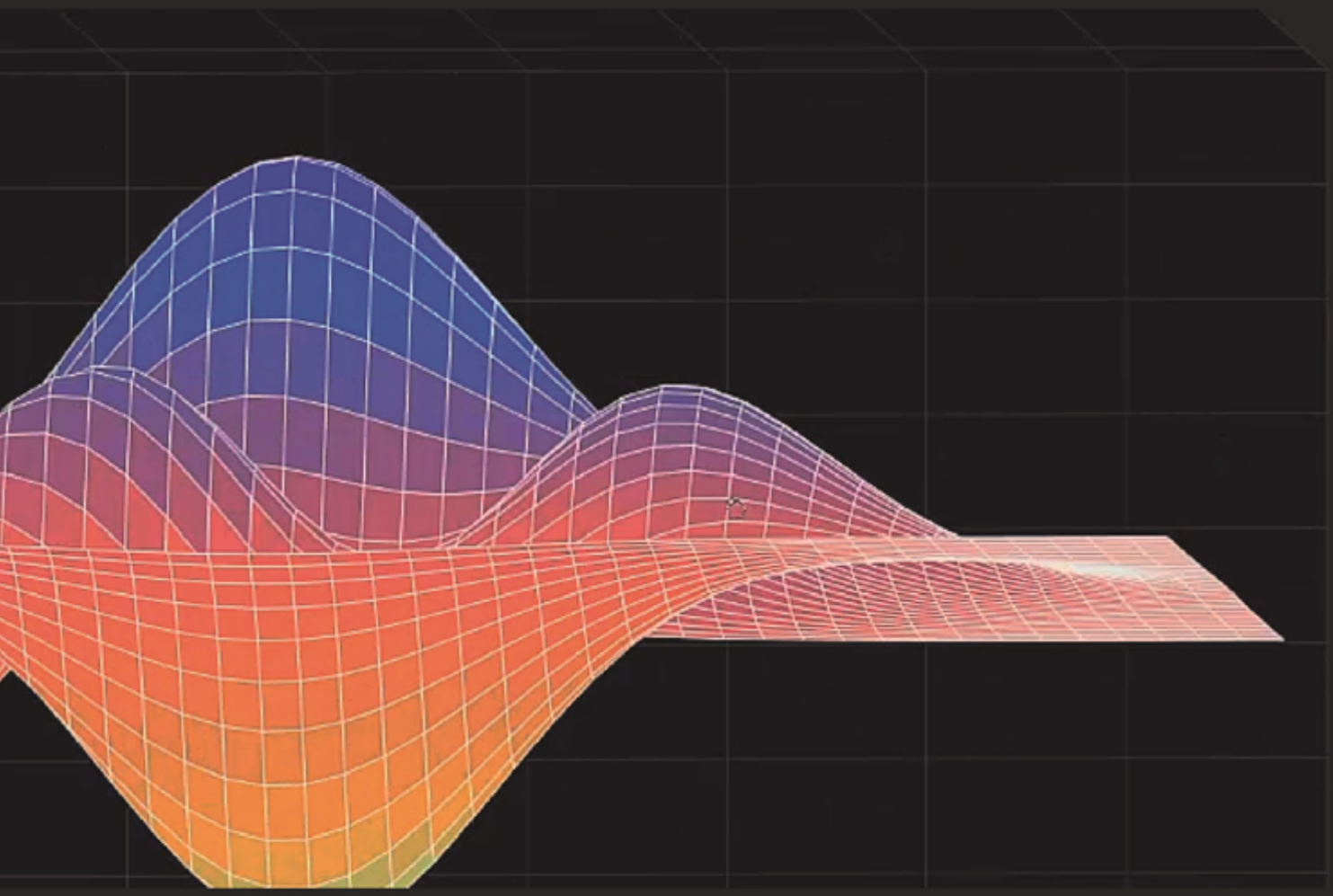
MINICURSO DE MatLab



Módulos 1 e 2
Introdução à Workspace, Dados, Matrizes e Estruturas.

UNICAMP RACING

Quem Somos

A Unicamp E-Racing é uma equipe formada por mais de 70 alunos de diferentes cursos (Eng. Elétrica, Mecânica, de Controle e Automação, da Computação e Física) que trabalham todo ano na construção de um monoposto elétrico de alta performance, baseando-se nas regras da competição promovida pela SAE (Sociedade dos Engenheiros da Mobilidade) representamos a Unicamp em campeonatos nacionais há 6 anos e tivemos a oportunidade de representar o Brasil em campeonatos internacionais em 4 ocasiões. Procuramos combinar nossos esforços e especialidades visando projetar, construir e validar um carro que desafie-nos como engenheiros e cientistas, nos levando a buscar soluções na fronteira da inovação e da tecnologia.

A ideia do mini-curso surgiu na tentativa de transmitirmos nosso conhecimento adquirido ao longo desses anos em um compilado de linhas de códigos e atividades, com uma abordagem mais dinâmica e aplicada, diferente do que se vê normalmente. Nós mesmos ministramos o curso e prestaremos o suporte necessário para que seja concluído com sucesso.

Conteúdo

1 Introdução à Apostila

A apostila a seguir apresenta um modelo linear, onde as variáveis utilizadas para apresentar exemplos podem ser utilizadas mais de uma vez. Desta forma é recomendado que as mesmas não sejam excluídas de um exemplo para o outro.

É recomendado que o usuário reproduza os exemplos em seu computador para observar os resultados e melhor absorver o conteúdo apresentado.

2 Introdução à Workspace

Ao iniciar o MATLAB® uma ou mais janelas aparecerão em sua tela. Isso depende da sua versão e da configuração deixada na última sessão de trabalho, podendo rearranjar as várias janelas, identificadas com nomes nas barras de títulos. A área de trabalho padrão da versão R2018a é mostrada na figura 1.1. Segue um sumário das funcionalidades das janelas.

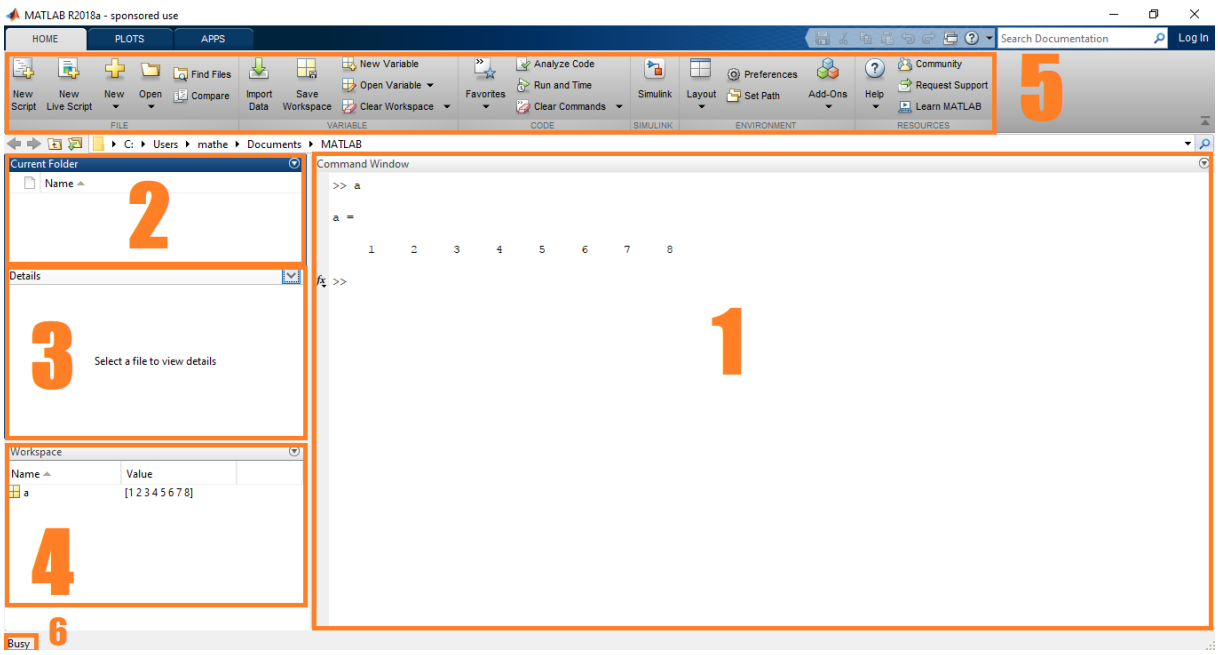


Figura 2.1: Área de Trabalho

1. **Command Window:** Permite digitar comandos e funções. Após digitar sua entrada e apertar ENTER, o MATLAB interpreta a informação digitada. Importante notar que a finalidade do command window é para implementações rápidas e testes, e é uma boa prática não fazer scripts nem tratar dados relevantes nessa área sem o devido cuidado.

2. **Current Folder:** Lista todos os arquivos contidos na pasta atual de trabalho, e destaca scripts e funções que serão utilizadas até arquivos para armazenar dados. Qualquer dado salvo será armazenado nesta pasta.

3. **Details:** Exibe detalhes de qualquer arquivo selecionado na janela Current Folder.

4. **Workspace:** É a janela que contém todas as variáveis de todas as tarefas executadas pelo MATLAB. Note que não há diferenciação da fonte que criou aquela variável, se dois scripts usassem uma variável de nome X por exemplo, essa poderia ser alterada por um ou outro script. Portanto é importante ter atenção com o nome das variáveis.

5. **Home Tab**: É onde estão vários dos ícones para facilitar o acesso às funcionalidades do MATLAB, como por exemplo: ícones para criação de variáveis, criação de scripts, mudança de layout da área de trabalho, entre outros.

6. **Status Bar:** Se encontra no canto inferior e indica se o MATLAB está realizando alguma tarefa (*Busy*) ou se está pronto para o próximo comando (*Ready*).

3 Dados

3.1 Operações Básicas

O MATLAB realiza vários tipos de operações, desde as mais simples, como soma e multiplicação até simulações complexas.

Todo resultado, se não for salvo dentro de uma variável escolhida pelo usuário, é salvo dentro da variável *ans*. Deve-se tomar cuidado pois somente é possível armazenar em *ans* o resultado mais recente, se alguma outra operação utilizar *ans* o resultado anterior é sobrescrito.

Diferentemente de outras linguagens de programação, no MATLAB não é necessário declarar variáveis antes de definir valores para as mesmas

```
1 + 2.5       % Retorna a resposta e armazena a resposta na variavel ans.
ans           % Retorna na tela o valor de ans
1 + 2.5;      % O ponto e vírgula omite o resultado da operação. O resultado fica salvo na variável ans
a = 1 + 2.5   % Salva na variável a e não em ans (tipo padrão: double). Nao precisa ser definida anteriormente. Exibe o resultado pois não há o ponto e vírgula
a = 12.345    % Sobrescreve o valor anterior com um novo valor.
a = 24; a = a + 25; % Ponto e vírgula pode separar operações de forma que elas escritas na mesma linha e sejam executadas uma após outra, isso não é uma boa prática para escrever programas, mas pode ser útil para executar comandos na Command Window
```

O MATLAB é *case-sensitive*, ou seja, ele diferencia letras maiúsculas das minúsculas. A variável *y* é diferente da *Y*, assim como a função *who*, que apresenta uma lista de todas as variáveis na *workspace*, é diferente da *WHO* (inexistente).

```
a
A
who      % Lista de todas as variáveis
WHO
whos     % Lista de todas as variáveis com detalhes
```

```
6 whos a  % Mostra detalhes importantes da variável 'a'
```

As operações *who* e *whos* mencionadas anteriormente se tratam de algumas das várias funções prontas que o MATLAB possui. Para uma explicação do funcionamento de uma função execute o comando *help*. Ele traz exemplos de como implementar a função, assim como o tipo e formato da saída, o formato necessário dos dados de entrada e explica todos os argumentos da função.

```
1 help who % Explicação da função who
```

Para mais explicações de cada função uma boa dica é usar o próprio site da *Mathworks* (`https://www.mathworks.com/help/`) que possui uma explicação completa em inglês e alguns exemplos de implementação de cada função.

4 Tipos de Dados

Toda variável utilizada possui um tipo específico, sendo que este define seu tamanho (em *bits*) e o modo como o MATLAB interpreta a mesma (por exemplo se é ponto flutuante ou não).

Embora para utilizações básicas o tipo de dado não costuma ser importante, em momentos quando é necessário uma alta precisão ou quando se trabalha com um grande volume de dados, escolher o tipo certo pode gerar um resultado mais preciso e mais rápido.

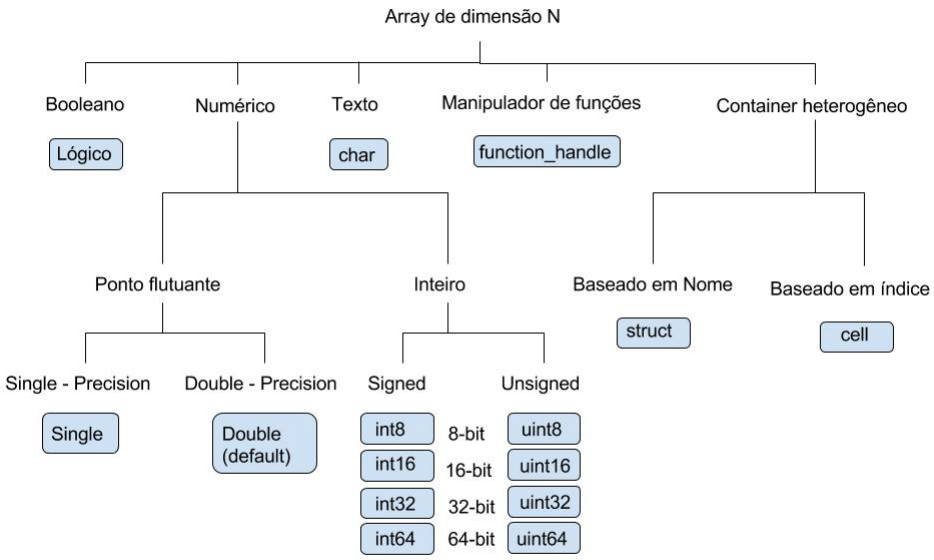


Figura 4.1: Tipos de dado

Classe	Uso esperado
double, single	Utilizado dados em frações numéricas. Precisão. Default do MATLAB.
int8, uint8, int16, uint16, int32, uint32, int64, uint64	Utilizados para números inteiros com ou sem sinais. Uso mais eficiente de memória. Escolha entre 4 tamanhos (8, 16, 32, 64 bits).
char	Utilizado para textos Para múltiplas strings, use *cell arrays*
logical	Usado em relações condicionais ou para testar estados. Pode ter um de dois valores: *true* ou *false*.
function_handle	Ponteiro para uma função. Habilita a passagem de uma função para outra. Pode chamar funções fora do escopo usual. Útil para fazer chamada de gráficos já criados. Salva em um arquivo MAT e restaura mais tarde.
struct	Seus campos armazenam *arrays* de várias classes e tamanhos. Acessa múltiplos campos/índices em uma única operação. O nome dos seus campos identificam seus conteúdos. Método simples de passagem de argumentos para uma função.
cell	Células armazenam *arrays* de várias classes e tamanhos. Permite liberdade para armazenar o quanto de dado for preciso. A manipulação de elementos é similar a *arrays*. Métodos simples de passagem de argumentos para uma função. Uso em listas separadas por vírgulas para eficiência.

Tabela 1: Tipos de dado

4.1 Classes Numéricas

Cada classe numérica apresenta um tamanho diferente, sendo necessário se atentar ao tamanho do dado que se deseja armazenar para evitar *overflows*.

Classe	Extensão de Valores	Função de Conversão
Signed inteiro 8-bit (1 byte)	-2^7 a 2^7-1	int8
Signed inteiro 16-bit (2 bytes)	-2^{15} a $2^{15}-1$	int16
Signed inteiro 32-bit (4 bytes)	-2^{31} a $2^{31}-1$	int32
Signed inteiro 64-bit (8 bytes)	-2^{63} a $2^{63}-1$	int64
Unsigned inteiro 8-bit (1 byte)	0 a 2^8-1	uint8
Unsigned inteiro 16-bit (2 bytes)	0 a $2^{16}-1$	uint16
Unsigned inteiro 32-bit (4 bytes)	0 a $2^{32}-1$	uint32
Unsigned inteiro 64-bit (8 bytes)	0 a $2^{64}-1$	uint64

Tabela 2: Diferenças entre classes de inteiros

Existem funções que executam a conversão de tipos de dado mas um cuidado especial é necessário na hora de utilizá-las, uma vez que pode-se perder informações na conversão de um tipo maior para um menor .

```
1 x = int16(255)
2 x_int8 = int8(x)  % As informações são perdidas devido ao
    overflow
3 x = int16(x_int8) % Não é mais possível recuperar o dado
    original
```

Exercício 1:

> Abra o gerenciador de tarefas do seu computador e faça os itens a seguir observando o uso de memória do MATLAB a cada item.
>
> **a)** Crie uma matriz 10000x10000 de *um's*.
>
> **b)** Lembrando que o formato padrão do MATLAB é *double*, mude o formato da matriz do item a para *int8*.

E, por último, a função *num2str* permite a conversão de um número para *string* enquanto a função *str2num* faz o contrário. O tratamento de strings será detalhada mais a frente.

```
num2str(1)
str2num('5')
str2num('5 0') %Devido ao espaçamento foi criado um vetor
```

4.2 Formatação de Dados

Quando são inseridos valores numéricos dentro do MATLAB, ele escolhe o formato em que será exibido ao usuário. Por exemplo, para variáveis do tipo *float*, o formato padrão é *short*, que apresenta 4 casas decimais.

É importante frisar que o formato de exibição da variável não afeta como o MATLAB armazena ela (se é um *int8* ou *int16*, por exemplo), apenas como ela é exibida ao usuário.

Quando o formato de exibição é mudado ele permanece da forma selecionada durante toda essa sessão do MATLAB, voltando ao seu formado padrão após ele ser fechado e aberto novamente.

```
help format
format short; a
format long; a
b = 1/2 + 2/3 + 2/5
format rat; b
```

4.3 Constantes Predefinidas

Existe uma série de constantes pré-definidas no MATLAB, como infinito (*inf*) e pi (π). Um cuidado especial deve ser tomado com essas constantes pois se uma variável for criada com o mesmo nome de uma delas, o MATLAB sobrescreve a mesma.Caso isso ocorra a função *clear* pode apagar a variável criada.

```
1/0
NaN
pi
i
j
i = 1    % Cuidado: sera sobrescrito
clear i;
```

Exercício 2:

> Abra o gerenciador de tarefas do seu computador e faça os itens a seguir observando o uso de memória do MATLAB a cada item.
>
> **a)** Crie uma matriz 10000x10000 de *um's*.
>
> **b)** Lembrando que o formato padrão do MATLAB é *double*, mude o formato da matriz do item a para *int8*.

4.4 Números Imaginários

O MATLAB possui suporte para operações com número imaginários.

```
b = 1 + i
c = 1 - j
sqrt(-1)
exp(i*pi/2)

current_1 = 50 + 50i; current_2 = -45 + 37i; current_3 = 40 -
    37i;
magniture = abs(current_1);
fase = phase(current_1);
total_current = current_1 + current_2 + current_3;
[total_theta, total_fase] = cart2pol(real(current_1), imag(
   current_1));
%Note que total_fase está em radianos. Para converter para
   graus podemos fazer por exemplo:
total_fase_graus = rad2deg(total_fase);
```

4.5 Operadores Lógicos

Uma expressão se diz lógica se os operadores são lógicos e os operandos são relações e/ou variáveis do tipo lógico. Eles são: & (*AND*), | (*OR*), ~ (*NOT*), XOR (*EXCLUSIVE OR*).

- O operador **E/AND** (&&) retorna verdadeiro se ambas as expressões unidas por ele também forem verdadeiras, enquanto um só (&) é uma operação lógica bit a bit entre as expressões;

```
        a = [0 0 1 1]
        b = [0 1 0 1]
        c_1 = a & b
        c_2 = and(a,b)
```

- O operador **OU/OR** (| |) retorna verdadeiro se alguma das expressões unidas a eles também forem, enquanto só (|) é, também, uma operação lógica;

```
a = [0 0 1 1]
b = [0 1 0 1]
c_1 = a | b
c_2 = or(a,b)
```

- O operador **NOT** ( ) inverte a entrada bit a bit;

```
a = [0 0 1 1]
c_1 = ~a
c_2 = not (a)
```

- O **XOR** é um operador lógico que retorna verdadeiro quando as entradas são diferentes.

```
a = [0 0 1 1]
b = [0 1 0 1]
c_1 = a & b
c_2 = xor(a,b)
```

4.6 Matrizes

O grande forte do MATLAB é o seu tratamento de matrizes, daí que vem seu nome ***MAT****rix* ***LAB****oratory*. Ele possui uma enorme gama de funções para operações entre matrizes, facilitando muito cálculos numéricos.

4.6.1 Vetores

Vetores são matrizes com somente uma linha ou uma coluna.

A função *size* indica o tamanho de um certo dado (constantes são consideradas como matrizes de uma linha e uma coluna). Já a função *numel* indica o número de elementos.

```
x = [1 0 0]
size(x)      % [numero de linhas, numero de colunas]
size(x,1)    % Numero de linhas
size(x,2)    % Numero de colunas
numel(x)     % Numero de elementos (linhas * colunas)
nrows_ncols    = size(x)
[nrows,ncols] = size(x)
```

Utilizando o operador **':'** podemos criar vetores regularmente espaçados. Usando *m:n:p*, sendo m o primeiro elemento, n o último e p o incremento entre cada valor.

A função *linspace* gera um vetor linearmente espaçado, sendo possível escolher o número de pontos

```
x = 1:10          % Por default o incremento é 1
x = 10:-1:-9
x = 1:0.1:2
x = 1:5:2         % Um elemento
x = 10:1:2        % Vazio
```

```
x = linspace(0,100,500) % 0 até 100 com 500 pontos
y = sin(x);             % Cria um vetor com os valores de
   seno para cada elemento de x
plot(y)                 % A função plot será explicada em
   detalhes mais a frente
```

Exercício 3:

> Plote o seno de zero até 2π com os seguinte parâmetros
>
> **a)** Amplitude 2, 10 pontos.
>
> **b)** Amplitude 2, 100 pontos.

O ponto e vírgula é usado para separar as colunas de uma matriz enquanto ' é utilizado para transpor a matriz.

```
y = [0; i; 0]
size(y)
y'  % Transposição do complexo conjugado
y.' % Transposição normal
```

4.6.2 Matrizes Padrão

Existe uma série de funções que criam matrizes especiais de tamanhos especificados pelo usuário, como matrizes de zeros (*zeros*), de um's (*ones*), de valores aleatórios menores que um(**rand**) e até a matriz identidade (*eye*)

```
A = zeros(2,3)
A = ones(2,3)
A = rand(2,3)
A = eye(3,3)
```

Como foi dito anteriormente, não é necessário declarar uma variável antes de utilizá-la no MATLAB, entretanto, se o usuário já souber o tamanho da mesma no início do seu *script*, é recomendado pré alocar esse espaço de memória criando uma matriz de zeros do tamanho desejado. Essa boa prática reduz o tempo de execução.

Exercício 4:

> Crie uma matriz diagonal 20x20 de valores aleatórios utilizando as funções *rand* e *eye*.

4.6.3 Acesso a Elementos

Elementos dentro de uma matriz podem ser acessados e alterados individualmente, assim como novos elementos podem ser incluídos mesmo em uma matriz já criada.

O MATLAB possui ainda o índice *end*, que representa o ultimo elemento de um certo dado. Ele é bem útil quando o tamanho do dado a ser trabalhado é variável.

```
x(1)
x(2)
```

```
z(3) = 1                    % Cria um vetor z e coloca na posição
    3 um zero
z(end+1) = 2     % Cria uma nova posicao
z(end-2)
z(end+1)         % Na leitura, "Index exceeds matrix
   dimensions"
```

Muitas vezes é desejado selecionar dados de uma matriz a partir de um certo padrão, como por exemplo uma linha ou coluna específica. O operador **:** permite efetuar tais operações

```
A = [1 2 3; 4 5 6]
A(1,:)                % Primeira linha
A(:,1)                % Primeira coluna
```

Do mesmo modo que novos elementos podem ser adicionados em uma matriz já pronta, novas matrizes podem ser concatenadas em uma já pronta.

```
A = [1 2 3]
B = [4 5 6]
C = [A B]
D = [A; B]
E = [A(:) B(:)]
```

4.6.4 Operações Básicas

Ao realizar operações de elementos deve-se levar em consideração que existem certas condições provindas da álgebra linear que são necessárias para algumas operações serem feitas, como por exemplo a multiplicação entre matrizes, que exige que o número de colunas da 1^a matriz seja igual ao número de linhas da 2^a.

Quando se deseja realizar operações elemento a elemento é necessário coloca um *ponto* antes da operação (para multiplicação **.***).

Vale notar que para as operações como “ .* ” e “ ./ “ as matrizes devem ter as mesmas dimensões ou um dos dois deve ser escalar.

```
a = [1 2 3];
b = [4 5 6];
c = [1 2 3 ; 1 2 3];
d = [1 2 3 4; 1 2 3 4];
4*a
a+b               % Operação de soma
a+c               % Soma 'a' em todas as linhas de 'c'
a+d               % Soma não é possível
a*b     % Produto escalar. Erro devido ao tamanho das
   matrizes
a*b'
a.*b    % Produto elemento a elemento
a./b    % Divisão elemento a elemento
a^2               % Só funciona com matriz quadrada
b.^a
```

4.6.5 Funções Básicas

Algumas funções básicas são úteis no tratamento de dados, como saber o valor mínimo (*min*), máximo (*max*) e médio (*mean*) de um certo dado.

```
a = [5 10 15 20; 25 30 35 40];
min(a)                    % Retorna o valor mínimo de cada
   coluna
max( a(:,2) )    % Valor máximo da segunda coluna de a
mean(a)
```

Exercício 5:

> Crie uma matriz 50x50 de valores aleatórios usando a função *rand*.
>
> **a)** Encontre os maiores valores de cada coluna.
>
> **b)** Encontre o maior valor absoluto.

4.6.6 Máscaras Lógicas

Máscaras lógicas podem ser utilizadas para procurar valores específicos dentro de matrizes.

```
A = [1 2 3; 1000 2000 3000]
A>100        % Máscara lógica
A( A>100 ); % Aplica a máscara lógica em A
```

4.6.7 Chars/Strings

Strings são tratadas pelo MATLAB como qualquer outro dado, podendo formar vetores e matrizes. Cada *char* ocupa uma posição do vetor/matriz.

```
txt1 = 'Text1'
txt1 = ['T','e','x','t','1'] % Strings tambem sao Matrizes no
    Matlab
txt1(4)
txt2 = 'Testtext2'
txt3 = [txt1,txt2]
txt4 = [txt1; txt2]                % Erro:tamanhos diferentes
   de string
txt4 = ['linha1';'linha2']

Texto1 = 'Hello'
whos;
Texto_Cortado    = Texto1(1:2)
Texto_Transp     = Texto1'
Texto_Linear     = Texto1(:)
clear Texto_Cortado Texto_Transp Texto_Linear

```

```
Texto2 = ' World!';
[Texto1 Texto2]
[Texto1; Texto2]
```

Variáveis do tipo *char* podem ser convertidas para single ou *double*, sendo os valores equivalentes aos sinais gráficos determinados pela tabela ASCII.

```
Texto1_doub = double(Texto1);      % Transforma em double (
   valores são os equivalentes as letras na tabela ASCII
whos
Texto1_R    = char(Texto1_doub)    % Transforma de volta em
   string
clear Texto1_doub Texto1_int Texto1_R
```

O MATLAB possui funções prontas para o tratamento de strings, como *strcmp* que compara duas strings, *strfind* que procura pedaços de texto específico e *lower*/*upper*, que transforma caracteres em minúsculos/maiúsculos.

```
strcmp(Texto1, Texto2)       % Retorna verdadeiro/falso (1/0)
strfind(Texto1, 'el')        % Retorna posição da primeira
   letra
strfind(Texto1, 'He')
strfind(Texto1, 'he')        % Case senstitive
lower(Texto1)                % Transforma em lowercase
upper(Texto1)
strfind(lower(Texto1), 'he')
```

4.7 Structs (estruturas)

Variáveis no MATLAB devem ser de somente um tipo, não é possível, por exemplo, criar uma matriz sendo a primeira coluna dados do tipo *char* e a segunda do tipo *double*. Sabendo que em alguns casos seria interessante essa "mistura" de dados, é necessária a utilização de *structs*. *Structs* são ótimas formas de criar bancos de dados.

```
Artigos.nome = 'O caso dos 10 negrinhos';
Artigos.tipo  = 'Livro';
Artigos.preco = 123.0;
Artigos.dim = [24 36 18];
Artigos.peso = 650;

struct('nome', 'O caso dos 10 negrinhos', 'tipo', 'Livro', '
   preco', 123.0, 'dim', [24 36 18], 'peso', 650) %Mesmo
   efeito

Artigos

Artigos(2).nome = 'Admiravel Mundo Novo';
Artigos(2).tipo  = 'Livro';
Artigos(2).preco = 82.0;
Artigos(2).dim = [24 30 10];
```

```
Artigos(2).peso = 376;

Artigos

Artigos(3).nome = 'Galileu';
Artigos(3).tipo  = 'Revista';
Artigos(3).preco = 'Gratis';
Artigos(3).dim = [24 36];
Artigos(3).peso = 100;
```

O acesso a elementos de uma *struct* é similar ao de elementos em uma matriz.

```
Artigos(1).tipo
Artigos(2).tipo
Artigos.nome
```

Como todo outro tipo de dado do MATLAB, novos campos podem ser inseridos ou removidos.

```
Artigos(3).numerodeserie='1234-567-890'     % Acrescenta novo
   campo
Artigos.numerodeserie
Artigos = rmfield(Artigos,'numerodeserie') % Apaga o campo
```

4.8 Cells (células)

Uma célula é um tipo de dado que, diferentemente de uma matriz, pode conter diferentes tipos de dado em cada um de seus elementos e os mesmos ainda podem ser de tamanhos diferentes. Ela é muito comumente utilizada para armazenar listas de strings, que possuem tamanhos diferentes e portanto não podem ser armazenados em, por exemplo, linhas de uma matriz.

O acesso a elementos de uma célula é feito de forma diferente, sendo utilizado chaves **{}** ao invés de parenteses A função *cell* cria um *array*, de tamanho especificado pelo usuário, de células vazias.

```
C = cell(1,5)
C{1} = 'Teste';
C{1}(1) % Primeiro elemento do primeiro elemento da célula
C{2} = 1;
C{3} = eye(3,3);
C
C{:}
conteudo1  = C{1}
celula1    = C(1)
whos
```

Quando se deseja aplicar uma função a todos os elementos de uma célula, utiliza-se a função *cellfun*

```
C{2} = 'Outro Teste';
C{3} = 'Mais um Teste';
```

```
cellfun(@isempty,C)
cellfun(@length,C)
cellfun(@(x) strfind(x,'Teste'), C, 'UniformOutput', false)
```

5 Scripts e Salvamento de Dados

Embora os exemplos apresentados nesta apostila foram todos executados na própria *Command Window*, não é uma boa prática escrever seus códigos na mesma, uma vez que embora exista um histórico, seu código não é realmente salvo.

O MATLAB possui dois tipos de formatos diferentes, o *.mat*, que salva a sua *workspace* (variáveis) e o *.m*, que salva seus códigos.

5.1 Tipo *.m*

Para criar um *script* deve-se clicar no ícone *New Script*, presenta na aba *Home*, como pode ser visto na figura 5.1. Um editor de texto aparecerá na tela.

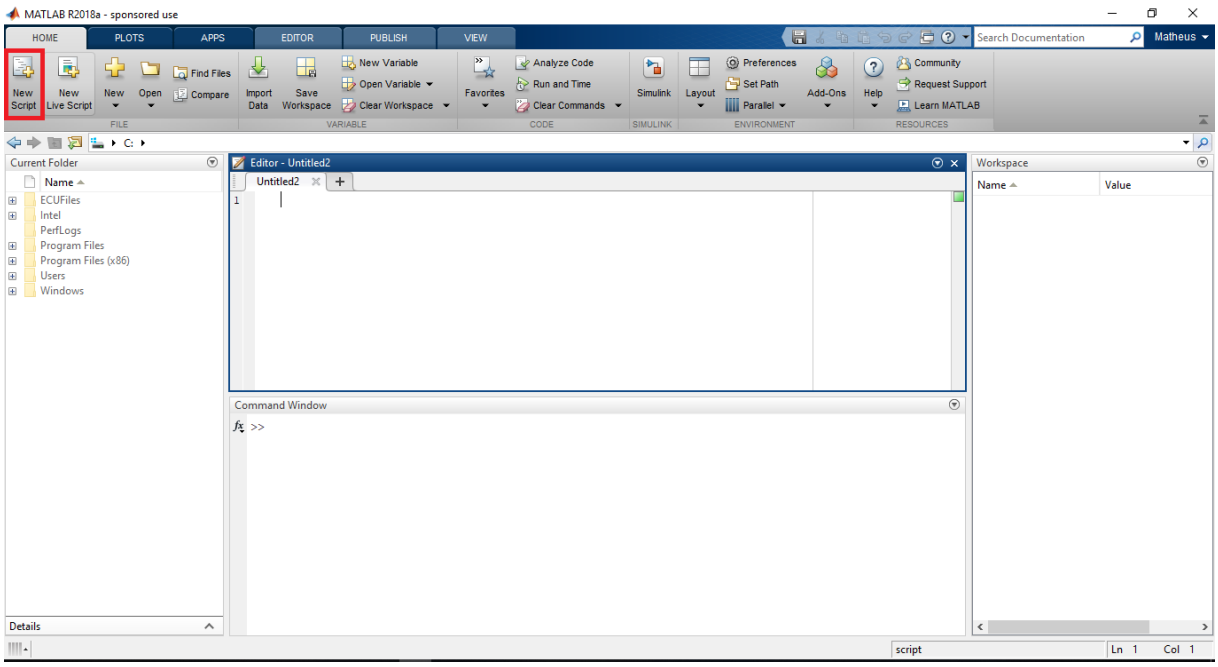


Figura 5.1: Localização do ícone *New Script*

Qualquer código escrito neste editor de texto pode ser executado por completo clicando no ícone *Run*, presente na aba *Editor*, como pode ser visto na figura 5.2. O atalho **F5** também executa o *script*.

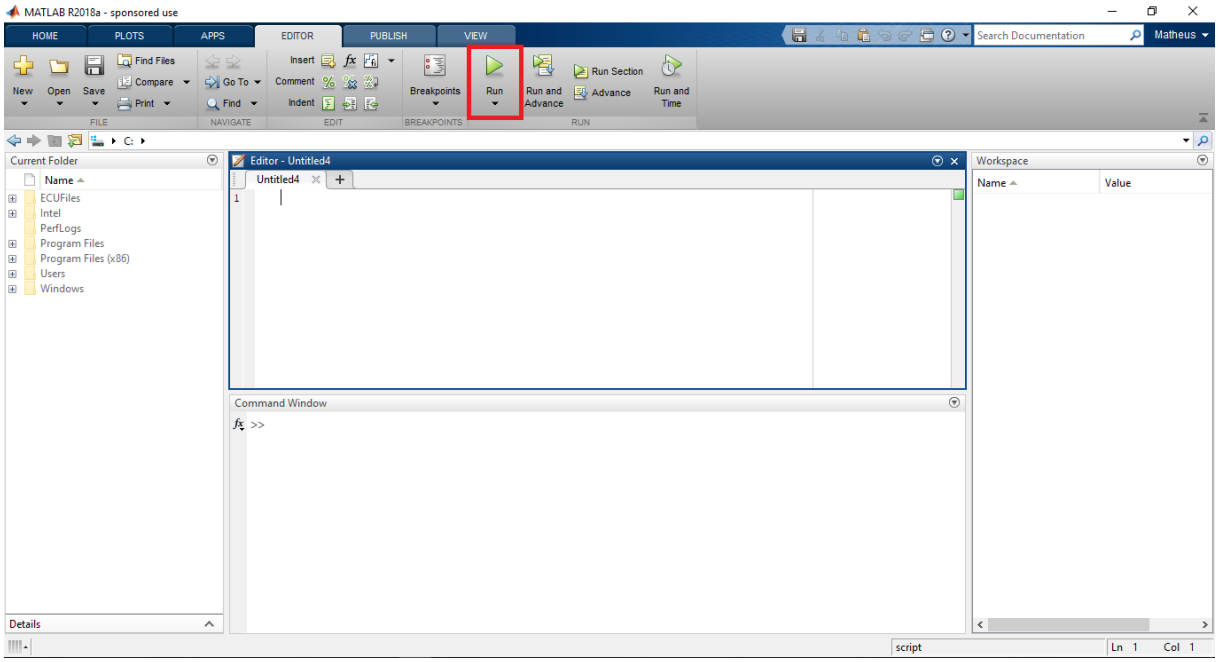


Figura 5.2: Localização do ícone *Run*

É possível separar seu *script* em sessões, sendo assim possível executar somente uma parte do mesmo. Isto é muito útil quando se deseja testar uma parte específica do código.

Sessões são separadas por porcentagens duplas (%%) e para executar uma sessão basta utilizar o atalho **Ctrl+Enter** ou clicar no ícone ao lado do **Run**.

5.2 Tipo *.mat*

Como dito anteriormente, quando as variáveis presentes na *workspace* são salvas o formato de arquivo criado é o *.m*.

A função *save* é a responsável por salvar enquanto a *load* por carregar. É possível salvar e carregar tanto a *workspace* inteira como apenas algumas variáveis específicas. Se não for especificado nos argumentos da função *save* o local onde a variável vai ser salva, a mesma é criada no *Current Folder*.

```
1  x = 1;
2  y = 2;
3  save('tmp.mat');          % Salva todas as variáveis
4  clear all
5  load('tmp.mat');
6  save('tmp_x.mat','x'); % Salva variáveis especificas
7  clear all
8  load('tmp_x.mat');
9  clear all
10 s = load('tmp.mat')      % Por default, salva em uma struct
11 clear all
12 y = load('tmp.mat','y')
```

6 Funções

Diferentemente de *scripts*, que executam comandos como se os mesmo fossem escritos na própria *Command Window*, funções permitem uma maior flexibilidade, possibilitando que sejam passados valores como entrada e que a saída seja referentes a sua entrada.

Para criar uma função deve-se criar um editor da mesma forma quando se está criando um *script*.

A estrutura de uma função esta apresentada a seguir, onde *y1,...,yN* são as saídas, *x1,...,xN* são as entradas e *myfun* é o nome da mesma.

```
1 function [y1,yN] = myfun(x1,xM)
2         <Código da função>
3 end
```

Exercício 6:

> Crie uma função que receba as coordenadas bidimensionais (x e y) de dois pontos e retorne a distância entre os mesmos.
> Lembrete: a distância D entre dois pontos a e b é dado por:
> $D = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$

7 Exercícios Extra

Exercício 7:

> A segurança de carros movidos a bateria é extremamente preocupante. Por isso, há um sistema de monitoramento de bateria (BMS) que deve desligar o carro em caso de sobre temperatura, sobretensão ou baixa tensão mínima. Todos os dados adquiridos são registrados para análise posterior. No arquivo *logBMS.mat* há a tensão do pack de baterias; a mínima tensão de uma célula individual; a máxima; a temperatura; a corrente total; e o tempo, em cada coluna, nesta ordem.
>
> **a)** Qual o valor inicial de cada um dos dados? E o final?
>
> **b)** Crie um vetor-coluna referente a cada dado.
>
> **c)** Crie uma estrutura com a data do teste (12/03/2017), tensão inicial e final do pack, a menor tensão mínima registrada, a menor tensão máxima registrada e a máxima temperatura.
>
> **d)** Ache em que instante de tempo as tensões máxima e mínima foram obtidas.
>
> **e)** Além do BMS, também utilizamos um sensor Hall para uma medição da corrente mais precisa. No arquivo *logHall.mat*, temos duas colunas, a primeira é a corrente em Amperes, e a segunda é o tempo em segundos. Observe a diferença de comportamento entre a corrente medida pelo BMS e pelo sensor Hall. Faça um gráfico (*plot*) do erro do BMS, além de apresentar na *Command Window* (função *disp*) o máximo erro registrado.
>
> **f)** Crie um vetor-coluna que indique a potência elétrica instantânea para todas as medidas, lembrando que a potência elétrica pode ser calculada por $P_e = V * I$.

g) Salve o arquivo agora com os dados já manipulados.