

Rio de Janeiro, 10 de setembro de 2017

Universidade do Estado do Rio de Janeiro

Instituto de Matemática e Estatística

**Computação Gráfica**

**Trabalho 1 – Exercícios de Introdução ao Octave**

**Aluno:** Leonardo Lima Marinho

**Matrícula:** 2014.1.00506.11

**1. Confira os tipos de dados do Octave/Matlab através do comando: typeinfo**

>> typeinfo

ans =

{

[1,1] = <unknown type>

[2,1] = cell

[3,1] = scalar

[4,1] = complex scalar

[5,1] = matrix

[6,1] = diagonal matrix

[7,1] = complex matrix

[8,1] = complex diagonal matrix

[9,1] = range

[10,1] = bool

[11,1] = bool matrix

[12,1] = string

[13,1] = sq\_string

[14,1] = int8 scalar

[15,1] = int16 scalar

[16,1] = int32 scalar

[17,1] = int64 scalar

[18,1] = uint8 scalar

[19,1] = uint16 scalar

[20,1] = uint32 scalar

[21,1] = uint64 scalar

[22,1] = int8 matrix

[23,1] = int16 matrix

[24,1] = int32 matrix

[25,1] = int64 matrix

[26,1] = uint8 matrix

[27,1] = uint16 matrix

[28,1] = uint32 matrix

[29,1] = uint64 matrix

[30,1] = sparse bool matrix

[31,1] = sparse matrix

[32,1] = sparse complex matrix

[33,1] = struct

[34,1] = scalar struct

[35,1] = class

[36,1] = cs-list

[37,1] = magic-colon

[38,1] = built-in function

[39,1] = user-defined function

[40,1] = dynamically-linked function

[41,1] = function handle

[42,1] = inline function

[43,1] = float scalar

[44,1] = float complex scalar

[45,1] = float matrix

[46,1] = float diagonal matrix

[47,1] = float complex matrix

[48,1] = float complex diagonal matrix

[49,1] = permutation matrix

[50,1] = null\_matrix

[51,1] = null\_string

[52,1] = null\_sq\_string

[53,1] = lazy\_index

[54,1] = onCleanup

[55,1] = octave\_java

[56,1] = object

}

**2. Tipos de imagem:**

**Imagem RGB:**

**• Crie 3 matrizes (RED, GREEN e BLUE), com as mesmas dimensões, contendo valores entre 0 e 1.**

>> RED = [1 0 0; 1 0 1; 1 0 0.5];

>> GREEN = [0 0 1; 1 0 0; 1 1 0.5];

>> BLUE = [0 0 0; 0 1 1; 1 1 0.5];

**• Para ver o conteúdo das matrizes, digite seus nomes e, em seguida *Enter*. É muito importante notar que o Octave/Matlab é *case sensitive*.**

>> RED

RED =

1.00000 0.00000 0.00000

1.00000 0.00000 1.00000

1.00000 0.00000 0.50000

>> GREEN

GREEN =

0.00000 0.00000 1.00000

1.00000 0.00000 0.00000

1.00000 1.00000 0.50000

>> BLUE

BLUE =

0.00000 0.00000 0.00000

0.00000 1.00000 1.00000

1.00000 1.00000 0.50000

**• Concatenar as três matrizes para criar uma única matriz tridimensional (RGB)  
usando a função cat (usar a ajuda para aprender sobre a função: help cat)**

>> RGB = cat (3, RED, GREEN, BLUE);

**• Visualize o conteúdo da matriz RGB. Os dois primeiros índices dos elementos da matriz definem, respectivamente, os números de linha e coluna. O terceiro índice refere-se a uma das três matrizes originais. Se considerarmos agora matriz RGB como uma imagem, os dois primeiros índices definem as coordenadas de um pixel, e o terceiro, um dos componentes RGB do pixel.**

>> RGB

RGB =

ans(:,:,1) =

1.00000 0.00000 0.00000

1.00000 0.00000 1.00000

1.00000 0.00000 0.50000

ans(:,:,2) =

0.00000 0.00000 1.00000

1.00000 0.00000 0.00000

1.00000 1.00000 0.50000

ans(:,:,3) =

0.00000 0.00000 0.00000

0.00000 1.00000 1.00000

1.00000 1.00000 0.50000

>> RGB(:,:,1)

ans =

1.00000 0.00000 0.00000

1.00000 0.00000 1.00000

1.00000 0.00000 0.50000

>> RGB(:,:,2)

ans =

0.00000 0.00000 1.00000

1.00000 0.00000 0.00000

1.00000 1.00000 0.50000

>> RGB(:,:,3)

ans =

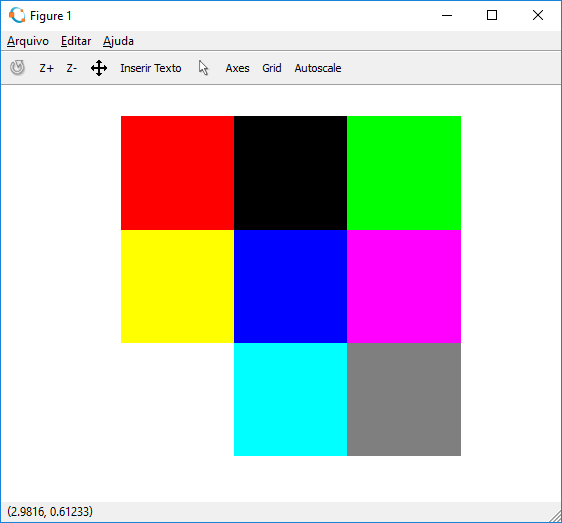
0.00000 0.00000 0.00000

0.00000 1.00000 1.00000

1.00000 1.00000 0.50000

**• Visualize a matriz resultante com a função: imshow (RGB);**

>> imshow (RGB);



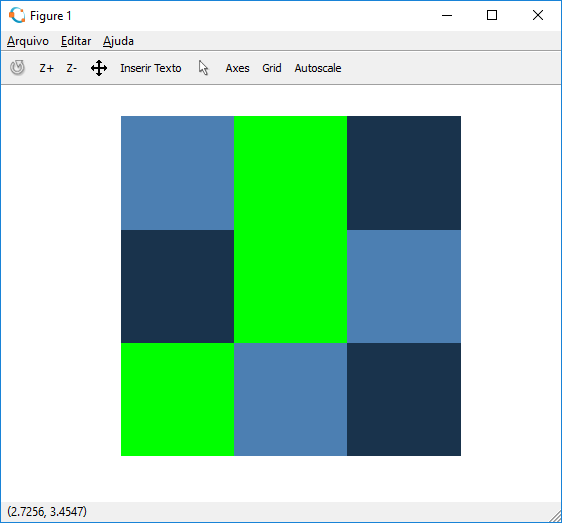
**Imagem Indexada:**

**• Crie uma matriz com o nome MAP, com três linhas e três colunas e valores entre 0 e 1, para representar um *mapa de cores*. Crie uma matriz bidimensional com o nome X, com valores inteiros entre 1 e 3. Visualize a imagem indexada usando: imshow (X, MAP);**

>> MAP = [0.3 0.5 0.7; 0 1 0; 0.1 0.2 0.3];

>> X = [1 2 3; 3 2 1; 2 1 3];

>> imshow (X, MAP);



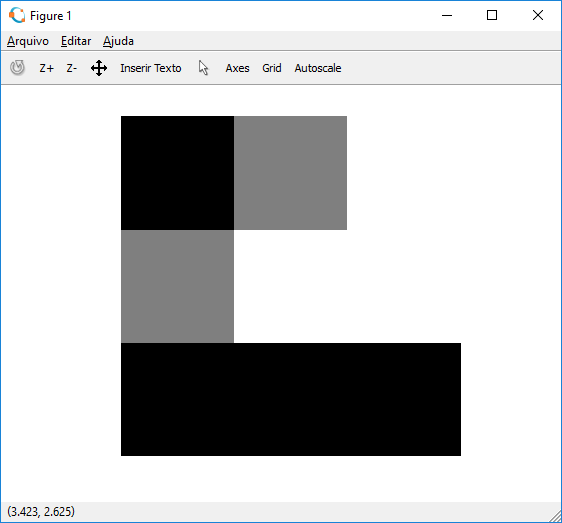
**Imagem de Níveis de Cinza:**

**• Crie uma matriz X com valores entre 0 e 1.**

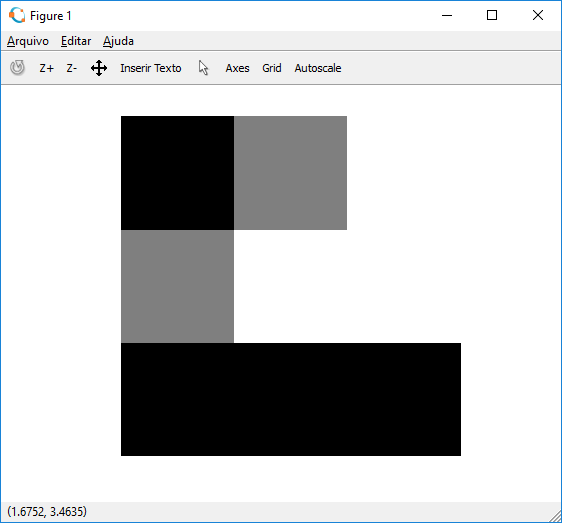
>> X = [0 0.5 1; 0.5 0.7 0.9; 0.1 0.2 0.3];

**• Visualize a matriz X como uma imagem de níveis de cinza (intensidade).**

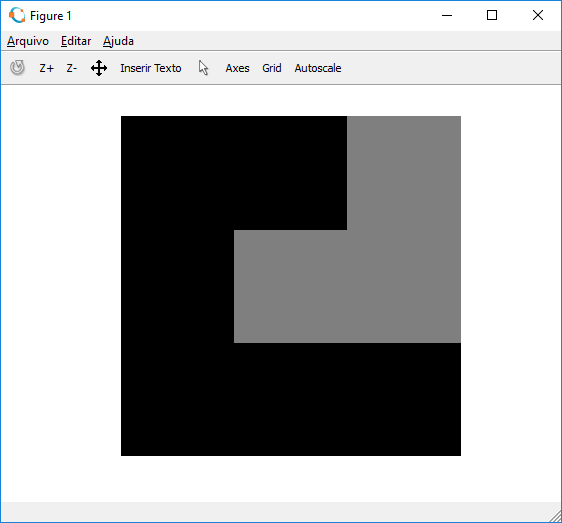
>> imshow(X);



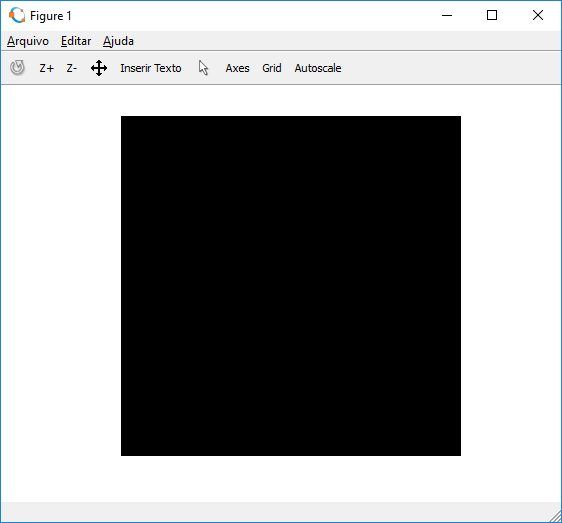
>> imshow(X, [0 1]);



>> imshow(X, [0 2]);



>> imshow(X, [0 16]);



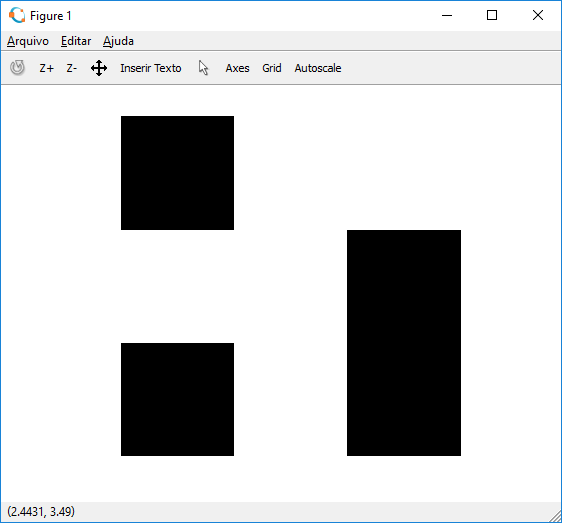
**Imagem Binária:**

**• Crie uma matriz X usando exclusivamente valores 0 ou 1 (imagem binária).**

>> X = [0 1 1; 1 1 0; 0 1 0];

**• Visualize a matriz X usando o comando: imshow (X);**

>> imshow(X);



**3. Leitura e gravação de imagens:**

**Leitura de Imagens:**

**• Estudar a função imread usando a ajuda do Octave/Matlab.**

**• Leia as imagens *arara\_full.png* e *arara\_full\_256.bmp* usando a função imread.**

>> a\_png = imread("D:/img/arara\_full.png");

>> a\_bmp = imread("D:/img/arara\_full\_256.bmp");

**• Tente descobrir, com a ajuda da função whos se as imagens são RGB ou indexadas (para saber mais sobre a função use help whos).**

>> whos a\_png

Variables in the current scope:

Attr Name Size Bytes Class

==== ==== ==== ===== =====

a\_png 296x460x3 408480 uint8

Total is 408480 elements using 408480 bytes

>> whos a\_bmp

Variables in the current scope:

Attr Name Size Bytes Class

==== ==== ==== ===== =====

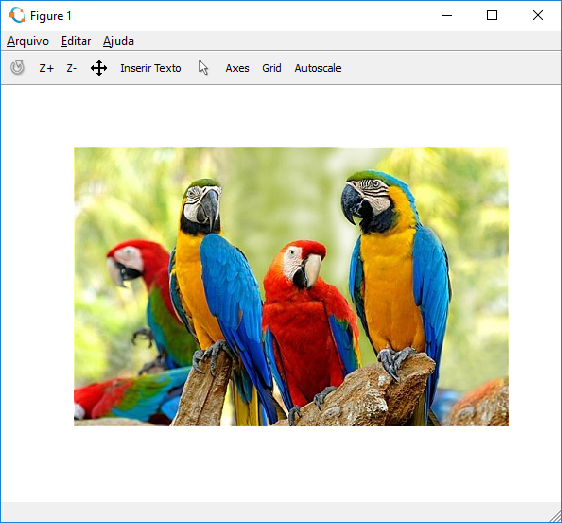
a\_bmp 296x460 136160 uint8

Total is 136160 elements using 136160 bytes

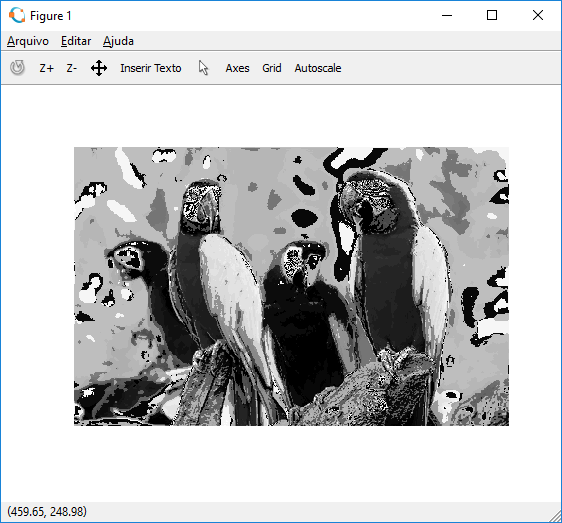
Como a imagem arara\_full.png possui 3 dimensões no campo size, ela é RGB. Já a imagem arara\_full\_256.bmp só possui 2 dimensões neste campo e, portanto, é indexada.

**• Visualize as imagens usando a função imshow**

>> imshow(a\_png);



>> imshow(a\_bmp);



**• Gravar em disco cópias das imagens lidas no exercício anterior com as extensões trocadas (*.bmp* ↔ *.png*) usando a função imwrite.**

>> imwrite(a\_png, "D:/img/arara\_full.bmp");

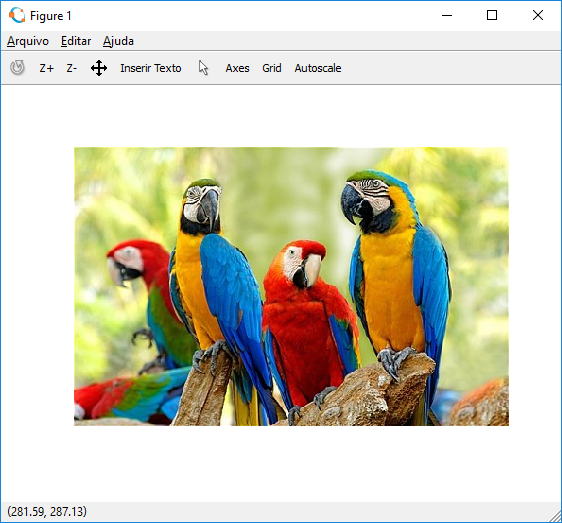
>> imwrite(a\_bmp, "D:/img/arara\_full\_256.png");

**• Ler as imagens gravadas (imread) e visualize-as (imshow).**

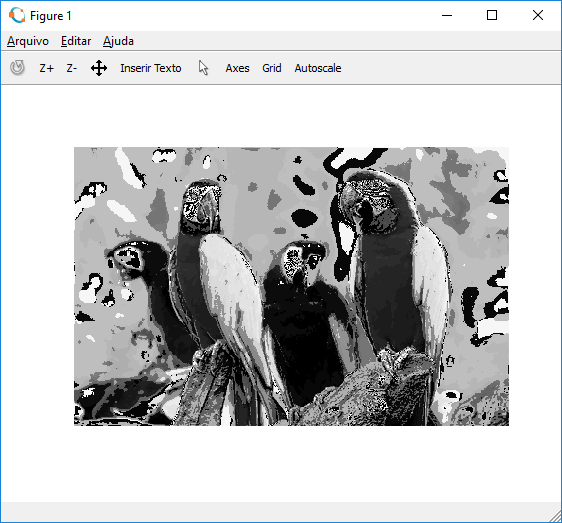
>> a\_bmp\_2 = imread("D:/img/arara\_full.bmp");

>> a\_png\_2 = imread("D:/img/arara\_full\_256.png");

>> imshow(a\_bmp\_2);



>> imshow(a\_png\_2);



**4. Conversão de imagens:**

**• Aprenda a usar as funções ind2gray, gray2ind, rgb2ind, ind2rgb, rgb2gray e gray2rgb. Observe os tipos de dados (double ou uint8) dos elementos das matrizes resultantes.**

**• Transformar as imagens criadas nos exercícios anteriores em diferentes tipos  
de imagem e visualizar as imagens resultantes.**

>> gray = imread("D:/img/arara\_full\_256.png");

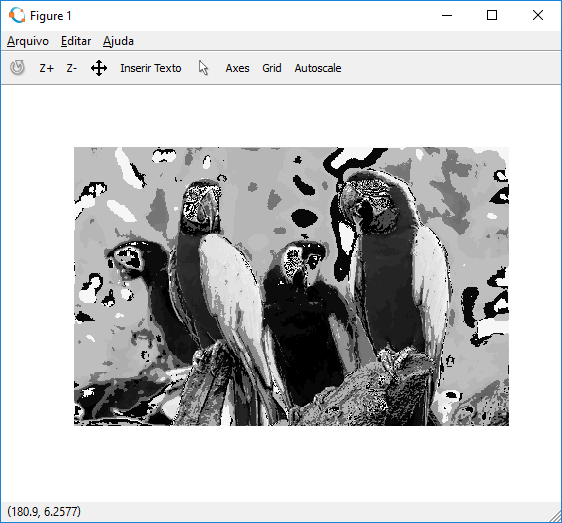
>> [indFromGray, mapIndFromGray] = gray2ind(gray);

>> grayFromInd = ind2gray(indFromGray, mapIndFromGray);

>> imshow(gray);

>> imshow(indFromGray, mapIndFromGray);

>> imshow(grayFromInd);



>> rgb = imread("D:/img/arara\_full.png");

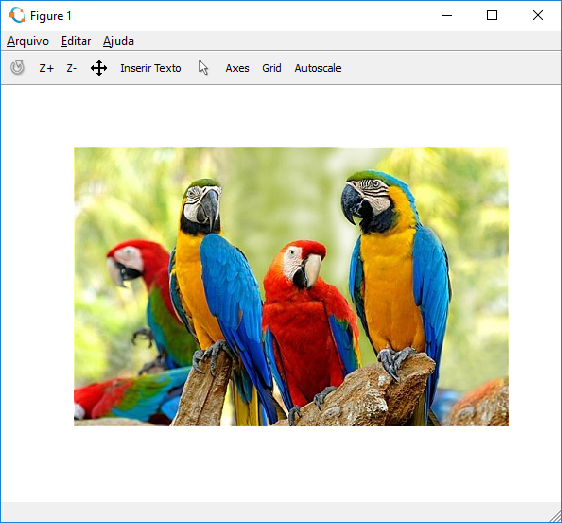
>> [indFromRgb, mapIndFromRgb] = rgb2ind(rgb);

>> rgbFromInd = ind2rgb(indFromRgb, mapIndFromRgb);

>> imshow(rgb);

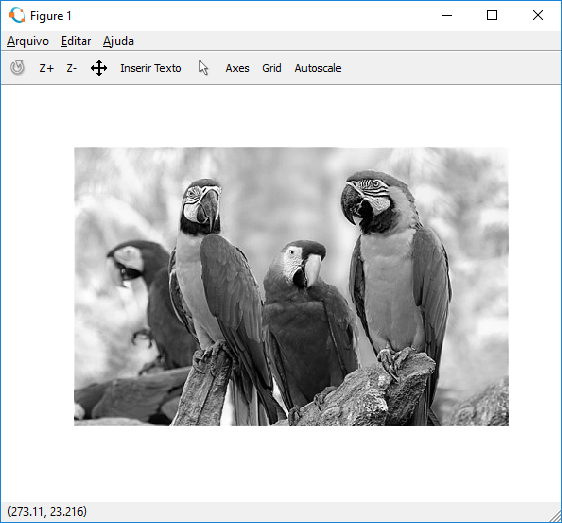
>> imshow(indFromRgb, mapIndFromRgb);

>> imshow(rgbFromInd);



>> rgb = imread("D:/img/arara\_full.png");

>> grayFromRgb = rgb2gray(rgb);



**5. Visualização:**

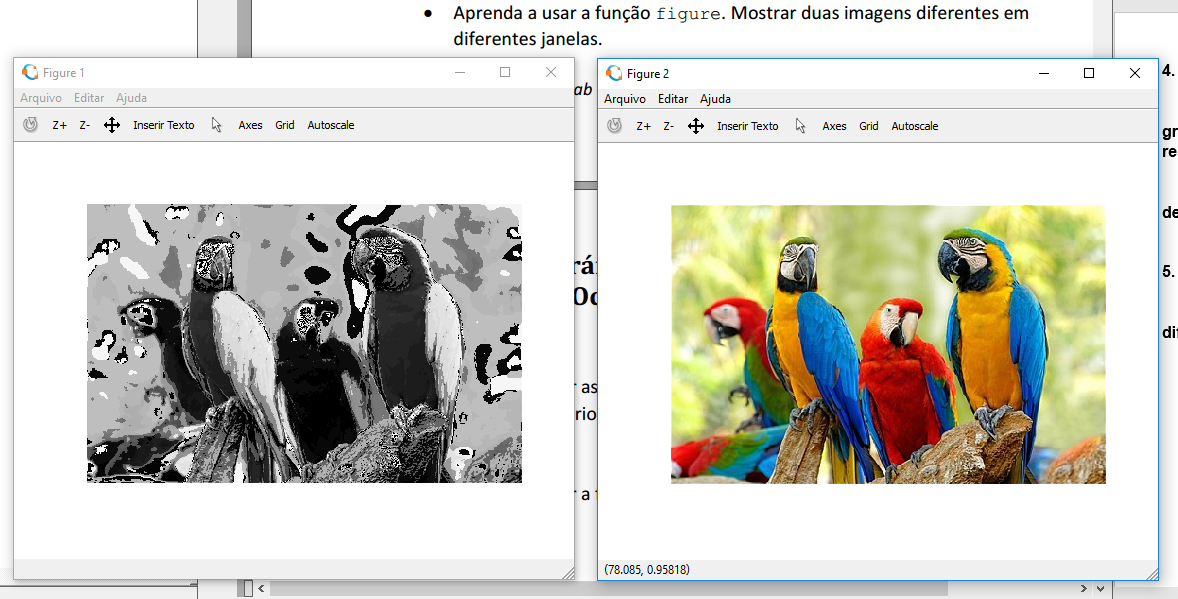
**• Aprenda a usar a função figure. Mostrar duas imagens diferentes em  
diferentes janelas.**

>> figure

>> imshow(a\_bmp\_2);

>> figure

>> imshow(a\_png\_2);

****

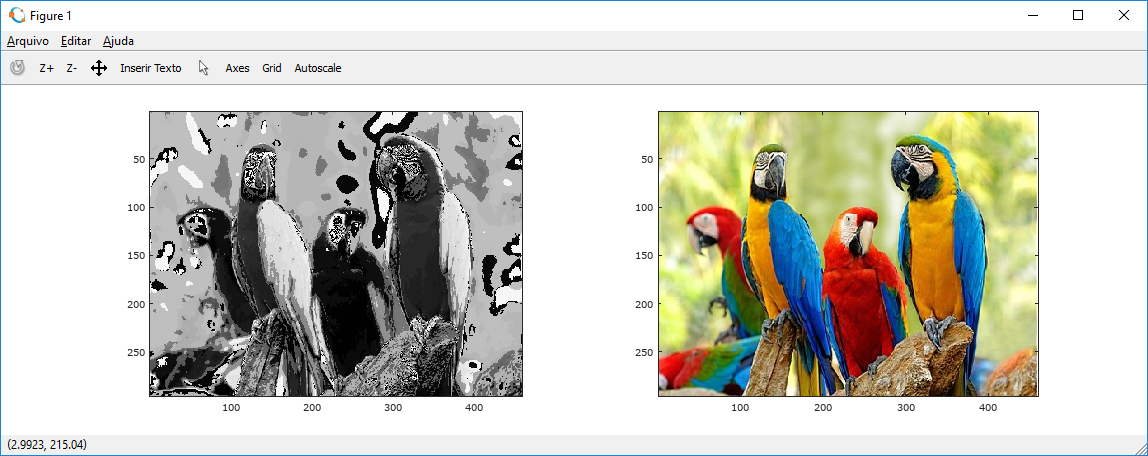
**• Aprenda a usar as funções subplot e subimage. Mostrar as duas imagens de exercícios anteriores em uma única janela usando estas funções.**

>> subplot(1,2,1);

>> subimage(a\_png\_2);

>> subplot(1,2,2);

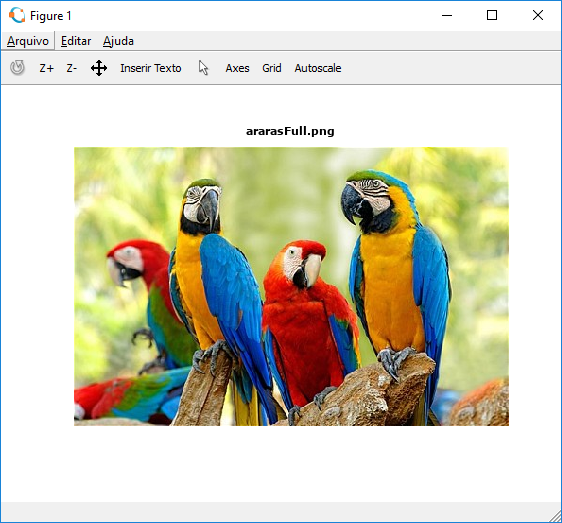
>> subimage(a\_bmp\_2);



**• Aprenda a usar a função title. Coloque títulos nas janelas dos exercícios anteriores.**

>> imshow(a\_png);

>> title ("ararasFull.png");



**6. Matrizes:**

**• Criar a matriz A usando o seguinte comando:**

>> A = [16 3 2 13 19; 5 10 11 8 3; 6 7 9 12 8; 4 15 14 1 13; 1 2 3 4 5];

**• Calcular a soma das quatro células dos cantos da matriz, referenciando os  
respectivos índices.**

>> A(1,1) + A(1,5) + A(5, 1) + A(5, 5)

ans = 41

**• Verifique a saída dos seguintes comandos:**

>> A

A =

16 3 2 13 19

5 10 11 8 3

6 7 9 12 8

4 15 14 1 13

1 2 3 4 5

>> A(1,3) + A (3,1)

ans = 8

>> A(1: 3,3)

ans =

2

11

9

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

ans =

3 2 13

10 11 8

7 9 12

15 14 1

>> A(:, 3)

ans =

2

11

9

14

3

>> A(1:3, :)

ans =

16 3 2 13 19

5 10 11 8 3

6 7 9 12 8

**Criar uma matriz 5x3 B, e aplicar o comando A(:, [3 5 2]) = B(:, 1:3)**

>> B = [10 20 30; 40 50 60; 15 10 5; 90 60 20; 10 10 10];

>> A(:, [3 5 2]) = B(:, 1:3)

A =

16 30 10 13 20

5 60 40 8 50

6 5 15 12 10

4 20 90 1 60

1 10 10 4 10

>> A(:)

ans =

16

5

6

4

1

30

60

5

20

10

10

40

15

90

10

13

8

12

1

4

20

50

10

60

10

>> A'

ans =

16 5 6 4 1

30 60 5 20 10

10 40 15 90 10

13 8 12 1 4

20 50 10 60 10

>> A(:, [1 2 2 3 3 3 4 4 4 4])

ans =

16 30 30 10 10 10 13 13 13 13

5 60 60 40 40 40 8 8 8 8

6 5 5 15 15 15 12 12 12 12

4 20 20 90 90 90 1 1 1 1

1 10 10 10 10 10 4 4 4 4

>> A([1 2 2 3 3 3 4 4 4 4], :)

ans =

16 30 10 13 20

5 60 40 8 50

5 60 40 8 50

6 5 15 12 10

6 5 15 12 10

6 5 15 12 10

4 20 90 1 60

4 20 90 1 60

4 20 90 1 60

4 20 90 1 60

>> A(A>10) = 0

A =

0 0 10 0 0

5 0 0 8 0

6 5 0 0 10

4 0 0 1 0

1 10 10 4 10

**• Estude os comandos zeros, ones, eye, size.**

>> zeros(2, 3)

ans =

0 0 0

0 0 0

>> ones(3, 2)

ans =

1 1

1 1

1. 1

>> eye(3)

ans =

Diagonal Matrix

1 0 0

0 1 0

0 0 1

>> size(B)

ans =

5 3

**• Use help arith para aprender sobre os operadores: "+", "-", "\*", "/", "./",  
"^", "\*". "^."**

**• Experimente com os operadores: "\*",". \*", "/", "./"**

>> C = [1 2 3; 7 8 9];

>> D = [4 5 6; 10 11 12];

>> E = [1 2; 3 4; 5 6];

>> C \* E

ans =

22 28

76 100

>> C .\* D

ans =

4 10 18

70 88 108

>> C / D

ans =

1.50000 -0.50000

0.50000 0.50000

>> C ./ D

ans =

0.25000 0.40000 0.50000

0.70000 0.72727 0.75000