Solving Linear Least-Squares Problems for Sparse Matrices

Linear Algebra and its Applications

Project Report

Praveen Kumar
MDS202225
praveenk@cmi.ac.in

Rohan Karthikeyan
MDS202226
rkarthikeyan@cmi.ac.in

Roudranil Das
MDS202227
roudranil@cmi.ac.in

Saikat Bera
MDS202228
saikatb@cmi.ac.in

Instructor: **Priyavrat Deshpande**
Associate Professor

Chennai Mathematical Institute

April 14, 2023

Abstract

In this paper, we explore Björck and Duff's algorithm [BD80] for solving sparse least squares problems. We present a detailed analysis of how it ensures numerical stability while also preserving much of the system's original sparsity.

Contents

1. Introduction

The standard system of linear equations bears the following form:

$$b = Ax + \epsilon$$

where,

- $b \in \mathbb{R}^m$ is the column vector of observed values of the dependent variable;
- $A \in \mathbb{R}^{m \times n}$ is a fixed known matrix;
- $x \in \mathbb{R}^n$ is the column vector of the solution to the system that has to be estimated; and
- $\epsilon \in \mathbb{R}^m$ is the column vector of random errors.

The most popular method of estimating the solutions is the method of **least squares**, which can be stated as:

$$\min_x \|b - Ax\|_2 \tag{1.1}$$

We make the standard assumptions that $m \geq 2$ and $\text{rank}(A) = n$, in which case (1.1) has a unique solution.

We are primarily interested in solving (1.1) when A is large and sparse. Wilkinson defined a sparse matrix as *one with enough zeros that it pays to take advantage of them.* This informal yet practical definition captures the essence of the goal of all the algorithms to solve (1.1). They must exploit the sparsity of the matrix to solve problems economically: **much faster** and using **far less memory** than if all the entries were stored and took part in explicit numerical computations. In addition to sparsity preservation, we must also emphasize numerical stability when evaluating algorithms for sparse least squares problems.

For solving large sparse systems, iterative methods are generally preferred because they need much less storage than direct methods. In this regard, the (preconditioned) Conjugate Gradient Least Squares (CGLS) [HS+52] or its stabilized version LSQR [PS82] are most commonly used. Even though such methods are outside the scope of this paper, it is worth nothing that these methods could run potentially faster, but only if their convergence is rapid. For the CGLS method, this means that a pre-conditioner must be chosen correctly.

In Section 2, we describe the well-known normal equations and the problems they present while solving sparse least squares problems. This motivates

our discussion on Björck and Duff's algorithm in Section 3. In Sections 4 and 5, we take a deep dive into the strategies used by the authors to ensure stability and preserve the sparsity of the linear system. In Section 6, we describe our implementation of the algorithm in MATLAB and the corresponding results obtained.

2. Normal equations and least squares problems

One of the most common ways of finding a solution to a system of linear equations is by solving a system of normal equations

$$A^T Ax = A^T b \tag{2.1}$$

Two most popular methods which solve the least squares problem through normal equations, use the *pseudo-inverse*, or the *Cholesky decomposition* on $A^T A$. Using the pseudo-inverse for $A^T A$ we can get the solution vector as $x = (A^T A)^{-1} A^T b$.

The Cholesky decomposition can be used when A has full rank, and it produces a lower triangular matrix L such that $A^T A = LL^T$. We can then obtain the solution vector x by back-solving two triangular systems,

$$Lc = A^T b \tag{2.2}$$

$$L^T x = c \tag{2.3}$$

Although these methods are computationally very efficient, we might have potential numerical difficulties that could be disastrous in some cases:

- The condition number of $A^T A$ is square of that of A, so we might not get an accurate solution if A itself is poorly-conditioned.
- If A is not of full column rank, then $A^T A$ is singular and the Cholesky algorithm breaks.
- The sparsity of A does not guarantee the sparsity of $A^T A$.

The upshot of all this is that forming and solving the normal equations requires relatively high working precision to guarantee suitable accuracy. But this would often entail an unacceptable increase in storage requirements for very large problems. The principal motivation for alternative methods is to alleviate these numerical difficulties.

3. Björck-Duff's Method

Here we will outline Björk and Duff's method. Before that we provide an insight into how Peters and Wilkinson structured their approach to solving this problem [PW70].

3.1. Peters-Wilkinson Elimination Method

Assume that $\text{rank}(A) = n$. Carrying out Gaussian elimination with row and column interchanges on $A_{m\times n}$ leads to a factorisation of the form

$$P_1 A P_2 = LU \tag{3.1}$$

where, P_1 and P_2 are permutation matrices of order m and n respectively. This results in an $m \times n$ unit lower triangular matrix L, and an $n \times n$ upper triangular matrix U. Recall that if O is an orthogonal $m \times n$ matrix, for $x \in \mathbb{R}^n$, we have,

$$\|Ox\|_2 = \|x\|_2 \text{ and } \left\|x^T O^T\right\|_2 = \|x\|_2 \tag{3.2}$$

Utilising the LU factorisation, the orthogonal invariance of the Euclidean norm and the change of variable $UP_2^T x = y$, we can simplify (1.1) as follows:

$$\begin{aligned} &\min_x \|b - Ax\|_2 \\ &= \min_x \|P_1 b - P_1 Ax\|_2 \\ &= \min_x \left\|P_1 b - LUP_2^T x\right\|_2 \\ &= \min_x \|P_1 b - Ly\|_2 \end{aligned} \tag{3.3}$$

One way to solve the modified equation is to use normal equations $L^T Ly = L^T P_1 b$. The solution x to (1.1) is then recovered by solving the triangular system $UP_2^T x = y$.

Peters and Wilkinson observed that **if we use an appropriate pivoting strategy** in the elimination (i.e., the choice of the permutation matrices), we can control the conditioning of L. Hence, we isolate any ill-conditioning of the matrix A in U (as we will see later, this is because we obtain a bound on the elements of L and not that of U). Thus it should be safe numerically to form and solve $L^T Ly = L^T P_1 b$.

This elimination scheme of Peters and Wilkinson is extended and adapted in the sparse setting by Björck and Duff. When A is sparse, the matrices P_1 and P_2 must be chosen such that

- Sparsity is preserved in matrices L and U;
- The conditioning of L is enhanced; and
- The factorisation must be numerically stable.

3.2. The adapted algorithm

The first part to the algorithm involves splitting the solution x into two parts, one of which does not involve the normal equations, $L^T L y = L^T P_1 b$. Using the partitioning,

$$L = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} \text{ and } P_1 b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{3.4}$$

where, $L_1 \in \mathbb{R}^{n \times n}$ and $b_1 \in \mathbb{R}^n$, define $c \in \mathbb{R}^n$ as the solution to

$$L_1 c = b_1$$

Let the vector $d \in \mathbb{R}^{m-n}$ be defined as $d \coloneqq b_2 - L_2 c$. Now, we observe that

$$P_1 b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} L_1 c \\ L_2 c + d \end{bmatrix} = c \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} + \begin{bmatrix} 0 \\ d \end{bmatrix} = Lc + \begin{bmatrix} 0 \\ d \end{bmatrix} \tag{3.5}$$

Now if x_c is any solution of the system $U P_2^T x = c$, then the residual norm corresponding to it is

$$\begin{aligned} \|b - Ax_c\|_2 &= \|P_1(b - Ax_c)\|_2 \\ &= \left\| Lc + \begin{bmatrix} 0 \\ d \end{bmatrix} - Lc \right\|_2 \\ &= \|d\|_2 \end{aligned} \tag{3.6}$$

Thus, if $\|d\|_2 < \epsilon$, where ϵ is some suitable tolerance, then x_c is a solution to (1.1) with a slightly perturbed right hand side b, and we can stop. This is obtained only at the cost of a forward elimination $L_1 c = b_1$ (to obtain c) and a back substitution $U P_2^T x = c$ (to obtain x).

What should we do if $\|d\|_2$ is larger? Then, we have, for an arbitrary x:

$$\begin{aligned} P_1(b - Ax) &= Lc + \begin{bmatrix} 0 \\ d \end{bmatrix} - LUP_2^T x \\ &= \begin{bmatrix} 0 \\ d \end{bmatrix} - Lz \end{aligned} \tag{3.7}$$

where, $UP_2^T x = c + z$.

Hence, x is a least squares solution of (1.1) if it satisfies $UP_2^T x = c + z$, where z is the solution of:

$$\min_z \left\| \begin{bmatrix} 0 \\ d \end{bmatrix} - Lz \right\|_2$$

Since, we formed L by pivoting on A, the above equation can be solved using the normal equations:

$$L^T L z = L^T \begin{bmatrix} 0 \\ d \end{bmatrix}$$

using the Cholesky decomposition $L^T L = HH^T$.

3.3. Pseudo code

The pseudo code of the algorithm outlined above is given below:

1: Björck-Duff's Elimination Method

Compute the $\mathsf{P}_1\mathsf{A}\mathsf{P}_2 = \mathsf{L}\mathsf{U}$ factorisation to produce sparse U and sparse and well-conditioned L
Split L into $\begin{bmatrix} \mathsf{L}_1 \\ \mathsf{L}_2 \end{bmatrix}$ where $\texttt{shape}(\mathsf{L}_1) = n \times n$
Split $\mathsf{P}_1\mathsf{b}$ into $\begin{bmatrix} \mathsf{b}_1 \\ \mathsf{b}_2 \end{bmatrix}$ where $\texttt{shape}(\mathsf{b}_1) = n \times 1$
Solve $\mathsf{L}_1\mathsf{c} = \mathsf{b}_1$
Solve $\mathsf{U}{\mathsf{P}_2}^T\mathsf{x}_1 = \mathsf{c}$
Set $\mathsf{d} \coloneqq \mathsf{b}_2 - \mathsf{L}_2\mathsf{c}$
if $\|\mathsf{d}\|_2 < \epsilon$ **then**
    Set $\mathsf{x} = \mathsf{x}_1$
else
    Solve $\mathsf{L}^T\mathsf{L}\mathsf{z} = \mathsf{L}^T \begin{bmatrix} 0 \\ \mathsf{d} \end{bmatrix}$
    Solve $\mathsf{U}{\mathsf{P}_2}^T\mathsf{x}_2 = \mathsf{z}$
    Set $\mathsf{x} = \mathsf{x}_1 + \mathsf{x}_2$
end
return x

4. Ensuring stability

One of the main questions is to ensure that our factorisation into L and U is numerically stable. Recall that the active submatrix at step k consists of all $A_{ij}^{(k)}, i \geq k, j \geq k$. The following 4 subsets of elements in the active submatrix will be necessary for the discussion:

S_{pc}	Subset of primary candidates: these are the non-zero entries of the active submatrix which will be searched.
S_{st}	A subset of S_{pc} which contains the elements that satisfy some stability criterion.
S_{sp}	A subset of S_{pc} which contains the elements that satisfy some sparsity criterion.
S_{piv}	$S_{st} \cap S_{sp}$: the set from which the pivot will be chosen.

The definitions S_{sp} and S_{st} need not be independent - one may be a subset of the other. The pivot is chosen from S_{piv}. If this set has more than one element, then sparsity considerations are made again to select the final element, because we would like to reduce cost by improving sparsity as much as possible while stability requirements are still met.

If at some step S_{piv} happens to be empty, then we must relax our sparsity requirements, redefining S_{sp}. Clearly our stability criterion must allow freedom for S_{st} to be large. In practice, we don't need to explicitly compute these 4 sets, because simplified procedures are available.

4.1. Choice of pivoting strategy

The permutation matrices obtained after the LU decomposition of A may be trivial (identity) or non trivial. When the factorisation is carried out using complete pivoting, both the permutation matrices turn out to be non-trivial. In practice, however, P_2 is an identity matrix when we use partial pivoting which tends to be sufficient to retain numerical stability of the factorization, unless the matrix A is singular or nearly so.

When the matrix is sparse, it is important for the factorisation to rely solely on the non-zero entries of the matrix. However new non-zero entries may be introduced in L and U matrices which were not present in A. These

new entries are referred to as ***fill-in***, as a result, $nnz(L+U) \geq nnz(A)$ where nnz denotes the number of non-zero entries.

- While *complete pivoting* guarantees a tight bound for the growth factor, it is not recommended from the viewpoint of sparsity because no freedom is allowed for controlling the fill-in.
- *Partial pivoting* allows us to take some advantage of sparsity, but it is too rigid a scheme; rigidity in the sense of "does it allow freedom for S_{st} to be large?"

The amount of fill-in can be controlled with the matrix ordering performed prior to the factorization and consequently, for the sparse case, both the matrices P_1 and P_2 are non-trivial. P_2 induces the column re-orderings that minimize fill-in and P_1 permutes rows so that pivots selected during the Gaussian elimination guarantee numerical stability. The search for the optimal ordering is an NP-complete problem [Yan81]. Therefore, many heuristics have been devised to find an ordering which approximates the optimal one.

The compromise pivoting strategy used is called threshold pivoting: the pivot is not necessarily the largest in absolute value in the current column (which is the case in the dense codes) but instead, it is just large enough to preserve numerical stability. This makes the pivoting much more efficient, especially with the complex data structures involved in sparse factorization.

All the non-zeros of the active submatrix are primary candidates and belong to S_{pc}. A tolerance u is chosen in the range $0 < u \leq 1$ and S_{st} is defined to be the set of all non-zeros $A_{ij}^{(k)}$ of S_{pc} such that one of the following conditions is met:

$$\begin{aligned} \left|a_{kk}^{(k)}\right| &\geq u \max_{k \leq i \leq m} \left|a_{ik}^{(k)}\right| \text{ or,} \\ \left|a_{kk}^{(k)}\right| &\geq u \max_{k \leq j \leq n} \left|a_{kj}^{(k)}\right| \end{aligned} \tag{4.1}$$

This relaxation allows the selection of smaller elements that are otherwise preferable. However, Duff et al. [DER17] provide that only the first condition is sufficient to ensure stability.

4.2. Growth of matrix elements

Now that we have our pivoting strategy, we do a simple analysis of the effect of using threshold pivoting on the numerical stability of a sparse

factorisation. We assume that each pivot is selected on sparsity grounds, but is required to satisfy the *threshold inequality* (4.1).

At a certain step k where $k < \min(i, j)$, we divide row k by the pivot $a_{kk}^{(k)}$ and store $l_{jk} \equiv a_{jk}^{(k+1)} = \frac{a_{jk}^{(k)}}{a_{kk}^{(k)}}$ in place of $a_{jk}^{(k)}$. The operation performed on $a_{ij}^{(k)}$ during step k using floating point arithmetic is expressed as

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik}u_{kj} + e_{ij}^{(k)}$$

where $u_{kl} = a_{kl}^{(k)}$ for $l > k$ and $e_{ij}^{(k)}$ is the error made by the floating point computations.

Define $a_{ij} = \max_k \left|A_{ij}^{(k)}\right|$ so that $\left|A_{ij}^{(k)}\right| \leq a_{ij}$ for $k \leq \min(i, j)$. In particular, when $k = 1$, the bound holds for the elements of the original matrix A. This inequality therefore implies a bound on the elements of L, and not necessarily that of U.

Our pivoting strategy bounds the modulus of the multipliers or equivalently

$$|l_{ik}| \leq u^{-1}, i > k$$

It also limits the growth in a single column during a single step of the reduction, according to the inequality

$$\begin{aligned}\max_i \left|a_{ij}^{(k+1)}\right| &\leq (1 + u^{-1}\mu_{jk}) \max_i \left|a_{ij}^{(k)}\right| \\ &= (1 + u^{-1})^{\mu_{jk}} \max_i \left|a_{ij}^{(k)}\right|, \ j > k\end{aligned}$$

holds, where

$$\mu_{jk} = \begin{cases} 0 & \text{if } l_{jk} = 0 \\ 1 & \text{if } l_{jk} \neq 0 \end{cases}$$

Hence, the overall growth at step k is given by:

$$\max_i \left|a_{ij}^{(k)}\right| \leq (1 + u^{-1})^{c_j} \max_i |a_{ij}| \tag{4.2}$$

where c_j is the number of non-zeros in the j^{th} column of L.

For an overall bound on $\rho = \max_{i,j,k} \left|a_{ij}^{(k)}\right|$, we find that:

$$\rho \leq (1 + u^{-1})^c \max_{i,j} |a_{ij}| \tag{4.3}$$

where $c = \max_j c_j$. From (4.2) we can see that ρ_j is usually small in the sparse case. Thus we may take $u < 1$ to allow greater freedom of pivot choice while still maintaining some control of stability.

4.3. Choice of the pivot threshold parameter

The choice of the pivot threshold parameter u not only influences the number of candidate pivots that are rejected, but also the stability of the factorisation. A large u means a tight bound on the magnitude of the entries in L at the expense of many entries of S_{pc} getting rejected. A smaller value of u on the other hand potentially reduces the number of rejected candidates, but factorisation may be less accurate.

In extreme cases, the entries of the factors may become unbounded. Commonly used values are $u = 0.1$ and $u = 0.01$, on the basis of extensive numerical experience.

5. Ensuring sparsity

In this section, we focus on local ordering methods for sparse matrices. Local orderings are heuristic. It is difficult to define what we mean by "best" and, given a definition, it is difficult to find such an ordering.

Ordering and local ordering

Ordering for sparse matrices refers to the reordering of rows and columns in a sparse matrix. Local orderings are a specific type of ordering for sparse matrices that consider the structure of the matrix in small sub-blocks. Local orderings aim to reduce the fill-in or the number of non-zero entries that are added to the matrix during factorization.

There are three stages in Peters and Wilkinson's method at which we can lose sparsity present in the original system:

- The original LU decomposition;
- The formation of the reduced normal equations $L^T L$; and
- The Cholesky factorisation of $L^T L$

Any attempt to find the smallest possible fill-in for the whole operation or even in the present computational stage would be unrealistic, as the search would be extensive and expensive. This is because we would not only require the updated sparsity pattern but must compute the fill-in associated with the pivot candidates.

5.1. Markowitz's ordering strategy

The ordering strategy of Markowitz [Mar57] has proved extremely successful for general purpose use when the order is not huge. Consider the active submatrix of all $A_{ij}^{(k)}$ with $i \geq k$ and $j \geq k$. Let $r_i^{(k)}$ and $c_j^{(k)}$ be the number of nonzero elements in row i and column j of $A^{(k)}$, respectively. The **Markowitz cost** of an element $a_{ij}^{(k)}$ is defined as

$$M_{ijk} = \left(r_i^{(k)} - 1\right)\left(c_j^{(k)} - 1\right) \tag{5.1}$$

M_{ijk} is equal to the number of matrix elements that will change value from $A^{(k)}$ to $A^{(k+1)}$ if $a_{ij}^{(k)}$ is chosen as the pivotal element. It is thus an upper bound for the amount of fill-in which can be produced if we choose $a_{ij}^{(k)}$. Let

$$M_k = \min\{M_{ijk} \mid i, j \geq k\}$$

The original Markowitz strategy amounts to, at any stage k, choosing a pivotal element with Markowitz cost M_k. This will not necessarily mean that we minimize the fill-in at stage k, but it is considerably easier to compute the Markowitz cost than to compute the amount of fill-in for each element in the active submatrix $A_{ij}^{(k)}$, and in practice, it is almost as good.

An advantage of using (5.1) rather than $r_i^{(k)} c_j^{(k)}$ is that this forces the algorithm to select a row singleton ($r_i^{(k)} = 1$) or a column singleton ($c_j^{(k)} = 1$) if either is present. Such a choice produces no fill-in at all.

How may we interpret this strategy? There are three possible ways:

- Finding the pivot that, after the first $k-1$ pivots have been chosen, modifies the least number of coefficients in the remaining submatrix.
- An approximation to the local minimum multiplication count, since using $a_{ij}^{(k)}$ as pivot requires $r_i^{(k)}(c_j^{(k)} - 1)$ multiplications; and
- An approximation to the choice of pivot that introduces the least fill-in at this stage.

However, to implement the Markowitz strategy, we would require knowledge about the updated sparsity pattern of the submatrix $A_{ij}^{(k)}$ at each stage of the elimination. As a result if we do not limit our search intelligently, it could require examining all the entries in the active submatrix at every stage. To limit the search, Curtis and Reid propose ordering the rows and the columns in increasing order of the number of non-zero elements and show that the search may be stopped rather quickly.

However since we are more concerned with keeping the number of non-zeros in L low, we can use a simpler strategy at stage $k\,(1 < k < n)$; select the column with the minimum $c_j^{(k)}$ and then choose the entry in that column in the row with the minimum $r_i^{(k)}$ among other entries in that column. This is called the **min. row in min. col.** strategy. The drawback is that although the pivot is in a column with few entries, it may be in a row with many entries.

A third criterion is provided, aiming to avoid creating non zeros in $L^T L$. Let s_i denote the number of non-zeros in the $i^{\text{th}}\,(k \leq i \leq m)$ row of L. An upper bound on the number of non-zeros added to the lower triangular part of $L^T L$ when the pivot is chosen from column j is

$$k_j = \sum_{i \in B_j} (s_i + 1) \tag{5.2}$$

where B_j is the set of row indices of the nonzero elements in column j. The column which minimises k_j is chosen as the pivotal column, and then the nonzero in this column with the least non-zeros in its row is taken as pivot, subject to satisfying the numerical criterion.

5.2. Minimum-degree ordering algorithm

How might we preserve sparsity when performing the Cholesky decomposition of $L^T L$ (see line 10 of the pseudo code)? First, observe that this matrix is symmetric positive definite and we can be sure that the diagonal pivots produce a stable factorization. Two things simplify. We do not have to carry numerical values to check for stability, and the search for the pivot is simplified to finding i such that:

$$r_i^{(k)} = \min_t r_t^{(k)}$$

leading to $a_{ii}^{(k)}$ as the pivot.

The algorithm is a symmetric analogue of Markowitz' method and was first proposed by Tinney and Walker [TW67] as **scheme 2**. Rose [Ros72] developed a graph theoretical model of Tinney and Walker's algorithm and renamed it the **minimum degree algorithm**, since it performs its pivot selection by choosing from a graph a node of minimum degree.

We now present the original minimum degree algorithm of Tinney and Walker in the context of the graph model of Rose. For a more detailed

history of the algorithm, we refer the reader to the survey of George and Liu [GL89].

Graph representation of a sparse matrix

The nonzero pattern of a symmetric n by n matrix A, can be represented by a graph $G^0 = (V^0, E^0)$, with nodes $V^0 = \{1, \ldots, n\}$ and edges E^0 such that an edge (i, j) is in E^0 *if and only if* $a_{ij} \neq 0$ and $i \neq j$. Since A is undirected, the graph G^0 is undirected.

Elimination graph

The **elimination graph** $G^k = \left(V^k, E^k\right)$, describes the nonzero pattern of the submatrix still to be factorised after the first k pivots have been chosen and eliminated. It is undirected, since the matrix remains symmetric as it is factorised. At step k, the graph G^k depends on G^{k-1} and the selection of the k^{th} pivot. To find G^k, the k^{th} pivot node p is selected from V^{k-1}. Edges are added to E^{k-1} to make the nodes adjacent to p in G^{k-1} a *clique* (a fully connected subgraph).

This addition of edges (fill-in) means that we cannot know the storage requirements in advance. The edges added correspond to the fill-in caused by the k^{th} step of factorisation. In this context, a fill-in is a nonzero entry L_{ij} where $(PAP^T)_{ij}$ is zero. The pivot node p and its incident edges are then removed from the graph G^{k-1} to yield the graph G^k.

Let $\mathsf{Adj}_{G^k(i)}$ denote the set of nodes adjacent to i in the graph G^k. When the k^{th} pivot is eliminated, the graph G^k is given by

$$\begin{aligned} V^k &= V^{k-1} \setminus \{p\} \\ E^k &= \left(E^{k-1} \cup (\mathsf{Adj}_{G^k}(p) \times \mathsf{Adj}_{G^k}(p))\right) \cap \left(V^k \times V^k\right) \end{aligned} \tag{5.3}$$

The minimum degree algorithm selects node p as the k^{th} pivot such that the degree of p, $t_p \equiv |\mathsf{Adj}_{G^{k-1}}(p)|$, is minimum (where $|\cdots|$ denotes the size of a set or the number of non-zeros in a matrix, depending on the context).

By minimizing the degree, the algorithm minimises the upper bound of the fill-in caused by the k^{th} pivot. Selecting p as the pivot creates at most $\binom{t_p}{2}$ new edges in G. Notice that the diagonal entry of minimum degree will always have Markowitz count equal to the minimum for any diagonal or off-diagonal entry. This property is maintained by choosing pivots from the diagonal, since symmetry is preserved.

The costliest part of the original minimum degree algorithm is the recomputation of the degrees of nodes adjacent to the current pivot element. Rather than keep track of the exact degree, the **approximate** minimum degree algorithm [ADD96; ADD04] finds an upper bound on the degree that is easier to compute. Rather surprisingly, this often produces a better ordering than minimum degree and leads to substantial savings in run time, particularly for very irregularly structured matrices. It has no effect on the quality of the ordering.

6. Implementation

In implementing this algorithm, we used the MATLAB programming language because of its vast array of sparse matrix routines that we could leverage in implementing this algorithm. Furthermore, we used the University of Florida sparse matrix collection [DH11] for some test sparse matrices that we could implement our algorithm on. The following table shows the outcome of our experiments on three such test matrices:

Matrix	**Dimensions**	$\|PAQ - LU\|_1$	$\|d\|_2$	$\|Ax - b\|_2$
ash219	219×85	14.000	107.771	63.556
well1033	1033×320	28.565	1.274	10740.945
illc1850	1850×712	32.194	8.014	17324.772

Table 1: Results on some test sparse matrices

We used the right hand side b vector if it was provided along with the matrix (typically, when the corresponding application domain was to solve a least square problem). This was the case with *well1033* and *illc1850*. For *ash219*, we generated b by applying a small perturbation to the vector obtained by summing each row of the matrix.

Furthermore, we set a threshold of $\epsilon = 100$ that determines whether or not our system is nearly consistent. Because of the high value of $\|d\|_2$ in the case of *ash219*, we proceed with the Cholesky decomposition of LL^T.

The sparsity patterns of the original matrix and the L and U factors are shown in the appendix. The values of the number of non-zero elements confirm our earlier claim that $nnz(L + U) \geq nnz(A)$.

References

[HS+52] Magnus R Hestenes, Eduard Stiefel, et al. "Methods of conjugate gradients for solving linear systems". In: *Journal of research of the National Bureau of Standards* 49.6 (1952), pp. 409–436.

[Mar57] Harry M Markowitz. "The elimination form of the inverse and its application to linear programming". In: *Management Science* 3.3 (1957), pp. 255–269.

[TW67] William F Tinney and John W Walker. "Direct solutions of sparse network equations by optimally ordered triangular factorization". In: *Proceedings of the IEEE* 55.11 (1967), pp. 1801–1809.

[PW70] Gwen Peters and James Hardy Wilkinson. "The least squares problem and pseudo-inverses". In: *The Computer Journal* 13.3 (1970), pp. 309–316.

[Ros72] Donald J Rose. "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations". In: *Graph theory and computing*. Elsevier, 1972, pp. 183–217.

[BD80] Åke Björck and Iain S Duff. "A direct method for the solution of sparse linear least squares problems". In: *Linear Algebra and its Applications* 34 (1980), pp. 43–67.

[Yan81] Mihalis Yannakakis. "Computing the minimum fill-in is NP-complete". In: *SIAM Journal on Algebraic Discrete Methods* 2.1 (1981), pp. 77–79.

[PS82] Christopher C Paige and Michael A Saunders. "LSQR: An algorithm for sparse linear equations and sparse least squares". In: *ACM Transactions on Mathematical Software (TOMS)* 8.1 (1982), pp. 43–71.

[GL89] Alan George and Joseph WH Liu. "The evolution of the minimum degree ordering algorithm". In: *SIAM review* 31.1 (1989), pp. 1–19.

[ADD96] Patrick R Amestoy, Timothy A Davis, and Iain S Duff. "An approximate minimum degree ordering algorithm". In: *SIAM Journal on Matrix Analysis and Applications* 17.4 (1996), pp. 886–905.

[ADD04] Patrick R Amestoy, Timothy A Davis, and Iain S Duff. "Algorithm 837: AMD, an approximate minimum degree ordering algorithm". In: *ACM Transactions on Mathematical Software (TOMS)* 30.3 (2004), pp. 381–388.

[DH11] Timothy A Davis and Yifan Hu. "The University of Florida sparse matrix collection". In: *ACM Transactions on Mathematical Software (TOMS)* 38.1 (2011), pp. 1–25.

[DER17] Iain S Duff, Albert Maurice Erisman, and John Ker Reid. *Direct methods for sparse matrices*. Oxford University Press, 2017.

A. Figures

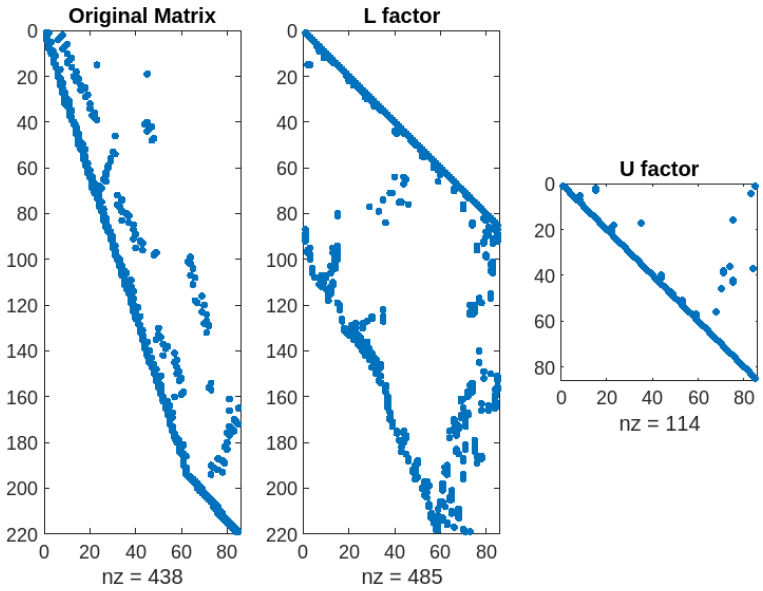

Figure 1: Sparsity patterns for *ash219*

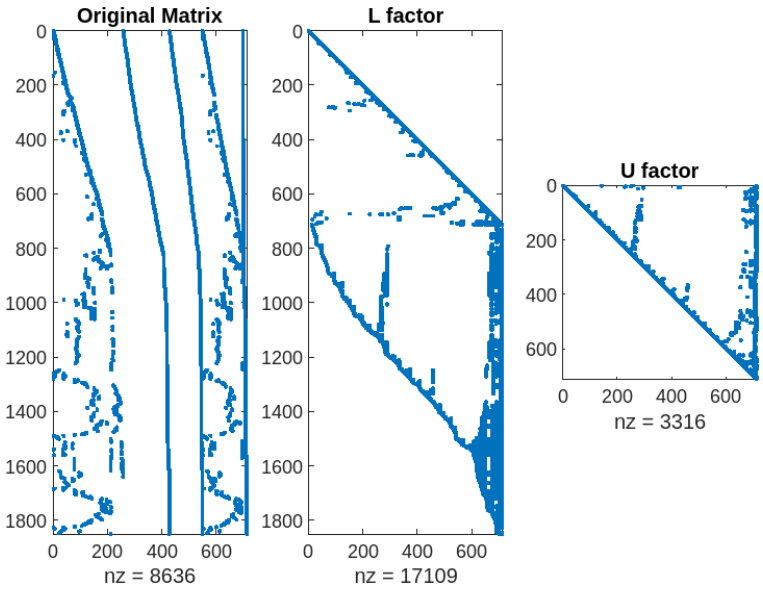

Figure 2: Sparsity patterns for *illc1850*