Universidade Tecnológica Federal do Paraná - UTFPR
Departamento Acadêmico de Eletrônica – DAELN
Prof. Gustavo B. Borba

ATIVIDADE 04
04.set.2018

ATENÇÃO
- Vale nota.
- Em dupla ou individual.
- Em dupla é em dupla: os dois fazem todas as atividades juntos!
- Leia todo o conteúdo! Os textos reforçam/complementam a teoria e os enunciados são bem claros.
- Apresentar quando todos estiverem funcionando ou até 22h30.

Arquivos necessários
1. moon.tif [MATLAB]
2. Fig0304(a)(breast_digital_Xray).tif
3. radio.tif
4. Fig0310(b)(washed_out_pollen_image).tif
5. coins.png [MATLAB]
6. pout.tif [MATLAB]

4.1) 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, para confirmar que a imagem de entrada é igual à imagem de saída. Também complemente o código incluindo instruções para verificar se as imagens realmente são iguais.
[[OM], Tutorial 8.1, Itens 1,2,3]

```
clear all, close all
I = imread('moon.tif');
y = uint8(0:255);
plot(y); xlim([0 255]); ylim([0 255]);
Ia = y(I + 1);
figure, subplot(1,2,1), imshow(I), title('Original');
subplot(1,2,2), imshow(Ia), title('Transformação');
```

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 *Fig0304(a)(breast_digital_Xray).tif* [[GW], Tópico 3.2.1, Figura 3.4; [GWm], Tópico 3.2.1]. Só pra saber: o MATLAB tem a função `imcomplement`, mas nesse exercício não é pra usá-la. Use o código do exemplo como template, pra aprender mais sobre o *MATLAB way-of-life* :-).

4.2) 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` [http://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`.

```
clear all, close all
I = imread('moon.tif');
y = uint8(0:255);
plot(y); xlim([0 255]); ylim([0 255]);
Ia = intlut(I,y);
figure, subplot(1,2,1), imshow(I), title('Original');
subplot(1,2,2), imshow(Ia), title('Transformação');
```

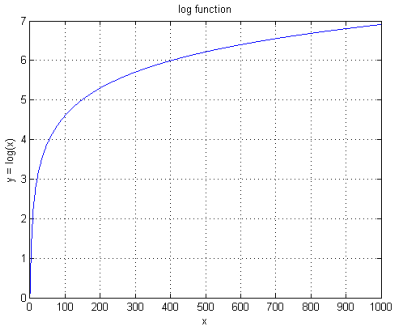
Crie uma função de transformação para obter o negativo da imagem *Fig0304(a)(breast_digital_Xray).tif* [[GW], Tópico 3.2.1, Figura 3.4; [GWm], Tópico 3.2.1], utilizando a função `intlut`. Só pra saber: o MATLAB tem a função `imcomplement`, mas nesse exercício não é pra usá-la. Use o código do exemplo como template, no mesmo espírito do exercício anterior.

4.3) Funções de transformação dos níveis de cinza ($\log_e$)
[[OM], Tópico 8.3.4; [GWm], Tópico 3.2.2, [GW], Tópico 3.2.2]

```
%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 $y = c*log(1+x)$. 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) = indeterminado* ($\rightarrow -\infty$).

Lembrando que, geralmente, *log* é o log_{10} (a base pode ser suprimida), log_e ou *ln* é o *logarítmo natural* ou *logarítmo 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. Porém, nos livros [OM, GWm, GW], *log* é a notação para log_e. Então, quando aparecer *log* nestes livros, a função correspondente no MATLAB é `log`. Ufa!

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 → 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 → uma faixa de poucos (faixa estreita) valores de níveis de cinza maiores na saída.

O efeito disto é a expanção 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 *log inversa* é $y = e^{(x/c)}-1$ [[OM], Tutorial 8.1, depois da Question 4].

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.

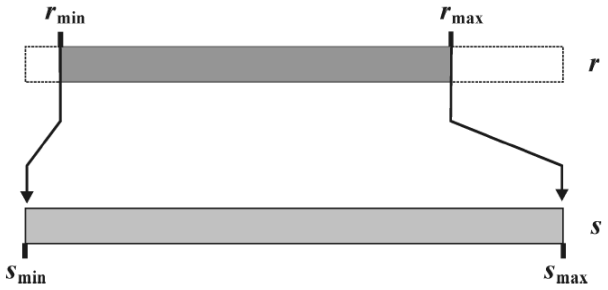
Criar uma função de transformação *log* (log_e) e aplicar na imagem *radio.tif*. Mostrar na mesma figure um plot da função, a imagem original e a ajustada. Não use normalização.

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

Aprendemos que a operação descrita a seguir é chamada de *normalização*.

$$s = \frac{L-1}{r_{max} - r_{min}} \cdot (r - r_{min})$$

[[OM], Equação 8.4]



[[OM], Figura 8.6]

Outros nomes utilizados para esta operação são *auto-contraste*, *contrast stretching* ou *histogram stretching*. Na verdade, os outros nomes talvez sejam até mais adequados, já que apenas 'normalização' não deixa claro se está realizando-se a operação *min(img) →0*, *max(img)→1*, ou apenas uma divisão por 255 (para imagens de entrada `uint8`).

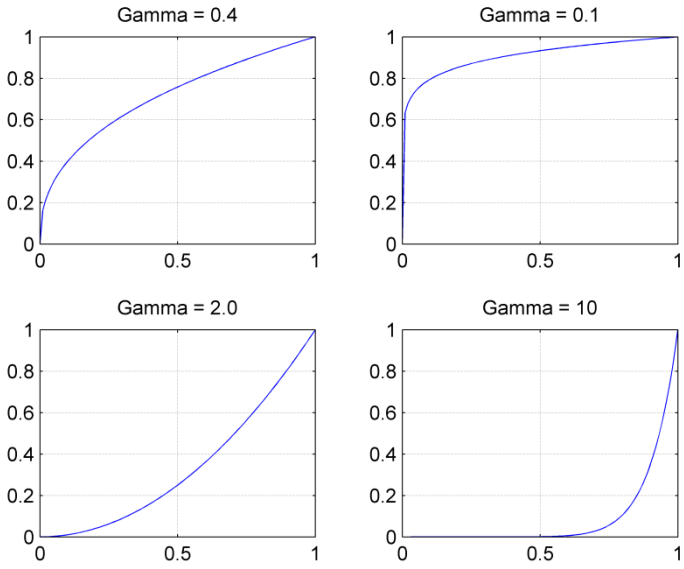
Fazer o auto-contraste de uma imagem de três formas diferentes: na unha, usando a função `mat2gray` e usando a função `imadjust` [http://www.mathworks.com/help/images/ref/imadjust.html]. Rode para a imagem *pout.tif*, mostrando as três saídas em uma mesma figure.

4.5) Funções de transformação dos níveis de cinza (gamma usando função imadjust)

[[GW], Tópico 3.2.3, Figura 3.8; [OM], Tópico 8.3.3; [GWm], Tópico 3.2.1, Figura 3.3]

```
%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 $y = c*x^{\gamma}$. 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`). γ 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.

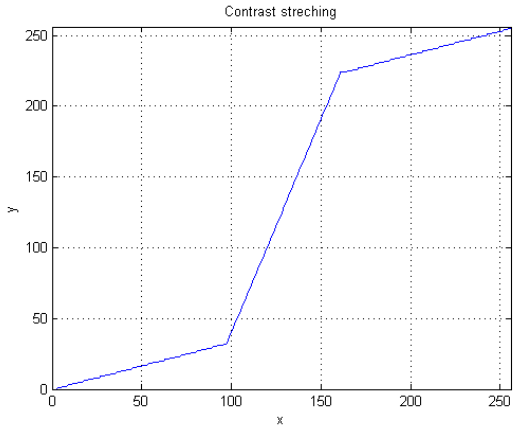
- Para $\gamma < 1$: A imagem 'fica mais clara'.
- Para $\gamma > 1$: A imagem 'fica mais escura'.
- Para $\gamma = 1$: Função identidade, isto é, a saída é igual à entrada (desde que c=1)

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 γ = 0,4, γ = 0,1 e γ = 2,0.

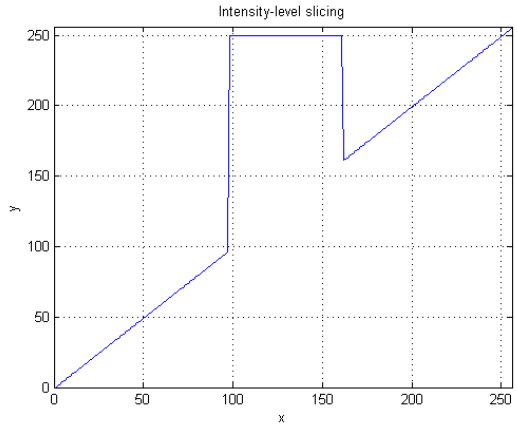
4.6) Funções de transformação dos níveis de cinza (piecewise linear usando intlut)

[[OM], Tópicos 8.3.5 e 8.4, Exemplos 8.4, 8.5 e 8.6; [GW], Tópico 3.2.4, Figuras 3.10 e 3.11]

```
%Contrast stretching
%Aloca uint8
%para depopis usar funcao intlut (y1 é a LUT)
y1 = uint8(zeros([1 256]));
%Equação da reta inferior y = (1/3)*x
y1(1:97) = (1/3)*(0:96);
%Equação da reta intermediária y = 3*x -256
y1(98:161) = 3*(97:160) - 256;
%Equação da reta superior y = (1/3)*x + 170
y1(162:256) = (1/3)*(161:255) + 170;
%Display
figure, plot(y1)
xlim([0 255]), ylim([0 255])
grid on
title('Contrast streching')
xlabel('x'), ylabel('y')
```




```
%Intensity-level slicing
%Aloca uint8
%para depopis usar funcao intlut (y2 é a LUT)
y2 = uint8(zeros([1 256]));
%Equação da reta inferior y = x (identidade)
y2(1:97) = 0:96;
%Equação da reta intermediária
%y = 250(um único nível de cinza cte)
y2(98:161) = 250;
%Equação da reta superior y = x (identidade)
y2(162:256) = 161:255;
%Display
figure, plot(y2)
xlim([0 255]), ylim([0 255])
grid on
title('Intensity-level slicing')
xlabel('x'), ylabel('y')
```



Contrast streching (alongamento do contraste) e intensity-level slicing (fatiamento de níveis de intensidade) são técnicas de ajuste de contraste que utilizam funções pecewise linear (pedaços lineares). Nos plots anteriores pode ser observada uma função piecewise linear para o contrast streching e outra para o intensity-level slicing.

A função contrast stretching mostrada no código mapeia:

- Reta inferior: os pixels escuros da faixa [0 96] em pixels mais escuros ainda na faixa [0 32], realizando assim uma compressão dos pixels escuros.
- Reta superior: os pixels claros da faixa [161 255] em pixels mais claros ainda na faixa [224 255], realizando assim uma compressão dos pixels claros.
- Reta intermediária: os pixels de nível médio da faixa [97 160] em pixel de nível médio na faixa [35 224], realizando assim um expansão dos pixels de nível de cinza médios.

O efeito disto é um destaque nos pixels de nível de cinza médio.

A função de intensity-level slicing mostrada mantém todos pixels na faixa [0 96] e [161 255] inalterados (função identidade) e os pixels da faixa [97 160] são mapeados para o valor 250.
O efeito disto é um destaque nos pixels da faixa correspondente ao 'slicing' (valor constante 250), como uma forma de selecionar o objeto para a facilitar a sua visualização, porém transformando toda a faixa de níveis de cinza do objeto em um único nível de cinza.

Aplique as funções de contrast streching e intensity-level slicing especificadas anteriormente (códigos exemplo) na imagem *Fig0310(b)(washed_out_pollen_image).tif*. Use a função `intlut` do MATLAB. Mostre a imagem original e as processadas em uma mesma figure.

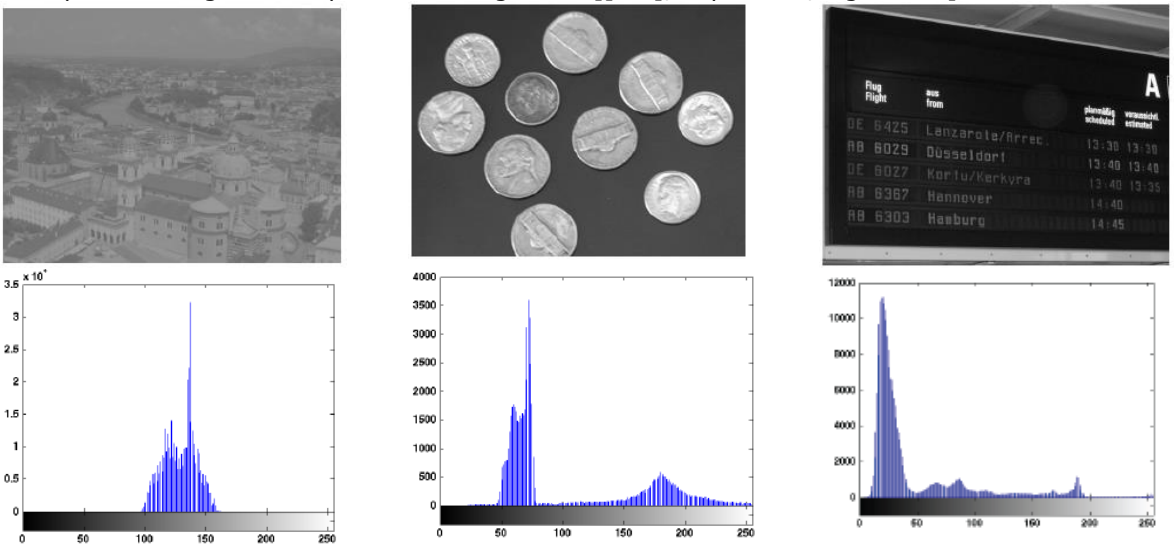
Meu to-do

Funções de transformação dos níveis de cinza (bit-plane slicing)

[[GW], Tópico 3.2.4, Figura 3.12; [OM], Tópico 5.4.3, Exemplo 5.2]

4.7) Histograma usando a função imhist

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



Obter os histogramas das imagens *coins.png* e *pout.tif* utilizando a função `imhist` do MATLAB. Mostrar cada par imagem/histograma em uma figure.

4.8) Cálculo do histograma na unha

Computar e plotar o histograma de uma imagem sem usar a função `imhist` ou `hist` ou .... Pode usar laços de repetição à vontade, mas tem que ser na unha. Mostrar uma figure com a imagem *pout.tif*, outra com o histograma que vc gerou e outra com o histograma do `imhist`, com o objetivo de comparar os dois. Dica: use um gráfico de barras (função `bar`).

4.9) Equalização do histograma usando a função histeq

Fazer a equalização do histograma de uma imagem usando a função `histeq` do MATLAB. Mostrar em uma figure a imagem original e o seu histograma e em outra figure a imagem processada e o seu histograma.

4.10) Equalização do histograma na unha

Fazer a equalização do histograma de uma imagem na unha. Pode usar a função `cumsum`. Mostrar a imagem *pout.tif* processada com o seu programa e com o `histeq(I,256)` do MATLAB e o histograma de cada uma. Os passos para a equalização do histograma estão descritos abaixo. O EXAMPLE 9.3 do livro [OM] também aborda a equalização de um histograma.

1. Obter o histograma da imagem original.
2. Normalizar este histograma [dividir por M*N (número de pixels da imagem)].
3. Obter a *cumulative distribution function* (cdf) [acumular o histograma do passo 2, isto é, *p(i) = Σ(j=0 até i) de p(j)*].
4. Transformar a cdf em *níveis de cinza arredondados* [multiplicar por 255 e transformar em `uint8`].
5. Aplicar o vetor do passo 4 como uma *função de transformação* sobre a imagem original, usando a função `intlut` do MATLAB.

Referências

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

[GWm] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, Digital image processing using MATLAB, Pearson Prentice Hall, 2004.

[GW] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, Digital image processing, Pearson Prentice Hall, 3rd ed, 2008.