Relatório de Modelo para a Tomografia Computadorizada

Amanda Costa amanda.cs@usp.br
João Pedro de Freitas jpfreitas2001@usp.br
Juan Nogueira juan.nog@usp.br
Maria Rita Xavier mrxavier74@usp.br
Octavio Augusto Potalej oapotalej@usp.br
Samuel Garcez samueltgarcez@usp.br

Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo

O relatório objetiva relacionar a densidade de raios X nos pixels de tomografias, a forma como seus feixes os incidem e a densidade medida no detector através de sistemas lineares que utilizam estes três parâmetros, bem como apresentar um método para encontrar as suas soluções aproximadas.

1 Introdução

O primeiro sistema comercial de tomografia computadorizada para fins médicos foi desenvolvido em 1971 por G. N. Hounsfield. Seu problema básico é construir uma imagem (tomografia) de uma seção transversal do corpo humano usando os dados coletados de milhares de feixes individuais de raios X, da grossura de um fio de cabelo, que são projetados e permanecem no plano da seção transversal que desejamos analisar. Esses dados são processados por um computador e a seção transversal computada é mostrada em sua tela. Dentro do pixel, os fótons que constituem o feixe são absorvidos pelo tecido a uma taxa proporcional a densidade de raios X do tecido e cada pixel é sombreado de acordo com ela. Diferentes tecidos no corpo humano possuem diferentes densidades e, assim, os vários órgãos e tecidos da seção são distinguidos claramente pela imagem. Em 1979, Houndsfield e A. M. Cormack foram laureados com o prêmio Nobel pelo seu trabalho pioneiro na área.

O método para a produzir, que descreveremos através da modelagem, é similar ao utilizado em escaneamentos do cérebro e a construção da seção transversal requer a solução de sistemas lineares enormes. Certos algoritmos, chamados de Técnicas de Reconstrução Algébrica (TRAs), podem ser usados para os resolver e suas soluções produzem a seção transversal em formato digital, como será mostrado.

2 O Modelo

2.1 A tomografia computadorizada

De acordo com Mourão (2015) [1], a palavra tomografia significa imagem em planos, bem como qualquer aparelho que gere imagens de um plano de corte de uma estrutura corporal. Apesar de diversos aparelhos realizarem funções similares à descrita, apenas a Tomografia Computadorizada (TC) apresenta imagens de um corpo por meio da perda gradual do fluxo de energia dos raios X que atravessam seu meio, devido às diferentes características que os tecidos apresentam.

Através desse processo de captação de imagens, é possível notar alterações no volume dos órgãos humanos e, dessa forma, realizar diagnósticos.

Além disso, desde seu surgimento em 1970, o TC já passou por diversas modificações. Devido ao advento do desenvolvimento tecnológico, o aparelho tornou-se mais eficiente em diversos aspectos, como na forma da aquisição das imagens, no tempo de duração do exame, nos diagnósticos que ele pode oferecer etc. Como forma de agrupar essas mudanças de acordo com as características apresentadas, elas foram divididas em cinco gerações. A seguir, serão detalhados os aspectos percentes à quinta geração, a qual é a mais moderna, porém menos utilizada no mercado.

No que tange ao modo de aquisição das imagens, ocorre a geração dos feixes de elétrons estacionários e integrados à arquiteura do próprio sistema de TC, o que faz dele um sistema estático. Dessa forma, a velocidade de aquisição das imagens é muito maior que a dos aparelhos de gerações anteriores. Com isso, o aparelho é ideal para captação de imagens de lugares que se movem em alta velocidade, como nos diagnósticos de anormalidades no coração, por exemplo. Por fim, como a dispersão dos feixes é cônica, o método também é eficaz na captação de imagens da região óssea, já que esse tecido apresenta uma absorção maior dos feixes do que os tecidos moles, e também porque é aplicada uma menor dose de radiação no paciente e, ainda, uma imagem com melhor qualidade é retornada.

2.2 Os parâmetros x, b e a

Em uma tomografia computadorizada tradicional, o paciente se encontra entre a fonte de raios X e o detector dos mesmos, conforme a figura 1, retirada do livro *Algebra Linear com Aplicações* [2], junto a figura 2 a ser vista.

Esse par é transladado e girado a cada medida, até que a quantidade de medidas desejadas seja alcançada.

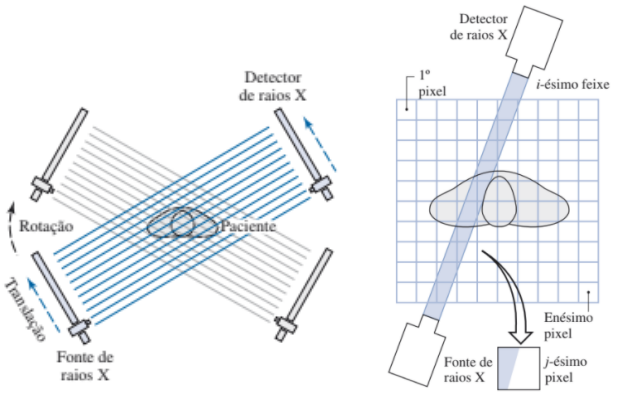

Figura 1

O que queremos é determinar a densidade de raios X de cada pixel, que será sombreado proporcionalmente a ela, em uma escala de cinza, ao ser reproduzido no monitor do computador. Para desenvolver o raciocínio que seguiremos, considere inicialmente um feixe de mesma largura de um pixel e paralelo aos seus lados, como a figura 2 esquematiza.

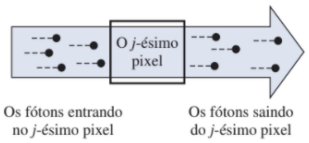

Figura 2

Como já dito, os fótons são absorvidos no pixel a uma taxa proporcional a densidade de raios X do tecido que está nele. Definiremos, assim, $\mathbf{x_j}$ como sendo a densidade de raios X do j-ésimo pixel.

$$x_j = ln(\frac{\text{número de fótons entrando no pixel j}}{\text{número de fótons saindo do pixel j}}) =$$

$$= -ln\left(\begin{array}{c}\text{fração de fótons que passam}\\ \text{pelo pixel j sem ser absorvida}\end{array}\right)$$

Perceba que se o mesmo feixe passar por uma fileira de pixels, o número de fótons saindo de um qualquer é igual ao número de fótons entrando no seguinte. Da enumeração dos pixels desta fileira (1, 2, ... , n), vem:

$$x_1 + x_2 + ... + x_n =$$

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

$$= -ln\left(\begin{array}{c}\text{fração de fótons que passa pela}\\ \text{fileira de pixels sem ser}\\ \text{absorvida}\end{array}\right)$$

Essa soma representa a densidade total de uma fileira de pixels. Definiremos, agora, $\mathbf{b_i}$ como sendo a densidade do i-ésimo feixe.

$$b_i =$$

$$= ln\left(\frac{\text{núm. fótons do feixe i entrando no detector sem paciente}}{\text{núm. fótons do feixe i entrando no detector com paciente}}\right) =$$

$$= -ln\left(\begin{array}{c}\text{fração de fótons do feixe i que}\\ \text{passa pela seção transversal sem}\\ \text{ser absorvida}\end{array}\right)$$

O numerador é obtido realizando uma medida de calibração sem o paciente e essa medida é armazenada no computador. Para cada feixe que passa por uma fileira de pixels, a fração de fótons do feixe que passa sem ser absorvida é igual a fração não detectada. Assim, se o i-ésimo feixe passa por dentro de cada pixel (e paralelo aos seus lados) de uma linha ou coluna de pixels enumerados j_1, j_2, ..., j_i, então temos:

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

Definiremos

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

e reescreveremos a equação, que chamaremos de i-ésima equação de feixe, como:

$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n = b_i$$

Observamos entretanto, como ilustrado na figura 1, que nem todos os feixes são paralelos às linhas ou colunas de pixels. Na verdade eles em geral não o são. Para lidarmos com isso, daremos outras definições de a_{ij} de maneira a reduzir a forma como o feixe atravessa os pixels ao caso que descrevemos anteriormente, onde sua travessia ocorre de forma paralela aos lados.

1) Método do Centro do Pixel (baixa exatidão)

$$a_{ij} = \begin{cases} 1, \text{ se o feixe i passa pelo pixel j} \\ 0, \text{ se isso não acontece} \end{cases}$$

2) Método da Reta Central (média exatidão)

$$a_{ij} = \frac{\text{comprimento da reta central do feixe i que fica no pixel j}}{\text{largura do pixel j}}$$

3) Método da Área (alta exatidão)

$$a_{ij} = \frac{\text{área do feixe i que fica no pixel j}}{\text{área do feixe i que ficaria no pixel j se o feixe i fosse paralelo aos seus lados}}$$

Independente do método que escolhamos usar entre os três, podemos chegar, finalmente, em uma sistema linear de m equações (equações de feixe) e n incógnitas (densidades de pixel), ao qual devido as proporções que pode atingir, utilizaremos métodos especiais (a serem descritos) para aproximarmos a sua solução. Consideraremos, ademais, o caso em que $m > n$, chamado de sobredeterminado.

Uma vez que para cada b há um conjunto de combinações de a_{ij} ($i = 1, ..., m$ e $j = 1, ..., n$), há um conjunto de equações baseadas nestas combinações. O sistema de equações abaixo é dito um Sistema de Equações Lineares.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \\ \vdots \quad \vdots \quad ... \quad \vdots \quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_m \end{cases}$$

2.3 A Técnica de Reconstrução Algébrica

Para solucionar um Sistema de Equações Lineares é possível utilizar diversos métodos matemáticos. Em particular, a abordagem mais adequada para o sistema aqui estudado é o método iterativo conhecido como Técnica de Reconstrução Algébrica. Uma reconstrução iterativa consiste em um processo algorítmico que, a partir de um ponto x_0 (dito iterada inicial), gera sucessivas iteradas x_n de modo tal que a sequência dos valores gerados é $(x_k) \to x^*$, isto é, converge para uma solução x^*.

A opção por esse método se dá em função da própria forma como as equações são obtidas, sujeitas a imprecisão devido a fatores físicos e de medição, de modo a não existir, em geral, um ponto de interseção de todos os hiperplanos. Assim, será possível encontrar a solução aproximada para os valores de x_j, correspondentes às densidades de raios X absorvido no j-ésimo pixel.

No intuito de explicar o mecanismo funcional da TRA, os conceitos de projeção ortogonal e produto interno dentro de um espaço vetorial são fundamentais.

2.3.1 Produto Interno

O produto interno, como uma transformação que utiliza de dois vetores, é um operador simétrico, distributivo, homogêneo e positivo.

Na prática, essas propriedades são os conceitos chave para definir se uma transformação é, ou não, um produto interno.

Dados u e v, dois vetores de $\mathbb{R}^n$, tal que

$$u = (u_1, u_2, ..., u_n) \text{ e } v = (v_1, v_2, ..., v_n)$$

o produto interno definido como

$$< u, v >$$

é a soma dos produtos de suas coordenadas:

$$< u, v >= u_1v_1 + ... + u_nv_n = \sum_{i=1}^{n} u_iv_i$$

A este produto interno dá-se o nome de Produto Interno Euclidiano ou Produto Interno Canônico de $\mathbb{R}^n$.

2.3.2 Ortogonalidade

Dados dois vetores x e y, eles serão ortogonais se, e somente se, o produto interno entre eles for nulo.

$$< x, y >= 0$$

2.3.3 Projeção Ortogonal

A projeção ortogonal é uma transformação linear $F : \mathbb{R}^n \to \mathbb{R}^n$, tal que x é um ponto (ou vetor) qualquer de $\mathbb{R}^n$ e $F(x)$ é o vetor correspondente a ele sobre algum elemento geométrico. Esse elemento pode ser um vetor reta, um outro vetor, um sólido, um plano... entre outros.

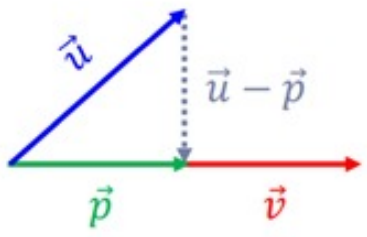

Figura 3

Por definição, se o produto interno existir no espaço vetorial, então a projeção ortogonal existe. A projeção de um vetor pode ser feita sobre outro vetor, reta ou sobre uma superfície, de maneira que, caso se queira projetar um vetor u em um vetor v, sua projeção será um outro vetor p, que será paralelo a v.

Dada a especificação de paralelismo,

$$p = av,$$

a pertence aos reais.

Portanto, se u está projetado sobre v, o produto entre v e a diferença entre u e a projeção p, $< v, u - p >$ é nulo. E além disso, o vetor p corresponde ao vetor v multiplicado por um escalar real.

Assim, podemos equacionar:

$$< u, v >$$

$$< u - av, av >= 0 \Rightarrow$$

$$\Rightarrow < u - av, av >=< u, av > - < av, av >\Rightarrow$$

$$\Rightarrow a < u, v > -a^2 < v, v >= 0$$

E, daqui:

$$a =< u, v > / < v, v >$$

em que a é o escalar que multiplica v para gerar o vetor p (projeção).

Portanto, a projeção de u em v, $p = av$ leva à formula da projeção ortogonal:

$$Proj_v u = v. < u, v > / < v, v >,$$

Ou, em notação matricial:

$$Proj_{\mathbf{v}}\mathbf{v} = \mathbf{w}$$
$$\text{w} = \text{v} \ . \ \text{u}^t\mathbf{v}(\mathbf{v}^t\mathbf{v})^{-1}$$

Definindo:

$$\boldsymbol{a}_i = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \quad \text{e} \quad \boldsymbol{x}_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{in} \end{bmatrix},$$

com $i = 1, \ldots, n$.

Definimos a equação geral de um hiperplano em $\mathbb{R}^n$ atribuindo a notação do problema:

$$a_i^t \cdot x_i = b$$

Seja z um vetor qualquer de Rn e z_p sua projeção ortogonal sobre o hiperplano e c^t um vetor ortogonal ao hiperplano

$$c^t \cdot x = b$$

,
A diferença entre os vetores z e z_p é igual à diferença entre suas projeções sobre o vetor ortogonal.

$$z - z_p = proj_{c^t} z - proj_{c^t} z_p$$

$$z - z_p = \frac{c^t z}{c^t c} c - \frac{c^t z_p}{c^t c} c$$

$$z - z_p = \left(\frac{c^t z}{c^t c} - \frac{c^t z_p}{c^t c} \right) c$$

Mas, uma vez que z_p é a projeção de z sobre o hiperplano, z_p pertence a ele.
Então $c^t z_p = b$.

$$z - z_p = \left(\frac{c^t z}{c^t c} - \frac{b}{c^t c} \right) c$$

E, isolando o vetor z_p:

$$z_p = z + \frac{(b - c^t z)}{c^t c} c$$

Que é a projeção do vetor z, dados z e a equação do hiperplano.

2.3.4 O Algoritmo

Admitimos um ponto incial x_0 qualquer - em geral será a origem, de entradas nulas.

A partir dele, calculamos sua projeção sobre $a_1^t \cdot x = b_1$, o hiperplano da primeira equação. Como visto acima, essa projeção é dada por:

$$x_1^1 = x_0 + \frac{b_1 - a_1^t x_0}{a_1^t a_1} a_1.$$

A fim de evitar confundir com o x_1 definido na seção 2.2, utilizou-se o sobrescrito 1 (em breve, porém, ele terá um sentido próprio). Agora, tendo x_1^1, fazemos a projeção sobre o hiperplano $a_2^t \cdot x = b_2$ da segunda equação. Isto é, obtemos o ponto

$$x_2^1 = x_1^1 + \frac{b_2 - a_2^t x_1^1}{a_2^t a_2} a_2.$$

O processo deverá ser repetido, sempre obtendo x_{i+1}^1 a partir de x_i^1, a e b, até calcular x_m^1. Na sequência, x_m^1 deverá ser projetado novamente no hiperplano 1 $a_1^t \cdot x = b_1$ e esta projeção será chamada x_1^2. Repare o significado do sobrescrito: x_1^2 é a iterada obtida no segundo ciclo sobre o hiperplano 1. Dando segmento à iteração, calcula-se x_2^2, x_3^2, ..., x_m^2, x_1^3, x_2^3 ... e assim consecutivamente.

As sucessivas projeções, portanto, geram m sequências:

$$\begin{cases} x_1^1, x_1^2, x_1^3, x_1^4, ... \\ x_2^1, x_2^2, x_2^3, x_2^4, ... \\ x_3^1, x_3^2, x_3^3, x_3^4, ... \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \ddots \\ x_m^1, x_m^2, x_m^3, x_m^4, ... \end{cases}$$

Cada uma dessas sequências converge. A prova desta afirmação foge ao escopo deste trabalho. Por hora, temos $(x_i) \to x_i^*$, onde $i = 1, 2, ..., m$. Todos esses m valores iterados são aproximações satisfatórias para a solução do sistema.

3 O Programa

A seguir, demonstra-se o trabalho computacional utilizado para aplicar o algoritmo e gerar imagens a partir dele. Na prática, a reconstrução de imagem através da tomografia demanda certo poder computacional para ser realizada. O feito para o problema apresentado é dividido em alguns pequenos processos, cuja união compõe, apesar de suas limitações, uma verdadeira simulação de tomografia.

3.1 Ferramentas de uso geral

Num primeiro momento, faz-se importante compreender quais são e como funcionam as ferramentas que são utilizadas no decorrer do processo de simulação. Estas se dividem em três grandes áreas: o cálculo de X e a simulação dos A.

3.1.1 Densidade de Pixel X

O algoritmo descrito em 2.3.4 é capaz de calcular x utilizando a matriz A de elementos a_{ij}, a matriz B de b_i e a fórmula de projeção. Sendo assim, o cálculo das densidades x_i é realizado dentro do método `densidade()` da classe `Tomografia` e itera os valores para cada x_i conforme uma pré-determinada quantidade de ciclos de iterações. O método `densidade(a, b, converter)` recebe como parâmetros, A, B e *converter*, onde *converter* é um valor boolean que determina se a densidade do x vai ser

retornada em forma de fração, ou em seu valor real. O ponto x_0 é criada pela própria função e sempre será um vetor em que todos os valores serão zero.

Para aumentar a precisão do algoritmo, escolhemos que ele irá repetir 50 ciclos, e assim programamos o algoritmo da seguinte maneira:

```
def densidade(self, a, b, converter):

        # Cria uma variável x0, que será o ponto
        ↪ inicial.

        x0 = []

        for i in range(len(a[0])):
            x0.append([0])

        # Execução da fórmula apresentada no artigo

        x = []

        for k in range(50): # Estrutura de repetição
        ↪ para aumentar a precisão dos resultados

            x.clear() # Limpa a lista para que
            ↪ somente o ultimo resultado seja
            ↪ salvo

            for i in range(len(a)): # Estrutura de
            ↪ repetição que resolve o sistema de n
            ↪ icógnitas e m equações usando
            ↪ projeção ortogonal

                # Conversão das listas para matrizes
                ↪ numpy

            x0 = np.asmatrix(x0)
            a[i] = np.matrix(a[i])

            if(i == 0): # caso esteja na primeira
            ↪ repetição, o algorítmo usará o x0 no
            ↪ lugar do x[i-1]

                if((np.transpose(a[i]) *
                ↪ np.asmatrix(a[i]) ) != 0):
                    aux = x0 + a[i] * (
                    ↪ (b[i]-(np.transpose(a[i]) *
                    ↪ x0)) / (np.transpose(a[i]) *
                    ↪ np.asmatrix(a[i]) ) )
                    x.append(aux)

                else:
                    x.append(0)

            else:
                if((np.transpose(a[i]) *
                ↪ np.asmatrix(a[i]) ) != 0):

                    aux = x[i-1] + a[i] * (
                    ↪ (b[i]-(np.transpose(a[i]) *
                    ↪ x[i-1])) /
                    ↪ (np.transpose(a[i]) *
                    ↪ np.asmatrix(a[i]) ) )
                    x.append(aux)

                else:
                    x.append(0)

            if(i == (len(a)-1)):
                x0 = x[i]
```

O resultado obtido será x_m^{50}, que é uma aproximação satisfatória para os valores de x.

3.1.2 Feixes A

Para a simulação de feixes, o algoritmo utilizado se baseia numa interpretação geométrica analítica da situação, compreendendo que cada pixel p considerado tem, como ponto central, um ponto no no plano cartesiano (x_p, y_p), e este, sendo um quadrado, tem seus vértices, dado seu tamanho determinado t, em $(x_p-\frac{t}{2}, y_p-\frac{t}{2})$, $(x_p+\frac{t}{2}, y_p-\frac{t}{2})$, $(x_p-\frac{t}{2}, y_p+\frac{t}{2})$ e $(x_p+\frac{t}{2}, y_p+\frac{t}{2})$.

Desse modo, independente de qual dos três métodos considerados for escolhido, um feixe de reta $f : \mathbb{R} \to \mathbb{R}$ passa sobre p se, e somente se, $\exists x_0, x_1 \in [x_p - \frac{t}{2}, x_p + \frac{t}{2}] \mid f(x_0), f(x_1) \in [y_p - \frac{t}{2}, y_p + \frac{t}{2}]$ (considerando também que x_0 e x_1 não necessariamente são diferentes, como é o caso de um feixe atravessar exatamente e apenas um vértice do pixel).

Eventualmente se utilizam retas auxiliares, que compõem um feixe propriamente dito. Assim, considerando que o feixe detém um tamanho T_p, onde sua reta central de reta f possui duas retas auxiliares f_1 e f_2, estando ambas a $\frac{T_p}{2}$ unidades de distância de f, um para cada lado, encontrar as equações de reta das auxiliares depende unicamente dessa distância.

Medir a distância d entre duas retas paralelas g e h, onde estas assumem seu formato geral, ou seja, $g : ax+by+c_g = 0$ e $h : ax + by + c_h = 0$, se resume a calcular, conforme a geometria analítica:

$$d = \frac{\mid c_h - c_g \mid}{\sqrt{a^2 + b^2}}$$

Seja a reta $y = mx + n$, seu formato geral será $-mx + y - n$, ou seja, se terá, para g, sendo m seu coeficiente angular e n_g, respectivamente para h também, que $g : -mx + y - n_g = 0$ e $h : -mx + y - n_g = 0$. Substituindo e encontrando assim uma expressão geral:

$$d = \frac{\mid n_g - n_h \mid}{\sqrt{m^2 + 1}}$$

Sendo $d = \frac{T}{2}$, é possível escrever:

$$\mid n_g - n_h \mid = \frac{T}{2}\sqrt{m^2 + 1}$$

Considerando o caso da reta central f e suas auxiliares, é fácil verificar que estas terão coeficiente linear sendo os resultados positivos e negativos do módulo, ou seja:

$$n_1 = n_f + \frac{T}{2}\sqrt{m^2 + 1}$$

$$n_2 = -n_f + \frac{T}{2}\sqrt{m^2 + 1}$$

Tendo, portanto, as equações para as três retas:

$$f(x) = mx + n_f$$

$$f_1(x) = mx + n_1 = mx + n_f + \frac{T}{2}\sqrt{m^2 + 1}$$

$$f_2(x) = mx + n_2 = mx - n_f + \frac{T}{2}\sqrt{m^2 + 1}$$

De resto, o algoritmo utilizado para cada método depende do que cada um pretende. De forma geral, há o uso

destas retas auxiliares nos métodos do Centro do Pixel (onde se busca verificar se o centro se localiza no intervalo entre f_1 e f_2) e no da Área (onde é buscado o cálculo da área ocupada por uma reta em relação à outra dentro do pixel). No caso da Reta Central, se utiliza apenas a reta f, se calculando onde f entra e sai em p, tendo então a razão da distância cartesiana entre estes dois pontos pelo comprimento do pixel (T).

O código-fonte dos algoritmos utilizados pode ser encontrado aqui, na classe `Metodos`.

4 Aplicação no exemplo 3x3

O caso simples de uma malha de dimensões 3x3 deve elucidar como se dão as operações apresentadas até aqui. Considere os dados da figura abaixo com os valores de b e os pixels atravessados por cada feixe.

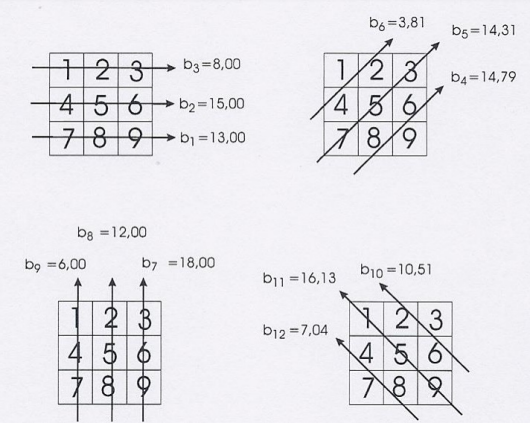

Figura 4

4.1 Montagem do sistema linear

Conforme a seção 2.2.2, aplicando ao caso $m = 12$ e $n = 9$, esses feixes dão origem ao sistema

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + ... + a_{1,9}x_9 = 13 \\ a_{2,1}x_1 + a_{2,2}x_2 + ... + a_{2,9}x_9 = 15 \\ \quad \vdots \qquad \vdots \qquad ... \qquad \vdots \qquad \vdots \\ a_{12,1}x_1 + a_{12,2}x_2 + ... + a_{12,9}x_9 = 7,04 \end{cases}$$

em que os valores a_{ij} dependem do método de cálculo utilizado. Com isso em vista, o programa construído em `tarefa3.py` pergunta ao usuário qual dos três métodos deseja executar.

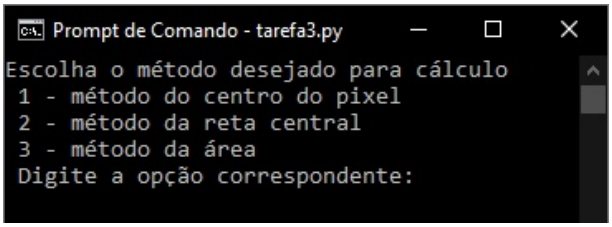

Figura 5

Para descrever o sistema linear, de qualquer método, basta utilizar o programa para visualizar a matriz A correspondente. A título de exemplo, neste relatório estará exibido a seguir o sistema alcançado pelo método da reta central.

$$\begin{cases} x_7 + x_8 + x_9 = 13 \\ x_4 + x_5 + x_6 = 15 \\ x_1 + x_2 + x_3 = 8 \\ 0,707x_6 + 0,707x_8 + 0,707x_9 = 14,79 \\ 1,414x_3 + 1,414x_5 + 1,414x_7 = 14,31 \\ 0,707x_1 + 0,707x_2 + 0,707x_4 = 3,81 \\ x_3 + x_6 + x_9 = 18 \\ x_2 + x_5 + x_8 = 12 \\ x_1 + x_4 + x_7 = 6 \\ 0,707x_2 + 0,707x_3 + 0,707x_6 = 10,51 \\ 1,414x_1 + 1,414x_5 + 1,414x_9 = 16,13 \\ 0,707x_4 + 0,707x_7 + 0,707x_8 = 7,04 \end{cases}$$

Nota-se claramente os significados geométricos que os valores de A carregam. A ver, os coeficientes da quinta equação $a_{5i} = 1,414$ representam a medida da diagonal de cada pixel ($\sqrt{2}$), que é justamente o segmento do feixe b_5 que é interno aos pixels x_3, x_5 e x_7.

4.2 Cálculo de densidade dos pixels

Imediatamente após a obtenção de A, o programa roda o algoritmo de iteração previamente exposto e calcula as densidades x_i. Repare que, com 12 equações, o programa exibirá como solução o valor que encontrar em x_{12}^{50}.

Podemos comparar os valores da densidade obtidos pelos três métodos:

Pelo Método do Centro:

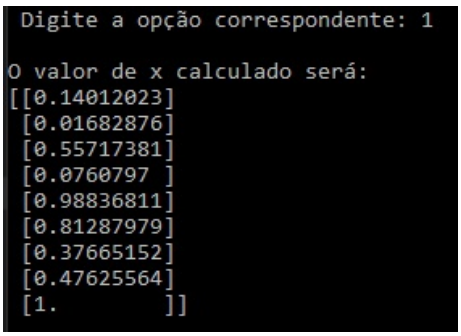

Figura 6

Pelo Método da Reta Central:

```
Digite a opção correspondente: 2

O valor de x calculado será:
[[0.01408667]
 [0.22554986]
 [0.40877041]
 [0.20371845]
 [0.45387088]
 [1.        ]
 [0.24271077]
 [0.64830102]
 [0.78615723]]
```

Figura 7

Pelo Método da Área:

```
Digite a opção correspondente: 3

O valor de x calculado será:
[[0.12004397]
 [0.40703858]
 [0.4135304 ]
 [0.18829618]
 [0.3258553 ]
 [1.        ]
 [0.39310338]
 [0.55414176]
 [0.92593068]]
```

Figura 8

4.3 Exibição da imagem

Por fim, o programa retorna ao usuário a imagem correspondente aos valores x_i encontrados. Novamente, podemos comparar as imagens oriundas de cada método:

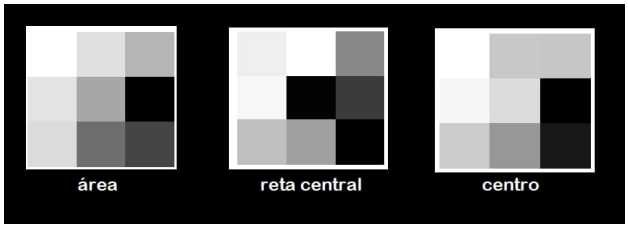

Figura 9

5 Para telas maiores

O processo de reconstrução completa de uma imagem a partir dela mesmo, se utilizando das ferramentas e transformações apresentadas, traz margens de erros, pois a simulação de um tomógrafo, principalmente se utilizando dos métodos apresentados, é problemática e carece de precisão. Para telas com poucos pixels, não há tanta distorção, mas para telas grandes o problema se mostra grande quanto. Analisemos o porquê.

5.1 Replicando o método conhecido

Uma tomografia que possua n pixels terá n^2 variáveis. Nessas condições, se faz suficiente utilizar apenas 4 direções de feixes para uma tela 3×3, pois haverão, se passados exatamente 3 retas em cada direção, 12 equações, que é um número maior que a quantidade de variáveis e portanto se trata de um sistema solucionável.

Porém, para matrizes grandes $n \times n$, são necessários ao menos n direções com n feixes cada para que se tenha a mesma quantidade de equações que variáveis.

O algoritmo utilizado foi capaz de gerar retas em quatro direções, o suficiente para o exemplo anterior, mas insuficiente para problemas maiores. Ainda assim, é interessante analisar teoricamente como pode ser um algoritmo mais preciso.

Como apresentado, os raios são descritos como retas. Nesse sentido, faz-se necessários automatizar o equacionamento de retas conforme uma certa quantidade de direções, de modo a se capturar todas as variáveis em um número igual ou maior de equações.

Ao tentar simular um tomógrafo real utilizando o programa, os problemas são mais evidentes. Ao pegar uma imagem nxn, podemos separá-la em pixels através das funções desenvolvidas e achar a densidade da cada um deles. Após isso, simular feixes e encontrar valores de a que nos permita chegar à uma imagem. Como discutido, a dificuldade maior reside em simular uma quantidade suficientemente grande de feixes e computar os respectivos pesos a.

Uma vez obtidos x e a, restaria calcular b. Tal operação é a mais simples, pois se resume à própria definição: dados A e X, B é o resultado da equação de cada feixe, ou seja:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ \vdots \quad \vdots \quad \ldots \quad \vdots \quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \end{cases}$$

Seu cálculo é realizado pela função `calcularB()` da seguinte forma:

```
def calcularB(A, X):
    b = [ ]
    for linha in A:
        soma = 0
        for i in range(len(linha)):
            soma += linha[i][0]*X[i]
        b.append(soma)
    return b
```

Agora que temos a e b, tentamos calcular novamente os valores de x, utilizando a função `densidade(a, b)`. Ocorre que, com um valor muito grande de n, é muito provável que o computador do usuário trave e precise ser reiniciado, o que nos limita a resoluções de imagem menores. No entanto, mesmo para imagens com poucas dezenas de dimensão, o resultado é insatisfatório, o que frustra nossa tentativa de aplicar o método em imagens reais . Como já colocado, devido à baixa quantidade de feixes simulados, não temos suficientes valores de a e a imagem fica irreconhecível.

Aqui temos uma imagem 32x32:

Figura 10

Aqui temos ela depois de sua reconstrução com 32 e 363 feixes, respectivamente:

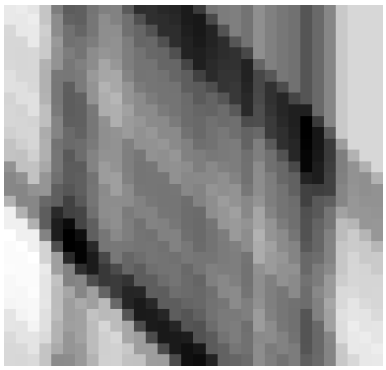

Figura 11

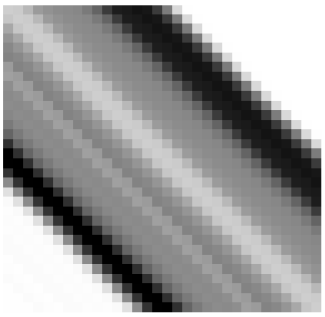

Figura 12

O algoritmo completo pode ser acessado *neste diretório*.

5.2 Uma nova abordagem

Apresentaremos um novo método capaz de simular as imagens obtidas em um tomógrafo de maneira mais precisa.

5.2.1 Transformações de Imagem

Dadas as ferramentas utilizadas em 3.1, é possível realizar a transformação de dados para a imagem e vice-versa. Os cálculos já foram apresentados, e resta tão somente analisar o processo de captura de dados a partir de uma imagem e seu inverso.

5.2.2 De imagem à informação

O processo de converter uma imagem em dados utilizáveis foi realizado através das bibliotecas `Numpy` e `PIL`. A primeira tem seu propósito em cálculos numéricos trabalhosos, mas que são realizados de forma bastante otimizados, e que inclusive foi utilizada no método da área para A, especificamente no caso em que eram considerados três pontos, tendo a área como a metade da determinante destes pontos. A segunda, porém, tem seu propósito em transformar imagens em objetos do Python, de modo que se possa trabalhar com estes matematicamente.

Assim, é possível obter os dados de uma imagem da seguinte forma:

```
from Numpy import asarray
from PIL import Image

image = Image.open(imagem)
data = asarray(image)
```

A variável `data` recebe um array de dados, sendo estes informações em RGB (*Red, Green, Blue*) ou RGBA (*Red, Green, Blue, Alpha*, onde o último se trata de um valor entre 0 e 1 para a opacidade de imagem). Não é possível fazer um trabalho grande com este array sem que haja uma devida conversão:

```
matriz = []
for i in data:
    l = [ ]
    for j in i:
        try:
            a = j[3]
        except:
            a = 1
        soma = 0
        if a != 0:
            for k in range(j):
                soma += k
            soma -= a
            soma /= 765
        l.append(soma)
    matriz.append(l)
```

A divisão por 765 se deve à soma do valor máximo de um RGB (255, 255, 255), assim podendo converter uma lista de três elementos em um valor entre 0 e 1, em escala de cinza.

A variável `matriz`, assim, corresponde ao vetor de densidades, ou seja, X, podendo ser útil no cálculo de B.

5.2.3 De informação à imagem

O processo inverso, de ter os dados e formar uma imagem, não se faz tão complexo. É utilizada a biblioteca `matplotlib`, e este processo se resume a estabelecer que um dado ponto de uma tabela detém determinada cor de fundo, sendo esta cor conforme o vetor de densidades X.

```
import matplotlib.pyplot as plt
def exibir(matriz):
    plt.imshow(matriz, aspect='auto', cmap="Greys")
    plt.axis('off')
    plt.show()
```

5.2.4 Exemplos

Aqui temos uma imagem 80x80:

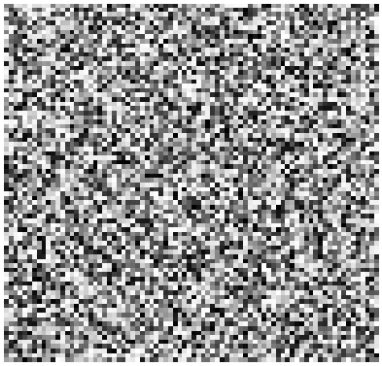

Figura 13

E aqui temos a reconstrução dessa imagem:

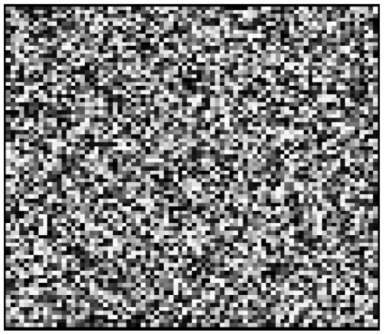

Figura 14

Agora uma imagem 512x512:

Figura 15

E sua reconstrução:

Figura 16

Referências

[1] A. P. Mourã, *Tomografia computadorizada: Tecnologias e aplicações.*

[2] C. Anton, Howard. Rorres, *Álgebra Linear com Aplicações.* bookman, 2012.