

Processamento de Imagens

Prof. MSc. Daniel Menin Tortelli

e-mail: danielmenintortelli@gmail.com

Skype: daniel.menin.tortelli

Site: <http://sites.google.com/site/danielmenintortelli/home>

Matlab

- O MATLAB é um software destinado a fazer cálculos com matrizes (MATLAB = **MAT**rix **LAB**oratory).
- MATLAB foi criado no fim dos anos 1970 por Cleve Moler, então presidente do departamento de ciência da computação da Universidade do Novo México.
- Ele logo se espalhou para outras universidades e encontrou um forte uso no âmbito da comunidade matemática aplicada.
- MATLAB é construído na linguagem MATLAB, às vezes chamada M-código ou simplesmente M.
- MATLAB é uma linguagem de programação e um conjunto de caixas de ferramentas especializadas de software.

Image Processing Toolbox™

- O *Image Processing Toolbox*™ fornece um conjunto abrangente de algoritmos de referência e aplicativos de fluxo de trabalho para processamento, análise, visualização e desenvolvimento de algoritmos de imagem.
- Ele permite realizar segmentação de imagem, aprimoramento de imagem, redução de ruído, transformações geométricas, registro de imagem e processamento de imagem 3D.
- As funções e aplicativos de visualização permitem explorar imagens, volumes 3D e vídeos; ajustar contraste; criar histogramas; e manipular regiões de interesse (ROIs).

Área de Trabalho do MATLAB

1) Diretório de Trabalho.

2) Comando **pwd** exibe o diretório padrão.

3) Comando **path** exibe as rotas de busca de arquivos e comandos.

4) Exibe o conteúdo das variáveis.

Área de Trabalho do MATLAB

The screenshot displays the MATLAB R2018b environment. The Command Window at the bottom shows the following commands and outputs:

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

>> B = [2 5; -3 -5]
B =
     2     5
    -3    -5

>> C = A + B
C =
     3     7
     0    -1
```

The Variables Editor (labeled 5) shows a 2x2 double matrix with the following values:

	1	2	3	4	5	6
1	3	7				
2	0	-1				
3						
4						

The Workspace (labeled 4) shows the following variables:

Name	Value
A	[1,2;3,4]
B	[2,5;-3,-5]
C	[3,7;0,-1]

1) Define uma matriz chamada **A** 2x2

2) Define uma matriz chamada **B** 2x2

3) Define uma matriz **C** 2x2 com o resultado da soma das matrizes A e B.

4) Exibe o conteúdo das variáveis/matrizes.

5) Exibe o conteúdo as matrizes em formato tabular.

Área de Trabalho do MATLAB

The screenshot displays the MATLAB Command Window and Workspace. The Command Window shows three steps of a script execution, each highlighted with a red box and a red number. The Workspace window shows the current state of the workspace.

Step 1: The Command Window shows the command `>> D = 15` and the output `D = 15`.

Step 2: The Command Window shows the command `>> I = imread('Imagem1.tif')` and the output `I = 10x10 uint8 matrix`. The output is a 10x10 matrix of uint8 values.

Step 3: The Command Window shows the command `>> whos` and the output showing the details of the variables in the workspace.

The Workspace window shows the following variables:

Name	Value
A	[1,2;3,4]
B	[2,5;-3,-5]
C	[3,7;0,-1]
D	15
I	10x10 uint8

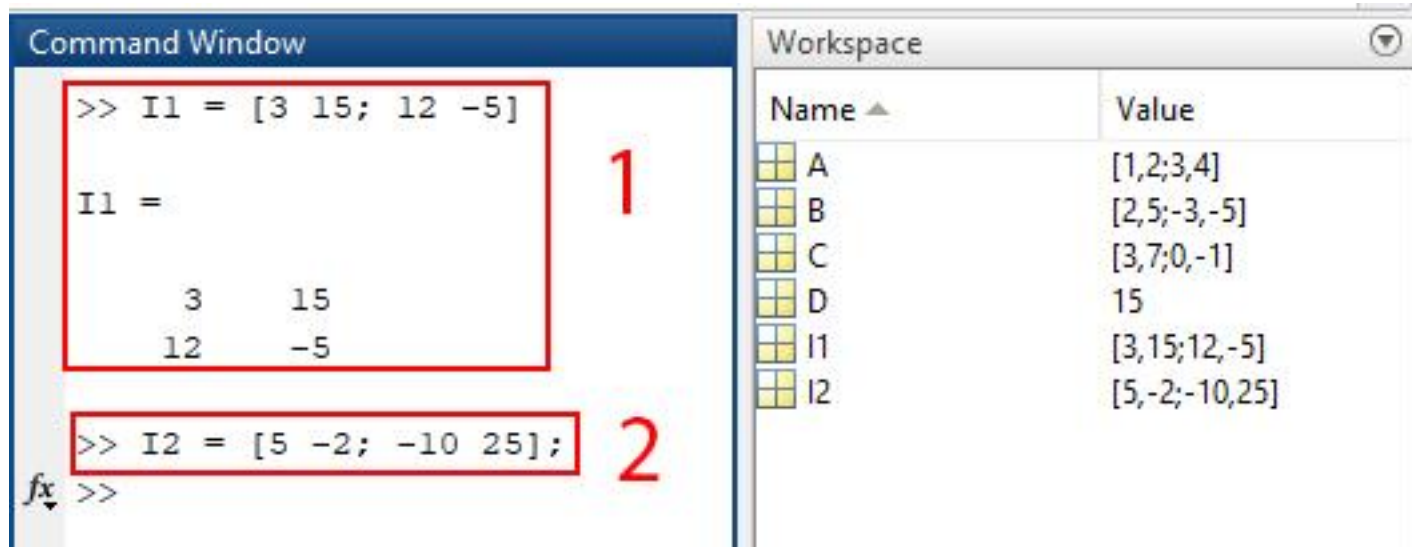
1) Define uma variável chamada **D** e atribui o valor **15**.

2) Carrega uma imagem e salva seu conteúdo na variável/matriz chamada **I**.

3) O comando **whos** exibe uma listagem com detalhes das variáveis existentes no espaço de trabalho.

Suprimindo a impressão do resultado dos comandos

- 1) Por padrão, o MatLab imprime o resultado do comando logo abaixo da linha de execução.
- 2) Para suprimir a impressão, sempre adicione um **ponto-e-vírgula (;)** no final de cada comando.



Gravação, Remoção e Carregamento de Dados

- A forma mais simples de salvar os resultados de uma seção do Matlab é usar o comando **save**:

```
>> save minhaseção1
```

- O comando **save** grava o estado atual das variáveis no arquivo **minhaseção1.mat** no diretório de trabalho.
- A extensão **.mat** no arquivo gravado indica que é um arquivo de dados do Matlab.

Gravação, Remoção e Carregamento de Dados

- Para remover variáveis específicas, usa-se o comando **clear**:

```
>> clear A B C D; whos
```

- O comando **clear** sem qualquer nome de variável remove TODAS as variáveis do espaço de trabalho.

```
>> clear; whos
```

- *Note que múltiplos comandos podem ser digitados na mesma linha, desde que sejam separados por ponto-e-vírgula (;)*

Gravação, Remoção e Carregamento de Dados

- Para recarregar os resultados de uma seção do Matlab, use-se o comando **load**:

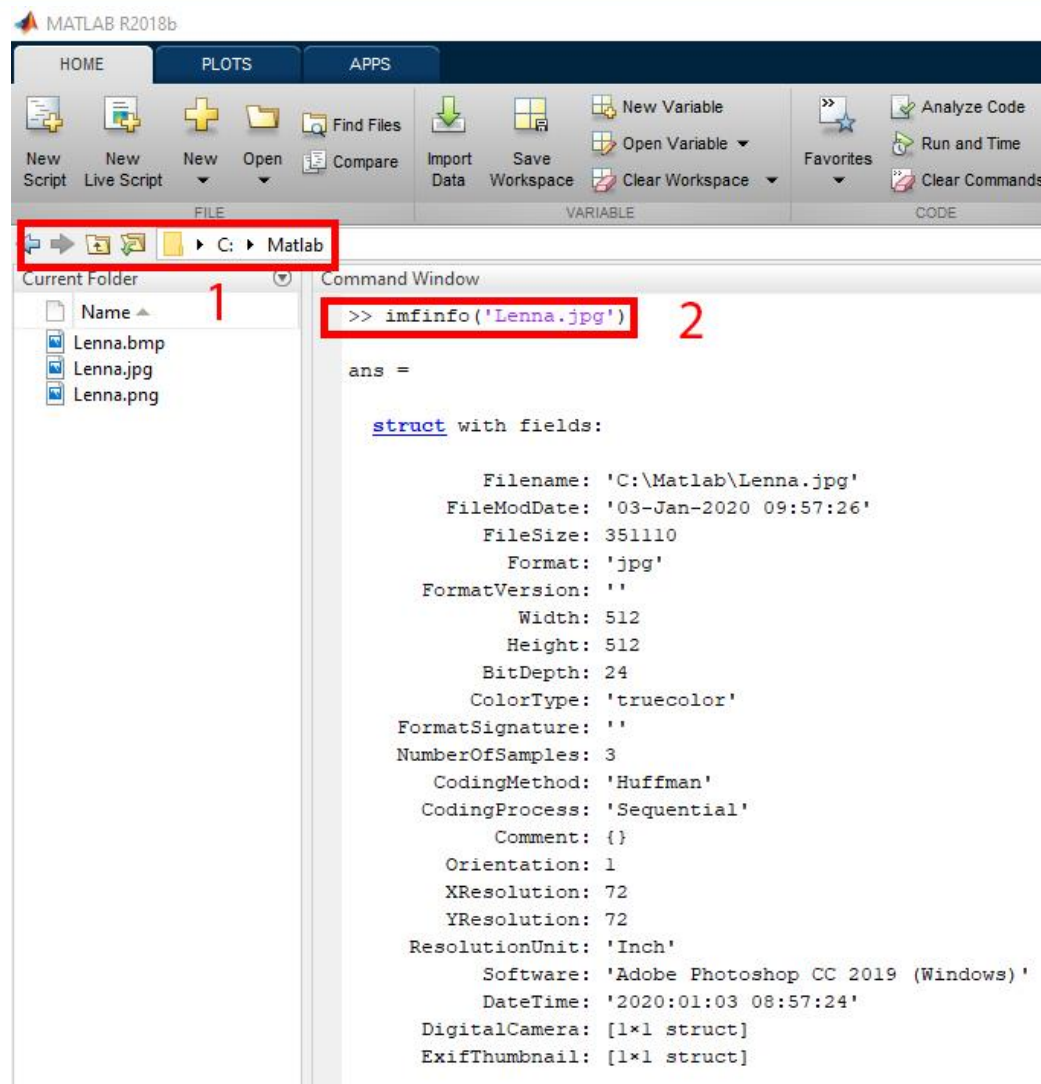
```
>> load minhaseção1; whos
```

- TODAS as variáveis salvas previamente serão recarregadas no espaço de trabalho.

Imagens no Matlab

- A leitura e escrita de imagens são feitas com as funções **imread** e **imwrite**.
- Estas funções suportam todos os formatos mais comuns de imagens e criam/exportam os apropriados conjuntos (matrizes) 2D/3D no âmbito do ambiente Matlab.
- A função **imfinfo** pode ser usada para examinar uma imagem e determinar todas as suas propriedades importantes, incluindo tipo, formato, tamanho e profundidade de bit.

Comando **imfinfo**



1) Configurar o diretório de trabalho que contém as imagens.

2) O comando **imfinfo** analisa a imagem e fornece informações como: FileSize, Format, Width, Height, BitDepth, ColorType...

3) O comando **clc** limpa a janela de comando.

Comando `imread`

MATLAB R2018b

HOME PLOTS APPS VARIABLE VIEW

Open Rows Columns
New from Selection Print 1 1
VARIABLE SELECTION EDIT

Insert Delete Transpose Sort

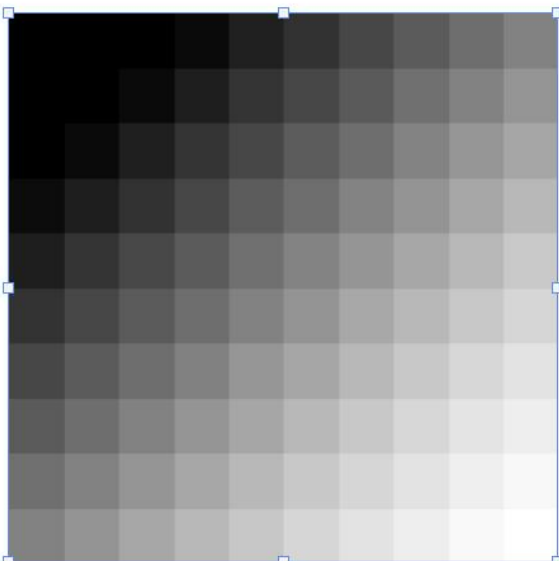
Current Folder
Name
Image1.tif
Lenna.bmp
Lenna.jpg
Lenna.png
Lenna.tif

Variables - I1
I1
10x10 uint8

	1	2	3	4	5	6	7	8	9	10	1
1	0	0	0	9	22	35	52	70	88	107	
2	0	0	9	21	36	52	69	89	108	127	
3	0	9	22	36	52	70	88	108	128	147	
4	10	21	35	52	70	88	108	127	148	167	
5	21	36	52	69	89	108	128	147	167	186	
6	36	52	70	88	107	127	148	167	186	203	
7	52	69	88	107	128	147	167	186	203	219	
8	70	88	108	128	147	167	185	203	220	233	
9	89	108	127	147	167	186	203	219	234	246	
10	108	127	147	167	185	203	219	233	246	255	
11											
12											

Workspace
Name Value
I1 10x10 uint8 2

Command Window
1
>> I1 = imread('Image1.tif')
I1 =
10x10 uint8 matrix
0 0 0 9 22 35 52 70 88 107
0 0 9 21 36 52 69 89 108 127
0 9 22 36 52 70 88 108 128 147
10 21 35 52 70 88 108 127 148 167
21 36 52 69 89 108 128 147 167 186
36 52 70 88 107 127 148 167 186 203
52 69 88 107 128 147 167 186 203 219
70 88 108 128 147 167 185 203 220 233
89 108 127 147 167 186 203 219 234 246
108 127 147 167 185 203 219 233 246 255



Comando **imread**

- 1) O comando **imread** lê os dados de uma imagem e monta uma matriz com as cores dos pixels que compõem a imagem.
 - 2) Os valores dos pixels da matriz resultante da leitura pode ser vista acessando a variável **I1** na aba **Workspace**.
- Como a imagem está no formato **grayscale**, ela possui 8bit por pixel, resultando em valores que variam de 0 (preto) à 255 (branco).

Comando imread

MATLAB R2018b

HOME PLOTS APPS

New Script New Live Script New Open Find Files Import Data Save Workspace New Variable Open Variable Clear Workspace Favorites Analyze Code Run and Time Clear Commands Layout Preferences Set Path Add-Ons Help Community Request Support Learn MATLAB

FILE VARIABLE CODE ENVIRONMENT RESOURCES

Current Folder

- dsotm.jpg
- Imagem1.tif
- Imagem2.tif
- img01.tif
- lenna.bmp
- Lenna.jpg
- Lenna.png
- Lenna.tif
- Untitled-2.jpg

Command Window

```
>> A = imread('imagem2.tif')  
  
10x10x3 uint8 array  
  
A(:,:,1) =  
  
0 255 255 255 0 0 255 0 0 0  
0 255 255 255 0 0 255 0 0 0  
0 255 255 255 0 0 255 0 0 0  
0 255 255 255 0 0 255 0 0 0  
0 255 255 255 0 0 255 0 0 0  
0 255 255 255 0 0 255 0 0 0  
0 255 255 255 0 0 255 0 0 0  
0 255 255 255 0 0 255 0 0 0  
0 255 255 255 0 0 255 0 0 0  
0 255 255 255 0 0 255 0 0 0  
  
A(:,:,2) =  
  
0 0 0 255 255 255 255 0 0 0  
0 0 0 255 255 255 255 0 0 0  
0 0 0 255 255 255 255 0 0 0  
0 0 0 255 255 255 255 0 0 0  
0 0 0 255 255 255 255 0 0 0  
0 0 0 255 255 255 255 0 0 0  
0 0 0 255 255 255 255 0 0 0  
0 0 0 255 255 255 255 0 0 0  
0 0 0 255 255 255 255 0 0 0  
0 0 0 255 255 255 255 0 0 0  
  
A(:,:,3) =  
  
0 0 0 255 0 0 255 255 255 0  
0 0 0 255 0 0 255 255 255 0  
0 0 0 255 0 0 255 255 255 0  
0 0 0 255 0 0 255 255 255 0  
0 0 0 255 0 0 255 255 255 0  
0 0 0 255 0 0 255 255 255 0  
0 0 0 255 0 0 255 255 255 0  
0 0 0 255 0 0 255 255 255 0  
0 0 0 255 0 0 255 255 255 0  
0 0 0 255 0 0 255 255 255 0
```

Workspace

Name	Value
A	10x10x3 uint8
ans	1x1 struct

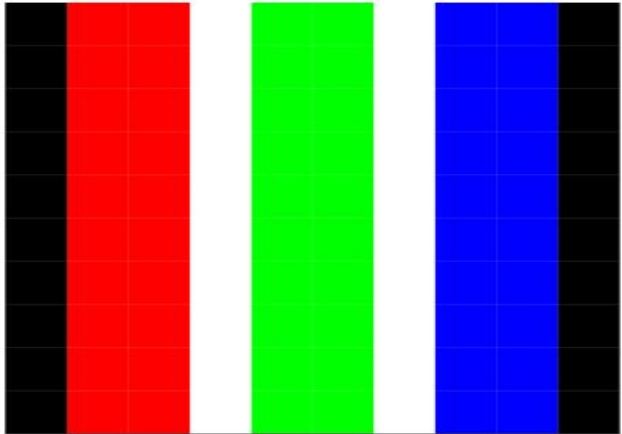
Details

Select a file to view details

R

G

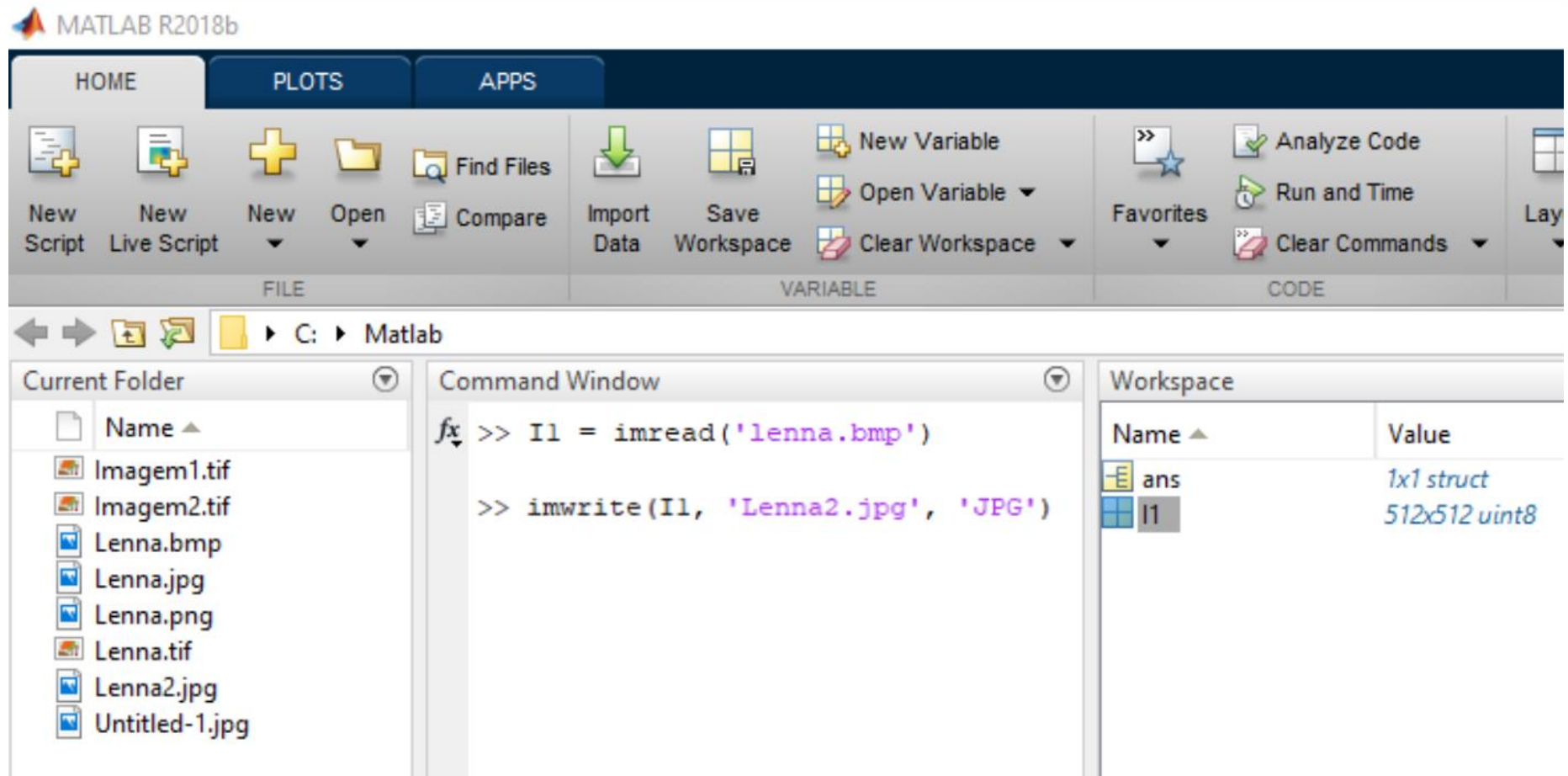
B



Comando **imread**

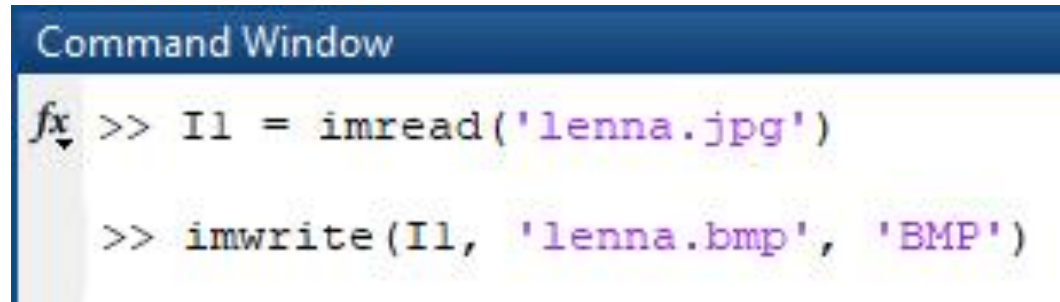
- 1) O comando **imread** lê os dados de uma imagem e monta uma matriz com as cores dos pixels que compõem a imagem.
 - 2) Os valores dos pixels da matriz resultante da leitura pode ser vista acessando a variável **A** na aba **Workspace**.
- Como a imagem está no formato **truecolor**, ela possui 8bit por pixel, resultando em valores que variam de 0 (preto) à 255 (branco).
 - Como a imagem está no formato RGB, a matriz mostra os valores **RGB** respectivamente para cada pixel.

Comando **imwrite**



Comando **imwrite**

- 1) O comando **imwrite** escreve no disco os dados da matriz lida previamente no Matlab através do comando **imread**.
- 2) Dessa forma, pode ser feita uma conversão de tipos de formato de imagem.
 - No exemplo, foi lida a imagem **Lenna.JPG** através do comando **imread** e o valor de seus pixels foram salvos na matriz **I1**.
 - Em seguida, foi usado o comando **imwrite** para gerar uma nova imagem no formato **BMP**, contendo os valores dos pixels da matriz **I1**.



```
Command Window  
fx >> I1 = imread('lenna.jpg')  
    >> imwrite(I1, 'lenna.bmp', 'BMP')
```

Comando **imwrite**

- Comparação dos tamanhos dos arquivos após a conversão da imagem no formato JPG (com compressão) para o formato BMP (sem compressão).

```
Command Window
>> imfinfo('lenna.jpg')

ans =

    struct with fields:

        Filename: 'C:\Matlab\lenna.jpg'
        FileModDate: '03-Jan-2020 09:57:26'
        FileSize: 351110
        Format: 'jpg'
        FormatVersion: ''
        Width: 512
        Height: 512
        BitDepth: 24
        ColorType: 'truecolor'
        FormatSignature: ''
        NumberOfSamples: 3
        CodingMethod: 'Huffman'
        CodingProcess: 'Sequential'
        Comment: {}
        Orientation: 1
        XResolution: 72
        YResolution: 72
        ResolutionUnit: 'Inch'
        Software: 'Adobe Photoshop CC 2019 (Windows)'
        DateTime: '2020:01:03 08:57:24'
        DigitalCamera: [1x1 struct]
        ExifThumbnail: [1x1 struct]
```

```
Command Window
>> imfinfo('lenna.bmp')

ans =

    struct with fields:

        Filename: 'C:\Matlab\lenna.bmp'
        FileModDate: '03-Jan-2020 11:53:22'
        FileSize: 786486
        Format: 'bmp'
        FormatVersion: 'Version 3 (Microsoft Windows 3.x)'
        Width: 512
        Height: 512
        BitDepth: 24
        ColorType: 'truecolor'
        FormatSignature: 'BM'
        NumColorMapEntries: 0
        Colormap: []
        RedMask: []
        GreenMask: []
        BlueMask: []
        ImageDataOffset: 54
        BitmapHeaderSize: 40
        NumPlanes: 1
        CompressionType: 'none'
        BitmapSize: 786432
        HorzResolution: 0
        VertResolution: 0
```

Exibindo Imagens

- Matlab oferece duas funções básicas para a exibição de imagens: **imshow** e **imagesc**.



- **imshow** requer que a matriz 2D especificada para exibição se conforme a um tipo de dado de imagem (por exemplo, imagens de intensidade/em cores com valor entre 0–1 ou 0–255).
- **imagesc** aceita matrizes de entrada de qualquer tipo de dado de Matlab (uint8, uint16 ou duplo) e qualquer faixa de valor numérico.

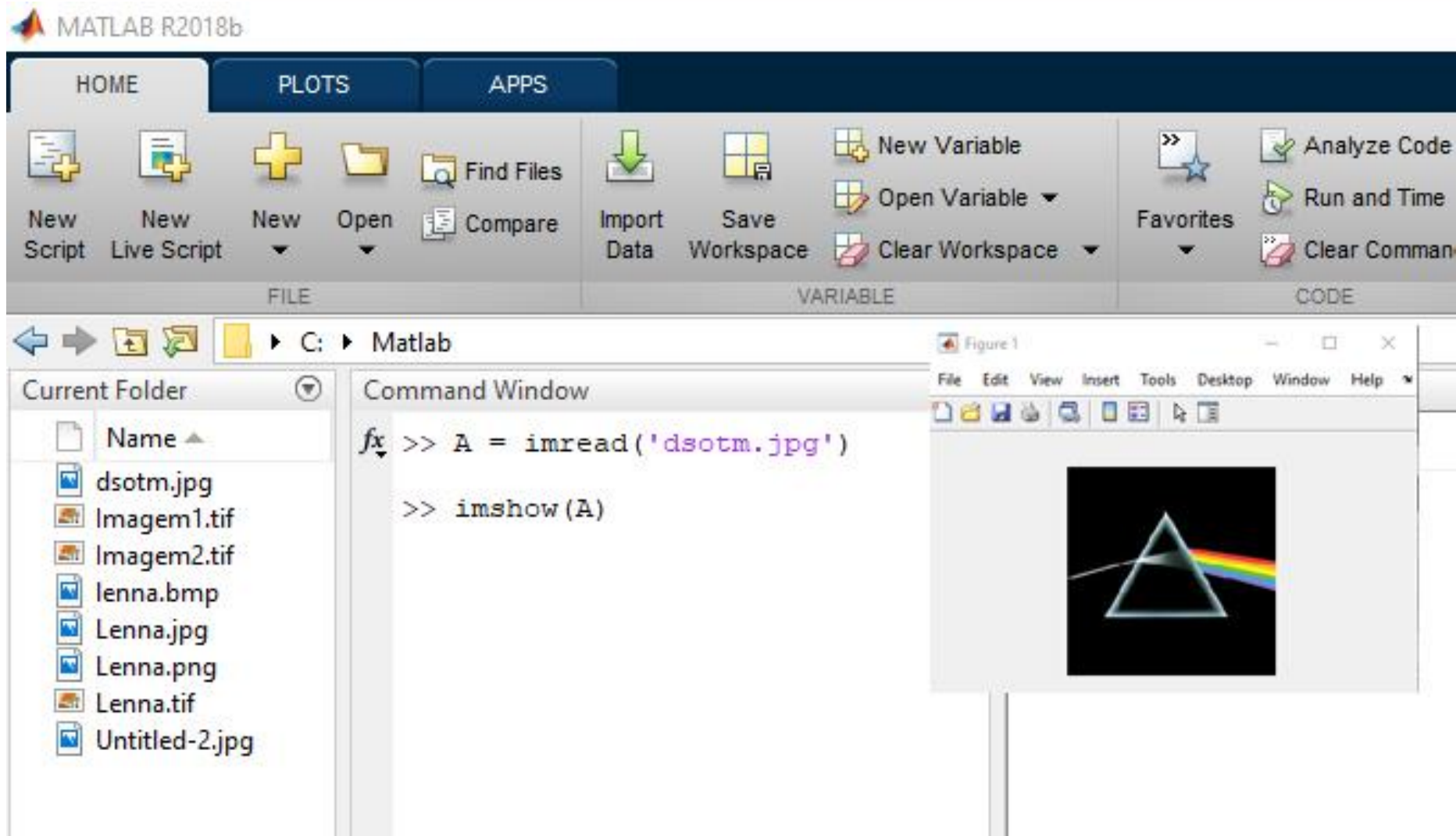
```
Command Window
>> imfinfo('dsotm.jpg')

ans =

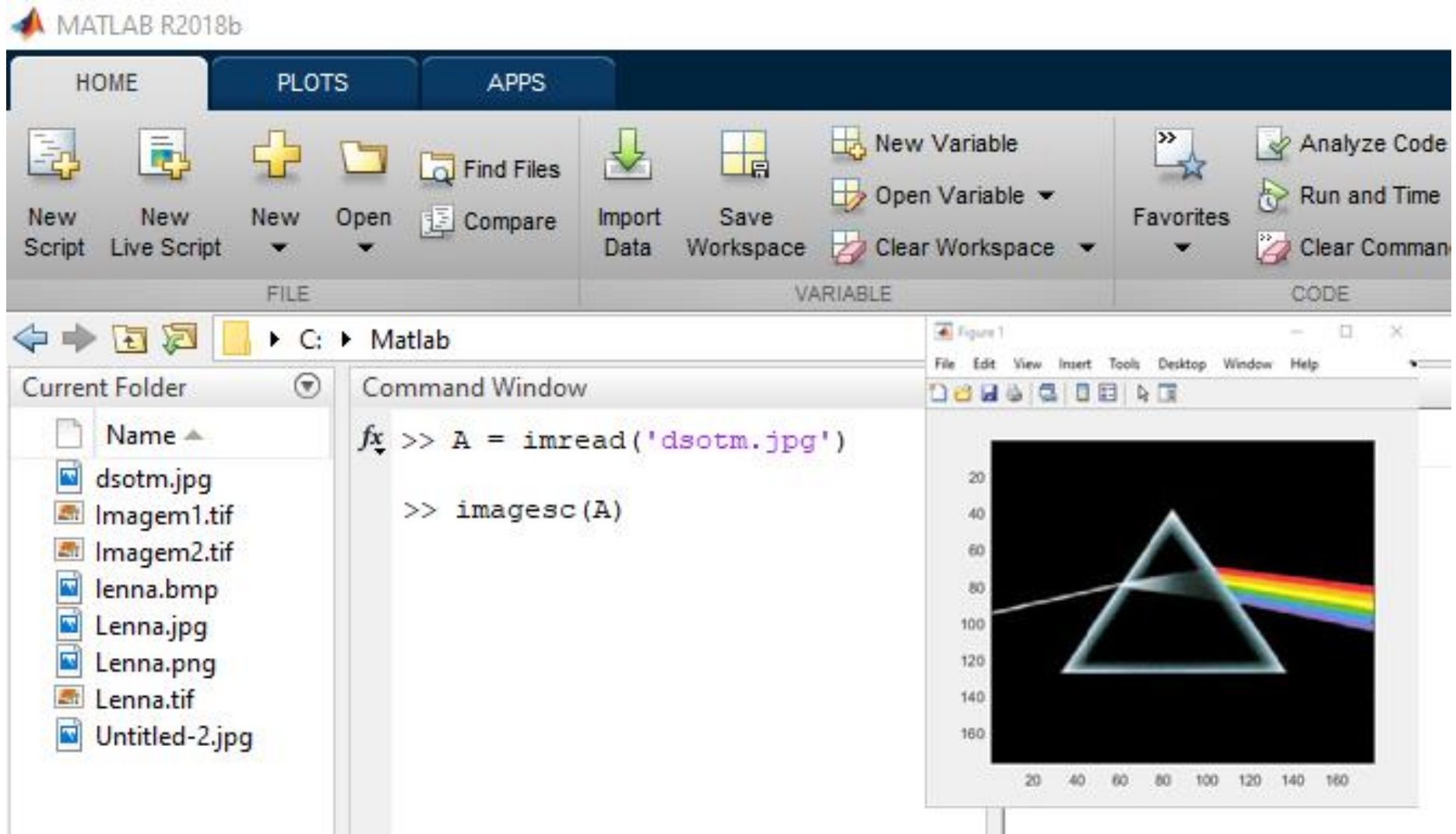
    struct with fields:

        Filename: 'C:\Matlab\dsotm.jpg'
        FileModDate: '14-Oct-2007 22:34:05'
        FileSize: 23980
        Format: 'jpg'
        FormatVersion: ''
        Width: 177
        Height: 177
        BitDepth: 24
        ColorType: 'truecolor'
        FormatSignature: ''
        NumberOfSamples: 3
        CodingMethod: 'Huffman'
        CodingProcess: 'Sequential'
        Comment: {}
        Orientation: 1
        XResolution: 72
        YResolution: 72
        ResolutionUnit: 'Inch'
        Software: 'Adobe Photoshop 7.0'
        DateTime: '2007:10:14 21:34:05'
        DigitalCamera: [1x1 struct]
        ExifThumbnail: [1x1 struct]
```

Comando **imshow**



Comando **imagesc**



Comando colormap

- O comando **colormap** seta um mapa de cores para a figura atualmente carregada através do comando **imagecsc**.
- O novo mapa de cores possui o mesmo comprimento (número de cores) do mapa de cores padrão.
- Também é possível criar mapas de cores personalizados...

Colormap Name

The following table lists the predefined colormaps.

Colormap Name	Color Scale
parula	
jet	
hsv	
hot	
cool	
spring	
summer	
autumn	
winter	
gray	
bone	
copper	
pink	
lines	
colorcube	
prism	
flag	
white	

Comando colormap

MATLAB R2018b

HOME PLOTS APPS

New Script New Live Script New Open Find Files Import Data Save Workspace New Variable Open Variable Clear Workspace Favorites Run and Time Analyze Code Clear Commands Preferences Set Path Add-ons Help Request Support Learn MATLAB

FILE VARIABLE CODE ENVIRONMENT RESOURCES

Current Folder: C:\Matlab

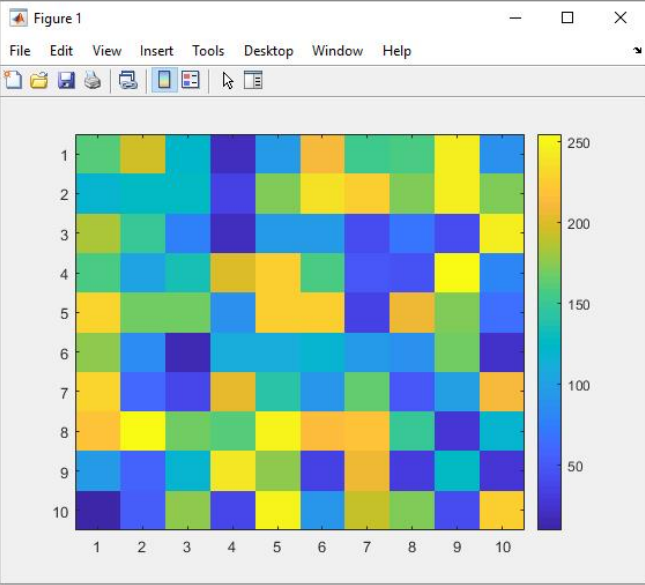
Command Window

```
>> B = rand(10) * 256 1  
B =  
160.1786 196.1943 121.1126 20.5452 97.5495 210.9769 153.0301 156.6406 246.9279 87.5802  
119.8283 128.3252 126.2434 34.5341 173.9857 236.3628 225.0596 170.6970 245.3914 171.3456  
182.8613 150.8035 78.5696 17.8776 96.3229 94.7235 40.9304 70.4761 41.7756 245.8185  
155.8603 105.1139 135.4633 200.8718 225.6838 155.5764 48.9188 46.5748 254.9401 78.9762  
229.8727 167.2406 169.5894 88.8154 225.0064 227.9631 36.4760 208.6101 173.4187 66.5081  
175.8468 83.6182 14.6392 111.7804 111.2665 119.8030 95.9290 88.0136 170.2107 22.7335  
231.8839 61.7484 36.7908 204.7292 142.4403 92.0504 165.1353 50.6910 98.0372 210.0775  
219.0078 254.6715 168.4212 162.3963 248.1761 216.1457 217.6345 151.2865 27.1450 117.0105  
94.8609 59.3440 118.2969 240.8537 175.1578 36.3874 207.7389 30.5049 127.5329 28.6346  
9.6880 53.0935 177.6007 37.6152 250.2673 92.1664 192.4239 172.0724 42.8788 226.7413  
  
>> imagesc(B) 2  
>> colormap 3
```

Workspace

Name	Value
B	10x10 double

Figure 1



1) Cria uma imagem 10x10 com valores de pixels variando de 0 – 256

2) Exibe a imagem da matriz contida na variável **B**.

3) Exibe a barra lateral na imagem, que representa o mapa de cores.

Comando colormap

Command Window

```
>> B = rand(10) * 256
```

B =

```
160.1786 196.1943 121.1126 20.5452 97.5495 210.9769 153.0301 156.6406 246.9279 87.5802
119.8283 128.3252 126.2434 34.5341 173.9857 236.3628 225.0596 170.6970 245.3914 171.3456
182.8613 150.8035 78.5696 17.8776 96.3229 94.7235 40.9304 70.4761 41.7756 245.8185
155.8603 105.1139 135.4633 200.8718 225.6838 155.5764 48.9188 46.5748 254.9401 78.9762
229.8727 167.2406 169.5894 88.8154 225.0064 227.9631 36.4760 208.6101 173.4187 66.5081
175.8468 83.6182 14.6392 111.7804 111.2665 119.8030 95.9290 88.0136 170.2107 22.7335
231.8839 61.7484 36.7908 204.7292 142.4403 92.0504 165.1353 50.6910 98.0372 210.0775
219.0078 254.6715 168.4212 162.3963 248.1761 216.1457 217.6345 151.2865 27.1450 117.0105
94.8609 59.3440 118.2969 240.8537 175.1578 36.3874 207.7389 30.5049 127.5329 28.6346
9.6880 53.0935 177.6007 37.6152 250.2673 92.1664 192.4239 172.0724 42.8788 226.7413
```

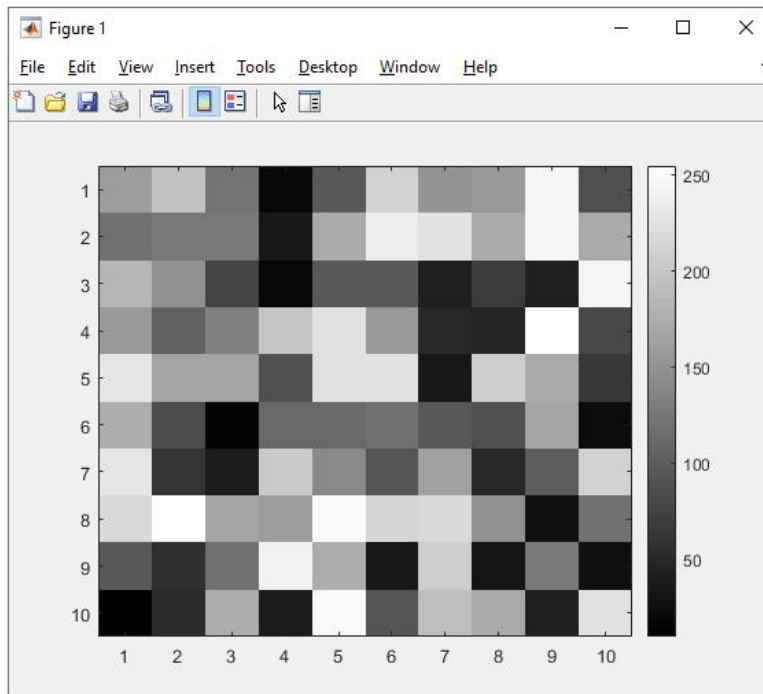
```
>> imagesc(B)
```

```
>> colorbar
```

```
>> colormap(gray)
```

```
>>
```

4



4) Muda o mapa de cores padrão para **escala de cinza (gray)**.

Comando colormap

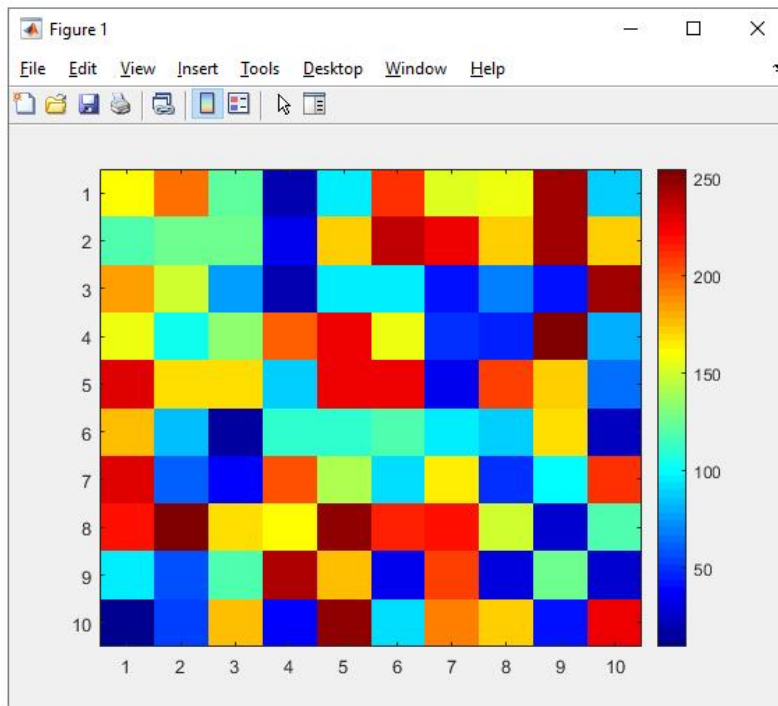
```
>> B = rand(10) * 256
```

```
B =
```

160.1786	196.1943	121.1126	20.5452	97.5495	210.9769	153.0301	156.6406	246.9279	87.5802
119.8283	128.3252	126.2434	34.5341	173.9857	236.3628	225.0596	170.6970	245.3914	171.3456
182.8613	150.8035	78.5696	17.8776	96.3229	94.7235	40.9304	70.4761	41.7756	245.8185
155.8603	105.1139	135.4633	200.8718	225.6838	155.5764	48.9188	46.5748	254.9401	78.9762
229.8727	167.2406	169.5894	88.8154	225.0064	227.9631	36.4760	208.6101	173.4187	66.5081
175.8468	83.6182	14.6392	111.7804	111.2665	119.8030	95.9290	88.0136	170.2107	22.7335
231.8839	61.7484	36.7908	204.7292	142.4403	92.0504	165.1353	50.6910	98.0372	210.0775
219.0078	254.6715	168.4212	162.3963	248.1761	216.1457	217.6345	151.2865	27.1450	117.0105
94.8609	59.3440	118.2969	240.8537	175.1578	36.3874	207.7389	30.5049	127.5329	28.6346
9.6880	53.0935	177.6007	37.6152	250.2673	92.1664	192.4239	172.0724	42.8788	226.7413

```
>> imagesc(B)
>> colorbar
>> colormap(gray)
>> colormap(jet)
```

5



5) Muda o mapa de cores padrão para **jet**.

Comando colormap

```
>> B = rand(10) * 256
```

B =

20.9545	204.2234	100.9449	51.7433	97.9240	28.9087	14.4303	166.9869	236.3189	204.2945
184.9004	145.2935	74.3423	172.7914	74.6739	194.0935	96.2809	11.3845	65.3276	0.2557
62.4530	233.5503	111.3399	189.9349	40.6027	233.1736	212.2853	228.6650	250.0136	137.3646
160.8391	26.3028	147.6798	226.0550	204.6246	23.6605	117.1835	6.1848	145.3880	91.1308
41.9577	22.0890	219.0346	67.0525	251.2060	198.6136	18.2213	106.1694	187.3895	125.8254
133.9528	213.8357	243.5921	227.8945	13.2102	58.8944	114.4882	202.9176	95.1140	168.9228
177.4390	208.0187	184.5179	86.7661	201.4570	148.2118	207.8768	139.1694	177.7857	87.0527
90.8138	139.0446	118.2223	10.4068	1.5895	29.0618	26.9563	51.5870	139.7658	13.9842
15.7736	150.8371	127.5380	90.7567	213.7228	70.5689	113.7511	240.5023	61.1813	219.6173
104.7993	99.2181	87.9482	22.3500	181.3536	3.4276	160.3783	195.6934	125.2799	77.0204

```
>> imagesc(B)
```

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

6

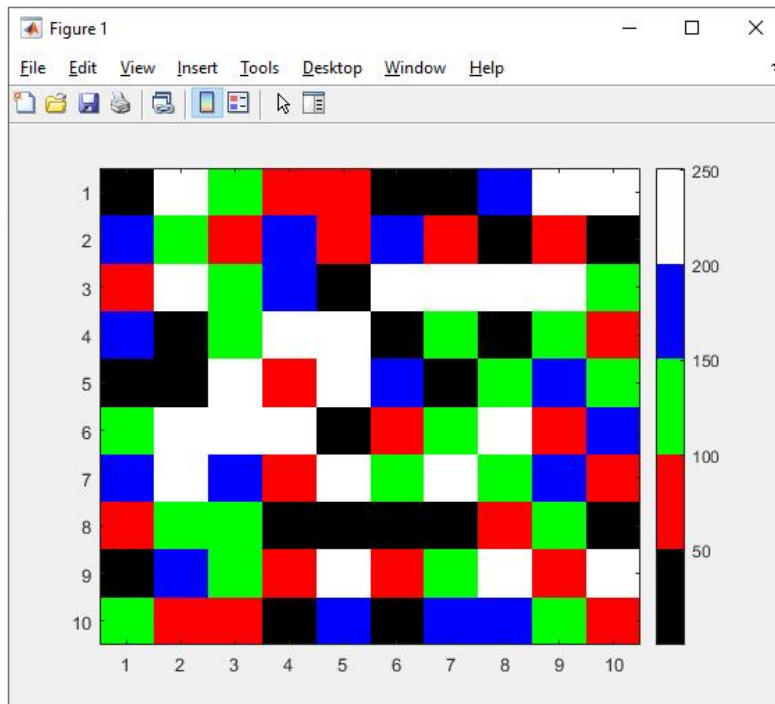
```
>> colorbar
```

```
>> colormap(jet)
```

```
>> colormap(mymap)
```

7

```
>>
```



6) Cria um mapa de cores personalizado. No exemplo, os valores entre 0 – 255 são divididos em 5 cores.

Valores entre:

0 – 50 = preto

51 – 100 = vermelho

101 – 150 = verde

151 – 200 = azul

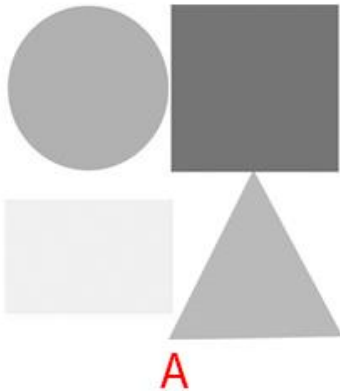
201 – 255 = branco

7) Aplica o mapa de cores na imagem atual.

Arquivos M – scripts e funções

Command Window

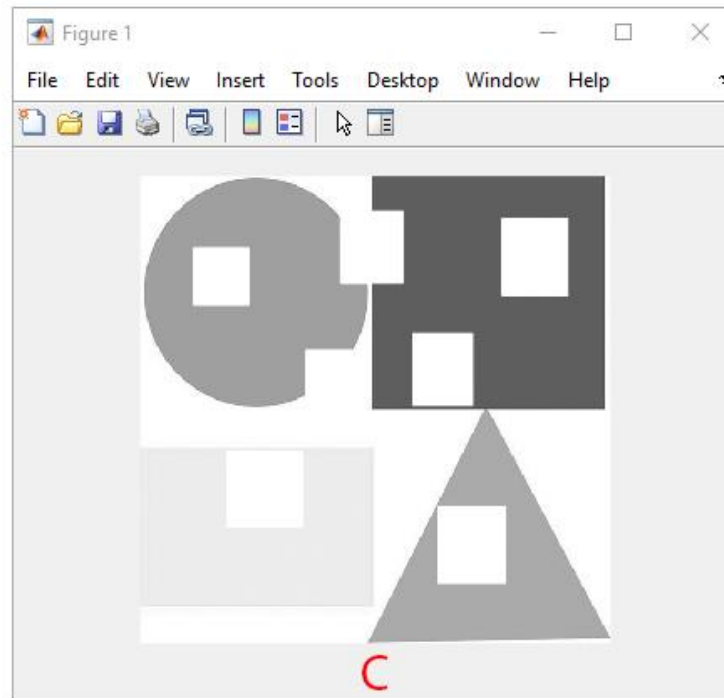
```
>> clear %Remove todas as variáveis do espaço de trabalho
>> A = imread('geo1.tif'); % Lê uma imagem TIFF 512x512
>> B = imread('geo2.tif'); % Lê uma imagem TIFF 256x256
>> imshow(A); pause(3); % Exibe a 1ª imagem e espera 3 segundos
>> imshow(B); pause(3); % Exibe a 2ª imagem e espera 3 segundos
>> B = imresize(B, size(A)); % Altera o tamanho da imagem B para ser igual ao tamanho de A
>> C = imadd(A,B); imshow(C); % Exibe a soma de 8 bits
>>
```



A



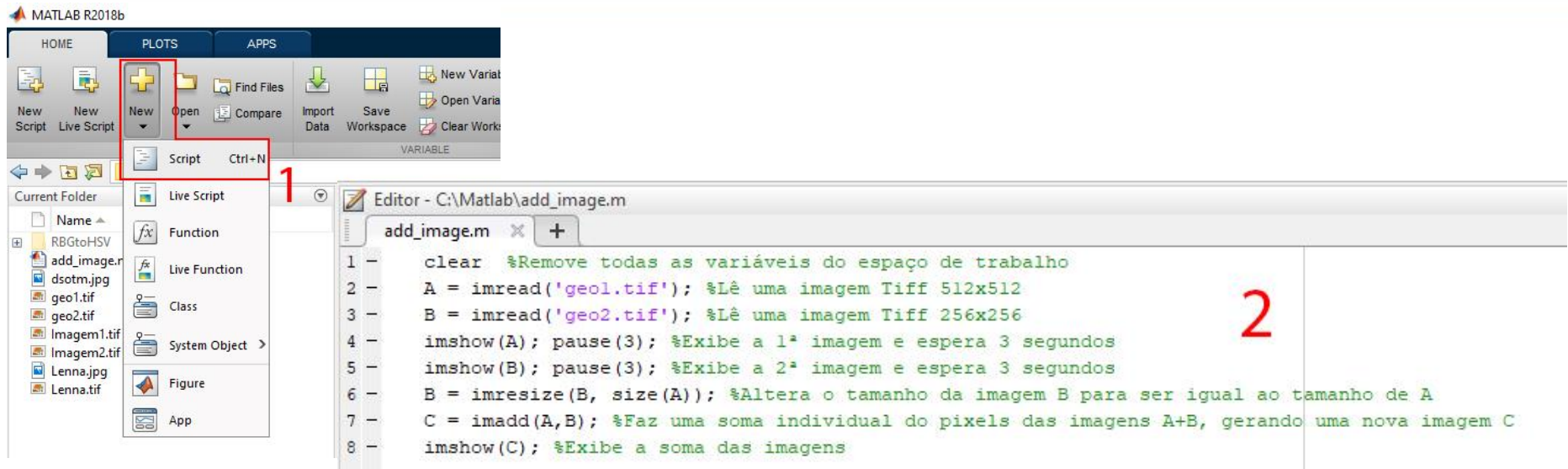
B



C

- A forma mais simples de programa de Matlab é o chamado arquivo de ***script***.
- Um arquivo de ***script*** é um arquivo de texto que contém uma sequência de comandos válidos do Matlab.
- No Matlab, os arquivos de ***script*** possuem a extensão **.M**

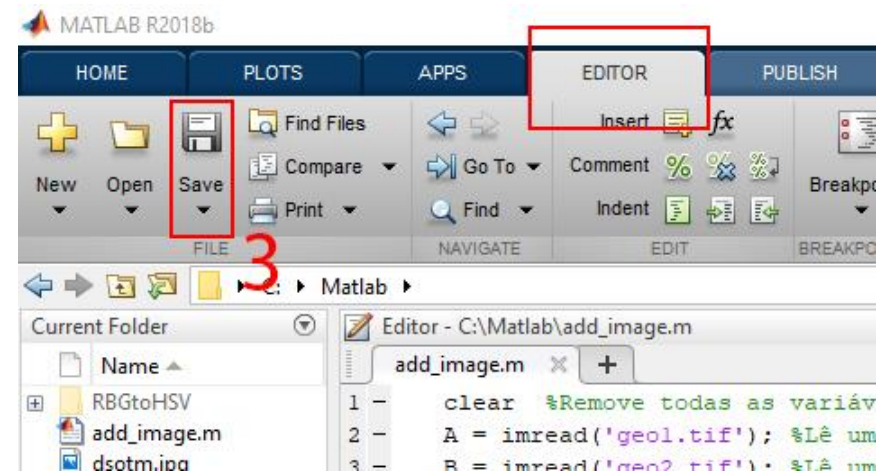
Criando um arquivo de script .M



1) Escolha a opção **New > Script** para criar um novo arquivo de texto para implementação do *script* de comandos do Matlab.

2) Digite o *script* com uma série de comandos que irão executar sequencialmente.

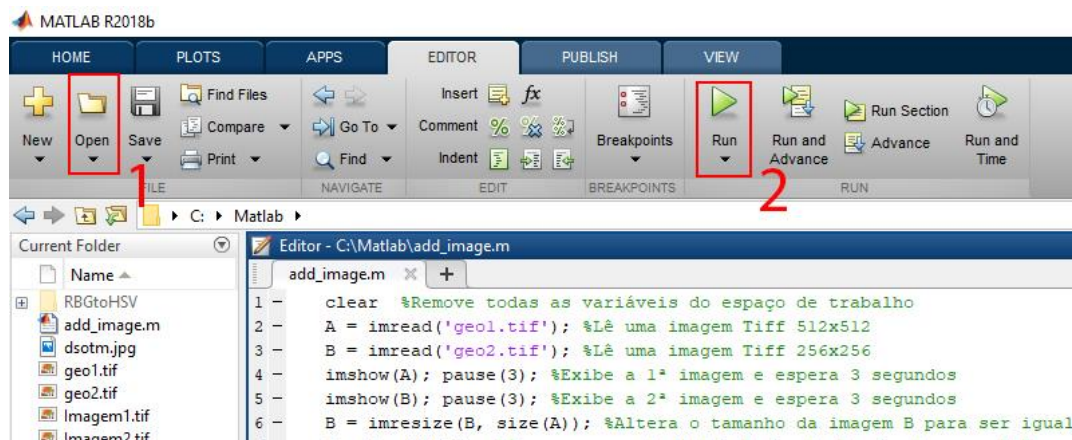
3) Na aba **Editor**, clique em **Save** para salvar o *script* com a extensão **.M**



Executando um script .M

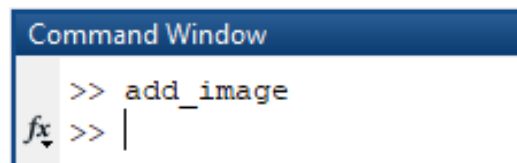
Método 1:

- 1) Na aba **Editor**, abra o arquivo de script .M na opção **Open**
- 2) Clique no botão **Run**.



Método 2:

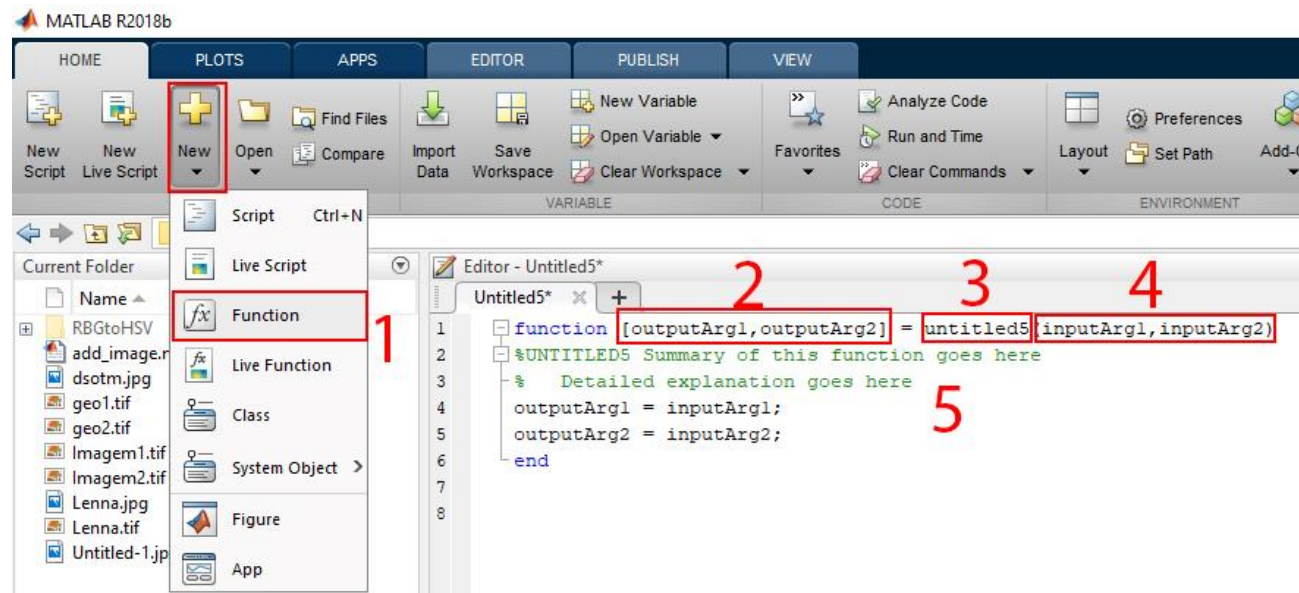
- 1) Se o script estiver no diretório de trabalho, apenas digite o nome do arquivo de script na janela de comandos.



Criando um arquivo de função

- O segundo tipo de arquivo de script .M é a função de Matlab.
- A principal diferença entre scripts e funções é que, quando uma função executa, somente os argumentos de saída declarados são retornados ao espaço de trabalho como variáveis.

- 1) Clique em **New > Function**
- 2) Na declaração da função, especificar uma ou mais argumentos de saída.
- 3) Atribuir o nome da função.
- 4) Definir os parâmetros de entrada da função.
- 5) Digitar a lógica dentro do corpo da função e finalizar a função com o comando **end**.



Criando um arquivo de função

Matlab R2018b

HOME PLOTS APPS EDITOR PUBLISH VIEW Search Documentation Sign In

FILE NAVIGATE EDIT BREAKPOINTS RUN

Current Folder: C:\Matlab\

- RBGtoHSV
- add_image.m
- dsotm.jpg
- geo1.tif
- geo2.tif
- geo3.tif
- Imagem1.tif
- Imagem2.tif
- Lenna.jpg
- Lenna.tif
- SubtrairImagens_Script.m
- SubtrairImagensFunc.m

Editor - C:\Matlab\SubtrairImagensFunc.m

```
1 function C = SubtrairImagens(A,B)
2 % Esta função aceita como entradas duas imagens A e B de
3 % tamanhos iguais. A saída é uma imagem representando a
4 % diferença C=A-B interpretada como dado de 8 bits (0-255)
5 % A imagem também é exibida na janela padrão de figuras
6 if size(A) ~= size(B)
7     disp('As imagens não tem o mesmo tamanho!');
8     return;
9 else
10    C = imsubtract(A,B);
11    imshow(C); % Exibe a imagem-diferença de 8 bits
12 end
```

Workspace

Name	Value
A	256x256 uint8
B	256x256 uint8
D	256x256 uint8

Command Window

```
>> clear % Remove todas as variáveis do espaço de trabalho
A = imread('geo2.tif'); % Lê a imagem A
B = imread('geo3.tif'); % Lê a imagem A
D = SubtrairImagensFunc(A,B); % Executa a função
whos; % Exibe variáveis
```

Name	Size	Bytes	Class	Attributes
A	256x256	65536	uint8	
B	256x256	65536	uint8	
D	256x256	65536	uint8	

fx >>

SubtrairImagens_Script.m (Scr...

Figure 1

File Edit View Inse Tool Desk Windc Hel

A B D

Construir Expressões Básicas em Matlab

- No Matlab, expressões empregam uma combinação de variáveis, operadores, números e funções.
- Por exemplo, na expressão a seguir:

```
>> x=1:10, y=sin(pi.*x./2)
```

x e **y** são variáveis.

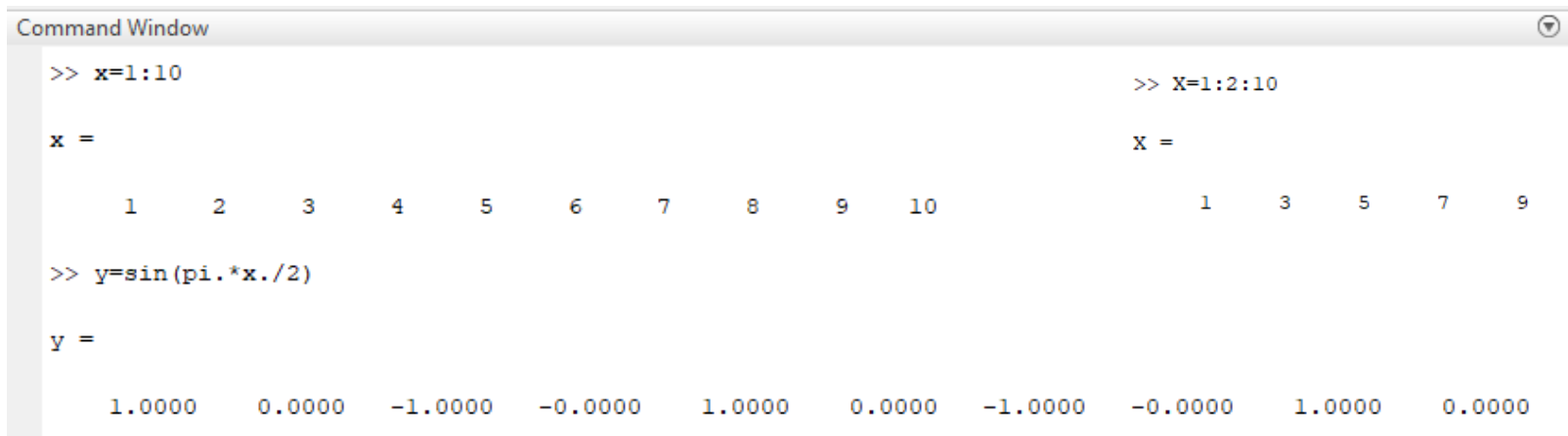
.* e **./** são operadores.

pi e **2** são números. (**pi** é um caso especial de uma constante numérica predefinida no Matlab).

sin é uma função.

Vetorização Implícita

- Digitando o primeiro comando, nota-se que x é um vetor de 10 elementos que contém os inteiros de 1 a 10.
- O segundo comando calcula o valor da expressão em cada valor do vetor de entrada x , produzindo um vetor de saída também de 10 elementos.
- Esse processo é conhecido como **Vetorização Implícita**, que ajuda a evitar laços iterativos e permite escrever expressões mais compactas e intuitivas.



```
Command Window
>> x=1:10
x =
     1     2     3     4     5     6     7     8     9    10

>> y=sin(pi.*x./2)
y =
  1.0000  0.0000 -1.0000 -0.0000  1.0000  0.0000 -1.0000 -0.0000  1.0000  0.0000

>> X=1:2:10
X =
     1     3     5     7     9
```

Vetorização Implícita

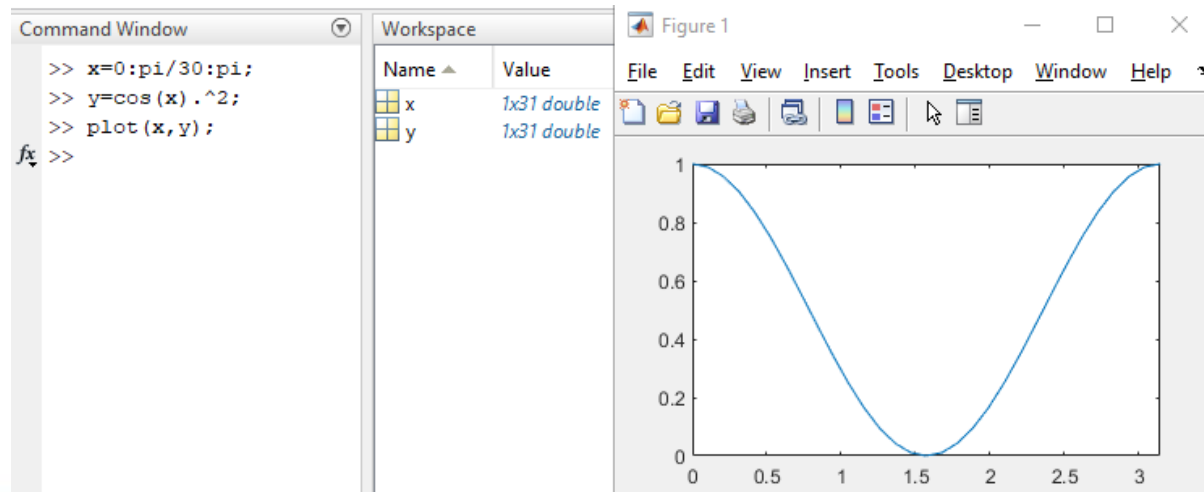
- Explore os seguintes exemplos:

```
>> x=0:pi/30:pi
```

%Define o vetor x de 30 elementos, com valores de 0 a pi

```
>> y=cos(x).^2; plot(x,y)
```

%Mostra $\cos^2 x$ nos valores especificados e faz um gráfico
%Gera $\cos^2 x$ e faz um gráfico da função



Vetorização Implícita

- Explore os seguintes exemplos:

```
>> A=[1 2; 3 4], B=[0 1; -1 0]    %Cria matrizes A e B 2 x 2
>> A-B                             %Subtrai a matriz B de A. Imprime o resultado na tela: ans = [1 1; 4 4]
>> A.^2                            %Eleva cada elemento de A ao quadrado. Imprime o resultado na
                                   %tela: ans = [1 4; 9 16]
```

Command Window

```
>> A=[1 2; 3 4], B=[0 1; -1 0]
```

A =

```
1    2
3    4
```

B =

```
0    1
-1   0
```

```
>> A - B
```

ans =

```
1    1
4    4
```

```
>> A .^ 2
```

ans =

```
1    4
9   16
```

Workspace

Name ▲	Value
A	[1,2;3,4]
ans	[1,4;9,16]
B	[0,1;-1,0]

Variáveis

- No Matlab, ***as variáveis não requerem comandos de declaração de tipos ou de tamanho.***
- Quando um novo nome de variável é encontrado, o espaço necessário para armazenagem é alocado automaticamente.

```
>> clear;           %Limpa o espaço de trabalho  
>> A=[1 0; 0 1]    %Espaço para armazenagem da matriz A 2 × 2 é criado automaticamente
```

- Adicionalmente, o espaço necessário para armazenagem da variável pode ser alterado dinamicamente.

Operadores

- Os operadores aritméticos no Matlab são, de acordo com a sua precedência:

Operador	Descrição
+	Adição
-	Subtração
./	Divisão (escalar)
.*	Multiplicação (escalar)
.^	Potenciação (escalar)
'	Transposto complexo conjugado
()	Especifica ordem de cálculo
/	Divisão (matricial)
*	Multiplicação (matricial)
^	Potenciação (matricial)

Operadores

Command Window

```
>> x=1:10, y=sqrt(x)
```

```
x =
```

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

```
y =
```

```
1.0000 1.4142 1.7321 2.0000 2.2361 2.4495 2.6458 2.8284 3.0000 3.1623
```

```
>> A=[1 2; 3 4], B=[0 1; -1 0], C = [2 1; 2 1]
```

```
A =
```

```
1 2  
3 4
```

```
B =
```

```
0 1  
-1 0
```

```
C =
```

```
2 1  
2 1
```

```
>> (A-B) - (B+C)
```

```
ans =
```

```
-1 -1  
3 3
```

Funções de conversão de tipos de dados

Function	Purpose
double	Converts to double precision number
single	Converts to single precision number
int8	Converts to 8-bit signed integer
int16	Converts to 16-bit signed integer
int32	Converts to 32-bit signed integer
int64	Converts to 64-bit signed integer
uint8	Converts to 8-bit unsigned integer
uint16	Converts to 16-bit unsigned integer
uint32	Converts to 32-bit unsigned integer
uint64	Converts to 64-bit unsigned integer

```
x = single([5.32 3.47 6.28]) .* 7.5
x = double([5.32 3.47 6.28]) .* 7.5
x = int8([5.32 3.47 6.28]) .* 7.5
x = int16([5.32 3.47 6.28]) .* 7.5
x = int32([5.32 3.47 6.28]) .* 7.5
x = int64([5.32 3.47 6.28]) .* 7.5
```

```
x =
    39.900    26.025    47.100

x =
    39.900    26.025    47.100

x =
    38    23    45

x =
    38    23    45

x =
    38    23    45

x =
    38    23    45
```

Operadores Matriciais e Escalares

Alerta!

É particularmente importante estar ciente, desde o início, da diferença entre as operações escalares de *divisão*, *multiplicação* e *potenciação* ($/$ $.$ $*$ $.$ $^$), que incluem um ponto antes do símbolo convencional, e seus equivalentes matriciais ($/$ $*$ $^$), que não incluem o ponto!

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

A =

```
1 2
3 4
```

```
>> B = [-1 3; 4 2]
```

B =

```
-1 3
4 2
```

```
>> BA1 = B .* A
```

BA1 =

```
-1 6
12 8
```

```
>> C = 5;
```

```
>> AC = A .* C
```

AC =

```
5 10
15 20
```

```
>> BA2 = B * A
```

BA2 =

```
8 10
10 16
```

$$B \cdot A = \begin{bmatrix} -1 & 3 \\ 4 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} (-1) \cdot 1 + 3 \cdot 3 & (-1) \cdot 2 + 3 \cdot 4 \\ 4 \cdot 1 + 2 \cdot 3 & 4 \cdot 2 + 2 \cdot 4 \end{bmatrix} = \begin{bmatrix} 8 & 10 \\ 10 & 16 \end{bmatrix}$$

Funções Matemáticas

- Matlab provê um grande número de funções matemáticas elementares.
- Para uma lista completa das funções elementares disponíveis, digite o comando:

>> help elfun

```
Command Window
>> help elfun
Elementary math functions.

Trigonometric.
  sin      - Sine.
  sind     - Sine of argument in degrees.
  sinh     - Hyperbolic sine.
  asin     - Inverse sine.
  asind    - Inverse sine, result in degrees.
  asinh    - Inverse hyperbolic sine.
  cos      - Cosine.
  cosd     - Cosine of argument in degrees.
  cosh     - Hyperbolic cosine.
  acos     - Inverse cosine.
  acosd    - Inverse cosine, result in degrees.
  acosh    - Inverse hyperbolic cosine.
  tan      - Tangent.
  tand     - Tangent of argument in degrees.
  tanh     - Hyperbolic tangent.
  atan     - Inverse tangent.
  atand    - Inverse tangent, result in degrees.
  atan2    - Four quadrant inverse tangent.
  atan2d   - Four quadrant inverse tangent, result in degrees.
  atanh    - Inverse hyperbolic tangent.
  sec      - Secant.
  secd     - Secant of argument in degrees.
  sech     - Hyperbolic secant.
```


Funções Matemáticas

Exponential.

- [exp](#) - Exponential.
- [expm1](#) - Compute $\exp(x)-1$ accurately.
- [log](#) - Natural logarithm.
- [log1p](#) - Compute $\log(1+x)$ accurately.
- [log10](#) - Common (base 10) logarithm.
- [log2](#) - Base 2 logarithm and dissect floating point number.
- [pow2](#) - Base 2 power and scale floating point number.
- [realpow](#) - Power that will error out on complex result.
- [reallog](#) - Natural logarithm of real number.
- [realsqrt](#) - Square root of number greater than or equal to zero.
- [sqrt](#) - Square root.
- [nthroot](#) - Real n-th root of real numbers.
- [nextpow2](#) - Next higher power of 2.

Rounding and remainder.

- [fix](#) - Round towards zero.
- [floor](#) - Round towards minus infinity.
- [ceil](#) - Round towards plus infinity.
- [round](#) - Round towards nearest integer.
- [mod](#) - Modulus (signed remainder after division).
- [rem](#) - Remainder after division.
- [sign](#) - Signum.

Complex.

- [abs](#) - Absolute value.
- [angle](#) - Phase angle.
- [complex](#) - Construct complex data from real and imaginary parts.
- [conj](#) - Complex conjugate.
- [imag](#) - Complex imaginary part.
- [real](#) - Complex real part.
- [unwrap](#) - Unwrap phase angle.
- [isreal](#) - True for real array.
- [cplxpair](#) - Sort numbers into complex conjugate pairs.

Arranjos (Matrizes e Vetores)

- Matlab foi originalmente projetado para, especificamente, permitir cálculo/manipulação de matrizes.
- Como objetos computacionais, matrizes são arranjos, ou seja, linhas e colunas ordenadas de valores numéricos.
- A forma mais simples para criar pequenos arranjos no Matlab é fornecer os elementos via janela de comandos, linha por linha.

Comandos Matlab

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

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

O que está sendo feito

```
%Cria uma matriz A  $2 \times 3$ 
```

```
%Cria uma matriz B  $2 \times 3$ 
```

Comentários

Os **colchetes** indicam que o conteúdo forma um arranjo. Um **ponto-e-vírgula** no interior dos colchetes indica o início de uma nova linha.

Arranjos (Matrizes e Vetores)

Um vetor é apenas um arranjo unidimensional:

```
>> vec=[0 1 1] %Cria um vetor-linha de 3 elementos, vec
```

Podemos transformar um vetor-linha em um vetor-coluna usando o operador de transposição:

```
>> vec=vec' %Define vec como a versão transposta dele próprio
```

ou, obviamente, transpor um arranjo bidimensional:

```
>> A' %Transpõe A
```

Podemos, também, criar arranjos de 'strings' que contêm texto:

```
>> alph=['ab';'cd'] %Cria um arranjo de caracteres  $2 \times 2$   
>> alph(2,2) %Exibe o elemento na 2ª linha e 2ª coluna (d)
```

Arranjos (Matrizes e Vetores)

Como acessar elementos individuais em um arranjo

Elementos individuais de um arranjo são acessados por meio de subscritos:

Comandos Matlab

```
>> vec(2)
```

```
>> A(1,2)
```

O que está sendo feito

%vec(2) é o segundo elemento de vec

%Retorna o elemento na 1ª linha e 2ª coluna

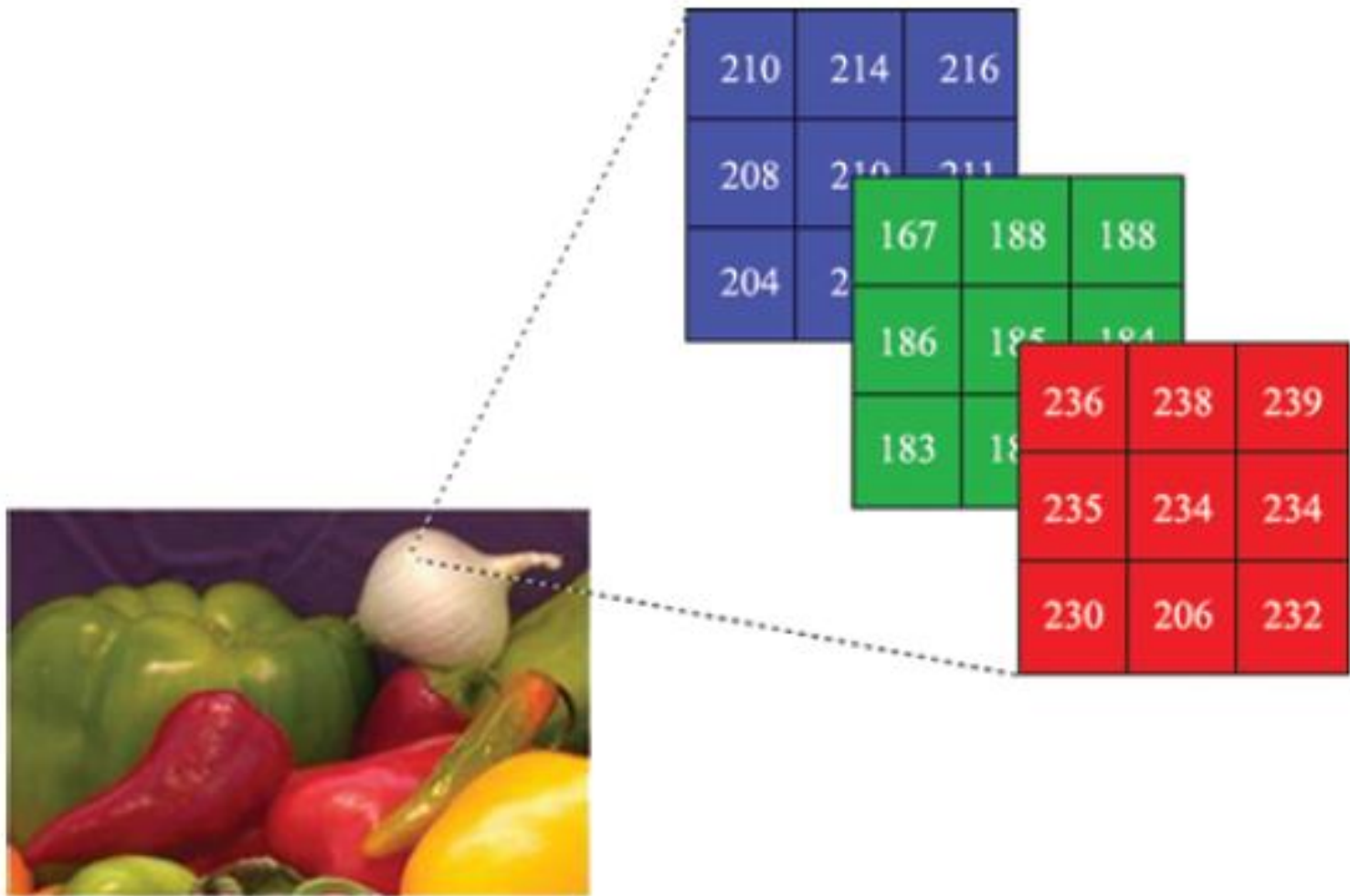
Podemos alterar ou corrigir valores de elementos de arranjos indexando-os individualmente:

```
>> A(1,2)=6
```

%Define o valor do elemento na 1ª linha e 2ª coluna como 6

- **Cuidado**: No Matlab, diferente de linguagens de programação como C/C++ ou Java, o índice do primeiro elemento dos arranjos é 1 e não 0.

Pixels de uma imagem colorida



Arranjos (Matrizes e Vetores)

```

Command Window

>> I = imread('Imagem3.tif')

10x10x3 uint8 array

I(:, :, 1) =

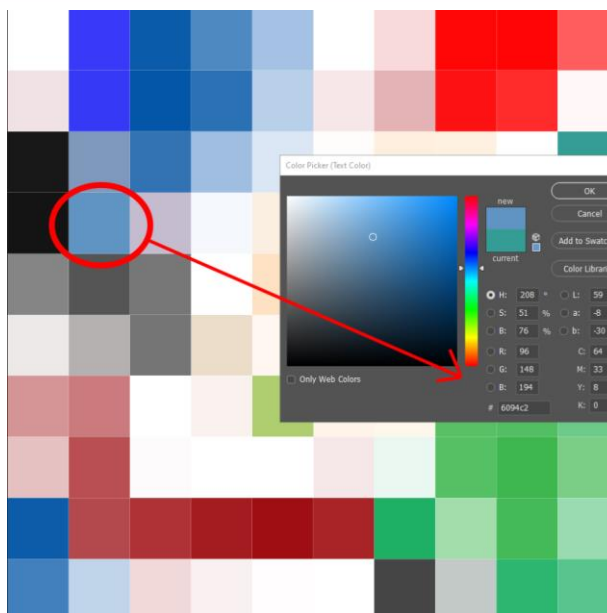
255  57  10  78  165  255  248  255  255  255
241  54   4  43  185  248  228  254  255  255
 24 126  51 159  220  255  254  254  255  54
 20  96 198  245  251  252  252  254  254  253
133  85 119  255  254  253  255  250  248  251
236 180 117  235  255  252  249  251  214  129
213 203 255  250  174  254  254  89  83  109
230 185 253  255  255  246  234  86  62  123
 12 180 174  165  159  168  30  162  68  154
 66 192 240 249  255  255  69  195  45  89

I(:, :, 2) =

255  57  91 137 194 255 218  7  6  94
227  57  87 113 208 231 179 17  44 248
 24 153 114 190 231 253 239 241 255 156
 20 148 188 248 239 203 202 229 245 224
133  85 119 255 226 206 252 194 165 207
233 177 117 221 247 187 177 196 238 211
149 123 255 242 206 244 247 192 191 202
193  79 251 255 255 232 248 192 183 206
 92  73  50  28  14  36 176 221 186 219
128 213 217 241 254 255  69 201 182 197

I(:, :, 3) =

255 248 170 194 229 255 219  7  6  94
229 246 168 182 235 233 181 10  44 248
 24 187 179 226 245 251 222 226 255 150
 20 194 207 252 226 150 148 203 234 189
133  85 119 255 195 157 250 126  64 153
233 177 117 202 240 118  91 131 221 169
151 125 255 239 111 233 236 102 100 136
194  82 251 255 255 232 241 100  79 134
170  77  54  32  18  39 101 171  88 178
189 233 217 241 254 255  69 198 112 142
    
```



- Por exemplo, para retornar o valor de um componente de cor de um pixel de uma imagem, o seguinte procedimento pode ser usado:

- 1) Ler uma imagem RGB (em cores)
- 2) Buscar dentro da matriz a posição da linha, coluna, componente de cor **R**(1) **G**(2) **B**(3) desejado:

```

>> I(4,2,1)      >> I(4,2,2)      >> I(4,2,3)

ans =             ans =             ans =

    uint8          uint8          uint8

    96             148             194
    R              G              B
    
```


Arranjos (Matrizes e Vetores)

Criado vetores:

- O operador de *dois-pontos* (`:`) pode ser usado para criar vetores e para definir subscritos de arranjos.
- Criando vetores com diferentes incrementos usando o operador de *dois-pontos*:

Comando de Matlab

```
>> x=1:10  
>> y=0:5:100
```

O que está sendo feito

%x = [1 2 3 ... 10]. O incremento é 1
%y = [0 5 10 ... 95 100]. O incremento é 5

Arranjos (Matrizes e Vetores)

Acessar grupos de elementos em um arranjo:

- O operador de *dois-pontos* (`:`) também pode ser usado para acessar grupos de elementos em um arranjo, especificando índices dos vetores:

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

```
A =
```

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

```
>> A1 = A(2:3, 3:4)
```

Linhas 2 e 3

```
A1 =
```

8	1
7	4

Colunas 3 e 4

Arranjos (Matrizes e Vetores)

Acessar todos os elementos de uma determinada linha ou coluna em um arranjo:

A =

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

```
>> A1 = A(:,1) % Extrai a 1ª coluna de A
```

A1 =

1
5
9
6

```
>> A2 = A(2,:) % Extrai a 2ª linha de A
```

A2 =

5 9 8 1 3

Exercícios

1) Carregue a imagem **SunMan.tif** no Matlab (use a função *imread*).

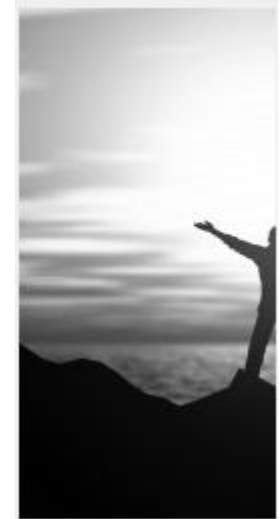


2) Extraia os 50% centrais dos pixels da imagem e exiba a imagem (use *imshow*).



Exercícios

3) Extraia as primeiras 128 colunas da imagem e as exiba como uma nova imagem.



4) Extraia as linhas 100 a 150, inclusive, da imagem e as exiba como uma nova imagem.



Concatenando Arranjos

- É possível concatenar (unir) arranjos e formar um arranjo maior.

```
>> A = [1 2; 3 4], B = [5 6; 7 8], C = [9 10 11 12]
```

A =

1	2
3	4

B =

5	6
7	8

C =

9	10	11	12
---	----	----	----

```
>> AB = [A B]
```

AB =

1	2	5	6
3	4	7	8

A, B são as **colunas** do novo arranjo, resultando em uma matriz 2x4

```
>> AB = [A;B]
```

AB =

1	2
3	4
5	6
7	8

A, B são as **linhas** do novo arranjo, resultando em uma matriz 4x2

```
>> ABC = [A B ; C]
```

ABC =

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

Concatenando Arranjos

- Os arranjos que formam os elementos de um arranjo concatenado *devem ter dimensões conformáveis*, ou seja, o arranjo resultante deve ser retangular.
- Por exemplo, a tentativa de formar arranjos a seguir não funcionará, gerando um erro:

```
>> A = [1 2; 3 4], B = [5 6; 7 8], C = [9 10 11 12]
```

A =

```
1    2
3    4
```

B =

```
5    6
7    8
```

C =

```
9    10    11    12
```

```
>> AC = [A;C]
```

Error using vertcat

Dimensions of arrays being concatenated are not consistent.

```
>> ABC = [A B C]
```

Error using horzcat

Dimensions of arrays being concatenated are not consistent.

Exercícios

5) Carregue as imagens **DuasCaras1.tif** e **DuasCaras2.tif** no Matlab (use a função *imread*).



6) Concatene essas imagens para formar uma única imagem e exiba o resultado (use a função *imagesc* ou *imshow*).



Exercícios

7) Aproveitando a imagem ao lado que foi gerada no exercício 5, crie um mosaico com imagens repetidas 2x3, como mostra o exemplo abaixo:

- Utilize a função **repmat** para criar a imagem estilo mosaico.



Criando arranjos simples com dimensões arbitrárias

- Matlab fornece funções embutidas para a criação e manipulação de arranjos simples de dimensões arbitrárias.

Função	Descrição	Exemplo simples
<code>zeros</code>	Arranjo de zeros	<code>A=zeros(4);</code>
<code>ones</code>	Arranjos de uns	<code>A=ones(4);</code>
<code>eye</code>	Matriz identidade	<code>A=eye(4);</code>
<code>repmat</code>	Faz cópias da matriz de entrada em uma estrutura de “azulejos”	<code>A=eye(4); B=repmat(A,1,2);</code>
<code>rand</code>	Arranjo aleatório – elementos entre 0 e 1	<code>A=rand(4);</code>
<code>randn</code>	Arranjo aleatório com distribuição normal	<code>A=randn(4);</code>
<code>linspace</code>	Vetor com incrementos lineares	<code>A=linspace(0,1,12);</code>
<code>logspace</code>	Vetor com incrementos logarítmicos	<code>A=logspace(0,1,12);</code>
<code>meshgrid</code>	Cria arranjo 2D a partir de 2 vetores de entrada	<code>[X,Y]=meshgrid(1:16,1:16);</code>

Função rand

```
>> A = rand
```

- Gera um número aleatório entre 0 - 1

```
A =
```

```
0.6553
```

```
>> A = rand(3)
```

- Gera uma matriz 3x3 com números aleatórios entre 0 – 1, seguindo uma distribuição uniforme.

```
A =
```

```
0.5074    0.8028    0.6503  
0.7879    0.1688    0.1860  
0.1175    0.2106    0.8007
```

```
>> A = rand(2,5)
```

- Gera uma matriz 2x5 com números aleatórios entre 0 – 1, seguindo uma distribuição uniforme.

```
A =
```

```
0.4997    0.1408    0.9336    0.3358    0.4251  
0.9217    0.1797    0.4451    0.2123    0.0729
```

Função rand

- Gera uma matriz 5x2 com números aleatórios entre -5 e 5, seguindo uma distribuição uniforme dos valores.

```
>> A = -5 + (5+5) * rand(5,2)
```

```
A =
```

```
-0.3949    2.0349  
 3.7849   -2.5296  
 0.1601   -3.1881  
 3.7859   -2.3672  
-3.6563    4.5312
```

- Em geral, pode-se gerar N números aleatórios no intervalo (a,b) com a fórmula: **$R = a + (b - a) .* \text{rand}(N,M)$**
- Onde:
 - **a** é o valor mínimo
 - **b** é o valor máximo
 - **N** é a quantidade de linhas
 - **M** é a quantidade de colunas

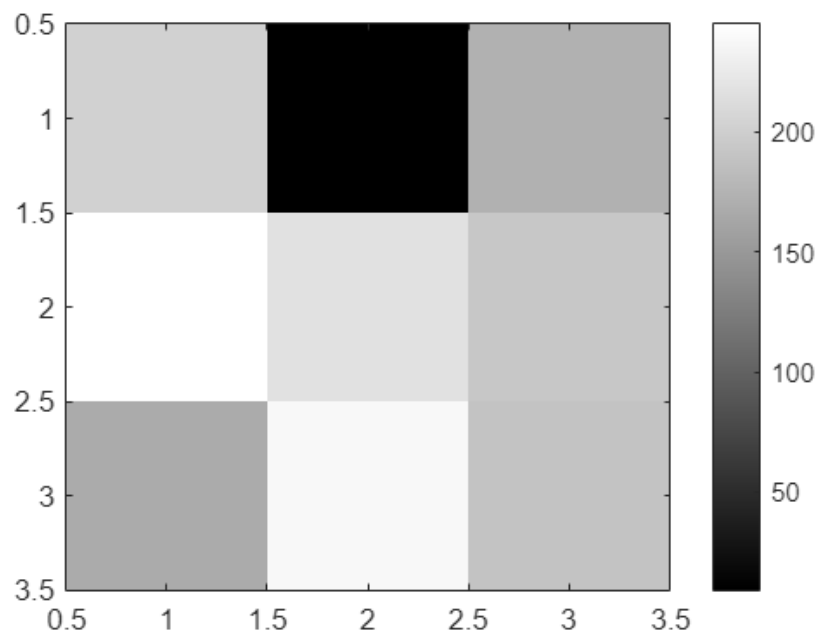
Função rand

- Gera uma matriz 3x3 com números aleatórios entre 0 e 255, seguindo uma distribuição uniforme dos valores.
- Converte os valores para **uint8**

```
A = uint8(0 + (255 + 0) * rand(3,3))  
  
imagesc(A);  
colorbar  
colormap("gray")
```

A = 3x3 uint8 matrix

202	9	173
245	217	193
167	238	189



Função rand

- Gera uma matriz 5x5x3 com números aleatórios entre 0 e 1, seguindo uma distribuição uniforme dos valores.

```
>> R = rand(5, 5, 3);  
>> imagesc(R)
```

val(:,:,1) =

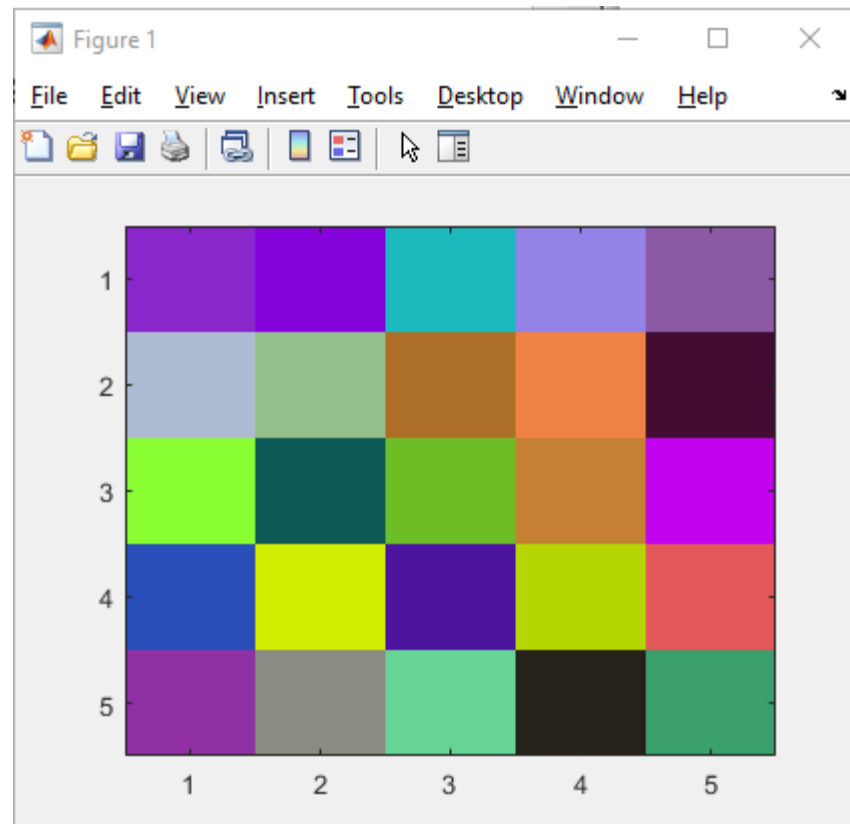
0.5375	0.5256	0.1089	0.5838	0.5508
0.6879	0.5812	0.6798	0.9314	0.2643
0.5358	0.0497	0.4265	0.7716	0.7590
0.1658	0.8253	0.2978	0.7135	0.8921
0.5668	0.5395	0.3973	0.1514	0.2303

val(:,:,2) =

0.1499	0.0146	0.7264	0.5091	0.3528
0.7332	0.7435	0.4357	0.5096	0.0467
0.9965	0.3496	0.7464	0.5021	0.0045
0.3062	0.9347	0.0774	0.8381	0.3446
0.1884	0.5490	0.8304	0.1275	0.6272

val(:,:,3) =

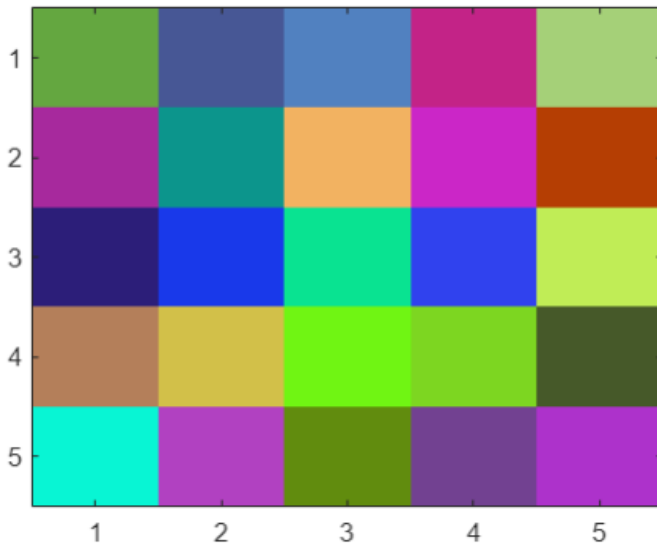
0.7991	0.8615	0.7421	0.9075	0.6456
0.8221	0.5472	0.1529	0.2622	0.2006
0.1981	0.3318	0.1472	0.2027	0.9354
0.7216	0.0003	0.6126	0.0101	0.3443
0.6385	0.5146	0.5822	0.1109	0.4273



Função rand

```
A = rand(5,5,3)
B = uint8(A * 255)
```

```
imagesc(A);
imagesc(B);
```



A =

A(:, :, 1) =

0.3922	0.2769	0.3171	0.7655	0.6463
0.6555	0.0462	0.9502	0.7952	0.7094
0.1712	0.0971	0.0344	0.1869	0.7547
0.7060	0.8235	0.4387	0.4898	0.2760
0.0318	0.6948	0.3816	0.4456	0.6797

A(:, :, 2) =

0.6551	0.3404	0.5060	0.1386	0.8143
0.1626	0.5853	0.6991	0.1493	0.2435
0.1190	0.2238	0.8909	0.2575	0.9293
0.4984	0.7513	0.9593	0.8407	0.3500
0.9597	0.2551	0.5472	0.2543	0.1966

A(:, :, 3) =

0.2511	0.5853	0.7537	0.5308	0.4694
0.6160	0.5497	0.3804	0.7792	0.0119
0.4733	0.9172	0.5678	0.9340	0.3371
0.3517	0.2858	0.0759	0.1299	0.1622
0.8308	0.7572	0.0540	0.5688	0.7943

B(:, :, 1) =

100	71	81	195	165
167	12	242	203	181
44	25	9	48	192
180	210	112	125	70
8	177	97	114	173

B(:, :, 2) =

167	87	129	35	208
41	149	178	38	62
30	57	227	66	237
127	192	245	214	89
245	65	140	65	50

B(:, :, 3) =

64	149	192	135	120
157	140	97	199	3
121	234	145	238	86
90	73	19	33	41
212	193	14	145	203

Função randi

```
>> R = randi(10)
```

- Gera um número aleatório entre 0 - 10

```
R =
```

```
4
```

```
>> R = randi(50, 3)
```

- Gera uma matriz 3x3 com números aleatórios entre 0 – 50, seguindo uma distribuição uniforme.

```
R =
```

```
30    46    49
30    33    48
12     3    43
```

```
>> R = randi(50, 3, 5)
```

- Gera uma matriz 3x5 com números aleatórios entre 0 – 50, seguindo uma distribuição uniforme.

```
R =
```

```
6    38    15     7    19
29    18    26    32    31
39    22    35    34    26
```

Função randi

- Para gerar números aleatórios inteiros, utilizar a função **randi** (ao invés de **rand**). O exemplo abaixo gera uma matriz 5x3 com números aleatórios entre 10 e 50:

```
>> R = randi([10 50], 5, 3)
```

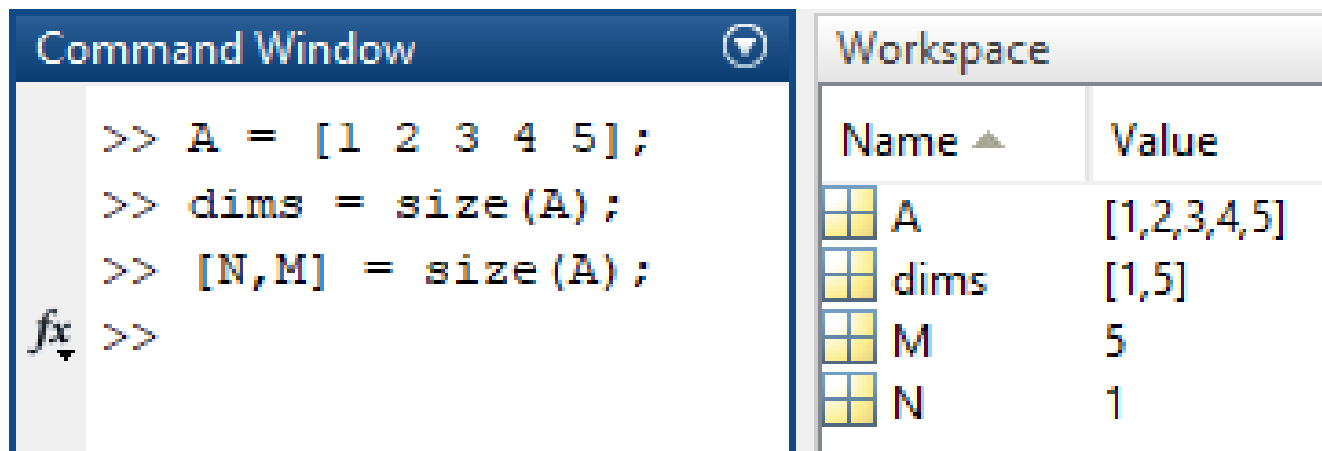
R =

49	44	27
46	34	15
21	44	33
31	25	39
20	36	19

- Use a fórmula: **R = randi([a b], N, M)**
- Onde:
 - **a** é o valor inteiro mínimo
 - **b** é o valor inteiro máximo
 - **N** é a quantidade de linhas
 - **M** é a quantidade de colunas

Determinar o tamanho de um arranjo

- Para determinar o tamanho de um arranjo e usar ou alocar essa informação a uma variável, usa-se a função **size**.
- A sintaxe básica é **[N, M] = size(A)**, em que os números de linhas e colunas da matriz **A** são alocados às variáveis **N** e **M**, respectivamente.



The screenshot displays the MATLAB Command Window and Workspace. The Command Window shows the following commands and their output:

```
>> A = [1 2 3 4 5];  
>> dims = size(A);  
>> [N,M] = size(A);  
fx >>
```

The Workspace window shows the following variables and their values:

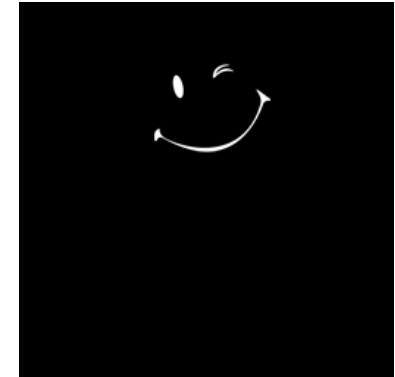
Name	Value
A	[1,2,3,4,5]
dims	[1,5]
M	5
N	1

Determinar o tamanho de um arranjo

- Tamanho de uma imagem Grayscale.

```
Command Window
>> A = imread('smile.tif');
>> dims = size(A);
>> [N,M] = size(A);
fx >>
```

Name ▲	Value
A	255x255 uint8
dims	[255,255]
M	255
N	255

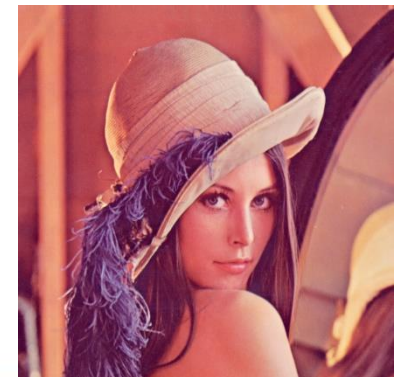


- Tamanho de uma imagem RGB.

- Imagens em cores RGB são compostas por 3 imagens 2D separadas, uma para cada um dos componentes (canais) Vermelho, Verde e Azul.*

```
Command Window
>> A = imread('lenna.jpg');
>> [N,M,RGB] = size(A);
>> dims = size(A);
fx >>
```

Name ▲	Value
A	512x512x3 uint8
dims	[512,512,3]
M	512
N	512
RGB	3



Funções para arranjos

- Matlab fornece diversas funções para manipulação de arranjos, tais como:

Função	Descrição
reshape	Altera a forma de um arranjo
fliplr	Inverte a ordem das colunas
flipud	Inverte a ordem das linhas
tril	Extraí a parte triangular inferior do arranjo
triu	Extraí a parte triangular superior do arranjo
rot90	Gira o arranjo de 90° no sentido anti-horário

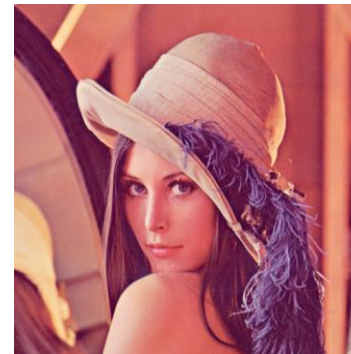
Funções para arranjos

- **fliplr** - Inverter a ordem das colunas:

```
>> A = imread('lenna.jpg');  
>> imshow(A);
```



```
>> B = fliplr(A);  
>> imshow(B);
```



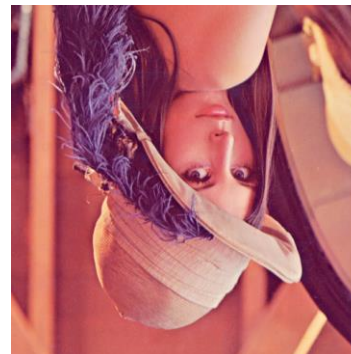
Funções para arranjos

- **flipud** - Inverter a ordem das linhas:

```
>> A = imread('lenna.jpg');  
>> imshow(A);
```



```
>> C = flipud(A);  
>> imshow(C);
```



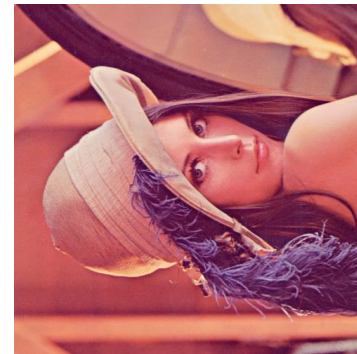
Funções para arranjos

- **rot90** – Gira o arranjo 90° no sentido anti-horário:

```
>> A = imread('lenna.jpg');  
>> imshow(A);
```



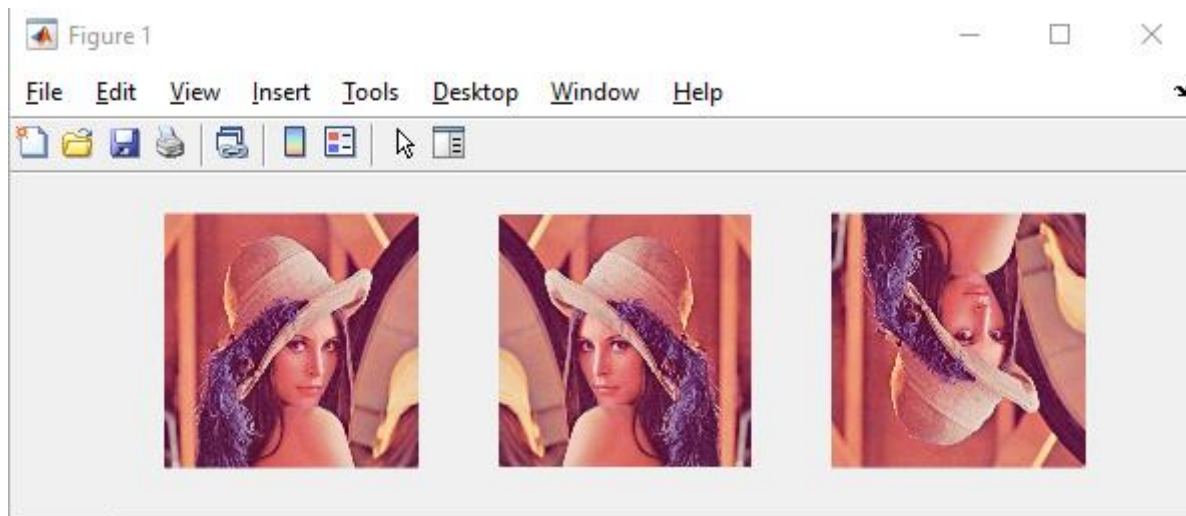
```
>> D = rot90(A);  
>> imshow(D);
```



Funções para arranjos

- **subplot** – Exibe imagens na mesma janela

```
>> subplot(1,3,1), imshow(A);  
>> subplot(1,3,2), imshow(B);  
>> subplot(1,3,3), imshow(C);
```



Técnica de Inversão de Imagem

- Na técnica de Inversão de Imagem, todos os elementos da matriz são substituídos para serem os elementos da linha superior para a linha inferior e os elementos da linha inferior para a linha superior.
- Em outras palavras, a imagem gira no eixo vertical.

```
% Parametros da função flip (1 - linhas 2 - columnas)
I = imread('onion.png'); % Original Image
I_mirror = flip(I, 2); % Mirror Image
I_reverse = flip(I, 1); % Reverse Image
I_mirrev = flip(I_reverse, 2); % Reverse + Mirror Image

figure,
subplot(2,2,1), imshow(I); title('Original Image');
subplot(2,2,2), imshow(I_mirror); title('Mirror Image');
subplot(2,2,3), imshow(I_reverse); title('Reverse Image');
subplot(2,2,4), imshow(I_mirrev); title('Reverse + Mirror Image');
```

Técnica de Inversão de Imagem

Original Image



Mirror Image



Reverse Image



Reverse + Mirror Image



Técnica de Rotação de Imagem

- **imrotate** – Rotaciona uma imagem usando um ângulo (em graus) e um método de interpolação:

`I_rotate = imrotate(Image Matrix Variable, Angle, Interpolation Method)`

Interpolation method

- 'nearest': Nearest-Neighbor Interpolation
- 'bilinear': Bilinear Interpolation
- 'bicubic': Bicubic Interpolation

Técnica de Rotação de Imagem

```
I = imread('pout.tif');  
I_rotate_nearest = imrotate(I, 60, "nearest");  
I_rotate_bilinear = imrotate(I, 90, "bilinear");  
I_rotate_bicubic = imrotate(I, 180, "bicubic");  
subplot(1,4,1), imshow(I); title('Original Image');  
subplot(1,4,2), imshow(I_rotate_nearest); title('Image Rotate Nearest');  
subplot(1,4,3), imshow(I_rotate_bilinear); title('Image Rotate Bilinear');  
subplot(1,4,4), imshow(I_rotate_bicubic); title('Image Rotate Bicubic');
```

Original Image



Image Rotate Nearest

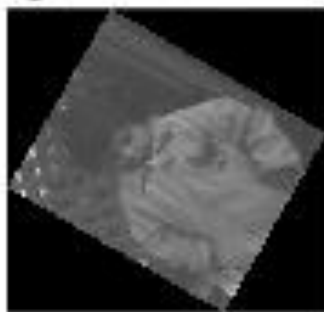


Image Rotate Bilinear



Image Rotate Bicubic



Técnica de Deslocamento de Imagem

- **imtranslate** – Reposiciona os pixels de uma imagem para outra posição na matriz. A translação pode ser feita nos eixos horizontal e vertical.

```
I = imread("pout.tif"); % Imagem Original
I_translate = imtranslate(I, [15,25]); % Translação (Shift)
subplot(1,2,1); imshow(I); title('Original Image');
subplot(1,2,2); imshow(I_translate); title('Shifting Image');
```

Original Image



Shifting Image

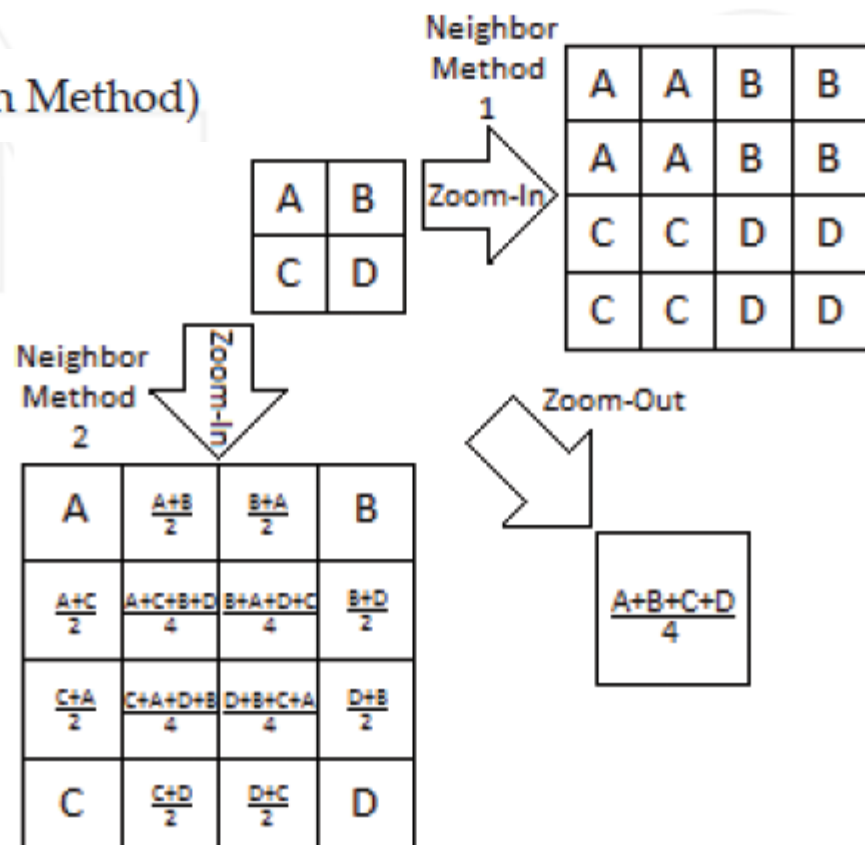


Técnica de modificação de resolução de Imagem

- **imresize** – Modifica a resolução de uma imagem digital, aplicando um efeito de *zoom-in* ou *zoom-out*. Se especificado um método de interpolação, a qualidade da imagem final é alterada.

$I_{\text{resize}} = \text{imresize}(I, \text{Resize Rate}, \text{Interpolation Method})$

- Se o parâmetro *Resize Rate* = 1, a imagem não sofre alterações.
- Se o parâmetro *Resize Rate* > 1, a imagem sofre Zoom-in.
- Se o parâmetro *Resize Rate* < 1, a imagem sofre Zoom-out.



Técnica de modificação de resolução de Imagem

```
I = imread("pout.tif");  
% Aumenta em 4x o tamanho da imagem (Zoom-in)  
I_zoomIn = imresize(I, 4);  
% Diminui em 80% o tamanho da imagem (Zoom-out)  
I_zoomOut = imresize(I, 0.2);  
subplot(1,3,1); imshow(I); title('Imagem Original');  
subplot(1,3,2); imshow(I_zoomIn); title('Zoom-In');  
subplot(1,3,3); imshow(I_zoomOut); title('Zoom-Out');
```

Imagem Original



Zoom-In



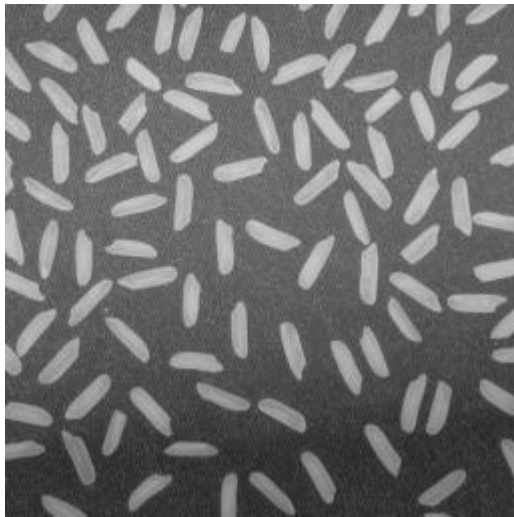
Zoom-Out



Exercícios

8) Carregue as imagens **rice.png** e **cameraman.tif**

- Reflita a imagem **rice** e relação ao eixo x, a imagem **cameraman** em relação ao eixo y e some as duas imagens com o operador de adição comum do Matlab.
- Exiba o resultado com a função ***imshow***.



Imagens Originais



Imagem Final

Operadores Comparativos e Lógicos

- Matlab inclui todos os operadores comparativos e lógicos convencionais usados em linguagens de programação.
- Esses operadores são usados geralmente em conjunto com comandos de controle de fluxo (IF).

Operador comparativo	Descrição
<	Menor que
<=	Menor ou igual a
~=	Não igual a
>	Maior que
>=	Maior ou igual a
==	Igual a

Operadores Comparativos e Lógicos

- **Operadores comparativos** podem ser usados para comparar dois arranjos ou um arranjo e um escalar.
- A saída de todas as expressões comparativas produzem conjuntos lógicos que contêm **1** (true) ou **0** (false).
- Note que a comparação é feita elemento a elemento.

```
>> A = [1 3 2 6];  
>> B = [0 4 3 4];  
>> R = A < B
```

R =

1×4 logical array

0 1 1 0

```
>> X = 3;  
>> R = (A == X)
```

R =

1×4 logical array

0 1 0 0

Operadores Comparativos e Lógicos

- **Operadores lógicos** permitem a combinação ou negação de expressões comparativas.
- A aplicação desses operadores a expressões comparativas também resulta em arranjos lógicos (0-1), que contêm o valor **1** onde a expressão é TRUE e o **0**, onde é FALSE.

Operador lógico de Matlab	Descrição
&	AND lógico (E lógico)
	OR lógico (OU lógico)
~	NOT lógico (NÃO lógico)

Operadores Comparativos e Lógicos

Aloca matrizes:

```
>> A = [1 2 3];  
>> B = [0 2 4];  
>> C = [1 3 2];
```

Somente o último elemento de B
satisfaz as duas condições:

```
>> (B > A) & (B > C)
```

```
ans =
```

```
1×3 logical array
```

```
0    0    1
```

NÃO lógico do exemplo anterior:

```
>> ~( (B > A) & (B > C) )
```

```
ans =
```

```
1×3 logical array
```

```
1    1    0
```

Todos os elementos de B são
diferentes dos elementos de C:

```
>> (A < C) | (B ~= C)
```

```
ans =
```

```
1×3 logical array
```

```
1    1    1
```


Operadores Comparativos e Lógicos

- **Operadores lógicos** permitem a combinação ou negação de expressões comparativas.
- A aplicação desses operadores a expressões comparativas também resulta em arranjos lógicos (0-1), que contêm o valor **1** onde a expressão é TRUE e o **0**, onde é FALSE.

Função lógica de Matlab	Descrição
and	AND lógico (E lógico)
or	OR lógico (OU lógico)
xor	OR exclusivo lógico (OU exclusivo lógico)
any	TRUE lógico se qualquer elemento for TRUE (VERDADE)
all	TRUE lógico se todos os elementos forem TRUE (VERDADE)
isempty	TRUE lógico se a matriz for vazia
isequal	TRUE lógico se matrizes forem idênticas
ismember	

Operadores Comparativos e Lógicos

Aloca matrizes:

```
>> A = [1 2 3];  
>> B = [0 2 4];  
>> C = [1 3 2];
```

E lógico entre arranjos A
< B < C: ans = [0 0 1]

```
>> and(A < B, C < B)
```

```
ans =
```

```
1×3 logical array
```

```
0    0    1
```

OU exclusivo entre arranjos

A < B e B < C: ans = [1 1 1]

```
>> xor(A < B, B < C)
```

```
ans =
```

```
1×3 logical array
```

```
1    1    1
```

A não tem elementos maiores que 4. A função **find** retorna o arranjo vazio. **isempty** testa isso. ans = 1

```
>> find(A > 4)
```

```
ans =
```

```
1×0 empty double row vector
```

Se qualquer elemento de
B > A, retorna TRUE
lógico: ans = 1

```
>> any(B > A)
```

```
ans =
```

```
logical
```

```
1
```

```
>> isempty(find(A > 4))
```

```
ans =
```

```
logical
```

```
1
```

Controle de Fluxo - IF

- Para controle de fluxo, o Matlab suporta o comando IF e as seguintes variantes da estrutura:

if ... (comandos) ... end

if ... (comandos) ... else ... (comandos) ... end

if ... (comandos) ... elseif ... (comandos)... else ... (comandos) ... end

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

```
>> x = 3;
```

```
>> if (find(A == x))
```

```
disp('Encontrei X')
```

```
else
```

```
disp('Não encontrei X')
```

```
end
```

```
Encontrei X
```

```
>> A = [ 2 9 -5 4];
```

```
>> B = [10 -3 -4 -6];
```

```
>> x = randi([-10, 10]);
```

```
>> r = A(2) + B(4);
```

```
>> if (r > x)
```

```
msg = sprintf('r: %d é maior do que x: %d', r, x);
```

```
disp(msg);
```

```
else
```

```
msg = sprintf('r: %d é menor do que x: %d', r, x);
```

```
disp(msg);
```

```
end
```

```
r: 3 é menor do que x: 10
```

Workspace	
Name ^	Value
A	[2,9,-5,4]
B	[10,-3,-4,-6]
msg	'r: 3 é menor do que x: 10'
r	3
x	10

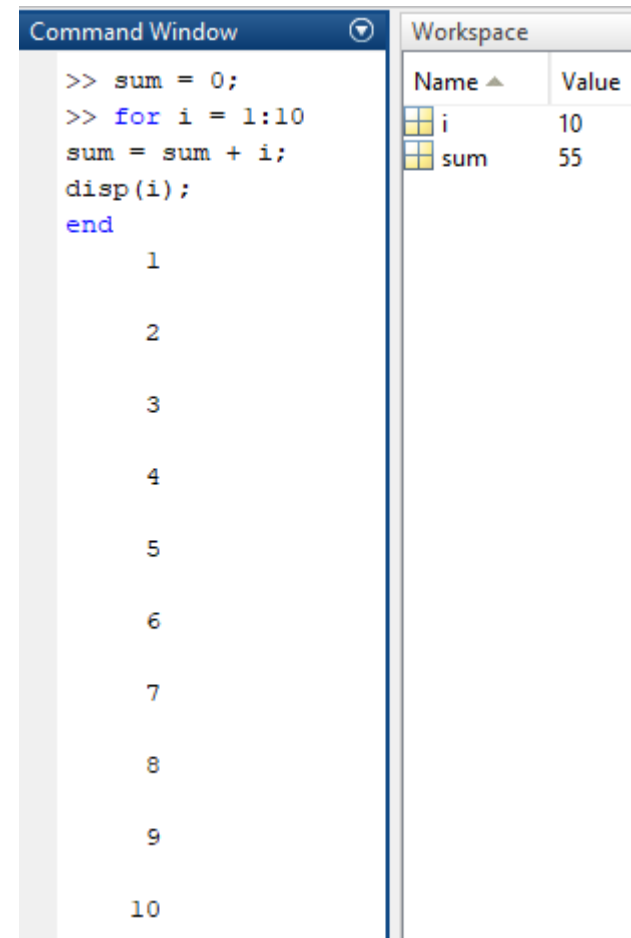
Laço FOR

- Laços FOR permitem que um conjunto de comandos seja executado certo número finito de vezes.

- A sintaxe é:

```
for x = arranjo  
    comandos  
end
```

- O laço FOR basicamente aloca à variável **x** a próxima coluna do vetor arranjo a cada iteração.



The screenshot shows the MATLAB Command Window and Workspace. The Command Window displays the following code and its output:

```
>> sum = 0;  
>> for i = 1:10  
    sum = sum + i;  
    disp(i);  
end  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

The Workspace window shows the current state of variables:

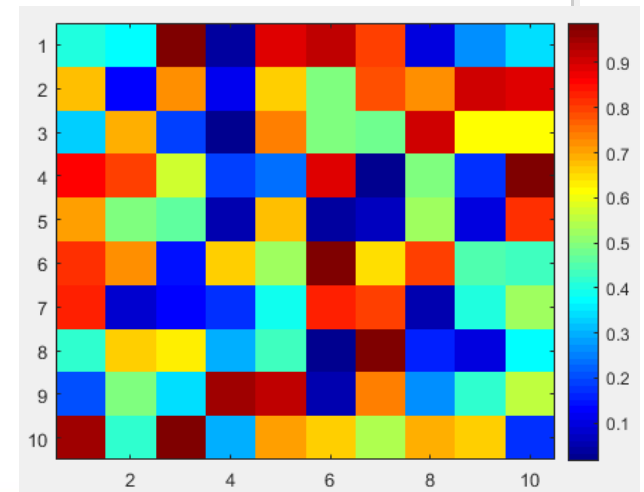
Name	Value
i	10
sum	55

Laço FOR

- Uma forma de construir e percorrer arranjos é com o uso de estruturas de repetição (loops).
- O exemplo abaixo aloca valores aleatórios em uma matriz 10x10.

```
>> A = zeros(10); % Cria uma matriz 10x10 preenchida com zeros
>> for i = 1:size(A) % i controla as linhas da matriz 1 à 10
for j = 1:size(A) % j controla as colunas da matriz 1 à 10
A(i,j) = rand; % Gera e aloca um número aleatório entre 0 e 1 na matriz
end % Fecha o bloco das colunas (j)
end % Fecha o bloco das linhas (i)

>> imagesc(A); % Imprime a imagem final
```

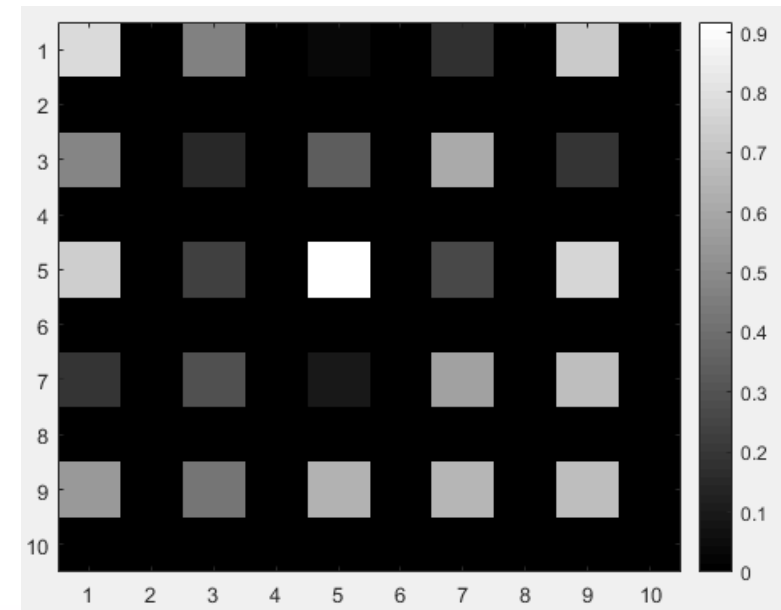


Laço FOR

- O exemplo abaixo aloca valores aleatórios em uma matriz 10x10, porém, o incremento é de 2 em 2.

```
>> A = zeros(10); % Cria uma matriz 10x10 preenchida com zeros
>> for i = 1:2:size(A) % i controla as linhas da matriz 1 à 10
for j = 1:2:size(A) % j controla as colunas da matriz 1 à 10
A(i,j) = rand;
end
end
>> imagesc(A);
```

	1	2	3	4	5	6	7	8	9	10
1	0.7847	0	0.4714	0	0.0358	0	0.1759	0	0.7218	0
2	0	0	0	0	0	0	0	0	0	0
3	0.4735	0	0.1527	0	0.3411	0	0.6074	0	0.1917	0
4	0	0	0	0	0	0	0	0	0	0
5	0.7384	0	0.2428	0	0.9174	0	0.2691	0	0.7655	0
6	0	0	0	0	0	0	0	0	0	0
7	0.1887	0	0.2875	0	0.0911	0	0.5762	0	0.6834	0
8	0	0	0	0	0	0	0	0	0	0
9	0.5466	0	0.4257	0	0.6444	0	0.6476	0	0.6790	0
10	0	0	0	0	0	0	0	0	0	0

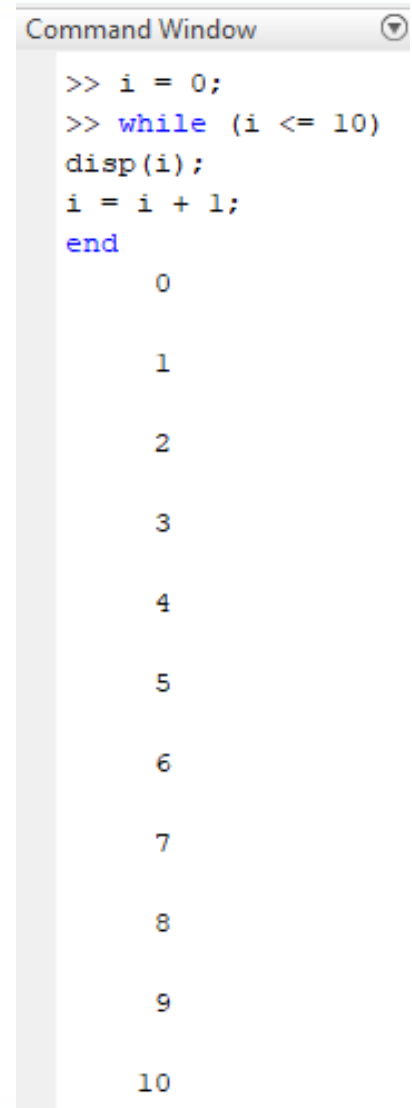


Laço WHILE

- A sintaxe do laço WHILE:

```
while expressão  
    comandos  
end
```

- No laço WHILE, **expressão** é uma expressão lógica e os **comandos** seguintes são executados repetidas vezes até que a expressão no comando *while* se torne logicamente **falsa**.



```
Command Window
```

```
>> i = 0;  
>> while (i <= 10)  
    disp(i);  
    i = i + 1;  
end
```

0
1
2
3
4
5
6
7
8
9
10

Exercícios

9) Escreva uma script no Matlab que carregue uma matriz 2D (uma imagem em grayscale) e:

- Mude a escala da matriz de modo que o máximo valor de pixel passe a ser 0 - 1.
- Calcule o valor médio de pixels da matriz alterada.
- Imprima as seguintes mensagens condicionadas na tela:
 - “A imagem é escura” se média < 0.4
 - “A imagem é normal” se média ≥ 0.4 E média < 0.6
 - “A imagem é clara” se média ≥ 0.6
- Teste sua função em uma ou duas imagens para verificar se funciona corretamente.

Exercícios

10) Escreva uma script no Matlab que carregue uma matriz 2D (uma imagem em grayscale) e:

- Para cada pixel da imagem, aumente ou diminua a sua intensidade, atribuindo um valor que será somado ou diminuído ao valor atual do pixel.
- Certifique-se de que o novo valor do pixel não exceda os limites 0 – 255.
- Exiba a imagem final.

Indexação de Arranjos Unidimensionais

- Indexação de um arranjo é uma forma de selecionar um subconjunto de elementos de um arranjo.

```
>> v = [16 5 9 4 2 11 7 14];
```

```
>> v(3)
```

```
%O subscrito pode ser um único valor  
%“Extrai o terceiro elemento” ans = 9
```

```
>> v([1 5 6])
```

```
%Extrai o primeiro, o quinto e o sexto elementos  
%ans = 16  2  11
```

```
>> v(3:7)
```

```
%Extrai os elementos 3 a 7, inclusive  
%ans =  9  4  2  11  7
```

Indexação de Arranjos Unidimensionais

```
>> v = [16 5 9 4 2 11 7 14];
```

```
>> v2 = v([5:8 1:4])
```

%Extrai e troca as duas metades de v
%v2 = 2 11 7 14 16 5 9 4

```
>> v(end)
```

%Extrai o último elemento
%ans = 14

```
v(5:end)
```

%Extrai do quinto ao último elementos, inclusive
%ans = 2 11 7 14

Indexação de Arranjos Unidimensionais

```
>> v = [16 5 9 4 2 11 7 14];
```

```
v(2:end-1)
```

```
%Extrai do segundo ao penúltimo elementos, inclusive  
%ans = 5  9  4  2  11  7
```

```
>> v([2 3 4]) = [10 15 20]
```

```
%Substitui elementos selecionados de v  
%v = 16  10  15  20  2  11  7  14
```

```
>> v([2 3]) = 30
```

```
%Substitui o segundo e o terceiro elementos por 30  
%v = 16  30  30  20  2  11  7  14
```

Indexação de Arranjos Bidimensionais

- **$M = \text{magic}(n)$** retorna uma matriz n -by- n composta por inteiros entre 1 e n^2 , onde a soma das linhas e colunas são exatamente iguais.
- **n** deve ser um inteiro maior ou igual a 3.

```
>> A = magic(4)
```

%Cria quadrado mágico 4×4 usando a
%função embutida *magic*

	1	2	3	4
1	16	2	3	13
2	5	11	10	8
3	9	7	6	12
4	4	14	15	1

Indexação de Arranjos Bidimensionais

```
>> A(2,4)
```

```
%Extrai o elemento na linha 2, coluna 4  
%ans = 8
```

```
>> A(2:4,1:2)
```

```
%Extrai linhas 2 a 4, colunas 1 a 2
```

```
>> A(3,:)
```

```
%Extrai a terceira linha  
%ans =      9   7   6  12
```

```
>> A(:,end)
```

```
%Extrai a última coluna  
%ans = 13 8 12 1
```

Função find

- A função **find** pode ser usada para extrair os índices de todos os elementos de um arranjo que satisfazem dada condição.

```
>> clear; A=1:10; i=find(A>5)           %i = [6 7 8 9 10]
```

Isso também pode ser usado em arranjos 2D para retornar os índices *linha-coluna*:

```
>> clear; A=[1 2 3; 0 1 2; -1 1 1]; [i,j]=find(A<=0)   %i=[2 3] j=[1 1]
```

Ou para extrair os *índices lineares* do arranjo 2D:

```
>> i=find(A<=0)           %i=[2 3]
```

Lista de Exercícios 1

Exercício 1.1 Usando os exemplos apresentados para a exibição de uma imagem em Matlab e para acessar uma posição de pixel, explore a adição/subtração de um valor escalar a/de uma posição de pixel, ou seja, $I(i, j) = I(i, j) + 25$ ou $I(i, j) = I(i, j) - 25$. Comece com a imagem em escala de cinza 'cell.tif' dos exemplos e posição de pixel (100, 20). Qual é o efeito da adição e da subtração na cor da escala de cinza?

Estenda a técnica a imagens em cores RGB, adicionando/subtraindo valores aos/dos três canais de cor, usando uma imagem de exemplo apropriada. Tente, ainda, adicionar valor a apenas um dos canais, deixando os outros dois inalterados. Qual é o resultado de cada uma dessas operações na cor do pixel?

Exercício 1.2 Com base na resposta ao Exercício 1.1, use a estrutura *for* de Matlab (para obter ajuda, use o comando *help for* na janela de comandos do Matlab) para varrer todos os pixels na imagem bem como escurecer ou clarear a imagem.

Você deve assegurar que seu programa não tente criar um valor de pixel que seja maior ou menor do que o valor que o pixel pode assumir. Por exemplo, uma imagem de 8 bits pode assumir valores na faixa 0–255 em cada posição de pixel, e o mesmo se aplica a cada canal de cor em uma imagem RGB em cores de 24 bits.

Lista de Exercícios 1

Exercício 1.3 Usando a imagem em escala de cinza 'cell.tif' dos exemplos, explore a exibição da imagem com diferentes mapas em falsa cor. A função de Matlab ***colormap*** pode assumir uma faixa de valores para especificar diferentes mapas de falsa cor: na janela de comando de Matlab, use o comando *help graph3d* e examine a lista de mapas de cor apresentada na seção *Color maps* [Mapas de cor]. Que diferentes aspectos e detalhes da imagem podem ser vistos se estes mapas de falsa cor forem usados no lugar da convencional escala de cinza?

Mapas de falsa cor também podem ser especificados numericamente como parâmetros no comando ***colormap***: use *help colormap* para obter mais informação.

Lista de Exercícios 1

Exercício 1.4 Carregue uma imagem em Matlab e, usando as funções apresentadas no Exemplo 1.1, salve-a uma vez como um arquivo no formato JPEG (por exemplo, teste.jpg) e outra no formato PNG (por exemplo, teste.png). A seguir, carregue as imagens dos dois arquivos que acabou de salvar como novas imagens em Matlab, 'ljpg' e 'lpng'.

Podemos esperar que essas duas imagens sejam exatamente iguais, pois tiveram origem em uma mesma imagem e foram salvas em diferentes formatos de arquivo. Se as compararmos subtraindo uma da outra e tomando a diferença absoluta em cada posição de pixel, podemos verificar se esta hipótese é válida.

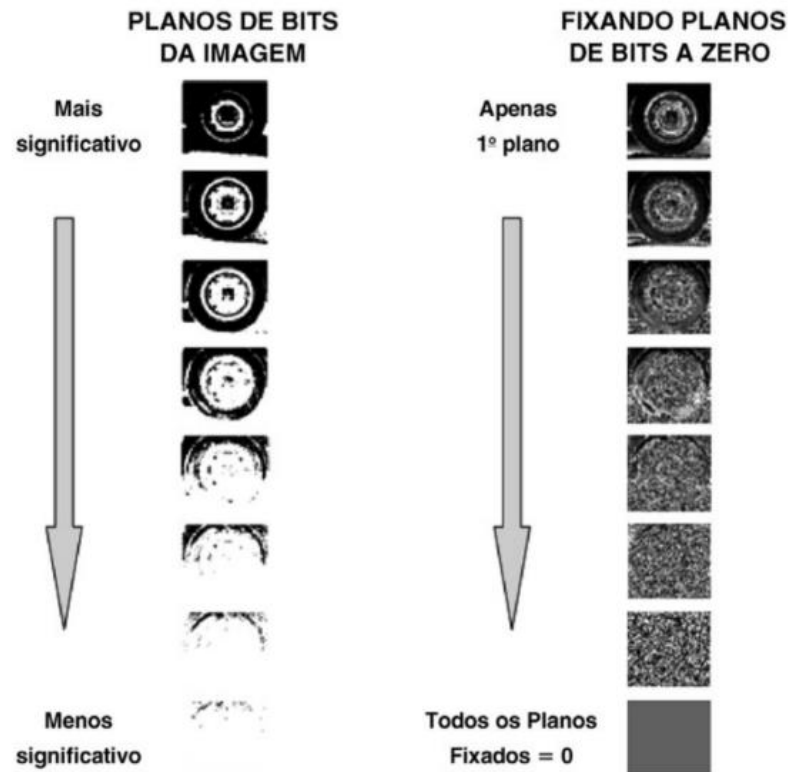
Use o comando de Matlab *imabsdiff* para gerar uma imagem-diferença entre 'ljpg' e 'lpng'. Use *imagesc* para exibir a imagem resultante.

A diferença entre estas duas imagens não é zero, como podíamos esperar; há um padrão de ruído relacionado à diferença entre as imagens introduzida pelo uso de compressão com perda (JPEG) e sem perda (PNG). A diferença que vemos deve-se à informação removida da imagem na versão JPEG. Se exibíssemos a diferença com *imshow*, tudo o que veríamos seria uma imagem preta, pois as diferenças são tão pequenas que os correspondentes pixels têm valores muito baixos (ou seja, escuros). O ajuste automático de valores e o mapa de falsa cor de *imagesc* permitem que visualizemos estes pequenos valores de pixels.

Lista de Exercícios 1

Exercício 1.5 Implemente um programa para efetuar a decomposição em plano de bits e extrair/exibir os resultantes planos de imagem como imagens independentes em Matlab.

Para exibir um mosaico de diferentes planos de bit de uma imagem, considere o uso da função *subplot*.



Exemplo de decomposição em planos de bits de uma imagem em escala de cinza.

Lista de Exercícios 1

Exercício 1.6 Usando a função *rgb2hsv* de Matlab, escreva um programa para exibir os canais de matiz (*hue*), saturação (*saturation*) e valor (*value*) de uma dada imagem em cores RGB.



Original



Red Channel



Green Channel



Blue Channel

Imagem Original



Matiz (tonalidade)



Saturação (intensidade)



Valor (Luminância)

