Capítulo 6 – Operações com Matrizes

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

6.1 Operações com Matrizes

- Matrizes Transpostas

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

Exemplo 1

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

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

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

No MATLAB a matriz transposta é denotada por A'.

Somatório de Produtos

É a soma escalar de dois vetores do mesmo tamanho.

Somatório de produtos = A . B = $\sum_{i=j}^{N} aibi$

Exemplo 2

A = [4 –1 3] e B = [-2 5 2]
A . B = (4) . (-2) + (-1) . (5) + (3) . (2)
A . B = (-8) + (-5) + (6)
A . B = -7

Comando *sum*

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

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

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

Multiplicação de Matrizes

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

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

Exemplo 3

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

O primeiro elemento do produto C = A.B é

(2).(-1) + (5).(-1) + (1).(5) = -2

Logo a matriz C será:

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

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

No MATLAB podem ser usados os seguintes comandos:

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

Matriz Power

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

Matriz Inversa

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

Se considerarmos duas matrizes A e B:

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

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

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

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

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

Exercícios para Praticar!

Sejam as matrizes:

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

Calcule:

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

Determinante

Seja a matriz

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

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

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

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

Aplicação à Solução de Problema: Peso Molecular de Proteínas

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

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

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

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

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

Lys Glu Met Asp Ser Glu

00010200000110010000

O nome do arquivo será chamado protein.dat.

1. PROBLEMA EM SI
   Calcular o peso molecular de um grupo de moléculas de proteínas.

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

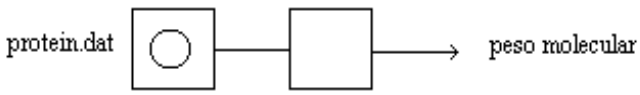


Diagrama de Entrada e Saída

3. SOLUÇÃO NO MATLAB

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

6.2 Manipulações com Matrizes

Comando *rot90*

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

Exemplo 4

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

B = rot90(A)

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

C = rot90(A,2)

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

Comando *fliplr*

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

Comando *flipud*

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

Exemplo 5

Seja a matriz A:

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

B = fliplr(A)

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

C = flipud(B)

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

Comando *reshape*

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

Exemplo 6

Seja a matriz A:

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

No MATLAB:

```
A = [2 5 6 -1;3 -2 10 0];
B = reshape(A,4,2);
C = reshape(A,1,8);
```

Comando *diag*

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

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

B = diag(A)

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

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

Exemplo 7

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

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

Comando *triu*

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

Exemplo 8

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

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

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

Comando *tril*

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

Exemplo 9

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

B = tril(A)

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

Aplicação à Solução de Problema: Alinhamento de Imagem

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

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

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

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

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

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

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

Supondo que:

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

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

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

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

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

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

Considerando o par de matrizes:

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

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

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

Isto acontece porque os valores se cancelam.

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

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

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

1. O PROBLEMA EM SI
   Determinar a melhor rotação de 90° no alinhamento de duas imagens.

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

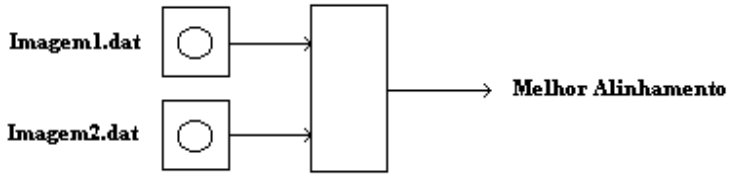


Diagrama de entrada e saída

4. UM EXEMPLO PARA AUXILIAR

Supor as duas matrizes:

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

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

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

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

5. SOLUÇÃO NO MATLAB

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

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

Capítulo 7 – Gráficos

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

7.1 Gráficos X – Y

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

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

Coordenadas Retangulares

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

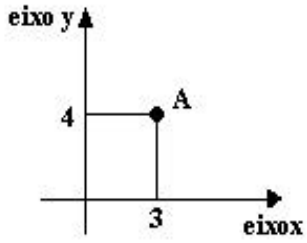


fig7.1 - coordenadas cartesianas

Legenda

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

Title(text) – Este comando escreve títulos no topo do gráfico plotado.
Xlabel(text) – Este comando escreve um texto abaixo do eixo x do gráfico plotado.
Ylabel(text) – Este comando escreve um texto ao lado do eixo y do gráfico plotado.
Text(x, y, text) – Este comando escreve um texto na tela do gráfico no ponto específico das coordenadas (x, y) usando os eixos dos gráficos. Se x e y são vetores o texto é escrito a cada ponto.
Text(x, y, text, sc) – Este comando escreve um texto na tela do gráfico no ponto especificado pelas coordenadas (x, y), assumindo que a esquina esquerda inferior é (0,0), e a esquina direita superior é (1,1).
gtext(text) – Este comando escreve um texto nas posições indicadas na tela do gráfico pelo mouse.
grid – Este comando acrescenta grades no gráfico plotado.

- *Comandos de plotar*

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

Os comandos para plotar gráficos lineares e logarítmico são:

plot(x, y) – Este comando gera gráficos lineares com valores de x e y, onde x representa a variável independente e y representa a variável dependente.
Semilogx(x, y) – Este comando gera gráfico usando escala linear para y e escala logarítmica para x.
Semilogy(x, y) – Este comando gera gráficos usando escala linear para x e escala logarítmica para y.
Loglog(x, y) – Este comando gera gráficos com escala logarítmica para ambos os eixos x e y.

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

7.2 Gráficos Polares

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

Coordenadas polares

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

O comando no MATLAB para gerar gráficos polares é:

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

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

```
theta = 0:2*pi / 100 : 2*pi;
r = theta / (2*pi);
polar(theta,r);
```

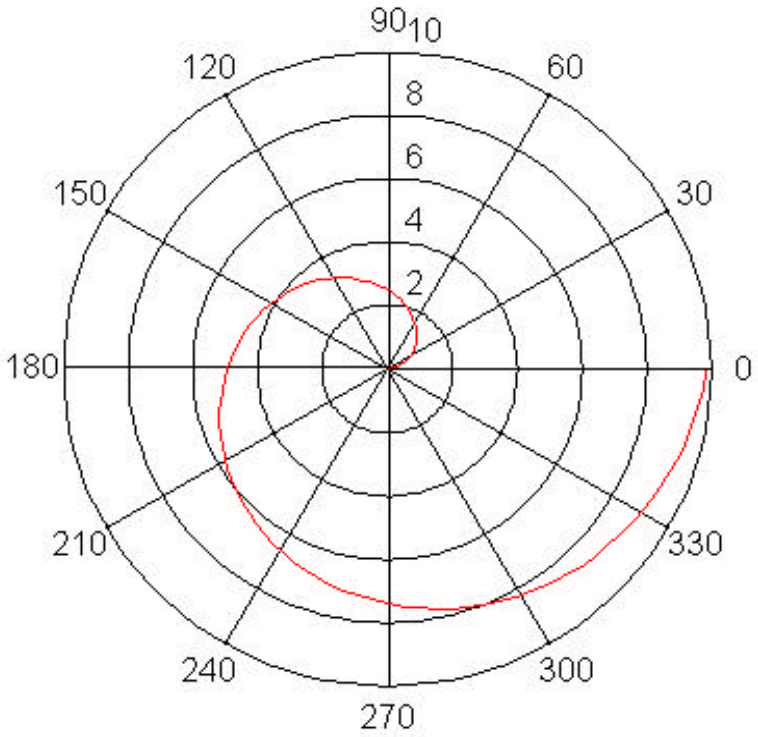


fig7.2 - gráfico polar

Transformação retangular / polar ; polar / retangular

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

As equações abaixo relacionam os sistemas polar e retangular:

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

E x e r c í c i o s

1) Converter de coordenada retangular para coordenada polar:
   a) (3, -2);
   b) (0.5, 1);

2) Converter de coordenada polar para coordenada retangular:
   a) (π, 1);
   b) (2.3, 0.5);

Gráficos de barras e degraus

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

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

Exemplo: a figura 7.3 mostra um gráfico de barra;

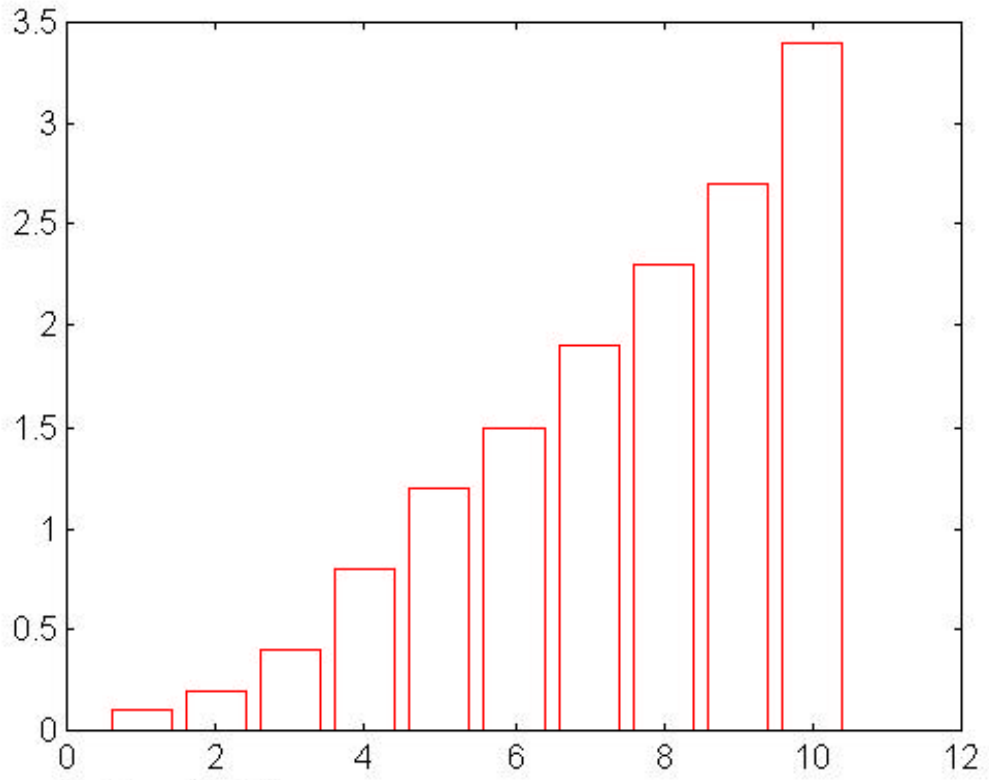


fig.7.3 - gráfico de barra

7.3 Opções

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

plot(x, y, w, z);

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

- *Estilo de linha e marcação*

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

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

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

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

Podemos também escolher as cores que serão usadas:

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

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

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

· *Escala*

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

axis - Este comando congela a escala de eixos para uma subseqüência de gráficos. A Segunda execução do comando retorna o sistema a escala automática.
axis(v)- v é um vetor de quatro elementos que contém a escala de valores,[xmin,xmax,ymin,ymax].

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

- *Subplot*

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

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

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

- *Controle de tela*

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

E x e r c í c i o

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

a) Gerar o gráfico linear desta função;
b) Gerar o gráfico desta função com escala logarítmica x;
c) Gerar o gráfico desta função com escala logarítmica y;
d) Gerar o gráfico loglog desta função;
e) Comparar as vantagens e desvantagens dos gráficos;

Solução:

7.4 Gráficos 3D

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

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

Exemplo:

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

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

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

Solução:

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

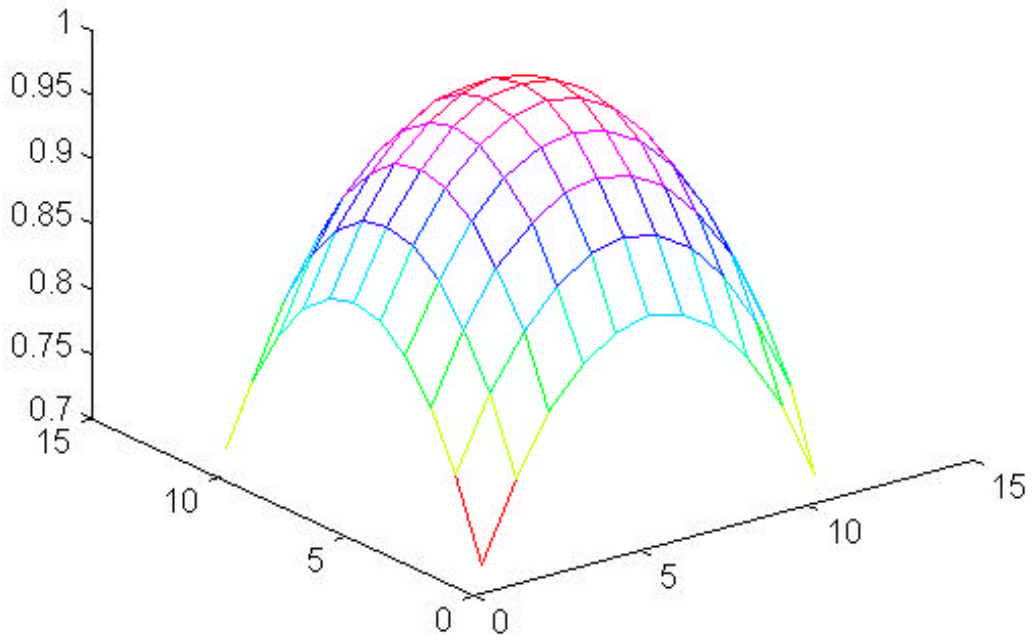


figura 7.4 - Gráfico 3D

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

Exemplo:

1) Rotação horizontal (figura 7.5):

Comando no MATLAB **:**

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

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

Solução:

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

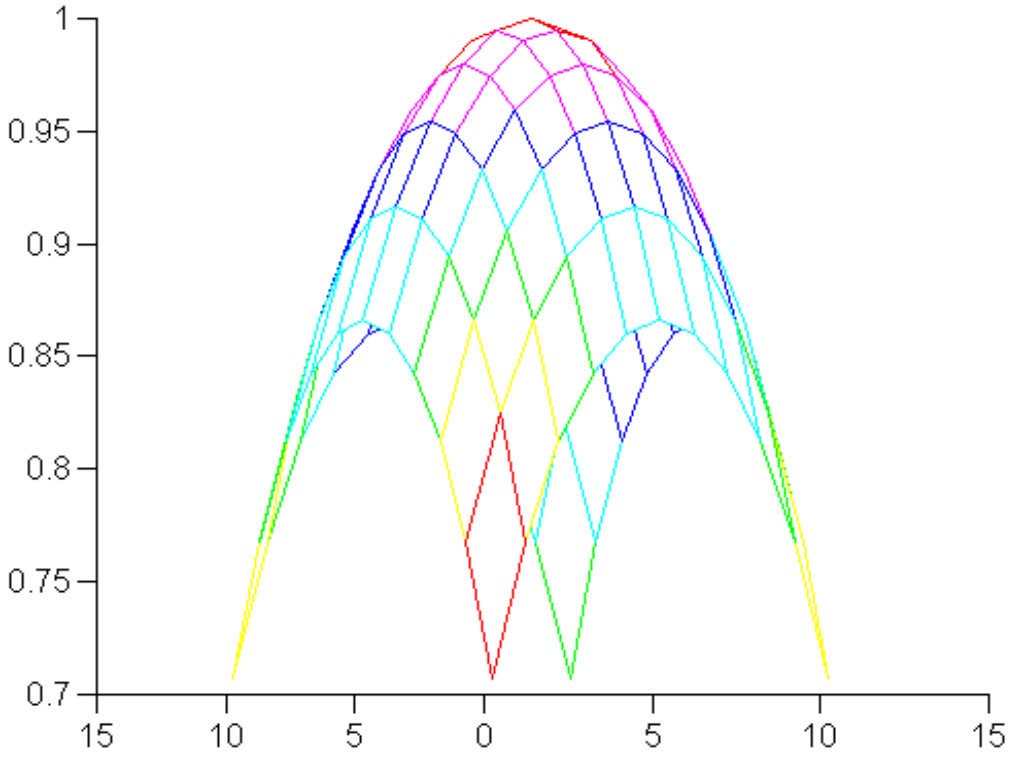


Figura 7.5 - Rotação horizontal

2) Rotação vertical (figura 7.6):

Comando no MATLAB

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

Solução:

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

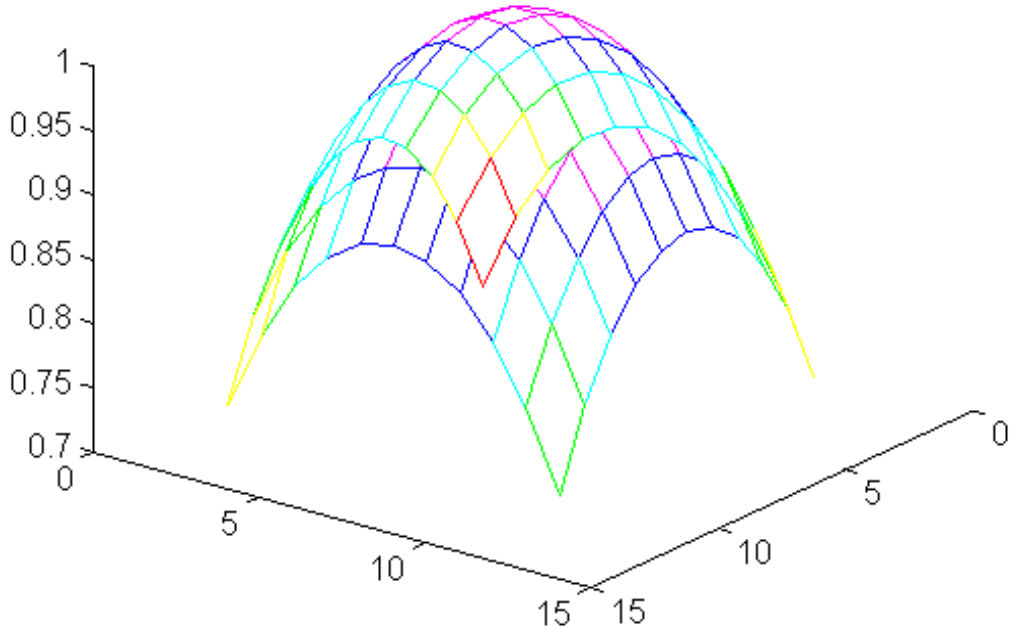


Figura 7.6 - Rotação vertical

E x e r c í c i o

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

Solução:

Aplicação a solução de problemas: Trajetória de um Satélite

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

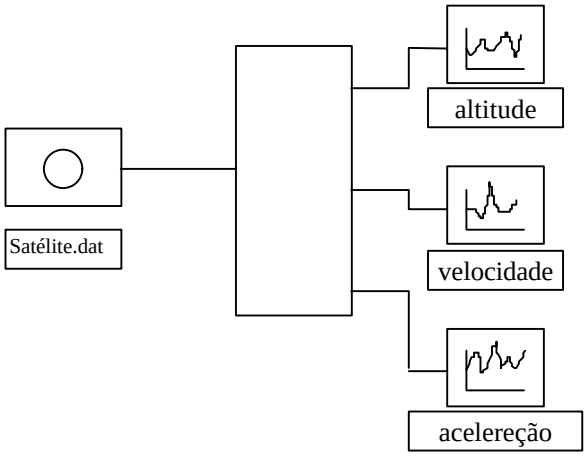
Para aumentar a bagagem científica de coleção de dados da parte mais elevada da atmosfera, os satélites auxiliam sistemas de telemetria para transmissão de informação.

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

1. PROBLEMA EM SI

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

2. DESCRIÇÃO ENTRADA / SAÍDA:



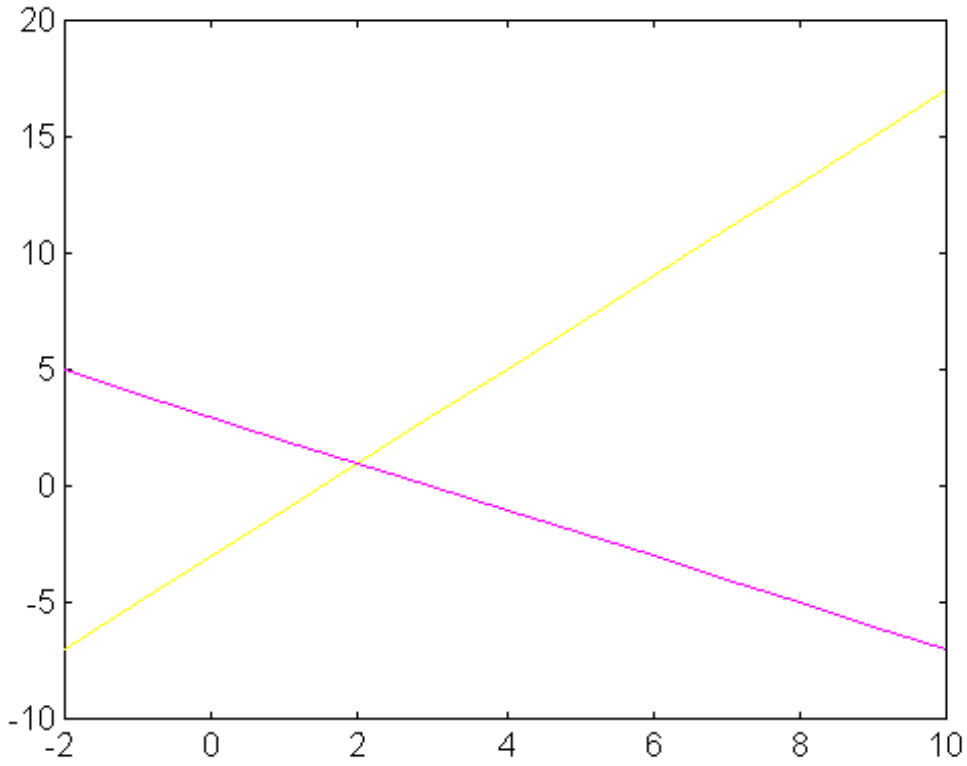
3. SOLUÇÃO MATLAB

Capítulo 8 - Solução a Sistemas de Equações Lineares

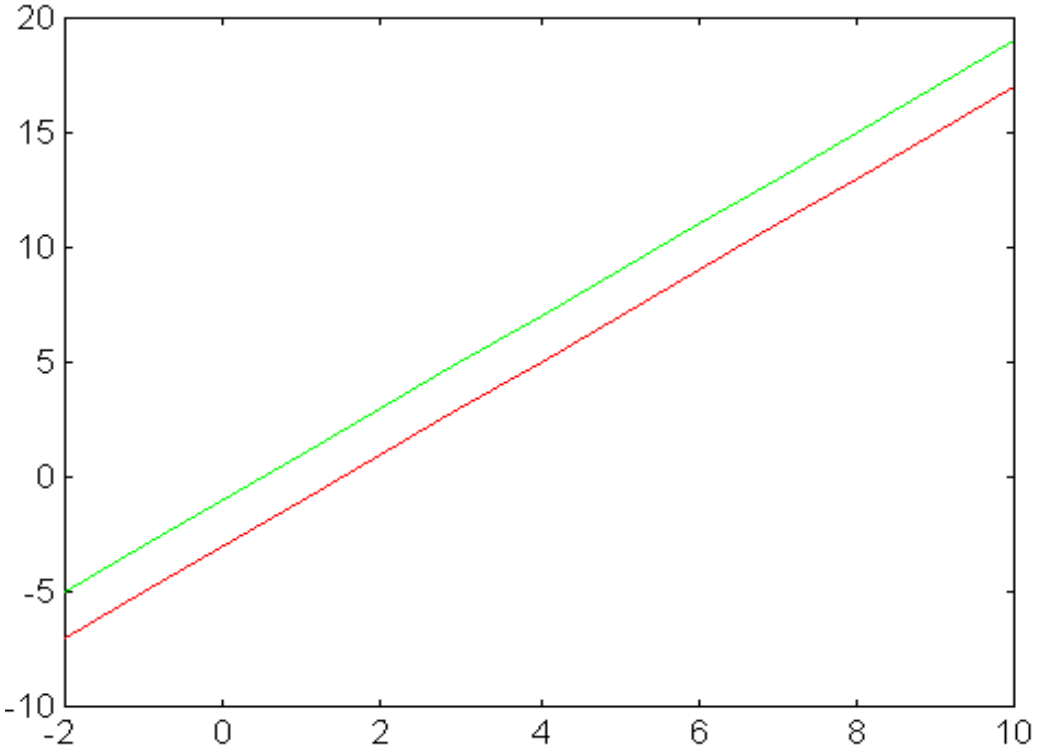
8.1 Interpretação gráfica

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

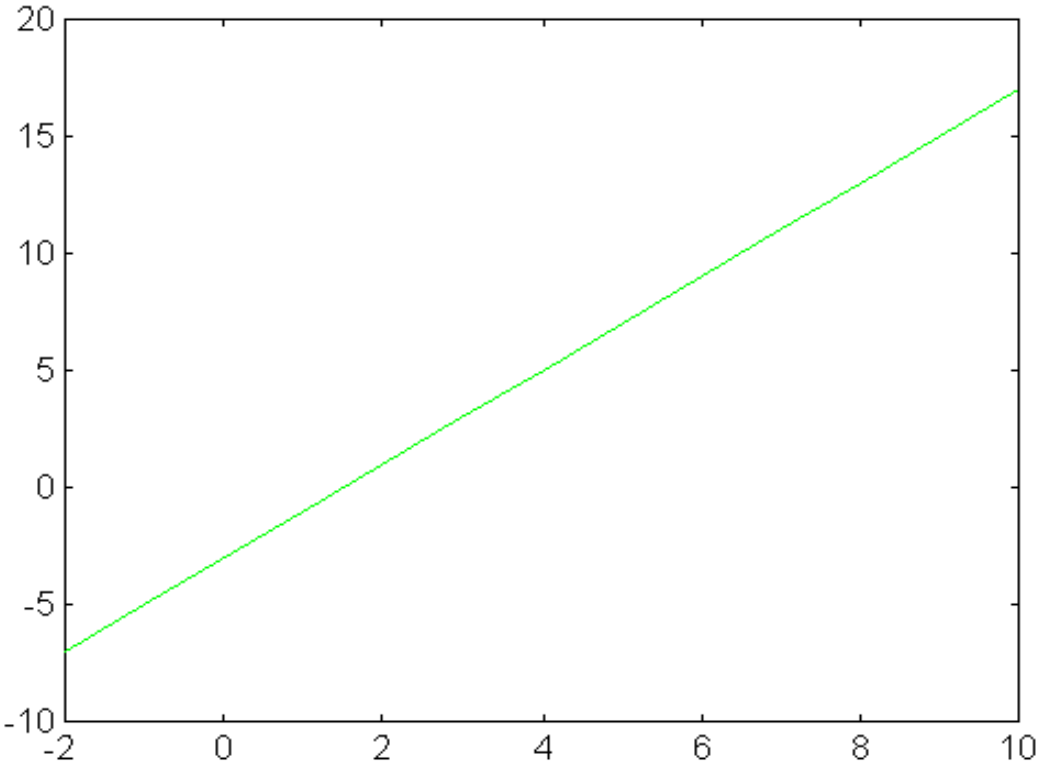
Uma equação linear com 2 variáveis, semelhante a 2x - y = 3, define uma linha reta e é escrita na forma y = mx + b, onde m é o coeficiente angular e b o coeficiente linear. Podemos escrever y = 2x - 3. Se tivermos 2 equações lineares, elas podem representar 2 diferentes retas que se interceptam em um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



(a) - Retas que se interceptam.



(b) - Retas paralelas



(c) - Retas iguais - fig8.1

Equações que representam duas retas que se interceptam podem ser facilmente identificadas porque possuem diferentes coeficientes angulares.
Exemplo: y = 2x -3 ; y = -x +3;

Equações que representam duas retas paralelas possuem o mesmo coeficiente angular e coeficientes lineares diferentes.
Exemplo: y = 2x - 3 ; y = 2x + 1;

Equações que representam a mesma reta são equações com mesmo coeficiente angular e mesmo coeficiente linear.
Exemplo: y = 2x - 3 ; 3y = 6x - 9;

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

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

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

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

8.2 Solução usando operação com matrizes

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

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

Podemos rescrever os sistemas de equações usando as seguintes matrizes:

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

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

$$Ax = B$$

Divisão de matrizes

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

```
A = [3, 2, -1; -1, 3, 2; 1, -1, -1];
B = [10; 5; -1];
x =A \ B;
```

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

Matriz inversa

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

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

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

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

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

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

No MATLAB podemos calcular essa expressão usando o comando:

```
X = inv(A)*B;
```

E x e r c í c i o s

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

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

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

Aplicação a solução de problemas: Análise de circuito elétrico

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

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

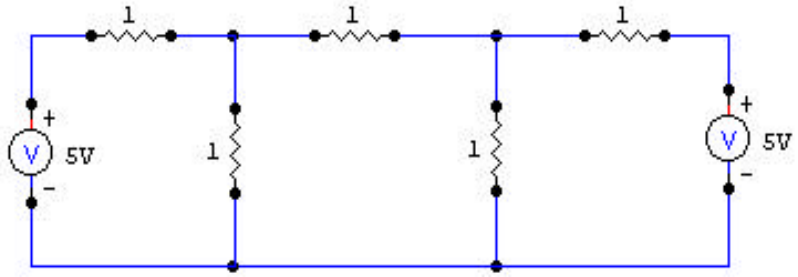
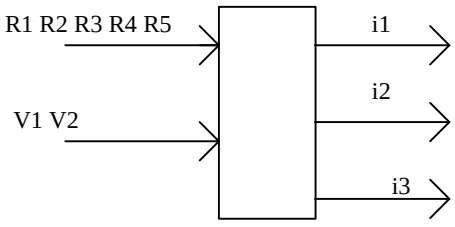


Figura 8.2 - Circuito com duas fontes de voltagem

$$\begin{bmatrix} (R1+R2)i1 & -R2i2 & +0i3 & = & V1 \\ -R2i1 & +(R2+R3+R4)i2 & -R4i3 & = & 0 \\ 0i1 & -R4i2 & +(R4+R5)i3 & = & -V2 \end{bmatrix}$$

Problema: Calcular as três correntes do circuito da figura 8.2 considerando os valores da figura para R1, R2, R3, R4, R5, V1 e V2.

Entrada / Saída:



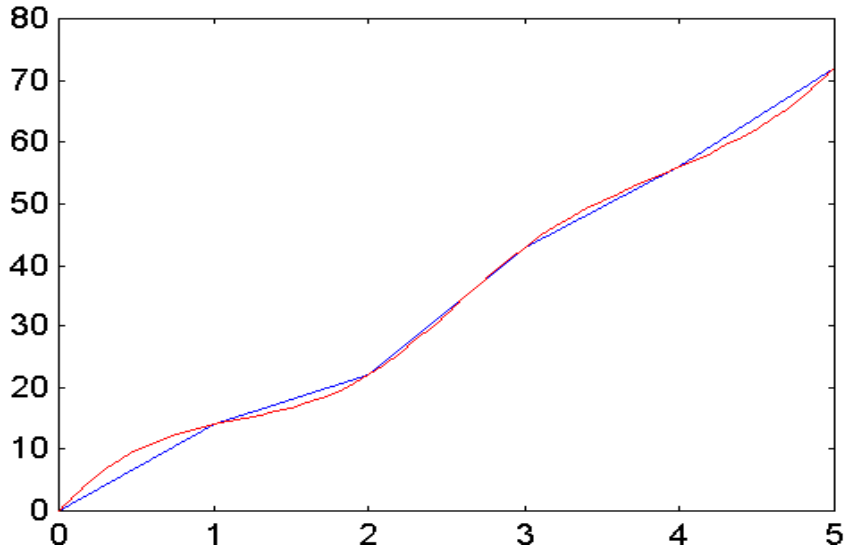
Capítulo 9 - Ajuste de Curvas e Interpolação

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

9.1 Interpolação

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

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



Interpolação linear

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

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

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

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

Comando table1

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

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

Exemplo 1

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

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

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

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

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

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

Os valores correspondentes serão y1 = 64.8 e y2 = 106.7.

Supondo agora que medimos a temperatura em três pontos do cilindro:

Tempo, s	T1	T2	T3
0	0	0	0
1	20	25	52
2	60	62	90
3	68	67	91
4	77	82	93
5	110	103	96

Guardando estas informações numa matriz, com as informações do tempo na primeira coluna:

```
dado2(:,1) = [ 0,1,2,3,4,5] ’;
dado2(:,2) = [0,20,60,68,77,110] ’;
dado2(:,3) = [0,25,62,67,82,103]’;
dado2(:,4) = [0,52,90,91,93,96]’;
```

Para determinar valores das temperaturas nestes três pontos no tempo de t = 2.6s, usamos os seguinte comando:

```
temps = table1 (dado2, 2.6);
```

Onde temps será um vetor contendo os três valores da temperatura: 64.8, 65.0 e 90.6.

Comando table2

Esse comando possibilita a interpolação bidimensional usando valores da primeira coluna e da primeira linha da tabela. É importante perceber que tanto os elementos da primeira coluna quanto os elementos da primeira linha devem estar ordenados crescentemente ou decrescentemente e que os valores de x e de y devem permanecer entre os limites da tabela.

Supomos agora que iniciamos um determinado processo incrementando uma velocidade constante dada em rotações por minuto, enquanto medimos a temperatura em um ponto da cabeça do cilindro. Então, se iniciarmos o processo e incrementarmos uma velocidade 2000 rpm em 5 segundos e registrarmos os valores de temperatura. Da mesma forma podemos continuar registrando os valores de temperaturas para os vários valores de velocidade:

Tempo, s	V1=2000	V2=3000	V3=4000	V4=5000	V5=6000
0	0	0	0	0	0
1	20	110	176	190	240
2	60	180	220	285	327
3	68	240	349	380	428
4	77	310	450	510	620
5	110	405	503	623	785

Desta forma podemos estimar a temperatura da cabeça do cilindro em qualquer tempo entre 0 e 5 segundos, e em qualquer velocidade entre 2000 e 6000 rpm.

Ao invés de calcularmos, o que seria bem mais complicado, podemos interpolar a função em questão.

Podemos agora guardar estas informações numa matriz dado3, e então usar o comando *table2* para calcular esta informação para nós:

Note que agora nós preenchemos as linhas com as informações da tabela, no exemplo anterior nós preenchemos as colunas.

```
dado3(1,:) = [0,2000,3000,4000,5000,6000];
dado3(2,:) = [0,0,0,0,0,0];
dado3(3,:) = [1,20,110,176,190,240];
dado3(4,:) = [2,60,180,220,285,327];
dado3(5,:) = [3,68,240,349,380,428];
dado3(6,:) = [4,77,310,450,510,620];
dado3(7,:) = [5,110,405,503,623,785];
temp = table2(dado3,3.1,3800)
```

A resposta será mostrada em temp = 336.68 F .

- *Spline*

Uma spline cúbica é uma curva suave construída passando através do conjunto de pontos. A curva entre cada par de pontos é determinada por um polinômio do terceiro grau, que é calculado para fornecer uma curva suave entre os pontos ao invés de ligá-los simplesmente.

Comando spline

É o comando que realiza no MATLAB uma spline cúbica. O primeiro argumento do comando *spline* é o x, o segundo é o y e o terceiro contém o valor do(s) ponto(s) aonde se deseja o valor da função. Lembrando que novamente os valores de x devem ser ordenados ou crescentemente ou decrescentemente, caso contrário surgirá uma mensagem de erro!

Exemplo 2

Supondo que queiramos usar a spline cúbica para calcular a temperatura na cabeça do cilindro no tempo t = 2.6 segundos, podemos usar os seguintes comandos:

```
x = [0,1,2,3,4,5];
y = [0,20,60,68,77,110];
temp1 = spline(x,y,2.6)
```

O valor de temp1 será 67.3.

Se quisermos usar estes processo para calcularmos a temperatura em diferentes momentos podemos usar os seguintes comandos:

```
temp2 = spline(x,y,[2.6,4.9]);
temp2 = [67.3,105.2]
```

Se quisermos ainda plotar uma curva spline abrangendo um outro intervalo de valores, podemos gerar um vetor x como o terceiro argumento do comando *spline*.

Exemplo 3

```
x = [0,1,2,3,4,5];
y = [0,20,60,68,77,110];
newx = 0: 0.1 :5;
newy = spline(x,y,newx);
axis([-1,6,-20,120]);
plot (x,y,newx,newy,x,y,´o’);
title (‘ Interpolação Spline ’);
xlabel(‘ Tempo,s ’);
ylabel(‘ Graus, F ’);
grid;
```

Note que na interpolação linear, o gráfico de x e y percorrem as coordenadas por meio de retas, enquanto que o gráfico de newx e newy representa a spline definida por interpolação cúbica.

Exercícios para Praticar!

Supondo que nossa tabela de valores seja;

Tempo,s	Temp, F
0,0	72,5
0,5	78,1
1,0	86,4
1,5	92,3
2,0	110,6
2,5	111,5
3,0	109,3
3,5	110,2
4,0	110,5
4,5	109,9
5,0	110,2

a. Gerar um gráfico que compare os dois tipos de interpolação já vistos.

b. Achar os valores da temperatura correspondentes aos seguintes valores de tempo t = [0.3,1.25,2.36,4.48] , usando a interpolação linear.
c. Achar os valores da temperatura correspondentes aos seguintes valores de tempo t = [0.3,1.25,2.36,4.48] , usando a spline.

Aplicação à Solução de Problemas : Braço Robótico

Assim como este sistema de manipulação existem vários outros usados em vários tipos de robôs, que se utilizam de um avançado sistema de controle para guiar um braço robótico para a posição desejada. Um dos anseios de um sistema de controle é que o caminho percorrido pelo braço ao se mover de um local para o outro, ao pegar ou soltar um objeto, seja feito regularmente, evitando assim possíveis 'trancos' durante o percurso.

O caminho percorrido pelo braço será definido através de coordenadas de pontos por onde o braço irá se mover. Então podemos utilizar a interpolação para definir uma curva suave, regida por estas coordenadas, para mostrar o comportamento desse braço ao longo de uma trajetória.

Uma parte importante no desenvolvimento do algoritmo ou da solução deste problema está na consideração de situações especiais. Neste problema nós assumimos que pontos nos quais o braço irá passar precisarão estar na ordem para mover o braço na trajetória desejada que será: posição inicial, posição intermediária, posição para pegar o objeto, posição para colocar o objeto no local desejado e finalmente posição inicial. E, consideraremos também que cada ponto conterá três coordenadas: x, y(que serão as coordenadas relativas a posição inicial), e uma terceira coordenada dizendo o código da respectiva posição, de acordo com a tabela abaixo:

Código	Posição
0	Inicial
1	Intermediária
2	Para pegar o objeto
3	Para deixar o objeto

Queremos então utilizar uma spline para visualizarmos o comportamento do braço robótico.

Método para a resolução do problema

1. PROBLEMA EM SI
   Desenhar uma curva suave utilizando a interpolação por spline que pode ser usada para guiar um braço robótico para uma determinada trajetória.

2. DESCRIÇÃO DA ENTRADA E DA SAÍDA

A entrada é constituída de um arquivo contendo as coordenadas x e y dos pontos pelos quais o braço robótico deverá passar.

A saída do programa será a curva correspondente ao comportamento do robô ao percorrer estes pontos.

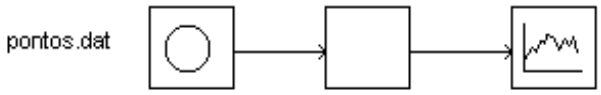


Diagrama de entrada e saída

3. SOLUÇÃO NO MATLAB

9.2 Ajuste de curvas pelo método dos mínimos quadrados

Supondo que tenhamos um conjunto de pontos originados de um determinado experimento e que queiramos plotar o seu gráfico. Se tentarmos traçar uma única reta entre esses pontos, somente um par destes pontos irão fazer parte da reta. O método dos mínimos quadrados poderá ser usado neste caso para achar uma única reta que mais se aproxime de todos os pontos. Embora essa reta seja a melhor aproximação possível, pode acontecer da reta não passar efetivamente por nenhum ponto.

Note que este método é muito diferente da interpolação porque esta passará por todos os pontos.

Vamos partir primeiro para a discussão do ajuste da reta para um conjunto de pontos e depois para o ajuste do polinômio através do conjunto de pontos.

Regressão linear

É o processo que determina a equação linear, ou seja, a função mais aproximada do comportamento dos pontos, que é calculada através do somatório dos mínimos quadrados das distâncias entre a reta e os pontos.

Como exemplo vamos ainda considerar aqueles valores de temperaturas do cilindro:

```
x = [0,1,2,3,4,5];
y = [0,20,60,68,77,110];
axis([-1,6,-20,120]);
```

Se simplesmente plotarmos o gráfico através do comando:

plot(x,y,x,y, ‘o’);

Ele ligará os pontos. Mas, se ao invés disso, estimarmos o comportamento da função em y1 = 20*x, e aí sim plotarmos este gráfico:

plot(x,y1,x,y, ‘o’)

Para medirmos a qualidade desta estimativa, devemos determinar a distância no eixo vertical de cada ponto à reta estimada e somá-las através do comando sum. Observe que somamos os quadrados das distâncias para evitar que algum valor seja anulado devido aos sinais.

somadist = sum ((y - y1) .^ 2);

Para achar a reta mais perto de todos os pontos devemos achar a menor soma dos quadrados das distâncias. Para isto devemos escrever a equação geral da reta : y = mx + b.

Os valores de m e b poderão ser calculados através do comando *polyfit*

Comando polyfit

Este comando acha os coeficientes do polinômio que estamos procurando. Mas, para isto devemos especificar o grau do polinômio. Este comando possui três argumentos: primeiro as coordenadas x e y, e depois o grau do polinômio.

Exemplo:

```
x = [0,1,2,3,4,5];
y = [0,20,60,68,77,110];
coef = polyfit(x,y,1);
m = coef (1);
b = coef (2);
ybest = m*x+b;
somadist = sum ((y - ybest) .^ 2 );
axis([-1,6,-20,120]);
plot(x,ybest,x,y, ‘o’ );
title (‘       ’)
xlabel (‘X’); ylabel(‘Y’);
grid;
```

Comando polyval

Este comando é empregado para estimar o mínimo polinômio quadrado de um conjunto de pontos. O primeiro argumento deste comando conterá os coeficientes do polinômio, o segundo argumento será um vetor com os valores de x para os quais desejamos o valor da função.

Exemplo:

```
ybest = polyval (coef,x);
```

Capítulo 10 - Análise polinomial

Este capítulo traz uma série de comandos no MATLAB para a análise polinomial. Primeiro vamos discutir meios de avaliar os polinômios e como trabalhar o seu comportamento. Uma aplicação deste conceito está na modelagem da altitude e velocidade de um balão. A seguir definiremos as raízes dos polinômios.

Polinômios normalmente aparecem em aplicações da Engenharia e na Ciência em geral porque eles constituem ainda bons modelos para representar sistemas físicos.

10.1 Avaliação do polinômio

Como exemplo vamos tomar o seguinte polinômio:

$$f(x) = 3x^4 - 0.5x^3 + x - 5.2$$

Se x assumir valores escalares, podemos escrever:

```
f(x) = 3*x ^4 - 0.5*x ^3 + x - 5.2;
```

Se x for um vetor ou uma matriz devemos escrever:

```
f(x) = 3* x .^4 - 0.5* x .^3 +  x - 5.2;
```

onde o tamanho da matriz f será o mesmo da matriz x.

Comando polyval

Este comando possui dois argumentos. O primeiro argumento contém os coeficientes do polinômio em questão e o segundo argumento contém a matriz para a qual desejamos avaliar o polinômio.

Exemplo 1

```
a = [3,-0.5,0,1,-5.2];
f = polyval(a,x);
```

Esses comandos também podem ser combinados em um só:

```
f = polyval([3,-0.5,0,1,-5.2],x);
```

O tamanho de f terá que ser igual ao tamanho de x, seja ele um escalar, vetor ou matriz.

Supondo que queiramos o valor da função $g(x) = -x^5 + 3x^3 - 2.5x^2 - 2.5$, para x no intervalo de[0,5]:

```
x:0:0.025:5;
a = [-1,0,3,-2,5,0,-2.5];
g = polyval(a,x)
```

Quando x for um escalar ou um vetor, *polyval* consegue calcular o valor da função operando elemento por elemento. Mas quando x for uma matriz usa-se o comando *polyvalm*:

```
f = polyvalm(a,x);
```
sendo a matriz x, uma matriz quadrada.

Operações Aritméticas

Podemos trabalhar com polinômios armazenando seus coeficientes em vetores, e trabalhar apenas com estes vetores.

- Soma e subtração

Para somar ou subtrair polinômios basta somar ou subtrair seus respectivos coeficientes. O MATLAB não apresenta um comando específico par somar polinômios. A soma ou subtração padrão funciona se ambos os vetores polinomiais forem do mesmo tamanho. Somemos os polinômios a seguir:

$g(x) = x^4 - 3x^2 - x + 2.4$

$h(x) = 4x^3 - 2x^2 + 5x - 16$

$som(x) = g(x) + h(x)$

$sub(x) = g(x) - h(x)$

Para multiplicar um polinômio por um escalar (sendo ele positivo ou negativo), basta definir o polinômio pelos seus coeficientes e efetuar a multiplicação. Multipliquemos o polinômio:

$g(x) = 3\ f(x)$

No MATLAB:
```
f = [ 3,-6,1];
g = 3 * f
```

- Multiplicação

A multiplicação polinomial é efetuada por meio do comando *conv* (que faz a convolução entre dois conjuntos). A multiplicação de mais de dois polinômios requer o uso repetido de *conv*.

```
m = conv(g,h)
```

- Divisão

No MATLAB a divisão de polinômios é feita através do comando *deconv*:

```
[q,r] = deconv(g,h)
```

Esse resultado nos diz que g dividido por h nos dá o polinômio de quociente q e resto r.

Aplicação à Solução de Problemas: Balões Meteorológicos

Balões são usados para reunir problemas de temperatura e pressão nas diferentes altitudes da atmosfera. O balão consegue ganhar altitude porque nele está presente um gás de menor densidade que o próprio ar ao seu redor. Durante o dia, devido a presença da luz solar, o gás Hélio se expande, se tornando mais denso que o ar e assim fazendo com que o balão suba. Durante a noite, o gás Hélio esfria e fica mais denso, e com isso o balão desce a baixa altitude. No dia seguinte o sol novamente esquenta o gás e o balão sobe. Com o passar dos dias, esse processo gera vários valores de altitude que geralmente podem ser aproximados por uma equação polinomial.

Assumindo que o seguinte polinômio represente a altitude em metros, durante as primeiras 48 horas de um balão:

$$h(t) = -0.12\ t^4 + 12\ t^3 - 380\ t^2 + 4100\ t + 220$$

onde t é medido em horas. O modelo polinomial para velocidade, obtido através da derivada, em metros por hora do balão é o seguinte:

$$v(t) = -0.48\ t^3 + 36\ t^2 - 760\ t + 4100$$

Método para a resolução do problema

1. PROBLEMA EM SI:

Usando o polinômio dado fazer o gráfico da altitude e da velocidade do balão em questão. E achar também a máxima altitude por ele atingida.

2. DIAGRAMA ENTRADA/SAÍDA:

Neste diagrama é mostrado que não existe nenhuma entrada externa ao programa. A saída consiste em dois gráficos e na altitude máxima atingida e o seu tempo correspondente.

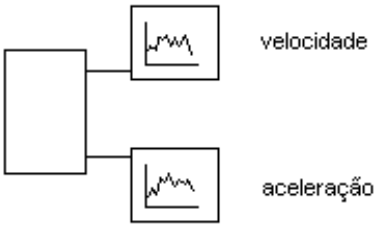


Diagrama de entrada e saída

3. SOBRE O PROGRAMA:

Queremos que apenas o programa faça o gráfico de acordo com as nossas informações e então calcule o máximo valor atingido no gráfico. Devemos também fazer que nosso programa converta metros por hora em metros por segundo.

4. SOLUÇÃO NO MATLAB:

Vamos usar o comando *polyval* para gerar os pontos para formar o gráfico. O comando *max* é usado para determinar o valor máximo da função.

10.2 Raízes de polinômios

Achar as raízes de um polinômio, isto é, os valores para os quais o polinômio é igual a zero, é um problema comum em muitas áreas do conhecimento, como por exemplo, achar as raízes de equações que regem o desempenho de um sistema de controle de um braço robótico, ou ainda equações que demonstram a arrancada ou freada brusca de um carro, ou analisando a resposta de um motor, e analisando a estabilidade de um filtro digital.

Se assumirmos que os coeficientes (a_1, a_2, ...) de um polinômio são valores reais, poderemos encontrar raízes complexas.

Se um polinômio é fatorado em termos lineares, fica fácil de identificar suas raízes, igualando cada termo a zero.

Um exemplo consiste no polinômio:

$f(x) = x^2 + x - 6$,

que ao ser fatorado se torna:

$f(x) = (x - 2) . (x + 3)$

As raízes da equação são os valores de x para os quais a função f(x) é igual a zero, ou seja, x = 2 e x = -3.

No gráfico, as raízes são valores onde a função corta o eixo x.

Um polinômio do terceiro grau tem exatamente três raízes que podem ser:

- três raízes reais;
- três raízes iguais;
- uma raiz real e duas raízes iguais;
- uma raiz real e um par conjugado de raízes complexas.

Se a função f(x) for um polinômio de grau n, ela terá exatamente n raízes. Estas n raízes podem conter múltiplas raízes ou raízes complexas.

No MATLAB, um polinômio é representado por um vetor linha dos seus coeficientes em ordem decrescente. Observe que os termos com coeficiente zero têm de ser incluídos. Dada esta forma, as raízes do polinômio são encontradas usando-se o comando *roots* do MATLAB.

Já que tanto um polinômio quanto suas raízes são vetores no MATLAB, o MATLAB adota a convenção de colocar os polinômios como vetores linha e as raízes como vetores coluna. Para ilustrar este comando vamos determinar as raízes do seguinte polinômio:

$f(x) = x^3 - 2x^2 - 3x + 10$

No MATLAB:

```
p = [1,-2,-3,10];
r = roots(p)
```

Lembrando que estes comandos podem ser dados de um só vez:

```
r = roots([1,-2,-3,10]);
```

Os valores das raízes serão: 2 + i, 2 - i e -2.

Agora, dadas as raízes de um polinômio, também é possível construir o polinômio associado. No MATLAB, o comando *poly* é encarregado de executar essa tarefa.

onde o argumento do comando *poly* é o vetor contendo as raízes do polinômio que desejamos determinar.

Exemplo 2

Sejam as raízes de um polinômio -1, 1 e 3. Determinar este polinômio.

No MATLAB:

```
a = poly ([-1,1,3]');
```

Exemplo 3

Determine as raízes dos seguintes polinômios e plote seu gráfico, com seu eixo apropriado, com o objetivo de verificar se o polinômio atravessa o eixo x bem nos locais das raízes.

a. $f(x) = x^3 - 5x^2 + 2x + 8$
b. $g(x) = x^2 + 4x + 4$
c. $h(x) = x^5 + 3x^4 - 11x^3 + 27x^2 + 10x - 24$
d. $i(x) = x^5 - 3x^3 + 4x^2 - 1$

Capítulo 11 - Integração e Diferenciação Numérica

A integração e diferenciação são conceitos fundamentais usados para resolver um grande número de problemas na Engenharia e na Ciência. Enquanto muitos destes problemas se usam de soluções analíticas, muitos requerem soluções numéricas para serem entendidos.

11.1 Integração Numérica

A integral de uma função f(x) no intervalo [a,b], é definida como sendo a área sob a curva percorrida por f(x) entre a e b.

$$k = \int_a^b f(x)\,dx$$

A avaliação numérica de uma integral é também chamada de quadratura (enfoque geométrico). O MATLAB possui três comandos para calcular a área sob uma função, em um domínio finito, que são: *trapz*, *quad* e *quad8*.

Regra do Trapézio

Quando a área sob a curva pode ser representada por trapézios e o intervalo [a,b], dividido em n partes iguais, a área aproximada poderá ser calculada através da seguinte fórmula:

$$Kt = \frac{b - a}{2n} \left(f(x_0) + 2\, f(x_1) + \ldots + 2f(x_{n-1}) + f(x_n) \right)$$

onde os valores de xi representam os pontos no final da cada trapézio e $x_0 = a$ e $x_n = b$.

A estimativa da integral melhora quando usarmos um maior número de componentes (como por exemplo trapézios), para aproximar a área sob a curva, pois quanto menor for o intervalo da função a curva tende a uma reta.

Comando quadratura

O MATLAB possui dois comandos para desenvolver a integração numérica. O comando *quad* usa uma forma adaptada da regra de Simpson, enquanto o comando *quad8* usa uma forma adaptada da regra de Newton-Cotes. O comando *quad8* funciona melhor em certas funções com certos tipos de singularidades como por exemplo:

$$k = \int_0^1 \sqrt{x}\,dx$$

Lembrando que uma singularidade é um ponto no qual uma função ou sua derivada não são definidas ou tendem para o infinito. Ambas as funções escrevem na tela uma mensagem quando detectam uma singularidade, mas ainda assim o valor estimado da integral é retornado.

A forma mais simples do comando *quad* requer três argumentos: o primeiro argumento é o nome da função no MATLAB que reconhece a função que estamos tratando; o segundo e o terceiro argumento são os limites inferior e superior a e b da integral.

Exemplo 1

$$k = \int_a^b \sqrt{x}\,dx \qquad \text{para a e b não negativos}$$

$$K = \frac{2}{3}\,(b^{3/2} - a^{3/2})$$

Os comandos *quad* e *quad8* podem ainda assumir um quarto argumento que é a tolerância, que corresponde a precisão. Se a tolerância for omitida, o valor default 0.001 será assumido pelo MATLAB.

Exercício 1

Seja a função f(x) = | x |. Resolva as integrais abaixo usando o MATLAB e compare com os resultados obtidos a mão.

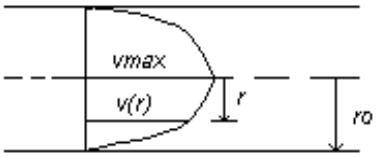
a. int[0.6,0.5] f(x) dx
b. int[0.5,-0.5] f(x) dx
c. int[0.0,-1.0] f(x) dx

Problema Aplicado: Análise de Escoamento de um Óleo num Oleoduto

A análise do fluxo de um líquido em duto tem aplicação em muitos sistemas diferentes, incluindo o estudo em veias e artérias no corpo humano, o sistema hidráulico de uma cidade, o sistema de irrigação de uma fazenda, o sistema de jato de tinta de uma impressora, etc.

O atrito de um fluxo ao passar num oleoduto circular gera a chamada velocidade de perfil no fluido.

O óleo que está em contato com as paredes do duto não está se movendo na mesma velocidade que o óleo no centro do fluido. O diagrama abaixo mostra como a velocidade do óleo varia de acordo com o diâmetro do duto e define as variáveis usadas para esta análise:



A velocidade de perfil é definida pela seguinte equação:

$$v(r) = vmax\ (1 - r / r_0)^{1/n}$$

onde n é um número inteiro entre 5 e 10 que define o contorno do escoamento do óleo. A velocidade média de escoamento do óleo pode ser calculada integrando-se a velocidade de perfil no intervalo de 0 a r_0.

$$A = 2\frac{vmax}{r0^2}$$

$$V = A\int_0^{r0} r\left(1 - \frac{r}{r0}\right)^{\left(\frac{1}{n}\right)} dr$$

Os valores de vmax e de n podem ser medidos experimentalmente, e o valor de r_0 é o próprio raio do tubo. Escreva um programa no MATLAB para integrar a velocidade de perfil e assim determinar a velocidade médio do óleo no duto.

Método Para a Resolução do Problema

1. O PROBLEMA EM SI
   Calcular a velocidade média do óleo em um duto.

2. DESCRIÇÃO DA ENTRADA E SAÍDA
   Os dados experimentais que serão tomados como entrada em nosso programa são a velocidade máxima vmax, o raio do duto r_0, e o valor de n.

   A saída de nosso programa será a velocidade média do óleo no duto.

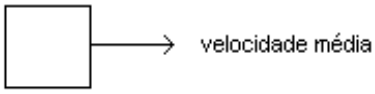


Diagrama de entrada e saída

3. FORMA QUE AJUDARÁ NA QUESTÃO

Plotar um gráfico da função $r\ (1 - r / r_0)^{1/n}$ e estimar o valor da integral através do cálculo da área sob a curva.

4. SOLUÇÃO NO MATLAB

11.2 Diferenciação Numérica

A derivada de uma função *f* em um ponto pode ser descrita graficamente como a inclinação da reta que tangencia a função naquele ponto.

Pontos da função onde a derivada é zero são chamados pontos críticos. São pontos onde a tangente é representada por uma linha horizontal e que, por isso, definem o local de máximo e de mínimo da função.

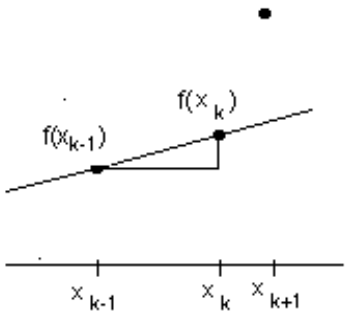
Podemos perceber ao analisar uma determinada função num determinado intervalo que o sinal da derivada pode mudar, e, se esse sinal muda, significa que dentro deste intervalo existe local de máximo e local de mínimo.

Podemos também analisar uma função pela sua derivada segunda. De modo que, se a derivada segunda de um ponto crítico é positiva, então o valor da função naquele ponto significa um local de mínimo. Da mesma forma, se a derivada segunda de um ponto crítico é negativa, então a função possui um local de máximo.

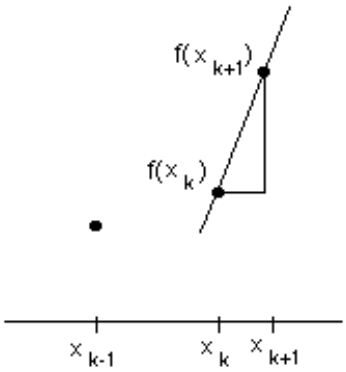
Derivação por expressões de diferenças

As técnicas de diferenciação numérica estimam a derivada de uma função em um ponto x_k através da aproximação da inclinação da reta tangente à curva neste ponto usando valores que a função assume em pontos perto de x_k. Essa aproximação pode ser feita de vários modos.

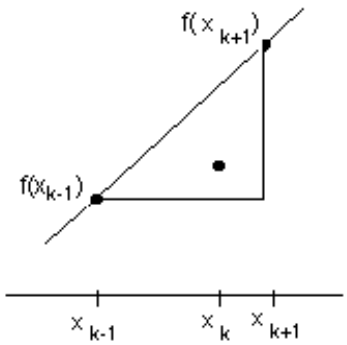
Assim, dependendo dos pontos, temos três técnicas:



(a)



(b)



(c)

A derivada segunda pode ser achada através da fórmula:

$$f''(x_k) = \frac{f'(x_k) - f'(x_k - 1)}{(x_k) - (x_k - 1)}$$

Comando diff

O comando *diff* calcula a diferença entre dois pontos adjacentes num vetor, gerando um novo vetor com a diferença (Se o comando *diff* for aplicado a uma matriz, ele irá operar como se cada coluna da matriz fosse um vetor).

Por exemplo, assumindo que o vetor x seja [0,1,2,3,4,5], e que o vetor y seja [2,3,1,5,8,10]. O vetor gerado por diff(x) será [1,1,1,1,1], enquanto que o gerado por diff(y) será [1,-2,4,3,2].

A derivada dy será calculada por diff(y) ./ diff(x). Note que estes valores de dy estarão corretos para ambas as formas de diferenças, backward ou forward. A diferença entre esses dois métodos para o cálculo da derivada é determinada pelos valores de x que correspondem à derivada dy. Se os valores correspondentes de x forem [1,2,3,4,5] então dy é calculado pela diferença backward; mas se os valores de x forem [0,1,2,3,4] então dy será calculado pelo método da diferença forward.

Supondo que desejamos analisar a função dada pelo seguinte polinômio:
$f(x) = x^5 - 3x^4 - 11x^3 + 27x^2 + 10x - 24$

Assumindo que queiramos calcular o valor de sua derivada no intervalo [-4,5], usando o método da diferença backward.

Chamando f '(x) de df e, xd os valores de x da derivada.

Temos no MATLAB que:

```
x = -4:0.1:5;
f = x .^5 - 3 * x .^4 - 11 * x.^3 + 27 * x .^2 + 10 * x - 24;
df = diff(y) . / diff(x);
xd = x(2:length(x) );
plot(f,x)
plot(df,xd)
axis([-4 5 -800 600]);
plot(f)
axis([-4 5 -200 1400]);
plot(df)
```

Podemos marcar os locais dos pontos críticos para essa função com os seguintes comandos:

```
produto = df(1 : length(df) - 1 ) .* df(2 : length(df) );
critico = xd (find (produto < 0) )
```

O comando *find* determina os índices dos locais do produto para os quais a derivada df(k) é igual a zero; esses índices são então usados com o vetor contendo os valores de xd para determinar os locais de pontos críticos.

Exercícios para Praticar !

1. Para cada polinômio abaixo, plote a função, sua derivada primeira e sua derivada segunda, no intervalo de [-10,10]. Depois ache os locais de mínimo, de máximo, e os pontos críticos

   a. g(x) = x3 - 5x2 + 2x + 8
   b. h(x) = x5 - 4x4 - 9x3 + 32x2 + 28x - 48
   c. i(x) = x7 - 5x3 + 14x2 - 12

Capítulo 12 - Equações Diferenciais Ordinárias

Nesta sessão iremos apresentar um grupo de equações de primeira ordem e suas soluções analíticas. Depois seguiremos com a descrição dos métodos de Runge - Kutta para a integração de equações de primeira ordem, onde então iremos comparar as soluções numéricas com as analíticas. Esse capítulo termina com a discussão quando se torna necessário converter equações diferenciais de ordem superiores para equações de primeira ordem.

12.1 Equações Diferenciais Ordinárias de Primeira Ordem

A equação diferencial de primeira ordem (ODE) é uma equação que pode ser escrita na seguinte forma:

$$y' = \frac{dy}{dx} = g(x,y)$$

onde x é a variável independente.

A solução da equação diferencial de primeira ordem (ODE) é a função y = f(x), tal que f ’(x) = g(x,y). O cálculo da solução envolve a integração de y ’ para obter y. A solução de uma equação diferencial é geralmente uma família de funções. A condição inicial é usualmente necessária na ordem para especificar uma única solução. A seguir serão representadas algumas soluções analíticas para equações diferenciais ordinárias.

Enquanto que as soluções analíticas para as equações diferenciais são preferenciais, muitas vezes requerem soluções muito complicadas. Para esses casos, uma técnica numérica se torna necessária. As técnicas numéricas mais comuns para resolver equações diferenciais ordinárias, são o método de Euler e o método de Runge-Kutta.

Tanto o método de Euler quanto o método de Runge-Kutta aproximam a função utilizando-se da expansão em série de Taylor.

Lembrando que a série de Taylor é uma expansão que pode ser usada para aproximar uma função cujas derivadas são definidas no intervalo contendo a e b. A expansão por série de Taylor para f(b) é:

$$f(b) = f(a) + (b - a)\, f'(a) + \frac{(b - a)^2}{2!} f''(a) + \ldots + (b - a)^n f(n)(a) + \ldots$$

Para as equações diferenciais de primeira ordem a serie de Taylor se torna:

$$f(b) \approx f(a) + (b - a)\, f'(a)$$

Para as equações diferenciais de segunda ordem:

$$f(b) \approx f(a) + (b - a)\, f'(a) + \frac{(b - a)^2}{2!} f''(a)$$

E, assim por diante.

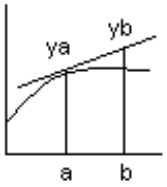
12.2 Método de Runge - Kutta

Os métodos mais populares para a integração da equação diferencial de primeira ordem são os métodos de Runge - Kutta. Esses métodos de aproximação de uma função se usam da expansão por série de Taylor. Desta forma, o método de Runge - Kutta de primeira ordem se utiliza da expansão de Taylor de primeira ordem, o método de Runge - Kutta de segunda ordem se utiliza da expansão de Taylor de segunda ordem, e, assim por diante. Lembrando que o método de Euler é equivalente ao método de Runge - Kutta de primeira ordem.

Método de Euler

$$yb = ya + (b - a)\, ya'$$

Esta equação estima o valor da função yb usando uma reta tangente a função no ponto a, conforme mostrado na figura abaixo:

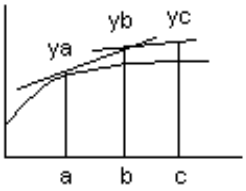


A equação diferencial é usada para calcular o valor de ya'.

Tendo estimado o valor da função yb no ponto b, podemos estimar o próximo valor da função yc, usando:

$$yc = yb + (b - a)\, yb'$$

Essa equação utilizará a tangente no ponto b para estimar o valor da função no ponto c, yc, como é mostrado na figura a seguir:



É preciso partir de uma condição inicial para dar início ao processo de estimativa de outros pontos da função f(x).

Comando ode

O MATLAB contém dois comandos para calcular soluções numéricas para equações diferenciais ordinárias: *ode23* e *ode45*; o comando *ode23* usa o método de Runge - Kutta para equações diferenciais de segunda e terceira ordem; o comando *ode45* usa o método de Runge - Kutta para equações diferenciais de quarta e quinta ordem. Os comandos *ode23* e *ode45* possuem os mesmos tipos de argumentos.

A forma mais simples do comando *ode23* requer quatro argumentos. O primeiro argumento é o nome da função, definida no MATLAB, que retorna o valor da equação diferencial y' = g(x,y) quando é fornecido valor para x e y. O segundo e o terceiro argumentos representam os limites no intervalo no qual nó desejamos calcular o valor da função y = f(x). O quarto argumento contém a condição inicial necessária para determinar a única solução para a equação diferencial ordinária. Nós assumimos que esse argumento representa o valor da função dentro do intervalo considerado. O comando *ode23* possui duas saídas: um conjunto de coordenadas x e, um conjunto de coordenadas y correspondentes, os quais representam os pontos da função y = f(x).

No MATLAB, primeiro temos que definir a função a qual desejamos avaliar as equações diferenciais, assumindo valores escalares de entrada para x e y.

Exemplo 1

Resolver a equação y' = g1(x,y) = $3x^2$ no intervalo [2,4], assumindo como condição inicial f(2) = 0,5.

Solução analítica: y = x^3 - 7.5

Solução no MATLAB:

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

```
[x,num_y] = ode23('g1',2,4,0.5)
anl_y^= x.^3 - 7.5;
subplot(211),plot(x,num_y,x,anl_y,'o');
title('Solução do Exemplo 1');
xlabel('X');
ylabel('y = f(x)');
grid;
```

O gráfico obtido conterá a comparação entre a solução numérica e a solução analítica.

Exemplo 2

Resolver a equação y' = g2(x,y) = $2x\cos^2 y$ no intervalo [0,2], assumindo como condição inicial f(0) = $\pi/4$.

Solução analítica: $y = \tan^{-1}(x^2 + 1)$

Solução no MATLAB:

```
function dy=g2(x,y)
dy=2*x*cos(y)^2;
[x,num_y] = ode23('g2',0,2,pi/4)
anl_y = atan(x*x+1);
subplot(211),plot(x,num_y,x,anl_y,'o');
title('Solução do Exemplo 2');
xlabel('X');
ylabel('y = f(x)');
grid;
```

O número de pontos calculados para a função y = f(x) pelo comando *ode23* ou *ode45* é determinado pelo MATLAB.

Os comandos *ode23* e *ode45* podem também ser usados com dois parâmetros adicionais. O quinto parâmetro pode ser usado para especificar a tolerância que estará relacionada com o tamanho do passo. O valor default para a tolerância é de 0.001 para o ode23 e 0.000001 para o ode45. O sexto parâmetro pode ser usado para requerer que a função escreva na tela imediatamente os resultados chamado traço. O valor default é zero, especificando nenhum traço para os resultados.

Exercícios para praticar!

1. Seja a equação:

y' = ga(x,y) = -y

a) Assumindo como condição inicial f(0) = -3.0, resolva, no MATLAB, essa equação diferencial no intervalo de [0,2] e plote o gráfico com os valores correspondentes de y.

b) Sendo y = -3 e-x , a solução analítica para esta equação, faça um novo gráfico que compare a solução analítica com a numérica.

Problema Aplicado: Aceleração de uma turbina UDF numa aeronave

Uma avançada turbina chamada de ventilador não canalizado (UDF) é uma das novas tecnologias mais promissoras que tem sido desenvolvida para o futuro transporte de aeronaves. Turbinas, que têm sido usadas por décadas, combinam o poder e a confiabilidade dos motores a jato com a eficiência dos propulsores. Eles constituem uma importante melhoria dos antigos propulsores movidos a pistão. Suas aplicações têm sido limitadas a pequenas aeronaves do tipo comutador, isto porque eles não são tão rápidos, nem poderosos quanto as turbinas usadas em grandes aeronaves. Esse tipo de turbina(UDF) implica em avanços significantes na tecnologia de propulsão. Novos materiais, aerodinâmica e velocidades de alta rotação habilitam esta turbina a voar tão rápido quanto as turbinas a jato, e com grande aproveitamento de combustível. A UDF é também menos barulhenta que o sistema convencional de turbinas.

Durante um teste de vôo de uma turbina UDF de uma aeronave, o motor é levado para um nível de 40 Newton, o que significa os 20 Kg da aeronave tendo alcançado uma velocidade de 180 m/s. As válvulas de regulação do motor são então levadas para atingir um nível de 60 Newtons, e a aeronave começa a acelerar.

A equação diferencial que determina a aceleração da aeronave é:

$$a = \frac{T}{m} - 0{,}000062\, v^2$$

onde:

$$a = \frac{dv}{dt}$$

T = nível atingido em Newtons
m = massa em kg
v = velocidade em m/s

Escreva no MATLAB um programa para determinar a nova velocidade depois de uma mudança no nível do motor através do gráfico da solução para a equação diferencial.

Método para a resolução do problema

1. O PROBLEMA EM SI
   Calcular a nova velocidade atingida pela aeronave depois de uma mudança no nível do motor.

2. DESCRIÇÃO DA ENTRADA E DA SAÍDA
   Como entrada nós temos a equação diferencial que define a aceleração da aeronave.

   Como saída, nós desejamos o gráfico da velocidade e da aceleração.

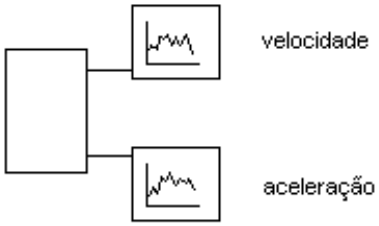


Diagrama de entrada e saída

3. SOLUÇÃO NO MATLAB

   Podemos usar o comando *ode23* para avaliar a nossa equação diferencial. A solução dessa equação diferencial nos fornecerá valores de velocidade, os quais poderão ser usados para determinar os valores da aceleração. Nós podemos então traçar ambos os gráficos de velocidade e aceleração num intervalo de 4 minutos para observar suas mudanças. A velocidade deverá aumentar e então estabilizar num novo valor, enquanto que a aceleração deverá diminuir até chegar a zero.

4. PROGRAMA

12.3 Equações Diferenciais Ordinárias de Ordens Superiores

Equações diferenciais de ordens superiores podem ser escritas como um sistema constituído por um conjunto de equações diferenciais de primeira ordem usando a mudança de variáveis.

Exemplo 3

Vamos considerar uma equação diferencial linear de segunda ordem:

$y'' = g(x,y,y') = y'(1 - y^2) - y$

Primeiro vamos definir duas novas funções:

$u_1(x) = y'$
$u_2(x) = y$

Nós então obtemos esse sistema de um conjunto de equações diferenciais de primeira ordem:

$u_1' = y'' = g(x, u_2, u_1) = u_1(1 - u_2^2) - u_2$
$u_2' = u_1$

O sistema contendo as equações diferenciais de primeira ordem pode ser resolvido pelo MATLAB através do comando ode. Entretanto, a função que é usada para avaliar a equação diferencial deve calcular os valores das equações diferenciais de primeira ordem em um vetor. A condição inicial deverá também ser um vetor contendo uma condição inicial para cada equação diferencial de primeira ordem: $y^{n-1}, y^{n-2}, ..., y', y$.

Para resolver as equações desenvolvidas no exemplo anterior, primeiro temos que definir a função para calcular os valores das equações diferenciais de primeira ordem:

```
function u_primo = eqns2(x,u)
u_primo(1) = u(1)*(1 - u(2)^2 - u(2);
u_primo(2) = u(1);
```

Então, para resolver o sistema de equações diferenciais de primeira ordem no intervalo [0,20] usando as condições iniciais y'(0) = 0.0 e y(0) = 0.25, podemos seguir os seguintes passos:

```
inicial = [0 0.25];
[x,num_x] = ode23('eqns2',0,20,inicial);
subplot(211), plot(x,num_y(:,1))
title('Primeira Derivada de y');
xlabel('x');grid;
subplot(212), plot(x,num_y(:,2))
title('y');
xlabel('x');grid;
```

Capítulo 13 - Decomposição e Fatorização de Matrizes

Este capítulo contém algumas das mais avançadas características de matrizes que são utilizadas na resolução de certos tipos de problemas de engenharia. O primeiro tópico, autovalores e autovetores, aparece em inúmeras aplicações. Depois de definir autovalores e autovetores e ilustrar suas propriedades com um exemplo simples, a função *eig* é apresentada para computação usando ambas. Uma aplicação que é utilizada para demostrar como autovalores e autovetores são utilizados para analisar a performance de algoritmos de adaptadores para redução de ruídos. O resto do capítulo continua com decomposição e fatorização que podem ser aplicados para a matriz A.

13.1 Autovalores e Autovetores

Assuma que A é uma matriz quadrada n x n. Seja X um vetor de uma coluna e "n" linhas e seja λ um escalar. Considere a seguinte equação:

$$AX = \lambda X \qquad (13.1)$$

Ambos os lados dessa equação são iguais com uma coluna de vetores com n linhas. Se X é completada com zeros, então esta equação é verdadeira para algum valor de λ, mas esta é uma solução trivial.

Os valores de λ para que X não seja completado com zeros são descritos pelos autovalores da matriz A, e os valores correspondentes de X são descritos pelos autovetores da matriz A.

A equação (13.1) pode ser utilizada para determinar a seguinte equação:

$$(A - \lambda I) X = 0 \qquad (13.2)$$

onde I é uma matriz identidade de n x n elementos. Esta equação representa um conjunto de equações homogêneas enquanto o lado direito da equação for igual a zero. Este conjunto de equações homogêneas possui soluções que não são triviais. A solução só é trivial quando o determinante for igual a zero.

$$\det (A - \lambda I) = 0 \qquad (13.3)$$

A equação (13.3) representa uma equação que é referida a equação característica da matriz A. A solução desta equação é obtida com os autovalores da matriz A.

Em muitas aplicações, é desejável selecionar os autovetores como tal $QQ^T = I$, onde Q é uma matriz cujas colunas são os autovetores. Este conjunto de autovetores representa um conjunto ortogonal, enquanto significa que ambos são normalizados e que eles são mutuamente ortogonais. (Um conjunto de vetores é ortonormal se o produto de vetores for igual a unidade, e o produto de um vetor com outro for zero.)

Para ilustrar estas relações entre a matriz A e estes autovalores e autovetores consideremos a matriz A :

$$A = \begin{bmatrix} 0.50 & 0.25 \\ 0.25 & 0.50 \end{bmatrix}$$

Os autovalores podem ser obtidos usando a equação característica:

$$\det(A - l\,I) = \det \begin{bmatrix} 0.5 - l & 0.25 \\ 0.25 & 0.5 - l \end{bmatrix}$$
$$= \lambda^2 - \lambda + 0.1875$$
$$= 0$$

Esta equação pode ser facilmente resolvida usando equação quadrática e obtemos $\lambda_o = 0.25$ e $\lambda_1 = 0.75$. (Se a matriz A tiver mais de 2 linhas e 2 colunas, determinar os auto- valores na mão pode se tornar uma formidável tarefa.) Os autovalores podem ser determinados utilizando os autovalores da equação (13.2), usando o valor 0,25:

ou

$$\begin{bmatrix} 0.5 - 0.25 & 0.25 \\ 0.25 & 0.5 - 0.25 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Mas este par de equações nos dá a seguinte equação:

$$x_1 = - x_2$$

Portanto, existem uma infinidade de autovetores que podem ser associados com o autovalor 0.25. Alguns desses autovetores são demonstrados agora:

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad \begin{bmatrix} 5 \\ -5 \end{bmatrix} \quad \begin{bmatrix} 0.2 \\ -0.2 \end{bmatrix}$$

Similarmente, pode se obter os autovetores de autovalor 0.75, que possui a seguinte relação:

$$x_1 = x_2$$

De novo obtemos uma infinidade de autovetores, como:

$$\begin{bmatrix}1.5\\1.5\end{bmatrix} \quad \begin{bmatrix}5\\-5\end{bmatrix} \quad \begin{bmatrix}0.2\\-0.2\end{bmatrix}$$

Para determinar um conjunto ortonormal de autovetores para um exemplo simples precisamos lembrar de como selecionar os autovetores, como $QQ^T = I$. Portanto consideremos o seguinte:

$$QQ^T = \begin{bmatrix} c_1 & c_2 \\ -c_1 & c_2 \end{bmatrix} \begin{bmatrix} c_1 & -c_1 \\ c_2 & -c_2 \end{bmatrix}$$

$$= \begin{bmatrix} c_1^2 + c_2^2 & -c_1^2 + c_2^2 \\ -c_1^2 + c_2^2 & c_1^2 + c_2^2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Resolvendo o conjunto de equações obtemos:

$$c_1^2 = c_2^2 = 0.5$$

Então $c_1 = c_2 = 0.707$ ou -0.707. Assim eles possuem várias variações para os mesmos valores, que podem ser utilizados para determinar o conjunto ortonormal de autovetores. Nós escolhemos o seguinte:

$$Q = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

Os cálculos para se obter os autovalores de um conjunto associado de autovetores ortonormais podem ser relativamente simples para uma matriz 2x2. Entretanto é evidente que fica muito difícil se aumentarmos o tamanho da matriz.

A função *eig* possui um argumento da matriz A. Esta função pode ser usada para retornar um vetor coluna que contenha apenas autovalores, como:

```
lambda = eig (A)
```

A função pode também ser usada para executar uma tarefa dupla. Neste caso para retornarmos duas matrizes quadradas: uma contém autovetores (X) como coluna e a outra contém autovalores (λ) na diagonal:

```
[Q,d] = eig (A)
```

Os valores de Q e d são como $QQ^T = I$ e $AQ = Qd$.

Nós podemos ilustrar a função eig com um exemplo:

```
A = [0.50, 0.25; 0.25, 0.50]
[Q,d] = eig (A)
```

Os valores de Q e d são obtidos:

$$Q = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

$$d = \begin{bmatrix} 0.25 & 0.00 \\ 0.00 & 0.75 \end{bmatrix}$$

Podemos facilmente verificar que $QQ^T = I$ e $AQ = Qd$.

P r a t i q u e !

Consideremos a matriz A:

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

Use o MATLAB para responder as questões:

1. Determine λ_1, λ_2, λ_3, os três autovalores de A
2. Determine um conjunto ortonormal de autovetores, X_1, X_2, X_3
3. Verifique se $\det(AX - \lambda I) = 0$ para os autovalores obtidos
4. Demonstre que $AQ = Qd$

Aplicação em Solução de Problemas: Adaptador p/ Redução de Ruído

Este equipamento é utilizado para reduzir o efeito de interferência de ruídos em um sinal. Por exemplo, um microfone que é utilizado para gravar sinais de voz de um grande auditório. Outro microfone é usado na parte de trás do auditório para colher principalmente os sinais de ruído. Através das técnicas para cancelamento de ruídos, as características do sinal de ruído podem ser determinadas usando o sinal de dois microfones. Os adaptadores são utilizados para reduzir o ruído oriundo da parte de trás do auditório, para poderem ser transmitidos para a sala de controle. Este processo resulta num sinal limpo e uma melhor comunicação .

Os algoritmos para os adaptadores estão acima do nível deste texto, mas a performance e a velocidade do algoritmo dependem das características dos sinais de entrada. Estas características determinam a superfície multidimensional quadrática para que obtenhamos um valor mínimo. Este mínimo é determinado ajustando o algoritmo para um ponto de partida para um único mínimo. Se a superfície quadrática tem um contorno circular o algoritmo não é necessário. A matriz R pode ser computada a partir dos sinais de entrada, e os autovalores da matriz R irão determinar o tipo de superfície de contorno a ser utilizada. Se os autovalores forem iguais, a superfície é circular. Quanto maior a variação dos autovalores, mais elíptica será a superfície. Os autovetores representam o eixo principal da superfície. Portanto para determinar a velocidade e a performance do algoritmo do adaptador com certeza do tipo de dados e para analisar a superfície, nós precisamos determinar os autovalores e autovetores da matriz R.

Escreva um programa para ler os valores da matriz com o nome dataR.mat, e depois calcule os autovalores e autovetores.

1. DESCRIÇÃO DO PROBLEMA
   Calcule os autovalores e autovetores de uma matriz.

2. DESCRIÇÃO DOS DADOS DE ENTRADA E SAÍDA
   A entrada é o arquivo dataR.mat e a saída são os autovalores e autovetores da matriz.

3. EXEMPLO MANUAL
   Assumamos que a matriz de entrada seja:

$$R = \begin{bmatrix} 0.50 & 0.25 \\ 0.25 & 0.50 \end{bmatrix}$$

   Como no exemplo que tratamos anteriormente sabemos que os autovalores são 0.25 e 0.75. Os autovetores são:

$$V_1 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$$

$$V_2 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

Como os autovalores não são iguais, sabemos que a superfície quadrática não possui um contorno circular e que a performance do adaptador será lenta e menos precisa.

4. SOLUÇÃO MATLAB

Neste programa usaremos um loop para imprimir os autovalores e autovetores.

```
load dataR;
[Q,d] = eig (R)
[m,n] = size(R)
for k = 1: m
        fprintf ( ‘ Autovalor %4.0f = %7.2f  \n’, k, d ( k, k ) );
        disp (‘Autovetor correspondente’)
        disp (Q(:,k) ‘)
end
```

5. TESTANDO

A saída do programa será:

```
Autovalor  1  =        0.25
Autovetor correspondente
        0.7071     -0.7071

Autovalor  2  =        0.75
Autovetor correspondente
        0.7071      0.7071
```

Este exemplo é um exemplo simples ao tratar de cancelamento de sinais, pois quando se filtra os sinais usados em comunicação com satélites, a matriz R possui milhares de linhas e colunas.

13.2 DECOMPOSIÇÃO e FATORIZAÇÃO

Nesta seção iremos apresentar três tipos de decomposição e fatorização de matrizes que podem ser utilizados para a solução de problemas que contenham matrizes. Algumas dessas técnicas decompõe a matriz em um produto de outras matrizes. O uso do produto fatorial reduz o número de cálculos necessários para a computação de muitas matrizes. Muitas técnicas numéricas que utilizam matrizes, as convertem em forma de decomposição e fatorização.

Fatorização Triangular

A fatorização triangular expressa uma matriz quadrada como um produto de duas matrizes triangulares – uma matriz inferior superior e uma matriz triangular superior. Esta fatorização é conhecida como fatorização LU (lower-upper).

Esta fatorização é muito usada para simplificar matrizes computacionais. Este é um dos passos para se determinar o determinante de uma matriz muito grande, como também a matriz inversa e para a solução de equações lineares simultâneas.

A fatorização pode ser realizada começando com uma matriz quadrada e uma matriz identidade de mesmo tamanho. Operações de linhas e colunas são realizadas na matriz A para reduzi-la a forma triangular superior; as mesmas operações são realizadas na matriz identidade para transformá-la na forma triangular inferior.

Para ilustrar pegamos as matrizes A e B:

$$A = \begin{bmatrix} 1 & 2 & -1 \\ -2 & -5 & 3 \\ 1 & -3 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 3 & 2 \\ -2 & -6 & 1 \\ 2 & 5 & 7 \end{bmatrix}$$

Usando a fatorização LU obtemos:

$$A = \begin{bmatrix} -0.5 & 1 & 0 \\ 1 & 0 & 0 \\ 0.5 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 & -5 & 3 \\ 0 & -0.5 & 0.5 \\ 0 & 0 & -2 \end{bmatrix} \qquad B = \begin{bmatrix} -0.5 & 0 & 1 \\ 1 & 0 & 0 \\ -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} -2 & -6 & 1 \\ 0 & -1 & 8 \\ 0 & 0 & 2.5 \end{bmatrix}$$

A função *lu* do MATLAB executa esta fatorização, e é especificada da seguinte forma:

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

O fator inferior é colocado na matriz L e o superior na matriz U. O produto de L e U é igual a A. Veja o exemplo:

```
A = [ 1, 2, -1; -2, -5, 3; -1, 3, 0 ];
[LA,UA] = lu ( A );
B = [ 1, 3, 2; -2, -6, 1; 2, 5, 7 ];
[LB,UB] = lu ( B );
```

Como resultado obtemos:

$$LA = \begin{bmatrix} -0.5 & 1 & 0 \\ 1 & 0 & 0 \\ 0.5 & 1 & 1 \end{bmatrix} \qquad UA = \begin{bmatrix} -2 & -5 & 3 \\ 0 & -0.5 & 0.5 \\ 0 & 0 & -2 \end{bmatrix}$$

$$LB = \begin{bmatrix} -0.5 & 0 & 1 \\ 1 & 0 & 0 \\ -1 & 1 & 0 \end{bmatrix} \qquad UB = \begin{bmatrix} -2 & -6 & 1 \\ 0 & -1 & 8 \\ 0 & 0 & 2.5 \end{bmatrix}$$

É facilmente verificável que: A = (LA)(UA) e B = (LB)(UB).

Fatorização QR

Esta técnica de fatorização é feita a partir do produto de uma matriz ortonormal e de uma matriz triangular superior. (Lembrando que uma matriz é ortonormal quando $QQ^T=I$). Não é necessário que a matriz A seja quadrada.

A menor solução para um sistema indeterminado AX=B é a solução de um sistema quadrado $RX=Q^TB$.

A função *qr* do MATLAB executa esta fatorização, e é especificada da seguinte forma:

[Q,R] = qr (A)

Para uma matriz A m x n, tamanho de Q é n x n, e o tamanho de R é m x n.

Decomposição de valor singular

O SVD (singular value decomposition) é um outro método para a fatorização de matrizes ortogonais. Esta é a decomposição mais confiável, mas isto pode requerer dez vezes mais tempo que a fatorização QR. A decomposição SVD decompões a matriz num produto dos fatores de outras três matrizes: A= USV ,onde U e V são matrizes ortogonais e S é diagonal. Os valores da matriz diagonal são chamados de valores singulares e por isso a decomposição recebe este nome. O número de valores singulares diferentes de zero é igual ao rank da matriz.

A função *svd* do MATLAB executa esta fatorização, e é especificada da seguinte forma:

[U,S,V] = svd (A)

Sumário MATLAB

eig	calcula os autovalores e autovetores de uma matriz
lu	calcula a decomposição LU de uma matriz
or	calcula a decomposição ortonormal de uma matriz
svd	calcula a decomposição SVD de uma matriz

Capítulo 14 – Processamento de Sinais

Este capítulo discute algumas funções que são relacionadas a processamento de sinais. Essas funções foram divididas em quatro categorias: análise do domínio da freqüência e domínio do tempo, análise de filtros, implementação de filtros e projeto de filtros.

Embora este capítulo discuta tanto a análise do processamento de sinais analógicos e digitais será dada maior ênfase no processamento de sinais digitais ou DSP.

14.1 Análise no Domínio da Freqüência

Recordemos que o sinal analógico é uma função contínua (f(t)) que representa uma informação, como por exemplo, um sinal de voz, o sinal da pressão sangüínea e sinais sísmicos. Para o processamento de sinais por um computador, o sinal analógico precisa ser amostrado num período de T segundos, gerando assim, um sinal digital com uma seqüência de valores derivados do sinal analógico original. Representamos o sinal digital como um sinal contínuo com a seguinte notação:

$$f_k = f(kT)$$

O sinal digital é uma seqüência de amostragens representadas por: $[f_k]$

O tempo que normalmente se escolhe para começar a amostragem e o zero e assim o primeiro intervalo de amostragem é f_0. Então se o sinal é amostrado com uma freqüência de 100 Hz, os primeiros três valores correspondentes ao sinal analógico são:

$f_0 = f(0T) = f(0.0)$
$f_1 = f(1T) = f(0.01)$
$f_2 = f(2T) = f(0.02)$

Estamos acostumados a ver os sinais digitais derivados de analógicos como uma seqüência de pontos, mas quando nós plotamos um sinal digital os pontos são conectados por segmentos de linha. Utilizamos o eixo Y para representar $[f_k]$ ou $f(kT)$, que é o sinal digital.

Os índices do MATLAB começam sempre com 1, como x(1), x(2) e assim por diante. Entretanto os índices para processamento de sinais utilizados são sempre valores negativos, como h_{-2}, h_{-1}, h_0 e assim por diante. É importante que as equações sejam escritas na mesma forma para que o usuário não se confunda.

Os sinais podem ser analisados de suas formas – domínio do tempo e domínio da freqüência. O domínio do tempo é representado por valores reais e o domínio da freqüência por valores complexos, que por sua vez, podem ser representados por senóides, que compõem o sinal.

A transformada discreta de Fourier (DFT) é usada para converter um sinal digital no domínio do tempo em um conjunto de pontos no domínio da freqüência. A entrada da transformada é um conjunto de N valores de tempo [f_k] : o algoritmo é calculado como um conjunto de valores complexos [F_K] que representam a informação no domínio da freqüência.

O algoritmo da transformada utiliza um número de cálculos muito grande, por isso utilizamos a Transformada de Fourier, que também converte o sinal no tempo para o domínio da freqüência.

A função *fft* do MATLAB calcula a transformada de Fourier. Esta função pode ser usada para uma ou duas entradas. Se entrarmos com um sinal simples no domínio do tempo obteremos como resposta um sinal contendo números complexos, que representam o domínio da freqüência.

Se o número de valores no domínio do tempo for igual a potência de 2, usaremos o método da transformada de Fourier. Se não, usaremos o método DFT.

Os valores no domínio da freqüência gerados pela função fft correspondem a uma freqüência de separação de 1/NT Hz, onde N é o número de amostras e T o período da função. Seja N=32, T=0.001, os valores de freqüência mostrados serão 0 Hz, 1/0.032Hz e 2/0.032 Hz e assim por diante.

Consideremos o seguinte exemplo:

```
N = 64;
T = 1/128;
k = 0 : N-1;
f = sin(2*pi*20*k*T);
F = fft(f);
magF = abs (F);
plot (k, magF), title (‘Magnitude de F(k)’),...
xlabel (‘k’), ylabel (‘| F ( k )|’),grid
```

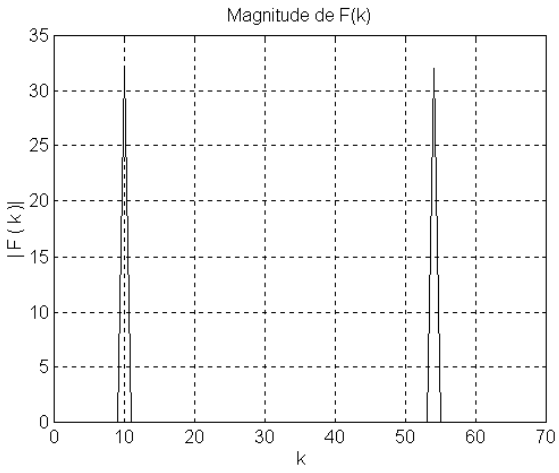


Figura 1 – Magnitude de Fk

Neste gráfico podemos notar a simetria gerada pela periodicidade da FFT; e da onda senoidal.

Podemos plotar também a magnitude de F_K pela freqüência em hertz(Hz).

```
hertz = k*(1/(N*T));
plot(hertz(1:N/2),magF(1:N/2)),...
title('Magnitude de F(k)'),...
xlabel ('Hz'), ylabel ('| F ( k )|'),grid
```

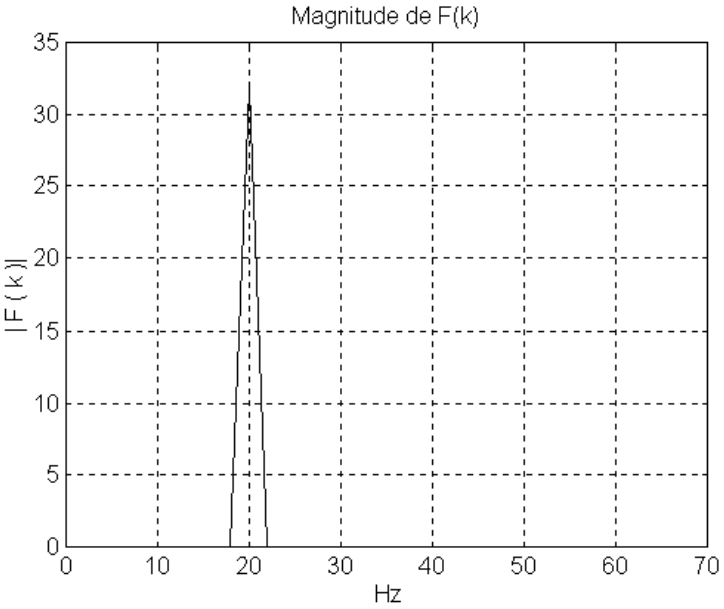


Figura 2 – Magnitude de Fk em hertz

A função *ifft* faz a transformada inversa de Fourier, ou seja, calcula o domínio do tempo [f_k] a partir de valores complexos [F_K].

O método FFT é uma ferramenta poderosa quando se trabalha com sinais digitais. Nossa discussão foi focada na magnitude de F_K , mas também é muito importante obtermos a fase de F_K.

P r a t i q u e !

Gere 128 pontos para os seguintes sinais. Plote o sinal no domínio do tempo. Usando o método da transformada de Fourier gere e plote o sinal no domínio da freqüência. Use a escala de Hz no eixo X. Assuma a taxa de amostragem de 1KHz. Verifique se os picos ocorrem onde era esperado para o domínio da freqüência.

1. $f_k = 2 \sin(2\pi 50kT)$
2. $g_k = \cos(250\pi kT) - \sin(200\pi kT)$
3. $h_k = 5 - \cos(1000kT)$
4. $m_k = 4 \sin(250\pi kT - \pi/4)$

14.2 Análise de Filtros

A função de transferência de um sistema analógico é descrita como H(s) e de um sistema digital como H(z). Estas funções de transferências descrevem o efeito do sistema a um sinal de entrada, e também o efeito de filtragem do sistema.

Como a função de transferência de um filtro define o efeito do filtro em termos de freqüência, podemos usar esta função de transferência para descrever uma faixa de freqüência. Por exemplo, um filtro passa baixa irá deixar passar todas as freqüências abaixo da freqüência de corte estabelecida. O passa banda irá deixar passar a banda de freqüência especificada. E o corta banda irá remover a banda de freqüência especificada.

Podemos definir a atuação dos filtros em 3 regiões básicas: banda de passagem, banda de transição e banda de corte. Estas regiões são definidas pela freqüência de corte w_c e pela freqüência de rejeição w_r.

Como uma função de transferência é uma função complexa, a análise dos filtros incluem gráficos de magnitude e fase. Para isso utilizam-se as funções *abs*, *angle* e *unwrap.* Adicionalmente, as funções *freqs* e *freqz* podem ser usadas para se achar os valores das funções H(s) e H(z) como nos exemplos a seguir.

Função de Transferência Analógica

Um filtro analógico é definido pela função de transferência H(s) onde s = jw. Na forma geral a função de transferência H(s) é a seguinte:

$$H(s) = \frac{B(s)}{A(s)} \qquad (14.1)$$

$$= \frac{b_0 s^n + b_1 s^{n-1} + b_2 s^{n-2} + \ldots + b_n}{a_0 s^n + a_1 s^{n-1} + a_2 s^{n-2} + \ldots + a_n}$$

Esta função de transferência corresponde a ordem n dos filtros analógicos.

$$H(s) = \frac{0.5279}{s^2 + 1.0275s + 0.5279}$$

$$H(s) = \frac{s^2}{s^2 + 0.1117s + 0.0062}$$

$$H(s) = \frac{1.05s}{s^2 + 1.05s + 0.447}$$

$$H(s) = \frac{s^2 + 2.2359}{s^2 + 2.3511s + 2.2359}$$

A função *freqs* calcula os valores de uma função complexa H(s), usando 3 argumentos de entrada. O primeiro é um vetor contendo os coeficientes do polinômio B(s) da Equação 14.1; o segundo é um vetor contendo os coeficientes do polinômio A(s); e o terceiro é um vetor com o valor da freqüência em 'rps'. Em geral, colocamos o alcance da freqüência para começar no zero e incluímos todos os parâmetros do filtro.
Programa:

```
W1 = 0:0.05:5.0;
B1 = [0.5279];
A1 = [1,1.0275,0.5279];
H1s = freqs(B1,A1,W1);

W2 = 0:0.001:0.3;
B2 = [1,0,0];
A2 = [1,0.1117,0.0062];
H2s = freqs(B2,A2,W2);

W3 = 0:0.01:10;
B3 = [1.05,0];
A3 = [1,1.05,0.447];
H3s = freqs(B3,A3,W3);

W4 = 0:0.005:5;
B4 = [1,0,2.2359];
A4 = [1,2.3511,2.2359];
H4s = freqs(B4,A4,W4);
clg
subplot (221),plot(W1,abs(h1s)),title('Filtro H1(s)'),...
xlabel ('w,rps'), y;label('Magnitude'), grid
subplot (222),plot(W2,abs(h2s)),title('Filtro H2(s)'),...
xlabel ('w,rps'), y;label('Magnitude'), grid
subplot (223),plot(W3,abs(h3s)),title('Filtro H3(s)'),...
xlabel ('w,rps'), y;label('Magnitude'), grid
subplot (224),plot(W4,abs(h4s)),title('Filtro H4(s)'),...
xlabel ('w,rps'), y;label('Magnitude'), grid
```

A fase de um filtro pode ser obtida utilizando-se a função able ou a unwrap. Como a fase de um número complexo é um ângulo em radianos, o ângulo está restrito ao intervalo 2π. Esta função, utiliza o intervalo de $-\pi$ a π e quando estes limites são ultrapassados é gerada uma descontinuidade.

A função unwrap remove as descontinuidades quando introduzimos a função angle. Como no exemplo: unwrap(angle(H)).

Função de Transferência Digital

A função de transferência digital é definida por H(z) onde z = e ^{jwt}. Na forma geral a função de transferência H(z) é a seguinte:

$$H(z) = \frac{B(z)}{A(z)} \qquad (14.2)$$

$$H(s) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + ... + b_n z^{-n}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + ... + a_n z^{-n}}$$

Esta função de transferência corresponde a ordem n dos filtros digitais.

$$H_1(z) = \frac{0.2066 + 0.4131z^{-1} + 0.2066z^{-2}}{1 - 0.3695z^{-1} + 0.1958z^{-2}}$$

$$H_2(z) = \frac{0.894 - 1.789z^{-1} + 0.894z^{-2}}{1 - 1.778z^{-1} + 0.799z^{-2}}$$

$$H_3(z) = \frac{0.42 - 0.42z^{-2}}{1 - 0.443z^{-1} + 0.159z^{-2}}$$

$$H_4(z) = \frac{0.5792 + 0.4425z^{-1} + 0.5792z^{-2}}{1 + 0.4425z^{-1} + 0.1584z^{-2}}$$

Se o denominador da função de transferência for igual a 1, o filtro é um FIR(resposta finita ao impulso) e se for diferente de 1 o filtro é um IIR(resposta infinita ao impulso). Ambos são muito utilizados no processamento de sinais digitais.

A função *freqz* calcula os valores de uma função complexa H(z), usando 3 argumentos de entrada. O primeiro é um vetor contendo os coeficientes do polinômio B(z) da Equação 14.2; o segundo é um vetor contendo os coeficientes do polinômio A(z); e o terceiro é para especificar o número de valores de freqüências normalizadas que se quer no intervalo de 0 a π. Este número de pontos define a resolução. Através deste ajuste da resolução podemos determinar os tipos de filtros e as freqüências críticas.

Programa:

```
B1 = [0.2066,0.4131,0.2066];
A1 = [1,-0.3695,0.1958];
[H1z,w1T] = freqz(B1,A1,100);


B2 = [0.894,-1.789,0.894];
A2 = [1,-1.778,0.799];
[H2z,w2T] = freqz(B2,A2,100);

B3 = [0.42,0,-0.42];
A3 = [1,-0.443,0.159];
[H3z,w3T] = freqz(B3,A3,100);

B4 = [0.5792,0.4425,0.5792];
A4 = [1,0.4425,0.1584];
[H41z,w1T] = freqz(B4,A4,100);
clg
subplot (221),plot(w1T,abs(H1z)),title('Fitro H1(z)'),...
xlabel ('w,rps'), y;label('Magnitude'), grid
subplot (222),plot(w2T,abs(H2z)),title('Fitro H2(z)'),...
xlabel ('w,rps'), y;label('Magnitude'), grid
subplot (223),plot(w3T,abs(H3z)),title('Fitro H3(z)'),...
xlabel ('w,rps'), y;label('Magnitude'), grid
subplot (224),plot(w4t,abs(H4z)),title('Fitro H4(z)'),...
xlabel ('w,rps'), y;label('Magnitude'), grid
```

A fase de um filtro digital pode ser plotada usando-se a função angle ou a unwrap.

P r a t i q u e !

Para estas funções de transferência, plote a magnitude. Use freqüência normalizada no eixo X para filtros digitais.

$$1.\ H(s) = \frac{s^2}{s^2 + \sqrt{2}s + 1}$$

$$2.\ H(s) = \frac{0.707z - 0.707}{z - 0.414}$$

$$3.\ H(s) = -0.163 - 0.058z^{-1} + 0.116z^{-2} + 0.2z^{-3} + 0.116z^{-4} - 0.058z^{-5} - 0.163z^{-6}$$

$$4.\ H(s) = \frac{5s + 1}{s^2 + 0.4s + 1}$$

14.3 Implementação de Filtros Digitais

Filtros analógicos são implementados com componentes eletrônicos como resistores e capacitores. Os filtros digitais são implementados por software. Os filtros digitais podem ser representados por funções de transferência ou equações diferenciais.

A relação entre o sinal de entrada x_n e o sinal de saída y_n é descrita pela equação diferencial, que está escrita na forma geral:

$$y_n = \sum_{k=-N_1}^{N_2} b_k x_{n-k} - \sum_{k=1}^{N_3} a_k y_{n-k}$$

Exemplos:

$$y_n = 0.04x_{n-1} + 0.17x_{n-2} + 0.25x_{n-3} + 0.17x_{n-4} + 0.04x_{n-5}$$

$$y_n = 0.42x_n - 0.42x_{n-2} + 0.44x_{n-1} - 0.16x_{n-2}$$

$$y_n = 0.33x_{n+1} + 0.33x_n + 0.33x_{n-1}$$

As três equações diferenciais representam diferentes tipos de filtros. A saída do primeiro depende somente dos valores anteriores do sinal de entrada. O segundo filtro requer valores não apenas de entrada mas também os valores anteriores da saída. O terceiro filtro depende apenas dos valores de entrada. Porém os valores de entrada necessários são os posteriores e isso pode ser problemático quando os dados são adquiridos em tempo real. Estes filtros são FIR porque os denominadores são iguais a 1.

A função *filter* do MATLAB assume a equação diferencial na seguinte forma:

$$y_n = \sum_{k=0}^{N_2} b_k x_{n-k} - \sum_{k=1}^{N_3} a_k y_{n-k}$$

que corresponde a seguinte função de transferência:

$$H(z) = \frac{B(z)}{A(z)}$$

$$H(s) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + ... + b_n z^{-n}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + ... + a_n z^{-n}}$$

Os dois primeiros argumentos da função filter são vetores com os coeficientes de $[b_k]$ e de $[a_k]$. O terceiro argumento é o sinal de entrada.

Exemplo1:

```
B = [0.0,0.04,0.17,0.25,0.17,0.04];
A = [1];
Y = filter(B,A,x);
```

Exemplo2:

```
B = [0.42,0.0,-0.42];
A = [-0.44,0.16];;
Y = filter(B,A,x);
```

Nós podemos usar a função filter para o terceiro filtro porque a equação não está ajustada na forma geral usada pela função. A terceira equação precisa começar em k=0 e não em k= -1. Neste caso implementamos o filtro utilizando um vetor aritmético. Assumindo um sinal de entrada x constrói-se um vetor x, que pode calcular o sinal de saída correspondente usando a seguinte forma:

```
N = length(x);
y(1) = 0.33*x(1) + 0.33*x(2)
for n=2:N-1
        y(n) = 0.33*x(n+1) + 0.33*x(n) + 0.33*x(n-1);
end
y(N) = 0.33*x(N-1) + 0.33*x(N);
```

Uma outra forma é a seguinte:

```
N = length(x);
y(1) = 0.33*x(1) + 0.33*x(2);
y(2:N-1) = 0.33*x(3:N) + 0.33*x(2:N-1) + 0.33*x(1:N-2);
y(N) = 0.33*x(N-1) + 0.33*x(N);
```

A seguinte função de transferência foi projetada para deixar passar freqüências entre 500Hz e 1500Hz, de um sinal de até 5KHz.

$$H(z) = \frac{0.42z^2 - 0.42}{z^2 - 0.443z + 0.159}.$$

Use os seguintes sinais de entrada no filtro. Plote a entrada e a saída do filtro no mesmo gráfico e determine o efeito do filtro no sinal de entrada.

1. $x_k = \sin(2\pi 1000kT)$
2. $x_k = 2\cos(2\pi 100kT)$
3. $x_k = -\sin(2\pi 2000kT)$
4. $x_k = \cos(2\pi 1600kT)$

14.4 Projeto de Filtros Digitais

A discussão será separada em duas técnicas: uma para filtros IIR e outra para filtros FIR.

Projeto de Filtros IIR usando protocolos analógicos

O MATLAB possui quatro tipos de filtros digitais baseados em projetos de filtros analógicos. Os filtros Butterworth são usados como passa-banda e corta-banda, os filtros Chebyshev Tipo I são usados com ripple na banda de passagem, os filtros Chebyshev Tipo II são usados com ripple na banda de corte. Os filtros elípticos possuem uma faixa de transição mais definida que o filtro Butterworth com as mesmas especificações.

As funções para se projetar filtros IIR digitais que usam protocolos analógicos possuem o seguinte formato:

```
[B,A] = butter (N,Wn);
[B,A] = cheby1butter (N,Rp,Wn);
[B,A] = cheby2utter (N,Rs,Wn);
[B,A] = ellip (N,Rp,Rs,Wn);
```

O argumento de entrada representa a ordem do filtro (N), o ripple (Rs e Rp), e a freqüência de corte normalizada (Wn). Os vetores de saída B e A são os vetores da expressão geral para filtros IIR e podem ser utilizados para se achar a função de transferência ou a equação diferencial.

Para se projetar filtros passa-banda os argumentos das funções são os mesmos do passa-baixa. Entretanto o vetor Wn precisa conter 2 elementos que representam as freqüências normalizadas especificadas da banda de freqüência, como Wn(1) e Wn(2).

Para se projetor filtros passa-alta, um parâmetro adicional com o nome ‘high’ precisa ser adicionado e ficam assim as novas formas:

```
[B,A] = butter (N,Wn,’high’);
[B,A] = cheby1butter (N,Rp,Wn,’high’);
[B,A] = cheby2utter (N,Rs,Wn,’high’);
[B,A] = ellip (N,Rp,Rs,Wn,’high’);
```

Para se projetar filtros corta-banda, os argumentos são os mesmos dos filtros passa-alta, mas com o termo ‘stop’ ao invés de ‘high’. O argumento Wn precisa ser um vetor que contenha 2 valores definem a banda de freqüência como Wn(1) e Wn(2) para serem rejeitadas.

Para ilustrar estas funções supomos que precisemos projetar um filtro passa-alta Chebyshev Tipo II de ordem 6. Nós queremos como limite de passabanda um ripple de 0,1 ou 20db. O filtro é para ser usado para um sinal de até 1KHz. A freqüência de corte é de 300HZ e a freqüência normalizada é de 300/500 ou 0.6. Os comandos para o projeto deste filtro e para plotar a magnitude característica são os seguintes:

```
[B,A] = cheby2 (6,20,0.6,’high’);
[H,wT] = freqz (B,A,100);
T = 0.001;
hertz = wT/(2*pi*T);
plot (hertz, abs(H)), title(‘Filtro Passaalta’),...
xlabel (‘Hz’), ylabel (‘Magnitude’),grid
```

Para aplicar neste filtro um sinal x basta usar a seguinte instrução:

```
Y = filter (B,A,x);
```

Projeto de um Filtro IIR

O MATLAB possui uma função para projetar filtros com o método Yule-Walker. Esta técnica de projeto pode ser usada para projetar formas arbitrárias, possibilitando projetar respostas freqüenciais multibandas.

O comando para projetar um filtro com esta função é o seguinte:

```
[B,A] = yulewalk(n,f,m);
```

Os vetores de saída B e A contém os coeficientes de n ordem do filtro IIR. Os vetores f e m especificam as características freqüência-magnitude do filtro. As freqüências em f precisam começar em 0, terminando em 1 e serem crescentes. A magnitude m precisa corresponder com a freqüência f, e representa a magnitude esperada para a freqüência correspondente. O exemplo a seguir nos mostra um projeto de um filtro com dois passabandas e que plota a magnitude de resposta na freqüência normalizada.

```
m = [0 0 1 1 0 0 1 1 0 0 ];
f = [0 .1 .2 .3 .4 .5 .6 .7 .8 1];
[B,A] = yulewalk (12,f,m);
[H,wT] = freqz(B,A,100);
plot (f,m,wT/pi,abs (H)),...
title('Filtro IIR com dois passabandas'),...
xlabel ('Freqüência Normalizada'),...
ylabel ('Magnitude'), grid
```

Projeto Direto de um Filtro FIR

Os filtros FIR são projetados pelo MATLAB usando o método de Parks-McClellan que usa o algoritmo de troca de Remez. Lembrando que os filtros FIR necessitam apenas do vetor B, pois o denominador é igual a 1. Portanto a função Remez calcula apenas um único vetor, como mostrado a seguir:

```
B = remez (n,f,m);
```

O primeiro argumento define a ordem do filtro, e os valores de f e m são similares ao da técnica do filtro de Yule-Walker.

O exemplo a seguir nos mostra um projeto de um filtro com dois passabandas e que plota a magnitude de resposta na freqüência normalizada.

```
m = [0 0 1 1 0 0 1 1 0 0 ];
f = [0 .1 .2 .3 .4 .5 .6 .7 .8 1];
B = remez (50,f,m);
[H,wT] = freqz(B,[1],100);
plot (f,m,wT/pi,abs (H)),...
title('Filtro FIR com dois passabandas'),...
xlabel ('Freqüência Normalizada'),...
ylabel ('Magnitude'), grid
```

P r a t i q u e !

Use as funções do MATLAB descritas nesta seção para projetar os seguintes filtros. Plote a magnitude do filtro projetado para confirmar se as características estão corretas.

1. Filtro IIR passabaixa com corte de 75Hz, onde usa-se uma freqüência de até 500Hz.(Use um filtro de ordem 5)
2. Filtro IIR passaalta com corte de 100Hz, onde usa-se uma freqüência de até 1KHz.(Use um filtro de ordem 6)
3. Filtro FIR passabaixa com corte de 75Hz, onde usa-se uma freqüência de até 500Hz.(Use um filtro de ordem 40)

4. Filtro FIR passabanda com faixa de freqüência entre 100Hz e 200Hz, onde usa-se uma freqüência de até 1KHz.(Use um filtro de ordem 80)

Aplicação na Solução de Problemas: Filtros para Separação de Canais

Imagens coletadas de uma nave espacial ou de satélites que circundam a terra são enviadas para a terra como um fluxo de dados. Estes fluxos de dados são convertidos em sinais digitais que contém informações que podem ser recombinadas e assim reconstruir as imagens originais. Informações coletadas por outros sensores também são transmitidas para a terra. A freqüência do sinal de informação do sensor depende dos tipos de dados que estão sendo medidos.

Técnicas de modulação podem ser usadas para mover o conteúdo da freqüência para a banda de freqüência especificada. Desta forma o sinal pode conter sinais múltiplos ao mesmo tempo. Por exemplo, supomos que queremos enviar 3 sinais em paralelo. O primeiro sinal contém componentes entre 0 e 100Hz, o segundo sinal contém componentes entre 500 e 1KHz, e o terceiro contém componentes entre 2KHz e 5KHz.

O sinal contendo estas três componentes é de até 10KHz. Para separar estas três componentes precisamos de um filtro passabaixa com corte de 100HZ, um passabanda com faixa de 500Hz a 1KHz, e um passa alta com corte de 2KHz. A ordem dos filtros precisa ser suficientemente grande para gerar um pequena banda de transição para que a componente de uma freqüência não contamine as outras componentes.

1. DESCRIÇÃO DO PROBLEMA
   Projete três filtros para serem usados com um sinal de até 10KHz. Um filtro é um passabaixa com corte de 100HZ, um passabanda com faixa de 500Hz a 1KHz, e um passa alta com corte de 2KHz.

2. DESCRIÇÃO DOS DADOS DE ENTRADA E SAÍDA
   Não há valores de entrada para este problema. Os valores de saída são coeficientes dos vetores que definem os três filtros $H_1(z)$, $H_2(z)$ e $H_3(z)$, como na figura a seguir:

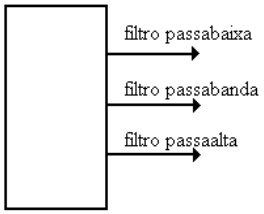


Figura 4 – Diagrama de I/O

3. EXEMPLO MANUAL

O espectro abaixo nos mostra a amostra de freqüência com os três sinais filtrados, Nós usaremos os filtros Butterworth. Precisaremos experimentar várias ordens para sabermos se as bandas de transição não irão interferir uma nas outras.

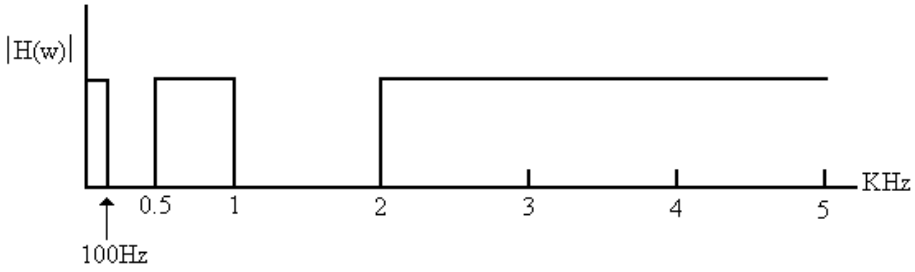


Figura 4 – Espectro de faixa dos filtros

4. SOLUÇÃO MATLAB

O seguinte programa MATLAB determina os valores da freqüência normalizada para as freqüências de corte pela função *butter*. Depois de calculados os coeficientes para os filtros, nós usamos a função *freqz* para plotar as características dos filtros. Lembrando que a função *freqz* normaliza as freqüências para valores entre 0 e π. Nós usamos Hz como unidade no eixo X, para facilitar a visualização das características dos filtros projetados.

5. TESTANDO

As magnitudes dos três filtros são mostradas no mesmo gráfico para podermos verificar se os filtros não se sobrepõem.

Sumário MATLAB

butter	projeta um filtro digital Butterworth
cheby1	projeta um filtro digital Chebyshev Tipo 1
cheby2	projeta um filtro digital Chebyshev Tipo 2
ellip	projeta um filtro digital elíptico
fft	calcula a freqüência de um sinal
filter	aplica um filtro digital em um sinal de entrada
freqs	calcula a freqüência analógica
freqz	calcula a freqüência digital
grpdelay	mede o grupo de retardo de um filtro digital
ifft	calcula a inversa da transformada de Fourier
remez	projeta um filtro digital FIR
unwrap	remove a descontinuidade 2π da fase angular
yulewalk	projeta um filtro digital IIR

PROBLEMAS:

1. Nós usaremos a soma de senóides de até 10KHz. O primeiro sinal contém uma soma de senóides com freqüências de 24Hz, 40 Hz e 75Hz. O segundo sinal contém uma soma de senóides com freqüências de 500Hz, 730 Hz e 850Hz. O terceiro sinal contém uma soma de senóides com freqüências de 3500Hz, 4000 Hz e 4200Hz. Escolha as amplitudes e fases para estas senóides. Plote 500 pontos para o sinal 1, 2 e 3 em diferentes gráficos.

2. Calcule e plote a magnitude e a fase dos três sinais do exercício anterior. Use Hz como unidade no eixo x .

3. Adicione três tempos no sinal gerado no exercício 1. Plote o sinal e sua magnitude usando o Hz como unidade no eixo x.

4. Aplique um filtro passabaixa, um passabanda e um passaalta no sinal do problema 3 e plote a magnitude da freqüência dos sinais de saída.

5. Para os seguintes filtros determine a banda de passagem, banda de transição e banda de corte. Use 0.7 para determinar as freqüências de corte e use 0.1 para determinar as freqüências de rejeição.

$$H(s) = \frac{0.5279}{s^2 + 1.0275s + 0.5279}$$

$$H(s) = \frac{s^2}{s^2 + 0.1117s + 0.0062}$$

$$H_1(z) = \frac{0.2066 + 0.4131z^{-1} + 0.2066z^{-2}}{1 - 0.3695z^{-1} + 0.1958z^{-2}}$$

$$H_2(z) = \frac{0.894 - 1.789z^{-1} + 0.894z^{-2}}{1 - 1.778z^{-1} + 0.799z^{-2}}$$

6. Projete um filtro para remover freqüências entre 500Hz e 1000Hz de um sinal de até 10KHz. Compare os resultados dos filtros elíptico e do filtro Yule-Walker, ambos de ordem 12. Plote a magnitude dos dois sinais.

7. Projete um filtro para remover freqüências entre 100Hz e 150Hz, e outro entre 500Hz e 600Hz de um sinal de até 2.5KHz. Compare os resultados usando FIR e IIR. Plote a magnitude dos dois sinais

Capítulo 15 - Matemática Simbólica

Introdução

Agora é possível instruir ao MATLAB que manipule expressões matemáticas, sem de fato usar números, que lhe permitam calcular com símbolos matemáticos, além de números. Esse processo é freqüentemente chamado de matemática simbólica. Aqui estão alguns exemplos de expressões simbólicas:

$\cos(x^2)$ $3x^2 + 5x - 1$ $v = d\ x^2$ $f = \int x^2\, dx$

A toolbox de Matemática Simbólica é uma coleção de funções para o MATLAB usadas para manipular e resolver expressões simbólicas. Há diversas ferramentas para combinar, simplificar, derivar, integrar e resolver equações diferenciais e algébricas. Outras ferramentas são utilizadas em álgebra linear para derivar resultados exatos para inversas, determinantes e formas canônicas e para encontrar os autovalores de matrizes simbólicas, sem o erro introduzido pelo cálculo numérico.

A aritmética de precisão variável que calcula simbolicamente e retorna um resultado para qualquer grau de precisão especificado, também está disponível no MATLAB.

As ferramentas contidas na toolbox de matemática simbólica foram criadas por meio de um poderoso programa de software chamado Maple, originalmente desenvolvido na Universidade de Waterloo, no Canadá. Quando você pedir ao MATLAB para executar alguma operação simbólica, ele solicitará ao Maple para fazê-lo e então retornará o resultado para a janela de comando do MATLAB. Por isso, fazer manipulações simbólicas no MATLAB é uma extensão natural do modo como você usa o MATLAB para processar números.

15.1 Expressões Simbólicas

Expressões simbólicas são strings de caracteres ou conjuntos de strings de caracteres que representam números, funções, operadores e variáveis. as variáveis não têm de ter valores previamente definidos. Equações simbólicas são expressões simbólicas que contêm um sinal de igualdade. A aritmética simbólicas é a prática de resolução dessas equações por meio da aplicação de regras conhecidas e de identidades a determinados símbolos, exatamente da forma que você aprendeu a resolvê-las em álgebra e cálculo. Matrizes simbólicas são conjuntos cujos elementos são expressões simbólicas.

Representações de Expressões Simbólicas no MATLAB

O MATLAB representa expressões simbólicas internamente como strings de caracteres para diferenciá-las de variáveis numéricas e operadores. Aqui estão alguns exemplos de expressões simbólicas com seus equivalentes em MATLAB:

Expressões simbólicas	Representação no MATLAB
$\frac{1}{2x^n}$	'1 (2 * x ^ n) '
$\cos(x^2)$ - sen (2x)	' cos(x ^ 2) - sin(2 * x) '
$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$	sym (' [a , b ; c , d] ')

Exercícios:

1. diff('cos(x)')
2. M = sym ('[a,b;c,d]')
3. determ(M)

Observe que, no primeiro exercício acima, a expressão simbólica foi definida implicitamente pelo uso de aspas simples para dizer ao MATLAB que ('cos(x)') seja uma expressão simbólica, em vez de uma expressão numérica, ao passo que, no segundo exemplo, a função *sym* foi usada para, explicitamente, dizer ao MATLAB que M = *sym* ('[a,b;c,d]') é uma expressão simbólica. Geralmente a função explícita sym não é necessária onde o MATLAB puder determinar, por si só o tipo de argumento.

Em MATLAB, a forma *func arg* é equivalente a func('arg'), onde *func* é uma função e *arg* é um argumento na forma de string de caracteres. Por exemplo, o MATLAB consegue descobrir que diff cos(x) e diff ('cos(x)') significam a mesma coisa, ou seja, diff(sym('cos(x)')), mas a primeira forma é certamente mais fácil de ser digitada. Contudo, a função *sym* é necessária. Isto fica claro no exemplo seguinte:

Exemplo

M=[a,b;c,d] M é uma matriz numérica usando valores de a até d
M='[a,b;c,d]' M é uma string de caracteres
M=sym('[a,b;c,d]') M é uma matriz simbólica

Aqui M foi definido de três maneiras: numericamente (se a, b, c e d tiverem sido predefinidos), como uma string de caracteres e como uma matriz simbólica.

Muitas funções simbólicas são suficientemente avançadas para converter strings de caracteres em expressões simbólicas de forma automática. Mas, em alguns casos, em especial na criação de um conjunto simbólico, a função sym deve ser usada para converter um string de caracteres especificamente em uma expressão simbólica. A forma implícita, ou seja, diff cos(x), é mais útil para tarefas simples que não referenciam resultados anteriores. Contudo, a forma mais simples (sem aspas) requer um argumento que é um string de caracteres simples sem espaços em seu interior:

Exemplo

```
diff x^2+3*x+5
diff x^2 + 3*x + 5
```

Na segunda equação os espaços pressupões strings de caracteres distintos.

As expressões simbólicas sem variáveis são chamadas de constantes simbólicas. Quando as constantes simbólicas são visualizadas, freqüentemente torna-se difícil distingui-las de números inteiros.

Exemplo

```
f=symop('(3*4-2)/5+1')
isstr(f)
```

A função isstr nada mas faz do que retornar se f é uma string de caracteres (1 = sim e 0 = não). Nesse caso, f representa a constante simbólica '3' não o número 3. O MATLAB armazena strings de caracteres como representação ASCII de caracteres. Consequentemente, se você realizar uma operação numérica em um string de caracteres, o MATLAB usa o valor ASCII de cada caractere na operação.

Exemplo

```
f +1
```

15.2 Variáveis Simbólicas

Quando se trabalha com expressões simbólicas contendo mais de uma variável, uma variável a é a variável independente, ele seleciona uma baseado na regra seguinte:

A variável independente padrão em uma expressão simbólica é uma letra minúscula única, que não seja i ou j, que não faça parte de uma palavra. Se não há tal caractere, é escolhido x. Se o caractere não for único, aquele mais próximo de x, alfabeticamente falando, é escolhido. Se houver empate, o caractere posterior no alfabeto será escolhido.

A variável independente padrão, algumas vezes conhecida como variável livre, na expressão '1 / (5+cos(x))' é 'x'; a variável livre na expressão '3*y+z' é 'y'; e a variável livre na expressão ' a+sin(t)' é 't'. A variável simbólica livre na expressão ' sin(pi/4) -

cos(3/5)' é 'x' porque esta expressão é uma constante simbólica, não contendo variáveis simbólicas.

Você pode pedir ao MATLAB para lhe dizer qual a variável em uma expressão simbólica é considerada por ele como a variável independente usando a função *symvar*.

Exemplo

```
symvar('a*x+y')          encontra a variável simbólica padrão
symvar ('a*t+s/(u+3)') u é mais próxima de x
symvar('sin(omega)')   'omega' não é um caracter único
symvar('3*i+4*j')        i e j são iguais a sqrt(-1)
symvar('y+3*s','t')      encontra a variável mais próxima de 't'
```

Se symvar não encontrar uma variável padrão, usando esta regra, considerará que ela não existe e retornará x. Isto será verdadeiro para expressões que contenham variáveis com múltiplos caracteres, bem como constantes simbólicas, as quais não contêm variáveis.

Muitos comandos dão a você a opção de especificar a variável independente, se desejado:

Exemplo

```
diff('x^n')                  deriva em relação à variável padrão 'x';
diff('x^n','n')              deriva em relação a n;
diff('sin(omega)')           deriva usando a variável padrão;
diff('sin(omega)','omega')   especifica a variável independente;
```

E x e r c í c i o s

Dada cada expressão simbólica, use a sintaxe do MATLAB para criar a expressão simbólica equivalente do MATLAB.

a . $f = ax^2 + bx + c$

b. $p = \dfrac{3s^2 + 2s + 1}{4s - 2}$

c. $r = e^{-2t}$

d. $\dfrac{d}{dx}\sqrt{3x^2 + 2x + 5}$

2. Encontre a variável independente padrão que é retornada por *symvar* nas expressões seguintes:

a . z = 3ac +4b -2
b. $x = 4a + 3c + b^2 + 1$
c. $q = r + \sqrt{3k^2 + 2p}$
d. $f = s^{3nt}$
e. $n = 3r + 2s^2 + 5$

15.3 Operações em Expressões Simbólicas

Depois de criar uma expressão simbólica, você provavelmente vai querer mudá-la de alguma forma. Você pode querer extrair parte de uma expressão, combinar duas expressões ou achar o valor numérico de uma expressão simbólica. Há muitas ferramentas simbólicas que lhe permitirão realizar essas tarefas.

Quase todas as funções simbólicas agem sobre expressões simbólicas e conjuntos simbólicos e retornam expressões simbólicas ou conjuntos. O resultado pode ás vezes se parecer com um número, mas é uma expressão simbólica representada internamente como um string de caracteres. Conforme foi dito antes, é possível descobrir se o resultado que parece ser um número é um inteiro ou um string de caracteres utilizando a função *isstr* do MATLAB.

Extração de Numeradores e Denominadores

Se sua expressão for um polinômio racional (uma razão de dois polinômios) ou puder ser expandida em um polinômio racional (incluindo aqueles com um denominador igual a 1), você pode extrair o numerador e o denominador usando *numden*. Dadas as expressões:

$$m = x^2$$

$$f = \frac{ax^2}{b - x}$$

$$g = \frac{3}{2}x^2 + \frac{2}{3}x - \frac{3}{5}$$

$$h = \frac{x^2 + 3}{2x - 1} + \frac{3x}{x - 1}$$

$$k = \begin{bmatrix} \frac{3}{2} & \frac{2x+1}{3} \\ \frac{4}{x2} & 3x+4 \end{bmatrix}$$

numden combina e racionaliza a expressão, se necessário, e retorna o numerador e o denominador resultantes. As linhas de comando do MATLAB para se fazer isso são:

```
m = 'x^2'
[n,d]= numden(m)

f = 'a*x^2 / (b - x)'

[n,d]=numden(f)

g = '3/2*x^2+2/3*x-3/5'

[n,d] = numden(g)

h = '(x^2+3)/(2*x-1)+3*x/(x-1)'

[n,d] = numden(h)

k = sym ('[3/2,(2*x+1)/3;4/x^2,3*x+4]')

[n,d] = numden(k)
```

Essa expressão , k, é um conjunto simbólico, *numden* retornou dois novos conjuntos, n e d, onde n é um conjunto de numeradores e d o conjunto de denominadores. Se você usar a forma s = numden(f), *numden* retorna apenas o numerador dentro da variável s.

15.4 Operações Algébricas Padrão

Diversas operações algébricas podem ser executadas em expressões simbólicas. As funções *symadd*, *symsub* e *symdiv* somam, subtraem, multiplicam e dividem, respectivamente, duas expressões, e *sympow* eleva uma expressão à potência da outra.

Exemplo

$f = 2x^2 + 3x - 5$

$g = x^2 - x + 7$

No MATLAB:

```
f = '2*x^2+3*x-5'
g = 'x^2-x+7
symadd(f,g)
symsub(f,g)
symmul(f,g)
symdiv(f,g)
sympow(f, '3*x')      %Encontra a expressão para f^3
```

Uma outra função de aplicação geral permite que você crie novas expressões a partir de outras variáveis simbólicas, expressões e operadores. *Symop* recebe até 16 argumentos separados por vírgula, cada um dos quais pode ser uma expressão simbólica, um valor numérico ou um operador (+,-,*,/,^,(,ou)). *Symop* então concatena os argumentos e retorna à expressão resultante.

Exemplo

```
f = 'cos(x)'
g = 'sin(2*x)'
symop(f, '/ ', g , '+' , 3)
```

Todas estas operações também trabalham com conjuntos de argumentos da mesma forma.

Operações Avançadas

O MATLAB tem a capacidade de realizar operações mais avançadas em expressões simbólicas. A função compose combina f(x) e g(x) em f(g(x)), a função finverse encontra o inverso funcional da expressão, e a função symsum encontra o somatório simbólico de uma expressão.

Dada as expressões:

1. $f = \frac{1}{1 + x^2}$

2. g=sen(x)

3. $h = \frac{1}{1 + u^2}$

4. k = sen(v)

No MATLAB

```
f = '1/(1+x^2)';
g = 'sin(x)';
h = '1/(1+u^2)';
k= 'sin(v)';
compose(f,g)
compose(g,f)
```

Compose pode também ser usada em funções que tem variáveis independentes diferentes:

```
compose ( h, k, 'u', 'v')
```

O inverso funcional de uma expressão, digamos f(x), é a expressão g(x) que satisfaz a condição g(f(x)) = x. Por exemplo, o inverso funcional de e^x é ln(x), já que $\ln(e^x) = x$.

A função *finverse* retorna o inverso funcional de uma expressão e avisa a você se o resultado não for único.

Exemplos

```
finverse ('x^2')
finverse ('a*x+b')
finverse ('a*b+c*d-a*z','a')
```

A função symsum encontra o somatório simbólico de uma expressão. Há quatro formas para a função:

MATLAB	Retorna
symsum(f)	$\sum_{0}^{x-1} f(x)$
symsum(f, a, b)	$\sum_{0}^{b} f(x)$
symsum(f, 's')	$\sum_{a}^{s-1} f(s)$
symsum(f, 's', a , b)	$\sum_{a}^{b} f(s)$

Exemplos

1. $\sum_{0}^{x-1} x^2$

2. $\sum_{1}^{n} (2n-1)^2$

3. $\sum_{1}^{\infty} \frac{1}{(2n-1)^2}$

15.5 Funções de Conversão

Essa seção apresenta ferramentas para conversão de expressões simbólicas em valores numéricos e vice-versa.

A função *sym* pode receber um argumento numérico e convertê-lo em uma representação simbólica. A função *numeric* faz o contrário, converte uma constante simbólica em um valor numérico.

Exemplo

```
phi= '(1+sqrt(5))/2'
numeric (phi)
```

A função *eval* faz com que o MATLAB calcule um string de caracteres. Dessa forma, *eval* é outra função que pode ser usada para converter constante simbólica em um número, ou para calcular uma expressão.

Exemplo

```
eval(phi)
```

Você já trabalhou com polinômios em MATLAB, usando vetores cujos elementos são os coeficientes dos polinômios. A função simbólica *sym2poly* converte um polinômio simbólico em seu vetor de coeficientes equivalente no MATLAB. A função *poy2sym* faz o inverso e possibilita que você especifique a variável a ser usada na expressão resultante.

Exemplo

```
f= '2*x^2+x^3-3*x+5'
n=sym2poly(f)
poly2sym(n)
poly2sym(n, 's')
```

Substituição de Variáveis

Quando se tem uma expressão simbólica em x, e que queira mudar a variável para y, usa-se a função *subs*.

Exemplo

```
f= ‘a*x^2+b*x+c’
subs(f, ‘s’, ‘x’)
h= subs(f, ‘2’, ‘x’)
```

15.6 Derivação e Integração

Derivação

A derivação de uma expressão simbólica usa a função diff em uma dentre as quatro formas:

Expressão	Resultado
f= ‘a*x^3+x^2-b*x-c’	Define uma expressão simbólica
diff(f)	Deriva em relação ao padrão (x)
diff(f, ‘a’)	Deriva f em relação a (a)
diff(f, 2)	Deriva f duas vezes em relação a (x)
diff(f, ‘a’,2)	Deriva a função f duas vezes em relação a (a)

A função *diff* também opera sobre conjuntos. Se f for um vetor simbólico ou matriz, *diff(f)* deriva cada elemento do conjunto:

Exemplo

```
f= sym(‘[a*x, b*x^2, c*x^3, d*s]’)
diff(f)
```

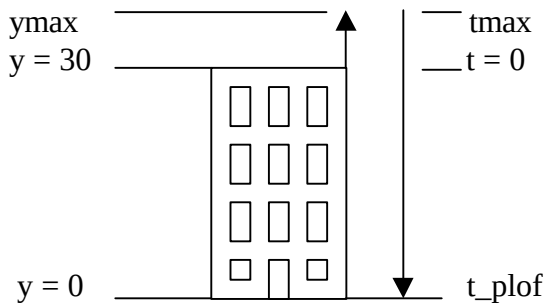
Integração

A função de integração *int*(f), onde f é uma expressão simbólica, tenta encontrar outra expressão simbólica F tal que *diff*(F)=f.

Expressão	Resultado
f= ‘sin(s+2*x)’	Cria uma função simbólica
int(f)	Integra em relação a x
int(f, ‘s’)	Integra em relação a s
int(f, pi/2, pi)	Integra em relação a x de pi/2 a pi
int(f, ‘s’, pi/2, pi)	Integra em relação a s de pi/2 a pi
int(f, ‘m’,‘n’)	Integra em relação a x de m para n

Exercício:

Hélio está em uma excursão com sua turma de escola, no alto do edifício Central. Ele pega um tomate maduro em sua mochila, debruça-se sobre a beirada do terraço e atira-o no ar. O tomate é jogado para cima, a uma velocidade inicial vo = 20m/s. O terraço encontra-se a yo = 30 metros acima do nível do solo. Onde estará o tomate a um intervalo arbitrário de t segundos mais tarde? Quando ele alcançará a altura máxima? Que altura acima do solo o tomate conseguirá alcançar? Quando o tomate atingirá o solo? Considere que não há resistência do ar e que g = 10 m/s.

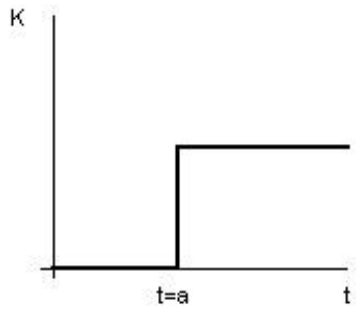


15.7 Transformadas

As transformadas são usadas com muita freqüência em Engenharia para mudar o campo de referência entre o domínio do tempo e o domínio s, domínio da freqüência ou domínio Z. Há muitas técnicas para analisar estados de equilíbrio e sistemas que sofrem mudanças muito suaves no domínio do tempo, mas os sistemas complexos quase sempre podem ser analisados mais facilmente em outros domínios.

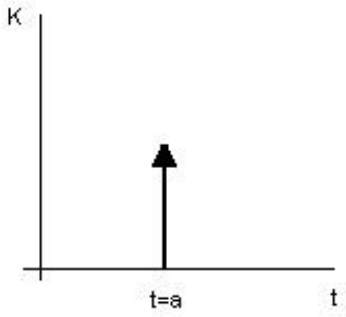
Funções Degrau e Impulso

Os problemas de Engenharia freqüentemente fazem uso da função Degrau u(t) e impulso δ(t) na descrição de sistemas. A função Degrau Ku(t - a), onde K é uma constante, é definida como Ku(t - a) = 0 para t < a e Ku(t - a) = K para t >a . Eis um gráfico da função Degrau Ku(t - a):



A função impulso δ(t) é a derivada da função Degrau u(t). A função Degrau K δ(t-a) é definida como K δ(t-a) = 0 para t < a e $\int_{\infty}^{-\infty} Kd(t-a)dt = K$ para t = a

Quando representada em gráfico ela é comumente representada como uma seta de amplitude K em t = a . Eis o gráfico de K δ(t-a):



Exemplo

```
u = 'k*Heaviside(t-a)'
d = diff(u)
int(d)
```

Transformada de Laplace

A Transformada de Laplace realiza a operação F(s)= $\int_0^\infty f(t)e^{-st}dt$. Para transformar f(t), no domínio do tempo, em F(s), no domínio de s.

A Transformada de Laplace da função cosseno amortecido e^{-at} cos(wt) é encontrada usando-se a função Laplace:

```
f=sym('exp(-a*t)*cos(w*t)')
F= laplace(f)
pretty(F)
laplace('Dirac(t)')
laplace('Heaviside(t)')
```

As expressões podem ser transformadas novamente para o domínio do tempo, usando-se o inverso da transformada de Laplace, invlaplace, que realiza a operação f(t). Usando F do exemplo acima temos:

```
invlaplace(F)
```

Transformada de Fourier

A Transformada de Fourier e sua inversa são muito usadas em análise de circuitos para determinar as características de um sistema em ambos os domínios de tempo e de freqüência. O MATLAB usa as funções fourier e invfourier para transformar expressões entre domínios. A Transformada de Fourier e sua inversa são definidas como:

$$F(w)= \int_{-\infty}^{\infty} f(t)e^{-jwt}dt \qquad f(t) = \frac{1}{} \int_{-\infty}^{\infty} F(w)e^{jwt}dw$$

O MATLAB usa um w para representar ω em expressões simbólicas.

Exemplo

```
f= 't*exp(-t^2)'
F= fourier(f)
invfourier(F)
```

Transformada Z

As transformadas de Laplace e Fourier são usadas para analisar sistemas de tempo contínuos. Transformadas Z, por outro lado, são usadas para analisar sistemas de tempo discreto. A Transformada Z é definida como:

$$F(z)=\sum_{n=0}^{\infty} f(v)z^{-n}$$

onde z é um número complexo.

A Transformada z e a Transformada z inversa são obtidas usando-se as funções *ztrans* e *invztrans*. O formato é similar ao das funções de transformadas de Laplace e Fourier.

Exemplo

```
f= '2 ^ n / 7- (-5) ^ n / 7'
G= ztrans(f)
pretty(G)
invtrans(G)
```