ÍNDICE

1. O MATLAB

Sua primeira versão foi escrita na Universidades do Novo México e na Universidade de Stanford no final da década de 1970. Destinava-se aos cursos de:

- Teoria Matricial
- Álgebra Linear
- Análise Numérica

1.1. Características

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 os problemas e as soluções são expressos em uma linguagem matemática familiar.

O MATLAB tem como elemento de dados básico uma matriz que não requer dimensionamento. Isso permite solucionar muitos problemas computacionais, principalmente os que envolvem formulações matriciais ou vetoriais, em uma fração do tempo que seria necessário para escrever um programa em uma linguagem como C ou FORTRAN.

1.2. Usos Típicos

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

Além disso, o MATLAB permite a construção de ferramentas reutilizáveis. Você pode facilmente criar suas próprias funções e programas especiais (conhecidos como arquivos M) em linguagem MATLAB.

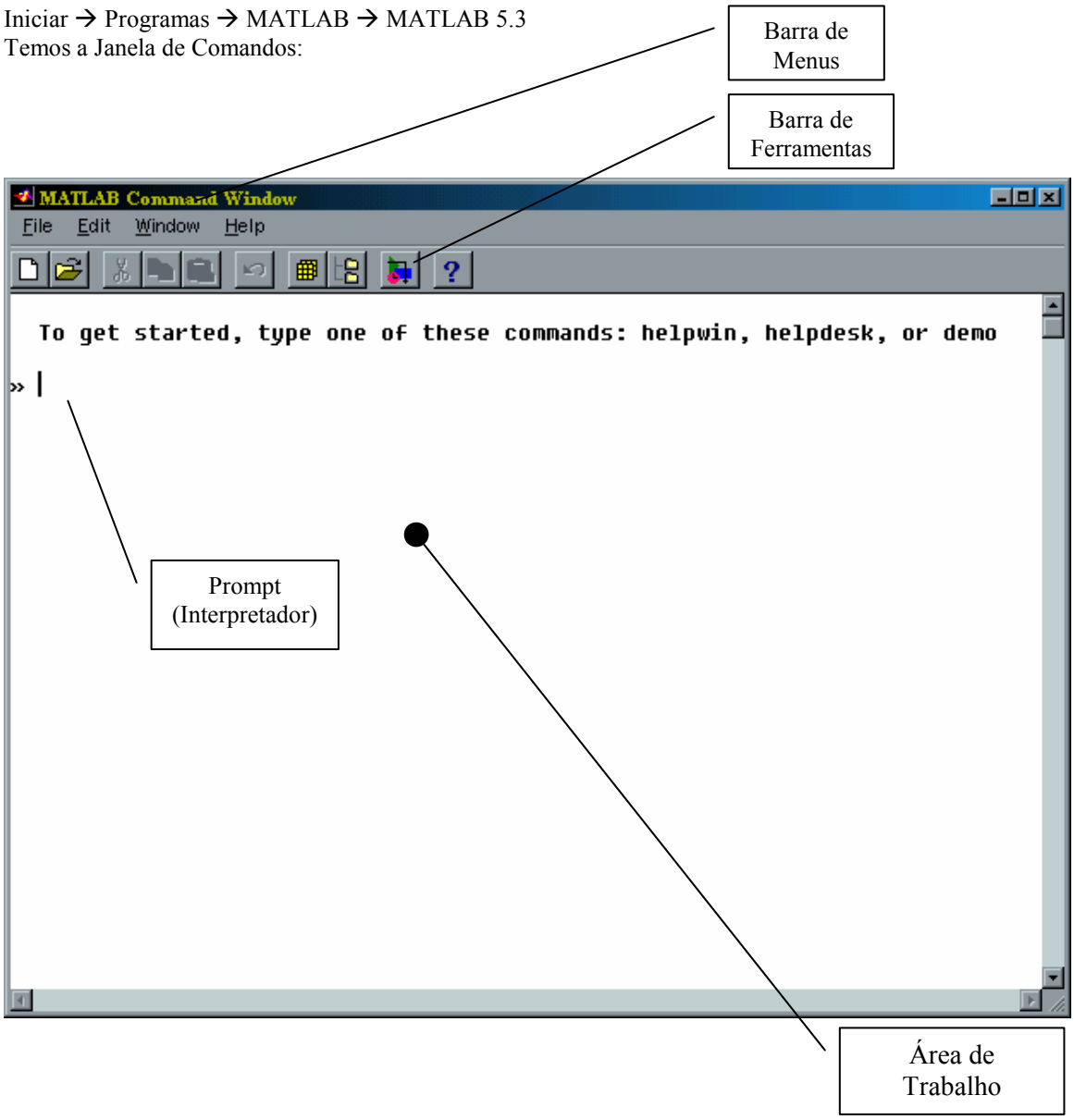
Denomina-se **toolbox** o conjunto de ferramentas para a resolução de problemas em áreas específicas. O MATLAB dispõe de diversas **toolboxes** desenvolvidas por alguns dos maiores pesquisadores do mundo.

1.3. Algumas Toolboxes Disponíveis

- Processamento de Sinais
- Identificação de Sistemas
- Otimização
- Sistemas de Controle
- Controle Robusto
- Redes Neurais
- Spline
- Controle Linear e Não-Linear
- Aplicações Financeiras
- Lógica Nebulosa
- Processamento de Imagens
- Matemática Simbólica
- Etc.

2. Trabalhando com o MATLAB

Iniciar → Programas → MATLAB → MATLAB 5.3
Temos a Janela de Comandos:



3. Operadores Mais Comuns

- Adição → +
- Subtração → -
- Multiplicação → *
- Divisão → / ou \
- Potenciação → ^

As expressões são calculadas da esquerda para a direita com a operação de potenciação tendo a maior ordem de precedência, seguida da multiplicação e divisão (ambas tendo a mesma ordem de precedência), por sua vez seguidas da adição e da subtração (ambas tendo a mesma ordem de precedência). Pode-se usar parênteses para alterar essa ordem. Nesse caso, o cálculo inicia-se nos parênteses amis internos e procede de dentro para fora.

Problema 1) Maria vai a papelaria e compra 4 borrachas por 25 centavos cada, 6 blocos de nota por 52 centavos cada e 2 rolos de fita por 99 centavos cada. Quantos itens Maria comprou e quanto eles custaram?

Inicialmente, podemos adotar o mesmo método da calculadora:

```
» 4+6+2

ans =

   12

» 4*25 + 6*52 + 2*99

ans =

  610
```

Observe que o MATLAB não se importa com espaços e chama o resultado de *ans* (abreviação de answer).

4. Refazendo Um Comando

Utilize a seta para cima do teclado para chamar os comandos anteriores ao prompt do MATLAB.
Utilize a seta para baixo do teclado para chamar os comandos posteriores ao prompt do MATLAB.
Utilize as setas laterais para mover o cursor dentro da linha de comando no prompt do MATLAB.
Digite o começo do comando e a seta para cima.
O Mouse pode ser usado em conjunto com a Área de Transferência para cortar, copiar e colar texto no prompt de comando.

5. Criando Variáveis

Alternativamente, podemos resolver problemas armazenando as informações em variáveis.

```
» borrachas=4

borrachas =

             4

» blocos=6

blocos =

   6

» fitas=2;
» itens=borrachas+blocos+fitas

itens =

   12

» custo=borrachas*25+blocos*52+fitas*99

custo =
```

610

O ponto-e-vírgula depois do fim da linha **» fitas=2;** diz ao MATLAB para processar a linha mas não apresentar a resposta.

Podemos ainda, perguntar qual foi o preço médio dos itens comprados.

```
» custo_medio=custo/itens

custo_medio =

   50.8333
```

5.1. Olhando as Variáveis

Para verificar o valor da variável *fitas*, basta introduzir o nome da mesma após o prompt:

```
» fitas

fitas =

    2
```

Para ver uma lista de todas as variáveis criadas, utilize o comando *who*:

```
» who

Your variables are:

ans         borrachas     custo_medio    itens
blocos      custo         fitas
```

As variáveis ficam no "Espaço de Trabalho" (Workspace) do MATLAB.

5.2. Nomes de Variáveis

- São sensíveis a maiúsculas e minúsculas. (Itens, itens, itEns e ITENS são todas variáveis MATLAB diferentes).
- Podem conter até 31 caracteres.
- Devem começar com uma letra, seguida de um número qualquer de letras, algarismos ou sublinhados. Caracteres de pontuação não são permitidos por terem um significado especial para o MATLAB. Não são permitidos caracteres acentuados, nem a cedilha e nem o ñ espanhol.

Nota: o comando *salvar* utiliza às vezes o nome da variável como nome de arquivo. Algumas plataformas não aceitam arquivos com mais de oito caracteres.

5.3. Variáveis Especiais

ans Variável-padrão usada para resultados.
pi Razão entre o perímetro e o diâmetro de uma circunferência.
eps Menor número que, somado a 1, resulta em um número de ponto flutuante maior do que 1.

```
» eps

ans =

   2.220446049250313e-016
```

inf Infinito.

```
» 2 / inf
ans =
   0
```

NaN (ou) **nan** Não-numerico.

```
» inf/inf
ans =
  NaN
```

i (e) **j** i = j = $\sqrt{-1}$ (Número Imaginário).

```
» a = 3i
a =
     0 + 3.0000i
```

5.4. Modificando as Variáveis

```
» borrachas=4;
» blocos=6;
» fitas=2;
» itens=borrachas+blocos+fitas

itens =

   12

» borrachas=6

borrachas =

    6

» itens

itens =

   12
```

Observe que, modificando o número de borrachas para 6, o valor de *itens* não se alterou. Isso ocorre porque o MATLAB não recalcula o número de itens baseado no novo valor da variável *borrachas*. Quando o MATLAB faz os cálculos, ele utiliza os valores conhecidos no momento em que o cálculo foi solicitado. Neste caso, para recalcular o número de itens, o custo total e o custo médio seria necessário chamar os comandos MATLAB apropriados e solicitar ao MATLAB que os recalculasse.

5.5. Eliminando Variáveis

» clear borrachas
exclui somente a variável *borrachas*.

» clear custo itens
exclui as variáveis *custo* e *itens*.

» clear cl*
usa o caractere-chave * para excluir todas as variáveis que comecem pelas letras *cl*.

6. Limpando o Espaço de Trabalho

As variáveis do espaço de trabalho do MATLAB podem ser excluídas usando-se o comando *clear*.

6.1. Limpando Toda a Área de Trabalho

```
» clear
» who
»
```

Exclui todas as variáveis do espaço de trabalho sem possibilidade de recuperação.
Outras opções para a função *clear* podem ser encontradas por meio do comando *help*.

7. Comentários e Pontuação

Todo texto depois do sinal de porcentagem **"%"** é considerado um comentário.

```
» borrachas=4     %Número de borrachas

borrachas =

     4
```

Pode-se colocar mais de um comando em uma linha, separando-os por vírgula *","* ou ponto-e-vírgula *";"*.

```
» borrachas=4, blocos=6; fitas=2

borrachas =

   4


fitas =

   2
```

8. Quebrando Uma Linha

Para a quebra de linha utiliza-se os tres pontos "...".

```
» custo_medio=custo/...
itens

custo_medio =

   50.8333
```

Não se pode utilizar "..." para dividir o nome de uma variável ou um comentário.

9. Números Complexos

Utiliza-se **"i"** ou **"j"** para indicar a parte imaginária de um número complexo. Os mesmos devem ser escritos junto ao número, sem separação.

```
» a = 2 + 3i, b = 5 -2j

a =

   2.0000 + 3.0000i


b =

   5.0000 - 2.0000i
```

Em geral, as operações com números complexos dão por resultado números complexos. Diferente de outras linguagens, o MATLAB trata da mesma forma as variáveis reais e complexas.

```
» c = 3 * (2 -sqrt(-4)*3)

c =

   6.0000 -18.0000i
```

Para transformar uma variável ou expressão em complexa deve-se utilizar a multiplicação.

```
» d = 6 + sin(pi/4)*i

d =

   6.0000 + 0.7071i
```

As operações matemáticas com números complexos são escritas da mesma maneira que aquelas com números reais.

```
» e=(a+b)/c     %Dos dados anteriores

e =

   0.0667 + 0.3667i

» verifique=j^2     %O quadrado de sqrt(-1) tem de ser igual a -1!
```

verifique =

-1

9.1. Identidade de Euler

Problema 2) Como exemplo de aritmética complexa, considere a identidade de Euler (pronuncia-se Oiler), que relaciona a forma **polar** à forma **retangular** de um número complexo:

$$C = M\angle\theta \equiv M \cdot e^{j\theta} = a + bi$$

em que a forma polar é dada por um **módulo M** e um **ângulo** θ e a retangular é dada por $a + bi$. As relações entre essas fórmulas são:

$$M = \sqrt{a^2 + b^2}$$
$$\theta = \tan^{-1}(b/a)$$
$$a = M\cos\theta$$
$$b = M\text{sen}\theta$$

No MATLAB, a conversão entre polar e retangular utiliza as funções *real*, *imag*, *abs* e *angle*. A função *abs* calcula o módulo de números complexos ou o valor absoluto de números reais. A função *angle* calcula o ângulo de um número complexo em radianos. As funções *real* e *imag* informam respectivamente a parte real e imaginária do número.

A função *conj* informa o conjugado complexo de um número complexo.

```
» c = 2 - 3i

c =

  2.0000 - 3.0000i

» mod_c = abs(c)

mod_c =

   3.6056

» angulo_c_r = angle(c)

angulo_c_r =

  -0.9828

» angulo_c_g=angulo_c_r*180/pi

angulo_c_g =

 -56.3099

» a = real(c)

a =
```

```
    2

» b = imag(c)

b =

   -3

» M = sqrt(a^2 + b^2)

M =

   3.6056

» a = M*cos(angulo_c_r)

a =

    2

» b = M*sin(angulo_c_r)

b =

   -3

» d = conj(c)

d =

  2.0000 + 3.0000i
```

10. Funções Científicas

sqrt(x) Raiz quadrada

```
» x = 3.4

x =

   3.4000

» sqrt(x)

ans =

   1.8439
```

sin(x) Seno

```
» sin(x)

ans =

  -0.2555
```

cos(x) Co-seno

```
» cos(x)

ans =

  -0.9668
```

tan(x) Tangente

```
» tan(x)

ans =

   0.2643
```

log(x) Logaritmo natural

```
» log(x)

ans =

   1.2238
```

log10(x) Logaritmo na base dez

```
» log10(x)

ans =

   0.5315
```

exp(x) Exponencial e^x

```
» exp(x)

ans =

  29.9641
```

rem(x,y) Resto da divisão de x por y

```
» a=6; b=10;
» rem(b,a)

ans =

   4

» rem(b,x)

ans =

   3.2000
```

gcd(x,y) Máximo divisor comum dos inteiros x e y

```
» gcd(a,b)

ans =

    2
» gcd(a,x)
??? Error using ==> gcd
Requires integer input arguments.
```

lcm(x,y) Mínimo múltiplo comum dos inteiros x e y

```
» lcm(a,b)

ans =

   30
```

ceil(x) Arredondamento para número inteiro na direção de mais infinito

```
» ceil(x)

ans =

    4
```

floor(x) Arredondamento para número inteiro na direção de menos infinito

```
» floor(x)

ans =

    3
```

round(x) Arredondamento para o número inteiro mais próximo

```
» round(x)

ans =

    3
```

fix(x) Arredondamento na direção de zero

```
» fix(x)

ans =

    3
```

sign(x) Função Sinal: Retorna o sinal do argumento

```
» sign(x)

ans =
```

```
     1

» sign(x*-2)

ans =

    -1

» sign(0)

ans =

     0

» sign(c)

ans =

  0.5145+ 0.8575i
```

Por quê?

```
» help sign     %Ajuda para o uso da função sign

 SIGN   Signum function.
    For each element of X, SIGN(X) returns 1 if the element
    is greater than zero, 0 if it equals zero and -1 if it is
    less than zero.  For complex X, SIGN(X) = X ./ ABS(X).
```

asin(x) Arco seno

```
» x=sqrt(2)/2

x =

    0.7071

» y=asin(x)

y =

    0.7854

» y_graus=y*180/pi

y_graus =

   45.0000
```

acos(x) Arco co-seno

```
» y=acos(x)

y =

    0.7854
```

```
» y_graus=y*180/pi

y_graus =

   45
```

atan(x) Arco tangente

```
» 4*atan(1)     %uma forma de aproximar pi


ans =


   3.1416

» 180/pi*atan(-2/3)

ans =

  -33.6901
```

atan2(y,x) Arco tangente de quatro quadrantes

```
» 180/pi*atan2(2,-3)

ans =

  146.3099

» 180/pi*atan2(-2,3)

ans =

  -33.6901

» 180/pi*atan2(-2,-3)

ans =

 -146.3099
```

Problema 3) Decaimento Radioativo

O polônio, um elemento radioativo, tem uma meia-vida de 140 dias, o que significa que, devido ao decaimento radioativo, a quantidade de polônio restante depois de 140 dias é a metade da original. Começando com 10 gramas de polônio hoje, quanto vai restar depois de 250 dias?

Solução:

Depois de meia-vida, ou 140 dias, restam $10 \cdot 0,5 = 5$ gramas. Depois de duas meias-vidas, ou 280 dias, restam $5 \cdot 0,5 = 10 \cdot 0,5 \cdot 0,5 = 10 \cdot (0,5)^2 = 2,5$ gramas. Portanto, a solução correta estaria entre 5 e 2,5 gramas, e a quantidade restante depois de um certo período de tempo é dada por:

$$\text{quantidade restante} = \text{quantidade inicial} \cdot (0,5)^{\text{tempo/meia-vida}}$$

Solução em MATLAB:

```
» quant_inic=10;
» meia_vid =140;
» tempo=250;
» quant_rest =quant_inic * 0.5^(tempo/meia_vida)

Q_final =

   2.9003
```

Problema 4) Cálculo de Juros

Você decidiu comprar um carro novo por R$ 18.500,00. O vendedor está oferecendo duas opções de financiamento:

a) Taxa de Juros de 0,24% ao mês paga durante 4 anos.

b) Taxa de Juros de 0,74% ao mês paga durante 4 anos, com desconto de fábrica de R$1.500,00.

Qual é o melhor negócio?

Solução:

O pagamento mensal P em um empréstimo de A reais, com uma taxa mensal de juros R, amortizado em M meses, é:

$$P = A\left[\frac{R(1+R)^M}{(1+R)^M - 1}\right]$$

gerando uma quantia total paga de T = P · M.

Solução em MATLAB:

```
» format bank % usa o formato de saída de banco
» A=18500; % total do empréstimo
» M=12*4; % número de meses
» FR=1500; % desconto de fábrica
» %primeira opção de financiamento
» R=0.24/100; % taxa mensal de juros
» P=A*(R*(1+R)^M/((1+R)^M -1)) % pagamento

P =

        408.50

» T1=P*M % custo total do carro

T1 =

     19608.22

» % segunda opção de financiamento
» R=0.74/100; % taxa mensal de juros
» P=(A-FR)*(R*(1+R)^M/((1+R)^M -1)) % pagamento

P =

      422.08
```

```
» T2=P*M %custo total do carro

T2 =

    20259.73

» Dif=T2-T1

Dif =

     651.51
```

Com base no resultado, percebemos que a melhor opção de financimento é a primeira.

11. Olhando as Variáveis mais Detalhadamente

```
» who

Your variables are:

meia_vida     quant_rest
quant_inic    tempo

» whos
  Name          Size         Bytes  Class

  meia_vida     1x1              8  double array
  quant_inic    1x1              8  double array
  quant_rest    1x1              8  double array
  tempo         1x1              8  double array

Grand total is 4 elements using 32 bytes
```

Além dessas funções, o item *Show Workspace* do menu *File* cria uma janela GUI, chamada de *Workspace Browser*, que contém as mesmas informações fornecidas pelo comando *whos*. Além disso, ela permite que você apague variáveis selecionadas. Esta janela também é criada quando pressionamos o item *Workspace Browser* na barra de ferramentas da janela de *Comandos*.

12. Salvando o Trabalho

O comando *diary* armazena, no diretório corrente (c:\MATLABr11\work\), a entrada de dados do usuário e tudo aquilo que é exibido na janela de comandos em um arquivo de texto no formato ASCII. Arquivos ASCII podem ser editados em qualquer editor comum de textos.

```
» diary aula1      %Armazena o diário no arquivo aula1.

» clear
» a=25

a =

     25.00

» b=2;
```

» diary off %Encerra o comando diary e depois fecha o arquivo.

O comando *save* permite o armazenamento de uma ou mais variáveis do espaço de trabalho do MATLAB no formato do arquivo de sua escolha.

File → Save Workspace As

» save

Saving to: MATLAB.mat
armazena todas as variáveis no formato binário do MATLAB no arquivo *MATLAB.mat*.

» save aula1
armazena todas as variáveis no formato binário do MATLAB no arquivo *aula1.mat*.

» save aula1 a b
armazena as variáveis *a* e *b*, em formato binário, no arquivo *aula1.mat*.

» save aula1 a b -ascii
armazena as variáveis *a* e *b*, em formato ASCII de 8 dígitos, no arquivo *aula1.mat*.

» save aula1 a b –ascii -doble
armazena as variáveis *a* e *b*, em formato ASCII de 16 dígitos, no arquivo *aula1.mat*.

13. Recuperando o Trabalho

O comando *load* usa a mesma sintaxe do comando *save*, com a diferença de que é empregado para recuperar variáveis no espaço de trabalho do MATLAB.

File → Load Workspace

» load

Loading from: MATLAB.mat
recupera todas as variáveis no formato binário do MATLAB do arquivo *MATLAB.mat*.

14. Formato de Saída

Os cálculos no MATLAB são feito com precisão dupla, mas o formato de apresentação da saída pode ser controlado pelos comandos abaixo:

format short	5 dígitos	**35.833**
format long	16 dígitos	**35.83333333333334**
format short e	5 dígitos + expoente	**3.5833e+01**
format long e	16 dígitos + expoente	**3.583333333333334e+01**
format short g	5 dígitos	**35.833**
format long g	16 dígitos	**35.83333333333334**
format hex	Hexadecimal	**4041eaaaaaaaaaab**
format bank	2 dígitos decimais	**35.83**
format rat	Aproximação racional	**215/6**
format +	Positivo, negativo ou zero	**+**

Uma vez acionado, o comando escolhido permanece até ser mudado.

Os comandos de formatação também podem ser acionados a partir do menu de barra do MATLAB. Selecione a opção Options, dentro dele selecione Numerical Format e mova o cursor sobre o formato desejado.

format compact suprime a maioria das linhas em branco; permite que mais informação seja colocada na tela ou na página. Ele independe de outros comandos de formatação.

15. Gerenciamento de Arquivos

Para verificar qual é o diretório atual pode-se recorrer ao comando *cd*.

» cd

C:\MATLABR11\work

Neste diretório serão salvos todos os arquivos e é nele que o MATLAB procura primeiramente por algum arquivo solicitado pelo usuário.

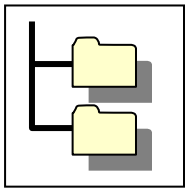
Quando o MATLAB não encontra o arquivo solicitado no diretório atual ele o procura nos diretórios do *path* seguindo a seqüência em que os diretórios aparecem.

» path

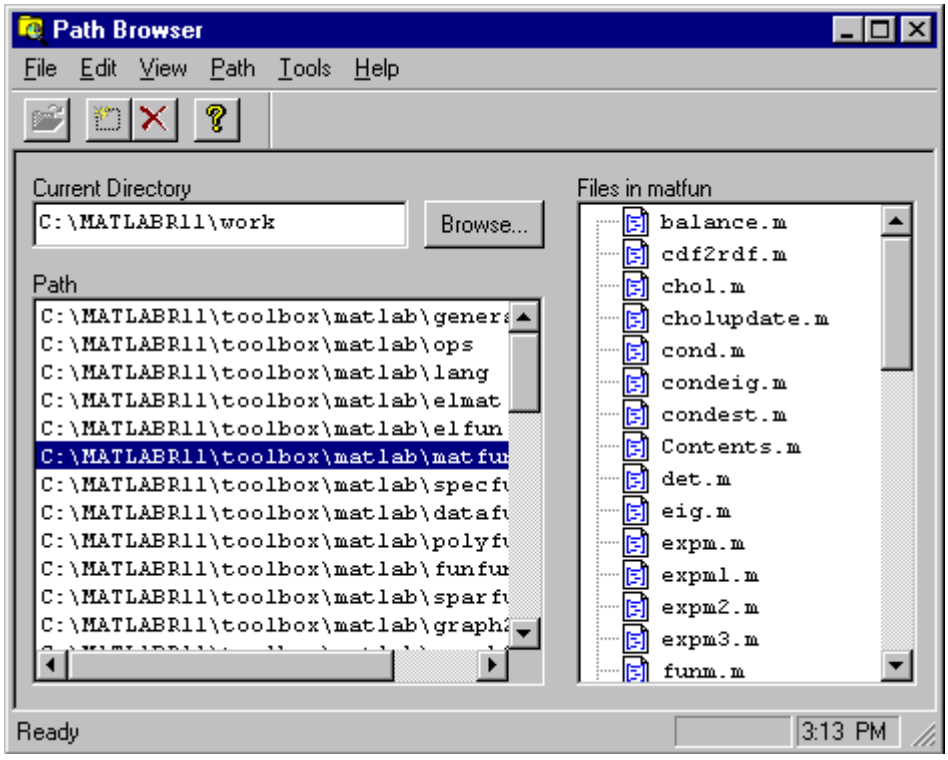
MATLABPATH

C:\MATLABR11\toolbox\MATLAB\general
C:\MATLABR11\toolbox\MATLAB\ops
C:\MATLABR11\toolbox\MATLAB\lang
C:\MATLABR11\toolbox\MATLAB\elmat
C:\MATLABR11\toolbox\MATLAB\elfun
…
C:\MATLABR11\toolbox\simulink\simdemos
C:\MATLABR11\toolbox\simulink\dee
C:\MATLABR11\toolbox\tour
C:\MATLABR11\work
C:\MATLABR11\toolbox\local

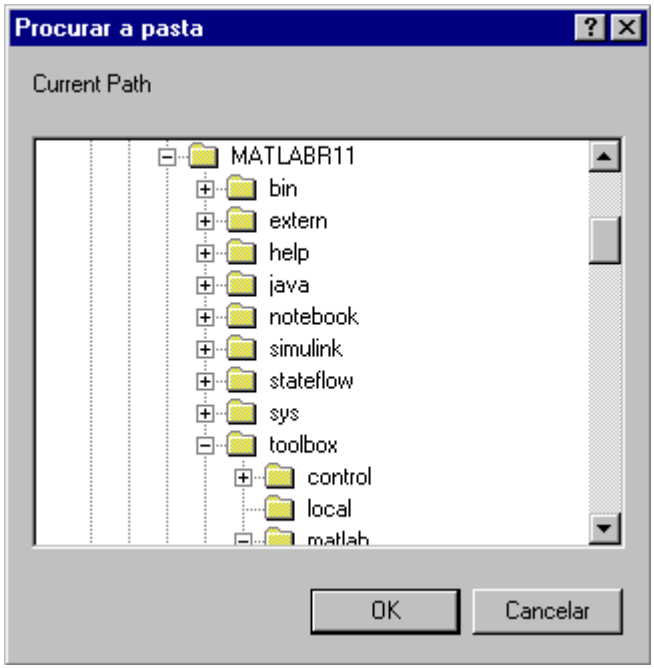
Outra forma de verificar quais são os diretórios do MATLAB é recorrendo ao ícone *Path Browser* que se encontra na Barra de Ferramentas.



Uma janela semelhante a seguinte é exibida:



Para que o diretório do usuário se torne o diretório de trabalho (diretório atual) deve-se clicar no botão *Browse*:



Em *Drives*, deve-se escolher o diretório raiz e em seguida, na caixa acima, o diretório desejado. Clicando em *OK*, o diretório escolhido passará a ser o diretório atual.

Clicando na linha do diretório atual, na parte central da janela do *path* aparecerá quais são os arquivos MATLAB que se encontram nesse diretório.

Se desejar adicionar um diretório ao *Path* basta clicar na opção *Add to Path...* do menu *Path,* escolher pela janela que aparecerá o diretório desejado, escolher adicionar na frente, *Add to Front* (primeiro diretório a ser pesquisado depois do diretório atual), ou no final do caminho *Add to Back* (último diretório a ser pesquisado

depois do diretório atual). O lugar onde um diretório fica no caminho de procura pode ser importante como veremos no decorrer do estudo.

16. Criando Arquivos M (Arquivos de Comandos)

Quando se tem uma sequência de comandos que se deseja repetir para valores diferentes das variáveis, utilizar o prompt pode ser pouco eficiente e cansativo.

O MATLAB permite que uma sequência de comandos seja gravada num arquivo, e que sejam executados como se eles tivessem sido introduzidos manualmente. Para issoo pelo prompt só o nome do arquivo.

Será criado agora um arquivo com base no **Problema 3**.

Para criar um arquivo M, primeiro certifique-se que o diretório atual é seu diretório de trabalho (comando *cd*), em caso de não ser utilize o **Path-Bowser** das ferramentas.

Selecione o item **New File** das ferramentas ou **New** do menu **File** . O que o MATLAB apresenta é uma janela em branco denominada MATLAB Editor/Debugger-[Untitled1]. Esta janela é um editor com características específicas para edição de programas MATLAB, como por exemplo identificar comandos, fazer indentação, identificar comentários, etc.

```
% Calculo do Decaimento Radiotivo
% Quantidade Inicial = quant_inic

quant_inic=10;
meia_vida=140;
tempo=250;
quant_rest=quant_inic * 0.5^(tempo/meia_vida);
disp(quant_rest)
```

Salve o arquivo com algum nome significativo, por exemplo **decaimento.m**.

Neste primeiro arquivo de comandos foi incluído o comando *disp(ans)* cuja função é a de apresentar o resultado sem identificar os nomes das variáveis. Desta forma, entrando com o nome do arquivo no prompt, visualizaremos apenas o resultado:

```
» decaimento
   2.9003
```

Fazendo:

```
» who

Your variables are:

meia_vida     quant_rest
quant_inic    tempo
```

Percebemos que apesar dos comandos estarem escritos num arquivo, uma vez executado, suas variáveis passam a pertencer ao Workspace.

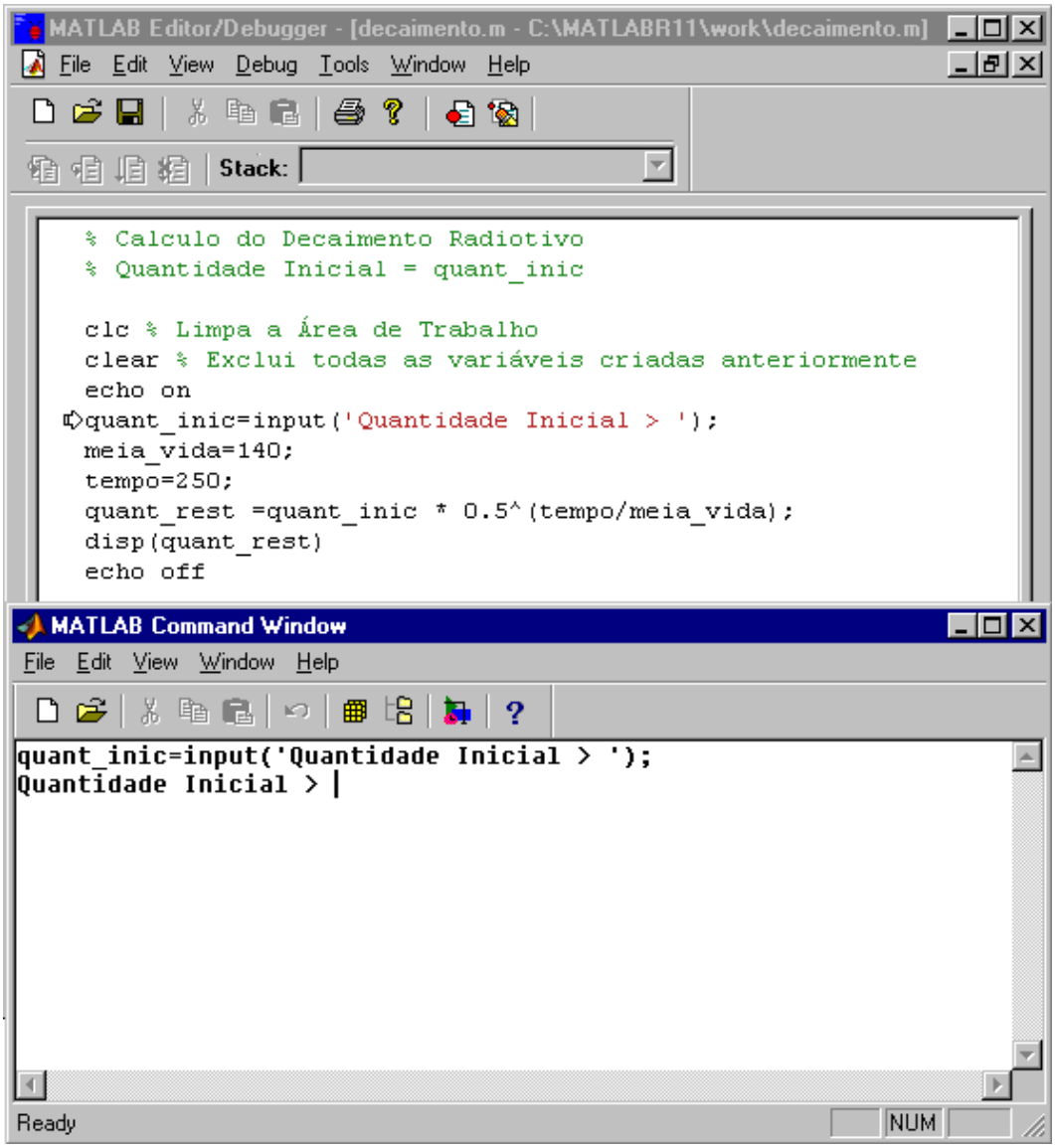
Suponha agora que a quantidade inicial deva ser informada pelo usuário. Neste caso, usaremos o comando *input*.

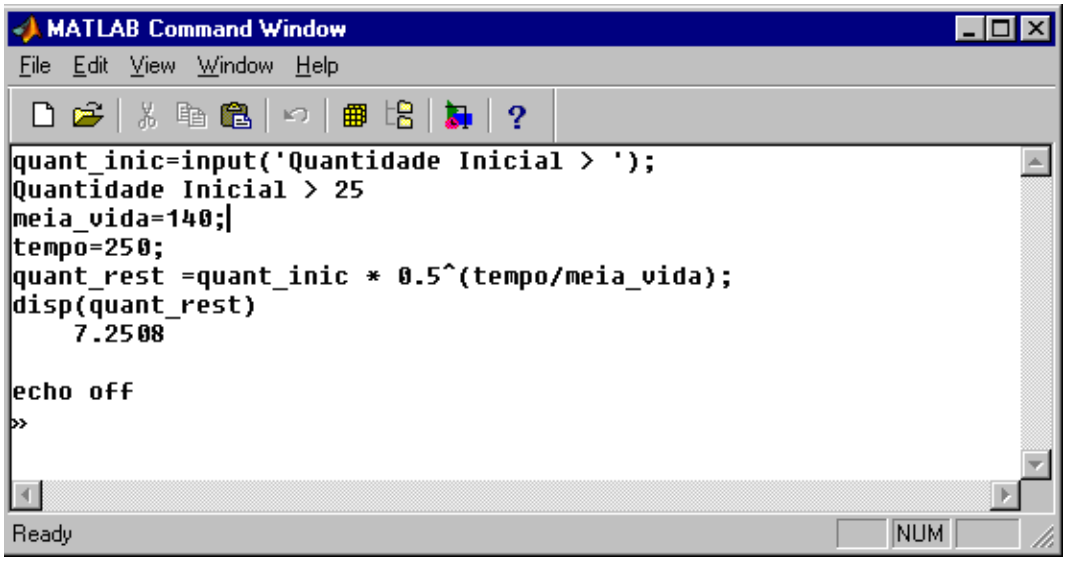
A função *input* pode ser misturada com outras operações, tal como as demais funções comuns, aceitando qualquer expressão válida co MATLAB.

Ema algumas situações, é desejável saber quais comandos foram utilizados para se obter uma certa resposta. Nestes casos, usamos o comando *echo* que tem a função de controlar a exibição dos comandos dos arquivos M na janela de comandos.

```
% Calculo do Decaimento Radiotivo
% Quantidade Inicial = quant_inic
```

```
clc % Limpa a Área de Trabalho
clear % Exclui todas as variáveis criadas anteriormente
echo on
quant_inic=input('Quantidade Inicial em g > ');
meia_vida=140;
tempo=250;
quant_rest =quant_inic * 0.5^(tempo/meia_vida);
disp(quant_rest)
echo off
```





Agora a variável **quant_inic** não está mais definida dentro do arquivo. Tem-se uma maior flexibilidade para a variação dos parâmetros sem que haja necessidade de repetição de comandos.

Outros comandos que são muito utilizados dentro de arquivos M para as funções de visualização e detecção de erros são os comandos *keyboard* e *pause*.

O comando *keyboard* tem a função de transferir temporariamente o controle para o teclado (pressione *return* para sair).

O comando *pause* suspende a execução até que o usuário pressione alguma tecla.

Por exemplo:

```
% Calculo do Decaimento Radiotivo
% Quantidade Inicial = quant_inic

quant_inic=input('Quantidade Inicial em g > ');
meia_vida=140;
tempo=250;
keyboard
quant_rest =quant_inic * 0.5^(tempo/meia_vida);
disp(quant_rest)
```

Executando obtemos:

```
» decaimento
Quantidade Inicial > 60
K» whos
  Name          Size         Bytes  Class

  meia_vida     1x1              8  double array
  quant_inic    1x1              8  double array
  quant_rest    1x1              8  double array
  tempo         1x1              8  double array

Grand total is 4 elements using 32 bytes

K» return
   17.4019
```

Nesta execução vemos o seguinte: Como o comando *keyboard* foi colocado antes dos cálculos, é possível modificar as variáveis já que elas ainda não foram utilizadas. Para sair do modo de controle do teclado é

necessário utilizar o comando *return*. Quando estamos no modo de controle de teclado, isto é indicado pelo prompt '**K»**'.

O comando *pause* é utilizado para esperar que o usuário aperte uma tecla qualquer para continuar com a execução dos comandos. Isto é utilizado quando o arquivo vai apresentando resultados intermediários e se deseja analisa-los. Também é possível utilizar o comando *pause(n)* onde *n* é o número de segundos que se deseja que a sequência seja interrompida.

Temos ainda o comando *waiforbuttonpress* que suspende a execução até que o usuário pressione uma tecla ou um botão do mouse.

17. Operações com Vetores e Matrizes

17.1. Criação de Vetores

A forma mais direita de criar um vetor em MATLAB é escreve-lo como é de costume em matemática, por exemplo:

```
» vet1 = [1, 2, 3, 2, 1]

vet1 =

    1    2    3    2    1
```

também pode-se escrever um vetor linha somente separando os elementos do vetor por espaços:

```
» vet1 = [ 1 2   3      2                1]

vet1 =

    1    2    3    2    1
```

Se o desejar é possível misturar as duas formas.

Uma das grandes vantagens do MATLAB é que ele consegue tratar variáveis matriciais e vetoriais (quase) do mesmo modo em que trata elementos individuais, por exemplo:

```
» vet2= sqrt(vet1)

vet2 =

   1.0000    1.4142    1.7321    1.4142    1.0000

» vet3=cos(vet2)

vet3 =

   0.5403    0.1559   -0.1606    0.1559    0.5403

» vet4=sqrt(vet3)

vet4 =

  Columns 1 through 4

  0.7351            0.3949                0 + 0.4007i   0.3949

  Column 5
```

0.7351

Observe no caso de **vet4** que a apresentação do resultado ficou um pouco confusa, devido a que o cálculo produziu um número complexo, e o MATLAB arranjou a apresentação para dar espaço a em caso de todos os elementos do vetor forem complexos. Como um elemento do vetor não tinha espaço na linha, foi colocado em baixo, e agregado à apresentação os índices das colunas (**Columns 1 through 4** e **Column 5**).

17.2. Endereçamento Vetorial

Os elementos dentro de um vetor podem ser endereçados pelo nome do vetor seguido do índice do elemento entre parêntese:

Nota: o primeiro índice em um vetor (e também nas matrizes) é "**1**".

```
» vet1(2)

ans =

   2

» vet4(3)

ans =

      0 + 0.4007i
```

Quando se deseja ter acesso a vários componentes pode-se utilizar vários índices na forma de um vetor:

```
» vet1([2 3 5])

ans =

   2    3    1

» vet1([2 5 3])

ans =

   2    1    3
```

Observe que a ordem dos índices pode ser arbitrária.

Quando os elementos que se deseja acessar estão em seqüência, pode-se utilizar ***:***

```
» vet1(2:5)

ans =

   2    3    2    1
```

ou

```
» vet1(1:2:5)

ans =

   1    3    1
```

Onde **1 : 2 : 5** indica os elementos que começam pelo índice 1 e vão de dois em dois até o índice 5.

De forma reversa:

» vet1(5:-1:2)

ans =

1 2 3 2

Isto porque o operador ***:*** por si só já serve para criar vetores.
Quando se deseja acessar o último elemento do vetor pode-se utilizar a palavra reservada *end*.

» vet1(3:end)

ans =

3 2 1

17.3. Construção de Vetores

Como foi visto, a forma mais direta de criar vetores é escrevendo eles normalmente em notação matemática. Entretanto muitas vezes são requerido vetores com características especiais, o que pode facilitar sua construção. Por exemplo desejo criar um vetor de elementos que será utilizado como base de tempo em segundos de zero a 10.

» tempo=0:10

tempo =

0 1 2 3 4 5 6 7 8 9 10

Pode ser que a base de tempo seja de 0 a 2 segundos em intervalos de 0.1 segundos.

» tempo=0:.1:2

tempo =

Columns 1 through 7

0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000

Columns 8 through 14

0.7000 0.8000 0.9000 1.0000 1.1000 1.2000 1.3000

Columns 15 through 21

1.4000 1.5000 1.6000 1.7000 1.8000 1.9000 2.0000

Que também pode ser criado como:

» tempo=(0:20)*.1

tempo =

Columns 1 through 7

0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000

Columns 8 through 14

0.7000 0.8000 0.9000 1.0000 1.1000 1.2000 1.3000

Columns 15 through 21

1.4000 1.5000 1.6000 1.7000 1.8000 1.9000 2.0000

Ou também com o auxilio da função *linspace*:

```
» linspace(0,2,21)

ans =

  Columns 1 through 7

         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000

  Columns 8 through 14

    0.7000    0.8000    0.9000    1.0000    1.1000    1.2000    1.3000

  Columns 15 through 21

    1.4000    1.5000    1.6000    1.7000    1.8000    1.9000    2.0000
```

Onde agora é indicado o primeiro elemento, o último e a quantidade de elemento que se deseja que o vetor possua, mas não posso indicar o incremento.

Também é possível criar vetores cujos elementos são as potências de 10, por exemplo:

```
» logspace(1,2,6)

ans =

   10.0000   15.8489   25.1189   39.8107   63.0957  100.0000
```

Que é equivalente a fazer:

```
» 10.^(1:.2:2)

ans =

   10.0000   15.8489   25.1189   39.8107   63.0957  100.0000
```

O operador *".^"* é um operador que indica que os expoentes devem ser aplicados um a um ao elemento 10. Isto é um assunto que será tratado com mais profundidade em outra seção.

Assim como é possível criar vetores individualmente de cada uma das formas, pode-se misturar-las para criar vetores das mais diversas formas, por exemplo:

```
» vetorestranho=[3:-1.2:-1,0 2.5,logspace(0.01,0.1,3)]

vetorestranho =

  Columns 1 through 7

    3.0000    1.8000    0.6000   -0.6000         0    2.5000    1.0233
```

Columns 8 through 9

1.1350 1.2589

Também posso utilizar vetores já criados ou parte deles para criar novos vetores:

» vetornovo=[vet1(1:3),vet4(5:-1:3)]

vetornovo =

Columns 1 through 4

1.0000 2.0000 3.0000 0.7351

Columns 5 through 6

0.3949 0 + 0.4007i

17.4. Vetores Coluna

Até agora só foram apresentado vetores linha, mas na prática eles são utilizados tanto nessa forma como na forma de coluna. Para criar um vetor coluna primeiramente posso utilizar como separador o ponto e vírgula *;* no lugar da vírgula. Por exemplo:

» vetc=[1;2;3;4;0]

vetc =

1
2
3
4
0

Todos os métodos apresentados para criar vetores linha não têm sua contrapartida para criar vetores coluna. Entretanto, como na notação matemática, na qual muitas vezes é utilizado a aspas simples **'**, o MATLAB interpreta esse símbolo como a transposição de elementos.

» vetc1=vet1'

vetc1 =

1
2
3
2
1

Assim é possível criar vetores coluna com qualquer método simplesmente criando linhas e logo transpondo:

» vetc3=logspace(0.02,0.2,5)'

vetc3 =

1.0471
1.1614

```
1.2882
1.4289
1.5849
```

Como acontece com a transposição matemática, se o vetor for formado por números complexos, sua transposição cria um vetor transposto e conjugado:

```
» ac=sqrt(-4:-1)

ac =

   0 + 2.0000i   0 + 1.7321i   0 + 1.4142i   0 + 1.0000i

» act=ac'

act =

   0 - 2.0000i
   0 - 1.7321i
   0 - 1.4142i
   0 - 1.0000i
```

Quando o que se deseja é a transposição dos elementos sem conjugar utiliza-se o operador ponto-aspas .'.

```
» act1=ac.'

act1 =

   0 + 2.0000i
   0 + 1.7321i
   0 + 1.4142i
   0 + 1.0000i
```

A operação transposição, como é de esperar transforma também um vetor coluna em um vetor linha:

```
» vecx=vetc3'

vecx =

  1.0471   1.1614   1.2882   1.4289   1.5849
```

Este operador como será visto mais na frente também opera com matrizes.

17.5. Criação de Matrizes

Como foi visto na seção anterior, agrupando elementos separados por vírgulas, constroem-se vetores linha, e separando-os por ponto e virgula permite construir vetores coluna. Combinando estes dois tipos de separadores permite construir matrizes.

```
» mat_a = [1, 2, 3 4; 4 5, 6 7;0 1 1 0]

mat_a =

   1   2   3   4
   4   5   6   7
```

```
    0    1    1    0
```

Neste primeiro exemplo pode-se observar que para indicar elementos separados nas linhas, como é feito com os vetores, pode-se faze-lo com vírgulas ou com espaços, entretanto, o indicador de "coluna seguinte" deve sempre estar presente ou ser substituído pelo **return** ou **enter**:

```
» mat_1=[0.1 0.4 0.3 1
0.9 0.8 0.1 0
1.1 0.5 0.4 0.2]

mat_1 =

    0.1000    0.4000    0.3000    1.0000
    0.9000    0.8000    0.1000         0
    1.1000    0.5000    0.4000    0.2000
```

Utilizando todas as formas já vistas para criar vetores, permite criar matrizes com as mesmas ferramentas. Por exemplo:

```
» vv=1:4;
» v1=8:-1:5;
» mat_b=[vv',v1']

mat_b =

    1    8
    2    7
    3    6
    4    5
```

O detalhe mais importante a se considerar é que todas as linhas (todas as colunas) de uma matriz devem ter a mesma quantidade de elementos. Este fato apesar de ser evidente nem sempre fica claro quando a matriz é criada por métodos diferentes dos de escrever direitamente um a um.

```
» mat_c=[1 2 3; 4 5]
Number of elements in each row must be the same.
» mat_c1=[vv',v1]
???  All matrices on a row in the bracketed expression must have the
 same number of rows.
```

O operador transposição como já foi antecipado, também funciona com matrizes.

```
» mat_d=[1:3;4:6]

mat_d =

    1    2    3
    4    5    6

» mat_dt=mat_d'

mat_dt =

    1    4
    2    5
    3    6
```

Pode-se observar que como os vetores e as matrizes compartilham os mesmos métodos de construção (e como veremos a mesma matemática), pode-se considerar os vetores como matrizes unidimensionais (compostas de uma linha ou de uma coluna) o que simplifica seu tratamento.

17.6. Matemática Escalar - Matriz

Como já foi observado em algumas situações, a operação escalar - vetor ou escalar - matriz é muito simples é segue a matemática *normal*.

```
» mat_d - 4

ans =

   -3   -2   -1
    0    1    2

» mat_a*-1

ans =

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

Entretanto em algumas situações o resultado desejado não é o que a matemática e o MATLAB darão como resposta. Para essas situações existem alguns operadores especiais. Por exemplo quando desejamos elevar todos os termos de uma matriz a um expoente devemos utilizar o operador **.^** já que o operador ^ por si só aplicado a uma matriz é o exponencial da matriz.

```
» mat_d.^2

ans =

    1    4    9
   16   25   36
```

No exemplo anterior se fosse utilizado **mat_d^2** o MATLAB daria uma mensagem de erro já que isso equivale a fazer **mat_d * mat_d** que pode apresentar um resultado válido somente se **mat_d** fosse quadrada (mesmo número de linhas que de colunas) como será visto mais na frente.

Este operador também funciona se tento elevar um número a diferentes expoentes que estão numa matriz.

```
» ex=[1 2 3;4 5 6;7 8 9]

ex =

    1    2    3
    4    5    6
    7    8    9

» 2.^ex

ans =

    2    4    8
   16   32   64
  128  256  512
```

17.7. Matemática Matriz - Matriz

Como acontece na matemática a soma e a subtração de duas matrizes só é possível se forem do mesmo tamanho, por exemplo:

```
» clear
» a=[1 2 3;4 5 6]

a =

   1   2   3
   4   5   6

» b=[9 8 7; 6 5 4]

b =

   9   8   7
   6   5   4

» c=b'

c =

   9   6
   8   5
   7   4

» b - a

ans =

   8   6   4
   2   0  -2

» a + c
??? Error using ==> +
Matrix dimensions must agree.
```

A multiplicação e a divisão elemento a elemento de duas matrizes também requer que a dimensão das duas matrizes seja a mesma. Entretanto a notação é um pouco diferente , no caso **.*** e **./** estando reservado o ***** e a **/** para a multiplicação e divisão matricial clássica.

```
» a.*b

ans =

   9  16  21
  24  25  24

» b./a

ans =

   9.0000   4.0000   2.3333
   1.5000   1.0000   0.6667
```

```
» d=[1 2 3;3 2 1]

d =

   1   2   3
   3   2   1

» e=[2 2 2;3 3 3]

e =

   2   2   2
   3   3   3

» d.^e

ans =

   1   4   9
  27   8   1
```

17.8. Multiplicação Matricial

A multiplicação matricial requer que a quantidade de colunas da primeira matriz seja igual ao número de linhas da segunda matriz:

```
» a,c % mostra as matrizes

a =

   1   2   3
   4   5   6


c =

   9   6
   8   5
   7   4

» c*a

ans =

   33   48   63
   28   41   54
   23   34   45
```

Na multiplicação matricial é muito importante a ordem dos termos:

```
» a*c

ans =

   46   28
```

```
   118    73
```

Onde observa-se que **c * a <> a * c**.

Deve-se prestar muita atenção a este detalhe já que é umas das principais fontes de erros na matemática matricial.

A divisão de matrizes é ainda mais complicado já que somente pode-se utilizar como divisor uma matriz quadrada já que a operação matriz exponencial ^ só é definida para matrizes quadradas. Dividir uma quantidade qualquer é igual a multiplica-la pela inversa do seu divisor, por exemplo:

```
» f=[1 2;3 4]

f =

    1    2
    3    4

» f^-1

ans =

   -2.0000    1.0000
    1.5000   -0.5000

» c*f^-1

ans =

   -9.0000    6.0000
   -8.5000    5.5000
   -8.0000    5.0000

» c/f

ans =

   -9.0000    6.0000
   -8.5000    5.5000
   -8.0000    5.0000
```

18. Tamanhos e Endereçamento

Para verificar quais são as variáveis que tenho no meu workspace, já foram vistos os comandos **who** e **whos**, utilizando este último comando posso ver o tamanho das variáveis definidas:

```
» whos
  Name      Size         Bytes  Class

  a         2x3             48  double array
  ans       2x3             48  double array
  b         2x3             48  double array
  c         3x2             48  double array
  d         2x3             48  double array
  e         2x3             48  double array
  ex        3x3             72  double array
  f         2x2             32  double array

Grand total is 49 elements using 392 bytes
```

onde pode-se observar que o tamanho (**size**) está indicado na forma clássica **nxm** onde **n** indica a quantidade de linhas da matriz e **m** a quantidade de colunas. Se se deseja saber qual é o tamanho individual de cada matriz pode-se utilizar a função **size(X)**

```
» tam_a=size(a)

tam_a =

   2   3
```

Esta função dá como resposta um vetor no qual o primeiro elemento indica a quantidade de linhas e o segundo a quantidade de colunas.

Também é possível saber só a quantidade de linhas da matriz:

```
» size(a,1)

ans =

   2
```

Ou a quantidade de colunas:

```
» size(a,2)

ans =

   3
```

A função **length(X)** retorna o número de linhas ou colunas (o que for maior) da matriz **X**.

```
» length(a)

ans =

   3

» length(c)

ans =

   3
```

A forma de indicar primeiro as linhas e depois as colunas é a forma geralmente utilizada pelo MATLAB para todas as operações, inclusive para se referir a um elemento da matriz, é utilizada essa forma.

```
» b(2,3)

ans =

   4
```

Está-se observando o elemento que fica na segunda linha e na terceira coluna da matriz **b**.

```
» c(1:2,2)

ans =

   6
```

```
     5
```

Neste último estão-se observando os elementos da primeira e segunda linha que se encontram na segunda coluna.

Quando se deseja obter os elementos de toda uma linha ou de toda uma coluna podem-se utilizar varios métodos. No primeiro, se é conhecido o tamanho da matriz, pode-se fazer:

```
» ex      %mostra a matriz ex

ex =

   1   2   3
   4   5   6
   7   8   9

» ex(2,1:3)

ans =

   4   5   6
```

Como com vetores, pode-se utilizar a palavra reservada **end** para indicar o último elemento.

```
» ex(2,1:end)

ans =

   4   5   6
```

O MATLAB permite ainda uma simplificação maior, utilizar simplesmente os dois pontos **:** para indicar toda a linha **ou** toda a coluna (isto também se aplica a vetores, entretanto não é utilizável).

```
» ex(2,:)

ans =

   4   5   6
```

Se o índice do elemento desejado for maior que o tamanho da matriz, o MATLAB acusará um erro.

```
» b(2,1:5)
??? Index exceeds matrix dimensions.
```

Problema 5) Decaimento Radioativo

O polônio, um elemento radioativo, tem uma meia-vida de 140 dias, o que significa que, devido ao decaimento radioativo, a quantidade de polônio restante depois de 140 dias é a metade da original. Começando com 10 gramas de polônio hoje, quanto vai restar depois de 250 dias?

Solução:

Usando a solução apresentada anteriormente, a quantidade restante depois de um certo período de tempo é dada por:

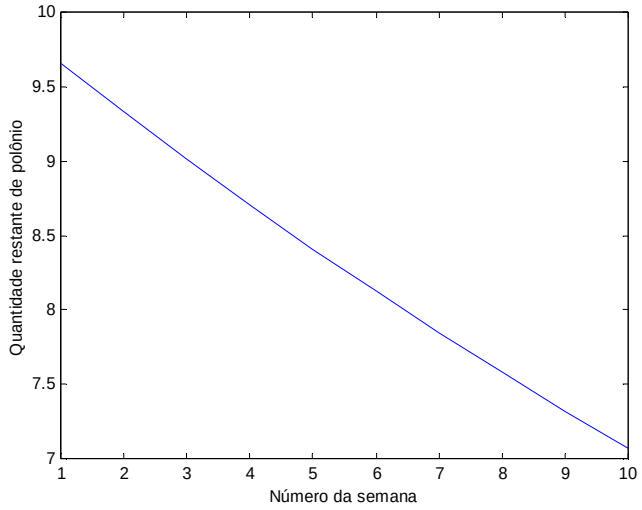
$$\text{quantidade restante} = \text{quantidade inicial} \cdot (0{,}5)^{\text{tempo/meia-vida}}$$

Solução em MATLAB:

```
» quant_inicial=10;
» meia_vida=140;
» tempo=7:7:70 %o último dia das próximas 10 semanas

tempo =

    7   14   21   28   35   42   49   56   63   70

» quant_rest=quant_inicial*0.5.^(tempo/meia_vida)

quant_rest =

  Columns 1 through 7

    9.6594    9.3303    9.0125    8.7055    8.4090    8.1225    7.8458

  Columns 8 through 10

    7.5786    7.3204    7.0711
```

O uso de matemática vetorial faz com que se torne fácil calcular a expressão para múltiplos valores de uma variável. Note que a exponenciação elemento a elemento **.^** é usada porque queremos calcular 0,5 elevado a cada componente do vetor. Um gráfico com esses dados pode ser traçado facilmente no MATLAB:

```
» plot(tempo/7,quant_rest)
» xlabel('Número da semana')
» ylabel('Quantidade restante de polônio')
```



Problema 6) O "Problema da Semana", estudado nas escolas de ensino médio, consiste em determinar todos os números menores que 1000 que são divisíveis por 7, mas têm resto 1 quando são divididos por 2, 3, 4, 5 e 6.

Solução:

Não há solução analítica para este problema, de modo que ela tem de ser encontrada por busca. Se você começar com todos os múltiplos de 7 menores que 1000. todos os números que não satisfazem as condições restantes podem ser desprezados, restando a solução desejada.

Solução em MATLAB:

```
%semana.m arquivo M de comandos para resolver o problema da
%semana

n=7:7:1000; %todos os múltiplos de 7 menores que 1000
numero=length(n) %número de possíveis soluções
n(rem(n,2)~=1)=[]; %os elemntos que não são solução são
                   %rejeitados
numero=length(n)
n(rem(n,3)~=1)=[]; %atribuindo a eles uma variável vazia
numero=length(n)
n(rem(n,4)~=1)=[]; %a função rem calcula o resto de uma
                   %divisão
numero=length(n)
n(rem(n,5)~=1)=[];
numero=length(n)
n(rem(n,6)~=1)=[]
```

A execução desse arquivo de comandos produz duas soluções:

```
» semana

numero =

  142


numero =

   71


numero =

   24


numero =

   12


numero =

    2


n =

  301   721
```

Claramente, não foi difícil eliminar a maioria das candidatas à solução.

Problema 7) No processo de manufatura nas seções de fundição de uma fábrica automotiva, peças são mergulhadas na água para resfriar e então mergulhadas em um banho ácido para clarear. Com o tempo, a

concentração do ácido decresce em conseqüência da água introduzida na imersão e da solução removida quando as peças são retiradas do banho ácido. Para manter a qualidade, a acidez da solução não deve ser menor que um determinado limite mínimo. Começando com uma concentração de 90% de ácido na solução, admitindo-se que a concentração mínima seja de 50%, e admitindo-se ainda que a água introduzida e removida em cada mergulho varia de 1% a 10% e que uma certa quantidade de ácido é também removida com a peça, de maneira que o volume da solução ácida permaneça constante, pergunta-se: quantas peças podem ser mergulhadas no banho ácido antes de cair abaixo da acidez mínima?

Solução:

Usando a solução apresentada anteriormente:

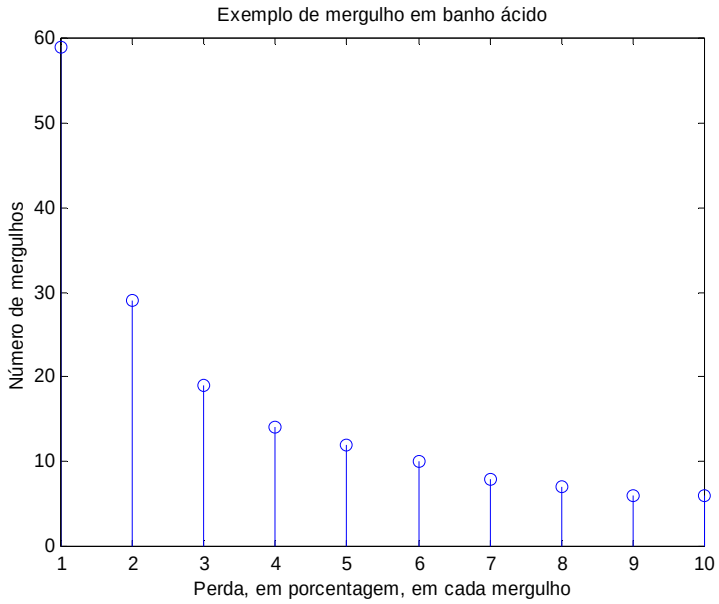
$$n = \frac{\log(con\_inicial / con\_\min)}{\log(1 + perda)}$$

Solução no MATLAB:

```
%arquivo m de comandos problema7

con_inicial=90;
con_min=50;
perda=1:10 %considera de 1% a 10%, em incrementos de 1%
n=floor(log(con_inicial/con_min)./log(1+perda/100))
stem(perda,n)
xlabel('Perda, em porcentagem, em cada mergulho')
ylabel('Número de mergulhos')
title('Exemplo de mergulho em banho ácido')
```

A execução desses comandos produz os seguintes dados e traça o gráfico de haster que aparece a seguir:

```
» problema7

perda =

   1   2   3   4   5   6   7   8   9  10


n =

  59  29  19  14  12  10   8   7   6   6
```



19. Funções de Construção de Matrizes

eye matriz identidade

```
» eye(2)

ans =

   1   0
   0   1
```

zeros matriz de zeros

```
» zeros(2,3)
ans =

   0   0   0
   0   0   0
```

ones matriz de 1's

```
» ones(2,1)

ans =

   1
   1
```

diag matriz diagonal

- Se **x** é um vetor, **diag(x)** é a matriz diagonal com **x** na diagonal;

```
»  x=[1 2 3 1 -1 4];
» diag(x)

ans =
```

```
1   0   0   0   0   0
0   2   0   0   0   0
0   0   3   0   0   0
0   0   0   1   0   0
0   0   0   0  -1   0
0   0   0   0   0   4
```

- Se **A** é uma matriz quadrada, então **diag(A)** é um vetor cujos componentes são os elementos da diagonal de **A**.

```
» A=[3 11 5; 4 1 -3; 6 2 1]

A =

   3   11    5
   4    1   -3
   6    2    1

» diag(A)

ans =

   3
   1
   1

» diag(diag(A))

ans =

   3   0   0
   0   1   0
   0   0   1
```

É possível construir uma matriz a partir de blocos. Exemplo, se **A** é uma matriz **3 x 3**, então:

```
» B = [A, zeros(3,2); zeros(2,3), eye(2)]

B =

   3   11    5    0    0
   4    1   -3    0    0
   6    2    1    0    0
   0    0    0    1    0
   0    0    0    0    1
```

B é matriz **5 x 5**.

rand matriz gerada aleatoriamente

```
» rand(5,4)

ans =

   0.9501   0.7621   0.6154   0.4057
   0.2311   0.4565   0.7919   0.9355
```

```
    0.6068    0.0185    0.9218    0.9169
    0.4860    0.8214    0.7382    0.4103
    0.8913    0.4447    0.1763    0.8936
```

magic quadrado mágico

```
» magic(3)

ans =

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

20. Polinômios

No MATLAB, os polinômios são representados por vetores linha contendo seus coeficientes em ordem decrescente. Por exemplo, o polinômio $x^4 – 12x^3 + 0x^2 + 25x + 116$ é introduzido como:

```
» p=[1 -12 0 25 116]

p =

     1   -12     0    25   116
```

As raízes do polinômio são encontradas usando-se a função *roots*:

```
» r=roots(p)

r =

  11.7473
   2.7028
  -1.2251 + 1.4672i
  -1.2251 - 1.4672i
```

Por convenção, os polinômios são vetores linha e as raízes são vetores coluna. A partir das raízes, também é possível construir o polinômio associado:

```
» pa=poly(r)

pa =

    1.0000  -12.0000   -0.0000   25.0000  116.0000
```

20.1. Multiplicação

A multiplicação polinomial é efetuada por meio da função *conv* (convolução entre dois vetores). Considerando os polinômios $a(x) = x^3 + 2x^2 + 3x + 4$ e $b(x) = x^3 + 4x^2 + 9x + 16$, seu produto é dado por:

```
» a=[1 2 3 4]; b=[1 4 9 16];
» c=conv(a,b)

c =

     1     6    20    50    75    84    64
```

Logo, $c(x) = x^6 + 6x^5 + 20x^4 + 50x^3 + 75x^2 + 84x + 64$.

20.2. Adição

Quando os dois polinômios forem de ordens diferentes, aquele que tiver menor ordem terá de ser preenchido com coeficientes iguais a zero, a fim de torná-lo da mesma ordem do polinômio de ordem mais alta.

```
» d=a+b

d =

   2    6   12   20

» e=c+[0 0 0 d]

e =

   1    6   20   52   81   96   84
```

20.3. Divisão

A divisão de um polinômio por outro pode ser feita com a função *deconv*.

```
» [q,r]=deconv(c,b)

q =

   1    2    3    4


r =

   0    0    0    0    0    0    0
```

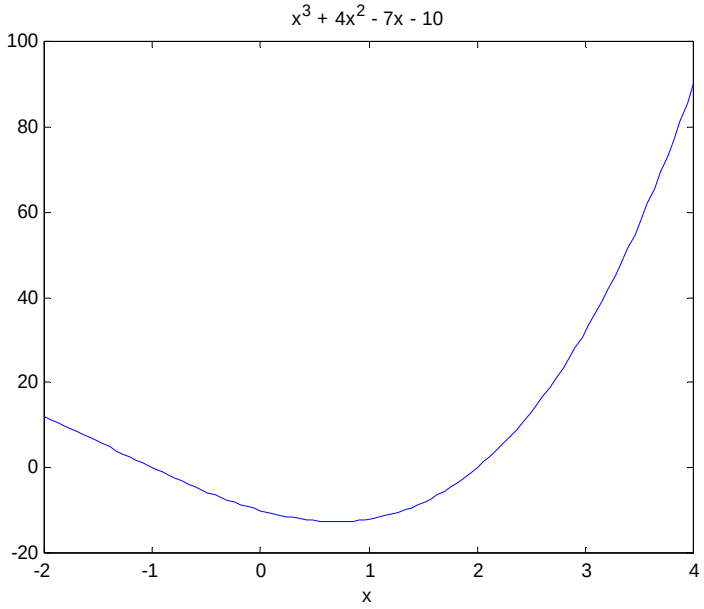
O resultado nos diz que *c* dividido por *b* nos dá o polinômio quociente *q* e o resto *r*.

20.4. Derivadas

```
» f=polyder(e)

f =

   6   30   80  156  162   96
```

20.5. Cálculo de Polinômios

» x=linspace(-2,4);
Escolhe 100 pontos entre -2 e 4.
» p=[1 4 -7 -10];
Define o polinômio $p(x)=x^3 + 4x^2 - 7x - 10$.
» v=polyval(p,x);
Calcula p(x) nos valores armazenados em x e armazena o resultado em v.
» plot(x,v), title('x^3 + 4x^2 - 7x - 10'), xlabel('x')
Plota o gráfico resultante.



20.6. Expansão em Frações Parciais

Dada a seguinte razão entre polinômios:

$$\frac{n(x)}{d(x)} = \frac{N_1 x^m + N_2 x^{m-1} + \cdots + N_{m+1}}{D_1 x^n + D_2 x^{n-1} + \cdots + D_{n+1}}$$

Temos:

```
» n=[1 -10 100]     %   numerador

n =

    1   -10   100

» d=[1 10 100 0]     %   denominador

d =

    1    10   100     0

» z= roots(n)     %   raízes de n(x)/d(x)

z =

  5.0000 + 8.6603i
  5.0000 - 8.6603i

» p=roots(d)     %   pólos de n(x)/p(x)

p =

      0
 -5.0000 + 8.6603i
 -5.0000 - 8.6603i
```

```
» [nd,dd]=polyder(n,d)     %   derivada em relação a x

nd =

      -1       20      -100      -2000     -10000


dd =

  Columns 1 through 6

       1       20       300       2000      10000          0

  Column 7

       0

» [r,p,k]=residue(n,d)

r =

  0.0000 + 1.1547i
  0.0000 - 1.1547i
  1.0000


p =

 -5.0000 + 8.6603i
 -5.0000 - 8.6603i
      0


k =

    []
```

Neste caso, a função *residue* retorna os coeficientes r dos resíduos (ou da expansão em frações parciais), seus pólos associados p e o termo polinomial k. Como a ordem do numerador é menor que a do denominador, não há termos direitos. Sendo assim, a expansão em frações parciais do polinômio racional dado anteriormente é:

$$\frac{n(x)}{d(x)} = \frac{1{,}1547i}{x+5-8{,}6603i} + \frac{-1{,}1547i}{x-5+8{,}6603i} + \frac{1}{x}$$

O polinômio racional pode ser obtido usando-se novamente a função *residue*.

```
» [nn,dd]=residue(r,p,k)

nn =

   1.0000  -10.0000  100.0000


dd =

   1.0000   10.0000  100.0000         0
```

A função *residue* realiza duas operações, sendo uma a inversa da outra, com base no número de argumentos de entrada e saída que são utilizados.

21. Sistemas de Equações Lineares

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 366 \\ 804 \\ 351 \end{bmatrix}$$

$$A \cdot x = b$$

$$x = A^{-1} \cdot b$$

Para resolver esse problema, é necessário fornecer **A** e **b**:

```
» A=[1 2 3; 4 5 6
7 8 0]

A =

   1   2   3
   4   5   6
   7   8   0

» b=[366;804;351]

b =

  366
  804
  351
```

Para que esse problema tenha solução única é necessário que o *determinante* da matriz A seja diferente de 0.

```
» det(A)

ans =

   27
```

Sendo isso verdade, há duas formas de usar o MATLAB para determinar a solução de $A \cdot x = b$ sendo uma delas mais recomendável. A maneira menos indicada, mas mais direta, consiste em calcular $x = A^{-1} \cdot b$ explicitamente:

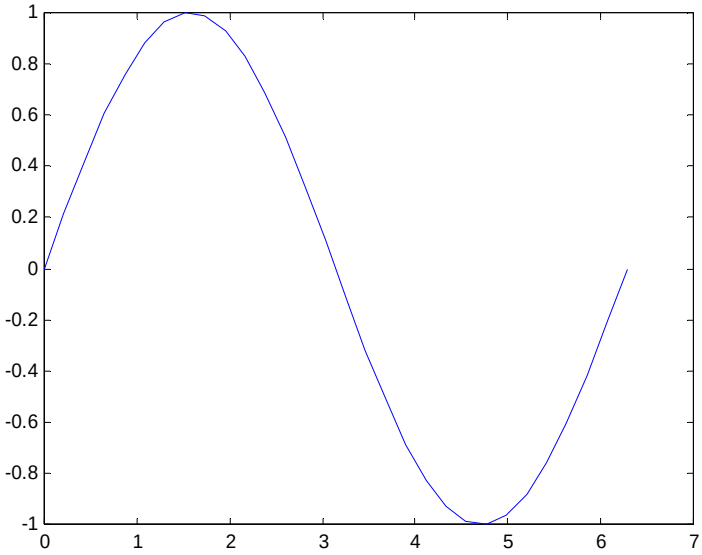
```
» x=inv(A)*b

x =

  25.0000
  22.0000
  99.0000
```

A maneira mais recomendável de resolver o sistema consiste em usar o operador matricial de divisão à esquerda.

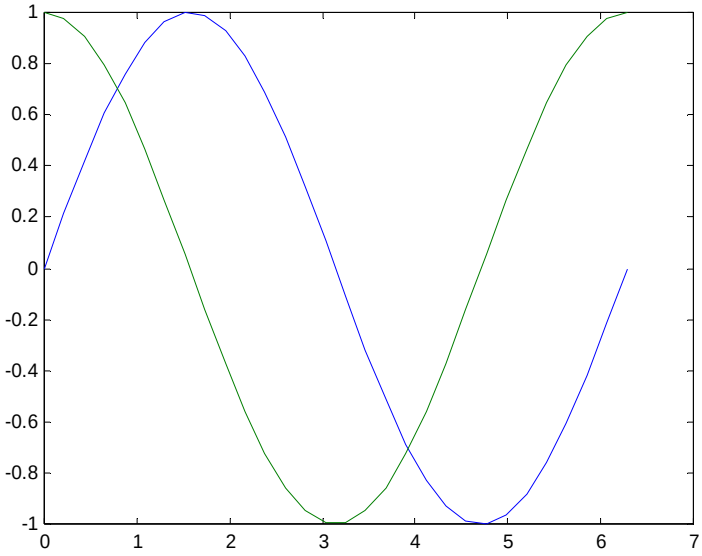
```
» x=A\b

x =

   25.0000
   22.0000
   99.0000
```

22. Gráficos Bidimensionais

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




```
» z=cos(x);
» plot(x,y,x,z)
```

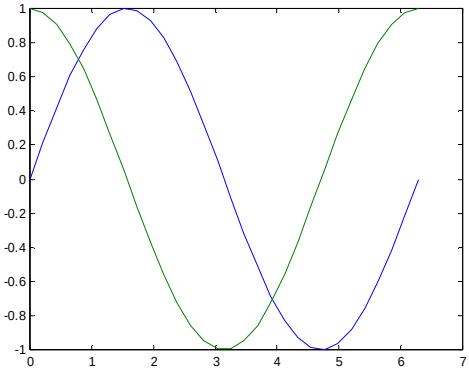


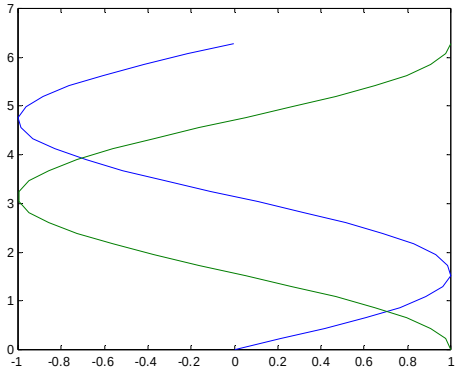
Neste exemplo, seno(x) *versus* x e cosseno(x) *versus* x foram representados no mesmo gráfico. O comando *plot* automaticamente desenhou a segunda curva em cor diferente na tela. Muitas curvas podem ser representadas de uma só vez por meio da introdução de pares adicionais de argumentos.

» W=[y;z]; % cria uma matriz do seno e do cosseno
» plot(x,W) % faz o gráfico das colunas de W versus x

Mudando a ordem os argumentos, o gráfico gira de 90 graus:

» plot(W,x) % faz o gráfico de x versus as colunas de W





Quando o comando *plot* é executado com apenas um argumento, como em plot(Y), ele age de maneira diferente, dependendo do tipo dos dados contidos em Y.

Se Y armazenar valores complexos, plot(Y) é interpretado como plot(real(Y),imag(Y)). Em todos os outros casos, a parte imaginária das componentes do vetor de entrada é ignorada.

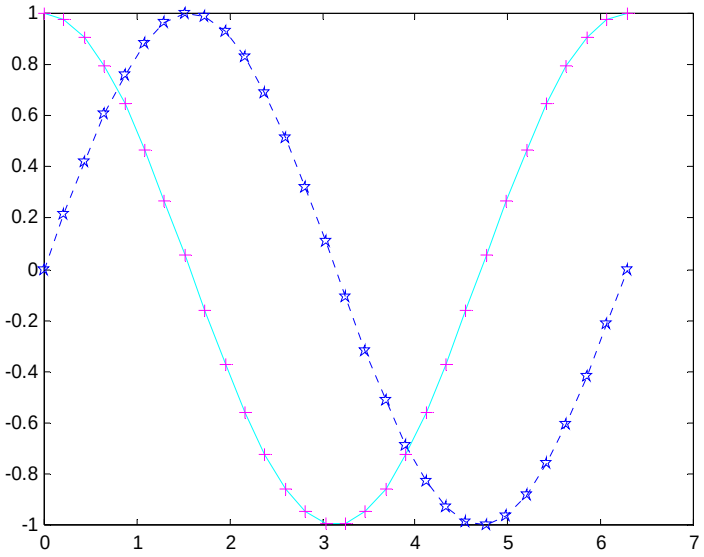
Por outro lado, se Y armazenar valores reais, plot(Y) é interpretado como plot(1:length(Y),Y), ou seja, é feito um gráfico de Y versus o índice de seus elementos. Quando Y é uma matriz, essa interpretação vale para cada uma de suas colunas.

22.1. Estilos de Linha, Marcadores e Cores

Símbolo	*Cor*	*Símbolo*	*Marcador*	*Símbolo*	*Tipo de Linha*
b	Azul	.	Ponto	-	Linha contínua
g	Verde	o	Círculo	:	Linha pontilhada
r	Vermelho	x	x	-.	Traços e pontos
c	Ciano	+	+	--	Linha tracejada
m	Magenta	*	Estrela		
y	Amarelo	s	Quadrado		
k	Preto	d	Losango		
w	Branco	v	Triângulo para baixo		
		^	Triângulo para cima		
		<	Triângulo para esquerda		
		>	Triângulo para direita		
		p	Pentagrama		
		h	Hexagrama		

O MATLAB começa com o azul e segue ciclicamente pelas sete primeiras cores da tabela para cada nova curva. O tipo de linha habitualmente usado é o contínuo. Não há marcador padrão.

Se uma cor, um marcador e um estilo de linha são especificados, a cor se aplicará tanto aos marcadores como à linha. Para conseguir um gráfico com uma cor diferente para os marcadores, faça dois gráficos para os mesmos dados alterando a especificação das curvas.

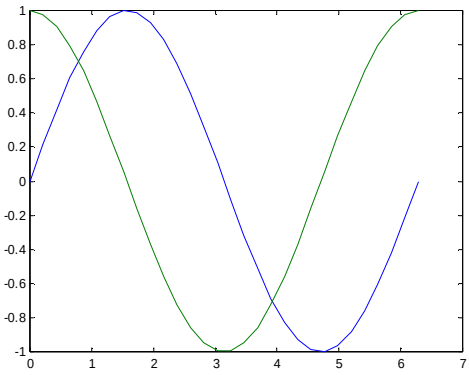


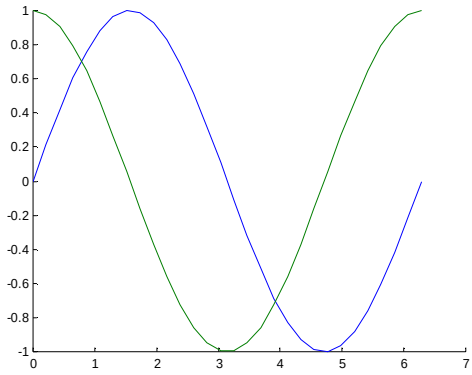
22.2. Estilos de Gráficos

O comando *colordef* permite que você selecione um estilo genérico para seus gráficos. O estilo-padrão é *colordef white*. Esse estilo usa um fundo branco para os eixos, um fundo cinza-claro para figuras, preto para legendas e azul, verde-escuro e vermelho para as três primeiras curvas traçadas. Se você prefere um fundo preto, use *colordef black*. Esse estilo usa um fundo preto para os eixos, um fundo cinza-escuro para figuras, branco para legendas e amarelo, magenta e ciano para os três primeiros gráficos traçados. Se você optar por *colordef none*, o MATLAB usará o padrão da versão 4. Esse estilo usa o preto como fundo para os eixos e figuras, o branco para legendas e amarelo, magenta e ciano para os três primeiros gráficos traçados.

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

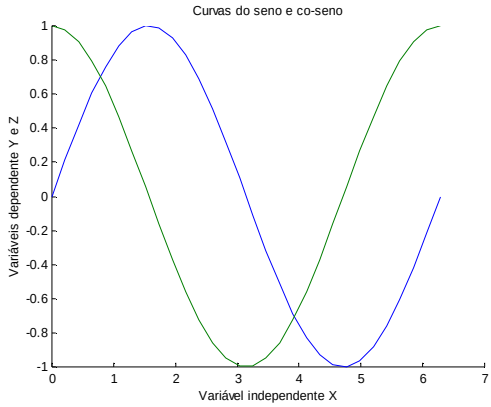
» plot(x,y,x,z)

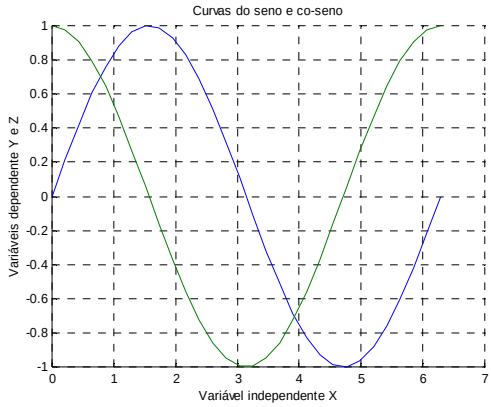




22.3. Grades, Eixos, Legendas e Títulos

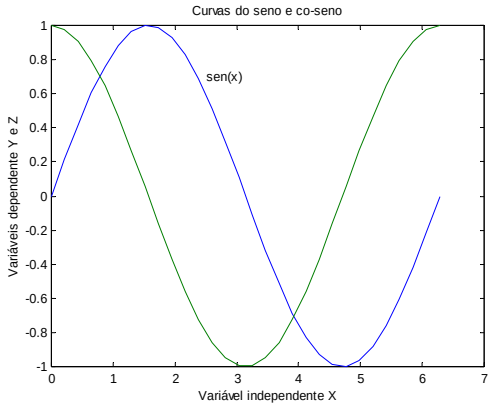
```
» xlabel('Variável independente X')     %   Legenda do eixo horizontal
» ylabel('Variáveis dependente Y e Z')     %   Legenda do eixo vertical
» title('Curvas do seno e co-seno')     %   Título do gráfico
» grid on      % Cria grade
```

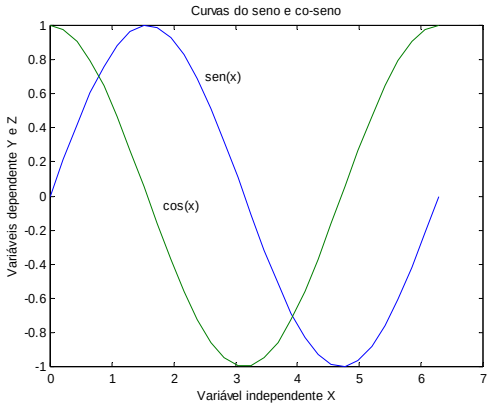




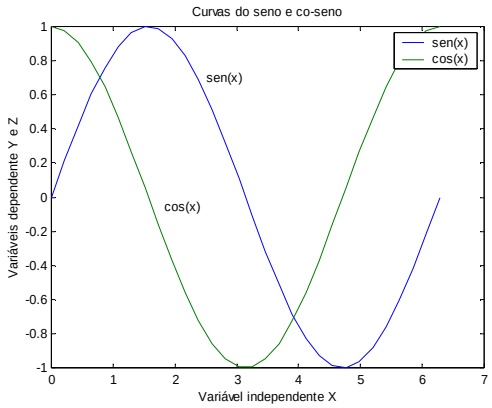

```
» grid off, box on      %   Restaura a caixa dos eixos e oculta a grade
» text(2.5,0.7,'sen(x)')      %   Adiciona ao gráfico um texto identificando a curva do seno na
posição (2.5,0.7)
» gtext('cos(x)')      %   Adiciona o texto desejado com o mouse.
```

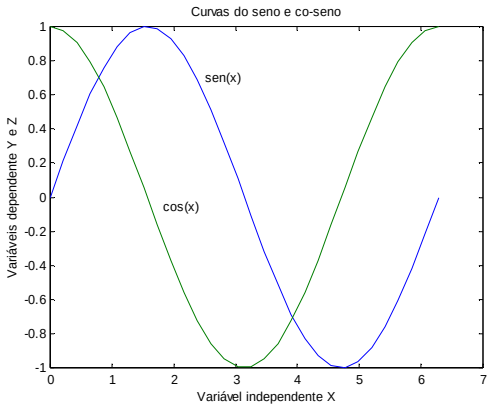
O comando *gtext* muda para a janela de figuras em uso e desenha nela uma cruz que segue o mouse até que você dê um clique com o mouse ou pressione uma tecla. O texto será colocado com o canto inferior esquerdo do primeiro caractere naquela posição.





» legend('sen(x)','cos(x)') % Exibe a legenda
» legend off % Apaga a legenda





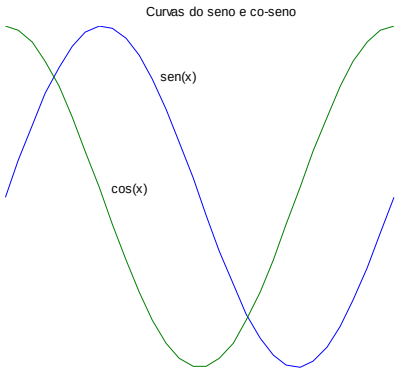
É possível mover e excluir a legenda e os textos através do gráfico.

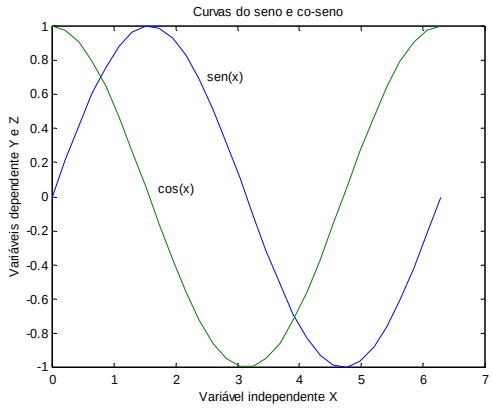
22.4. Personalização de Eixos

Comandos	***Descrição***
axis on	Devolve a cor de fundo e os nomes dos eixos, os marcadores e, caso tenham sido definidas, a grade e a caixa em torno do gráfico.
axis off	Retira a cor de fundo e os nomes dos eixos, a grade, a caixa e os marcadores. Mantém o título e os textos definidos pelos comandos *text* e *gtext*.
axis([xmin xmax ymin ymax])	Define os valores mínimos e máximos dos eixos com base nos valores fornecidos pelo vetor linha.
V=axis	Guarda em V o vetor linha que contém os dados de escala do gráfico atual: [xmin xmax ymin ymax]
axis auto axis('auto')	Retorna o escalamento dos eixos para o modo automático padrão: xmin=min(x), xmax=max(x) etc.
axis manual	Congela os dados de escala atuais, de modo que, usando o comando *hold on*, os gráficos subseqüentes mantenham os mesmos limites.
axis xy	Usa o sistema de coordenadas cartesianas (que é o padrão), de modo que a *origem do gráfico* (o par com as menores coordenadas) apareça no canto inferior esquerdo. O eixo horizontal cresce da esquerda para a

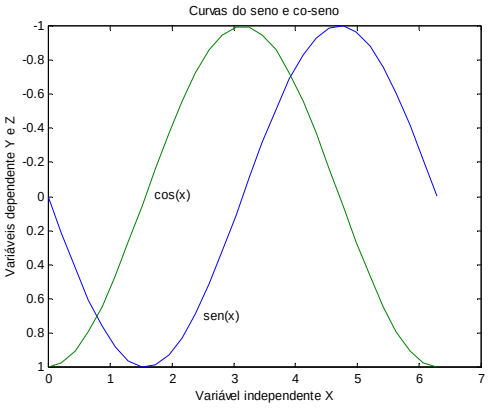
	direita e o vertical cresce de baixo para cima.
axis ij	Usa o sistema de coordenadas matriciais, de modo que a origem do gráfico apareça no canto superior esquerdo. O eixo horizontal cresce da esquerda para a direita, mas o eixo vertical cresce de cima para baixo.
axis square	Faz com que o gráfico atual tenha a forma de um quadrado, em lugar do retângulo habitual.
axis equal	Define os valores de escalamento para ambos os eixos como sendo iguais.
axis tightequal axis tight	O mesmo que *axis equal*, mas ajustando a caixa do desenho de modo a torná-la tão pequena quanto possível.
Axis vis3d	Mantém a proporção entre os eixos, evitando que seja alterada quando há mudança de ponto de vista.
Axis normal	Desfaz o efeito dos comandos *axis square*, *equal*, *tight* e *vis3d*.

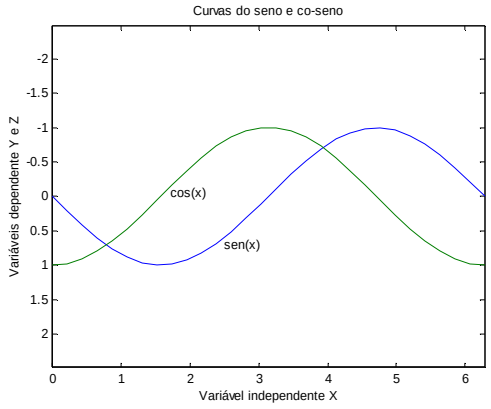
» axis off % Retirando os eixos
» axis on % Devolvendo os eixos



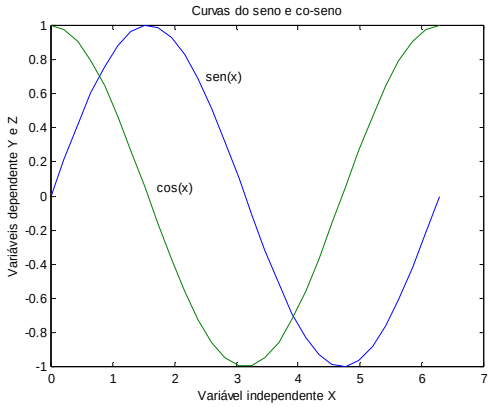


» axis ij % Virando o gráfico de cabeça para baixo
» axis square equal % Comando axis com dois parâmetros





» axis xy normal % Voltando aos parâmetros originais do MATLAB



22.5. Impressão de Figuras

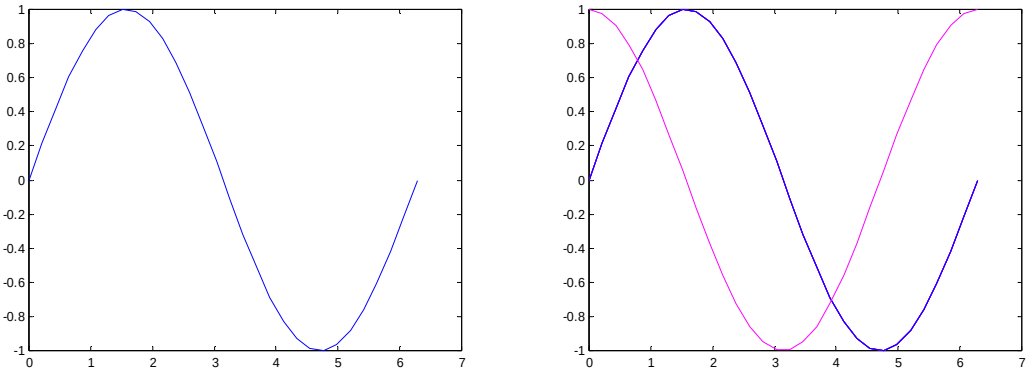
Para imprimir uma janela de figuras, dê um clique com o mouse ou use o comando *figure(n)* para ativá-la e, então, use o comando *print*.

```
» figure(1)
» print      %   Imprime o gráfico atual em sua impressora
» orient     %   Orientação atual

ans =

portrait

» orient landscape      %   Imprime na horizontal
» orient tall      %   Espicha a figura para preencher a página na vertical
```

22.6. Manipulação de Gráficos

É possível adicionar curvas a um gráfico que já existe usando o comando *hold*.

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

» hold on
» ishold      %   Esta função lógica retorna 1 se hold on está em uso

ans =

    1

» plot(x,z,'m')
» hold off
» ishold      %   Hold on não está mais em uso

ans =

    0
```

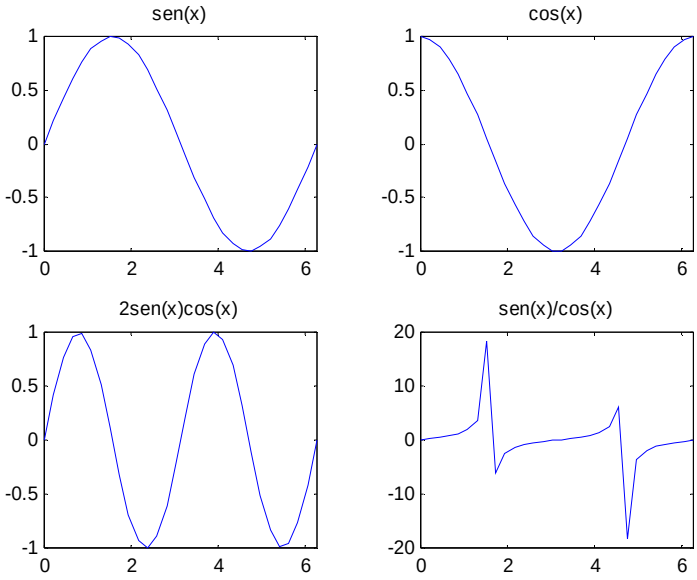


Para desenhar dois ou mais gráficos em diferentes janelas, use o comando *figure* na janela de *comandos*, ou o item *New Figure* do menu *File* da janela de comandos ou da janela de figuras. O comando *figure* sem parâmetros cria uma nova janela.

É possível escolher uma janela específica para ser usada com padrão selecionando-a com o mouse ou utilizando o comando *figure(n)*, em que *n* é o número da janela que será usada pelos comandos *plot* subseqüentes.

Uma janela pode conter mais de um conjunto de eixos. O comando *subplot(m,n,p)* subdivide a janela de figuras atual em uma matriz com *m* por *n* regiões nas quais se pode traçar gráficos e ativa a região de ordem *p*. Os subgráficos são numerados da esquerda para a direita, primeiro na linha superior, depois na segunda linha e daí por diante.

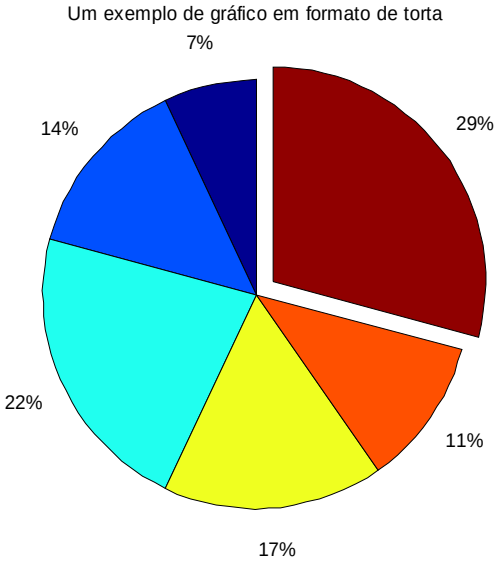
```
» x=linspace(0,2*pi,30);
» y=sin(x);
» z=cos(x);
» a=2*sin(x).*cos(x);
» b=sin(x)./(cos(x)+eps);
» subplot(2,2,1)     %   Selecionando o gráfico no alto, à esquerda
» plot(x,y), axis([0 2*pi -1 1]), title('sen(x)')
» subplot(2,2,2)     %   Selecionando o gráfico no alto, à direita
» plot(x,z), axis([0 2*pi -1 1]), title('cos(x)')
» subplot(2,2,3)     %   Selecionando o gráfico em baixo, à esquerda
» plot(x,a), axis([0 2*pi -1 1]), title('2sen(x)cos(x)')
» subplot(2,2,4)     %   Selecionando o gráfico em baixo, à direita
» plot(x,b), axis([0 2*pi -20 20]), title('sen(x)/cos(x)')
```



22.7. Outros Recursos dos Gráficos Bidimensionais

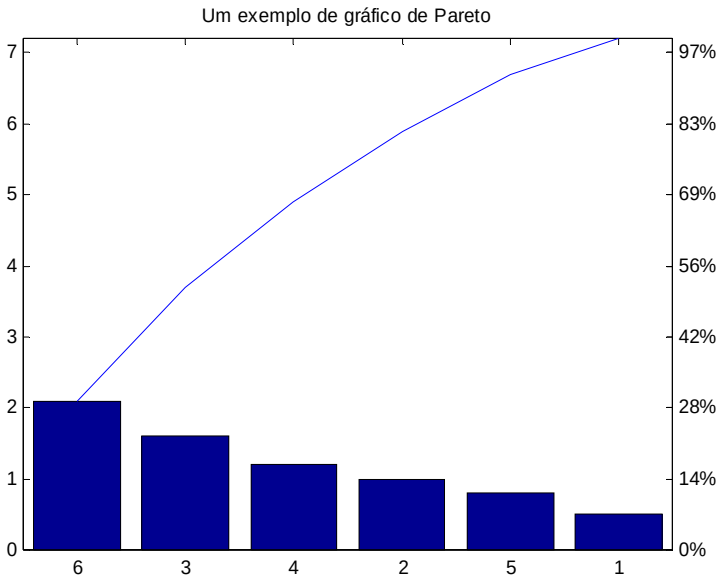
- *loglog* é equivalente ao comando *plot*, exceto pelo fato de escalas logarítmicas serem usadas para ambos os eixos.
- *semilogx* é equivalente ao comando *plot*, exceto pelo fato de uma escala logarítmica ser usada para o eixo *x*, enquanto o eixo *y* usa uma escala linear.
- *semilogy* é equivalente ao comando *plot*, exceto pelo fato de uma escala logarítmica ser usada para o eixo *y*, enquanto o eixo *x* usa uma escala linear.
- *area(x,y)* é equivalente a *plot(x,y)*, exceto pelo fato de que a área entre os valores *0* e *y* é preenchida. O valor de *y* que serve como base para o preenchimento pode ser especificado, sendo igual a zero em caso contrário.
- Gráficos em formato de torta (ou pizza) podem ser criados usando o comando *pie(a,b)*, em que *a* é um vetor contendo valores e *b* é um vetor lógico opcional que descreve quais "fatias" devem ser separadas do resto.

```
» a=[.5 1 1.6 1.2 .8 2.1];
» pie(a,a==max(a));      %   Traça o gráfico e separa a maior fatia
» title('Um exemplo de gráfico em formato de torta')
```



- Utilizando o gráfico de Pareto, os dados são mostrados na forma de barras, em ordem decrescente, juntamente com uma curva que fornece os valores acumulados associados aos pontos.

» pareto(a);
» title('Um exemplo de gráfico de Pareto')



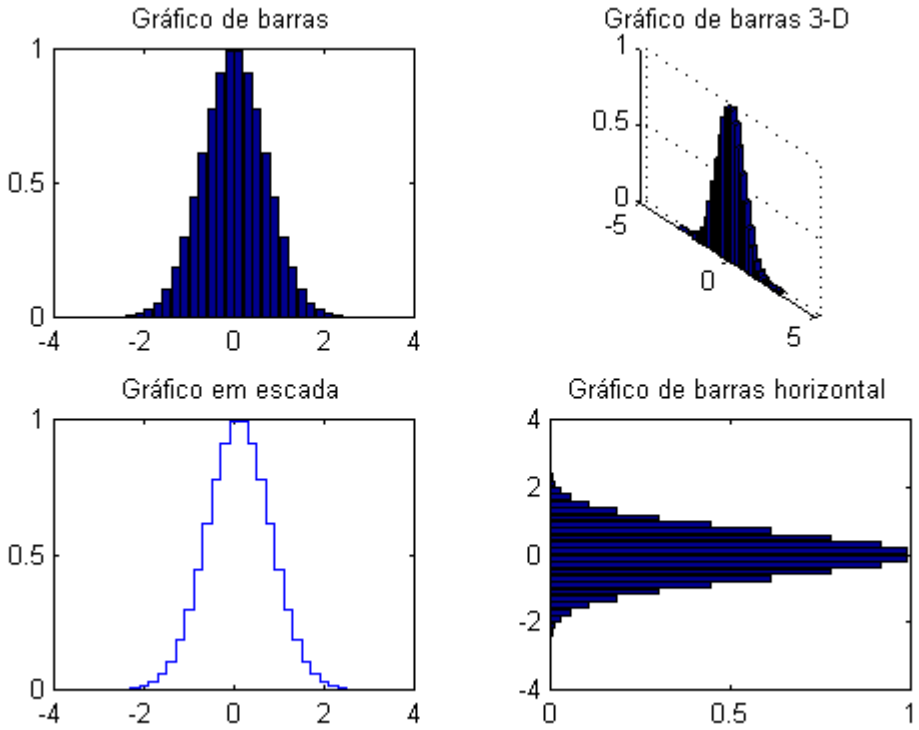
- Para traçar curvas diferentes no mesmo gráfico usando escalas diferentes par o eixo y, usa-se a função *plotyy*.

» x=-2*pi:pi/10:2*pi;
» y=sin(x);
» z=2*cos(x);
» subplot(2,1,1), plot(x,y,x,z)
» title('Dois gráficos com a mesma escala.')
» subplot(2,1,2), plotyy(x,y,x,z)

» title('Dois gráficos com escalas diferentes.');

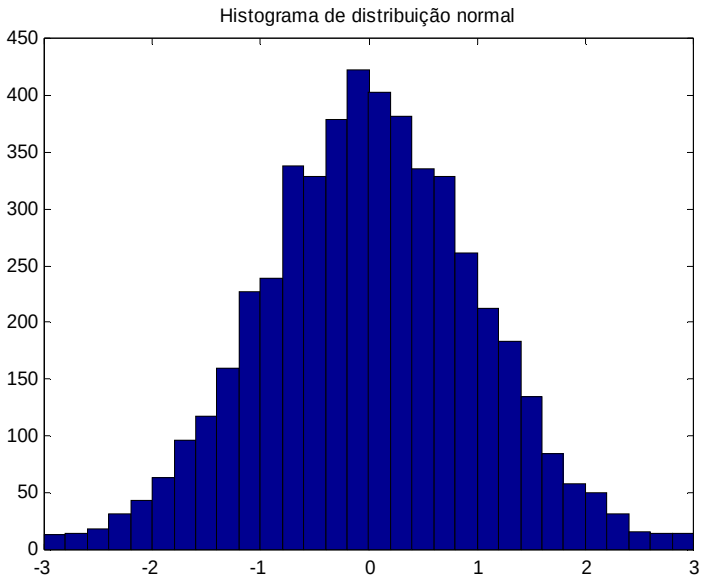
- Gráficos de barras e em escada podem ser gerados por meio dos comandos *bar*, *bar3*, *barh* e *stairs*. Apresenta-se agora um exemplo de curvas em forma de sino:

```
» x=-2.9:0.2:2.9;
» y=exp(-x.*x);
» subplot(2,2,1)
» bar(x,y)
» title('Gráfico de barras')
» subplot(2,2,2)
» bar3(x,y)
» title('Gráfico de barras 3-D')
» subplot(2,2,3)
» stairs(x,y)
» title('Gráfico em escada')
» subplot(2,2,4)
» barh(x,y)
» title('Gráfico de barras horizontal')
```



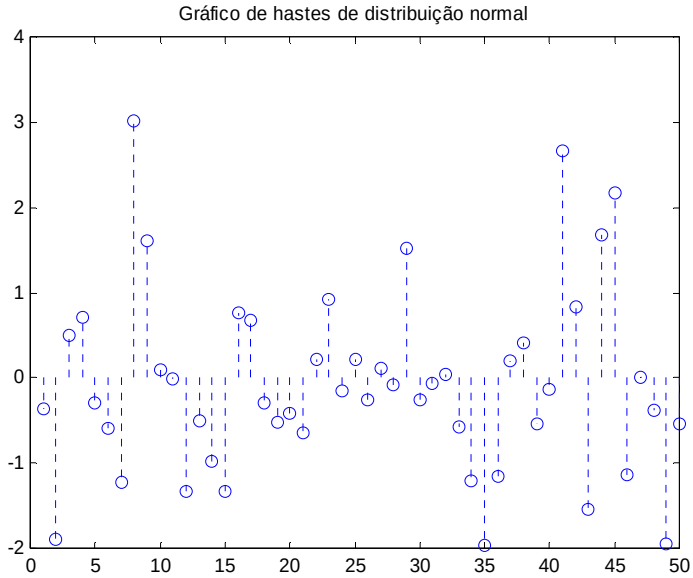
- O comando *hist(y)* cria um histograma de dez divisões para os dados no vetor y.*hist(y,n)*, em que *n* é um escalar, desenha um histograma com *n* divisões. *hist(y,x)*, em que *x* é um vetor, desenha um histograma usando as divisões especificadas em *x*. Apresenta-se agora um exemplo de histograma de uma curva em forma de sino a partir de dados Gaussianos:

```
» x=-2.9:0.2:2.9;      %   Especifica as divisões a serem usadas
» y=randn(5000,1);     %   Cria 5000 pontos aleatórios, com distribuição normal
» hist(y,x);           %   Desenha o histograma
» title('Histograma de distribuição normal')
```



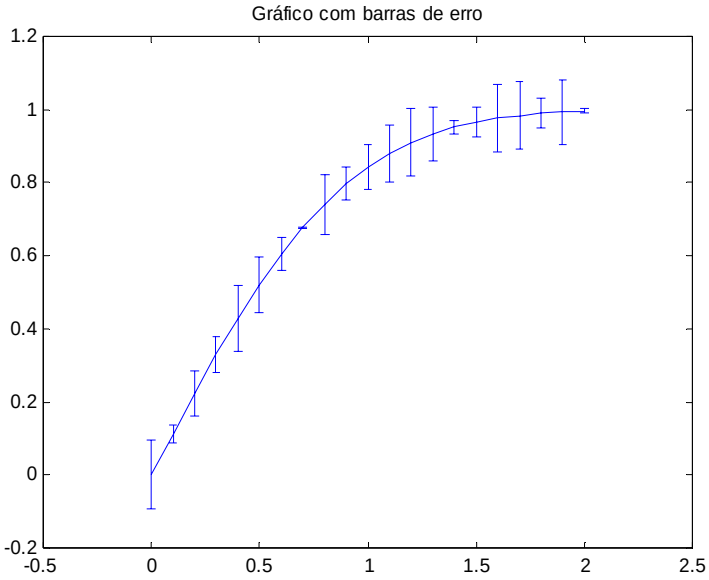
- Seqüências discretas de dados podem ser representadas graficamente por meio da função *stem*. s*tem(z)* cria um gráfico dos dados contidos no vetor *z* conectados ao eixo horizontal por uma linha. Um argumento opcional de string de caracteres pode ser usado para especificar o tipo de linha. *stem(x,z)* cria um gráfico de pontos *z* nos valores especificados de *x*.

```
» z=randn(50,1);      %   Cria dados aleatórios com distribuição normal
» stem(z,':');      %   Cria um gráfico de hastes com linhas pontilhadas
» title('Gráfico de hastes de distribuição normal')
```



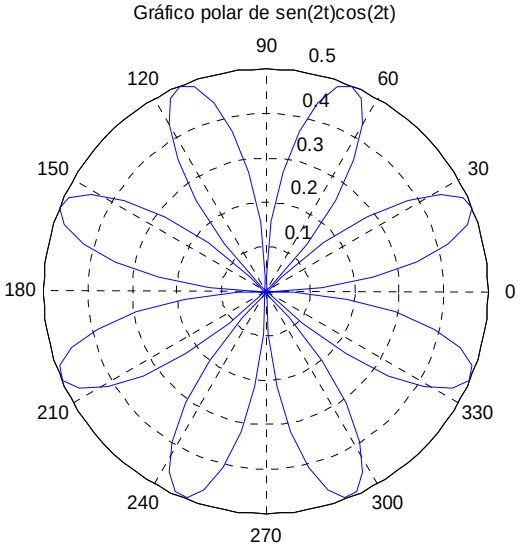
- Pode-se colocar barras de erro nos gráficos. *errorbar(x,y,e)* cria um gráfico do vetor *x versus* vetor *y*, com barras de erro especificadas pelo vetor *e*. Todos os vetores dever ter o mesmo comprimento. Para cada ponto (x_i,y_i), uma barra de erro é desenhada a uma distância e_i acima e e_i abaixo de cada ponto.

```
» x=linspace(0,2,21);      %   Cria um vetor
» y=erf(x);      %   y é a função erro de x
» e=rand(size(x))/10;      %   e contém valores de erro aleatórios
» errorbar(x,y,e);      %   Cria o gráfico
» title('Gráfico com barras de erro')
```



- Gráficos em coordenadas polares podem ser criados pelo comando *polar(t,r,S)*, em que *t* é o vetor que contém o ângulo em radianos, *r* é o vetor que contém o raio e *S* é uma string de caracteres opcional que pode ser usada para descrever as cores, os símbolos ou o estilo de linha. Veja o comando *plot* para uma descrição de strings apropriadas.

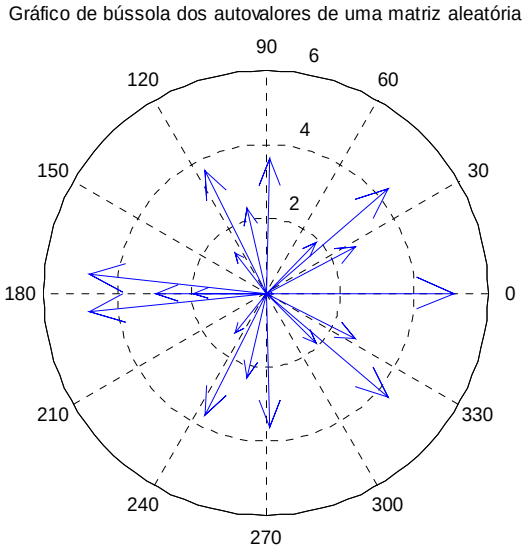
```
» t=linspace(0,2*pi);
» r=sin(2*t).*cos(2*t);
» polar(t,r)
» title('Gráfico polar de sen(2t)cos(2t)')
```



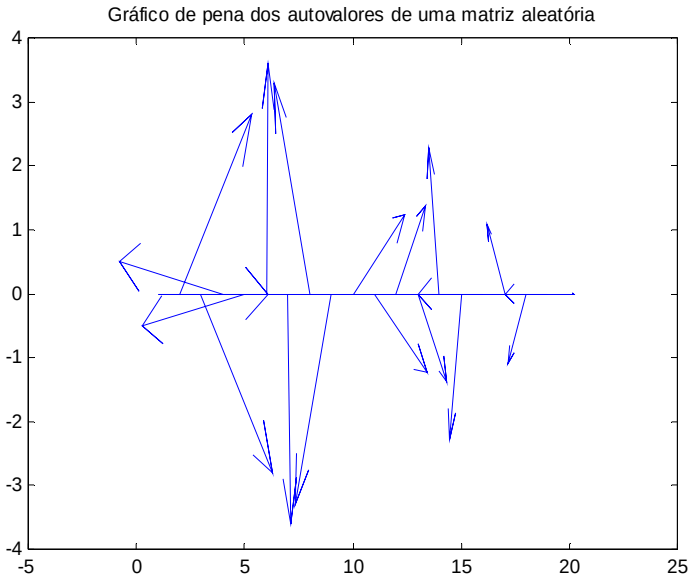
- Dados complexos podem ser representados graficamente com os comandos *compass* (bússola) e *feather* (pena). *compass(z)* desenha um gráfico que mostra o ângulo e o módulo dos elementos complexos de *z* na forma de setas partindo da origem. *feather(z)* cria o gráfico com os mesmo dados usando setas que

saem de pontos eqüidistantes em uma linha horizontal. *compass(x,y)* e *feather(x,y)* são equivalentes a *compass(x+i*y)* e *feather(x+i*y)*.

» z=eig(randn(20,20));
» compass(z)
» title('Gráfico de bússola dos autovalores de uma matriz aleatória')

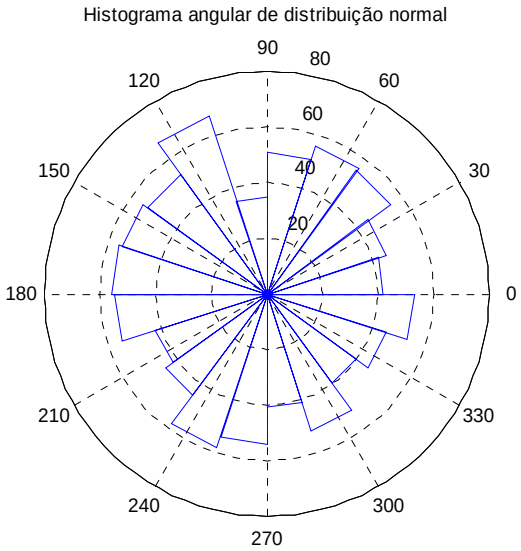


» feather(z)
» title('Gráfico de pena dos autovalores de uma matriz aleatória')



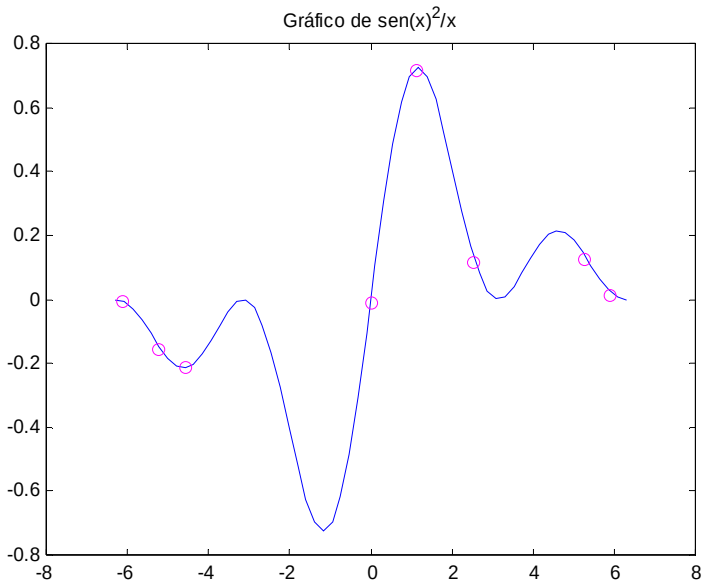
- O comando *rose(v)* desenha um histograma polar com 20 divisões para os ângulos no vetor *v*. *rose(v,n)*, em que *n* é um escalar, desenha um histograma com *n* divisões. *rose(v,x)* em que *x* é um vetor, desenha um histograma usando as divisões especificadas em *x*. A seguir, um exemplo de um histograma em ângulo:

```
» v=randn(1000,1)*pi;
» rose(v)
» title('Histograma angular de distribuição normal')
```



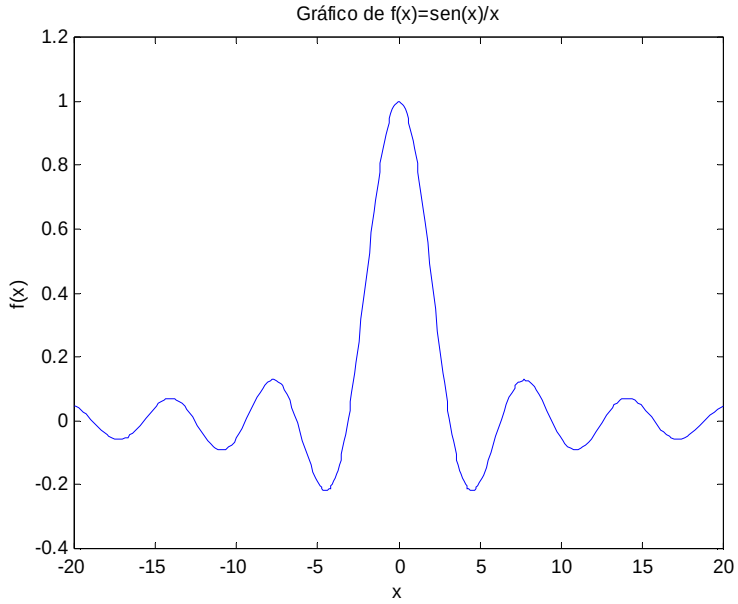
- O comando *ginput* oferece uma forma para selecionar pontos de um gráfico em uso por meio do mouse. *[x,y]=ginput(n)* toma até *n* pontos dos eixos atuais e retorna suas coordenadas nos vetores coluna *x* e *y*. Se *n* é omitido, um número ilimitado de pontos é reunido até que a tecla *Return* ou *Enter* seja pressionada. Como exemplo, criaremos o gráfico de uma função e então marcaremos alguns pontos selecionados com o mouse.

```
» x=linspace(-2*pi,2*pi,60);
» y=sin(x).^2./(x+eps);
» plot(x,y)
» title('Gráfico de sen(x)^2/x')
» [a,b]=ginput(8);      %   toma até 8 pontos
» hold on
» plot(a,b,'mo')      %   Plota os pontos coletados
» hold off
```



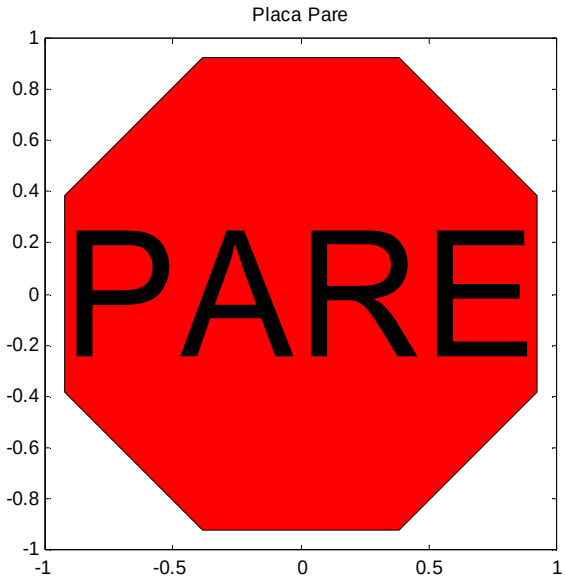
- O comando *fplot* oferece uma maneira fácil de criar o gráfico de uma função de uma variável entre limites especificados, sem criar um conjunto de dados para a variável. *fplot('fun',[xmin xmax])* cria um gráfico da função *fun* sobre o intervalo xmin ≤ x ≤ xmax com escalamento automático do eixo *y*. *fplot('fun',[xmin xmax ymin ymax])* especifica também os limites do eixo y.

» fplot('sin(x)./x',[-20 20 -.4 1.2])
» title('Gráfico de f(x)=sen(x)/x')
» xlabel('x')
» ylabel('f(x)')



- O comando *fill(x,y,'c')* preenche o polígono bidimensional definido pelos vetores coluna *x* e *y* com a cor definida por *c*. Os vértices do polígono são especificados pelos pares (x_i,y_i). Se necessário, o polígono é fechado pela conexão do último vértice ao primeiro.

```
» t=(1/8:2/8:15/8)'*pi;      %   Vetor coluna[pi/8;3pi/8;...;15pi/8]
» x=sin(t);
» y=cos(t);
» fill(x,y,'r')      %   Um círculo preenchido com vermelho usando somente 8 pontos
» axis square
» text(0,0,'PARE','FontSize',88,'HorizontalAlignment','center')
» title('Placa Pare')
```



23. Ajustes de Curvas e Interpolação

Quando é necessário descrever dados obtidos experimentalmente por meio de uma função analítica, tem-se duas alternativas. Usando interpolação, parte-se do pressuposto de que os dados estejam corretos e procura-se alguma maneira de descrever o que acontece entre os pontos dados. Usando ajuste de curvas ou regressão, o objetivo é encontrar alguma curva suave que "melhor ajuste" aos dados, mas que não necessariamente passe por quiasquer dos pontos.

23.1. Ajuste de Curvas

O "melhor ajuste" é interpretado como sendo a minimização da soma dos quadrados dos erros nos pontos dados e a curva utilizada é restrita a polinômios. Isso é denominado ajuste de curvas por quadrados mínimos a um polinômio. A função *polyfit* resolve o problema do ajuste de curvas pelo método dos quadrados mínimos.

```
» x=[0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1];
» y=[-.447 1.978 3.28 6.16 7.08 7.34 7.66 9.56 9.48 ...
9.30 11.2];
```

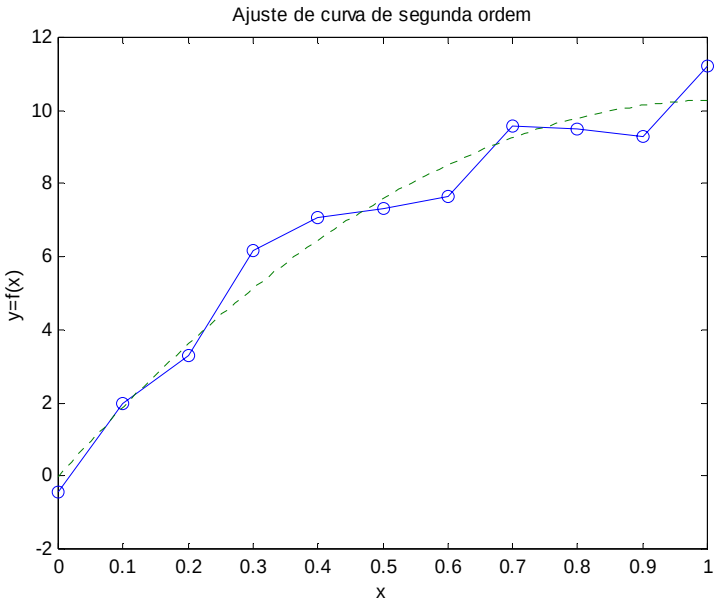
Para usar *polyfit*, além dos dados anteriores, devemos fornecer a ordem ou grau do polinômio. Se escolhermos *n=1* para o grau do polinômio, a melhor aproximação linear será encontrada. Isso é freqüentemente chamado de regressão linear. Por outro lado, se escolhermos *n=2*, um polinômio quadrático será encontrado.

```
» n=2;
» p=polyfit(x,y,n)

p =
```

-9.8108 20.1293 -0.0317

A saída de *polyfit* é um vetor linha com os coeficientes do polinômio. Aqui a solução é $y = -9{,}8108x^2 + 20{,}1293x - 0{,}0317$. Para comparar a solução do ajuste de curva aos pontos dados, traçamos ambas as curvas:

» xi=linspace(0,1,100);
Cria os dados do eixo x para traçar o polinômio.
» z=polyval(p,xi);
Chama a função *polyval* para calcular o polinômio p nos pontos dados em xi.
» plot(x,y,'-o',xi,z,':')
Mostra os dados originais (x,y), marcando os pontos com 'o' e unindo-os com retas e traça o polinômio definido pelos dados xi e z, usando uma linha pontilhada.
» xlabel('x'), ylabel('y=f(x)')
» title('Ajuste de curva de segunda ordem')
Coloca nomes nos eixos e no gráfico.



A escolha do grau do polinômio é, de certa forma, arbitrária. São necessários dois pontos para definir uma reta ou um polinômio de primeira ordem. São necessários três pontos para definir um polinômio quadrático ou de segunda ordem. Seguindo-se essa progressão, necessita-se de $n+1$ pontos para especificar de forma única um polinômio de ordem n. Dessa forma, no caso anterior, em que há 11 pontos, poderíamos escolher um polinômio até a décima ordem. Entretanto, dadas as propriedades numéricas pobres de polinômios de ordem elevada, não se deve escolher uma ordem maior que a necessária. Além disso, à medida que a ordem do polinômio aumenta, a aproximação torna-se menos suave, já que os polinômios de ordem superior podem ser derivados mais vezes até que se tornem zero. Por exemplo, escolhe-se um polinômio de décima ordem:

» pp=polyfit(x,y,10);
» format short e % Muda o formato de visualização de números
» pp.' % Mostra os coeficientes do polinômio como uma coluna

ans =

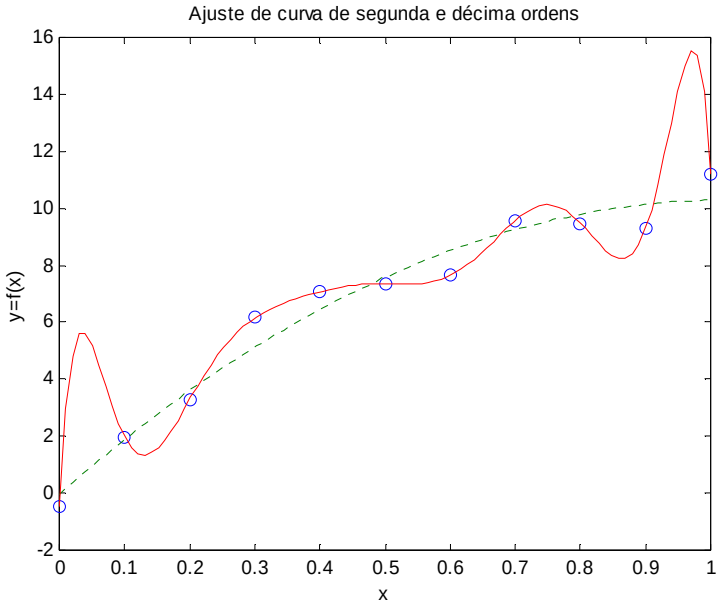
-4.6436e+005
2.2965e+006

```
-4.8773e+006
 5.8233e+006
-4.2948e+006
 2.0211e+006
-6.0322e+005
 1.0896e+005
-1.0626e+004
 4.3599e+002
-4.4700e-001
```

Observe a ordem de grandeza dos coeficientes do polinômio nesses caso, em comparação com aqueles obtidos no ajuste quadrático anterior. Observe também a diferença de sete ordens de grandeza entre o menor (-4,4700e-01) e o maior (5,8233e+06) coeficiente.

```
» zz=polyval(pp,xi);      %   Calcula o polinômio de décima ordem
» plot(x,y,'o',xi,z,':',xi,zz)      %   Representa graficamente os dados
» xlabel('x'), ylabel('y=f(x)')
» title('Ajuste de curva de segunda e décima ordens')
```

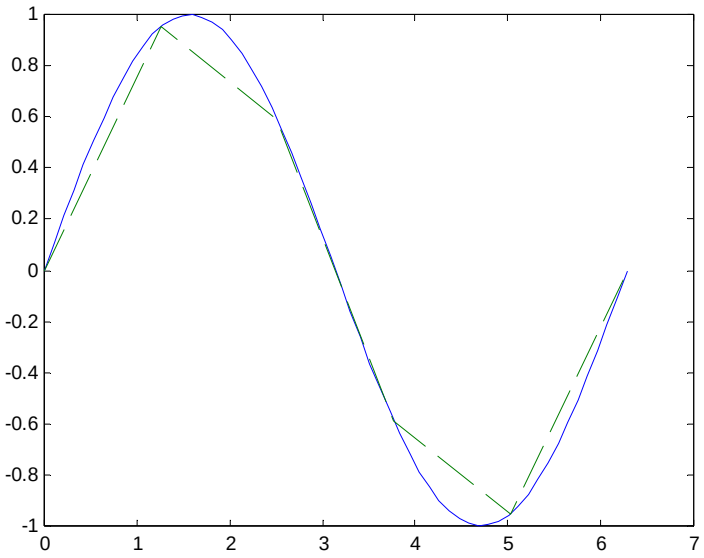
Na figura abaixo, os dados originais estão marcados com ‘o’, o ajuste de curvas quadrático encontra-se pontilhado e o ajuste de curva de décima ordem está representado por uma linha contínua. Observe as ondulações que aparecem entre os pontos dados nos extremos esquerdo e direito da curva do ajuste de décima ordem. A filosofia do “quanto mais melhor” não necessariamente se aplica aqui.



23.2. Interpolação Unidimensional

Por definição, o MATLAB desenha linhas retas interligando os pontos usados para criar um gráfico. Essa interpolação linear considera que os valores intermediários caem em uma linha reta entre os pontos definidos. À medida que se têm mais dados e a distância entre eles diminui, a interpola;cão linear torna-se mais precisa.

```
» x1=linspace(0,2*pi,60);
» x2=linspace(0,2*pi,6);
» plot(x1,sin(x1),x2,sin(x2),'--')
» xlabel('x'), ylabel('sen(x)'), title('Interpolação linear')
```

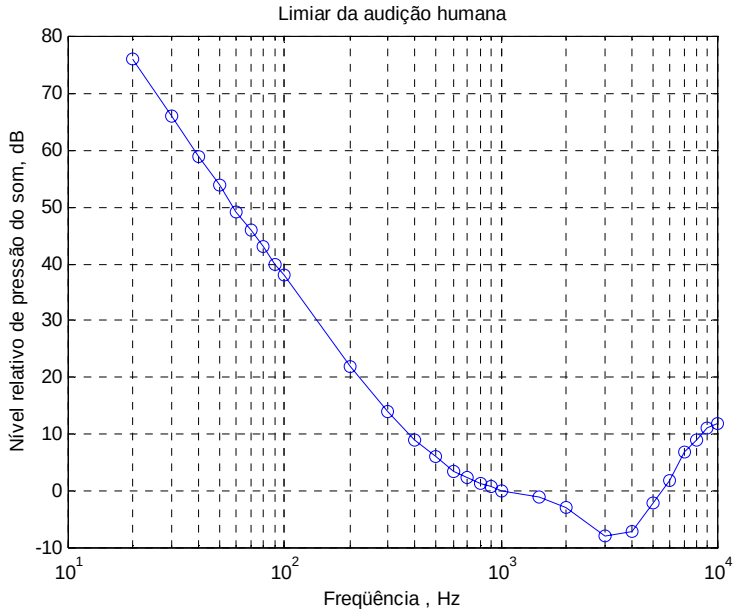


Dos dois gráficos da função seno, o que tem 60 pontos é muito mais suave e mais preciso entre os pontos do que o que só tem 6 pontos.

O MATLAB apresenta diversas funções de interpolação, *interp1* interpola dados unidimensionais; *interp2* interpola dados bidimensionais,; *interp3* interpola dados tridimensionais; e *interpn* interpola dados de dimensões maiores.

Para ilustrar a interpolação unidimensional, tem-se o exemplo do limiar da audição, isto é, o nível mínimo de som perceptível ao ouvido humano que varia com a freqüência. Medidas típicas são:

```
» Hz=[20:10:100 200:100:1000 1500 2000:1000:10000];      %   Freqüência em Hertz
» semilogx(Hz,nps,'-o')
» nps=[76 66 59 54 49 46 43 40 38 22 ...
14 9 6 3.5 2.5 1.4 0.7 0 -1 -3 ...
-8 -7 -2 2 7 9 11 12];      %   Nível de pressão do som em decibéis (dB)
» semilogx(Hz,nps,'-o')
» xlabel('Freqüência , Hz')
» ylabel('Nível relativo de pressão do som, dB')
» title('Limiar da audição humana')
» grid on
```



Com base na Figura acima, a audição humana é mais sensível a sons por volta de 3 kHz. Com esses dados, vamos estimar, por diferentes modos, o nível de pressão do som em uma freqüência 2,5 kHz.

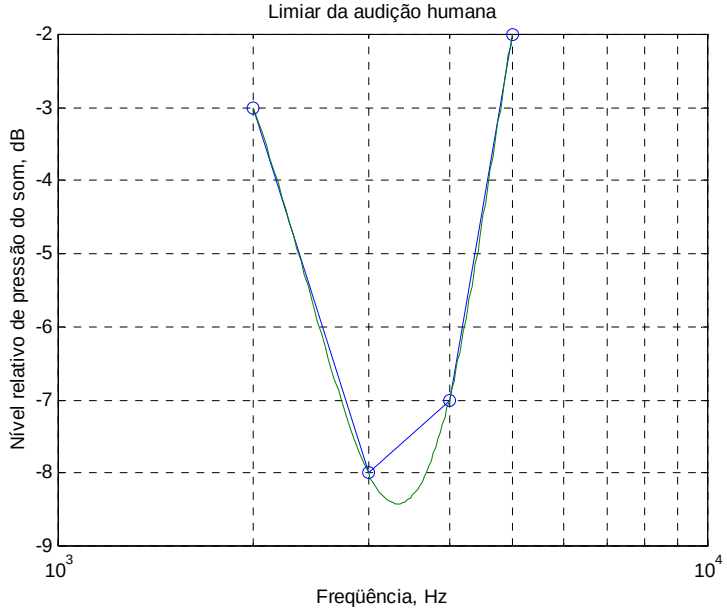
```
» s=interp1(Hz,nps,2.5e3)      %   Interpolação Linear

s =

  -5.5000

» s=interp1(Hz,nps,2.5e3,'linear')      %   Interpolação Linear novamente

s =

  -5.5000

» s=interp1(Hz,nps,2.5e3,'cubic')      %   Interpolação Cúbica

s =

  -5.8690

» s=interp1(Hz,nps,2.5e3,'spline')      %   Interpolação por Splines

s =

  -5.8690

» s=interp1(Hz,nps,2.5e3,'nearest')      %   Vizinho mais próximo

s =

   -8
```

O primeiro dos dois resultados retorna exatamente o que é mostrado na figura em 2,5 kHz, já que o MATLAB interpola linearmente entre os pontos dados em gráficos. As repostas da interpolação cúbica e interpolação usando splines são próximas entre si, uma vez que ambas ajustam cúbicas, isto é, polinômios de

terceira ordem, ainda que de modos diferentes. O pior resultado de interpolação é aquele fornecido pelo vizinho mais próximo, pois, nesse caso, o valor obtido nada mais é que o dado de entrada mais próximo do valor dado.

Dado um conjunto de pontos, usa-se a interpolação por splines para calcular os dados em intervalos menores.

```
» Hzi=linspace(2e3,5e2);      %   Olhando muito perto do mínimo
» npsi=interp1(Hz,nps,Hzi,'cubic');      %   Interpolando perto do mínimo
» i=find(Hz>=2e3 & Hz<=5e3);
» %   Determinando os índices dos dados originais perto do mínimo
» semilogx(Hz(i),nps(i),'-o',Hzi,npsi)      %   Representando graficamente dados velhos e novos
» xlabel('Freqüência, Hz')
» ylabel('Nível relativo de pressão do som, dB')
» title('Limiar da audição humana')
» grid on
```



A linha tracejada corresponde à interpolaçào linear, a linha contínua, à interpolação cúbica e os dados originais encontram-se marcados com ‘o’. Buscando uma resolução mais refinada no eixo das freqüências e usando interpolação cúbica, temos uma estimativa mais suave do nível de pressão do som. A inclinação da solução cúbica não muda abruptamente nos pontos dados.

Assim, pode-se fazer uma estimativa melhor da freqüência de maior sensibilidade:

```
» [nps_min,i]=min(npsi)      %   Mínimo e índice do mínimo

nps_min =

  -8.4245


i =

   45

» Hz_min=Hzi(i)      %   Freqüência no mínimo

Hz_min =
```

3.3333e+003

O ouvido humano é mais sensível a sons próximos de 3,3 kHz.

Interp1 possui duas restrições, não se pode pedir resultados fora do limite da variável independente: por exemplo, interp1(Hz,nps,1e5) conduziria a um erro (resultaria em NaN). A variável independente tem de ser monotônica, isto é, precisa sempre crescer ou sempre decrescer. Hz é monotônica.

24. Referências Bibliográficas

- HANSELMAN, Duane, LITTLEFIELD, Bruce. *"MATLAB 5 – Guia do Usuário"*. Makron Books. 1997.
- TRINDADE, Marcelo, SAMPAIO, Rubens. *"Introdução ao MATLAB"*. PUC-RIO. 2002.
- SILVA, José Demisio Simões. *"Introdução ao Ambiente MATLAB"*. 1997. http://www.eletrica.ufpr.br/edu/pds_/manual.html
- NETO, Luiz Antonio Salgado. *"Aprendendo o MATLAB"*. 1995. http://www.lasalgado.eng.br/trabtec/apmlab/