



## Notas de aula e prática Octave #06

### Arquivos necessários

1. Lenna256g.png [Adaptada de <https://en.wikipedia.org/wiki/Lenna>]
2. Lenna256g\_ng20.png [Adaptada de <https://en.wikipedia.org/wiki/Lenna>]

### 6a) Separabilidade dos filtros lineares [[BB], Tópico 6.3.2]

Os filtros lineares recebem este nome porque podem ser implementados a partir da operação da convolução (a convolução é linear, só pra reforçar). O filtro Gaussiano e o box filter (média), por exemplo, são filtros lineares. O filtro da mediana, por exemplo, é não-linear. Relembrando algumas propriedades da convolução, no contexto de imagens:

- Linearidade: multiplicar a imagem por um escalar constante  $s$  e depois aplicar a convolução, dá o mesmo resultado que aplicar a convolução e depois multiplicar por  $s$ . Ainda, somar duas imagens e depois aplicar a convolução, dá o mesmo resultado que aplicar a convolução em cada imagem individualmente e somá-las.

$$(s \cdot I) * H = I * (s \cdot H) = s \cdot (I * H)$$

$$(I_1 + I_2) * H = (I_1 * H) + (I_2 * H)$$

- Comutatividade: convoluir o filtro com a imagem ou a imagem com o filtro (!) dá o mesmo resultado.

$$I * H = H * I$$

- Associatividade: sucessivos filtros lineares podem ser aplicados em qualquer ordem.

$$A * (B * C) = (A * B) * C$$

Agora, vamos considerar que podemos expressar o kernel de convolução desejado através da convolução de diferentes kernels. Um kernel que pode ser expresso dessa forma é chamado de **separável** (*separable*):

$$H = H_1 * H_2 * \dots * H_n$$

Então, a convolução de uma imagem com esse kernel é

$$I * H = I * (H_1 * H_2 * \dots * H_n)$$

Mas como a convolução é associativa, podemos escrever:

$$I * H = (\dots((I * H_1) * H_2) * \dots H_n)$$

Para ilustrar a coisa toda e compreender a implicação prática disso, vamos usar como exemplo um box filter 3x3. O kernel desse filtro é separável e pode ser escrito como mostrado a seguir (se quiser testar, faça  $H_x = [1/3 \ 1/3 \ 1/3]$ ;  $H_y = [1/3; 1/3; 1/3]$ ;  $H_{xy} = \text{conv2}(H_x, H_y);$ ).

$$\underbrace{\begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}}_{H_x} * \underbrace{\begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix}}_{H_y} = \underbrace{\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}}_{H_{xy}}$$

Isso significa que, para obter a imagem filtrada  $F$ , ao invés de convoluir  $H_{xy}$  com a imagem de entrada  $I$ , podemos fazer:

$$F = (I * H_x) * H_y$$

Isto é, convoluir separadamente: convoluir primeiramente  $I$  com  $H_x$  e, o resultado disso, convoluir com  $H_y$ .

Na prática: a convolução de  $H_{xy}$  com a imagem de entrada requer 9 ( $3 \times 3 = 9$ ) multiplicações para cada pixel. Já a convolução com  $H_x$  e depois com  $H_y$  requer 6 ( $3 + 3 = 6$ ) multiplicações para cada pixel. Então, no final das contas, a separabilidade permite a implementação mais eficiente de filtros lineares. Quanto maiores forem as dimensões do kernel, maior é este ganho.

"In general, the number of operations for a 2D filter grows quadratically with the filter size (side length) but only linearly if the filter is x/y-separable. Clearly, separability is an eminent bonus for the implementation of large linear filters" [[BB], Tópico 6.3.2, página 102].

### IMPORTANTE

- O filtro Gaussiano é separável. Em [[BB], Tópico 6.3.2] há uma análise detalhada.
- A versão separável não é a forma mais eficiente de implementar o box filter. O método que utiliza *integral images*, por exemplo, popularizado por [VJ], é mais eficiente (o documento [KD] apresenta uma boa revisão do método. Na Wikipedia, está em [IIw]).

## 6.1) Filtro separável

Elabore um script para comparar o desempenho do box filter 5x5 com a sua versão separável. Para capturar o tempo de execução no MATLAB, utilize as funções `tic` e `toc`. Não utilize `imfilter`, nem `conv`, nem `conv2` nem... faça tudo na unha, com `for` dentro de `for` à vontade. Assim, é possível garantir que você estará comparando laranjas com laranjas. Resposta disponível em a06\_01.m.

Esse exercício é interessante também para você reforçar o procedimento de uma convolução 2D e visualizar como é implementada em um nível mais baixo. Pra te ajudar, aí vai um código em C, mas fique à vontade pra implementar como achar melhor (desde que a comparação dos desempenhos ainda seja válida. Você pode reaproveitar código do exercício que pedia pra implementar o filtro da mediana na unha, de aulas anteriores). Só por curiosidade, em [JT] há  $n+1$  maneiras de se implementar convolução 2D.

[MW], Tópico 8.2, página 99:

"A C program segment to compute a simple discrete convolution is given next. It convolves a 3x3 mask with an MxM image to give a MxM convolved result. This convolution ignores a single pixel edge around the image.

```
for (i=1; i<M-2; ++i)
    for (j=1; j<M-2; ++j)
        Result[i][j] = Image[i-1][j-1] * mask[0][0] +
                        Image[i-1][j]   * mask[0][1] +
                        Image[i-1][j+1] * mask[0][2] +
                        Image[i][j-1]   * mask[1][0] +
                        Image[i][j]     * mask[1][1] +
                        Image[i][j+1]   * mask[1][2] +
                        Image[i+1][j-1] * mask[2][0] +
                        Image[i+1][j]   * mask[2][1] +
                        Image[i+1][j+1] * mask[2][2];"
```

## 6b) Filtros adaptativos

Conforme vimos na aula anterior e observamos na prática, através das atividades, ao suavizar o ruído de uma imagem acaba-se também suavizando as bordas, gerando o efeito indesejado de blurring e perda dos detalhes finos da imagem. Em imagens médicas, por exemplo, esta deterioração das bordas pode ser até mais prejudicial que em imagens convencionais, já que detalhes importantes das estruturas anatômicas ou de lesões podem ser corrompidos.

O objetivo de um filtro adaptativo é remover o máximo possível do ruído da imagem, comprometendo o mínimo possível a integridade das bordas e dos detalhes da imagem. Um filtro adaptativo não-linear simples para a suavização do ruído Gaussiano é o MMSE (Minimum Mean Square Error) [[SV], Equação 4; [MW] Tópico 11.4, Equação 26; [GD], Tópico 8.3.1, Equação 8.14; [GW], Tópico 5.3.3, Equação 5.3-12 (o que aqui chamamos de MMSE não recebe nenhum nome por GW)]. Cada pixel de saída do MMSE é obtido aplicando-se a equação a seguir para cada pixel correspondente da imagem de entrada. O MMSE também opera sobre uma janela quadrada, que define a vizinhança de pixels da imagem de entrada a serem utilizados no cálculo da saída. Para a descrição deste filtro, o termo *janela* é mais adequado que o termo *máscara*. O termo máscara costuma ser mais utilizado no contexto da convolução.

$$\hat{f}(i, j) = \left(1 - \frac{\sigma_n^2}{\sigma_l^2}\right) r(i, j) + \frac{\sigma_n^2}{\sigma_l^2} m_l$$

Onde  $r(i, j)$  é a imagem a ser filtrada (com ruído),  $\sigma_l^2$  é a variância dos pixels pertencentes à janela e  $m_l$  a média dos pixels pertencentes à janela.  $\sigma_n^2$  é a variância do ruído, que será uma constante. A variância do ruído ( $\sigma_n^2$ ) deve ser estimada antes e utilizada como um parâmetro do filtro, da seguinte forma: obter a variância de pixels pertencentes a uma região escura (sem sinal, portanto apenas ruído) e homogênea (sem bordas) da imagem.

As imagens a seguir mostram a cameraman original, com ruído, e processada com o MMSE.



## 6.2) Filtros adaptativos (MMSE)

Interpretando a equação, explique o princípio de funcionamento do MMSE. Dica: suponha as situações nas quais i) a variância da janela é igual à variância estimada do ruído e ii) a variância da janela é alta em relação à variância estimada do ruído. Lembre que: a principal finalidade do MMSE é suavizar o ruído e preservar as bordas e que a variância da janela é alta quando a ela está posicionada sobre uma borda. Resposta disponível nos slides.

## 6.3) Filtros adaptativos (MMSE)

Crie um script que implementa o filtro adaptativo descrito. Lembre: os parâmetros de entrada do filtro são a matriz imagem a ser filtrada, as dimensões da janela, sempre quadrada, e a variância estimada do ruído (uma constante). Utilize a função `imcrop` para permitir a interação do usuário para selecionar a região da imagem que será utilizada para o cálculo da variância do ruído.

Teste o filtro para uma janela de 5x5 para a imagem *Lenna256g\_ng20.png* (ruído Gaussiano). Mostrar os resultados (imagem de entrada e imagem de saída). Compare com a saída do filtro da média (box filter) de janela de mesmas dimensões para verificar a afirmação de que este filtro é melhor em preservar as bordas e se ele suaviza o ruído tanto quanto o filtro da média. **OBS:** se quiser testar com imagens mais/menos degradadas pelo ruído Gaussiano, ou outro tipo de ruído, basta inserir o tipo e o nível de ruído desejado na Lenna original (*Lenna256g.png*). Para isso, pode-se utilizar a função `imnoise`, conforme mostrado no item 5b (aulas anteriores). Resposta disponível em a06\_03.m.

### 6c) Métricas objetivas para a medida de qualidade de imagem

Quando comparamos imagens visualmente e dizemos, por exemplo, “o ruído na imagem A foi mais suavizado que na imagem B” ou “a imagem C está menos borrada e portanto melhor que a imagem D”, estamos fazendo uma avaliação *qualitativa*. Estas avaliações são importantes, mas pouco relevantes cientificamente, pois não podem ser reproduzidas por outros pesquisadores para a comparação de diferentes algoritmos. É possível coletar a opinião de uma quantidade significativa de pessoas a respeito da qualidade da imagem, com base em um sistema de pontuação pré-definido. Este é um método subjetivo de avaliação de qualidade de imagem, denominado *mean opinion score* (MOS). Um MOS é uma métrica válida, pois é *quantitativa*, mas difícil de ser obtida, já que envolve voluntários, isto é, observadores humanos. Assim, são necessárias métricas *quantitativas objetivas* para a medida do desempenho dos algoritmos de filtragem. As métricas utilizadas para isto são denominadas de *métricas objetivas para a medida de qualidade de imagem* [WBL]. Atualmente, não existe uma métrica objetiva universalmente aceita para a medida de qualidade de imagem. Uma das métricas objetivas do tipo *full-reference* (definição a seguir) mais adotadas é a *mean structural similarity* (MSSIM) [WBo], disponível no MATLAB a partir da versão 2014a, função `ssim` da IPT (<http://www.mathworks.com/help/images/ref/ssim.html>). Não está disponível no Image Package do Octave, mas há implementações por aí, como a dos próprios autores, por exemplo: <https://ece.uwaterloo.ca/~z70wang/research/ssim/#usage>.

As métricas objetivas podem ser classificadas ainda em função da necessidade ou não da imagem original (sem ruído, não degradada) para o seu cálculo. Se uma métrica requer a imagem original ela é do tipo *full-reference*. Se não é necessária a imagem original, a métrica é do tipo *no-reference* ou *blind-metric*.

Uma métrica objetiva bastante utilizada é a *relação sinal-ruído* (SNR, signal-to-noise ratio). Genericamente,  $SNR = \text{sinal/ruído}$ . Quanto maior for a SNR, melhor, já que maior é o sinal em relação ao ruído nele presente.

- [WB], Tópico 2.6, Equação 2.5:

$$SNR_L = \frac{m_s}{\sigma_n}$$

A  $SNR_L$  é uma métrica do tipo *no-reference*, utiliza apenas a imagem que se deseja avaliar.  $m_s$  é a média do sinal, estimada considerando-se os pixels de uma região clara (sinal) e homogênea (sem bordas) da imagem.  $\sigma_n$  é o desvio padrão do ruído, estimado considerando-se os pixels de uma região escura (sem sinal, portanto, apenas ruído) e homogênea da imagem. Quanto maior o valor de  $SNR_L$ , melhor.

- [ASP], Equação 10:

$$PSNR = 20 \log \frac{L}{\sqrt{\sum_{i=1}^M \sum_{j=1}^N \frac{(\hat{f}(i,j) - f(i,j))^2}{MN}}} [dB]$$

A PSNR (o P significa *Peak*) é uma métrica do tipo *full-reference*.  $L$  é a faixa dinâmica dos pixels (255 para imagens de 8 bits por pixel),  $M$  e  $N$  são o número linhas e colunas da imagem,  $\hat{f}(i,j)$  os pixels da imagem que se quer avaliar e  $f(i,j)$  os pixels da imagem original (sem ruído). Quanto maior o valor de PSNR, melhor. Disponível no Octave, função `psnr` do Image Package (<https://octave.sourceforge.io/image/function/psnr.html>).

- [GW], Tópico 5.8, Equação 5.8-4:

$$MSE = \frac{\sum_{i=1}^M \sum_{j=1}^N (f(i,j) - \hat{f}(i,j))^2}{MN}$$

A *mean squared error* (MSE) é do tipo *full-reference*.  $M$  e  $N$  são o número linhas e colunas da imagem,  $f(i,j)$  os pixels da imagem que se quer avaliar e  $\hat{f}(i,j)$  os pixels da imagem original (sem ruído). Quanto menor o valor de MSE, melhor. Disponível no Octave, função `immse` do Image Package (<https://octave.sourceforge.io/image/function/immse.html>).

#### 6.4) Avaliação quantitativa dos algoritmos de filtragem de ruído (SNR)

Implemente a métrica  $SNR_L$ . Utilize a função `imcrop` para permitir a interação do usuário para selecionar as regiões da imagem correspondente ao sinal ( $m_s$ ) e ao ruído ( $\sigma_n$ ). Obtenha os valores da  $SNR_L$  para a imagem *Lenna256g\_ng20.png* (ruído Gaussiano), filtrada com a média 3x3 e 7x7. Use as mesmas regiões para as medidas nas duas imagens. É possível afirmar que esta métrica captura com fidelidade as diferenças percebidas pelo sistema visual humano na qualidade das imagens? Pelo tipo da pergunta, você já deve ter notado que a resposta é não. Justifique utilizando os resultados do exercício. Resposta disponível em a06\_04.m.

#### Referências

- [BB] Wilhelm Burger, Mark Burge, Digital image processing – an algorithmic introduction using Java, Springer, 2010.
- [GW] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, Digital image processing, Pearson Prentice Hall, 3<sup>rd</sup> ed, 2008.
- [VJ] Paul Viola, Michael Jones, Rapid object detection using a boosted cascade of simple Features, Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - CVPR 2001, vol. 1, p. I:511–518, 2001.
- [KD] Konstantinos G. Derpanis, Notes in Computer vision, Integral image-based representations, Department of Computer Science and Engineering, York University, 2007. Disponível em [http://www.cse.yorku.ca/~kosta/CompVis\\_Notes/integral\\_representations.pdf](http://www.cse.yorku.ca/~kosta/CompVis_Notes/integral_representations.pdf)
- [IIw] Wikipedia, Summed Area Table [http://en.wikipedia.org/wiki/Summed\\_area\\_table](http://en.wikipedia.org/wiki/Summed_area_table).
- [WB] Wolfgang Birkfellner, Applied medical image processing: a basic course, CRC Press, 2011.
- [ASP] C. B. Ahn, Y. C. Song, D. J. Park, Adaptive Template Filtering for Signal-to-Noise Ratio Enhancement in Magnetic Resonance Imaging, IEEE Transactions on Medical Imaging, vol. 18, no. 6, p. 549-556, June 1999.
- [WBL] Zhou Wang, Alan C. Bovik, Ligang Lu, Why is image quality assessment so difficult?, IEEE International conference on Acoustics, Speech & Signal Processing, May 2002. Disponível em <https://ece.uwaterloo.ca/~z70wang/publications/icassp02a.html>
- [WBo] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, Eero P. Simoncelli, Image quality assessment: from error visibility to structural similarity, IEEE Transactions on Image Processing, vol. 13, no. 4, p. 600-612, 2004. Disponível em <https://ece.uwaterloo.ca/~z70wang/publications/ssim.pdf>
- [GD] Geoff Dougherty, Digital image processing for medical applications, Cambridge University Press, 2009. Imagens disponíveis em [www.cambridge.org/dougherty](http://www.cambridge.org/dougherty)
- [SV] Sun, X.Z., Venetsanopoulos, A.N., Adaptive schemes for noise filtering and edge detection by use of local statistics, IEEE Transactions on Circuits and Systems, vol. 35, n. 1, p. 57-69, DOI 10.1109/31.1700, 1998.
- [MW] Harley R. Myler, Arthur R. Weeks, Computer imaging recipes in C, Prentice-Hall, 1993.
- [JT] Jan Thorbecke, 2D Convolution Algorithms, 2000. Disponível em <http://janth.home.xs4all.nl/Publications/html/conv2d.html>