

Universidade Estadual de Campinas  
MC970 - Introdução ao Processamento de Imagem Digital  
Prof. Dr. Hélio Pedrini

## **Trabalho 2**

Mateus de Lima Almeida  
242827

28 de abril de 2025

# 1 Introdução

O presente trabalho aborda a implementação, realização e exploração de técnicas de meios-tons e de manipulação de imagens no domínio da frequência sugeridas em sala na matéria de Introdução ao Processamento de Imagem Digital (MC920). Com o objetivo de introduzir, solidificar e explorar o conhecimento aos conceitos de técnicas de meios-tons e de processamento de imagens no domínio da frequência através da utilização da transformada de Fourier. Neste presente relatório, imagens foram transformadas, processadas, e seus resultados foram analisados.

Este relatório detalha a metodologia adotada na execução dos exercícios sugeridos e também realiza a discussão de seus resultados, bem como a exploração de hipóteses surgidas no processo, junto também com a discussão de seus testes e resultados.

## 2 Materiais e Métodos

Para todas as tarefas a seguir, os materiais utilizados foram a linguagem Python 3.9 junto com as bibliotecas OpenCV2, Scikit-Image, Numpy e Pillow. Para utilização das bibliotecas e execução dos códigos, também foi utilizado um ambiente virtual (venv), onde foram instaladas as bibliotecas citadas.

### 2.1 Organização de Arquivos

A pasta do projeto foi organizada de modo a facilitar a visualização dos resultados de cada exercício de maneira independente. Dentro da pasta estão:

- Arquivos Python de cada exercício (de 1 a 2)
- Arquivo *requirements.txt* com as dependências necessárias do projeto.
- Pasta *images* que contém:
  - Imagens PNG que serão os Inputs dos exercícios
  - Pasta *outputs* que contém pastas (enumeradas de 1 a 2) contendo as imagens-saída de cada exercício (as pastas contendo as saídas de cada exercício podem ter, dentro delas, pastas separando algumas categorias de saída, como coloridas, monocromáticas e zooms, apenas por uma questão de facilidade na análise).

### 2.2 Versão Python e Bibliotecas

A versão de Python utilizada, junto com as bibliotecas e suas versões foram as seguintes:

- Python 3.9.1
- Pillow 11.1.0
- Opencv-python 4.11.0.86
- Numpy 2.0.2
- Matplotlib 3.9.4

## 2.3 Execução dos códigos

Os códigos feitos para realizar as operações e transformações sugeridas neste trabalho foram todos executados dentro de um ambiente virtual *venv*, então é recomendável que para areprodução seja feito o mesmo. É necessário que as dependências de *requirements.txt* sejam instaladas.

Para executar um determinado programa, basta inserir o comando básico para rodar um arquivo *.py* no terminal (certifique-se de estar na pasta correta).

```
python ex1.py
```

O arquivo exemplo utilizado foi o *ex1.py*, mas a pasta contém arquivos enumerados de 1 à 2. Em seguida, basta digitar o nome da imagem que se deseja usar, ou seja, se desejamos executar *ex2.py* e usar a imagem *city.png*, deve-se fazer:

```
python ex2.py  
Digite o nome da imagem (sem '.png'): city
```

Para o arquivo *ex1.py*, após inserir o nome da imagem, o programa oferecerá todas as opções de varredura da seguinte forma:

Distribuições:

```
1 - floyd_steinberg  
2 - stevenson_acre  
3 - burkes  
4 - sierra  
5 - stucki  
6 - jarvis_judice_ninke  
7 - Todas (aplica todas as distribuições e varreduras, gerando 12 imagens)
```

Qual distribuição deseja aplicar? (Digite o número):

Para selecionar, basta digitar o número que corresponde à opção desejada. Caso a opção selecionada não seja a 7, o programa perguntará que tipo de varredura deseja-se aplicar da seguinte forma:

Varreduras:

```
1 - Esquerda para direita  
2 - Alternada (zigue-zague)
```

Qual varredura deseja aplicar? (Digite o número):

Para selecionar, basta apenas digitar o número correspondente à varredura desejada. Caso a opção selecionada para distribuição seja a opção 7 (Todas as distribuições), o programa automaticamente executará todas os tipos de distribuição utilizando tanto a varredura esquerda\_direita quanto a varredura alternada, gerando 12 imagens.

A imagem selecionada deve estar presente na pasta *images*. A seguir, uma lista das imagens presentes na pasta, que podem ser usadas como input:

- Imagens Monocromáticas:
  - baboon\_monocromatica
  - blue\_sky

- city

- gradient

- iwakura\_lain

- Imagens RGB:

- baboon\_colorida

- windmill

Caso seja desejado, mais imagens podem ser adicionadas manualmente na pasta *images* para serem usadas como input.

## 3 Resultados, testes e discussão

### 3.1 Técnicas de Meios-Tons

#### 3.1.1 Floyd-Steinberg e Tipos de Varreduras

O objetivo da primeira tarefa proposta foi aplicar técnicas de meios-tons com difusão de erro em imagens. As distribuições propostas foram as seguintes:

$$\begin{array}{ll}
 \text{Floyd e Steinberg} \rightarrow & \begin{bmatrix} f(x, y) & \frac{7}{16} \\ \frac{3}{16} & \frac{5}{16} \end{bmatrix} \\
 & \text{Stevenson e Arce} \rightarrow \begin{bmatrix} \frac{12}{200} & & & f(x, y) & \frac{32}{200} \\ & \frac{12}{200} & \frac{26}{200} & \frac{30}{200} & \frac{16}{200} \\ \frac{5}{200} & & \frac{26}{200} & \frac{12}{200} & \frac{12}{200} \\ & \frac{12}{200} & & \frac{12}{200} & \frac{5}{200} \end{bmatrix} \\
 \text{Burkes} \rightarrow & \begin{bmatrix} f(x, y) & \frac{8}{32} & \frac{4}{32} \\ \frac{2}{32} & \frac{4}{32} & \frac{8}{32} \end{bmatrix} \quad \text{Sierra} \rightarrow \begin{bmatrix} f(x, y) & \frac{5}{32} & \frac{3}{32} \\ \frac{2}{32} & \frac{4}{32} & \frac{5}{32} \\ \frac{2}{32} & \frac{3}{32} & \frac{4}{32} \\ \frac{3}{32} & \frac{2}{32} & \frac{3}{32} \end{bmatrix} \\
 \text{Stucki} \rightarrow & \begin{bmatrix} f(x, y) & \frac{8}{42} & \frac{4}{42} \\ \frac{2}{42} & \frac{4}{42} & \frac{8}{42} \end{bmatrix} \quad \text{Jarvis, Judice e Ninke} \rightarrow \begin{bmatrix} f(x, y) & \frac{7}{48} & \frac{5}{48} \\ \frac{3}{48} & \frac{5}{48} & \frac{7}{48} \\ \frac{1}{48} & \frac{3}{48} & \frac{5}{48} \\ \frac{1}{48} & \frac{1}{48} & \frac{3}{48} \end{bmatrix}
 \end{array}$$

As técnicas de meios-tons acima foram implementadas de forma iterativa, onde a cada pixel iterado, foi calculado um erro, que representa a diferença entre o pixel atual (valor exato) e o pixel de uma versão quantizada da mesma imagem (valor aproximado). A imagem quantizada foi feita seguindo a regra de que, todo pixel com valor superior à 127 teria seu valor mudado para 255, e os pixels com valor menor ou igual à 127 teriam seu valor mudado para 0. Desta forma, a imagem quantizada é uma versão preta e branca (apenas valores 0 e 255) da imagem monocromática original.

A cada iteração, o erro atual foi distribuído entre os pixels vizinhos seguindo o padrão das matrizes mostradas, onde o valor do erro foi multiplicado por cada peso e aplicado ao pixel daquela posição. Por exemplo, usando a técnica de *Floyd e Steinberg*, a cada pixel  $f(x, y)$  analisado, o erro foi multiplicado por  $7/16$  e somado ao pixel da direita, multiplicado por  $5/16$  e somado ao pixel abaixo, e assim por diante.

No código, as matrizes de distribuição de erros foram guardadas em um dicionário e representadas por listas contendo, para cada peso presente na matriz, o valor do peso e uma lista com a "coordenada" daquele peso na matriz. Como no exemplo abaixo:

```
[[[1, -1], 3/16], [[1, 0], 5/16], [[1, 1], 1/16], [[0, 1], 7/16]]
```

Observando o exemplo para o método de *Floyd e Steinberg* acima, um dos elementos da matriz seria uma lista com  $3/16$ , e as coordenadas  $(1, -1)$ , onde o valor 1 significa que o alvo é 1 pixel abaixo do pixel atual, e o valor -1 significa que o alvo está 1 pixel à esquerda. Esses valores fazem sentido tendo em mente o sistema de coordenadas de imagem, que tem sua origem no canto superior esquerdo e cresce para baixo. Todos os demais valores e as demais matrizes seguem a mesma lógica.

O resultado esperado é uma imagem com 2 níveis de cinza (preto e branco, apenas), mas que, devido ao posicionamento e agrupamento dos pixels, causa a impressão de que existem mais níveis de cinza na imagem. A imagem escolhida para realizar os testes iniciais foi a imagem *baboon\_monocromatica.png* (Figura 8).

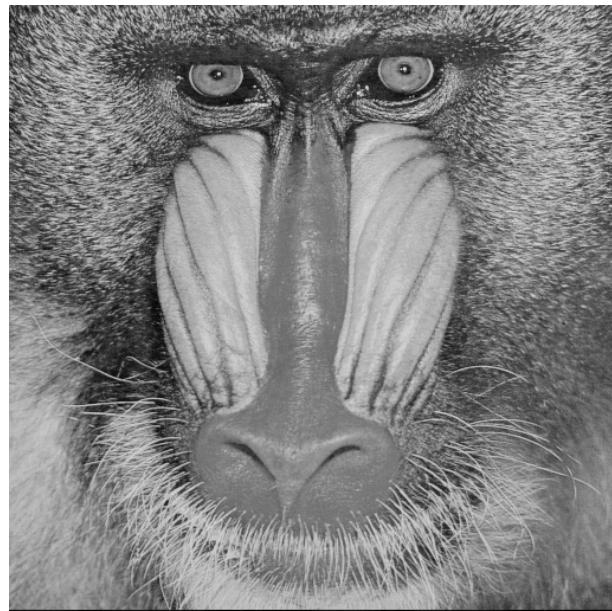


Figura 1: *baboon\_monocromatica.png*

Após aplicar na figura acima a técnica de meios-tons com a abordagem de *Floyd e Steinberg*, o resultado obtido é o visto nas Figuras 2 e 3:

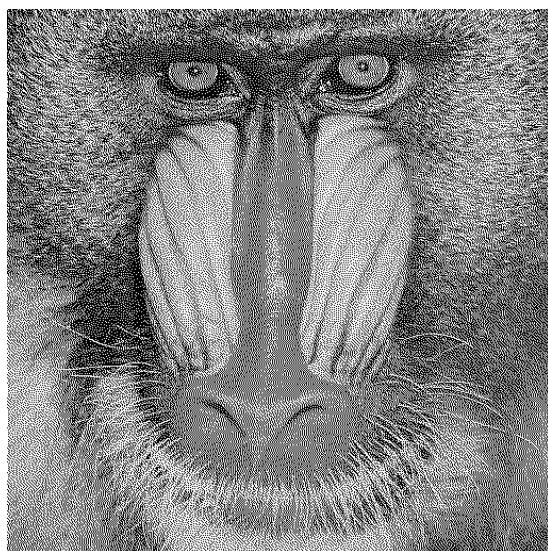


Figura 2: floyd steinberg aplicada



Figura 3: zoom na imagem resultante

Como é possível ver nas imagens acima, o resultado obtido combina com a descrição do resultado esperado para a saída. A imagem resultante se assemelha muito (principalmente se vista de longe/menor) com a imagem original, porém, ao olharmos mais atentamente (e de forma mais próxima) para o resultado (Figura 3), podemos notar que a imagem é composta apenas por pixels brancos (valor 255) ou pretos (valor 0).

Como foi explicado anteriormente, o método utilizado anteriormente consistiu em iterar pixel a pixel, sempre da esquerda para a direita e de cima para baixo (Figura 4), porém, como sugerido no comando da tarefa, a ordem em que a iteração foi feita pode gerar alguns padrões indesejados na figura. A solução para isso é modificar a ordem de iteração para alternar a ordem de acordo com as linhas, como um padrão "zigue-zague" (Figura 5).

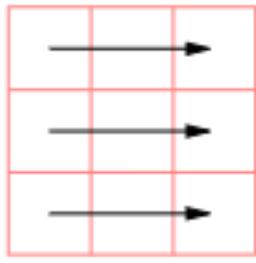


Figura 4: varredura da esquerda à direita

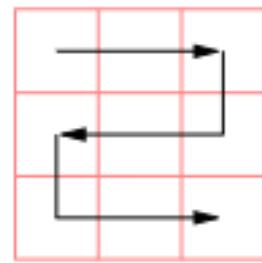


Figura 5: varredura alternada

Ao utilizar a varredura alternada, nas linhas onde a direção da iteração é da direita para a esquerda, a matriz de distribuição de erros também é invertida (espelhada de forma horizontal), para que os erros sejam sempre propagados no mesmo sentido com relação à direção da iteração. Aplicando agora a mesma abordagem de *Floyd e Steinberg*, porém utilizando varredura alternada, obtemos o resultado a seguir (Figura 6):

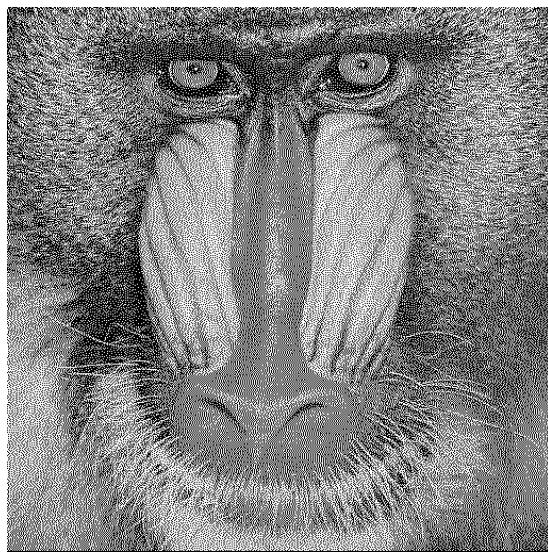


Figura 6: floyd steinberg aplicada com varredura alternada



Figura 7: zoom na imagem resultante

Como é possível observar, com a imagem escolhida, os resultados são extremamente parecidos com os obtidos através da varredura convencional, porém, realizando uma operação

de subtração entre as imagens, isto é, subtraindo cada pixel da Figura 6 do seu pixel correspondente da Figura 2, é possível ver que existem diferenças entre os resultados, essa subtração pode ser vista a seguir:

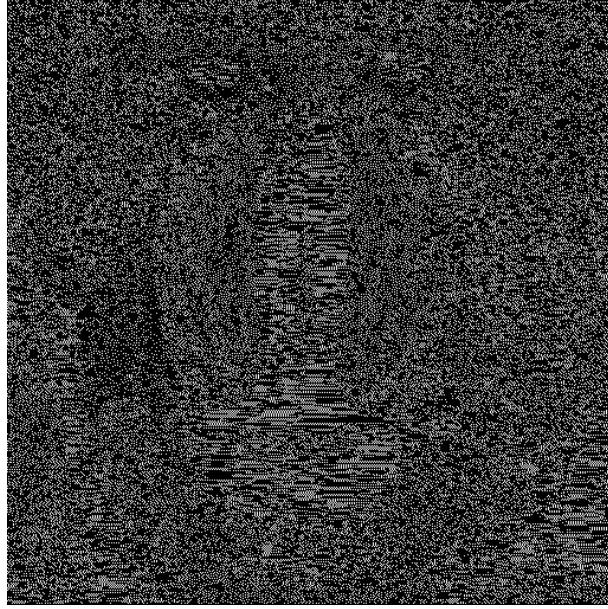


Figura 8: subtração entre resultados das varreduras

Naturalmente, como as duas imagens de entrada só possuem valores 0 e 255, os pontos brancos da subtração indicam locais onde os dois pixels eram diferentes (em casos onde a subtração deu um valor negativo, o valor foi tratado como módulo).

Porém, como o intuito inicial de executar uma varredura diferente era remover/atenuar padrões indesejados que podem surgir durante o processo, e com a varredura convencional não obtivemos padrões indesejados grandes e relevantes o suficiente para serem notados. Portanto, faz-se necessário utilizar outra imagem se quisermos observar diferenças entre as varreduras. A fim de poder observar claramente a diferença entre as duas varreduras, foi escolhida uma imagem onde o surgimento de padrões indesejados já é conhecido neste tipo de técnica: degradês. Isso acontece pois degradês são, essencialmente compostos por muitos meios-tones entre preto e branco, então, ao aplicar a quantização binária na imagem, e depois aplicando o algoritmo, o resultado precisa simular essa transição gradual apenas através do padrão de pontos. Como os erros são propagados para direções específicas (no caso de *Floyd e Steinberg*, mais para baixo e para a direita), padrões refletindo esses formatos começam a surgir, pois o erro começa a ser acumulado e propagado sempre para a mesma direção (nesse caso, por causa da direção da matriz, há o surgimento de padrões diagonais). A imagem escolhida para demonstrar isso é *gradient.png* (Figura 9).



Figura 9: gradient.png

Após aplicar *Floyd e Steinberg* com a varredura convencional e a varredura alternada, obtemos:

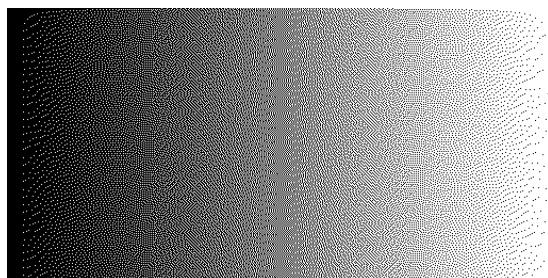


Figura 10: gradient.png com varredura convencional

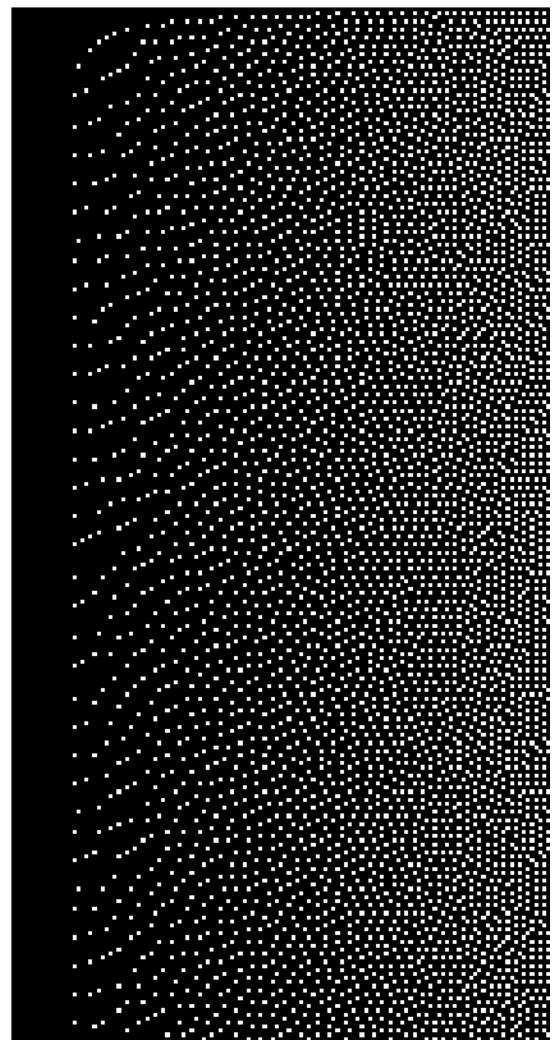


Figura 11: zoom na imagem resultante

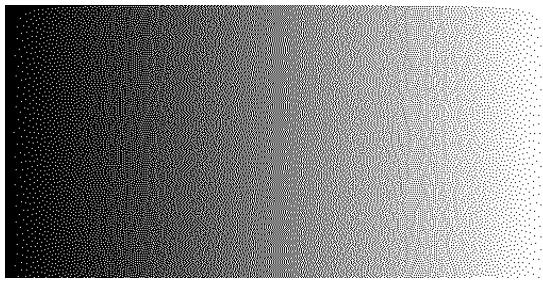


Figura 12: gradient.png com varredura alternada

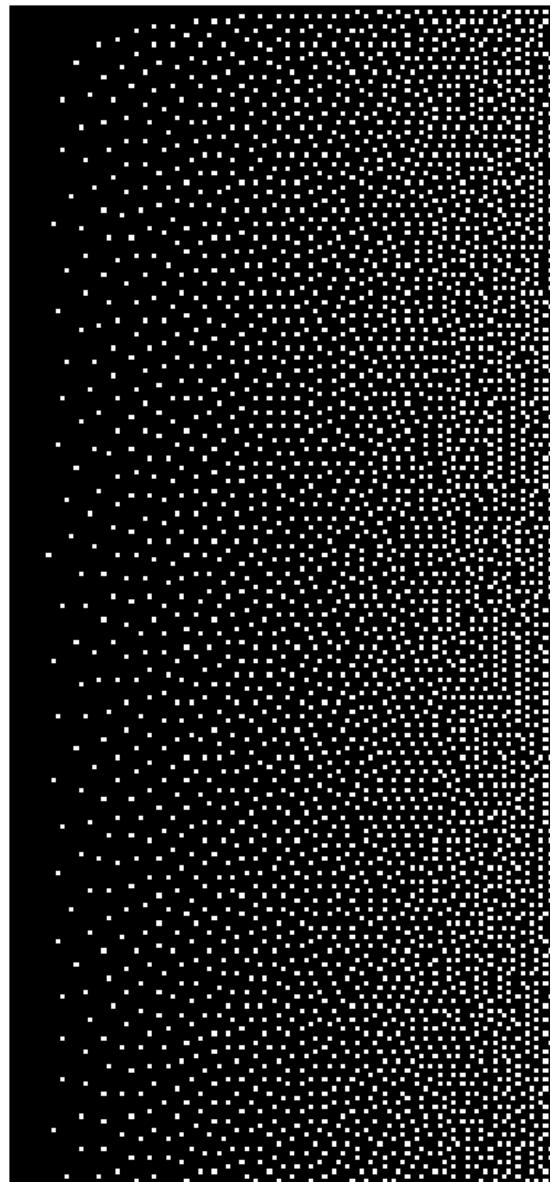


Figura 13: zoom na imagem resultante

Como é possível observar acima nas Figuras 10 e 12 e em seus respectivos zooms, na imagem resultante da varredura convencional, ao final do degradê (ocorre nas duas pontas, embora apenas a ponta esquerda tenha sido selecionada no zoom), padrões (linhas/curvas) diagonais de pixels começam a se formar, enquanto na imagem gerada através da varredura alternada, esses padrões diagonais não existem, e os pixels estão dispostos de forma mais uniforme no espaço. Em imagens onde existem grandes áreas com degradês de cores, como fotos onde o céu aparece, ou imagens com grandes áreas de sombra/penumbra, existe uma grande chance do aparecimento desse tipo de padrão indesejado, e, como visto, a varredura alternada é uma boa ferramenta para solucionar esse problema.

### 3.1.2 Análise das demais abordagens

Como visto anteriormente, além da abordagem de *Floyd e Steinberg*, também foram sugeridas outras 5 abordagens para aplicação da técnica de meios-tonos. A aplicação dessas demais abordagens foi feita da mesma forma descrita na seção passada, mudando apenas

os valores nas listas que representam cada matriz. Os resultados podem ser vistos a seguir (Figura 14 a Figura 19) e em seus respectivos zooms (Figura 20 a Figura 25), por uma questão de simplicidade, só serão apresentados os resultados obtidos com a varredura alternada.

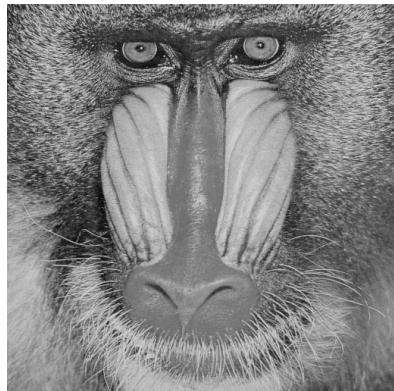


Figura 14: Imagem original

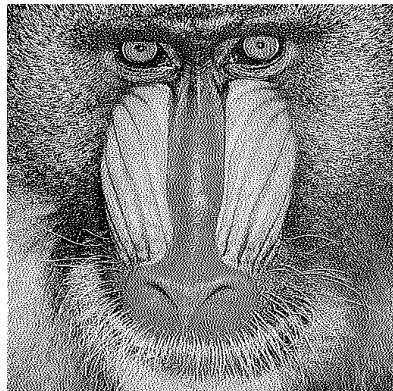


Figura 15: Stevenson e Arce

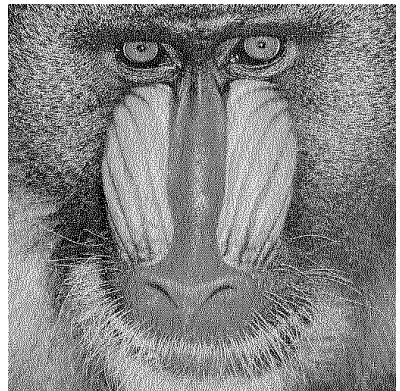


Figura 16: Burkes

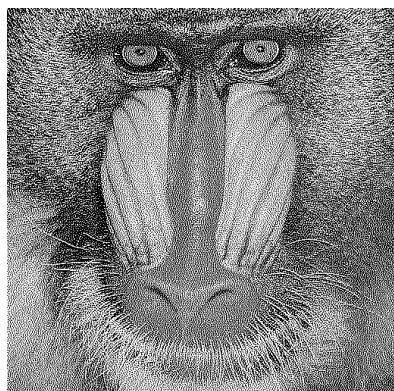


Figura 17: Sierra

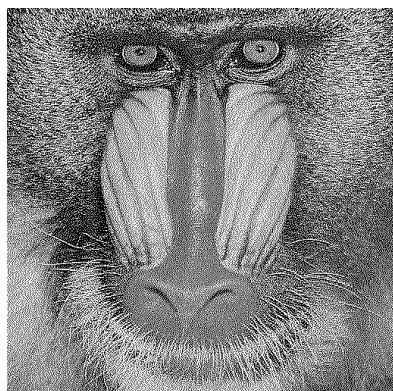


Figura 18: Stucki

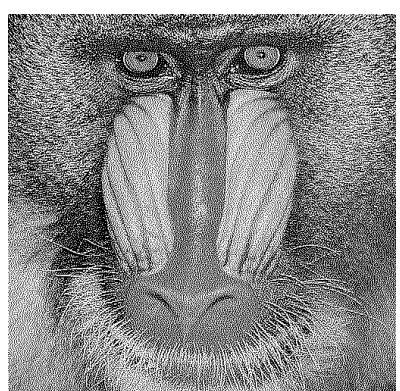


Figura 19: Jarvis, Judice e Ninke

Olhando as imagens mais de perto:



Figura 20: Zoom na imagem original

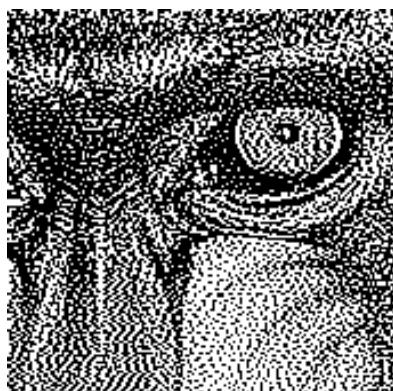


Figura 21: Zoom em Stevenson e Arce

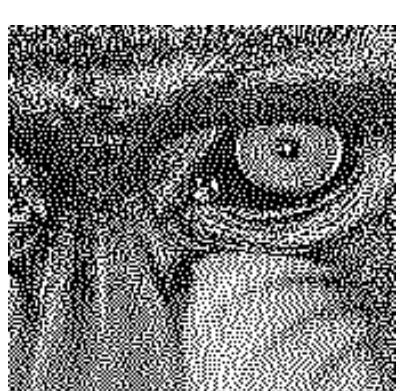


Figura 22: Zoom em Burkes

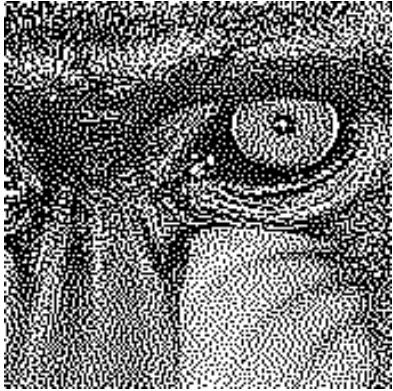


Figura 23: Zoom em Sierra

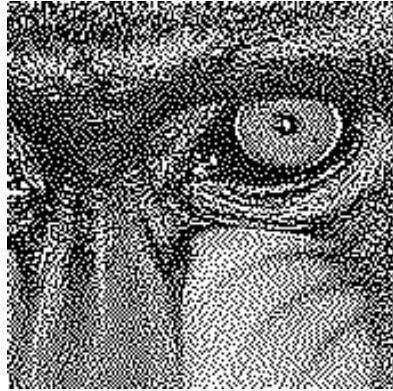


Figura 24: Zoom em Stucki

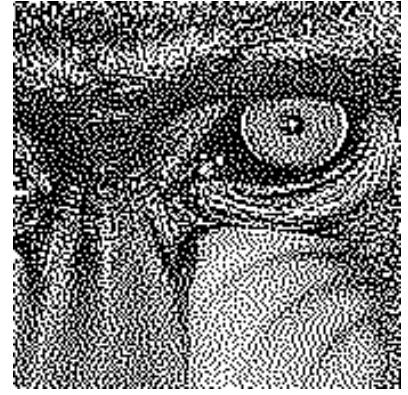


Figura 25: Zoom em Jarvis, Judice e Ninke

Poucas conclusões podem ser tomadas apenas olhando as imagens de forma rápida e superficial, mas uma conclusão que pode ser tomada prontamente é que a abordagem de *Stevenson e Arce* resultou em uma imagem com áreas escuras mais densas, isto é, mais pixels pretos agrupados, quando comparada às outras imagens. Isso pode ser explicado pelo formato da matriz de distribuição de *Stevenson e Arce*, que além de ser maior e mais ampla que as outras, também realiza uma distribuição mais leve, onde os pesos têm valores menores (pois existem mais pixels vizinhos a serem modificados). A combinação dessa distribuição ampla e "leve" faz com que as áreas escuras prevaleçam pois a distribuição demora mais para compensar as partes claras da imagem, causando o resultado visto, onde regiões escuras possuem pixels pretos mais agrupados.

Também por uma questão de simplicidade, e espaço, não serão feitas todas as comparações possíveis entre os 5 resultados, ou seja, em vez de realizar comparações exaustivamente entre as abordagens, serão escolhidas duas abordagens com resultados parecidos para realizarmos uma comparação mais aprofundada sobre suas possíveis diferenças. A comparação será realizada entre os resultados obtidos através das abordagens de *Burkes* e *Stucki*.

$$\begin{array}{c}
 \text{Burkes} \\
 \left[ \begin{array}{ccccc}
 \frac{2}{32} & \frac{4}{32} & f(x, y) & \frac{8}{32} & \frac{4}{32} \\
 & & \frac{8}{32} & \frac{4}{32} & \frac{2}{32}
 \end{array} \right] \\
 \text{Stucki} \\
 \left[ \begin{array}{ccccc}
 \frac{2}{42} & \frac{4}{42} & f(x, y) & \frac{8}{42} & \frac{4}{42} \\
 \frac{1}{42} & \frac{2}{42} & & \frac{4}{42} & \frac{2}{42} \\
 & & & \frac{2}{42} & \frac{1}{42}
 \end{array} \right]
 \end{array}$$

Embora as diferenças sejam quase imperceptíveis nas imagens mostradas, ao analisar as matrizes das duas abordagens e compará-las, podemos chegar à algumas conclusões, sendo a principal delas o fato de que a matriz de *Stucki* possui uma linha a mais que a matriz de *Burkes*, fazendo com que o erro seja distribuído para mais pixels. Isso faz com que o resultado obtido por *Stucki* seja levemente mais suavizado do que o resultado de *Burkes*, e da mesma forma, podemos concluir que *Burkes*, com sua matriz menor, produz um resultado final que preserva mais detalhes nítidos, principalmente em regiões de transição (detalhes esses que são um pouco mais suavizados no caso de *Stucki*).

O código feito para aplicação da técnica descrita acima também funciona com imagens com múltiplas bandas de cor (RGB). Ao utilizar uma imagem colorida como entrada, o código automaticamente reconhece a presença de múltiplas bandas, e aplica a técnica separadamente em cada uma das bandas de cor, juntando-as novamente no final do processo. O resultado é semelhante ao resultado do caso monocromático: uma imagem muito

semelhante à original, mas apenas formado pelos valores extremos das bandas (Vermelho, Verde e Azul), e, nesse caso, também pelas combinações desses extremos (branco, preto, ciano, amarelo e magenta). Essas combinações se dão pois os valores R, G e B, que vão de 0 a 255 cada (nesse caso assumem apenas o valor 0 ou 255), podem se combinar, já que cada um está em uma banda diferente, ou seja, um pixel pode assumir o valor (0, 255, 255), que resulta na cor ciano, (0, 0, 0), que resulta em preto, etc. Os resultados, assim como no caso monocromático, são bem parecidos, logo, por uma questão de simplicidade, será mostrado apenas o resultado da aplicação da abordagem de *Burkes* com varredura alternada.

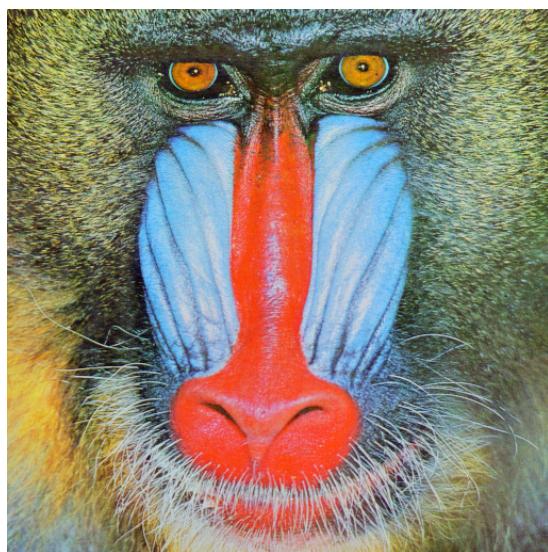


Figura 26: baboon\_colorida.png

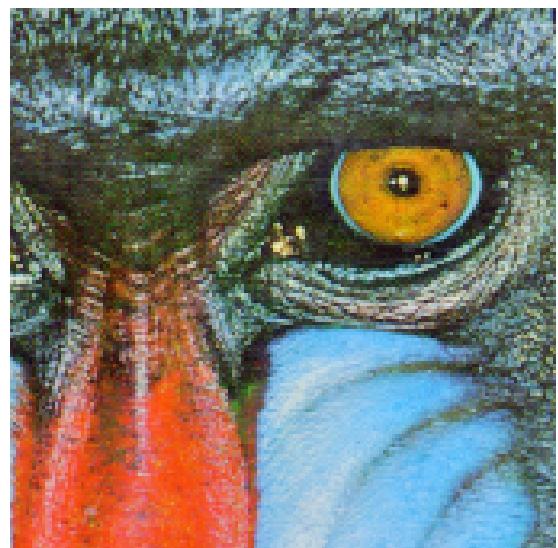


Figura 27: Zoom em baboon\_colorida.png

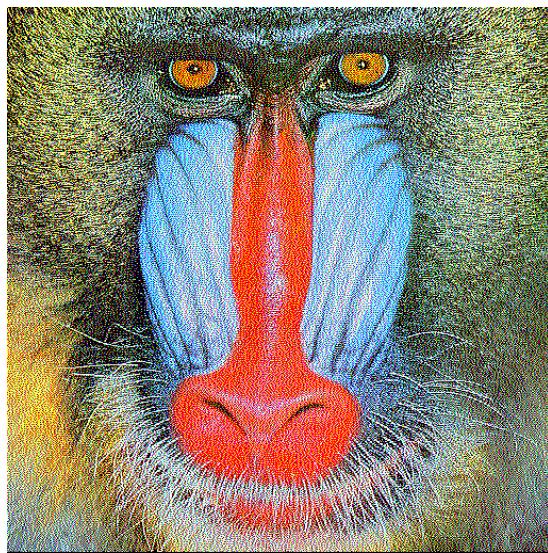


Figura 28: Resultado da aplicação de Burkes

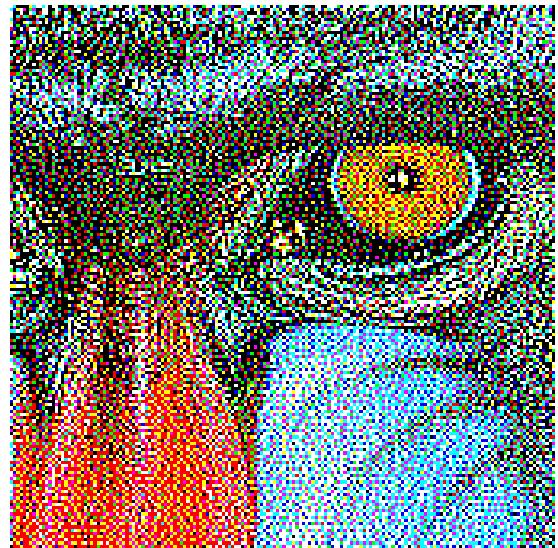


Figura 29: Zoom no resultado

## 3.2 Transformações no Domínio da Frequência

### 3.2.1 Aplicação de Filtros no Domínio da Frequência

O objetivo da segunda tarefa proposta foi aplicar a transformada rápida de Fourier em imagens, explorando o processamento dessas imagens no domínio de frequência. Primeiramente, para explorar essas transformações no domínio da frequência, alguns filtros (passa-baixa, passa-alta, passa-faixa e rejeita-faixa) foram aplicados na imagem *baboon\_monocromatica.png*.

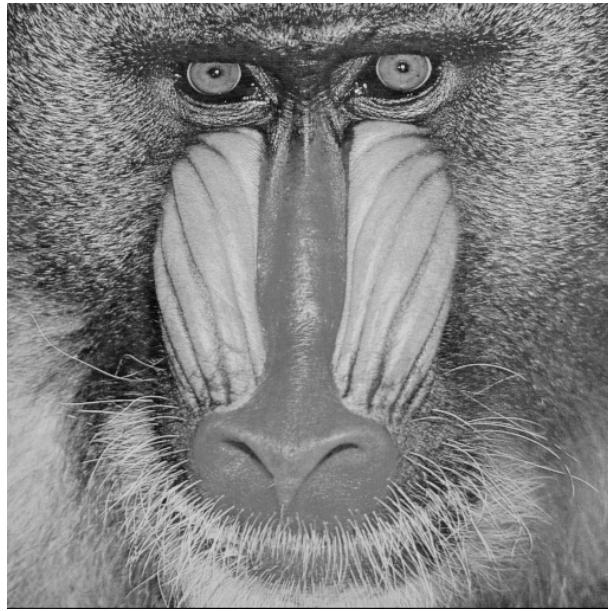


Figura 30: baboon\_monocromatica.png

Os filtros podem ser aplicados (e foram, no trabalho passado) utilizando a operação de convolução do filtro com a imagem, porém, desta vez, foi utilizada a seguinte propriedade da convolução:

$$f(x) * g(x) \xleftrightarrow{\mathcal{F}} F(\omega) \cdot G(\omega)$$

Ou seja, a convolução de duas funções  $f(x)$  e  $g(x)$  no domínio do tempo (ou espaço) é equivalente à multiplicação de suas transformadas de Fourier  $F(\omega)$  e  $G(\omega)$  no domínio da frequência. A recíproca também é verdadeira.

Para realizar isso, foi utilizada a função *Fast Fourier Transform (FFT)* para aplicar a transformada de Fourier na imagem. Após isso, o espectro de frequência foi centralizado, isso faz com que a frequência zero fique no centro, ou seja, quanto menor a frequência, mais próxima do centro (Figura 31). Isso é feito para simplificar a criação e aplicação dos filtros, que passam a ser círculos ou "rosquinhas" / anéis. A aplicação da transformada e centralização do espectro são facilmente realizadas usando a função FFT da seguinte forma:

```
fourier = np.fft.fft2(img)
fourier_centro = np.fft.fftshift(fourier)
```

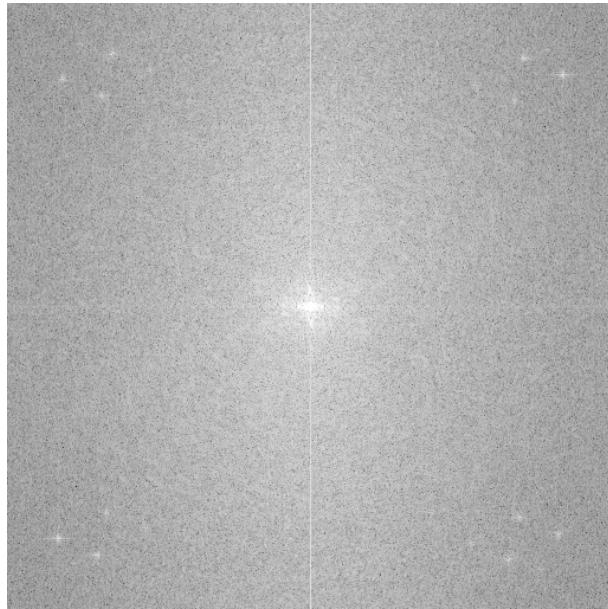


Figura 31: baboon\_monocromatica.png no domínio da frequência centralizado (espectro centralizado)

Após a centralização, filtros passa-baixa, passa-alta, passa-faixa e rejeita-faixa foram criados. Para criá-los foi criada, para cada filtro, uma figura nas dimensões da imagem formada por 0s e 1s, onde os valores 0 bloqueiam/zeram as frequências e os valores 1 as deixam passar (pois o respectivo pixel no espectro é multiplicado por 1, portanto, continua o mesmo). Devido à centralização e ao fato de as aplicações serem realizadas com uma simples multiplicação (pois estamos no domínio de frequência), naturalmente, filtros passa-baixa podem ser representados pelo formato de uma imagem com um círculo central formado por 1s e o restante da imagem é formado por 0s, dessa forma, ao multiplicarmos o filtro pela imagem, apenas os valores mais centrais (menores frequências) serão mantidos. Os filtros passa-alta são o oposto, um círculo formado por 0s com o restante sendo 1s. Já os filtros passa-faixa possuem um formato de anel onde os pixels do anel são 1s, e, por fim, os filtros rejeita-faixa são o oposto dos passa-faixa. As imagens dos filtros aplicados ao espectro podem ser vistas nas Figuras 32, 33, 52 e 35.

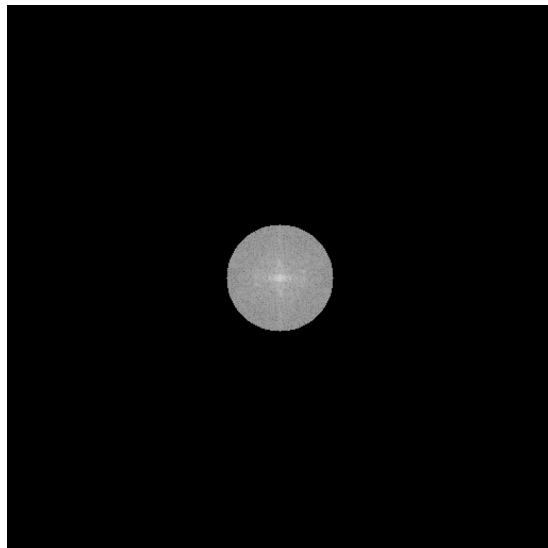


Figura 32: núcleo passa-baixa

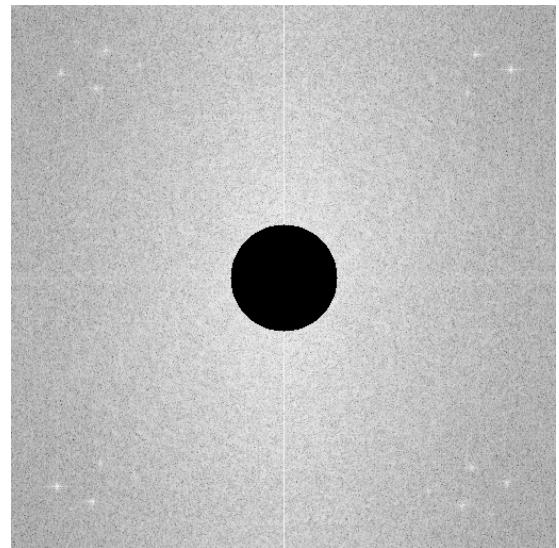


Figura 33: núcleo passa-alta

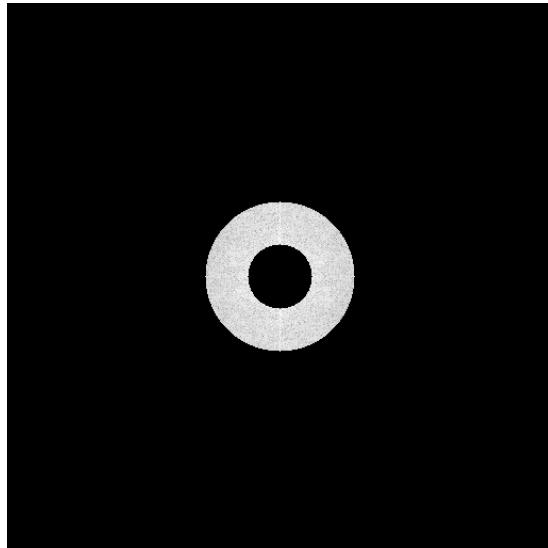


Figura 34: núcleo passa-faixa

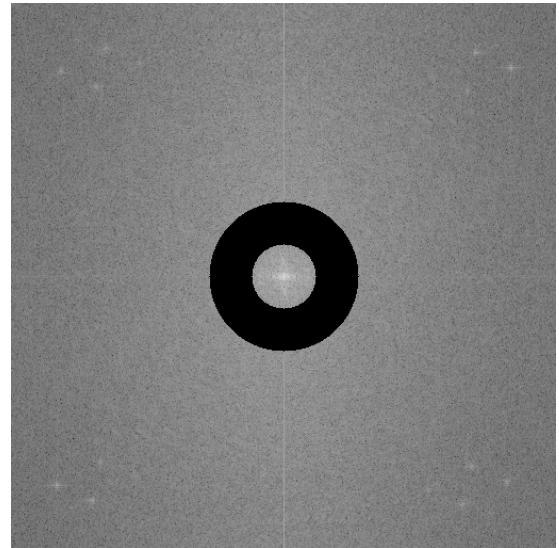


Figura 35: núcleo rejeita-faixa

Ao observar os filtros acima, fica mais intuitivo perceber que os raios dos círculos/anéis representam, na verdade, os limiares de frequência a serem cortados. Mais adiante neste relatório realizaremos testes quanto à escolha desses valores. Nesta ocasião, os valores selecionados para os raios foram: 50 para os raios do passa-baixa e passa-alta e 30 e 70 para os raios dos anéis do passa-baixa e passa-alta. Na prática, isso é feito da seguinte forma:

```
passa_baixa_mascara = passa_baixa(img, 50)
passa_alta_mascara = passa_alta(img, 50)
passa_faixa_mascara = passa_faixa(img, 70, 30)
rejeita_faixa_mascara = rejeita_faixa(img, 70, 30)
```

Quanto à aplicação dos filtros nas imagens, fica implícito que, após realizar a transformada, centralização e multiplicação, é necessário reverter a centralização e a transformada

para obter novamente uma imagem no domínio espacial, ou seja, uma imagem que pode ser vista e compreendida. Para isso, basta utilizar a FFT novamente, porém com métodos diferentes:

```
def aplicar_filtro(fourier, mascara):
    fourier_filtrada = fourier * mascara
    fourier_s_centro = np.fft.ifftshift(fourier_filtrada)
    img_aux = np.abs(np.fft.ifft2(fourier_s_centro))
    return img_aux
```

O trecho de código mostrado acima mostra como é feita a implementação do processo no código. Nesse caso, a função de aplicação do filtro, além de realizar a aplicação multiplicando a imagem pelo filtro (máscara), também realiza a operação inversa da centralização com `np.fft.ifftshift()` e a inversa da transformada com `np.abs(np.fft.ifft2())` (a função `np.abs()` é utilizada para tratar os valores complexos presentes após a aplicação da inversa de Fourier).

Após aplicar esse processo inverso em cada uma das imagens com os filtros aplicados mostradas acima, podemos ver quais resultados cada filtro causou na imagem original (Figuras 32, 33, 52 e 35).



Figura 36: baboon\_monocromatica.png  
após aplicação do passa-baixa

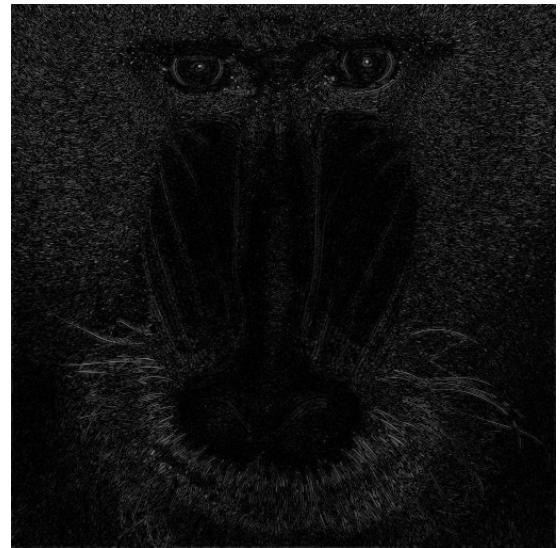


Figura 37: baboon\_monocromatica.png  
após aplicação do passa-alta

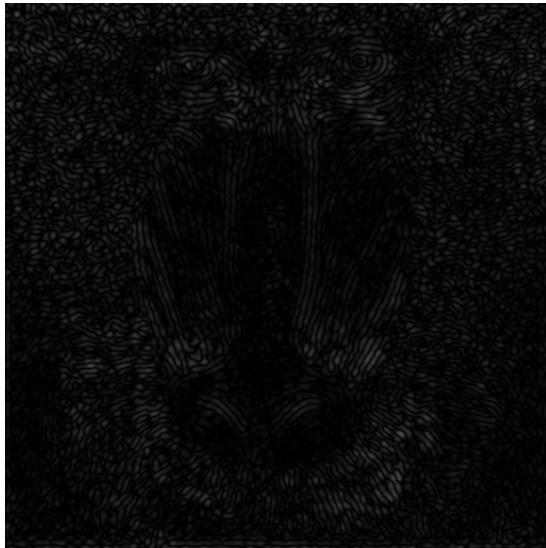


Figura 38: baboon\_monocromatica.png  
após aplicação do passa-faixa

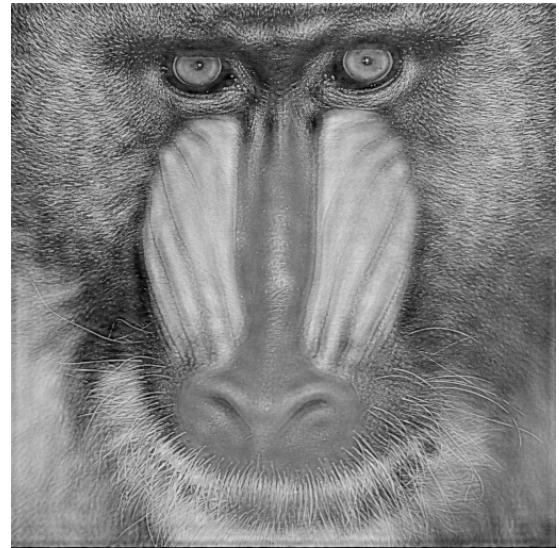


Figura 39: baboon\_monocromatica.png  
após aplicação do rejeita-faixa

Como é possível ver nos resultados mostrados acima, a imagem resultante da aplicação do filtro passa-baixa se tornou mais borrada, com os detalhes (pelos, texturas, etc) suavizados pois o filtro permite apenas a passagem das frequências mais baixas, que formam as componentes mais suaves da imagem. Já a imagem resultante da aplicação do filtro passa-alta ficou preta quase inteiramente (pois as frequências baixas, que representam as cores mais claras, são bloqueadas), mas realçou as bordas e texturas da imagem original, isso acontece pois o filtro permite apenas a passagem das frequências mais altas, que formam as bordas e detalhes mais finos da imagem. A imagem resultante da aplicação do filtro passa-faixa resultou em um meio termo entre os resultados anteriores, já que o filtro descarta as frequências mais baixas e mais altas, tanto as regiões mais suaves quanto os detalhes e bordas mais finas da imagem são descartados. Já o filtro rejeita-faixa apresenta o efeito oposto do filtro passado, mantendo apenas as áreas suaves das baixas frequências e os detalhes das altas frequências, resultando em uma imagem que se torna um misto entre desfoque e nitidez.

Embora os resultados observados anteriormente sejam satisfatórios e condizentes com o esperado, é inevitável se perguntar: que tipos de mudanças teríamos caso mudássemos os raios das máscaras? É por isso que a seguir, alguns testes serão feitos variando os raios. Por uma questão de espaço e praticidade, serão mostradas apenas as imagens finais, sem as imagens dos núcleos, já que a única alteração nessas imagens seria o tamanho do círculo/anel.

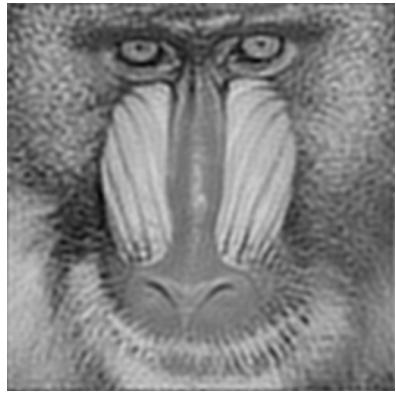


Figura 40: Resultado original do passa-baixa (raio 50)

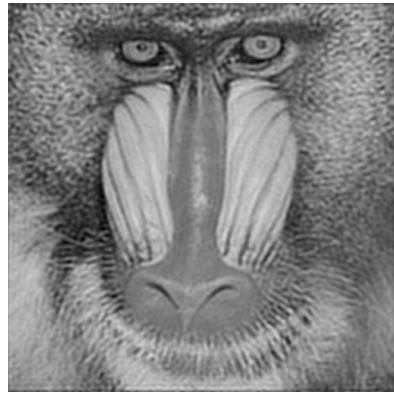


Figura 41: Resultado com passa-baixa de raio 70

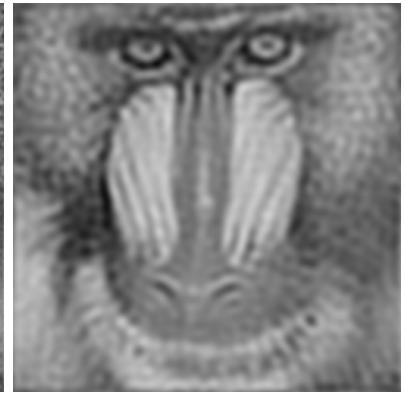


Figura 42: Resultado com passa-baixa de raio 30

Como é possível observar nas Figuras 40, 41 e 42, a diminuição do raio da máscara do filtro passa-baixa faz com que o efeito de desfoque fique mais intenso, isso acontece pois o filtro deixa passar ainda menos frequências, fazendo com que a imagem perca mais definição. Naturalmente, o aumento do raio do passa-baixa causa o efeito inverso.

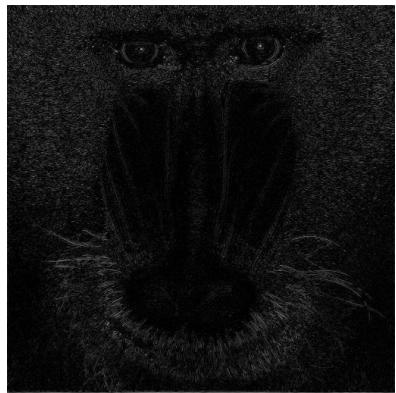


Figura 43: Resultado original do passa-alta (raio 50)

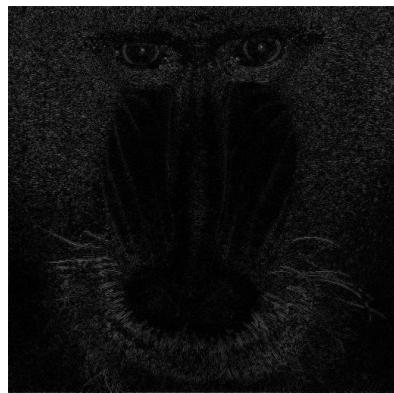


Figura 44: Resultado com passa-alta de raio 70

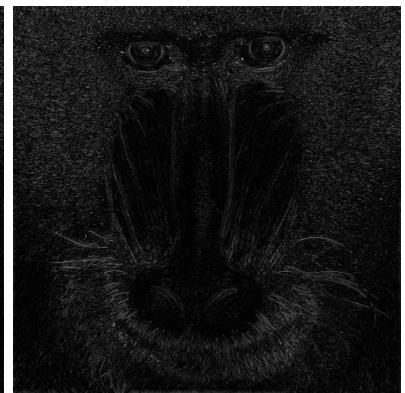


Figura 45: Resultado com passa-alta de raio 30

Como é possível observar nas Figuras 43, 44 e 45, a diminuição do raio da máscara do filtro passa-alta possui a lógica inversa do filtro passa-baixa: quando o raio é diminuído, o filtro permite que mais frequências passem, fazendo a imagem ter um pouco mais de definição, pegando mais detalhes da imagem original.

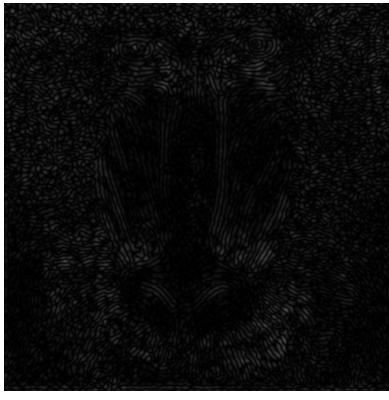


Figura 46: Resultado original do passa-faixa (raios 30 e 70)



Figura 47: Resultado com Figura 48: Resultado com passa-faixa de raios 20 e 80 passa-faixa de raios 40 e 60

Como é possível observar nas Figuras 46, 47 e 48, ao tornarmos o anel da máscara mais largo (raios 20 e 80), o filtro permite a passagem de mais frequências médias, fazendo a imagem apresentar mais detalhes, ao contrário de quando deixamos o anel mais estreito, fazendo com que a imagem se torne mais apagada pois menos frequências passam.

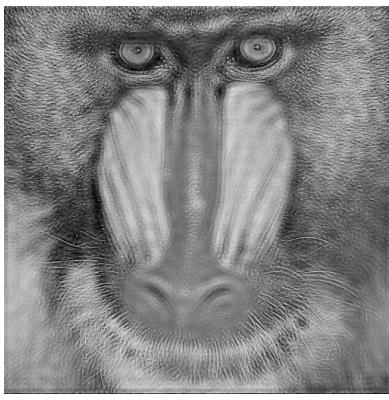


Figura 49: Resultado original do rejeita-faixa (raios 30 e 70)



Figura 50: Resultado com Figura 51: Resultado com rejeita-faixa de raios 20 e 80 rejeita-faixa de raios 40 e 60

Por fim, observando as Figuras 49, 50 e 51, é possível notar que os resultados seguem a lógica oposta do caso anterior, onde um anel mais largo bloqueará mais frequências, fazendo a imagem perder definição, enquanto o estreitamento do anel faz com que a imagem ganhe definição.

### 3.2.2 Compressão de Imagem no Domínio da Frequência

Além da aplicação dos filtros mostrada anteriormente, também é possível realizar outras transformações úteis no domínio da frequência. Nesta etapa da tarefa, será realizada uma compressão da imagem *baboon\_monocromatica.png* através da eliminação de frequências menos significativas, fazendo com que o tamanho de armazenamento necessário para guardar a imagem seja reduzido. Para realizar isso, o mesmo processo de aplicação das transformada e centralização foi aplicado na imagem, porém, em vez de aplicar filtros, algumas frequências foram retiradas do espectro. Para retirar frequências do espectro, é necessário

primeiro definir um limiar e só então zerar os valores abaixo do limiar selecionado. O processo de definição do limiar e retirada das frequências menores foi realizado da seguinte forma:

```
limiar = np.percentile(np.abs(fourier_centro), 90)
fourier_comprimida = np.where(np.abs(fourier_centro) < limiar, 0, fourier_centro)
img_comprimida = np.abs(np.fft.ifft2(np.fft.ifftshift(fourier_comprimida)))
```

Acima, a função `np.percentile()` é utilizada para definir o limiar usando a imagem (já no domínio da frequência) e o valor 90 como parâmetros. Essa função, na prática, retorna um valor de frequência em `fourier_centro` tal que 90% dos valores da imagem estão abaixo dele. Em seguida, os valores de `fourier_centro` menores que o limiar são zerados, e são aplicadas a inversa do deslocamento e a inversa da transformada, para retornar a imagem ao domínio espacial. Repare que é necessário zerar os valores **menores** que o limiar pois essas são as componentes fracas da imagem, isto é, as frequências que carregam informações mais sutis e cuja eliminação não causará alterações drásticas na imagem. O resultado dessa compressão pode ser visto na Figura 53 abaixo:

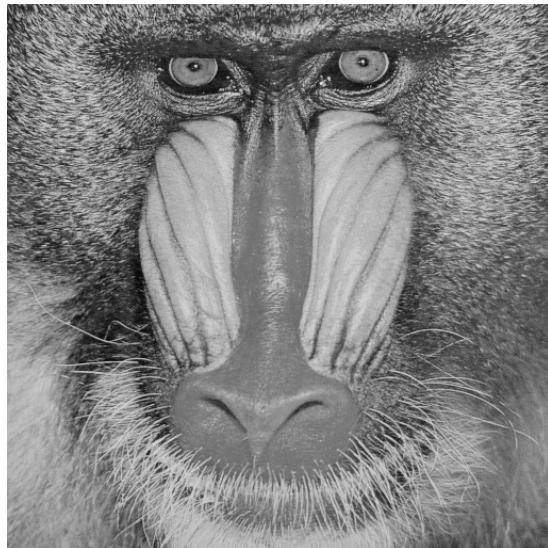


Figura 52: baboon\_monocromatica.png  
original

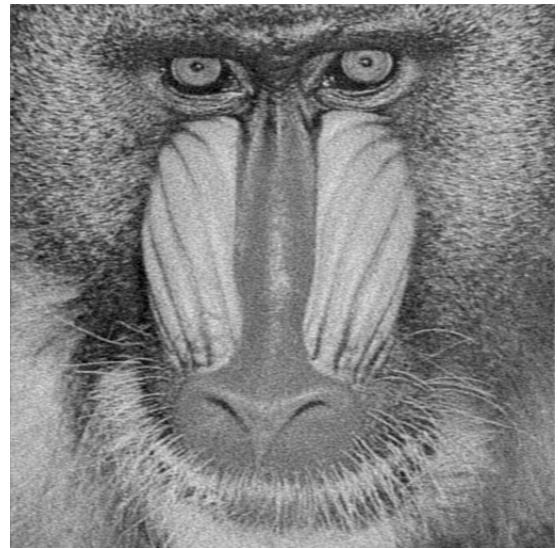


Figura 53: baboon\_monocromatica.png  
comprimida



Figura 54: Zoom em baboon\_monocromatica.png original



Figura 55: Zoom em baboon\_monocromatica.png comprimida

Como é possível observar nas imagens do resultado, fica evidente que, com a compressão, embora o resultado seja muito parecido com a imagem original, detalhes e nitidez da imagem são perdidos (principalmente quando analisada de perto). Analisando o tamanho dos arquivos dos resultados, temos que a imagem original possui 201 KB, enquanto a comprimida possui 174 KB, portanto, obtivemos uma redução de aproximadamente 13,43% no tamanho necessário para armazenar a imagem. Para uma análise melhor das imagens, também é interessante observar os histogramas das imagens. Abaixo, podem ser vistos os histogramas que relacionam frequência com intensidade dos pixels da imagem

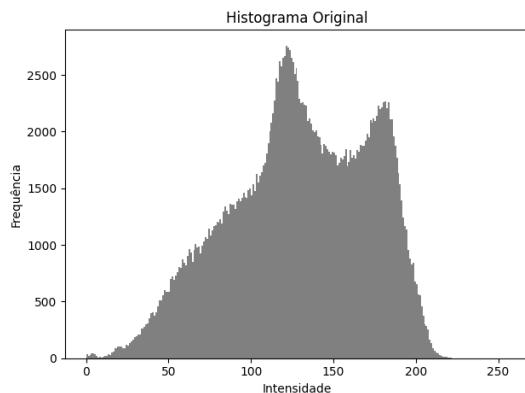


Figura 56: Histograma da imagem original

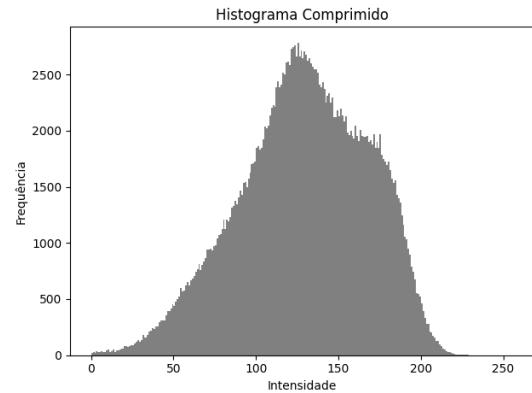


Figura 57: Histograma da imagem comprimida

Como é possível observar no histograma da imagem original, a distribuição das intensidades apresenta dois picos principais, significando que existem duas faixas de tons predominantes. Já no histograma da imagem comprimida, a forma geral do histograma foi preservada, mas agora, além de existir um único pico, as variações de frequência ficaram mais suaves.

Em um primeiro momento, é natural notar que valor do limiar escolhido parece particularmente alto, porém, ao analisar resultados com limiares menores, podemos notar que

as diferenças na compressão para limiares menores não são tão interessantes, por exemplo, observemos os resultados para uma compressão com limiar 70 na função `np.percentile()`.



Figura 58: Zoom em compressão com limiar 70

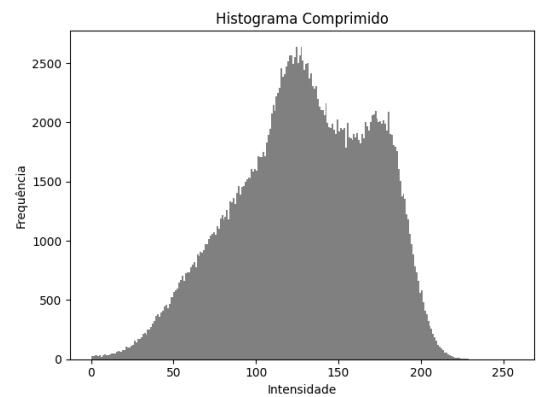


Figura 59: Histograma da imagem comprimida (limiar 70)

Observando os resultados acima, é possível notar que a imagem perdeu um pouco de definição, mas seu histograma ainda apresenta formato parecido com o histograma original, tendo dois picos definidos. Porém, o resultado que realmente chama a atenção (negativamente) é o ganho de espaço/tamanho obtido pela compressão: enquanto a imagem original possui 201 KB, o resultado da compressão com limiar 70 possui 193 KB, uma redução aproximada de apenas 4%.

Também é possível realizar os mesmos testes com valores acima de 90. Nesse caso a seguir, o valor escolhido foi 98.



Figura 60: Zoom em compressão com limiar 98

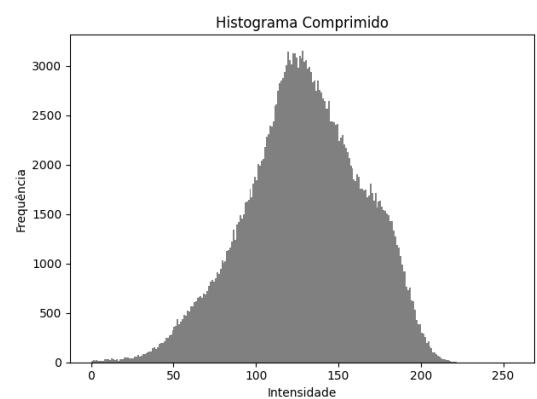


Figura 61: Histograma da imagem comprimida (limiar 98)

No caso acima, é possível notar uma queda radical na qualidade/definição da imagem

e uma suavização ainda maior no histograma, assim como esperado. A redução obtida utilizando o limiar 98 é de aproximadamente 27%, com a imagem resultante tendo 146 KB. Tais resultados corroboram as expectativas previamente estipuladas: quanto mais frequências eliminamos da imagem, mais leve ela se torna, ao mesmo passo que mais qualidade ela perde.