

## Notas de aula e prática MATLAB #03

### Arquivos necessários

1. 5.1.09.tiff [<http://sipi.usc.edu/database/database.php>]
2. xray01.tif [adaptada de <https://wexnermedical.osu.edu/blog/facts-about-dense-breast-tissue>]
3. radio.tif [[http://www.ogemarques.com/wp-content/uploads/2014/11/Tutorial\\_Images.zip](http://www.ogemarques.com/wp-content/uploads/2014/11/Tutorial_Images.zip)]
4. vpfig.png [adaptada de Volnei A. Pedroni, Eletrônica Digital Moderna e VHDL, 2020, Fig. 4.6]
5. 42049\_20-200.png [adaptada de <https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/seqbench/BSDS300/html/dataset/images/gray/42049.html>]
6. gDSC04422m16.png
7. phpEMp3Xr.png
8. Fig0362.tif [Rafael C. Gonzalez, Richard E. Woods and Steven L. Eddins. Digital Image Processing Using MATLAB. 3rd ed. Pearson Prentice Hall, 2020. Capítulo 3. Imagem disponível em <https://www.imageprocessingplace.com>]

### 3.1) Funções de transformação dos níveis de cinza (identidade e negativo)

O código a seguir cria uma *função de transformação* (ajuste) do tipo *identidade*. Para o warm up deste exercício, entenda o código e execute-o.

```
clc, clear, close all

g = checkerboard;
figure; imshow(g)

gu = im2uint8(g);
% Só pra lembrar:
% gu2 = uint8(g.*255); % igual im2uint8(g)
% igual = isequal(gu,gu2) % igual é '1'

% Função de transformação
y = uint8(0:1:255);
gIdx = double(gu)+1; % double(gu) pq 255+1...
% ...em um uint8 ocorre overflow (MATLAB não
% ..."gira" fazendo voltar pra 0; ele trunca em 255).
% +1 pq MATLAB é one-based indexing.

gtu = y(gIdx); % saída gtu é da mesma classe
% da função de transformação y
figure; imshow(gtu);
% igual = isequal(gu,gtu) % igual é '1'
```

O *negativo* de uma imagem é equivalente ao negativo fotográfico. É obtido fazendo-se  $n = P - p$  para todos os pixels da imagem, onde  $P$  é o máximo valor de nível de cinza possível (255 para imagens uint8) e  $p$  é o valor do pixel.

O negativo pode ser utilizado para facilitar a visualização de objetos em determinados tipos de imagens. Para o caso de imagens de exames de mamografia, por exemplo, há especialistas que preferem visualizar o negativo da imagem original. O argumento é que, na imagem original, os detalhes são mais difíceis de visualizar pois ficam envoltos por grandes áreas escuras.

Crie uma função de transformação para obter o negativo da imagem *xray01.png*. Só pra saber: o MATLAB tem a função `imcomplement` [<https://www.mathworks.com/help/images/ref/imcomplement.html>], mas nesse exercício não é pra usá-la. Obrigatoriamente, use o código do exemplo como template. Resposta disponível em a03\_01.m.

### 3.2) Funções de transformação dos níveis de cinza (negativo usando a função `intlut`)

Outra maneira de implementar funções de transformação no MATLAB é utilizando-se a função `intlut` [<https://www.mathworks.com/help/images/ref/intlut.html>], que implementa uma *lookup table* (LUT). O código a seguir cria uma *função de transformação* (ajuste) do tipo *identidade* usando a função `intlut`.

```
clc, clear, close all

g = imread('5.1.09.tiff');
figure; imshow(g)

% Função de transformação
y = uint8(0:255);
figure, plot(y), xlim([0 255]), ylim([0 255])
% Usando intlut
gt = intlut(g,y);
figure, imshow(gt)

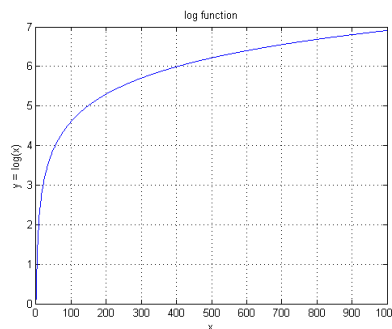
% Se verifica == 0, imagens iguais
verifica = double(g) - double(gt);
verifica = sum(verifica(:));
```

Crie uma função de transformação para obter o negativo da imagem *5.1.09.tiff*, utilizando a função `intlut`. Só pra saber: o MATLAB tem a função `imcomplement`, mas nesse exercício não é pra usá-la. Obrigatoriamente, use o código do exemplo como template, no mesmo espírito do exercício anterior. Resposta disponível em *a03\_02.m*.

### 3.3) Funções de transformação dos níveis de cinza ( $\log_e$ )

```
clc, clear, close all
%Funcao log
%Logaritmo neperiano (base e)
y = log(1:1000);

%Display
figure
plot(y)
grid on
title('log function')
xlabel('x')
ylabel('y = log(x)')
```



A função de transformação do tipo logarítmica é  $y = c \cdot \log(1+x)$ .  $x$  é o pixel da imagem e  $c$  uma constante geralmente igual a 1. Também pode ser menor que 1, para manter a saída dentro da faixa dinâmica (255 para imagens `uint8`). O  $1+$  serve para evitar o  $\log(0) = \text{indeterminado}$  ( $\rightarrow \infty$ ).

Lembrando que, geralmente,  $\log$  é o  $\log_{10}$  (a base pode ser suprimida),  $\log_e$  ou  $\ln$  é o *logaritmo natural* ou *logaritmo neperiano* [<http://en.wikipedia.org/wiki/Logarithm>, Tópico Particular bases]. Atenção, pois no MATLAB: função `log10` é o  $\log$  e função `log` é o  $\log_e$ .

Observando a curva da função logarítmica plotada anteriormente, conclui-se que ela mapeia os níveis de cinza da entrada para a saída da seguinte forma:

- Uma faixa de poucos valores baixos (faixa estreita) de níveis de cinza na entrada  $\rightarrow$  uma faixa de muitos valores (faixa ampla) de níveis de cinza na saída
- A faixa restante de valores mais altos de níveis de cinza na entrada  $\rightarrow$  uma faixa de poucos (faixa estreita) valores de níveis de cinza maiores na saída.

O efeito disto é a expansão dos pixels escuros, para deixá-los mais claros e visíveis, e por consequência a compressão dos pixels claros, para que todos os níveis de cinza caibam na faixa dinâmica da imagem.

A função *gamma* tem a mesma finalidade da *log*, é mais versátil e portanto mais utilizada. A função *log* é útil para comprimir os valores dos pixels de imagens com variações muito grandes nos valores dos pixels. A aplicação típica está na visualização da transformada de Fourier de uma imagem, que pode assumir valores de 0 até  $10^6$ , por exemplo.

Aplicar o  $\log$  ( $\log_e$ ) na imagem *radio.tif*. Não use auto-contraste (`mat2gray`). Resposta disponível em *a03\_03.m*.

### 3.4) Auto-contraste (também chamado de normalização) (na unha, usando `mat2gray` e `imadjust`)

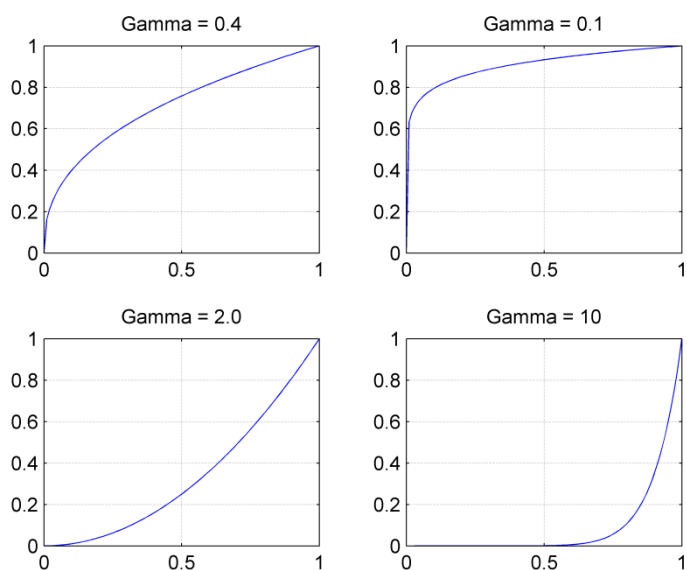
Outros nomes utilizados para esta operação são *normalização*, *contrast stretching* ou *histogram stretching*. Apenas 'normalização' não deixa claro se está realizando-se a operação  $\min(img) \rightarrow 0$ ,  $\max(img) \rightarrow 1$ , ou apenas uma divisão por 255 (para imagens de entrada `uint8`).

Fazer o auto-contraste da imagem `42049_20-200.png` de três formas diferentes: na unha, usando a função `mat2gray` e usando a função `imadjust` [<https://www.mathworks.com/help/images/ref/imadjust.html>]. Os resultados devem ser visualmente iguais. Resposta disponível em `a03_04.m`.

### 3.5) Funções de transformação dos níveis de cinza (gamma usando função `imadjust`)

```
clc, clear, close all
%Funcao Gamma
x = 0:0.01:1;
y1 = x.^0.4;
y2 = x.^0.1;
y3 = x.^2.0;
y4 = x.^10;

%Display
figure
subplot(2,2,1), plot(x,y1)
grid on
title('Gamma = 0.4')
subplot(2,2,2), plot(x,y2)
grid on
title('Gamma = 0.1')
subplot(2,2,3), plot(x,y3)
grid on
title('Gamma = 2.0')
subplot(2,2,4), plot(x,y4)
grid on
title('Gamma = 10')
```



A função de transformação do tipo *gamma* é  $y = c \cdot x^\gamma$ .  $x$  é o pixel da imagem e  $c$  uma constante geralmente igual a 1. Ao usar a função *gamma*, é mais comum que os pixels da imagem de entrada estejam na faixa  $[0 \dots 1]$ .  $\gamma$  é a constante que determina o formato da curva.

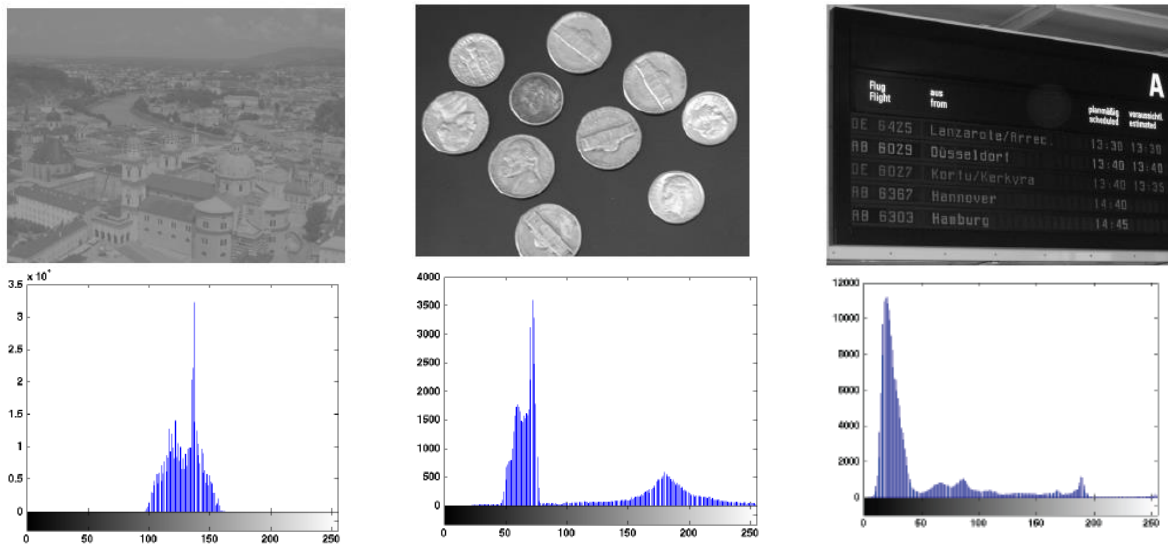
Observando a curva da função *gamma* plotada anteriormente, conclui-se que ela mapeia os níveis de cinza da entrada para a saída como descrito a seguir.

- Para  $\gamma < 1$ : A imagem 'fica mais clara'.
- Para  $\gamma > 1$ : A imagem 'fica mais escura'.
- Para  $\gamma = 1$ : Função identidade, isto é, a saída é igual à entrada (desde que  $c=1$ )

Usar a função `imadjust` [<http://www.mathworks.com/help/images/ref/imadjust.html>] para aplicar a função de transformação *gamma* na imagem `radio.tif`. Mostrar cada imagem em uma figure: original e as processadas com  $\gamma = 0,4$ ,  $\gamma = 0,1$  e  $\gamma = 2,0$ . Resposta disponível em `a03_05.m`.

### 3.6) Histograma usando a função imhist

Exemplos de imagens e respectivos histogramas [[OM], Tópico 9.3, Figura 9.2]:



Obter os histogramas das imagens *gDSC04422m16.png*, *42049\_20-200.png* e *vpfig.png*, utilizando a função `imhist` do MATLAB. Resposta disponível em *a03\_06.m*.

### 3.7) Explore o *live script* *a03\_07.mlx*

Apresenta um exemplo da aplicação de várias das operações ponto-a-ponto e baseadas no histograma para melhoramento de imagens.

### Referências

[OM] Oge Marques, *Practical image and video processing using MATLAB*, Wiley, 2011.