



*Universidade Estadual de Campinas
Faculdade de Engenharia Mecânica
Departamento de Projeto Mecânico
Laboratório de Dinâmica de Estruturas e Máquinas*



Guia Prático de Matlab

Capítulo I

Noções Básicas

Docente Responsável:

Prof. Dr. Milton Dias Jr.

Autores:

Hugo Heidy Miyasato

Leonardo Gimenes

Vinicius Gabriel Segala Simionatto

Campinas, 2009

Seção 1.1 – Reconhecendo o programa

Sobre o Matlab

O Matlab é um programa desenvolvido pela MathWorks, uma empresa Norte-Americana. Basicamente, ele é um ambiente desenvolvido para computação numérica. Consequentemente, este ambiente suporta manipulação de matrizes e vetores em geral, geração de gráficos de funções e dados, tanto bi quanto tridimensionais, implementação de algoritmos e rotinas através de sua linguagem de programação, criação de interfaces gráficas (GUIs), e comunicação com programas desenvolvidos em outras linguagens.

Apesar de ser um ambiente para computação numérica, as versões mais recentes do Matlab também suportam computação simbólica (através do toolbox MuPAD) e modelagem gráfica em múltiplos domínios, através da ferramenta Simulink.

História

O Matlab foi desenvolvido em 1970, pelo então chairman do Departamento de Ciência da Computação da Universidade do Novo México, Cleve Moler. O nome dado ao programa é uma contração de “matrix laboratory”, e o intuito inicial de Cleve era que seus alunos pudessem usar os pacotes matemáticos LINPACK e EISPACK, ambos desenvolvidos em Fortran, sem ter que aprender esta linguagem de programação.

Com a boa aceitação do programa, e a sua divulgação em eventos, Jack Little, um engenheiro da área de desenvolvimento de controle, se uniu a Cleve, e depois de reescreverem o programa em C, a MathWorks decidiu continuar o desenvolvimento do programa.

A partir de 2000, o Matlab faz uso do pacote LAPACK para a manipulação de matrizes e vetores. Sua primeira aplicação na engenharia foi na área de controle, por causa de Jack Little, mas logo, o programa passou a englobar muitos outros domínios, e também a ser usado no ensino de álgebra linear, análises numéricas e processamento de imagens e outros sinais digitais.

- **O ambiente**

Ao abrir o programa, você perceberá que a tela está dividida em três partes, duas ao lado esquerdo, e uma ao lado direito. No canto superior esquerdo, você encontrará uma janela chamada “Current Directory”. Esta é a pasta na qual você estará trabalhando, e quando você tiver que salvar ou abrir um arquivo, é nesta pasta que ele deve estar.

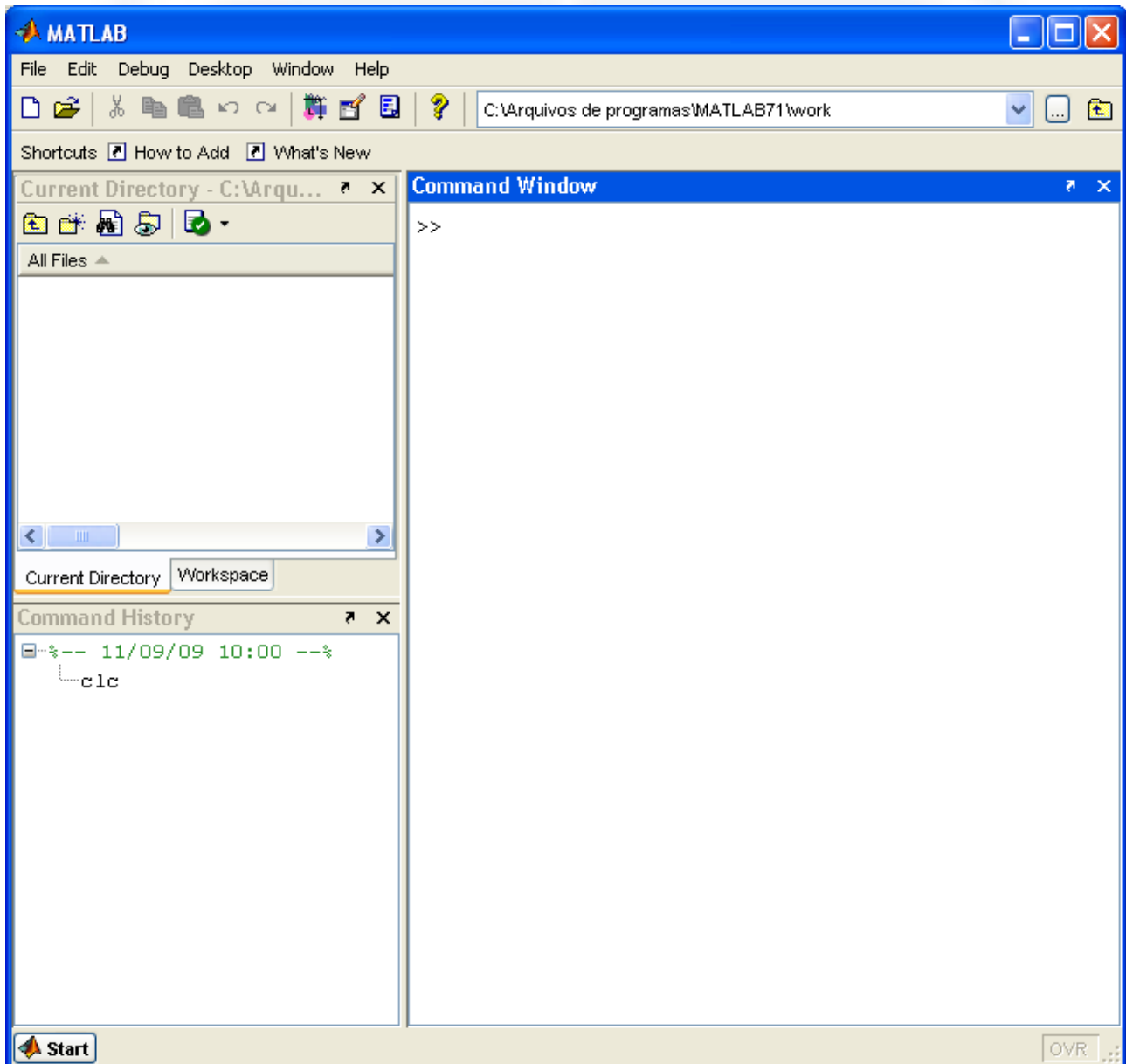


Figura 1: Janela inicial do Matlab

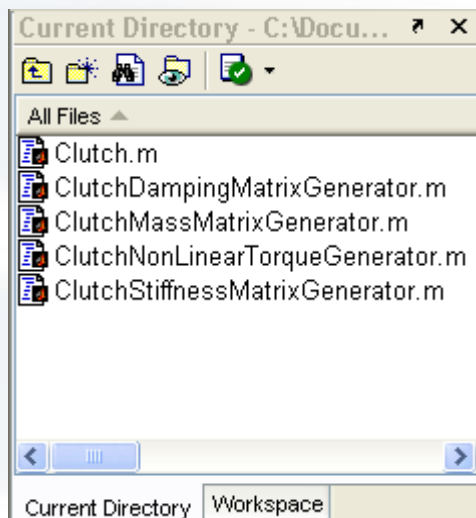


Figura 2: Janela "Current Directory"

Para mudar a pasta de trabalho, você deve procurar na barra de ferramentas superior, um menu cujo nome também é "Current Directory". Ao lado dele, ao apertar um botão com o símbolo "...", você terá acesso a uma janela, onde poderá configurar a sua pasta de trabalho. Esta é a configuração mais importante a se fazer antes de iniciar seu trabalho.

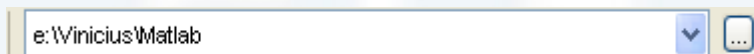


Figura 3: Mecanismo de configuração do ambiente de trabalho.

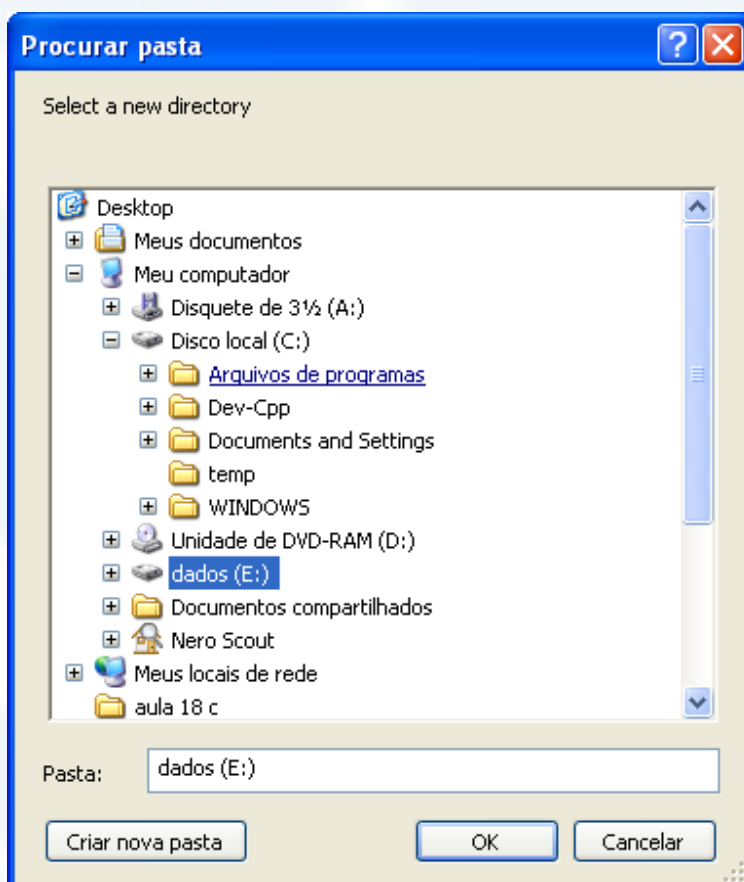


Figura 4: Janela para configurar pasta de trabalho.

Lembre-se: A boa organização com uso de pastas e padrões de nomes para seus arquivos irá facilitar seu trabalho como desenvolvedor.

Ainda do lado esquerdo da tela, entre a janela superior e a inferior, você encontrará abas com os nomes “Current Directory” e “Workspace”. A primeira deve estar selecionada se na janela superior esquerda você puder ver seus arquivos.

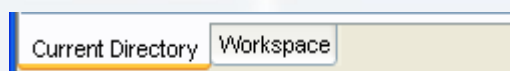


Figura 5: Abas “Current Directory” e “Workspace”

Ao clicar na aba “Workspace”, esta janela passará a mostrar todas as variáveis que estão sendo utilizadas no ambiente de trabalho. Estas variáveis não estão armazenadas em disco, mas sim na memória RAM, e podem ser acessadas a qualquer momento através do prompt de comando. Por padrão, a janela “Workspace” irá apresentar o nome das variáveis, seu valor, e sua classe, ou tipo. Ao clicar no título de uma destas colunas, você poderá exibir também nesta janela o tamanho da variável (por exemplo número de linhas e colunas no caso de um matriz), e o tamanho destas variáveis na memória RAM em bytes.

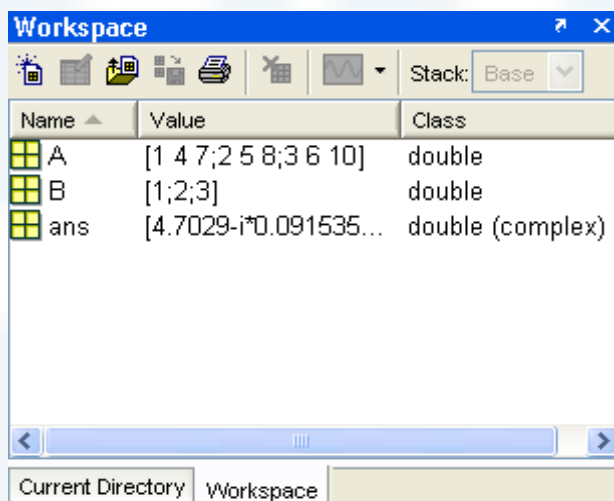


Figura 6: Janela “Workspace”

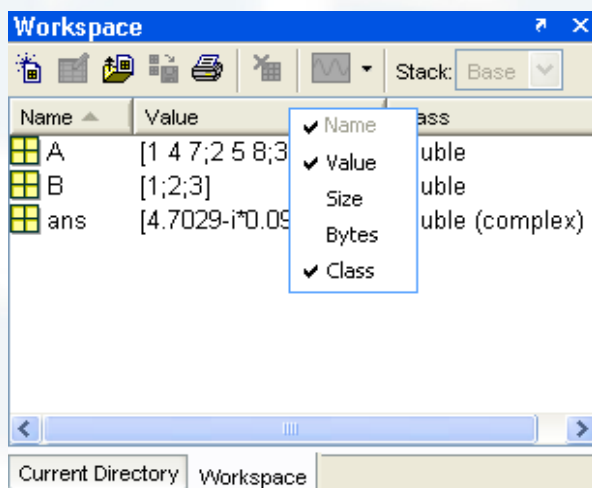


Figura 7: Mostrando mais informações sobre as variáveis.

Todas as operações no Matlab retornam algum valor, e este valor sempre deve ser salvo em uma variável. Caso você faça uma operação e decida não salvar seu resultado em nenhuma variável, o Matlab, por padrão, salvará este resultado em uma variável chamada “ans”, quer ela exista ou não. No canto inferior esquerdo da tela, há uma janela chamada “Command History”. Como diz o nome, ela é um histórico de todos os comandos digitados no prompt, por ordem cronológica. Isto pode ser muito útil quando se testa um novo código.

À direita da tela, você irá encontrar o prompt de comando, ou “Command Prompt” do programa que é o lugar onde serão digitados e executados os comandos desejados pelo usuário. Sempre que o Prompt de comando exibir o símbolo “>>” à esquerda, significa que o Matlab está pronto e aguardando novos comandos.



Figura 8: “Command Window”

Faça o teste:

Primeiro, faça com que a janela superior esquerda seja a “Workspace”. Então, no prompt de comando, digite um número qualquer e aperte Enter. Veja o que acontece:

```
>> 27  
  
ans =  
  
    27  
  
>>
```

Digitar um número no prompt do Matlab é uma operação que retorna o valor do número digitado. Como decidimos não guardar este valor em nenhuma variável, este valor foi guardado em ans. Nos próximos capítulos, você irá aprender como guardar seu resultado em uma variável desejada. Observe que, na janela “Workspace” surgiu a variável ans, e também está registrado lá seu valor e seu tipo, ou classe. Enquanto isso, o último comando registrado em “Command History” é o comando que você deu.

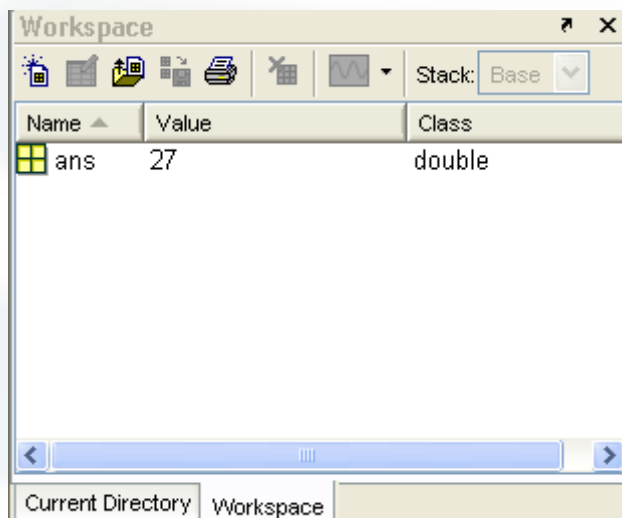


Figura 9: Variável “ans” no “Workspace”



Figura 10: Variável “ans” no “Command Prompt”

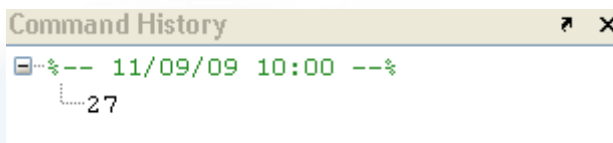


Figura 11: Comandos executados no “Command History”

Agora, dê um duplo clique na variável “ans”, na janela “Workspace”. Uma nova janela aparecerá, com uma tabela, onde, se quiser, você poderá editar esta variável. Esta janela se chama “Array Editor”

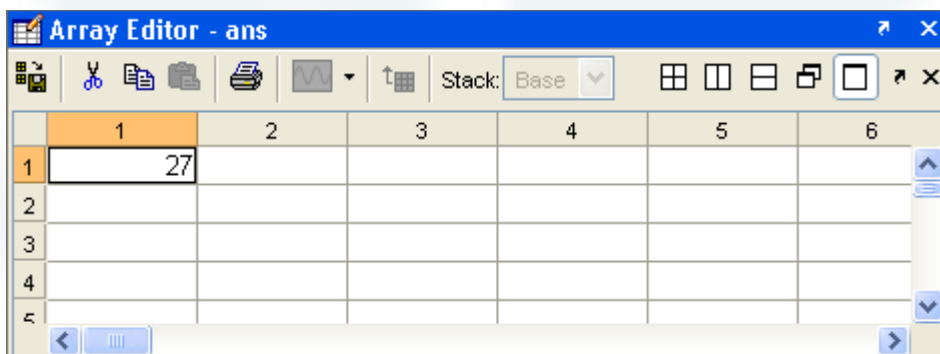


Figura 12: “Array Editor”

Se você der um duplo clique no comando registrado na “Command History”, ele será executado novamente.

IMPORTANTE: Ao terminar de digitar um comando no “Command Prompt”, você deve sempre apertar Enter para que este comando seja executado.

Outra maneira de se recuperar um comando antigo, é apertando para cima e para baixo (“↑”, “↓”), enquanto o seu cursor estiver no “Command Prompt”. Você pode também digitar parte do início do comando desejado e apertar para cima. O prompt então buscará apenas os comandos que iniciam com a mesma sequência de caracteres.

Faça o teste:

Digite um número maior do que 100 e menor do que 199 no seu Matlab, e em seguida aperte Enter. O número ficará guardado em ans, como visto anteriormente:

```
>> 132

ans =

    132

>>
```

Agora, digite “1” no seu prompt e aperte para cima. O número que você digitou anteriormente aparece novamente. Se você quer executar este comando de novo, é só apertar Enter.

A disposição das janelas neste programa também pode ser alterada para uma que mais lhe agrade. Para isso, você pode simplesmente, como em ambiente Windows, arrastar as janelas pelo título e acompanhar pelo contorno cinza desenhado na tela, onde ela permanecerá, depois que o botão do mouse for solto. Você poderá também fechar as janelas que achar desnecessárias, ou ainda separar algumas janelas do Matlab com o recurso “dock / undock”. Para fechar as janelas, clique no “x” que está no canto superior direito da janela. Para separar a janela do Matlab, clique em na seta curva para cima, e para fazer com que ela retorne ao programa, clique em na seta curva para baixo.



Figura 13: Setas para dar “Dock” e “Undock” em janelas

As janelas que foram fechadas podem ser reabertas através do menu “Desktop”, na aba de menus superior do Matlab. Além disso, no submenu “Desktop Layout” você encontrará alguns padrões de organização destas janelas.

- ***Help***

De maneira exemplar, tudo o que existe no Matlab foi muito bem documentado. Além disso, sempre há duas maneiras de se acessar a ajuda do programa: ou pelo prompt de comando, onde serão exibidas apenas informações escritas, ou pelos “docs”, onde você poderá ver figuras, informações e exemplos sobre tudo o que deseja fazer.

Pelo prompt de comando basta digitar “help” e o que você deseja obter ajuda sobre. Siga o exemplo a seguir:


```
>> help abs
ABS      Absolute value.
        ABS(X) is the absolute value of the elements of X. When
        X is complex, ABS(X) is the complex modulus (magnitude) of
        the elements of X.

        See also sign, angle, unwrap, hypot.

        Overloaded functions or methods (ones with the same name in other
        directories)
            help frd/abs.m
            help iddata/abs.m
            help sym/abs.m

        Reference page in Help browser
            doc abs

>>
```

Para acessar o help na forma de “docs”, basta apertar “F1”. Então, você descobrirá esta janela:

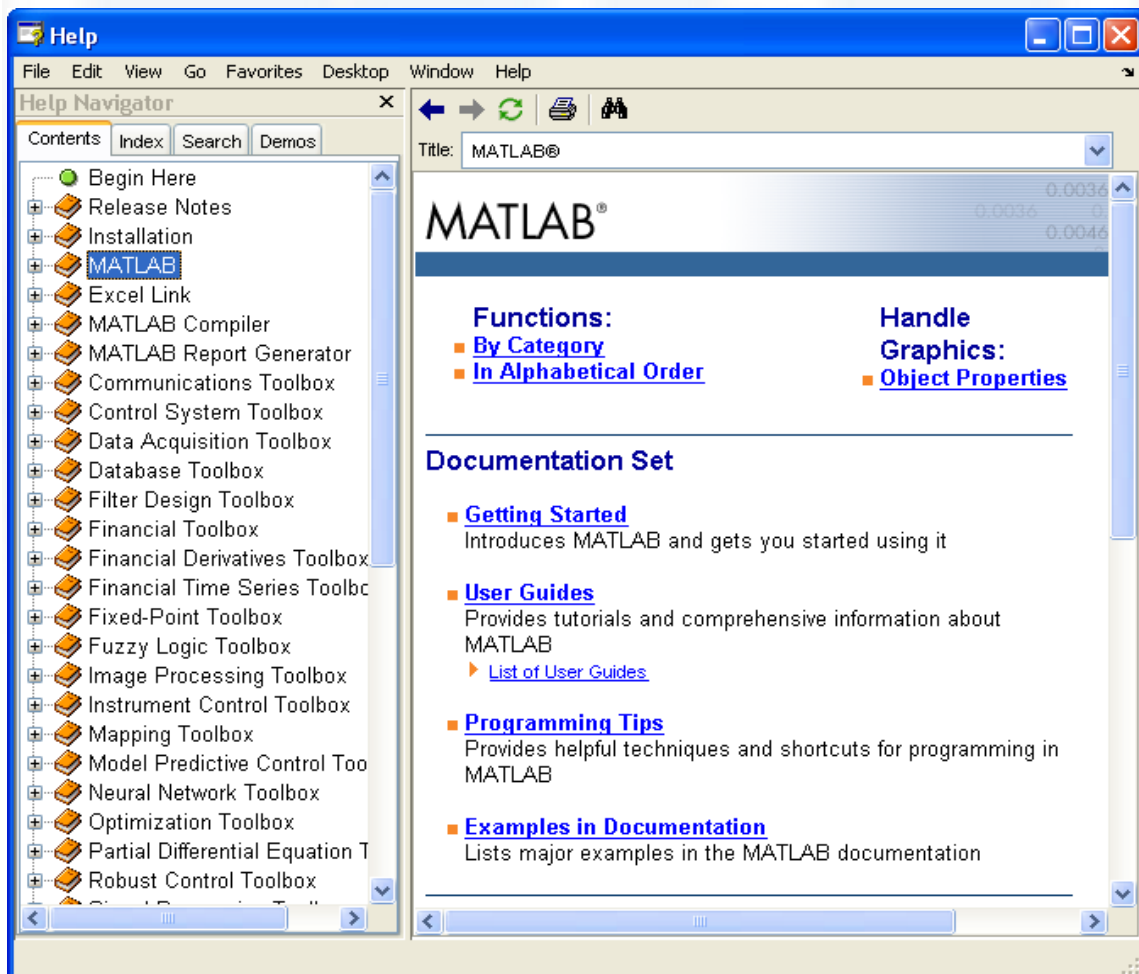


Figura 14: “Help” do Matlab

Nela, basta clicar na aba “Search”, digitar o que procura e apertar enter. Ao fazer isto, será exibida a você uma lista com todos os resultados encontrados para a sua busca, de forma que você poderá selecionar qual “doc” deseja ver.

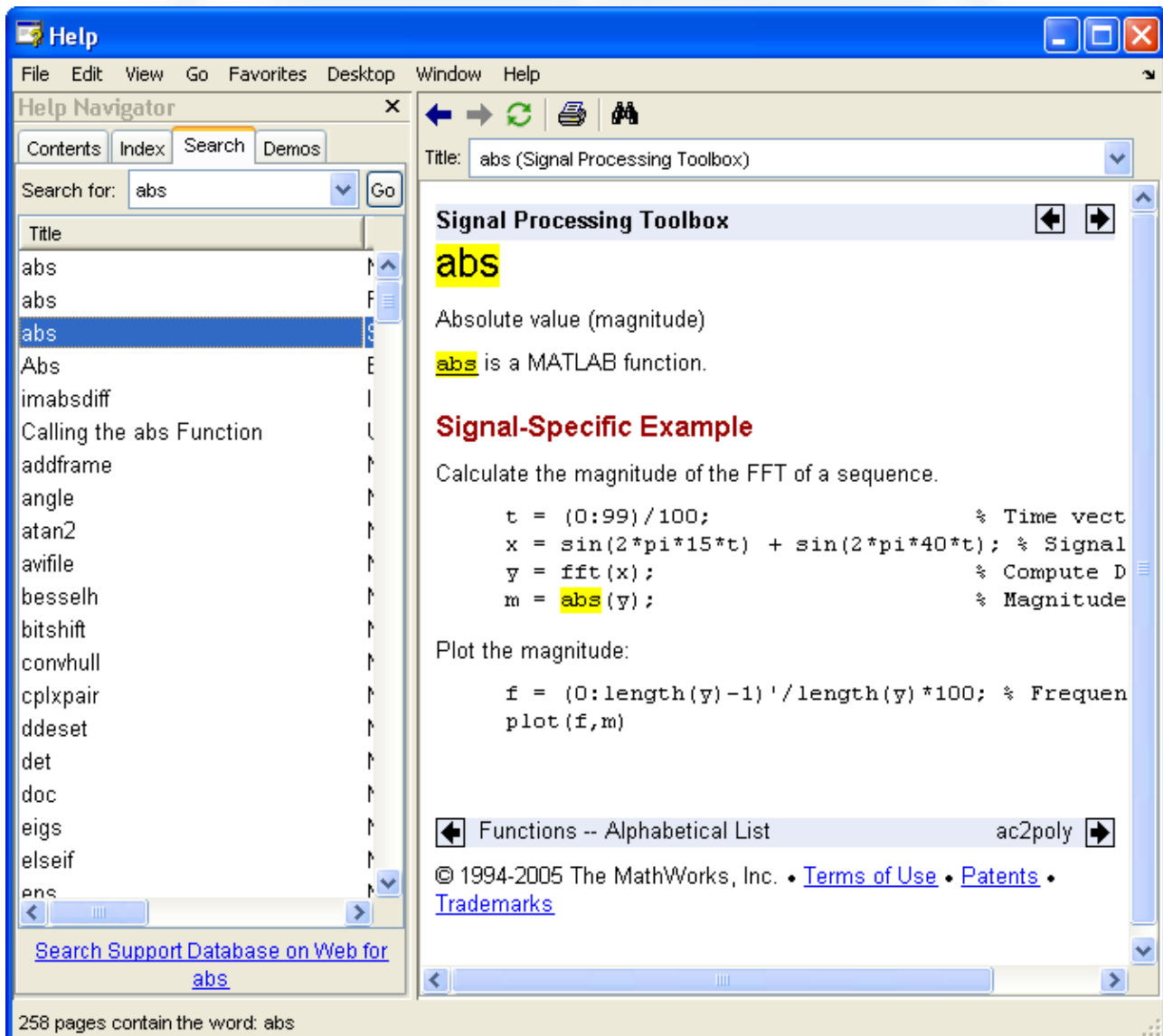


Figura 15: Doc da função abs

Não se esqueça: O Matlab é um programa de 40 anos, desenvolvido por profissionais de muitas áreas. Portanto, é possível que parte ou grande parte do seu trabalho esteja pronta, e implementada de forma extremamente eficiente. Portanto, sempre que possível, procure no Help do Matlab o que você deseja implementar e só implemente a funcionalidade desejada quando esta não existir.

Seção 1.2. Tipos de variáveis

Uma grande vantagem que o Matlab apresenta para o usuário é a sua capacidade de realizar diversos cálculos com vetores e matrizes. Além disso, diferentemente de outras linguagens de programação como C, C++, JAVA, nas quais os dados são caracterizados por tipos escolhidos pelo programador descrevendo precisão numérica ou caracteres alfanuméricos (int, double, long, char, float, short etc.), no Matlab esses dados são descritos automaticamente. Por exemplo, na tabela abaixo, pode-se observar que em outras linguagens existe mais de uma maneira de declarar uma variável com o valor 1, cada uma ocupando espaço em memória diferente. Em Matlab, a notação é mais simplificada:

| C, C++, JAVA | Matlab |
|--|--------|
| double c = 1; int c = 1; float c = 1; long c = 1; short c = 1; | c = 1; |

- Variáveis**

Para o nome das variáveis, existe diferenciação entre letras maiúsculas e minúsculas, além da existência de caracteres que não são admitidos. Basicamente, o nome da variável não pode ser iniciada por número e em nenhuma posição possuir algum operador da sua linguagem (por exemplo, +, -, *, %, @, /, <, >, etc.).

Exemplos:

| | |
|--|---|
| <pre>>> 1ad = 2 ??? 1ad = 2 Error: Unexpected MATLAB expression. >> &ad = 2 ??? &ad = 2 Error: Unexpected MATLAB operator. >> >ad = 2 ??? >ad = 2 Error: Unexpected MATLAB operator.</pre> | <pre>>> a1d = 2 a1d = 2 >> ad1 = 2 ad1 = 2 >> a&d = 2 ??? a&d = 2 Error: The expression to the left of the equals sign is not a valid target for an assignment.</pre> |
|--|---|

Nomes para variáveis podem ter poucos ou muitos caracteres.

Exemplos:

```
>> Pneumoultramicroscopicossilicovulcanoconiotico = 100

Pneumoultramicroscopicossilicovulcanoconiotico =

    100

>> Pneumoultramicroscopicossilicovulcanoconiose = 200

Pneumoultramicroscopicossilicovulcanoconiose =

    200

>> Hipopotomonstrosesquipedaliofobia = 300

Hipopotomonstrosesquipedaliofobia =

    300
```

Na linguagem de programação do Matlab, as variáveis são consideradas como matrizes de 1x1.

Exemplo:

c = 1 representa o mesmo que c = [1]

IMPORTANTE: Para a notação de casas decimais, é utilizado ponto “.”. A vírgula (“,”) é utilizada como um separador de comandos, podendo gerar erros de lógica.

Exemplo:

| | |
|--|---|
| <pre>>> a = 1.987 a = 1.9870</pre> | <pre>>> a = 1,987 a = 1 (valor anterior a virgula) ans = 987 (valor depois da vírgula) - equivale a: >> a = 1 a = 1 >> 987 ans = 987</pre> |
|--|---|

– Números Complexos

Números complexos são representados com i ou j multiplicando a parte imaginária do número. Podem ser obtidos utilizando-se a notação: `complex(parte real, parte imaginária)`.

Exemplo:

`a = 1+2*i` ou `a = 1+2*j` ou `a = complex(1,2)`

Portanto, não é recomendável a utilização de i e j como variável, pois pode gerar erros de cálculo.

Exemplos:

| | |
|---|---|
| <pre>>> i = 2 i = 2 >> a = 1+2*i (representa número imaginário?) a = 5 (resultado real)</pre> | <pre>>> b = 1+3*i (b é número imaginário) b = 1.0000 + 3.0000i >> i = 2 (i recebe valor real) i = 2 >> b = 1+3*i b = 7 (b passa a ser número real)</pre> |
|---|---|

– Funções para números complexos

Sendo b um número complexo:

abs(b) – calcula o seu módulo

arg(b) – calcula o argumento ou fase em radianos (a maioria das funções do Matlab utiliza ângulos em radianos)

real(b) – retorna somente a sua parte real

imag(b) – retorna um número real que é a parte imaginária do número

Exemplos:

| | |
|--|--|
| <pre>>> a = 2+3*i a = 2.0000 + 3.0000i >> modulo = abs(a) modulo = 3.6056 >> fase = angle(a) fase = 0.9828 radianos)</pre> | <pre>>> parte_real = real(a) parte_real = 2 >> parte_imaginaria = imag(a) parte_imaginaria = 3 (Nota: "_" - underline - pode ser utilizado para o nome de variaveis)</pre> |
|--|--|

– Vetores e Matrizes

Na sintaxe tanto de ambos, inicia-se com o uso de colchete “[” para indicar o início de um vetor/matriz e outro colchete “]” para indicar o seu fim. Colunas são indicadas por espaço “ ” e linhas por ponto e vírgula “;”. Parênteses “()” podem ser utilizados internamente ao código para organizar a disposição dos seus elementos.

| Matriz/Vetor | Em linguagem Matlab |
|---|---|
| $a = [1 \ 2 \ 3 \ 4]$ | $a = [1 \ 2 \ 3 \ 4]$ |
| $B = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$ | $B = [1;2;3;4]$ |
| $mat = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ | $mat = [1 \ 2; 3 \ 4]$ |
| $C = [1+2*i \ 2 \ 3+6*i \ 4]$ | $C = [(1+2*i) \ 2 \ (3+6*i) \ 4]$ |
| $E = \begin{bmatrix} 1 & 5*i & 3 \\ 4 & 5 & 6+3*i \\ 7 & 8 & 9 \end{bmatrix}$ | $E = [1 \ 5*i \ 3 ; 4 \ 5 \ 6+3*i ; 7 \ 8 \ 9]$ |

– Infinito – inf

É a representação do infinito, sendo resultado de operações de divisão por zero ou em casos em que o cálculo atinge valores acima do máximo permitido pela precisão numérica.

Exemplos:

| | |
|--|--|
| <pre>>> 1/0 Warning: Divide by zero. ans = Inf >> 1000/0 Warning: Divide by zero. ans = Inf</pre> | <pre>>> exp(1000) ans = Inf >> log(0) Warning: Log of zero. ans = -Inf</pre> |
|--|--|

– NaN (Not a Number)

É obtido quando ocorrem operações do tipo 0 dividido por 0 ou inf dividido por inf.

Exemplo:

| | |
|---|--|
| <pre>>> 0/0 Warning: Divide by zero. ans = NaN</pre> | <pre>>> inf/inf ans = NaN</pre> |
|---|--|

– Boolean

É o resultado de operações lógicas é representado por *true* ou *false*.

– Caracteres

Em Matlab, caracteres são identificados entre aspas simples `' '`. Possuem significado na forma de texto.

Exemplo:

| | |
|--|-------------------------------------|
| <pre>>> a = 'B' a = B (variável a possui letra B)</pre> | <pre>>> c = 'E' c = E</pre> |
|--|-------------------------------------|

– String

É o conjunto de caracteres organizados na forma de vetor. É comumente utilizada em legendas de gráficos, para nomear campos de *struct* ou *cell*.

Exemplo:

| | |
|---|---|
| <pre>>> vet = ['ma' 't' 'l' 'ab'] vet = matlab >> vet = 'matlab' vet = matlab</pre> | <pre>>> vet = ['ma' 't' 'l' 'ab' 'MATLAB'] vet = matlabMATLAB >> vet = ['ma' 't' 'l' ' ' 'ab' 'MATLAB'] vet = matl abMATLAB</pre> |
|---|---|

– Struct

É um conjunto de dados no qual o programador pode agrupar diversos campos com seus respectivos valores, nomeando-os com o uso de *string*. Possui como sintaxe básica:

```
s = struct('field1', values1, 'field2', values2, ...)
```

Em *field*, o campo deve ser nomeado e os valores devem ser atribuídos em *values*.

```
s = struct('field1', {}, 'field2', {}, ...)
```

Exemplo:

| | |
|---|--|
| <pre>>> s = struct('campo1',{1 2 3 4 5},'campo2',{6 7 8 9 0}) s = 1x5 struct array with fields: campo1 campo2 >> s(1) ans = campo1: 1 campo2: 6 >> s(2) ans = campo1: 2 campo2: 7</pre> | <pre>>> s(3) ans = campo1: 3 campo2: 8 >> s(4) ans = campo1: 4 campo2: 9 >> s(5) ans = campo1: 5 campo2: 0</pre> |
|---|--|

– Cell

Uma Cell é uma matriz onde se pode guardar qualquer tipo de variável em qualquer uma de suas posições, sem que o programa gere erros. Este tipo de dados é muito usado para guardar Strings, pois guardar Strings em Matrizes exige que elas possuam o mesmo número de caracteres, o que dificulta o processo de programação.

Basicamente, em uma Cell, pode-se guardar qualquer tipo de dado, mesmo que ele seja outra Cell. Para acessar os campos de uma Cell, basta utilizar a mesma sintaxe das matrizes, porém, no lugar dos parênteses (“(” e “)”) utilizamos chaves (“{” e “}”).

Exemplos:

```
>> A = {'Matlab', 1:10 ; struct('a',1), eye(10)}
A =
    'Matlab'      [ 1x10 double]
    [1x1 struct]  [10x10 double]
>> B = {'123',A}
B =
    '123'      {2x2 cell}
>> B{1,2}
ans =
    'Matlab'      [ 1x10 double]
    [1x1 struct]  [10x10 double]
>>
```

Seção 1.3 – Operadores

Para facilitar a manipulação de números, matrizes, vetores, e executar outras funcionalidades o Matlab dispõe de muitos operadores. Um operador é um símbolo que indica uma operação entre argumentos.

Lembre-se: O bom conhecimento dos operadores do Matlab tornará seu programa mais simples, seguro e eficiente, e facilitará em muito o seu trabalho.

Caracteres Especiais

Os caracteres especiais no Matlab são parte da sintaxe do programa, e provavelmente serão os mais utilizados:

- **Colon “:”**

O colon (ou “dois pontos”) tem alguns usos muito interessantes. O primeiro que se deve citar é a criação de vetores. Assim, se digitarmos no prompt de comando `a:b`, sendo o valor de “a” maior do que o de “b”, o programa automaticamente criará um vetor linha com valores partindo de “a”, de um em um, até o menor valor mais próximo de “b”.

Exemplo 1:

```
>> 1:10

ans =

     1     2     3     4     5     6     7     8     9    10

>>
```

Exemplo 2:

```
>> 1.7:6.9

ans =

    1.7000    2.7000    3.7000    4.7000    5.7000    6.7000

>>
```

Porém, se usarmos duas vezes o colon para criar um vetor, podemos especificar um intervalo diferente de 1 entre os números. Ao digitarmos no prompt `a:b:c`, obedecendo às seguintes condições:

- $b > 0$ se $c > a$
- $b < 0$ se $c < a$
- $|b| < |a - c|$
- $b \neq 0$

O Matlab criará um vetor cujo primeiro elemento é “a”, o intervalo entre os elementos é “b”, e o último elemento é o de menor módulo que for mais próximo de “c”, ou seja, ele criará um vetor de “a” até “c”, de “b” em “b”.

Exemplo 1:

```
>> 2:2:10  
  
ans =  
  
     2     4     6     8    10  
  
>>
```

Exemplo 2:

```
>> 1:0.5:3  
  
ans =  
  
    1.0000    1.5000    2.0000    2.5000    3.0000  
  
>>
```

Exemplo 3:

```
>> 10:-1:3  
  
ans =  
  
    10     9     8     7     6     5     4     3  
  
>>
```

Exemplo 4:

```
>> 2.1:0.7:5  
  
ans =  
  
    2.1000    2.8000    3.5000    4.2000    4.9000  
  
>>
```

Exemplo 5:

```
>> 5:-0.7:2  
  
ans =  
  
    5.0000    4.3000    3.6000    2.9000    2.2000  
  
>>
```

O último uso para o colon é no acesso de múltiplas posições em uma matriz ou vetor, e será abordado a parte, ao final deste capítulo.

- **Parênteses “(” “)”**

Os parênteses são muito utilizados no Matlab. Eles têm três utilidades básicas: estabelecer ordem de precedência entre operações, identificar argumentos de funções e acessar posições em uma matriz.

- Ordem de precedência entre operações:

Basicamente, o Matlab executará primeiro as operações que estão dentro dos parênteses, para depois executar as operações que estão fora dele, tal qual manda a sintaxe matemática à qual estamos habituados.

Exemplo 1:

```
>> 8/2*3
ans =
    12
>>
```

Neste caso, quando digitamos a operação 8/2*3, para o Matlab, as operações de divisão “/” e multiplicação “*” estão no mesmo nível de hierarquia. Assim sendo, ele as executa na ordem, da esquerda para a direita. Deste modo, a conta que o Matlab interpretou foi:

$$8/2*3 = \left(\frac{8}{2}\right) \times 3 = 4 \times 3 = 12$$

Exemplo 2:

```
>> 8/(2*3)
ans =
    1.3333
>>
```

Neste caso, estabelecemos que a operação 2*3 deveria ser executada primeiro, e depois, 8 deveria ser dividido pelo resultado. Deste modo, a conta que o Matlab interpretou foi:

$$8/(2*3) = \frac{8}{(2 \times 3)} = \frac{8}{6} = 1.333...$$

IMPORTANTE: Alguns operadores no Matlab têm hierarquia sobre outros. Isto significa, que as operações que envolvem estes operadores são executadas primeiro nos comandos dados. Bem

como na álgebra comum, a divisão (“/”) e a multiplicação (“*”) têm preferência sobre a soma (“+”) a subtração (“-”), o que significa que o Matlab executará primeiro as multiplicações, divisões, e operações internas aos parênteses, da esquerda para a direita. Logo após isto, ele executará as operações de soma e subtração.

– Identificar argumentos de funções

É muito comum que se precise passar argumentos para as funções para que elas possam retornar os resultados desejados. Para se invocar a função, é necessário digitar:

nome_da_funcao(arg1,arg2,...,argn)

Onde *nome_da_funcao* é a função que se deseja invocar, e dentro dos parênteses, são passados os argumentos. Se for necessário mais de um argumento, eles devem ser separados por vírgulas “,”.

Exemplo 1:

Neste exemplo, iremos invocar a função *abs* que retorna o valor absoluto de um número:

```
>> abs(-3)

ans =

     3

>>
```

– Acessar posições em um vetor ou matriz

Para acessar uma posição de um vetor ou matriz, fazemos uso da sintaxe:

nome_da_matriz(i, j, k, ... , n)

Onde *nome_da_matriz* é o nome da Matriz cuja posição desejamos acessar, e o índice da posição é dado pelos argumentos *i, j, k, ... , n* separados por vírgulas. Note que a matriz pode ter de 1 a “n” dimensões.

Exemplo:

```
>> 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

>>

Observe que para o caso de matrizes bidimensionais, a primeira coordenada é o número da linha onde está a posição desejada, e a segunda coordenada é o número da coluna da respectiva posição.

- **Colchetes “[” “]”**

Os colchetes são usados para a construção de matrizes e vetores em geral. Os argumentos internos aos colchetes devem ser separados por vírgulas (“,”) ou espaços, se estiverem na mesma linha da matriz, e ponto e vírgula(“;”) ou quebra de linha se pertencerem a linhas diferentes.

Exemplo 1:

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

```
A =
```

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

```
>>
```

Exemplo 2:

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

```
A =
```

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

```
>>
```

Exemplo 3:

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

```
A =
```

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

```
>>
```

Neste exemplo, foi digitado no prompt:

$A = [1 \ 2 \ 3 \text{ (enter)} \ 4 \ 5 \ 6 \text{ (enter)} \ 7 \ 8 \ 9] \text{ (enter)}$

E o Matlab só criou a matriz quando fechamos o colchete que estava aberto.

Os argumentos internos dos vetores também podem ser outros vetores ou matrizes. O programa irá concatená-los se as dimensões de ambos forem compatíveis.

Exemplo 1:

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

A =

     1     2     3
     4     5     6
     7     8     9

>> B = [10 11 12; 13 14 15]

B =

    10    11    12
    13    14    15

>> [A;B]

ans =

     1     2     3
     4     5     6
     7     8     9
    10    11    12
    13    14    15

>>
```

Exemplo 2:

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

A =

     1     4     7
     2     5     8
     3     6     9

>> B = [10 13; 11 14; 12 15]

B =

    10    13
    11    14
    12    15

>> [A B]
```

```
ans =
    1     4     7    10    13
    2     5     8    11    14
    3     6     9    12    15
```

```
>>
```

Exemplo 3:

```
>> [ 1:3 ; 4:6 ; 7:9 ]
```

```
ans =
    1     2     3
    4     5     6
    7     8     9
```

```
>>
```

- **Chaves “{” “}”**

As chaves têm um uso análogo ao dos colchetes. Porém, elas criarão Cells e não matrizes ou vetores. Desta forma, não será possível, fazendo o uso de chaves, concatenar vetores, matrizes, strings, variáveis, ou até mesmo Cells. Quando se faz o uso das chaves, o retorno do comando será uma nova Cell com os campos passados.

Exemplo 1:

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

```
A =
    1     4     7
    2     5     8
    3     6     9
```

```
>> B = [10 13; 11 14; 12 15]
```

```
B =
    10    13
    11    14
    12    15
```

```
>> {A B}
```

```
ans =
    [3x3 double]    [3x2 double]
```

```
>>
```

Note que, ao invés de concatenar as matrizes lado a lado, o resultado é uma nova Cell onde o primeiro campo é a matriz A e o segundo é a matriz B.

Exemplo 2:

```

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

A =

     1     4     7
     2     5     8
     3     6     9

>> B = [10 13; 11 14; 12 15]

B =

    10    13
    11    14
    12    15

>> {[A B] {A; B}}

ans =

    {1x2 cell}    {2x1 cell}

>>

```

Neste caso, o retorno do comando é uma Cell que possui uma Cell em cada uma de suas posições. A primeira Cell é uma Cell linha, com as matrizes A e B. A segunda é uma Cell coluna com as matrizes A e B. Note que as dimensões das matrizes não são compatíveis para que elas sejam concatenadas em coluna. Mesmo assim, a Cell foi criada, pois as matrizes não foram concatenadas, mas sim guardadas em posições diferentes da Cell.

As chaves também servem para acessar os campos da Cell. A sintaxe é a mesma da matriz, mas ao invés de usar parênteses, devemos usar chaves na frente do nome da Cell.

Exemplo:

```

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

A =

     1     4     7
     2     5     8
     3     6     9

>> B = [10 13; 11 14; 12 15]

B =

    10    13
    11    14
    12    15

>> C = {A B}

```

```

C =
      [3x3 double]      [3x2 double]
>> C{1,1}
ans =
      1      4      7
      2      5      8
      3      6      9
>> C{1,2}
ans =
     10     13
     11     14
     12     15

```

- **Ponto “.”**

O ponto tem duas funções no Matlab: separador de casas decimais (diferente do padrão brasileiro, que é a vírgula), e meio de acesso a campos de struct.

- Separador de casas decimais

No Matlab, digitar 3.5 e 3,5 tem uma diferença significativa. E esta diferença é pelo fato de que ambas as sentenças estão certas, mas significam comandos absolutamente diferentes. Deste modo, este erro não gera nenhuma mensagem de erro, mas seus cálculos sofrerão grandes penas.

O ponto (“.”) é o real separador de casas decimais. Neste ambiente, a vírgula (“,”) serve somente para separar comandos que estão digitados na mesma linha.

IMPORTANTE: Digitar 3,5 é o mesmo que digitar 3, apertar enter, digitar 5 e apertar enter novamente. Ou pior: digite A = 3,5 e veja o que acontece!

Exemplo 1:

```

>> 3.5
ans =
     3.5000
>> A = 3.5
A =
     3.5000
>>

```

Esta é a forma correta de se fazer. Observe que no primeiro comando, o valor de `ans` é 3.5, e no segundo comando, o valor de `A` é 3.5 .

Exemplo 2:

```
>> 3,5
ans =
    3
ans =
    5
>>
```

Esta é a forma incorreta. Neste caso, foram enviados ao programa dois comandos em uma linha. Deste modo, primeiro a variável `ans` vale 3 e depois ela passa a valer 5, quando ao final do processo, era desejado que ela tivesse o valor 3.5.

Exemplo 3:

```
>> A = 3,5
A =
    3
ans =
    5
>>
```

Neste caso, o erro é ainda mais grave, pois a variável `A` é criada, e ela termina com o valor 3, enquanto a variável `ans` que não deveria ter sido criada, foi criada também e possui o valor 5. O desejado era que `ans` não fosse criada e `A` possuísse o valor 3.5 .

– Meio de acesso a campos de struct

Quando se possui uma struct, é necessário saber qual o valor de seus campos internos. Para isso, utilizamos o ponto. Para acessar o campo, tanto para guardar quanto para resgatar um valor, a sintaxe é:

nome_da_struct.nome_do_campo

Onde *nome_da_struct* é o nome da variável onde está salva a struct, e *nome_do_campo* é o nome do campo interno da struct que se deseja acessar.

Exemplo 1:

Imagine que a nossa struct vai representar um círculo. Para isso, precisaremos de 4 campos: coordenadas x e y do centro, raio e cor. Para isso, criamos a variável Círculo, e guardamos nela uma struct com os campos necessários:

```
>> circulo = struct('X0', 0, 'Y0', 0, 'R', 3, 'Cor', 'vermelho')

circulo =

    X0: 0
    Y0: 0
     R: 3
  Cor: 'vermelho'

>>
```

Note que a sintaxe para criar uma struct é muito simples: basta utilizar a função struct, e passar como argumentos o nome de cada campo, e em seguida seu valor. Se digitarmos struct('a',b,'c',d), criaremos uma struct com o campo “a”, que guarda o valor b, e com o campo “c” que guarda o valor d. Continuando com o exemplo do círculo, agora vamos acessar e alterar o valor de seu raio.

Acessando o valor do raio:

```
>> circulo.R

ans =

     3

>>
```

Com o simples uso do ponto, conseguimos acessar o valor desejado com agilidade. Agora, vamos alterar o valor de seu raio:

```
>> circulo.R = 4

circulo =

    X0: 0
    Y0: 0
     R: 4
  Cor: 'vermelho'

>>
```

Agora, fazendo o mesmo uso do ponto, conseguimos alterar um valor interno da struct. Além disso, o Matlab mostrou todos os campos da struct, com seu novo valor.

- ***Reticências “...”***

As reticências neste ambiente têm a função única de permitir que uma linha grande de código seja

separada em várias linhas, para que possamos entender melhor o código gerado. Quando digitarmos nossos códigos em arquivos, seu uso será melhor justificado. Mas, como todos os outros comandos, as reticências também podem ser utilizadas no prompt de comando. E seu uso é simples: basta digitar “...” ao final de sua linha e apertar enter, que o Matlab não executará o que foi digitado. Ele executará todas as linhas digitadas desta maneira em conjunto, quando uma delas (a última) for digitada sem as reticências.

Exemplo 1:

Neste exemplo, criaremos a mesma struct do círculo, porém em várias linhas:

```
>> circulo = struct('X0', 0, ...
'Y0',0,...
'R' ,3,...
'Cor','vermelho')

circulo =

    X0: 0
    Y0: 0
     R: 3
   Cor: 'vermelho'

>>
```

IMPORTANTE: Tudo o que for digitado depois das reticências é considerado comentário, ou seja, não é código, e não será interpretado.

Exemplo 2:

```
>> circulo = struct('X0', 0, ... Crio um círculo com X0 = 0
'Y0',0,... Com Y0 = 0
'R' ,3,... Com R = 3
'Cor','vermelho') % e vermelho

circulo =

    X0: 0
    Y0: 0
     R: 3
   Cor: 'vermelho'

>>
```

Note que na última linha, não havia reticências, então foi necessário utilizar o caractere “%” para inserir um comentário a partir dali.

- ***Vírgula “,”***

No Matlab, a vírgula assume quatro funções: Separar argumentos em uma função, separar coordenadas em uma matriz, separar comandos em uma linha com vários comandos, e separar as colunas de uma matriz.

– Separar argumentos de uma função

Exemplo 1:

```
>> strcat('mat','lab')  
  
ans =  
  
matlab  
  
>>
```

Neste caso, invocamos a função `strcat`, que recebe como argumentos um conjunto de strings retorna uma string resultante da concatenação das strings passadas como argumento, da esquerda para a direita. É possível perceber que os argumentos são separados por vírgula (“,”).

– Separar coordenadas de uma matriz

Exemplo 2:

```
>> 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  
  
>>
```

Neste exemplo, criamos a matriz `A`, e acessamos a posição que se encontra na linha 2 e na coluna 3 desta matriz. Observe que no comando de acesso à posição, o número da linha e o da coluna são separados por vírgula (“,”).

– Separar comandos em uma linha com múltiplos comandos

Exemplo 3:

```
>> A=1, B=2, 3+4, 5*2, 3/7, A*B  
  
A =  
  
     1
```

```

B =

    2

ans =

    7

ans =

   10

ans =

    0.4286

ans =

    2

>>

```

Agora, as vírgulas são usadas para separar os comandos que foram escritos em uma linha única. Observe que o programa exibe 6 saídas, uma para cada comando.

– Separar as colunas de uma matriz

Ao escrever uma matriz, as colunas podem ser separadas por espaços ou vírgulas. Porém, o uso das vírgulas pode tornar a visualização do código mais clara, quando os elementos da matriz não são números, mas operações:

Exemplo 4:

```

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

A =

    1    2    3
    4    5    6
    7    8    9

>>

```

Exemplo 5:

```

>> A = [1, -2, 3+4; 4*7, 5+4, 6+1; 7-3.4, 8.7, 9]

A =

    1.0000   -2.0000    7.0000
   28.0000    9.0000    7.0000

```

```

3.6000    8.7000    9.0000
>>

```

- **Semicolon “;”**

O semicolon ou “ponto e vírgula” tem 2 usos: omitir a impressão da saída, e separar linhas da matriz.

- Omitir impressão de saída:

Exemplo 1:

```

>> A=10;
>> A

A =

    10

>>

```

Note que quando executamos o comando seguido de semicolon (“;”), o ambiente não exibe nenhuma mensagem, mas executa o comando. Quando invocamos a variável A, o valor 10 que colocamos nela, está lá. Esta é uma grande vantagem do Matlab, pois imprimir mensagens na tela é uma operação muito demorada, e pode comprometer em muito a velocidade de execução da sua rotina.

Lembre-se: Omitir mensagens desnecessárias pode reduzir muito o tempo de processamento de sua rotina. Porém, na etapa de desenvolvimento da rotina, exibir estas mensagens pode ser muito útil para corrigir, ou “debugar” seu código.

- Separar as linhas da matriz

Podemos usar o semicolon para separar as linhas de uma matriz sem ter que quebrar uma linha para tanto.

Exemplo 2:

```

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

A =

     1     2     3
     4     5     6
     7     8     9

>>

```


- **Porcentagem “%”**

Quando o Matlab identifica o operador “%” em uma linha, tudo o que está à sua frente é considerado comentário e não é executado como código. Note que, no prompt de comando, os comentários ficam da cor verde.

Exemplo 1:

```
>> A=1% Observações importantes sobre este comando
A =
    1
>>
```

- **Ponto de exclamação “!”**

Este operador, se colocado antes de um comando, envia a mensagem que está após ele para o sistema operacional. Ao usar este comando, você pode digitar qualquer comando da sintaxe do MS-DOS, que ele será reconhecido.

Exemplo 1:

```
>> !dir
O volume na unidade C não tem nome.
O número de série do volume , 7863-32EB

Pasta de C:\Arquivos de programas\MATLAB71\work

18/09/2008  15:14    <DIR>          .
18/09/2008  15:14    <DIR>          ..
                0 arquivo(s)          0 bytes
                2 pasta(s) 88.871.419.904 bytes disponíveis
>>
```

- **Igual “=”**

O sinal de igual “=” serve para guardar um valor em uma variável, posição de matriz ou vetor, campo de struct, cell ou qualquer outro lugar onde se possa salvar um valor.

Exemplo 1:

Salvando em uma variável.

```
>> A=1
A =
    1
>>
```

Exemplo 2:

Salvando em uma posição de matriz.

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

```
A =
```

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

```
>> A(2,3) = 15
```

```
A =
```

```
    1    2    3
    4    5   15
    7    8    9
```

```
>>
```

Exemplo 3:

Salvando em um campo de uma struct

```
>> circulo = struct('X0', 0, 'Y0', 0, 'R', 3, 'Cor', 'vermelho')
```

```
circulo =
```

```
    X0: 0
    Y0: 0
     R: 3
   Cor: 'vermelho'
```

```
>> circulo.R = 4
```

```
circulo =
```

```
    X0: 0
    Y0: 0
     R: 4
   Cor: 'vermelho'
```

```
>>
```

Exemplo 4:

Salvando em um campo de uma cell.

```
>> minhaCell = {'yuhuuuu', [1 2 3], 7}
```

```
minhaCell =
```

```
    'yuhuuuu'    [1x3 double]    [7]
```

```
>> minhaCell{1,3} = 'aaa'

minhaCell =

    'yuhuuuu'    [1x3 double]    'aaa'

>>
```

Operadores Aritméticos

Naturalmente, os operadores matemáticos são suados exclusivamente para fazer operações aritméticas, sejam elas entre matrizes, números escalares ou complexos, vetores, ou combinações convenientes destes. Estes operadores seguem a todas as regras da álgebra comum, como regras de precedência e concordância entre número de linhas e colunas, nos casos matriciais e vetoriais.

- **Mais “+”**

A operação de soma pode ser feita entre números reais ou complexos, matrizes de mesmo número de linhas e colunas, ou vetores de mesmo número de linhas ou colunas.

Exemplo 1:

```
>> 37+45

ans =

    82

>>
```

Exemplo 2:

```
>> 3+4*i-6-7.2*i

ans =

   -3.0000 - 3.2000i

>>
```

Exemplo 3:

```
>> [1 2 3] + [6 5 4]

ans =

     7     7     7

>>
```

Exemplo 4:

```
>> [1 0 0; 0 1 0; 0 0 1] + [0 0 1; 0 1 0; 1 0 0]
ans =
     1     0     1
     0     2     0
     1     0     1
>>
```

- **Menos “-”**

A subtração funciona exatamente da mesma maneira que a soma, ou seja, pode ser feita entre números reais ou complexos, matrizes de mesmo número de linhas e colunas, ou vetores de mesmo número de linhas ou colunas.

- **Multiplicação “*”**

O operador “*” executa multiplicação entre escalares, escalares e matrizes, e entre matrizes (que podem ser também vetores) desde que elas obedeçam à regra de multiplicação de matrizes, que diz que o número de colunas da matriz da esquerda deve ser igual ao número de linhas da matriz da direita.

Exemplo 1:

Multiplicação entre escalares

```
>> 5*2
ans =
    10
>>
```

Exemplo 2:

Multiplicação escalar - matriz

```
>> 3*[1 2;3 4]
ans =
     3     6
     9    12
>>
```

Exemplo 3:

Multiplicação matricial entre vetores

```
>> [1 2 3]*[4;5;6]

ans =

    32

>>
```

Exemplo 4:

Multiplicação vetor - matriz

```
>> [1 2]*[3 4;5 6]

ans =

    13    16

>>
```

Exemplo 5:

Multiplicação entre matrizes:

```
>> [1 2 3; 4 5 6]*[1 6; 2 5; 3 4]

ans =

    14    28
    32    73

>>
```

- ***Divisão à direita “/” e à esquerda “\”***

Os operadores de divisão à direita e à esquerda podem ser aplicados entre escalares, escalares e matrizes, matrizes e vetores, e entre matrizes.

- Aplicação em escalares

A divisão comum, à qual estamos habituados, é executada com a barra de divisão à direita. Assim, quando digitamos a/b , o Matlab executa a seguinte operação:

$$a/b = a \times \frac{1}{b} = \frac{a}{b}$$

Portanto quando a divisão é à direita, o termo que é invertido é o da direita.

Quando utilizamos a divisão à esquerda, a operação que o Matlab executa é exatamente a inversa.

Ele inverte o termo da esquerda, e o multiplica pelo termo da direita:

$$a \setminus b = \frac{1}{a} \times b = \frac{b}{a}$$

Exemplo 1:

```
>> 3/2
ans =
    1.5000
>>
```

Exemplo 2:

```
>> 3\2
ans =
    0.6667
>>
```

– Aplicação entre escalares e matrizes

O princípio de aplicação destes operadores entre escalares e matrizes é um pouco diferente. Neste caso, na divisão à direita, o escalar deve sempre estar à direita, e na divisão à esquerda, o escalar deve sempre estar à esquerda. Isto significa que quando a operação é executada entre um escalar e uma matriz, o Matlab consegue inverter sempre o escalar, mas nunca a matriz.

$$[A]/b = [A] \times \frac{1}{b}$$

E no caso da divisão à esquerda, temos:

$$b \setminus [A] = \frac{1}{b} \times [A]$$

Exemplo 1:

```
>> A = [1 4 7; 2 5 8; 3 6 10]
A =
     1     4     7
     2     5     8
     3     6    10
>> A/3
```

```
ans =
    0.3333    1.3333    2.3333
    0.6667    1.6667    2.6667
    1.0000    2.0000    3.3333

>>
```

Exemplo 2:

```
>> A = [1 4 7; 2 5 8; 3 6 10]

A =
     1     4     7
     2     5     8
     3     6    10

>> 3\A

ans =
    0.3333    1.3333    2.3333
    0.6667    1.6667    2.6667
    1.0000    2.0000    3.3333

>>
```

– Aplicação entre matrizes e vetores

Vetores são entidades matemáticas que não possuem operação inversa definida. Portanto, a divisão à direita, quando se trata de vetores e matrizes, acontecerá somente se o elemento da direita for uma matriz e o da esquerda um vetor com o número de colunas igual ao número de linhas da matriz. Analogamente, a divisão à esquerda ocorrerá se o elemento da esquerda for uma matriz, e o da direita for um vetor com o número de linhas igual ao número de colunas da matriz. Isto ocorre pois o Matlab interpreta estes comando da seguinte forma:

Seja A uma matriz quadrada e B um vetor coluna, que tem tantas linhas quantas colunas tiver a matriz A. Deste modo, a divisão à esquerda será interpretada como:

$$A \setminus B = [A]^{-1} \times \{B\}$$

Seja agora A uma matriz quadrada e B um vetor linha que possui tantas linhas quantas colunas tiver a matriz A. Agora, a divisão à direita será interpretada como:

$$B / A = \{B\} \times [A]^{-1}$$

IMPORTANTE: Caso as operações obedeçam às condições de quantidade de linhas e colunas de cada elemento, e a matriz A não for quadrada, os comandos ainda assim funcionarão. Porém ao invés de calcular a inversa da matriz A, o Matlab calculará a pseudo-inversa da matriz A, pelo método de mínimos quadrados.

Lembre-se: O bom uso dos operadores do Matlab pode facilitar a sua vida como programador. Imagine que você tem que resolver um sistema de equações linear da forma:

$$[A]\{x\} = \{B\}$$

Se o sistema é definido, sua resposta é dada por:

$$\{x\} = [A]^{-1}\{B\}$$

O que, no Matlab é calculado como:

```
x = A\B
```

Este é um dos usos da divisão à esquerda.

- ***Acento circunflexo “^”***

O acento circunflexo representa a operação de potência. Sempre, um dos dois argumentos deve ser um número escalar. Se o argumento da direita for um número escalar, ele elevará o argumento da esquerda à potência do argumento da direita, como conhecemos normalmente. Caso contrário, o Matlab fara uso da exponencial matricial. Em ambos os casos, o argumento não escalar deve ser uma matriz quadrada.

- O argumento da direita é um escalar

Este é o caso das potências comuns.

Exemplo 1:

```
>> 5^2
ans =
    25
>>
```

Exemplo 2:

```
>> 5.3^1.7
ans =
    17.0321
>>
```


Exemplo 3:

```
>> [1 2;3 4]^2
ans =
     7     10
    15     22
>>
```

Exemplo 4:

```
>> [1 2;3 4]^1.76
ans =
    4.7029 - 0.0915i    6.6674 + 0.0419i
   10.0011 + 0.0628i   14.7040 - 0.0287i
>>
```

– O argumento da esquerda é um escalar.

Sendo a operação exponencial matricial definida pela função *expm*, A uma matriz quadrada, e b um escalar, o Matlab interpretará o comando como:

$$b \wedge A = \text{expm}(\ln(b) A)$$

• Aspas simples “ ’ ”.

As aspas simples têm dois usos no Matlab: escrita de strings e calcular o complexo conjugado transposto de matrizes ou vetores.

Exemplo 1:

Escrita de strings:

```
>> A = 'matlab'
A =
matlab
>>
```

Exemplo 2:

Calcular o complexo conjugado transposto de uma matriz:

```
>> A = [1 2;3 4]
A =
     1     2
     3     4
>> A'
ans =
     1     3
     2     4
>> A = [1 2;0 i]
A =
     1.0000     2.0000
           0      0 + 1.0000i
>> A'
ans =
     1.0000           0
     2.0000      0 - 1.0000i
>>
```

- ***Ponto antes dos operadores***

Normalmente, deseja-se fazer operações entre matrizes, que não são definidas na álgebra. As mais comuns são as operações termo-a-termo, por exemplo, elevar todos os termos de uma matriz ao quadrado, multiplicar ou dividir duas matrizes ou vetores termo-a-termo. Para os operadores “*”, “\”, “/”, “^” e “.”, o Matlab permite este tipo de operação desde que se coloque um ponto (“.”) antes do operador.

Exemplo 1:

Multiplicação termo a termo:

```
>> A = [1 2;3 4], B = [5 6;7 8], A.*B
A =
     1     2
     3     4
B =
```

```

    5    6
    7    8

ans =

    5    12
   21    32

>>

```

Divisão termo a termo à direita:

```

>> A = [1 2;3 4], B = [5 6;7 8], A./B

A =

    1    2
    3    4

B =

    5    6
    7    8

ans =

    0.2000    0.3333
    0.4286    0.5000

>>

```

Divisão termo a termo à esquerda:

```

>> A = [1 2;3 4], B = [5 6;7 8], A.\B

A =

    1    2
    3    4

B =

    5    6
    7    8

ans =

    5.0000    3.0000
    2.3333    2.0000

>>

```

Elevação de todos os termos ao quadrado:

```
>> A = [1 2;3 4], A.^2

A =

     1     2
     3     4

ans =

     1     4
     9    16

>>
```

No caso da aspas simples, o uso do ponto faz com que se calcule somente a transposta da matriz, sem obter seu complexo-conjugado.

Exemplo 2:

Comparação do uso de “'” e “.”” em uma matriz complexa:

```
>> A = [1 2*i;3 4]

A =

     1.0000             0 + 2.0000i
     3.0000             4.0000

>> A'

ans =

     1.0000             3.0000
     0 - 2.0000i         4.0000

>> A.'

ans =

     1.0000             3.0000
     0 + 2.0000i         4.0000

>>
```

Operadores Relacionais

Os operadores relacionais representam comparações entre valores. Em caso positivo, ele retornará o valor “true” que no Matlab é representado pelo número 1. Em caso negativo, o valor retornado será “false” representado pelo número 0.

Veja no exemplo abaixo, a representação dos valores “true” e “false” no Matlab.

Exemplo 1:

```
>> true
ans =
     1
>> false
ans =
     0
>>
```

Na tabela abaixo, estão todos os operadores relacionais do Matlab e suas funções básicas:

| Operador | Nome | Sintaxe | Retorna true quando: | Retorna false quando: |
|----------|----------------|------------|-----------------------------------|-----------------------------------|
| < | Menor | $a < b$ | <i>“a” é menor do que “b”</i> | <i>“a” é maior ou igual a “b”</i> |
| <= | Menor ou igual | $a \leq b$ | <i>“a” é menor ou igual a “b”</i> | <i>“a” é maior do que “b”</i> |
| > | Maior | $a > b$ | <i>“a” é maior do que “b”</i> | <i>“a” é menor ou igual a “b”</i> |
| >= | Maior ou igual | $a \geq b$ | <i>“a” é maior ou igual a “b”</i> | <i>“a” é menor do que “b”</i> |
| == | Igual | $a == b$ | <i>“a” é igual a “b”</i> | <i>“a” não é igual a “b”</i> |
| ~= | Diferente | $a \neq b$ | <i>“a” não é igual a “b”</i> | <i>“a” é igual a “b”</i> |

Normalmente, os operadores “a” e “b” são números escalares, mas pode-se usar também vetores ou matrizes, desde que ambos tenham o mesmo tamanho. Neste caso, o resultado da comparação será um vetor ou matriz, de mesmo tamanho dos argumentos, com o valor “true” nas posições onde a condição for verdadeira, e false onde ela for falsa.

Exemplo 1:

```
>> A = [1 3 5 7 9];
>> B = [1 3 7 8 9];
>> A==B
ans =
     1     1     0     0     1
>> A~=B
ans =
     0     0     1     1     0
>> A>B
```

```

ans =
    0    0    0    0    0

>> A<=B

ans =
    1    1    1    1    1

>>

```

Operações Lógicas

As operações lógicas são utilizadas quando, para a identificação de um resultado, mais de uma condição deve ser obedecida. Por exemplo, quando queremos identificar, a partir de uma nota de 0 a 10, qual o conceito (“A”, “B”, “C”, “D” ou “E”) que o aluno obteve. Assim, o conceito do aluno é “A” se sua nota for menor ou igual a 10 E maior ou igual a 8. Este “E” é uma operação lógica que diz que o resultado da comparação será verdadeiro se ambas as condições forem obedecidas simultaneamente.

As operações lógicas básicas conhecidas são o “E”, o “OU” e o “NÃO”, mais conhecidos como “AND”, “OR” e “NOT”. Elas funcionam da seguinte forma:

| Operador | Nome | Sintaxe | True somente se: | Equivalente Matemático |
|----------|------|---------|--|------------------------|
| && | and | C1&&C2 | C1 e C2 forem verdadeiros | Multiplicação |
| | or | C1 C2 | Pelo menos uma das condições forem verdadeiras | Soma |
| ~ | not | ~C1 | C1 for falso | --- |

Como os operadores relacionais retornam 1 se a condição for verdadeira e 0 se ela for falsa, as operações AND e OR têm equivalentes matemáticos que são a Multiplicação e a Soma. No caso do AND, se uma das condições for falsa, o produto entre todas será 0, o que equivale a “false” no Matlab. Esta operação somente retornará “true” se todas as condições forem obedecidas.

No caso do OR, se pelo menos uma das condições for verdadeira, a soma dos resultados será diferente de 0. Então, qualquer resultado diferente de 0 no Matlab será interpretado como verdadeiro (“true”).

Exemplo 1:

```

>> A = 9;
>> (A>5) && (A<10)

ans =
    1

>> (A>5) && (A<8)

```

```

ans =
    0
>> (A>5) || (A<8)
ans =
    1
>> (A>5) || (A<10)
ans =
    1
>> ~(A==8)
ans =
    1
>> (A==8)
ans =
    0

```

Veja que neste exemplo foram feitos os testes para o valor de “A” igual a 9. Foi testado respectivamente se:

- *A é maior do que 5 e menor do que 10? (está entre 5 e 10)*
- *A é maior do que 5 e menor do que 8? (está entre 5 e 8)*
- *A é maior do que 5 ou menor do que 8?*
- *A é maior do que 5 ou menor do que 10?*
- *A não é igual a 8?*
- *A é igual a 8?*

Indexação vetorial

A indexação vetorial, no Matlab é uma maneira muito prática de se acessar partes de matrizes ou vetores. No lugar da coordenada da linha e/ou da coluna que desejamos acessar, podemos passar como argumentos vetores com todas as posições que queremos, de uma só vez. Desta forma, podemos acessar, pegar partes de matrizes, e alterar matrizes rapidamente.

Exemplo 1:

Acessando parte de uma matriz:

```

>> A = [1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15; 16 17 18 19 20; 21 22 23 24 25]
A =

```

```

1      2      3      4      5
6      7      8      9     10
11     12     13     14     15
16     17     18     19     20
21     22     23     24     25

>> A(2:4,3:5)

ans =

      8      9     10
     13     14     15
     18     19     20

>>
```

Observe que, neste caso, acessamos as posições que estavam nas linhas de 2 a 4, e nas colunas que estavam de 3 a 5. Então, conseguimos “recortar” um pedaço da matriz.

Exemplo 2:

Alterando parte de uma matriz:

```

>> A = [1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15; 16 17 18 19 20; 21 22 23 24 25]

A =

      1      2      3      4      5
      6      7      8      9     10
     11     12     13     14     15
     16     17     18     19     20
     21     22     23     24     25

>> A(2:4,3:5) = eye(3)

A =

      1      2      3      4      5
      6      7      1      0      0
     11     12      0      1      0
     16     17      0      0      1
     21     22     23     24     25

>>
```

Agora, nas posições desejadas, conseguimos inserir outra matriz menor.

- **Indexação condicional**

Ao invés de passarmos linhas e colunas de uma matriz, podemos passar como argumento uma matriz de mesmo tamanho, porém, preenchida com valores “true” ou “false”. Neste caso, o Matlab retornará um vetor coluna, com todos os valores onde, na matriz de argumento, havia um valor “true”.

Exemplo 1:

Retornar todos os elementos da matriz, que são maiores do que 12:

```
>> A = [1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15; 16 17 18 19 20; 21 22 23 24 25]

A =

     1     2     3     4     5
     6     7     8     9    10
    11    12    13    14    15
    16    17    18    19    20
    21    22    23    24    25

>> A(A>12)

ans =

    16
    21
    17
    22
    13
    18
    23
    14
    19
    24
    15
    20
    25

>>
```

Quando passamos como argumento a operação ($A>12$), o resultado dela é uma matriz do mesmo tamanho de A (neste caso 5x5) com valores true nas posições onde o valor era maior do que 13, e false nas restantes. O que o programa fez, foi retornar os valores de A que estavam nas posições onde a matriz do argumento apresentava um “true”.

Exemplo 2:

Retornar os elementos que obedecem a condições de outra matriz:

```
>> A = [1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15; 16 17 18 19 20; 21 22 23 24 25]

A =

     1     2     3     4     5
     6     7     8     9    10
    11    12    13    14    15
    16    17    18    19    20
    21    22    23    24    25
```

```
>> A(eye(5)==1)

ans =

     1
     7
    13
    19
    25

>>
```

Neste caso, fizemos o Matlab retornar os valores de A que estivessem nas mesmas posições onde os valores de uma matriz identidade 5x5 valessem 1, ou seja, pegamos os valores da diagonal de A .

Seção 1.4 - Funções básicas

- *Alocar vetores com valores seqüenciais*

- A partir da função linspace

$x = \text{linspace}(a,b)$ é gerado um vetor de 100 elementos entre os valores a e b .

$x = \text{linspace}(a,b,n)$ é gerado um vetor de n pontos entre os valores de a e b .

Exemplos:

Vetor em ordem crescente (10 pontos entre 1 e 2):

```
>> a = 1
a =
    1
>> b = 2
b =
    2
>> x=linspace(a,b,10)
x =
    1.0000    1.1111    1.2222    1.3333    1.4444    1.5556    1.6667
    1.7778    1.8889    2.0000
```

Vetor em ordem decrescente (10 pontos entre 2 e 1):

```
>> x = linspace(2,1,10)
x =
    2.0000    1.8889    1.7778    1.6667    1.5556    1.4444    1.3333
    1.2222    1.1111    1.0000
```

- Uma outra maneira de gerar vetores:

$x = a:\text{delta}:b$ é gerado um vetor de valores igualmente espaçados de delta de a até b .

Exemplos:

Vetor em ordem crescente (pontos entre 1 e 2 espaçados de 0.1):

```
>> x = 1:0.1:2  
  
x =  
  
    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000  
    1.7000    1.8000    1.9000    2.0000
```

Vetor em ordem decrescente (pontos entre 2 e 1 espaçados de 0.25):

```
>> a = 1  
  
a =  
  
    1  
  
>> b = 2  
  
b =  
  
    2  
  
>> x = b:-0.25:a  
  
x =  
  
    2.0000    1.7500    1.5000    1.2500    1.0000
```

– **Uma aplicação para os comandos**

Gerar vetores de tempo igualmente espaçados

```
>> dt = 0.001           %intervalo de amostragem  
  
dt =  
  
    1.0000e-003  
  
>> t0 = 0               %instante inicial  
  
t0 =  
  
    0  
  
>> tf = 3               %instante final  
  
tf =  
  
    3  
  
>> t = t0:dt:tf;        %vetor de tempo de 0 a 3 s com intervalo de amostragem 0.001s
```

– Uma utilidade para vetores

Com a palavra reservada *end* é possível delimitar o fim do vetor quando posições deste devem ser selecionadas.

Exemplo:

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

A =

     1     2     3     4     5     6     7     8     9

>> B = A(4:end)    %seleciono da posicao 4 ate o fim do vetor

B =

     4     5     6     7     8     9

>> C = A(6:end)    %seleciono da posicao 6 ate o fim do vetor

C =

     6     7     8     9
```

• *Alocação de Matrizes e Vetores*

Para determinadas aplicações, é necessário gerar matrizes ou vetores vazios para posteriormente preencher suas posições durante a execução de determinado algoritmo.

A alocação pode ser feita com o comando *matriz = zeros(nlinhas, ncolunas)*, gerando uma matriz com dimensões de nlinhas e ncolunas, com valores 0 em todas as suas posições.

Exemplo:

| | |
|---|--|
| <pre>>> A = zeros(3,2) A = 0 0 0 0 0 0 >> B = zeros(1,5) B = 0 0 0 0 0</pre> | <pre>>> C = zeros(4,3) C = 0 0 0 0 0 0 0 0 0 0 0 0 >> D = zeros(3,1) D = 0 0 0</pre> |
|---|--|

É possível alocar matrizes de dimensões 3D, 4D, etc. Na forma *matriz = zeros(dim1, dim2, dim3,*

dim4, etc.), temos dim1 = numero de linhas, dim2 = numero de colunas, e assim por diante.

Exemplo:

| | |
|---|--|
| <pre>>> A = zeros(2,2,2) %matriz 3D %2 camadas de matriz 2x2 A(:,:,1) = 0 0 0 0 A(:,:,2) = 0 0 0 0</pre> | <pre>>> A = zeros(2,2,2,2) %matriz 4D A(:,:,1,1) = 0 0 0 0 A(:,:,2,1) = 0 0 0 0 A(:,:,1,2) = 0 0 0 0 A(:,:,2,2) = 0 0 0 0</pre> |
|---|--|

– Gerando Matriz com valor 1 em todas as posições

A alocação pode ser feita com o comando *matriz = ones(nlinhas, ncolunas)*, gerando uma matriz com dimensões de n_{linhas} e n_{colunas} , com valores 1 em todas as suas posições.

Exemplo:

| | |
|--|---|
| <pre>>> A = ones(3,2) A = 1 1 1 1 1 1 >> B = ones(1,5) B = 1 1 1 1 1</pre> | <pre>>> C = ones(4,3) C = 1 1 1 1 1 1 1 1 1 1 1 1 >> D = ones(3,1) D = 1 1 1</pre> |
|--|---|

A grande vantagem desse comando é que, ao se multiplicar a matriz com valores 1 por uma variável, é obtida uma matriz com essa variável em todas as posições.

Exemplo:

| | |
|--|--|
| <pre>>> F = 3*ones(3,2)</pre> <pre>F =</pre> <pre> 3 3 3 3 3 3 </pre> | <pre>>> H = 5*ones(1,4)</pre> <pre>H =</pre> <pre> 5 5 5 5 </pre> |
|--|--|

– Uma aplicação para esse comando

É possível adicionar ou subtrair um valor de todas as posições de uma matriz sem a necessidade de correr as posições uma a uma.

Exemplo:

| | |
|--|--|
| <pre>>> A = [1 2 3;4 5 6; 7 8 9]</pre> <pre>A =</pre> <pre> 1 2 3 4 5 6 7 8 9 </pre> <pre>>> B = 2*ones(3,3)</pre> <pre>B =</pre> <pre> 2 2 2 2 2 2 2 2 2 </pre> | <pre>>> C = A+B</pre> <pre>C =</pre> <pre> 3 4 5 6 7 8 9 10 11 </pre> <p>%matriz A somada de 2 em todas as %posicoes</p> |
|--|--|

– Gerando Matriz Identidade

A alocação pode ser feita com o comando *identidade = eye(n)*, gerando uma matriz identidade com dimensão n x n.

Exemplo:

| | |
|---|--|
| <pre>>> A = eye(2)</pre> <pre>A =</pre> <pre> 1 0 0 1 </pre> <pre>>> B = eye(3)</pre> <pre>B =</pre> <pre> 1 0 0 0 1 0 0 0 1 </pre> | <pre>>> C = eye(4)</pre> <pre>C =</pre> <pre> 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 </pre> |
|---|--|

– Gerando matriz diagonal

Pode ser obtida a partir do comando $matriz = diag(vetor)$, resultando numa matriz contendo as posições do vetor de dimensão $n \times 1$ (vetor coluna) ou $1 \times n$ (vetor linha), na ordem em que aparecem no vetor. Nesse caso, o elemento da posição (1,0) ou (0,1) do vetor será posicionado na posição (1,1) da matriz diagonal, assim por diante. Os elementos de fora da diagonal serão 0.

Exemplo:

| | |
|---|---|
| <pre>>> v = [1 2 3] v = 1 2 3 >> X = diag(v) X = 1 0 0 0 2 0 0 0 3</pre> | <pre>>> u = [4; 9; 8] u = 4 9 8 >> Y = diag(u) Y = 4 0 0 0 9 0 0 0 8</pre> |
|---|---|

Com a notação $matriz = diag(vetor, k)$, a matriz resultante diagonal (dimensões $n \times n$) a partir do vetor é posicionada k posições acima ($k > 0$) ou abaixo ($k < 0$) da diagonal principal de uma matriz de dimensões $(n+k) \times (n+k)$.

Exemplo:

| | |
|--|--|
| <pre>>> v = [1 2 3] v = 1 2 3 >> X = diag(v, 2) X = 0 0 1 0 0 0 0 0 2 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0</pre> | <pre>>> X = diag(v, -2) X = 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 2 0 0 0 0 0 3 0 0</pre> |
|--|--|

– Obtendo a diagonal de uma matriz

Com a notação $vetor = diag(matriz\ quadrada)$, é obtido um vetor coluna com a diagonal da matriz quadrada dada.

Exemplo:

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

```
A =
```

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

```
>> M = diag(A)
```

```
M =
```

```
    1
    5
    9
```

Com a notação `vetor = diag(matriz não quadrada)`, é obtido um vetor coluna com elementos até a máxima posição onde é possível obter valor = `matriz(n,n)`

Exemplo:

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

```
A =
```

```
    1    2    3
    4    5    6
```

```
>> M = diag(A)
```

```
M =
```

```
    1
    5
```

```
>> B = [1 4; 2 5; 3 6]
```

```
B =
```

```
    1    4
    2    5
    3    6
```

```
>> L = diag(B)
```

```
L =
```

```
    1
    5
```

– Uma aplicação para os comandos

Remover de matriz resíduos numéricos indesejáveis resultantes de cálculos anteriores, sabendo-se que, pela teoria, a matriz resultante deve possuir somente a diagonal principal.

Exemplo:

```
%suponha que F foi resultado de
%sucessivos
%calculos matriciais que deixaram
%residuos %numericos de valor baixo na
%matriz
```

```
>> F = (10e-16)*ones(3,3)
```

```
F =
```

```
% a matriz H contem os residuos
%numericos de % F - o Matlab mostra
%somente 4 casas
%decimais
% acessando os elementos é possível
%verificar
```

```
>> H(1,1)
```

```
ans =
```

| | |
|----------------------|-------------|
| 1.0e-015 * | |
| 1.0000 1.0000 1.0000 | 1.0000 |
| 1.0000 1.0000 1.0000 | >> H(1,2) |
| 1.0000 1.0000 1.0000 | ans = |
| >> G = diag([1 2 3]) | 1.0000e-015 |
| G = | >> H(1,3) |
| 1 0 0 | ans = |
| 0 2 0 | 1.0000e-015 |
| 0 0 3 | >> H(2,3) |
| >> H = F+G | ans = |
| H = | 1.0000e-015 |
| 1.0000 0.0000 0.0000 | |
| 0.0000 2.0000 0.0000 | |
| 0.0000 0.0000 3.0000 | |

Para remover esse valores indesejáveis, sabendo-se previamente que a matriz H deve ser diagonal, realizamos, finalmente:

| | |
|----------------------|------------------------------|
| >> H = diag(diag(H)) | %onde observamos que, agora, |
| H = | %todas as posicoes sao zero |
| 1.0000 0 0 | >> H(1,2) |
| 0 2.0000 0 | ans = |
| 0 0 3.0000 | 0 |
| | >> H(2,3) |
| | ans = |
| | 0 |
| | >> H(1,3) |
| | ans = |
| | 0 |

– Tamanho de Vetores e Matrizes

O comando *size(matriz)* retorna o número de linhas e colunas da matriz.

Exemplo:

```
>> M = zeros(10,300);
>> [nlin,ncol] = size(M) %obtenho o numero de linhas e colunas separadamente
```

```

nlin =
    10

ncol =
    300

>> s = size(M)           %obtenho linhas e colunas na forma de um vetor s
s =
    10    300             %na primeira posicao temos o numero de linhas
                           %na segunda posicao temos o numero de colunas

```

Uma variação do comando *size(matriz,dim)* retorna a quantidade de posições na dimensão (*dim*) desejada. No caso de uma matriz bidimensional, *dim* = 1 indica o número de linhas, enquanto que *dim* = 2 indica o número de colunas. Para o caso tridimensional, *dim* = 3 representa o número de camadas e assim, sucessivamente. A variável *dim* seleciona o valor no vetor de dimensões a ser calculado pela função.

Exemplo:

| | |
|---|--|
| <pre> >> A = zeros(2,3,4) %matriz 3D %camada 1 - matriz 2 x 3 A(:, :, 1) = 0 0 0 0 0 0 %camada 2 - matriz 2 x 3 A(:, :, 2) = 0 0 0 0 0 0 %camada 3 - matriz 2 x 3 A(:, :, 3) = 0 0 0 0 0 0 %camada 4 - matriz 2 x 3 A(:, :, 4) = 0 0 0 0 0 0 </pre> | <pre> >> lin = size(A,1) %numero de linhas lin = 2 >> col = size(A,2) %numero de colunas col = 3 >> cam = size(A,3) %numero de camadas cam = 4 >> v = size(A) %vetor de dimesoes v = 2 3 4 </pre> |
|---|--|

– Uma aplicação para o comando

Pode-se utilizar esse comando para alocar matrizes com dimensões iguais a uma matriz original dada.

Exemplo:

| | |
|--|--|
| <pre>>> M = [1 2 3;4 5 6;7 8 9]</pre> <p>M =</p> <pre> 1 2 3 4 5 6 7 8 9</pre> <pre>>> B = zeros(size(M))</pre> <p>B =</p> <pre> 0 0 0 0 0 0 0 0 0</pre> | <pre>>> A = ones(size(M))</pre> <p>A =</p> <pre> 1 1 1 1 1 1 1 1 1</pre> |
|--|--|

Obtendo o número de linhas e colunas é possível correr as posições da matriz para realizar alguma operação desejada.

Exemplo:

| | |
|--|---|
| <pre>>> M = [1 2 3;4 5 6;7 8 9];</pre> <pre>A = zeros(size(M));</pre> <pre>for lin = 1:size(M,1)</pre> <pre> for col = 1:size(M,2)</pre> <pre> A(lin,col) = M(lin,col)^2;</pre> <pre> end</pre> <pre>end</pre> <pre>>> A</pre> <p>A =</p> <pre> 1 4 9 16 25 36 49 64 81</pre> | <pre>%aloco a matriz A com a mesma dimensão da matriz M</pre> <pre>%corro linhas</pre> <pre>%corro colunas</pre> <pre>%elevo elemento da matriz ao seu quadrado</pre> |
|--|---|

– **Ordenação de elementos de matrizes/vetores**

O comando *sort(A)* ordena as colunas de uma matriz de dimensão n x m em ordem crescente. A variação do comando *sort(A,1)* ordena as colunas, enquanto que *sort(A,2)* ordena as linhas da matriz. A ordenação em ordem decrescente é feita por *sort(A,'descend')*.

Exemplo:

| | |
|--|---|
| <pre>>> A = [4 1 0; 7 6 9; 2 3 5] A = 4 1 0 7 6 9 2 3 5 >> B = sort(A) B = 2 1 0 4 3 5 7 6 9 >> C = sort(A,1) %equivale a sort(A) C = 2 1 0 4 3 5 7 6 9</pre> | <pre>%ordeno colunas em ordem decrescente >> F = sort(A,'descend') F = 7 6 9 4 3 5 2 1 0 %ordeno linhas em ordem crescente >> D = sort(A,2) D = 0 1 4 6 7 9 2 3 5 %ordeno linhas em ordem decrescente >> E = sort(A,2,'descend') E = 4 1 0 9 7 6 5 3 2</pre> |
|--|---|

Para vetores *sort(vetor)* ordena os elementos em ordem crescente. Para *sort(vetor, 'descend')*, os elementos são ordenados de forma decrescente.

Exemplo:

```
>> vet = [1 3 2 5 6 4 0.1 2]
vet =
    1.0000    3.0000    2.0000    5.0000    6.0000    4.0000    0.1000
    2.0000

>> D = sort(vet)
D =
    0.1000    1.0000    2.0000    2.0000    3.0000    4.0000    5.0000
    6.0000

>> E = sort(vet,'descend')
E =
    6.0000    5.0000    4.0000    3.0000    2.0000    2.0000    1.0000
    0.1000
```

– Inverter colunas de uma matriz

O comando *fliplr(A)* – *flip matrices left right* - realiza a inversão das colunas de uma matriz da esquerda para a direita. Em vetor linha, esse comando inverte a posição dos valores dos seus elementos.

Exemplo:

| | |
|---|---|
| <pre>>> A = [4 1 0; 7 6 9; 2 3 5] A = 4 1 0 7 6 9 2 3 5 >> fliplr(A) ans = 0 1 4 9 6 7 5 3 2</pre> | <pre>>> vet = [1 3 2 5 6 4 0.1 2] vet = 1.0000 3.0000 2.0000 5.0000 6.0000 4.0000 0.1000 2.0000 >> fliplr(vet) ans = 2.0000 0.1000 4.0000 6.0000 5.0000 2.0000 3.0000 1.0000</pre> |
|---|---|

– Inverter linhas de uma matriz

O comando *flipud(A)* – *flip matrices up down* - realiza a inversão das linhas de uma matriz da de baixo para cima. Em vetor coluna, esse comando inverte a posição dos valores dos seus elementos.

Exemplo:

| | |
|---|---|
| <pre>>> A = [4 1 0; 7 6 9; 2 3 5] A = 4 1 0 7 6 9 2 3 5 >> flipud(A) ans = 2 3 5 7 6 9 4 1 0</pre> | <pre>>> vet = [2;4;5;1;3;7] vet = 2 4 5 1 3 7 >> flipud(vet) ans = 7 3 1 5 4 2</pre> |
|---|---|

– Limpar variável utilizada

Utilizando-se o comando *clear* em frente à uma variável do programa, ela é automaticamente desalocada da memória do programa.

Exemplo:

Variáveis no *Workspace* do programa:

| Name ▲ | Value | Class |
|--------|---------------------|--------|
| A | [4 1 0;7 6 9;2 3 5] | double |
| B | [2 1 0;4 3 5;7 6 9] | double |
| C | [4 1 0;9 7 6;5 3 2] | double |
| D | [0.1 1 2 2 3 4 5 6] | double |
| E | [6 5 4 3 2 2 1 0.1] | double |
| F | [7 6 9;4 3 5;2 1 0] | double |
| G | [1 0 0;0 2 0;0 0 3] | double |
| H | [1 0 0;0 2 0;0 0 3] | double |
| L | [1;5] | double |
| M | [1 2 3;4 5 6;7 8 9] | double |
| ans | [7;3;1;5;4;2] | double |
| cam | 4 | double |
| col | 3 | double |
| lin | 2 | double |
| m | 1 | double |
| n | 3 | double |
| ncol | 300 | double |
| nlin | 10 | double |
| s | [10 300] | double |
| v | [2 3 4] | double |
| vet | [2;4;5;1;3;7] | double |

Figura 16: *Workspace* com variáveis

```
>> clear A
>> clear B
>> clear F
>> clear G
>> clear vet
```

Variáveis no *Workspace* após os comandos:

| Name ▲ | Value | Class |
|--------|---------------------|--------|
| C | [4 1 0;9 7 6;5 3 2] | double |
| D | [0.1 1 2 2 3 4 5 6] | double |
| E | [6 5 4 3 2 2 1 0.1] | double |
| H | [1 0 0;0 2 0;0 0 3] | double |
| L | [1;5] | double |
| M | [1 2 3;4 5 6;7 8 9] | double |
| ans | [7;3;1;5;4;2] | double |
| cam | 4 | double |
| col | 3 | double |
| lin | 2 | double |
| m | 1 | double |
| n | 3 | double |
| ncol | 300 | double |
| nlin | 10 | double |
| s | [10 300] | double |
| v | [2 3 4] | double |

Figura 17: Workspace após a limpeza de variáveis

Na tentativa de acessá-las novamente, observamos que todas foram apagadas:

```
>> A
??? Undefined function or variable 'A'.

>> B
??? Undefined function or variable 'B'.

>> F
??? Undefined function or variable 'F'.

>> G
??? Undefined function or variable 'G'.

>> vet
??? Undefined function or variable 'vet'.
```

– Limpar todas as variáveis do Workspace

Para desalocar toda a memória do *Workspace*, deve ser utilizado o comando *clear all*.

Exemplo:

A partir do *Workspace* resultante do exemplo anterior, temos:

```
>> clear all
```


Temos que o *Workspace* está completamente vazio:

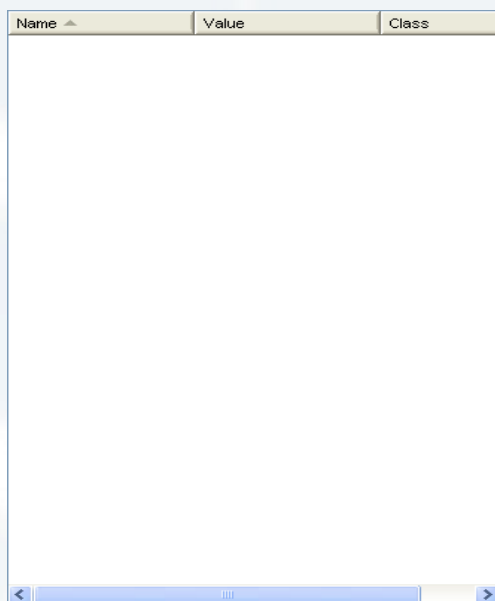


Figura 18: Workspace após comando “clear all”

– Limpar prompt de comando do Matlab

Ao se utilizar o Matlab, podemos observar que o prompt de comando fica repleto de comandos anteriores:

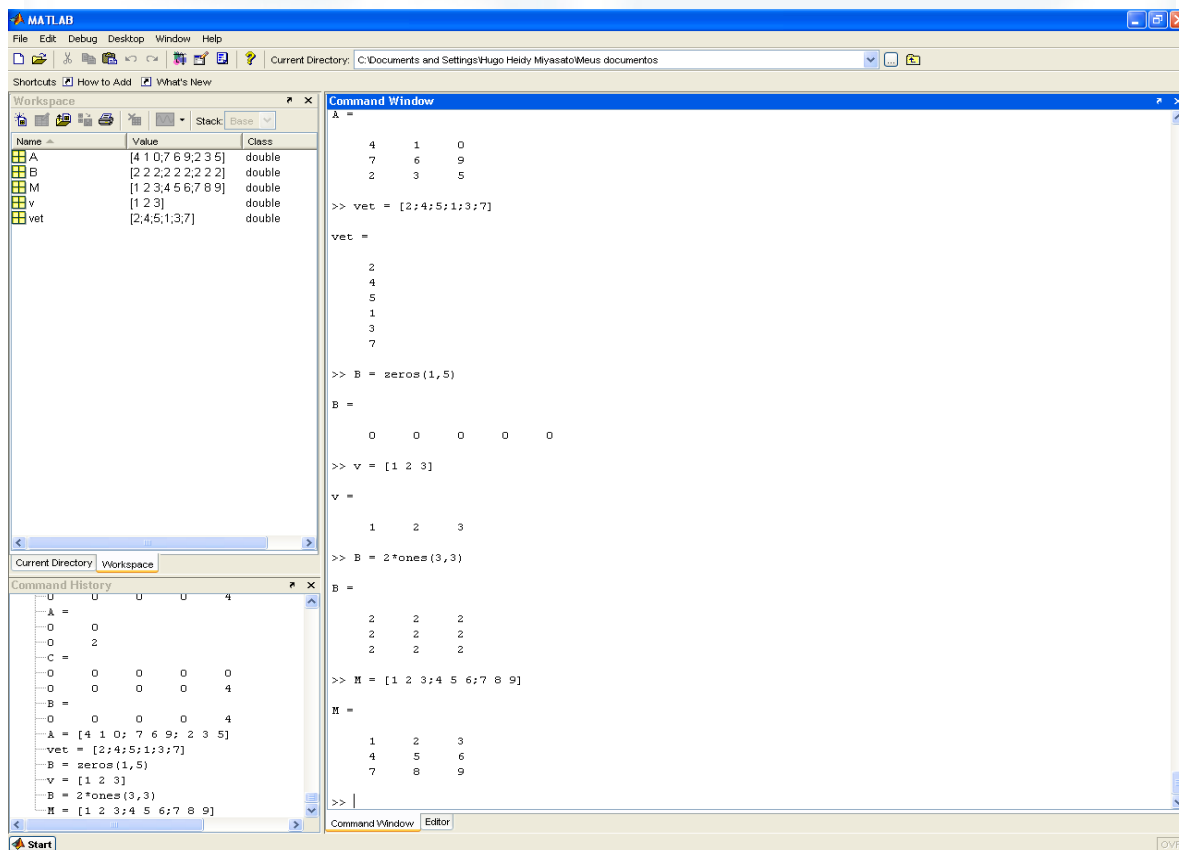


Figura 19: Prompt de comando com mensagens

Para melhorar a visualização existe o comando `clc`, que remove o texto que está no *prompt*:

```
>> clc
```

Após este comando, a *Command Window* está limpa, porém sem qualquer alteração nas variáveis presentes no *workspace* ou no *Command History*:

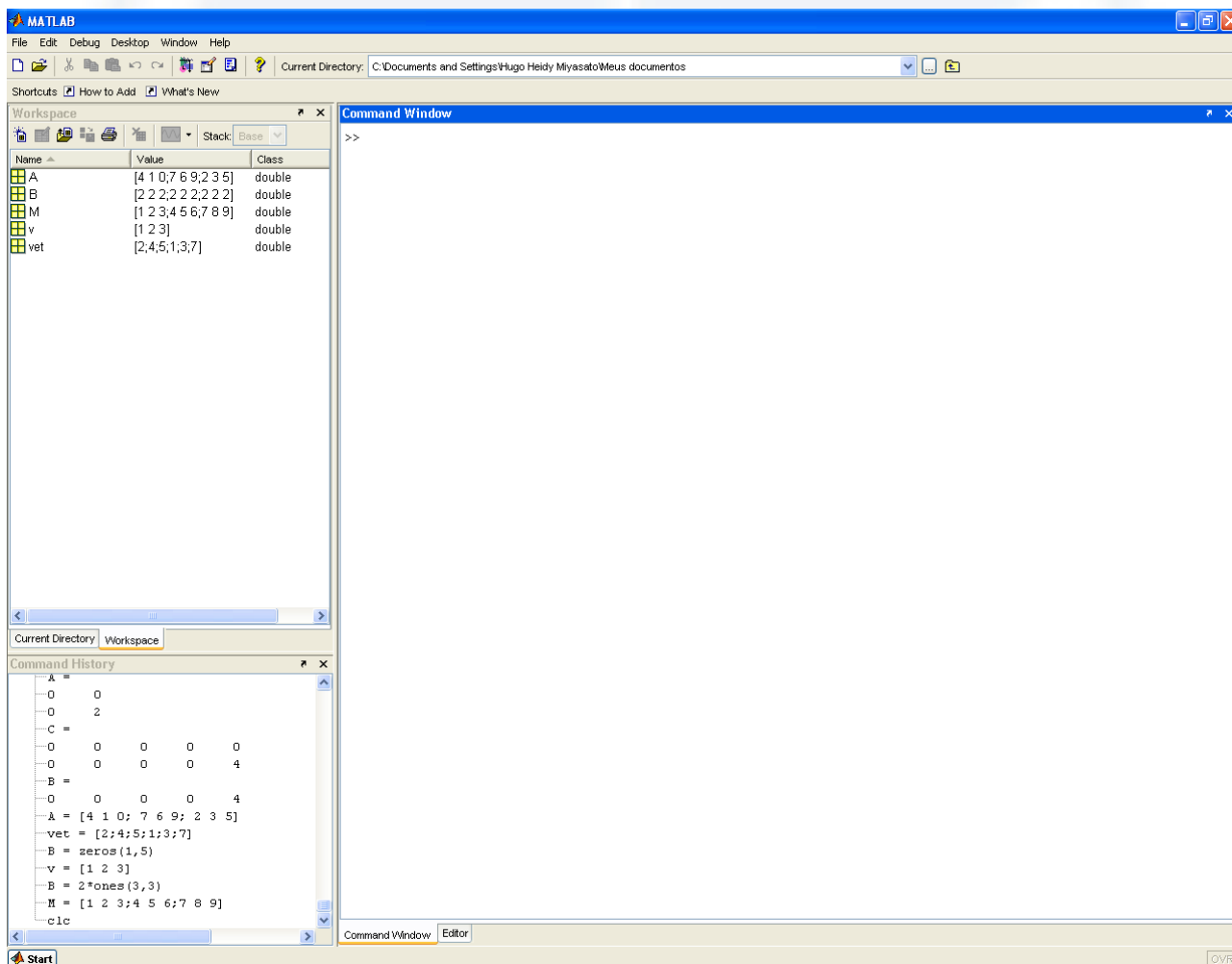


Figura 20: Prompt de comando após comando “clc”

Dessa maneira, todas as variáveis anteriores podem ser recuperadas sem qualquer alteração nos seus valores.

Seção 1.5 - M-file

– Barra de ferramentas do editor



Figura 21: Barra de ferramentas do editor de texto

– Principais botões/comandos da barra de ferramentas

| Ícone | Função | Atalho |
|---|-------------------------|----------|
|  | Novo M-file | Ctrl + N |
|  | Abrir arquivo | Ctrl + O |
|  | Salvar arquivo | Ctrl + S |
|  | Recortar | Ctrl + X |
|  | Copiar | Ctrl + C |
|  | Colar | Ctrl + V |
|  | Desfazer | Ctrl + Z |
|  | Refazer | Ctrl + Y |
|  | Imprimir | Ctrl + P |
|  | Encontrar e substituir | Ctrl + F |
|  | Salvar e rodar programa | F5 |

– Compilar arquivo no editor

Diferentemente de outras linguagens de programação (C,C++,JAVA,etc.), em Matlab, ao ser salvo o arquivo, não existe a necessidade que o seu código seja compilado para que o programa funcione.

– Inserir comentário

Em Matlab, o símbolo “%” indica comentário. As linhas indicadas por esse símbolo não são levadas em conta para o funcionamento do programa, sendo somente um recurso para a documentação e organização do código-fonte. Com isso, o programador pode descrever o que ocorre em determinado trecho ou em todo o programa, permitindo que outras pessoas possam realizar modificações futuras.

Exemplo:

```
%este trecho de programa realiza a alocação de matriz para um calculo
A = zeros(300,300);
```

– Comentar trecho de programa

O comando ctrl+R transforma em comentário o trecho de código selecionado no editor, enquanto que ctrl+T transforma o trecho comentado novamente em código fonte para o programa.

Exemplo:

| | |
|---|---|
| <p>1 – Código no editor</p> <pre>1 %programa teste 2 3 - M = [1 2 3;4 5 6;7 8 9] 4 - B = 2*ones(3,3) 5 - v = [1 2 3] 6 - A = [4 1 0; 7 6 9; 2 3 5] 7 - C = sort(A,1)</pre> | <p>2 – Seleção do trecho de código com o mouse</p> <pre>1 %programa teste 2 3 - M = [1 2 3;4 5 6;7 8 9] 4 - B = 2*ones(3,3) 5 - v = [1 2 3] 6 - A = [4 1 0; 7 6 9; 2 3 5] 7 - C = sort(A,1)</pre> |
| <p>3 – comando ctrl+R comenta trecho selecionado</p> <pre>1 %programa teste 2 3 % M = [1 2 3;4 5 6;7 8 9] 4 % B = 2*ones(3,3) 5 % v = [1 2 3] 6 - A = [4 1 0; 7 6 9; 2 3 5] 7 - C = sort(A,1)</pre> | <p>4 – seleção de trecho comentado</p> <pre>1 %programa teste 2 3 % M = [1 2 3;4 5 6;7 8 9] 4 % B = 2*ones(3,3) 5 % v = [1 2 3] 6 - A = [4 1 0; 7 6 9; 2 3 5] 7 - C = sort(A,1)</pre> |

4 – comando ctrl+T transforma comentário em código

```
1 %programa teste
2
3 - M = [1 2 3;4 5 6;7 8 9]
4 - B = 2*ones(3,3)
5 - v = [1 2 3]
6 - A = [4 1 0; 7 6 9; 2 3 5]
7 - C = sort(A,1)
```

• Criação de Script

Os scripts são arquivos que contêm comandos do Matlab sequenciais, permitindo ao programador gravar as operações ao invés de ter que digitá-las no *Command Window* toda vez que seja necessário.

Exemplo:

1 – Criar novo M-file

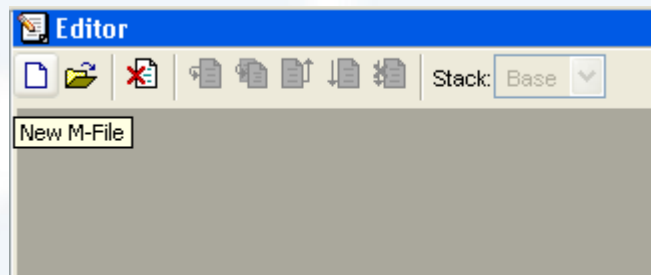


Figura 21: Botão para criar um novo M-file

2 – Novo M-file criado

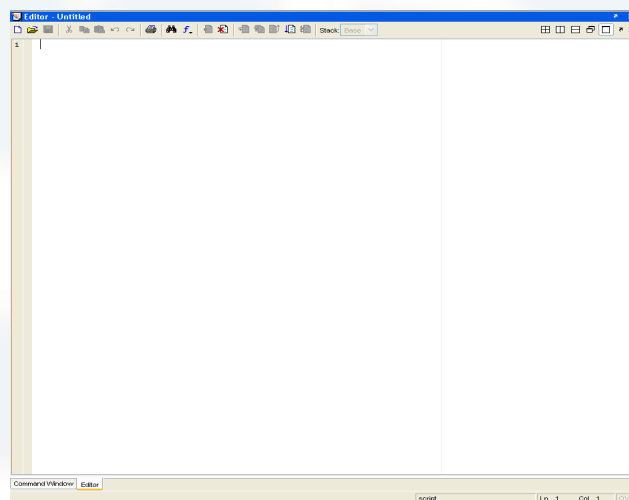


Figura 22: Criação do novo M-file

3- Digitar comandos desejados

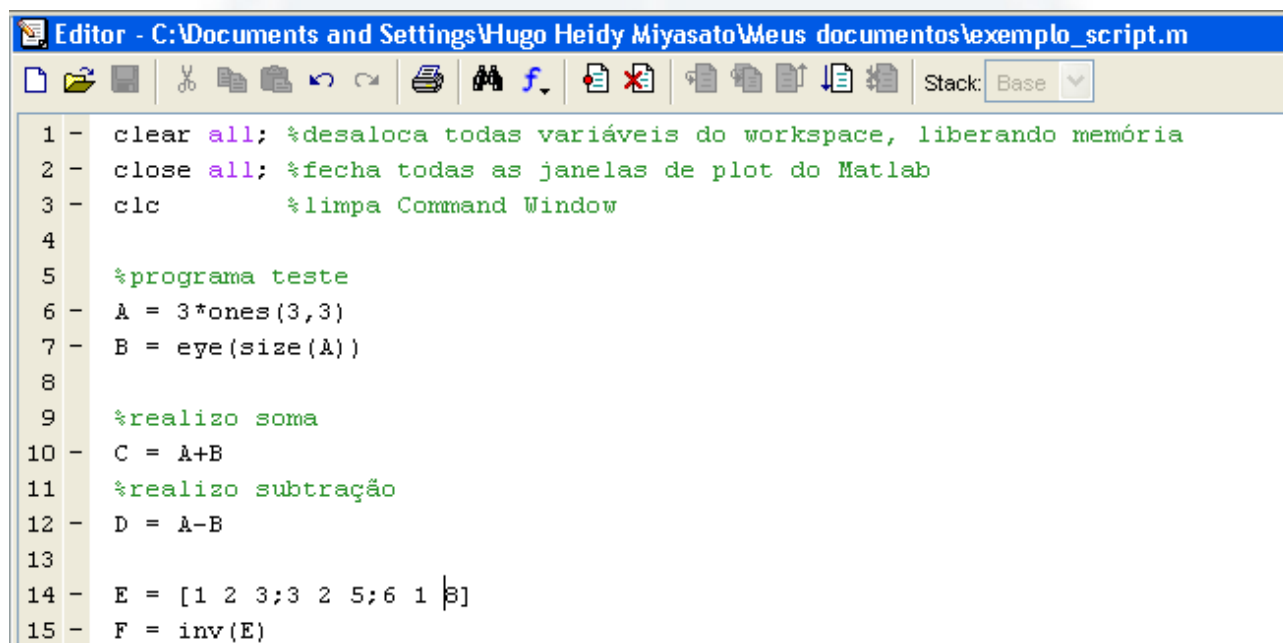


Figura 23: M-file com os códigos produzidos

Nota: é recomendável a utilização dos comandos *clear all*, *close all*, *clc* no início do código pois o Matlab apresentará somente os resultados referentes ao script que será rodado, tanto no *Command Window* quanto no *Workspace*, facilitando a visualização dos mesmos e evitando confusão com outro *script* que seja utilizado anterior ou posteriormente.

4 – Salvar arquivo

Para poder rodar o script criado é necessário salvá-lo anteriormente.

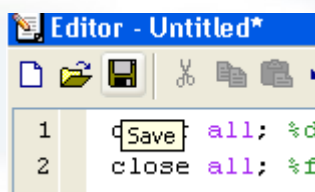


Figura 24: Botão para salvar o M-file

Alternativamente, com o botão F5 (*Save and Run*) é requisitado o nome e local para salvar o arquivo antes de sua execução:

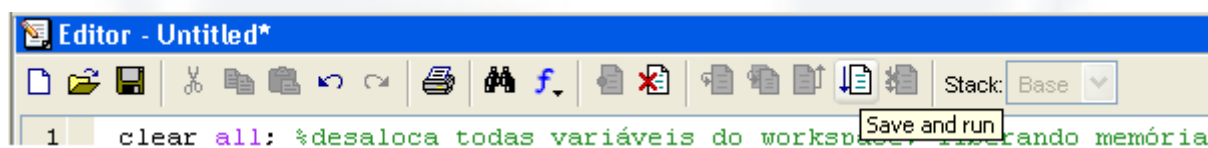


Figura 25: Botão para salvar e executar o M-file

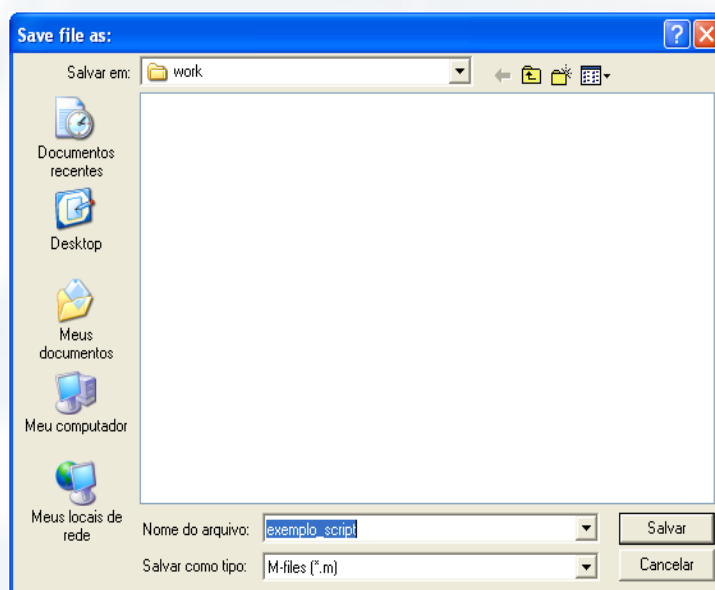


Figura 26: Janela para salvar o M-file

Surgindo a janela para determinar o nome e o local para salvar o arquivo. O local padrão é uma pasta denominada *work*, podendo ser alterada:

5 – Rodar arquivo

Após salvar o arquivo, F5 roda o arquivo:



Figura 27: Botão para executar o arquivo

Caso esteja se utilizando outro diretório, pode surgir a janela apresentada a seguir, para que o *Current Directory* seja a pasta onde está salvo o *M-file*.

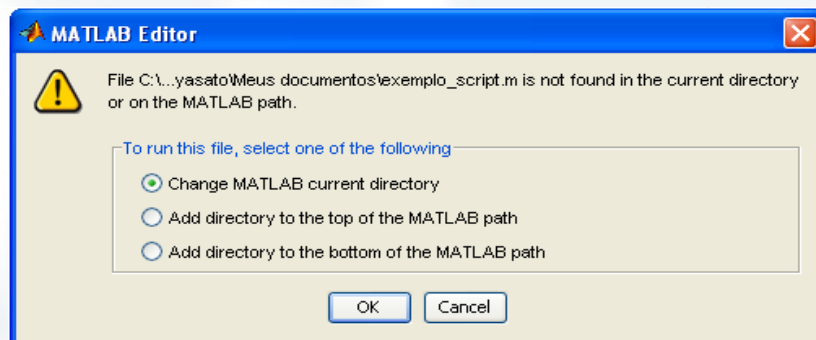
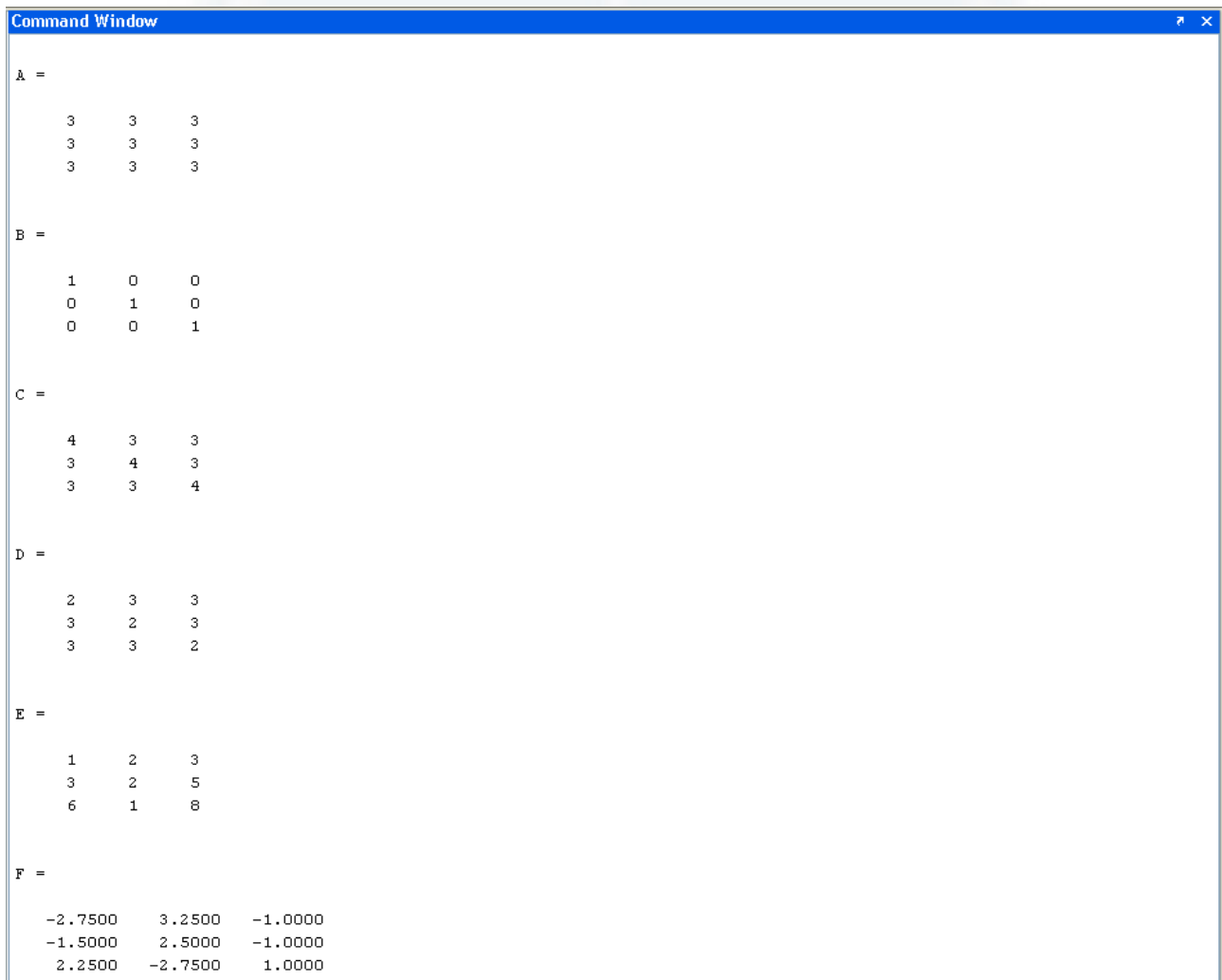


Figura 28: Tela para confirmação de mudança de diretório de trabalho

Clicando OK, o código é executado:



```
Command Window

A =

     3     3     3
     3     3     3
     3     3     3

B =

     1     0     0
     0     1     0
     0     0     1

C =

     4     3     3
     3     4     3
     3     3     4

D =

     2     3     3
     3     2     3
     3     3     2

E =

     1     2     3
     3     2     5
     6     1     8

F =

    -2.7500    3.2500   -1.0000
    -1.5000    2.5000   -1.0000
     2.2500   -2.7500    1.0000
```

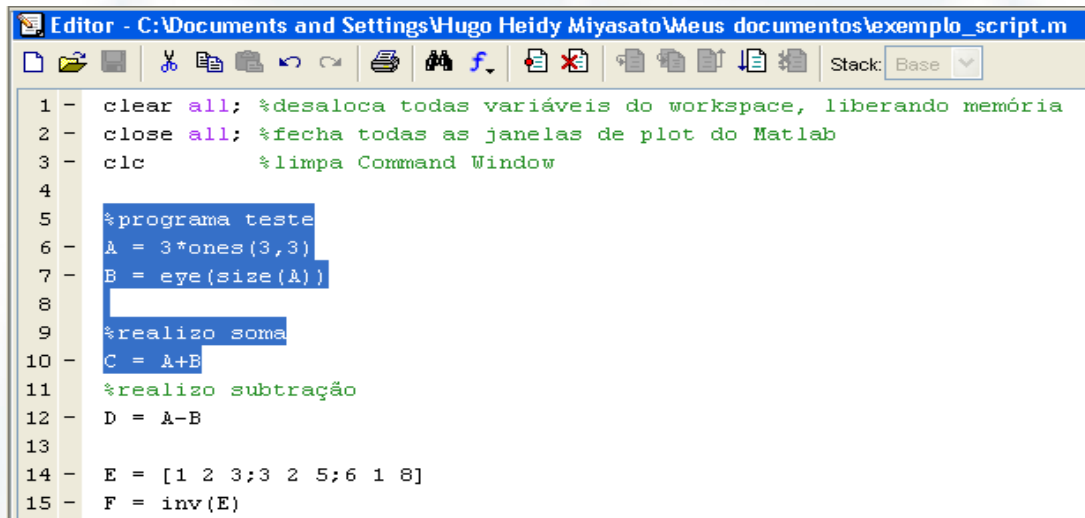
Figura 29: Execução do código do M-file

– Rodar somente um trecho do código-fonte

Com F9 é possível rodar trechos do programa como se ele tivesse sido digitado diretamente no *Command Window*.

Exemplo:

1 – Seleção de trecho de código:

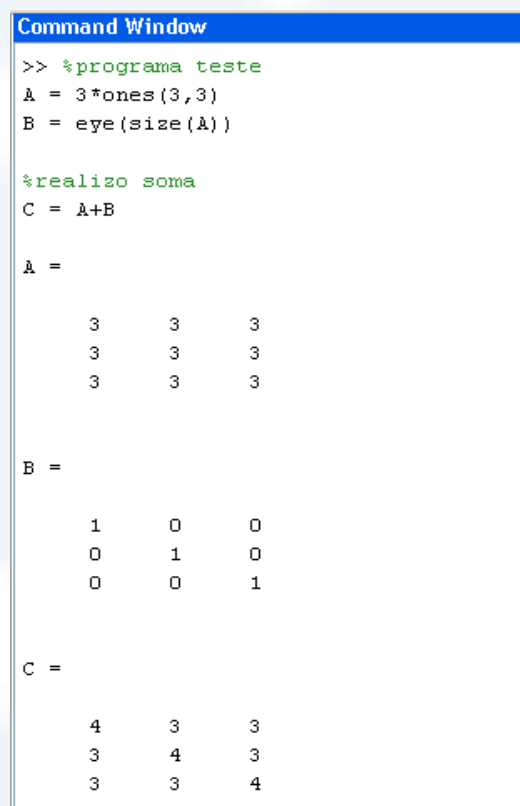


```

1 - clear all; %desaloca todas variáveis do workspace, liberando memória
2 - close all; %fecha todas as janelas de plot do Matlab
3 - clc      %limpa Command Window
4
5 - %programa teste
6 - A = 3*ones(3,3)
7 - B = eye(size(A))
8
9 - %realizo soma
10 - C = A+B
11 - %realizo subtração
12 - D = A-B
13
14 - E = [1 2 3; 3 2 5; 6 1 8]
15 - F = inv(E)
    
```

Figura 30: Seleção do trecho de código a ser executado

2 – Após apertar o botão F9, o código selecionado é executado no Command Window:



```

>> %programa teste
A = 3*ones(3,3)
B = eye(size(A))

%realizo soma
C = A+B

A =

     3     3     3
     3     3     3
     3     3     3

B =

     1     0     0
     0     1     0
     0     0     1

C =

     4     3     3
     3     4     3
     3     3     4
    
```

Figura 31: Execução do código selecionado

- **Criando Funções**

A sintaxe das funções é:

$$\text{function}[\text{out1}, \text{out2}, \text{out3}, \dots, \text{outN}] = \text{nome_funcao}(\text{in1}, \text{in2}, \text{in3}, \dots, \text{inM})$$

onde *out* são os resultados que serão fornecidos pela função de nome *nome_funcao*. Os parâmetros de entrada são indicados por *in*.

Exemplo:

Função que resolve polinômio de segundo grau utilizando a fórmula de Bhaskara

```
function[x1,x2] = Bhaskara(a,b,c)

%esta funcao resolve um polinomio de 2° grau do tipo
%a*x^2+b*x+c = 0

aux = sqrt(b^2-4*a*c);

%raizes do polinomio
x1 = (-b+aux)/(2*a);
x2 = (-b-aux)/(2*a);
```

Nota: O M-file deve ser salvo com o mesmo nome dado para a função

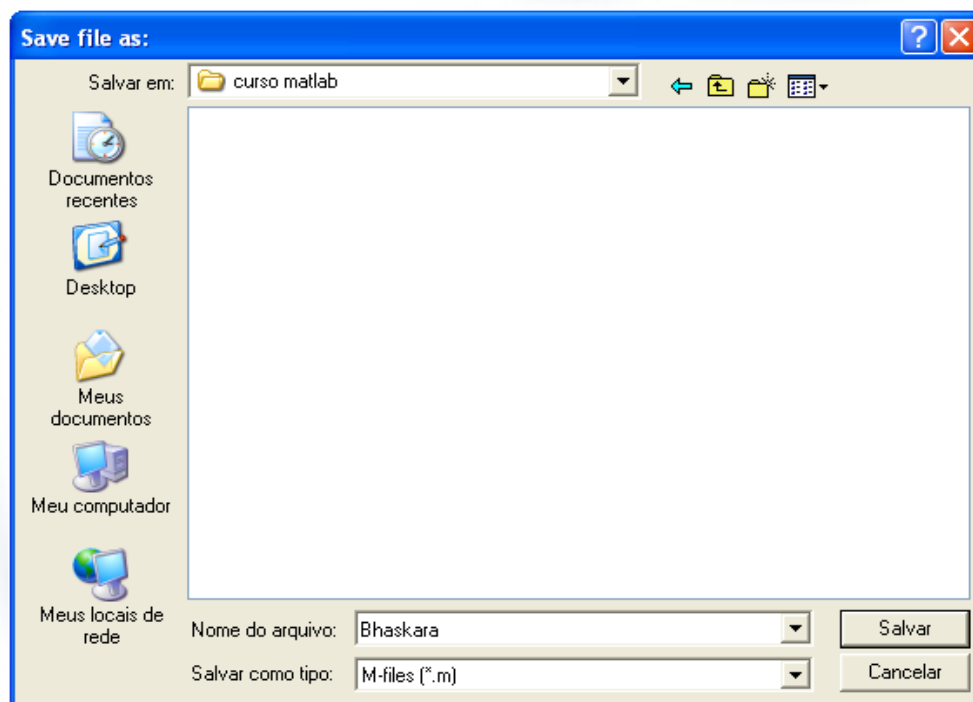


Figura 32: M-file sendo salvo com o mesmo nome da function

Caso o *Current Directory* não seja a pasta onde está salva a função, ela não pode ser utilizada no

Command Window:

```
>> Bhaskara(1,2,1)
??? Undefined command/function 'Bhaskara'.

>> Bhaskara(2,4,4)
??? Undefined command/function 'Bhaskara'.
```

Mas, caso o *Current Directory* esteja corretamente configurado, a função pode ser utilizada:

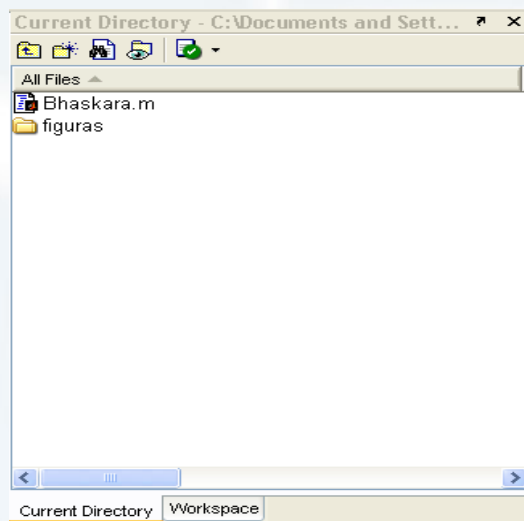


Figura 33: Função salva no Current Directory

Command Window:

```
Command Window

>> [x1,x2]=Bhaskara(1,-4,3)

x1 =

     3

x2 =

     1

>> [x1,x2]=Bhaskara(1,-2,1)

x1 =

     1

x2 =

     1
```

Figura 34: Função gerada sendo chamada no command prompt

A função criada também pode ser utilizada num *Script* do Matlab:

```
clear all;
close all;
clc;

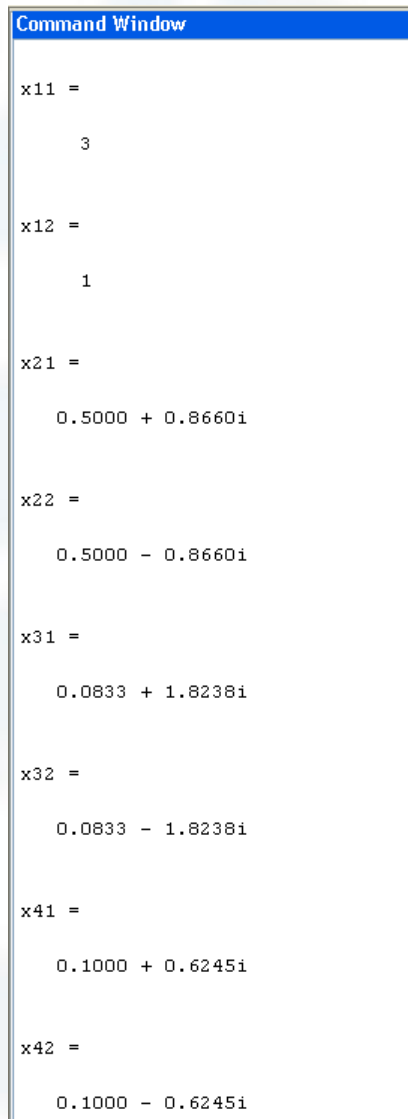
%polinomio 1
[x11,x12]=Bhaskara(1,-4,3)

%polinomio 2
[x21,x22]=Bhaskara(1,-1,1)

%polinomio 3
[x31,x32]=Bhaskara(3,-0.5,10)

%polinomio 4
[x41,x42]=Bhaskara(5,-1,2)
```

Command Window:



```
Command Window

x11 =

     3

x12 =

     1

x21 =

    0.5000 + 0.8660i

x22 =

    0.5000 - 0.8660i

x31 =

    0.0833 + 1.8238i

x32 =

    0.0833 - 1.8238i

x41 =

    0.1000 + 0.6245i

x42 =

    0.1000 - 0.6245i
```

Figura 35: Função gerada sendo chamada por meio de um script

É possível incluir a função no mesmo arquivo que o *Script* que o utiliza, porém o Script precisa ser transformado numa função sem parâmetros de entrada ou saída. Desta maneira, com F5, o código é executado normalmente.

Exemplo:

```
function [] = teste_function()

clear all; %desaloca todas variáveis do workspace, liberando memória
close all; %fecha todas as janelas de plot do Matlab
clc       %limpa Command Window

%polinomio 1
[x11,x12]=Bhaskara(1,-4,3)

%polinomio 2
[x21,x22]=Bhaskara(1,-1,1)

%polinomio 3
[x31,x32]=Bhaskara(3,-0.5,10)

%polinomio 4
[x41,x42]=Bhaskara(5,-1,2)

end

%função auxiliar inserida no fim do código
function [x1,x2] = Bhaskara(a,b,c)

%esta funcao resolve um polinomio de 2° grau do tipo
%a*x^2+b*x+c = 0

aux = sqrt(b^2-4*a*c);

%raizes do polinomio
x1 = (-b+aux)/(2*a);
x2 = (-b-aux)/(2*a);

end
```

Seção 1.6 – Palavras reservadas

A palavras reservadas a serem abordadas nessa seção são *if*, *else*, *elseif*, *switch*, *case*, *otherwise*, *break*, *continue*, *for*, *while* e *end*.

– If

A estrutura do comando if é dada por:

```
if expressão lógica
    comandos caso a expressão lógica seja verdadeira
end
```

Na expressão lógica, podem ser utilizados os operadores (&&, >, <, ==, ~=, <=, >=, etc.) para realizar comparações entre variáveis ou comparações entre expressões. Logo após os comandos, caso a expressão seja verdadeira, existe a palavra *end*, indicando o fim dos comandos referentes à comparação if anterior, sendo equivalente ao fechamento de chaves “{}” em outras linguagens de programação (C, C++, JAVA).

Exemplo:

```
clear all
close all
clc

%variaveis
a = 1;
b = 2;
c = 3;
d = 4;

if (a<b) && (c<d) %expressao logica
    %comandos caso expressao seja verdadeira
    %e mostrada uma string/frase (vetor de palavras) no Command Window com o comando disp
    %os numeros sao convertidos em caracteres com o comando int2str
    disp([int2str(a) ' é menor que ' int2str(b) ' E ' int2str(c) ' é menor que ' int2str(d)])
end %fim dos comandos caso expressao logica seja verdadeira

if (a<b) || (c>d) %expressao logica
    %comandos caso expressao seja verdadeira
    disp([int2str(a) ' é menor que ' int2str(b) ' OU ' int2str(c) ' é maior que ' int2str(d)])
end %fim dos comandos caso expressao logica seja verdadeira
```

Saída no Command Window:

```
1 é menor que 2 E 3 é menor que 4
1 é menor que 2 OU 3 é maior que 4
```

O comando if também pode tratar a comparação entre elementos de matrizes e vetores. Reforçando-se que, na expressão lógica também podem existir elementos numéricos e operações como soma, divisão, multiplicação, subtração, etc:

Exemplo:

```

clear all;
close all;
clc;

%teste if
A = 2
B = [1 2 3 4]
C = [4 5;6 7]
D = [10 11 12 13 14]
E = [10 11;12 13]

%variavel
if A>1
    disp('A é maior que 1') %demonstro resultado da comparação no Command Window
end

%vetor - verifico se todos elementos são menores que 5
if B<5
    disp(['Todos elementos do vetor B são menores que 5'])
end

%matriz - verifico se todos os elementos são maiores que 3
if C>3
    disp(['Todos elementos da matriz C são maiores que 3'])
end

%comparo numero de 2 elementos
if size(B,2)<size(D,2)
    disp('Numero de colunas do vetor B é menor que o numero de colunas do vetor D')
end

%incluo operação de soma na expressão lógica
%verifico se todos os elementos de C são menores todos elementos da matriz E somados de %1
if C<E+1
    disp('Todos elementos da matriz C são menores que os elementos da matriz E+1')
end

```

Saída no Command Window:

```

A é maior que 1
Todos elementos do vetor B são menores que 5
Todos elementos da matriz C são maiores que 3
Numero de colunas do vetor B é menor que o numero de colunas do vetor D
Todos elementos da matriz C são menores que os elementos da matriz E+1

```

– Elseif

Esse recurso do Matlab permite uma melhor visualização quando ocorrem muitas cláusulas *if* encadeadas, reduzindo o tamanho do código e a possibilidade de erros de digitação ou de posicionamento das palavras de comparação no código. Com *elseif* não é necessário colocar as cláusulas *end* das comparações internas.

Exemplo:

| SEM <i>elseif</i> | COM <i>elseif</i> |
|--|--|
| <pre> clear all close all clc %variaveis a = 1; b = 2; c = 3; d = 4; %codigo sem elseif if a==0 disp('a é igual a 0') else if b==a disp('a é igual a b') else if c>d disp('c é maior que d') else disp('a é diferente de 0') disp('a é diferente de b') disp('c é menor que d') end end end end </pre> <p>Saída no Command Window:</p> <pre> a é diferente de 0 a é diferente de b c é menor que d </pre> | <pre> clear all close all clc %variaveis a = 1; b = 2; c = 3; d = 4; %codigo com elseif if a==0 disp('a é igual a 0') elseif b==a disp('a é igual a b') elseif c>d disp('c é maior que d') else disp('a é diferente de 0') disp('a é diferente de b') disp('c é menor que d') end </pre> <p>Saída no Command Window:</p> <pre> a é diferente de 0 a é diferente de b c é menor que d </pre> |

– Switch

A cláusula switch é utilizada para tornar o código mais claro evitando a utilização de if encadeados ou de elseif, tornando o código mais claro e limpo. A sua sintaxe é dada por:

switch expressao

case valor1

comandos para o caso da expressão ser igual a valor1

case valor2

comandos para o caso da expressão ser igual a valor2

...

case valor_n

comandos para o caso da expressão ser igual a valor_n

otherwise

comandos para o caso da expressão ser diferente de todos os casos anteriores

end

Exemplo:

| Com <i>if</i> | Com <i>elseif</i> | Com <i>switch</i> |
|--|--|---|
| <pre>clear all close all clc cont = 3; %expressão com if if cont ==1 disp('cont vale 1') else if cont==2 disp('cont vale 2') else if cont ==3 disp('cont vale 3') else disp('cont não vale 1, 2 ou 3') end end end end</pre> | <pre>clear all close all clc cont = 3; %expressão com elseif if cont ==1 disp('cont vale 1') elseif cont==2 disp('cont vale 2') elseif cont ==3 disp('cont vale 3') else disp('cont não vale 1, 2 ou 3') end</pre> | <pre>clear all close all clc cont = 3; %expressão com switch switch cont case 1 disp('cont vale 1') case 2 disp('cont vale 2') case 3 disp('cont vale 3') otherwise disp('cont não vale 1, 2 ou 3') end</pre> |
| Saída no Command Window: | Saída no Command Window: | Saída no Command Window: |
| cont vale 3 | cont vale 3 | cont vale 3 |

Na expressão podem ser utilizados operandos tais como soma, multiplicação, divisão, subtração, etc.

Exemplo:

| |
|--|
| <pre>clear all close all clc a = 1 b = 2 c = 3 %expressão com switch switch a*b+c case 1 disp('a*b+c vale 1') case 2 disp('a*b+c vale 2') case 3 disp('a*b+c vale 3') otherwise disp(['a*b+c vale ' int2str(a*b+c)]) end</pre> |
| Saída no Command Window: |
| a*b+c vale 5 |

– **While**

É um laço de execução possuindo a seguinte estrutura:

```
while expressao
    comandos enquanto a expressao for verdadeira
end
```

Enquanto a expressão for verdadeira, os comandos contidos no laço serão executados. Dessa forma, é uma expressão útil para procedimentos repetitivos ou algoritmos expressos matematicamente.

Da mesma maneira que o if, podem ser utilizados os operadores (&&, >, <, ==, ~=, <=, >=, etc.) para realizar comparações entre variáveis ou comparações entre expressões, além de operações matemáticas (+, -, *, /, etc.).

Exemplo:

Cálculo dos termos de uma PG (Progressão Geométrica).

$$a_n = a_1 r^{n-1}$$

Onde,

a_1 – primeiro termo da PG

r – razão da PG

```
clear all
close all
clc

%calculo dos termos de uma P.G.

%primeiro termo
a1 = 1;
%razao da PG
r = 2
%total de termos considerados
n = 10
cont = 1
%vetor contendo os termos
a = zeros(1,n);
%carrego primeiro termo
a(1,1) = a1;

cont = 2;

%calculo os termos da P.G.
while cont<=n %expressao
    a(1,cont) = a1*r^cont;
    cont = cont+1;%incremento variavel cont
end

%mostro termos da PG
disp(a)
```

Command Window:

Columns 1 through 9

| | | | | | | | | |
|-----------|------|---|---|----|----|----|-----|-----|
| | 1 | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
| 512 | | | | | | | | |
| Column 10 | | | | | | | | |
| | 1024 | | | | | | | |

– Break

Com esse comando, é possível interromper a execução do laço mesmo que a expressão não tenha sido satisfeita, ignorando todos os comandos seguintes da iteração.

Exemplo:

A partir do exemplo anterior, interrompo o cálculo da PG quando o contador chegar a 5, antes da expressão do laço ser satisfeita.

```
clear all
close all
clc

%calculo dos termos de uma P.G.
%primeiro termo
a1 = 1;
%razao da PG
r = 2
%total de termos considerados
n = 10
cont = 1
%vetor contendo os termos
a = zeros(1,n);
%carrego primeiro termo
a(1,1) = a1;

cont = 2;

%calculo os termos da P.G.
while cont<=n
    if cont ==5
        break; %interrompo o laço caso cont seja igual a 5
    else
        a(1,cont) = a1*r^cont;
        cont = cont+1;
    end
end

%mostro termos da PG
disp(a)
```

Command Window:

```
1      4      8     16      0      0      0      0      0      0
```

– Continue

Com esse comando, o laço passa para o próxima iteração, pulando o passo que seria executado e todos os comandos seguintes.

Exemplo:

| Com <i>continue</i> | Sem <i>continue</i> |
|--|---|
| <pre> clear all close all clc %calculo dos termos de uma P.G. %primeiro termo a1 = 1; %razao da PG r = 2 %total de termos considerados n = 10 cont = 1 %vetor contendo os termos a = zeros(1,n); %carrego primeiro termo a(1,1) = a1; cont = 2; %calculo os termos da P.G. while cont<=n a(1,cont) = a1*r^cont; cont = cont+1; if cont<n %para nao acessar posicao inexistente cont+1 = 11 continue; %pulo comando seguinte %seguindo para a proxima iteracao do loop disp(a(1,cont-1)) %nunca e realizado end end disp('Pulou commando e nao mostrou valor') </pre> <p>Command Window:</p> <p>Pulou commando e nao mostrou valor</p> | <pre> clear all close all clc %calculo dos termos de uma P.G. %primeiro termo a1 = 1; %razao da PG r = 2 %total de termos considerados n = 10 cont = 1 %vetor contendo os termos a = zeros(1,n); %carrego primeiro termo a(1,1) = a1; cont = 2; %calculo os termos da P.G. while cont<=n a(1,cont) = a1*r^cont; cont = cont+1; if cont<n disp(a(1,cont-1)) end end disp('Não Pulou commando e mostrou valor') </pre> <p>Command Window:</p> <p>1</p> <p>4</p> <p>8</p> <p>16</p> <p>32</p> <p>64</p> <p>128</p> <p>256</p> <p>Não Pulou commando e mostrou valor</p> |

– For

Da mesma forma que o while é um laço, porém leva em conta o incremento de uma variável a cada iteração dentro de intervalo especificado para deterimnar quando o laço é determinado.

Sintaxe com incremento de 1 a cada iteração:

```

for x = valor_inicial:valor_final
    comandos enquanto x não atingir o valor final
end

```

Sintaxe com incremento com passo especificado a cada iteração:

for $x = \text{valor_inicial}:\text{incremento}:\text{valor_final}$
 comandos enquanto x não atingir o valor final
end

Por se tratar de um laço, um código utilizando o *for* pode ser gerado com o *while*.

Exemplos:

Gerar matriz contendo o valor da sua posição de linha multiplicada por sua coluna.

| Com <i>while</i> | Com <i>for</i> |
|--|--|
| <pre>clear all close all clc %aloco matriz A = zeros(5,5); lin = 1;%inicio pela primeira linha while lin<=size(A,1) col = 1; %inicio pela primeira coluna while col<=size(A,2) A(lin,col) = lin*col; col = col+1; end lin = lin+1; end disp('Matriz Final') disp(A)</pre> <p>Command Window:</p> <pre>Matriz Final 1 2 3 4 5 2 4 6 8 10 3 6 9 12 15 4 8 12 16 20 5 10 15 20 25</pre> | <pre>clear all close all clc %aloco matriz A = zeros(5,5); for lin = 1:size(A,1) %corro linhas da matriz for col = 1:size(A,2) %corro cada coluna da linha atual A(lin,col) = lin*col; end end disp('Matriz Final') disp(A)</pre> <p>Command Window:</p> <pre>Matriz Final 1 2 3 4 5 2 4 6 8 10 3 6 9 12 15 4 8 12 16 20 5 10 15 20 25</pre> |

Nota: não é recomendável, para aplicações que necessitem de tempo de processamento rápido, a utilização de laços. Procure no Help (F1) do software para verificar a existência de uma função que gere a matriz desejada.

Exemplo:

Baseado no exemplo anterior. *For* com incremento determinado.

| Preencho somente linhas pares e colunas ímpares | Preencho somente linhas ímpares e colunas pares |
|--|--|
| <pre>clear all close all clc %aloco matriz A = zeros(5,5); %corro linhas pares da matriz for lin = 2:2:size(A,1) %corro colunas ímpares for col = 1:2:size(A,2) A(lin,col) = lin*col; end end disp('Matriz Final') disp(A)</pre> <p>Command Window:</p> <pre>Matriz Final 0 0 0 0 0 2 0 6 0 10 0 0 0 0 0 4 0 12 0 20 0 0 0 0 0</pre> | <pre>clear all close all clc %aloco matriz A = zeros(5,5); %corro linhas ímpares da matriz for lin = 1:2:size(A,1) %corro colunas pares for col = 2:2:size(A,2) A(lin,col) = lin*col; end end disp('Matriz Final') disp(A)</pre> <p>Command Window:</p> <pre>Matriz Final 0 2 0 4 0 0 0 0 0 0 0 6 0 12 0 0 0 0 0 0 0 10 0 20 0</pre> |