Material de Apoio 1

Introdução ao Matlab

1. Trabalhando com Vetores e Matrizes

O aplicativo *Matlab* é uma das ferramentas mais úteis para as áreas de processamento de sinais. O "Mat" do nome *Matlab* está relacionado com "matriz", ou seja, é um laboratório que permite a manipulação eficiente de matrizes.

A forma mais básica de se criar uma matriz no *Matlab* consiste em entrar com todos os seus elementos, conforme a expressão de exemplo a seguir, que cria uma matriz 3x3:

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

O *Matlab* mostrará o seguinte resultado:

```
A =

     1     2     3

     4     5     6

     7     8     9
```

Caso queiramos omitir o resultado mostrado pelo *Matlab*, podemos incluir um ";" no final da expressão:

```
>> A = [1 2 3; 4 5 6; 7 8 9];
```

Um vetor nada mais é que uma matriz **1xN**, ou seja, uma matriz cujos N elementos estejam dispostos em uma única linha.

```
>> v = [5 2 9 4 1 0 0];
```

Podemos também concatenar elementos de matrizes diferentes em uma única matriz:

```
>> a = [1 2 3];

>> b = [4 5 6];
```

```
>> c = [a b]

c =

      1     2     3     4     5     6
```

Ou podemos concatenar os elementos na forma de linhas de uma nova matriz:

```
>> C = [a ; b]

C =

      1     2     3

      4     5     6
```

É importante observar que este último comando não funcionará se o número de colunas de "a" e "b" não for o mesmo.

O *Matlab* também trabalha com matrizes cujos elementos são números complexos:

```
>> A = [1+j -5+3j; -2-j 1-j]

A =

      1.0000 + 1.0000i        -5.0000 + 3.0000i

      -2.0000 - 1.0000i 1.0000 - 1.0000i
```

No caso de uma matriz complexa, podemos utilizar as funções **abs** e **angle** para extrair o módulo e a fase de cada elemento da matriz.

```
>> abs(A)

ans =

    1.4142    5.8310

    2.2361    1.4142

>> angle(A)

ans =

    0.7854    2.6012

   -2.6779   -0.7854
```

É muito comum desejarmos criar vetores que assumam uma progressão aritmética (sendo a mais comum: 1, 2, 3, 4, ...). Uma forma simples de criarmos vetores com essa característica é dada pelo seguinte comando:

```
>> B = 1:1:10

B =

      1     2     3     4     5     6     7     8     9     10
```

Como vimos, esse comando é formado por três elementos: o primeiro indica o início da série, o segundo representa o incremento, e o terceiro indica o valor máximo que pode ser assumido pelo último elemento do vetor.

É possível ter acesso às componentes (ou elementos) de uma matriz utilizando-se índices para realizar o endereçamento vetorial. Considere, por exemplo, o seguinte vetor:

```
>> X = pi*[0:0.1:0.5]
X =
         0    0.3142    0.6283    0.9425    1.2566    1.5708
```

Podemos acessar o primeiro elemento do vetor através do comando:

```
>> X(1)
ans =
     0
```

Podemos acessar o segundo elemento do vetor através do comando:

```
>> X(2)
ans =
    0.3142
```

Na verdade, podemos extrair os elementos do vetor na ordem que desejarmos. O seguinte comando extrai o primeiro, o quinto e o terceiro elementos do vetor definido:

```
>> X([1 5 3])
ans =
         0    1.2566    0.6283
```

Estude ainda os seguintes exemplos:

```
>> X(4:-1:2)
ans =
    0.9425    0.6283    0.3142

>> X(4:end)
ans =
    0.9425    1.2566    1.5708
```

Operações semelhantes são válidas para matrizes. No próximo exemplo, definimos uma matriz 3x3. É mostrado como acessar o elemento presente na segunda linha e terceira coluna, como acessar todos os elementos da segunda coluna e como acessar os elementos da terceira linha, respectivamente.

```
>> A = [1 2 3;4 5 6;7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9

>> A(2,3)
ans =
     6

>> A(:,2)
ans =
     2
     5
     8

>> A(3,:)
ans =
     7     8     9
```

2. Operadores

Os operadores básicos no *Matlab* são muito simples: adição (+), subtração (-), multiplicação (*), divisão (/) e potenciação (^).

É muito importante lembrar que os operadores no *Matlab* realizam operações sobre matrizes, ou seja, se multiplicarmos uma matriz "A" por uma matriz "B", a operação A*B resultará em uma matriz de multiplicação matricial. Dito desta forma soa bastante óbvio. No entanto é bastante comum cometermos o seguinte erro:

Inserimos um vetor "a" e desejamos multiplicar os elementos de "a" pelos elementos de "b" em uma operação elemento-a-elemento:

```
>> a = [1 2 3 4];

>> b = [0 1 1 0];
```

Se inserimos o comando:

```
>> a*b
```

o *Matlab* mostrará uma mensagem de erro, pois ele interpretará que estamos tentando multiplicar uma matriz 1x4 por outra matriz 1x4, o que não é possível!

Para indicar ao *Matlab* que estamos realizando uma operação elemento-a-elemento, devemos usar o operador ".". Assim, uma multiplicação ponto-a-ponto deve utilizar ".*":

```
>> a.*b

ans =

     0     2     3     0
```

Observe que o mesmo é válido para os operadores de DIVISÃO e POTENCIAÇÃO.

Os operadores lógicos no *Matlab* são: "e" (&), "ou" (|) e "não" (~).

Os operadores relacionais no *Matlab* são == (igual), > (maior), < (menor), ~= (diferente), >= (maior ou igual) e <= (menor ou igual).

Existe ainda o operador transposição, conforme mostra o seguinte exemplo:

```
>> a = 1:3
a =
     1     2     3

>> b = a'
b =
     1
     2
     3
```

Caso o operador transposição seja utilizado sobre uma matriz (ou vetor) que contenha elementos complexos, será retornado a matriz conjugada transposta. Para que seja retornada apenas a matriz transposta, deve-se utilizar o operador ponto juntamente com o operador transposição:

```
>> A = [1+2*j 3+6*j ; 10-j 5+9*j]
A =
   1.0000 + 2.0000i   3.0000 + 6.0000i
  10.0000 - 1.0000i   5.0000 + 9.0000i

>> A'
ans =
   1.0000 - 2.0000i  10.0000 + 1.0000i
   3.0000 - 6.0000i   5.0000 - 9.0000i

>> A.'
ans =
   1.0000 + 2.0000i  10.0000 - 1.0000i
   3.0000 + 6.0000i   5.0000 + 9.0000i
```

3. Funções Básicas

A seguir serão mostradas algumas funções importantes:

zeros(m,n) : cria uma matriz mxn preenchida com 0's.

```
>> zeros(1,3)

ans =

      0     0     0
```

ones(m,n) : cria uma matriz mxn preenchida com 1's.

```
>> ones(1,3)

ans =

      1     1     1
```

size(t) : exibe o tamanho da matriz t. A função **length(t) :** mostrará o comprimento do vetor.

```
>> A = randn(2,3)
A =
   -0.4326    0.1253   -1.1465
   -1.6656    0.2877    1.1909

>> size(A)
ans =
     2     3

>> length(A)
ans =
     3
```

cos(x), sin(x) : realiza as funções cosseno e seno respectivamente. É realizada sobre os elementos da matriz (em radianos).

```
>> x = [0 0 0];

>> cos(x)

ans =
```

```
      1     1     1

>> sin(x)

ans =

      0     0     0
```

exp(x) : realiza as função exponencial (e^x). É realizada sobre os elementos da matriz.

```
>> x = [1 2 3];

>> exp(x)

ans =

      2.7183            7.3891            20.0855
```

Funções igualmente importantes são **log**, **sqrt**, **tan**, **atan**, **atan2**, **min**, **max**, **find**, **sort**, **fliplr**, **repmat** e **reshape**. Para uma descrição das operações realizadas por cada uma dessas funções, basta digitar no prompt do *Matlab* o comando *help* seguido do nome da função de interesse.

fft(x), ifft(x) : realizam, respectivamente, a transformada discreta de Fourier e a transformada discreta inversa de Fourier sobre os elementos de um vetor x.

rand(m,n): cria uma matriz mxn com números aleatórios uniformemente distribuídos entre 0 e 1.

```
>> rand(1,3)

ans =

    0.9501        0.2311      0.6068
```

randn(m,n): cria uma matriz mxn com números aleatórios que seguem uma distribuição normal (gaussiana) de média nula e variância unitária.

wavwrite(y, Fs, N, 'filename'): função que permite escrever o conteúdo de um vetor y em um arquivo de som (no caso filename.wav). *Fs* é a taxa de amostragem do sinal (em amostras por segundo) e N permite definir o número de bits por amostra (8, 16, 24 ou 32 bits).

wavread('filename'): função que retorna o conteúdo de um arquivo de som (no caso filename.wav) no formato de um vetor. Suporta arquivos de 24 e 32 bits por amostra.

wavrecord(N,Fs): função que permite gravar N amostras de um sinal de áudio a uma taxa de Fs amostras por segundo.

4. Scripts e Funções Definidas pelo Usuário

Ao invés de digitar todos os comando no prompt do *Matlab*, longas seqüências de comandos podem ser armazenadas de arquivos de extensão ".m" denominados scripts. Dessa forma, pode-se economizar tempo (evitando a repetição de toda uma seqüência de comandos) e reaproveitar códigos já desenvolvidos.

O *Matlab* permite também que o usuário defina suas próprias funções. Para tal, deve-se criar um arquivo tipo ".m" cujo nome deve ser o mesmo da função a ser definida. Dentro do arquivo deve ser definida a função da seguinte forma:

```
function [saida1,saida2,…] = Nome_Funcao(parametro_a, parametro_b, ...)

comandos...
```

Por exemplo, podemos construir uma função que retorna a área e o perímetro de um quadrado, sendo fornecido o valor de comprimento de seu lado. Definimos em um arquivo denominado *ap_quad.m* os seguintes comandos:

```
function [area,perimetro]=ap_quad(lado)
% Função que retorna a área e o perímetro de um quadrado a
% partir da medida de seu lado.

area = lado.^2;
perimetro = 4*lado;
```

Note que "**%**" é utilizado para introduzir uma linha de comentário. Podemos então utilizar a função definida. No prompt de comando, podemos digitar:

```
>> l = 2;

>> [a,p] = ap_quad(l)

a =

     4

p =

     8
```

5. Controle de Fluxo

Os comandos elementares de controle de fluxo são: **if/else, for** e **while.**

A sintaxe do comando if/else é a seguinte.

```
if expressão

 comandos...

else

 comandos...

end
```

Caso haja mais de uma expressão a ser testada, podemos usar o elseif.

```
if expressão

 comandos1...

elseif expressão

 comandos2...

else

 comandos3...

end
```

A sintaxe do comando for é a seguinte.

```
for variavel_contador = 1:k

 comandos...

end
```

É possível também aninhar vários laços **for**.

A sintaxe do comando while é a seguinte.

```
while expressão
 comandos...
end
```

Existe também o comando **try/catch**, de sintaxe:

```
try
 comandos1...
catch
 comandos2...
end
```

O bloco *comandos1* é executado. Se um erro ocorrer, a execução é de *comandos1* é interrompida (sem interromper a rotina como um todo) e o bloco *comandos2* é executado.

Existem ainda os comandos **switch/case**, **continue**, **break** e **return**. Os detalhes quanto a sintaxe e utilização destes comandos pode ser encontrada na documentação do *Matlab*.

6. Recursos Gráficos

A seguir são apresentadas algumas funções de plotagem:

plot(a): realiza um gráfico cujo eixo X é representado pelo índice dos elementos de "a" e o eixo Y pelo valor dos elementos de "a". Os pontos são ligados entre si.

plot(a, b): realiza um gráfico dos pontos (a_n, b_n) dos vetores "a" e "b". Os pontos são ligados entre si.

O comando plot apresenta várias opções de entrada, como alterar a cor e o tipo de linha utilizada. Para maiores detalhes, utilize o comando ***help plot***.

stem(a) : idêntico a plot(a), mas os pontos são representados por uma barra vertical e não são ligados entre si.

stem(a, b) : idêntico a plot(a, b), mas os pontos são representados por uma barra vertical e não são ligados entre si.

Podemos também adicionar títulos e legendas aos nossos gráficos com as funções **title(), xlabel() e ylabel()**:

Considere, por exemplo, o caso de plotar o gráfico da função $f(x) = (x-1)(x-2)$. Podemos utilizar o seguinte código:

```
% Criamos o vetor que representa os valores de x
x = 0:0.05:4;
% Definição da função f(x)
f = (x-1).*(x-2);
plot(x,f)
xlabel('x')
ylabel('f(x)')
```

7. Exercícios

Observação: não é necessária a entrega de nenhum dos exercícios.

1) Considere as matrizes abaixo e, utilizando um script Matlab, faça as seguintes operações:

$$A = \begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 14 & 83 & 23 & 0 \end{vmatrix} \qquad B = \begin{vmatrix} 7.4 & 10 & 0 \\ -4.01 & 2 & 3 \\ 0.1 & 10 & 0 \end{vmatrix}$$

a) Defina as matrizes A e B;

b) Atribua o elemento a_{32} à variável *c1;*

c) Atribua o elemento b_{22} à variável *c2;*

d) Atribua os elementos a_{11}, a_{12} e a_{13} a um vetor *d1;*

e) Atribua a 3ª coluna da matriz *B* a um vetor *d2;*

f) Atribua a 4ª linha de *A* à 2ª linha de *A.*

g) É possível realizar a operação A + B ? Justifique.

2) Plote, em um mesmo gráfico, as funções $f(t) = \frac{\text{sen}(2t) \cdot \cos(t)}{2 + \text{sen}(t)}$ e $g(t) = 0.025t^2 + 0.03t - 0.5$. Utilize diferentes estilos de cor e de linha para cada um dos gráficos. Inclua também uma legenda para identificar cada uma das curvas. Para o intervalo de tempo, utilize uma resolução de 0.005s e considere $-7 \leq t \leq 6$. Não se esqueça de acrescentar um título e identificar os eixos do gráfico. Por meio da ferramenta *Data Cursor* (identificada pelo ícone ), determine as raízes da equação $f(t) - g(t) = 0$.

3) Gere uma senóide com freqüência de 50 Hz e amplitude 3 V, amostrada a uma taxa de amostragem de 500 Hz (500 amostras por segundo). Adicione a esta senóide um sinal aleatório cujos valores de amplitude seguem uma distribuição normal com variância de 2 V^2 e média nula. Plote o sinal resultante. Determine os valores máximo e mínimo deste sinal. Não se esqueça de incluir título e unidades. Procure plotar o sinal de forma que suas características fiquem claras. *Lembre-se ainda que se* X *é uma variável aleatória de variância* σ_X^2, *e* $Y = aX$, *onde* a *é uma constante, então a variância de* Y *será* $\sigma_Y^2 = a^2\sigma_X^2$.

4) Sinais discretos no tempo são representados por uma seqüência de números, formalmente escrito como $x = \{x[n]\}, -\infty < n < \infty, n \in Z$. Considere o caso de uma seqüência finita x de tamanho *N*, como, por exemplo, $x = \{x[0], x[1], \ldots, x[N-1]\}$ Neste caso, a potência média do sinal discreto é definida por:

$$P_x = \frac{1}{N}\sum_{k=0}^{N-1}|x[k]|^2 \qquad (1)$$

Escreva uma função em *Matlab* que receba como entrada um seqüência (vetor) e retorne como parâmetro de saída a potência média dessa seqüência. Gere também um script para testar a função elaborada.

5) Um sistema ***acumulador*** pode ser definido pela equação:

$$y[n] = \sum_{k=-\infty}^{n} x[k] \qquad (2)$$

Note que o n-ésimo ponto da saída y é dado pela soma de todas as amostras da entrada x até o ponto n. Escreva, em *Matlab*, uma função que receba como entrada o vetor x e retorne o vetor y. Escreva também um *script* que permita testar e validar a implantação de sua função.

6) Considere o sinal definido pela expressão:

$$x(t) = \begin{cases} t\,, 0 \le t \le 1 \\ 0\,, \text{caso contrário} \end{cases} \tag{3}$$

Este sinal será processado de duas formas distintas, conforme mostrado na Figura 1. No primeiro caso, é gerado um sinal $v(t) = x(-t)$, que é em seguida atrasado em duas unidades temporais, resultando no sinal $y(t) = v(t-2)$. Na segunda situação, é gerado um sinal $w(t) = x(t-2)$, que é depois refletido em torno da origem, resultando no sinal $z(t) = w(-t)$.

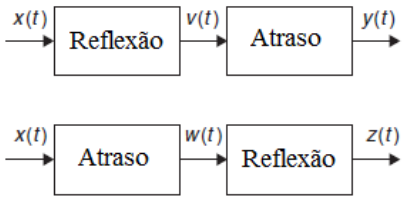

Figura 1

a) Obtenha expressões matemáticas para os sinais apresentados nos diferentes pontos dos diagramas de blocos;
b) Utilizando do *Matlab*, faça as operações descritas e gere o gráfico de cada um dos sinais considerados;
c) A partir dos resultados obtidos anteriormente, a reflexão em torno da origem temporal e o deslocamento no tempo são comutativas? Justifique.

7) Qualquer sinal $x(t)$ pode ser representado pela soma de uma componente par, $x_e(t)$, e uma componente ímpar, $x_o(t)$; isto é $x(t) = x_e(t) + x_o(t)$, em que:

$$x_e(t) = \frac{1}{2}\left[x(t) + x(-t)\right] \tag{4}$$

$$x_o(t) = \frac{1}{2}\left[x(t) - x(-t)\right] \tag{5}$$

Considere o sinal mostrado na Figura 2. Obtenha:

a) As componentes par e ímpar desse sinal;

b) Utilizando o *Matlab*, gere as componentes par e ímpar, e faça os gráficos do sinal e suas componentes. Comprove que $x(t) = x_e(t) + x_o(t)$;

c) Mostre que, se $x(t)$ é um sinal de energia, então é válida a relação $\int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |x_e(t)|^2 dt + \int_{-\infty}^{\infty} |x_o(t)|^2 dt$.

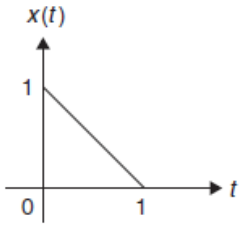

Figura 2

8) Considere o sinal definido pela expressão:

$$x(t) = \begin{cases} te^{-2t}, 0 \le t \le 4 \\ 0, \text{caso contrário} \end{cases} \quad (6)$$

Utilizando o *Matlab*, defina os novos sinais $y(t) = x(2t)$ e $z(t) = x(t/2)$. Faça o gráfico de $x(t)$, $y(t)$ e $z(t)$. Qual deles é uma versão expandida (no tempo) de $x(t)$? E qual é uma versão comprimida (no tempo) de $x(t)$? Justifique seus resultados.

8. Referências Bibliográficas

HANSELMAN, D. LITTLEFIELD, B. *Matlab 6 – Curso Completo*. Prentice Hall.

Material adaptado de apostilas do prof. João Leite, ENE/UnB - 2012.