Aula 01 - Familiarização com o Matlab®

1 Introdução

O Matlab é um programa de simulação numérica muito útil no estudo de problemas e no desenvolvimento de projetos em Engenharia, sendo utilizado em universidades e empresas ao redor do mundo.

O principal motivo desse sucesso é a utilização de vetores e matrizes para representar e manipular dados de uma forma simples (Matlab = *Matrix Laboratory*). Assim, o Matlab é uma excelente plataforma para explorar de forma aplicada os conhecimentos adquiridos nos cursos de Álgebra Linear.

O objetivo desta aula é um primeiro contato com o Matlab por meio de alguns exemplos básicos. Veremos como definir e operar com vetores e matrizes, gráficos, *scripts* e funções.

Nas primeiras seções, vamos trabalhar diretamente na janela de comando (*command window*), em que você pode digitar diretamente os comandos após o *prompt* `>>`. Depois, vamos aprender a escrever nossos *scripts* e funções no editor do Matlab.

Você pode adquirir uma versão para estudante do Matlab em `http://www.mathworks.com/`. Alternativamente, quase todas as atividades a serem desenvolvidas ao longo do curso podem ser realizadas usando os programa gratuitos Octave (`https://www.gnu.org/software/octave/`) ou Scilab (`http://www.scilab.org/`) . Apesar das interfaces de ambos não serem tão amigável quanto no Matlab, os comandos básicos são muito parecidos. Você é convidado a testá-los e conversar com os professores em caso de dúvidas.

2 Criando Matrizes

Matrizes são os elementos básicos sobre os quais opera-se no Matlab. A forma mais direta de se criar uma matriz no Matlab é digitar os números entre colchetes, usando espaço ou vírgula para separar diferentes entradas. Use um ponto e vírgula ou um `[Enter]` para indicar o fim de uma linha.

Digite na linha de comando os exemplos a seguir:

```
>> A = [1 2; 3 4; 5 -6] [Enter]
A =
    1  2
    3  4
    5 -6

>> B = [1 -2 3      [Enter]
      4 5 -6]       [Enter]
B =
    1  -2   3
    4   5  -6

>> x = [4; 3;2]      [Enter]
x = 4
    3
    2
```

```
>> X = [4, 3,2]      [Enter]
X =
     4 3 2
```

Para ver uma matriz criada, basta digitar seu nome seguido por `[Enter]`. Note que o Matlab diferencia maiúsculas e minúsculas, assim `X` é diferente de `x`.

Exercício 1: Faça o Matlab mostrar na tela as 4 matrizes do exemplo anterior. Note a forma como elas são apresentadas.

Exercício 2: Crie as seguintes matrizes no Matlab. Escreva no espaço ao lado os comandos que você utilizou para criá-las.

a) $C = \begin{bmatrix} -4 & 8 & -1 \\ 5 & 0 & 3 \\ 6 & 2 & 10 \end{bmatrix}$

b) $D = \begin{bmatrix} 2 & -1 \\ 0.1 & 3 \\ -2 & 1 \end{bmatrix}$

c) $\mathbf{vec} = \begin{bmatrix} 3 \\ -5 \\ 1 \end{bmatrix}$

Quando M é uma matriz, o comando `size(M)` retorna um vetor com duas entradas que são o número de linhas e colunas de M. Por exemplo,

```
>> size(A)     [Enter]
ans =
      3   2
```

retorna as dimensões da matriz A do exemplo anterior.

Observações

i. Para que o Matlab execute mas não apresente na tela o resultado de um comando, basta acrescentar um `;` ao seu final.

ii. `ans` é o nome de uma variável à qual o Matlab atribui automaticamente a saída de uma função, caso não seja atribuído uma variável a ela no comando que a invocou. Veja o exemplo anterior com o `size`.

iii. Se você ficar na dúvida sobre a utilização de uma função `<X>` do Matlab, usar `help <X>`, pode lhe ajudar. Tente digitar `help size`, por exemplo.

iv. Para visualizar as variáveis definidas num dado instante no espaço de trabalho, use os comandos `who` ou `whos`. Para limpar todas as variáveis da memória, use `clear all`. Para limpar a *command window* use `clc`.

3 Acessando elementos e concatenando matrizes

Para ver o conteúdo de uma matriz que foi armazenada, basta digitar o seu nome na linha de comando. Você pode ver ou modificar o valor armazenado em uma determinada posição de uma matriz `C` digitando `C(#linha,#coluna)` sendo `#linha` e `#coluna` os índices das linhas e colunas desejadas. No Matlab, os índices sempre são inteiros começando de 1.

Exercício 3: Considerando a matriz `C` do Exercício 2, digite os seguintes comandos, anote as saídas e as justifique.

a)
```
>> C, C(3,1)
>> C(3,1)=-9
```

b)
```
>> C, C(:,2)
>> C(:,2)=[1;1;0]
```

c)
```
>> C, C([1 3],[2 3])
>> C([1 3],[2 3]) = [-2 4; 6 7]
```

d)
```
>> C, C([1 3],:)
>> C([1 3],:) = C([3 1],:)
```

e)
```
>> C, C(3,:)
>> C(3,:) = [0 1 2]
```

f)
```
>> vetor = C(:)
>> C'
```

g) `>> ultimoelementolinha3 = C(3,end)`

Note que o `:` em `C(:,3)` significa "todas as linhas". No caso, estamos selecionando os elementos que estão em todas as linhas e terceira coluna da matriz `C`. Já `C(:)` faz com que o Matlab crie um vetor coluna com todos os elementos da matriz `C`. O `C'` ou `transpose(C)` é usado para transpor a matriz `C`.

O Matlab permite concatenar ou juntar matrizes de tamanhos consistentes em uma nova matriz maior. Veja os exemplos no Exercício 4 a seguir.

Exercício 4: Considerando as matrizes criadas na Seção 2, digite os seguintes comandos, anote as saídas e as justifique.

a) `>> [C D]`

b) `>> [D C]`

c) `>> [C;B]`

d) `>> [B;C]`

e) `>> [B C]`

---

4 Algumas funções básicas para trabalhar com matrizes

Os comandos `eye`, `zeros`, `ones`, `diag` são utilizados para criarmos rapidamente algumas matrizes que aparecem muito frequentemente. Seu uso pode ser facilmente entendido por meio de exemplos.

Exercício 5: Considerando as matrizes criadas no Exercício 2, digite os seguintes comandos, anote as saídas e as justifique.

a) `>> eye(3)`

b) `>> zeros(3)`

c) `>> ones(size(D))`

d) `>> diag([4 5 6 7])`

e) `>> diag([ones(1,5) zeros(1,3) ones(1,2)])`

---

5 Criando vetores

Um vetor é uma matriz especial constituída por uma única linha ou uma única coluna. Existem alguns comandos especialmente úteis para se criar vetores.

Uma primeira forma de se criar longos vetores rapidamente é por meio do operador `:`. O formato geral é:

```
Vetor = valor inicial: passo: valor final
```

Quando o passo é unitário, ele pode ser omitido.

Exemplos de utilização:

a) Gerar um vetor `x` com os números inteiros de zero a cinco.

```
>> x = 0:5
x =
0     1     2     3     4     5
```

b) Gerar um vetor `y` indo de 0 a 1 com passo de 0.1.

```
>> y = 0:0.1:1
y =
         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000
0.7000 0.8000    0.9000    1.0000
```

c) Gere um vetor contendo os números inteiros entre zero e 10 em ordem crescente seguidos pelos mesmos em ordem decrescente.

```
>> x = [0:10 10:-1:0]
x =
0     1     2     3     4     5     6     7     8     9    10    10
9     8     7     6     5     4     3     2     1     0
```

Exercício 6: Gerar um vetor `x` de números pares decrescentes de 48 a -2.

Exercício 7: Construa um vetor constituído pelos números pares de 0 a 10 seguido pelos números ímpares de 0 a 10.

A função linspace é uma forma prática de se gerar vetores quando sabemos quantos pontos ele deve ter. Ela tem o seguinte formato:

```
Vetor = linspace(valor inicial, valor final, número de pontos)
```

Exemplos de utilização:

a) Gere um vetor de 1000 pontos com valores entre zero e 1 igualmente espaçados.

```
>> v = linspace(0,1,1000);
```

b) Repita o exemplo a), mas com os valores em ordem decrescente.

```
>> v = linspace(1,0,1000);
```

Exercício 8: Gere um vetor x de 5 000 pontos com valores entre `-pi` e `pi`.

---

6 Aritmética com matrizes e vetores

O Matlab permite somar (`+`), subtrair (`-`), multiplicar (`*`) e dividir (`/`) matrizes de dimensões apropriadas. Além disso, são bastante úteis as operações de multiplicação e divisão "ponto a ponto", obtidas utilizando-se os operadores `.*` e `./`, respectivamente.

Quase todas as funções (trigonométricas, exponenciais e outras) podem ser aplicadas a uma matriz ou a um vetor sendo que elas operam elemento a elemento.

Exemplos de aplicação:

a) Sendo `x = [2 3 7]` e `y = [0 -1 3]` escreva a resposta de cada um desses comandos executados no Matlab.

i. `>> x+y` `[2 2 10]`

ii. `>> x-y` `[2 4 4]`

iii. `>> x.*y` `[0 -3 21]`

b) Como gerar um vetor de números igualmente espaçados entre 1 e 11 a partir do vetor `x = 0:0.001:1`?

```
>> v = 10*x+1
```

7 Gráficos

Outra característica interessante do Matlab para um Engenheiro é a facilidade de se construir gráficos com ele de uma maneira muito simples. Dois comandos muito utilizados são:

```
plot(vetor.abscissa, vetor.ordenada, 'modo');
    stem(vetor.abscissa, vetor.ordenada);
```

O comando `plot` traça um gráfico constituído pelos pontos cujas abscissas estão em `vetor.asbscissa` e ordenadas estão em `vetor.ordenada`. Claramente esses vetores precisam ter mesmo comprimento. A *string* `modo` indica a forma (cor, formato da linha ligando os pontos, etc.) como o gráfico será traçado. Veja os exemplos e use `help plot` para mais detalhes. A função `stem` traça um gráfico da sequência em seu segundo argumento como *palitos* com círculos no valor dos dados usando seu primeiro argumento como abscissa. Veja os exemplos a seguir.

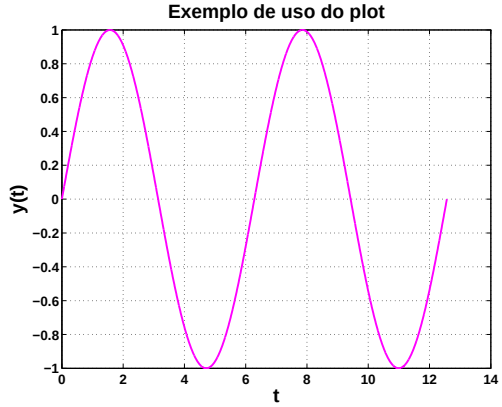
Seguem-se alguns comandos úteis quando se trabalha com gráficos. Consulte o `help` para saber em detalhes como eles funcionam.

- `grid` - coloca linhas de grade no gráfico.
- `title` - permite acrescentar um título ao gráfico.
- `xlabel` - permite acrescentar um título no eixo das abscissas.
- `ylabel` - permite acrescentar um título no eixo das ordenadas.
- `hold` - não apaga o gráfico atual antes de fazer o seguinte.
- `axis` - controla a escala e aparência dos eixos.

Exemplos de aplicação:

a) Faça um gráfico da função $y(t) = \sin(t)$ para $t \in [0, 4\pi]$.

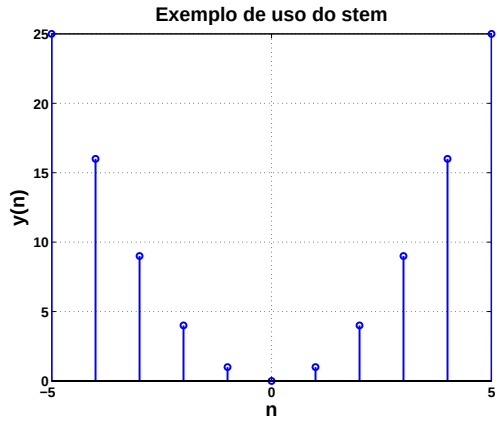
```
>> t = linspace(0,4*pi,5000);
>> y = sin(t);
>> plot(t,y,'m-'); % 'm-' = pontos ligados por linha contínua magenta
>> xlabel('t');
>> ylabel('y(t)');
>> grid;
>> title('Exemplo de uso do plot')
```



Note que ao usar o `plot` com o modo `-` ou sem especificar um formato de linha em especial, o Matlab liga os pontos obtidos tendo-se a impressão de uma curva contínua. Porém, na realidade, temos sempre um conjunto finito de pontos!

b) Faça um gráfico da função $y(n) = n^2$ para $n \in \mathbb{Z}, -5 \leq n \leq 5$

```
>> n = -5:5;
>> y = n.^2;
>> stem(n,y)
>> xlabel('n');
>> ylabel('y(n)');
>> grid;
>> title('Exemplo de uso do stem')
```



Exercício 9: Faça gráficos de $y(t) = \sin^2\left(\frac{\pi}{12}t\right)$ e $z(t) = \cos^2\left(\frac{\pi}{12}t\right)$ para $-30 \leq t \leq 30$ na mesma figura. O gráfico de $y(t)$ deverá ficar em ciano e o de $z(t)$ em preto.

---

8 *Scripts* e Funções

Até este ponto, todas as nossas interações com o Matlab foram por meio da linha de comando. Digitamos comandos ou funções na linha de comando e o Matlab interpreta nossa entrada e toma a ação apropriada. Isso é razoável quando nossa sessão de trabalho é curta e não repetitiva. No entanto, essa forma de trabalhar é inviável para executar uma longa sequência de comandos. Fica muito difícil detectar e corrigir erros. Além do mais, muitas vezes é interessante armazenar a sequência de comandos para utilizarmos novamente quando quisermos.

Um *script* é uma sequência de comandos e funções comuns, como as usadas na linha de comando. São arquivos-textos e podem ser criados usando qualquer editor de texto. Um *script* é invocado na linha de comando digitando-se o nome do arquivo. Aí o Matlab executa os comandos e funções no arquivo como se eles tivessem sido digitados diretamente na linha de comando.

Outras informações relevantes sobre os *scripts*:

- *scripts* podem invocar outros *scripts*.
- os arquivos de *scripts* têm a terminação `.m`. Por isso, os *scripts* são chamado de *arquivos-m*, assim como as funções que veremos mais adiante;
- os *scripts* operam sobre as variáveis do espaço de trabalho;

- os nomes de variáveis Matlab podem ter qualquer comprimento. Nos seus *scripts* use nomes de variáveis que tenham significado no contexto do programa, o que facilita muito a posterior correção de erros.

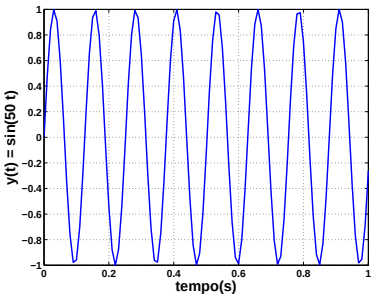
Suponha, por exemplo, que desejemos fazer um gráfico da função $y(t) = \sin at$ em que a é um parâmetro que queremos variar. Usando o editor de texto do Matlab (para abrí-lo basta ditar `edit` na linha de comando), podemos escrever um script chamado `plotdata.m` como mostrado a seguir.

```
% Este é um script para fazer um grafico da função y = sin(a*t)
% O valor de a precisa existir no espaco de trabalho antes
% de se chamar este script
t = 0:0.01:1;
y = sin(a*t);
plot(t,y);
xlabel ('tempo(s)');
ylabel(['y(t) = sin(' num2str(a) ' t)']);
grid on;
```

É importante salvar o *script* no mesmo diretório em que se está trabalhando na linha de comando. Caso contrário, ao tentar executar o *script* o Matlab não encontrará o arquivo e exibirá uma mensagem de erro. Esse erro é muito comum quando estamos começando a trabalhar com *scripts*.

Uma vez digitado e salvo é muito fácil utilizar o *script*. Veja os exemplos a seguir:

```
>> a = 50;
>> plotdata
```

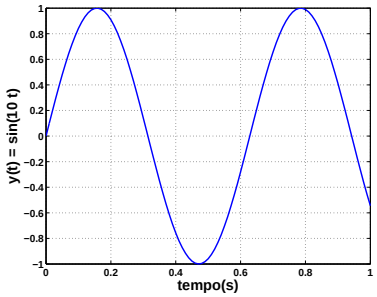



```
>> a = 10;
>> plotdata
```

Alternativamente, você poderia incluir a linha `a=10` diretamente no *script* e alterá-la toda vez que quiser variar o valor de `a`.

Ao escrever *scripts* é sempre interessante utilizar comentários, linhas que começam com `%`. Se você escrever linhas de comentário antes do começo das instruções do *script*, ao utilizar o comando `help nomearq` o Matlab apresenta essas linhas na tela. Por exemplo,

```
>> help plotdata
  Este é um script para fazer um grafico da função y = sin(a*t)
  O valor de a precisa existir no espaco de trabalho antes
  de se chamar este script
```



Assim como os *scripts*, as funções definidas pelo usuário estão entre os recursos mais importantes e utilizados do Matlab. Uma função é um *script* que recebe um ou mais argumentos e pode devolver um ou mais argumentos.

O formato de uma função no Matlab é o seguinte:

```
function [outarg1, outarg2,...] = fname(inarg1, inarg2,...)
% Um comentário
% Mais um comentário
....
(código executável)
....
```

A primeira linha, iniciada com `function` é o que diferencia uma função de um *script*. Nessa estrutura, `fname` é o nome da função criada e deve ser o nome do arquivo-m em que foi salva a função. `inarg1`, `inarg2`,... são os argumentos de entrada e `outarg1`, `outarg2`,... são os argumentos de saída.

Importante: diferentemente de um *script*, uma função não opera sobre as variáveis que estão criadas no espaço de trabalho e nem cria novas variáveis lá. Ela só opera sobre os argumentos de entrada e somente os argumentos de saída são gravados no espaço de trabalho e são acessíveis.

Como um exemplo muito simples, vamos criar uma função que recebe como argumento duas matrizes A e B e retorna a soma delas.

```
function R = somateste(A,B);
%Funcao para somar duas matrizes
R = A+B;
```

Uma vez que você tenha salvado este arquivo como `somateste.m` no diretório corrente, você pode usá-lo como nos exemplos a seguir:

```
>> somateste(2, 4)
ans =
     6
>> matriz1 =[1 2; 3 4];
>> matriz2 = [0 1; 3 3];
>> res = somateste(matriz1,matriz2)
res =

     1     3
     6     7
```

Exercício 10: Algo que deve-se ter à mente ao se iniciar o uso do Matlab é que a representação matricial praticamente elimina a necessidade de utilização de laços `for` ou `while` simplificando e acelerando muito os programas. O Matlab trabalha de forma otimizada com essas representações. Ou seja, evitar um laço em um programa Matlab pode representar uma economia muito grande de tempo de simulação. Em outras palavras, em Matlab, sempre que possível (ou seja, quase sempre!) não utilize laços `for` ou `while`!

Para ilustrar a situação, o *script* a seguir traça o gráfico de $y(t) = t^2$, $0 \leq t \leq 1$ de duas formas diferentes: uma explorando o uso do operador ponto-a-ponto `.^` e outra usando um laço `for`. Execute o *script* e comente sobre a diferença de tempo de execução das duas formas.

```
clear all;
clc;
tic %Zera cronômetro
t = linspace(0,1,1e7);
y = x.^2;
figure(1); plot(t,y);
disp('Tempo sem for');
toc %Mostra tempo gasto

clear all;
tic %Zera cronômetro
tempo = linspace(0,1,1e7);
for indice = 1:length(tempo),
    t(indice) = tempo(indice);
    y(indice) = tempo(indice)^2;
end
figure(2); plot(t,y);
disp('Tempo com for');
toc %Mostra tempo gasto
```

Exercício 11: Gere um vetor `x` de 1 000 valores aleatórios com distribuição uniforme no intervalo $[0, 1]$. Como você verá em detalhes no curso de Probabilidades, numa distribuição uniforme no intervalo $[0, 1]$ você tem probabilidades iguais de sortear números que pertençam a intervalos de mesmo comprimento. Assim, por exemplo, as probabilidades de um valor sorteado estar nos intervalo $[0, 0.5]$ ou $[0.5, 1]$ são iguais.

Dica: use a função `rand` (não sabe como usar? Use o `help`!).

Exercício 12: Considerando o vetor `x` gerado no Exercício 11, escreva um *script* que:

a) calcule a média dos valores no vetor `x`

b) calcule o mínimo e o máximo entre os valores de `x`

c) faça um gráfico dos valores de `x`, destacando os pontos de mínimo e máximo (você pode colocar estrelas nesses pontos, por exemplo).

Dica: Como vimos no Exercício 10, no "jeito Matlab de programar" você deve evitar ao máximo o uso de laços (`for`, `while`, etc.) nos programas. Será que existem funções que calculem máximo, mínimo e média de vetores? Pesquise...

Exercício 13: Escreva uma sequência de comandos do Matlab que forneça um vetor contendo 100 valores aleatórios uniformemente distribuídos no intervalo -1 a 1 e que faça um gráfico desse sinal.

Exercício 14: Escreva uma sequência de comandos Matlab que gere um gráfico do sinal $x(n) = \cos\left(\frac{\pi}{8}n\right) + 0.2r(n)$, sendo $r(n)$ um vetor de números aleatórios com distribuição uniforme entre -1 e 1. Faça $n \in \mathbb{Z}/0 \leq n \leq 99$.

Exercício 15: Escreva uma função Matlab chamada `pulso2graf` cujas entradas sejam dois números inteiros a e b com $a < b$. A função deverá fazer o gráfico de um pulso com amplitude 2 no intervalo $a \leq n \leq b$, considerando apenas valores de n inteiros. O gráfico deve começar em $a - 2$ e terminar em $b + 2$. Por exemplo, ao digitarmos:

`>> pulso2graf(2,8);` devemos obter a figura