 2 dois lugares são reservados após a vírgula. A seqüência *\t* provocará um espaço horizontal entre um valor e outro. A variável *fid* no comando *fprintf* é o inteiro identificador do arquivo onde os dados são escritos, o qual foi retornado quando o arquivo foi aberto ou criado com o comando *fopen*.

Alguns formatos permitidos para imprimir dados são: *%n.me , %n.mE , %n.mf , %nd , %c , %s*. A tabela a seguir descreve o significado destes formatos.

Especificador	Descrição
n	número de dígitos decimais
m	precisão decimal
e, E	formato exponencial
f	ponto fixo
d	inteiros
c	caractere simples
s	palavras (string)

A tabela seguinte descreve uma seqüência de caracteres tal como *\t* e seu significado.

Caractere	Descrição
\b	retrocesso (do cursor)
\f	alimentação do formulário
\n	linha nova
\r	retorno ao inicio da linha de impressão
\t	espaço horizontal
\\	barra invertida

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.

```
%ex32
>> 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 é a sua derivada: f = x^2-2x+3 ; df = 2x-2');
```

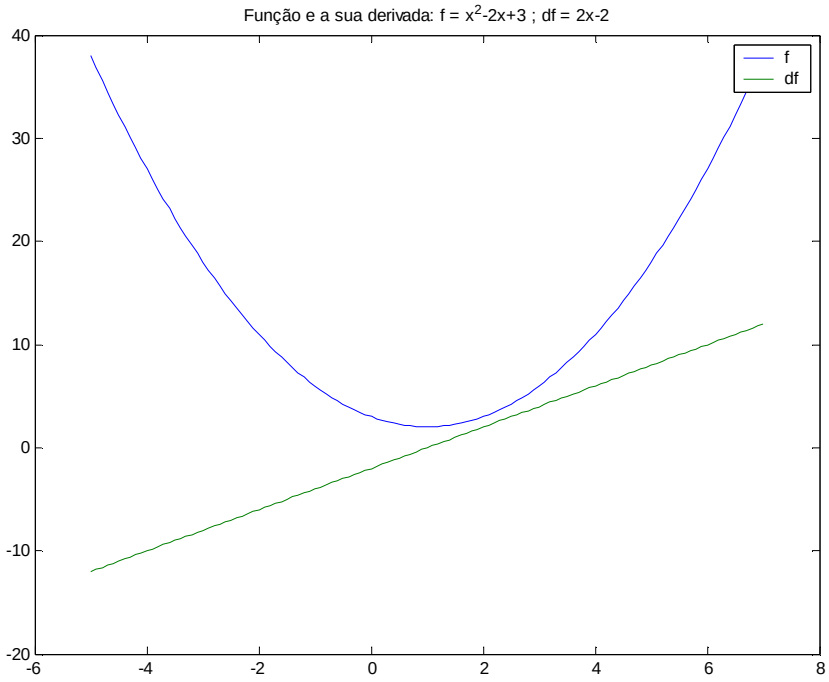

Figura 24: Curvas contínuas de um polinômio é a sua derivada

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 cociente *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*. A figura 24 mostra a função polinomial e a sua derivada analítica.

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'*.

13 Análise numérica de funções

É possível trabalhar com funções definidas em arquivos com extensão ".m" e em funções definidas como texto. Este tipo de definição faz-se necessária quando é desejável utilizar alguns comandos de integração, otimização e resolução de equações diferenciais no Matlab.

Para obter o gráfico de funções definidas como texto utiliza-se o comando *fplot*. Por exemplo:

```
%ex33
>> f = '2*exp(-x).*sin(x)';%função definida como texto
>> fplot(f,[0 8]);%gráfico de f no intervalo [0 8]
>> text(2,0.4,'\leftarrow Função f');%coloca o texto no ponto
                                    %(2,0.4)
```

Pode-se achar o mínimo de uma função dentro de um intervalo usando o comando *fmin*. O máximo de uma função *f* corresponde ao mínimo de *-f*, portanto pode-se usar este mesmo comando para achar o máximo de *f*.

```
>> hold on;% o próximo gráfico é mostrado junto com o gráfico
          %de f
>> f2 = '-2*exp(-x).*sin(x)';% definição de -f
>> fplot(f2,[0 8],'k-');%gráfico de f2 com linha preta
          %contínua
>> text(2,-0.4,'\leftarrow Função -f');%coloca o texto no
                                      %ponto (2,-0.4)
>> xmin = fmin(f,2,5);%acha o mínimo de f no intervalo [2,5]
>> xmax = fmin(f2,0,3);%o máximo de f no intervalo [0,3]
>> x = xmax;%x é a variável independente de f
>> ymax = eval(f,x) %valor máximo de f
>> x = xmin;%x é a variável independente de f
>> ymin = eval(f,x) %valor mínimo de f
>> title('Gráfico das funções f e -f');%título do gráfico
```

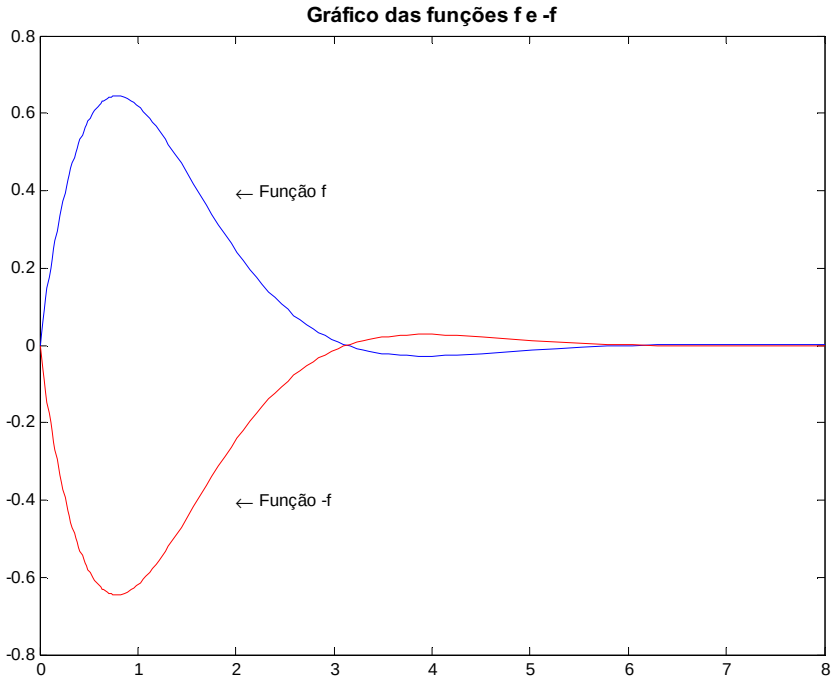

Figura 25: Pontos extremos de f: x máximo 0.7854, x mínimo 3.9270

Os valores máximos e mínimos de *f* são obtidos avaliando *f* em *xmax* e *xmin* respectivamente. Assim *f(xmax=0.7854) = 0.6448* e *f(xmin=3.9270) = -0.0279*.

```
>> xzero = fzero(f,3.1);%calcula o zero de f próximo a
                        %x = 3.1
>> x = xzero;% x = 3.1416
>> yzero = eval(f,x);% yzero = 1.0584e-17
```

Existe um comando chamado *ginput* que funciona em modo interativo. A sintaxe é *[x,y] = ginput(2)*, depois de dado "enter" aparecem duas linhas, uma horizontal e outra vertical que se cruzam encima da janela do gráfico. Este comando permite, clicando com o "mouser" encima do gráfico, obter as coordenadas do ponto encima do qual foi clicado. O argumento 2 significa que serão pedido as coordenadas (x,y) de dois pontos.

Os mesmos comandos acima podem ser usados para funções definidas em arquivos executáveis ".m". Por exemplo:

```
%Arquivo func2.m
function y=func2(x)
y = 2*exp(-x).*sin(x);
```

Depois em modo interativo (ou em um arquivo principal .m):

```
>> fplot('func2',[0 8]);%Grafica a função y definida no
arquivo func2.m
>> xmin = fmin('func2',0,8);% procura o mínimo de y no
%intervalo [0 8]
>> xzero = fzero('func2',3.1);%acha o zero de y próximo ao
                              %ponto 3.1
```

Obtendo-se *xmin = 3.9270* e *xzero = 3.1416*, como anteriormente. A figura 25 mostra os gráficos de *f* e *-f.*

14 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 métodos.

14.1 Integração

Na versão 6.0 do 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+x2)) %antiderivada de 1/(1+x^2)
ans =
atan(x)
>> int(sin(alpha*u),alpha) % integra com respeito a alpha e
                           %não u
ans =
-1/u*cos(alpha*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*(sin(t)2-4)
>> int([exp(t),exp(alpha*t)]) % integra o par c/r a variável
                              %comum t
ans =
[ exp(t), 1/alpha*exp(alpha*t)] %duas integrais, respectivas
                               %componentes
>> A = [cos(x*t),sin(x*t); -sin(x*t),cos(x*t)];%dois pares de
                                                %integrais
>> int(A,t)
ans =
[ sin(x*t)/x, -1/x*cos(x*t)] % integrais do primeiro par c/r
                            %a t
[ 1/x*cos(x*t), sin(x*t)/x] %integrais do segundo par c/r a t
```

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.

$$\int_a^b f(x)dx \tag{8}$$

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 x^2 no intervalo [0,1]
>> quad('func',a,b);%integra x^2 no intervalo [0,1]
>> quad(@func,a,b);%integra x^2 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
```

Formas válidas são:

```
>> dblquad(inline('x.*y'),0,1,0,1);
>> dblquad(@func4,0,1,0,1);%func4.m define uma function
                           %retornando f acima
```

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

Para aumentar a precisão na visualização do resultado numérico pode-se usar o comando *format*, por padrão o formato numérico é *short* com 4 dígitos após a vírgula. O formato *long* produz a seguinte saída:

```
>> format long;
>> 1/12300123453560
ans =
   8.129999701023911e-14
>> format short; % é a saída default
>> 1/12300123453560
ans =
   8.1300e-14
```

O seguinte exemplo corresponde a um problema mais elaborado onde define-se uma função por partes utilizando o comando *if-end* e depois integrando esta função. Vários cuidados devem ser observados para não obter resultados errôneos.

O seguinte arquivo define por partes uma função contínua na reta toda.

```
% arquivo funcao.m
% Este exemplo define uma função por partes.
```

```
% A variável x deve ser escalar para validar
% corretamente a condição if.
function f = funcao_por_partes(x)
if x >= 0 & x <= 1   % [0,1]
   f = x;
elseif x >= 1   % [1,+∞]
   f = -0.2*x.3+1.2;
elseif x <= 0   % [-∞,0]
   f = -sin(x);
end
```

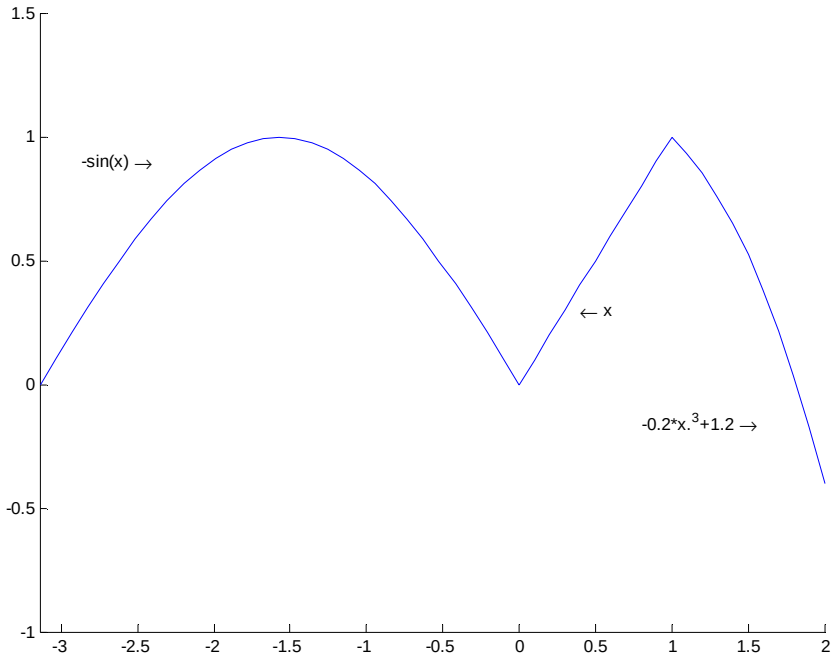

Figura 26: Gráfico de uma função definida por partes

Se esta função deve ser integrada então isto deve ser feito por partes. Por exemplo, para integrar *f* no intervalo [-pi,2], deve-se integrar em 3 partes, uma integral para cada um dos seguintes intervalos: [-pi,0], [0,1] e [1,2].

```
disp('Este arquivo integra a função definida em funcao.m');
disp('no intervalo [-pi,2]');
q1 = quad(@funcao,-pi,0);
q1
q2 = quad(@funcao,0,1);
```

```
q2
q3 = quad(@funcao,1,2);
q3
q = q1+q2+q3;
q
disp(['O valor da integral é : ',num2str(q)]);
```

Para fazer o gráfico da função *f* as mesmas condições devem ser observadas.

```
%ex36
hold on;
axis([-pi 2 -1 1.5]);
x = -pi:pi/30:0;
plot(x,funcao(x));
x = 0:0.1:1;
plot(x,funcao(x));
x = 1:0.1:2;
plot(x,funcao(x));
text(-2.7,0.9,' {-sin(x)} \rightarrow');
text(0.4,0.3,'\leftarrow {x}');
text(1.6,0.5,'\leftarrow {-0.2*x.3+1.2}');
```

14.2 Diferenciacão

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

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \tag{9}$$

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

$$\frac{df}{dx} = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0} \tag{10}$$

Quando fixa-se x_0, a derivada na curva neste ponto é a inclinação da tangente da curva no ponto $(x_0, f(x_0))$.

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

```
%ex37
>> syms x;%declara x simbólico
>> df = diff('x^ 2',x);%deriva simbolicamente
>> df
df =
2*x
>> diff('cos(a*x)',x)
ans =
-sin(a*x)*a
```

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 =
-sin(a*x)*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.

```
%ex38
x = linspace(-1,1,30);%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');
```

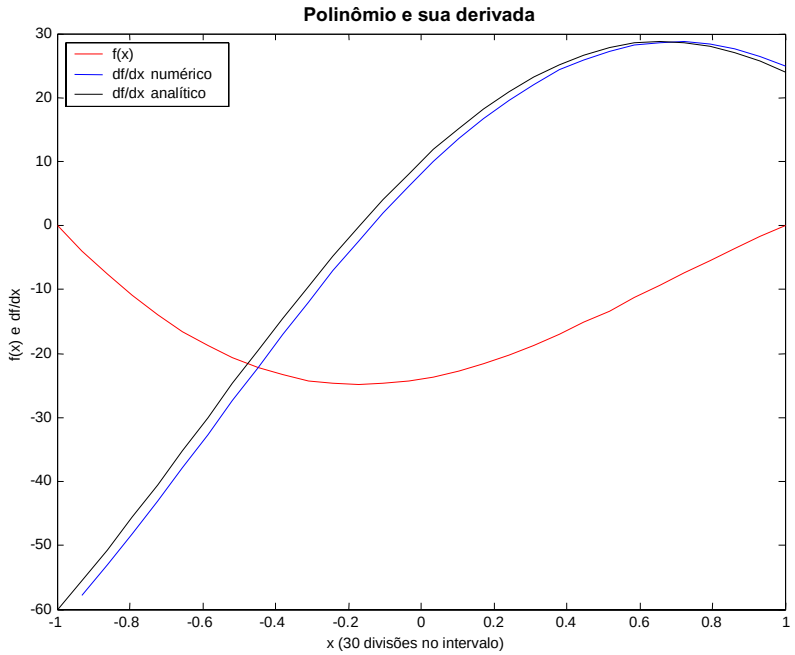

Figura 27: Derivada numérica: 30 divisões do intervalo [-1,1]

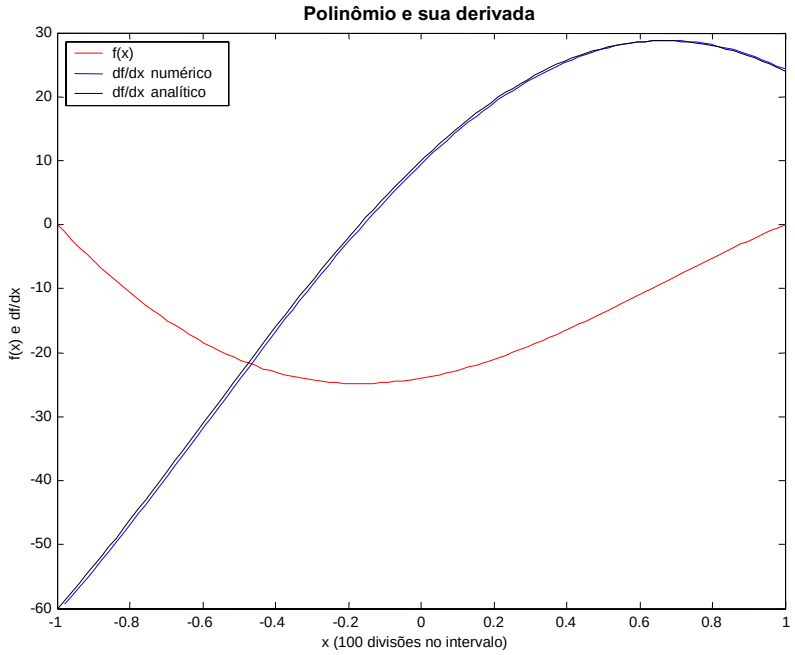

Figura 28: Derivada numérica: 100 divisões no intervalo [-1,1]

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 figura 27 foram tomados 30 divisões do intervalo [-1,1]. A figura 28 mostra a mesma situação anterior, mas agora foram consideradas 100 divisões no intervalo. Percebe-se claramente a diferença de precisão entre os casos considerados.

15 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. Aqui serão exemplificados os comandos *ode23* e *ode45* 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 primeiro exemplo 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):

$$y' = g_1(x, y) = 3x^2, [2,4] \tag{11}$$

$$\text{Condição inicial} : y(2) = 0.5 \tag{12}$$

$$\text{Solução analítica} : y = x^3 - 7.5 \tag{13}$$

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:

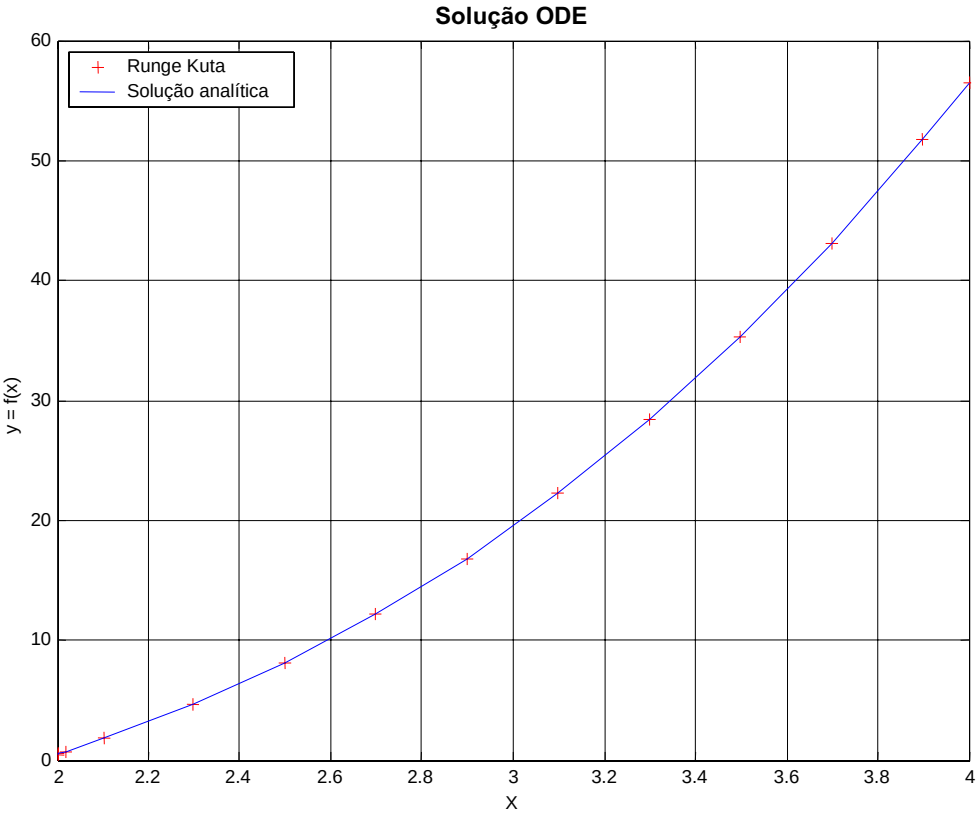

Figura 29: Gráfico das soluções exata e numérica da equação y'=3x2

```
%ex39
% 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 método ode23
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 29.

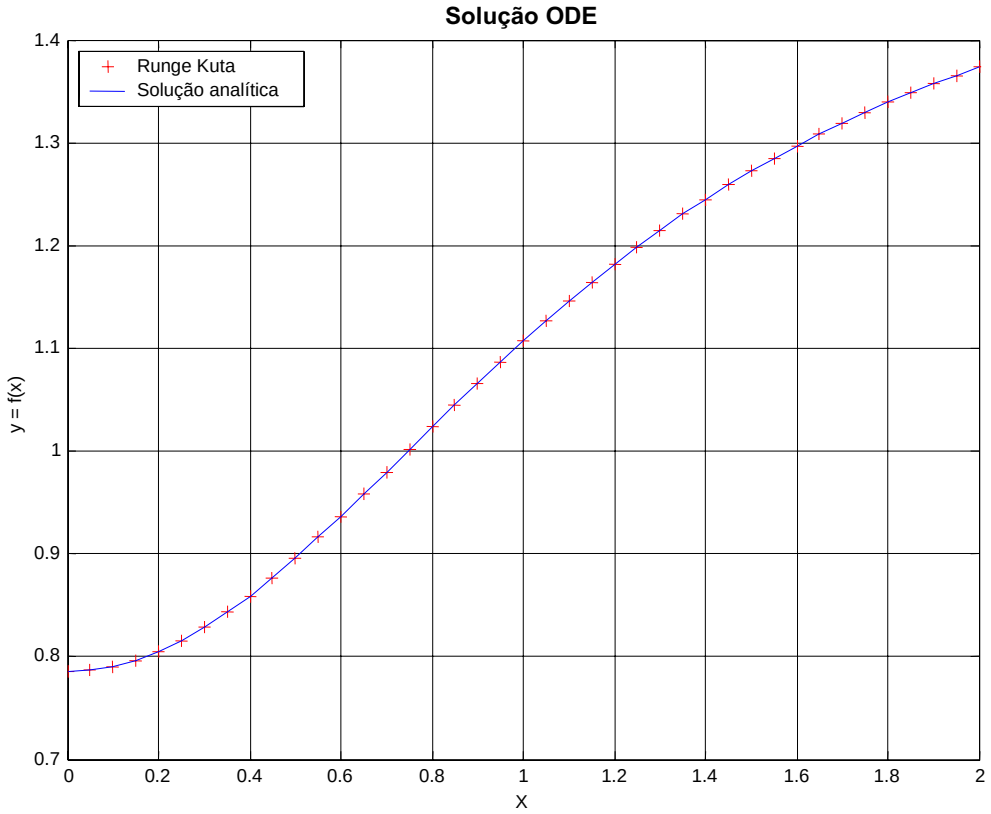

Figura 30: Soluções numérica e exata da equação y'=2xcos^ 2(y)

No segundo exemplo também é conhecida a solução exata. Resolver a equação:

$$y' = g_2(x, y) = 2x\cos^2(y) \text{ em } [0,2] \tag{14}$$

$$\text{Condição inicial} : y(0) = \pi / 4 \tag{15}$$

$$\text{Solução analítica} : y = ArcTan(x^2 + 1) \tag{16}$$

Crie o arquivo ".m" contendo a *function* definida pelo lado direito da equação.

```
function dy = g2(x,y)
dy = 2*x.*cos^ 2(y)
```

O arquivo que resolve a equação diferencial pelo método Runge-Kuta é dado a seguir.

```
%ex40
[x,num_y] = ode45('g2',[0,2],pi/4); %solução aproximada
anl_y = atan(x.*x+1); %solução exata
plot(x,num_y,'r+',x,anl_y,'b:');
title('Solução ODE');
xlabel('X');
ylabel('y = f(x)');
legend( 'Runge Kuta','Solução analítica');
grid;
anl_y-num_y % erro cometido
```

O erro cometido pelas soluções analítica e numérica, como mostrado pela última linha, é da ordem 10^{-6}, conforme verificado pela proximidade dos pontos da solução numérica a solução exata na figura 30.

15.1 Equação diferencial ordinária de ordem superior

Quando a equação é de ordem maior do que 1, como no próximo exemplo, ainda é possível utilizar os comandos *ode23*, *ode45*, etc. Para tal deve-se reescrever a equação simples como um sistema linear de equações de primeira ordem. Em qualquer caso sempre o número de condições deve ser igual à ordem da equação.

Exemplo: resolver a seguinte equação de segunda ordem:

$$y''+y=0 \text{ em } [0,2\pi] \tag{17}$$

$$\text{Condição inicial}: y'(0)=1/2,\ y(0)=1 \tag{18}$$

$$\text{Solução analítica}: y=\frac{1}{2}\text{sen}(x)+\cos(x) \tag{19}$$

No caso proposto pode-se escrever:

$$z1 = y, \quad z2 = y' \tag{20}$$

Então das equações e seguem-se:

$$\frac{dz_1}{dx} = z'_1 = y' = z_2 \quad \text{e} \quad \frac{dz_2}{dx} = z'_2 = y'' = -z_1 \tag{21}$$

Crie-se o arquivo contendo a seguinte function.

```
%EDO de ordem 2
%função teste para ode45
function dz = func3(x,z)
dz = [z(2), -z(1)];
```

O arquivo contendo a resolução pelo Matlab e ode45 escreve-se da seguinte forma.

```
%ex41
intervalo=[0 2*pi]; % intervalo em que a equação será
                    %resolvida.
zo=[1 1/2]; % Condições iniciais em x=0
[x,z]=ode45('func3',intervalo,zo);% retorna os valores da
                           %variável independente x e um
                           %vetor z de 2 colunas, sendo a
                           %primeira correspondente a z1 e a
                           %segunda a z2.
yan= 0.5*sin(x) + cos(x); %solução exata
dyan = 0.5*cos(x)-sin(x); %derivada da solução exata
subplot(2,1,1);
plot(x,z(:,1),'r+:',x,yan,'b-');%gráfico de soluções exata e
                          %numérica (funções)
legend('y numérico', 'y analítico');
subplot(2,1,2);
plot(x,z(:,2),'r+:',x,dyan,'b-');%gráfico de soluções exata e
                           %numérica (derivadas)
legend('dy/dx numerica','dy/dx analítica');
yan-z(:,1) % erro no valor da função
dyan-z(:,2) % erro na derivada
```

O erro cometido, tanto no valor da função como no valor da derivada, nos pontos escolhidos pelo método foram da ordem no máximo 10^{-3}, como pode ser verificado pela saída das últimas duas linhas do arquivo. Por exemplo, a saída para *(dyan-z(:,2))'* toma a forma:

```
ans =
   1.0e-03 *
Columns 1 through 10
0 0.0000 0.0000 0.0000 0.0000 0.0099 0.0139 0.0073 0.0035
0.0422
Columns 11 through 20
0.0610 0.0315 0.0061 0.0419 0.0606 0.0220 -0.0181 0.0004
0.0117 -0.0207
Columns 21 through 30
-0.0590 -0.0633 -0.0614 -0.0725 -0.0913 -0.1132 -0.1178 -
0.0994 -0.0876 -0.1148
Columns 31 through 40
-0.1202 -0.0759 -0.0346 -0.0538 -0.0556 -0.0002 0.0560 0.0529
0.0556 0.1003
Columns 41 through 45
0.1490 0.1607 0.1629 0.1803 0.2046
```

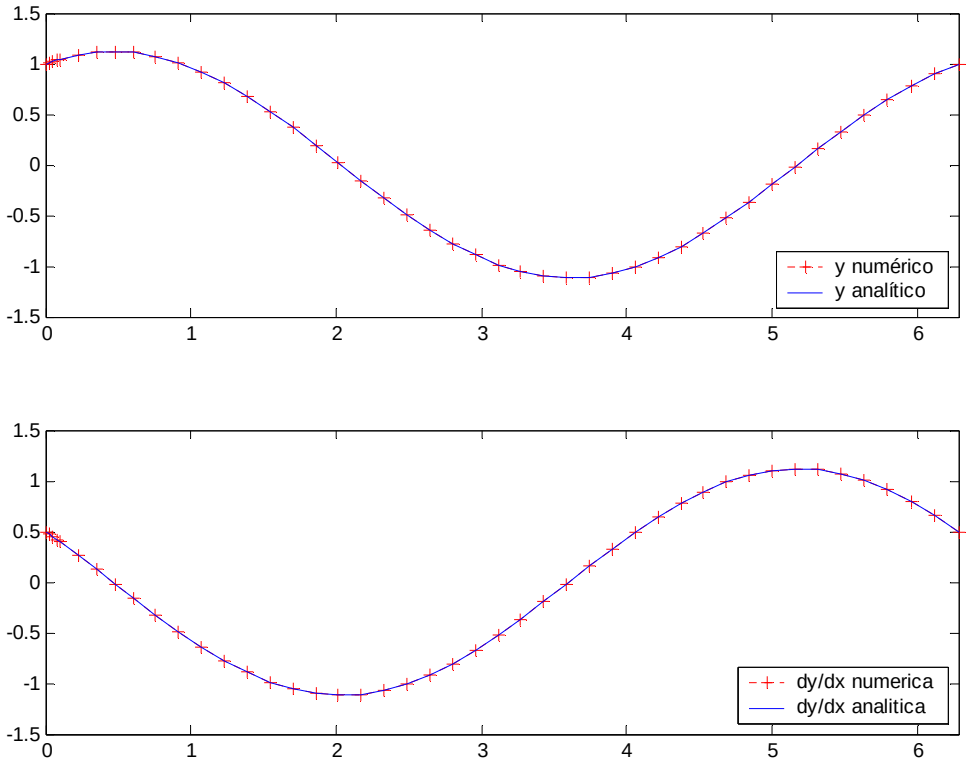

Figura 31: Comparação solução numérica e exata: função e derivada

A figura 31 mostra a boa concordância entre os dados numéricos e a solução exata, tanto para o valor da função como da sua derivada.

Observações

A ordem de uma equação diferencial refere-se à derivada mais alta que aparece na equação. O exemplo a seguir corresponde a uma equação de quinta ordem.

$$y^{(5)} + y^{(2)} + y = x^7 \tag{22}$$

onde $y = y(x)$ e $y^{(n)} = \frac{d^n y}{dx^n}$.

O grau de uma equação diferencial define-se como o expoente da derivada de maior ordem. O seguinte exemplo corresponde a uma equação de grau 3.

$$\left(y^{(5)}\right)^3 + y^{(2)} + y^4 = x^7 \tag{23}$$

onde $\left(y^{(5)}\right)^3 = y^{(5)} * y^{(5)} * y^{(5)} * y^{(5)} * y^{(5)}$ e $y^4 = y * y * y * y$.

16 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 (veja subseção 9.1). A decomposição LU serve, por exemplo, para determinar o determinante de uma matriz grande, ou a matriz inversa.

16.1 Fatorização triangular - LU

Como já vimos na subseção 9.1, 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*

16.2 Decomposição - QR

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

$$Q^T * Q = I \tag{24}$$

Isto quer dizer que *Q* e a sua transposta Q^T são inversas. As linhas da matriz *Q* possuem a seguinte propriedade: denote por q_i a linha *i* de *Q*, então $q_i * q_j^T = \delta_{ij}$, 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), equação 25. O mesmo aplica-se as colunas de *Q*.

$$(q_i q_i) = \sum_{k=1}^{n} q_i^k * q_j^k = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases} \tag{25}$$

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
```

16.3 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*.

16.4 Autovalores e autovetores

Entre outras aplicações o problema de autovalores encontra utilidade na transmissão de sinais de comunicação. Um autovalor λ e um autovetor x estão associados a uma matriz A da seguinte maneira:

$$Ax = \lambda x \tag{26}$$

Logo a determinação destes esta associada à resolução de um sistema linear de equações. A primeira vista uma solução trivial deste sistema é $x = 0, \lambda = 0$. Mas podem existir valores não nulos que também resolvem o sistema. No Matlab os autovalores e autovetoers são 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:

$$A * V = V * D \tag{27}$$

Ou em termos de vetores

$$A * v_i = v_i * d_{ii}, i = 1,2,...,n \tag{28}$$

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

17 Comentário final

Existem ainda um grande número de funções e métodos definidos para uso no Matlab que aqui não foram mencionados, até por uma questão de espaço e tempo. Pode-se dizer que a capacidade para trabalhar com o desenvolvimento da parte gráfica é um dos pontos

fortes do Matlab. Por último quer-se salientar que este estudo pretende introduzir ao pesquisador na filosofia de trabalho do Matlab como ferramenta de ajuda científica, e que um estudo apurado dependerá da dedicação à pesquisa na área de atuação de cada usuário.

18 Referências Bibliográficas

[1] Curso de MATLAB 5.1 : Introdução à Solução de Problemas de Engenharia, Apostila preparada pela Faculdade de Engenharia da UERJ, Universidade do Estado do Rio de Janeiro, 2000.

[2] Versão de Estudante, Curso de Matlab 5. Guia do usuário. D. Hanselman e B. Littlefield, Makron Books do Brasil, São Paulo - Brasil, 1999.

[3] MATLAB, The Language of Technical Computing. Matlab Function Reference. Computation, Visualization and Programming. The Math Works Inc. Volumes I, II, e III (Revised for 6.0).