Summer 2016

Geometric-Based Algorithm for a Full Row-Rank System Matrix Along Multiple Directions in DT

Igor Lutsenko

GEOMETRIC-BASED ALGORITHM FOR A FULL ROW-RANK SYSTEM MATRIX ALONG MULTIPLE DIRECTIONS IN DT

by

IGOR LUTSENKO

(Under the Direction of Jiehua Zhu)

ABSTRACT

Discrete tomography (DT) is an image reconstruction procedure that deals with computational synthesis of a cross-sectional image of an object from either transmission or reflection data collected by penetrating an object with X-rays from a small number of different directions, and whose range of the underlying function is discrete. Image reconstruction using algebraic approach is time consuming and the computation cost depends on the size of the system matrix. More scanning directions provide an increase in the reconstructed image quality, however they increase the size of the system matrix dramatically. Deletion of linearly dependent rows of this matrix is necessary to reduce computational cost, and is sometimes a requirement for certain reconstruction software. A geometric-based algorithm is derived in this study that will remove linearly dependent rows of the system matrix generated along an arbitrary number of scanning directions. Numerical experiments indicate that the proposed algorithm reduces the system matrix to a full row-rank.

Key Words: Discrete Tomography, Strip-Based Projection, Line-Based Projection, Full Row-Rank Matrix

2009 *Mathematics Subject Classification*: 92C55

GEOMETRIC-BASED ALGORITHM FOR A FULL ROW-RANK SYSTEM MATRIX ALONG MULTIPLE DIRECTIONS IN DT

by

IGOR LUTSENKO

B.S., Valdosta State University, 2013

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment

of the Requirement for the Degree

MASTER OF SCIENCE

STATESBORO, GEORGIA

GEOMETRIC-BASED ALGORITHM FOR A FULL ROW-RANK SYSTEM MATRIX ALONG MULTIPLE DIRECTIONS IN DT

by

IGOR LUTSENKO

Major Professor: Jiehua Zhu

Committee: Xiezhang Li

Yan Wu

Electronic Version Approved:

July 2016

ACKNOWLEDGMENTS

I wish to thank my advisor, Dr. Jiehua Zhu, for her direction and aid in this project, and my thesis defense committee members, Dr. Xiezhang Li and Dr. Yan Wu. I also wish to thank the entire Department of Mathematical Sciences for their support and help. Finally, I wish to extend my gratitude toward my parents who have provided me with this opportunity to study abroad.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Discrete Tomography (DT)

Computed tomography (CT) is an image reconstruction procedure that deals with computational synthesis of a cross-sectional image of an object from either transmission or reflection data collected by penetrating an object with waves from numerous different directions. The discovery of CT techniques in 1970s has had a revolutionary impact in diagnostic medicine, as it enables health providers to view internal organs, or bone fractures, with groundbreaking precision and safety to the patient. CT′s ability to take non-invasive photographs has been popular in various industrial applications, besides medical imaging, where cutting an object may ruin its value. For example, CT is often used to provide non-destructive evaluations for discovering manufacturing flaws in parts before their use, resulting in greater reliability and greater safety for workers; to identify the presence and facilitate the recovery/extraction of natural resources, such as oil, coal, gas, etc; to provide non-destructive testing and quality control of fresh fruits and vegetable, enhancing the safety of food. Generally, CT involves X-rays for forming images of tissues based on their X-ray attenuation coefficient, which we will use in this study. More recent developments in medical imaging has also been successfully accomplished with radioisotopes, ultrasound, and magnetic resonance, though the imaged parameter being different in each case [4].

Consider an X-ray source which transmits X-ray photons through an object towards a digital detector that records the amount of photons that have passed through that object and were not absorbed. The X-ray source moves around the object to collect transmission data from numerous view points. The fundamental assumption of CT is that there exists an unknown underlying function f that describes the X-ray attenuation of the image of an object. The main goal of CT is to reconstruct this

function using the projection data collected. Line integrals are the most common measures for these data, which are collectively know as projections.

The algorithms for reconstruction of the image from these projections are traditionally developed for a real valued range of the underlying function. However, the CT reconstruction algorithms are not appropriate in the cases where the underlying function's range is discrete, though the domain could be discrete or continuous [3], hence Discrete Tomography (DT) has been developed. The goals of discrete tomography, as we see in the field, have to do with obtaining f, perhaps only partially or approximately, from weighted sums over subsets of its domain in the discrete case and from wighted integrals over subspaces of its domain in the continuous case. In many applications these sums or integrals are known only approximately. Becasue of this, the most essential aspect of discrete tomography is that knowing the discrete range of f may allow us to its value points where this could not have been done without this knowledge. In addition, the knowledge of discrete range of the function may allow for it to be determined from less data than what are necessary for general functions. DT is full of interesting questions and has its own theory and reconstruction algorithms in many fascinating applications [9]. The name, Discrete Tomography, is due to Larry Shepp, who organized the first meeting devoted to the topic, which was help in 1994. There are many publications on the development of reconstruction methods in DT, the first being a reconstruction of binary images proposed by Ryser [8] and Gale [1] independently. The reconstruction methods are usually reduced to a formulation of an optimization problem. An important difference between DT and conventional tomography, such as CT, is the fact that only few projections are sufficient enough for a high quality image reconstruction.

1.2 Line-Based Projection Model

Different algebraic approaches for discrete tomography are used to model the raw detector reading, or the projection data. The most commonly used model is the line-based [9] projection model. In this model the projection equations are constructed along X-rays considered as integral lines, and the equation for each line is defined by the quantity of pixels along the line.

For the line-based projection model, let Ω be a $N \times N$ integer lattice set, with N^2 square cells. It can be described by a set of points $\Omega = \{(i,j) : 1 \leqslant i,j \leqslant N\}$. The index of each cell is identified by the coordinate of the top right corner of the cell. A binary image is thought of as a binary function f defined on Ω, where $f(i,j)$ is the value of f for the $(i,j)^{\text{th}}$ cell. Suppose that the N^2 cells are arranged in the order from bottom to top column-by-column and the corresponding one-dimensional vector $\mathbf{u}$ representing the unknown image values is given by:

$$\mathbf{u} = [f(1,1), ..., f(1,N), f(2,1)..., f(2,N), ..., f(N,1), ..., f(N,N)]^T. \tag{1.1}$$

A set of directions $\{(X_d, Y_d)\}_{d=1}^n$ in the form of $(1,0), (0,1), (1,\pm 1), (q,\pm p)$, or $(p,\pm q)$, where p, q are co-prime, s.t. $gcd(p,q) = 1$ and $1 \leqslant q < p$. In addition, N is a multiple of pq. The parallel lines in a direction (X_d, Y_d) can be characterized by:

$$X_d y = Y_d x + t, \qquad t \in \mathbb{Z}. \tag{1.2}$$

A system generated by a line-based projection model along a set of directions $\{(X_d, Y_d)\}_{d=1}^n$ is denoted by

$$\sum_{X_d y = Y_d x + t} f(i,j) = h_{d,t}, \quad t \in \mathbb{Z}, d = 1, 2, ..., n, \quad \text{or} \quad M\mathbf{u} = \mathbf{h}, \tag{1.3}$$

where $f(i,j) = 0$ for $(i,j) \notin \Omega$ and $h_{d,t}$ is the number of points which are crossed by the line $X_d y = Y_d x + t$ at exactly top right corner. M is a binary matrix and vector $\mathbf{h}$ is the projection data.

As an example for the line-based projection model, consider the scanning of a binary image with size $N = 6$ along one direction $(X_1, Y_1) = (3, -2)$, as shown in Fig. 1.1. After examining line $t = 20$, it is evident from the figure that this line crosses cells $(1, 6)$ and $(4, 4)$ at exactly top right hand corner. Thus the equation for the line $t = 20$ becomes

$$1 * f(1, 6) + 1 * f(4, 4) = h_{20}.$$

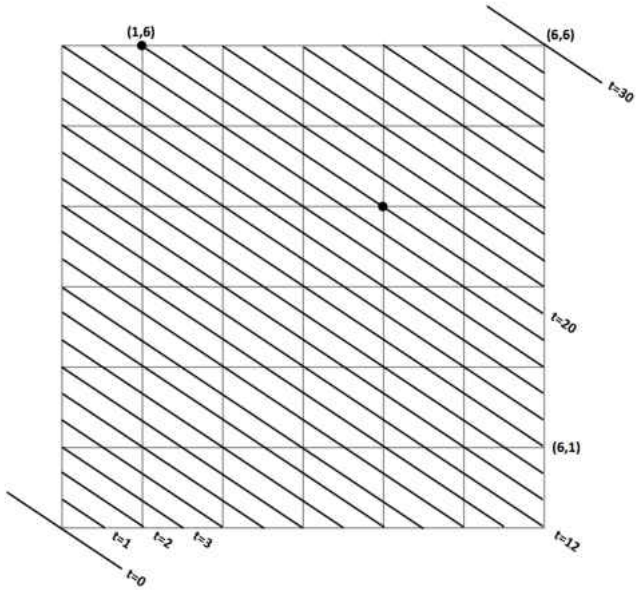


Figure 1.1: Line-Based Projection Model Scanning

A full matrix M in Eq. (1.3), consisting of all 30 equations, generated by the line-based projection model along this direction, is shown in the Fig. 1.2. Equations along a positive direction can be achieved in an analogous way.

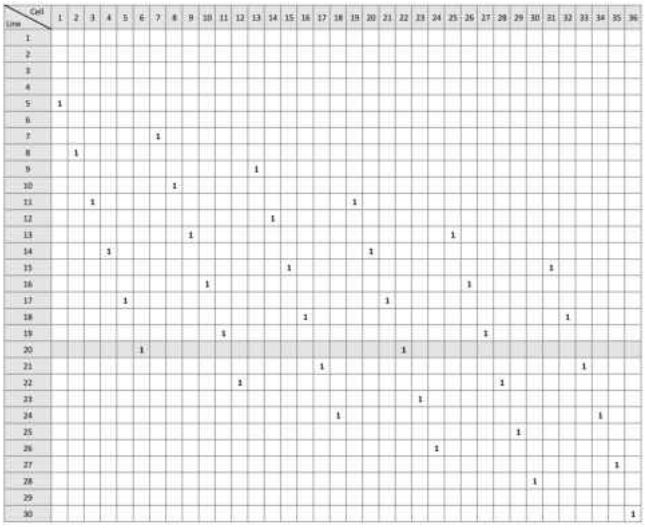


Figure 1.2: Line-Based Projection Model System Matrix

A DT problem here is to find an integer solution f to the system in Eq (1.3). Because the X-ray beams of finite width are reduced to integral lines, the solution to this system requires a major approximation when the data $\{h_{d,t}\}$ is obtained [9].

1.3 Strip-Based Projection Model

The strip-based projection model allows us to work with the actual finite width of the X-rays, as it takes into account the fractional area of each cell through which the X-ray beam passes. Thus the strip-based projection model fits the real projection data and is closer to reality, in some applications, than the line-based projection model. It avoids the error between the theoretical model and the real projection data originating in the line-based projection model. The collected data of the X-ray detector in the strip-based projection model represents the sum of the gray levels corresponding to square cells belonging to the strip [9].

The lattice set Ω can be viewed as a set of unit square cells, indexed by non-negative integers, where the $(i,j)^{\text{th}}$ cell is enclosed between the lines: $x = i-1, x = i, y = j-1$, and $y = j$. A binary image can also be viewed as a binary function $f(i,j)$ defined on the cells of Ω. The two representations of Ω are in one-to-one correspondance to each other [9]. Let $S_{d,t}$ be the strip between the consecutive parallel lines: $X_d\, y = Y_d\, x+t-1$ and $X_d\, y = Y_d\, x+t$. Also let $u_{d,t}(i,j)$ represent the fractional area between the strip $s_{d,t}$ and the $(i,j)^{\text{th}}$ cell. Thus, the system generated by the strip-based projection model along a set of directions $\{(X_d, Y_d)\}_{d=1}^{n}$ can be expressed as:

$$\sum_{1 \leqslant i,j \leqslant N} \mu_{d,t}(i,j) f(i,j) = k_{d,t}, \;\; t \in \mathbb{Z}, d = 1,2,...,n, \;\; \text{or } C\mathbf{u} = \mathbf{k}, \tag{1.4}$$

where $f(i,j) = 0$ for $(i,j) \notin \Omega$ and $k_{d,t}$ is the sum of the areas of the intersection between the strip $s_{d,t}$ and the cells of Ω. Again, the vector on the right-hand side is the projection data.

As an example for the strip-based projection model, consider the same setting as in the previous example, with $N = 6$ lattice set and one negative scanning direction $(X_1, Y_1) = (3, -2)$. The 20^{th} strip between the lines $t = 19$ and $t = 20$ is shown in Fig. 1.3.

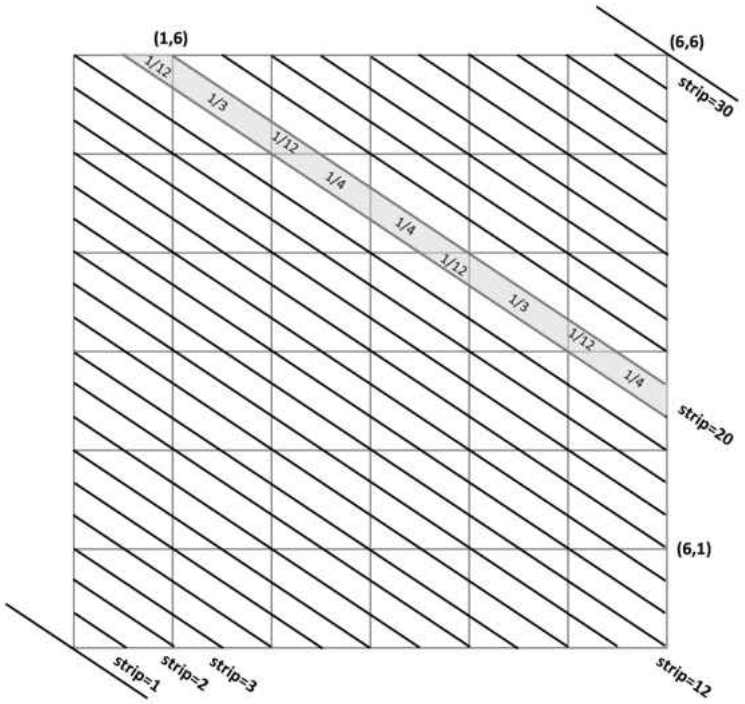
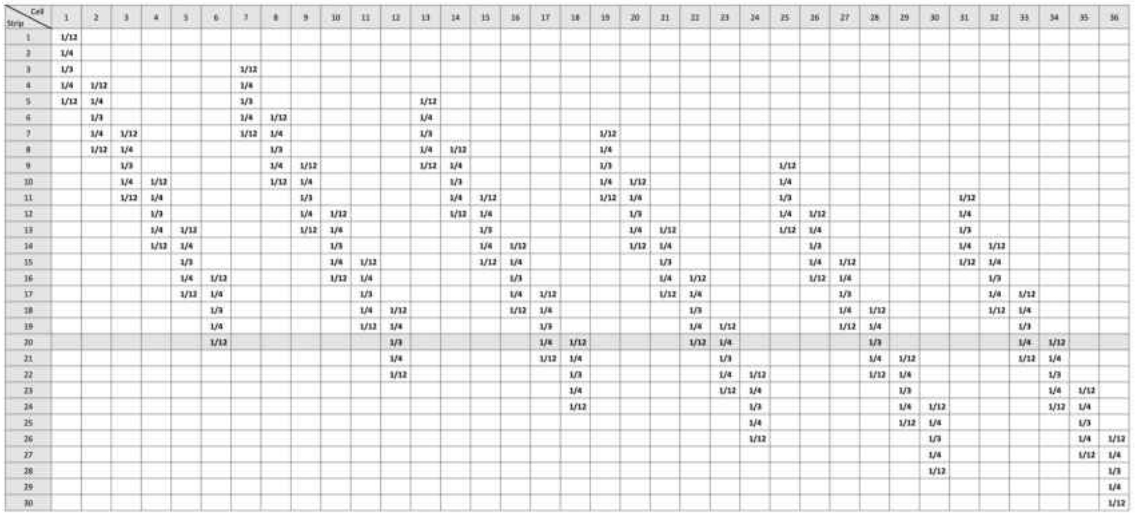


Figure 1.3: Strip-Based Projection Model Scanning

Once the areas of intersection of this strip and each intersecting cell is calculated, an equation for the 20^{th} strip can be written as follows:

$$\begin{aligned} &\tfrac{1}{12}f(1,6) + \tfrac{1}{3}f(2,6) + \tfrac{1}{4}f(3,5) + \tfrac{1}{12}f(3,6) + \tfrac{1}{12}f(4,4) \\ &+\tfrac{1}{4}f(4,5) + \tfrac{1}{3}f(5,4) + \tfrac{1}{4}f(6,3) + \tfrac{1}{12}f(6,4) = k_{20}. \end{aligned} \tag{1.5}$$

Concatenating vertically equations of all 30 strips results in a 30×36 system matrix, shown in Fig. 1.4.



Strip \ Cell	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
1	1/12																																			
2	1/4																																			
3	1/3						1/12																													
4	1/4	1/12					1/6																													
5	1/12	1/4					1/3						1/12																							
6		1/3					1/6	1/12					1/4																							
7		1/4	1/12				1/12	1/4					1/3						1/12																	
8		1/12	1/4					1/3					1/4	1/12					1/6																	
9			1/3					1/4	1/12				1/12	1/4					1/3						1/12											
10			1/4	1/12				1/12	1/4					1/3					1/6	1/12					1/4											
11			1/12	1/4					1/3					1/4	1/12				1/12	1/4					1/3						1/12					
12				1/3					1/4	1/12				1/12	1/6					1/3					1/4	1/12					1/4					
13				1/4	1/12				1/12	1/6					1/3					1/4	1/12				1/12	1/4					1/3					
14				1/12	1/6					1/3					1/6	1/12				1/12	1/4					1/3					1/4	1/12				
15					1/3					1/6	1/12				1/12	1/4					1/3					1/4	1/12				1/12	1/4				
16					1/4	1/12				1/12	1/4					1/3					1/4	1/12				1/12	1/6				1/3					
17					1/12	1/6					1/3					1/6	1/12				1/12	1/6					1/3				1/6	1/12				
18						1/3					1/4	1/12				1/12	1/6					1/3					1/6	1/12			1/12	1/4				
19						1/4					1/12	1/4					1/3					1/6	1/12				1/12	1/4				1/3				
20						1/12						1/3					1/6	1/12				1/12	1/4					1/3				1/4	1/12			
21												1/4					1/12	1/4					1/3					1/4	1/12				1/12	1/4		
22												1/12						1/3					1/4	1/12					1/12	1/4				1/3		
23																		1/4					1/12	1/4						1/3				1/4	1/12	
24																		1/12						1/3						1/6	1/12				1/12	1/4	
25																								1/4						1/12	1/4				1/3	
26																								1/12							1/3				1/4	1/12
27																															1/4				1/12	1/4
28																															1/12					1/3
29																																				1/4
30																																				1/12

Figure 1.4: Strip-Based Projection Model System Matrix

A DT problem for the strip-based projection model is to find an integer solution f to the system in Eq. (1.4).

For each integer t with $-np+1 \leqslant t \leqslant nq$, let two disjoint and complementary subsets of $N = \{1, 2, 3, ..., n\} : S_{t,1} = \{l \in N, (ql - t) \bmod (p) \geqslant q\}$ and $S_{t,2} = \{l \in N, (ql-t) \bmod (p) < q\}$. The strip-based projection equations from the t^{th} strip along the directions $(X_d, Y_d) = (q, p)$ and $(X_d, Y_d) = (q, -p)$ are in the forms presented in Eqs. (1.6) and (1.7), respectively:

$$\sum_{l \in S_{t,1}} \frac{1}{p} f([w]+1, l) \quad + \sum_{l \in S_{t,2}} \left(\left(\tfrac{1}{p} - \tfrac{1}{2pq} - \tfrac{1}{q}\{w\} \right) f([w], l) + \left(\tfrac{1}{2pq} + \tfrac{1}{q}\{w\} \right) f([w]+1, l) \right) = h_{d,t}, \tag{1.6}$$

$$\sum_{l \in S_{t,1}} \frac{1}{p} f([w]+1, l) \quad + \sum_{l \in S_{t,2}} \left(\left(\tfrac{1}{p} - \tfrac{1}{2pq} - \tfrac{1}{q}\{w\} \right) f([w], l) + \left(\tfrac{1}{2pq} + \tfrac{1}{q}\{w\} \right) f([w]+1, l) \right) = h_{d,t}, \tag{1.7}$$

where in Eq. (1.6) $w = \frac{ql-t}{p}$ and in Eq. (1.7) $w = \frac{ql+Np-t}{p}$ with $1 \leqslant t \leqslant N(p+q)$ [9].

In order to build an efficient algorithm to reconstruct the function f, it is important to note the rank and the dimension of the system matrix C given in Eq. (1.4), generated along one direction (X_d, Y_d), where N is a multiple of $X_d|Y_d|$, and $X_d|Y_d| \neq 0$. Then, by [2],

$$C \in R^{(a_d+|b_d|)N \times N^2} \quad \text{and} \quad \text{rank}(C) = (a_d + |b_d|)N - a_d|b_d|. \tag{1.8}$$

1.4 Equivalence Between the System Matrices of the Two Models

Consider the two systems $C\mathbf{u} = \mathbf{k}$ and $M\mathbf{u} = \mathbf{h}$ generated by the strip-based projection in Eq. (1.3) and the line-based projection in Eq. (1.4), respectively, along one direction $(X_d, Y_d) = (q, -p)$. Furthermore, $1 \leqslant q < p$, $\gcd(p, q) = 1$, and N is a multiple of pq. The equivalence between system matrices C and M is studied in a theorem by Zhu and Li in [9] and is summarized as follows. The system matrix C, is in the form of

$$C = [C_1 \quad C_2 \quad ... \quad C_N] \ \in \ R^{(p+q)N \times N^2}, \tag{1.9}$$

where

$$C_i = [\mathbf{c}_1^{(i)} \quad \mathbf{c}_2^{(i)} \quad ... \quad \mathbf{c}_N^{(i)}] \ \in \ R^{(p+q)N \times N}, \quad 1 \leqslant i \leqslant N, \tag{1.10}$$

where

$$\mathbf{c}_j^{(i)} = \begin{bmatrix} \mathbf{0}_{(i-1)p+(j-1)q} \\ \mathbf{y} \\ \mathbf{0}_{(N-i)p+(N-j)q} \end{bmatrix}, \quad 1 \leqslant i, j \leqslant N, \tag{1.11}$$

where

$$\mathbf{y} = [\frac{1}{2pq} \frac{3}{2pq} \ ... \ \frac{2q-1}{2pq} \frac{1}{p} \ ... \ \frac{1}{p} \frac{2q-1}{2pq} \frac{2q-3}{2pq} \ ... \ \frac{1}{2pq}]^T \in R^{p+q}. \tag{1.12}$$

Similarly, the system matrix M is in the form of

$$M = [M_1 \quad M_2 \quad ... \quad M_N] \ \in \ R^{(p+q)N \times N^2}, \tag{1.13}$$

where

$$M_i = [\mathbf{m}_1^{(i)} \quad \mathbf{m}_2^{(i)} \quad ... \quad \mathbf{m}_N^{(i)}] \ \in \ R^{(p+q)N \times N}, \quad 1 \leqslant i \leqslant N, \tag{1.14}$$

where

$$\mathbf{m}_j^{(i)} = \begin{bmatrix} \mathbf{0}_{(i-1)p+(j-1)q} \\ \mathbf{e}_1 \\ \mathbf{0}_{(N-i)p+(N-j)q} \end{bmatrix}, \quad 1 \leqslant i, j \leqslant N, \tag{1.15}$$

where $\mathbf{e}_1$ is the first column of the identity matrix I_{p+q}. Even though the first $p+q-1$ lines do not pass through any grid point of Ω and thus the first $p+q-1$ rows of the system matrix M are all zero rows, we keep them in order for the dimensions of system matrices C and M to be the same. By a sequence of row operations each of the two matrices can be obtained from the other. Thus both C and M are $(p+q)N \times N^2$ and are row equivalent. Furthermore by [9], if M and C are system matrices generated by line-based and strip-based projection methods, respectively, along many directions $(X_d, Y_d)_{d=1}^n$, then both M and C are $N \sum_{d=1}^n (X_d + |Y_d|)$ by N^2 matrices and are also row equivalent with

$$\text{rank}(C) = \text{rank}(M) = N \sum_{d=1}^{n} (X_d + |Y_d|) - \sum_{d=1}^{n} X_d \sum_{d=1}^{n} |Y_d|. \tag{1.16}$$

Since both system matrices are row-equivalent, we will only refer to one of them, system matrix M. These results allow us to transform the strip-based projection system to the line-based projection system, and vice versa. We can apply the techniques developed for one of the systems to the other, where it is convenient. For example, all of the existing reconstruction algorithms based on the line-based projection model can be accordingly applied to the strip-based projection system.

1.5 Full Row-Rank System Matrix Generated Along One Direction

As larger matrices require more resources for the image reconstruction procedure, it is beneficial to reduce the matrix' size while keeping its rank. Suppose M is underdetermined. The deletion of dependent rows of M will transform it into a full row-rank matrix, while preserving its essential structure, i.e rank, and thus reducing the cost of solving the system in Eq. (1.3). In addition, some l_1 minimization software packages for image reconstruction, such as $l_1 - Magic$ package [5], require a full row-rank system.

The procedure for removing all of the dependent rows of M is described by Zhu and Li [10], and is summarized as follows. Let $M \in R^{(p+q)N \times N^2}$ be the system matrix generated by the strip-based projection model along one direction $(q, -p)$, s.t. $1 \leqslant q < p$ and $\gcd(p, q) = 1$. Let H be a set, such that

$$\begin{aligned} H &= H_1 \cup H_2 \cup H_3, \quad \text{where} \\ H_1 &= \{h \in K : 0 < h \leqslant pq \quad \text{and} \quad h \notin 1 + T\}, \\ H_2 &= \{(p+q)N + 1 - h : h \in H_1\}, \\ H_3 &= \{h \in K : (p+q)(N-1) - qN + 1 < h \leqslant (p+q)N - qN\}, \end{aligned} \tag{1.17}$$

where K is the set of all positive integers less than or equal to $(p+q)N$ and

$$T = \{t \in \mathbb{Z} : 0 \leqslant t < pq; t = pu + qv + 1; u, v \in \mathbb{Z}; 0 \leqslant u, v \leqslant N - 1\}. \tag{1.18}$$

Then the indexes of the linearly dependent rows of the system matrix M are the indexes in the set H. Thus the matrix obtained from deleting all rows with indexes in H from M will result in a new full-row rank system matrix rM, and $\text{rank}(rM)=\text{rank}(M)$. If we replace the i^{th} row of M with a zero row and h_i, in the right hand side of Eq. (1.3), with zero, $i \notin T$, then all nonzero rows of the modified M will be maximum linearly independent rows of M. The resultant system is called the reduced binary system matrix (RBSM) generated along one direction (q, p).

As an example of this procedure, consider the scanning of a binary image with size 20×20 along the direction $(4, -5)$. Thus $N = 20$, $q = 4$, and $p = 5$. A linear system $M\mathbf{u} = \mathbf{h}$ is generated by the strip-based projection model along this direction. The matrix M is of size 180×400 and $\text{rank}(M) = 160$. Then the set T in Eq. (1.18) and the set K are:

$$T = \{0, 4, 5, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20\},$$

$$K = \{N \in \mathbb{Z} : 0 < N \leqslant 180\}.$$

Using the notation in (1.17), it follows that:

$$H_1 = \{2, 3, 4, 7, 8, 12\},$$

$$H_2 = \{169, 173, 174, 177, 178, 179\},$$

$$H_3 = \{93, 94, 95, 96, 97, 98, 99, 100\}.$$

Thus deleting the rows from M with indexes in

$$\begin{aligned} H &= H_1 \cup H_2 \cup H_3 \\ &= \{2, 3, 4, 7, 8, 12, 93, 94, 95, 96, 97, 98, 99, 100, 169, 173, 174, 177, 178, 179\} \end{aligned}$$

will result in reduced M, rM, being of size 160×400 with $\text{rank}(rM) = \text{rank}(M) = 160$, and thus rM is of full row-rank.

For the projections along one scanning direction, the method above provides an efficient and precise way of locating linearly dependent rows of the system matrix M, generated by the strip-based projection model. In the case of multiple scanning directions, the system matrix is a vertical concatenation of component matrices for each direction. The method above is not sufficient in the case of multiple directions, as it only finds rows of each component matrix that are dependent to the rows of the same component matrix. In order to make the whole system matrix of full row-rank,

we would also need to find rows of each component matrix that are dependent to the rows of each of the preceding component matrices. The goal of our research is to construct an algorithm that will precisely locate all indexes of each component matrix regardless of the number of scanning directions. Upon removing these indexes from each component matrix, the system matrix M will be of full row-rank.

CHAPTER 2

FULL ROW-RANK SYSTEM MATRIX GENERATED ALONG TWO DIRECTIONS

2.1 Full Row-Rank System Generated Along Two Scanning Directions

Image reconstruction using algebraic approaches is time consuming and the computation cost for the procedure depends on the size of the system matrix M. More scanning directions provide an increase in the reconstructed image quality, however they increase the size of the system matrix M dramatically. Thus the deletion of the dependent nonzero rows along with the zero rows of M reduces the size of M, furthermore transforms it into a full row-rank matrix, preserving all the information embedded in the original matrix M, thus reducing the computational cost for image reconstruction. In addition, some reconstruction algorithms require a full row-rank system matrix to perform the image reconstruction [6]. In this chapter we extend our research from a case of one scanning direction to two.

Let

$$M\mathbf{u} = \mathbf{h} \tag{2.1}$$

be the system of equations, defined in Eq. (1.3), generated by a line-based projection model along two distinct scanning directions $(q, \pm p)$ and $(b, \pm a)$ such that $\gcd(a, b) = \gcd(p, q) = 1$. The components of the system in Eq. (1.3) can be written as:

$$M = \begin{bmatrix} M_1 \\ M_2 \end{bmatrix} \quad \text{and} \quad \mathbf{h} = \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{bmatrix}, \tag{2.2}$$

where M_1 and M_2 are the reduced binary system matrices (RBSM) generated along scanning directions $(q, \pm p)$ and $(b, \pm a)$, respectively. The binary system matrix M is of size $(p + q + a + b)N$ by N^2 with $\text{rank}(M) = (p + q + a + b)N - (p + a)(q + b)$. In order to reduce this system matrix M to a full row-rank matrix, we need to remove

pq zero rows from M_1 and ab zero rows from M_2 along with $bp + aq$ non-zero rows of M_2. A study by Li, Wang, Wu, and Zhu [6] provides an explicit way of finding these $bp + aq$ nonzero rows of M_2. The rows of the resulting system matrix are maximum linearly independent rows of M making M of full row-rank. In this study the authors presented two cases, first of which is for scanning directions of the same sign, both positive or both negative, while the second case is for scanning directions of opposite signs.

Case 1.

Let M be a system matrix generated along two distinct scanning directions of the same sign, $(q, -p)$ and $(b, -a)$ or (q, p) and (b, a), s.t. $\gcd(p, q) = \gcd(a, b) = 1$. The removal of the $bp + aq$ nonzero rows of M_2 whose indexes are the values in the set $delM_2$, given in Eq. (2.3), along with the removal of pq zero rows of M_1 and ab zero rows of M_2 will result in a full row-rank system matrix M.

$$\begin{aligned} delM_2 &= Z \cup Y_B \cup Y'_B, \quad \text{where} \\ Z &= \{ab + bp + 1, ..., aq + ba\}, \\ Y_B &= \{au + bv + 1 : 0 \leqslant u \leqslant b - 1 \ \text{ and } \ 0 \leqslant v \leqslant p - 1\}, \\ Y'_B &= \{au + bv + 1 : N - b \leqslant u \leqslant N - 1 \ \text{ and } \ N - p \leqslant v \leqslant N - 1\}. \end{aligned} \tag{2.3}$$

Case 2.

Let M be a system matrix generated along two distinct scanning directions of an opposite sign, $(q, -p)$ and (b, a) or (q, p) and $(b, -a)$, s.t. $\gcd(p, q) = \gcd(a, b) = 1$. The removal of the $bp + aq$ nonzero rows of M_2 whose indexes are the values in the set $delM_2$, given in Eq. (2.4), along with the removal of pq zero rows of M_1 and ab zero rows of M_2 will result in a full row-rank system matrix M.

$$delM_2 = \{a(N - 1 - q) + 1, ..., a(N - 1 - b) + b(p + a)\}. \tag{2.4}$$

2.2 Geometric Interpretation for Locating Dependent Rows Along Two Scanning Directions

In order to implement the results in Eq. (2.3) and Eq. (2.4) in computational software, such as MATLAB, an interpretation of these results is beneficial. We first consider Case 1 in Section 2.1, in which both scanning directions have the same sign. In addition, we restrict the slopes to be in ascending order, considering the sign, thus for the two negative directions, $-\frac{p}{q} < -\frac{a}{b}$, and for the two positive, $\frac{p}{q} < \frac{a}{b}$. From [6] we obtain the translation of the results via geometry. The domain of sets Z, Y_B and Y'_B in Eq. (2.3) can be plotted on a (u, v) coordinate plane as shown in the Fig. 2.1 and Fig. 2.2. Each point in this domain, excluding certain points which are defined later, evaluated at $au + bv + 1$, provides a linearly dependent row index of M_2. It follows from geometry that the two slopes, $\frac{p}{q}$ and $\frac{a}{b}$ can be enclosed within Quadrant I, as presented in the Fig. 2.1.

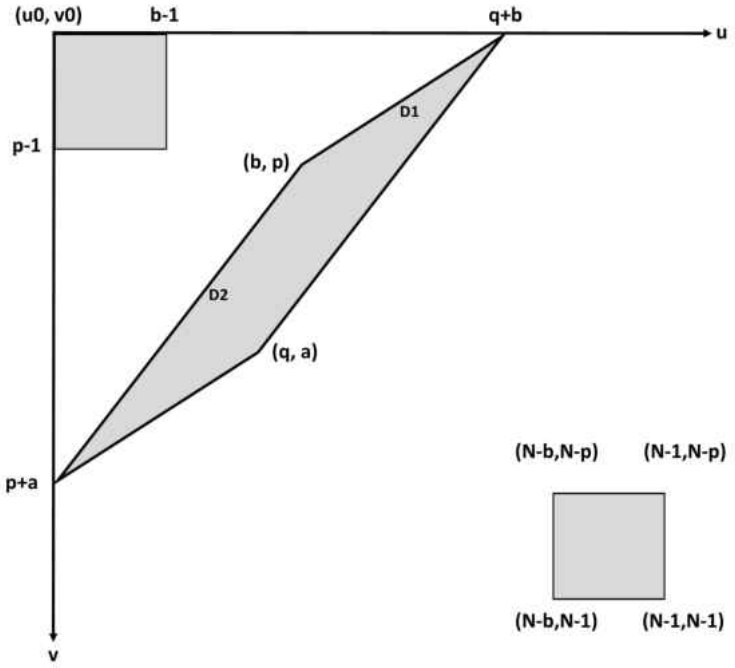


Figure 2.1: Scanning With 2 Positive Directions

For the sub case of both positive directions, the domain of the set Y_B, $\{0 \leqslant u \leqslant b-1$ and $0 \leqslant v \leqslant p-1\}$, is the rectangle, positioned in the corner of the origin $(u0, v0)$. This rectangle includes bp number of points (u, v). Similarly, the domain of the set Y'_B, $\{N-b \leqslant u \leqslant N-1,\ N-p \leqslant v \leqslant N-1\}$, can be visualized as the rectangle of the same size as in Y_B, but positioned in the bottom right corner of the Quadrant 1, also containing bp number of points (u, v). Similarly, referring to Fig. 2.2, for the sub case of both negative directions, the domains Y_B and Y'_B are represented by two rectangles, each also containing bp number of points, however the domain of the set Y_B is now $\{0 \leqslant u \leqslant q-1,\ 0 \leqslant v \leqslant a-1\}$, while the domain for the set Y'_B is $\{N-q \leqslant u \leqslant N-1,\ N-a \leqslant v \leqslant N-1\}$.

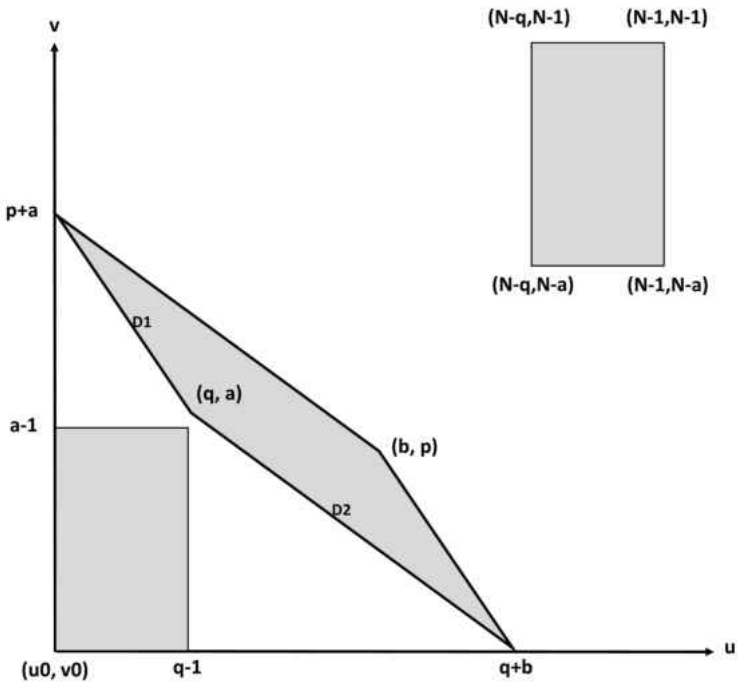


Figure 2.2: Scanning With 2 Negative Directions

While the coordinate plane representaton of the domains of Y_B and Y'_B is somewhat evident, the domain of the set Z is more complicated. Authors in [6] have shown

that the set Z in Eq. (2.3) can be written as $au + bv + 1$ for some $0 \leqslant u, v \leqslant N - 1$. These points (u, v) are in fact all of the discrete interior points and one vertex of the parallelogram that is built from the two positive or negative directions. The parallelogram is built in such a way that the two opposite vertexes of the parallelogram are on the axes $u0, v0$, as shown in Fig. 2.1 and in Fig. 2.2. Evaluating these interior points and any one vertex will result in a set Z in Eq. (2.3). To conclude, the required $aq + bp$ number of (u, v) points that result in $aq + bp$ dependent rows indexes of M_2 inside and on the edge of the two rectangles, and all interior points of the parallelogram, including only one vertex. Evaluating these $aq + bp$ number of points at $au + bv + 1$ will result in a vector $delM_2$ in Eq. (2.3), consisting of all linearly dependent rows of component matrix M_2 to be deleted.

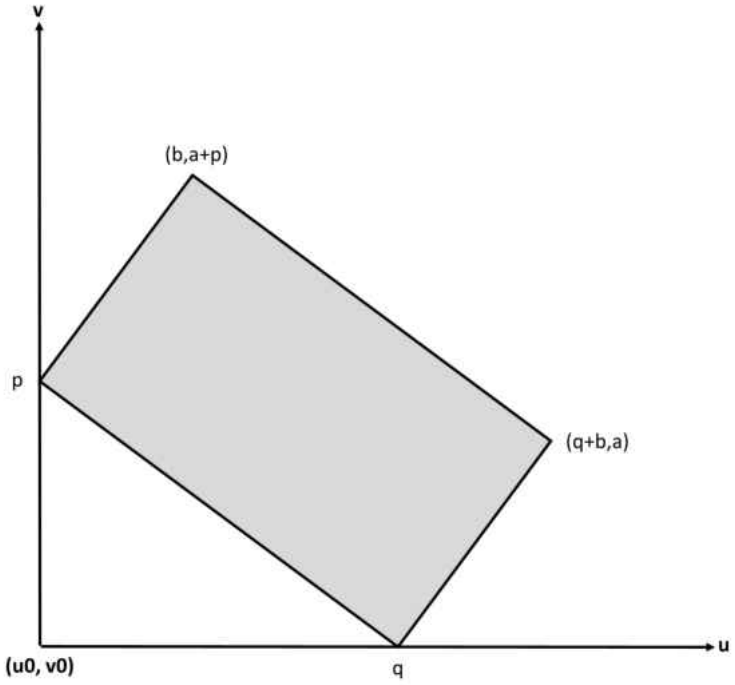


Figure 2.3: Scanning With 2 Directions of Different Signs

We now consider Case 2 in Section 2.1, for scanning along two directions using different slope signs. Based on the study in [6], all of the interior points and one vertex of a parallelogram constructed from these two scanning directions, evaluated at $au + bv + 1$, are representative of the set $delM_2$ in Eq. (2.4). This is also a total of $aq + bp$ points. We construct a parallelogram in Quadrant I using the two slopes of different signs, in the same manner as for Case 1, except that we position it as shown in Fig. 2.2.

2.3 Geometric-Based Algorithm for Scanning With Two Directions

Now that the geometric interpretation is described, and we are able to visualize the results, we can move forward with writing an algorithm in MATLAB for finding the dependent row indexes of the system matrix M_2. The algorithm is called *twodir* and will be used throughout the rest of the paper. Some pseudo code is given in this section, however the exact code can be found in Appendix A. This method considers all three possibilities, which are scanning with: two positive directions, two negative, and different direction signs. Each of the three scenarios has a different geometric interpretation, thus each needs to be dealt with separately and should have its own part in the algorithm. *twodir* will identify which of the three scenarios is taking place, and, based on the scenario's geometric interpretation from previous section, will locate and evaluate the needed points (u, v) at $au + bv + 1$ to an output vector $delM_2$ consisting of all rows of M_2 that are dependent to M_1. The rest of the section will describe the detailed process involved in each step of *twodir*.

TWODIR

Input : $D1, D2$: Scanning directions

N : Image size N by N, s.t. N is a multiple of pq and ab

$u0, v0$: Origin of Quadrant 1 that encloses the needed paralellogram

sv : Axes shift when mixing signs of directions

Output : $delM_2$: Nonzero dependent rows of the second matrix M_2

We start the method in line 1 by checking if the signs of the two scanning directions are both positive. If this condition holds, we go on toward working with the two rectangles and the parallelogram shown in Fig. 2.1. Lines 1.1 - 1.2 compute the row indexes of M_2 associated with the rectangle that is closer to the axes origin. As we find (u, v) points within this rectangle, we evaluate them at $au + bv + 1$, which results in a one dimensional vector *Rect*, where the dependent rows indexes associated with the first rectangle are stored.

1. **if** $\text{sign}(\frac{p}{q}) + \text{sign}(\frac{a}{b}) == 2$

    1.1. **for** $u = 0$ until $b - 1$

        1.1.1. **for** $v = v0 - sv$ until $v0 - sv + p - 1$

            1.1.1.1. temp $=$ [temp, $au + bv + 1$];

        1.1.2. **end**;

        1.1.3. Rect = [Rect, temp];

    1.2. **end**;

The next step, in lines 1.3 - 1.4, is to compute the indexes associated with the second rectangle in the opposite corner from the origin. This is done exactly as with the first rectangle, but with different bounds for u and v. Again, we evaluate the located points at $au + bv + 1$ and store them in a one dimensional vector $Rect2$. This vector includes indexes of dependent rows of M_2 that are associated with the second rectangle.

1.3. **for** $u = N - b$ until $N - 1$

1.3.1. **for** $v = N - v0 + sv - p$ until $N - v0 + sv - 1$

1.3.1.1. temp2 $=$ [temp2, $au + bv + 1$];

1.3.2. **end**;

1.3.3. Rect2 = [Rect2, temp2];

1.4. **end**;

Now that the rectangles are dealt with, we move on toward computing the dependent row indexes associated with the parallelogram in Fig. 2.1, which is done in lines 1.5 - 1.13. We begin by creating a two-dimensional matrix IP containing the points to be checked for being interior points of the parallelogram. The top row of IP contains u coordinates, while the bottom row consists of v coordinates.

1.5. **for** $i = u0$ until $u0 + q + b$

1.5.1. **for** $j = v0$ until $v0 + p + a$

1.5.1.1. xq = [xq, (i)];

1.5.1.2. yq = [yq, (j)];

1.5.2. **end**;

1.6. **end**; IP = [xq; yq];

Next we declare the four vertexes of the parallelogram in vectors xv and yv, lines

1.7 - 1.8. As in the previous step, xv contains u points, while yv contains v points. Then, in line 1.9, a vector in is created using a built-in MATLAB function, *inpolygon*, [7],for finding interior or on the edge points of a polygon. Vector in contains indexes of points of IP that are, in fact, interior or on the edge of the parallelogram. We use a built-in MATLAB function, *inpolygon*, for finding interior or on the edge points of a polygon.

1.7. xv $= [u0,\ u0 + b,\ u0 + q + b,\ u0 + q]$;

1.8. yv $= [v0 + p + a,\ v0 + p,\ v0,\ v0 + a]$;

1.9. in $=$ find(inpolygon(xq, yq, xv, yv) $==$ 1);

Lines 1.10 - 1.13 extract the needed points from IP and place them into a matrix UV, after which the (u, v) points in UV are evaluated at $au + bv + 1$, resulting in a vector $Paral$, containing the indexes of dependent ros of M_2 associated with the interior points and one vertex of the parallelogram. An additional step is required to remove three out four vertexes from UV, which is not included here, but is present in the exact code. Finally, in line 1.14, we combine $Rect$, $Rect2$, and $Paral$, to create the needed output vector $delM_2$.

1.10. **for** $i = 1$ until the number of columns of in

1.10.1. UV$(:, i) =$ IP$(:,$ in$(i))$;

1.11. **end**;

1.12. **for** $i = 1$ until the number of columns of UV

1.12.1. Paral $=$ [Paral, $au + bv + 1$];

1.13. **end**;

1.14. delM$_2$ $=$ [Rect Rect2 Paral];

This completes the part of the *twodir* algorithm that works with both positive scanning directions. In the case of both negative directions, the method skips to line 2. This portion of the algorithm is identical to the one of both positive directions, but since the rectangles and the parallelogram in Fig. 2.2 have rotated counterclockwise by 90 degrees, we need to re-declare the points in lines 1.1, 1.1.1, 1.3, and 1.3.1. The parallelogram, even though it is rotated, still has the same vertex points, but in a different order, so no change is necessary. So lines 2.1-2.14 consist exactly of the same lines as 1.1-1.14, except the changes in the following lines:

2. **if** $\text{sign}(\frac{p}{q}) + \text{sign}(\frac{a}{b}) == -2$

2.1. **for** $u = u0$ until $u0 + q - 1$

2.1.1 **for** $v = 0$ until $a - 1$

2.3 **for** $u = N - u0 - q$ until $N - u0 - 1$

2.3.1 **for** $v = N - a$ until $N - 1$

If the input scanning directions have different signs, the method skips to line 3 to work with Fig. 2.3. The dependent row indexes of M_2 associated with the parallelogram are computed in the similar manner as for same direction signs scenario associated with a parallelogram, except that the location of parallelogram's vertexes in lines 3.3 and 3.4 are slightly changed.

3. **else**

3.1. **for** $i = u0$ until $u0 + q + b$

3.1.1. **for** $j = v0$ until $v0 + p + a$

2.1.1.1. xq = [xq, (i)];

2.1.1.2. yq = [yq, (j)];

3.1.2. **end**;

3.2. **end**; IP = [xq; yq];];

3.3. xv $= [u0, u0 + b, u0 + q + b, u0 + q]$;

3.4. yv $= [v0 - p, v0 - p - a, v0 - a, v0]$;

3.5. in = find(inpolygon(xq, yq, xv, yv) == 1);

3.6. **for** $i = 1$ until the number of columns of in];

3.6.1. UV$(:, i)$ = IP$(:, \text{in}(i))$;

3.7. **end**;

3.8. **for** $i = 1$ until the number of columns of UV

3.8.1. Paral = [Paral, $au + bv + 1$];

3.9. **end**;

3.10. delM$_2$ = Paral;

4. **end**

2.4 Example

We demonstrate the results in Section 2.3 using a numerical example in this section. Consider the scanning of an image with size 30×30 along directions (q, p)=(5,1) and (b, a)=(3,2). The matrix M in Eq. (2.1) consists of M_1 and M_2, RBSMs generated along scanning directions $(5, 1)$ and $(3, 2)$, respectively. The dimension and rank of M is 330×900, with rank$(M) = 306$. The component matrices M_1 and M_2 have 5 and 6 zero rows, respectively. In addition to these 11 zero rows, matrix M has 13 nonzero rows that should be removed to result in a full row-rank matrix. In order to compute these non-zero row indexes, we call on *twodir* algorithm above, by:

$$twodir(D1, D2, N, u0, v0, sv) = twodir([5, 1], [3, 2], 30, 0, 0, 0).$$

The method outputs a set $delM_2 = Z \cup Y_B \cup Y'_B = \{10, 11, 12, 13, 14, 15, 16\} \cup \{1, 3, 5\} \cup \{142, 144, 146\}$. This numerical experiment verifies that the removal of the rows in M_2 with indexes in $delM_2$ along with 5 zero rows of M_1 and 6 zero rows of M_2, results in a full row rank matrix rM with dimension of 306×900 and $\text{rank}(rM) = \text{rank}(M) = 306$.

CHAPTER 3

FULL ROW-RANK SYSTEM GENERATED ALONG MANY DIRECTIONS OF THE SAME SLOPE SIGN

3.1 All Positive Scanning Directions

3.1.1 Three and Four Positive Scanning Directions

In practice, as X-Ray source moves around the object, it creates projections from many directions. The main goal of this research is to propose an algorithm that will provide the dependent row indexes of system matrix M regardless of the number of directions and their signs. This would help reconstruct quality CT images using less computational power.

We first describe the process involving three positive scanning directions (q, p), (b, a) and (d, c), such that N is a multiple of pq, ab, and cd, and $\gcd(p, q) = \gcd(a, b) = \gcd(c, d) = 1$. Additionally, the three directions are in ascending order, such that $\frac{p}{q} < \frac{a}{b} < \frac{c}{d}$. The components of the system in Eq. (2.1) can be written as:

$$M = \begin{bmatrix} M_1 \\ M_2 \\ M_3 \end{bmatrix} \quad \text{and} \quad \mathbf{h} = \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{bmatrix}, \tag{3.1}$$

where M_1, M_2 and M_3 are the reduced binary system matrices (RBSM), generated along scanning directions (q, p), (b, a), and (d, c), respectively. In order to find all linearly dependent rows of M, we first start by finding rows of M_2 that are dependent to M_1. This is done by applying previously established *twodir* algorithm to matrices M_1 and M_2. This procedure stays exactly the same as how it is described in Chapter 2. Once indexes of M_2 are calculated, we proceed by adding the third matrix M_3 and our goal now is to find the rows of M_3, that are dependent to M_1 and M_2, respectively.

The idea is to apply the *twodir* algorithm to each pair, M_1 and M_3, and, M_2 and M_3, but to do so, we need to know the input parameters for *twodir*, which are unique for each pair. Thus it is necessary to construct a geometric interpretation as in Section 2.2 using all three scanning directions. The figure will provide us with the required input parameters for the algorithm. Graphically, from these three directions we construct the needed parallelograms and their respective rectangles, on a (u, v) plane as shown in Fig. 3.1.

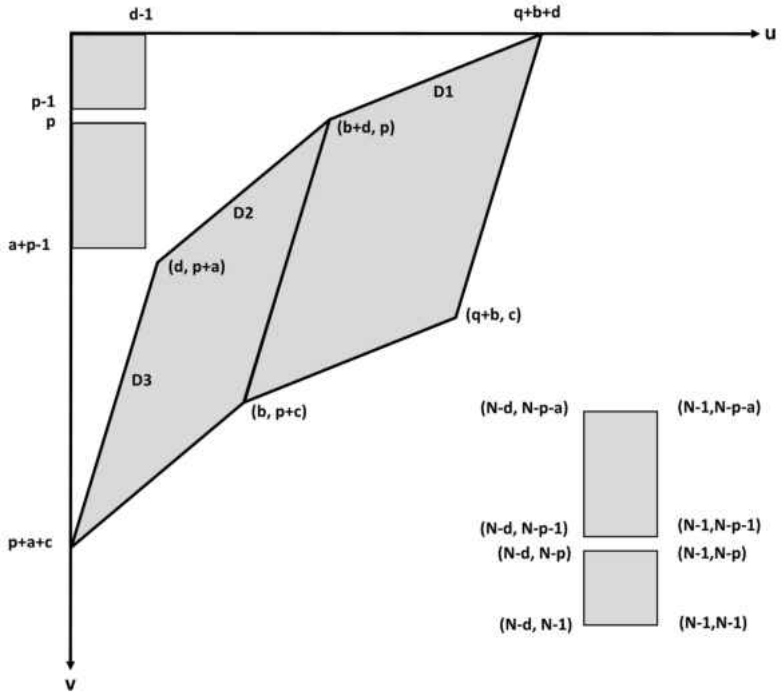


Figure 3.1: Scanning With 3 Positive Directions

Each application of *twodir* in the case of three positive scanning directions will find (u, v) points associated with 2 rectangles and one parallelogram, and since we call on *twodir* twice, we will ultimately manage both parallelograms and all four rectangles, shown in the Fig. 3.1. As we work with matrices M_1 and M_3, using Fig. 3.1, it is evident that the $u0$ axis, shown in Fig. 2.1, needs to be positioned at $u0 = b$,

in order for the opposite vertexes of this parallelogram to touch the axes. $v0$ axis stays at $v0 = 0$ as no shift is necessary here. Similarly, working with the matrices M_2 and M_3, and the parallelogram constructed from the respective scanning directions in Fig. 3.1, it is evident that the $v0$ axis is shifted to $v0 = p$, while $u0$, in this case, is unchanged from $u0 = 0$. The summary of the MATLAB algorithm for three positive scanning directions is shown below.

Three Positive Directions Algorithm

Input : $D1, D2, D3$: Scanning directions

N : Image size N by N, s.t. N is a multiple of pq, ab and cd

Output : $delM_2$: Nonzero dependent rows of the second matrix M_2

$delM_3$: Nonzero dependent rows of the third matrix M_3

1. $delM_2 = twodir(D1, D2, N, 0, 0, 0)$,

   2.1. $delM_{13} = twodir(D1, D3, N, b, 0, 0)$,

   2.2. $delM_{23} = twodir(D2, D3, N, 0, p, 0)$,

2. $delM_3 = [delM_{13}\ delM_{23}]$.

The same geometry and reasoning can be extended to a case with four positive scanning directions, (q, p), (b, a), (d, c), and (f, e), with the same restrictions as in the case of three directions. The procedure for four directions is all that we have shown for three directions, plus one more iteration of computing rows of matrix M_4 that are linearly dependent to all the preceding matrices, M_1, M_2, and M_3. For the third iteration, we position the three parallelograms and their corresponding rectangles as shown in Fig. 3.2, in a very similar manner as before. After determining the

enclosing axes for each of the parallelograms, we, in the same manner, proceed with the MATLAB implementation of the procedure.

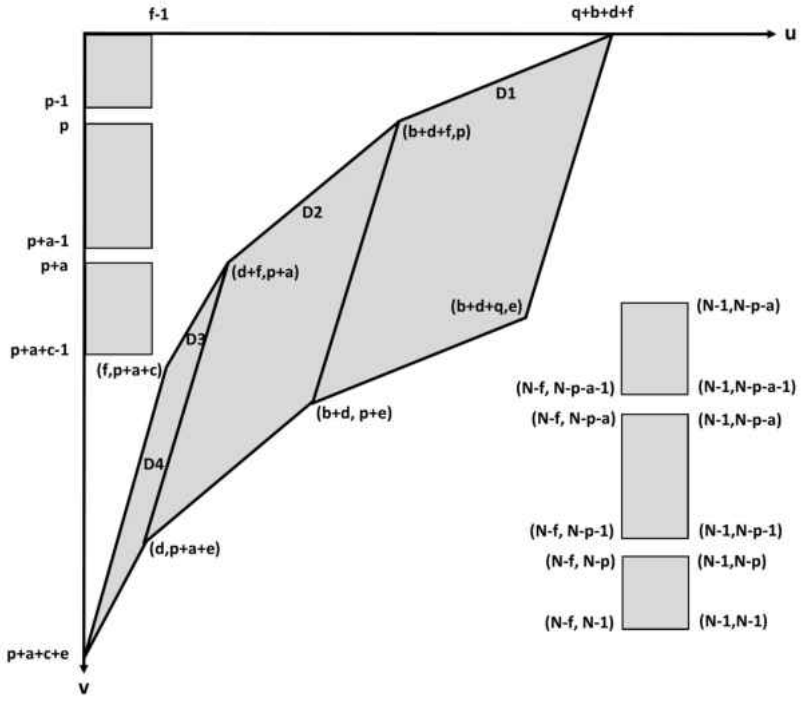


Figure 3.2: Scanning With 4 Positive Directions

Four Positive Directions Algorithm

Input: $D1, D2, D3, D4$: Scanning directions

N : Image size N by N, s.t. N is a multiple of pq, ab, cd and ef

Output: $delM_2$: Nonzero dependent rows of the second matrix M_2

$delM_3$: Nonzero dependent rows of the third matrix M_3

$delM_4$: Nonzero dependent rows of the fourth matrix M_4

$$
\begin{aligned}
1. \quad & delM_2 &&= twodir(D1, D2, N, 0, 0, 0), \\
2.1. \quad & delM_{13} &&= twodir(D1, D3, N, b, 0, 0), \\
2.2. \quad & delM_{23} &&= twodir(D2, D3, N, 0, p, 0), \\
2. \quad & delM_3 &&= [delM_{13}\ delM_{23}], \\
3.1. \quad & delM_{14} &&= twodir(D1, D4, N, b+d, 0, 0), \\
3.2. \quad & delM_{24} &&= twodir(D2, D4, N, d, p, 0), \\
3.3. \quad & delM_{34} &&= twodir(D3, D4, N, 0, p+a, 0), \\
3. \quad & delM_4 &&= [delM_{14}\ delM_{24}\ delM_{34}].
\end{aligned}
$$

We can keep adding positive directions in an increasing manner and continue the process. In the next section we will provide an algorithm that will automatically locate indexes of linearly dependent rows of each component matrix that make up the system matrix M, regardless of the number of positive scanning directions.

3.1.2 Many Positive Scanning Directions

In this section we extend the system matrix M defined in Eq. (2.1) to be the system matrix generated along n distinct scanning directions (X_{D_1}, Y_{D_1}), (X_{D_2}, Y_{D_2}), ... , (X_{D_n}, Y_{D_n}), such that $\frac{Y_{D_1}}{X_{D_1}} < \frac{Y_{D_2}}{X_{D_2}} < ... < \frac{Y_{D_n}}{X_{D_n}}$, $\gcd(X_{D_1}, Y_{D_1}) = \gcd(X_{D_2}, Y_{D_2}) =$ ... $= \gcd(X_{D_n}, Y_{D_n}) = 1$, and N is a multiple of the product of X_i and Y_i for $i = 1, 2, ..., n$. In addition, $\Sigma_{i=1}^{n} X_{Di} < N$ and $\Sigma_{i=1}^{n} Y_{Di} < N$. In practice, the last condition is insignificant as N is large, but, if needed, N can be increased, by scalar multiplying it, without affecting any of the other conditions, and while leaving the scanning directions unchanged. The components of the system in Eq. (2.1) can be written as:

$$M = \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_n \end{bmatrix} \quad \text{and} \quad \mathbf{h} = \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \vdots \\ \mathbf{h}_n \end{bmatrix}. \tag{3.2}$$

In order to make the system matrix M of full-row rank, we take each component matrix, M_2 through M_n, and find indexes of rows that are dependent to all the previous component matrices. This is an iterative process, and in the last iteration we apply the *twodir* algorithm to $n-1$ pairs, direction D_n and each of the directions in the set $\{D_1, D_2, ..., D_{n-1}\}$, hence the $n-1$ iterations. As in previous sections, we construct Fig. 3.3 containing all $n-1$ parallelograms and $2(n-1)$ rectangles, which would help us establish an iterative algorithm for precisely locating all dependent rows of the system matrix M. Essentially our goal is to construct a MATLAB algorithm *ndir*, that will combine all three cases, in particular n positive directions, n negative directions, and n directions of different signs. We proceed with the explanation of the first case.

NDIR Positive

Input: *Dir* : Scanning directions [D1; D2; ... ; Dn]

n : Image size N by N, s.t. N is a multiple of the product of X and Y of each scanning direction

Output: $delM_2$: Nonzero dependent rows of the second matrix M_2

$delM_3$: Nonzero dependent rows of the third matrix M_3

$\vdots$

$delM_n$: Nonzero dependent rows of the nth matrix M_n

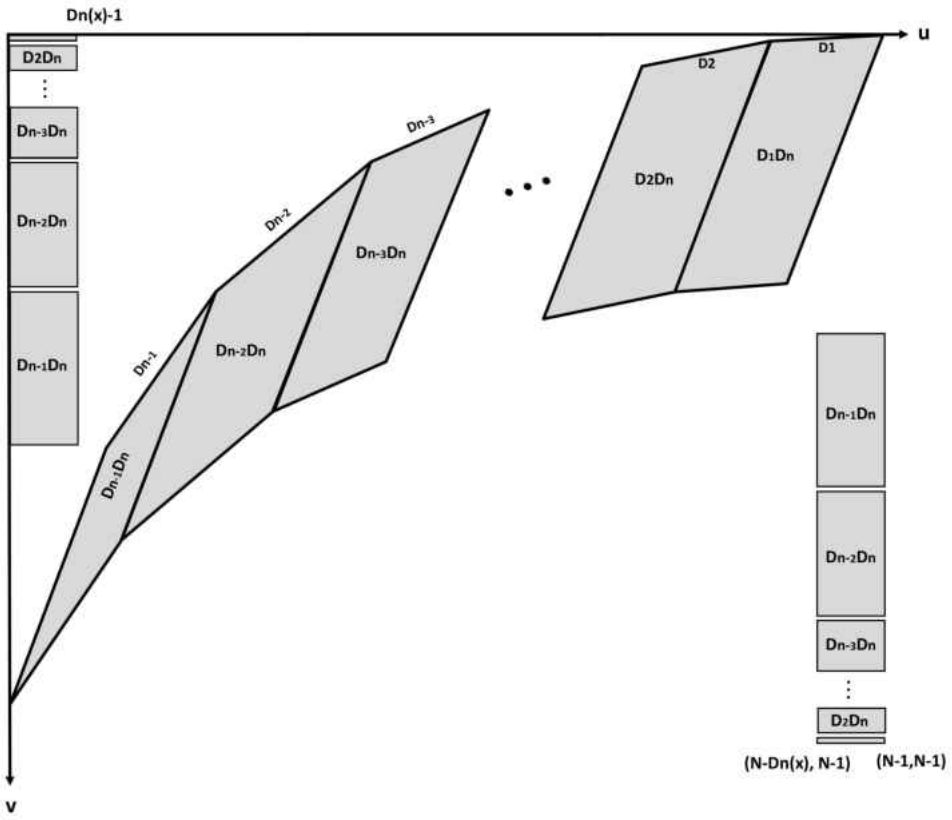


Figure 3.3: Scanning With n Positive Directions

1. for $i = 2$ until number of rows in Dir

   1.1.$v = [\ ]$,

   1.2.for $j = 1$ until $i - 1$

      1.2.1. $v = [v,\ twodir(Dir(j,:), Dir(i,:), N, \text{sum}(X(j+1:i-1)),$

      $\text{sum}(Y(1:j-1)), 0)]$,

   1.3.end,

   1.4.eval($['delM'\ num2str\ (i)' = v']$),

2. end.

In the main iteration loop, index i represents the number of given scanning directions. As i runs from 2 to the index of the last direction, n, we acquire the needed $n-1$ iterations. Within each iteration, we let index j run from 1 to $i-1$, where j represents all the scanning directions preceding the current direction i. Now, using Figure 3.3, let us consider any parallelogram, labeled $D_{n-j}D_n$, for which $i = n$. In order to enclose this particular parallelogram in axes $u0, v0$ as explained in Section 2.2 for positive scanning directions, it is evident that the $u0$ axis needs to be shifted to $u0 = X_{D_{n-1}} + X_{D_{n-2}} + ... + X_{D_{j+1}} = sum(X(j+1 : n-1))$. Therefore, for any i, the $u0$ axis needs to be shifted to $u0 = sum(X(j+1 : i-1))$. Similarly, for the same parallelogram, the $v0$ axis needs to be shifted to $v0 = Y_{D_1} + Y_{D_2} + ... + Y_{D_{j-1}} = sum(Y(1 : j-1))$. For each direction i we restart the vector v and we keep concatenating it with its previous self. Once all js are deprived, we create a vector $delM_i$ containing all rows of the matrix M_i that are dependent to all preceding matrices. Once each of the $delM_i$ vectors are calculated, we proceed with removing from each matrix M_i all the rows whose indexes are in $delM_i$ and all the zero rows. Upon the completion of this process for each of the component matrix M_i, and removing zero rows from M_1, the resulting reduced system matrix M in Equation 3.2 is of full-row rank. An example will be presented at the end of this chapter to verify these results.

3.2 All Negative Scanning Directions

3.2.1 Three and Four Negative Scanning Directions

We now extend this process to a scenario where all of the scanning directions are of a negative sign. The process here is analogous to the one with all positive directions, except for few minor changes. Due to the lack of further complexity here, we will

describe the geometric interpretation for three and four slopes in one example of four negative scanning directions. We will then move toward generalizing this idea for an unlimited number of negative scanning directions.

The the system matrix M generated along four scanning directions $(q, -p)$, $(b, -a)$, $(d, -c)$ and $(f, -e)$, such that $\gcd(p, q) = \gcd(a, b) = \gcd(c, d) = \gcd(e, f) = 1$. Again, directions here are in ascending order, considering the sign, thus $-\frac{p}{q} < -\frac{a}{b} < -\frac{c}{d} < -\frac{e}{f}$. The system matrix M is composed of four component matrices, M_1, M_2, M_3 and M_4. The process of locating linearly dependent rows of M_2, M_3 and M_4 is exactly the same as it is described for four positive directions, however the input parameters $u0$ and $v0$ need to be investigated. We first apply the *twodir* algorithm to component matrices M_1 and M_2, which is described in Chapter 2. We move on by adding a third negative scanning direction, and in order to see the input parameters, we construct Fig. 3.4. For the next iteration, we add fourth negative direction and use Fig. 3.5 to locate the needed parameters.

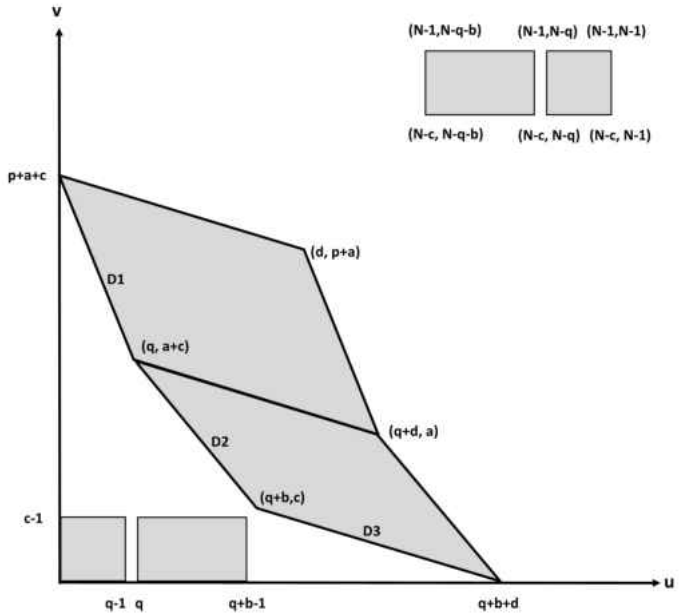


Figure 3.4: Scanning With 3 Negative Directions

Referring back to Chapter 2, the axes, that enclose the parallelogram, based on two negative directions, are placed in the bottom left corner of each parallelogram. Thus in Fig. 3.4, in order to enclose the parallelogram D_1D_3, we need to keep $u0$ at $u0 = 0$, but shift $v0$ to $v0 = a$. Similarly, for the parallelogram D_2D_3, we shift $u0$ to $u0 = q$, and keep $v0$ at $v0 = 0$.

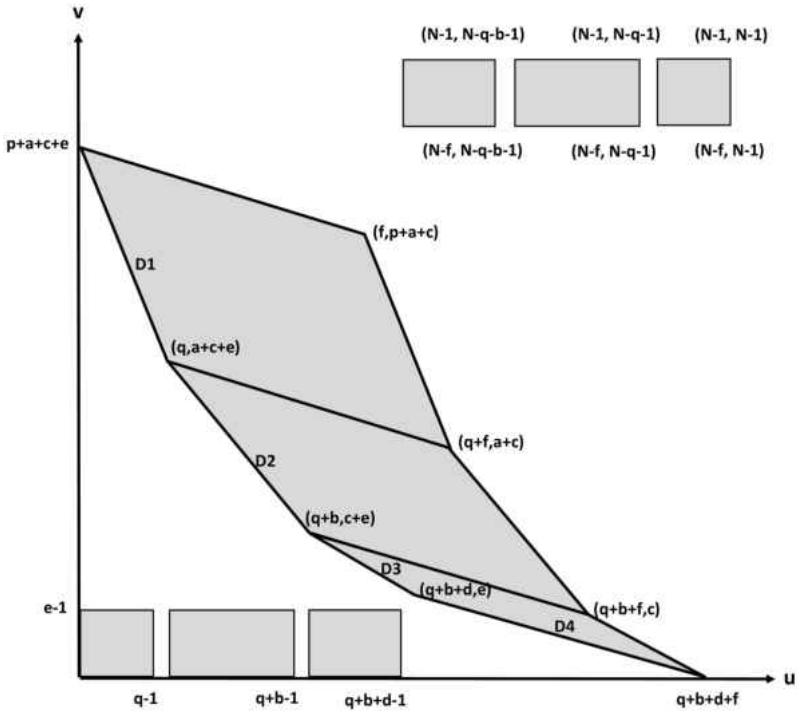


Figure 3.5: Scanning With 4 Negative Directions

Furthermore, referring to Fig. 3.5, we see that for the the parallelogram D_1D_4 we keep $u0$ at $u0 = 0$, and shift $v0$ to $v0 = a + c$. For the the parallelogram D_2D_4 we shift $u0$ to $u0 = q$, and shift $v0$ to $v0 = c$. Lastly, for the the parallelogram D_3D_4 we shift $u0$ to $u0 = q + b$, and leave $v0$ at $v0 = 0$. The summary of a MATLAB process for four negative scanning directions is shown below.

Four Negative Directions Algorithm

Input : $D1, D2, D3, D4$: Scanning directions

N : Image size N by N, s.t. N is a multiple of pq, ab, cd and ef

Output : $delM_2$: Nonzero dependent rows of the second matrix M_2

$delM_3$: Nonzero dependent rows of the second matrix M_3

$delM_4$: Nonzero dependent rows of the second matrix M_4

1. $delM_2 = twodir(D1, D2, N, 0, 0, 0)$,

   2.1. $delM_{13} = twodir(D1, D3, N, 0, a, 0)$,

   2.2. $delM_{23} = twodir(D2, D3, N, q, 0, 0)$,

2. $delM_3 = [delM_{13}\ delM_{23}]$,

   3.1. $delM_{14} = twodir(D1, D4, N, 0, a + c, 0, ,$

   3.2. $delM_{24} = twodir(D2, D4, N, q, c, 0)$,

   3.2. $delM_{34} = twodir(D3, D4, N, q + b, 0, 0)$,

3. $delM_4 = [delM_{14}\ delM_{24}\ delM_{34}]$.

We can keep adding negative directions in an increasing manner and continue the process exactly as with positive directions. We now move toward providing an algorithms that will automatically locate indexes of linearly dependent rows of each component matrix that make up the system matrix C, regardless of the number of negative scanning directions.

3.2.2 Many Negative Scanning Directions

We further extend the system matrix M defined in Eq. (2.1) to be the system matrix generated along n negative distinct scanning directions $(X_{D_1}, -Y_{D_1})$, $(X_{D_2}, -Y_{D_2})$,...,

$(X_{D_n}, -Y_{D_n})$, such that $-\frac{Y_{D_1}}{X_{D_1}} < -\frac{Y_{D_2}}{X_{D_2}} < ... < -\frac{Y_{D_n}}{X_{D_n}}$, and $\gcd(X_{D_1}, Y_{D_1}) = \gcd(X_{D_2}, Y_{D_2}) = ... = \gcd(X_{D_n}, Y_{D_n}) = 1$. In addition, $\Sigma_{i=1}^{n} X_{D_i} < N$ and $\Sigma_{i=1}^{n} Y_{D_i} < N$. Components of the system in Eq. (2.1) can be written in the same way as in Eq. (3.2). In order to make the system matrix M of full-row rank, we take each component matrix, M_2 through M_n, and find indexes of rows that are dependent to all the previous component matrices. This, again, is an iterative process, and in the last iteration we apply the *twodir* algorithm to $n-1$ pairs, direction D_n and each of the directions in the set $\{D_1, D_2, ..., D_{n-1}\}$, hence there are $n-1$ iterations. We construct Fig. 3.6 containing all $n-1$ parallelograms and $2(n-1)$ rectangles, which would help us construct an iterative algorithm that would precisely locate all dependent rows of the system matrix M. We proceed with constructing the portion of *ndir* algorithm that deals with scanning using all negative directions.

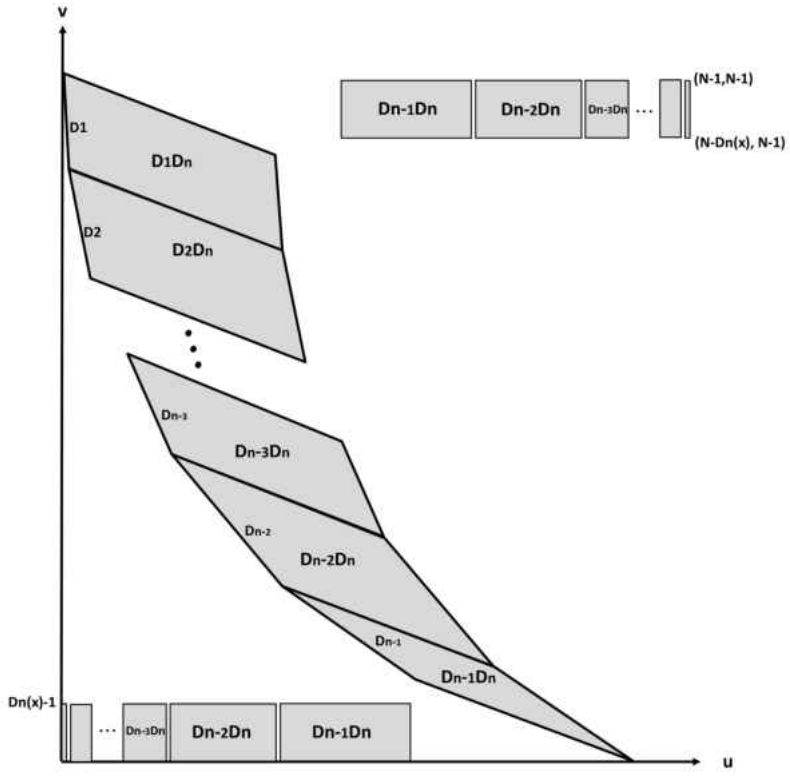


Figure 3.6: Scanning With n Negative Directions

NDIR Negative

Input: Dir : Scanning directions $[D_1; D_2; ...; D_n]$

N : Image size N by N, s.t. N is a multiple of pq, ab, cd and ef

Output: $delM_2$: Nonzero dependent rows of the second matrix M_2

$delM_3$: Nonzero dependent rows of the second matrix M_3

$\vdots$

$delM_n$: Nonzero dependent rows of the second matrix M_n

1. for $i = 2$ until number of rows in Dir

1.1.$v = [\ ]$,

1.2.for $j = 1$ until $i - 1$

1.2.1. $v = [v,\ twodir(Dir(j,:), Dir(i,:), n, \text{sum}(X(1 : j-1)),$

$\text{sum}(Y(j+1 : i-1)), 0)]$,

1.3.end,

1.4.eval($['delM'\ num2str\ (i)' = v']$),

2. end.

The process here is almost identical to the one described in Section 3.2 using all positive scanning direction. Here, again, in the main iteration loop, index i represents the number of given scanning directions. As i runs from 2 to the index of the last direction, n, we acquire the needed $n - 1$ iterations. Within each iteration, we let index j run from 1 to $i-1$, where j represents all the directions preceding the current direction i. However now, using Fig. 3.4, let us consider any parallelogram, labeled $Dn - jDn$, for which $i = n$. In order to enclose this particular parallelogram in axes $u0, v0$ as explained in Section 2.2 for negative scanning directions, it is evident

that the $u0$ axis needs to be shifted to $u0 = X_{D_1} + X_{D_2} + ... + X_{D_{j-1}} = sum(X(1 : j-1))$. Similarly, for the same parallelogram, the $v0$ axis needs to be shifted to $v0 = Y_{D_{n-1}} + Y_{D_{n-2}} + ... + Y_{D_{j+1}} = sum(Y(j+1 : n-1))$. Therefore, for any i, the $v0$ axis needs to be shifted to $v0 = sum(Y(j+1 : i-1))$. For each direction i we restart the vector v and we keep adding it to itself until j completes its cycle. Once all js are deprived, we create a vector $delM_i$ containing all rows of the matrix M_i that are dependent to all preceding matrices. Once each of the $delM_i$ vectors are calculated, we proceed with removing from each matrix M_i all the rows whose indexes are in $delM_i$ and all the zero rows. Upon the completion of this process for each of the component matrix M_i, and removing zero rows from M_1, the resulting reduced system matrix M in Eq. (3.2) is of full-row rank.

3.3 Examples

In this section we demonstrate the results from Chapter 3 using a pair of numerical examples. Consider the scanning of an image with size 48×48 along an array of 7 positive directions, in particular

$$Dir = \begin{bmatrix} 3 & 4 & 1 & 2 & 1 & 1 & 1 \\ 2 & 3 & 1 & 3 & 2 & 3 & 6 \end{bmatrix}.$$

The matrix $M = [M_1; M_2; ...; M_7]$, consisting of 7 component RBSMs generated along 7 positive scanning directions, has the dimension and rank of M is 1584×2304, with $\text{rank}(M) = 1324$. M has 36 zero rows. In addition to these 36 zero rows, matrix M has 224 nonzero linearly dependent rows that should be removed to result in a full row-rank matrix. In order to compute these non zero row indexes, we use the MATLAB algorithm for all positive scanning directions to compute and/or remove all of the dependent rows. The method produced a total of 224 values within 6 vectors, each vector containing the indexes of dependent rows to be removed from

corresponding component matrices. The vectors are:

$$delM_2 = [1, 4, 5, 7, 8, 10, 11, 14, 39, 149, 152, 153, 155, 156, 158, 159, 162],$$

$$delM_3 = [1 - 5, 12, 13, 43 - 47],$$

$$delM_4 = [1, 3 - 12, 14, 28 - 39],$$

$$delM_5 = [1 - 9, 16 - 26, 62 - 70],$$

$$delM_6 = [1 - 11, 18 - 39, 83 - 93],$$

$$delM_7 = [1 - 14, 21 - 78, 149 - 162].$$

Upon removing the rows represented by the indexes in each of these vectors from their respective component matrix, along with removing all the zero rows from the system matrix M, the resulting reduced matrix rM is of size 1324 $\times$ 2304, with rank(rM) = 1324. The matrix is now full-row rank, requiring around 28 percent less computational power for reconstructions, while preserving all of its original image data.

For another example, consider the scanning of an image with size 36 $\times$ 36 along an array of 6 negative directions, in particular

$$Dir = \begin{bmatrix} 1 & 1 & 1 & 2 & 1 & 3 \\ -6 & -3 & -2 & -3 & -1 & -2 \end{bmatrix}.$$

The matrix $M = [M_1; M_2; ...; M_6]$, consisting of 6 components RBSMs generated along 6 negative scanning directions, has the dimension and rank of M is 936 $\times$ 1296, with rank(M) = 783. M has 24 zero rows. In addition to these 24 zero rows, matrix M has 129 nonzero linearly dependent rows that should be removed to result in a full row-rank matrix. In order to compute these non zero row indexes, we use the MATLAB algorithm for all negative scanning directions to compute and/or remove all of the dependent rows. The method produced a total of 129 values within 5 vectors, each vector containing the indexes of dependent rows to be removed from corresponding

component matrices. The vectors are:

$$delM_2 = [1, 2, 3, 7, 8, 9, 67, 68, 69],$$

$$delM_3 = [1, ..., 4, 7, ..., 11, 49, ..., 52],$$

$$delM_4 = [1, 3, ..., 11, 16, ..., 28, 76, 78, ..., 84, 86],$$

$$delM_5 = [1, ..., 5, 7, ..., 15, 31, ..., 35],$$

$$delM_6 = [1, 3, ..., 12, 14, 19, ..., 51, 73, 75, ..., 84, 86].$$

Upon removing the rows represented by the indexes in each of these vectors from their respective component matrix, along with removing all the zero rows from the system matrix M, the resulting reduced matrix rM is of size 783×1296, with $\text{rank}(M) = 783$. The matrix is now full-row rank, requiring around 27 percent less computational power for reconstructions, while preserving all of its original image data.

CHAPTER 4

FULL ROW-RANK SYSTEM GENERATED ALONG MULTIPLE DIRECTIONS OF VARIOUS SIGNS

4.1 Many Directions of Various Signs

In this chapter we extend the process for locating all linearly dependent rows of the system matrix M by combining all the previously discussed cases into a scenario where the scanning directions contain both negative and positive slopes. Let the matrix M, defined in Eq. (2.1), be the system matrix generated along n distinct increasing scanning directions:

$(X_{D_1}, -Y_{D_1})$, $(X_{D_2}, -Y_{D_2})$, ... , $(X_{D_k}, -Y_{D_k})$, $(X_{D_{k+1}}, Y_{D_{k+1}})$, ... , (X_{Dn}, Y_{Dn}) with conditions, such that: $-\frac{Y_{D_1}}{X_{D_1}} < -\frac{Y_{D_2}}{X_{D_2}} < ... < -\frac{Y_{D_k}}{X_{D_k}} < \frac{Y_{D_{k+1}}}{X_{Dk+1}} < ... < \frac{Y_{D_n}}{X_{D_n}}$; $\gcd(X_{D_1}, Y_{D_1}) = \gcd(X_{D_2}, Y_{D_2}) = ... = \gcd(X_{D_k}, Y_{D_k}) = \gcd(X_{D_{k+1}}, Y_{D_{k+1}}) = ... = \gcd(X_{D_n}, Y_{D_n}) = 1$; furthermore $\Sigma_{i=1}^{n} X_{D_i} < N$ and $\Sigma_{i=1}^{n} Y_{D_i} < N$.

The system M can be written as

$$M = \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_k \\ M_{k+1} \\ \vdots \\ M_n \end{bmatrix} \quad \text{and} \quad \mathbf{h} = \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \vdots \\ \mathbf{h}_k \\ \mathbf{h}_{k+1} \\ \vdots \\ \mathbf{h}_n \end{bmatrix}. \tag{4.1}$$

Note that since the scanning directions are in ascending order, the first set of directions in the set $\{D_1, D_2, ..., D_k\}$ are of a negative sign, while the directions in the set $\{D_{k+1}, ..., D_n\}$ are of a positive sign. Our goal is to construct an algorithm, that, as before, will locate all the linearly dependent rows of M, by taking each component matrix M_2 through M_n and locating indexes of its rows that are dependent to all the

preceding matrices. In order to proceed with the explanation of this procedure, we first construct Fig. 4.1 consisting of parallelograms and rectangles that are generated by the given set of n directions. We construct the portion of *ndir* algorithm that work with a set of directions of different signs.

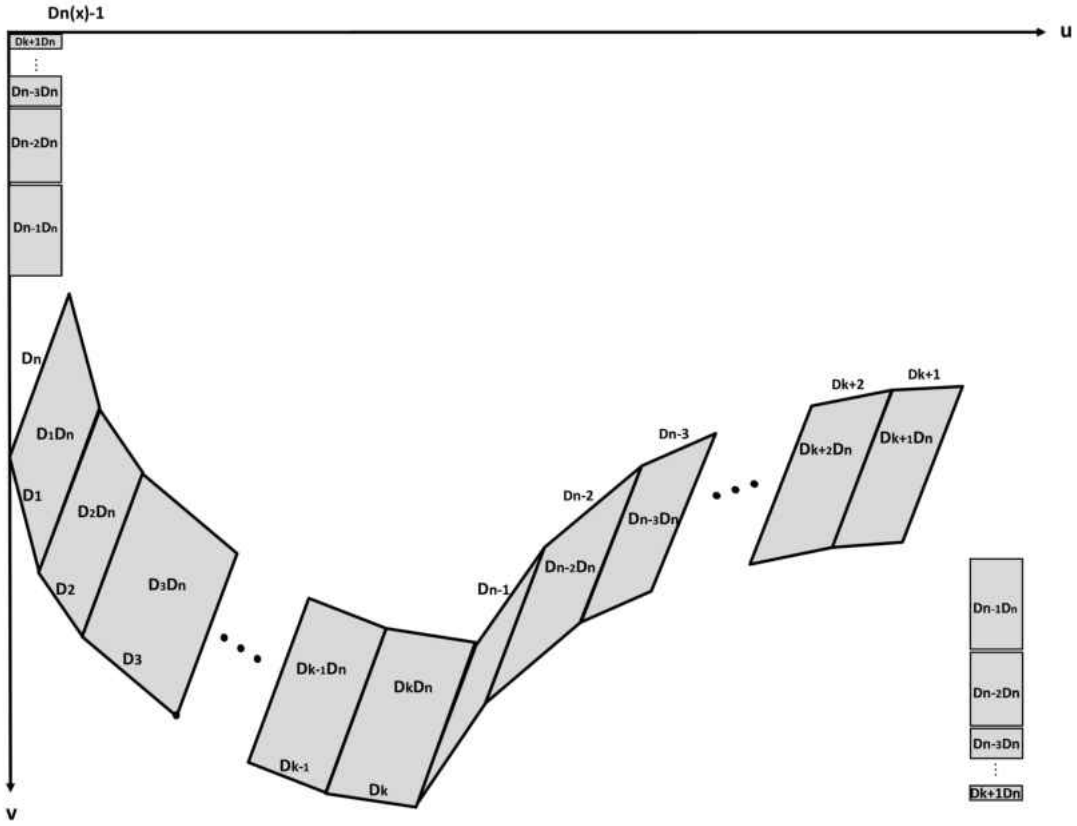


Figure 4.1: Scanning With n Directions of Different Signs

NDIR Different Signs

Input: Dir : Scanning directions [D1; D2; ... ; Dk; Dk+1; ... Dn]

N : Image size N by N, s.t. N is a multiple of the product of X and Y of each scanning direction

Output: $delM_2$: Nonzero dependent rows of the second matrix M_2

$delM_3$: Nonzero dependent rows of the third matrix M_3

$\vdots$

$delM_n$: Nonzero dependent rows of the nth matrix M_n

As we start the algorithm, we first need to locate the index k that tells us how many negative directions are present. This is easily done and is shown in line 1 below:

1. $k = length(find(signDir == -1))$.

Now that we know exactly how many negative directions are present in Dir, we proceed with the method for locating dependent row indexes for each of the component matrices M_2 through M_n. We split the main iteration loop for this portion of $ndir$ into two separate loops, first of which works with directions from 1 to k, while the second loop works with directions $k+1$ through n. It is evident that for the first k directions, which are all of a negative sign, we would adopt the exact procedure that was used for n negative directions, described in Section 3.2.2, where $n = k$. Thus for the first loop of $k-1$ iterations, lines 2 through 3, where i runs from 2 to k, while j runs from 1 to $i-1$, the $twodir$ parameters stay the same as presented in Section 3.2.2.

2. for $i = 2$ until k

   2.1. $v = [\ ]$,

   2.2. for $j = 1$ until $i - 1$

      5.2.1. $v = [v,\ twodir(Dir(j,:), Dir(i,:), n, \text{sum}(X(1:j-1)),$

      $\text{sum}(Y(j+1:i-1)), 0)]$,

   2.3. end,

   2.4. eval($['delM'\ num2str\ (i)' = v']$),

3. end.

Once the first loop is complete, we have located the dependent row indexes for each of the component matrices M_2 through M_k and stored them in corresponding vectors $delM_2$ through $delM_k$. The next step is to add the first positive direction, indexed at D_{k+1}, and apply the *twodir* method to component matrix pairs M_1 and M_{k+1}, through M_k and M_{k+1}. We then proceed by adding the rest of positive scanning directions, one at a time, and proceed as before. In order to generalize the parameters for the *twodir* method, we refer back to Fig. 4.1 which provides a visualization for the last $(n-1)$st iteration. From this figure we construct the second loop that runs through positive directions $k+1$ through n. For every i here, j, represents all directions preceding i, consists of a negative and a positive directions. The first k indexes of j are directions of negative slopes, while i here represents a positive slope. Thus, when applying *twodir* method to a pair D_j and D_i where $j \leqslant k$, we are working with a pair of negative and positive slopes, and thus only interior points and one vertex are representatives of linearly dependent rows here. These parallelograms are shown on Fig. 4.1, and are denoted D_1D_n through D_kD_n. The parameters $u0$ and $v0$ are the axes that enclose each of these parallelograms in a way described in Chapter 2, in

a case of two directions of different signs. For example, consider parallelogram D_1D_n in Fig. 4.1. $u0$ for this parallelogram is at $u0 = 0$, while $v0 = Y_{D_{k+1}} + ... + Y_{D_n} + Y_{D_1}$. Now consider the next parallelogram to the right, D_2D_n. $u0$ here shifts to $u0 = X_{D_1}$, while $v0 = Y_{D_{k+1}} + ... + Y_{D_n} + Y_{D_1} + Y_{D_2}$. Similarly, parallelogram D_3D_n has $u0, v0$ its axes at $u0 = X_{D_1} + X_{D_2}$ and $v0 = Y_{D_{k+1}} + ... + Y_{D_n} + Y_{D_1} + Y_{D_2} + Y_{D_3}$. The pattern here is clear, and thus for the last k^{th} parallelogram D_kD_n, constructed from a negative and positive slopes, the $u0$ and $v0$ shift is at $u0 = X_{D_1} + X_{D_2} + ... + X_{D_{k-1}}$ and $v0 = Y_{D_{k+1}} + ... + Y_{D_n} + Y_{D_1} + Y_{D_2} + ... + Y_{D_k}$. A more general call on *twodir* method is shown in line 4.2.1.1.

4. for $i = k + 1$ until n, the number of rows in Dir

4.1. $v = [\ ]$,

4.2. for $j = 1$ until $i - 1$

4.2.1 if $j \leqslant k$

4.2.1.1. $v = [v,\ twodir(Dir(j,:), Dir(i,:), N, \text{sum}(X(1 : j-1)),$

$\text{sum}(Y(1 : j)) + \text{sum}(Y(k + 1 : i)), 0)]$,

As j runs from 1 to $i - 1$, and $j > k$, the parallelograms constructed with pairs of directions D_{k+1} and D_n through D_{n-1} and D_n are all of positive direction pairs. As we call on *twodir* method for these pair, it would generate dependent indexes from each of the parallelograms and two rectangles associated with it. In order to generally define the parameters for these parallelograms, we first consider parallelogram $D_{k+1}D_n$. The enclosing axes for it are $u0 = X_{D_1} + ... + X_{D_k} + X_{D_{k+2}} + ... + X_{D_{n-1}}$ and $v0 = Y_{D_1} + ... + Y_{D_k}$. Next, for the parallelogram $D_{k+2}D_n$, the axes are $u0 = X_{D_1} + ... + X_{D_k} + X_{D_{k+3}} + ... + X_{D_{n-1}}$ and $v0 = Y_{D_1} + ... + Y_{D_{k+1}}$. Finally, for the last parallelogram $D_{n-1}D_n$, the axes are $u0 = X_{D_1} + ... + X_{D_k}$ and $v0 = Y_{D_1} + ... + Y_{D_{n-2}}$. This generalizes to the $u0$ and $v0$ parameters shown in line 4.2.2.1, where $u0 = sum(X(1 : k)) + sum(X(j+1 : i-1))$,

and $v0 = sum(Y(1 : j-1))$. As we work not only with parallelograms here but also with two rectangles for each parallelogram, we need to include a third parameter, sv. No matter which case we work with, the rectangles have to be positioned near the main axes u, v, such that $u = 0 : X_{D_n} - 1$ and v varies based on the direction, however the first rectangle starts at 0. When working with two or more positive slopes, this is automatically done without using sv or $sv = 0$. However, in the case of directions of different signs, if sv is not used, the rectangles would be positioned in a column at $u = 0 : X_{D_n} - 1$, but the first rectangle $D_{k+1}D_n$ would start at $v = sum(Y(1 : k))$. Thus we require the new parameter $sv = sum(Y(1 : k))$ to shift the column of rectangles to $v = 0$. The summary of the complete algorithm for the case of directions of different signs is shown below in lines 4 through 5.

4.2.2 elseif $j > k$

4.2.2.1. $v = [v,\ twodir(Dir(j,:), Dir(i,:), N,$

$\text{sum}(X(1 : k)) + \text{sum}(X(j + 1 : i - 1)),$

$\text{sum}(Y(1 : j - 1)), \text{sum}(Y(1 : k)))],$

4.2.3 end

4.3. end,

4.4. eval($['delM'\ num2str\ (i)' = v']$),

5. end.

Now that the three portions of *ndir* are derived, we combine them into one program, the exact code of which is found in Appendix B.

4.2 Examples

To finalize our research, we demonstrate with several examples the use of *ndir* method to locate and then remove all linearly dependent rows from the system matrix M, in addition to zero rows, to form a reduced system matrix rM, which is then tested to be of full row rank by using a built-in MATLAB function *rank*. In addition to the dimensions and ranks of original and reduced matrices, we provide the time that *ndir* took to locate and remove the dependent rows on an 2015 Apple MacBook Pro with 2.5 GHz Intel Core i7 processor.

Consider the scanning of an image with size 108×108 along an array of eleven directions, in particular:

$$Dir = \begin{bmatrix} 2 & 4 & 3 & 4 & 9 & 9 & 3 & 4 & 3 & 4 & 2 \\ -9 & -9 & -4 & -3 & 2 & 4 & 2 & 3 & 4 & 9 & 9 \end{bmatrix}$$

The matrix $M = [M_1; M_2; ...; M_{11}]$, consisting of 11 RBSMs generated along 11 directions, 4 of which are positive and 7 negative, has the dimension and rank of M is 11340×11664, with rank(M)=8614. Upon the completion of *ndir* method, the reduced system matrix rM is of size 8614×11664 and rank(rM)=8614. The elapse time is 0.521571 seconds.

Consider another example with scanning of an image with size 100×100 along an array of eight directions, in particular:

$$Dir = \begin{bmatrix} 2 & 5 & 5 & 25 & 25 & 1 & 5 & 5 \\ -5 & -4 & -2 & -4 & -2 & 1 & 2 & 4 \end{bmatrix}$$

The matrix $M = [M_1; M_2; ...; M_8]$, consisting of 8 RBSMs generated along 8 directions, 5 of which are positive and 3 negative, has the dimension and rank of M is

9700×10000, with rank(M)=7948. Upon the completion of *ndir* method, the reduced system matrix rM is of size 7948×10000 and rank(rM)=7948. The elapse time is 0.424827 seconds.

CHAPTER 5

CONCLUSION AND FUTURE WORK

Throughout this work we have proposed a computational algorithm that precisely locates the indexes of linearly dependent rows of a reduced binary system matrix generated along a number of scanning directions. This algorithm has been verified via a large set of numerical examples.

For the image reconstruction procedure, it is customary in industry and academia to use block-scheme iteration reconstruction algorithms. The idea is to split the original system matrix M into many blocks, each block generated from a small number of scanning directions, such as three or four. Dependent rows are then found within the set of those three or four component matrices for each block. Some image reconstruction software packages require the full row-rank blocks. In contrast, our algorithm can be applied to the entire system matrix M, and sequential iteration image reconstruction algorithms can be applied. As a result of applying out algorithm to the entire system matrix M at once, many more rows will be removed compared to working with each block independently. To visualize this, let us consider the second example in Section 4.2. There are eight scanning directions. Let us split the system matrix into two blocks, each with four directions. From the first block of four component matrices *ndir* will remove 415 linearly dependent rows. From the second block of the last four directions *ndir* will remove 243 rows. In total, there are 658 dependent rows to be removed. If we apply *ndir* to the entire system matrix, we are to remove 1531 linearly dependent rows. This is an improvement that could lead to faster image reconstruction time, and thus reduce the computational cost.

In the future we would like to improve the method by adding a set of directions in the form of $(\pm 1, 0)$ and $(0, \pm 1)$. With the addition of these scanning directions, our set will include all possible scanning direction. Lastly, we would like to rigorously

prove the method to verify its accuracy and effectiveness.

REFERENCES

[1] D. Gale, *A theorem on flows in networks*, Pacific Journal of Mathematics, 7:1073-1082 (1957).

[2] L. Hajdu and R. Tijdeman, *Algebraic aspects of discrete tomography*, Journal für die Reine und Angewandte Mathematik, 534:119-128 (2001).

[3] G.T, Herman, and A. Kuba, *Discrete Tomography: Foundations, Algorithms and Applications*, Birkhäuser, Boston, MA, 1999.

[4] C. Kak and M. Slaney *Principles of Computerized Tomographic Imaging*, IEEE Press, New York, NY 1988.

[5] E. Candes and J. Romberg, l_1-*Magic*, (2016, May 15). Retrieved from http://statweb.stanford.edu/ candes/l1magic/

[6] X. Li, H. Wang, Y. Wu, and J. Zhu, *A full row-rank system matrix generated along two directions in discrete tomography*, Journal on Applied Mathematics and Computation, 218:107-114 (2011).

[7] *Points located inside or on edge of polygonal region*, (2016, March 20). Retrieved from www.mathworks.com/help/matlab/ref/inpolygon.html

[8] H. J. Ryser *Combinatorial properties of matrices of zeros and ones*, Canadian Journal of Mathematics, 9:371-377 (1957).

[9] J. Zhu, X. Li, Y. Ye, and G. Wang, *Analysis on the strip-based projection model for discrete tomography*, Journal on Discrete Applied Mathematics, 156:2359-2367 (2008).

[10] J. Zhu and X. Li, *A full row-rank system matrix generated by the strip-based projection model in discrete tomography*, Journal on Applied Mathematics and Computation, 216:3536-3540 (2010).

Appendix A

TWODIR

```
function Bdel = twodir(D1,D2,N,u0,v0,sv)

% The function determines dependent rows of the second matrix M_2
% corresponding to the second direction D2.

% Inputs:
% D1,D2: scanning directions
% N:     image size N by N, N is a multiple of p*q and a*b
% u0,v0: oordinates for the corner of a rectangle enclosing enclosing
%        the parallelogram
% sv:    vertical axes shift for rectangles
%        needed when more than 2 scanning directions are present

% Output:
% Bdel:  nonzero rows in the second matrix M_2.

% Variables:
% Paral: dependent row indices in M_2 corresponding to the parallelogram
% Rect:  dependent row indices in M_2 corresponding to the first rectangle
% Rect2: dependent row indices in M_2 corresponding to the second rectangle

%Initialization
Dir = [D1; D2];
signDir = sign(Dir(:,1).*Dir(:,2));
X = abs(Dir(:,1));
Y = abs(Dir(:,2));
Paral = []; Rect = []; Rect2=[];
```

```
if sum(signDir) == -2     %--
    %indices from parallelogram
        % creating vertices to check
        xq=[]; yq=[];
        for i=(u0):u0+(sum(X))
             for j=(v0):v0+(sum(Y))
                xq = [xq, (i)];
                yq = [yq, (j)];
            end
        end
        IP=[xq; yq];
        % verteces of the parallelogram
        xv = [u0, u0+X(2,1), u0+sum(X), u0+X(1,1)];
        yv = [v0+sum(Y), v0+Y(1,1), v0, v0+Y(2,1)];
        % running built-in inpolygon method
        rng default
        in = find(inpolygon(xq,yq,xv,yv)==1);
        [inm, inn]=size(in);
        % vector of points that are verteces and are internal points
        % of the parallelogram
        for i=1:inn
            UV(:,i)=IP(:,in(i));
        end
        % removing 3 out 4 verteces
        remove1 = [u0 v0+sum(Y)];
        remove2 = [u0+sum(X) v0];
        remove3 = [u0+X(2,1) v0+Y(1,1)];
        UV(:,find(ismember(UV',remove1,'rows')))=[];
        UV(:,find(ismember(UV',remove2,'rows')))=[];
        UV(:,find(ismember(UV',remove3,'rows')))=[]; [UVm UVn]=size(UV);
```

```
        % computing indeces of M_2 based on parallelogram
        for i=1:UVn
            Paral = [Paral Y(2,1)*UV(1,i) + X(2,1)*UV(2,i) + 1];
        end

    %indices from 1st rectangle
        %u=[]; v=[];
        for u = u0 : u0 +  X(1,1) - 1
             temp=[];
            for v = 0 : 0 + Y(2,1) - 1
                 temp = [temp, abs(Dir(2,:))*[v;u] + 1];
            end
            Rect = [Rect temp];
        end

     %indices from 2st rectangle
        %u=[];v=[];
        for u = (N-u0-X(1,1)):(N-u0-1)
            temp2 = [];
             for v = (N-Y(2,1)):(N-1)
                temp2 = [temp2, abs(Dir(2,:))*[v;u] + 1];
            end
            Rect2 = [Rect2 temp2];
         end
         Bdel = [Paral Rect Rect2];
end

if sum(signDir) == 2    %++
   %indices from parallelogram
        % creating vertices to check
        xq=[]; yq=[];
```

```
for i=(u0):u0+(sum(X))
     for j=(v0):v0+(sum(Y))
        xq = [xq, (i)];
        yq = [yq, (j)];
    end
end
IP=[xq; yq];
% verteces of the parallelogram
xv = [u0, u0+X(2,1), u0+sum(X), u0+X(1,1)];
yv = [v0+sum(Y), v0+Y(1,1), v0, v0+Y(2,1)];
% running built-in inpolygon method
rng default
in = find(inpolygon(xq,yq,xv,yv)==1);
[inm, inn]=size(in);
% vector of points that are verteces and are internal points
% of the parallelogram
for i=1:inn
    UV(:,i)=IP(:,in(i));
end
 % removing 3 out 4 verteces
remove1 = [u0 v0+sum(Y)];
remove2 = [u0+sum(X) v0];
remove3 = [u0+X(1,1) v0+Y(2,1)];
UV(:,find(ismember(UV',remove1,'rows')))=[];
UV(:,find(ismember(UV',remove2,'rows')))=[];
UV(:,find(ismember(UV',remove3,'rows')))=[]; [UVm UVn]=size(UV);
% computing indeces of M_2 based on parallelogram
for i=1:UVn
    Paral = [Paral Y(2,1)*UV(1,i) + X(2,1)*UV(2,i) + 1];
end
```

```
    %indices from 1st rectangle
        for u = 0 : X(2,1) - 1
             temp=[];
            for v = v0-sv : v0-sv + Y(1,1) - 1
                 temp = [temp, abs(Dir(2,:))*[v;u] + 1];
            end
            Rect = [Rect temp];
        end

    %indices from 2st rectangle
        for u = (N-X(2,1)):(N-1)
            temp2 = [];
             for v = (N-v0+sv-Y(1,1)):(N-v0+sv-1)
                temp2 = [temp2, abs(Dir(2,:))*[v;u] + 1];
            end
            Rect2 = [Rect2 temp2];
        end
        Bdel = [Paral Rect Rect2];
end

if sum(signDir) == 0 %-+
     %indices from parallelogram
         % creating vertices to check
         xq=[]; yq=[];
         for i=(u0):(u0+sum(X))
             for j=(v0-sum(Y)):(v0)
                 xq = [xq, (i)];
                 yq = [yq, (j)];
             end
         end
         IP=[xq; yq];
```

```
        % verteces of the parallelogram
        xv = [u0, u0+X(2,1), u0+sum(X), u0+X(1,1)];
        yv = [v0-Y(1,1), v0-sum(Y), v0-Y(2,1), v0];
        % running built-in inpolygon method
        rng default
        in = find(inpolygon(xq,yq,xv,yv)==1);
        [inm, inn]=size(in);
        % vector of points that are verteces and are internal points
        % of the parallelogram
        for i=1:inn
            UV(:,i)=IP(:,in(i));
        end
         % removing 3 out 4 verteces
        remove1 = [u0+X(2,1) v0-sum(Y)];
        remove2 = [u0+sum(X) v0-Y(2,1)];
        remove3 = [u0+X(1,1) v0];
        UV(:,find(ismember(UV',remove1,'rows')))=[];
        UV(:,find(ismember(UV',remove2,'rows')))=[];
        UV(:,find(ismember(UV',remove3,'rows')))=[];
        [UVm UVn]=size(UV); Bdel=[];
        % computing indeces of M_2 based on parallelogram
        for i=1:UVn
            Bdel = [Bdel Y(2,1)*UV(1,i) + X(2,1)*UV(2,i) + 1];
        end

end
```

Appendix B

NDIR

```
function [rC] = ndir(Dir,N)

% The program locates and then removes the dependent row indexes
% from M. It then compares the rank of the original system
% matrix M with the rank of the reduced one rM. If the original system
% matrix is over—determined, the program terminates

% Inputs:
% Dir:   set of scanning directions [D1; D2; ... ; Dn]
% N:     image size N by N,  N is a multiple of XiYi, ... , XnYn

% Output:
% rM:    reduced system matrix

% Sorting directions to be in ascending order
Dir(:,3)=Dir(:,2)./Dir(:,1);
[x,y]=sort(Dir(:,3),1,'ascend');
col1=Dir(:,1); col2=Dir(:,2);
temp1=col1(y); temp2=col2(y);
Dir=[temp1 temp2];

% initialization
[a b]=size(Dir);
signDir = sign(Dir(:,1).*Dir(:,2));
X = abs(Dir(:,1));
Y = abs(Dir(:,2));
% directions in original order
```

```
fprintf('input directions :     '); Dir

% obtaining component matrices:
for i=1:a
    M{i} = getmat(Dir(i,:), N);
    Mlength(i) = length(M{i}(:,1));
end

% concatinating component matrices to form C=[M1;M2;...;Mn]
C = cat(1, M{1,:});
[om, on] = size(C);

if om > N^2
   fprintf('Original matrix M is over-determined. i.e. number of rows
   exceeds number of columns. The algorithm is not applicable');
   return
end

%rank based on the paper
r = sum(X+Y)*N-sum(X)*sum(Y);
fprintf('Rank based on the paper: '); r

%Printng original system matrix info
fprintf('original matrix M=[M1;M2;...;Mn], generated from D1,D2,...,Dn :');
fprintf(' size = %4d x %d, ', om, on);
fprintf(' rank = %4d \n', r);

% master loop otputing already reduced matrices based on twodir.m
if sign(Dir(1,2))+sign(Dir(a,2))==2   % all positive directions
    % applying twodir method
```

```
    for i=2:a
        v = [];
        for j=1:(i-1)
            v = [v, twodir(Dir(j,:),Dir(i,:),N,sum(X(j+1:i-1)),
                    sum(Y(1:j-1)),0)];
        end
        eval(['delM' num2str(i) '= v;']);
        % removing dependent rows from corresponding component matrix
        v = setdiff(1:Mlength(i),v);
        M{i} =  M{i}(v,:);
    end

elseif sign(Dir(1,2))+sign(Dir(a,2))==-2   % all negative directions
    for i=2:a
        v = [];
        for j=1:(i-1)
            v = [v, twodir(Dir(j,:),Dir(i,:),N,sum(X(1:j-1)),
                    sum(Y((j+1):(i-1))),0)];
        end
        eval(['delM' num2str(i) '= v;']);
        v = setdiff(1:Mlength(i),v);
        M{i} =  M{i}(v,:);
    end
elseif sign(Dir(1,2))+sign(Dir(a,2))==0   % mixed directions
    % computing the number of negative directions, k

    k = length(find(signDir == -1));
    %applying twodir method to first k directions
    tic
    for i=2:k
        v = [];
```

```
        for j=1:i-1
            v = [v, twodir(Dir(j,:),Dir(i,:),N,sum(X(1:j-1)),
                    sum(Y((j+1):(i-1))),0)];
        end
        eval(['delM' num2str(i) '= v;']);
        v = setdiff(1:Mlength(i),v);
        M{i} =  M{i}(v,:);
    end
    for i=(k+1):a
        v = [];
        for j=1:(i-1)
            % applying twodir method to negative and positive
              directions
            if j<=k
                v = [v, twodir(Dir(j,:),Dir(i,:),N,sum(X(1:j-1)),
                        sum(Y(1:j))+sum(Y(k+1:i)),0)];
            % applying twodir method to positive directions
            elseif j>k
                v = [v, twodir(Dir(j,:),Dir(i,:),N,
                        sum(X(1:k))+sum(X(j+1:i-1)),sum(Y(1:j-1)),
                        sum(Y(1:k)))];
            end
            eval(['delM' num2str(i) '= v;']);
        end
        v = setdiff(1:Mlength(i),v);
        M{i} =  M{i}(v,:);
    end
    toc
end
```

```
% removing zero rows from the reduced system matrix
rC = cat(1, M{1,:});
Mdelz = find(sum(rC,2)==0); zeroarray=Mdelz';
rC(zeroarray,:)=[];
[rCm rCn]=size(rC);

%Printng reduced system matrix data
fprintf('reduced matricx rM=[rM1;rM2;...;rMn], generated from
         D1,D2,...,Dn : ');
fprintf(' size = %4d x %d, ', rCm, rCn);
fprintf(' rank = %4d \n', rank(rC));
```