Projeto 2 PDI - 2021.2

O grupo é formado por:

- Joan Vitor
- Matheus Morais
- Renilton Ribeiro
- Yooh Brito

Introdução

O primeiro passo, na maioria dos sistemas de compressão de imagens e vídeo, é identificar a presença de redundância espacial (semelhança entre um pixel e os pixels em sua vizinhança) em cada imagem, campo ou frame do vídeo. Isto pode ser feito aplicando-se a Transformada Discreta de Cosenos (DCT) ao longo da imagem. A DCT é um processo sem perda (lossless) e reversível.

Neste trabalho, a DCT (1D e 2D) (direta e inversa) deve ser desenvolvida utilizando as equações estudadas em sala de aula, sem o uso de bibliotecas prontas para esse fim.

1 - Dada uma imagem I em níveis de cinza, de dimensões RxC, desenvolva um programa para:

1.1 Exibir o módulo normalizado (expansão de histograma para [0, 255]) da DCT de I, sem o nível DC, e o valor (numérico) do nível DC

1.2. Encontrar e exibir uma aproximação de I obtida preservando o coeficiente DC e os n coeficientes AC mais importantes de I, e zerando os demais. O parâmetro n é um inteiro no intervalo [0, RxC-1].

2. Desenvolva um programa para filtrar uma imagem I de dimensões RxC, utilizando um filtro de Butterworth passa-baixas no domínio da frequência. A função de transferência do filtro é dada por:

$$H(d(k,l)) = \frac{1}{\sqrt{1 + \left(\frac{d(k,l)}{fc}\right)^{2n}}}$$

em que d(k,l) é a distância euclidiana do coeficiente (k,l) até a origem, fc é a distância de corte até a origem e n >= 1 é a ordem do filtro.

3. Desenvolva um programa para filtrar um sinal s, em formato .wav, com N amostras, utilizando um filtro de Butterworth passa-baixas no domínio da frequência. A função de transferência do filtro é dada por:

$$H(f) = \frac{1}{\sqrt{1 + \left(\frac{f}{fc}\right)^{2n}}}$$

em que f é a frequência em Hz, fc é a frequência de corte em Hz e n > 1 é a ordem do filtro.

Fundamentação Teórica

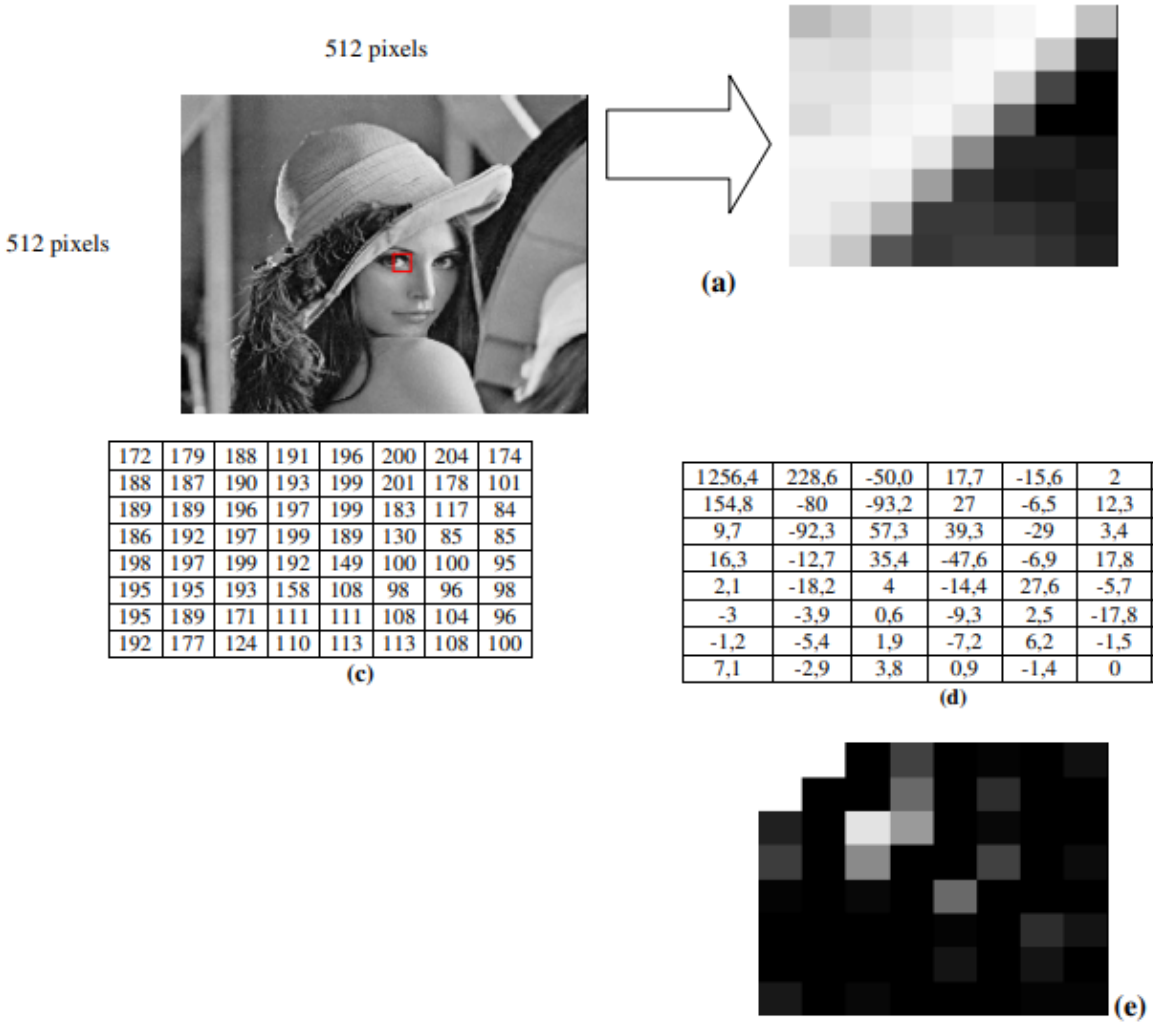
No presente trabalho, foram utilizados uma imagem 2D, nas dimensões RxC, e um audio em formato .wav, estes que ao decorrer do código são transformados em arrays e matrizes de números com pontos flutuantes. Essa transformação é necessária para que os filtros possam ser aplicados, estes que são descritos abaixo:

1 . Transformada Discreta de Cosseno ou DCT 1D

Uma transformada discreta de cosseno (DCT) expressa uma sequência finita de pontos de dados em termos de uma soma de funções de cosseno que oscilam em diferentes frequências. O DCT, proposto pela primeira vez por *Nasir Ahmed* em 1972, é uma técnica de transformação amplamente usada no processamento de sinais e compressão de dados. É utilizado na maioria das mídias digitais, incluindo imagens digitais, vídeos digitais, áudios digitais, televisão digital, rádio digital e codificação de voz. Os DCTs também são importantes para várias outras aplicações em ciência e engenharia, como processamento de sinal digital, dispositivos de telecomunicações, redução do uso da largura de banda da rede e métodos espectrais para a solução numérica de equações diferenciais parciais.

Matematicamente, corresponde a uma simples multiplicação por uma matriz, que tem a propriedade de converter dados de amplitude espacial (os valores dos pixels) em coeficientes representando freqüências espaciais. Para cada dimensão de blocos a ser usada (a mais usada é de 8 x 8 pixels), existe uma matriz de DCT fixa. Realizar a transformação implica simplesmente em em recolher os 64 pixels deste bloco da imagem, fazer o cálculo da DCT para estes valores, obtendo-se novos 64 valores, que são chamados coeficientes da DCT. Este processo é repetido para todos os blocos da imagem.

Supõe-se que se calcula a DCT em blocos de 8 x 8 pixels. No exemplo, o bloco em questão localiza-se na região do olho da Lena e está identificado pelo quadrado superposto à figura. Considere o bloco de 8 x 8 pixels mostrado graficamente na figura 1b, cujos valores dos pixels são mostrados na figura 1c. Observe que o primeiro pixel deste bloco tem o valor 172, que corresponde a um tom de cinza. É aplicada a DCT aos valores mostrados na figura 1c, obtendo-se assim os coeficientes da DCT, mostrados numericamente na figura 1d. A figura 1e é uma visualização gráfica dos coeficientes da DCT deste bloco. Observe que os coeficientes possuem valores positivos e negativos. O primeiro coeficiente (canto superior esquerdo) 2 possui o maior valor, enquanto os últimos coeficientes (próximos ao canto inferior direito) possuem valores pequenos. Estes valores pequenos normalmente são desprezados (o que corresponde a substituí-los por zero), sem que a imagem sofra grandes deformações. A possibilidade de desprezar os coeficientes menos significativos sem muita perda da qualidade da imagem é justamente a razão principal de se usar a DCT na codificação de imagens (e vídeo), pois permite economia de bits.



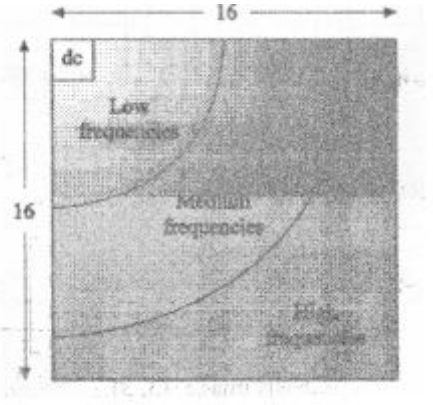
172	179	188	191	196	200	204	174
188	187	190	193	199	201	178	101
189	189	196	197	199	183	117	84
186	192	197	199	189	130	85	85
198	197	199	192	149	100	100	95
195	195	193	158	108	98	96	98
195	189	171	111	111	108	104	96
192	177	124	110	113	113	108	100

1256,4	228,6	-50,0	17,7	-15,6	2	-2,7	5,8
154,8	-80	-93,2	27	-6,5	12,3	2	0,7
9,7	-92,3	57,3	39,3	-29	3,4	6,3	1,5
16,3	-12,7	35,4	-47,6	-6,9	17,8	-2,1	4,4
2,1	-18,2	4	-14,4	27,6	-5,7	-12,9	-1,4
-3	-3,9	0,6	-9,3	2,5	-17,8	12,3	6,1
-1,2	-5,4	1,9	-7,2	6,2	-1,5	6,2	-11,8
7,1	-2,9	3,8	0,9	-1,4	0	2	2,9

Figura 1: **(a)** Imagem de 512x512 pixels; **(b)** Visualização do bloco 8x8 com os pixels da imagem representados em tons de cinza; **(c)** Valores dos pixels correspondentes a região com um quadrado vermelho na imagem (a); **(d)** Valores dos coeficientes DCT do bloco; **(e)** Visualização da DCT do bloco.

Verifica-se que o primeiro valor da matriz DCT é mais importante, ele é chamado de valor médio ou valor DC, os outros coeficientes são chamados de valores AC. A propriedade importante da DCT é que ela transforma os pixels de um domínio onde “todos são iguais”, para um novo domínio, onde há hierarquia. O primeiro coeficiente da DCT (o DC) é mais importante que o 64º (coeficiente AC de mais alta freqüência). A 512 pixels

A figura 2 mostra a distribuição de freqüência em uma DCT de duas dimensões de 16 x 16 pixel



A fórmula da DCT direta, que transforma um vetor x[n] em coeficientes X[k], com k variando de 0 a N-1 é:

$$X[k] = \alpha[k] \sum_{n=0}^{N-1} x[n] \cos\left(\frac{\pi (2n+1) k}{2N}\right)$$

A fórmula da DCT inversa, que obtém x[n], com n variando de 0 a N-1, a partir dos coeficientes DCT X[k] é:

$$x[n] = \sum_{k=0}^{N-1} \alpha[k] \;\; X[k] \;\; \cos\left(\frac{\pi (2n+1) k}{2N}\right)$$

Onde em ambos os casos:

$$\alpha[k] = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } \; k = 0 \\ \sqrt{\frac{2}{N}} & \text{for } \; k = 1, 2, \ldots, N-1 \end{cases}$$

Para imagens, é utilizado a DCT bindimensional ou DCT-2D, está pode ser representada pela seguinte fórmula:

$$F(u,v) = \frac{C_u}{2}\frac{C_v}{2}\sum_{y=0}^{7}\sum_{x=0}^{7} f(x,y)\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right]$$

A sua inversa, é demonstrada pela seguinte fórmula:

$$f(x,y) = \sum_{u=0}^{7}\sum_{v=0}^{7} F(u,v)\frac{C_u}{2}\frac{C_v}{2}\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right]$$

Onde em ambos os casos:

$$C_u = \begin{cases} \frac{1}{\sqrt{2}} & \text{if u} = 0, \\ 1 & \text{if u} > 0 \end{cases}; C_v = \begin{cases} \frac{1}{\sqrt{2}} & \text{if v} = 0, \\ 1 & \text{if v} > 0 \end{cases}$$

2. Expansão de Histograma

Em uma imagem, o seu histograma representa a frequencia de distribuição de níveis de cinza na imagem, é representado por um diagrama a qual o seu eixo x representa o nível de cinza específico e o eixo y representa o número de ocorrências de cada nível de cinza. Cada pixel de uma imagem tem uma cor que foi produzida por uma combinação de cores primárias (vermelho, verde e azul, ou RGB). Cada uma dessas cores pode ter um brilho que varia de 0 a 255 em uma imagem digital com profundidade de bits de 8-bits. Um histograma RGB é produzido quando o computador varre a imagem em cada um desses valores de brilho RGB e conta quantos pixels há em cada nível de 0 a 255. Outros tipos de histogramas existem, mas todos têm mais ou menos a mesma cara do exemplo abaixo:

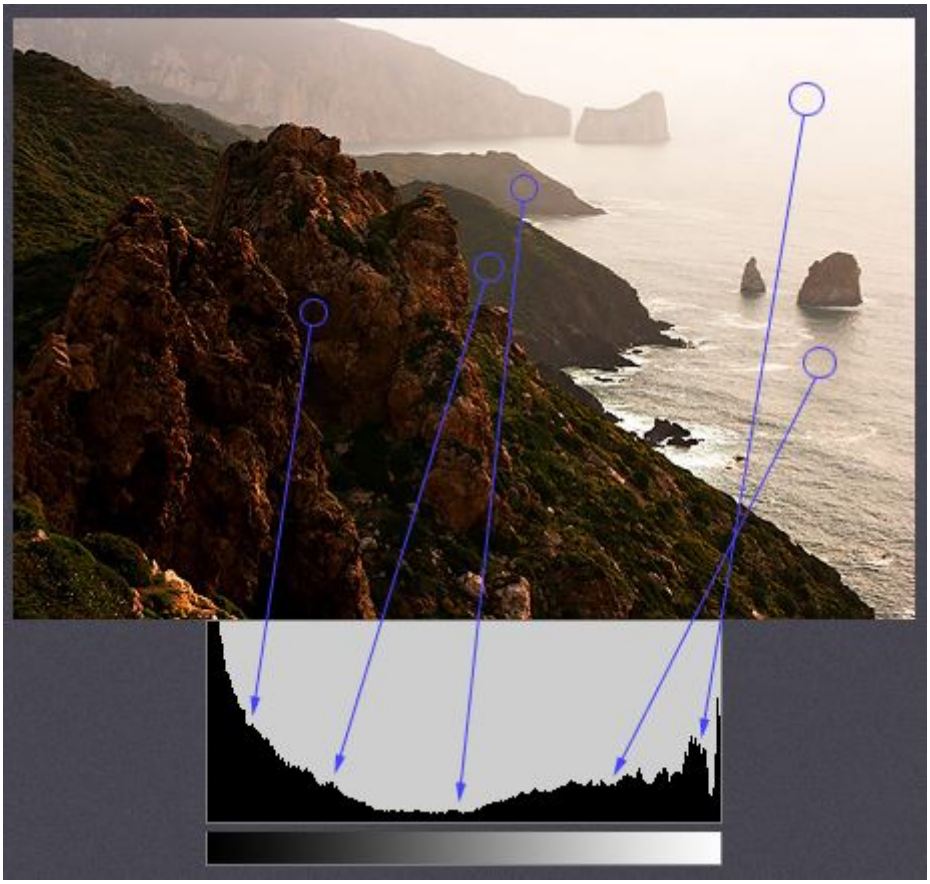


Figura 2: Representação de Histograma de uma Imagem

A Expansão de um Histograma, ou Normalização, é um técnica de normalização, tal técnica realiza uma redistribuição dos níveis de cinza de uma imagem, podendo gerar imagens mais ricas. Sua fórmula pode ser definida como:

$$s = T(r) = round\left(\frac{r - r_{\min}}{r_{\max} - r_{\min}} (L - 1) \right)$$

A seguir, a aplicação direta da expansão de um histograma:

Figura 3: Imagens não normalizada

Figura 4: Imagens normalizada

3. Filtro Butterworth

O filtro Butterworth é um tipo de projeto de filtros eletrônicos. Ele é desenvolvido de modo a ter uma resposta em frequência o mais plana o quanto for matematicamente possível na banda passante.Estes foram descritos primeiramente pelo engenheiro britânico Stephen Butterworth na sua publicação "On the Theory of Filter Amplifiers", Wireless Engineer (também chamada de Experimental Wireless and the Radio Engineer), vol. 7, 1930, pp. 536-541.

A resposta em frequência de um filtro Butterworth é muito plana (não possui ondulações) na banda passante, e se aproxima do zero na banda rejeitada. Quando visto em um gráfico

logarítmico, esta resposta desce linearmente até o infinito negativo. Para um filtro de primeira ordem, a resposta varia em −6 dB por oitava (−20 dB por década). (Todos os filtros de primeira ordem, independentemente de seus nomes, são idênticos e possuem a mesma resposta em frequência.) Para um filtro Butterworth de segunda ordem, a resposta em frequência varia em −12 dB por oitava, em um filtro de terceira ordem a variação é de −18 dB, e assim por diante. Os filtros Butterworth possuem uma queda na sua magnitude como uma função linear com ω.

Exemplo de um filtro passa-baixas Butterworth de segunda ordem O Butterworth é o único filtro que mantém o mesmo formato para ordens mais elevadas (porém com uma inclinação mais íngreme na banda atenuada) enquanto outras variedades de filtros (Bessel, Chebyshev, elíptico) possuem formatos diferentes para ordens mais elevadas.

Comparado com um filtro Chebyshev do Tipo I/Tipo II ou com um filtro elíptico, o filtro Butterworth possui uma queda relativamente mais lenta, e portanto irá requerer uma ordem maior para implementar um especificação de banda rejeitada particular. Entretanto, o filtro Butterworth apresentará uma resposta em fase mais linear na banda passante do que os filtros Chebyshev do Tipo I/Tipo II ou elípticos.

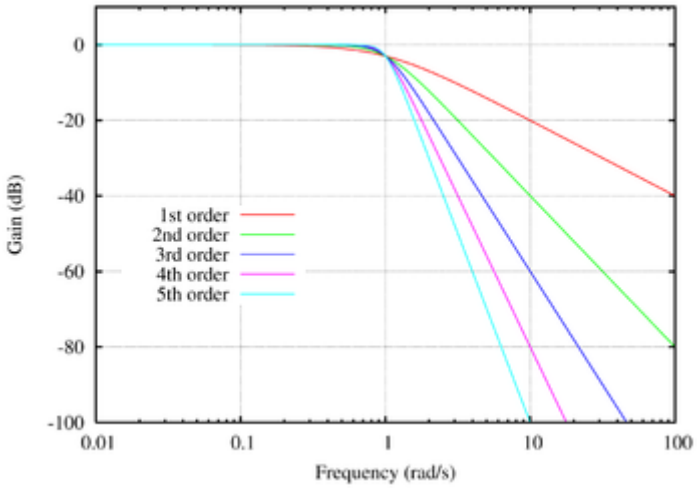


Figura 5: Filtros passa-baixas Butterworth de ordens 1 a 5

3.1 Filtro Passa-Baixas

Os filtros passivos passa-baixa são circuitos que permitem a passagem de sinais de baixa frequência e reduzem a intensidade de sinais de alta frequência, ou seja, a partir de uma frequência de referência ele permite que frequências mais baixas que ela passem livremente e frequências mais altas sejam atenuadas, por isso o seu ganho será de no máximo 1.

Um Filtro Passa-Baixas para uma imagem I de dimensões RxC pode ser definido por:

$$H(d(k,l)) = \frac{1}{\sqrt{1+\left(\frac{d(k,l)}{f_c}\right)^{2n}}}$$

Onde:

D(k,L) -> Distância euclidiana do coeficiente (k,l) até a origem

Fc -> Distância de corte até a origem

n >= 1 -> A ordem do filtro

Um Filtro Passa-Baixas para filtrar um sinal s, em formato .wav, com N amostras pode ser definido por:

$$H(f) = \frac{1}{\sqrt{1+\left(\frac{f}{f_c}\right)^{2n}}}$$

Onde:

f -> frequência em Hz

fc -> frequência de corte em Hz

n > 1 é a ordem do filtro.

Materiais e Métodos

1. Imports e Arquivos utilizados

O presente trabalho, foi integralmente desenvolvido utilizando a linguagem de programação Python, também foram utilizados:

- Uma imagem bidimensional do tipo png(Portable Network Graphics)

Figura 6: Imagem da Lena

- Um arquivo de Audio do tipo wav.

Além disso, foram utilizadas, como suporte, bibliotecas nativas do ambiente Python para auxiliar no desenvolvimento da aplicação, como, por exemplo:

- Matplotlib, uma biblioteca de software para criação de gráficos e visualizações de dados em geral.
- SciPy, um pacote principal de rotinas científicas em Python, que se destina a operar de forma eficiente em matrizes.
- Numpy, biblioteca que suporta o processamento de grandes, multi-dimensionais arranjos e matrizes, juntamente com uma grande coleção de funções matemáticas de alto nível para operar sobre estas matrizes.
- Math, biblioteca responsável por realizar operações matematicas, como calculo de raizes e cossenos.
- Time, responsável por cronomentrar o tempo de execução da aplicação.
- PIL, adiciona suporte à abertura e gravação de muitos formatos de imagem diferentes.
- Colorsys, define conversões bidirecionais de valores de cores entre cores expressas no espaço de cores RGB (Red Green Blue).
- OpenCV, biblioteca multiplataforma para o desenvolvimento de aplicativos na área de Visão computacional.
- Imageio, biblioteca que auxilia na leitura e escrita de dados.

A partir da fundamentação teórica adquirida de pesquisas feitas de artigos e dissertações, além dos slides fornecidos na disciplina de Introdução ao Processamento Digital de Imagens foi possível a resolução dos problemas propostos na Introdução. Logo, segue a caracterização da solução:

2. Transformada Discreta de Cosseno ou DCT 1D

Conforme proposto antes, a Transformada Discreta de Cosseno em ambiente unidimensional pode ser demonstrada através da seguinte formula:

$$X[k] = \alpha[k] \sum_{n=0}^{N-1} x[n] \cos\left(\frac{\pi (2n+1) k}{2N}\right)$$

Diante disso, foi necessário criar uma função capaz de realizar esse calculo. Para isto, a função, denominada DCT_1D, recebe único parametro, um array chamado data, primeiramente a DCT_1D inicializa uma várivel N que recebe como valor o tamanho do array data, onde N representa o número de elementos do array, um array X de tamanho igual a N e a constante ak que recebe o resultado da raiz quadrada de 2 divido por N (2.0/N). Seguindo o código um laço For é criado, onde o número de repetições k percorrerá o intervalo de (0, N-1), 0 á N-1, dentro do laço, a variavel ck é inicializada, se acaso k for igual a 0, ck é igual a raiz de 1.0 dividido por 2.0, se acaso k for diferente de 0, ck é igual a 1, também é criada uma variavel s, inicializada com 0, está variavel representa o valor do somatorio dos cossenos da DCT. Esse somátorio é realizado atraves de um novo For, em que n pertence ao intervalo (0, N-1), que inicializa as variaveis a1 e a2, são os numerados do cosseno, e nn, denominador do cosseno, respectivamente:

- a1 = 2*pi*k*n
- a2 = k*pi
- nn = 2.0*N

Ao fim do for, é atribuido ao somatorio s o resultado da multiplicação de X[n] e o cosseno de ((a1/nn) + (a2/nn)). Após finalizado o segundo for, é atribuido ao array X, na posição k o resultado da multiplicação de ak, ck e s:

- X[n] = ak*ck*s

A DCT_1D finaliza retornando o array X.

```python
@jit(nopython=True)
def DCT_1D(data):
    N = len(data)
    X = np.zeros(N)
    aK = math.sqrt(2.0/N)
    for k in range(N):
        if (k == 0):
            ck = math.sqrt(1.0/2.0)
        else:
            ck = 1
        s = 0
        for n in range(N):
            a1 = 2.0 * math.pi * k * n
            a2 = k * math.pi
            nn = 2.0 * N
            s += data[n] * math.cos((a1/nn) + (a2/nn))
        X[k] = aK * ck * s
    return X
```

Figura 7: Função DCT_1D

2 . Inversa Transformada da Discreta de Cosseno ou DCT 1D

A Transformada Discreta de Cosseno Inversa, ou IDCT-1, é representada pela função IDCT_1D que recebe como unico parametro um array, denominado data. Primeiramente a DCT_1D inicializa uma várivel N que recebe como valor o tamanho do array data, em analogia á formula da Figura 2, N representa o número de elementos do array, em analogia á formula da Figura 2, N representa o número de elementos do array, e um array vazia X de tamanho igual a N. Seguindo o fluxo, uma variavel ak recebe o resultado da raiz quadrada de 2 divido por N, calculo que so foi possivel com o uso da biblioteca math do python, então é criado um For em que n percorrerá o intervalo (0, N-1), dentro do For:

- Inicializa a variavel s, somatorio, com o valor 0.
- Cria outro For em que k pertence ao intervalo (0,N-1):
    - Inicializa as variaveis que são os numerados do calculo do cosseno:
    - a1 = 2.0*pi*n
    - a2 = k*pi
    - nn = 2.0 * N
    - Se k = 0
        - ck = raiz de 1.0/2.0
    - Senão
        - ck = 1

- Soma a s a seguinte formula: ck*data[k]*cos[(a1/nn) + (a2/nn)]

```python
@jit(nopython=True)
def IDCT_1D(data):
    N = len(data)
    x = np.zeros(N)

    aK = math.sqrt(2.0/N)

    for n in range(N):
        s = 0
        for k in range(N):
            a1 = 2.0 * math.pi * k * n
            a2 = k * math.pi
            nn = 2.0 * N
            cK = math.sqrt(1.0/2.0) if k == 0 else 1

            s += cK * data[k] * math.cos((a1/nn) + (a2/nn))

        x[n] = aK * s

    return x
```

Figura 8: Função IDCT_1D

3. Transformada Discreta de Cosseno Bidimensional ou DCT-2D

Para aplicar o calculo da DCT-2D foi utilizado a função DCT_1D dentro de um looping do tipo For, passando como parametros o array contendo os dados da imagem da Lena. Primeiramente, aplica-se o DCT_1D nas linhas, sentido horizontal, finalizado o For, ocorrer uma transposição do resultado da DCT_1D na imagem, por fim, repete a aplicação da DCT_1D.

```python
# Cria matrizes vazias
imgDCT = np.zeros(lena256.shape)
img_DCT_horizontal = np.zeros(lena256.shape)
img_DCT_vertical = np.zeros(lena256.shape)

# DCT nas linhas - horizontal
for i, linha in enumerate(lena256):
    img_DCT_horizontal[i] = DCT_1D(linha)

# Transposição dos dados
img_input_vertical = img_DCT_horizontal.T

# DCT nas colunas - vertical
for i, linha in enumerate(img_input_vertical):
    img_DCT_vertical[i] = DCT_1D(linha)
```

Figura 9: Aplicação da DCT 2D

4. Transformada Discreta de Cosseno Inversa Bidimensional ou IDCT-2D

Para aplicar o IDCT-2D, foi criada uma função denominada idct2d, onde recebe como parametro a matriz da imagem com o DCT aplicado, dentro da função será aplicado a função IDCT_1D nas linhas da matriz, e após, será Transposta. O mesmo procedimento se repete para as colunas e por fim retorna uma lista da matriz.

```
# DCT nas linhas - horizontal
for i, linha in enumerate(imgDCT):
    img_IDCT_horizontal[i] = IDCT_1D(linha)

# Transposição dos dados
img_input_vertical = img_IDCT_horizontal.T

# DCT nas colunas - vertical
for i, linha in enumerate(img_input_vertical):
    img_IDCT_vertical[i] = IDCT_1D(linha)

img_IDCT_vertical = img_IDCT_vertical.T
```

Figura 8: Função idct2d

5. Transformada rápida de Fourier

Para a aplicação do Filtro Butterworth, antes é necessario, converter a imagem no dominio original, que no caso é a propria imagem , para a imagem no dominio da frequencia, para isto, necessita-se ser aplicada a imagem, a Transformada rápida de Fourier, ou FFT. A função que representa essa converção é chamada de aplica_filtro, a qual recebe como parametros, a imagem e o filtro(Butterworth). Primeiramente aplica_filtro, armazena linhas e colunas e cria uma matriz de zeros e depois percorre a matriz atribuindo o valor de n, n=1, nas posições das coordenadas i e j. A partir da função fftshift, da biblioteca Numpy, ele rearranja a transformada de Fourier movendo as frequências zero para o centro do array, e armazena na variavel primeira_shift_fft, depois é chamada a função fft2, onde ela retorna a FFT-2D, então a partir da função ifft2, retorna a inversa da transformada de Fourier 2D e aplica o filtro, inverte o shift e armazena o resultado de acordo com a matriz.

```python
def aplica_filtro(img, filtro):

    # Armazena linhas e colunas e cria uma matriz de zeros
    linhas, colunas = img.shape
    matriz = np.zeros((linhas, colunas))
    n = 1

    # Percorre a matriz atribuindo n na posição das coordenadas i e j
    for i in range(linhas):
        for j in range(colunas):
            matriz[i, j] =n
            n =-n
        n =-n

    # Rearranja a transformada de Fourier movendo as frequências zero para o centro do array
    # Como é uma matriz, troca o primeiro quadrante da (imagem * matriz) pelo terceiro e o segundo pelo quarto
    # Basicamente é como se processasse vários sinais 1-D e atribuisse nas linhas
    # E computa a transformada de fourier e inverte as frequências zero de cada coluna
    primeira_shift_fft = (fftshift(img * matriz))

    # Retorna a transformada de Fourier 2D
    primeira_fft2 = (fft2(primeira_shift_fft))

    # Retorna a inversa da transformada de Fourier 2D e aplica o filtro
    inversa_fft2 = (ifft2(primeira_fft2 * filtro))

    # Inverte o shift
    inversa_shift_fft = (ifftshift(inversa_fft2))

    # Armazena o resultado de acordo com a matriz
    resultado = (inversa_shift_fft * matriz)

    return np.real(resultado)
```

Figura 9: Função aplica_filtro

6. Filtro Butterworth

O Filtro Butterwoth passa-baixas é representado pela função filtro_butterworth, que recebe como parametro a imagem, a distancia do corte e a ordem do filtro. Primeiramente, a função salva a quantidade de colunas e linhas em duas variaveis, col e row, respectivamente, que serão utilizadas na criação de dois arrays, rowsIndex e colsIndex, estes que o tamanho é determinado pelo numero de linhas e colunas da imagem, ambas as matrizes armazena os indexadores, que serão utilizados posteriormente na função. Apos isso, a filtro_butterworth cria uma matriz, denominada D, de dimensões RxC, ou seja, R é igual ao valor de row e C é o valor de col. Para cada elemento de D, será aplicado a equação de Butterworth, e por fim, será aplicado o passa-baixas em D, retornando o resultado do paixa-baixas em D.

```python
def filtro_butterworth(img, D0, n, lowpass = True):

    # Armazena e indexa o tamanho de colunas e linhas
    row, col = img.shape
    rowsIndex=np.zeros((row, col))
    colsIndex=np.zeros((row, col))

    # Para a diagonal, ambos os indexadores recebem i
    if row == col :
        for i in range(row):
            for j in range(col):
                rowsIndex[i, j] = i
        colsIndex=rowsIndex.T
    # Para o resto recebem suas determinadas coordenadas i e j
    else:
        for i in range(row):
            for j in range(col):
                rowsIndex[i, j] = i
                colsIndex[i, j] = j

    # Cria uma matriz de 0
    D=np.zeros((row, col))

    # Aplica a equação em cada elemento de D
    for i in range(row):
        for j in range(col):
            D[i, j] = np.sqrt(pow((rowsIndex[i, j] - row//2), 2) + pow((colsIndex[i, j] - col//2), 2))

    # Aplicando a passa-baixa
    if lowpass:
        L = 1 /(1 + (D / D0) ** (2 * n))
        return L
```

Figura 10: Filtro Butterworth

7. Filtro Butterworth - Audio

O Filtro Butterworth antes de ser aplicado, realizamos a conversão para DCT, para levarmos o audio ao domínio da frequência, utilizando a função DCT_1D_audio, já realizando a aplicação na expressão do filtro, junto com a operação do passa-baixa. Após a conversão, chamamos então a função IDCT_1 para voltar o audio ao domínio espacial com o resultado da aplicação do filtro.

```python
def filtro_butterworth(audio, fc):

    # Obtém o tamanho do audio e define o i e o n
    tamanho_do_audio = audio.size
    n = 3
    i=0

    # Adquirimos o os fatores para realizar as operações
    fa_utilizado = 1 / (tamanho_do_audio - 1)
    f1 = fa_utilizado / (2 * (tamanho_do_audio - 1))
    frequencia_hertz = f1 * fc

    # Aplicamos a equação de butterworth com a passa baixa
    for i in range(i, tamanho_do_audio, 1):
        if(i < fc):
            audio[i] = audio[i] * (1 / np.sqrt((1 + (np.power(((f1 * (i + 1)) / frequencia_hertz), 2 * n)))))
        else:
            audio[i] = 0

    return
```

```python
# Resultado
frq, audio = wavfile.read('maisumasemana.wav')
corte = 8000

audio_DCT = DCT_1D(audio)

filtro_butterworth(audio_DCT, corte)

audio_IDCT = IDCT_1D(audio_DCT)

wavfile.write('resultado.wav', frq, audio_IDCT.astype(np.int16))
```

Figura 9: Função aplica_filtro

Resultados

Abaixo podemos observar na prática o uso da linguagem Python em conjunto com alguns de seus frameworks os quais auxiliaram na extração, leitura e manipulação de imagens e áudio para o desenvolvimento do trabalho, exibindo assim seus resultados para comprovamos o uso eficaz dos filtros ensinados em sala de aula.

Imports e Acesso a Arquivos

Para o desenvolvimento do trabalho foi utilizada a linguagem de programação Python em conjunto com as suas seguintes bibliotecas:

```
from scipy.io import wavfile
import numpy as np
import numpy
import math
```

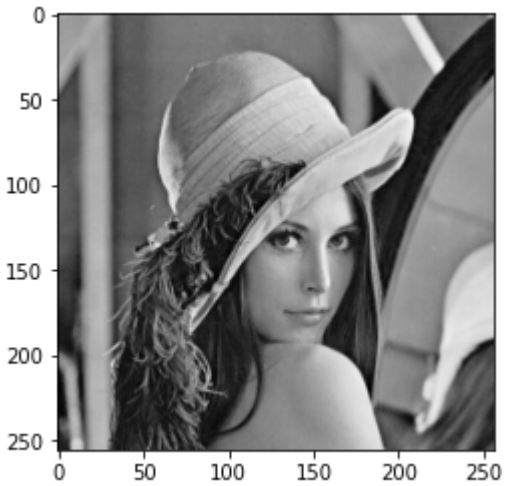
```
import math
from PIL import Image
from scipy import fftpack
from numba import jit
import cv2
from matplotlib.pyplot import imshow
from google.colab.patches import cv2_imshow
from numpy.fft import fft2,fftshift,ifft2,ifftshift
from scipy.signal import butter, lfilter, freqz, filtfilt
from matplotlib import pyplot as plt
```

Importando dados da imagem utilizada:

```
!wget -O 'lena256.png' 'https://i.ibb.co/8KkxSKF/lena256.png'
```

Imagem Utilizada:

```
lena256 = Image.open('lena256.png')
lena256 = np.asarray(lena256)
plt.imshow(lena256, cmap="gray")
plt.show()
```



Importando dados do áudio utilizado:

```
!wget -O 'maisumasemana.wav' 'https://www.dropbox.com/s/nym5313z3unt8nv/MaisUmaSemana.
```

Questão 1

DCT (1D e 2D).

Definindo a fórmula da DCT 1D:

```
@jit(nopython=True)
def DCT_1D(data):

    N = len(data)
    X = np.zeros(N)
    aK = math.sqrt(2.0/N)

    #Fórmula de Cossenos Discretos
    for k in range(N):
        if (k == 0):
            ck = math.sqrt(1.0/2.0)
        else:
            ck = 1
        s = 0
        for n in range(N):
            a1 = 2.0 * math.pi * k * n
            a2 = k * math.pi
            nn = 2.0 * N
            s += data[n] * math.cos((a1/nn) + (a2/nn))
        X[k] = aK * ck * s

    return X
```

Aplicando a DCT 2D:

```
# Cria matrizes vazias
imgDCT = np.zeros(lena256.shape)
img_DCT_horizontal = np.zeros(lena256.shape)
img_DCT_vertical = np.zeros(lena256.shape)

# DCT nas linhas - horizontal
for i, linha in enumerate(lena256):
    img_DCT_horizontal[i] = DCT_1D(linha)

# Transposição dos dados
img_input_vertical = img_DCT_horizontal.T

# DCT nas colunas - vertical
for i, linha in enumerate(img_input_vertical):
    img_DCT_vertical[i] = DCT_1D(linha)

# Transposição dos dados
imgDCT = img_DCT_vertical.T
```

1.1 Exibir o módulo normalizado (expansão de histograma para [0, 255]) da DCT de I, sem o nível DC, e o valor (numérico) do nível DC

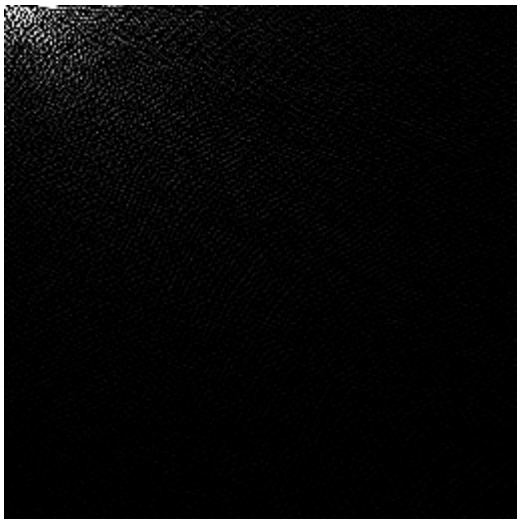
Nível DC:

```
# Exibição
imgDCT_with_DC = imgDCT.copy()
print("Nivel DC: {}".format(imgDCT[0][0]))
```

```
Nivel DC: 31883.60937500001
```

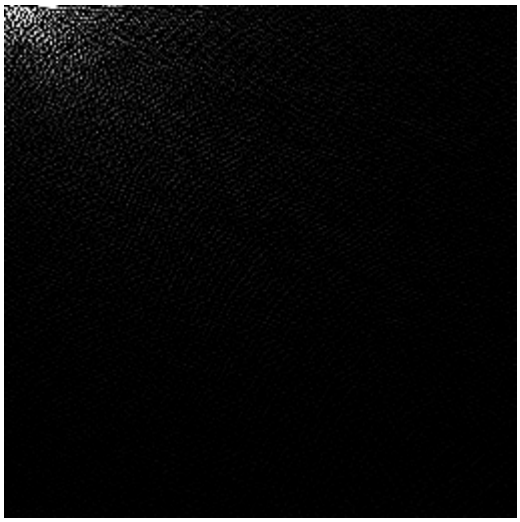
DCT Com nivel DC:

```
# Exibição
dct_image = Image.fromarray(imgDCT)
dct_image = dct_image.convert("P")
display(dct_image)
```



DCT sem nível DC:

```
# Exibição
imgDCT[0][0] = 0
dct_image = Image.fromarray(imgDCT)
dct_image = dct_image.convert("P")
display(dct_image)
```



Módulo normalizado da DCT:

```
# Armazena dados a serem utilizados
imagem_antiga = np.copy(imgDCT)
imagem_nova = np.ndarray(imagem_antiga.shape)

# Armazena o Rmax
r_max = np.max(imagem_antiga)

# Define o fator da expansão
fator = (256-1)/(r_max)

# Cria uma nova imagem depois de realizar a operação com o fator para normalização
imagem_nova[:,:] = (imagem_antiga[:,:]) * fator

# Exibição
dct_image = Image.fromarray(imagem_nova)
dct_image = dct_image.convert("P")
display(dct_image)
```

[redacted]

[redacted]

IDCT 1D e 2D

Definição da fórmula da IDCT 1D

```
# Conversão de retorno para o Dominio Espacial
@jit(nopython=True)
def IDCT_1D(data):

    N = len(data)
    x = np.zeros(N)

    aK = math.sqrt(2.0/N)

    for n in range(N):
        s = 0
        for k in range(N):
            a1 = 2.0 * math.pi * k * n
            a2 = k * math.pi
            nn = 2.0 * N
            cK = math.sqrt(1.0/2.0) if k == 0 else 1

            s += cK * data[k] * math.cos((a1/nn) + (a2/nn))

        x[n] = aK * s

    return x
```

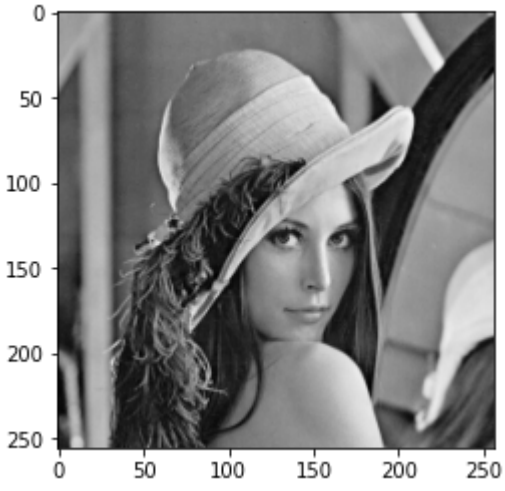
Aplicando a IDCT 2D:

```
# Cria matrizes vazias
imgIDCT = np.zeros(imgDCT.shape)
img_IDCT_horizontal = np.zeros(imgDCT.shape)
img_IDCT_vertical = np.zeros(imgDCT.shape)

imgDCT = numpy.array(imgDCT)
```

```
# DCT nas linhas - horizontal
for i, linha in enumerate(imgDCT):
    img_IDCT_horizontal[i] = IDCT_1D(linha)

# Transposição dos dados
img_input_vertical = img_IDCT_horizontal.T

# DCT nas colunas - vertical
for i, linha in enumerate(img_input_vertical):
    img_IDCT_vertical[i] = IDCT_1D(linha)

img_IDCT_vertical = img_IDCT_vertical.T
```

Resultado ao aplicar a IDCT2D:

```
plt.imshow(img_IDCT_vertical, cmap="gray")
plt.show()
```



1.2. Encontrar e exibir uma aproximação de I obtida preservando o coeficiente DC e os n coeficientes AC mais importantes de I, e zerando os demais. O parâmetro n é um inteiro no intervalo [0, RxC-1]

Definindo os passos para a resolução:

```
# Define o N, e armazena a imagem
img_selected_freq = []
samples = 100
imgDCT_original = imgDCT_with_DC.copy()

# Achata a imagem para 1 dimensão e pega o tamanho dela para depois ordenar
imgDCT_original = imgDCT_original.flatten()
imgDCT_original_ac = imgDCT_original[1:]
size_original_ac = len(imgDCT_original_ac)

# Retorna os dados de cada coordenada e valor para a lista e armazena na tupla
imgDCT_original_tuple =  list(np.ndenumerate(imgDCT_original))

# RC-N, calcula o valor absoluto utilizando o tamanho da imagem original menos o n
index_reset = np.abs(imgDCT_original_ac).argsort()[:len(imgDCT_original) - samples]

# Substituição dos valores que não são importantes por zero
for i in range (0, size_original_ac):
    if(i in index_reset):
        imgDCT_original_ac[i] = 0

# Retorna ao formato original, e em seguida o shape padrão
imgDCT_original[1:] = imgDCT_original_ac
imgDCT_original = imgDCT_original.reshape(lena256.shape)
```

Expansão de histograma:

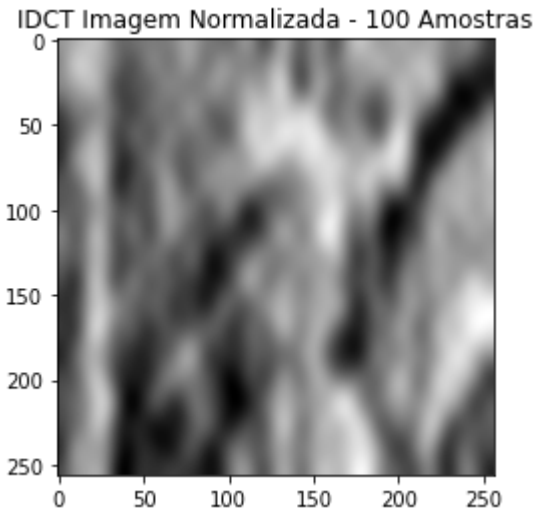
```
# Armazena a imagem normalizada
imagem_antiga = imgDCT_original
norm_img = np.ndarray(imagem_antiga.shape)

# Armazena o Rmax
r_max = np.max(imagem_antiga)

# Define o fator da expansão
fator = (256-1)/(r_max)

# Cria uma nova imagem depois de realizar a operação com o fator para normalização
norm_img[:,:] = (imagem_antiga[:,:]) * fator
```

Função para aplicar IDCT:

```
def apply_IDCT(data):

    # Armazena formato da imagem e inicia contador de tempo
    imgIDCT = np.zeros(lena256.shape)

    # Aplicação de IDCT na horizontal
    for i, linha in enumerate(data):
        imgIDCT[i] = IDCT_1D(linha)

    imgIDCT= imgIDCT.T

    # Aplicação de IDCT na vertical
    for i, linha in enumerate(imgIDCT):
        imgIDCT[i] = IDCT_1D(linha)

    imgIDCT  = imgIDCT.T
    data_image = imgIDCT.copy()

    return imgIDCT, data_image
```

Exibição da imagem normalizada:

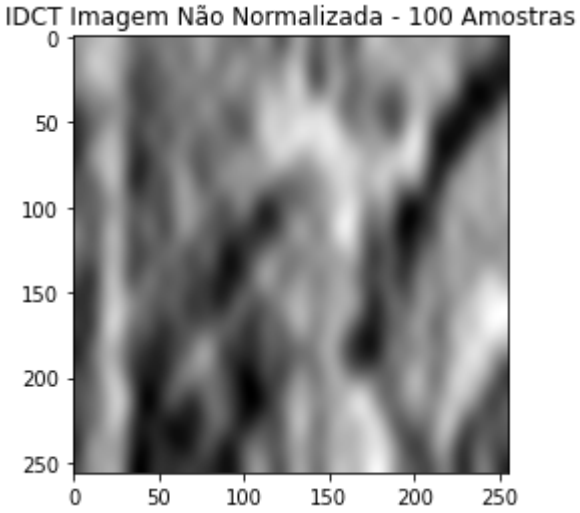
```
# Aplica IDCT na imagem normalizada
imgIDCT, data_image_norm = apply_IDCT(norm_img)

# Exbição
plt.imshow(imgIDCT, cmap="gray")
plt.title("IDCT Imagem Normalizada - {} Amostras".format(samples))
plt.show()
```



Exibição da imagem não normalizada:

```
# Aplica IDCT na imagem não normalizada
```

```
data_image_non_normalized = imgDCT_original.copy()
imgIDCT, data_image_non_orm = apply_IDCT(data_image_non_normalized)

# Exibição
plt.imshow(imgIDCT, cmap="gray")
plt.title("IDCT Imagem Não Normalizada - {} Amostras".format(samples))
plt.show()
```



Questão 2

Função para definir como aplicar o filtro:

```
def aplica_filtro(img, filtro):

    # Armazena linhas e colunas e cria uma matriz de zeros
    linhas, colunas = img.shape
    matriz = np.zeros((linhas, colunas))
    n = 1

    # Percorre a matriz atribuindo n na posição das coordenadas i e j
    for i in range(linhas):
        for j in range(colunas):
            matriz[i, j] =n
            n =-n
        n =-n

    # Rearranja a transformada de Fourier movendo as frequências zero para o centro do
    # Como é uma matriz, troca o primeiro quadrante da (imagem * matriz) pelo terceiro
    # Basicamente é como se processasse vários sinais 1-D e atribuisse nas linhas
    # E computa a transformada de fourier e inverte as frequências zero de cada coluna
    primeira_shift_fft = (fftshift(img * matriz))

    # Retorna a transformada de Fourier 2D
    primeira_fft2 = (fft2(primeira_shift_fft))
```

```
    # Retorna a inversa da transformada de Fourier 2D e aplica o filtro
    inversa_fft2 = (ifft2(primeira_fft2 * filtro))

    # Inverte o shift
    inversa_shift_fft = (ifftshift(inversa_fft2))

    # Armazena o resultado de acordo com a matriz
    resultado = (inversa_shift_fft * matriz)

    return np.real(resultado)
```

Função para definir como aplicar o filtro:

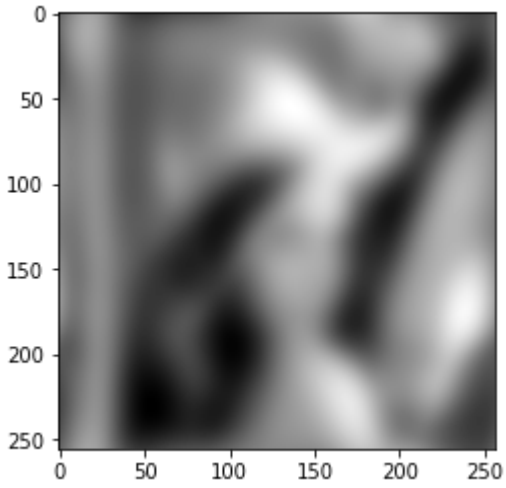
```
def filtro_butterworth(img, D0, n, lowpass = True):

    # Armazena e indexa o tamanho de colunas e linhas
    row, col = img.shape
    rowsIndex=np.zeros((row, col))
    colsIndex=np.zeros((row, col))

    # Para a diagonal, ambos os indexadores recebem i
    if row == col :
        for i in range(row):
            for j in range(col):
                rowsIndex[i, j] = i
        colsIndex=rowsIndex.T
    # Para o resto recebem suas determinadas coordenadas i e j
    else:
        for i in range(row):
            for j in range(col):
                rowsIndex[i, j] = i
                colsIndex[i, j] = j

    # Cria uma matriz de 0
    D=np.zeros((row, col))

    # Aplica a equação em cada elemento de D
    for i in range(row):
        for j in range(col):
            D[i, j] = np.sqrt(pow((rowsIndex[i, j] - row//2), 2) + pow((colsIndex[i, j

    # Aplicando a passa-baixa
    if lowpass:
        L = 1 /(1 + (D / D0) ** (2 * n))
        return L
```

Exibição da imagem após filtro:

```
# Salva a imagem em uma variável e passa a imagem, o D0 e o n para o filtro
imagem_original = cv2.imread("lena256.png", 0)
filtro = filtro_butterworth(imagem_original, 5, 2)
```

```
# Exibe o resultado
imagem_resultante = aplica_filtro(imagem_original, filtro)
plt.imshow(imagem_resultante, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7fed1f04da50>
```



Questão 3

```
def filtro_butterworth(audio, fc):

    # Obtém o tamanho do audio e define o i e o n
    tamanho_do_audio = audio.size
    n = 3
    i=0

    # Adquirimos o os fatores para realizar as operações
    fa_utilizado = 1 / (tamanho_do_audio - 1)
    f1 = fa_utilizado / (2 * (tamanho_do_audio - 1))
    frequencia_hertz = f1 * fc

    # Aplicamos a equação de butterworth com a passa baixa
    for i in range(i, tamanho_do_audio, 1):
        if(i < fc):
            audio[i] = audio[i] * (1 / np.sqrt((1 + (np.power(((f1 * (i + 1)) / freque
        else:
            audio[i] = 0

    return


# Resultado
frq, audio = wavfile.read('maisumasemana.wav')
corte = 8000

audio_DCT = DCT_1D(audio)

filtro_butterworth(audio_DCT, corte)
```

```
audio_IDCT = IDCT_1D(audio_DCT)

wavfile.write('resultado.wav', frq, audio_IDCT.astype(np.int16))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: WavFileWarning: C
```

Considerações finais

O presente trabalho, obteve exito ao resolver as questões propostas pela discinplina de Introdução ao Processamento Digital de Imagens, onde, a partir do suporte da linguagem Python e suas bibliotecas, auxiliou na conclusão. Embora, o desenvolvimento do projeto foi seguido por diversas dificuldades, principalmente em relação as questões 2 e 3, quanto em relação a transferir as formulas dos filtros Butterworth e o Passa-Baixas, além de utilizar a bibliteca Numpy para aplicar a formula de Fourier, parte essencial para a aplicação do filtro. Em Relação a questão 3, por motivos que não foram possiveis de resolver, no áudio final, exportado pelo código, não foi possível detectar qualquer tipo de som audivel.

Referências

Disponível em: http://www.ic.uff.br/~aconci/CosenosTransformada.pdf
acessado: 24/05/2022.
Disponível em : https://stringfixer.com/pt/Discrete_cosine_transform
acessado: 24/05/2022.
Disponível em : https://stringfixer.com/pt/Discrete_cosine_transform
acessado: 25/05/2022.
Disponivel em : http://www.ic.uff.br/~aconci/CosenosTransformada.pdf
acessado: 30/05/2022
Disponível em : https://users.cs.cf.ac.uk/Dave.Marshall/Vision_lecture/node20.html
acessado: 25/05/2022.

✓ 2m3s conclusão: 09:35