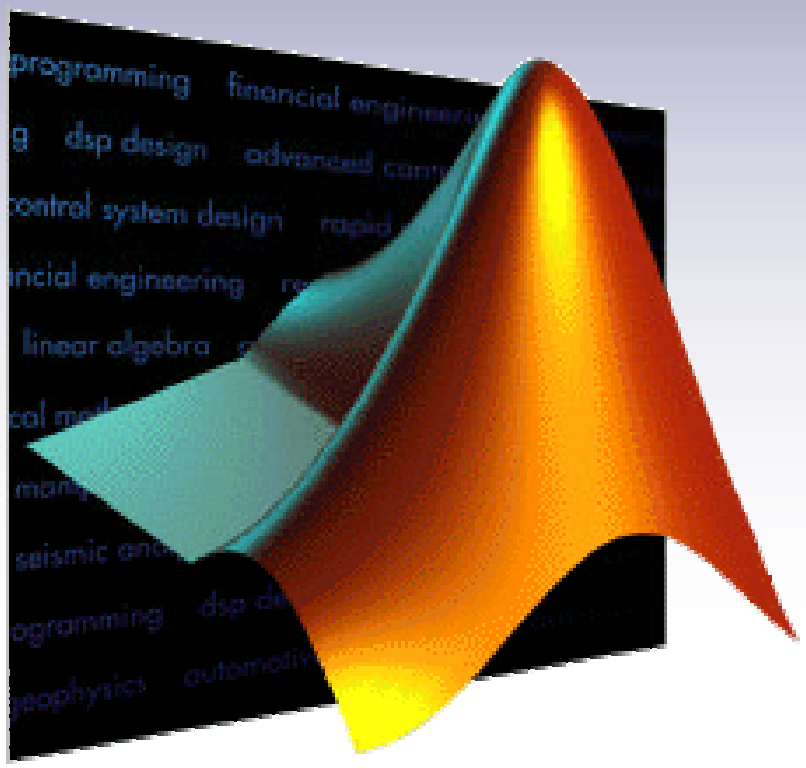
2ª EDIÇÃO

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 ‘ Engeneering 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 tecnico-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

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/0. 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.

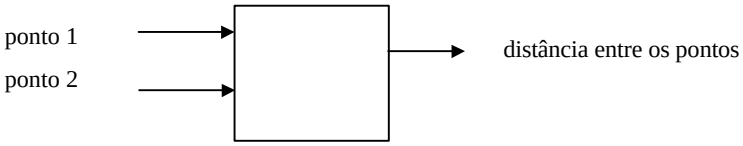


Figura 1.1 – Diagrama I/O

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:

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

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:

$$d = \sqrt{s1^2 + s2^2}$$
$$d = \sqrt{(4-1)^2 + (7-5)^2}$$
$$d = \sqrt{13}$$
$$d = 3{,}61$$

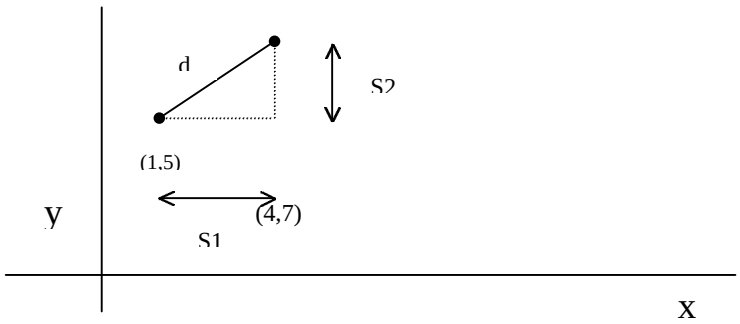


Figura 1.2 – Distância entre dois pontos.

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

Se uma matriz contiver m linhas e n colunas, então conterá um total de m . 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 MALTAB 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.

E x e r c í c i o s

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.

E x e r c í c i o s

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];
» c = 2*a
 c =
 2     4     6
```

```
» T = [ 1.1  2.4   3.7];
» U = 2*T
 U =
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.

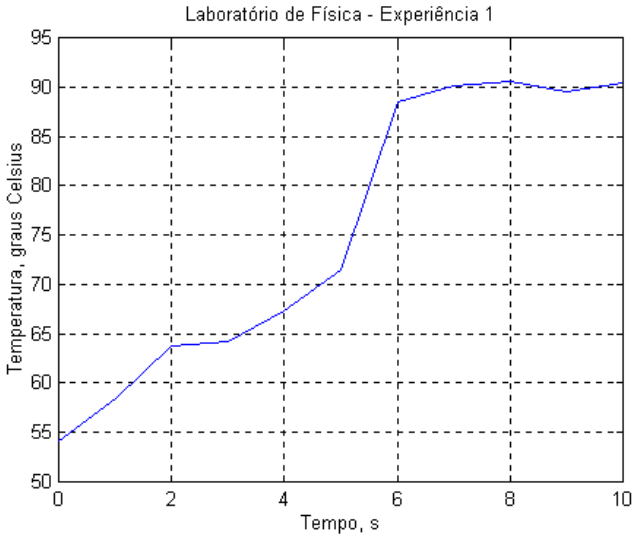
```
plot (x, y)
```

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.

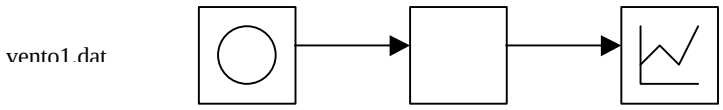


Diagrama I/O

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 18^0 . 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.
√-1	i,j	Estas variáveis são inicialmente agrupadas ao valor √-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 x 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 x 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 x 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 x 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
Multiplicação	a x b	a*b
Divisão Direita	$\frac{a}{b}$	a/b
Divisão Esquerda	$\frac{b}{a}$	a\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	Multiplicação 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:

```
area = 0.5*h*(B + b);
```

Suponha que omitamos os parênteses:

```
area = 0.5*altura*B + b;
```

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

```
area = (0.5*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 convertemos 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.

E x e r c í c i o s

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

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

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 dentre 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 distinguí-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.\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:

C = A.^2;
D = A.^B;

Os vetores C e D serão os seguintes:

C = [4 25 36]
D = [4 125 7776]

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:

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

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

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

Exercícios

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

A = [2 -1 5 0] 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.\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 seqüência 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.

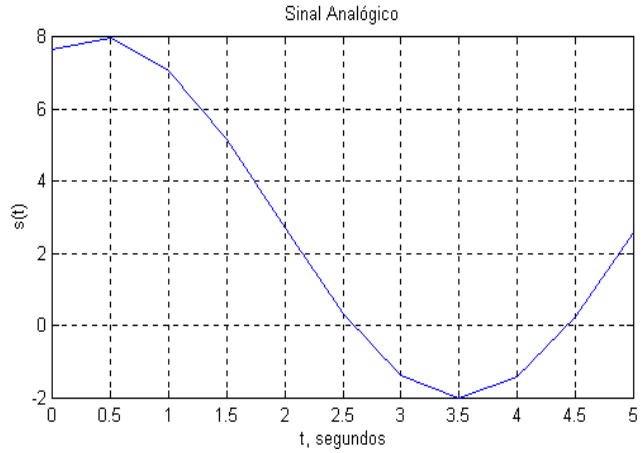


Figura 3,1 – Sinal Analógico ou contínuo

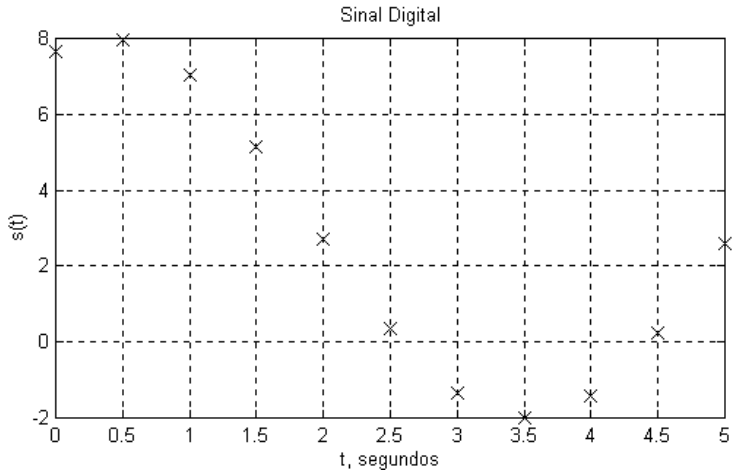


Figura 3..2 – Amostra ou Sinal Digital

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.

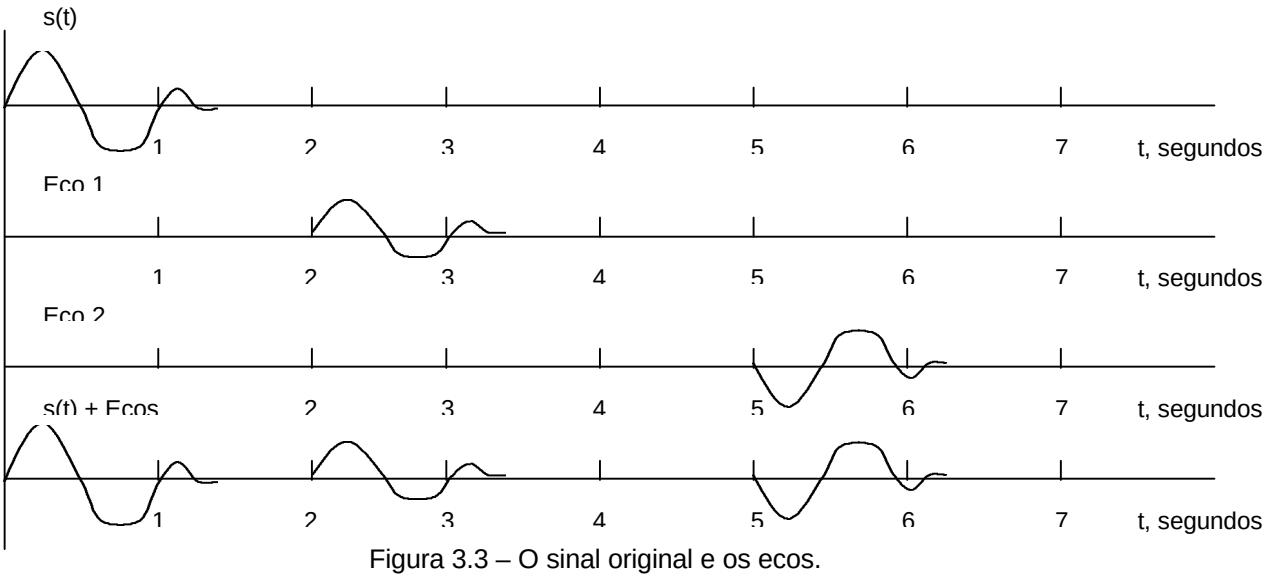


Figura 3.3 – O sinal original e os ecos.

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.

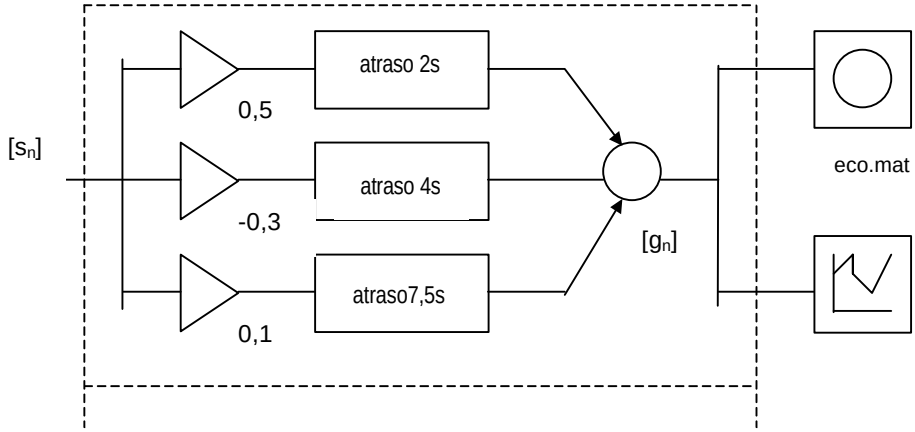


Figura 3.4 – Diagrama entrada / saída

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):

Tempo,s	*Valor do sinal*
2,0	(0,5) . (0,0) = 0,0
2,1	(0,5) . (0,5) = 0,25
2,2	(0,5) . (1,0) = 0,5

Tempo,s	*Valor do sinal*
4,0	(- 0,3) . (0,0) = 0,0
4,1	(- 0,3) . (0,5) = - 0,15
4,2	(- 0,3) . (1,0) = - 0,3

Tempo,s	*Valor do sinal*
2,0	(0,1) . (0,0) = 0,0
2,1	(0,1) . (0,5) = 0,05
2,2	(0,1) . (1,0) = 0,1

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

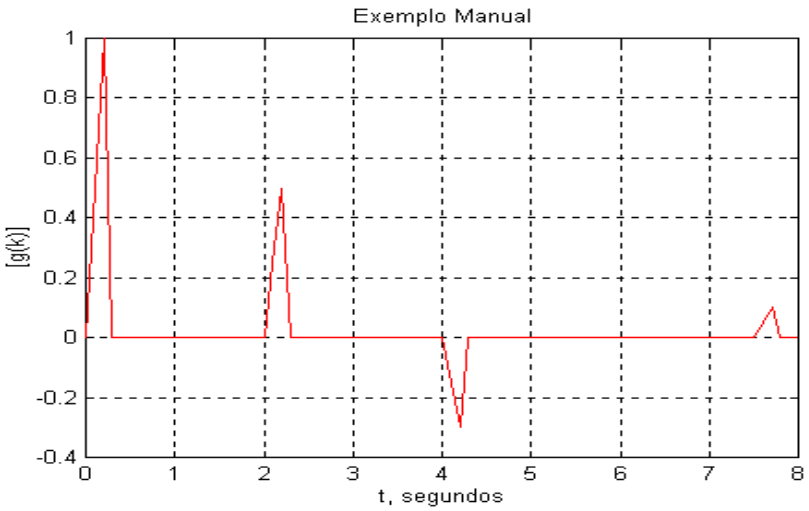


Figura 3.5 – Sinal Original mais os três ecos.

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 valore 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(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 sensitivy*" 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:

abs (x)	Calcula o valor absoluto de x.
sqrt(x)	Calcula a raiz quadrada de x.
round(x)	Arredonda o valor de x para o inteiro mais próximo.
fix(x)	Arredonda o valor de x para o inteiro mais próximo de zero.
floor(x)	Arredonda o valor de x para o inteiro mais próximo de - ∞
ceil(x)	Arredonda o valor de x para o inteiro mais próximo de ∞
sign(x)	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.
Rem(x,y)	Retorna o resto da divisão x/y. Por exemplo, rem (25,4) é 1, e rem (100,21) é 16.
Exp(x)	Esta função retorna ao valor de e^x, onde *e* é a base para logaritmo natural

	ou aproximadamente 2.718282.
log(x)	Retorna a ln x, o logaritmo natural de x para a base *e*.
Log10(x)	Retorna a $\log_{10}$x, ou seja, o logaritmo de x na base 10.

E x e r c í c i o s

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 Trigonométricas

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° = π radianos:

```
ângulo_graus = ângulo_radianos*(180/pi);
ângulo_radianos = ângulo_graus*(pi/180);
```

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

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

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

E x e r c í c i o s

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 contêm 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 (*so*und *n*avigation *a*nd *r*anging) 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 freqüência 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 freqüências.

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 freqüência da senóide em ciclos por segundo, ou Hertz(Hz).

Se a freqüência 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 freqüência 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 freqüência é dada por

$$f = 1/P$$

onde f é a freqüência 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:

$$g(t) = A\,\text{sen}\,(2pft)$$

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

$$g(t) = A\,\text{sen}\,(2pft + \phi)$$

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 :

$$s(t) = \begin{cases} \sqrt{\frac{2E}{PD}}\cos(2pfc), & 0 \le t \le PD \\ 0 & para\ os\ demais\ ins\tan tes \end{cases}$$

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

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

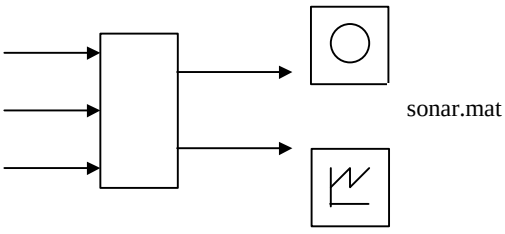
Escreva um programa MATLAB que permita ao usuário entrar com valores de E, PD, e f_c para gerar um sinal sonar. Armazene os valores do sinal em um arquivo MAT denominado sonar.mat. A amostragem do sinal deve cobrir a duração do pulso e conter 10 amostras de cada período de x(t). Além disso, adicione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

$$a_1x^n + a_2x^{n-1} + a_3x^{n-2} + \ldots + n_{a-1}\,x^2 + a_nx + a_{n+1} = 0$$

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 √-1. (O MATLAB também reconhece o uso de j para representar √-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:

real (x)	Calcula a parte real do número complexo x.
imag(x)	Calcula a parte imaginária do número complexo x.
conj(x)	Calcula o conjugado do número complexo x.
abs(x)	Calcula o módulo do número complexo x.
angle(x)	Calcula o ângulo usando o valor de atan2 (imag(x), real(x)), e portanto o ângulo está entre -π 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 raio3,6.

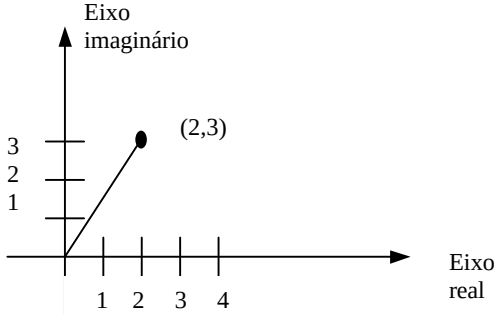


Figura 3.10 – Plano Complexo

Conversão

- retangular a polar

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

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

- polar a retangular

$a = r \cos \theta$
$b = r \text{ sen } \theta$

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;
```

E x e r c í c i o s

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:

a = [2 4 6]
b = [3 5 1]

Então, o valor de a < b será o vetor [1 1 0], enquanto o valor de a ~ = 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:

~ (b = = c | 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 | 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 b = 1.5 k = -3

1. a < 10.0
2. a + b >= 6.5
3. k ~= 0
4. b – k > a
5. ~(a = = 3*b)
6. –k < = k + 6
7. a < 10 & a > 5
8. abs (k) > 3 | 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:

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

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

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

Calculamos a velocidade correta com estes comandos:

```
if d < = 1        0
        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 freqüentemente 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.

E x e r c í c i o s

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 variável = 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 *velocidade(1), 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
```

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 conterá o próximo valor do vetor;
- Se a expressão for uma matriz, então a cada iteração a variável conterá 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.

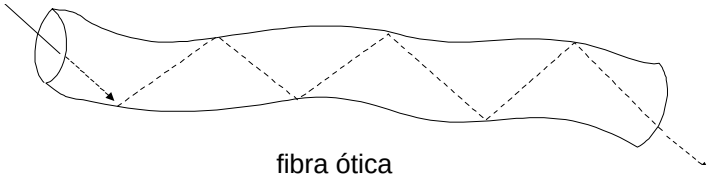
E x e r c í c i o s

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 count = -2:14
3. for k = -2:-1:10
4. for time = 10:-1:0
5. for time = 10:5
6. for 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:



fibra ótica

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

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

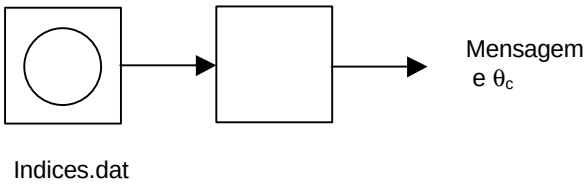
Escreva um programa MATLAB que determina se a luz será ou não transmitida por dois materiais que forma um guia. Suponha que um arquivo de dados ASCII denominado indices.dat contenha o número de possíveis índices de refração da fibra seguido pelo índice de refração do meio que o envolve. O programa deve 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:

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

O guia de luz transmitirá luz para todos os ângulos de incidência maiores que $41{,}82^0$.

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
```

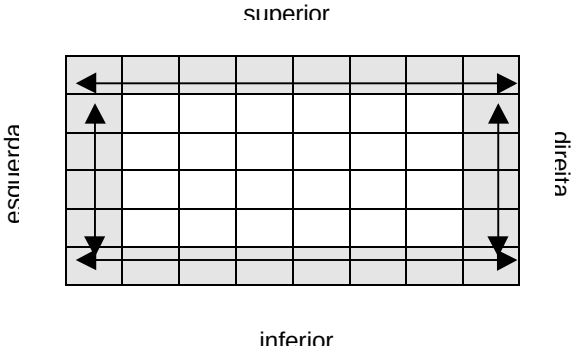
Se a expressão for verdadeira, então o grupo de comandos A é executado. Depois destes comandos serem executados, a condição é novamente questionada. Se for verdadeira, o grupo de comandos é novamente executado. Quando a condição for falsa, o controle pula para o comando posterior ao comando *end*. As 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

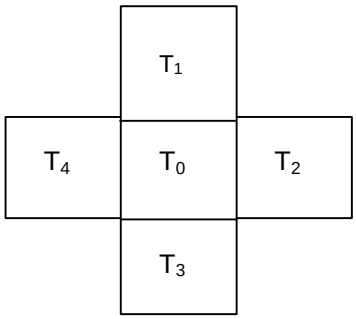
O projeto de novos materiais para o aperfeiçoamento das características do fluxo de ar acerca dos veículos envolve a análise dos materiais para não somente o fluxo de ar mas também para propriedades como a distribuição de temperatura. Neste problema, consideramos a distribuição em uma fina placa metálica 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:

$$T_0 = \frac{T1 + T2 + T3 + T4}{4}$$



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.

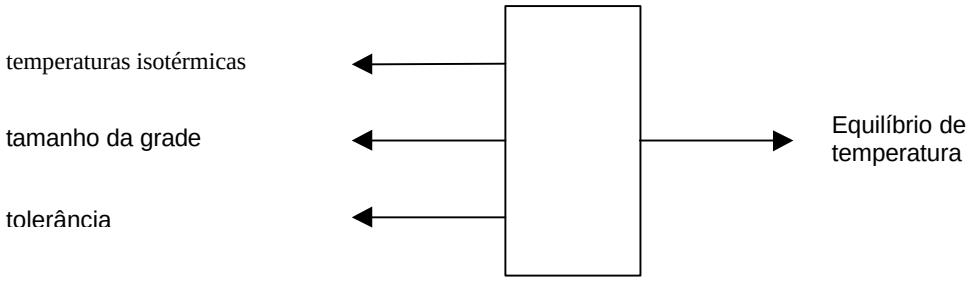
Portanto, usamos uma matriz de temperaturas antigas para calcular uma matriz de novas temperaturas e para determinar se alguma temperatura muda acima da tolerância. Quando movemos as novas temperaturas para o conjunto antigo. Quando nenhuma das temperaturas muda acima da tolerância, supomos que o equilíbrio 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 100º e 50º, e nós iniciamos todos os pontos restantes do zero. Usamos uma tolerância de 40º. 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 iteração 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 60^0 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:

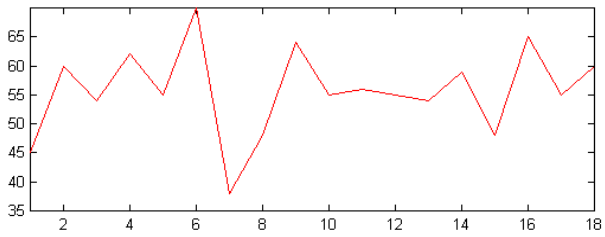


Figura 5.1 - Percentagem de lucro apresentada pela corretora A durante 18 dias.

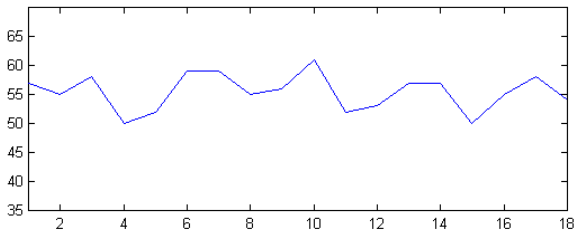


Figura 5.2 - Percentagem de lucro apresentada pela corretora B durante 18 dias.

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

- Média mais alta de percentagem de lucro;
- Maior percentagem de lucro;
- Menor variação de Percentagem 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)
ans =
55.7222
```

```
» mean (corretoraB)
ans =
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)
ans =
70
```

```
» max(corretoraB)
ans =
61
```

E a menor percentagem?

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

```
»min(corretoraA)
ans =
38
```

```
» min(corretoraB)
ans =
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:

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

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

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

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

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 $(\sigma)^2$. 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 =
     6
     6
     6
     8
     8
     8
     8
     8
     8
     8
     8
     8
     8
    10
    10
```

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.

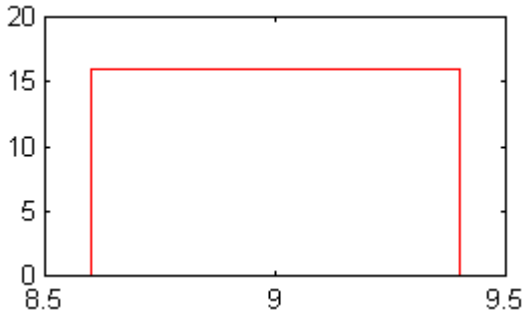


Figura 5.3 - Histograma de Direito

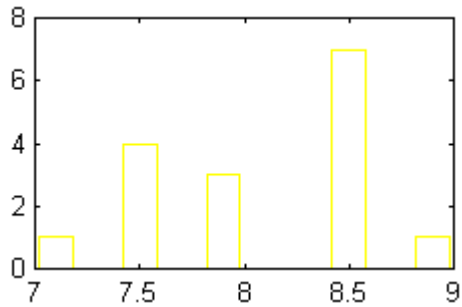


Figura 5.4 - Histograma de Economia

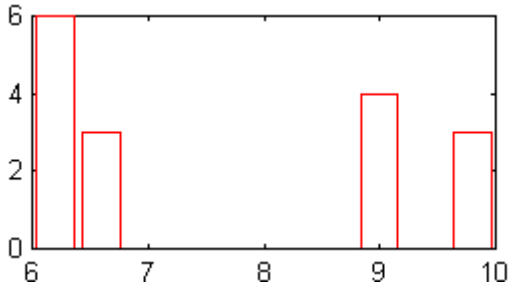


Figura 5.5 - Histograma de Metodologia

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
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.

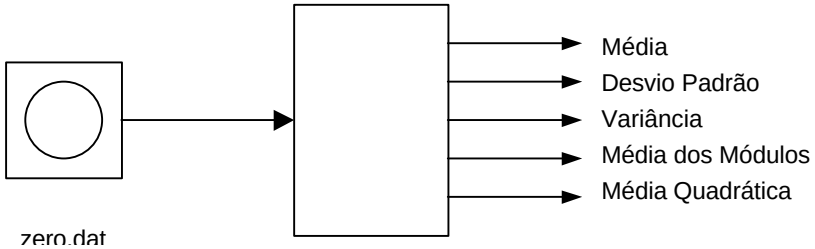


Figura 5.6 – diagrama I/O

3. EXEMPLO MANUAL

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

[2.5 8.2 -1.1 -0.2 1.5]

Usando uma calculadora, calculamos os seguintes valores:

a. média = (2.5 + 8.2 - 1.1 - 0.2 + 1.5)/5 = 2.18
b. variância = $[(2.5 - \mu)^2 + (8.2 - \mu)^2 + (-1.1 - \mu)^2 + (-0.2 - \mu)^2 + (1.5 - \mu)^2]/4$ = 13.307
c. desvio padrão = $\sqrt{13.307} = 3.648$
d. média quadrática = $(2.5^2 + 8.2^2 + (-1.1)^2 + (-0.2)^2 + 1.5^2)/5 = 15.398$
e. 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 um 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’) Rand (‘normal’)	Especificam se os números aleatórios desejados serão uniformemente ou normalmente distribuídos.
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 ¼, pois neste intervalo corresponde a ¼ do total. Então,

$$P\ (270^0 \leq X \leq 360^0) = ¼$$

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

$$P(a \leq X \leq b) = \frac{b-a}{360^0}$$

Construindo um histograma da variável X:

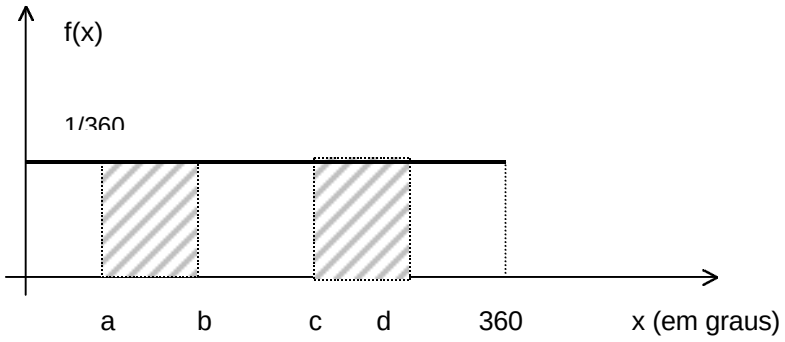


Figura 5.7 – histograma da variável X

O histograma corresponde à função:

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

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

$$P(a \leq X \leq b) = \int_a^b f(x)dx = \int_a^b \frac{1}{360^0}dx = \frac{b-a}{360^0}$$

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 a e b (a<b) reais, se a sua f.d.p. é dada por:

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

- Gráfico:

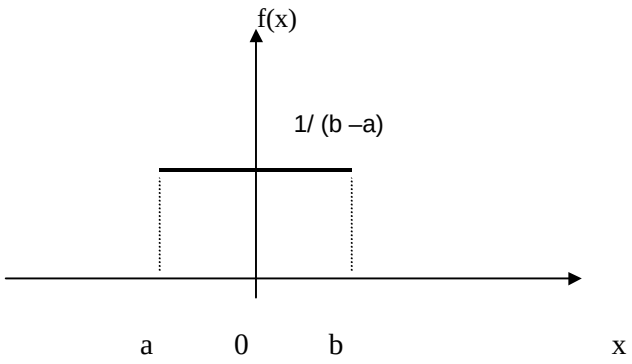


Figura 5.9 – distribuição uniforme

- 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: (μ, σ^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.

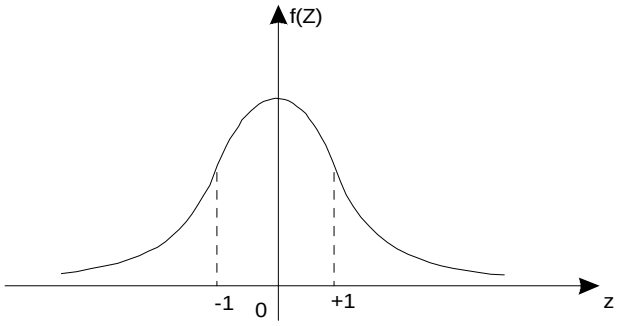


Figura 5.9 – Curva Normal Padrão

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

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

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

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 seqüência 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) . 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*SINAL) – 5

Exercícios

Determine os comandos do MATLAB necessários para gerar 10 números aleatórios com as características específicas. Verifique sua 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 -π.

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 . 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.

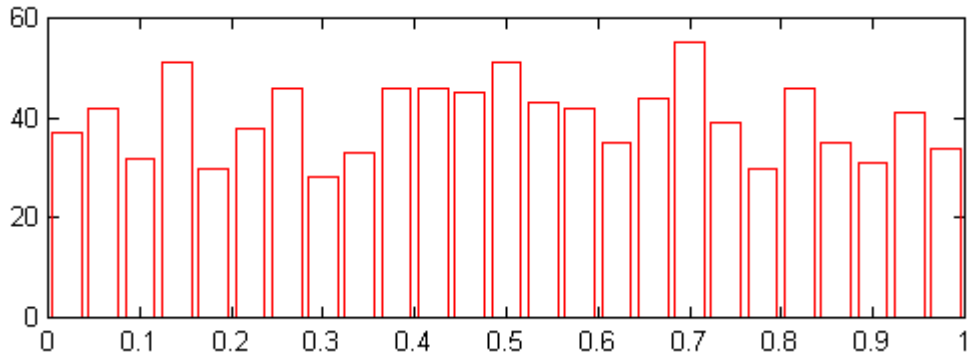


Figura 5.10 - Histograma para números aleatórios uniformemente distribuídos

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).

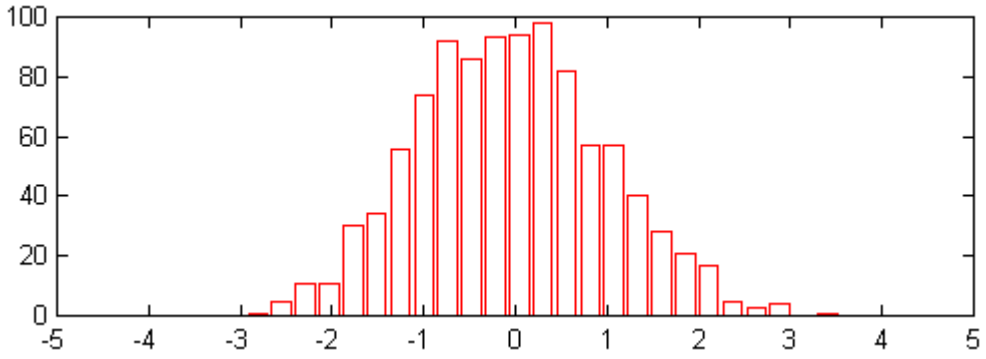


Figura 5.11 - Histograma para números aleatórios normalmente distribuídos

E x e r c í c i os

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.

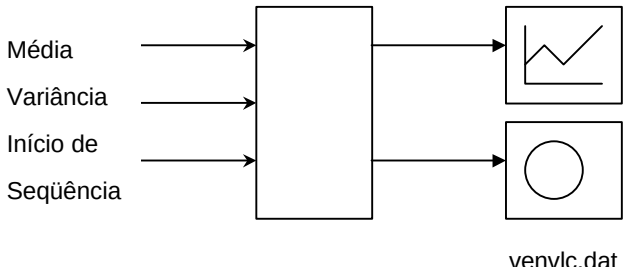
Escreva um programa MATLAB para gerar um seqüência de números aleatórios para simular uma hora de dados de velocidade do vento, o qual é atualizado a cada 10 segundos. (Uma hora de dados é então representada por 361 valores.) De uma análise do padrão real de vento, foi determinado que a velocidade do 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 10milhas 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 pelo 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 do 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:

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 4sen2*pt* é igual a 16/2, ou 8.

E x e r c í c i o s

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:

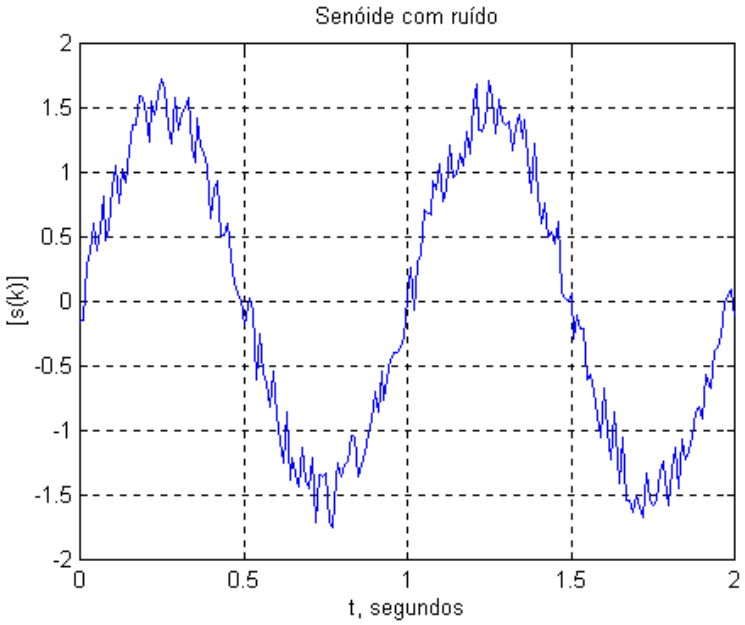
$$\text{Energia do ruído} = \frac{(1.5^2)/2}{46} = 0.024$$

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 freqüência 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. Ë 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.

E x e r c í c i o s

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.