Sparse Givens resolution of large system of linear equations: Applications to image reconstruction[☆]

María-José Rodríguez-Alvarez [a,*], Filomeno Sánchez [b], Antonio Soriano [b], Amadeo Iborra [a]

[a] *Instituto de Matemática Multidisciplinar, Universidad Politécnica de Valencia, Building 8G (2° Floor), Camino de Vera, E-46022 Valencia, Spain*

[b] *IFIC, Centro mixto CSIC-Univ. de Valencia, Edificio Institutos de Investigación, Paterna, E-46071 Valencia, Spain*

A R T I C L E I N F O

Article history:
Received 28 September 2009
Received in revised form 17 February 2010
Accepted 8 March 2010

Keywords:
Givens rotations
QR-factorization
Computed tomography
Image reconstruction

A B S T R A C T

In medicine, computed tomographic images are reconstructed from a large number of measurements of X-ray transmission through the patient (projection data). The mathematical model used to describe a computed tomography device is a large system of linear equations of the form $AX = B$. In this paper we propose the *QR* decomposition as a direct method to solve the linear system. *QR* decomposition can be a large computational procedure. However, once it has been calculated for a specific system, matrices Q and R are stored and used for any acquired projection on that system. Implementation of the *QR* decomposition in order to take more advantage of the sparsity of the system matrix is discussed.

1. Introduction

Computed tomography has a significant role in medicine and industrial applications [1–4]. As a result of this, continuous efforts are carried out in order to improve image reconstruction algorithms.

The mathematical model used to describe a computed tomography device is a large system of linear equations of the form $AX = B$. Fig. 1 shows a scanner detector. The matrix A ($m \times n$ sparse matrix) describes the system geometry. The element a_{ij} of matrix A is proportional to the area that a ray beam i intersects with the pixel j, ($j = 1, \dots n$, where n is the size of the image to be reconstructed). Consequently, the A matrix is obtained from the geometrical characteristics of the scanner and independently of the scanned object. X is the unknown matrix to be reconstructed and represents the scanned object. The x_{ij} element is proportional to the X-ray absorption of the ray i by pixel j. Finally, B ($m - vector$) contains the X-ray transmission data (projections), measured by the detectors.

In this paper we study the *QR* decomposition techniques for a sparse matrix used to described the computed tomography system geometry [3,4]. Nevertheless results are applicable to any linear sparse systems of equations which have the same requirements: full-rank matrices. It is desirable to have no more than 5% of non-zero elements, because fill-in could cause problems.

A sparse matrix is one of the main data structures used in large-scale scientific and engineering applications for representing linear systems of equations. In large and sparse linear systems, as usually occurs in computer tomography, there are thousands of equations, but each individual equation only depends on a few variables. This leads to equations where most of the coefficients are zero. As a pixel is intersected by two or three rays at the most, only two or three coefficients (a_{ij}) aren't zero. This implies that for each equation having n unknowns, only two or three of them have a non-zero coefficient.

[☆] This work is partially supported by Generalitat Valenciana GVPRE/2008/303 and the Spanish M.E.C. grant MTM2009-08587.

[*] Corresponding author. Tel.: +34 9638 79665.
E-mail addresses: mjrodri@imm.upv.es, mjrodri@mat.upv.es (M.-J. Rodríguez-Alvarez), filomeno@ific.uv.es (F. Sánchez), Antonio.Soriano@ific.uv.es (A. Soriano), amibcar@fiv.upv.es (A. Iborra).

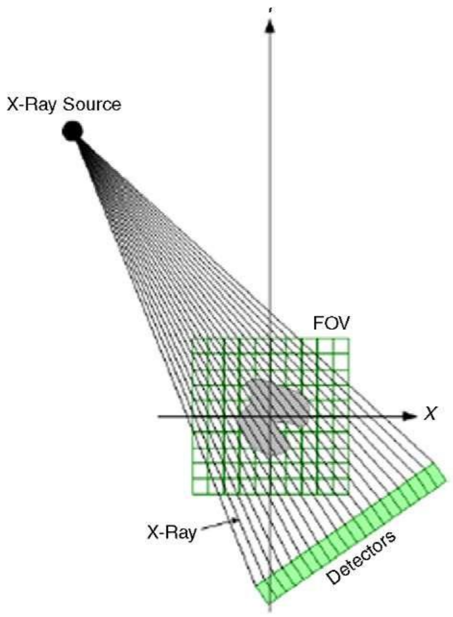

Fig. 1. The figure shows a fan-beam scanner in which the detectors, X-ray source, ray projections and field of view (FOV) are marked. A fan-beam is emitted by the X-ray source which is attenuated by the object. Finally, the attenuated beam is recorded by the detectors.

In computer tomography no more than 3% of the coefficients are non-zero elements, mostly around 2%, depending on the geometrical description of the scanner.

One way of solving a linear system $AX = B$ involves a decomposition matrix A into Q and R [5] matrices, where R is a triangular matrix and Q an orthogonal matrix. If the A matrix is decomposed into Q and R matrices, $A = Q \cdot R$, then the new linear system to be solved is:

$$R \cdot X = Q^T \cdot B. \tag{1}$$

As has been stated above, the A matrix only depends on the geometry of the scanner and the Q and R matrices are calculated for a given scanner only once. When an object is scanned, the Q and R matrices, previously calculated and stored, are used to solve the triangular system (1). In this way, to obtain the final reconstructed image, it is only necessary to solve a triangular linear system.

A is a $m \times n$ matrix and verifies $m \geq n$. For $m < n$ we compute the QR decomposition of A^T. We assume that A has full rank. Orthogonal decompositions have been used extensively for small and dense matrices, because it is well known that such a decomposition is numerically stable. The storage and the operation count requirements in the decomposition are $O(mn)$ and $O(mn^2)$ respectively [6].

This paper is organized as follows. In Section 2, the sparse matrix structure is presented. Section 3 describes the permutation procedure. This part of the algorithm is quite a bit faster and it allows us to speed up the rest of the algorithm. Section 4 describes the way we have followed for the implementation of the QR decomposition algorithm. In Section 5 we present the results we have obtained with our procedure. Finally, in Appendix details on the pseudo-code developed are presented.

2. Sparse matrix representation

The data structure for a sparse matrix is a very important issue. The data structure must be compact and easily accessed by the applications it has been designed for. Sparse row-wise representation of a matrix A with m rows and n columns is given by three one-dimensional arrays.

For instance, consider the sparse matrix

$$A = \begin{bmatrix} a_{11} & 0 & 0 \\ 0 & 0 & a_{23} \\ a_{31} & a_{32} & 0 \\ a_{41} & 0 & 0 \end{bmatrix}.$$

If nz is the number of non-zero elements (five in this case), for a sparse row-wise storing scheme it is necessary to have two arrays of nz elements (RA and JA), and one array of $m + 1$ ($4 + 1 = 5$ in this case), elements IA. RA contains the matrix non-zero elements $RA = a_{11}, a_{23}, a_{31}, a_{32}, a_{41}$, ordered by increasing row index and then by column. JA contains the column

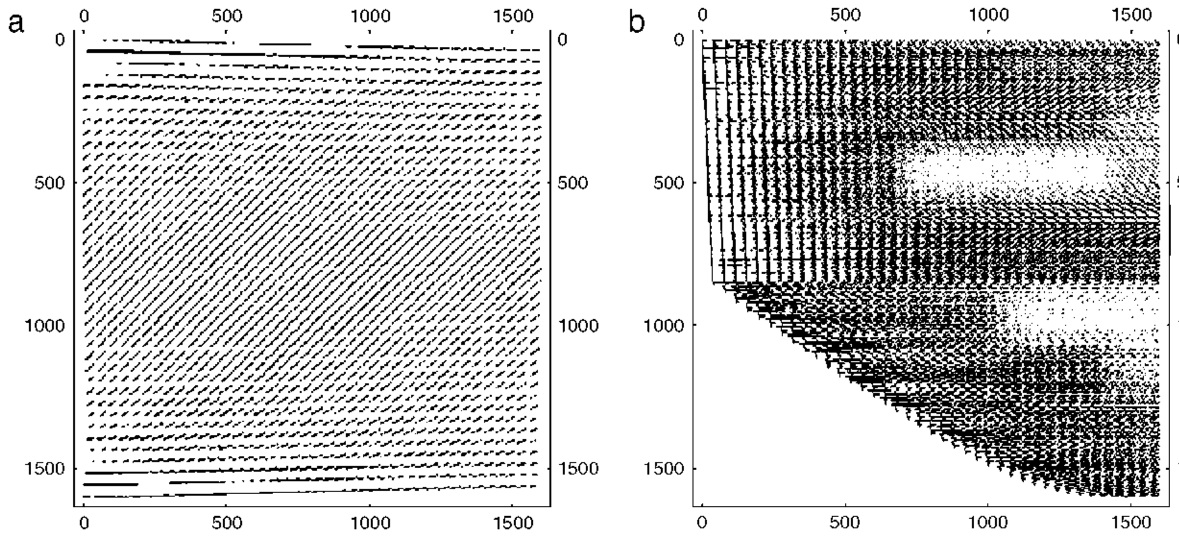

Fig. 2. Effect of row ordering algorithm. (a) Original 1600 × 1600 matrix obtained for a computer tomography system with 2% of non-zero elements. (b) The final matrix obtained with the proposed ordering algorithm. Black points represent the non-zero element positions.

index of the elements stored in RA, in this example $JA = 1, 3, 1, 2, 1$. Finally, the IA array contains the pointer element to the first element of each row in RA (and also in JA). In this example $IA = 1, 2, 3, 5, 6$. Note that element $(m + 1)$ points to the first empty element.

$$\begin{array}{rcccccc} IA = & 1 & 2 & 3 & & 5 & 6 \\ & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow \\ RA = & a_{11} & a_{23} & a_{31} & a_{32} & a_{41} & \\ JA = & 1 & 3 & 1 & 2 & 1 & \end{array} \tag{2}$$

Notice that the column indices in JA are ordered within a given row. It is worth emphasizing that certain applications don't require an ordered row-wise format, e.g. some sparse matrix multiplications [7]. In our case, we use sparse ordered row-wise, because we must "find" quickly non-zero elements placed in the inferior half-triangle of the A matrix. In the same way the sparse order column-wise is also used:

$$\begin{array}{rcccccc} JA = & 1 & & & 4 & 5 & 6 \\ & \downarrow & & & \downarrow & \downarrow & \downarrow \\ RA = & a_{11} & a_{31} & a_{41} & a_{23} & a_{32} & \\ IA = & 1 & 3 & 4 & 2 & 1 & \end{array} \tag{3}$$

Notice tat JA and IA interchange their roles with respect to the row-wise representation. A is stored in the sparse row-wise scheme.

3. Givens rotations and row ordering

We propose to solve the linear system $AX = B$ by QR factorization based on Givens rotations [5], [8,9]. Givens-based QR decomposition factorizes the $m \times n$ matrix A into the product QR, where Q is orthogonal and R is upper triangular (or upper trapezoidal if $m < n$).

In order to reduce the number of Givens rotations and consequently the number of fill-ins, is important to use a new row ordering strategy [10]. We propose a very simple algorithm which reduces considerably the number of eliminations and consequently the number of fill-ins.

The proposed ordering algorithm assigns to each row a label with the column number of the first non-zero element in that row. Then it orders rows by increasing label number. By permuting rows, we can reduce by 70% the number of eliminations. Eliminations could produce fill-ins which need more rotations and so on. Permuting rows is a quasi-instantaneous process, less than 10^{-2} s for matrices with 10^5 rows on a 32-bit 2 GHz microprocessor with 2 GB of RAM.

Fig. 2 shows the original 1600×1600 matrix obtained for a computer tomography system with 2% of non-zero elements. In this figure is also shown the final matrix obtained with the proposed ordering algorithm. It can be seen that the proposed ordering algorithm efficiently reduces the number of eliminations by about 70%. The new order for the rows is stored in an array P of m elements.

4. Algorithms

In this section we describe two new algorithms for sparse matrix decomposition. The first algorithm computes the R matrix and stores data in order to compute the Q matrix. The second algorithm computes the Q matrix. These algorithms are implemented in two separate programs, although both algorithms share a common data structure.

4.1. *R algorithm*

The R matrix is stored in three one-dimensional arrays in the same way as was done for the A matrix. RR contains the matrix elements. Its length, which increases with R calculations, is the number of non-zero elements. JR contains the column index of the elements stored in RR. IR contains the pointer to rows. The structure for the R algorithm is:

```
(1) Read the matrix $A$. Suppose $m$ rows, $n$ columns and $nz$ non-zero elements stored in ordered sparse row-wise.
(2) Apply ordering algorithm and update matrix $A$, then save $P$ array with permutations in an external file.
(3) For columns $i = 1$ to $n$.
    (3a) For row $j = m$ to $i + 1$
    (3b) If $a_{ij} \neq 0$ then make Givens rotations' element $a_{ij}$
    (3c) Update $R$ matrix:
    (3d) Store position $(i, j)$ and $r = c/s$, where $c$ and $s$ are cosine and sine of Givens rotations [5], for future calculations of "Q" matrix.
    (3a) Continue
(3) Continue
(4) Update $A = R$
```

Steps (1) and (2) above read and order the A matrix. After that, the algorithm (loop 3) searches and rotates the non-zero elements. This search is made by increasing order on columns, and for a given column, from the last row up to the diagonal positions. To update matrix R only involves modifying the values of two rows, which are very quickly overwritten because of the data structure used.

When the algorithm finishes, the matrix A has been substituted by the triangular R matrix.

4.2. *Q algorithm*

In this section the Q algorithm [5] is developed for the sparse structure described in Section 2. For a matrix A $m \times n$, the matrix Q contains the computed product of $m + 1$ matrices of $m \times m$ order

$$Q = Q_0 \cdot Q_1 \cdot Q_2 \ldots Q_{j-1} \cdot Q_j \ldots Q_m \tag{4}$$

where Q_0 is the $m \times m$ identity matrix. To calculate Q it is necessary to manipulate three matrices, which we are going to denote as Q_{new}, Q_j and Q_{old}. The matrix Q_{new} is calculated by forward accumulation:

$$Q_{new} = Q_{old} \cdot Q_j$$

where $Q_{old} = Q_0 \cdot Q_1 \ldots Q_{j-1}$. When the algorithm finishes $Q_{new} = Q$. The matrix Q_j isn't calculated and stored as a whole matrix. The Q_j matrix contains the identity matrix of $(j-1)$ order in the first $(j-1)$ rows and columns, whereas the remainder in these rows and columns are zeros. The rest of the $(m - j + 1) \times (m - j + 1)$ submatrix is denoted by $\widehat{Q}_j$.

$$Q_j = \left(\begin{array}{c|c} I_{j-1} & 0 \\ \hline 0 & \widehat{Q}_j \end{array}\right)$$

where I_{j-1} is the identity matrix of $j - 1$ order and $\widehat{Q}_j$ is the matrix corresponding to all the Givens rotations of the column j. Due to the I_{j-1} identity matrix, $(j - 1)$ columns at least remain unalterable in Q_{new} calculations. In any case, Q_{new} has k unalterable columns, with $k \geq j$, Q_{k+1} being the first matrix in the rest of the forward product not equal to the identity matrix. Notice that if no rotations were done in a column of matrix A the corresponding Q_j matrix in Eq. (4) is the identity matrix.

$$\begin{array}{ccccc} Q_{old} & \cdot & Q_j & = & Q_{new} \\ \downarrow & & \downarrow & & \downarrow \\ \left(\begin{array}{c|c} 1 \text{ to } j-1 & j \text{ to } m \\ \text{fixed} & \text{not fixed} \\ \text{columns} & \text{columns} \\ Q_{def} & \widehat{Q} \end{array}\right) & \cdot & \left(\begin{array}{c|c} I_{j-1} & 0 \\ \hline 0 & \widehat{Q}_j \end{array}\right) & = & \left(\begin{array}{c|c} 1 \text{ to } k & k+1 \text{to m} \\ \text{fixed} & \text{to} m \text{ not} \\ \text{columns} & \text{fixed} \\ Q_{def} & Q_{new} \end{array}\right) \end{array}$$

So $k \geq j$ columns of Q_{new} remain unalterable during the rest of the algorithm and can be saved in an external file if it is necessary to deallocate memory. Only $\widehat{Q} \cdot \widehat{Q}_j$ is calculated in each step:

$$\widehat{Q} \cdot \widehat{Q}_j = \left(\begin{array}{c|c} j \text{ to } k-1 & k+1 \text{ to } m \\ \text{columns} & \text{columns} \\ \text{to be added to} & \\ Q_{def} & \widehat{Q_{new}} \end{array} \right) \tag{5}$$

4.2.1. Scheme of Q algorithm

The general scheme for the Q algorithm is very easy, and it isn't necessary to have in RAM all the data from Givens rotations or the matrix Q_{new}. Let us denote J_1: Column number of the last matrix included in the forward product $Q_{old} = Q_0 \cdot Q_1 \ldots Q_{j1}$.$J_2$: Column number for the matrix we are including in the current iteration.J_3: Column number for the matrix we are going to include in the next iteration. nH: number of columns of matrix A which have been modified. This means the number of Q_j which are distinct from the identity matrix. $Qlist$: contains the list of the columns of matrix A which have been modified, i.e. the indices of Q_j matrices which are distinct from the identity matrix.

```
(1) For $j = Qlist[1]$ to $j = Qlist[nH]$
   (1a) For $jj = J_1$ to $J_2 - 1$
         * Calculate columns $J_1$ to $J_{J2-1}$
         * Add columns calculated to the external file $Q_{def}$ in sparse column-wise format.
   (1a) continue
   (1b) For $jj = J_1$ to $jj = m$
         * Calculate rows since column $J_2$ to $m$
         * Save as $\widehat{Q_{new}}$.
   (1b) continue
 • Update $\widehat{Q} = \widehat{Q_{new}}$
(1) Continue
```

The loop(1) above indicates that the calculations are necessary only for those Q_j matrices not equal to the identity matrix. Loop (1a) calculates j to $k-1$ columns to be added to Q_{def} from Eq. (5) in sparse column-wise format (3). Loop (1b) calculates $\widehat{Q_{new}}$ in sparse row-wise format (2) from Eq. (5).

Finally a backward substitution strategy has been used to solve the upper triangular system and the final solution for Eq. (1) is obtained. The pseudo-code is included in Appendix.

5. Discussion and conclusions

The proposed algorithm has been tested with two different algorithms, *LAPACK* [11] library *SGELSS* and Maximum Likelihood Expectation Maximization algorithm MLEM [3], [12,13], which is a widely used reconstruction algorithm in computer tomography. The *LAPACK* [11] library *SGELSS* is used to compare the advantages and drawbacks of sparse against dense structures. As MLEM is a sparse algorithm, we have used it in order to compare its results with our algorithm's results.

A matrix A of $(20\,000 \times 5000)$ with 2% of non-zero elements is used. The matrix A corresponds to a real computer tomography system. For the calculations, we have used a 32-bit 2 GHz microprocessor with 2 GB of RAM. The image to be reconstructed (B-vector) has been obtained with a synthetic model.

The differences between the reconstructed images are negligible within the significant digits for parameters like contrast and signal-to-noise ratio, usually used to compare results in computer tomography [3,14]. Nevertheless, computing time and maximum size of the system matrices are quite different.

It isn't possible to solve bigger systems with our PC with *LAPACK* dense format, but computing times are very good; *LAPACK* only needs $\approx$75 min to solve a $(20\,000 \times 5000)$ system. Our algorithm works more slowly (600 min for the same system) but the limit in the size of matrix A is only defined by the number of non-zero elements. We have managed up to 10^9 non-zero elements. This implies that assuming 1% of sparsity, our matrix in dense format can have up to 10^{11} elements.

It should be noticed that although *QR* factorization is a very slow process, this must be made only once for a computer tomography system. It can be stored as software of the computer tomograph and used to reconstruct any image in a few seconds. Then to reconstruct an image only implies to solve an upper triangular system and to make a product (quasi-instant procedure). Fortunately, this second process to solve the triangular system, only take a few seconds for both systems, *LAPACK* and the sparse algorithm.

We have compared the results obtained with the procedure described in this paper, with those obtained by the authors in a previous work with the MLEM algorithm [3]. Even though images are very similar (differences in quality parameters such as contrast and signal-to-noise ratio are negligible), results are quite a bit different in CPU time consumed. The MLEM algorithm needs several minutes to obtain an image while for the sparse *QR* procedure only a few seconds are needed, our procedure being about 60 times faster. This is due to the fact that *QR* decomposition allows us to store the biggest part of the reconstruction algorithm (as the Q and R matrices don't change during the reconstruction process), and it is only necessary to solve a triangular system to reconstruct an image. However, the MLEM algorithm doesn't allow us to use any precomputed result that could be reutilized and/or stored. The algorithm has to start from scratch for each reconstructed image.

Finally, we can conclude that our algorithm is capable of reconstructing images from computer tomography, by getting at the same time the desirable features of the dense based algorithms (i.e. their speed) and the sparse based algorithms (i.e. the maximum size of the matrix system).

Appendix

It is easy to develop the R algorithm from the scheme from Section 4.1, but more information is necessary to make the Q algorithm because there are several subroutines implied.

The same notation as in Section 4 is used. We are going to use Q_{def} to store row columns which aren't going to be modified. Data to construct the matrices $\widehat{Q}$, $\widetilde{Q_{new}}$ are stored in three arrays in sparse row-wise format and Q_{def} in column-wise format.

Let us to denote:

- $M[i,:]$: row i of M.
- $M[:,j]$: column j of M.
- $M[i:l,j]$ column j since row i to l.
- $L(j:k)$: elements 1 to k of the list L.
- $\tau(j)$: contains τ quotient between sine and cosine of a Givens rotation ordered by columns [5]. It will be used to calculate Q_j matrix.
- $IQ(j)$: list of row position of rotations of matrix Q_j.
- $JQ(j)$: pointers to the first element of each column rotation in $\tau(j)$ and $IQ(j)$.

A.1. Subroutines

Two subroutines or functions are needed to construct rows and columns of matrix Q_j, (or using (5) to simplify $\widehat{Q_j}$). These subroutines need as input the position of the row rotations (j refers to the column), and the list of τ values of rotations made in the j column of the A matrix. The output is the row (or the column).

- *Subroutine_Q_columns*(j, $\tau(j)$, $IQ(j)$, $Q_j[i:,j]$): calculates $Q_j[:,j]$ in dense format. Inputs are $IQ(j)$, list of positions of rotations in the j column of A, $\tau(j)$, list of rotations in the j column of A.
- *Subroutine_Q_rows*(i, j, $\tau(j)$, $IQ(j)$, $Q_j[i,:]$): calculates $Q_j[i,:]$ in sparse format. Inputs are the same as in the above subroutine besides data of the row it must be constructed i.

We have implemented two subroutines to make the sparse product of two matrices M_1 and M_2 to obtain $M_3 = M_1 \cdot M_2$. One subroutine is designed to obtain rows of M_3 (Subroutine_To_rows) which is used to obtain $\widetilde{Q_{new}}$ and the other one to obtain columns (Subroutine_To_columns) used to obtain Q_{def}.

- *Subroutine_To_rows*(i, j, $\tau(j)$, $IQ(j)$, $M_1[i,:]$, $M_3[i,:]$): inputs of this subroutine are the same as *Subroutine_Q_rows*. This subroutine is called to calculate rows of Q_j and M_1. The output is row i of matrix M_3, where $M_3 = M_1 \cdot M_2$ and $M_2 = Q_j$.
- *Subroutine_To_columns*(j, $\tau(j)$, $IQ(j)$, $M_1[:,j]$, $M_3[:,j]$) inputs of this subroutine are the same as *Subroutine_Q_column* besides column j of matrix M_1. This subroutine is called to calculate columns of Q_j. The output is column j of matrix M_3, where $M_3 = M_1 \cdot M_2$ and $M_2 = Q_j$. The columns calculated aren't going to be used again in this algorithm. Columns calculated here are stored in an external file, in sparse column-wise format.

A.2. Pseudo-code of Q algorithm

```
• $J_1 = J(1), J_2 = J(2), J(nH + 1) = m$.
• For $jj = J_1$ to $J_2 - 1$
    – Call Subroutine_Q_columns($J_1$, $\tau(JQ(J_1) : JQ(J_2 - 1))$, $IQ(JQ(J_1) : JQ(J_2 - 1))$, $Q_j[i:,j]$).
    – Save in an external file $Q_{def}$ in sparse column-wise format.
• continue
• For $ij = J_1$ to $m$
    – Call Subroutine_Q_Columns($ij$, $J_1$, $\tau(JQ(J_1) : JQ(J_2 - 1))$, $IQ(JQ(J_1) : JQ(J_2 - 1))$, $Q_j[:, ij]$).
    – Save as $Q$ in sparse row-wise format.
• continue
• For $jj = 2$ to $nH - 1$
    – Actualize $J_1 = J(jj - 1)$; $J_2 = J(jj)$; $J_3 = J(jj + 1)$;
    – For $jj = J_2$ to $J_3 - 1$
        ∗ Call Subroutine_To_columns($jj$, $J_1$, $\tau(JQ(J_1) : JQ(J_2 - 1))$, $IQ(JQ(J_1) : JQ(J_2 - 1))$, $Q[:, jj]$, $Q_{def}[:, jj]$)
    – continue
    – For $jj = 2$ to $m$
        ∗ For $ii = 1$ to $m$
        ∗ Call $Subroutine\_To\ rows(ii, jj, \tau(JQ(J_1) : JQ(J_2 - 1)), IQ(JQ(J_1) : JQ(J_2 - 1)), Q[ii, J_2 : m], Q_{new}[J_2, J_2 : m])$
    – continue
    – $Q = Q_{def}$
• continue
• output of the algorithm is $Q$ matrix stored in sparse row-wise format, which is very useful because we must do $Q^T B$
  product.
```

References

[1] L. Jian, L. Litaoa, C. Penga, S. Gia, W. Zhifang, Rotating-polar coordinate ART applied in industrial CT image reconstruction, NDT & Eint. 40 (4) (2007) 333–336.

[2] G.T. Herman, Image reconstruction from projections: the fundamentals of computed tomography, Med. Phys. 9 (3) (1982) 446–448.

[3] C. Mora, M.J. Rodríguez-Alvarez, J.V. Romero, New pixellation scheme for CT algebraic reconstruction to exploit matrix symmetries, Comput. Math. Appl. 56 (3) (2008) 715–726.

[4] C. Mora, M.J. Rodríguez-Alvarez, I. Baeza, Blobs-bases algebraic reconstruction methods using polar grids, in: Modelling for Engineering and Medicine, 2008.

[5] G.H. Golub, C.F. Van Loan, Matrix Computations, 3 ed., The Johns Hopkins University Press, 1996.

[6] E.G.Y. Ng, Row elimination in sparse matrices using rotations, Thesis University of Waterloo, 1983.

[7] F.G. Gustavson, Two fast algorithms for sparse matrices: Multiplication and permuted transpositions, ACM Trans. Math. Softw. 4 (3) (1978) 250–269.

[8] T.A. Davis, Direct Methods for Sparse Linear Systems, Siam, 2006 (Chapter 7).

[9] W. Givens, Computation of plane unitary rotations transforming a general matrix to triangular form, J. Soc. Ind. Appl. Math. 6 (1958) 26–50.

[10] A. George, E. Ng, On row and column orderings for sparse least square problems systems, SIAM J. Sci. Statist. Comput. 8 (1983) 390–409.

[11] http://www.netlib.org/lapack, last update 2009.

[12] E. Levitan, G.T. Herman, A maximum a posteriori probability expectation maximization algorithm for image reconstruction in emission tomography, IEEE Trans. Med. Imaging MI-6 (1987) 185–192.

[13] L.A. Shepp, Y. Vardi, Maximum likelihood reconstruction for emission tomography, IEEE Trans. Med. Imaging 1 (2) (1982) 113–122.

[14] K.D. Allert, S. Vangala, F.A. DiBianca, Novel materials for low-contrast phantoms for computed tomography, J. Xray Sci. Technol. 15 (1) (2007) 9–18.