Introdução ao Octave/Matlab

Anny Caroline Correa Chagas
Ciência da Computação, UERJ

21 de Julho de 2019

1 Exercício 1

Confira os tipos de dados do Octave/Matlab através do comando: `help datatypes`.

Resultado:
O comando correto para o Octave é o `typeinfo()`, que retorna uma matriz com todos os tipos de dados instalados no ambiente [1]. (veja o resultado do comando na Figura 1).

```
1 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 matri
  [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
}
```

Figura 1: Resultado do comando typeinfo()

2 Tipos de imagem

2.1 RGB

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

   Exemplo (três matrizes 3x3):

   ```
   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];
   ```

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

3. 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).
   Exemplo:
   `RGB = cat (3, RED, GREEN, BLUE);`

4. 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.

   Exemplo (experimente os comandos abaixo):

   ```
   RGB(:,:,1)
   RGB(:,:,2)
   RGB(:,:,3)
   ```

5. Visualize a matriz resultante com a função:
   `imshow (RGB);`

Resultado:

```
#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];

```

```
#2
disp(">> RED");
RED
disp(">> GREEN");
GREEN
disp(">> BLUE");
BLUE

#3
#help cat
RGB = cat (3, RED, GREEN, BLUE);

#4
disp(">> RGB");
RGB #visualizar o conteúdo da matriz
disp(">> RGB(:,:,1)");
RGB(:,:,1)
disp(">> RGB(:,:,2)");
RGB(:,:,2)
disp(">> RGB(:,:,3)");
RGB(:,:,3)

#5
imshow(RGB);
```

O resultado da execução pode ser observado nas Figuras 2, 3 e 4

```
>> 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
```

Figura 2: Visualização das matrizes RED, GREEN e BLUE

```
>> 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
```

Figura 3: Visualização da matriz RGB

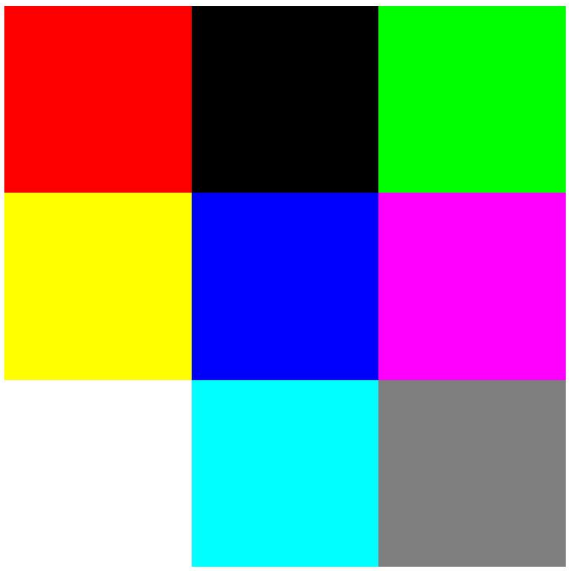

Figura 4: Figura gerada pela matriz RGB

2.2 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);
```

Resultado:

```
MAP = [1 0 0; 0 1 0; 0 0 1];
X = [1 2 3; 3 1 2];
imshow(X,MAP);
```

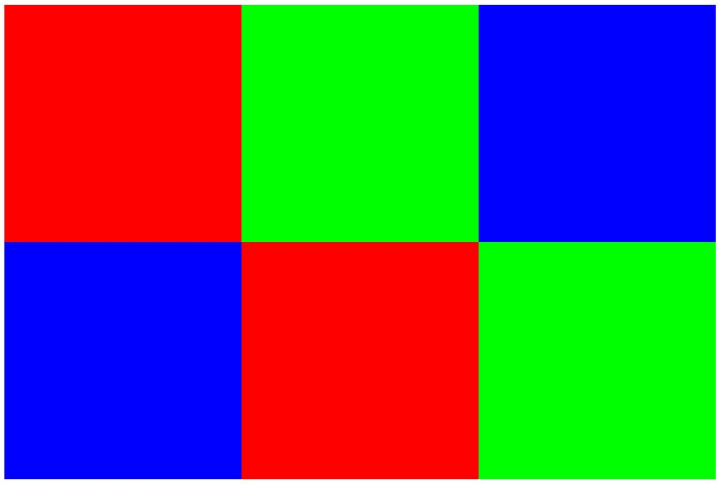

Figura 5: Figura gerada pelas matrizes X e MAP

2.3 Imagem de Níveis de Cinza

1. Crie uma matriz X com valores entre 0 e 1.
2. Visualize a matriz X como uma imagem de níveis de cinza (intensidade).

   Experimente os comandos abaixo:

   ```
   imshow (X);
   imshow (X, [0 1]);
   imshow (X, [0 2]);
   imshow (X, [0 16]);
   ```

Resultados:

```
X = [0.1 0.5 0.9];
imshow(X,MAP);
figure;
imshow(X,[0 1]);
figure;
imshow(X,[0 2]);
figure;
imshow(X,[0 16]);
```

Figura 6: imshow (X);

Figura 7: imshow (X, [0 1]);

Figura 8: imshow (X, [0 2]);

Figura 9: imshow (X, [0 16]);

2.4 Imagem Binária

1. Crie uma matriz X usando exclusivamente valores 0 ou 1 (imagem binária).
2. Visualize a matriz X usando o comando: imshow (X);

Resultado:

```
X = [1 1 0 0 1; 0 0 1 1 0; 0 1 0 1 0; 0 1 1 0 0; 1 0 0 1 1];
imshow(X);
```

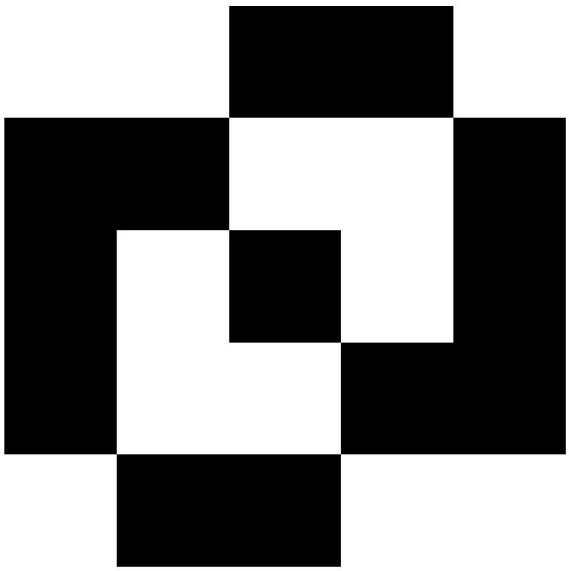

Figura 10: Imagem binária

3 Leitura e gravação de imagens

3.1 Leitura de Imagens

1. Estudar a função `imread` usando a ajuda do Octave/Matlab.
2. Leia as imagens `arara_full.png` e `arara_full_256.bmp` usando a função `imread`.
3. Tente descobrir, com a ajuda da função `whos` se as imagens são RGB ou indexadas (para saber mais sobre a função use `help whos`).
4. Visualize as imagens usando a função `imshow`.

Resultados:

```
1  #1
2  help imread;
3
4  #2
5  araraPNG = imread("imgs/arara_full.png");
6  araraBMP = imread("imgs/arara_full_256.bmp");
7
8  #3
9  whos;
10
11 #2 e 3
12 [araraBMPCorreta MAP] = imread("imgs/arara_full_256.bmp");
13
14 #4
15 figure;
16 imshow(araraPNG);
17 figure;
18 imshow(araraBMP);
19 figure;
20 imshow(araraBMPCorreta, MAP);
```

A imagem em png, por ser uma imagem RGB (observe o tamanho 296x460x3 na Figura 11) foi lida corretamente na linha 5. Entretanto, a imagem em bmp, por ser uma imagem indexada, não foi lida corretamente: na linha 6 não foi obtido o mapa de cores, por isso a Figura 13 exibiu uma imagem em preto e branco.
Por conta disso, uma nova leitura foi feita, na linha 12, para obter tanto a matriz da imagem quanto o mapa de cores, resultando na imagem mostrada na Figura 14.

```
>> lle3Lei
Variables in the current scope:

   Attr Name          Size                     Bytes  Class
   ==== ====          ====                     =====  =====
        araraBMP    296x460                   136160  uint8
        araraPNG    296x460x3                 408480  uint8

Total is 544640 elements using 544640 bytes
```

Figura 11: whos;

Figura 12: imshow(araraPNG);

Figura 13: imshow(araraBMP);

Figura 14: imshow(araraBMPCorreta, MAP);

3.2 Escrita de Imagens

1. Gravar em disco cópias das imagens lidas no exercício anterior com as extensões trocadas (.bmp <->.png) usando a função `imwrite`.

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

Resultados:

```
#exercicio anterior
araraPNG = imread("imgs/arara_full.png");
[araraBMP MAP] = imread("imgs/arara_full_256.bmp");

#1
imwrite(araraPNG, "arara.bmp");
imwrite(araraBMP, MAP, "arara.png");

#2
araraBMP = imread("arara.bmp");
[araraPNG MAP] = imread("arara.png");

figure;
imshow(araraBMP);
figure;
imshow(araraPNG, MAP);
```

Figura 15: Imagem bmp (antiga png) — linha 14

Figura 16: Imagem jpg (antiga bmp) — linha 16

4 Conversão de Imagens:

1. 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.

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

Resultados:

```
#ind2gray
[X MAP] = imread("imgs/arara_full_256.bmp");
imshow(X, MAP);
G = ind2gray(X, MAP);
figure
imshow(G);

#gray2ind
[X, MAP] = gray2ind(G);
figure
imshow(X, MAP);

#rgb2ind
RGB = imread("imgs/arara_full.png");
[X, MAP] = rgb2ind(RGB);
figure
imshow(X,MAP);

#ind2rgb
[X MAP] = imread("imgs/arara_full_256.bmp");
RGB = ind2rgb(X, MAP);
figure
imshow(RGB);

#rgb2gray
RGB = imread("imgs/arara_full.png");
G = rgb2gray(RGB);
figure
imshow(G);

#gray2rgb
#não existe
```

Figura 17: ind2gray

Figura 18: gray2ind

Figura 19: rgb2ind

Figura 20: ind2rgb

Figura 21: rgb2gray

```
>> gray2rgb
error: 'gray2rgb' undefined near line 1 column 1
```

Figura 22: gray2rgb

5 Visualização

1. Aprenda a usar a função `figure`. Mostrar duas imagens diferentes em diferentes janelas.
2. Aprenda a usar as funções `subplot` e `subimage`. Mostrar as duas imagens de exercícios anteriores em uma única janela usando estas funções.
3. Aprenda a usar a função `title`. Coloque títulos nas janelas dos exercícios anteriores.

Resultado:

```
ELC = imread("imgs/Einstein_low_contrast.png");
EMC = imread("imgs/Einstein_med_contrast.png");

#1
figure;
imshow(ELC);
figure;
imshow(EMC);

#2

#SUBPLOT
figure;
subplot(1,2,1);
imshow(ELC);
subplot(1,2,2);
imshow(EMC);

#SUBIMAGE
figure;
subplot(1,2,1);
subimage(ELC);
subplot(1,2,2);
subimage(EMC);

#3
#TITLE
figure;
imshow(ELC);
title ("Título");

#TITULO NA JANELA
figure('name','Título');
imshow(ELC);
```

```
35 figure('name', 'Título sem número','NumberTitle','off');
36 imshow(EMC);
```

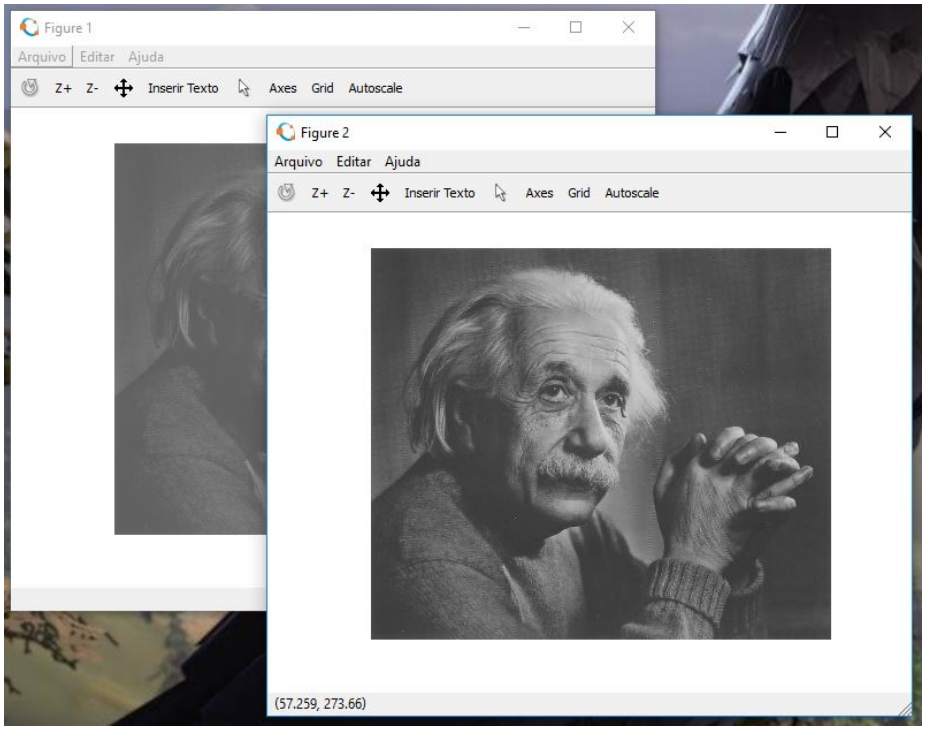

Figura 23: Imagens diferentes em janelas diferentes

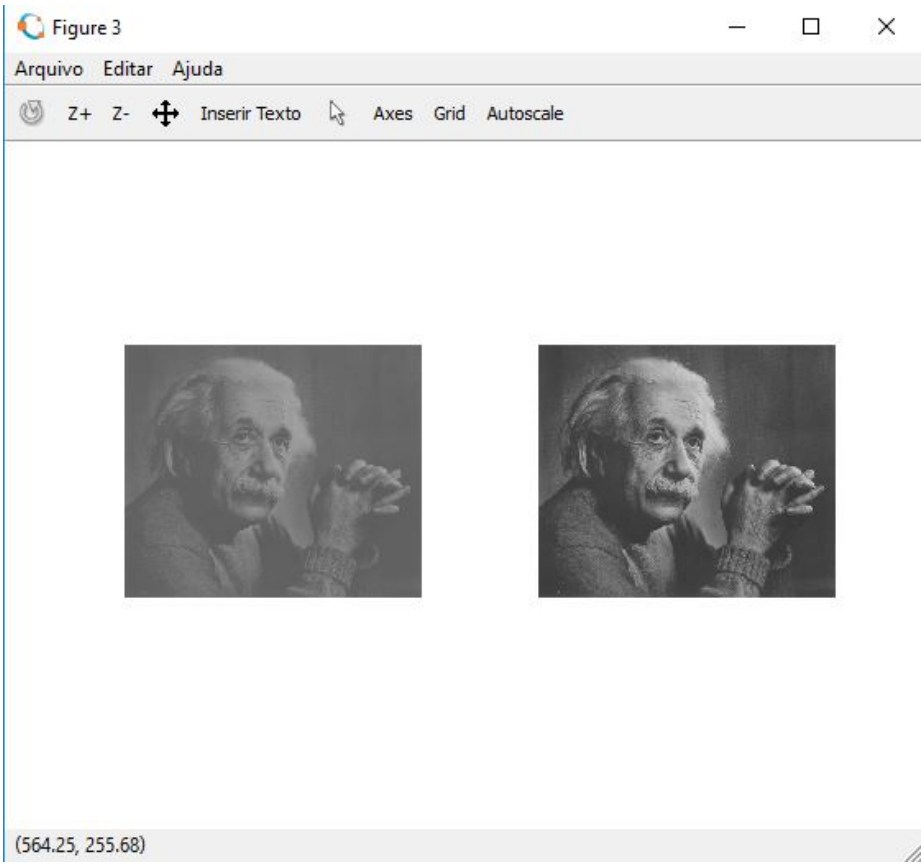

Figura 24: Imagens diferentes em uma única janela com `subplot`

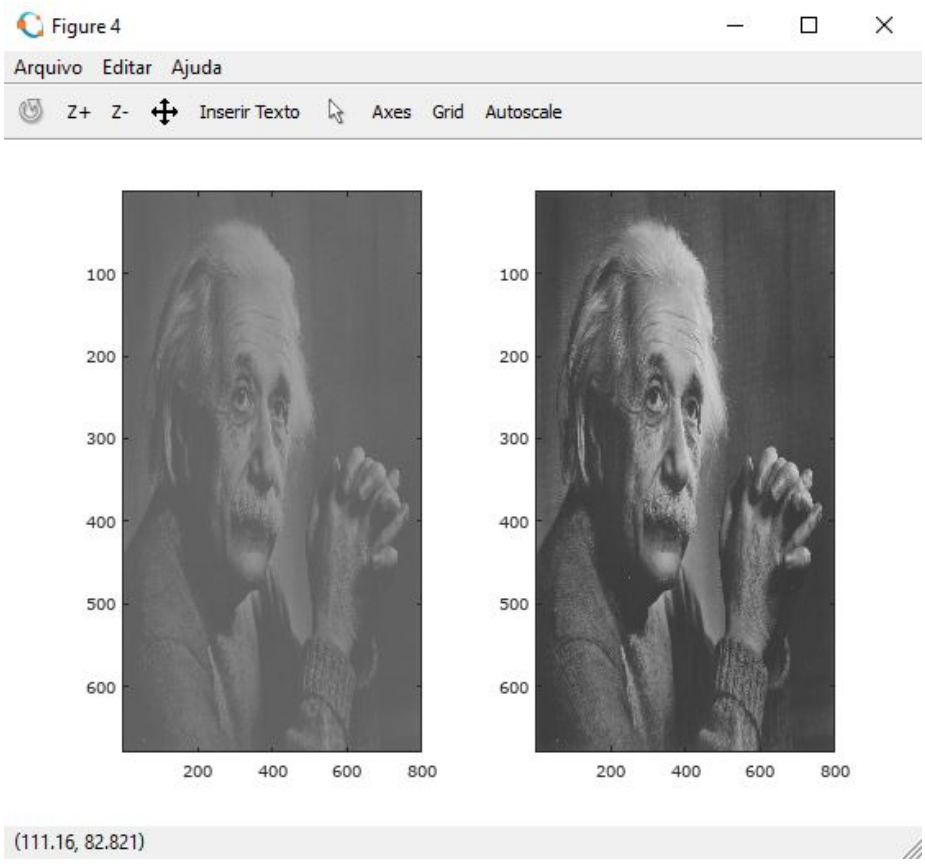

Figura 25: Imagens diferentes em uma única janela com `subimage`

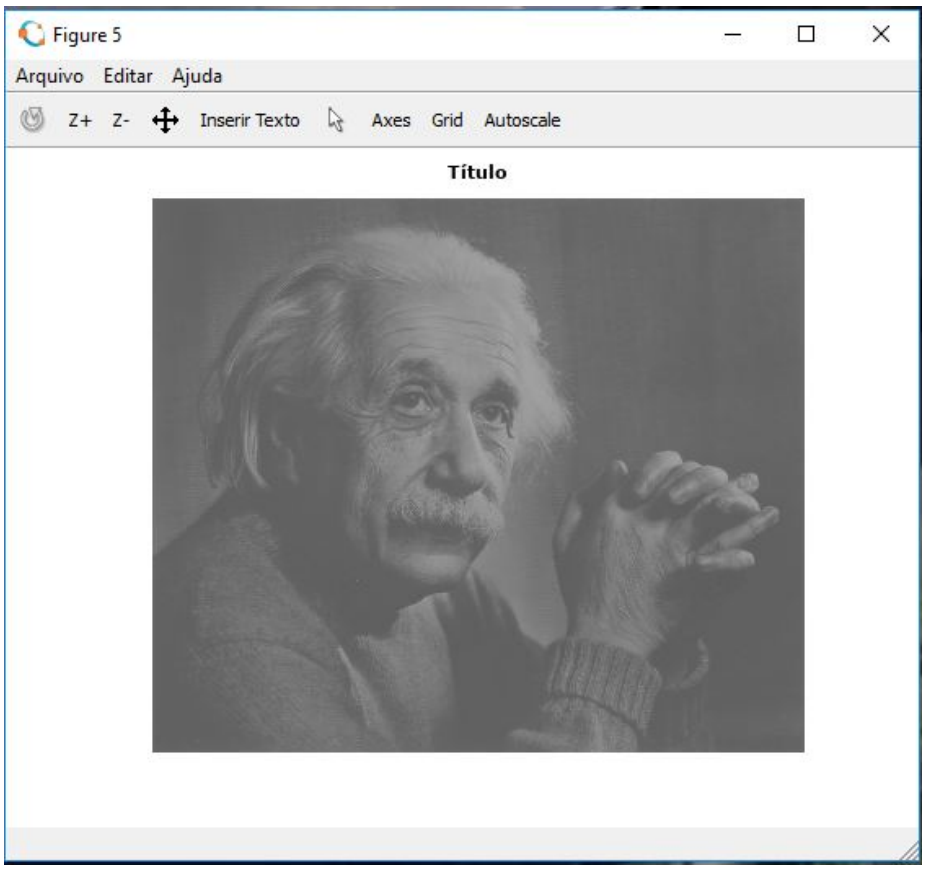

Figura 26: Título com a função `title`

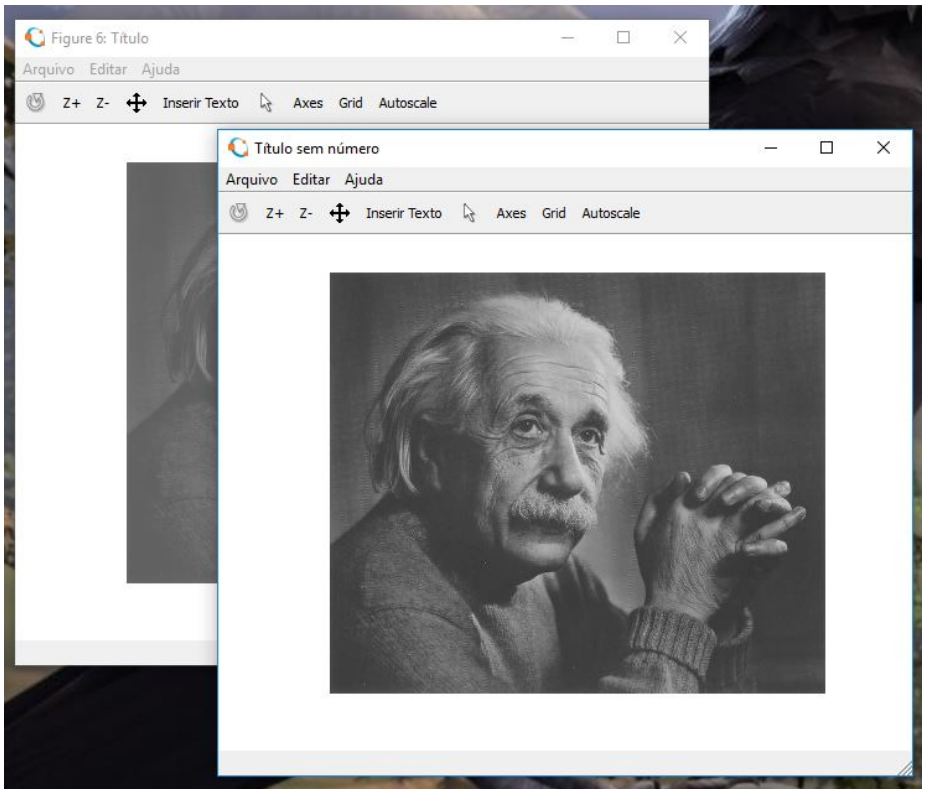

Figura 27: Alterando o título da janela [2]

6 Matrizes

1. 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]`

```
>> 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 =

   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
```

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

```
>> A(1,1) + A(1,5) + A(5,5) + A(5,1)
ans =  41
```

3. Verifique a saída dos seguintes comandos:
   `A`
   `A(1,3) + A(3,1)`
   `A(1:  3,3)`
   `A(1:4,2:4)`
   `A(:, 3)`
   `A(1:3, :)`

```
>> 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
```

4. Criar uma matriz 5x3 B, e aplicar os comandos:
   `A(:, [3 5 2]) = B(:, 1:3)`
   `A(:)`
   `A'`
   `A(:, [1 2 2 3 3 3 4 4 4 4])`
   `A([1 2 2 3 3 3 4 4 4 4], :)`
   `A(A>10) = 0`

```
>> B = [16 3 2; 5 10 11; 6 7 9; 4 15 14; 1 2 3]
B =

   16    3    2
    5   10   11
    6    7    9
    4   15   14
    1    2    3
```

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

    0    2   16    0    3
    5   11    5    8   10
    6    9    6    0    7
    4   14    4    1   15
    1    3    1    4    2

>> A(:)
ans =

    0
    5
    6
    4
    1
    2
   11
    9
   14
    3
   16
    5
    6
    4
    1
    0
    8
    0
    1
    4
    3
   10
    7
   15
    2

>> A'
ans =

    0    5    6    4    1
    2    0    9    0    3
    0    5    6    4    1
```

```
    0    8    0    1    4
    3   10    7    0    2

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

   0   2   2   0   0   0   0   0   0   0
   5   0   0   5   5   5   8   8   8   8
   6   9   9   6   6   6   0   0   0   0
   4   0   0   4   4   4   1   1   1   1
   1   3   3   1   1   1   4   4   4   4

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

    0    2    0    0    3
    5    0    5    8   10
    5    0    5    8   10
    6    9    6    0    7
    6    9    6    0    7
    6    9    6    0    7
    4    0    4    1    0
    4    0    4    1    0
    4    0    4    1    0
    4    0    4    1    0

>> A(A>10) = 0
A =

    0    2    0    0    3
    5    0    5    8   10
    6    9    6    0    7
    4    0    4    1    0
    1    3    1    4    2
```

5. Estude os comandos zeros, ones, eye, size.

   (a) **zeros, zeros(n), zeros(m,n), zeros(m,n,k), zeros([m n ...]), zeros(...,"uint8"):** retorna uma matriz preenchida com zeros [3].

```
>> zeros
ans = 0
>> zeros(1)
ans = 0
```

```
>> zeros(3)
ans =

   0   0   0
   0   0   0
   0   0   0

>> zeros(2,3)
ans =

   0   0   0
   0   0   0

>> zeros(2,3,4)
ans =

ans(:,:,1) =

   0   0   0
   0   0   0

ans(:,:,2) =

   0   0   0
   0   0   0

ans(:,:,3) =

   0   0   0
   0   0   0

ans(:,:,4) =

   0   0   0
   0   0   0

>> zeros([2 3 4])
ans =

ans(:,:,1) =

   0   0   0
   0   0   0

ans(:,:,2) =
```

```
   0   0   0
   0   0   0

ans(:,:,3) =

   0   0   0
   0   0   0

ans(:,:,4) =

   0   0   0
   0   0   0

>> zeros(2,"uint8")
ans =

  0  0
  0  0
```

(b) **ones, ones(n), ones(m,n), ones(m,n,k), ones([m n ...]), ones(...,"uint8"):** retorna uma matriz preenchida com "um"s [4].

```
>> ones
ans =  1
>> ones(1)
ans =  1
>> ones(3)
ans =

   1   1   1
   1   1   1
   1   1   1

>> ones(2,3)
ans =

   1   1   1
   1   1   1

>> ones(2,3,4)
ans =

ans(:,:,1) =

```

```
   1   1   1
   1   1   1

ans(:,:,2) =

   1   1   1
   1   1   1

ans(:,:,3) =

   1   1   1
   1   1   1

ans(:,:,4) =

   1   1   1
   1   1   1

>> ones([2 3 4])
ans =

ans(:,:,1) =

   1   1   1
   1   1   1

ans(:,:,2) =

   1   1   1
   1   1   1

ans(:,:,3) =

   1   1   1
   1   1   1

ans(:,:,4) =

   1   1   1
   1   1   1

>> ones(2,"uint8")
ans =

  1  1
```

```
  1  1
```

(c) **eye(n), eye(m,n), eye([m n], eye(..., class):** retorna uma matriz identidade [5].

```
>> eye
ans =  1
>> eye(1)
ans =  1
>> eye(2,3)
ans =

Diagonal Matrix

   1   0   0
   0   1   0

>> eye([2 3])
ans =

Diagonal Matrix

   1   0   0
   0   1   0

>> eye(3,"uint8")
ans =

  1  0  0
  0  1  0
  0  0  1
```

(d) **size(X), size(X, dim):** retorna um vetor linha contendo o número de elementos em cada dimensão (ou de uma dimensão, se passado o segundo parâmetro) da matriz passada por parâmetro [6].

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

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

```

```
   1   2   3   4
   5   6   7   8

>>
>>
>> size(A)
ans =

   1   3

>>
>>
>> size(A,2)
ans =  3
>>
>>
>> size(B)
ans =

   2   4

>> [linhas colunas] = size(B);
>> linhas
linhas =  2
>> colunas
colunas =  4
>>
>>
>> [linhas sobrou] = size(ones(2, 3))
linhas =  2
sobrou =  3
>>
>>
>> [linhas sobrou] = size(ones(2, 3, 4))
linhas =  2
sobrou =  12
```

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

7. Experimente com os operadores: ”*”,”. *”, ”/”, ”./”

   (a) +
   Soma duas matrizes ou dois valores. Se forem duas matrizes, os o número de colunas e linhas devem ser iguais [7].

```
>> 1+2
ans =  3
>> [1 2] + [3 4]
ans =

   4   6

>> [1 2] + [3 4 5]
error: operator +: nonconformant arguments (op1 is 1x2,
 ↪  op2 is 1x3)
```

(b) -
Subtrai duas matrizes ou dois valores. Se forem duas matrizes, os o número de colunas e linhas devem ser iguais [7].

```
>> 1-2
ans = -1
>> [1 2] - [3 4]
ans =

  -2  -2

>> [1 2] - [3 4 5]
error: operator -: nonconformant arguments (op1 is 1x2,
 ↪  op2 is 1x3)
```

(c) *
Multiplicação de matrizes ou valores escalares. Em uma multiplicação x*y, onde x e y são matrizes, o número de colunas de x deve ser igual ao número de linhas em y [7].

```
>> 1 * 2
ans =  2
>> [1 2] * [3 4]
error: operator *: nonconformant arguments (op1 is 1x2,
 ↪  op2 is 1x2)
>> [1 2] * [3; 4]
ans =  11
```

(d) /
Divisão simples [7].

```
>> 4/2
ans =  2
```

```
>> [2 2]/[2 2]
ans =  1.00000
>> [2 2]/[4 4]
ans =  0.50000
>> [2 2]/[4 4 4]
error: quotient: nonconformant arguments (op1 is 1x2,
 ↪  op2 is 1x3)
```

(e) ./
Divisão elemento por elemento [7].

```
>> 4./2
ans =  2
>> [2 2]./[2 2]
ans =

   1   1

>> [2 2]./[4 4]
ans =

   0.50000   0.50000

>> [2 2]./[4 4 4]
error: quotient: nonconformant arguments (op1 is 1x2,
 ↪  op2 is 1x3)
```

(f) ^
Em x ^ y, onde y é um escalar, calcula o valor de $\prod_{i=1}^{y} x$

```
>> 2^3
ans =  8
>> [1 2; 3 4]^2
ans =

    7   10
   15   22
```

(g) .^
Operador "power"elemento por elemento.

```
>> 2.^3
ans =  8
>> [2 2; 2 2].^3
```

```
ans =

   8   8
   8   8

>> [2 2; 2 2].^[1 2]
ans =

   2   4
   2   4

>> [2 2; 2 2].^[1; 2]
ans =

   2   2
   4   4

>> [2 2; 2 2].^[1 2; 3]
error: vertical dimensions mismatch (1x2 vs 1x1)
>> [2 2; 2 2].^[1; 2 3]
error: vertical dimensions mismatch (1x1 vs 1x2)
>> [2 2; 2 2].^[1 2; 3 4; 5 6]
error: operator .^: nonconformant arguments (op1 is
↪  2x2, op2 is 3x2)
```

Referências

[1] "GNU Octave typeinfo," https://www.gnu.org/software/octave/doc/v4.0.3/Data-Types.html, acessado: 2017-09-12.

[2] "Stack Overflow how to change the window title of a matlab plotting figure?" https://stackoverflow.com/questions/4684363/how-to-change-the-window-title-of-a-matlab-plotting-figure, acessado: 2017-09-17.

[3] "Octave io zeros," https://octave.sourceforge.io/octave/function/zeros.html, acessado: 2017-09-17.

[4] "Octave io ones," https://octave.sourceforge.io/octave/function/ones.html, acessado: 2017-09-17.

[5] "Octave io eye," https://octave.sourceforge.io/octave/function/eye.html, acessado: 2017-09-17.

[6] "Octave io size," https://octave.sourceforge.io/octave/function/size.html, acessado: 2017-09-17.

[7] "Octave io arithmetic operators," https://www.gnu.org/software/octave/doc/v4.2.0/Arithmetic-Ops.html, acessado: 2017-09-17.