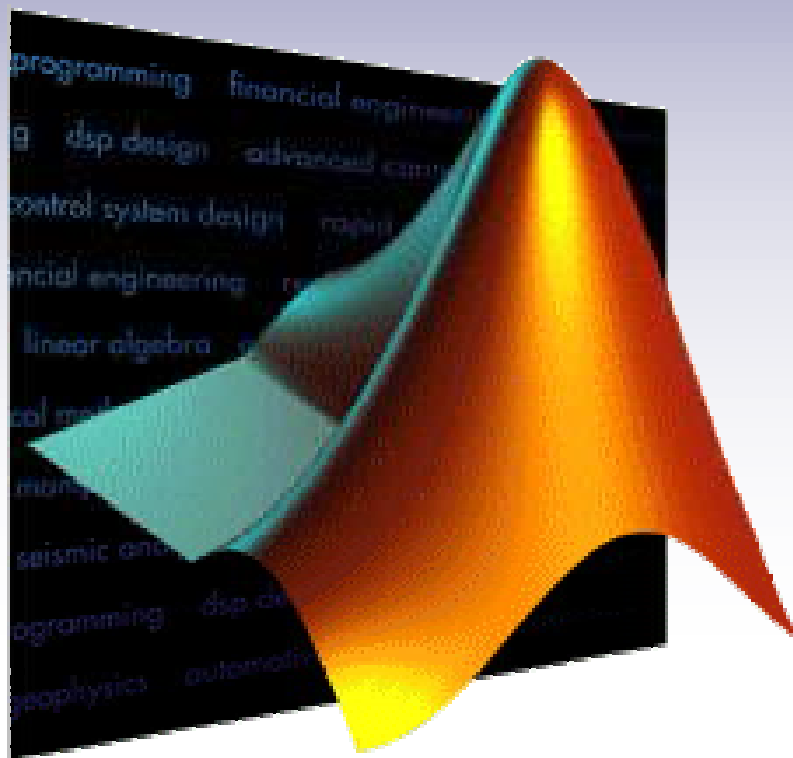


2ª EDIÇÃO

Revista e Ampliada

Curso de MATLAB 5.1

Introdução à Solução de Problemas de Engenharia



The
**MATH
WORKS**
Inc.



Faculdade de
Engenharia



Laboratório de
Engenharia Elétrica

Programa Prodenge / Sub-Programa Reenge
Universidade do Estado do Rio de Janeiro

AGRADECIMENTOS

Estas notas sobre o uso da versão 5.1 do MATLAB são o resultado do trabalho persistente dos alunos da Faculdade de Engenharia da UERJ, bolsistas de iniciação Tecnológica do Projeto REENGE - Joana Figueiredo Konte, Jorge Luís Pinheiro Teixeira, Pat Evie Alves - e da estagiária Luciana Faletti que se encarregaram de dar corpo à segunda edição de um curso de Introdução à Solução de Problemas de Engenharia usando a metodologia da Profa. Delores M. Etter, autora da obra 'Engineering Problem Solving with MATLAB' que inspirou, de perto, a confecção desta apostila. A este grupo entusiasmado de jovens, aderiram outros estagiários do Laboratório de Engenharia Elétrica, como Hélio Justino Mattos Filho. A todos eles os cumprimentos pelo êxito e pela forma como se envolveram de corpo e alma na execução das tarefas. O sucesso obtido na implementação de ambos os cursos não é sem dúvida fruto de uma obra isolada. Dela participaram, com entusiasmo a equipe técnico-administrativa do Laboratório de Engenharia Elétrica, cujos membros contribuíram com a dedicação que lhes é peculiar, através do suporte e infra-estrutura e o envolvimento direto com os alunos e com a coordenação do projeto. Um muito obrigado à equipe formada pelos funcionários Alberto Avelar Santiago, André Vallim Stachlewski, José Emílio Gomes, Jair Medeiros Júnior, João Elias Souza da Costa, Luiz Roberto Franco Fagundes Filho, Marcos Augusto Mafra, Antônio Marcos Medeiros Corrêa, Sueli Ferreira dos Santos e pela Srta. Carla Aparecida Caldas de Almeida, do curso de Pós-Graduação 'latu-senso' em Engenharia Mecatrônica da UERJ. Uma palavra de reconhecimento especial ao diretor Dr. Nival Nunes de Almeida, coordenador geral do REENGE, pelo apoio e pelo incentivo dado à viabilização de inúmeras atividades no âmbito da faculdade como um todo e do LEE em particular. À Profa. Maria Eugênia Mosconi de Golveia, vice-diretora da faculdade de Engenharia uma palavra de gratidão pelo empenho em viabilizar juntamente com o diretor as solicitações de estágio interno no LEE. Ao grupo de colaboradores silenciosos da administração pelo apoio nas atividades no âmbito de suas competências, o obrigado sincero da Orientação do trabalho. Ao CNPq que patrocinou as bolsas que permitiram este trabalho mediante os recursos alocados pela FINEP, o nosso agradecimento.

Bernardo Severo da Silva Filho
Orientador e chefe do Lab. De Engenharia Elétrica

Índice

1	INTRODUÇÃO À SOLUÇÃO DE PROBLEMAS	1
2	MATRIZES, VETORES E ESCALARES	4
2.1	Definindo matrizes no MATLAB	5
	Método Simples	6
	Arquivos MAT e ASCII	6
	Operador dois pontos	8
	Comando Input	8
	Imprimindo matrizes	11
	Comando format	11
	Comando disp	12
	Comando fprintf	12
2.2	Gráficos X-Y	13
	Aplicação à Solução de Problemas: Análise de um túnel de vento	15
3	CÁLCULOS FUNDAMENTAIS E MATRIZES ESPECIAIS	16
3.1	Valores Especiais e Matrizes Especiais	16
	Magic Square	17
	Matriz de Zeros	17
	Matriz de um's	17
	Matriz identidade	17
	Triângulo de Pascal	17
3.2	Operações entre escalares	18
	Hierarquia em operações aritméticas	19
	Limites Computacionais	21
3.3	Operações de Conjuntos	21
	Aplicação à solução de problemas: Ecos em sinais de comunicação	25
3.4	Funções Elementares	29
	Funções matemáticas elementares	30
	Funções trigonométricas	31
	Funções hiperbólicas	32
	Funções de Arquivos M	32
	Aplicação à solução de problemas: sinais de sonar	34
3.5	Números Complexos	36
	Operações aritméticas com complexos	37
	Coordenadas polares e retangulares	37

4	CONTROLE DE FLUXO	40
4.1	Operadores lógicos e relacionais	40
4.2	Tomada de decisões	42
	Estrutura if-then-else	42
4.3	Loop FOR	45
	Comando break	47
	Aplicação à solução de problemas: fibras óticas	47
4.4	Loop WHILE	49
	Aplicação à solução de problemas: equilíbrio de temperatura	50
5	MEDIDAS ESTATÍSTICAS	54
5.1	Funções para análise de dados	56
	Desvio médio, variância e desvio padrão	56
	Comando sort	60
	Histograma	61
	Aplicação à solução de problemas: análise do sinal de voz	64
5.2	Números Aleatórios	66
	Função número aleatório	66
	Função Densidade de Probabilidade	66
	Modelo uniforme	68
	Modelo normal	68
	Histograma: comando hist	71
	Aplicação à Solução de Problemas: simulador de voo	73
5.3	Relação Sinal/Ruído	75
	Energia de um sinal	75
	Cálculo de SNR	76
	Adicionando um ruído a um sinal existente	77
6	OPERAÇÕES COM MATRIZES	79
6.1	Operações com matrizes	79
	Matriz transposta	79
	Somatório de produtos	79
	Comando sum	80
	Multiplicação de matrizes	80
	Matriz Power	81
	Matriz inversa	81
	Determinante	82
	Aplicação à Solução de Problemas: peso molecular de proteínas	82

6.2	Manipulação com matrizes	84
	Comando rot90	84
	Comando fliplr	84
	Comando flipud	84
	Comando reshape	85
	Comando diag	85
	Comando triu	86
	Comando tril	87
	Aplicação à Solução de Problemas: alinhamento de imagens	87
7	GRÁFICOS	91
7.1	Gráficos X-Y	91
	Coordenadas retangulares	91
	Legendas	91
7.2	Gráficos Polares	92
	Coordenadas Polares	92
	Transformações retangular/polar	93
	Gráficos de barras e degrau	94
7.3	Opções	94
7.4	Gráficos 3D	97
	Aplicação à Solução de Problemas: trajetória de um satélite	100
8	SOLUÇÕES DE SISTEMAS DE EQUAÇÕES LINEARES	101
8.1	Interpretação Gráfica	101
8.2	Solução usando operações matriciais	103
	Divisão de matrizes	104
	Matriz Inversa	104
	Aplicação à Solução de Problemas: análise de um circuito elétrico	105
9	INTERPOLAÇÃO E AJUSTE DE CURVAS	106
9.1	Interpolação	106
	Interpolação linear	107
	Função table1	107
	Função table2	109
	Comando spline	110
	Aplicação à Solução de Problemas: braço robótico	112

9.2	Ajuste de curvas	113
	Regressão Linear	113
	Comando polyfit	114
	Comando polyval	115
10	ANÁLISE POLINOMIAL	116
10.1	Avaliação do polinômio	116
	Comando polyval	116
	Operações Aritméticas	117
	Aplicação à Solução de Problemas: balões meteorológicos	118
10.2	Raízes de polinômios	120
11	INTEGRAÇÃO NUMÉRICA E DIFERENCIAÇÃO	122
11.1	Integração Numérica	122
	Regra Trapezoidal e Regra de Simpson	122
	Comando Quadratura	122
	Aplicação à Solução de Problemas: análise de escoamento de um óleo num oleoduto	123
11.2	Diferenciação Numérica	125
	Derivação por expressão de diferença	126
	Comando diff	127
12	EQUAÇÕES DIFERENCIAIS ORDINÁRIAS	129
12.1	Equações Diferenciais Ordinárias de Primeira Ordem	129
12.2	Método de Runge-Kutta	130
	Aproximação de Primeira Ordem (método de Euler)	130
	Comando ODE	131
	Aplicação à solução de problemas: aceleração de uma turbina UDF numa aeronave	133
12.3	Equações Diferenciais de Ordens Superiores	135
13	FATORAÇÃO E DECOMPOSIÇÃO DE MATRIZES	137
13.1	Autovalores e autovetores	137
	Aplicação à solução de problemas: adaptador para redução de ruídos	141
13.2	Decomposição e Fatoração	143

Fatoração Triangular	143
Fatoração QR	144
14 PROCESSAMENTO DE SINAIS	146
14.1 Análise no domínio da frequência	146
14.2 Análise de filtros	149
Função de Transferência Analógica	149
Função de Transferência Digital	151
14.3 Implementação de Filtros Digitais	153
14.4 Projetos de Filtros Digitais	155
Filtros IIR	156
Filtros FIR	157
Aplicação à solução de problemas: filtros para separação de canais	158
15 MATEMÁTICA SIMBÓLICA	161
15.1 Expressões Simbólicas	161
Representações de Expressões Simbólicas no MATLAB	162
15.2 Variáveis Simbólicas	163
15.3 Operações em expressões simbólicas	165
15.4 Operações Algébricas Padrão	166
Operações Avançadas	167
15.5 Funções de Conversão	169
15.6 Derivação e Integração	170
15.7 Transformadas	171
Transformada de Laplace	173
Transformada de Fourier	173
Transformada Z	174

Capítulo 1 – Uma Introdução à Solução de Problemas

A solução de problemas é parte essencial não somente dos cursos de engenharia mas também dos cursos de Matemática, Física, Química e Ciência da Computação. Logo, é importante uma base sólida em solução de problemas. Também é de grande auxílio um embasamento suficiente para trabalhar em todas estas áreas, para que não tenhamos que aprender uma técnica para problemas de matemática, e uma técnica diferente para problemas de física, e assim por diante. A técnica de solução de problemas que apresentamos trabalhos para problemas de engenharia e pode ser seguida de perto para resolver problemas em outras áreas; mas, supõe-se que estamos usando o MATLAB para ajudar a resolvê-los.

O processo ou metodologia para resolução de problemas que usaremos ao longo do texto possui cinco passos:

1. Enunciar o problema claramente.
2. Descreva a informação de entrada e saída.
3. Trabalhar o problema manualmente.
4. Desenvolver uma solução MATLAB.
5. Testar a solução usando uma variedade de grupo de dados.

Descreveremos cada um dos passos usando o exemplo do cálculo da distância entre dois pontos em um plano.

1. ENUNCIADO DO PROBLEMA

O primeiro passo é enunciar o problema claramente. É extremamente importante que o enunciado seja conciso para evitar desentendimentos. Para este exemplo, o enunciados do problema é:

Calcule a distância em linha reta entre dois pontos num plano.

2. DESCRIÇÃO ENTRADA/SAÍDA

O segundo passo é descrever cuidadosamente a informação que é dada para resolver o problema e então identificar os valores a serem calculados. Estes itens representam a entrada e a saída para o problema e agregadamente podem ser chamados entrada/saída, ou I/O. Para muitos problemas, é útil usar um diagrama que mostra a entrada e a saída. Algumas vezes, este tipo de diagrama é chamado de “caixa preta” porque não estamos definindo para este ponto todos os passos para determinar a saída, mas estamos mostrando a informação que é usada para calcular a saída. Para este exemplo, poderíamos usar o diagrama na figura 1.1.

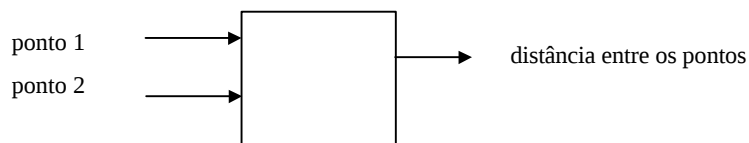


Figura 1.1 – Diagrama I/O

3. EXEMPLO MANUAL

O terceiro passo é trabalhar o problema manualmente ou com uma calculadora, usando um pequeno grupo de dados. É um passo muito importante e não deve ser ignorado por mais simples que seja o problema. É um item no qual você trabalha os detalhes da solução do problemas. Se você não pode pegar um simples grupo de números e calcular a saída (seja manualmente ou com uma calculadora), então você não está pronto para executar o próximo passo; você deve reler o problemas e talvez consultar material de referência. Uma vez que pode trabalhar o problema de um simples grupo de dados, então você está pronto para desenvolver um algoritmo ou um esboço passo a passo da solução. Este esboço é convertido para os comandos MATLAB para que possamos usar o computador para fazer todos os cálculos. O exemplo manual para o este exemplo é mostrado a seguir:

Suponha que os pontos p_1 e p_2 tenham as seguintes coordenadas:

$$p_1 = (1,5), \quad p_2 = (4,7)$$

Queremos calcular a distância entre dois pontos, que é a hipotenusa de um triângulo retângulo, conforme mostra a figura 1.2. Usando o Teorema de Pitágoras, podemos calcular a distância d com a seguinte equação:

$$\begin{aligned} d &= \sqrt{s_1^2 + s_2^2} \\ d &= \sqrt{(4-1)^2 + (7-5)^2} \\ d &= \sqrt{13} \\ d &= 3,61 \end{aligned}$$

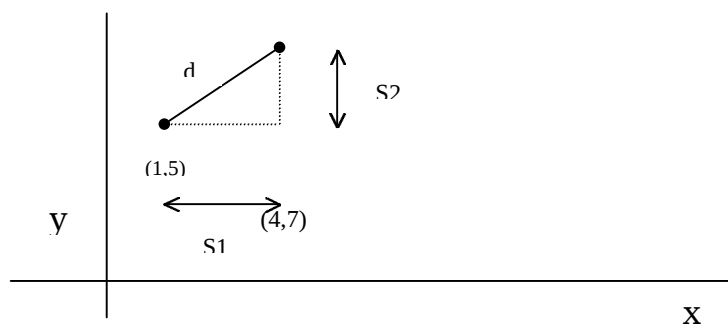


Figura 1.2 – Distância entre dois pontos.

4. SOLUÇÃO MATLAB

No próximo capítulo, falaremos sobre os comandos MATLAB. Contudo, da solução você pode ver que os comandos são muito similares às equações que foram usadas no exemplo manual. O sinal de porcentagem é usado para anteceder comentários que explicam os comandos MATLAB.

```
%  
%      Este programa calcula e imprime  
%      distância, em linha reta, entre dois pontos.  
p1 = [1,5];           % ponto 1 inicial  
p2 = [4,7];           % ponto2 inicial  
d = sqrt (sum ((p2-p1).^2)) % calcular distância
```

5. TESTANDO

O passo final em nosso processo de solução de problemas é testar a solução. Primeiramente, devemos testar a solução com os dados do exemplo manual, já que calculamos a solução. Quando os comandos MATLAB na solução são executados, o computador mostra a seguinte saída:

```
d = 3.6056
```

Esta saída coincide com o valor que calculamos no exemplo manual. Se a solução MATLAB não coincidir com o exemplo manual, devemos rever ambas soluções a fim de encontrar o erro. Uma vez que a solução trabalha com o exemplo manual, devemos também testá-la com vários grupos de dados para certificar que a solução é válida para outras séries de dados.

Capítulo 2 - Matrizes, Vetores e Escalares

A capacidade de visualização dos dados é um fator importante na solução de problemas de engenharia. Às vezes, o dado é um simples número como o raio de um círculo. Outras, um grupo de coordenadas x-y-z que representam os quatro vértices de uma pirâmides com uma base triangular no espaço. Podemos representar o exemplos citados usando um tipo especial de estrutura de dados denominada matriz. *Matriz* é uma tabela de números dispostos em m linhas e n colunas. Assim, um simples número pode ser considerado uma matriz com uma linha e uma coluna, uma coordenada x-y pode ser considerada uma matriz com uma linha e duas colunas, e um grupo de quatro coordenadas x-y-z pode ser considerada uma matriz com quatro linhas e três colunas. Como exemplo, temos:

$$A = [3.5] \quad B = [1.5 \ 3.1] \quad C = \begin{bmatrix} -1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Se uma matriz contiver m linhas e n colunas, então conterá um total de $m \cdot n$ elementos. Cada elemento da matriz é indicado por índices, a_{ij} . O primeiro, i, indica a linha, o segundo, j, indica a coluna onde o elemento se encontra. Assim, o elemento $a_{1,2}$ da matriz B é 3.1. Se o número de linhas e colunas forem iguais, então dizemos que a matriz é uma *matriz quadrada*. Se a matriz tiver apenas uma linha e uma coluna, podemos dizer que o valor é um *escalar*, se a matriz contiver apenas uma linha ou uma coluna, ao matriz é chamada vetor-linha ou vetor-coluna, respectivamente.

Exercícios

Responda às seguintes questões sobre esta matriz:

$$G = \begin{bmatrix} 0.6 & 1.5 & 2.3 & -0.5 \\ 8.2 & 0.5 & -0.1 & -2.0 \\ 5.7 & 8.2 & 9.0 & 1.5 \\ 0.5 & 0.5 & 2.4 & 0.5 \\ 1.2 & -2.3 & -4.5 & 0.5 \end{bmatrix}$$

1. Qual é a ordem de G?
2. G é uma matriz quadrada?
3. Dê as referências para todas as posições que contém o valor 0.5.
4. Dê as referências para todas as posições que contém valores negativos.

Definindo Matrizes no MATLAB

Suponha que queiramos agora criar as matrizes A, B e C usando o MATLAB. Há vários métodos de definição de matrizes no MATLAB. Vejamos cada um:

Modo mais simples:

Nome da matriz = [a_{11} a_{12} a_{13} ... a_{1n} ; a_{21} a_{22} a_{23} ... a_{2n} ; ... ; a_{m1} a_{m2} a_{m3} ... a_{mn}];

Assim, as matrizes A, B e C serão representadas por:

A = [3.5];

B = [1.5, 3.1];

C = [-1,0,0; 1,1,0; 1,-1,0; 0,0,2];

O nome da matriz deve começar com uma letra e conter no máximo 19 caracteres que podem ser números, letras ou caracter sublinhado, e aparece ao lado esquerdo do sinal de igual. O lado direito contém os dados entre colchetes por ordem de linhas. O ponto-e-vírgula separa as linhas, e os valores das linhas podem estar separados por vírgulas ou por espaços. O valor pode conter um sinal de + ou -, e um ponto decimal, mas não pode conter uma vírgula, como 32,154.

Quando definimos uma matriz, o MATLAB imprime o valor da matriz na próxima linha a menos que coloquemos um ponto-e-vírgula depois da definição. Tente entrar com as matrizes A, B e C sem o ponto-e-vírgula.

Você também pode definir uma matriz digitando uma cada linha separadamente. Como exemplo, a matriz C:

```
C = [ -1  0  0
      1  1  0
      1 -1  0
      0  0  2];
```

Se quisermos, por exemplo, definir um vetor-linha F com 10 valores, também podemos fazer:

```
F = [1 52 64 197 42 -42 55 82 22 109]
F = [1 52 64 197 42 -42, ...
     55 82 22 109]
```

Esta forma é muito usada quando a linha de uma matriz é extensa. Podemos terminar uma linha com uma vírgula seguida de três ou mais pontos, e continuar a entrar com os valores restantes na próxima linha da área de trabalho do MATLAB.

Podemos também definir uma matriz usando outra que já definida. Por exemplo, considere as seguintes matrizes:

```
B = [ 1.5 , 3.1];  
S = [3.0 B];
```

Estes comandos equivalem a:

```
S = [ 3.0 1.5 3.1];
```

Podemos também mudar e adicionar valores na matriz usando um referência entre parênteses. Assim, o seguinte comando;

```
S (2) = -1.0;
```

Muda o segundo valor da matriz S de 1.5 para -1.0.

A ordem da matriz pode ser alterada. Se executarmos o seguinte comando:

```
S(4) = 5.5
```

Então a matriz S terá quatro valores em vez de três. Se executarmos o comando:

```
S(8) = 9.5;
```

Então a matriz S terá 8 elementos, e os valores de S(5), S(6) e S(7) são automaticamente nulos, já que não foram atribuídos valores para eles.

Exercícios

Determine a ordem das matrizes a seguir. Verifique suas respostas usando o MATLAB.

1. $A = [1, 0, 0, 0, 0, 1];$

2. $B = [2; 4; 6; 10];$

3. $C = [5 \ 3 \ 5 ; 6 \ 2 \ -3];$

4. $D = [3 \ 4$

$5 \ 7$

$9 \ 10];$

5. $E = [3 \ 5 \ 10 \ 0; 0 \ 0 \ 0 \ 3; 3 \ 9 \ 9 \ 8];$

6. $T = [\ 4 \ 24 \ 9];$

$Q = [T \ 0 \ T];$

7. $X = [3 \ 6];$

8. $R = [C; X, 5];$

9. $V = [C(2,1) ; B];$

10. $A(2,1) = -3;$

As matrizes também podem ser definidas através de informação armazenada em arquivos. O MATLAB trabalha com dois tipos diferentes de arquivos: Os arquivos MAT e os arquivos ASCII.

Os arquivos MAT

Os arquivos MAT são gerados por um programa MATLAB usando o comando *save*, que contém o nome do arquivo e as matrizes que devem ser armazenadas. A extensão *.mat* é automaticamente adicionada ao nome do arquivo. Assim, para salvar matrizes A, B e C, em um arquivo *.mat* nomeado “teste_1” devemos fazer:

```
save teste_1 A B C;
```

Para recuperar as matrizes no programa MATLAB, usamos o comando:

```
load teste_1
```

Arquivos ASCII

Um arquivo ASCII que será usado juntamente com um programa MATLAB deve conter informação exclusivamente numérica, e cada linha do arquivo deve conter o mesmo número de dados. O arquivo pode ser gerado utilizando um processador de texto ou, por exemplo, utilizando programas como o Fortran ou ainda, por um programa MATLAB usando a seguinte forma do comando *save*:

```
save teste_1.dat R /ascii
```

Cada linha da matriz R será escrita para linhas distintas no arquivos de dados. Recomenda-se utilizar a extensão *.dat* para ser mais fácil distingui-los dos arquivos MAT e dos arquivos M.

O comando *load* seguido do nome do arquivo irá recuperar a informação da matriz R.

```
load teste_1.dat;
```

Operador Dois Pontos (:)

Suponha que queiramos armazenar a primeira coluna da matriz *data1* em um vetor *x*, e a segunda coluna em um vetor *y*. O uso do operador dois pontos (:) é útil na criação de matrizes ou vetores. Dependendo do argumento, pode significar todas as linhas ou todas as colunas da matriz-referência. Para o nosso exemplo, temos:

```
data1 = [0.0,0.0; 0.1 0.2; 0.3 0.6];  
x = data1 ( : , 1);  
y = data1 ( : , 2 );
```

Os elementos do vetor x correspondem à primeira coluna de data1. O segundo comando cria um vetor y cujos elementos correspondem à segunda coluna da matriz data1. Se quiséssemos criar um vetor z cujos elementos sejam os elementos da primeira linha da matriz data1, devemos fazer:

```
z = data1(1, : );
```

Se o operador dois pontos for usado na seguinte notação:

```
H = 1 : 8;
```

A matriz H contém os valores 1, 2, 3, 4, 5, 6, 7 e 8. O operador “ : ” entre os dois números inteiros gera todos os inteiros entre os dois números especificados. Se for usado para separar três números, os dois pontos gerarão valores entre o primeiro e terceiro números, usando o segundo número como incremento. A notação abaixo gera um vetor-linha denominado TEMPO que contém os números de 0.0 a 5.0 com incrementos de 0.5:

```
TEMPO = 0.0 : 0.5 : 5.0;
```

O incremento também pode ser um valor negativo como:

```
VALORES = 10 : -1: 0;
```

Os elementos de VALORES são 10, 9, 8, 7, 6, ... 0.

O operador dois pontos pode também ser usado para selecionar uma sub-matriz de uma outra matriz. Por exemplo, considere a matriz abaixo:

```
C = [-1,0,0;1,1,0; 1,-1,0; 0,0,2];
```

Se executarmos os comandos:

```
PARTE_1 = C ( : , 2:3);  
PARTE_2 = C (3:4, 1:2);
```

Definimos as matrizes:

```
PARTE_1 = [ 0 0; 1 0; -1 0; 0 2];  
PARTE_2 = [1 -1; 0 0];
```

Observações:

- O MATLAB reconhece uma matriz ‘vazia’. Há várias maneiras de gerá-la. Como exemplo, temos:

$A = []$
 $B = 4:-1:5$

- A expressão $C(:, :)$ equivale a uma longa matriz coluna que contém a primeira coluna de C, seguida pela segunda coluna de c e assim por diante.

Exercícios

Determine as ordens e o conteúdo das matrizes abaixo. Use a matriz G como referência.

$$G = \begin{bmatrix} 0,6 & 1,5 & 2,3 & -0,5 \\ 8,2 & 0,5 & -0,1 & -2,0 \\ 5,7 & 8,2 & 9,0 & 1,5 \\ 0,5 & 0,5 & 2,4 & 0,5 \\ 1,2 & -2,3 & -4,5 & 0,5 \end{bmatrix}$$

Verifique suas respostas usando o MATLAB.

1. $A = G(:, 2);$
2. $B = G(4, :);$
3. $C = [10 : 15];$
4. $D = [4:9; 1:6];$
5. $E = [-5,5];$
6. $F = [0.0:0.1:1.0];$
7. $T1 = G(4 : 5, 1:3);$
8. $T2 = G(1: 2 : 5, :);$

Solução:

Comando Input

Você pode entrar com os valores da matriz, via teclado, utilizando o comando *input* que mostra um texto e então espera por uma entrada. Considere o comando:

```
z = input ( 'Valores de z: ');
```

Quando este comando é executado, o texto “Valores de z: ” é mostrado na tela. O usuário pode entrar com uma expressão como [5.1 6.3 -18.0] o qual especifica valores para z. Já que o comando *input* termina com um ponto-e-vírgula, os valores de z não são imprimidos quando o comando é executado.

Imprimindo Matrizes

O modo mais simples de imprimir uma matriz é entrar com seu nome. O nome da matriz é repetido, os valores da matriz serão imprimidos na próxima linha. Existem vários comandos que podem ser usados para alterar a saída a ser imprimida.

Comando *format*

Suponha os comandos abaixo:

» a = [1 2 3];	» T = [1.1 2.4 3.7];
» c = 2*a	» U = 2*T
c =	U =
2 4 6	2.2000 4.8000 7.4000

Por definição, se o elemento de uma matriz for um número inteiro, o MATLAB apresenta o resultado como número inteiro. Se o elemento for um número real, o MATLAB apresenta-o com cinco dígitos significativos, ou seja, quatro dígitos à direita do ponto decimal. Podemos alterar o formato numérico utilizando o comando *format*.

Exemplo: Seja uma variável A que armazene a raiz quadrada de 2.

```
» A = sqrt(2)
```

De acordo com o formato numérico escolhido, a variável A pode estar apresentada sob a forma:

Comando MATLAB	Variável A	Descrição
Format long	1.41421356237310	16 dígitos
Format short	1.4142	5 dígitos – formato numérico padrão
Format short e	1.4142e+000	5 dígitos - notação científica
Format long e	1.414213562373095e+000	16 dígitos – notação científica
format +	+	“+” para valores positivos e “-” para valores negativos
format rat	1393/985	aproximação racional
format hex	3ff6a09e667f3bcd	formato hexadecimal

Comando *disp*

Quando quisermos exibir o conteúdo de uma matriz sem imprimir seu nome ou imprimir um pequeno texto, usamos o comando *disp*. Assim, se a variável *temp* contiver um valor de temperatura em graus Celsius, podemos imprimir o valor em uma linha de comando e a unidade na linha posterior:

```
disp(temp); disp ('graus Celsius')
```

Se o valor de *temp* for 78, então a saída será:

```
78 graus Celsius
```

Comando *fprintf*

O comando *fprintf* nos permite imprimir textos e conteúdo de matrizes. Podemos também especificar o formato numérico. Sua forma geral é:

```
fprintf (formato, matriz)
```

O modo formato contém o texto e as especificações que são:

- % e indica que os valores da matriz serão impressos em notação exponencial
- % f indica que os valores da matriz serão impressos em notação decimal ou em notação fixa, isto é, o usuário pode especificar o número de algarismos significativos juntamente com o ponto decimal.
- % g pode indicar as duas formas acima, dependendo de qual delas será a mais curta.

O modo matriz denota a variável cuja matriz está armazenada.

Um simples exemplo de aplicação do comando *fprintf* é mostrado abaixo:

```
fprintf ('A temperatura é %f graus Celsius \n', temp)
```

A saída seria:

A temperatura é 78.0000 graus Celsius

Se modificarmos o comando para esta forma:

```
fprintf ('A temperatura é \n %f graus Celsius \n', temp)
```

Então, a saída seria:

A temperatura é
78.0000 graus Celsius

Os formatos específicos %f, %e, e %g também podem conter informação para especificar o número de casas decimais a imprimir e o número de algarismos significativos, juntamente com o ponto decimal, conforme explicado no início da seção. Considere o seguinte comando:

```
fprintf ('A temperatura é %4.1f graus Celsius \n', temp)
```

A saída mostrará o valor de *temp* com 4 algarismos, sendo que um destes será um ponto decimal, conforme mostramos abaixo:

A temperatura é 78.0 graus Celsius

Gráficos X-Y

Suponhamos que queremos plotar os valores de uma matriz em vez de imprimi-los. Podemos usar o MATLAB para plotar gráficos. Nesta seção, mostraremos como gerar um simples gráfico x-y de dados armazenados em dois vetores. Então, sem conhecer alguns comandos, você pode imediatamente começar usando o MATLAB para gerar gráficos.

Suponha que queira plotar os dados de temperatura a seguir coletados em uma experiência de física:

Tempo, s	Temperatura, °C
0	54.2
1	58.5
2	63.8
3	64.2
4	67.3
5	71.5
6	88.5
7	90.1
8	90.6
9	89.5
10	90.4

Suponha também que os dados relativos ao tempo estejam armazenados em um vetor denominado *x*, e que os relativos à temperatura estejam armazenados em um vetor denominado *y*. Para plotar estes pontos, simplesmente usamos o comando *plot*, onde *x* e *y* são vetores-linha ou vetores-coluna.

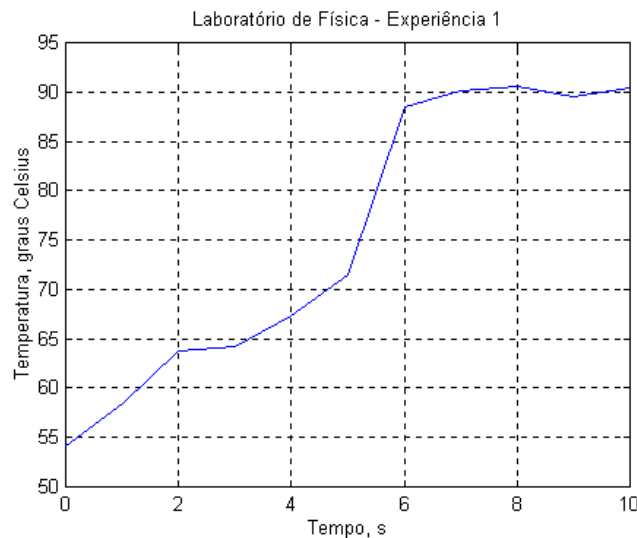
`plot (x, y)`

O gráfico é gerado automaticamente. A prática mostra que um bom gráfico deve incluir unidades, título e uma breve descrição. Logo, podemos aperfeiçoá-lo como os seguintes comandos:

Title	Adiciona um título ao gráfico.
Xlabel	Inclui uma descrição na direção do eixo-x
Y label	Inclui uma descrição na direção do eixo-y
Grid	Adiciona linhas de grade ao gráfico
Whitebg	Muda a cor de fundo do gráfico para branco.

Assim,

```
plot (x,y), ...  
title ('Laboratório de Física - Experiência 1'), ...  
xlabel ('Tempo, s'), ...  
ylabel ('Temperatura, graus Celsius'), ...  
grid  
whitebg
```



Os três pontos usados depois dos quatro comandos são usados para que o MATLAB execute os seis comandos em uma única vez. Para aprender mais opções para gerar gráficos x-y e outros tipos de gráficos, veja o capítulo 7.

Aplicação à Solução de Problemas: Análise de Dados de um Túnel de Vento

Um túnel de vento é uma câmara de teste construída para produzir diferentes velocidades de vento, ou números Mach (razão entre a velocidade do vento e a velocidade do som). Modelos em escala precisa de aeronaves podem ser equipados sobre suportes de medições de força na câmara de teste, e as medidas das forças sobre o modelo podem ser feitas para diferentes velocidades de vento e ângulos do modelo relativo à direção da velocidade. Ao final de um longo teste de túnel de vento, muitos grupos de dados são coletados e podem ser usados para determinar o lift, drag e outras características da performance aerodinâmica do novo modelo para várias velocidades de operação e posições.

Usamos esta aplicação várias vezes em nossos problemas ao longo do texto. Nesta seção, supomos que os dados coletados do teste do túnel de vento foram armazenados em um arquivo ASCII denominado vento1.dat. Gostaríamos de visualizar o gráfico dos dados para verificar se os sensores sobre o modelo em escala parecem trabalhar adequadamente. Suponhamos que cada linha do arquivo contém um ângulo de vôo em graus e um correspondente coeficiente de lift. Para este exemplo, usamos os seguintes dados:

Ângulo de Vôo (graus)	-4	-2	0	2	4	6	8	10	12	14	15
Coeficiente de Sustentação	-0,202	-0,050	0,108	0,264	0,421	0,573	0,727	0,880	1,027	1,150	1,195

Ângulo de Vôo (graus)	17	18	19	20	21
Coeficiente de Sustentação	1,225	1,250	1,245	1,221	1,177

Mesmo que pareça simples ler e plotar os dados usando o MATLAB, usaremos a metodologia descrita no capítulo anterior para mostrar é igualmente simples o processo que nos permite estruturar nossas idéias no desenvolvimento na solução de problemas.

1. ENUNCIADO DO PROBLEMA

Gerar um gráfico do ângulo de vôo e coeficiente de lift.

2. DESCRIÇÃO ENTRADA/SAÍDA

Sempre que for possível, usaremos um diagrama I/O, conforme mostrado na figura a seguir. Neste exemplo, lemos as informações contidas em um arquivo e usamos o MATLAB para plotá-las. O diagrama contém um símbolo de um disquete para representar o arquivo que é a entrada (observe que colocamos o nome do arquivo abaixo do símbolo) e um símbolo de um gráfico para representar a saída, que é o gráfico dos dados.

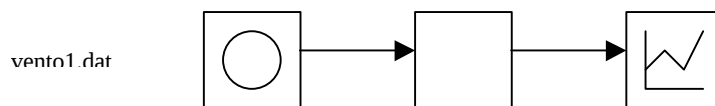


Diagrama I/O

3. EXEMPLO MANUAL

Apesar de ser apenas um gráfico, devemos estudar superficialmente uma pequena parte dos dados e determinar, grosseiramente, como seria o gráfico correspondente. Neste exemplo, se examinarmos os dados podemos perceber que inicialmente o coeficiente de lift é $-0,2$ e que o mesmo cresce até alcançar um máximo de $1,25$ para um ângulo de 18° . Se o gráfico que obtivermos for muito diferente do que esperávamos (por exemplo, valor inicial $0,7$ e um máximo de $1,177$ para um ângulo de 21 graus), então devemos novamente checar os dados e os comandos MATLAB usados.

4. SOLUÇÃO MATLAB

5. TESTANDO

Capítulo 3 - Cálculo Fundamentais e Matrizes Especiais

As operações de adição, subtração, multiplicação e divisão são a maioria das operações fundamentais usadas por engenheiros e cientistas. Podemos executar outras operações de rotina, como o cálculo da raiz quadrada ou o logaritmo de um valor ou a tangente de um ângulo. Estas operações podem ser executadas sobre um valor simples (um escalar), aplicadas a uma lista de valores (vetor), ou aplicadas a um grupo de valores armazenados em uma matriz. Neste capítulo aprenderemos como executar todas estas operações e funções. E também, aprenderemos como usar números complexos no MATLAB.

3.1 Valores Especiais e Matrizes Especiais

O MATLAB contém um grupo de constantes pré-definidas, valores e matrizes especiais úteis para uso em programas do MATLAB.

- Valores Especiais

π	pi	O valor de π é automaticamente armazenado nesta variável.
$\sqrt{-1}$	i,j	Estas variáveis são inicialmente agrupadas ao valor $\sqrt{-1}$. Veja a seção 3.5 para uma discussão completa sobre números complexos.
∞	inf	Esta variável é a representação do MATLAB para infinito, o qual ocorre tipicamente como o resultado de uma divisão por zero. Uma mensagem de aviso é imprimida, se você mostrar o resultado da divisão, o valor será ∞ .
Not-a-number	NaN	Ocorre em grande parte quando a expressão é indefinida, como a divisão de zero por zero.
	clock	Exibe a hora atual em um vetor linha de seis elementos contendo ano, mês, dia, hora, minute e segundos.
	date	Exibe a data atual como por exemplo, 20-Jun-92.
	ans	Variável usada para armazenar valores calculados por uma expressão que é calculada mas não armazenada em uma variável nomeada.

- Matrizes Especiais

O MATLAB contém um grupo de funções que geram matrizes especiais. Algumas destas matrizes tem aplicação específica às técnicas numéricas discutidas posteriormente.

Magic Square

Uma matriz *magic square* de ordem n é uma matriz $n \times n$ constituída de números inteiros de 1 a n^2 . Os elementos a_{ij} da matriz estão dispostos de forma tal que o somatório de cada linha é igual ao somatório de uma coluna.

Forma Geral: `magic (n)` matriz *square magic* de ordem n .

Assim, para saber o quadrado mágico de ordem 3, o prompt do MATLAB deve apresentar:

`magic (3)`

Zeros

Esta função gera uma *matriz zero*, isto é, uma matriz cujos elemento a_{ij} são nulos.

Forma Geral: `zeros(n)` Gera uma matriz zero, quadrada, de ordem n .
`zeros(m,n)` Gera uma matriz zero de ordem $m \times n$.

Ones

A função ones gera uma matriz cujo valor dos elementos a_{ij} é unitário.

Argumento: `ones(n)` Gera uma matriz quadrada de ordem n .
`ones(m,n)` Gera uma matriz de ordem $m \times n$.

Eye

A matriz identidade pode ser gerada pelo MATLAB através da função *eye*. Uma matriz identidade é uma matriz escalar de qualquer ordem cujos elementos a_{ij} são iguais a 1 para $i = j$. Apresenta o mesmo formato que as funções anteriores. O formato “`eye(n)`” gera uma matriz identidade de ordem n . Já o formato “`eye (m,n)`” gera uma matriz de ordem $m \times n$.

Pascal

Cria uma matriz cujas diagonais lembram o triângulo de Pascal. Assim, se usarmos o comando `pascal(5)`, a seguinte matriz é gerada:

1	1	1	1	1
1	2	3	4	5
1	3	6	10	15
1	4	10	20	35
1	5	15	35	70

3.2 Operações entre Escalares

Cálculos aritméticos são identificados usando expressões. Uma expressão pode ser tão simples como uma constante, ou pode ter matrizes e constantes combinadas com operações aritméticas. Nesta seção, discutiremos operações envolvendo somente escalares. Na seção posterior, estendemos as operações incluindo operações elemento por elemento entre escalares e matrizes ou entre duas matrizes.

As operações aritméticas entre dois escalares são mostradas na tabela 3.1. Uma expressão pode ser resolvida e armazenada em uma variável específica, como no comando seguinte, o qual especifica que os valores em *a* e *b* serão adicionados, e a soma armazenada em uma variável *x*:

```
x = a + b
```

Este comando deve ser interpretado como o valor em *b* adicionado ao valor em *a*, e a soma é armazenado em *x*. Se nós interpretamos os comandos desta forma, então nós preocupamos pelo seguinte comando MATLAB válido.

```
count = count + 1
```

É óbvio que esta instrução não é um comando algébrico válido, mas o MATLAB explica que 1 é adicionado ao valor em *count*, e o resultado será armazenado nesta variável. Ou seja, o valor em *count* será acrescido de 1 (ou incrementado por 1).

É importante reconhecer que uma variável pode armazenar somente um valor por vez. Por exemplo, suponha que as seguintes instruções serão executadas seguidamente;

```
Time = 0.0  
Time = 5.0
```

O valor 0.0 é armazenado na variável *time* quando a primeira instrução é executado e então substituído pelo valor 5.0 quando a segunda instrução é executada.

Quando você entra com uma expressão sem especificar uma variável para armazenar o resultado, o mesmo é automaticamente armazenado em uma variável denominada *ans*. Cada vez que um novo valor é armazenado em *ans*, o valor anterior é perdido.

Tabela 3.1 – Operações aritméticas entre dois escalares

Operação	Forma Algébrica	MATLAB
Adição	$a + b$	$a + b$
Subtração	$a - b$	$a - b$
Multiplificação	$a \times b$	$a*b$
Divisão Direita	$\frac{a}{b}$	a/b
Divisão Esquerda	$\frac{b}{a}$	$a \backslash b$
Exponenciação	a^b	a^b

Hierarquia em Operações Aritméticas

Sabendo que várias operações pode ser combinadas em uma simples expressão aritmética, é importante conhecer a ordem nas quais as operações serão executadas. A tabela 3.2 contém a ordem de prioridade das operações aritméticas no MATLAB. Note que esta prioridade também segue a prioridade algébrica padrão.

Tabela 3.2 Hierarquia em operações aritméticas

Prioridade	Operação
1	Parênteses
2	Exponenciação, esquerda à direita
3	Multiplificação e Divisão, esquerda à direita
4	Adição e Subtração, esquerda à direita

Suponha que queremos calcular a área de um trapézio, e também suponha que a variável *base* contenha o comprimento da base e que *altura_1* e *altura_2* contenham as duas alturas. A área de um trapézio pode ser calculada usando o seguinte enunciado:

$$\text{area} = 0.5 * h * (B + b);$$

Suponha que omitamos os parênteses:

$$\text{area} = 0.5 * \text{altura} * B + b;$$

Este enunciado seria executado como se fosse o enunciado a seguir:

$$\text{area} = (0.5 * \text{altura} * B) + b;$$

Note que embora a resposta incorreta tenha sido calculada, não há mensagens de erro imprimidas alertando-nos quanto ao erro. Portanto, é importante estar cauteloso quando convertamos equações para comandos do MATLAB. Adicionar parênteses extras é uma maneira

fácil para ter certeza que os cálculos são feitos na ordem que você quer. Se uma expressão é longa, divida-a em várias expressões. Por exemplo, considere a seguinte equação:

$$f = \frac{x^3 - 2x^2 + 6,3}{x^2 + 0,5005x - 3,14}$$

O valor de f poderia ser calculado usando os seguintes comandos, onde x é um escalar:

```
numerador = x^3 - 2*x^2 + x + 6.3  
denominador = x^2 + 0.5005*x - 3.14  
f = numerador/ denominador
```

É melhor usar várias equações que são mais fáceis de compreender que apenas uma, que requer maior cuidado na hora de imaginar a ordem das operações.

Exercícios

Dê os seguintes comandos do MATLAB para calcular os seguintes valores. Suponha que as variáveis nas equações são escalares e tenham valores determinados.

1. Coeficiente de fricção entre um pneu e o pavimento:

$$\text{Fricção} = \frac{v^2}{30s}$$

2. Fator de correção em cálculo de pressão:

$$\text{fator} = 1 + \frac{b}{v} + \frac{c}{v^2}$$

3. Distância entre dois pontos:

$$\text{Slope} = \frac{y^2 - y^1}{x^2 - x^1}$$

4. Resistência de um circuito paralelo:

$$\text{resistência} = \frac{1}{\frac{1}{r_1} + \frac{1}{r_2} + \frac{1}{r_3}}$$

5. Perda de pressão de um cano de fricção

$$\text{perda} = f \cdot p \cdot \frac{1}{d} \cdot \frac{v^2}{2}$$

Limites Computacionais

Para a maioria dos computadores, a escala de valores estende-se de 10^{-308} a 10^{308} , o que deve ser suficiente para acomodar grande parte dos cálculos. Contudo, é possível obter resultados que estejam fora deste alcance, como mostramos a seguir:

Suponha que executamos os seguintes comandos:

```
x = 2e200;  
y = 1e200;  
z = x*y;
```

Como o alcance é de 10^{-308} a 10^{308} , então os valores de x e y estão dentro do limite estabelecido. Mas, o valor de z é $2e400$, e este valor ultrapassa o alcance. Este erro é chamado *overflow* porque o expoente do resultado de uma operação aritmética é demasiadamente alto para ser armazenado na memória do computador. No MATLAB, o resultado de um expoente *overflow* é infinito(∞).

Suponha agora que executamos os seguintes comandos:

```
x = 2.5e-200;  
y = 1e200;  
z = x/y;
```

O erro de *underflow* é um erro similar causado pelo expoente do resultado de uma operação aritmética ser pequeno demais para ser armazenado na memória do computador. Os valores de x e y novamente estão dentro do alcance permitido, mas o valor de z deve ser $2.5e-400$. Se o expoente é menor que o mínimo, causamos um erro de *underflow*. No MATLAB, o resultado de *underflow* é zero.

Sabemos que a divisão por zero é uma operação inválida. Se uma expressão resulta em uma divisão por zero no MATLAB, o resultado da divisão é ∞ . O MATLAB imprimirá uma mensagem de aviso e logo a seguir o cálculo continua. As operações posteriores usam como ∞ resultado da divisão.

3.3 Operações de Conjuntos

Uma operação de conjunto é uma operação elemento por elemento. Por exemplo, suponha que A e B sejam vetores-linha com cinco elementos. Um modo de gerar um novo vetor C com valores que sejam produtos dos valores correspondentes em A e B é o seguinte:

```
C(1) = A(1)*B(1);  
C(2) = A(2)*B(2);  
C(3) = A(3)*B(3);  
C(4) = A(4)*B(4);  
C(5) = A(5)*B(5);
```

Estes comandos são essencialmente comandos escalares porque cada comando multiplica um simples valor por um outro e armazena o produto em um terceiro valor. Para indicar que executamos uma multiplicação elemento por elemento entre duas matrizes de mesma ordem, usamos um ponto antes da operação. Assim, os cinco comandos acima podem ser substituídos pelo seguinte:

$$C = A .* B;$$

Se omitirmos o ponto estaremos executando uma operação matricial. Operações matriciais é o tema que será discutido no capítulo 6.

Para as operações de adição e subtração, as operações de conjunto e matriciais são idênticas, e então não precisamos distingui-las. Contudo, as operações de conjunto para multiplicação, divisão e exponenciação são diferentes das operações matriciais para multiplicação, divisão e exponenciação e por isso devemos usar o ponto quando queremos especificar uma operação de conjunto.

Uma operação elemento por elemento, ou operações de conjuntos, aplicam-se não somente para operações entre duas matrizes de mesma ordem como também em operações entre um escalar e um não escalar. Contudo, a multiplicação de uma matriz por um escalar e a divisão esquerda de uma matriz por um escalar podem ser escritas de modo ou de outro. Assim, os dois comandos em cada grupo de comandos abaixo são equivalentes para uma matriz não escalar A.

```
B = 3*A;  
B = 3.*A;
```

```
C = A/5;  
C = A ./5;
```

As matrizes resultantes B e C terão a mesma ordem de A.

Para mostrar as operações de conjunto para vetores, considere os seguintes vetores-linha:

```
A = [2 5 6]  
B = [2 3 5]
```

Se calculamos o produto elemento a elemento de A e B usando o seguinte enunciado:

$$C = A .* B$$

Então, C conterá os seguintes valores:

$$C = [4 \ 15 \ 30]$$

O MATLAB tem dois operadores de divisão – uma divisão que usa o símbolo “/” e outra que usa o símbolo “\”. O comando para divisão *direita*:

$$C = A./B;$$

Irá gerar um novo vetor no qual cada elemento de A é dividido pelo elemento correspondente de B. Assim, C conterá os seguintes valores:

$$C = [1 \ 1.667 \ 1.2]$$

O comando para divisão *esquerda*:

$$C = A.\backslash B$$

Irá gerar um novo vetor no qual cada elemento é o elemento correspondente de B dividido pelo elemento correspondente de A. Então, C conterá os seguintes valores:

$$C = [1 \ 0.6 \ 0.833]$$

A exponenciação de conjunto também é uma operação elemento por elemento. Por exemplo, usamos os mesmos valores para A e B, considere os comandos:

$$\begin{aligned} C &= A.^2; \\ D &= A.^B; \end{aligned}$$

Os vetores C e D serão os seguintes:

$$\begin{aligned} C &= [4 \ 25 \ 36] \\ D &= [4 \ 125 \ 7776] \end{aligned}$$

A operação também é válida para uma base escalar e um expoente vetor, como o exemplo a seguir:

$$C = 3.0.^A;$$

que gera um vetor com os seguintes valores;

$$C = [9 \ 243 \ 729]$$

Este vetor poderia também ser calculado com a seguinte instrução:

$$C = (3).^A;$$

Contudo, a instrução a seguir é incorreta:

```
C = 3.^A;
```

O MATLAB supõe que o ponto é parte da constante 3, e então fazer uma exponenciação matricial, que discutiremos no capítulo 6. Se inserirmos um espaço antes do ponto, como se segue:

```
C = 3 .^A;
```

Então, o comando tentaria fazer a exponenciação elemento por elemento conforme desejávamos. Estes exemplos indicam que devemos ter cuidado quando especificarmos operações de conjuntos. Se não tiver certeza que o que escreveu é a expressão correta, sempre teste-a com simples exemplos como aqueles que usamos.

```
d = [1:5; -1: -1: -5];  
z = ones(2,5)  
s = d - z  
p = d.*s  
sq = d.^3;
```

Os valores destas matrizes são mostrados a seguir:

$$d = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ -1 & -2 & -3 & -4 & -5 \end{bmatrix} \quad z = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$s = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ -2 & -3 & -4 & -5 & -6 \end{bmatrix} \quad p = \begin{bmatrix} 0 & 2 & 6 & 12 & 20 \\ 2 & 6 & 12 & 20 & 30 \end{bmatrix}$$

$$sq = \begin{bmatrix} 1 & 8 & 27 & 64 & 125 \\ -1 & -8 & -27 & -64 & -125 \end{bmatrix}$$

Exercícios

Dê os valores no vetor C depois execute os seguintes enunciados, onde A e B contêm os valores mostrados. Cheque suas respostas usando o MATLAB.

$$A = [2 \ -1 \ 5 \ 0] \quad B = [3 \ 2 \ -1 \ 4]$$

1. $C = A - B$;
2. $C = B + A - 3$;
3. $C = 2*A + A.^B$;

4. $C = B ./ A;$
5. $C = B . \backslash A;$
6. $C = A .^B;$
7. $C = (2) .^B + A;$
8. $C = 2 * B / 3.0 * A;$

Solução de Problemas Aplicados à Engenharia: Ecos em Sinais de Comunicação

Uma interessante pesquisa está sendo feita atualmente para desenvolver sistemas de computadores que respondam a comandos verbais. O projeto do tal sistema supõe que o microfone colhe o comando de voz e tem uma representação nítida da fala. Infelizmente, sensores como os microfones apresentam distorções, denominadas ruído. Os sistemas com comunicações duas vias também raramente tem ecos que são inadvertidamente introduzidos pela instrumentação. Por essa razão, um sistema reconhecedor de voz deve ser capaz de executar algum processamento do sinal de voz para remover algumas das distorções e componentes indesejáveis, tal como os ecos, tentando antes reconhecer as palavras. Como forma de testar um programa que foi projetado para remover ecos, devemos estar aptos a gerar um sinal digital e adicionar ecos ao mesmo. Podemos então avaliar a performance do programa que é suposta para remover os ecos. Nesta seção, definimos sinais digitais, e então desenvolveremos um programa MATLAB para adicionar ecos a um sinal digital.

Sinais Digitais

Um sinal é uma função (normalmente em relação ao tempo) que representa informação. Esta informação ou dado são coletados com um sensor. Alguns dos mais comuns tipos de sensores são microfones, que medem acústica ou dados sonoros (como a fala); sismômetro, que mede intensidade de tremor de terra; fotocélulas, que medem a intensidade da luz; termistores, o qual medem a temperatura; e osciloscópios, que medem tensões. Os sensores são normalmente conectados à outra peça da instrumentação chamada conversor analógico-digital (A/D), que amostra o sinal periodicamente e grava o tempo e os valores do sinal que possam ser armazenados em um arquivo de dados. O sinal original é normalmente uma função contínua (ou analógica); a sequência de valores coletados do sinal original é denominada sinal digital. A figura 3.1 contém um exemplo de sinal analógico coletado de um sinal contínuo. O sinal analógico é composto de um grupo de coordenadas x-y e assim poderiam facilmente ser armazenadas em um arquivo de dados, e então ler um programa MATLAB. Quando plotamos um sinal analógico, geralmente ligamos os pontos com segmentos de reta em vez de plotar apenas os pontos.

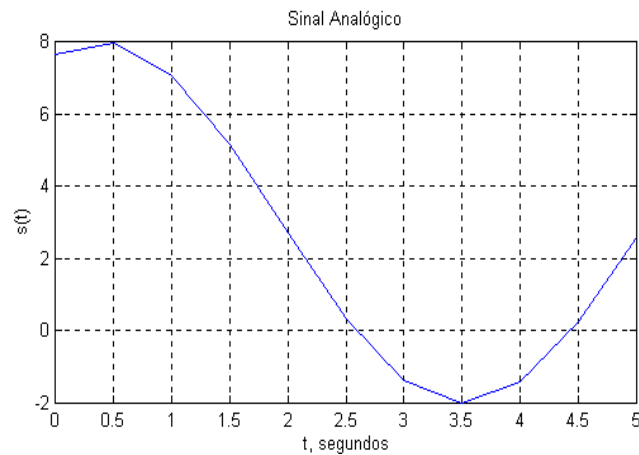


Figura 3,1 – Sinal Analógico ou contínuo

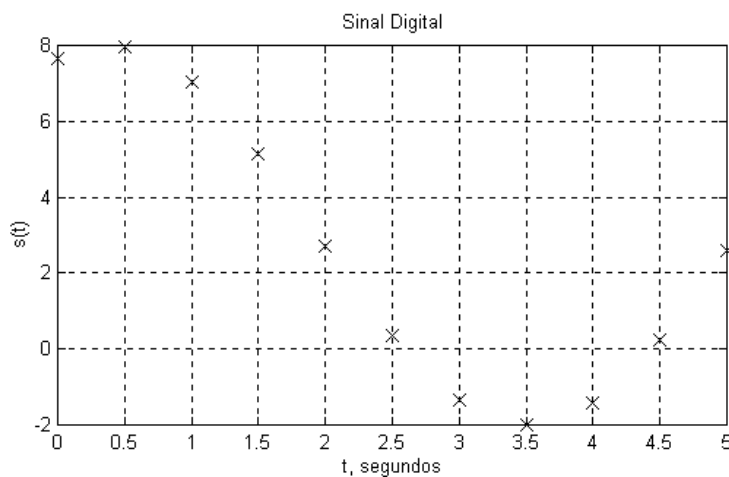


Figura 3..2 – Amostra ou Sinal Digital

Gerando ecos em um sinal

Um eco de um sinal é representado por uma versão atenuada do sinal original e que ocorre atrasado no tempo em relação ao sinal original. Por exemplo, a figura 3.3 contém um sinal original $s(t)$ no primeiro esquema. O segundo esquema contém um eco do sinal original que foi atenuado aproximadamente 50% (ou 0,5) do sinal original. O terceiro esquema contém um eco do sinal original atenuado em 30% e atrasado 5 segundos em relação ao sinal original; este é um ROLLED eco porque os valores do eco são negativos do eco esperado. O quarto esquema contém o sinal original mais os dois ecos adicionados ao mesmo.

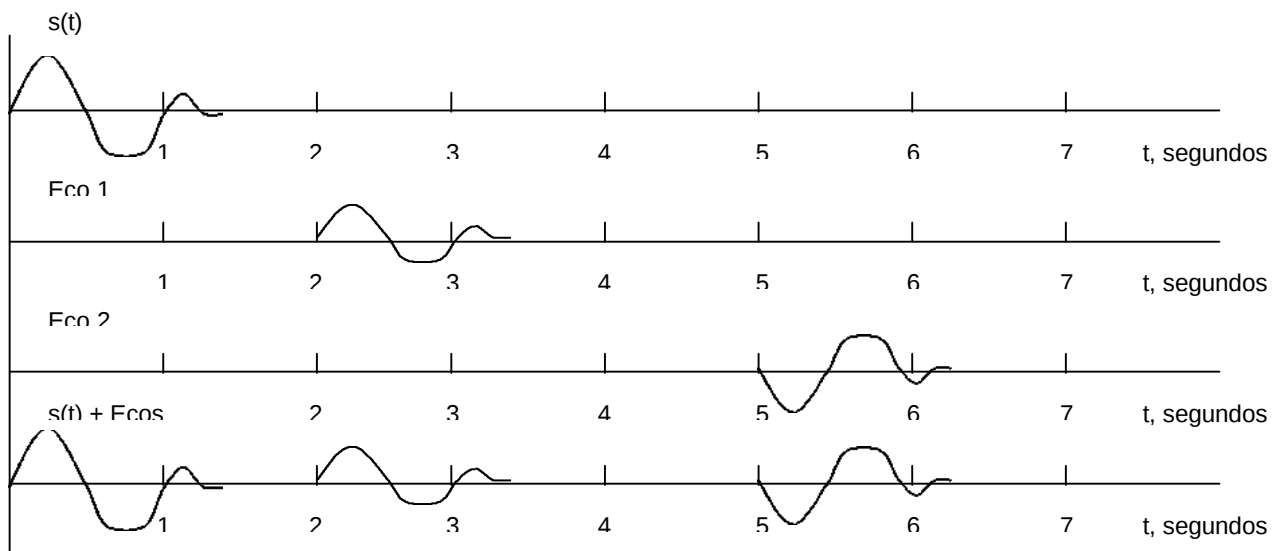


Figura 3.3 – O sinal original e os ecos.

Suponha que um sinal original foi coletado um período de 10 segundos, com um intervalo de tempo de amostragem de 0,1 segundos. O seguinte grupo de coordenadas foram coletados no primeiro segundo, e todos os valores do sinal depois estes valores foram zerados:

Tempo(s)	0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1,0
Valor do sinal	0,0	0,5	1,0	1,5	2,0	2,5	3,0	3,5	4,0	4,5	5,0

Escreva um programa do MATLAB que gera um sinal que contém o sinal original com três ecos adicionados ao mesmo. O primeiro eco é atenuado em 0,5 e atrasado em 2 segundos; o segundo eco tem um tempo de atraso de 4 segundos e atenuado em 0,3 segundos; o terceiro eco é atrasado em 7,5 segundos e atenuado em 0,1. Plote o gráfico do sinal original e o sinal com ecos em um arquivo MAT denominado eco.mat.

1. ENUNCIADO DO PROBLEMA

Dado um sinal original, gerar um novo sinal contendo o sinal original mais três ecos específicos adicionados a ele.

2. DESCRIÇÃO ENTRADA/SAÍDA

O retângulo tracejado contém uma figura detalhada do processo de geração de ecos do sinal de entrada [sn]. Este sinal é atrasado e multiplicado por um fator escalar (representado pelo triângulo) para gerar cada eco. Então, o sinal original e todos os ecos que são adicionados juntos em um novo sinal [gn], o qual é plotado e armazenado um arquivo de dados chamado eco.mat.

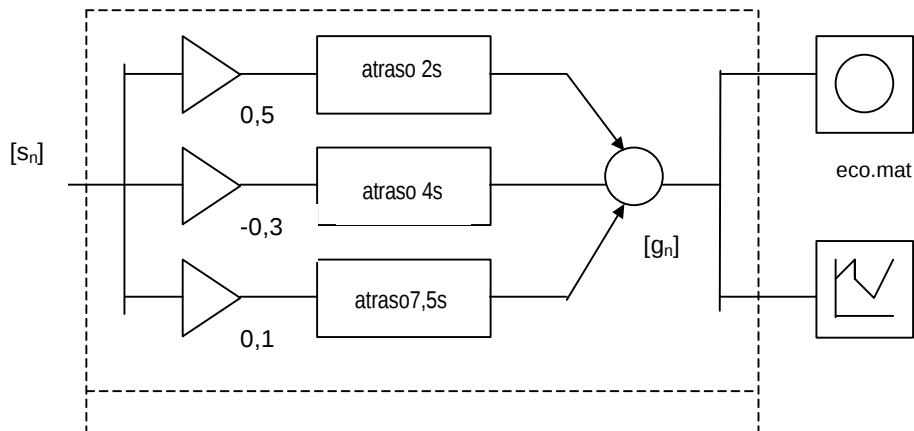


Figura 3.4 – Diagrama entrada / saída

3. EXEMPLO MANUAL

Para um exemplo manual, usamos os três primeiros valores do sinal original:

Tempo (s)	Valor do sinal
0,0	0,0
0,1	0,5
0,2	1,0

Os ecos específicos então tem os seguintes valores (não-nulos):

<i>Tempo,s</i>	<i>Valor do sinal</i>
2,0	$(0,5) \cdot (0,0) = 0,0$
2,1	$(0,5) \cdot (0,5) = 0,25$
2,2	$(0,5) \cdot (1,0) = 0,5$

<i>Tempo,s</i>	<i>Valor do sinal</i>
4,0	$(-0,3) \cdot (0,0) = 0,0$
4,1	$(-0,3) \cdot (0,5) = -0,15$
4,2	$(-0,3) \cdot (1,0) = -0,3$

<i>Tempo,s</i>	<i>Valor do sinal</i>
2,0	$(0,1) \cdot (0,0) = 0,0$
2,1	$(0,1) \cdot (0,5) = 0,05$
2,2	$(0,1) \cdot (1,0) = 0,1$

A soma do sinal original mais os três ecos são mostrados na figura 3.5.

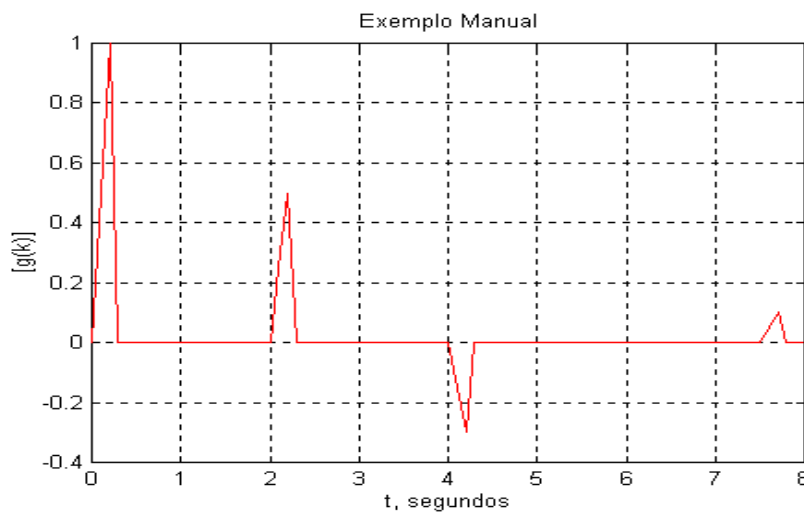


Figura 3.5 – Sinal Original mais os três ecos.

3.4 Funções Elementares

As expressões aritméticas raramente requerem outros cálculos que não sejam a adição, subtração, multiplicação, divisão, e exponenciação. Por exemplo, muitas expressões requerem o uso de logaritmos, exponenciais, e funções trigonométricas. O MATLAB nos permite usar funções para executar estes tipos de cálculos em vez de nos exigirem calculá-los usando operações aritméticas básicas. Por exemplo, se quisermos calcular o seno de um ângulo e armazenar o resultado em b, podemos usar o seguinte comando:

```
b = sin(angle);
```

A função *sin* supõe que o argumento está em radianos. Se o argumento contém um valor em graus, podemos convertê-lo de graus para radianos dentro da função referência:

```
b = sin (angle*pi/180);
```

Poderíamos também fazer a conversão em comandos separados:

```
angle_radians = angle*pi/180;  
b = sin(angle_radians);
```

Estes comandos são válidos se *angle* é um escalar ou se *angle* é uma matriz. Se *angle* for uma matriz, então a função será aplicada elemento por elemento aos valores na matriz.

Agora que já vimos vários exemplos de funções, iniciaremos uma revisão das regras relativa às funções. Uma função é uma referência que representa uma matriz. Os argumentos ou parâmetros da função estão contidos em parênteses seguindo do nome da função. Uma função pode não conter argumentos, um argumento ou muitos argumentos, dependendo de sua definição. Por exemplo, *pi* é uma função que não tem argumento; quando usamos a função referência *pi*, o valor

para pi automaticamente substitui a função referência. Se uma função contém mais que um argumentos, é muito importante dar os argumentos em ordem correta. Algumas funções também exigem que os argumentos estejam unidades específicas. Por exemplo, as funções trigonométricas supõem que os argumentos estão em radianos. No MATLAB, algumas funções usam o número de argumentos para determinar a saída da função. Por exemplo, a função zeros pode ter um ou dois argumentos, pelos quais determinamos a saída.

Uma função referência não pode ser usada ao lado esquerdo de um sinal de igualdade, desde que este represente um valor e não uma variável. Funções podem aparecer à direita de um sinal de igualdade e em expressões. Uma função de referência pode também ser parte do argumento de uma outra função de referência. Por exemplo, o seguinte comando calcula o logaritmo do valor absoluto de x:

$$\log_x = \log(\text{abs}(x))$$

Quando uma função é usada para calcular o argumento de uma outra função, tenha certeza de fechar o argumento de cada função em seu próprio grupo de parênteses. Esta acomodação da função é também chamada composição de funções. Nomes de funções devem estar em letras minúsculas a menos que o “*case sensitivity*” esteja desativado.

Agora discutiremos várias categorias de funções que são freqüentemente usadas em cálculos de engenharia. Outras funções serão apresentadas no decorrer dos capítulos tão logo debatermos tópicos relevantes.

Funções Matemáticas Elementares

As funções matemáticas elementares incluem funções para executar um número de cálculos comuns como o cálculo de valor absoluto e a raiz quadrada. Além disso, também incluímos um grupo de funções usadas em arredondamentos. Mostraremos a seguir uma lista destas funções com uma breve descrição:

<code>abs(x)</code>	Calcula o valor absoluto de x.
<code>sqrt(x)</code>	Calcula a raiz quadrada de x.
<code>round(x)</code>	Arredonda o valor de x para o inteiro mais próximo.
<code>fix(x)</code>	Arredonda o valor de x para o inteiro mais próximo de zero.
<code>floor(x)</code>	Arredonda o valor de x para o inteiro mais próximo de $-\infty$
<code>ceil(x)</code>	Arredonda o valor de x para o inteiro mais próximo de ∞
<code>sign(x)</code>	Se x é menor que zero, a função retorna ao valor -1 ; se x for igual a zero, retorna ao valor zero; caso contrário, a função retorna ao valor 1.
<code>Rem(x,y)</code>	Retorna o resto da divisão x/y. Por exemplo, <code>rem(25,4)</code> é 1, e <code>rem(100,21)</code> é 16.
<code>Exp(x)</code>	Esta função retorna ao valor de e^x , onde e é a base para logaritmo natural

ou aproximadamente 2.718282.

<code>log(x)</code>	Retorna a $\ln x$, o logaritmo natural de x para a base e .
<code>Log10(x)</code>	Retorna a $\log_{10}x$, ou seja, o logaritmo de x na base 10.

Exercícios

Calcule as seguintes expressões, e então verifique sua resposta no MATLAB.

1. `round (-2.6)`
2. `fix(-2.6)`
3. `floor(-2.6)`
4. `ceil(-2.6)`
5. `sign(-2.6)`
6. `abs(round(-2.6))`
7. `sqrt(floor(10.7))`
8. `rem(15,2)`
9. `floor(ceil(10.8))`
10. `log10(100) + log10(0.001)`
11. `abs(-5.5)`
12. `round([0:0.3:2,1:0.74:4])`

Funções Trigonômicas

As funções trigonométricas supõem que os ângulos estejam representados em radianos. Para converter para graus ou de graus para radianos, use as seguintes conversões, sabendo que $180^\circ = \pi$ radianos:

$$\begin{aligned}\text{ângulo_graus} &= \text{ângulo_radianos} * (180/\pi); \\ \text{ângulo_radianos} &= \text{ângulo_graus} * (\pi/180);\end{aligned}$$

A seguir uma lista de funções trigonométricas com uma breve descrição:

<code>sin(x)</code>	Calcula o seno de x , em radianos.
<code>cos(x)</code>	Calcula o cosseno de x , em radianos.
<code>tan(x)</code>	Calcula a tangente de x , em radianos.
<code>asin(x)</code>	Calcula o arcoseno de x , onde x deve estar entre -1 e 1 . A função apresenta um ângulo em radianos entre $-\pi/2$ e $\pi/2$.
<code>acos(x)</code>	Calcula o arcocosseno de x , onde x deve estar entre -1 e 1 . A função apresenta um ângulo em radianos entre 0 e π .
<code>atan(x)</code>	Calcula o arcotangente de x , onde x deve estar entre -1 e 1 . A função apresenta um ângulo em radianos entre $-\pi/2$ e $\pi/2$.
<code>atan2(x,y)</code>	Calcula o arcotangente do valor de y/x . A função apresenta um ângulo em radianos estará entre $-\pi$ e π , dependendo dos sinais de x e y .

As outras funções trigonométricas podem ser calculados usando as seguintes equações:

$$\sec x = 1 / \cos x$$

$$\csc x = 1 / \sin x$$

$$\cot x = 1 / \tan x$$

Funções Hiperbólicas

Funções Hiperbólicas são funções de e^x ; as funções hiperbólicas inversas são funções de $\ln x$. Estas funções são úteis em aplicações como o projeto de alguns tipos de filtros digitais. O MATLAB inclui várias funções hiperbólicas, como as mostradas nesta breve descrição:

<code>sinh(x)</code>	Calcula o seno hiperbólico de x.
<code>cosh(x)</code>	Calcula o cosseno hiperbólico de x.
<code>tanh(x)</code>	Calcula a tangente hiperbólica de x.
<code>asinh(x)</code>	Calcula o seno hiperbólico inverso de x.
<code>acosh(x)</code>	Calcula o cosseno hiperbólico inverso de x.
<code>atanh(x)</code>	Calcula a tangente hiperbólica inversa de x.

Exercícios

Dê as expressões MATLAB para calcular os seguintes valores, dado o valor de x.

1. `coth x`
2. `sec x`
3. `acoth x`
4. `csc x`
5. `asech x`
6. `acsc x`

Funções de arquivo M

O MATLAB apresenta uma estrutura que nos permite criar funções sob a forma de arquivos M. Como exemplo, considere uma função que esteja em um arquivo-M denominado `circum.m`:

```
function c = circum ( r )
% CIRCUM Circunferência de um círculo de raio r.
% Para matrizes, CIRCUM ( r ) retorna uma matriz
% que contém as circunferências de círculos com raios iguais
% aos valores no vetor original.
c = pi*2*r;
```

Assim, se o prompt do MATLAB apresentar:

```
r = [0 1.4 pi];  
a = circum (r );
```

Os elementos da matriz A corresponderão as circunferências de círculos de raios 0, 1,4 e π , respectivamente.

Para esta função também são válidos os comandos:

```
a = 5.6;  
disp (circum(a))
```

```
c = [1.2 3; 5 2.3];  
circum ( c );
```

Assim, *circum* passa a ser uma função MATLAB assim como *ones*, *sin* e outras. A parte comentada no arquivo *circum.m* é usada quando digitarmos *help circum* no prompt do MATLAB.

Há algumas regras para escrever uma função de arquivo M:

- A função deve começar com uma linha contendo a palavra *function*, seguida pelo argumento de saída, um sinal de igual, e o nome da função. Os argumentos para a função devem estar entre parênteses. Esta linha define os argumentos de entrada e saída;
- As primeiras linhas devem ser comentários porque serão exibidas quando o menu *help* for usado juntamente com o nome da função , como *help circum*;
- A única informação retornada da função é contida nos argumentos de saída, que são, obviamente, matrizes. Verificar se a função inclui um comando que assegure um valor ao argumento de saída.
- Uma função que possui mais de uma variável de saída como por exemplo:
`function [ dist, vel, acel] = motion (x)`
Deve apresentar as variáveis de saída dentro de colchetes. Além disso, todos os valores devem ser calculados dentro da função.
- Uma função que tenha múltiplos argumentos de entrada deve listar os argumentos no comando *function*, como mostramos no exemplo a seguir, que tem dois argumentos de entrada:

```
function error = mse (w,d)
```

- As variáveis especiais *nargin* e *nargout* podem ser usadas para determinar o número de argumentos de entrada passadas para uma função e o número de argumentos de saída solicitados quando a função é chamada.

Solução de Problemas Aplicados à Engenharia: Sinais de Sonar

O estudo do sonar (*sound navigation and ranging*) inclui a geração, transmissão, e recepção de energia sonora na água. Dentre as aplicações destacamos: mapeamento geológico, medidas de sinal biológico, navegação submarina e exploração mineral. Um sistema sonar ativo transmite um sinal que é normalmente um sinal senoidal de frequência conhecida. As reflexões ou os ecos do sinal são recebidos e analisados para prover informações sobre o meio envolvente. Um sistema sonar passivo não transmite sinais mas coleta-os de sensores e os analisa baseado em suas frequências.

Nesta seção, descreveremos as senóides, já que é um sinal básico usando em sistemas sonar. Depois, desenvolveremos um programa MATLAB para gerar um sinal sonar.

Geração de um Sinal Senoidal

Uma senóide é uma função seno escrita em função do tempo:

$$g(t) = \text{sen}(2\pi ft)$$

onde f é a frequência da senóide em ciclos por segundo, ou Hertz(Hz).

Se a frequência de uma senóide for 5 HZ, teremos:

$$g(t) = \text{sen}(2\pi 5t) = \text{sen}(10\pi t)$$

Então haverá cinco ciclos da senóide em um segundo, ou seja, a frequência de uma senóide é 5 HZ. O período P de uma senóide é o intervalo de tempo que corresponde a um ciclo; portanto, o período desta senóide é 0,2 segundos. A relação entre período e frequência é dada por

$$f = 1/P$$

onde f é a frequência em Hz e P é o período em segundos.

Se a senóide é multiplicada por um escalar A , a equação pode ser escrita nesta forma:

$$g(t) = A \text{sen}(2\pi ft)$$

O escalar é também chamado de amplitude da senóide. Uma senóide com um ângulo de fase ϕ em radianos pode ser escrita como:

$$g(t) = A \text{sen}(2\pi ft + \phi)$$

Se o ângulo de fase for igual a $\pi/2$ radianos, a senóide poderá ser escrita sob termos de uma função seno ou cosseno, e pode ou não incluir um ângulo de fase.

Gerando um sinal sonar

Conforme já dito, um dos tipos de sinais usados em sistemas de sonar é um sinal senoidal. As senóides podem ser representadas pela equação:
onde :

$$s(t) = \begin{cases} \sqrt{\frac{2E}{PD}} \cos(2\pi f_c t), & 0 \leq t \leq PD \\ 0 & \text{para os demais instantes} \end{cases}$$

E é a energia transmitida,
PD é a duração do pulso em segundos,
 f_c é a frequência em Hertz.

Duração de um sinal sonar podem alcançar de uma fração de milisegundos a alguns segundos; e as frequências, de poucas centenas de Hz a dezenas de KHz dependendo do sistema e do alcance de operação desejado.

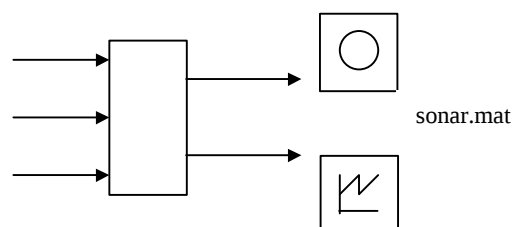
Escreva um programa MATLAB que permita ao usuário entrar com valores de E, PD, e f_c para gerar um sinal sonar. Armazene os valores do sinal em um arquivo MAT denominado sonar.mat. A amostragem do sinal deve cobrir a duração do pulso e conter 10 amostras de cada período de $x(t)$. Além disso, adicione um período de 200 pontos de silêncio depois do pulso.

1. ENUNCIADO DO PROBLEMA

Escreva um programa para gerar um sinal sonar que contenha 10 amostras de cada período de uma senóide específica, cobrindo uma duração de tempo determinada.

2. DESCRIÇÃO ENTRADA/SAÍDA

Os valores de E (energia transmitida em joules), PD (duração do pulso em segundos), e f_c (frequência em Hz) são os valores de entrada. A saída é um arquivo denominado sonar.mat, que contém os valores de tempo e sinal para a duração do pulso sonar, como mostramos na figura 3.7. Também plotamos o sinal sonar.



3. EXEMPLO MANUAL

Para um exemplo manual, usamos os valores a seguir:

$E = 500$ joules
 $PD = 5$ milisegundos (ms)
 $f_c = 3,5$ KHz

O período da senóide é $1/3500$, ou aproximadamente 0,3 ms. Assim, para ter 10 amostras por período, o intervalo da amostragem precisa ser aproximadamente 0,03 ms. A duração do pulso é 0,5 ms, e portanto precisamos de 167 amostras do sinal:

$$s(t) = \sqrt{\frac{2E}{PD}} \cos(2\pi f_c t)$$

$$s(t) = \sqrt{\frac{1000}{0,005}} \cos(2\pi(3500)t)$$

$$s(t) = 447,2 \cos(2\pi 3500t)$$

Os primeiros valores do sinal sonar são calculados com aproximação de uma casa decimal.

t (ms)	0,00	0,03	0,06	0,09	0,12	0,15	0,18	0,21	0,24	0,27	0,30	0,33
s(t)	447,2	353,4	111,2	-177,6	-391,9	-441,7	-306,1	-42,1	239,6	420,8	425,3	251,4

Adicionaríamos 200 pontos de silêncio através de dados adicionais com seus tempos correspondentes e valores de sinais.

4. SOLUÇÃO MATLAB

3.5 Números Complexos

As soluções de muitos problemas de engenharia como sistema de controle para um braço mecânico ou análise da estabilidade de um circuito elétrico envolvem a busca das raízes de uma equação da seguinte forma:

$$y = f(x)$$

onde as raízes são os valores de x para qual y é igual a zero.

Considere a forma geral para um polinômio de grau n :

$$a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + \dots + a_{n-1} x^2 + a_n x + a_{n+1} = 0$$

Um polinômio de grau n terá n raízes, sendo que algumas podem ser raízes múltiplas ou raízes complexas. Nesta seção discutiremos as operações com números complexos e as funções MATLAB que os usam.

Operações Aritméticas com Números Complexos

Os comandos MATLAB reconhecem os números complexos usando i para representar $\sqrt{-1}$. (O MATLAB também reconhece o uso de j para representar $\sqrt{-1}$. Esta notação é mais usada na Engenharia Elétrica). O comando a seguir define uma variável complexa:

```
x = 1 - 0.5*i;
```

Quando executamos operações entre dois complexos, o MATLAB automaticamente executa os cálculos necessários. Se uma operação for entre um número real e um complexo, o MATLAB supõe que a parte imaginária do número real é igual a zero. O MATLAB inclui várias funções que são específicas aos números complexos:

<code>real(x)</code>	Calcula a parte real do número complexo x .
<code>imag(x)</code>	Calcula a parte imaginária do número complexo x .
<code>conj(x)</code>	Calcula o conjugado do número complexo x .
<code>abs(x)</code>	Calcula o módulo do número complexo x .
<code>angle(x)</code>	Calcula o ângulo usando o valor de <code>atan2(imag(x), real(x))</code> , e portanto o ângulo está entre $-\pi$ e π .

Estas funções tornam mais fácil converter o complexo da forma polar para retangular.

Coordenadas polar e retangulares

Podemos representar um número complexo em um plano com eixos real e imaginário. Os números reais representam o eixo x , e os números imaginários representam o eixo y , e os números com partes real e imaginária representam o resto do plano.

Quando representamos um número complexo com uma parte real e imaginária, como $2 + i3$, estamos usando uma notação retangular. A figura 3.10 mostra que o número complexo pode ser escrito com um ângulo θ e raio r em relação à origem. Esta forma é chamada de notação polar, e o ponto $2 + i3$ pode ser representado em notação polar com um ângulo de 0,98 radianos e um raio 3,6.

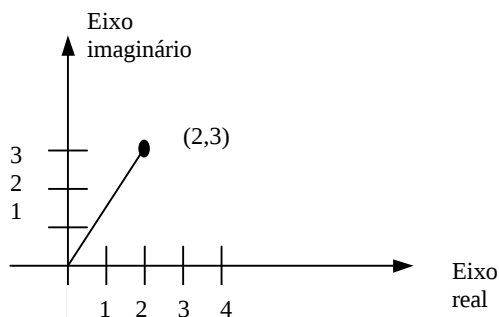


Figura 3.10 – Plano Complexo

Conversão

- retangular a polar

$$r = \sqrt{a^2 + b^2}$$

$$\theta = \tan^{-1} \frac{b}{a}$$

- polar a retangular

$$a = r \cos \theta$$

$$b = r \sin \theta$$

Se x é um número complexo, então o módulo e a fase podem ser calculados com os seguintes comandos:

```
r = abs (x);  
theta = angle (x);
```

Para calcular o número complexo usando módulo e fase determinados, usamos o comando:

```
y = r*exp (i*theta);
```

Podemos calcular a parte real e a parte imaginária de um complexo com os comandos:

```
a = real (x);  
b = imag (x);
```

Para calcular o complexo com partes real e imaginária específicas, usamos:

```
y = a + i*b;
```

Exercícios

Converter os números complexos nos problemas abaixo. Verifique suas respostas usando o MATLAB.

1. $3 - i2$
2. $-i$
3. -2
4. $0,5 + i$

Converter os valores abaixo para forma retangular. Verifique suas respostas usando as funções MATLAB.

5. e^i
6. $e^{ip0,75}$
7. $0,5 e^{i2,3}$
8. $3,5e^{i3p}$

Capítulo 4 - Controle de Fluxo

4.1 Operadores Lógicos e Relacionais

Operadores Relacionais

O MATLAB tem operadores relacionais que podem ser usados para comparar duas matrizes de mesma ordem ou para comparar uma matriz e um escalar, como os mostrados a seguir:

Operador	Descrição
<	Menor que
<=	Menor ou igual a
>	Maior que
>=	Maior ou igual a
==	Igual a (no sentido de condição)
~=	Não igual a

A finalidade dos operadores é fornecer respostas a perguntas do tipo falso/verdadeiro. Assim, se a comparação for verdadeira, atribui-se o valor 1; se for falsa, o valor 0.

Considere a expressão lógica a seguir:

$$a < b$$

Se a e b forem escalares, então o valor da expressão será 1 (verdadeira) se a for menor que b ; caso contrário, a expressão será 0 (falsa). Se a e b forem vetores com os valores a seguir:

$$\begin{aligned}a &= [2 \ 4 \ 6] \\b &= [3 \ 5 \ 1]\end{aligned}$$

Então, o valor de $a < b$ será o vetor $[1 \ 1 \ 0]$, enquanto o valor de $a \sim b$ será $[1 \ 1 \ 1]$.

Operadores Lógicos

Podemos combinar expressões usando os operadores lógicos do MATLAB. Os operadores são representados pelos seguintes símbolos.

Operadores	Descrição
------------	-----------

&	e
	ou
~	não

Quando duas expressões são unidas por *e* ; o resultado será 1 (verdadeiro) se ambas expressões forem verdadeiras, para expressões unidas por *ou*, o resultado será 1 (verdadeiro) se uma ou ambas expressões forem verdadeiras. Assim, para a seguinte expressão lógica

$$a < b \ \& \ b < c$$

O resultado será 1 (verdadeiro) somente se $a < b < c$; e falso (0) para todos resultados diferentes. Além disso, a operação só será válida se as matrizes resultantes ($a < b$ e $b < c$) tiverem o mesmo tamanho.

Uma expressão pode conter vários operadores lógicos, como a expressão abaixo:

$$\sim (b == c \mid b == 5.5)$$

O MATLAB analisaria primeiro, as expressões $b == c$ e $b == 5.5$ (obviamente, por causa do uso de parênteses). O resultado seria inversamente pelo operador *não*. Assim, suponha $b == 3$ e $c == 5$. Nenhuma das expressões é verdadeira, logo, a expressão $b == c \mid b == 5.5$ é falsa. Aplicando o operador *não*, o valor da expressão é alterado e a mesma torna-se verdadeira.

A prioridade dos operadores lógicos, do mais alto ao mais baixo, é: *não*, *e*, e *ou*.

Exercícios

Determine se as expressões nos problema 1 a 8 são verdadeiras ou falsas. Depois, verifique suas respostas usando o MATLAB. Lembre que ao verificá-las , você precisa entrar com a expressão. Suponha que as variáveis tenham os valores indicados abaixo:

$$a = 5.5 \qquad b = 1.5 \qquad k = -3$$

1. $a < 10.0$
 2. $a + b \geq 6.5$
 3. $k \sim 0$
 4. $b - k > a$
 5. $\sim(a == 3*b)$
 6. $-k \leq k + 6$
 7. $a < 10 \ \& \ a > 5$
 8. $\text{abs}(k) > 3 \mid k < b - a$
-

4.2 Tomada de Decisões

Estrutura If – Else – End

```
if    expressão
      Comandos
End
```

Se a expressão lógica for verdadeira, os comandos entre *if* e *end* são executados. Como exemplo, temos:

```
if a < 50
    count = count + 1;
    sum = sum + a;
end
```

Suponha que a seja um escalar. Se $a < 50$, então *count* é incrementada por 1 e a é adicionada à *sum*; caso contrário, os comandos não serão executados. Se a não for um escalar, então *count* é incrementado por 1 e a é adicionada à *sum* somente se cada elemento em a for menor que 50.

A próxima estrutura contém um parâmetro *if* dentro de outro parâmetro *if*:

```
if    expressão 1
      grupo de comandos A
      if expressão 2
        grupo de comandos B
      end
      grupo de comandos C
end
grupo de comandos D
```

Se a expressão 1 for verdadeira, os grupos de comandos A e C são executados. Se a expressão 2 também for verdadeira, o grupo de comandos B é executado antes do grupo de comandos C. Se a expressão 1 for falsa, imediatamente salta-se para os comandos D. Como exemplo, temos:

```
if a < 50
    count = count + 1
    sum = sum + a;
    if b > a
        b = 0;
    end
end
```

Novamente, suponha que a e b sejam escalares. Então, se $a < 50$ aumentaremos *count* por 1 e adicionaremos a à *sum*. Se $b > a$, então b será igual a zero. Se a não for menor que 50, então pula-se diretamente para o segundo end. Se a e nem b forem escalares, então b é maior que a somente se cada par de elementos correspondentes de a e b são valores nos quais $b > a$. Se a ou b é um escalar, então a matriz é comparada ao escalar.

Instrução *Else*

Esta instrução permite que executemos um comando se a expressão lógica é verdadeira e um diferente comando se a expressão é falsa. A forma geral do comando *if* combinada à instrução *else* é mostrada a seguir:

```
if      expressão
      grupo de comandos A
else
      grupo de comandos B
end
```

Se a expressão lógica é verdadeira, então o grupo de comandos A é executado. Caso contrário, o grupo de comandos B é executado.

Como exemplo, suponha que um táxi esteja passando entre dois edifícios. Considere que a variável d contenha a distância do veículo ao edifício mais próximo. Se o carro estiver a 10 metros do edifício, a velocidade é calculada usando a seguinte equação:

$$\text{velocidade} = 0,425 + 0,00175d^2$$

Se o táxi estiver a uma distância maior que 10 metros, use a equação a seguir:

$$\text{velocidade} = 0,625 + 0,12d - 0,00025d^2$$

Calculamos a velocidade correta com estes comandos:

```
if d <= 10
    velocidade = 0.425 + 0.00175*d^2
else
    velocidade = 0.625 + 0.12d - 0.00025*d^2
end
```

Quando há muitas alternativas a serem executadas, pode ser mais difícil determinar quais expressões lógicas devam ser verdadeiras (ou falsas) para executar cada grupo de comandos. Neste caso, a cláusula *elseif* é frequentemente usada para simplificar o programa lógico:

```
if      expressão 1
    grupo de comandos A
elseif  expressão 2
    grupo de comandos B
elseif  expressão 3
    grupo de comandos C
end
```

Se a expressão 1 for verdadeira, somente o grupo de comandos A é executado. Se a expressão 1 for falsa e a expressão 2 for verdadeira, então somente o segundo grupo de comandos é executado. Se as expressões 1 e 2 forem falsas e a expressão 3 for verdadeira, então somente o grupo de comandos C é executado. Se mais de uma expressão lógica for verdadeira, a primeira que for verdadeira determina qual grupo de comandos será executado. Se nenhuma das expressões lógicas forem verdadeiras, então nenhum dos comandos dentro da estrutura if é executado.

Else e *elseif* podem ser combinadas dentro de uma estrutura if-else-end, como mostramos a seguir:

```
if      expressão 1
    grupo de comandos A
elseif  expressão 2
    grupo de comandos B
elseif  expressão 3
    grupo de comandos C
else
    grupo de comandos D
end
```

Se nenhuma das expressões lógicas forem verdadeiras, então o grupo de comandos D é executado.

Exercícios

Nos problemas 1 a 7, dê os comandos MATLAB necessários para executar os passos indicados. Suponha que as variáveis são escalares.

1. Se *time* é maior que 50, então incremente-a por 1.
2. Quando a raiz quadrada de *poly* for menor que 0,001, imprima o valor de *poly*;
3. Se a diferença entre *volt_1* e *volt_2* for maior que 2, imprimir os valores de *volt_1* e *volt_2*;
4. Se o valor de *den* for menor que 0, 003; atribua zero a *result*; caso contrário, atribua a *result num* dividido por dez;

5. Se o logaritmo natural de x for maior ou igual a 10, atribua zero a *time* e incremente-o por *count*;
 6. Se *dist* for maior que 50 e *time* for maior que 10, incremente *time* por 2; caso contrário, incremente *time* por 5.
 7. Se *dist* for maior ou igual a 100, incremente *time* por 10. Se *dist* estiver entre 50 e 100, incremente *time* por 1. Caso contrário incremente *time* por 0,5.
-

4.3 Loop FOR

O MATLAB contém dois comandos para gerar loops, o comando *for* e o comando *while*. Nesta seção, discutiremos o comando *for*, e a seção 4.4 discutiremos o comando *while*.

O comando *for* tem a estrutura a seguir:

```
for variável = expressão
    Grupo de comandos A
end
```

Os comandos entre as instruções *for* e *end* são executados uma vez para cada coluna da expressão matricial. A cada iteração, a variável é atribuída para a próxima coluna da matriz, isto é, durante o i -ésimo ciclo do loop, temos que $\text{variável} = \text{expressão matricial}(:, i)$. Veja o exemplo a seguir:

Suponha que temos um grupo de valores que representam a distância de um táxi da torre mais próxima. Queremos gerar um vetor que contenha as respectivas velocidades. Se o táxi está a 10 metros do edifício, usamos a equação:

$$\text{velocidade} = 0,425 + 0,00175d^2$$

Se o táxi estiver a uma distância maior que 10 metros, use a equação a seguir:

$$\text{velocidade} = 0,625 + 0,12d - 0,00025d^2$$

Como a escolha da equação da velocidade depende do valor de d , devemos determinar separadamente $d(1)$, $d(2)$, e assim por diante. Contudo, não queremos o mesmo para calcular $\text{velocidade}(1)$, $\text{velocidade}(2)$ e assim por diante. Logo, usaremos um loop, com a variável usada como subscrito.

Na primeira solução, supomos que existiam 25 elementos no vetor d .

```
for d = 1:25
    if d <= 10
        velocidade = 0.425 + 0.00175*d^2
    else
        velocidade = 0.625 + 0.12*d - 0.00025*d^2
    end
end
```

Na próxima solução, supomos que o tamanho do vetor d é desconhecido. Contudo, usamos a função `size` para determinar o número de vezes que queremos executar o loop.

```
for k = 1:size(d,2)
    if d(k) <= 10
        velocidade = 0.425 + 0.00175*d(k)^2
    else
        velocidade = 0.625 + 0.12*d - 0.00025*d(k)^2
    end
end
```

As regras para um loop *for* são:

- Se o conjunto *for* uma matriz vazia, o loop não será executado. O fluxo de controle passará ao próximo comando após a instrução *end*;
- Se a expressão *for* um escalar, o loop será executado uma única vez;
- Se a expressão *for* um vetor-linha, então a cada iteração a variável conterà o próximo valor do vetor;
- Se a expressão *for* uma matriz, então a cada iteração a variável conterà a próxima coluna da matriz;
- Uma vez completo o loop *for* a variável contém o último valor usado.
- Se o operador dois-pontos é usado para definir a expressão matricial usando o formato:

for k = início: incremento: limite

Então o número de vezes que o loop executará pode ser calculado usando a equação:

$$\text{floor}\left(\frac{\text{limite} - \text{início}}{\text{incremento}}\right) + 1$$

Se este valor *for* negativo, o loop não será executado. Portanto, se um comando *for* contiver a seguinte informação:

for k = 5: 4: 83

Então, o número de vezes em que executa-se o loop será:

$$\text{floor} \left(\frac{83 - 5}{4} \right) + 1 = \text{floor} \left(\frac{78}{4} \right) + 1 = 20$$

O valor de k seria 5, depois 9, 13, e assim por diante até o valor final 81. O loop não seria executado para $k = 85$ porque é maior que o limite, 83.

- Comando *break*

O comando *break* pode ser usado para sair de um loop antes que o mesmo seja completo. É freqüentemente usado se houver um erro detectado dentro do loop.

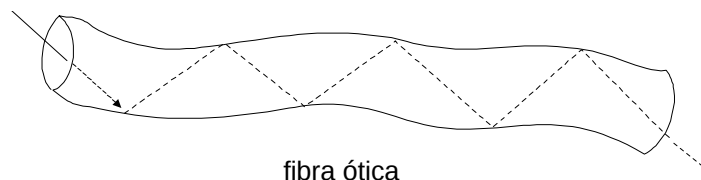
Exercícios

Determine o número de vezes que o loop *for* definido pelos comandos a seguir são executados. Verifique sua resposta .

1. for $k = 3:20$
2. for $\text{count} = -2:14$
3. for $k = -2:-1:10$
4. for $\text{time} = 10:-1:0$
5. for $\text{time} = 10:5$
6. for $\text{index} = 52 : -12$

Aplicação à Solução de Problemas: Fibras Óticas

Se a luz está direcionada para o extremo de uma longa haste de vidro ou plástico, a luz será totalmente refletida pelas paredes, ziguezagueando e segue adiante até chegar a outra extremidade. Este interessante fenômeno ótico pode ser usado para transmitir luz e imagens regulares, de um lugar para outro. Se “guia de luz”, a luz seguirá a forma da haste e emergirá somente na extremidade, como mostramos na figura a seguir:



A fibra ótica é uma fibra de vidro muito fina. Se os extremos das fibras são polidos e o arranjo espacial é o mesmo em ambos extremos (um feixe coerente), a fibra pode ser usada para transmitir uma imagem, e o feixe é chamado condutor de imagem. Se as fibras não tem o mesmo arranjo para ambos extremos (feixe incoerente), a luz é transmitida em vez da imagem, e por esta razão é chamada guia de luz. Por causa da flexibilidade das fibras óticas, as guias de luz e condutores de imagens são usados em instrumentos projetados para permitir a observação visual de objetos ou áreas que de outro modo seriam inacessíveis. Por exemplo, um endoscópio é um instrumento usado por físicos para examinar o interior do corpo de um paciente com somente um pequeno corte. Um endoscópio usa uma fibra ótica para transmitir a luz necessária dentro do corpo de um paciente. As fibras óticas podem ser usadas para transmitir feixes de luz LASER, o qual pode ser usado para desobstruir artérias, quebrar pedras nos rins, e limpar cataratas.

Este fenômeno de reflexão total interna origina-se da lei de Snell e depende unicamente dos índices de refração para os material considerados para guias de luz. Um guia de luz é composto de dois materiais — o núcleo central e o material que o envolve, denominado de blindagem. O material que compõe o núcleo tem o índice de refração mais alto que o meio que o envolve. Quando a luz incide em uma interface entre dois meios com índices de refração diferentes, parte dela é refletida a outra parte é refratada. A quantidade de luz refratada depende dos índices de refração dos materiais e o ângulo de incidência da luz. Se a luz incidente na interface entre dois meios provém de um material com maior índice de refração, então parte da luz atravessa a interface. O ângulo de incidência onde a luz é totalmente refletida pela superfície é chamado de ângulo crítico θ_c . Como o ângulo crítico depende dos índices de refração de dois materiais, podemos calcular este ângulo e determinar se a luz que entra em uma haste a um ângulo particular permanecerá no interior da mesma. Suponha que n_2 é o índice de refração do meio envolvente, e n_1 é o índice de refração da haste. Se n_2 é maior que n_1 , a haste não transmitirá luz; caso contrário, o ângulo crítico pode ser determinado pela equação

$$\text{sen } \theta_c = n_2 / n_1$$

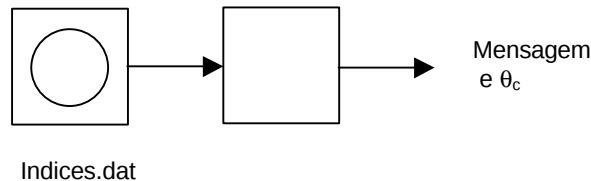
Escreva um programa MATLAB que determina se a luz será ou não transmitida por dois materiais que forma um guia. Suponha que um arquivo de dados ASCII denominado `indices.dat` contenha o número de possíveis índices de refração da fibra seguido pelo índice de refração do meio que o envolve. O programa deve determinar se os materiais formarão um guia de luz e, então, para quais ângulos a luz entra no guia.

1. ENUNCIADO DO PROBLEMA

Determine se os materiais especificados formarão ou não um guia de luz. Se eles não formarem, calcule os ângulos para o qual a luz pode entra na haste e ser transmitida.

2. DESCRIÇÃO ENTRADA \ SAÍDA

Como mostramos na figura abaixo, a entrada ao programa é um arquivo de dados contendo os índices de refração para os guias de luz em potencial. A saída é uma mensagem indicando se a luz é ou não transmitida e o ângulos para quais pode entrar no guia.



3. EXEMPLO MANUAL

O índice de refração do ar é 1,0003 e o índice do vidro é 1,5. Se formarmos um guia de luz de vidro envolvido pelo ar, o ângulo crítico θ_c pode ser calculado como mostramos a seguir:

$$\theta_c = \sin^{-1} (n_2 / n_1) = \sin^{-1} (1,0003 / 1,5) = \sin^{-1} (0,66687) = 41,82^\circ$$

O guia de luz transmitirá luz para todos os ângulos de incidência maiores que $41,82^\circ$.

4. SOLUÇÃO DO MATLAB

5. TESTANDO

4.4 Loops WHILE

O loop *while* é uma importante estrutura para repetição de um grupo de comandos quando a condição especificada for verdadeira. O formato geral para esta estrutura de controle é:

```

while expressão
    grupo de comandos A
end
  
```

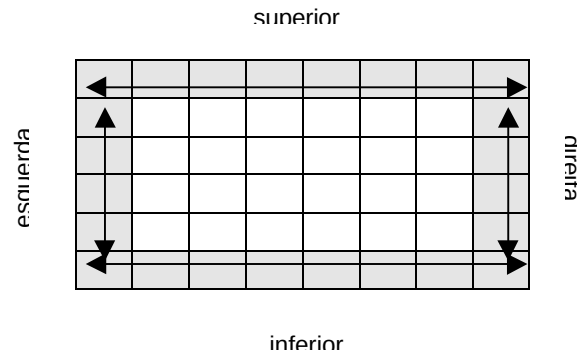
Se a expressão for verdadeira, então o grupo de comandos *A* é executado. Depois destes comandos serem executados, a condição é novamente questionada. Se for verdadeira, o grupo de comandos é novamente executado. Quando a condição for falsa, o controle pula para o comando posterior ao comando *end*. As variáveis modificadas no grupo de comandos *A* devem incluir as variáveis na expressão, ou o valor da expressão nunca será mudado. Se a expressão for verdadeira (ou é um valor não-nulo), o loop torna-se um loop infinito. (Lembre-se que você pode usar *^c* para sair um loop infinito).

Podemos mostrar que o uso de um loop *while* em um grupo de comandos que adicionam valores em um vetor para uma soma até o valor negativo ser alcançado. Como todos os valores no vetor podem ser positivos, a expressão no loop *while* deve acomodar a situação:

```
x = [ 1 2 3 4 5 6 7 8 9 ];
sum = 0;
k = 1;
while x(k) >= 0 & k < size(x,2)
    sum = sum + x(k);
    k = k + 1;
end
```

Aplicação à Solução de Problemas: Equilíbrio de Temperatura

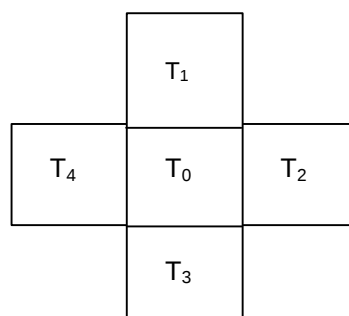
O projeto de novos materiais para o aperfeiçoamento das características do fluxo de ar acerca dos veículos envolve a análise dos materiais para não somente o fluxo de ar mas também para propriedades como a distribuição de temperatura. Neste problema, consideramos a distribuição em uma fina placa metálica tal que esta alcance um ponto de equilíbrio térmico. A placa é projetada para ser usada em uma aplicação na qual as temperaturas de todos os quatro lados da placa metálica estejam a temperaturas constantes ou a temperaturas isotérmicas. A temperatura para os outros pontos da placa é uma função da temperatura dos pontos envolventes. Se considerarmos as placas para serem semelhantes a uma grade, então uma matriz poderia ser usada para armazenar as temperaturas dos pontos correspondentes sobre a placa. A figura abaixo mostra uma grade para uma placa que está sendo analisada com seis medidas de temperatura ao longo dos lados e oito temperaturas ao longo das partes superior e inferior. Os pontos isotérmicos nos quatros lados são sombreados. Um total de 48 valores de temperaturas estão representados.



Grade de uma placa metálica

As temperaturas isotérmicas sobre os quatro lados seriam especificadas; supomos que os lados superior, esquerdo e direito são mantidos à mesma temperatura enquanto que o lado inferior da placa é mantida a uma temperatura diferente. Os pontos restantes são inicialmente selecionados para uma temperatura arbitrária, normalmente zero. A nova temperatura de cada ponto interno é calculado como a média das quatro temperaturas envolventes (veja a figura a seguir) e é dada por:

$$T_0 = \frac{T_1 + T_2 + T_3 + T_4}{4}$$



Pontos que envolvem T_0

Depois de calcular a nova temperatura para um ponto interno, a diferença entre a antiga e a temperatura recente é calculada. Se a mudança de temperatura for maior que algum valor de tolerância especificado, a placa já não está em equilíbrio térmico, e o processo inteiro é repetido.

Usamos as duas matrizes para as temperaturas, uma das antigas temperaturas e uma das recentes temperaturas. Precisamos de duas matrizes porque supomos que a temperatura muda para todos os pontos que ocorrem simultaneamente, sempre que as calculamos. Se usarmos uma única matriz, estaríamos atualizando informação sem que antes estivéssemos com a informação antiga. Por exemplo, suponha que estamos calculando a nova temperatura para a posição (3, 3). O novo valor é a média das temperaturas nas posições (2, 3), (3, 2), (3, 4), e (4, 3). Quando calculamos a nova temperatura para a posição (3, 4), novamente calculamos uma média, mas queremos usar o antigo valor na posição (3, 3), e não o seu valor atualizado.

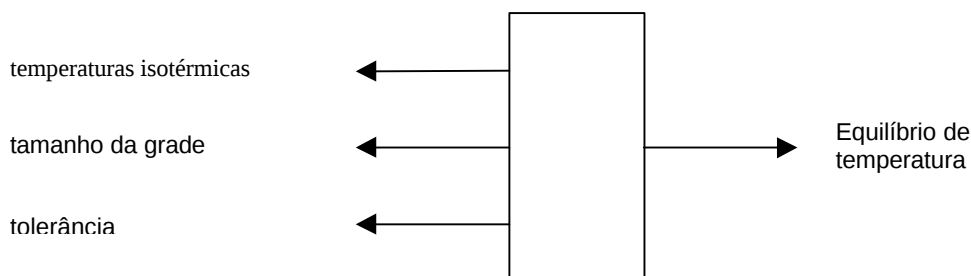
Portanto, usamos uma matriz de temperaturas antigas para calcular uma matriz de novas temperaturas e para determinar se alguma temperatura muda acima da tolerância. Quando movemos as novas temperaturas para o conjunto antigo. Quando nenhuma das temperaturas muda acima da tolerância, supomos que o equilíbrio foi alcançado, e imprimimos as temperaturas finais.

1. ENUNCIADO DO PROBLEMA

Determine os valores de equilíbrio para uma placa de metal com lados isotérmicos.

2. DESCRIÇÃO ENTRADA / SAÍDA

Como mostra a figura abaixo, a entrada é o tamanho da grade da placa, as temperaturas isotérmicas e o valor de tolerância. A saída é o grupo de temperaturas de equilíbrio para a placa de metal.



3. EXEMPLO MANUAL

Tenha a certeza que compreende o processo, examinamos um simples caso, estudando cada iteração. Supondo que a matriz contém quatro linhas e quatro colunas. As temperaturas isotérmicas são 100° e 50°, e nós iniciamos todos os pontos restantes do zero. Usamos uma tolerância de 40°. O grupo inicial de temperaturas e as sucessivas iterações ao equilíbrio térmico são mostradas a seguir:

Temperaturas Iniciais

100	100	100	100
100	0	0	100
100	0	0	100
50	50	50	50

Primeira Iteração

100	100	100	100
100	50	50	100
100	37,5	37,5	100
50	50	50	50

Segunda Iteração

100	100	100	100
100	71,875	71,875	100
100	59,375	59,375	100
50	50	50	50

Como nenhuma das temperaturas alteradas entre a primeira e segunda interação ultrapassam a tolerância de 40^0 , as temperaturas na segunda interação estão também em equilíbrio.

4. SOLUÇÃO DO MATLAB

5. TESTANDO

Capítulo 5 - Medidas Estatísticas

Analisar dados coletados de ensaios de engenharia é uma parte importante da avaliação dos mesmos. O alcance da análise estende-se dos mais simples cálculos de dados, como a média aritmética, à mais complexa análise que calcula medidas como o desvio padrão ou variância dos dados. Medidas como estas são medidas estatísticas porque suas propriedades não são exatas. Por exemplo, o seno de 60° é uma medida exata pois o valor é sempre o mesmo toda vez que o calculamos, mas a velocidade máxima que atingimos com o nosso carro é uma medida estatística porque varia dependendo de parâmetros como a temperatura, condições da estrada, e se estamos nas montanhas ou no deserto. Não só podemos medir as propriedades e características de dados estatísticos como também usar o computador para gerar seqüências de valores (números aleatórios) com características específicas. Neste capítulo, aprenderemos a usar as funções para análise de dados do MATLAB e a gerar seqüências de números aleatórios com características específicas.

5.1 Funções para Análise de Dados

Para se estudar o desempenho de duas companhias corretoras de ações, selecionou-se de cada uma delas amostras aleatórias das ações negociadas. Para cada ação selecionada, computou-se a porcentagem de lucro apresentada durante um período de tempo. Os dados estão a seguir:

Corretora A 45 60 54 62 55 70 38 48 64 55 56 55 54 59 48 65 55 60
Corretora B 57 55 58 50 52 59 59 55 56 61 52 53 57 57 50 55 58 54

Os gráficos para os dados das corretoras A e B são mostrados abaixo para podermos comparar os dois conjuntos de dados:

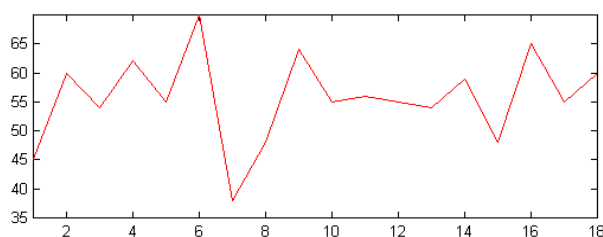


Figura 5.1 - Porcentagem de lucro apresentada pela corretora A durante 18 dias.

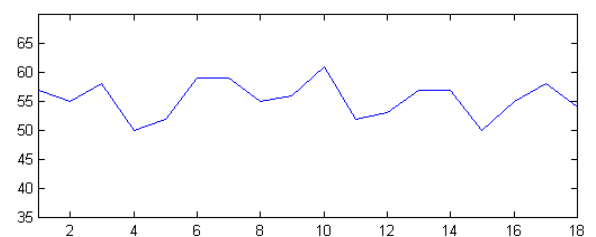


Figura 5.2 - Porcentagem de lucro apresentada pela corretora B durante 18 dias.

Para decidir qual corretora obteve melhor desempenho, alguns critérios foram considerados como:

- Média mais alta de porcentagem de lucro;
- Maior porcentagem de lucro;
- Menor variação de Porcentagem de lucro;

É visualmente perceptível que ao longo dos dezoito dias a corretora B apresentou menor variação de lucro. Facilmente também observaríamos que a Corretora A obteve o maior percentual de lucro de ações. Contudo, não é possível sabermos quantitativamente estas e outras informações como a média de percentagem do lucros das ações para cada corretora apenas com a observação dos gráficos. Para isso temos que calcular as grandezas necessárias para determinar qual corretora obteve melhor desempenho. Seria um pouco trabalhoso se o fizéssemos manualmente. O MATLAB pode perfeitamente auxiliar-nos nestes casos porque contém uma série de funções que contribuem para uma análise mais precisa dos dados. Algumas destas funções podem ser aplicadas ao exemplo das corretoras como as mostradas a seguir:

Qual a média percentual de lucro das ações durante os 18 dias de observação?

mean : calcula a média aritmética de um grupo de valores. Assim, para as corretoras A e B, temos:

» mean (corretoraA)	» mean (corretoraB)
ans =	ans =
55.7222	55.4444

Qual corretora alcançou a mais alta percentagem de lucros?

A função *max* determina a maior percentagem de lucro em cada corretora. A função *max* determina o maior valor de um conjunto de dados.

» max(corretoraA)	» max(corretoraB)
ans =	ans =
70	61

E a menor percentagem?

As menores margens de lucro obtidas por cada corretora são dadas pelo comando min .

» min(corretoraA)	» min(corretoraB)
ans =	ans =
38	50

Qual corretora apresenta menor variação de percentual de lucro de ações?

As duas corretoras tiveram médias bastantes próximas. Contudo, a média, por ser uma medida representativa de posição central, mascara toda a informação sobre a variabilidade dos dados das corretoras A e B. É necessário uma medida que resuma a variabilidade de dois grupos de valores, permitindo compara-los, conforme algum critério estabelecido. O critério mais comum é aquele que mede a concentração de dados em torno de sua média e, as medidas mais usadas são: o *desvio médio* e a *variância*.

Desvio Médio, Variância e Desvio Padrão

Sabendo que a média dos valores da corretora A é 55,72, os desvios $x_i - \bar{x}$ são: 10,72; - 4,28; 1,72; - 6,28; 0,72; -14,28, ... Para qualquer conjunto de dados, a soma dos desvios é igual a zero. É mais conveniente usarmos ou o total dos desvios em valor absoluto ou o total dos quadrados dos desvios. Assim, teríamos:

$$\sum_{i=0}^{18} |x_i - \bar{x}| = 10,72 + 4,28 + 1,72 + 6,28 + 0,72 + 14,28 + \dots + 0,72 + 4,28 = 100,4436$$

$$\sum_{i=0}^{18} (x_i - \bar{x})^2 = 1,56 + 0,44 + 2,56 + 5,44 + 3,44 + 3,56 + \dots + 2,56 + 1,44 = 1060,5294$$

O *desvio médio* e a *variância* são definidos, usando os termos acima:

$$\text{DesvioMédio} = \frac{\sum_{i=1}^n |x_i - \bar{x}|}{n} \qquad \text{Variância} = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n}$$

Então, *desvio médio* e a *variância* dos dados da corretora A são respectivamente 5,5802 e 58,9183. Para a corretora B, seriam 2,5556 e 9,9084. A corretora A tem maior variabilidade em porcentagem de lucro de ações, segundo o desvio médio. Isto significa que o percentual de lucro de ações da corretora B é mais homogêneo que o da corretora A.

Para evitar erros de interpretação — a variância é uma medida que expressa um desvio quadrático médio — usamos o desvio padrão (σ), que é definido como a raiz quadrada da variância (σ^2). O MATLAB tem uma função específica para o cálculo de desvio padrão denominada std. Portanto, para o exemplo das corretoras:

» std(corretoraA)

ans =7.6758

» std(corretoraB)

ans =3.1478

Para o MATLAB, a variância de um conjunto de dados é definida por:

$$\sigma^2 = \frac{\sum_{k=1}^N (x_k - \mu)^2}{N - 1}$$

onde μ é a média do conjunto de valores. O denominador $N - 1$ deve ser usado toda vez que estivermos trabalhando com uma amostra. Quando não houver a preocupação em saber se trabalhamos com população ou amostra, podemos usar o denominador n . Para grandes amostras pouca diferença fará se usarmos um ou outro.

Os comandos *mean*, *std*, *max* e *min* determinam, respectivamente, a média, o desvio padrão, o máximo valor e o valor mínimo de um vetor. No exemplo acima, usamos os dados da corretora A e armazenamos em um vetor denominado “corretoraA” e fizemos o mesmo para a corretora B. Mas estas funções também são válidas para matrizes. Elas retornam um vetor-linha contendo, respectivamente, a média, o desvio padrão, o máximo valor, o mínimo valor de cada coluna. Um exemplo de aplicação de medidas estatísticas para matrizes é dado a seguir:

A JR Indústria de Alimentícios, desejando melhorar o nível de seus funcionários em cargos de chefia, montou um curso experimental e indicou 15 funcionários para sua primeira turma. Os dados referentes à seção a que pertencem e notas obtidas estão na tabela a seguir.

Usando os dados da tabela, determine:

- A média em cada disciplina;
- As menores notas em cada disciplina e os funcionários que as obtiveram;

E,

- Dispor as notas de Administração em forma crescente;
- Comparar os funcionários das seções de Vendas e Técnicas e determinar a maior nota destes funcionários em cada disciplina;
- Comparar a distribuição dos dados nas disciplinas de Direito, Metodologia e Economia.

Func.	Seção	ADM	DIR	RED	ECO	ING	MET
1	Pessoal	8.0	9.0	8.6	8.5	9.0	9.0
2	Pessoal	8.0	9.0	7.0	8.0	9.0	6.5
3	Pessoal	8.0	9.0	8.0	8.5	8.0	9.0
4	Pessoal	6.0	9.0	8.6	8.5	8.0	6.0
5	Pessoal	8.0	9.0	8.0	9.0	9.0	6.5
6	Técnica	10.0	9.0	8.0	8.5	8.0	6.0
7	Técnica	8.0	9.0	9.0	8.0	9.0	10.0
8	Técnica	10.0	9.0	8.0	7.5	8.0	9.0
9	Técnica	8.0	9.0	10.0	8.0	10.0	10.0
10	Técnica	8.0	9.0	7.0	8.5	7.0	6.5
11	Vendas	8.0	9.0	9.0	7.5	7.0	9.0
12	Vendas	8.0	9.0	7.0	7.0	8.0	10.0
13	Vendas	8.0	9.0	8.0	7.5	9.0	6.0
14	Vendas	6.0	9.0	9.0	7.5	4.0	6.0
15	Vendas	6.0	9.0	4.0	8.5	7.0	6.0

ADM – administração
DIR – direito
RED – redação
ECO – economia
ING – inglês
MET – metodologia

Sabemos que a função mean calcula a média dos elementos de um vetor. Em uma matriz, a função calcula um vetor linha que contém a média de cada coluna. Para o exemplo do curso de aperfeiçoamento, podemos considerar a lista de funcionários e suas respectivas notas em cada disciplina, uma matriz 15 x 6, onde as colunas representam as disciplinas do curso, facilitando o cálculo da média da turma em cada matéria. Então, definindo a matriz no MATLAB.

```
»TURMA = [ 8.0 9.0 8.6 8.5 9.0 9.0
            8.0 9.0 7.0 8.0 9.0 6.5
            8.0 9.0 8.0 8.5 8.0 9.0
            6.0 9.0 8.6 8.5 8.0 6.0
            8.0 9.0 8.0 9.0 9.0 6.5
            10.0 9.0 8.0 8.5 8.0 6.0
            8.0 9.0 9.0 8.0 9.0 10.0
            10.0 9.0 8.0 7.5 8.0 9.0
            8.0 9.0 10.0 8.0 10.0 10.0
            8.0 9.0 7.0 8.5 7.0 6.5
            8.0 9.0 9.0 7.5 7.0 9.0
            8.0 9.0 7.0 7.0 8.0 10.0
            8.0 9.0 8.0 7.5 9.0 6.0
            6.0 9.0 9.0 7.5 4.0 6.0
            6.0 9.0 4.0 8.5 7.0 6.0];
```

E denominando o vetor que representa a média da turma em cada disciplina de MEDIA_TURMA, temos:

```
» MEDIA_TURMA = mean (TURMA)
```

```
» MEDIA_TURMA =
```

```
7.8667  9.0000  7.9467  8.0667  8.0000  7.7000
```

A saída do MATLAB mostra que as médias da turma nas disciplinas de Administração, Direito, Redação, Economia, Inglês e Metodologia são, respectivamente, 7.8667, 9.0000, 7.9467, 8.0667, 8.0000 e 7.7000.

Se quiséssemos apenas a média da turma em Redação, usaríamos o operador dois pontos (:), conforme mostraremos a seguir:

```
» MEDIA_RED = mean (TURMA (:, 3))
```

```
» MEDIA_RED =
```

```
7.9467
```

ou ainda, a média dos funcionários da seção de Pessoal em Direito:

```
» MEDIA_PDIR = mean (TURMA(1:5, 2))
```

```
» MEDIA_PDIR =
```

```
9.0000
```

- Menores notas e os funcionários que as obtiveram

A função *min* retorna o menor valor de um vetor. Em uma matriz, a função retorna um vetor-linha cujos elementos são os menores valores de cada coluna. Além disso, apresenta a forma $[y, i] = \min(x)$ onde armazena os menores elementos de cada coluna em um vetor *y*, e os índices (respectivas linhas) no vetor *i*. se existirem valores idênticos, o índice do primeiro é retornado. Logo, se usarmos esta forma da função *min* na matriz TURMA, podemos determinar a menor nota em cada disciplina o funcionário que a obteve. No caso, o vetor *y* denominar-se-á GRAU_MIN.

```
» [ GRAU_MIN, i ] = min (TURMA)
```

```
» GRAU_MIN =
```

```
6  9  4  7  4  6
```

```
i =
```

```
14  1 15 12 14  4
```

Assim, o funcionário 14 obteve a menor nota (6,0) em Administração, o mesmo acontece em Inglês. Observe que atribuiu-se ao funcionário 1 o menor grau em Direito, disciplina em que todos tiveram o mesmo grau. Conforme vimos, quando há valores mínimos idênticos, o índice do primeiro valor é retornado. Por isso, a atribuição do funcionário 1.

Esta forma é válida também para a função *max*.

- Notas de Administração em Forma Ascendente

O MATLAB contém uma função que distribui, em ordem ascendente, os elementos de um vetor ou os elementos de cada coluna de uma matriz. Sua forma geral é:

`sort(x)`

Como queremos apenas as notas da turma em Administração em ordem ascendente, usamos a função *sort*. Porém, devemos antes selecionar a disciplina de Administração, utilizando o operador dois pontos (:),

```
» ADM = TURMA( : , 1);  
» GRAU_ASC = sort (ADM)
```

```
» GRAU_ASC =
```

Poderíamos calcular diretamente sem precisar definir o vetor-coluna ADM:

```
» GRAU_ASC = sort (TURMA( : , 1))
```

Se quiséssemos somente as três menores notas, faríamos:

```
» GRAU_ASC = sort (TURMA( : , 1));  
» ASC_3 = sort (GRAU_ASC ( 1: 3 , 1))
```

A função *sort* também apresenta a forma `[y, I] = sort (x)` onde os valores dispostos em forma crescente são armazenados na matriz *y* e seus índices, na matriz *i*. Se *x* contiver valores complexos, os elementos serão distribuídos de forma ascendente de acordo com seus valores absolutos.

Tente fazer:

```
» [y,i] = sort (TURMA)
```

- Comparação entre as distribuições de dados de Direito, Metodologia e Economia.

Para auxiliar na comparação entre dados de variáveis, usaremos um tipo especial de gráfico que particularmente relevante às medidas estatísticas discutidas nesta seção. Um *histograma* é um gráfico que mostra a distribuição de um grupo de valores. No MATLAB, o histograma calcula o número de valores a distribuir em 10

barras igualmente espaçadas entre os valores máximos e mínimo de um grupo de valores. A forma geral para plotar um histograma é mostrada a seguir:

» hist (x)

onde x é um vetor contendo os dados a serem usados no histograma. Se quisermos plotar a segunda coluna em uma matriz, podemos usar o operador *dois pontos* como mostrado a seguir:

» hist (x (:, 2))

O comando *hist* nos permite selecionar o número de barras. Portanto, se quisermos aumentar a resolução dos histograma tal que usemos 25 barras, em vez de 10, faremos:

» hist (x , 25)

A informação usada para plotar também pode ser armazenada em vetores. Considere os seguintes comandos:

» [n , x] = hist (GRAU_MIN);

» [n , x] = hist (GRAU_MIN, 6);

Nenhum destes comandos plota um histograma. O primeiro comando calcula valores para dois vetores, *n* e *x*. O vetor *n* contém os dados para 10 barras, e o vetor *x* contém o ponto que corresponde ao meio de cada alcance da barra. O segundo comando é similar, mais armazena 6 dados em *n*, e 6 pontos de meio barra em *x*. Estes vetores são usados geralmente em gráficos do tipo *bar*, que será discutido no capítulo 7.

Agora, podemos usar o histograma para comparar e indicar as diferenças existentes entre as distribuições de dados de Direito, Metodologia e Economia.

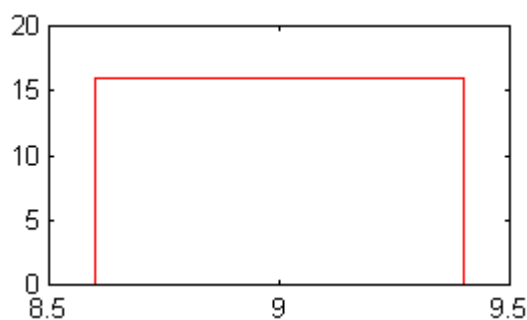


Figura 5.3 - Histograma de Direito

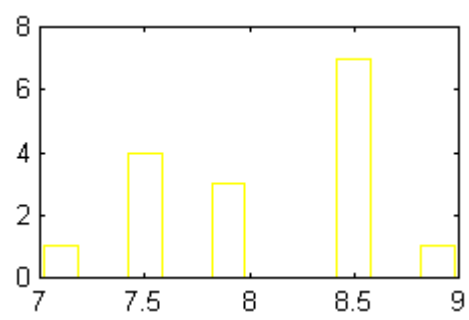


Figura 5.4 - Histograma de Economia

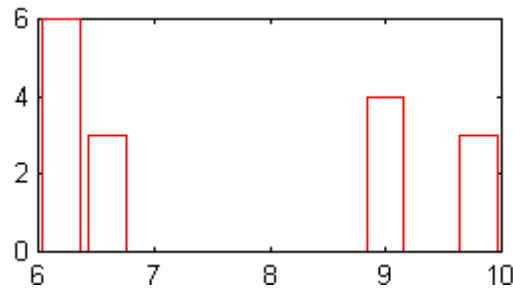


Figura 5.5 - Histograma de Metodologia

Observe as diferenças entre os histogramas. Enquanto o histograma de Direito indica que todos os alunos obtiveram grau 9,0; em Economia, indica que apenas 1 funcionário conseguiu o mesmo grau e em Metodologia, quatro funcionários. Também em Metodologia, há maior percentagem de notas menores que 7,0. Podemos afirmar que as notas são mais homogêneas em Direito. O que é confirmado se usarmos funções já vistas como *mean* e *std*.

Para terminar a parte de funções de análise de dados, mais um exemplo. Seja o quadro abaixo que mostra a taxa de juros durante os últimos 18 meses. Suponha que alguém lhe perguntasse como acompanhar o acúmulo de taxa de juros mês a mês.

Mês/ano	Jan/97	Fev/97	Mar/97	Abr/97	Mai/97	Jun/97	Jul/97
Juros a.m (%)	1,25	1,16	1,13	1,12	1,4	1,16	1,16

Mês/ano	Ago/97	Set/97	Out/97	Nov/97	Dez/97	Jan/98	Fev/98
Juros a.m (%)	1,13	1,15	1,10	2,04	1,82	1,65	0,95

Provavelmente, você diria que bastaria somar à taxa de juros inicial, os juros do mês seguinte e assim por diante. É um exemplo típico de *soma cumulativa*.

No MATLAB, a função que corresponde à soma cumulativa dos elementos de um vetor é *cumsum* conforme mostrado a seguir:

```
» JUROS = [1.25 1.16 1.13 1.12 1.4 1.16 1.16 1.13 1.15 1.10 2.04 1.82 1.65 0.95];
» ACUM_JUROS = cumsum (JUROS)
ACUM_JUROS =
```

```
Columns 1 through 7
```

```
1.2500 2.4100 3.5400 4.6600 6.0600 7.2200 8.3800
```

Columns 8 through 14

9.5100 10.6600 11.7600 13.8000 15.6200 17.2700 18.2200

Logo, o primeiro elemento do vetor ACUM_JUROS é 1,25 ; o segundo será 1,25 + 1,16 ; o terceiro, 1,25 + 1,16 + 1,13 e assim por diante.

Se quisermos saber o total de juros durante este intervalo de tempo, usamos a função *sum* :

» TOTAL_JUROS = sum(JUROS)

TOTAL_JUROS =

18.2200

As funções *prod* (x) e *cumprod* (x) são análogas às funções *sum* (x) e *cumsum*(x), onde *prod*(x) calcula o produto dos elementos de um vetor ou produto de cada coluna, em uma matriz; *cumprod* (x) calcula o produto acumulativo dos elementos de um vetor x.

Há também a função *median* que calcula a mediana dos elementos de um vetor x. Mediana é a realização que ocupa a posição central da série de observações quando estas estão ordenadas segundo suas grandezas (crescente ou decrescentemente). Assim, se as cinco observações de uma variável, como erros de impressão na primeira página forem 20, 16, 14, 8 e 19; a mediana é o valor 16. Se o número de observações é par, usa-se como mediana a média aritmética das duas observações centrais. Logo, se as observações de uma variável, forem 20, 14, 16 e 8, a mediana é

$$\text{Mediana} = (14 + 16) / 2 = 15.$$

Exercícios

Determine as matrizes representadas pelas funções a seguir. Depois, use o MATLAB para verificar suas respostas. Suponha que w, x, e y sejam as matrizes:

$$w = [0 \ 3 \ -2 \ 7]$$
$$x = [3 \ -1 \ 5 \ 7]$$

$$y = \begin{bmatrix} 1 & 3 & 7 \\ 2 & 8 & 4 \\ 6 & -1 & -2 \end{bmatrix}$$

1. `max (w)`
2. `min(y)`
3. `min (w, x)`
4. `[ z, i ] = max(y)`
5. `mean (y)`
6. `median(w)`
7. `cumprod(y)`
8. `sort(2*w + x)`
9. `sort (y)`
10. `std (w)`
11. `std(x)^2`
12. `std (y(:, 2))`
13. `std (y)`
15. `std (y).^2`

Aplicação à Solução de Problemas: Análise do Sinal de Voz

Suponha que queiramos projetar um sistema que reconheça as palavras que representam os dez algarismos: “ zero”, “um”, “dois”, ..., “ nove”. Uma das primeiras coisas que devemos fazer é analisar os dados para as dez seqüências correspondentes (ou sinais) para verificar se há algumas medidas estatísticas nas quais possamos distingui-los. As funções para análise de dados do MATLAB nos permite facilmente calcular estas medidas. Podemos então imprimir uma tabela de medidas e procurar por aquelas que nos permita distinguir valores.

Atualmente, podemos usar sinais de voz para calcular um número de medidas estatísticas que poderiam ser usadas como parte de um algoritmo de reconhecimento de dígitos. Os dados para cada dígito contém mil valores. Escreva um programa MATLAB para ler um arquivo de dados ASCII denominado zero.dat e calcule as seguintes informação: média, desvio padrão, variância, a média do módulo.

Já discutimos média, desvio padrão e variância. A média quadrática é o valor médio quadrático será discutido com mais detalhes na próxima seção. A média dos módulo é a média dos valores absolutos do conjunto de dados.

1. ENUNCIADO DO PROBLEMA

Calcular as medidas estatísticas a seguir para um sinal de voz.

2. DESCRIÇÃO ENTRADA/SAÍDA

O diagrama abaixo mostra o arquivo que contém os dados como a entrada e as várias medidas estatísticas como a saída.

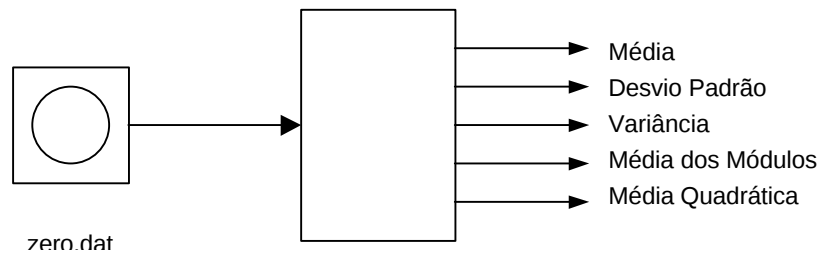


Figura 5.6 – diagrama I/O

3. EXEMPLO MANUAL

Para um exemplo manual, suponha que um sinal contenha a seguinte sequência de valores:

[2.5 8.2 -1.1 -0.2 1.5]

Usando uma calculadora, calculamos os seguintes valores:

- média = $(2.5 + 8.2 - 1.1 - 0.2 + 1.5)/5 = 2.18$
- variância = $[(2.5 - \mu)^2 + (8.2 - \mu)^2 + (-1.1 - \mu)^2 + (-0.2 - \mu)^2 + (1.5 - \mu)^2]/4 = 13.307$
- desvio padrão = $\sqrt{13.307} = 3.648$
- média quadrática = $(2.5^2 + 8.2^2 + (-1.1)^2 + (-0.2)^2 + 1.5^2)/5 = 15.398$
- média dos módulos = $(|2.5| + |8.2| + |-1.1| + |-0.2| + |1.5|)/5 = 15.398$

4. SOLUÇÃO DO MATLAB

5. TESTANDO

5.2 Números Aleatórios

Números aleatórios não são definidos por uma equação. Contudo, possuem certas características que os definem. Há muitos problemas que pedem o uso de números aleatórios no desenvolvimento de uma solução. Em alguns casos são usados para desenvolver a simulação de um problema complexo. A simulação pode ser testada diversas vezes para analisar os resultados e cada teste representa uma repetição do experimento. Também usamos números aleatórios para aproximar seqüências de ruído. Por exemplo, o que ouvimos no rádio é uma seqüência de ruído. Se estivermos testando um programa que use um arquivo de entrada que representa um sinal de rádio, poderíamos gerar ruídos e adicioná-los ao sinal de voz ou a uma música em seqüência para prover mais sinais reais.

Função Número Aleatório

A função *rand* no MATLAB gera números aleatórios no intervalo [0,1]. Os números aleatórios podem ser uniformemente distribuídos no intervalo [0,1] ou podem ser distribuídos normalmente, nos quais média e variância são, respectivamente, 0 e 1. Um valor é usado para iniciar uma seqüência aleatória.

Rand (n)	Retorna uma matriz de ordem n cujos elementos são números aleatórios entre 0 e 1.
Rand (m,n)	Matriz com m linhas e n colunas cujos elementos são números aleatórios entre 0 e 1.
Rand(size (A))	Matriz de mesma ordem que a matriz A.
Rand ('uniform')	Especificam se os números aleatórios desejados serão uniformemente ou normalmente distribuídos.
Rand ('normal')	
Rand ('seed', n)	Determina n como o valor inicial de uma seqüência de números aleatórios.
Rand ('seed')	Retorna ao valor atual de início do gerador de números aleatórios.

Função Densidade de Probabilidade

Suponha que o ponteiro dos segundos de um relógio elétrico possa parar a qualquer instante por defeitos técnicos. Como há infinitos pontos nos quais o ponteiro pode parar, cada uma com igual probabilidade, cada ponto teria a probabilidade de ocorrer igual a zero. Contudo, podemos determinar a probabilidade de o ponteiro parar numa região entre dois valores quaisquer. Assim, a probabilidade de o ponteiro parar no intervalo entre os números 9 e 12 é $\frac{1}{4}$, pois neste intervalo corresponde a $\frac{1}{4}$ do total. Então,

$$P(270^0 \leq X \leq 360^0) = \frac{1}{4}$$

onde X é medido em graus.

Sempre poderemos achar a probabilidade de o ponteiro parar num ponto qualquer de um intervalo, por menor que seja, compreendido entre os números a e b , de forma que

$$P(a \leq X \leq b) = \frac{b - a}{360^\circ}$$

Construindo um histograma da variável X :

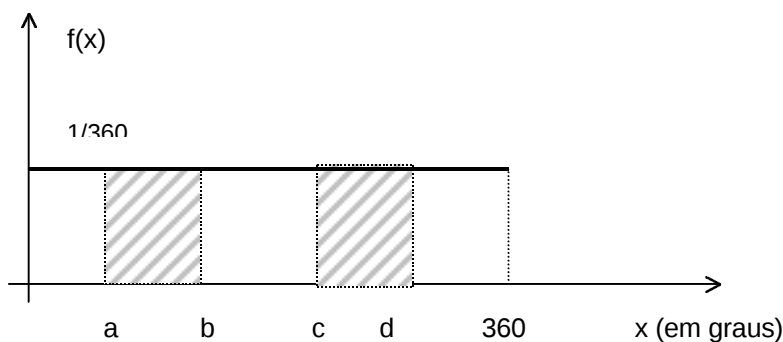


Figura 5.7 – histograma da variável X

O histograma corresponde à função:

$$f(x) = \begin{cases} 0, & \text{se } X < 0^\circ \\ 1/360, & \text{se } 0^\circ \leq X < 360^\circ \\ 0, & \text{se } x \geq 360^\circ \end{cases}$$

A área correspondente ao intervalo $[a,b]$ indica a probabilidade de a variabilidade de a variável estar entre a e b . Matematicamente,

$$P(a \leq X \leq b) = \int_a^b f(x) dx = \int_a^b \frac{1}{360^\circ} dx = \frac{b - a}{360^\circ}$$

A função $f(x)$ é chamada função densidade de probabilidade (f.d.p). A f.d.p. determina a região onde há maior probabilidade de uma variável X assumir um valor pertencente a um intervalo.

Existem alguns modelos freqüentemente usados para representar a f.d.p. de uma variável aleatória contínua como :

a) Modelo Uniforme

- Definição: Uma variável aleatória X tem distribuição uniforme com $\underline{a}$ e $\underline{b}$ ($a < b$) reais, se a sua f.d.p. é dada por:

$$f(x) = \begin{cases} \frac{1}{b-a}, & \text{se } a < x < b \\ 0, & \text{demais pontos.} \end{cases}$$

- Gráfico:

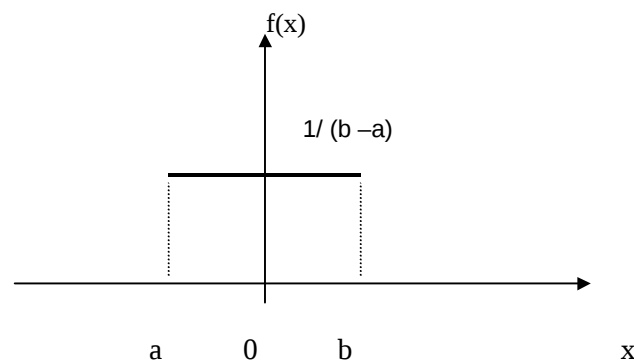


Figura 5.9 – distribuição uniforme

- Variância (σ^2)

$$\sigma^2 = \frac{(b-a)^2}{12}$$

b) Modelo Normal

A variável X tem distribuição normal com parâmetros μ e σ^2 , denotada por $X: (\mu, \sigma^2)$, onde $-\infty < \mu < +\infty$ e $0 < \sigma^2 < +\infty$, se sua f.d.p. é dada por:

$$f(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Para $-\infty < \mu < +\infty$.

- Gráfico: A curva normal é particular para média μ e variância σ^2 .

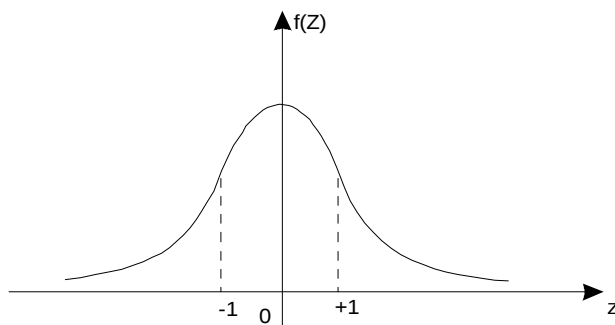


Figura 5.9 – Curva Normal Padrão

Como a probabilidade de a variável X ocorrer num intervalo, matematicamente, é a área sob a curva, teríamos valores diferentes de probabilidade para cada valor de μ e de σ . Por isso, usamos a curva normal *padrão* ou *reduzida* ($\mu = 0$, $\sigma^2 = 1$) – ver figura acima, denotada por $N(0,1)$. Se $X: N(\mu, \sigma^2)$, então a variável aleatória Z com distribuição $N(0, 1)$ é definida por:

$$Z = \frac{X - \mu}{\sigma}$$

Exemplo: Suponha que as amplitudes de vida de dois aparelhos elétricos, D1 e D2, tenham distribuições $N(42, 36)$ e $N(45, 9)$, respectivamente. Se o aparelho é para ser usado por um período de 45 horas, qual aparelho deve ser preferido? E se for por um período de 49 horas?

a) 45 horas

$$P(D1 > 45) = P\left(\frac{T - 42}{6} > \frac{45 - 42}{6}\right) = P\left(Z > \frac{1}{6}\right) = 0,5 - 0,192 = 0,308$$

$$P(D2 > 45) = P\left(\frac{T - 45}{3} > \frac{45 - 45}{3}\right) = P(Z > 0) = 0,5$$

b) 49 horas

$$P(D1 > 49) = P\left(\frac{T-42}{6} > \frac{49-42}{6}\right) = P\left(Z > \frac{7}{6}\right) = 0,5 - 0,377 = 0,123$$

$$P(D2 > 49) = P\left(\frac{T-45}{3} > \frac{49-45}{3}\right) = P\left(Z > \frac{4}{3}\right) = 0,5 - 0,407 = 0,093$$

O comando *hist* pode ser usado para estimar a f.d.p. de um sinal qualquer. Se tivermos mil valores e plotarmos o histograma, plotaremos um versão de f.d.p. para números aleatórios. Antes, aprenderemos como o MATLAB pode ser usado para gerar números aleatórios uniforme ou normalmente distribuídos. Por exemplo, se quisermos gerar uma sequência de 10 números aleatórios uniformemente distribuídos entre os valores 0 e 1, devemos fazer:

```
rand('seed',0)
rand('uniform')
rand(10,1)
```

Podemos converter um número r que está uniformemente distribuído entre 0 e 1 para um valor situado entre um intervalo qualquer. Por exemplo, se quisermos gerar 10 números aleatórios uniformes entre -5 e 5, devemos primeiro gerar os números entre 0 e 1. Logo em seguida, usamos a equação:

$$x = (b - a) \cdot r + a$$

onde b e a são, respectivamente, os limites inferior e superior (no caso, 5 e -5). Assim, suponha que armazenamos os dez números aleatórios determinados anteriormente em um vetor coluna denominado SINAL. Para convertê-los em valores entre -5 e 5 usamos:

$$x = (10 \cdot \text{SINAL}) - 5$$

Exercícios

Determine os comandos do MATLAB necessários para gerar 10 números aleatórios com as características específicas. Verifique suas respostas.

- Números uniformemente distribuídos entre 0 e 10,0
- Números uniformemente distribuídos entre -1 e 1.
- Números uniformemente distribuídos entre -20 e -10.
- Números uniformemente distribuídos entre 4,5 e 5,0
- Números uniformemente distribuídos entre π e $-\pi$.

O MATLAB também gera valores distribuídos normalmente cuja média é igual a zero e a variância, igual a 1. Se quisermos modificar estes valores para uma outra distribuição, multiplicamos os valores pelo desvio padrão da distribuição desejada e os adicionamos à média desta distribuição. Portanto, se r é um número qualquer com média 0 e variância 1, a equação a seguir irá gerar um novo número aleatório x com desvio padrão a e média b :

$$x = a \cdot r + b$$

Os comando abaixo geram números normalmente distribuídos com média 5 e variância igual a 2:

```
rand('seed',0)
rand('normal')
s = sqrt(2)*rand(10,1) + 5
```

Os valores a serem impressos por este programa são:

```
s =
6.6475
5.8865
5.1062
5.4972
4.0150
7.3987
5.0835
7.5414
5.3734
6.2327
```

Funções Densidade de Probabilidade: Histograma

Conforme dito anteriormente, podemos usar um histograma para avaliar a f.d.p de um sinal qualquer. Se tivermos mil valores, e os plotarmos, estaremos plotando uma versão da f.d.p. para os números aleatórios. Por exemplo, suponha que geramos mil números aleatórios entre 0 e 1, e os armazenamos no vetor `U_VALORES`. Podemos usar o comando `hist` para plotar a f.d.p. usando as 25 barras:

```
rand('uniform')
U_VALORES = rand(1000, 1)
hist(U_VALORES, 25)
```

O gráfico é mostrado na figura abaixo. Os números aleatórios estão distribuídos entre 0 e 1 e a distribuição é relativamente alisada.

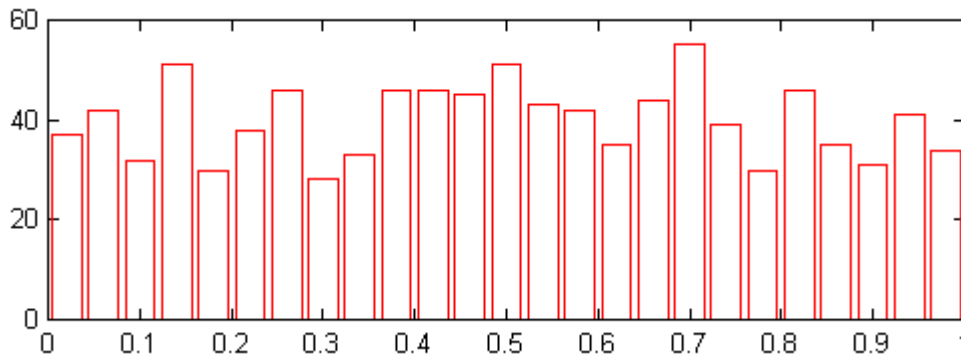


Figura 5.10 - Histograma para números aleatórios uniformemente distribuídos

Suponha agora que queiramos gerar mil números aleatórios distribuídos normalmente (cuja distribuição é $N(0,1)$) e armazená-los em um vetor coluna denominado G_VALORES. Podemos plotar a função com os comandos:

```
rand('normal')
G_VALORES = rand(1000,1);
hist(G_VALORES, 25)
```

O gráfico correspondente é mostrado a seguir. Conforme esperávamos, a distribuição atinge seu valor máximo onde a média é igual a zero, e a maioria dos valores está entre dois desvios padrões (-2 a 2).

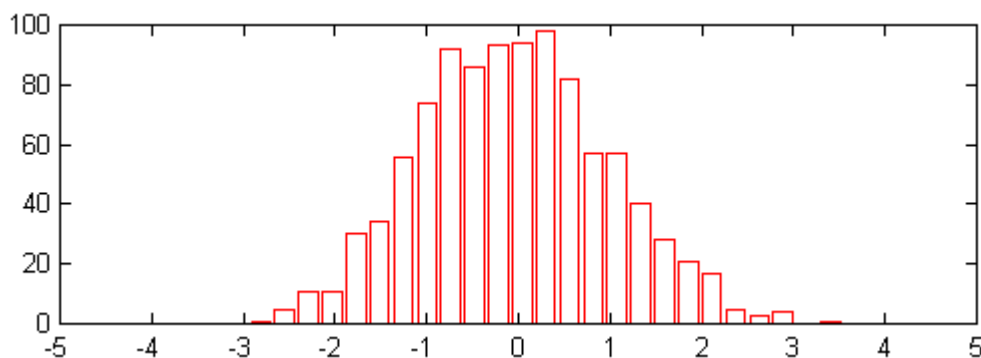


Figura 5.11 - Histograma para números aleatórios normalmente distribuídos

Exercícios

Use o MATLAB para gerar mil valores com as características desejadas. Calcule a média e a variância de mil valores, e compare-as aos valores especificados. Estes valores devem ser semelhantes. Também plote um histograma dos valores, usando 25 barras.

1. Números normalmente distribuídos com $\mu = 5$ e $\sigma^2 = 0,5$;
2. Números normalmente distribuídos com $\mu = -5.5$ e $\sigma = 0,25$;
3. Números normalmente distribuídos com $\mu = -5.5$ e $\sigma = 1,25$;
4. Números normalmente distribuídos com $\mu = \pi$ e $\sigma = \pi/8$;

Aplicação à Solução de Problemas: Simulador de Vôo

As simulações em computadores são usadas para gerar situações que modelam ou imitam uma situação do mundo real. Algumas simulações computacionais são desenvolvidas para jogos como pôquer ou damas. Para jogar, você indica seu movimento e o computador então selecionará uma resposta apropriada. Outros jogos usam a computação gráfica para desenvolver uma interação tal como o uso do teclado ou do mouse para jogar. Nas simulações mais sofisticadas, como um simulador de vôo o computador não somente responde à entrada do usuário mas também gera valores como temperaturas, velocidade do vento, e os locais de outras aeronaves. Os simuladores também simulam emergências que ocorrem durante o vôo de uma aeronave. Se toda esta informação gerada pelo computador for sempre a mesma série de informação, então o valor do simulador seria grandemente reduzido. É importante que estejamos aleatoriamente à geração de dados. As simulações de Monte Carlo usam números aleatórios para gerar valores que modelam os eventos.

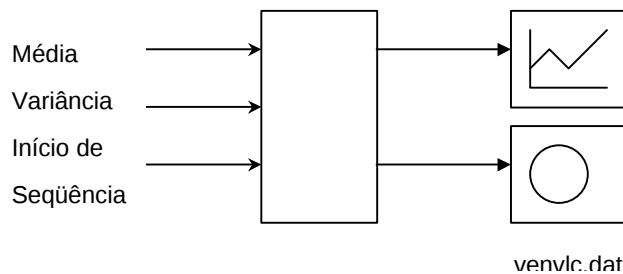
Escreva um programa MATLAB para gerar um seqüência de números aleatórios para simular uma hora de dados de velocidade do vento, o qual é atualizado a cada 10 segundos. (Uma hora de dados é então representada por 361 valores.) De uma análise do padrão real de vento, foi determinado que a velocidade do vento pode ser modelada como um número aleatório distribuído normalmente. A média e a variância são exclusivamente para uma região específica e período do ano e são como parâmetros de entrada. Além do mais, supõe-se que o avião tem 1% de chance de voar sob uma pequena tempestade. A duração do tempo que o avião está em uma pequena tempestade é três minutos. Quando o plano está numa pequena tempestade, a velocidade do vento aumenta em 10 milhas por hora. Também, há 0.01% de chances que o avião voará para um , o qual microrrajadas, o qual permanece por um minuto e aumenta a velocidade do vento para 50 milhas por hora. Plote o tempo e a velocidade e salve-os em um arquivo ASCII nomeado vento_vel.dat.

1. ENUNCIADO DO PROBLEMA

Gerar uma hora de velocidade simulada do vento usando estatísticas desenvolvidas para área de vôo.

2. DESCRIÇÃO ENTRADA/SAÍDA

A figura abaixo, a entrada para o programa é as estatísticas do tempo no plano de vôo, as quais são representadas pela média e variância da velocidade do vento em tempo normal. A saída é o gráfico e o arquivo de dados contendo os valores das velocidades do vento simuladas.



3. EXEMPLO MANUAL

Esta simulação usa várias seqüências diferentes de números aleatórios. A velocidade do vento é um número aleatório normalmente distribuído com média e variância particulares. As possibilidades de encontrar uma tempestade ou microrrajadas são dadas como valores percentuais e podem ser modelados como números uniformemente distribuídos. Suporemos que ocorrerá uma tempestade se um número aleatório uniforme entre 0 e 1 tiver um valor entre 0.0 e 0.01, e que ocorrerá um microrrajadas se o número aleatório uniforme estiver entre 0.01 e 0.0101.

4. SOLUÇÃO MATLAB

5. TESTANDO

5.3 Relação Sinal/Ruído

Quando a geração de sinais é usada em técnicas de testes de engenharia, freqüentemente queremos gerar seqüências como uma senóide com ruído adicionado. De ruído pode ser especificado como uma relação sinal/ruído, ou SNR. A SNR é definida em termos de energia de um sinal. Discutiremos energia de um sinal e então retornaremos a definição matemática de SNR.

Energia de um Sinal

Intuitivamente, a energia é uma medida da amplitude de um sinal. Quanto maiores forem os valores da amplitude, maior é a energia do sinal. Como a amplitude pode ser positiva ou negativa, a energia é definida em termos dos quadrados das amplitudes, para que os valores sejam sempre positivos. A energia em um sinal x (representado por um vetor x) pode ser avaliada pela média quadrática do sinal:

$$\text{energia} \approx \frac{\sum_{k=1}^N x_k^2}{N}$$

Observe que isto pode ser facilmente calculado usando a função MATLAB *sum*:

Energia = sum (x.^2)

pode ser mostrado que a energia em um sinal é igual a soma dos quadrados da variância e média:

$$\text{energia} = \mu^2 + \sigma^2$$

No MATLAB, isto pode ser calculado com o comando a seguir:

Energia = std(x) ^2 + mean (x) ^2;

Se o sinal for uma senóide, pode ser mostrado matematicamente que a energia do sinal é igual a metade do quadrado da amplitude da senóide. Assim, a energia da senóide $4\sin 2\pi t$ é igual a $16/2$, ou 8.

Exercícios

Dê os comandos do MATLAB para gerar mil valores da seqüência indicada. Calcule a energia usando o valor médio quadrático e então calcule-o usando a média e a variância; os valores devem ser semelhantes.

1. Valores uniformes entre 0 e 10;
2. Valores uniformes entre -2 e 4;
3. Valores uniformes com média 0 e variância 1,0;
4. Valores uniformes com média -0.5 e variância 4,0;
5. Valores normalmente com média 0 e variância 2,0;
6. Valores normalmente com média 0 e variância 0,5;
7. Valores normalmente com média -2,5 e variância 0,5.

Cálculo de SNR

Uma relação sinal/ruído é a relação entre a energia em um sinal e a energia em um ruído. Por exemplo, uma SNR de 1 especifica que a relação entre a energia do sinal e a energia do ruído é 1:1. Se tivermos uma senóide com uma amplitude 3 adicionada a um ruído uniforme entre -1 e 1, podemos calcular a SNR usando as medidas de energia de dois sinais. A energia da senóide é 9/2, e a energia do sinal é igual a $2^2/12$ ou 1/3. Contudo, a SNR pode ser calculada como:

$$SNR = \frac{9/2}{1/3} = 13.5$$

Para um sinal geral com amplitude A, e ruído uniforme entre a e b, a SNR pode ser calculada usando os seguintes comandos MATLAB:

$$SNR = ((A^2) / 2) / ((b - a)^2 / 12);$$

Para ilustrar com outro exemplo, suponha que queiramos gerar 201 pontos de um sinal que contenha uma senóide de 1Hz com ruído cuja média é 0 em uma SNR de 46. A senóide deve ter uma amplitude de 1.5 e um ângulo de fase de 0.0, e ser uma amostra de 100 Hz (cuja média é uma amostra de tempo de 1/100 segundos). A SNR é igual a energia da senóide dividida pela energia do ruído.

$$SNR = \frac{\text{energia do sinal}}{\text{energia do ruído}} = \frac{(1.5^2) / 2}{\text{energia do ruído}} = 46$$

Assim, para a energia do ruído, temos:

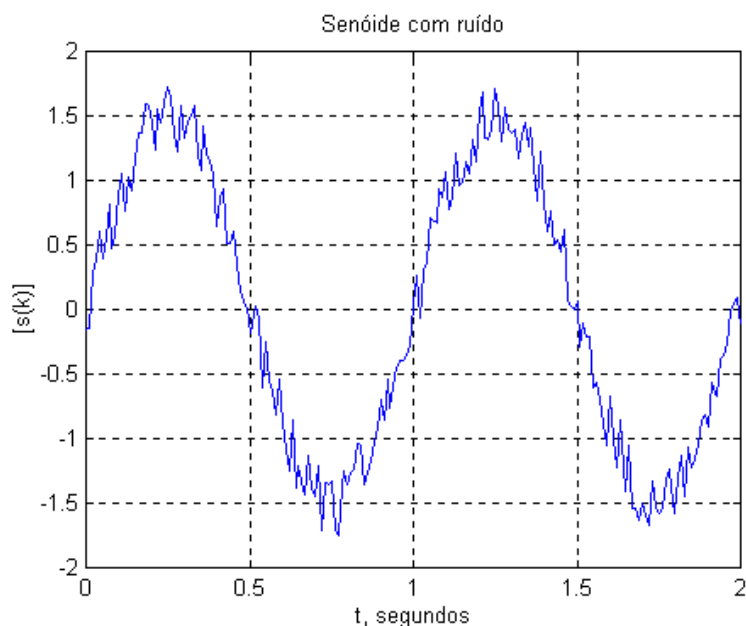
$$\text{Energia do ruído} = \frac{(1.5^2) / 2}{46} = 0.024$$

Como o ruído é especificado para ter média zero, a energia do ruído é igual a variância. Assim, a variância do ruído é 0.024. Já que o ruído é uniforme e tem média zero, ele alcança entre a e -a. Logo, a variância é igual a $(2a)^2/12$ e $0.024 = (2a)^2/12$ ou $a = 0.27$.

Podemos gerar o sinal de ruído desejado e adicioná-lo à senóide para obter a SNR desejada. Os comandos para gerar estes sinal são mostrados a seguir:

```
%
% Gerar e plotar senóide mais ruído
%
rand('seed',0);
rand('uniform');
t = 0: 0.01: 2.0;
s = 1.5*sin(2*pi*t) + (0.54*rand(1,201) - 0.27);
plot(t, s),...
title('Senóide com ruído'),...
xlabel('t, segundos'), ...
ylabel('[s(k)]'), ...
grid
```

O gráfico é mostrado a seguir. Observe que há dois períodos do sinal em dois segundos. Isto corresponde ao fato que a frequência é 1 Hz, de modo que período é 1 segundo.



Adicionando ruído a um sinal existente

Suponha que queiramos adicionar um ruído a um sinal já coletado e armazenado em um arquivo de dados. Se quisermos adicionar ruídos que mantenham uma SNR especificada, precisaremos avaliar a energia do sinal tal que possamos determinar a energia apropriada para o sinal de ruído. Uma boa avaliação da energia de uma sinal é a média quadrática do valor do sinal, o qual pode ser facilmente calculado usando o MATLAB. Podemos determinar a energia necessária para o ruído.

Sabemos que a energia é uma função de média e variância, então precisaríamos de um destes valores especificados em ordem para determinar o outro. É desejável para o ruído ter média zero, para isto supomos que não há outra informação considerável. Podemos calcular a variância necessária e gerar o ruído e adicioná-lo ao sinal existente.

Exercícios

Gerar e plotar um sinal composto de 100 pontos de uma amostra senoidal de 5 Hz a 50 Hz mais um ruído de média zero como especificado a seguir:

1. Ruído uniforme com SNR igual a 5;
2. Ruído uniforme com SNR igual a 1;
3. Ruído uniforme com SNR igual a 0,2;
4. Ruído normal com SNR igual a 5;
5. Ruído normal com SNR igual a 1;
6. Ruído normal com SNR igual a 0,2.

Capítulo 6 – Operações com Matrizes

Uma matriz é um conveniente meio para representar dados experimentais. Nos capítulos anteriores, nós discutimos cálculos matemáticos e funções que poderiam ser aplicadas elemento a elemento presente nas matrizes. Neste capítulo, nós apresentaremos um conjunto de operações e funções que podem ser aplicadas às matrizes como um todo, ao invés de lidarmos com os elementos individualmente. Vamos primeiro considerar um conjunto de operações matemáticas aplicadas às matrizes. E depois vamos considerar um grupo de funções que ajudam na manipulação das matrizes.

6.1 Operações com Matrizes

- Matrizes Transpostas

A transposta de uma matriz é uma nova matriz onde as colunas são formadas pelas linhas da matriz original.

Exemplo 1

$$A = \begin{bmatrix} 2 & 5 & 1 \\ 7 & 3 & 8 \\ 4 & 5 & 21 \\ 16 & 13 & 0 \end{bmatrix}$$

$$A^t = \begin{bmatrix} 2 & 7 & 4 & 16 \\ 5 & 3 & 5 & 13 \\ 1 & 8 & 21 & 0 \end{bmatrix}$$

Podemos notar que o elemento da posição (3,1) foi movido para a posição (1,3). De fato, quando se acha a matriz transposta de uma matriz temos a troca de elementos das posições (i,j) para as posições (j,i).

No MATLAB a matriz transposta é denotada por A'.

Somatório de Produtos

É a soma escalar de dois vetores do mesmo tamanho.

$$\text{Somatório de produtos} = A \cdot B = \sum_{i=1}^N a_i b_i$$

Exemplo 2

$$\begin{aligned}A &= [4 \ -1 \ 3] \text{ e } B = [-2 \ 5 \ 2] \\A \cdot B &= (4) \cdot (-2) + (-1) \cdot (5) + (3) \cdot (2) \\A \cdot B &= (-8) + (-5) + (6) \\A \cdot B &= -7\end{aligned}$$

Comando *sum*

Quando A e B forem ambos vetores linha ou ambos vetores coluna, temos que:
Somatório de produtos = `sum (A .*B)`;

Quando A for um vetor linha e B um vetor coluna, temos que:
Somatório de produtos = `sum (A' .*B)`;

Quando A for um vetor coluna e B um vetor linha, temos que:
Somatório de produtos = `sum (A .*B')`;

Multiplicação de Matrizes

A multiplicação de duas matrizes corresponde ao somatório de produtos das linhas *i* da primeira matriz e das colunas *j* da Segunda matriz. Como o somatório de produtos requer que os vetores tenham o mesmo número de elementos, então o número de colunas de A deve ser igual ao número de linhas de B.

Se A tem 2 linhas e 3 colunas, e B tem 3 linhas e 3 colunas, então o produto A.B terá 2 linhas e 3 colunas.

Exemplo 3

$$A = \begin{bmatrix} 2 & 5 & 1 \\ 0 & 3 & -1 \end{bmatrix} \quad B = \begin{bmatrix} -1 & 0 & 2 \\ -1 & 4 & -2 \\ 5 & 2 & 1 \end{bmatrix}$$

O primeiro elemento do produto $C = A.B$ é

$$(2) \cdot (-1) + (5) \cdot (-1) + (1) \cdot (5) = -2$$

Logo a matriz C será:

$$C = \begin{bmatrix} -2 & 22 & -5 \\ -8 & 10 & -7 \end{bmatrix}$$

Neste exemplo não se pode ter B.A pois o número de colunas de B não é igual ao número de linhas de A.

No MATLAB podem ser usados os seguintes comandos:

```
A = [2 5 1;0 3 -1];  
B = [1 0 2;-1 4 -2;5 2 1];  
C = A * B
```

Matriz Power

É uma matriz quando elevada a um fator. Quando se tem uma matriz quadrada e se deseja calcular $A*A$, usa-se a operação A^2 . Lembrando que A^4 equivale a $A*A*A*A$.

Matriz Inversa

Por definição o inverso de uma matriz quadrada A é a matriz A^{-1} .

Se considerarmos duas matrizes A e B:

$$A = \begin{bmatrix} 2 & 1 \\ 4 & 3 \end{bmatrix} \quad B = \begin{bmatrix} 1.5 & -.5 \\ -2 & 1 \end{bmatrix}$$

Quando calculamos os produtos A.B e B.A e obtemos as matrizes:

$$AB = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad BA = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Temos que as matrizes A e B são inversas, ou seja, $A = B^{-1}$ e $B = A^{-1}$.

No MATLAB, para obtermos uma matriz inversa devemos fornecer a matriz original A e executar o comando `inv(A)`.

Exercícios para Praticar!

Sejam as matrizes:

$$A = \begin{bmatrix} 2 & 1 \\ 0 & -1 \\ 3 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 3 \\ -1 & 5 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 2 \\ -1 & -2 \\ 0 & 2 \end{bmatrix} \quad D = \begin{bmatrix} 1 & 2 \end{bmatrix}$$

Calcule:

1. AB
 2. DB
 3. BC'
 4. $B^{-1}B$
 5. $(AC')^{-1}$
 6. $(AC')^{-1}(AC')$
-

Determinante

Seja a matriz

$$A = \begin{bmatrix} 1 & 3 \\ -1 & 5 \end{bmatrix}$$

O determinante de $A = |A|$ é definido pela expressão:

$$a_{11} \cdot a_{22} - a_{21} \cdot a_{12}$$

No MATLAB, o comando utilizado para se achar o determinante de uma matriz é $\det(A)$.

Aplicação à Solução de Problema: Peso Molecular de Proteínas

A sequência de proteínas é a sofisticada parte do equipamento que executa a função chave em engenharia genética. A sequência pode determinar a ordem de aminoácidos que caracteriza a cadeia de proteínas. Essa ordem de aminoácidos é que auxilia a Engenharia Genética na identificação do tipo de gene da proteína. Enzimas são usadas para dissolver as ligações de genes vizinhos, e assim, separar os genes mais importantes expostos no DNA.

Existem vinte tipos diferentes de aminoácidos. As moléculas de proteínas tem centenas de aminoácidos articulados em uma ordem específica. A sequência de aminoácidos de uma molécula de proteína tem sido identificada e computada pelo peso molecular dos aminoácidos.

O primeiro passo está em arquivar os dados que conterão os números e tipos de moléculas de aminoácidos em cada molécula de proteína.

Assumindo que os dados do arquivo são gerados pelas sequências de aminoácidos, cada linha de dados do arquivo corresponde a uma proteína, portanto, contendo os vinte inteiros correspondentes aos vinte aminoácidos em ordem alfabética como na tabela.

Por esta razão, a linha contém os seguintes valores gerados pela proteína:

Lys Glu Met Asp Ser Glu

00010200000110010000

O nome do arquivo será chamado protein.dat.

1. PROBLEMA EM SI

Calcular o peso molecular de um grupo de moléculas de proteínas.

2. DESCRIÇÃO DA ENTRADA E DA SAÍDA

A figura abaixo contém um diagrama mostrando que a entrada é um arquivo contendo os aminoácidos identificados em um grupo de moléculas de proteínas. A saída do programa são os seus respectivos pesos moleculares.



Diagrama de Entrada e Saída

3. SOLUÇÃO NO MATLAB

```

protein = [0 0 0 1 0 2 0 0 0 0 0 1 1 0 0 1 0 0 0 0;0 1 0 0 0 1 1 0 0 3 0 0 0 0 0 0 0 1 0 0];
pm = [89 175 132 132 121 146 146 75 156 131 131 174 149 165 116 105 119 203 181 117];
pesomol = protein * pm';
  
```

6.2 Manipulações com Matrizes

Comando *rot90*

Uma matriz A pode sofrer uma rotação de 90° usando-se o comando *rot90*.

Exemplo 4

$$A = \begin{bmatrix} 2 & 1 & 0 \\ -2 & 5 & -1 \\ 3 & 4 & 6 \end{bmatrix}$$

$$B = \text{rot90}(A)$$

$$B = \begin{bmatrix} 0 & -1 & 6 \\ 1 & 5 & 4 \\ 2 & -2 & 3 \end{bmatrix}$$

$$C = \text{rot90}(A, 2)$$

$$C = \begin{bmatrix} 6 & 4 & 3 \\ -1 & 5 & -2 \\ 0 & 1 & 2 \end{bmatrix}$$

Comando *fliplr*

Esse comando troca o lado esquerdo com o direito de uma matriz.

Comando *flipud*

Esse comando troca a parte de cima com a parte de baixo de uma matriz.

Exemplo 5

Seja a matriz A :

$$A = \begin{bmatrix} 1 & 2 \\ 4 & 8 \\ -2 & 0 \end{bmatrix}$$

`B = fliplr(A)`

$$B = \begin{bmatrix} 2 & 1 \\ 8 & 4 \\ 0 & -2 \end{bmatrix}$$

`C = flipud(B)`

$$C = \begin{bmatrix} 0 & -2 \\ 8 & 4 \\ 2 & 1 \end{bmatrix}$$

Comando *reshape*

Esse comando reescreve a matriz com diferente número de linhas e colunas.

Exemplo 6

Seja a matriz A:

$$A = \begin{bmatrix} 2 & 5 & 6 & -1 \\ 3 & -2 & 10 & 0 \end{bmatrix}$$

No MATLAB:

`A = [2 5 6 -1; 3 -2 10 0];`

`B = reshape(A,4,2);`

`C = reshape(A,1,8);`

Comando *diag*

Esse comando extrai os elementos da diagonal principal da matriz A e os coloca em um vetor coluna. Desta forma, temos:

$$A = \begin{bmatrix} 3 & 4 & 5 \\ 7 & 6 & 5 \\ 0 & 4 & 3 \end{bmatrix}$$

`B = diag(A)`

$$B = \begin{bmatrix} 3 \\ 6 \\ 3 \end{bmatrix}$$

Se o comando `diag` for aplicado a um vetor ao invés de uma matriz com linhas e colunas, este comando vai gerar uma matriz quadrada cuja diagonal principal será o vetor dado.

Exemplo 7

```
V = [1 2 3];  
A = diag(V)
```

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

Comando *triu*

Este comando trata uma matriz preenchendo com zeros nos lugares dos antigos elementos localizados abaixo da diagonal principal.

Exemplo 8

$$A = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 3 & 6 & 9 & 12 \\ 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```
B = triu(A)
```

$$B = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 0 & 6 & 9 & 12 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

Comando *tril*

É similar ao comando *triu*, porém essa função mantém a matriz da diagonal principal para baixo.

Exemplo 9

$$A = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 3 & 6 & 9 & 12 \\ 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

$$B = \text{tril}(A)$$

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 6 & 0 & 0 \\ 4 & 3 & 2 & 0 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

Aplicação à Solução de Problema: Alinhamento de Imagem

Cada ponto de uma imagem é definido como elemento de figura.

Uma boa resolução de imagem é representada por uma matriz com muitos elementos, enquanto que uma baixa resolução de imagem é representada por uma matriz com poucos.

Por exemplo, uma boa resolução de imagem pode ser representada por uma matriz com 1024 linhas e 1024 colunas, ou um total de mais de milhões de números.

Cada valor de imagem é um código que representa uma determinada intensidade de luz. A intensidade de luz pode ser codificada para representar a cor, ou pode ser codificada para representar a variação de cor cinza.

No exemplo seguinte assumimos que a imagem é representada por uma matriz com 5 linhas e 6 colunas. Assumimos também que cada valor da matriz se encontra de 0 a 7, representando, desta forma, as tonalidades de cinza. O valor 0 representa o branco, o 7 representa o preto e os outros valores representam as devidas tonalidades de cinza. A amostra que estamos tratando é definida pela matriz abaixo:

$$M = \begin{bmatrix} 0 & 0 & 2 & 6 & 2 & 0 \\ 0 & 1 & 0 & 6 & 6 & 0 \\ 1 & 0 & 0 & 2 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Vamos supor que tenhamos duas imagens de um mesmo objeto, de mesma resolução e de um mesmo código de escala de cinza. Só não sabemos se as duas imagens estão alinhadas de um mesmo modo. Para determinar o alinhamento correto nós podemos tomar uma imagem como constante, manipular operações, como rotacionar, para a outra imagem, e então comparar as duas imagens. As imagens estarão alinhadas quando os valores representados nas matrizes forem exatamente os mesmos.

Supondo que:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 2 & 0 & 0 & 2 & 0 \\ 2 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Para alinhar B com A podemos rotacionar B de 270 graus no sentido anti-horário (ou de 90 graus no sentido horário).

Para determinar se as duas imagens possuem os mesmos valores(ou estão alinhadas) observando as diferenças entre os elementos correspondentes nas duas matrizes. Isto pode ser feito utilizando os seguintes comandos no MATLAB:

```
dif = sum (sum (image1 - image2));
```

Infelizmente essa soma pode ser igual a zero mesmo que as matrizes não sejam as mesmas.

Considerando o par de matrizes:

$$A = \begin{bmatrix} 1 & 7 \\ 5 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}$$

```
dif = sum (sum (A -B));
```

```
dif = sum (sum (C));
```

Isto acontece porque os valores se cancelam.

Se tivéssemos valores absolutos isso não ocorreria. Logo, se após a diferença elevarmos a matriz ao quadrado só teríamos valores positivos antes da soma. Podemos fazer isso no MATLAB através dos seguintes comandos:

```
distância = sum (sum (image1 - image2) .^2;
```

Agora as duas imagens estarão alinhadas se a distância for zero.

1. O PROBLEMA EM SI

Determinar a melhor rotação de 90° no alinhamento de duas imagens.

2. DESCRIÇÃO DA ENTRADA E DA SAÍDA

A figura abaixo mostra um diagrama ilustrando que as duas imagens são lidas de dois arquivos e a saída é o melhor alinhamento entre as duas imagens.

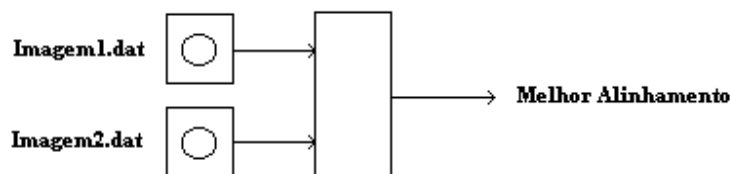


Diagrama de entrada e saída

4. UM EXEMPLO PARA AUXILIAR

Supor as duas matrizes:

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

Se rotacionarmos a matriz B de 0°, 90°, 180° e 270° no sentido anti-horário temos respectivamente:

$$B = \begin{bmatrix} 1 & 3 \\ 3 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 3 & 4 \\ 1 & 3 \end{bmatrix} \quad B = \begin{bmatrix} 4 & 3 \\ 3 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 3 & 1 \\ 4 & 3 \end{bmatrix}$$

Se calcularmos a distância (ou o somatório das diferenças entre dois elementos) C e entre essas quatro versões de D rotacionadas, acharemos os valores 19, 7, 1 e 13 respectivamente. Entretanto a mínima distância é 1, e o alinhamento de 180° é o melhor alinhamento usando a rotação de 90° no sentido anti-horário

5. SOLUÇÃO NO MATLAB

```
load imagem1.dat
load imagem2.dat
for k = 0:3
    a = rot90(imagem2,k);
    distance( k + 1 ) = sum(sum(imagem1 - a) .^2));
end

[minval,minloc] = min(distance);
fprintf('Melhor Alinhamento da Imagem de %3.0f graus \n', (minloc - 1)* 90)
fprintf('(anti-horário)\n')
```

Capítulo 7 – Gráficos

Engenheiros usam gráficos para analisar e resolver problemas e situações. Por isso é muito importante aprendermos a interpretar e gerar gráficos e suas formas. Neste capítulo vamos aprender como o MATLAB pode nos ajudar a gerar gráficos.

7.1 Gráficos X – Y

É muito comum engenheiros e cientistas usarem gráficos x - y. Os dados que nós plotamos são usualmente lidos por um arquivo ou calculados em nossos programas.

Geralmente assumimos que valores de x representam variáveis independentes e que valores de y representam variáveis dependentes. Os valores de y podem ser calculados usando as funções de x, ou os valores de x e y podem ser retirados de experiência.

Coordenadas Retangulares

Os pontos retangulares identificam os pontos no sistema de coordenadas cartesianas com suas posições ao longo dos eixos horizontal e vertical como na figura 7.1.

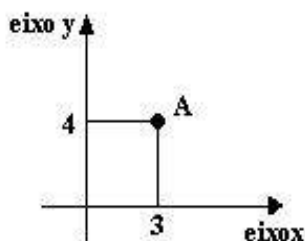


fig7.1 - coordenadas cartesianas

Legenda

Os comandos para se adicionar títulos, linhas de grade e inserir textos estão relacionados a seguir:

Title(text) – Este comando escreve títulos no topo do gráfico plotado.

Xlabel(text) – Este comando escreve um texto abaixo do eixo x do gráfico plotado.

Ylabel(text) – Este comando escreve um texto ao lado do eixo y do gráfico plotado.

Text(x, y, text) – Este comando escreve um texto na tela do gráfico no ponto específico das coordenadas (x, y) usando os eixos dos gráficos. Se x e y são vetores o texto é escrito a cada ponto.

Text(x, y, text, sc) – Este comando escreve um texto na tela do gráfico no ponto especificado pelas coordenadas (x, y), assumindo que a esquina esquerda inferior é (0,0), e a esquina direita superior é (1,1).

gtext(text) – Este comando escreve um texto nas posições indicadas na tela do gráfico pelo mouse.

grid – Este comando acrescenta grades no gráfico plotado.

- *Comandos de plotar*

Geralmente assumimos que y e x são eixos divididos com o mesmo intervalo de espaço. Esses gráficos são chamados de lineares. As vezes temos que usar uma escala logarítmica em um ou ambos os eixos.

Os comandos para plotar gráficos lineares e logarítmico são:

plot(x, y) – Este comando gera gráficos lineares com valores de x e y , onde x representa a variável independente e y representa a variável dependente.

Semilogx(x, y) – Este comando gera gráfico usando escala linear para y e escala logarítmica para x .

Semilogy(x, y) – Este comando gera gráficos usando escala linear para x e escala logarítmica para y .

Loglog(x, y) – Este comando gera gráficos com escala logarítmica para ambos os eixos x e y .

Obs.: É importante lembrar que logaritmo de valores negativos e zero não existem, logo se tentarmos plotar um gráfico semilog ou log com valores negativos ou zeros, aparecerá no MATLAB uma mensagem informando que esses valores serão omitidos do gráfico.

7.2 Gráficos Polares

Gráficos polares são úteis quando valores são representados por ângulo e grandeza (magnitude). Por exemplo se medirmos a intensidade luminosa ao redor de uma fonte de luz, podemos representar a informação com um ângulo fixando eixos e magnitude representando intensidade.

Coordenadas polares

Um ponto é representado em coordenadas polares por um ângulo θ e uma magnitude r . O valor de θ é geralmente dado entre 0 e 2π . A magnitude é um valor positivo que representa a distância do eixo que fornece o ângulo até o ponto.

O comando no MATLAB para gerar gráficos polares é:

polar(theta,r) – Este comando generaliza gráficos polares com ângulo θ (em radiano) e magnitude r correspondente.

Exemplo: Os comando para a construção do gráfico da figura 7.2;

```
theta = 0:2*pi / 100 : 2*pi;
```

```
r = theta / (2*pi);
```

```
polar(theta,r);
```

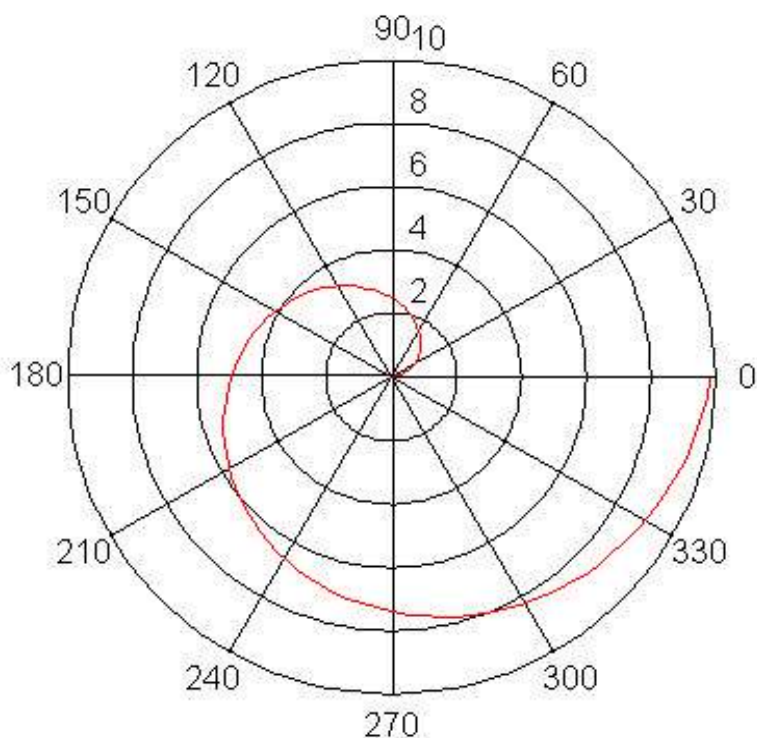


fig7.2 - gráfico polar

Transformação retangular / polar ; polar / retangular

Às vezes devido a praticidade é interessante transformarmos coordenadas de um sistema para outro.

As equações abaixo relacionam os sistemas polar e retangular:

- ✓ polar / retangular _____ $x = r \cos \theta$; $y = r \sin \theta$;
- ✓ retangular / polar _____ $r = \sqrt{x^2 + y^2}$; $\theta = \text{atan}(y/x)$;

Exercícios

1) Converter de coordenada retangular para coordenada polar:

- a) (3, -2);
- b) (0.5, 1);

2) Converter de coordenada polar para coordenada retangular:

- a) $(\pi, 1)$;
- b) (2.3, 0.5);

Gráficos de barras e degraus

Os gráficos são similares, porém as linhas verticais que marcam o eixo x nos gráficos de barras são omitidas nos gráficos de degraus.

Comandos:

bar(x, y) – Este comando gera gráficos de barras com elementos do vetor y localizados no vetor x, contém o mesmo espaço entre os valores.

stairs(y) – Este comando gera um gráfico de degraus com os elementos do vetor y localizados no vetor x, contendo o mesmo espaço entre os valores.

stairs(x,y) – Este comando gera um gráfico de degraus com os elementos do vetor y.

Exemplo: a figura 7.3 mostra um gráfico de barra;

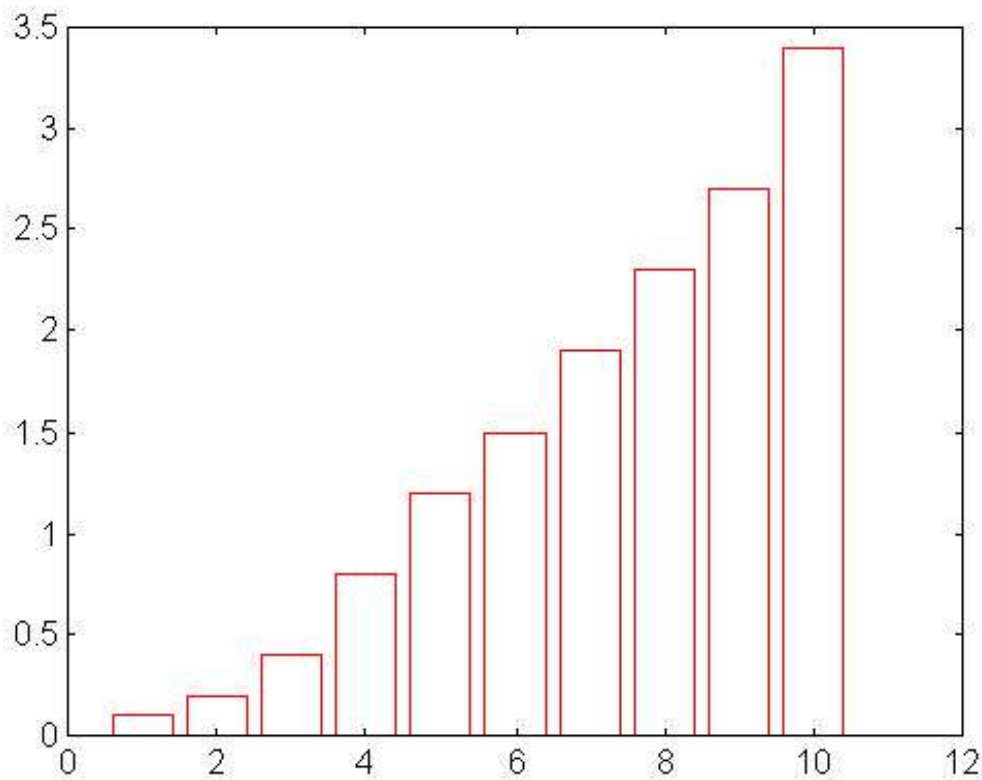


fig.7.3 - gráfico de barra

7.3 Opções

✓ Gráficos múltiplos => Para plotar curvas múltiplas no mesmo gráfico deve se usar vários argumentos no comando plotar como a seguir:

```
plot(x, y, w, z);
```

Quando se executa este comando a curva correspondente a x, y e a curva correspondente a w, z são plotadas no mesmo gráfico. O MATLAB seleciona linhas diferentes para as curvas plotadas.

• Estilo de linha e marcação

O comando `plot(x, y)` nos mostra uma linha plotada representando os vetores y e x, mas podemos selecionar outros tipos de linha. Também podemos selecionar plotar pontos ao invés de linhas. A seguir as diferentes opções de linhas e marcações:

Tipo de linha	Indicador	Tipo de ponto	Indicador
Solid	-	point	.
Dashed	--	plus	+
Dotted	:	star	*
Dashdot	-.	Circle	o
		x-mark	x

O comando a seguir representa linha sólida com tipo de ponto x-mark

```
plot(x, y, x, y, 'x')
```

Podemos também escolher as cores que serão usadas:

Cor	Indicadores
Vermelho	r
verde	g
azul	b
Branco	w
Invisível	i

O comando seguinte representa linha sólida azul para os vetores x, y e plotando pontos vermelhos x-mark:

```
plot(x, y, 'b', x, y, 'xr');
```

• Escala

A escala dos eixos no matlab é automática, porém se você quiser rearrumar a escala de seus eixos você pode usar o comando `axis`. Existe várias formas de se usar o comando `axis`:

axis - Este comando congela a escala de eixos para uma subsequência de gráficos. A Segunda execução do comando retorna o sistema a escala automática.

axis(v)- v é um vetor de quatro elementos que contém a escala de valores,[xmin,xmax,ymin,ymax].

Esses comandos tem um uso especial quando se quer comparar curvas de diferentes gráficos, pôs pode ser difícil a comparação quando as curvas possuem diferentes eixos e escalas.

- *Subplot*

O comando subplot é usado quando se quer visualizar dois ou mais gráficos ao mesmo tempo.

```
Subplot(211), plot(x,y)
Subplot(212), plot(y,x)
```

Esse comando significa que teremos 2 gráficos sendo o primeiro (plot(x,y)) colocado no canto superior esquerdo da tela e o segundo colocado no canto superior direito da tela.

- *Controle de tela*

gcf	_____	Apresenta uma janela com gráfico;
clc	_____	Limpa a janela de comando;
clg	_____	Limpa a janela do gráfico;

Exercício

Gerar 12 pontos de uma função para os valores de x começando de x=0 e incrementando de 0.5; $y = 5x.^2$:

- Gerar o gráfico linear desta função;
- Gerar o gráfico desta função com escala logarítmica x;
- Gerar o gráfico desta função com escala logarítmica y;
- Gerar o gráfico loglog desta função;
- Comparar as vantagens e desvantagens dos gráficos;

Solução:

7.4 Gráficos 3D

A rede de superfície pode ser gerada por um conjunto de valores em uma matriz. Cada ponta na matriz representa o valor da superfície que corresponde ao ponto na tela.

Para gerar um arquivo que representa uma superfície 3D, primeiramente calculamos o conjunto dos valores de x e y que representam as variáveis independentes e depois calculamos os valores de z que representa os valores da superfície. O comando no MATLAB para plotar gráficos 3D é mesh(z). O comando meshgrid tem os argumentos do vetor x e y, ou seja transforma o domínio especificado pelos vetores x e y em vetores que podem ser usados em cálculos de funções de 2 variáveis e construção de gráfico 3D.

Exemplo:

Gerar o gráfico 3D da função $-0.5 < x < 0.5$; $-0.5 < y < 0.5$; (Figura 7.4)

$$f(x,y) = z = 1 - x^2 - y^2$$

$$1 = |x^2 + y^2 + z^2|$$

Solução:

```
[xgrid,ygrid]=meshgrid(-0.5:0.1:0.5;-0.5:0.1:0.5);
```

```
z=sqrt(abs(1 - xgrid.^2 - ygrid.^2));
```

```
mesh(z);
```

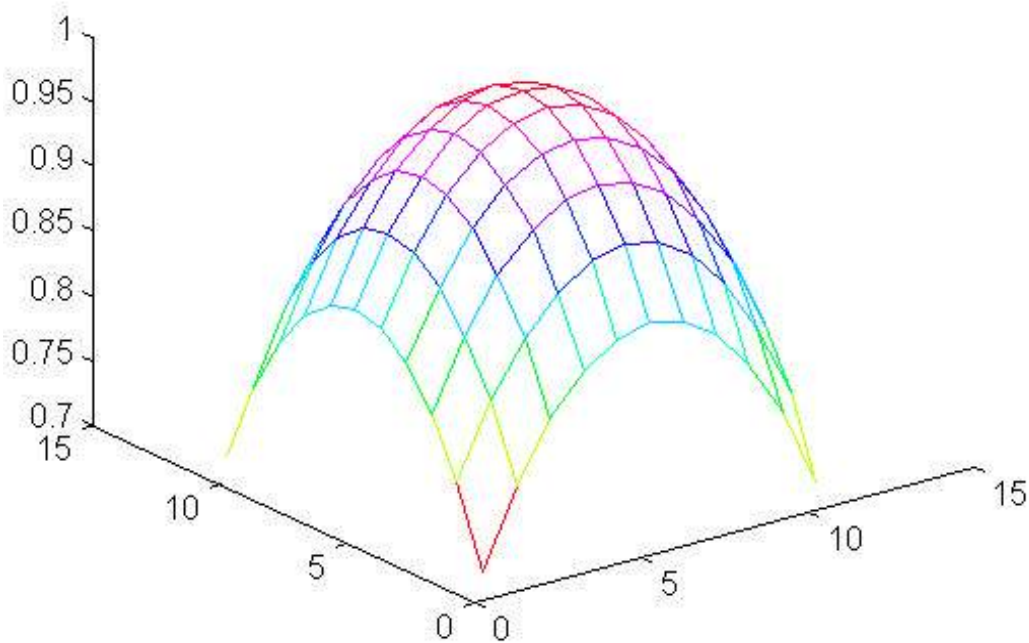


figura 7.4 - Gráfico 3D

Quando geramos redes de superfície 3D podemos querer escolher a posição de visão que será definida com os termos azimuth (rotação horizontal) e vertical elevation que especifica os graus (rotação vertical).

Exemplo:

1) Rotação horizontal (figura 7.5):

Comando no MATLAB :

$$f(x,y) = z = 1 - x^2 - y^2$$

$$1 = |x^2 + y^2 + z^2|$$

Solução:

```
[xgrid,ygrid]=meshgrid(-0.5:0.1:0.5;-0.5:0.1:0.5);
z=sqrt(abs(1 - xgrid.^2 - ygrid.^2));
mesh(z,[-37.5,0]);
```

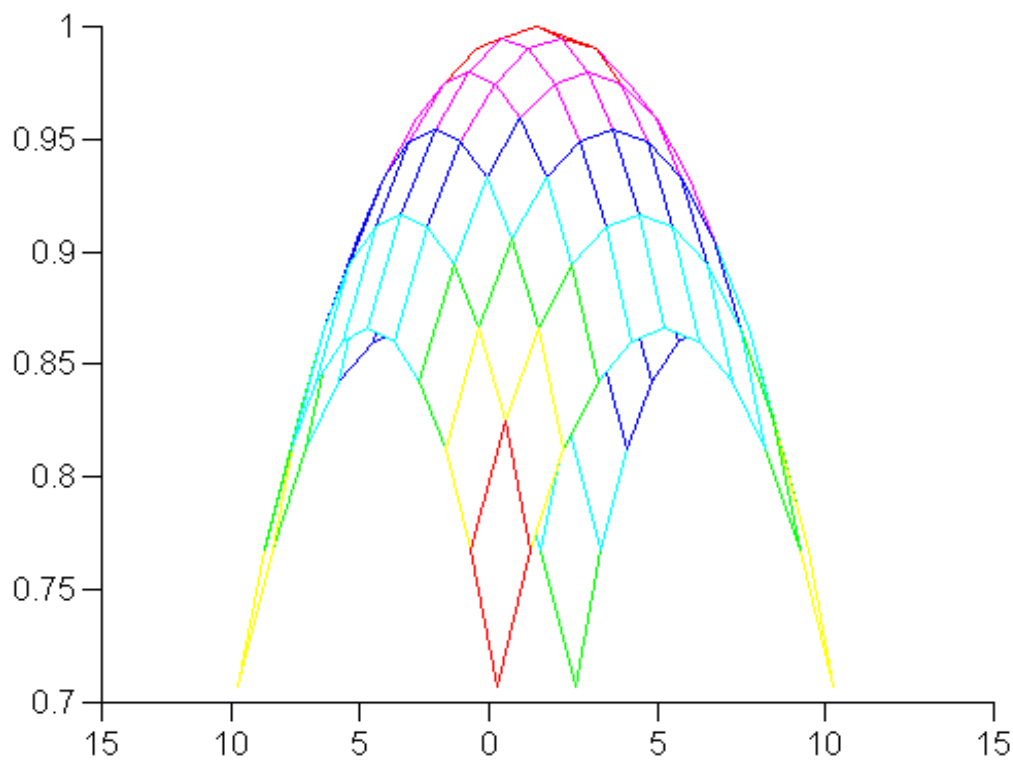


Figura 7.5 - Rotação horizontal

2) Rotação vertical (figura 7.6):

Comando no MATLAB

$$f(x,y)=z=1-x^2-y^2$$

$$1=|x^2+y^2+z^2|$$

Solução:

```
[xgrid,ygrid]=meshgrid(-0.5:0.1:0.5;-0.5:0.1:0.5);
z=sqrt(abs(1 - xgrid.^2 - ygrid.^2));
mesh(z,[-37.5,-30]);
```

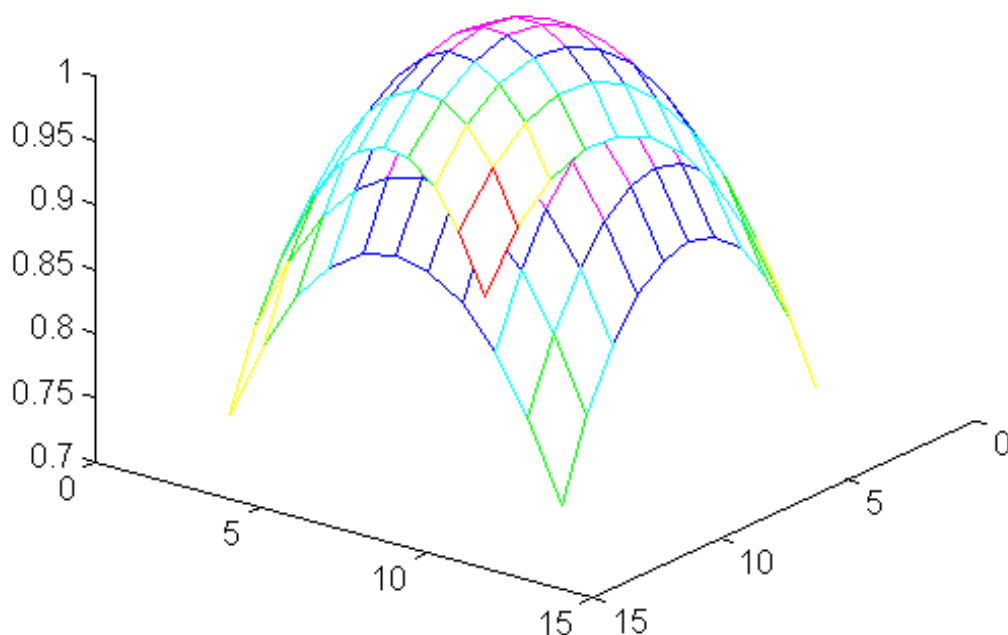


Figura 7.6 - Rotação vertical

Exercício

Gerar o gráfico 3D da função $z=f(x, y)= x*\exp(-x^2-y^2)$
para $-2 < x < 2$, $-2 < y < 2$:

Solução:

Aplicação a solução de problemas: Trajetória de um Satélite

Satélites são usados para investigar diferentes níveis de atmosfera para obter informações semelhantes as que são usadas para monitorar os níveis de ozônio na atmosfera.

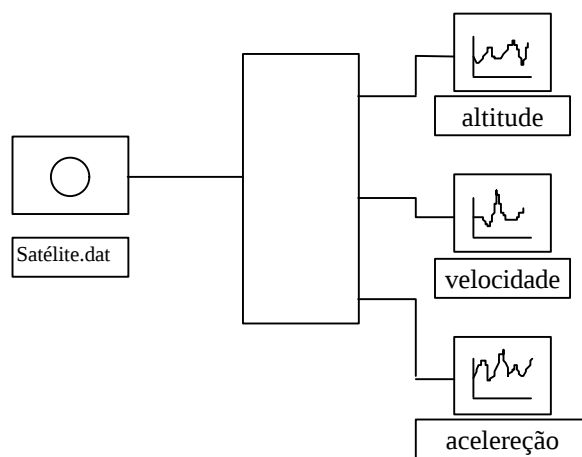
Para aumentar a bagagem científica de coleção de dados da parte mais elevada da atmosfera, os satélites auxiliam sistemas de telemetria para transmissão de informação.

Nessa seção nós assumimos que temos um arquivo contendo altitude, velocidade e aceleração, para um conjunto de dados relativos a uma trajetória de dois estágios do simulador.

1. PROBLEMA EM SI

Queremos gerar gráficos desses arquivos (altitude, velocidade, aceleração) para determinar se a performance dos dois estágios são similares.

2. DESCRIÇÃO ENTRADA / SAÍDA:



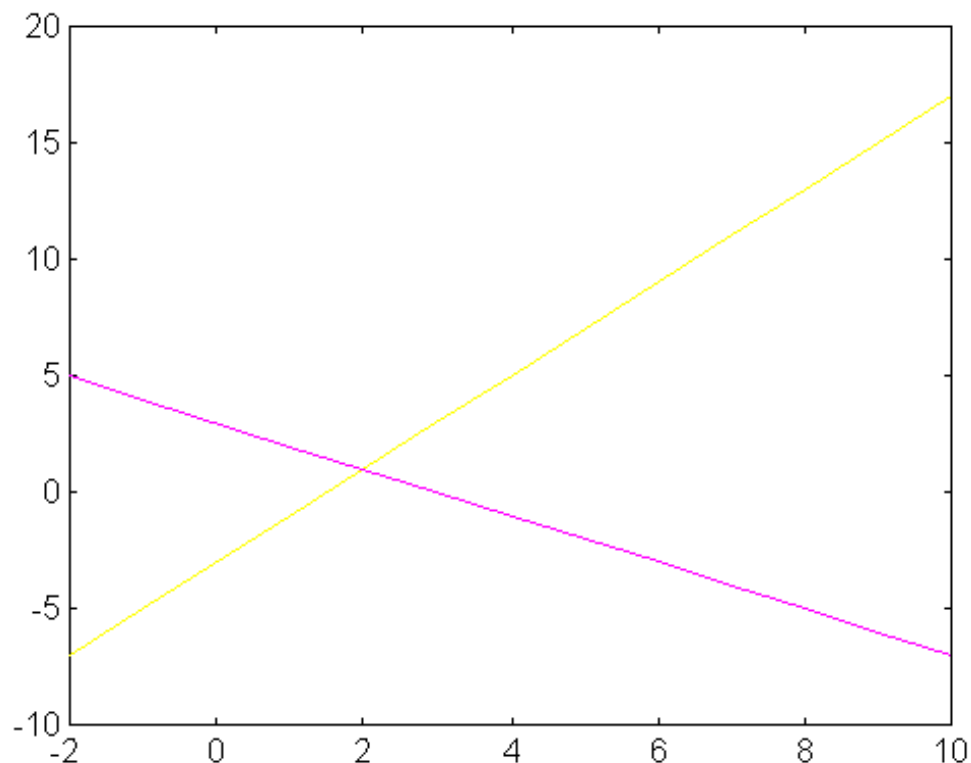
3. SOLUÇÃO MATLAB

Capítulo 8 - Solução a Sistemas de Equações Lineares

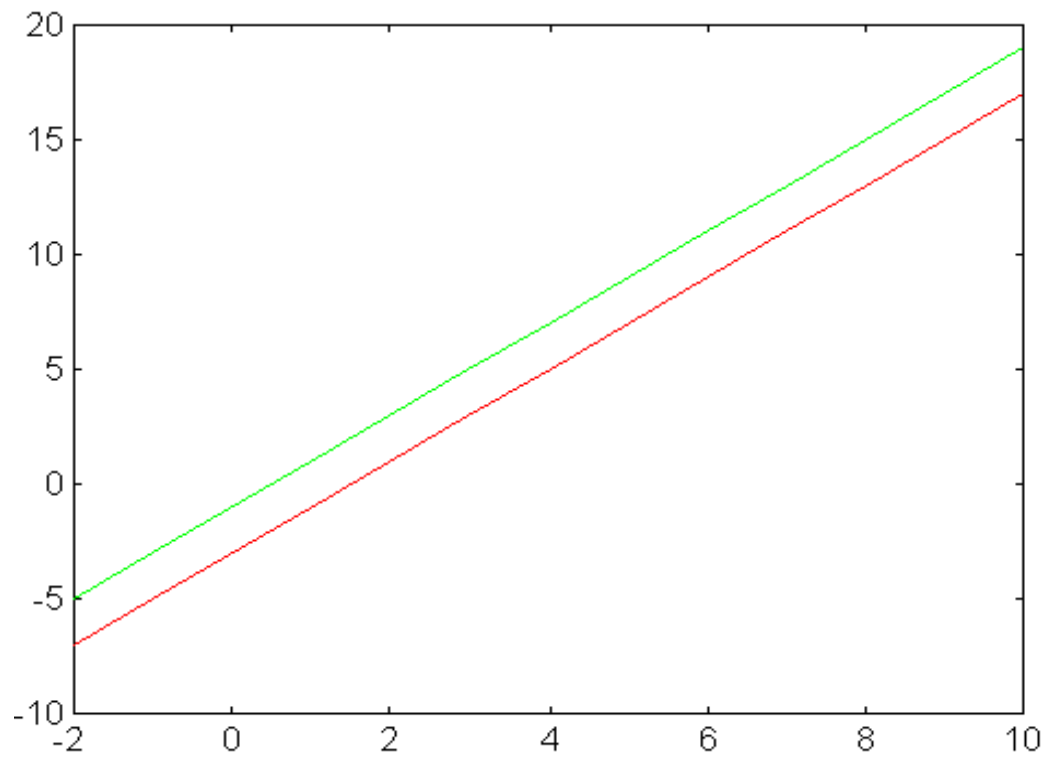
8.1 Interpretação gráfica

A interpretação gráfica é necessária para solução a sistemas de equações lineares ocorrente freqüentemente em problemas de engenharia. A vários métodos existentes para solucionar sistemas de equações, mas eles envolvem operações demoradas com grande oportunidade de erro. Entretanto temos que entender o processo para que possamos corrigir e interpretar os resultados do computador.

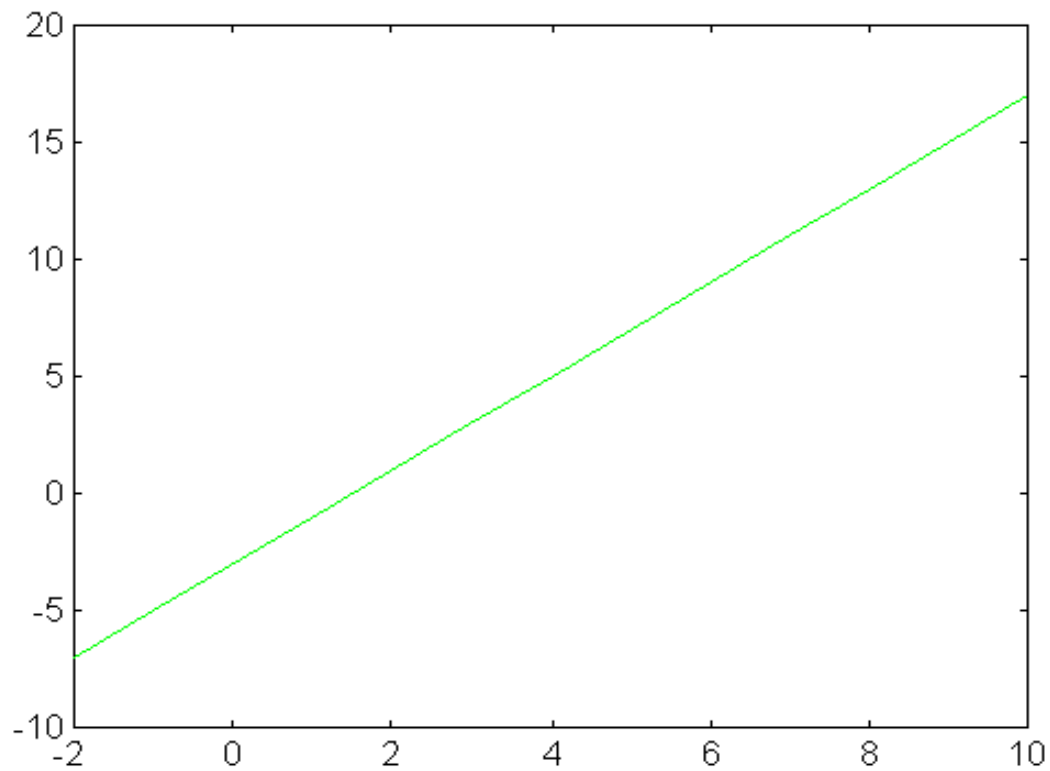
Uma equação linear com 2 variáveis, semelhante a $2x - y = 3$, define uma linha reta e é escrita na forma $y = mx + b$, onde m é o coeficiente angular e b o coeficiente linear. Podemos escrever $y = 2x - 3$. Se tivermos 2 equações lineares, elas podem representar 2 diferentes retas que se interceptam em um mesmo ponto, ou elas podem representar 2 retas paralelas que nunca se interceptam ou ainda podem representar a mesma reta. Estas possibilidades são vistas na figura 8.1.



(a) - Retas que se interceptam.



(b) - Retas paralelas



(c) - Retas iguais - fig8.1

Equações que representam duas retas que se interceptam podem ser facilmente identificadas porque possuem diferentes coeficientes angulares.

Exemplo: $y = 2x - 3$; $y = -x + 3$;

Equações que representam duas retas paralelas possuem o mesmo coeficiente angular e coeficientes lineares diferentes.

Exemplo: $y = 2x - 3$; $y = 2x + 1$;

Equações que representam a mesma reta são equações com mesmo coeficiente angular e mesmo coeficiente linear.

Exemplo: $y = 2x - 3$; $3y = 6x - 9$;

Se a equação linear contém 3 variáveis x , y , z então ela representa um plano em espaço tridimensional.

Se temos duas equações com três variáveis, elas podem representar dois planos que se interceptam em uma linha, ou podem representar dois planos paralelos ou ainda podem representar o mesmo plano.

Essas idéias podem ser estendidas para mais de três variáveis porém se torna difícil a visualização desta situação.

Em muitos sistemas de engenharia estamos interessados em determinar se existe uma solução comum para sistemas de equações. Se a solução comum existe então podemos determiná-la. Vamos discutir dois métodos para solução de sistemas de equação usando MATLAB.

8.2 Solução usando operação com matrizes

Considerando o sistema seguinte de três equações com três variáveis desconhecidas.

$$\begin{bmatrix} 3x & +2y & -z & = & 10 \\ -x & +3y & +2z & = & 5 \\ x & -y & -z & = & -1 \end{bmatrix}$$

Podemos rescrever os sistemas de equações usando as seguintes matrizes:

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

$$x = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$B = \begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}$$

Usando multiplicação de matrizes, o sistemas de equações pode ser escrito na forma:

$$Ax = B$$

Divisão de matrizes

No MATLAB, um sistema de equações simultânea pode ser resolvido usando divisão de matrizes. A solução da equação da matriz $Ax = B$ pode ser calculada usando divisão $A \setminus B$.

Exemplo: $Ax = B$

$$A = [3, 2, -1; -1, 3, 2; 1, -1, -1];$$

$$B = [10; 5; -1];$$

$$x = A \setminus B;$$

O vetor x contém os seguintes valores -2; 5; -6. Para confirmar se os valores de x estão corretos podemos multiplicar $A * x$ e veremos que o resultado será B .

Matriz inversa

O sistema de equações pode ser resolvido usando matriz inversa. Por exemplo assumimos que A , x , B são matrizes definidas a seguir:

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

$$x = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$B = \begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}$$

Então $A * x = B$. Suponha que multiplicamos ambos os lados da equação da matriz por A^{-1} então temos:

$$A^{-1} * A * x = A^{-1} * B$$

Mas $A^{-1} * A$ é igual a matriz identidade I , então temos:

$$I * x = A^{-1} * B \text{ ou} \\ x = A^{-1} * B;$$

No MATLAB podemos calcular essa expressão usando o comando:

$$X = \text{inv}(A) * B;$$

Exercícios

1) Resolver os sistemas de equações com os métodos acima e se possível plotar os gráficos.

$$\text{a) } \begin{bmatrix} -2x & + & y & = & -3 \\ x & + & y & = & 3 \end{bmatrix}$$

b)
$$\begin{bmatrix} -2x + y = -3 \\ -2x + y = 1 \end{bmatrix}$$

Aplicação a solução de problemas: Análise de circuito elétrico

A análise de circuito elétrico frequentemente envolve o encontro de soluções de conjunto de equações. Essas equações são usadas para descrever as correntes que entram e que saem dos nós, ou a voltagem em cada malha.

A figura 8.2 nos mostra um circuito com duas fontes de voltagem. As três equações que descrevem a voltagem ao redor dos três laços são:

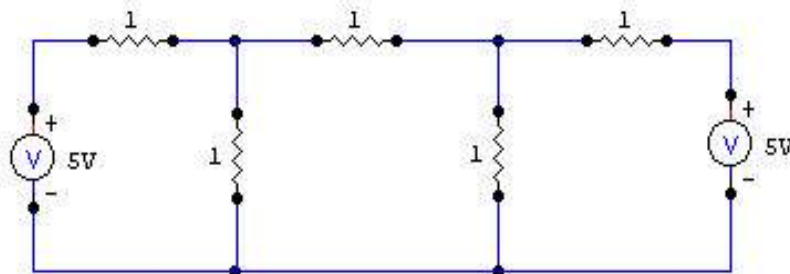
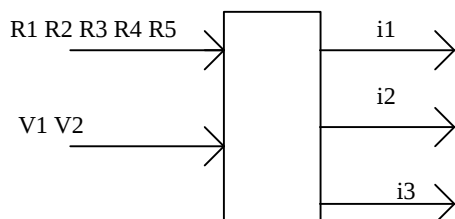


Figura 8.2 - Circuito com duas fontes de voltagem

$$\begin{bmatrix} (R_1+R_2)i_1 & -R_2i_2 & +0i_3 & = & V_1 \\ -R_2i_1 & +(R_2+R_3+R_4)i_2 & -R_4i_3 & = & 0 \\ 0i_1 & -R_4i_2 & +(R_4+R_5)i_3 & = & -V_2 \end{bmatrix}$$

Problema: Calcular as três correntes do circuito da figura 8.2 considerando os valores da figura para R_1 , R_2 , R_3 , R_4 , R_5 , V_1 e V_2 .

Entrada / Saída:



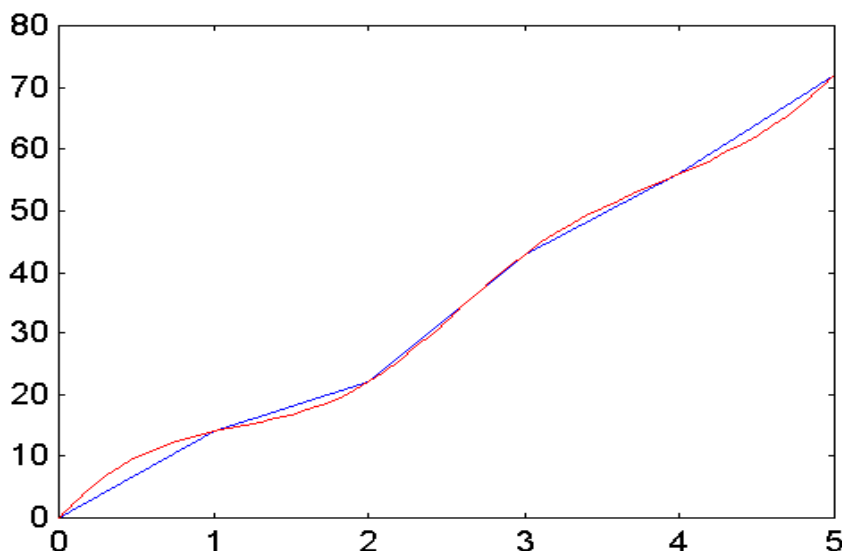
Capítulo 9 - Ajuste de Curvas e Interpolação

Em diversas áreas do conhecimento, com frequência se torna necessário descrever os dados obtidos experimentalmente oriundos de um experimento ou fenômeno físico. Essas informações podem ser tomadas como coordenadas de pontos que definem uma certa função analítica $f(x)$. Podemos ainda usar estes pontos para estimar valores da função que não estejam dentre os iniciais. Outro problema de engenharia é quando não é necessário que a função vá diretamente para todos os pontos dados e sim, para uma estimativa mais apropriada do comportamento da função. Há duas alternativas para resolver este problema. Na 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; o outro método é chamado de ajuste de curvas ou regressão, que tem como objetivo achar alguma curva suave que melhor se ajuste aos dados, mas que não necessariamente passe por quaisquer dos pontos.

9.1 Interpolação

A interpolação é definida como sendo uma forma de estimar os valores de uma função entre aqueles dados por algum conjunto de pontos de dados. A interpolação é uma ferramenta valiosa quando não se pode calcular rapidamente a função nos pontos intermediários desejados. Por exemplo isto ocorre quando os pontos de dados resultam de medições experimentais ou de procedimentos computacionais demorados.

Nesta seção vamos apresentar dois tipos de interpolação. A interpolação linear, que considera que os valores intermediários caem em uma linha reta entre os pontos definidos. Neste método se torna claro que, à medida em que se têm mais pontos de dados e a distância entre eles diminui, a interpolação linear se torna mais precisa. E a interpolação spline, que considera que alguma curva suave se ajusta aos pontos, onde esta suposição é a de que um polinômio de terceira ordem, isto é, um polinômio cúbico seja usado para modelar cada segmento entre pontos consecutivos e que a inclinação de cada polinômio cúbico se ajuste nos pontos de dados.



Interpolação linear

Uma das técnicas mais usadas para estimar o comportamento de uma determinada função entre dois pontos dados é a interpolação linear.

Supondo que tenhamos apenas duas coordenadas de uma função qualquer e, que podemos estimar seu comportamento linearmente, ou seja através de uma reta entre esses pontos. Então poderemos assim determinar o comportamento da função em qualquer ponto deste intervalo por meio de uma simples semelhança de triângulos, onde a equação geral é:

$$f(b) = f(a) + \frac{b - a}{c - a} (f(c) - f(a))$$

A interpolação linear é possível no MATLAB através do uso dos comandos *table1* e *table2*.

Comando table1

Este comando proporciona a interpolação linear em uma dimensão usando para isto uma tabela contendo as informações a serem trabalhadas. O primeiro argumento deste comando se refere à tabela contendo as informações. O segundo se refere ao valor de x para o qual queremos interpolar o valor da função.

O comando irá até a primeira coluna da tabela e achar os dois pontos consecutivos, entre os quais estará o nosso ponto a ser interpolado. O comando então acha o valor da função no ponto escolhido. É importante notar que na hora de alocar os valores na tabela, eles devem estar ordenados crescentemente ou decrescentemente, e o valor a ser interpolado deverá estar entre o primeiro e último valores da primeira coluna da tabela, caso contrário surgirá uma mensagem de erro!

Exemplo 1

Supondo que queiramos determinar o comportamento térmico da cabeça de um cilindro a ser implementado num carro. Supondo também que os valores experimentais referentes ao Tempo e a Temperatura sejam;

Tempo, s	Temp., F
0	0
1	20
2	60
3	68
4	77
5	110

Para alocarmos estas informações devemos usar uma matriz, onde o tempo será preenchido na primeira coluna através dos seguintes comandos:

```
dado1(:,1) = [0,1,2,3,4,5]';  
dado2(:,2) = [0,20,60,68,77,110]';
```

Podemos usar o comando *table1* para interpolar a temperatura correspondente a um determinado tempo no intervalo de 0 a 5 segundos:

```
y1 = table1 (dado1, 2.6);  
y2 = table1 (dado1, 4.9);
```

Os valores correspondentes serão $y1 = 64.8$ e $y2 = 106.7$.

Supondo agora que medimos a temperatura em três pontos do cilindro:

Tempo, s	T1	T2	T3
0	0	0	0
1	20	25	52
2	60	62	90
3	68	67	91
4	77	82	93
5	110	103	96

Guardando estas informações numa matriz, com as informações do tempo na primeira coluna:

```
dado2(:,1) = [ 0,1,2,3,4,5] ' ;  
dado2(:,2) = [0,20,60,68,77,110] ' ;  
dado2(:,3) = [0,25,62,67,82,103] ' ;  
dado2(:,4) = [0,52,90,91,93,96] ' ;
```

Para determinar valores das temperaturas nestes três pontos no tempo de $t = 2.6s$, usamos os seguinte comando:

```
temps = table1 (dado2, 2.6);
```

Onde temps será um vetor contendo os três valores da temperatura: 64.8, 65.0 e 90.6.

Comando table2

Esse comando possibilita a interpolação bidimensional usando valores da primeira coluna e da primeira linha da tabela. É importante perceber que tanto os elementos da primeira coluna quanto os elementos da primeira linha devem estar ordenados crescentemente ou decrescentemente e que os valores de x e de y devem permanecer entre os limites da tabela.

Supomos agora que iniciamos um determinado processo incrementando uma velocidade constante dada em rotações por minuto, enquanto medimos a temperatura em um ponto da cabeça do cilindro. Então, se iniciarmos o processo e incrementarmos uma velocidade 2000 rpm em 5 segundos e registrarmos os valores de temperatura. Da mesma forma podemos continuar registrando os valores de temperaturas para os vários valores de velocidade:

Tempo, s	V1=2000	V2=3000	V3=4000	V4=5000	V5=6000
0	0	0	0	0	0
1	20	110	176	190	240
2	60	180	220	285	327
3	68	240	349	380	428
4	77	310	450	510	620
5	110	405	503	623	785

Desta forma podemos estimar a temperatura da cabeça do cilindro em qualquer tempo entre 0 e 5 segundos, e em qualquer velocidade entre 2000 e 6000 rpm.

Ao invés de calcularmos, o que seria bem mais complicado, podemos interpolar a função em questão.

Podemos agora guardar estas informações numa matriz dado3, e então usar o comando *table2* para calcular esta informação para nós:

Note que agora nós preenchemos as linhas com as informações da tabela, no exemplo anterior nós preenchemos as colunas.

```
dado3(1,:) = [0,2000,3000,4000,5000,6000];  
dado3(2,:) = [0,0,0,0,0,0];  
dado3(3,:) = [1,20,110,176,190,240];  
dado3(4,:) = [2,60,180,220,285,327];  
dado3(5,:) = [3,68,240,349,380,428];  
dado3(6,:) = [4,77,310,450,510,620];  
dado3(7,:) = [5,110,405,503,623,785];  
temp = table2(dado3,3.1,3800)
```

A resposta será mostrada em $\text{temp} = 336.68 \text{ F}$.

- *Spline*

Uma spline cúbica é uma curva suave construída passando através do conjunto de pontos. A curva entre cada par de pontos é determinada por um polinômio do terceiro grau, que é calculado para fornecer uma curva suave entre os pontos ao invés de ligá-los simplesmente.

Comando spline

É o comando que realiza no MATLAB uma spline cúbica. O primeiro argumento do comando *spline* é o x, o segundo é o y e o terceiro contém o valor do(s) ponto(s) aonde se deseja o valor da função. Lembrando que novamente os valores de x devem ser ordenados ou crescentemente ou decrescentemente, caso contrário surgirá uma mensagem de erro!

Exemplo 2

Supondo que queiramos usar a spline cúbica para calcular a temperatura na cabeça do cilindro no tempo $t = 2.6$ segundos, podemos usar os seguintes comandos:

```
x = [0,1,2,3,4,5];  
y = [0,20,60,68,77,110];  
temp1 = spline(x,y,2.6)
```

O valor de temp1 será 67.3.

Se quisermos usar estes processo para calcularmos a temperatura em diferentes momentos podemos usar os seguintes comandos:

```
temp2 = spline(x,y,[2.6,4.9]);  
temp2 = [67.3,105.2]
```

Se quisermos ainda plotar uma curva spline abrangendo um outro intervalo de valores, podemos gerar um vetor x como o terceiro argumento do comando *spline*.

Exemplo 3

```
x = [0,1,2,3,4,5];  
y = [0,20,60,68,77,110];  
newx = 0: 0.1 :5;  
newy = spline(x,y,newx);  
axis([-1,6,-20,120]);  
plot (x,y,newx,newy,x,y,'o');  
title (' Interpolação Spline ');  
xlabel(' Tempo,s ');  
ylabel(' Graus, F ');  
grid;
```

Note que na interpolação linear, o gráfico de x e y percorrem as coordenadas por meio de retas, enquanto que o gráfico de $newx$ e $newy$ representa a spline definida por interpolação cúbica.

Exercícios para Praticar!

Supondo que nossa tabela de valores seja;

Tempo,s	Temp, F
0,0	72,5
0,5	78,1
1,0	86,4
1,5	92,3
2,0	110,6
2,5	111,5
3,0	109,3
3,5	110,2
4,0	110,5
4,5	109,9
5,0	110,2

a. Gerar um gráfico que compare os dois tipos de interpolação já vistos.

- b. Achar os valores da temperatura correspondentes aos seguintes valores de tempo $t = [0.3, 1.25, 2.36, 4.48]$, usando a interpolação linear.
- c. Achar os valores da temperatura correspondentes aos seguintes valores de tempo $t = [0.3, 1.25, 2.36, 4.48]$, usando a spline.

Aplicação à Solução de Problemas : Braço Robótico

Assim como este sistema de manipulação existem vários outros usados em vários tipos de robôs, que se utilizam de um avançado sistema de controle para guiar um braço robótico para a posição desejada. Um dos anseios de um sistema de controle é que o caminho percorrido pelo braço ao se mover de um local para o outro, ao pegar ou soltar um objeto, seja feito regularmente, evitando assim possíveis ‘trancos’ durante o percurso.

O caminho percorrido pelo braço será definido através de coordenadas de pontos por onde o braço irá se mover. Então podemos utilizar a interpolação para definir uma curva suave, regida por estas coordenadas, para mostrar o comportamento desse braço ao longo de uma trajetória.

Uma parte importante no desenvolvimento do algoritmo ou da solução deste problema está na consideração de situações especiais. Neste problema nós assumimos que pontos nos quais o braço irá passar precisarão estar na ordem para mover o braço na trajetória desejada que será: posição inicial, posição intermediária, posição para pegar o objeto, posição para colocar o objeto no local desejado e finalmente posição inicial. E, consideraremos também que cada ponto conterá três coordenadas: x, y (que serão as coordenadas relativas a posição inicial), e uma terceira coordenada dizendo o código da respectiva posição, de acordo com a tabela abaixo:

Código	Posição
0	Inicial
1	Intermediária
2	Para pegar o objeto
3	Para deixar o objeto

Queremos então utilizar uma spline para visualizarmos o comportamento do braço robótico.

Método para a resolução do problema

1. PROBLEMA EM SI

Desenhar uma curva suave utilizando a interpolação por spline que pode ser usada para guiar um braço robótico para uma determinada trajetória.

2. DESCRIÇÃO DA ENTRADA E DA SAÍDA

A entrada é constituída de um arquivo contendo as coordenadas x e y dos pontos pelos quais o braço robótico deverá passar.

A saída do programa será a curva correspondente ao comportamento do robô ao percorrer estes pontos.

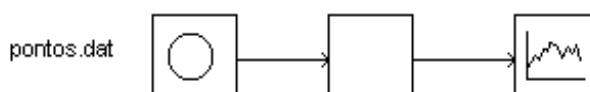


Diagrama de entrada e saída

3. SOLUÇÃO NO MATLAB

9.2 Ajuste de curvas pelo método dos mínimos quadrados

Supondo que tenhamos um conjunto de pontos originados de um determinado experimento e que queiramos plotar o seu gráfico. Se tentarmos traçar uma única reta entre esses pontos, somente um par destes pontos irão fazer parte da reta. O método dos mínimos quadrados poderá ser usado neste caso para achar uma única reta que mais se aproxime de todos os pontos. Embora essa reta seja a melhor aproximação possível, pode acontecer da reta não passar efetivamente por nenhum ponto.

Note que este método é muito diferente da interpolação porque esta passará por todos os pontos.

Vamos partir primeiro para a discussão do ajuste da reta para um conjunto de pontos e depois para o ajuste do polinômio através do conjunto de pontos.

Regressão linear

É o processo que determina a equação linear, ou seja, a função mais aproximada do comportamento dos pontos, que é calculada através do somatório dos mínimos quadrados das distâncias entre a reta e os pontos.

Como exemplo vamos ainda considerar aqueles valores de temperaturas do cilindro:

```
x = [0,1,2,3,4,5];  
y = [0,20,60,68,77,110];  
axis([-1,6,-20,120]);
```

Se simplesmente plotarmos o gráfico através do comando:

```
plot(x,y,x,y, 'o');
```

Ele ligará os pontos. Mas, se ao invés disso, estimarmos o comportamento da função em $y_1 = 20 \cdot x$, e aí sim plotarmos este gráfico:

```
plot(x,y1,x,y, 'o')
```

Para medirmos a qualidade desta estimativa, devemos determinar a distância no eixo vertical de cada ponto à reta estimada e somá-las através do comando `sum`. Observe que somamos os quadrados das distâncias para evitar que algum valor seja anulado devido aos sinais.

```
somadist = sum ((y - y1) .^ 2);
```

Para achar a reta mais perto de todos os pontos devemos achar a menor soma dos quadrados das distâncias. Para isto devemos escrever a equação geral da reta : $y = mx + b$.

Os valores de m e b poderão ser calculados através do comando *polyfit*

Comando polyfit

Este comando acha os coeficientes do polinômio que estamos procurando. Mas, para isto devemos especificar o grau do polinômio. Este comando possui três argumentos: primeiro as coordenadas x e y , e depois o grau do polinômio.

Exemplo:

```
x = [0,1,2,3,4,5];
y = [0,20,60,68,77,110];
coef = polyfit(x,y,1);
m = coef (1);
b = coef (2);
ybest = m*x+b;
somadist = sum ((y - ybest) .^ 2 );
axis([-1,6,-20,120]);
plot(x,ybest,x,y, 'o' );
title (' ')
xlabel ('X'); ylabel('Y');
grid;
```

Comando polyval

Este comando é empregado para estimar o mínimo polinômio quadrado de um conjunto de pontos. O primeiro argumento deste comando conterá os coeficientes do polinômio, o segundo argumento será um vetor com os valores de x para os quais desejamos o valor da função.

Exemplo:

```
ybest = polyval (coef,x);
```

Capítulo 10 - Análise polinomial

Este capítulo traz uma série de comandos no MATLAB para a análise polinomial. Primeiro vamos discutir meios de avaliar os polinômios e como trabalhar o seu comportamento. Uma aplicação deste conceito está na modelagem da altitude e velocidade de um balão. A seguir definiremos as raízes dos polinômios.

Polinômios normalmente aparecem em aplicações da Engenharia e na Ciência em geral porque eles constituem ainda bons modelos para representar sistemas físicos.

10.1 Avaliação do polinômio

Como exemplo vamos tomar o seguinte polinômio:

$$f(x) = 3x^4 - 0.5x^3 + x - 5.2$$

Se x assumir valores escalares, podemos escrever:

$$f(x) = 3*x.^4 - 0.5*x.^3 + x - 5.2;$$

Se x for um vetor ou uma matriz devemos escrever:

$$f(x) = 3* x.^4 - 0.5* x.^3 + x - 5.2;$$

onde o tamanho da matriz f será o mesmo da matriz x .

Comando polyval

Este comando possui dois argumentos. O primeiro argumento contém os coeficientes do polinômio em questão e o segundo argumento contém a matriz para a qual desejamos avaliar o polinômio.

Exemplo 1

```
a = [3,-0.5,0,1,-5.2];  
f = polyval(a,x);
```

Esses comandos também podem ser combinados em um só:

```
f = polyval([3,-0.5,0,1,-5.2],x);
```

O tamanho de f terá que ser igual ao tamanho de x , seja ele um escalar, vetor ou matriz.

Supondo que queiramos o valor da função $g(x) = -x^5 + 3x^3 - 2.5x^2 - 2.5$, para x no intervalo de $[0,5]$:

```
x:0:0.025:5;  
a = [-1,0,3,-2,5,0,-2.5];  
g = polyval(a,x)
```

Quando x for um escalar ou um vetor, *polyval* consegue calcular o valor da função operando elemento por elemento. Mas quando x for uma matriz usa-se o comando *polyvalm*:

```
f = polyvalm(a,x);  
sendo a matriz x, uma matriz quadrada.
```

Operações Aritméticas

Podemos trabalhar com polinômios armazenando seus coeficientes em vetores, e trabalhar apenas com estes vetores.

- Soma e subtração

Para somar ou subtrair polinômios basta somar ou subtrair seus respectivos coeficientes. O MATLAB não apresenta um comando específico par somar polinômios. A soma ou subtração padrão funciona se ambos os vetores polinomiais forem do mesmo tamanho. Somemos os polinômios a seguir:

$$g(x) = x^4 - 3x^2 - x + 2.4$$

$$h(x) = 4x^3 - 2x^2 + 5x - 16$$

$$\text{som}(x) = g(x) + h(x)$$

$$\text{sub}(x) = g(x) - h(x)$$

Para multiplicar um polinômio por um escalar (sendo ele positivo ou negativo), basta definir o polinômio pelos seus coeficientes e efetuar a multiplicação. Multipliquemos o polinômio:

$$g(x) = 3 f(x)$$

No MATLAB:

```
f = [ 3,-6,1];  
g = 3 * f
```

- Multiplicação

A multiplicação polinomial é efetuada por meio do comando *conv* (que faz a convolução entre dois conjuntos). A multiplicação de mais de dois polinômios requer o uso repetido de *conv*.

$$m = \text{conv}(g,h)$$

- Divisão

No MATLAB a divisão de polinômios é feita através do comando *deconv*:

$$[q,r] = \text{deconv}(g,h)$$

Esse resultado nos diz que *g* dividido por *h* nos dá o polinômio de quociente *q* e resto *r*.

Aplicação à Solução de Problemas: Balões Meteorológicos

Balões são usados para reunir problemas de temperatura e pressão nas diferentes altitudes da atmosfera. O balão consegue ganhar altitude porque nele está presente um gás de menor densidade que o próprio ar ao seu redor. Durante o dia, devido a presença da luz solar, o gás Hélio se expande, se tornando mais denso que o ar e assim fazendo com que o balão suba. Durante a noite, o gás Hélio esfria e fica mais denso, e com isso o balão desce a baixa altitude. No dia seguinte o sol novamente esquentando o gás e o balão sobe. Com o passar dos dias, esse processo gera vários valores de altitude que geralmente podem ser aproximados por uma equação polinomial.

Assumindo que o seguinte polinômio represente a altitude em metros, durante as primeiras 48 horas de um balão:

$$h(t) = -0.12 t^4 + 12 t^3 - 380 t^2 + 4100 t + 220$$

onde *t* é medido em horas. O modelo polinomial para velocidade, obtido através da derivada, em metros por hora do balão é o seguinte:

$$v(t) = -0.48 t^3 + 36 t^2 - 760 t + 4100$$

Método para a resolução do problema

1. PROBLEMA EM SI:

Usando o polinômio dado fazer o gráfico da altitude e da velocidade do balão em questão. E achar também a máxima altitude por ele atingida.

2. DIAGRAMA ENTRADA/SAÍDA:

Neste diagrama é mostrado que não existe nenhuma entrada externa ao programa. A saída consiste em dois gráficos e na altitude máxima atingida e o seu tempo correspondente.

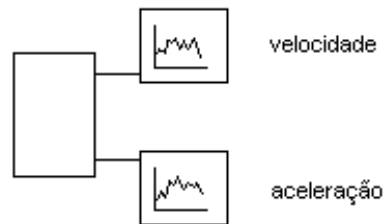


Diagrama de entrada e saída

3. SOBRE O PROGRAMA:

Queremos que apenas o programa faça o gráfico de acordo com as nossas informações e então calcule o máximo valor atingido no gráfico. Devemos também fazer que nosso programa converta metros por hora em metros por segundo.

4. SOLUÇÃO NO MATLAB:

Vamos usar o comando *polyval* para gerar os pontos para formar o gráfico. O comando *max* é usado para determinar o valor máximo da função.

10.2 Raízes de polinômios

Achar as raízes de um polinômio, isto é, os valores para os quais o polinômio é igual a zero, é um problema comum em muitas áreas do conhecimento, como por exemplo, achar as raízes de equações que regem o desempenho de um sistema de controle de um braço robótico, ou ainda equações que demonstram a arrancada ou freada brusca de um carro, ou analisando a resposta de um motor, e analisando a estabilidade de um filtro digital.

Se assumirmos que os coeficientes ($a_1, a_2, \dots$) de um polinômio são valores reais, poderemos encontrar raízes complexas.

Se um polinômio é fatorado em termos lineares, fica fácil de identificar suas raízes, igualando cada termo a zero.

Um exemplo consiste no polinômio:

$$f(x) = x^2 + x - 6,$$

que ao ser fatorado se torna:

$$f(x) = (x - 2) \cdot (x + 3)$$

As raízes da equação são os valores de x para os quais a função $f(x)$ é igual a zero, ou seja, $x = 2$ e $x = -3$.

No gráfico, as raízes são valores onde a função corta o eixo x .

Um polinômio do terceiro grau tem exatamente três raízes que podem ser:

- três raízes reais;
- três raízes iguais;
- uma raiz real e duas raízes iguais;
- uma raiz real e um par conjugado de raízes complexas.

Se a função $f(x)$ for um polinômio de grau n , ela terá exatamente n raízes. Estas n raízes podem conter múltiplas raízes ou raízes complexas.

No MATLAB, um polinômio é representado por um vetor linha dos seus coeficientes em ordem decrescente. Observe que os termos com coeficiente zero têm de ser incluídos. Dada esta forma, as raízes do polinômio são encontradas usando-se o comando *roots* do MATLAB.

Já que tanto um polinômio quanto suas raízes são vetores no MATLAB, o MATLAB adota a convenção de colocar os polinômios como vetores linha e as raízes como vetores coluna. Para ilustrar este comando vamos determinar as raízes do seguinte polinômio:

$$f(x) = x^3 - 2x^2 - 3x + 10$$

No MATLAB:

```
p = [1,-2,-3,10];  
r = roots(p)
```

Lembrando que estes comandos podem ser dados de um só vez:

```
r = roots([1,-2,-3,10]);
```

Os valores das raízes serão: $2 + i$, $2 - i$ e -2 .

Agora, dadas as raízes de um polinômio, também é possível construir o polinômio associado. No MATLAB, o comando *poly* é encarregado de executar essa tarefa.

onde o argumento do comando *poly* é o vetor contendo as raízes do polinômio que desejamos determinar.

Exemplo 2

Sejam as raízes de um polinômio -1 , 1 e 3 . Determinar este polinômio.

No MATLAB:

```
a = poly([-1,1,3]');
```

Exemplo 3

Determine as raízes dos seguintes polinômios e plote seu gráfico, com seu eixo apropriado, com o objetivo de verificar se o polinômio atravessa o eixo x bem nos locais das raízes.

- a. $f(x) = x^3 - 5x^2 + 2x + 8$
- b. $g(x) = x^2 + 4x + 4$
- c. $h(x) = x^5 + 3x^4 - 11x^3 + 27x^2 + 10x - 24$
- d. $i(x) = x^5 - 3x^3 + 4x^2 - 1$

Capítulo 11 - Integração e Diferenciação Numérica

A integração e diferenciação são conceitos fundamentais usados para resolver um grande número de problemas na Engenharia e na Ciência. Enquanto muitos destes problemas se usam de soluções analíticas, muitos requerem soluções numéricas para serem entendidos.

11.1 Integração Numérica

A integral de uma função $f(x)$ no intervalo $[a,b]$, é definida como sendo a área sob a curva percorrida por $f(x)$ entre a e b .

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

A avaliação numérica de uma integral é também chamada de quadratura (enfoque geométrico). O MATLAB possui três comandos para calcular a área sob uma função, em um domínio finito, que são: *trapz*, *quad* e *quad8*.

Regra do Trapézio

Quando a área sob a curva pode ser representada por trapézios e o intervalo $[a,b]$, dividido em n partes iguais, a área aproximada poderá ser calculada através da seguinte fórmula:

$$K_t = \frac{b - a}{2n} (f(x_0) + 2 f(x_1) + \dots + 2f(x_{n-1}) + f(x_n))$$

onde os valores de x_i representam os pontos no final de cada trapézio e $x_0 = a$ e $x_n = b$.

A estimativa da integral melhora quando usarmos um maior número de componentes (como por exemplo trapézios), para aproximar a área sob a curva, pois quanto menor for o intervalo da função a curva tende a uma reta.

Comando quadratura

O MATLAB possui dois comandos para desenvolver a integração numérica. O comando *quad* usa uma forma adaptada da regra de Simpson, enquanto o comando *quad8* usa uma forma adaptada da regra de Newton-Cotes. O comando *quad8* funciona melhor em certas funções com certos tipos de singularidades como por exemplo:

$$k = \int_0^1 \sqrt{x} dx$$

Lembrando que uma singularidade é um ponto no qual uma função ou sua derivada não são definidas ou tendem para o infinito. Ambas as funções escrevem na tela uma mensagem quando detectam uma singularidade, mas ainda assim o valor estimado da integral é retornado.

A forma mais simples do comando *quad* requer três argumentos: o primeiro argumento é o nome da função no MATLAB que reconhece a função que estamos tratando; o segundo e o terceiro argumento são os limites inferior e superior a e b da integral.

Exemplo 1

$$k = \int_a^b \sqrt{x} dx \quad \text{para a e b não negativos}$$

$$K = \frac{2}{3} (b^{3/2} - a^{3/2})$$

Os comandos *quad* e *quad8* podem ainda assumir um quarto argumento que é a tolerância, que corresponde a precisão. Se a tolerância for omitida, o valor default 0.001 será assumido pelo MATLAB.

Exercício 1

Seja a função $f(x) = |x|$. Resolva as integrais abaixo usando o MATLAB e compare com os resultados obtidos a mão.

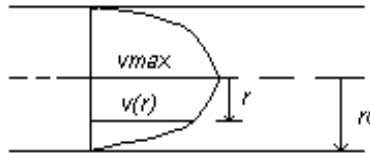
- a. $\text{int}[0.6,0.5] f(x) dx$
- b. $\text{int}[0.5,-0.5] f(x) dx$
- c. $\text{int}[0.0,-1.0] f(x) dx$

Problema Aplicado: Análise de Escoamento de um Óleo num Oleoduto

A análise do fluxo de um líquido em duto tem aplicação em muitos sistemas diferentes, incluindo o estudo em veias e artérias no corpo humano, o sistema hidráulico de uma cidade, o sistema de irrigação de uma fazenda, o sistema de jato de tinta de uma impressora, etc.

O atrito de um fluxo ao passar num oleoduto circular gera a chamada velocidade de perfil no fluido.

O óleo que está em contato com as paredes do duto não está se movendo na mesma velocidade que o óleo no centro do fluido. O diagrama abaixo mostra como a velocidade do óleo varia de acordo com o diâmetro do duto e define as variáveis usadas para esta análise:



A velocidade de perfil é definida pela seguinte equação:

$$v(r) = v_{max} \left(1 - \frac{r}{r_0}\right)^{1/n}$$

onde n é um número inteiro entre 5 e 10 que define o contorno do escoamento do óleo. A velocidade média de escoamento do óleo pode ser calculada integrando-se a velocidade de perfil no intervalo de 0 a r_0 .

$$A = 2 \frac{v_{max}}{r_0^2}$$

$$V = A \int_0^{r_0} r \left(1 - \frac{r}{r_0}\right)^{\left(\frac{1}{n}\right)} dr$$

Os valores de v_{max} e de n podem ser medidos experimentalmente, e o valor de r_0 é o próprio raio do tubo. Escreva um programa no MATLAB para integrar a velocidade de perfil e assim determinar a velocidade média do óleo no duto.

Método Para a Resolução do Problema

1. O PROBLEMA EM SI

Calcular a velocidade média do óleo em um duto.

2. DESCRIÇÃO DA ENTRADA E SAÍDA

Os dados experimentais que serão tomados como entrada em nosso programa são a velocidade máxima v_{max} , o raio do duto r_0 , e o valor de n .

A saída de nosso programa será a velocidade média do óleo no duto.

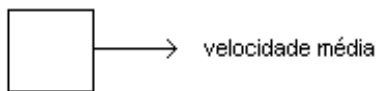


Diagrama de entrada e saída

3. FORMA QUE AJUDARÁ NA QUESTÃO

Plotar um gráfico da função $r(1 - r/r_0)^{1/n}$ e estimar o valor da integral através do cálculo da área sob a curva.

4. SOLUÇÃO NO MATLAB

11.2 Diferenciação Numérica

A derivada de uma função f em um ponto pode ser descrita graficamente como a inclinação da reta que tangencia a função naquele ponto.

Pontos da função onde a derivada é zero são chamados pontos críticos. São pontos onde a tangente é representada por uma linha horizontal e que, por isso, definem o local de máximo e de mínimo da função.

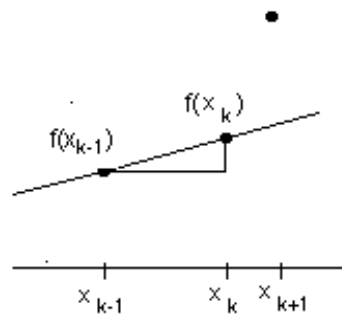
Podemos perceber ao analisar uma determinada função num determinado intervalo que o sinal da derivada pode mudar, e, se esse sinal muda, significa que dentro deste intervalo existe local de máximo e local de mínimo.

Podemos também analisar uma função pela sua derivada segunda. De modo que, se a derivada segunda de um ponto crítico é positiva, então o valor da função naquele ponto significa um local de mínimo. Da mesma forma, se a derivada segunda de um ponto crítico é negativa, então a função possui um local de máximo.

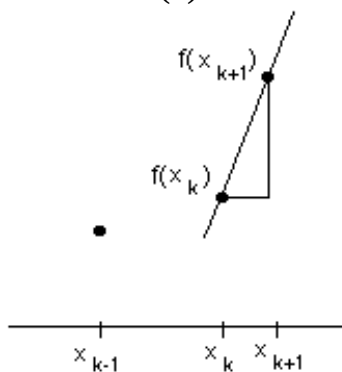
Derivação por expressões de diferenças

As técnicas de diferenciação numérica estimam a derivada de uma função em um ponto x_k através da aproximação da inclinação da reta tangente à curva neste ponto usando valores que a função assume em pontos perto de x_k . Essa aproximação pode ser feita de vários modos.

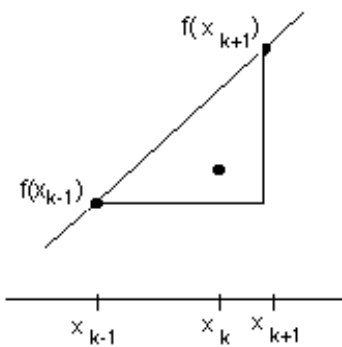
Assim, dependendo dos pontos, temos três técnicas:



(a)



(b)



(c)

A derivada segunda pode ser achada através da fórmula:

$$f''(x_k) = \frac{f'(x_k) - f'(x_k - 1)}{(x_k) - (x_k - 1)}$$

Comando diff

O comando *diff* calcula a diferença entre dois pontos adjacentes num vetor, gerando um novo vetor com a diferença (Se o comando *diff* for aplicado a uma matriz, ele irá operar como se cada coluna da matriz fosse um vetor).

Por exemplo, assumindo que o vetor *x* seja [0,1,2,3,4,5], e que o vetor *y* seja [2,3,1,5,8,10]. O vetor gerado por *diff(x)* será [1,1,1,1,1], enquanto que o gerado por *diff(y)* será [1,-2,4,3,2].

A derivada *dy* será calculada por *diff(y) ./ diff(x)*. Note que estes valores de *dy* estarão corretos para ambas as formas de diferenças, backward ou forward. A diferença entre esses dois métodos para o cálculo da derivada é determinada pelos valores de *x* que correspondem à derivada *dy*. Se os valores correspondentes de *x* forem [1,2,3,4,5] então *dy* é calculado pela diferença backward; mas se os valores de *x* forem [0,1,2,3,4] então *dy* será calculado pelo método da diferença forward.

Supondo que desejamos analisar a função dada pelo seguinte polinômio:

$$f(x) = x^5 - 3x^4 - 11x^3 + 27x^2 + 10x - 24$$

Assumindo que queiramos calcular o valor de sua derivada no intervalo [-4,5], usando o método da diferença backward.

Chamando $f'(x)$ de *df* e, *xd* os valores de *x* da derivada.

Temos no MATLAB que:

```
x = -4:0.1:5;
f = x.^5 - 3 * x.^4 - 11 * x.^3 + 27 * x.^2 + 10 * x - 24;
df = diff(f) ./ diff(x);
xd = x(2:length(x));
plot(f,x)
plot(df,xd)
axis([-4 5 -800 600]);
plot(f)
axis([-4 5 -200 1400]);
plot(df)
```

Podemos marcar os locais dos pontos críticos para essa função com os seguintes comandos:

```
produto = df(1 : length(df) - 1 ) .* df(2 : length(df) );  
critico = xd (find (produto < 0) )
```

O comando *find* determina os índices dos locais do produto para os quais a derivada $df(k)$ é igual a zero; esses índices são então usados com o vetor contendo os valores de xd para determinar os locais de pontos críticos.

Exercícios para Praticar !

1. Para cada polinômio abaixo, plote a função, sua derivada primeira e sua derivada segunda, no intervalo de $[-10,10]$. Depois ache os locais de mínimo, de máximo, e os pontos críticos

- a. $g(x) = x^3 - 5x^2 + 2x + 8$
- b. $h(x) = x^5 - 4x^4 - 9x^3 + 32x^2 + 28x - 48$
- c. $i(x) = x^7 - 5x^3 + 14x^2 - 12$

Capítulo 12 - Equações Diferenciais Ordinárias

Nesta sessão iremos apresentar um grupo de equações de primeira ordem e suas soluções analíticas. Depois seguiremos com a descrição dos métodos de Runge - Kutta para a integração de equações de primeira ordem, onde então iremos comparar as soluções numéricas com as analíticas. Esse capítulo termina com a discussão quando se torna necessário converter equações diferenciais de ordem superiores para equações de primeira ordem.

12.1 Equações Diferenciais Ordinárias de Primeira Ordem

A equação diferencial de primeira ordem (ODE) é uma equação que pode ser escrita na seguinte forma:

$$y' = \frac{dy}{dx} = g(x,y)$$

onde x é a variável independente.

A solução da equação diferencial de primeira ordem (ODE) é a função $y = f(x)$, tal que $f'(x) = g(x,y)$. O cálculo da solução envolve a integração de y' para obter y . A solução de uma equação diferencial é geralmente uma família de funções. A condição inicial é usualmente necessária na ordem para especificar uma única solução. A seguir serão representadas algumas soluções analíticas para equações diferenciais ordinárias.

Enquanto que as soluções analíticas para as equações diferenciais são preferenciais, muitas vezes requerem soluções muito complicadas. Para esses casos, uma técnica numérica se torna necessária. As técnicas numéricas mais comuns para resolver equações diferenciais ordinárias, são o método de Euler e o método de Runge-Kutta.

Tanto o método de Euler quanto o método de Runge-Kutta aproximam a função utilizando-se da expansão em série de Taylor.

Lembrando que a série de Taylor é uma expansão que pode ser usada para aproximar uma função cujas derivadas são definidas no intervalo contendo a e b . A expansão por série de Taylor para $f(b)$ é:

$$f(b) = f(a) + (b - a) f'(a) + \frac{(b - a)^2}{2!} f''(a) + \dots + (b - a)^n f^{(n)}(a) + \dots$$

Para as equações diferenciais de primeira ordem a serie de Taylor se torna:

$$f(b) \approx f(a) + (b - a) f'(a)$$

Para as equações diferenciais de segunda ordem:

$$f(b) \approx f(a) + (b - a) f'(a) + \frac{(b - a)^2}{2!} f''(a)$$

E, assim por diante.

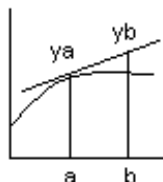
12.2 Método de Runge - Kutta

Os métodos mais populares para a integração da equação diferencial de primeira ordem são os métodos de Runge - Kutta. Esses métodos de aproximação de uma função se usam da expansão por série de Taylor. Desta forma, o método de Runge - Kutta de primeira ordem se utiliza da expansão de Taylor de primeira ordem, o método de Runge - Kutta de segunda ordem se utiliza da expansão de Taylor de segunda ordem, e, assim por diante. Lembrando que o método de Euler é equivalente ao método de Runge - Kutta de primeira ordem.

Método de Euler

$$y_b = y_a + (b - a) y'_a$$

Esta equação estima o valor da função y_b usando uma reta tangente a função no ponto a , conforme mostrado na figura abaixo:

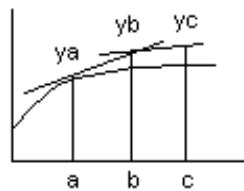


A equação diferencial é usada para calcular o valor de y'_a .

Tendo estimado o valor da função y_b no ponto b , podemos estimar o próximo valor da função y_c , usando:

$$y_c = y_b + (b - a) y'_b$$

Essa equação utilizará a tangente no ponto b para estimar o valor da função no ponto c , y_c , como é mostrado na figura a seguir:



É preciso partir de uma condição inicial para dar início ao processo de estimativa de outros pontos da função $f(x)$.

Comando ode

O MATLAB contém dois comandos para calcular soluções numéricas para equações diferenciais ordinárias: *ode23* e *ode45*; o comando *ode23* usa o método de Runge - Kutta para equações diferenciais de segunda e terceira ordem; o comando *ode45* usa o método de Runge - Kutta para equações diferenciais de quarta e quinta ordem. Os comandos *ode23* e *ode45* possuem os mesmos tipos de argumentos.

A forma mais simples do comando *ode23* requer quatro argumentos. O primeiro argumento é o nome da função, definida no MATLAB, que retorna o valor da equação diferencial $y' = g(x,y)$ quando é fornecido valor para x e y . O segundo e o terceiro argumentos representam os limites no intervalo no qual nós desejamos calcular o valor da função $y = f(x)$. O quarto argumento contém a condição inicial necessária para determinar a única solução para a equação diferencial ordinária. Nós assumimos que esse argumento representa o valor da função dentro do intervalo considerado. O comando *ode23* possui duas saídas: um conjunto de coordenadas x e, um conjunto de coordenadas y correspondentes, os quais representam os pontos da função $y = f(x)$.

No MATLAB, primeiro temos que definir a função a qual desejamos avaliar as equações diferenciais, assumindo valores escalares de entrada para x e y .

Exemplo 1

Resolver a equação $y' = g_1(x,y) = 3x^2$ no intervalo $[2,4]$, assumindo como condição inicial $f(2) = 0,5$.

Solução analítica: $y = x^3 - 7.5$

Solução no MATLAB:

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

```
[x,num_y] = ode23('g1',2,4,0.5)
anl_y^= x.^3 - 7.5;
subplot(211),plot(x,num_y,x,anl_y,'o');
title('Solução do Exemplo 1');
xlabel('X');
ylabel('y = f(x)');
grid;
```

O gráfico obtido conterá a comparação entre a solução numérica e a solução analítica.

Exemplo 2

Resolver a equação $y' = g_2(x,y) = 2x\cos^2 y$ no intervalo $[0,2]$, assumindo como condição inicial $f(0) = \pi/4$.

Solução analítica: $y = \tan^{-1}(x^2 + 1)$

Solução no MATLAB:

```
function dy=g2(x,y)
dy=2*x*cos(y)^2;
[x,num_y] = ode23('g2',0,2,pi/4)
anl_y = atan(x*x+1);
subplot(211),plot(x,num_y,x,anl_y,'o');
title('Solução do Exemplo 2');
xlabel('X');
ylabel('y = f(x)');
grid;
```

O número de pontos calculados para a função $y = f(x)$ pelo comando *ode23* ou *ode45* é determinado pelo MATLAB.

Os comandos *ode23* e *ode45* podem também ser usados com dois parâmetros adicionais. O quinto parâmetro pode ser usado para especificar a tolerância que estará relacionada com o tamanho do passo. O valor default para a tolerância é de 0.001 para o *ode23* e 0.000001 para o *ode45*. O sexto parâmetro pode ser usado para requerer que a função escreva na tela imediatamente os resultados chamado traço. O valor default é zero, especificando nenhum traço para os resultados.

Exercícios para praticar!

1. Seja a equação:

$$y' = g(x,y) = -y$$

- a) Assumindo como condição inicial $f(0) = -3.0$, resolva, no MATLAB, essa equação diferencial no intervalo de $[0,2]$ e plote o gráfico com os valores correspondentes de y .
- b) Sendo $y = -3 e^{-x}$, a solução analítica para esta equação, faça um novo gráfico que compare a solução analítica com a numérica.

Problema Aplicado: Aceleração de uma turbina UDF numa aeronave

Uma avançada turbina chamada de ventilador não canalizado (UDF) é uma das novas tecnologias mais promissoras que tem sido desenvolvida para o futuro transporte de aeronaves. Turbinas, que têm sido usadas por décadas, combinam o poder e a confiabilidade dos motores a jato com a eficiência dos propulsores. Eles constituem uma importante melhoria dos antigos propulsores movidos a pistão. Suas aplicações têm sido limitadas a pequenas aeronaves do tipo comutador, isto porque eles não são tão rápidos, nem poderosos quanto as turbinas usadas em grandes aeronaves. Esse tipo de turbina(UDF) implica em avanços significantes na tecnologia de propulsão. Novos materiais, aerodinâmica e velocidades de alta rotação habilitam esta turbina a voar tão rápido quanto as turbinas a jato, e com grande aproveitamento de combustível. A UDF é também menos barulhenta que o sistema convencional de turbinas.

Durante um teste de vôo de uma turbina UDF de uma aeronave, o motor é levado para um nível de 40 Newton, o que significa os 20 Kg da aeronave tendo alcançado uma velocidade de 180 m/s. As válvulas de regulação do motor são então levadas para atingir um nível de 60 Newtons, e a aeronave começa a acelerar.

A equação diferencial que determina a aceleração da aeronave é:

$$a = \frac{T}{m} - 0,000062 v^2$$

onde:

$$a = \frac{dv}{dt}$$

T = nível atingido em Newtons

m = massa em kg

v = velocidade em m/s

Escreva no MATLAB um programa para determinar a nova velocidade depois de uma mudança no nível do motor através do gráfico da solução para a equação diferencial.

Método para a resolução do problema

1. O PROBLEMA EM SI

Calcular a nova velocidade atingida pela aeronave depois de uma mudança no nível do motor.

2. DESCRIÇÃO DA ENTRADA E DA SAÍDA

Como entrada nós temos a equação diferencial que define a aceleração da aeronave.

Como saída, nós desejamos o gráfico da velocidade e da aceleração.

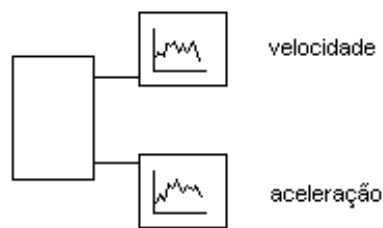


Diagrama de entrada e saída

3. SOLUÇÃO NO MATLAB

Podemos usar o comando `ode23` para avaliar a nossa equação diferencial. A solução dessa equação diferencial nos fornecerá valores de velocidade, os quais poderão ser usados para determinar os valores da aceleração. Nós podemos então traçar ambos os gráficos de velocidade e aceleração num intervalo de 4 minutos para observar suas mudanças. A velocidade deverá aumentar e então estabilizar num novo valor, enquanto que a aceleração deverá diminuir até chegar a zero.

4. PROGRAMA

12.3 Equações Diferenciais Ordinárias de Ordens Superiores

Equações diferenciais de ordens superiores podem ser escritas como um sistema constituído por um conjunto de equações diferenciais de primeira ordem usando a mudança de variáveis.

Exemplo 3

Vamos considerar uma equação diferencial linear de segunda ordem:

$$y'' = g(x, y, y') = y'(1 - y^2) - y$$

Primeiro vamos definir duas novas funções:

$$\begin{aligned} u_1(x) &= y' \\ u_2(x) &= y \end{aligned}$$

Nós então obtemos esse sistema de um conjunto de equações diferenciais de primeira ordem:

$$\begin{aligned} u_1' &= y'' = g(x, u_2, u_1) = u_1(1 - u_2^2) - u_2 \\ u_2' &= u_1 \end{aligned}$$

O sistema contendo as equações diferenciais de primeira ordem pode ser resolvido pelo MATLAB através do comando `ode`. Entretanto, a função que é usada para avaliar a equação diferencial deve calcular os valores das equações diferenciais de primeira ordem em um vetor. A condição inicial deverá também ser um vetor contendo uma condição inicial para cada equação diferencial de primeira ordem: $y^{n-1}, y^{n-2}, \dots, y', y$.

Para resolver as equações desenvolvidas no exemplo anterior, primeiro temos que definir a função para calcular os valores das equações diferenciais de primeira ordem:

```
function u_primo = eqns2(x,u)
u_primo(1) = u(1)*(1 - u(2)^2 - u(2);
u_primo(2) = u(1);
```

Então, para resolver o sistema de equações diferenciais de primeira ordem no intervalo $[0, 20]$ usando as condições iniciais $y'(0) = 0.0$ e $y(0) = 0.25$, podemos seguir os seguintes passos:

```
inicial = [0 0.25];  
[x,num_x] = ode23('eqns2',0,20,inicial);  
subplot(211), plot(x,num_y(:,1))  
title('Primeira Derivada de y');  
xlabel('x');grid;  
subplot(212), plot(x,num_y(:,2))  
title('y');  
xlabel('x');grid;
```

Capítulo 13 - Decomposição e Fatorização de Matrizes

Este capítulo contém algumas das mais avançadas características de matrizes que são utilizadas na resolução de certos tipos de problemas de engenharia. O primeiro tópico, autovalores e autovetores, aparece em inúmeras aplicações. Depois de definir autovalores e autovetores e ilustrar suas propriedades com um exemplo simples, a função *eig* é apresentada para computação usando ambas. Uma aplicação que é utilizada para demonstrar como autovalores e autovetores são utilizados para analisar a performance de algoritmos de adaptadores para redução de ruídos. O resto do capítulo continua com decomposição e fatorização que podem ser aplicados para a matriz A.

13.1 Autovalores e Autovetores

Assuma que A é uma matriz quadrada $n \times n$. Seja X um vetor de uma coluna e “n” linhas e seja λ um escalar. Considere a seguinte equação:

$$AX = \lambda X \quad (13.1)$$

Ambos os lados dessa equação são iguais com uma coluna de vetores com n linhas. Se X é completada com zeros, então esta equação é verdadeira para algum valor de λ , mas esta é uma solução trivial.

Os valores de λ para que X não seja completado com zeros são descritos pelos autovalores da matriz A, e os valores correspondentes de X são descritos pelos autovetores da matriz A.

A equação (13.1) pode ser utilizada para determinar a seguinte equação:

$$(A - \lambda I) X = 0 \quad (13.2)$$

onde I é uma matriz identidade de $n \times n$ elementos. Esta equação representa um conjunto de equações homogêneas enquanto o lado direito da equação for igual a zero. Este conjunto de equações homogêneas possui soluções que não são triviais. A solução só é trivial quando o determinante for igual a zero.

$$\det (A - \lambda I) = 0 \quad (13.3)$$

A equação (13.3) representa uma equação que é referida a equação característica da matriz A. A solução desta equação é obtida com os autovalores da matriz A.

Em muitas aplicações, é desejável selecionar os autovetores como tal $QQ^T = I$, onde Q é uma matriz cujas colunas são os autovetores. Este conjunto de autovetores representa um conjunto ortogonal, enquanto significa que ambos são normalizados e que eles são mutuamente ortogonais. (Um conjunto de vetores é ortonormal se o produto de vetores for igual a unidade, e o produto de um vetor com outro for zero.)

Para ilustrar estas relações entre a matriz A e estes autovalores e autovetores consideremos a matriz A :

$$A = \begin{bmatrix} 0.50 & 0.25 \\ 0.25 & 0.50 \end{bmatrix}$$

Os autovalores podem ser obtidos usando a equação característica:

$$\begin{aligned} \det(A - \lambda I) &= \det \begin{bmatrix} 0.5 - \lambda & 0.25 \\ 0.25 & 0.5 - \lambda \end{bmatrix} \\ &= \lambda^2 - \lambda + 0.1875 \\ &= 0 \end{aligned}$$

Esta equação pode ser facilmente resolvida usando equação quadrática e obtemos $\lambda_0 = 0.25$ e $\lambda_1 = 0.75$. (Se a matriz A tiver mais de 2 linhas e 2 colunas, determinar os autovalores na mão pode se tornar uma formidável tarefa.) Os autovalores podem ser determinados utilizando os autovalores da equação (13.2), usando o valor 0,25:

ou

$$\begin{aligned} \begin{bmatrix} 0.5 - 0.25 & 0.25 \\ 0.25 & 0.5 - 0.25 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{aligned}$$

Mas este par de equações nos dá a seguinte equação:

$$x_1 = -x_2$$

Portanto, existem uma infinidade de autovetores que podem ser associados com o autovalor 0.25. Alguns desses autovetores são demonstrados agora:

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad \begin{bmatrix} 5 \\ -5 \end{bmatrix} \quad \begin{bmatrix} 0.2 \\ -0.2 \end{bmatrix}$$

Similarmente, pode se obter os autovetores de autovalor 0.75, que possui a seguinte relação:

$$x_1 = x_2$$

De novo obtemos uma infinidade de autovetores, como:

$$\begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix} \quad \begin{bmatrix} 5 \\ -5 \end{bmatrix} \quad \begin{bmatrix} 0.2 \\ -0.2 \end{bmatrix}$$

Para determinar um conjunto ortonormal de autovetores para um exemplo simples precisamos lembrar de como selecionar os autovetores, como $QQ^T = I$. Portanto consideremos o seguinte:

$$\begin{aligned} QQ^T &= \begin{bmatrix} c_1 & c_2 \\ -c_1 & c_2 \end{bmatrix} \begin{bmatrix} c_1 & -c_1 \\ c_2 & -c_2 \end{bmatrix} \\ &= \begin{bmatrix} c_1^2 + c_2^2 & -c_1^2 + c_2^2 \\ -c_1^2 + c_2^2 & c_1^2 + c_2^2 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned}$$

Resolvendo o conjunto de equações obtemos:

$$c_1^2 = c_2^2 = 0.5$$

Então $c_1 = c_2 = 0.707$ ou -0.707 . Assim eles possuem várias variações para os mesmos valores, que podem ser utilizados para determinar o conjunto ortonormal de autovetores. Nós escolhemos o seguinte:

$$Q = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

Os cálculos para se obter os autovalores de um conjunto associado de autovetores ortonormais podem ser relativamente simples para uma matriz 2×2 . Entretanto é evidente que fica muito difícil se aumentarmos o tamanho da matriz.

A função *eig* possui um argumento da matriz A . Esta função pode ser usada para retornar um vetor coluna que contenha apenas autovalores, como:

$$\text{lambda} = \text{eig}(A)$$

A função pode também ser usada para executar uma tarefa dupla. Neste caso para retornarmos duas matrizes quadradas: uma contém autovetores (X) como coluna e a outra contém autovalores (λ) na diagonal:

$$[Q,d] = \text{eig}(A)$$

Os valores de Q e d são como $QQ^T = I$ e $AQ = Qd$.

Nós podemos ilustrar a função eig com um exemplo:

$$A = [0.50, 0.25; 0.25, 0.50]$$

$$[Q,d] = \text{eig}(A)$$

Os valores de Q e d são obtidos:

$$Q = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

$$d = \begin{bmatrix} 0.25 & 0.00 \\ 0.00 & 0.75 \end{bmatrix}$$

Podemos facilmente verificar que $QQ^T = I$ e $AQ = Qd$.

Pratique!

Consideremos a matriz A:

$$A = \begin{bmatrix} 4 & 3 & 0 \\ 3 & 6 & 2 \\ 0 & 2 & 4 \end{bmatrix}$$

Use o MATLAB para responder as questões:

1. Determine $\lambda_1, \lambda_2, \lambda_3$, os três autovalores de A
2. Determine um conjunto ortonormal de autovetores, X_1, X_2, X_3
3. Verifique se $\det(A - \lambda I) = 0$ para os autovalores obtidos
4. Demonstre que $AQ = Qd$

Aplicação em Solução de Problemas: Adaptador p/ Redução de Ruído

Este equipamento é utilizado para reduzir o efeito de interferência de ruídos em um sinal. Por exemplo, um microfone que é utilizado para gravar sinais de voz de um grande auditório. Outro microfone é usado na parte de trás do auditório para colher principalmente os sinais de ruído. Através das técnicas para cancelamento de ruídos, as características do sinal de ruído podem ser determinadas usando o sinal de dois microfones. Os adaptadores são utilizados para reduzir o ruído oriundo da parte de trás do auditório, para poderem ser transmitidos para a sala de controle. Este processo resulta num sinal limpo e uma melhor comunicação .

Os algoritmos para os adaptadores estão acima do nível deste texto, mas a performance e a velocidade do algoritmo dependem das características dos sinais de entrada. Estas características determinam a superfície multidimensional quadrática para que obtenhamos um valor mínimo. Este mínimo é determinado ajustando o algoritmo para um ponto de partida para um único mínimo. Se a superfície quadrática tem um contorno circular o algoritmo não é necessário. A matriz R pode ser computada a partir dos sinais de entrada, e os autovalores da matriz R irão determinar o tipo de superfície de contorno a ser utilizada. Se os autovalores forem iguais, a superfície é circular. Quanto maior a variação dos autovalores, mais elíptica será a superfície. Os autovetores representam o eixo principal da superfície. Portanto para determinar a velocidade e a performance do algoritmo do adaptador com certeza do tipo de dados e para analisar a superfície, nós precisamos determinar os autovalores e autovetores da matriz R.

Escreva um programa para ler os valores da matriz com o nome dataR.mat, e depois calcule os autovalores e autovetores.

1. DESCRIÇÃO DO PROBLEMA

Calcule os autovalores e autovetores de uma matriz.

2. DESCRIÇÃO DOS DADOS DE ENTRADA E SAÍDA

A entrada é o arquivo dataR.mat e a saída são os autovalores e autovetores da matriz.

3. EXEMPLO MANUAL

Assumamos que a matriz de entrada seja:

$$R = \begin{bmatrix} 0.50 & 0.25 \\ 0.25 & 0.50 \end{bmatrix}$$

Como no exemplo que tratamos anteriormente sabemos que os autovalores são 0.25 e 0.75. Os autovetores são:

$$V_1 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$$

$$V_2 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

Como os autovalores não são iguais, sabemos que a superfície quadrática não possui um contorno circular e que a performance do adaptador será lenta e menos precisa.

4. SOLUÇÃO MATLAB

Neste programa usaremos um loop para imprimir os autovalores e autovetores.

```
load dataR;
[Q,d] = eig (R)
[m,n] = size(R)
for k = 1: m
    fprintf ( ' Autovalor %4.0f = %7.2f \n', k, d ( k, k ) );
    disp ('Autovetor correspondente')
    disp (Q(:,k) ')
end
```

5. TESTANDO

A saída do programa será:

```
Autovalor 1 =      0.25
Autovetor correspondente
      0.7071   -0.7071
```

```
Autovalor 2 =      0.75
Autovetor correspondente
      0.7071    0.7071
```

Este exemplo é um exemplo simples ao tratar de cancelamento de sinais, pois quando se filtra os sinais usados em comunicação com satélites, a matriz R possui milhares de linhas e colunas.

13.2 DECOMPOSIÇÃO e FATORIZAÇÃO

Nesta seção iremos apresentar três tipos de decomposição e fatorização de matrizes que podem ser utilizados para a solução de problemas que contenham matrizes. Algumas dessas técnicas decompõem a matriz em um produto de outras matrizes. O uso do produto fatorial reduz o número de cálculos necessários para a computação de muitas matrizes. Muitas técnicas numéricas que utilizam matrizes, as convertem em forma de decomposição e fatorização.

Fatorização Triangular

A fatorização triangular expressa uma matriz quadrada como um produto de duas matrizes triangulares – uma matriz inferior superior e uma matriz triangular superior. Esta fatorização é conhecida como fatorização LU (lower-upper).

Esta fatorização é muito usada para simplificar matrizes computacionais. Este é um dos passos para se determinar o determinante de uma matriz muito grande, como também a matriz inversa e para a solução de equações lineares simultâneas.

A fatorização pode ser realizada começando com uma matriz quadrada e uma matriz identidade de mesmo tamanho. Operações de linhas e colunas são realizadas na matriz A para reduzi-la a forma triangular superior; as mesmas operações são realizadas na matriz identidade para transformá-la na forma triangular inferior.

Para ilustrar pegamos as matrizes A e B:

$$A = \begin{bmatrix} 1 & 2 & -1 \\ -2 & -5 & 3 \\ 1 & -3 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 3 & 2 \\ -2 & -6 & 1 \\ 2 & 5 & 7 \end{bmatrix}$$

Usando a fatorização LU obtemos:

$$A = \begin{bmatrix} -0.5 & 1 & 0 \\ 1 & 0 & 0 \\ 0.5 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & -5 & 3 \\ 0 & -0.5 & 0.5 \\ 0 & 0 & -2 \end{bmatrix} \quad B = \begin{bmatrix} -0.5 & 0 & 1 \\ 1 & 0 & 0 \\ -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} -2 & -6 & 1 \\ 0 & -1 & 8 \\ 0 & 0 & 2.5 \end{bmatrix}$$

A função `lu` do MATLAB executa esta fatorização, e é especificada da seguinte forma:

$$[L,U] = \text{lu} (A);$$

O fator inferior é colocado na matriz L e o superior na matriz U. O produto de L e U é igual a A. Veja o exemplo:

$$\begin{aligned} A &= [1, 2, -1; -2, -5, 3; -1, 3, 0]; \\ [LA,UA] &= \text{lu} (A); \\ B &= [1, 3, 2; -2, -6, 1; 2, 5, 7]; \\ [LB,UB] &= \text{lu} (B); \end{aligned}$$

Como resultado obtemos:

$$LA = \begin{bmatrix} -0.5 & 1 & 0 \\ 1 & 0 & 0 \\ 0.5 & 1 & 1 \end{bmatrix} \quad UA = \begin{bmatrix} -2 & -5 & 3 \\ 0 & -0.5 & 0.5 \\ 0 & 0 & -2 \end{bmatrix}$$
$$LB = \begin{bmatrix} -0.5 & 0 & 1 \\ 1 & 0 & 0 \\ -1 & 1 & 0 \end{bmatrix} \quad UB = \begin{bmatrix} -2 & -6 & 1 \\ 0 & -1 & 8 \\ 0 & 0 & 2.5 \end{bmatrix}$$

É facilmente verificável que: $A = (LA)(UA)$ e $B = (LB)(UB)$.

Fatorização QR

Esta técnica de fatorização é feita a partir do produto de uma matriz ortonormal e de uma matriz triangular superior. (Lembrando que uma matriz é ortonormal quando $QQ^T = I$). Não é necessário que a matriz A seja quadrada.

A menor solução para um sistema indeterminado $AX=B$ é a solução de um sistema quadrado $RX=Q^TB$.

A função *qr* do MATLAB executa esta fatorização, e é especificada da seguinte forma:

$$[Q,R] = qr(A)$$

Para uma matriz A $m \times n$, tamanho de Q é $n \times n$, e o tamanho de R é $m \times n$.

Decomposição de valor singular

O SVD (singular value decomposition) é um outro método para a fatorização de matrizes ortogonais. Esta é a decomposição mais confiável, mas isto pode requerer dez vezes mais tempo que a fatorização QR. A decomposição SVD decompõe a matriz num produto dos fatores de outras três matrizes: $A = USV$, onde U e V são matrizes ortogonais e S é diagonal. Os valores da matriz diagonal são chamados de valores singulares e por isso a decomposição recebe este nome. O número de valores singulares diferentes de zero é igual ao rank da matriz.

A função *svd* do MATLAB executa esta fatorização, e é especificada da seguinte forma:

$$[U,S,V] = svd(A)$$

Sumário MATLAB

eig	calcula os autovalores e autovetores de uma matriz
lu	calcula a decomposição LU de uma matriz
or	calcula a decomposição ortonormal de uma matriz
svd	calcula a decomposição SVD de uma matriz

Capítulo 14 – Processamento de Sinais

Este capítulo discute algumas funções que são relacionadas a processamento de sinais. Essas funções foram divididas em quatro categorias: análise do domínio da frequência e domínio do tempo, análise de filtros, implementação de filtros e projeto de filtros.

Embora este capítulo discuta tanto a análise do processamento de sinais analógicos e digitais será dada maior ênfase no processamento de sinais digitais ou DSP.

14.1 Análise no Domínio da Frequência

Recordemos que o sinal analógico é uma função contínua ($f(t)$) que representa uma informação, como por exemplo, um sinal de voz, o sinal da pressão sanguínea e sinais sísmicos. Para o processamento de sinais por um computador, o sinal analógico precisa ser amostrado num período de T segundos, gerando assim, um sinal digital com uma sequência de valores derivados do sinal analógico original. Representamos o sinal digital como um sinal contínuo com a seguinte notação:

$$f_k = f(kT)$$

O sinal digital é uma sequência de amostragens representadas por: $[f_k]$

O tempo que normalmente se escolhe para começar a amostragem é o zero e assim o primeiro intervalo de amostragem é f_0 . Então se o sinal é amostrado com uma frequência de 100 Hz, os primeiros três valores correspondentes ao sinal analógico são:

$$\begin{aligned}f_0 &= f(0T) = f(0.0) \\f_1 &= f(1T) = f(0.01) \\f_2 &= f(2T) = f(0.02)\end{aligned}$$

Estamos acostumados a ver os sinais digitais derivados de analógicos como uma sequência de pontos, mas quando nós plotamos um sinal digital os pontos são conectados por segmentos de linha. Utilizamos o eixo Y para representar $[f_k]$ ou $f(kT)$, que é o sinal digital.

Os índices do MATLAB começam sempre com 1, como $x(1)$, $x(2)$ e assim por diante. Entretanto os índices para processamento de sinais utilizados são sempre valores negativos, como h_{-2} , h_{-1} , h_0 e assim por diante. É importante que as equações sejam escritas na mesma forma para que o usuário não se confunda.

Os sinais podem ser analisados de suas formas – domínio do tempo e domínio da frequência. O domínio do tempo é representado por valores reais e o domínio da frequência por valores complexos, que por sua vez, podem ser representados por senóides, que compõem o sinal.

A transformada discreta de Fourier (DFT) é usada para converter um sinal digital no domínio do tempo em um conjunto de pontos no domínio da frequência. A entrada da transformada é um conjunto de N valores de tempo $[f_k]$: o algoritmo é calculado como um conjunto de valores complexos $[F_k]$ que representam a informação no domínio da frequência.

O algoritmo da transformada utiliza um número de cálculos muito grande, por isso utilizamos a Transformada de Fourier, que também converte o sinal no tempo para o domínio da frequência.

A função *fft* do MATLAB calcula a transformada de Fourier. Esta função pode ser usada para uma ou duas entradas. Se entrarmos com um sinal simples no domínio do tempo obteremos como resposta um sinal contendo números complexos, que representam o domínio da frequência.

Se o número de valores no domínio do tempo for igual a potência de 2, usaremos o método da transformada de Fourier. Se não, usaremos o método DFT.

Os valores no domínio da frequência gerados pela função *fft* correspondem a uma frequência de separação de $1/NT$ Hz, onde N é o número de amostras e T o período da função. Seja $N=32$, $T=0.001$, os valores de frequência mostrados serão 0 Hz, $1/0.032$ Hz e $2/0.032$ Hz e assim por diante.

Consideremos o seguinte exemplo:

```
N = 64;
T = 1/128;
k = 0 : N-1;
f = sin(2*pi*20*k*T);
F = fft(f);
magF = abs(F);
plot(k, magF), title('Magnitude de F(k)'),...
xlabel('k'), ylabel('| F ( k )|'),grid
```

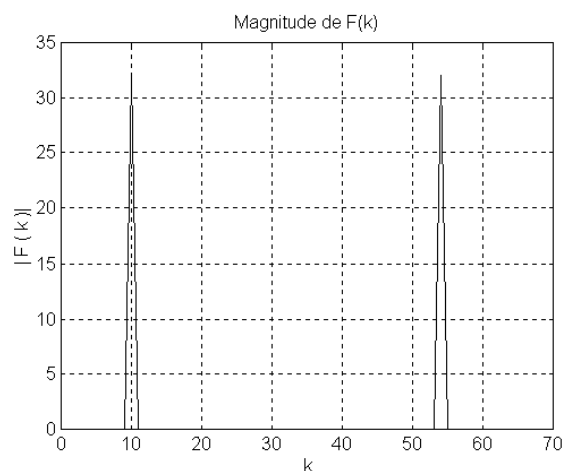


Figura 1 – Magnitude de F_k

Neste gráfico podemos notar a simetria gerada pela periodicidade da FFT; e da onda senoidal.

Podemos plotar também a magnitude de F_K pela frequência em hertz(Hz).

```
hertz = k*(1/(N*T));
plot(hertz(1:N/2),magF(1:N/2)),...
title('Magnitude de F(k)'),...
xlabel ('Hz'), ylabel ('| F ( k )|'),grid
```

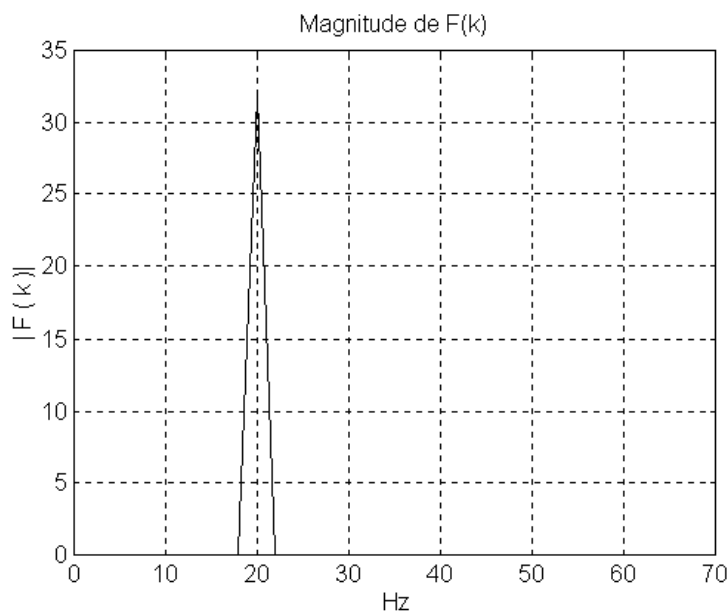


Figura 2 – Magnitude de Fk em hertz

A função *ifft* faz a transformada inversa de Fourier, ou seja, calcula o domínio do tempo $[f_k]$ a partir de valores complexos $[F_K]$.

O método FFT é uma ferramenta poderosa quando se trabalha com sinais digitais. Nossa discussão foi focada na magnitude de F_K , mas também é muito importante obtermos a fase de F_K .

Pratique!

Gere 128 pontos para os seguintes sinais. Plote o sinal no domínio do tempo. Usando o método da transformada de Fourier gere e plote o sinal no domínio da frequência. Use a escala de Hz no eixo X. Assuma a taxa de amostragem de 1KHz. Verifique se os picos ocorrem onde era esperado para o domínio da frequência.

1. $f_k = 2 \sin(2\pi 50kT)$
2. $g_k = \cos(250\pi kT) - \sin(200\pi kT)$
3. $h_k = 5 - \cos(1000kT)$
4. $m_k = 4 \sin(250\pi kT - \pi/4)$

14.2 Análise de Filtros

A função de transferência de um sistema analógico é descrita como $H(s)$ e de um sistema digital como $H(z)$. Estas funções de transferências descrevem o efeito do sistema a um sinal de entrada, e também o efeito de filtragem do sistema.

Como a função de transferência de um filtro define o efeito do filtro em termos de frequência, podemos usar esta função de transferência para descrever uma faixa de frequência. Por exemplo, um filtro passa baixa irá deixar passar todas as frequências abaixo da frequência de corte estabelecida. O passa banda irá deixar passar a banda de frequência especificada. E o corta banda irá remover a banda de frequência especificada.

Podemos definir a atuação dos filtros em 3 regiões básicas: banda de passagem, banda de transição e banda de corte. Estas regiões são definidas pela frequência de corte ω_c e pela frequência de rejeição ω_r .

Como uma função de transferência é uma função complexa, a análise dos filtros incluem gráficos de magnitude e fase. Para isso utilizam-se as funções *abs*, *angle* e *unwrap*. Adicionalmente, as funções *freqs* e *freqz* podem ser usadas para se achar os valores das funções $H(s)$ e $H(z)$ como nos exemplos a seguir.

Função de Transferência Analógica

Um filtro analógico é definido pela função de transferência $H(s)$ onde $s = j\omega$. Na forma geral a função de transferência $H(s)$ é a seguinte:

$$H(s) = \frac{B(s)}{A(s)} \quad (14.1)$$

$$= \frac{b_0 s^n + b_1 s^{n-1} + b_2 s^{n-2} + \dots + b_n}{a_0 s^n + a_1 s^{n-1} + a_2 s^{n-2} + \dots + a_n}$$

Esta função de transferência corresponde a ordem n dos filtros analógicos.

$$H(s) = \frac{0.5279}{s^2 + 1.0275s + 0.5279}$$

$$H(s) = \frac{s^2}{s^2 + 0.1117s + 0.0062}$$

$$H(s) = \frac{1.05s}{s^2 + 1.05s + 0.447}$$

$$H(s) = \frac{s^2 + 2.2359}{s^2 + 2.3511s + 2.2359}$$

A função *freqs* calcula os valores de uma função complexa $H(s)$, usando 3 argumentos de entrada. O primeiro é um vetor contendo os coeficientes do polinômio $B(s)$ da Equação 14.1; o segundo é um vetor contendo os coeficientes do polinômio $A(s)$; e o terceiro é um vetor com o valor da frequência em 'rps'. Em geral, colocamos o alcance da frequência para começar no zero e incluimos todos os parâmetros do filtro.

Programa:

```
W1 = 0:0.05:5.0;
B1 = [0.5279];
A1 = [1,1.0275,0.5279];
H1s = freqs(B1,A1,W1);

W2 = 0:0.001:0.3;
B2 = [1,0,0];
A2 = [1,0.1117,0.0062];
H2s = freqs(B2,A2,W2);

W3 = 0:0.01:10;
B3 = [1.05,0];
A3 = [1,1.05,0.447];
H3s = freqs(B3,A3,W3);

W4 = 0:0.005:5;
B4 = [1,0,2.2359];
A4 = [1,2.3511,2.2359];
H4s = freqs(B4,A4,W4);
clg
subplot (221),plot(W1,abs(h1s)),title('Filtro H1(s)'),...
xlabel ('w,rps'), y;label('Magnitude'), grid
subplot (222),plot(W2,abs(h2s)),title('Filtro H2(s)'),...
xlabel ('w,rps'), y;label('Magnitude'), grid
subplot (223),plot(W3,abs(h3s)),title('Filtro H3(s)'),...
xlabel ('w,rps'), y;label('Magnitude'), grid
subplot (224),plot(W4,abs(h4s)),title('Filtro H4(s)'),...
xlabel ('w,rps'), y;label('Magnitude'), grid
```

A fase de um filtro pode ser obtida utilizando-se a função *angle* ou a *unwrap*. Como a fase de um número complexo é um ângulo em radianos, o ângulo está restrito ao intervalo 2π . Esta função, utiliza o intervalo de $-\pi$ a π e quando estes limites são ultrapassados é gerada uma descontinuidade.

A função *unwrap* remove as descontinuidades quando introduzimos a função *angle*. Como no exemplo: *unwrap(angle(H))*.

Função de Transferência Digital

A função de transferência digital é definida por $H(z)$ onde $z = e^{j\omega t}$. Na forma geral a função de transferência $H(z)$ é a seguinte:

$$H(z) = \frac{B(z)}{A(z)} \quad (14.2)$$

$$H(s) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots + b_n z^{-n}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \dots + a_n z^{-n}}$$

Esta função de transferência corresponde a ordem n dos filtros digitais.

$$H_1(z) = \frac{0.2066 + 0.4131z^{-1} + 0.2066z^{-2}}{1 - 0.3695z^{-1} + 0.1958z^{-2}}$$

$$H_2(z) = \frac{0.894 - 1.789z^{-1} + 0.894z^{-2}}{1 - 1.778z^{-1} + 0.799z^{-2}}$$

$$H_3(z) = \frac{0.42 - 0.42z^{-2}}{1 - 0.443z^{-1} + 0.159z^{-2}}$$

$$H_4(z) = \frac{0.5792 + 0.4425z^{-1} + 0.5792z^{-2}}{1 + 0.4425z^{-1} + 0.1584z^{-2}}$$

Se o denominador da função de transferência for igual a 1, o filtro é um FIR(resposta finita ao impulso) e se for diferente de 1 o filtro é um IIR(resposta infinita ao impulso). Ambos são muito utilizados no processamento de sinais digitais.

A função *freqz* calcula os valores de uma função complexa $H(z)$, usando 3 argumentos de entrada. O primeiro é um vetor contendo os coeficientes do polinômio $B(z)$ da Equação 14.2; o segundo é um vetor contendo os coeficientes do polinômio $A(z)$; e o terceiro é para especificar o número de valores de frequências normalizadas que se quer no intervalo de 0 a π . Este número de pontos define a resolução. Através deste ajuste da resolução podemos determinar os tipos de filtros e as frequências críticas.

Programa:

```
B1 = [0.2066,0.4131,0.2066];  
A1 = [1,-0.3695,0.1958];  
[H1z,w1T] = freqz(B1,A1,100);
```

```
B2 = [0.894,-1.789,0.894];  
A2 = [1,-1.778,0.799];  
[H2z,w2T] = freqz(B2,A2,100);
```

```
B3 = [0.42,0,-0.42];  
A3 = [1,-0.443,0.159];  
[H3z,w3T] = freqz(B3,A3,100);
```

```
B4 = [0.5792,0.4425,0.5792];  
A4 = [1,0.4425,0.1584];  
[H4z,w4T] = freqz(B4,A4,100);  
clf
```

```
subplot (221),plot(w1T,abs(H1z)),title('Fitro H1(z)'),...  
xlabel ('w,rps'), y;label('Magnitude'), grid  
subplot (222),plot(w2T,abs(H2z)),title('Fitro H2(z)'),...  
xlabel ('w,rps'), y;label('Magnitude'), grid  
subplot (223),plot(w3T,abs(H3z)),title('Fitro H3(z)'),...  
xlabel ('w,rps'), y;label('Magnitude'), grid  
subplot (224),plot(w4T,abs(H4z)),title('Fitro H4(z)'),...  
xlabel ('w,rps'), y;label('Magnitude'), grid
```

A fase de um filtro digital pode ser plotada usando-se a função `angle` ou a `unwrap`.

Pratique !

Para estas funções de transferência, plote a magnitude. Use frequência normalizada no eixo X para filtros digitais.

$$1. H(s) = \frac{s^2}{s^2 + \sqrt{2}s + 1}$$

$$2. H(s) = \frac{0.707z - 0.707}{z - 0.414}$$

$$3. H(s) = -0.163 - 0.058z^{-1} + 0.116z^{-2} + 0.2z^{-3} + 0.116z^{-4} - 0.058z^{-5} - 0.163z^{-6}$$

$$4. H(s) = \frac{5s + 1}{s^2 + 0.4s + 1}$$

14.3 Implementação de Filtros Digitais

Filtros analógicos são implementados com componentes eletrônicos como resistores e capacitores. Os filtros digitais são implementados por software. Os filtros digitais podem ser representados por funções de transferência ou equações diferenciais.

A relação entre o sinal de entrada x_n e o sinal de saída y_n é descrita pela equação diferencial, que está escrita na forma geral:

$$y_n = \sum_{k=-N_1}^{N_2} b_k x_{n-k} - \sum_{k=1}^{N_3} a_k y_{n-k}$$

Exemplos:

$$y_n = 0.04x_{n-1} + 0.17x_{n-2} + 0.25x_{n-3} + 0.17x_{n-4} + 0.04x_{n-5}$$

$$y_n = 0.42x_n - 0.42x_{n-2} + 0.44x_{n-1} - 0.16x_{n-2}$$

$$y_n = 0.33x_{n+1} + 0.33x_n + 0.33x_{n-1}$$

As três equações diferenciais representam diferentes tipos de filtros. A saída do primeiro depende somente dos valores anteriores do sinal de entrada. O segundo filtro requer valores não apenas de entrada mas também os valores anteriores da saída. O terceiro filtro depende apenas dos valores de entrada. Porém os valores de entrada necessários são os posteriores e isso pode ser problemático quando os dados são adquiridos em tempo real. Estes filtros são FIR porque os denominadores são iguais a 1.

A função *filter* do MATLAB assume a equação diferencial na seguinte forma:

$$y_n = \sum_{k=0}^{N_2} b_k x_{n-k} - \sum_{k=1}^{N_3} a_k y_{n-k}$$

que corresponde a seguinte função de transferência:

$$H(z) = \frac{B(z)}{A(z)}$$

$$H(s) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots + b_n z^{-n}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \dots + a_n z^{-n}}$$

Os dois primeiros argumentos da função `filter` são vetores com os coeficientes de $[b_k]$ e de $[a_k]$. O terceiro argumento é o sinal de entrada.

Exemplo1:

```
B = [0.0,0.04,0.17,0.25,0.17,0.04];  
A = [1];  
Y = filter(B,A,x);
```

Exemplo2:

```
B = [0.42,0.0,-0.42];  
A = [-0.44,0.16];;  
Y = filter(B,A,x);
```

Nós podemos usar a função `filter` para o terceiro filtro porque a equação não está ajustada na forma geral usada pela função. A terceira equação precisa começar em $k=0$ e não em $k=-1$. Neste caso implementamos o filtro utilizando um vetor aritmético. Assumindo um sinal de entrada x constrói-se um vetor x , que pode calcular o sinal de saída correspondente usando a seguinte forma:

```
N = length(x);  
y(1) = 0.33*x(1) + 0.33*x(2)  
for n=2:N-1  
    y(n) = 0.33*x(n+1) + 0.33*x(n) + 0.33*x(n-1);  
end  
y(N) = 0.33*x(N-1) + 0.33*x(N);
```

Uma outra forma é a seguinte:

```
N = length(x);  
y(1) = 0.33*x(1) + 0.33*x(2);  
y(2:N-1) = 0.33*x(3:N) + 0.33*x(2:N-1) + 0.33*x(1:N-2);  
y(N) = 0.33*x(N-1) + 0.33*x(N);
```

Pratique !

A seguinte função de transferência foi projetada para deixar passar frequências entre 500Hz e 1500Hz, de um sinal de até 5KHz.

$$H(z) = \frac{0.42z^2 - 0.42}{z^2 - 0.443z + 0.159}.$$

Use os seguintes sinais de entrada no filtro. Plote a entrada e a saída do filtro no mesmo gráfico e determine o efeito do filtro no sinal de entrada.

1. $x_k = \sin(2\pi 1000kT)$
2. $x_k = 2 \cos(2\pi 100kT)$
3. $x_k = -\sin(2\pi 2000kT)$
4. $x_k = \cos(2\pi 1600kT)$

14.4 Projeto de Filtros Digitais

A discussão será separada em duas técnicas: uma para filtros IIR e outra para filtros FIR.

Projeto de Filtros IIR usando protocolos analógicos

O MATLAB possui quatro tipos de filtros digitais baseados em projetos de filtros analógicos. Os filtros Butterworth são usados como passa-banda e corta-banda, os filtros Chebyshev Tipo I são usados com ripple na banda de passagem, os filtros Chebyshev Tipo II são usados com ripple na banda de corte. Os filtros elípticos possuem uma faixa de transição mais definida que o filtro Butterworth com as mesmas especificações.

As funções para se projetar filtros IIR digitais que usam protocolos analógicos possuem o seguinte formato:

```
[B,A] = butter (N,Wn);  
[B,A] = cheby1butter (N,Rp,Wn);  
[B,A] = cheby2utter (N,Rs,Wn);  
[B,A] = ellip (N,Rp,Rs,Wn);
```

O argumento de entrada representa a ordem do filtro (N), o ripple (Rs e Rp), e a frequência de corte normalizada (Wn). Os vetores de saída B e A são os vetores da expressão geral para filtros IIR e podem ser utilizados para se achar a função de transferência ou a equação diferencial.

Para se projetar filtros passa-banda os argumentos das funções são os mesmos do passa-baixa. Entretanto o vetor Wn precisa conter 2 elementos que representam as frequências normalizadas especificadas da banda de frequência, como Wn(1) e Wn(2).

Para se projetar filtros passa-alta, um parâmetro adicional com o nome 'high' precisa ser adicionado e ficam assim as novas formas:

```
[B,A] = butter (N,Wn,'high');  
[B,A] = cheby1butter (N,Rp,Wn,'high');  
[B,A] = cheby2utter (N,Rs,Wn,'high');  
[B,A] = ellip (N,Rp,Rs,Wn,'high');
```

Para se projetar filtros corta-banda, os argumentos são os mesmos dos filtros passa-alta, mas com o termo 'stop' ao invés de 'high'. O argumento Wn precisa ser um vetor que contenha 2 valores definem a banda de frequência como Wn(1) e Wn(2) para serem rejeitadas.

Para ilustrar estas funções supomos que precisemos projetar um filtro passa-alta Chebyshev Tipo II de ordem 6. Nós queremos como limite de passabanda um ripple de 0,1 ou 20db. O filtro é para ser usado para um sinal de até 1KHz. A frequência de corte é de 300HZ e a frequência normalizada é de 300/500 ou 0.6. Os comandos para o projeto deste filtro e para plotar a magnitude característica são os seguintes:

```
[B,A] = cheby2 (6,20,0.6,'high');  
[H,wT] = freqz (B,A,100);  
T = 0.001;  
hertz = wT/(2*pi*T);  
plot (hertz, abs(H)), title('Filtro Passaalta'),...  
xlabel ('Hz'), ylabel ('Magnitude'),grid
```

Para aplicar neste filtro um sinal x basta usar a seguinte instrução:

```
Y = filter (B,A,x);
```

Projeto de um Filtro IIR

O MATLAB possui uma função para projetar filtros com o método Yule-Walker. Esta técnica de projeto pode ser usada para projetar formas arbitrárias, possibilitando projetar respostas frequenciais multibandas.

O comando para projetar um filtro com esta função é o seguinte:

```
[B,A] = yulewalk(n,f,m);
```

Os vetores de saída B e A contém os coeficientes de n ordem do filtro IIR. Os vetores f e m especificam as características frequência-magnitude do filtro. As frequências em f precisam começar em 0, terminando em 1 e serem crescentes. A magnitude m precisa corresponder com a frequência f, e representa a magnitude esperada para a frequência correspondente. O exemplo a seguir nos mostra um projeto de um filtro com dois passabandas e que plota a magnitude de resposta na frequência normalizada.

```
m = [0 0 1 1 0 0 1 1 0 0];  
f = [0 .1 .2 .3 .4 .5 .6 .7 .8 1];  
[B,A] = yulewalk (12,f,m);  
[H,wT] = freqz(B,A,100);  
plot (f,m,wT/pi,abs (H)),...  
title('Filtro IIR com dois passabandas'),...  
xlabel ('Frequência Normalizada'),...  
ylabel ('Magnitude'), grid
```

Projeto Direto de um Filtro FIR

Os filtros FIR são projetados pelo MATLAB usando o método de Parks-McClellan que usa o algoritmo de troca de Remez. Lembrando que os filtros FIR necessitam apenas do vetor B, pois o denominador é igual a 1. Portanto a função Remez calcula apenas um único vetor, como mostrado a seguir:

```
B = remez (n,f,m);
```

O primeiro argumento define a ordem do filtro, e os valores de f e m são similares ao da técnica do filtro de Yule-Walker.

O exemplo a seguir nos mostra um projeto de um filtro com dois passabandas e que plota a magnitude de resposta na frequência normalizada.

```
m = [0 0 1 1 0 0 1 1 0 0];  
f = [0 .1 .2 .3 .4 .5 .6 .7 .8 1];  
B = remez (50,f,m);  
[H,wT] = freqz(B,[1],100);  
plot (f,m,wT/pi,abs (H)),...  
title('Filtro FIR com dois passabandas'),...  
xlabel ('Frequência Normalizada'),...  
ylabel ('Magnitude'), grid
```

Pratique!

Use as funções do MATLAB descritas nesta seção para projetar os seguintes filtros. Plote a magnitude do filtro projetado para confirmar se as características estão corretas.

1. Filtro IIR passabaixa com corte de 75Hz, onde usa-se uma frequência de até 500Hz.(Use um filtro de ordem 5)
 2. Filtro IIR passaalta com corte de 100Hz, onde usa-se uma frequência de até 1KHz.(Use um filtro de ordem 6)
 3. Filtro FIR passabaixa com corte de 75Hz, onde usa-se uma frequência de até 500Hz.(Use um filtro de ordem 40)
-

4. Filtro FIR passabanda com faixa de frequência entre 100Hz e 200Hz, onde usa-se uma frequência de até 1KHz.(Use um filtro de ordem 80)

Aplicação na Solução de Problemas: Filtros para Separação de Canais

Imagens coletadas de uma nave espacial ou de satélites que circundam a terra são enviadas para a terra como um fluxo de dados. Estes fluxos de dados são convertidos em sinais digitais que contém informações que podem ser re combinadas e assim reconstruir as imagens originais. Informações coletadas por outros sensores também são transmitidas para a terra. A frequência do sinal de informação do sensor depende dos tipos de dados que estão sendo medidos.

Técnicas de modulação podem ser usadas para mover o conteúdo da frequência para a banda de frequência especificada. Desta forma o sinal pode conter sinais múltiplos ao mesmo tempo. Por exemplo, supomos que queremos enviar 3 sinais em paralelo. O primeiro sinal contém componentes entre 0 e 100Hz, o segundo sinal contém componentes entre 500 e 1KHz, e o terceiro contém componentes entre 2KHz e 5KHz.

O sinal contendo estas três componentes é de até 10KHz. Para separar estas três componentes precisamos de um filtro passabaixa com corte de 100HZ, um passabanda com faixa de 500Hz a 1KHz, e um passa alta com corte de 2KHz. A ordem dos filtros precisa ser suficientemente grande para gerar um pequena banda de transição para que a componente de uma frequência não contamine as outras componentes.

1. DESCRIÇÃO DO PROBLEMA

Projete três filtros para serem usados com um sinal de até 10KHz. Um filtro é um passabaixa com corte de 100HZ, um passabanda com faixa de 500Hz a 1KHz, e um passa alta com corte de 2KHz.

2. DESCRIÇÃO DOS DADOS DE ENTRADA E SAÍDA

Não há valores de entrada para este problema. Os valores de saída são coeficientes dos vetores que definem os três filtros $H_1(z)$, $H_2(z)$ e $H_3(z)$, como na figura a seguir:

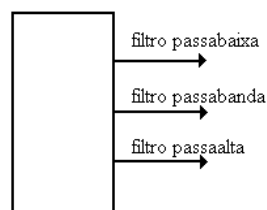


Figura 4 – Diagrama de I/O

3. EXEMPLO MANUAL

O espectro abaixo nos mostra a amostra de frequência com os três sinais filtrados, Nós usaremos os filtros Butterworth. Precisaremos experimentar várias ordens para sabermos se as bandas de transição não irão interferir uma nas outras.

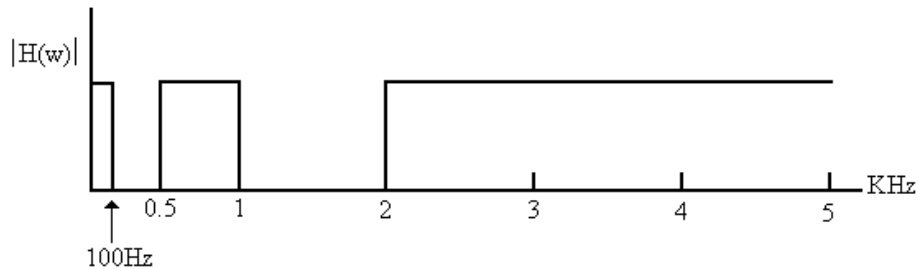


Figura 4 – Espectro de faixa dos filtros

4. SOLUÇÃO MATLAB

O seguinte programa MATLAB determina os valores da frequência normalizada para as frequências de corte pela função *butter*. Depois de calculados os coeficientes para os filtros, nós usamos a função *freqz* para plotar as características dos filtros. Lembrando que a função *freqz* normaliza as frequências para valores entre 0 e π . Nós usamos Hz como unidade no eixo X, para facilitar a visualização das características dos filtros projetados.

5. TESTANDO

As magnitudes dos três filtros são mostradas no mesmo gráfico para podermos verificar se os filtros não se sobrepõem.

Sumário MATLAB

<code>butter</code>	projeta um filtro digital Butterworth
<code>cheby1</code>	projeta um filtro digital Chebyshev Tipo 1
<code>cheby2</code>	projeta um filtro digital Chebyshev Tipo 2
<code>ellip</code>	projeta um filtro digital elíptico
<code>fft</code>	calcula a frequência de um sinal
<code>filter</code>	aplica um filtro digital em um sinal de entrada
<code>freqs</code>	calcula a frequência analógica
<code>freqz</code>	calcula a frequência digital
<code>grpdelay</code>	mede o grupo de retardo de um filtro digital
<code>ifft</code>	calcula a inversa da transformada de Fourier
<code>remez</code>	projeta um filtro digital FIR
<code>unwrap</code>	remove a descontinuidade 2π da fase angular
<code>yulewalk</code>	projeta um filtro digital IIR

PROBLEMAS:

1. Nós usaremos a soma de senóides de até 10KHz. O primeiro sinal contém uma soma de senóides com frequências de 24Hz, 40 Hz e 75Hz. O segundo sinal contém uma soma de senóides com frequências de 500Hz, 730 Hz e 850Hz. O terceiro sinal contém uma soma de senóides com frequências de 3500Hz, 4000 Hz e 4200Hz. Escolha as amplitudes e fases para estas senóides. Plote 500 pontos para o sinal 1, 2 e 3 em diferentes gráficos.
2. Calcule e plote a magnitude e a fase dos três sinais do exercício anterior. Use Hz como unidade no eixo x .
3. Adicione três tempos no sinal gerado no exercício 1. Plote o sinal e sua magnitude usando o Hz como unidade no eixo x.
4. Aplique um filtro passabaixa, um passabanda e um passaalta no sinal do problema 3 e plote a magnitude da frequência dos sinais de saída.
5. Para os seguintes filtros determine a banda de passagem, banda de transição e banda de corte. Use 0.7 para determinar as frequências de corte e use 0.1 para determinar as frequências de rejeição.

$$H(s) = \frac{0.5279}{s^2 + 1.0275s + 0.5279}$$

$$H(s) = \frac{s^2}{s^2 + 0.1117s + 0.0062}$$

$$H_1(z) = \frac{0.2066 + 0.4131z^{-1} + 0.2066z^{-2}}{1 - 0.3695z^{-1} + 0.1958z^{-2}}$$

$$H_2(z) = \frac{0.894 - 1.789z^{-1} + 0.894z^{-2}}{1 - 1.778z^{-1} + 0.799z^{-2}}$$

6. Projete um filtro para remover frequências entre 500Hz e 1000Hz de um sinal de até 10KHz. Compare os resultados dos filtros elíptico e do filtro Yule-Walker, ambos de ordem 12. Plote a magnitude dos dois sinais.
7. Projete um filtro para remover frequências entre 100Hz e 150Hz, e outro entre 500Hz e 600Hz de um sinal de até 2.5KHz. Compare os resultados usando FIR e IIR. Plote a magnitude dos dois sinais

Capítulo 15 - Matemática Simbólica

Introdução

Agora é possível instruir ao MATLAB que manipule expressões matemáticas, sem de fato usar números, que lhe permitam calcular com símbolos matemáticos, além de números. Esse processo é freqüentemente chamado de matemática simbólica. Aqui estão alguns exemplos de expressões simbólicas:

$$\cos(x^2)$$

$$3x^2 + 5x - 1$$

$$v = d \ x^2$$

$$f = \int x^2 \, dx$$

A toolbox de Matemática Simbólica é uma coleção de funções para o MATLAB usadas para manipular e resolver expressões simbólicas. Há diversas ferramentas para combinar, simplificar, derivar, integrar e resolver equações diferenciais e algébricas. Outras ferramentas são utilizadas em álgebra linear para derivar resultados exatos para inversas, determinantes e formas canônicas e para encontrar os autovalores de matrizes simbólicas, sem o erro introduzido pelo cálculo numérico.

A aritmética de precisão variável que calcula simbolicamente e retorna um resultado para qualquer grau de precisão especificado, também está disponível no MATLAB.

As ferramentas contidas na toolbox de matemática simbólica foram criadas por meio de um poderoso programa de software chamado Maple, originalmente desenvolvido na Universidade de Waterloo, no Canadá. Quando você pedir ao MATLAB para executar alguma operação simbólica, ele solicitará ao Maple para fazê-lo e então retornará o resultado para a janela de comando do MATLAB. Por isso, fazer manipulações simbólicas no MATLAB é uma extensão natural do modo como você usa o MATLAB para processar números.

15.1 Expressões Simbólicas

Expressões simbólicas são strings de caracteres ou conjuntos de strings de caracteres que representam números, funções, operadores e variáveis. As variáveis não têm de ter valores previamente definidos. Equações simbólicas são expressões simbólicas que contêm um sinal de igualdade. A aritmética simbólica é a prática de resolução dessas equações por meio da aplicação de regras conhecidas e de identidades a determinados símbolos, exatamente da forma que você aprendeu a resolvê-las em álgebra e cálculo. Matrizes simbólicas são conjuntos cujos elementos são expressões simbólicas.

Representações de Expressões Simbólicas no MATLAB

O MATLAB representa expressões simbólicas internamente como strings de caracteres para diferenciá-las de variáveis numéricas e operadores. Aqui estão alguns exemplos de expressões simbólicas com seus equivalentes em MATLAB:

Expressões simbólicas	Representação no MATLAB
$\frac{1}{2x^n}$	<code>'1 (2 * x ^ n) '</code>
$\cos(x^2) - \sin(2x)$	<code>' cos(x ^ 2) - sin(2 * x) '</code>
$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$	<code>sym (' [a , b ; c , d] ')</code>

Exercícios:

1. `diff('cos(x)')`
2. `M = sym ('[a,b;c,d]')`
3. `determ(M)`

Observe que, no primeiro exercício acima, a expressão simbólica foi definida implicitamente pelo uso de aspas simples para dizer ao MATLAB que `('cos(x)')` seja uma expressão simbólica, em vez de uma expressão numérica, ao passo que, no segundo exemplo, a função `sym` foi usada para, explicitamente, dizer ao MATLAB que `M = sym ('[a,b;c,d]')` é uma expressão simbólica. Geralmente a função explícita `sym` não é necessária onde o MATLAB puder determinar, por si só o tipo de argumento.

Em MATLAB, a forma `func arg` é equivalente a `func('arg')`, onde `func` é uma função e `arg` é um argumento na forma de string de caracteres. Por exemplo, o MATLAB consegue descobrir que `diff cos(x)` e `diff ('cos(x)')` significam a mesma coisa, ou seja, `diff(sym('cos(x)'))`, mas a primeira forma é certamente mais fácil de ser digitada. Contudo, a função `sym` é necessária. Isto fica claro no exemplo seguinte:

Exemplo

`M=[a,b;c,d]` M é uma matriz numérica usando valores de a até d
`M='[a,b;c,d]'` M é uma string de caracteres
`M=sym('[a,b;c,d]')` M é uma matriz simbólica

Aqui M foi definido de três maneiras: numericamente (se a, b, c e d tiverem sido predefinidos), como uma string de caracteres e como uma matriz simbólica.

Muitas funções simbólicas são suficientemente avançadas para converter strings de caracteres em expressões simbólicas de forma automática. Mas, em alguns casos, em especial na criação de um conjunto simbólico, a função `sym` deve ser usada para converter um string de caracteres especificamente em uma expressão simbólica. A forma implícita, ou seja, `diff cos(x)`, é mais útil para tarefas simples que não referenciam resultados anteriores. Contudo, a forma mais simples (sem aspas) requer um argumento que é um string de caracteres simples sem espaços em seu interior:

Exemplo

```
diff x^2+3*x+5
diff x^2 + 3*x + 5
```

Na segunda equação os espaços pressupõem strings de caracteres distintos.

As expressões simbólicas sem variáveis são chamadas de constantes simbólicas. Quando as constantes simbólicas são visualizadas, freqüentemente torna-se difícil distingui-las de números inteiros.

Exemplo

```
f=symop('(3*4-2)/5+1')
isstr(f)
```

A função `isstr` nada mais faz do que retornar se `f` é uma string de caracteres (1 = sim e 0 = não). Nesse caso, `f` representa a constante simbólica '3' não o número 3. O MATLAB armazena strings de caracteres como representação ASCII de caracteres. Consequentemente, se você realizar uma operação numérica em um string de caracteres, o MATLAB usa o valor ASCII de cada caractere na operação.

Exemplo

```
f +1
```

15.2 Variáveis Simbólicas

Quando se trabalha com expressões simbólicas contendo mais de uma variável, uma variável `a` é a variável independente, ele seleciona uma baseado na regra seguinte:

A variável independente padrão em uma expressão simbólica é uma letra minúscula única, que não seja `i` ou `j`, que não faça parte de uma palavra. Se não há tal caractere, é escolhido `x`. Se o caractere não for único, aquele mais próximo de `x`, alfabeticamente falando, é escolhido. Se houver empate, o caractere posterior no alfabeto será escolhido.

A variável independente padrão, algumas vezes conhecida como variável livre, na expressão '`1 / (5+cos(x))`' é '`x`'; a variável livre na expressão '`3*y+z`' é '`y`'; e a variável livre na expressão '`a+sin(t)`' é '`t`'. A variável simbólica livre na expressão '`sin(pi/4)`' -

$\cos(3/5)$ é 'x' porque esta expressão é uma constante simbólica, não contendo variáveis simbólicas.

Você pode pedir ao MATLAB para lhe dizer qual a variável em uma expressão simbólica é considerada por ele como a variável independente usando a função *symvar*.

Exemplo

<code>symvar('a*x+y')</code>	encontra a variável simbólica padrão
<code>symvar('a*t+s/(u+3)')</code>	u é mais próxima de x
<code>symvar('sin(omega)')</code>	'omega' não é um caracter único
<code>symvar('3*i+4*j')</code>	i e j são iguais a $\sqrt{-1}$
<code>symvar('y+3*s','t')</code>	encontra a variável mais próxima de 't'

Se *symvar* não encontrar uma variável padrão, usando esta regra, considerará que ela não existe e retornará x. Isto será verdadeiro para expressões que contenham variáveis com múltiplos caracteres, bem como constantes simbólicas, as quais não contêm variáveis.

Muitos comandos dão a você a opção de especificar a variável independente, se desejado:

Exemplo

<code>diff('x^n')</code>	deriva em relação à variável padrão 'x';
<code>diff('x^n','n')</code>	deriva em relação a n;
<code>diff('sin(omega)')</code>	deriva usando a variável padrão;
<code>diff('sin(omega)','omega')</code>	especifica a variável independente;

Exercícios

Dada cada expressão simbólica, use a sintaxe do MATLAB para criar a expressão simbólica equivalente do MATLAB.

a. $f = ax^2 + bx + c$

b. $p = \frac{3s^2 + 2s + 1}{4s - 2}$

c. $r = e^{-2t}$

d. $\frac{d}{dx} \sqrt{3x^2 + 2x + 5}$

2. Encontre a variável independente padrão que é retornada por *symvar* nas expressões seguintes:

a. $z = 3ac + 4b - 2$

b. $x = 4a + 3c + b^2 + 1$

c. $q = r + \sqrt{3k^2 + 2p}$

d. $f = s^{3nt}$

e. $n = 3r + 2s^2 + 5$

15.3 Operações em Expressões Simbólicas

Depois de criar uma expressão simbólica, você provavelmente vai querer mudá-la de alguma forma. Você pode querer extrair parte de uma expressão, combinar duas expressões ou achar o valor numérico de uma expressão simbólica. Há muitas ferramentas simbólicas que lhe permitirão realizar essas tarefas.

Quase todas as funções simbólicas agem sobre expressões simbólicas e conjuntos simbólicos e retornam expressões simbólicas ou conjuntos. O resultado pode às vezes se parecer com um número, mas é uma expressão simbólica representada internamente como um string de caracteres. Conforme foi dito antes, é possível descobrir se o resultado que parece ser um número é um inteiro ou um string de caracteres utilizando a função *isstr* do MATLAB.

Extração de Numeradores e Denominadores

Se sua expressão for um polinômio racional (uma razão de dois polinômios) ou puder ser expandida em um polinômio racional (incluindo aqueles com um denominador igual a 1), você pode extrair o numerador e o denominador usando *numden*. Dadas as expressões:

$$m = x^2$$

$$f = \frac{ax^2}{b-x}$$

$$g = \frac{3}{2}x^2 + \frac{2}{3}x - \frac{3}{5}$$

$$h = \frac{x^2 + 3}{2x - 1} + \frac{3x}{x - 1}$$

$$k = \begin{bmatrix} \frac{3}{2} & \frac{2x+1}{3} \\ \frac{4}{x^2} & 3x+4 \end{bmatrix}$$

numden combina e racionaliza a expressão, se necessário, e retorna o numerador e o denominador resultantes. As linhas de comando do MATLAB para se fazer isso são:

```
m = 'x^2'
[n,d]= numden(m)

f = 'a*x^2 / (b - x)'

[n,d]=numden(f)

g = '3/2*x^2+2/3*x-3/5'

[n,d] = numden(g)

h = '(x^2+3)/(2*x-1)+3*x/(x-1)'

[n,d] = numden(h)

k = sym ('[3/2,(2*x+1)/3;4/x^2,3*x+4]')

[n,d] = numden(k)
```

Essa expressão , k, é um conjunto simbólico, *numden* retornou dois novos conjuntos, n e d, onde n é um conjunto de numeradores e d o conjunto de denominadores. Se você usar a forma s = numden(f), *numden* retorna apenas o numerador dentro da variável s.

15.4 Operações Algébricas Padrão

Diversas operações algébricas podem ser executadas em expressões simbólicas. As funções *symadd*, *symsub* e *symdiv* somam, subtraem, multiplicam e dividem, respectivamente, duas expressões, e *sympow* eleva uma expressão à potência da outra.

Exemplo

$$f = 2x^2 + 3x - 5$$

$$g = x^2 - x + 7$$

No MATLAB:

```
f = '2*x^2+3*x-5'
g = 'x^2-x+7'
symadd(f,g)
symsub(f,g)
symmul(f,g)
symdiv(f,g)
sympow(f, '3*x')    %Encontra a expressão para f3
```

Uma outra função de aplicação geral permite que você crie novas expressões a partir de outras variáveis simbólicas, expressões e operadores. *Symop* recebe até 16 argumentos separados por vírgula, cada um dos quais pode ser uma expressão simbólica, um valor numérico ou um operador (+,-,*,/,^,(ou)). *Symop* então concatena os argumentos e retorna a expressão resultante.

Exemplo

```
f = 'cos(x)'
g = 'sin(2*x)'
symop(f, '/', g, '+', 3)
```

Todas estas operações também trabalham com conjuntos de argumentos da mesma forma.

Operações Avançadas

O MATLAB tem a capacidade de realizar operações mais avançadas em expressões simbólicas. A função *compose* combina $f(x)$ e $g(x)$ em $f(g(x))$, a função *finverse* encontra o inverso funcional da expressão, e a função *symsum* encontra o somatório simbólico de uma expressão.

Dada as expressões:

1. $f = \frac{1}{1+x^2}$

2. $g = \sin(x)$

3. $h = \frac{1}{1+u^2}$

4. $k = \sin(v)$

No MATLAB

```
f = '1/(1+x^2)';
g = 'sin(x)';
h = '1/(1+u^2)';
k = 'sin(v)';
compose(f,g)
compose(g,f)
```

Compose pode também ser usada em funções que tem variáveis independentes diferentes:

```
compose ( h, k, 'u', 'v')
```

O inverso funcional de uma expressão, digamos $f(x)$, é a expressão $g(x)$ que satisfaz a condição $g(f(x)) = x$. Por exemplo, o inverso funcional de e^x é $\ln(x)$, já que $\ln(e^x) = x$.

A função *finverse* retorna o inverso funcional de uma expressão e avisa a você se o resultado não for único.

Exemplos

```
finverse ('x^2')
finverse ('a*x+b')
finverse ('a*b+c*d-a*z','a')
```

A função *symsum* encontra o somatório simbólico de uma expressão. Há quatro formas para a função:

MATLAB	Retorna
<code>symsum(f)</code>	$\sum_{0}^{x-1} f(x)$
<code>symsum(f, a, b)</code>	$\sum_{0}^b f(x)$
<code>symsum(f, 's')</code>	$\sum_a^{s-1} f(s)$
<code>symsum(f, 's', a , b)</code>	$\sum_a^b f(s)$

Exemplos

1. $\sum_0^{x-1} x^2$

2. $\sum_1^n (2n-1)^2$

3. $\sum_1^\infty \frac{1}{(2n-1)^2}$

15.5 Funções de Conversão

Essa seção apresenta ferramentas para conversão de expressões simbólicas em valores numéricos e vice-versa.

A função *sym* pode receber um argumento numérico e convertê-lo em uma representação simbólica. A função *numeric* faz o contrário, converte uma constante simbólica em um valor numérico.

Exemplo

```
phi= '(1+sqrt(5))/2'  
numeric (phi)
```

A função *eval* faz com que o MATLAB calcule um string de caracteres. Dessa forma, *eval* é outra função que pode ser usada para converter constante simbólica em um número, ou para calcular uma expressão.

Exemplo

```
eval(phi)
```

Você já trabalhou com polinômios em MATLAB, usando vetores cujos elementos são os coeficientes dos polinômios. A função simbólica *sym2poly* converte um polinômio simbólico em seu vetor de coeficientes equivalente no MATLAB. A função *poly2sym* faz o inverso e possibilita que você especifique a variável a ser usada na expressão resultante.

Exemplo

```
f= '2*x^2+x^3-3*x+5'  
n=sym2poly(f)  
poly2sym(n)  
poly2sym(n, 's')
```

Substituição de Variáveis

Quando se tem uma expressão simbólica em x , e que queira mudar a variável para y , usa-se a função *subs*.

Exemplo

```
f= 'a*x^2+b*x+c'
subs(f, 's', 'x')
h= subs(f, '2', 'x')
```

15.6 Derivação e Integração

Derivação

A derivação de uma expressão simbólica usa a função *diff* em uma dentre as quatro formas:

Expressão	Resultado
$f = 'a*x^3+x^2-b*x-c'$	Define uma expressão simbólica
$\text{diff}(f)$	Deriva em relação ao padrão (x)
$\text{diff}(f, 'a')$	Deriva f em relação a (a)
$\text{diff}(f, 2)$	Deriva f duas vezes em relação a (x)
$\text{diff}(f, 'a', 2)$	Deriva a função f duas vezes em relação a (a)

A função *diff* também opera sobre conjuntos. Se f for um vetor simbólico ou matriz, $\text{diff}(f)$ deriva cada elemento do conjunto:

Exemplo

```
f= sym(['a*x, b*x^2, c*x^3, d*s'])
diff(f)
```

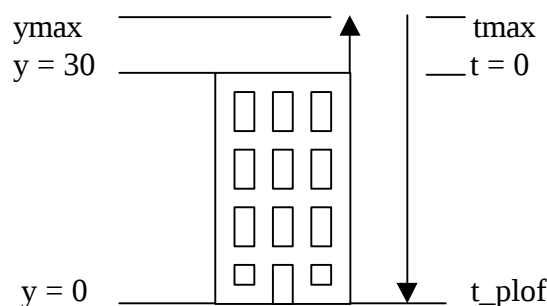
Integração

A função de integração *int*(f), onde f é uma expressão simbólica, tenta encontrar outra expressão simbólica F tal que $\text{diff}(F)=f$.

Expressão	Resultado
$f = \sin(s+2*x)$	Cria uma função simbólica
$\text{int}(f)$	Integra em relação a x
$\text{int}(f, 's')$	Integra em relação a s
$\text{int}(f, \pi/2, \pi)$	Integra em relação a x de $\pi/2$ a π
$\text{int}(f, 's', \pi/2, \pi)$	Integra em relação a s de $\pi/2$ a π
$\text{int}(f, 'm', 'n')$	Integra em relação a x de m para n

Exercício:

Hélio está em uma excursão com sua turma de escola, no alto do edifício Central. Ele pega um tomate maduro em sua mochila, debruça-se sobre a beirada do terraço e atira-o no ar. O tomate é jogado para cima, a uma velocidade inicial $v_0 = 20\text{m/s}$. O terraço encontra-se a $y_0 = 30$ metros acima do nível do solo. Onde estará o tomate a um intervalo arbitrário de t segundos mais tarde? Quando ele alcançará a altura máxima? Que altura acima do solo o tomate conseguirá alcançar? Quando o tomate atingirá o solo? Considere que não há resistência do ar e que $g = 10\text{ m/s}^2$.

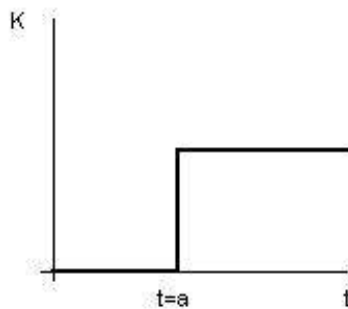


15.7 Transformadas

As transformadas são usadas com muita frequência em Engenharia para mudar o campo de referência entre o domínio do tempo e o domínio s , domínio da frequência ou domínio Z . Há muitas técnicas para analisar estados de equilíbrio e sistemas que sofrem mudanças muito suaves no domínio do tempo, mas os sistemas complexos quase sempre podem ser analisados mais facilmente em outros domínios.

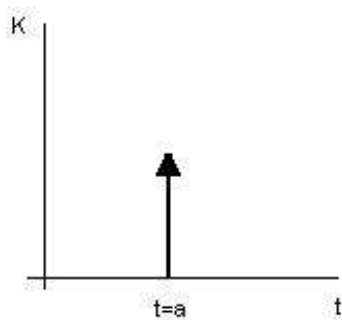
Funções Degrau e Impulso

Os problemas de Engenharia freqüentemente fazem uso da função Degrau $u(t)$ e impulso $\delta(t)$ na descrição de sistemas. A função Degrau $Ku(t - a)$, onde K é uma constante, é definida como $Ku(t - a) = 0$ para $t < a$ e $Ku(t - a) = K$ para $t > a$. Eis um gráfico da função Degrau $Ku(t - a)$:



A função impulso $\delta(t)$ é a derivada da função Degrau $u(t)$. A função Degrau $K\delta(t-a)$ é definida como $K\delta(t-a) = 0$ para $t < a$ e $\int_{-\infty}^{\infty} K\delta(t-a)dt = K$ para $t = a$.

Quando representada em gráfico ela é comumente representada como uma seta de amplitude K em $t = a$. Eis o gráfico de $K\delta(t-a)$:



Exemplo

```
u = 'k*Heaviside(t-a)'
d = diff(u)
int(d)
```

Transformada de Laplace

A Transformada de Laplace realiza a operação $F(s) = \int_0^{\infty} f(t) e^{-st} dt$. Para transformar $f(t)$, no domínio do tempo, em $F(s)$, no domínio de s .

A Transformada de Laplace da função cosseno amortecido $e^{-at} \cos(\omega t)$ é encontrada usando-se a função Laplace:

```
f=sym('exp(-a*t)*cos(w*t)')
F= laplace(f)
pretty(F)
laplace('Dirac(t)')
laplace('Heaviside(t)')
```

As expressões podem ser transformadas novamente para o domínio do tempo, usando-se o inverso da transformada de Laplace, `invlaplace`, que realiza a operação $f(t)$. Usando F do exemplo acima temos:

```
invlaplace(F)
```

Transformada de Fourier

A Transformada de Fourier e sua inversa são muito usadas em análise de circuitos para determinar as características de um sistema em ambos os domínios de tempo e de frequência. O MATLAB usa as funções `fourier` e `invfourier` para transformar expressões entre domínios. A Transformada de Fourier e sua inversa são definidas como:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt \quad f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega t} d\omega$$

O MATLAB usa um w para representar ω em expressões simbólicas.

Exemplo

```
f= 't*exp(-t^2)'
F= fourier(f)
invfourier(F)
```

Transformada Z

As transformadas de Laplace e Fourier são usadas para analisar sistemas de tempo contínuos. Transformadas Z, por outro lado, são usadas para analisar sistemas de tempo discreto. A Transformada Z é definida como:

$$F(z) = \sum_{n=0}^{\infty} f(n)z^{-n}$$

onde z é um número complexo.

A Transformada z e a Transformada z inversa são obtidas usando-se as funções *ztrans* e *invztrans*. O formato é similar ao das funções de transformadas de Laplace e Fourier.

Exemplo

```
f = '2 ^ n / 7 - (-5) ^ n / 7'
G = ztrans(f)
pretty(G)
invtrans(G)
```