A Fast Algorithm of Solving Large Sparse Equations

Renyan GE
R. & D. Dept., SOKKIA Co., Ltd.
260-63 Yanagi-Cho, Hase, Atsugi, Kanagawa 243
E-mail: GHA12214@niftyserve.or.jp
JAPAN

Commission V, Working Group 1

KEY WORDS: Large Sparse Positive Definite Matrices, Choleski Decomposition, Partial Inverse

ABSTRACT

In this paper a fast algorithm for solving large sparse, positive definite matrices is discussed. These equations arise in many surveying problems that are solved by the least squares adjustment, such as network adjustment, photogrammetric aerotriangulation, control network for deformation observation, leveling and applied satellite geodesy.

1. INTRODUCTION

In the least squares adjustment the main computational burden rests on solving the system of linear normal equation

$$AX = b \qquad (1.1)$$

Where A is a square and symmetric (semi-)positive definite n by n matrix. This is especially true if the system is very large that is quite common in the field of surveying. Fortunately, often not all entries of A are nonzero, and usually the rate of fill decreases with the size of the matrix.

If we distinguish among zero elements and nonzero elements of a matrix, we get the concept of the sparse matrix. There are two important advantages when we use these sparse matrices. Firstly, if we operate only upon the nonzeros, we save time; secondly, only the nonzeros have to be stored. However, the special data structures and special algorithms that give an overhead in time and storage are needed. Furthermore, we have to design special algorithms in a way that they use and preserve sparsity.

The most popular algorithm used for the solution of the system (1.1) is based upon Choleski decomposition of the normal matrix A. A positive matrix A can be written as the product of a low triangular matrix L and an upper triangular matrix L^T:

$$A = LL^T \qquad (1.2)$$

The solution of the system (1.1) then goes in two steps:

1. let $Y = L^T X$, solve Y from $LY = b$ by forward solution;

2. solve X from $L^T X = Y$ by backward solution.

The lower triangular factor L (the Choleski factor) is computed by a process called Choleski factorization. There are thus three steps: Choleski factorization, forward substitution and backward substitution, of which Choleski factorization is the most time consuming.

If the normal matrix A is sparse then the Choleski factor L is generally also a sparse matrix. The Choleski factor L has at least the same nonzero as the lower triangle of A (neglecting numerical cancellations), but usually there are also nonzero in L which were not present in A. These elements are called **fill-in**. The amounts of fill in and the location of fill-in elements depends on the order in which the pivots of the Choleski factor are chosen, and hence on the order of the unknowns. The solution of the permuted system:

$$(PAP^T)(PX) = Pb \qquad (1.3)$$

with permutation matrix P^* and solution $X = P^T(PX)$, is identical (except for small round of) to the solution of the system (1.1).

There is no need to pivot or exchange rows or columns for stability reason. Therefore we are free to choose the order of the unknowns, or the permutation matrix P, in such a way as to minimize fill-in. The Minimum Envelope Strategy is one of the method to find an ordering to permute A such that the nonzeros are confined to a specific region as small as possible called 'envelope'. The fill-in will occur only within the envelope.

2. THE COMPUTER REPRESENTATION OF A LARGE SPARSE, POSITIVE DEFINITE MATRIX

In ordinary matrix calculus we deal with full rectangular matrices. These matrices can be stored in a rather straightforward way, whereby individual elements a_{ij} can be accessed very easily.

With large sparse matrices we try to store and operate on the

* P is orthogonal, thus $P^{-1} = P^T$.

nonzero only, which can result in considerable saving in computing requirements. So, firstly we have to specify a storage scheme or data structure in order to store a large sparse matrix in computer memory, and secondly we have to design algorithms which use this data structure in the most profitable sense.

There are many different storage schemes: simple ones and very complex ones. In this paper we introduce Envelope method:

- Envelope, or variable band storage (for symmetric matrices only); all elements within the envelope around the nonzeros, zeros and nonzeros alike, are stored. The set of elements a_{ij} which are stored is called $Envelope(A)$. Note that only half of the matrix is stored.

$$Envelope(A) = \{a_{ij} \mid f_i \le j \le i\} \qquad (2.1)$$

where

$$f_i = \min\{k \mid a_{ik} \ne 0, 1 \le k \le i\} \qquad (2.2)$$

is the first nonzero element in row i .

The access time for a single element is not that important for Choleski factorization and ordering algorithms, since most of the time entire rows or columns are needed.

The nonzero storage is the most complex storage scheme. The envelope storage scheme involves the overhead of the sparse matrix storage schemes, and besides access time is low. In this scheme however, also some zeros are stored and hence will be operated upon.

The envelope storage scheme of the sparse symmetric matrix is that *VE* (*VectorEnvelope*) stores first nonzero element and all elements up to diagonal of every row (zeros and nonzeros alike). So the *i*'th row of *A* needs at least $\beta_i(A)+1$ units of memory space; *VE* needs $\sum_{i=1}^{N}(\beta_i+1)$ units memory; and *PD* (*PointerDiagonal*) needs *N* units memory. So the total storage space for sparse symmetric matrix is:

$$\sum_{i=1}^{N}(\beta_i+1)+N=\sum_{i=1}^{N}\beta_i+2N \qquad (2.3)$$

An advantage of this kind storage scheme is that all of the fill-in elements are located between the first nonzero element and diagonal element of *i*'th row when *A* is factored by Choleski decomposition, the *L* elements can be stored in *VE*, while the other elements of *A* are no need moving and storing.

3. THE SPARSE MATRIX AS A GRAPH

A graph $G=(V,E)$ consists of a set nodes (or vertices) *V*, and a set of edges *E*. We may associate the symmetric *n* by *n* matrix *A* with a labeled undirected graph $G^A=(V^A,E^A)$. The set of nodes corresponds to the diagonal entries of the matrix, the set of edges to the off-diagonal entries. Labeled means that each node has a unique number corresponding to a row, column or unknown; undirected means that we do not distinguish between the edge from node *v* to *w* and the edge from *w* to *v*, i.e. the matrix is symmetric. For any *n* by *n* permutation matrix $P \ne I$, the unleveled graph of *A* and PAP^T are the same, but the associated labeling are different.

4. CHOLESKI DECOMPOSITION

4.1 The Inner Product Method

In the least squares adjustment the coefficient matrix A of linear normal equation (1.1) is a square and symmetric (semi-)positive definite n by n matrix, which has a unique triangular factorization LL^T, where L is a lower triangular matrix with positive diagonal entries.

The linear normal equation (1.1) can be expressed by followings:

$$A = LL^T \qquad (4.1)$$

There are a number of methods to perform the choleski factorization. In some paper the 'inner product', the 'bordering' and the 'outer product' method is distinguished. The inner product method will be described here, since it has less operations comparing with others. It can be described as follows.

```
for $i = 1,\cdots,N$
$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$
for $j = i+1, i+2, \cdots, N$                    (4.2)
$l_{ji} = \left(a_{ji} - \sum_{k=1}^{i-1} l_{jk} l_{ik}\right) \Big/ l_{ii}$
end of loop $j$
end of loop $i$
```

These formulae can be derived directly by equating the elements of *A* to the corresponding elements of the product LL^T.

4.2 Sparsity Consideration

In most practical applications the matrix to be factored is sparse. If the sparse symmetric positive definite matrix *A* has a Choleski factorization LL^T, then the matrix *L* is usually sparse too, but also newly created nonzeros, called fill-in. According to the envelope storage scheme of a sparse symmetric matrix, the algorithm of the inner product of Choleski factorization can be described as follows.

It is easy to modify the algorithm (4.2) with the help of a matrix interface between a matrix element a_{ij} and a vector VE (VectorEnvelope) with its address pointer vector PD (PointerDiagonal), and a switch of which are only located in the envelope should be exceeded.

5. SOLVING LINEAR NORMAL EQUATIONS BASED ON CHOLESKI FACTORIZATION

5.1 General Form

As we described in Chapter 1, there are three steps to solve linear normal equations based on Choleski factorization: Choleski factorization, forward substitution and backward substitution.

After Choleski decomposition of the normal matrix A, a low triangular matrix L and an upper triangular matrix L^T are obtained. So, the forward substitution and the backward substitution can be expressed as follows respectively.

forward substitution:

```
for $i = 1,\cdots,N$
    $y_i = \left(b_i - \sum_{k=1}^{i-1} l_{ik} y_k\right) / l_{ii}$        (5.1)
end of loop $i$
```

backward substitution :

```
for $i = N,\cdots,1$
    $x_i = \left(y_i - \sum_{k=i+1}^{N} l_{ki} x_k\right) / l_{ii}$        (5.2)
end of loop $i$
```

5.2 Sparsity Consideration

After Choleski decomposition of the normal matrix A, according to the envelope storage scheme of a sparse symmetric matrix, the algorithm of the forward substitution and the backward substitution can be described as follows.

It is easy to modify the algorithm (5.1) and (5.2) with the help of a matrix interface between a matrix element a_{ij} and a vector VE (VectorEnvelope) with its address pointer vector PD (PointerDiagonal), and a switch of which are only located in the envelope should be exceeded.

6. PARTIAL INVERSE

The inverse of the normal matrix plays an important role in least square's problems: it is the covariance matrix of the least squares estimations $\hat{x}$ and it is also very important for quality control of the adjustment system.

The inverse of a general regular and square matrix A can be computed by solving the system $AB=I$, with $B = A^{-1}$. If we already have factored A, the inverse can be computed column by column by repeated forward and back substitution with column e_i of I. This is a very straightforward way of computing the inverse of a matrix.

For sparse matrices, repeated forward and back substitution is not a very elegant way for computing the inverse of a matrix. The inverse of a sparse matrix is generally a full matrix. However, we do not always need all these elements. For most applications, the sparse inverse is sufficient, and other elements are not needed. And also for some special purpose, only one or a few elements of the inverse have to be computed. For this reason, we will introduce a technique to compute a large number of elements of the inverse, corresponding to the nonzeros in the Choleski factor, and modifying it to meet the needs of some special purposes. The inverse that is computed by this technique is called the partial inverse or sparse inverse.

6.1 The General Case

Let A be an $n \times n$ symmetric positive definite matrix, with inverse $A^{-1} = B$, and L be the lower triangular matrix from the factorization $A = LL^T$, then we have the following derived forms.

$$A = LL^T \Rightarrow L^T BL = I \qquad (6.1)$$

By a recursive partitioning technique, we can compute a subset of the inverse in an economic manner.

The algorithm for computing the inverse of A should be expressed as follows.

```
$\bar{B}_n = 1/l_{nn}$
for $i = n-1,\cdots,1$
    $\bar{b}_i = -\bar{B}_{i+1} \bar{l}_i / l_{ii}$        (6.2)
    $b_{ii} = \left(1/l_{ii} - \bar{l}_i^T \bar{b}_i\right) / l_{ii}$
end of loop $i$
```

6.2 Sparsity Consideration

It has been discussed in Chapter 4, in most practical applications the matrix to be factored is sparse, then the matrix L is usually sparse too, but also newly fill-in created.

On the basis of the algorithm (6.2), let us consider the computation of a single element b_{ij} in the i^{th} column of the inverse, i.e.,

$$b_{ij} = \bar{b}_j^T \bar{l}_i / l_{ii} : \qquad (6.3)$$

Assume that $\bar{l}_i$ is a sparse vector, then only those elements of $\bar{b}_j$ (the j^{th} column of the inverse) are needed which correspond to a nonzero in $\bar{l}_i$. It turns out that, when l_{ij} itself is a nonzero, the required elements in $\bar{b}_j$ correspond also to nonzero in the Choleski factor. This becomes plausible when we consider that the fill-in, created in the i^{th} elimination step of Choleski factorization, is given by $Nonzero\left(\bar{l}_i \bar{l}_i^T\right)$. Hence, it is possible to compute only the elements b_{ij} of the inverse which correspond to nonzeros in the Choleski factor, i.e. $(i,j) \in Nonzero(L)$.

There are two forms of algorithm (6.2) for computation. One is to compute the inverse elements row by row, and another is to compute the inverse element column by column. The second form can be expressed as follows.

```
for $i = n,\cdots,1$
    for $j = n,\cdots,i+1$
        $b_{ji} = -\sum_{k=i+1}^{n} b_{jk} l_{ki} / l_{ii}$
    end of loop $j$                                  (6.3)
    $b_{ii} = \left(1/l_{ii} - \sum_{k=i+1}^{n} b_{ik} l_{ki}\right) / l_{ii}$
end of loop $i$
```

An algorithm of form one solves the inverse from right to left

row by row, its needs overhead space to store a matrix *L.* While the second algorithms compute the inverse element's column by column, only a vector is needed for storing a column elements of matrix *L.* Here, the second method is suggested for our purpose.
The operation count is about twice the operation count of the Choleski factorization process. However, due to additional overheads, the actual execution time of the sparse inverse is approximately three times the time needed for Choleski factorization. For most applications the sparse inverse is sufficient, and other elements are not needed.

REFERENCE

References from Books:
A. George and J.W.H. Liu, 1981, Computer Solution of Large Sparse Positive definite System, Prentice-Hall, Inc. Engwood Cliffs, New Jersey 07632.

References from Other Literature:
Richard A. Snay, 1976, Reducing the Profile of Sparse Symmetric Matrices, NOAA Technical Memorandum NOS NGS-4.