Tomografia Computadorizada MAP2110

Professor:	Nelson Mugayar Kuhl	NºUSP
Grupo:	Enzo Valentim Cappelozza	12556736
	Henrique Fujikawa Tokunaga	12675207
	Lucas Panfilo Donaire	12556552
	Marcos Martins Marchetti	11910868
	Ygor Peniche Maldonado	10271558

25 de junho de 2021

Conteúdo

Lista de Figuras

1 Introdução

Tomografia Computadorizada nomeia a técnica de reconstrução de imagens de seções transversais a partir de fluxos de Raios X, proporcionando um maior detalhamento quando comparado ao método convencional de Raios X.

A técnica consiste na emissão de Raios X em direção ao corpo de um paciente. Os Raios X possuem determinada quantidade padronizada de fótons. Parte destes, podem *colidir* com o corpo do paciente, o qual é composto de diferentes órgãos e tecidos. Estes *absorvem ou refletem* parte dos fótons devido as propriedades de suas composições orgânicas particulares. Dessa forma, os raios que colidem com o corpo e o atravessam, chegam à um receptor que detecta a *densidade* do feixe pela quantidade de fótons com que chega ao receptor.

Em geral, os feixes emitidos ou chegam com uma mesma densidade ao receptor, caso não colidam com nenhuma parte do corpo, ou chegam com uma densidade diferente, devido à absorção ou reflexão, de acordo com o tecido ou órgão com o qual colide. O feixe é captado e processado por um computador que implementa algorítimos que serão descritos posteriormente. O processo é repetido em diferentes ângulos n vezes com a rotação do sistema emissor-receptor até que se obtenham dados satisfatórios.

A partir das densidades de cada feixe de Raio X, podemos organizar os dados em um sistema de matrizes sobredeterminado. Por meio do uso de ferramentas computacionais aplicados as densidades dos feixes, podemos trabalhar na geração de uma imagem de interesse. Em geral, busca-se realizar a construção da imagem de uma seção transversal do corpo de um paciente para fins médicos.

No esquema a seguir podemos visualizar um esboço de como a seção transversal de uma região do corpo é obtida a partir dos pixels de uma imagem. Note que no exemplo abordamos uma imagem em duas dimensões. A tomografia computadorizada atual nos permite, caso o sistema translade tanto horizontalmente quanto verticalmente, reconstruir imagens em três dimensões, inclusive com distinção detalhada de órgãos e tecidos.

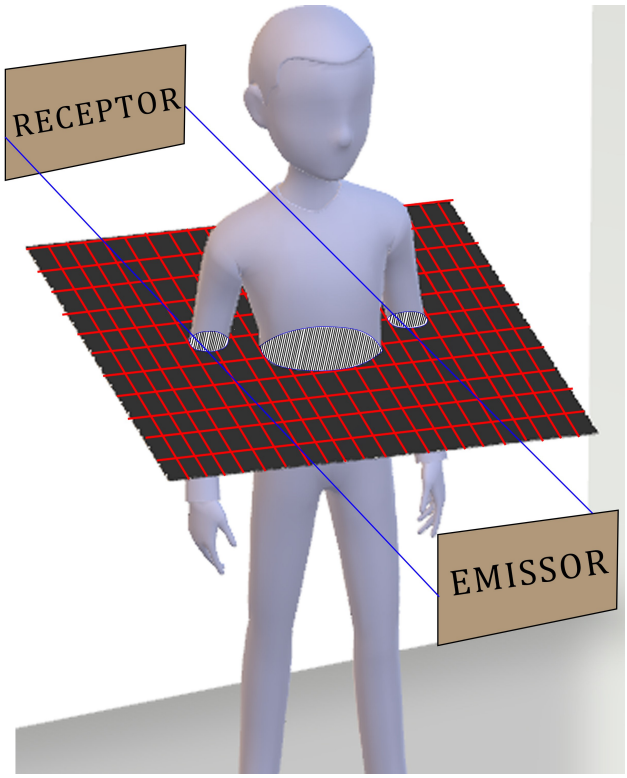


Figura 1: Seção transversal no sistema emissor-receptor

Tome, por exemplo que a região hachurada é objeto de interesse clínico. Tal como no exemplo, no presente trabalho trataremos de imagens em duas dimensões. Nesse sentido, apresentamos as variáveis de nosso estudo como:

A densidade de Raios X absorvida pelo j-ésimo pixel é denotada por x_j e é definida por:

$$x_j = ln\left(\frac{número\ de\ fótons\ entrando\ no\ j-ésimo\ pixel}{número\ de\ fótons\ saindo\ do\ j-ésimo\ pixel}\right)$$

Usando a propriedade logarítmica $ln(a/b) = -ln(b/a)$, temos

$$x_j = -ln\left(\begin{matrix}fração\ de\ fótons\ que\ passa\ pelo\\ j-ésimo\ pixel\ sem\ ser\ absorvida.\end{matrix}\right)$$

Tendo em vista que o número de fótons que saem de um pixel é igual ao número de fótons que entram no pixel seguinte, podemos afirmar que:

$$x_1 + x_2 + \cdots + x_n = ln\left(\frac{número\ de\ fótons\ entrando\ no\ primeiro\ pixel}{número\ de\ fótons\ saindo\ do\ n-ésimo\ pixel}\right)$$

$= -ln\left(fração\ de\ fótons\ que\ passa\ pela\ linha\ de\ n\ pixels\ sem\ ser\ absorvida\right)$

Em que os pixels são enumerados 1, 2,..., n

Desse modo, a densidade de Raios X de uma fileira é dada pela soma das densidades dos pixels individuais daquela fileira. Denotando a densidade do i-ésimo feixe de um "escaneamento"por b_i, temos que ela é dada por:

$$b_i = ln\begin{pmatrix} número\ de\ fotótons\ do\ i-ésimo\ feixe\ entrando \\ no\ detector\ sem\ ter\ a\ seção\ transversal\ no\ campo\ de\ visão \end{pmatrix}$$

=

$-ln\left(fração\ de\ fótons\ do\ i-ésimo\ feixe\ que\ passa\ pela\ seção\ transversal\ sem\ ser\ absorvida\right)$

$$\begin{pmatrix} fração\ de\ fótons\ do\ feixe\ que\ passa \\ pela\ fileira\ de\ pixelssemserabsorvida \end{pmatrix} = \begin{pmatrix} fração\ de\ fótons\ do\ feixe\ que\ passa \\ pela\ seção\ transversal\ sem\ ser\ absorvida \end{pmatrix}$$

Assim, a partir das equações (1) e (2) temos:

$$x_1 + x_2 + \cdots + x_n = b_i$$

Nela, a densidade b_i é obtida paor meio do aparelho de tomografia e $x_1, x_2, ..., x_n$ são densidades desconhecidas de cada pixel que devem ser determinadas.

Se o $i-ésimo$ feixe passa por um conjunto de pixels que numeramos como $j_1, j_2, ..., j_i$ então temos:

$$x_{j1} + x_{j2} + ... + x_{ji} = b_i$$

Caso definirmos

$$a_{ij} = \begin{cases} 1, \text{ se } j = j_1, j_2, ..., j_i \\ 0, \text{ caso contrário} \end{cases}$$

Podemos escrever a equação anterior como:

$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{iN}x_N = b_i$$

Sendo chamada de i-ésima equação de feixe.

Tendo em vista que os feixes de Raio X não passam necessariamente perpendicularmente em cada pixel, podemos atribuir diferentes valores para a_{ij} diferentes de 1 a fim de obter melhor precisão do valor para cada pixel. As maneiras de se definir a_{ij} são:

- O Método do Centro do Pixel
- O Método da Reta Central
- O Método da Área

Supondo que foram emitidos M feixes de Raios X no total, as M equações de feixe de um "escaneamento"completo serão dadas por:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1N}x_N = b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{iN}x_N = b_2 \\ \\ a_{M1}x_1 + a_{M2}x_2 + ... + a_{MN}x_N = b_M \end{cases}$$

Obtemos, portanto, um sistema linear de M equações (equações do feixe) e N incógnitas (densidades de cada pixel) no qual será considerado o caso sobredeterminado, em que $M > N$, pois é o caso de ocorrência nos tomógrafos. Vale ressaltar que esse sistema não possui solução matemática exata em razão dos erros experimentais e de modelagem inerentes ao problema. Dessa forma, faremos uso de um algoritmo que permite encontrar uma solução aproximada para o sistema linear.

2 Algoritmos

O algorítmo abaixo parte das *Técnicas de Reconstrução Algébrica* (TRA). Um paralelo realizado em Python pode ser encontrado nos anexos deste trabalho. Então, temos que :

$$x_k^{(p)} = x_{k-1}^{(p)} + \frac{b_k - a_k^t x_{k-1}^{(p)}}{a_k^t a_k} a_k$$

equivale a uma projeção em cada um dos hiperplanos. Criamos então outra função, que assume $x_0^{(p+1)} = x_m^{(p)}$ e repete a primeira até os números passarem a mudar menos de 10^{-10} por rodada.

Tentamos também fazer uma função que, dado o lado do quadrado, gerasse a matriz A para o método escolhido entre os três. Conseguimos com sucesso programar os valores de A para os feixes horizontais e verticais, pois é fácil padronizar uma lei algébrica para eles. Porém, nossa tentativa esbarrou em problemas em uma generalização algébrica de quais pixels cada feixe diagonal ia passar, então decidimos manter apenas as maiores diagonais do quadrado, a diagonal principal e a secundária. No caso da matriz 3x3, a diagonal principal é a que passa pelos pixels [1, 5, 9], e a secundária pelos [3,5,7].

Pelo algorítmo reconstruímos as imagens do exercício 3, e testamos outras 2, comentadas e inseridas com o anexo do python. Um exemplo de reconstrução de imagem se faz presente na página 11.

3 Equações

A partir das densidades dos feixes medidos neste exemplo seguinte, podemos construir as 12 equações a partir dos valores de densidade.

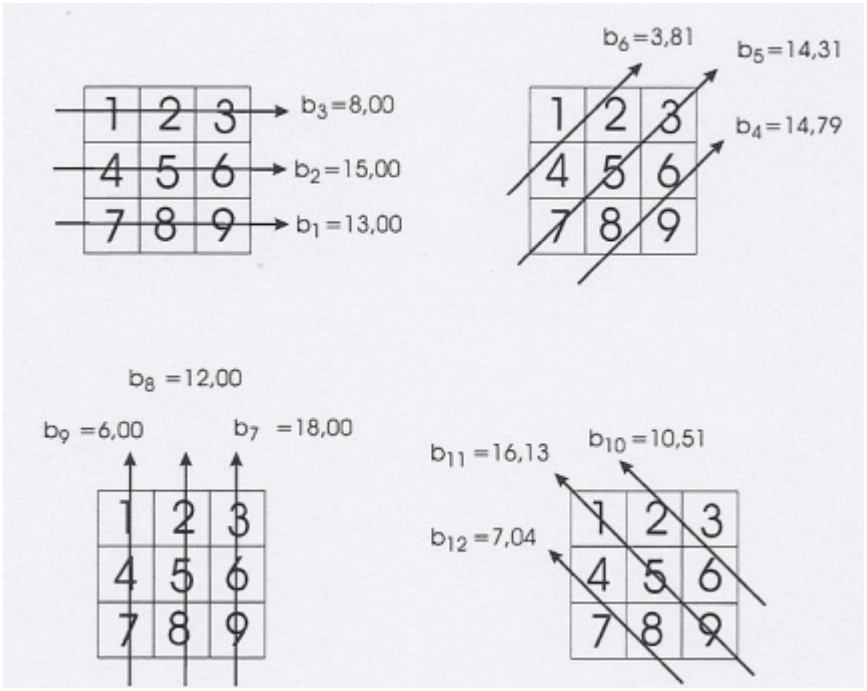


Figura 2: Feixes de Raio X e suas medidas no detector

Para os cálculos será fixado a largura do pixel em 1.

3.1 Método do Centro do Pixel

Pelo Método do Centro do Pixel, temos que o a_{ij}, é dado por:

$$\begin{cases} 1 \ se \ o \ i-ésimo \ feixe \ passa \ pelo \ centro \ do \ j-ésimo \ pixel \\ 0 \ caso \ contrário \end{cases}$$

Para os casos dos feixes horizontais e verticais, a coluna ou linha toda assume valor 1 e o resto 0. Esses casos são os feixes 1, 2, 3, 7, 8 e 9 do exercício.

Para os casos de feixes diagonais, trata-se de determinar quais pixels tem seu centro interno ao feixe, para assmir o valor 1 neles, e isso é relativamente simples: no exercício, há 6 feixes diagonais, os feixes 4, 5, 6, 10, 11 e 12, que passam, respectivamente pelos pixels [6, 8, 9], [3, 5, 7], [1, 2, 4], [2, 3, 6], [1, 5, 9], [4, 7, 8].

Assim, as equações do feixe são:

$$\begin{cases} x_7 + x_8 + x_9 = 13,00 \\ x_4 + x_5 + x_6 = 15,00 \\ x_1 + x_2 + x_3 = 8,00 \\ x_6 + x_8 + x_9 = 14,79 \\ x_3 + x_5 + x_7 = 14,31 \\ x_1 + x_2 + x_4 = 3,81 \\ x_3 + x_6 + x_9 = 18,00 \\ x_2 + x_5 + x_8 = 12,00 \\ x_1 + x_4 + x_7 = 6,00 \\ x_2 + x_3 + x_6 = 10,51 \\ x_1 + x_5 + x_9 = 16,13 \\ x_4 + x_7 + x_8 = 7,04 \end{cases}$$

Utilizando a fórmula do algoritmo generalizado, calculado através do programa em python no anexo, obtemos os seguintes resultados:

$$\begin{cases} x_1 = 1.3100000003736945 \\ x_2 = 0.6133333261584362 \\ x_3 = 5.303333350087902 \\ x_4 = 2.143333342913005 \\ x_5 = 7.49 \\ x_6 = 4.593333323753661 \\ x_7 = 1.773333316578765 \\ x_8 = 3.1233333405082297 \\ x_9 = 7.329999999626303 \end{cases}$$

O resultado da reconstrução da imagem é a imagem 3 - página 11

O algorítmo foi programado de forma que só ia parar as projeções quando a diferença entre as incógnitas fosse menor que 10^{-10}. Assim, como houveram 193 rodadas de projeções pra isso, os valores, apesar de praticamente os mesmos do artigo com 45 rodadas, apresentam leve variância, mas até mais precisão.

3.2 Método da Reta Central

Pelo Método da Reta Central, temos que o a_{ij}, é dado por:

$$\left(\frac{Comprimento\ da\ reta\ central\ do\ i-ésimo\ feixe\ que\ fica\ no\ j-ésimo\ pixel}{Largura\ do\ j-ésimo\ pixel} \right)$$

Como temos a largura do pixel, é necessário que seja encontrado o comprimento da reta central dos feixes.

Para os casos em que os feixes passam verticalmente e horizontalmente pelo centro dos pixels (matrizes 1 e 3), a medida do comprimento da reta central é a própria largura do pixel, ou seja, 1.

Nos casos em que o feixes passam diagonalmente pelos pixels (matrizes 2 e 4), deve-se obter as medidas dos comprimentos dos feixes. Para os feixes que passam diagonalmente pelos pixels [7, 5, 3] e [1, 5, 9] têm-se que, por pitagoras, a medida do comprimento da reta central, quando passa por 1 pixel, é de $\sqrt{2}$ [1]. Nos demais feixes, pode-se utilizar a distância dada entre a reta central de feixes, que é de 1 pixel, ou seja, igual a 1.

Utilizando o feixe que passa no pixel [5], temos que a medida de sua diagonal é de $\sqrt{2}$. A distância da diagonal do ponto central do pixel até sua lateral é de $\frac{\sqrt{2}}{2}$, e ao subtrair de 1 (distância entre os feixes), tem-se a distância restante até o feixe adjascente, que passa em [1, 9] na matriz 2 e em [3, 7] na matriz 4, e é dada por $1 - \frac{\sqrt{2}}{2}$. Como esta é a distância até o centro do quadrado, que pode ser desenhado com ínicio onde o feixe passa pelo pixel, tem-se que a medida do comprimento do feixe neste instante é de $2 \cdot (1 - \frac{\sqrt{2}}{2})$ [2]. Para os pixels [2, 4, 6, 8], tem-se a medida da diagonal do pixel subatraído pela distância entre os feixes. Como em [2], basta multiplicar por 2, obtendo $2 \cdot (\sqrt{2} - 1)$ [3]. Desta maneira, as equações de feixe são:

$$
\begin{cases}
x_7 + x_8 + x_9 = 13,00 \\
x_4 + x_5 + x_6 = 15,00 \\
x_1 + x_2 + x_3 = 8,00 \\
\\
2 \cdot (\sqrt{2} - 1) \cdot x_6 + 2 \cdot (\sqrt{2} - 1) \cdot x_8 + 2 \cdot (1 - \frac{\sqrt{2}}{2}) \cdot x_9 = 14,79 \\
\sqrt{2} \cdot x_3 + \sqrt{2} \cdot x_5 + \sqrt{2} \cdot x_7 = 14,31 \\
2 \cdot (1 - \frac{\sqrt{2}}{2}) \cdot x_1 + 2 \cdot (\sqrt{2} - 1) \cdot x_2 + 2 \cdot (\sqrt{2} - 1) \cdot x_4 = 3,81 \\
\\
x_3 + x_6 + x_9 = 18,00 \\
x_2 + x_5 + x_8 = 12,00 \\
x_1 + x_4 + x_7 = 6,00 \\
\\
2 \cdot (\sqrt{2} - 1) \cdot x_2 + 2 \cdot (1 - \frac{\sqrt{2}}{2}) \cdot x_3 + 2 \cdot (\sqrt{2} - 1) \cdot x_6 = 10,51 \\
\sqrt{2} \cdot x_1 + \sqrt{2} \cdot x_5 + \sqrt{2} \cdot x_9 = 16,13 \\
2 \cdot (\sqrt{2} - 1) \cdot x_4 + 2 \cdot (1 - \frac{\sqrt{2}}{2}) \cdot x_7 + 2 \cdot (\sqrt{2} - 1) \cdot x_8 = 7,04
\end{cases}
$$

Utilizando a fórmula do algoritmo generalizado, calculado através do pro-

grama em python no anexo, obtemos os seguintes resultados:

$$\begin{cases} x_1 = 2.098965791757338 \\ x_2 = 1.398821682528026 \\ x_3 = 3.9607546785997756 \\ x_4 = 1.5590027038167802 \\ x_5 = 4.310348216238125 \\ x_6 = 8.48719409588834 \\ x_7 = 1.7862986873734144 \\ x_8 = 5.675925120670143 \\ x_9 = 4.9963183725435485 \end{cases}$$

O resultado da reconstrução da imagem é a imagem 4 - página 12

3.3 Método da Área

Para o método da área, foram usadas as seguintes retas parametrizadas com o parâmetro t. Onde r_0 representa o feixe que passa pela origem, as retas cujos índices são ímpares representam os limites do alcance de cada feixe, e as retas cujos índices são pares representam os feixes em questão. E devido à simetria do problema, podemos simplesmente "rebater", para a esquerda, os valores dos índices à direita da diagonal definida por r_0.

$$\begin{aligned} r_0 &: y = x - t \times \sqrt{2},\ t = 0 \\ r_1 &: y = x - t \times \sqrt{2},\ t = 1 \\ r_2 &: y = x - t \times \sqrt{2},\ t = 2 \\ r_3 &: y = x - t \times \sqrt{2},\ t = 3 \\ r_4 &: y = x - t \times \sqrt{2},\ t = 4 \end{aligned}$$

Abaixo podemos analisar um Grid 3×3 relativo aos 9 pixels, tal como solicitado.

O método usado define os coeficientes a_{ij} de cada uma das equações lineares como:

$$\left(\frac{\textit{Área total contemplada pelo feixe}}{\textit{Área total que o feixe contemplaria caso fosse perpendicularmente incidente}} \right)$$

Desse modo, as equações de feixe são:

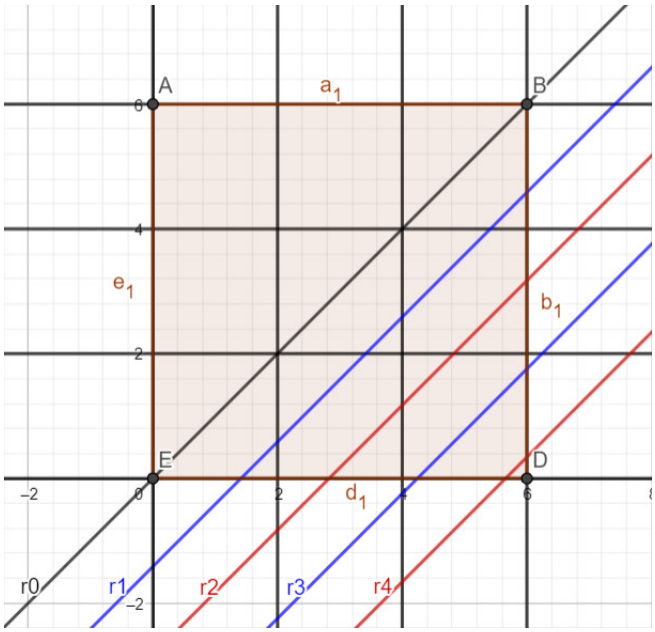


Figura 3: Ilustração das retas e das áreas

$$
\begin{cases}
x_7 + x_8 + x_9 = 13,00 \\
x_4 + x_5 + x_6 = 15,00 \\
x_1 + x_2 + x_3 = 8,00 \\
0.0425x_3 + 0.0425x_5 + 0.75x_6 + 0.0425x_7 + 0.75x_8 + 0.6125x_9 = 14,79 \\
0.25x_2 + 0.915x_3 + 0.25x_4 + 0.915x_5 + 0.25x_6 + 0.915x_7 + 0.25x_8 = 14,31 \\
0.6125x_1 + 0.75x_2 + 0.0425x_3 + 0.75x_4 + 0.0425x_5 + 0.0425x_6 = 3,81 \\
x_3 + x_6 + x_9 = 18,00 \\
x_2 + x_5 + x_8 = 12,00 \\
x_1 + x_4 + x_7 = 6,00 \\
0.0425x_1 + 0.75x_2 + 0.6125x_3 + 0.0425x_5 + 0.75x_6 + 0.0425x_9 = 10,51 \\
0.915x_1 + 0.25x_2 + 0.25x_4 + 0.915x_5 + 0.25x_6 + 0.25x_8 + 0.915x_9 = 16,13 \\
0.0425x_1 + 0.75x_4 + 0.0425x_5 + 0.6125x_7 + 0.75x_8 + 0.0425x_9 = 7,04
\end{cases}
$$

$$\begin{cases} x_1 = 2.9974775772355478 \\ x_2 = 1.001974861222238 \\ x_3 = 4.0021066831980185 \\ x_4 = 1.007300736414646 \\ x_5 = 4.986648885173407 \\ x_6 = 9.008179159721546 \\ x_7 = 1.9968839418640043 \\ x_8 = 6.013298884851567 \\ x_9 = 4.991376353088405 \end{cases}$$

O resultado da reconstrução da imagem é a imagem 5 - página 12

4 Reconstrução da Imagem

Para realizar a reconstrução da imagem, primeiramente foi aplicado a técnina Min-Max Normalization em cada matriz X $[x_1, x_9]$. A técnica consiste em reescalar os valores da matriz proporcionalmente ao seus valores máximo e mínimo, obtendo como resultado uma nova matriz com os valores entre [0, 1]:

$$\begin{cases} \dfrac{x - min(x)}{max(x) - min(x)} \end{cases}$$

Desta maneira garantimos que os resultados obtidos com cada método produzam imagens com os resultados proporcionais. Após ao reescalonamento, foram atribuídas cores em escala de cinza ao resultados, no qual cores mais escuras representam valores próximo a 1, enquanto cores claras representam valores próximo de zero.

Aplicando o algoritmo generalizado para reconstrução da imagem, utilizando o programa em python no anexo, obtemos:

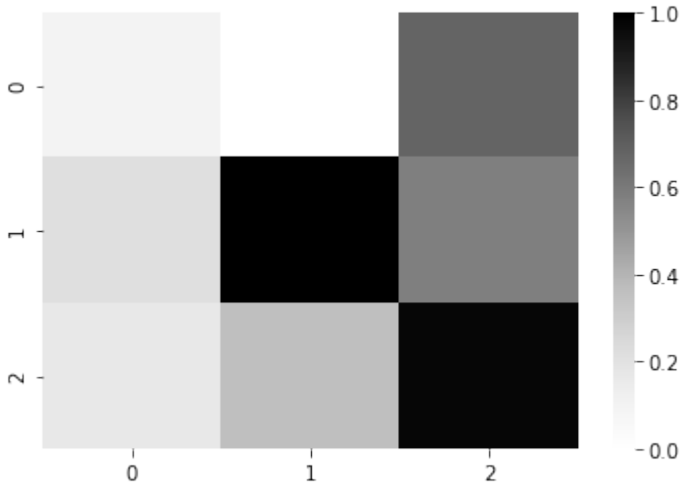


Figura 4: Centro do pixel

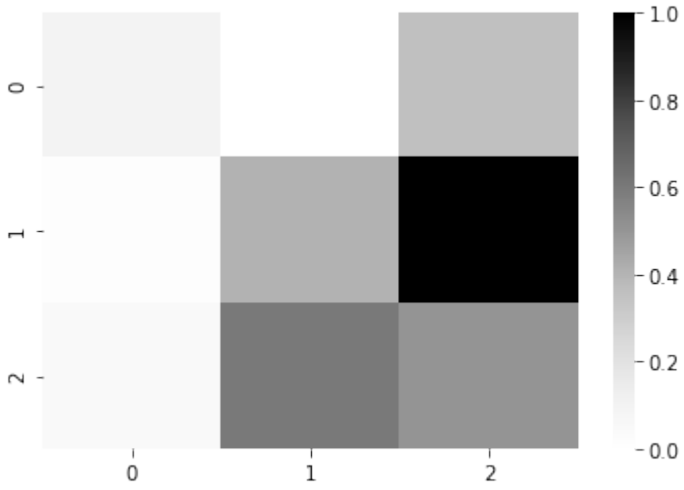


Figura 5: Reta central

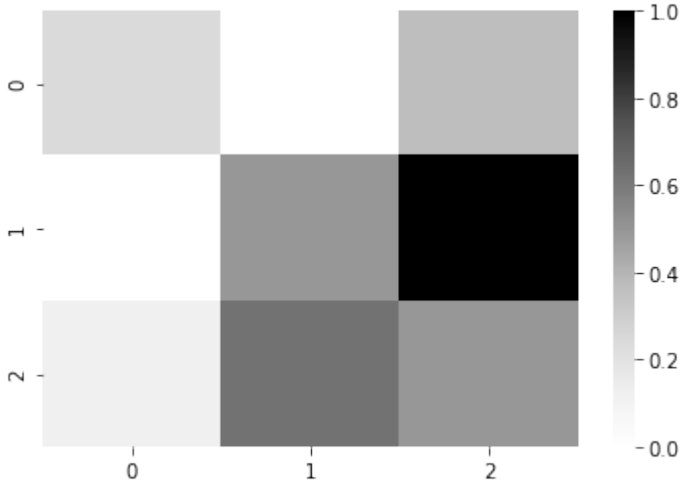


Figura 6: Método da Área

5 Simulações

Usando nosso algorítmo que cria a matriz A "incompleta", e criando o B através do processo reverso de criação de imagens, tentamos alterar levemente os valores da matriz B, o tal "ruído".
Nossa primeira tentativa foi usando uma foto de 6400 pixels do símbolo do IME com a face do Arquimedes, mas após muitas correções para adaptar o algorítmo, tivemos uma surpesa: o processador do computador usado não conseguiu fazer

tantas contas com matrizes de ordem 6400. Tivemos então que pegar uma imagem de 100 pixels e refizemos: dessa vez deu certo, apesar que a imagem não ficou parecida, pois as diagonais ausentes fizeram falta

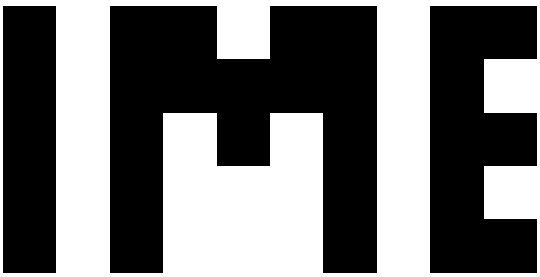


Figura 7: Imagem utilizada

6 Anexos

In [1]:

```
import numpy as np
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
import matplotlib.image as img
```

Tomografia Computadorizada

Funções para a Projeção Ortogonal

In [2]:

```
def mult(x, y):
    return np.dot(x, y)
```

In [3]:

```
def rod(a_, b_, xt_): # função de 1 rodada de projeções
    k_ = 0
    var = [0] * m

    while k_ < m:
        ai = np.array(a_[k_])
        at = np.transpose(ai)
        v1 = (b_[k_] - mult(at, xt_))
        v1 = v1[0]
        v2 = mult(at, ai)
        v2 = v2[0][0]

        if k_ == 0:
            proj = xt_ + (v1 / v2) * np.transpose(ai)
            xt_ = np.transpose(proj)

        else:
            proj = np.transpose(xt_) + (v1 / v2) * np.transpose(ai)
            xt_ = np.transpose(proj)
        var[k_] = proj
        k_ += 1
    return var
```

In [4]:

```
def calculate(a_, b_, xt_, epslon=10e-10):
    #função para rodar as projeções até aproximar resultados consecutivos
    i = 0
    dif = 1

    while dif > epslon:
        k = rod(a_, b_, xt_)
        xt_ = k[m-1][0]
        xt = np.transpose(xt_)
        k2 = rod(a_, b_, xt_)
        dif = abs(k[0][0][0] - k2[0][0][0])
        # print(k2[m-1])
        i += 1

    all_x = []
    # print(k2[m-1])
    for j in range(n):
        print(f'x{j+1} =', end=' ')
        print(k2[m-1][0][j])
        all_x.append(k2[m-1][0][j])

    return np.reshape(all_x, (-1, 1))
```

In [ ]:

```
```

Função para Criação da Imagem

In [5]:

```
def create_image(x_list):
    # Abaixo o reescalonamento Min-Max Normalization
    scl = MinMaxScaler(feature_range=(0, 1))
    scl.fit(x_list)
    scaled_x_list = scl.transform(x_list)

    # Tranformando a matriz em 3x3
    scaled_x_list = np.reshape(scaled_x_list, (-1, 3))
    print(scaled_x_list)

    sns.heatmap(scaled_x_list, vmin=0, vmax=1, cmap="Greys")
```

In [ ]:

```
```

Medidas dos Feixes no detector

In [6]:

```
b = [13.0, 15.0, 8.0, 14.79, 14.31, 3.81, 18.0, 12.0, 6.0, 10.51, 16.13, 7.04] #matriz b do

x0 = np.array([0., 0., 0., 0., 0., 0., 0., 0., 0.]) #ponto inicial antes das projeções - nã
xt = np.transpose(x0)
```

Método do Centro do Pìxel

Matriz obtida pelo Método do Centro do Pixel, aplicado a Figura 6 (feixes de Raios X)

1 - se o i-ésimo feixe passa pelo centro do j-ésimo pixel;

0 - caso contrário

In [7]:

```
m = 12

n = 9

a = [[[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [1.0], [1.0], [1.0]],
     [[0.0], [0.0], [0.0], [1.0], [1.0], [1.0], [0.0], [0.0], [0.0]],
     [[1.0], [1.0], [1.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]],

     [[0.0], [0.0], [0.0], [0.0], [0.0], [1.0], [0.0], [1.0], [1.0]],
     [[0.0], [0.0], [1.0], [0.0], [1.0], [0.0], [1.0], [0.0], [0.0]],
     [[1.0], [1.0], [0.0], [1.0], [0.0], [0.0], [0.0], [0.0], [0.0]],

     [[0.0], [0.0], [1.0], [0.0], [0.0], [1.0], [0.0], [0.0], [1.0]],
     [[0.0], [1.0], [0.0], [0.0], [1.0], [0.0], [0.0], [1.0], [0.0]],
     [[1.0], [0.0], [0.0], [1.0], [0.0], [0.0], [1.0], [0.0], [0.0]],

     [[0.0], [1.0], [1.0], [0.0], [0.0], [1.0], [0.0], [0.0], [0.0]],
     [[1.0], [0.0], [0.0], [0.0], [1.0], [0.0], [0.0], [0.0], [1.0]],
     [[0.0], [0.0], [0.0], [1.0], [0.0], [0.0], [1.0], [1.0], [0.0]]]
```

Resultado:

In [8]:

```
all_x = calculate(a, b, xt)
```

```
x1 = 1.310000000373695
x2 = 0.6133333261584377
x3 = 5.303333350087901
x4 = 2.1433333429130053
x5 = 7.490000000000001
x6 = 4.5933333237536615
x7 = 1.7733333165787661
x8 = 3.1233333405082293
x9 = 7.329999999626306
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

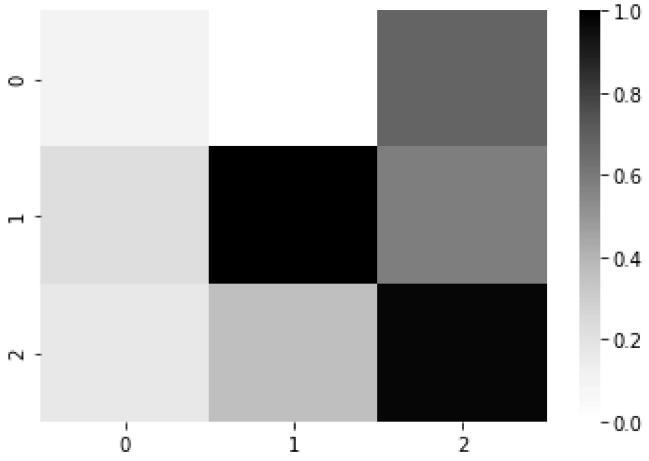
```
```

Imagem Gerada

In [9]:

```
create_image(all_x)
```

```
[[0.10130877 0.         0.68201648]
 [0.22249152 1.         0.57876878]
 [0.16868638 0.36500243 0.97673291]]
```



Método da Reta Central

Comprimento da reta central do i-ésimo feixe que fica no j-ésimo pixel / largura do j-ésimo pixel

In [10]:

```
2*(np.sqrt(2) - 1)
```

Out[10]:

```
0.8284271247461903
```

In [11]:

```
2*(1 - (np.sqrt(2)/2))
```

Out[11]:

```
0.5857864376269049
```

In [12]:

```
m = 12

n = 9

a = [[[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [1.0], [1.0], [1.0]],
     [[0.0], [0.0], [0.0], [1.0], [1.0], [1.0], [0.0], [0.0], [0.0]],
     [[1.0], [1.0], [1.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]],

     [[0.0], [0.0], [0.0], [0.0], [0.0], [2*(np.sqrt(2) - 1)], [0.0], [2*(np.sqrt(2) - 1)],
     [[0.0], [0.0], [np.sqrt(2)], [0.0], [np.sqrt(2)], [0.0], [np.sqrt(2)], [0.0], [0.0]],
     [[2*(1- (np.sqrt(2)/2))], [2*(np.sqrt(2) - 1)], [0.0], [2*(np.sqrt(2) - 1)], [0.0], [0

     [[0.0], [0.0], [1.0], [0.0], [0.0], [1.0], [0.0], [0.0], [1.0]],
     [[0.0], [1.0], [0.0], [0.0], [1.0], [0.0], [0.0], [1.0], [0.0]],
     [[1.0], [0.0], [0.0], [1.0], [0.0], [0.0], [1.0], [0.0], [0.0]],

     [[0.0], [2*(np.sqrt(2) - 1)], [2*(1- (np.sqrt(2)/2))], [0.0], [0.0], [2*(np.sqrt(2) -
     [[np.sqrt(2)], [0.0], [0.0], [0.0], [np.sqrt(2)], [0.0], [0.0], [0.0], [np.sqrt(2)]],
     [[0.0], [0.0], [0.0], [2*(np.sqrt(2) - 1)], [0.0], [0.0], [2*(1- (np.sqrt(2)/2))], [2*
```

In [ ]:

```

```

In [13]:

```
all_x = calculate(a, b, xt)
```

```
x1 = 2.098965791757338
x2 = 1.3988216825280262
x3 = 3.9607546785997756
x4 = 1.5590027038167802
x5 = 4.310348216238125
x6 = 8.48719409588834
x7 = 1.7862986873734144
x8 = 5.675925120670143
x9 = 4.9963183725435485
```

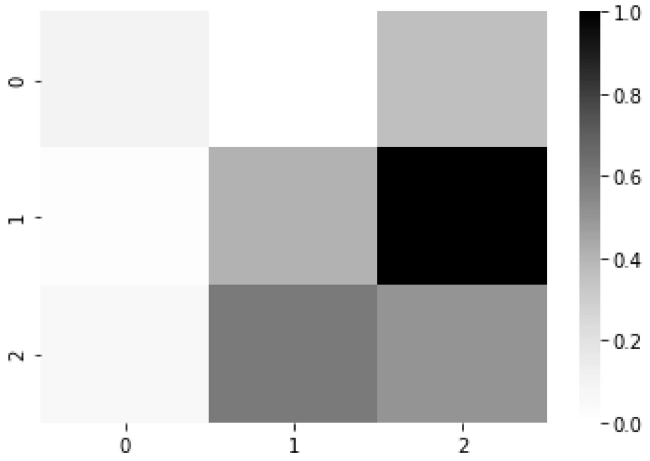
```

```

Imagem Gerada

In [14]:

```
create_image(all_x)
```

```
[[0.09877361 0.         0.36142754]
 [0.02259772 0.41074683 1.        ]
 [0.05466375 0.60339711 0.50752084]]
```



Método da Área

In [15]:

```
m = 12

n = 9

a = [[[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [1.0], [1.0], [1.0]],
     [[0.0], [0.0], [0.0], [1.0], [1.0], [1.0], [0.0], [0.0], [0.0]],
     [[1.0], [1.0], [1.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]],

     [[0.0], [0.0], [0.0425], [0.0], [0.0425], [0.75], [0.0425], [0.75], [0.6125]],
     [[0.0], [0.25], [0.915], [0.25], [0.915], [0.25], [0.915], [0.25], [0.0]],
     [[0.6125], [0.75], [0.0425], [0.75], [0.0425], [0.0], [0.0425], [0.0], [0.0]],

     [[0.0], [0.0], [1.0], [0.0], [0.0], [1.0], [0.0], [0.0], [1.0]],
     [[0.0], [1.0], [0.0], [0.0], [1.0], [0.0], [0.0], [1.0], [0.0]],
     [[1.0], [0.0], [0.0], [1.0], [0.0], [0.0], [1.0], [0.0], [0.0]],

     [[0.0425], [0.75], [0.6125], [0.0], [0.0425], [0.75], [0.0], [0.0], [0.0425]],
     [[0.915], [0.25], [0.0], [0.25], [0.915], [0.25], [0.0], [0.25], [0.915]],
     [[0.0425], [0.0], [0.0], [0.75], [0.0425], [0.0], [0.6125], [0.75], [0.0425]]]
```

In [16]:

```
all_x = calculate(a, b, xt)
```

```
x1 = 2.9974775772355486
x2 = 1.0019748612222354
x3 = 4.0021066831980185
x4 = 1.0073007364146442
x5 = 4.986648885173406
x6 = 9.008179159721545
x7 = 1.996883941864005
x8 = 6.013298884851569
x9 = 4.991376353088403
```

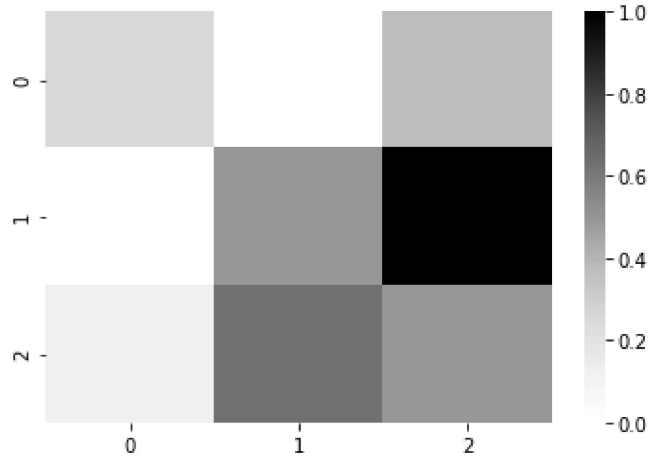
In [ ]:

```

```

In [17]:

```
create_image(all_x)
```

```
[[2.49244541e-01 0.00000000e+00 3.74725864e-01]
 [6.65218497e-04 4.97698269e-01 1.00000000e+00]
 [1.24267261e-01 6.25930071e-01 4.98288745e-01]]
```



Esboço de um algorítmo generalizado para o método do centro do pixel

Conseguimos calcular todos vetores de feixes horizontais e verticais (for 1 e for 2), mas uma fórmula generalizante para as diagonais envolveria o programa saber sobre o tamanho de diversas diagonais, e não conseguimos fazer nada fora a diagonal principal (for 3) e a secundária (for 4) - por isso o número de equações de um quadrado NxN seria 2N + 2, ou seja, o sistema não seria sobredeterminado.

In [18]:

```
def centro(n):
    r = int(n**0.5) # lado do quadrado
    a = [[]]
    K = 2*r + 2 #tamanho da matriz A
    a += [[]] * K

    for i in range(r): # for 1 - setas da esquerda pra direita
        lista1 = [[0]]*n
        for j in range(r):
            lista1[i*r + j] = [1]
        a[i] = lista1

    for i in range(r): #for 2 - setas de cima pra baixo
        lista3 = [[0]]*(n)
        for j in range(r):
            lista3[i + r*j]=[1]
        a[r + i] = lista3

    for i in range(1): # for 3 - setas de esq baixo -> dir cima
        lista2 = [[0]]*n
        for j in range(r):
            lista2[j + j*r] = [1] #comentário 1 *
        a[2*r ] = lista2

    # for 4 - setas de esq baixo -> dir cima
    lista4 = [[0]]*n
    for j in range(1,r+1):
        lista4[j*r - j] = [1] # comentário 2
    a[2*r + 1] = lista4

    return a
```

Comentários 1 e 2 da função acima: a função é a mesma independente do método, exceto que nas nas linhas das diagonais principais (for 3 e for 4), o método da reta central teria o valor de raíz de 2

O método da área é mais complicado pois os feixes das diagonais principais também pegam pixels fora da diagonal princpal em si

Usando esse método para o exercício 3, pelo método do centro do pixel, com uma matriz B reduzida, temos:

In [19]:

```
b = [8, 15, 13, 6, 12, 18, 16.13, 14.31]
m = 8
n = 9
a = centro(9)
```

In [20]:

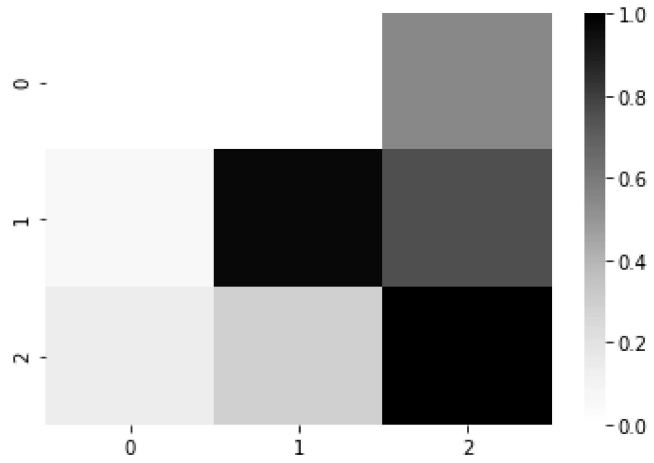
```
all_x = calculate(a, b, xt)
```

```
x1 = 1.6583333336673098
x2 = 1.5933333336892468
x3 = 4.7483333333406454
x4 = 1.9266666670225772
x5 = 7.146666666652043
x6 = 5.9266666670225785
x7 = 2.4150000000073133
x8 = 3.260000000355911
x9 = 7.325000000333977
```

In [21]:

```
create_image(all_x)
```

```
[[0.01134051 0.         0.55045071]
 [0.05815644 0.9688863  0.75603373]
 [0.14335563 0.2907822  1.        ]]
```



Esboço do algorítmo generalizado para o método da reta central

In [22]:

```
def reta_central(n):
    r = int(n**0.5) # lado do quadrado
    a = [[]]
    K = 2*r + 2 # tamanho da matriz A
    a += [[]] * K

    for i in range(r): # for 1 - setas da esquerda pra direita
        lista1 = [[0]]*n
        for j in range(r):
            lista1[i*r + j] = [1]
        a[i] = lista1

    for i in range(r): # for 2 - setas de cima pra baixo
        lista3 = [[0]]*(n)
        for j in range(r):
            lista3[i + r*j]=[1]
        a[r + i] = lista3

    for i in range(1): # for 3 - setas de esq baixo -> dir cima
        lista2 = [[0]]*n
        for j in range(r):
            lista2[j + j*r] = [np.sqrt(2)] # ver comentário 1 *
        a[2*r ] = lista2

    # for 4 - setas de esq baixo -> dir cima
    lista4 = [[0]]*n
    for j in range(1,r+1):
        lista4[j*r - j] = [np.sqrt(2)] # ver comentário 2
    a[2*r + 1] = lista4

    return a
```

In [23]:

```
b = [8, 15, 13, 6, 12, 18, 16.13 , 14.31]
m = 8
n = 9
a = reta_central(9)
```

In [24]:

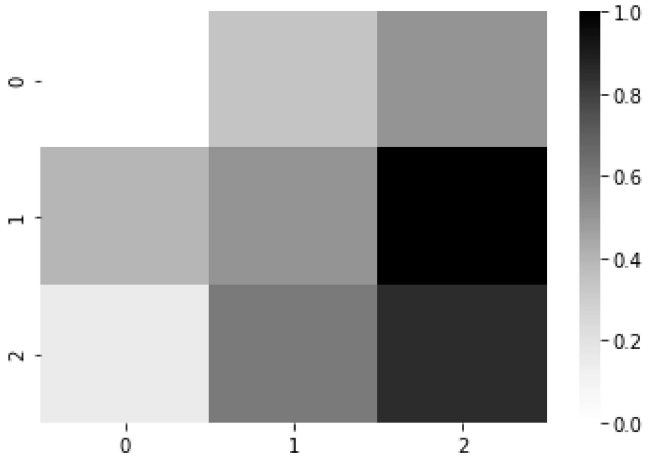
```
all_x = calculate(a, b, xt)
```

```
x1 = 0.7820944532476889
x2 = 3.0792782629497735
x3 = 4.138627282827193
x4 = 3.412611596283107
x5 = 4.174776806458442
x6 = 7.4126115962831065
x7 = 1.8052939494938596
x8 = 4.7459449296164395
x9 = 6.448761119914356
```

In [25]:

```
create_image(all_x)
```

```
[[0.         0.34645621 0.50622489]
 [0.39672881 0.51167688 1.        ]
 [0.1543167  0.5978192  0.85463419]]
```



In [ ]:

```

```

5) Reconstrução da Imagem

In [26]:

```
image = img.imread('ime10x10.png')
```

In [27]:

```
image.shape
```

Out[27]:

```
(10, 10)
```

In [28]:

```
matriz_b_row = {}
for i in range(image.shape[0]):
    matriz_b_row[f'b_{i+1}'] = np.sum(image[i])
```

In [29]:

```
matriz_b_row
```

Out[29]:

```
{'b_1': 10.0,
 'b_2': 10.0,
 'b_3': 3.0,
 'b_4': 3.0,
 'b_5': 4.0,
 'b_6': 6.0,
 'b_7': 5.0,
 'b_8': 10.0,
 'b_9': 10.0,
 'b_10': 10.0}
```

In [30]:

```
image_col_sum = np.sum(image, axis=0)
```

In [31]:

```
image_col_sum
```

Out[31]:

```
array([ 5., 10.,  5.,  8.,  8.,  8.,  5., 10.,  5.,  7.], dtype=float32)
```

In [32]:

```
matriz_b_col = {}
for i in range(image_col_sum.shape[0]):
    matriz_b_col[f'b_{i+11}'] = image_col_sum[i]
```

In [33]:

```
matriz_b_col
```

Out[33]:

```
{'b_11': 5.0,
 'b_12': 10.0,
 'b_13': 5.0,
 'b_14': 8.0,
 'b_15': 8.0,
 'b_16': 8.0,
 'b_17': 5.0,
 'b_18': 10.0,
 'b_19': 5.0,
 'b_20': 7.0}
```

In [ ]:

```

```

In [34]:

```
digonal_list_1 = []
for i in range(image.shape[0]):
    digonal_list_1.append(image[i][i])
```

In [35]:

```
diagonal_1_sum = np.sum(digonal_list_1)
diagonal_1_sum
```

Out[35]:

```
6.0
```

In [36]:

```
digonal_list_2 = []
for i in range(image.shape[0]):
    digonal_list_2.append(image[i][9-i])
```

In [37]:

```
diagonal_2_sum = np.sum(digonal_list_2)
diagonal_2_sum
```

Out[37]:

```
9.0
```

In [38]:

```
matriz_b_total = []
```

In [39]:

```
for value in matriz_b_row.values():
    matriz_b_total.append(value)
```

In [40]:

```
for value in matriz_b_col.values():
    matriz_b_total.append(value)
```

In [41]:

```
matriz_b_total.append(diagonal_1_sum)
matriz_b_total.append(diagonal_2_sum)
```

In [42]:

```
len(matriz_b_total)
```

Out[42]:

```
22
```

In [43]:

```
noise = np.random.normal(-.005, .005, image.shape)
image_noise = np.array(image) + noise
```

In [44]:

```
image_transposta = image.T
```

In [45]:

```
image_transposta_noise = image_transposta + noise
```

In [ ]:

```

```

In [46]:

```
digonal_list_1_noise = []
for i in range(image_noise.shape[0]):
    digonal_list_1_noise.append(image_noise[i][i])
```

In [47]:

```
digonal_list_2_noise = []
for i in range(image_noise.shape[0]):
    digonal_list_2_noise.append(image_noise[i][9-i])
```

In [ ]:

```

```

In [48]:

```
matriz_a = []
for lista in image_noise:
    matriz_a.append(lista)
```

In [49]:

```
for lista in image_transposta_noise:
    matriz_a.append(lista)
```

In [50]:

```
# matriz_a.append(digonal_list_1_noise)
# matriz_a.append(digonal_list_2_noise)
```

In [ ]:

```

```

In [51]:

```
def centro(n):
    r = int(n**0.5) # lado do quadrado
    a = [[]]
    K = 2*r + 2 #tamanho da matriz A
    a += [[]] * K

    for i in range(r): # for 1 - setas da esquerda pra direita
        lista1 = [0]*n
        for j in range(r):
            lista1[i*r + j] = 1
        a[i] = lista1

    for i in range(r): #for 2 - setas de cima pra baixo
        lista3 = [0]*(n)
        for j in range(r):
            lista3[i + r*j] = 1
        a[r + i] = lista3

    for i in range(1): # for 3 - setas de esq baixo -> dir cima
        lista2 = [0]*n
        for j in range(r):
            lista2[j + j*r] = 1 #comentário 1 *
        a[2*r ] = lista2

    # for 4 - setas de esq baixo -> dir cima
    lista4 = [0]*n
    for j in range(1,r+1):
        lista4[j*r - j] = 1 # comentário 2
    a[2*r + 1] = lista4

    return a
```

In [52]:

```
matriz_a2 = centro(100)
```

In [ ]:

```

```

In [53]:

```
print(len(matriz_a2[0]))
```

```
100
```

In [54]:

```
matriz_zero = np.zeros(100)
```

In [55]:

```
m = 22
n = 100
```

In [56]:

```
def rod_v2(a, b, xt): # função de 1 rodada de projeções
    k = 0
    var = [0]*int(m)

    while k < m:
        ai = np.array(a[k])
        at = np.transpose(ai)
        #print(len(b))
        v1 = (b[k] - mult(at,xt))
        v2 = mult(at,ai)
        # print(v2)

        if k==0:
            proj = xt + (v1/v2)*np.transpose(ai)
            xt = np.transpose(proj)

        else:
            proj = np.transpose(xt) + (v1/v2)*np.transpose(ai)
            xt =np.transpose(proj)

        # print(f'proj = {proj}, xt = {xt}, v1/v2 = {v1/v2}')
        # print(xt)
        # print(v1/v2)
        var[k] = proj
        k += 1


    return var
```

In [57]:

```
def calculate_v2(a, b, xt, epslon=10e-10):
    #função para rodar as projeções até aproximar resultados consecutivos
    i = 0
    dif = 1

    while dif > epslon:
        k = rod_v2(a,b,xt)
        xt = k[int(m-1)]
        xt = np.transpose(xt)
        k2 = rod_v2(a,b,xt)
        # print(k[m-1][0])
        dif = abs(k[m-1][0] -  k2[m-1][0])
        # print(dif)
        # break
        # dif = dif[0]
        i +=1
        if i > 100:
            break


    all_x = []
    # print(k2)
    for j in range(n):
        # print(len(k2[0][0]))
        print(f'x_{j+1} =', end = ' ')
        print(k2[m-1][j])
        all_x.append(k2[m-1][j])

    return np.reshape(all_x, (-1, 1))
```

In [ ]:

```

```

In [58]:

```
resultado = calculate_v2(a=matriz_a2, b=matriz_b_total, xt=matriz_zero, epslon=10e-2)
```

```
x_1 = 0.6800000000000002
x_2 = 1.2820000000000003
x_3 = 0.7820000000000003
x_4 = 1.0820000000000003
x_5 = 1.0820000000000003
x_6 = 1.0820000000000003
x_7 = 0.7820000000000003
x_8 = 1.2820000000000003
x_9 = 0.7820000000000003
x_10 = 1.1800000000000004
x_11 = 0.782
x_12 = 1.18
x_13 = 0.782
x_14 = 1.082
x_15 = 1.082
x_16 = 1.082
x_17 = 0.782
x_18 = 1.282
x_19 = 0.9800000000000002
x_20 = 0.982
x_21 = 0.08200000000000002
x_22 = 0.5820000000000001
x_23 = -0.02000000000000003
x_24 = 0.382
x_25 = 0.382
x_26 = 0.382
x_27 = 0.08200000000000002
x_28 = 0.7800000000000002
x_29 = 0.08200000000000002
x_30 = 0.28200000000000003
x_31 = 0.08200000000000002
x_32 = 0.5820000000000001
x_33 = 0.08200000000000002
x_34 = 0.27999999999999997
x_35 = 0.382
x_36 = 0.382
x_37 = 0.28000000000000014
x_38 = 0.5820000000000001
x_39 = 0.08200000000000002
x_40 = 0.28200000000000003
x_41 = 0.18199999999999997
x_42 = 0.6819999999999999
x_43 = 0.18199999999999997
x_44 = 0.48199999999999993
x_45 = 0.3799999999999999
x_46 = 0.68
x_47 = 0.18199999999999997
x_48 = 0.6819999999999999
x_49 = 0.18199999999999997
x_50 = 0.38199999999999995
x_51 = 0.382
x_52 = 0.882
x_53 = 0.382
x_54 = 0.682
x_55 = 0.8800000000000002
x_56 = 0.58
```

```
x_57 = 0.382
x_58 = 0.882
x_59 = 0.382
x_60 = 0.582
x_61 = 0.28200000000000003
x_62 = 0.782
x_63 = 0.28200000000000003
x_64 = 0.7800000000000002
x_65 = 0.5820000000000001
x_66 = 0.5820000000000001
x_67 = 0.18
x_68 = 0.782
x_69 = 0.28200000000000003
x_70 = 0.48200000000000004
x_71 = 0.782
x_72 = 1.282
x_73 = 0.9800000000000002
x_74 = 1.082
x_75 = 1.082
x_76 = 1.082
x_77 = 0.782
x_78 = 1.18
x_79 = 0.782
x_80 = 0.982
x_81 = 0.7819999999999998
x_82 = 1.48
x_83 = 0.7819999999999998
x_84 = 1.0819999999999999
x_85 = 1.0819999999999999
x_86 = 1.0819999999999999
x_87 = 0.7819999999999998
x_88 = 1.2819999999999998
x_89 = 0.6799999999999997
x_90 = 0.9819999999999998
x_91 = 0.9800000000000002
x_92 = 1.282
x_93 = 0.782
x_94 = 1.082
x_95 = 1.082
x_96 = 1.082
x_97 = 0.782
x_98 = 1.282
x_99 = 0.782
x_100 = 0.8799999999999999
```

In [ ]:

In [59]:

```
def create_image_v2(x_list):
    # Abaixo o reescalonamento Min-Max Normalization
    scl = MinMaxScaler(feature_range=(0, 1))
    scl.fit(x_list)
    scaled_x_list = scl.transform(x_list)

    # Tranformando a matriz em 3x3
    scaled_x_list = np.reshape(scaled_x_list, (-1, 10))
    print(scaled_x_list)

    sns.heatmap(scaled_x_list, vmin=0, vmax=1, cmap="Greys")
```

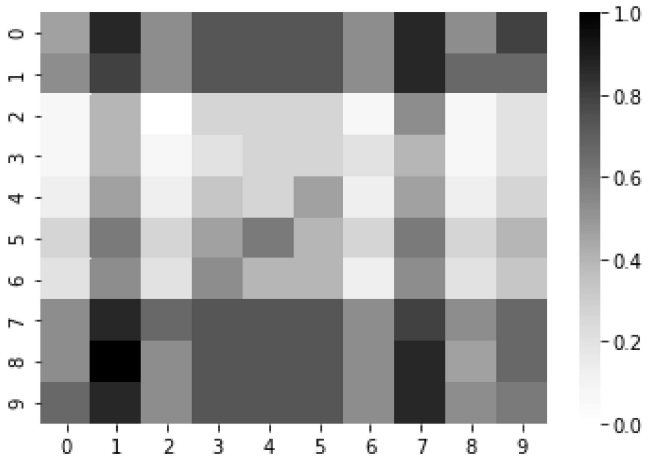
In [ ]:

```
```

In [60]:

```
create_image_v2(resultado)
```

```
[[0.46666667 0.868      0.53466667 0.73466667 0.73466667 0.73466667
  0.53466667 0.868      0.53466667 0.8        ]
 [0.53466667 0.8        0.53466667 0.73466667 0.73466667 0.73466667
  0.53466667 0.868      0.66666667 0.668      ]
 [0.068      0.40133333 0.         0.268      0.268      0.268
  0.068      0.53333333 0.068      0.20133333]
 [0.068      0.40133333 0.068      0.2        0.268      0.268
  0.2        0.40133333 0.068      0.20133333]
 [0.13466667 0.468      0.13466667 0.33466667 0.26666667 0.46666667
  0.13466667 0.468      0.13466667 0.268      ]
 [0.268      0.60133333 0.268      0.468      0.6        0.4
  0.268      0.60133333 0.268      0.40133333]
 [0.20133333 0.53466667 0.20133333 0.53333333 0.40133333 0.40133333
  0.13333333 0.53466667 0.20133333 0.33466667]
 [0.53466667 0.868      0.66666667 0.73466667 0.73466667 0.73466667
  0.53466667 0.8        0.53466667 0.668      ]
 [0.53466667 1.         0.53466667 0.73466667 0.73466667 0.73466667
  0.53466667 0.868      0.46666667 0.668      ]
 [0.66666667 0.868      0.53466667 0.73466667 0.73466667 0.73466667
  0.53466667 0.868      0.53466667 0.6        ]]
```



In [ ]:

In [ ]:

In [ ]:

Referências.

- Anton & Rorres, Álgebra Linear com Aplicações, 10ªed, Bookman, 2012
- Costa, F.M. Agustini, E. , Álgebra Linear e Formação de Imagens: a Tomografia Computadorizada, FAMAT em Revista – Número 05 – Setembro de 2005.