ILUT : A dual threshold incomplete LU factorization

Research presentation for SC, Monsoon ’25

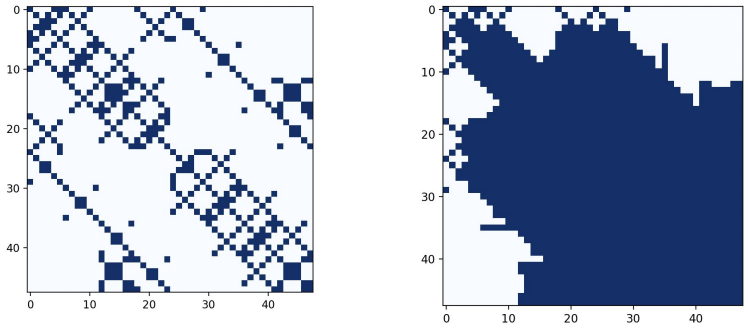
Himanshu (2023241)
Siddharth (2023524)

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

INTRODUCTION

Often in modelling physical situations like refineries, power grids, bridges and dams, we end up with solving $Ax = b$,
where A is large, square, non-singular but sparse. For example, a matrix for modelling oil reservoir was of size 1080 * 1080, but having only 20,000 nonzero entries.

A drawback with using simple LU factorization is that the (L + U) matrix ends up having far more nonzero entries than A. So we decide a compromise between the accuracy of LU and the sparsity of $(L + U)$.



The grid on the left is a 48 x 48 matrix from structural engineering having 400 nozeros (the pixels mean nonzero entries). The grid on the right is it's (L + U) having 1706 nonzeros.

LU Factorization (IKJ Gaussian Elimination)

This is an LU factorization. The algorithm is:

```
LU(A):
// A is a nonsingular n X n matrix with nonzero diagonals
L = I_n // initialize L
for i in [2, ... n]:
        // main loop, we will work on row 'i' of A
        for k in [1, ... i - 1]:
                // make A[i][k] zero
                L[i][k] = A[i][k] / A[k][k]
                A[i][k] = 0
                for j in [k + 1, ... n]:
                        A[i][j] = A[i][j] - (L[i][k] * A[k][j])
// A becomes the U part
```

All the incomplete LU factorizations will essentially be modifications of this algorithm.

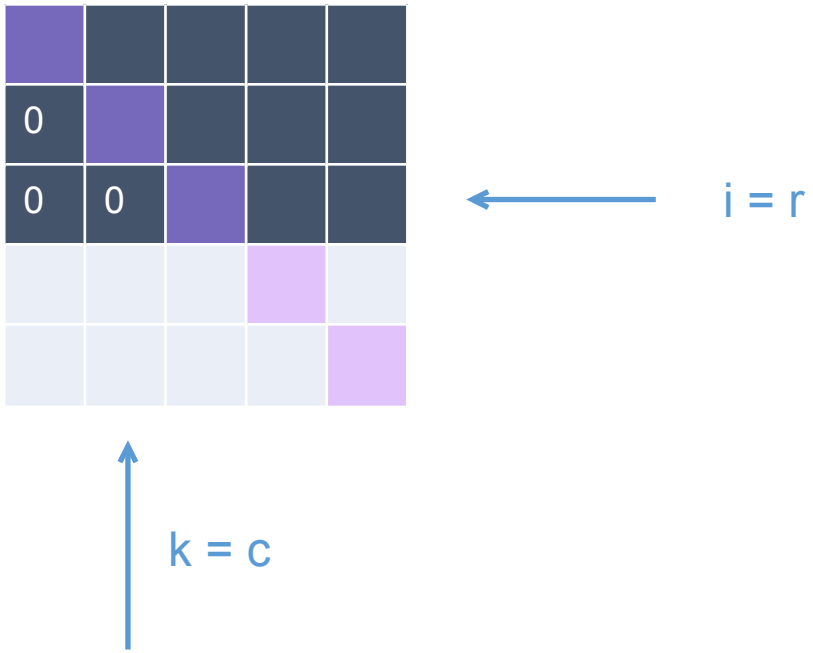


Figure: The A matrix

At the end of the second loop, say k = c, (when i = r), the position (r, c) holds 0. And at the end of this iteration of the main loop, the first i - 1 positions of row i hold 0 and for all the rows inculding and above this row, we have generated their parts of the L and U matrix.

ILU(0) Factorization

This is the simplest ILU factorization. The algorithm is:

```
ILU0(A):
    // A is a nonsingular n X n matrix with nonzero diagonals
    L = I_n // initialize L
    for i in [2, ... n]:
        // main loop, we will work on row 'i' of A
        for k in [1, ... i - 1]:
            // make A[i][k] zero only if it currently isn't
            if A[i][k] != 0:
                L[i][k] = A[i][k] / A[k][k]
                A[i][k] = 0
                for j in [k + 1, ... n]:
                    // keep zero elements untouched
                    if A[i][j] != 0:
                        A[i][j] = A[i][j] - (L[i][k] * A[k][j])
    // A becomes the U part
```

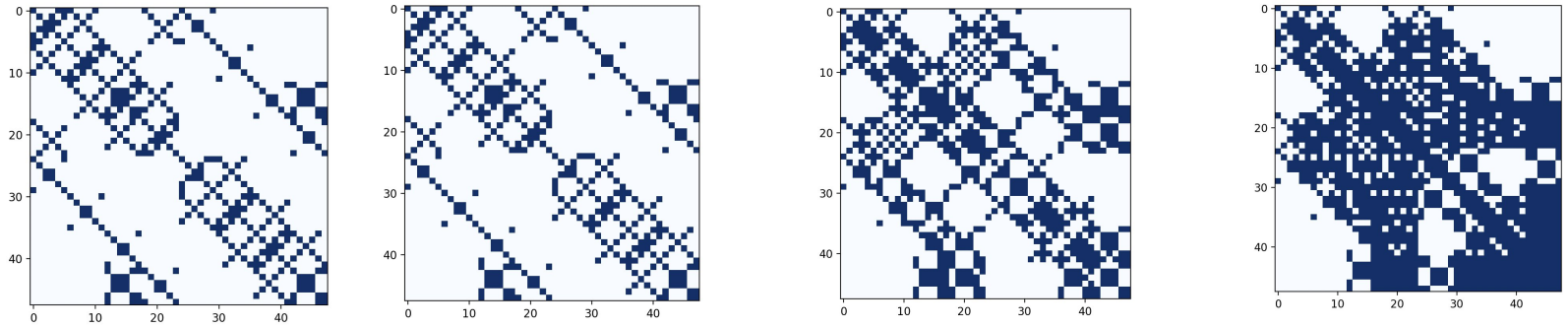
By keeping zero elements of the original matrix untouched, we *preserve the sparsity* of the original A in (L + U).

Level - based ILU(k) Factorization

Such a strict factorization condition in ILU(*0*) may not generate a accurate enough factors. We decide to follow a rule: set an element to zero, if in the elimination step it becomes small enough.

A 'level', say k is assigned to each element. We map each element to ε^k . The smaller an element becomes, the higher it's level goes.
Where $0 < \varepsilon \leq 1$. So initially all nonzero elements have level = *0* while zeros have level ∞.

Apart from initializing the levels of each element, we need to add only two lines to the innermost (j) loop to get this algorithm.



From left to right, we show the sparsity of original matrix, (L + U) of ILU(0), ILU(1) and ILU(2). the third and fourth images have 764 and 1312 nonzeros, respectively.

```
ILU(A, k):
// A is a nonsingular n X n matrix with nonzero diagonals, k is max allowed level
L = I_n // initialize L
level = 0_{n,n}  // initialize levels
for i in [1, ... n]:
        for j in [1, ... n]:
                if A[i][j] == 0:
                        level[i][j] = ∞

for i in [2, ... n]:
        // main loop, we will work on row 'i' of A
        for k in [1, ... i - 1]:
                // make A[i][k] zero
                L[i][k] = A[i][k] / A[k][k]
                A[i][k] = 0
                for j in [k + 1, ... n]:
                        A[i][j] = A[i][j] - (L[i][k] * A[k][j])
                        level[i][j] = min(level[i][j],level[i][j] + level[k][j] + 1)
                        if level[i][j] > k:
                                A[i][j] = 0 // drop it

// A becomes the U part
```

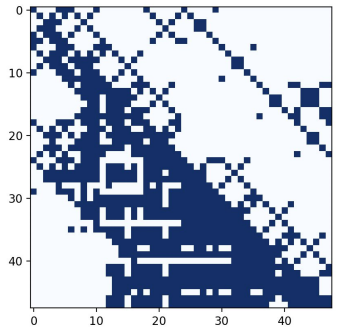
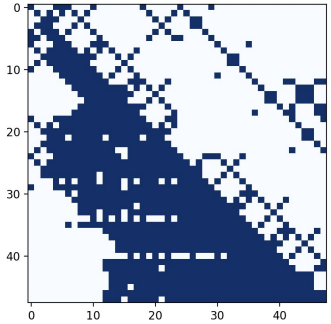
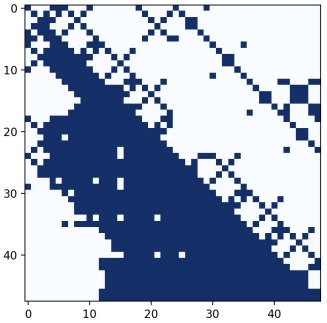
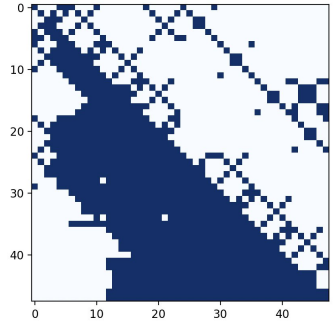
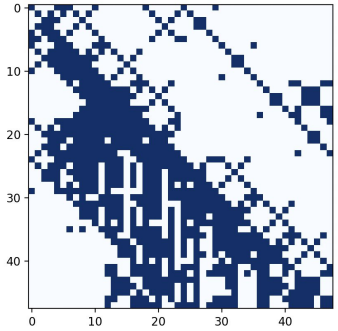
ILU(τ, p) : Dual thresholding

```
for i in [1, ... n]:
      w = A[i][*] // initialize w to be a copy of the i-th row of A
      norm_wl = norm2(w[1], w[2], ... w[i - 1])

      for k in [1, ... i - 1]:
            if w[k] != 0:
                  // first dropping rule
                  if abs(w[k]) < τ * norm_wl:
                        w[k] = 0
        else:
           w[k] = w[k] / U[k][k]
           for j in [k + 1, ... n]:
              w[j] = w[j] - (w[k] * U[k][j])

   // second dropping rule
   drop_with_threshold(w, p - 1, tau, 1, i)
   drop_with_threshold(w, p - 1, tau, i + 1, n + 1)
   // in each row of L and U, keep atmost ‘p’ absolute largest
   elements

   for j in [1, ... i - 1]: L[i][j] = w[j] // write to L
   for j in [i, ... n]: U[i][j] = w[j] // write to U
```

We list the results of ILU(tau, p=48) for tau = 10^{-5}, 10^{-6}, 10^{-7} and 10^{-8} and for ILU(tau=10^{-6}, p=20). In the third and fourth images, there are 1039 and 1050 nonzeros respectively.

Tau guides the dropping threshold, while p controls number of nonzero entries per row in L and U.

Theortical Analysis:

- If the diagonal elements of the original matrix are positive while the off-diagonal elements are negative, then under certain conditions of diagonal dominance the matrices generated during the elimination will have the same property.
- If the original matrix is diagonal dominant then the transformed matrices will also have the property of being diagonally dominant under certain conditions.

These properties will be analyzed in detail in this section.

- Row evolution definition: "We denote the row vector resulting from line 4 of Algorithm 3.2 by $u_i^{k+1,*}$."
  At the start of elimination step k, the working row is $u_i^{k,*}$, After processing column k, the new version of the row is $u_i^{k+1,*}$
  For positions to the left of the pivot ($j \leq k$), those entries are set to zero (they're eliminated): $u_{ij}^{k+1,*} = 0, for\ j \leq k$
- The elimination formula — Equation (2) $l_{ik} = \frac{u_{ik}^k}{u_{kk}^k}$
  This is the standard elimination multiplier — same as in Gaussian elimination:
  u_{kk}^k is the pivot (the diagonal entry of the current pivot row).
  u_{ik}^k is the element below that pivot being eliminated.
- Drop small multiplier -- If l_{ik} small enough, set l_{ik}=0.
  If the multiplier is below the drop tolerance threshold, it's discarded.
  This avoids storing very small numbers that would only add fill-ins and increase computational cost.
- Update of the working row — Equation (3) $u_{ij}^{k+1,*} = u_{ij}^{k,*} - l_{ik}u_{kj}^{k,*} - r_{ij}^k,\ \ j = k+1, \ldots, N$
  Subtract l_{ik} times the pivot row $u_k^{k,*}$ from the current row $u_i^{k,*}$.

The last term, r_{ij}^{k}, is a correction for dropped elements:

- $r_{ij}^{k} = 0$ if nothing is dropped.
- or equals the fill-in entry that was removed due to the drop rule.

- Initialization "Initially $u_i^{1,*} = a_i$" That is, at the start (before any elimination), the working row is simply the i-th row of A.
- " r_{ij}^{k} is an element subtracted from a fill-in element which is being dropped."
  When a new nonzero (a "fill-in") appears during elimination, If its magnitude is below the tolerance threshold, It's not kept in the factorization, Instead, the algorithm records this omission as a correction term. So r_{ij}^{k} represents the dropped parts of the computation.
- End of the i-th step — Equation (4) After finishing elimination for row i: $u_{i,*} = u_{i,*}^{i-1,*}$ That is, the final row of U (upper triangular factor) is the result of eliminating all previous rows.
- Relation between $a_{i,}$ *U and L*. At this point, the paper gives the algebraic relationship: $a_{i,*} = \sum_{k=1}^{i} l_{ik} u_{k,*}^{k,*} + r_{i,*}$
  where $r_{i,*} = \sum_{k=1}^{i} r_i^{(k)}$ is the total sum of dropped fill-in corrections.
  This equation expresses how the original row a_i is related to the earlier rows u_k of U, the multipliersl_{ik}, and the accumulated drop error r_i.
- The existence result which we will prove will only be valid for certain modifica- tions of the basic ILUT(p,τ) described earlier. We will consider an ILUT strategy which uses one of the two following modifications,

Modification 1 Elements generated in the original nonzero pattern of A are not subject to the dropping rule, regardless of their size. Elements in other locations are subject to the same dropping rule as before.
Modification 2 For any i < N let a; j, be the element of largest modulus among the elements a;j, j = i+ 1,...N. Then elements generated in position (i, j;) during the ILUT procedure are not subject to the dropping rule.

- The two modifications are safeguards to prevent ILUT from "zeroing out" crucial elements during dropping.
- The first is strict — keep every original nonzero.
- The second is lighter — just keep the biggest one per row.
- Since the first automatically satisfies the second, the paper only proves the theory for the second case, as it's both sufficient and more general.

Following axelsson and barker [3], Before the theorem, the paper defines the type of matrix for which the proof holds :-

- $\widehat{M}$ Matrix : A matrix H is called an $\widehat{M}$ matrix if it`s entries h_{ij} satisfy three conditions. the matrix is denoted as H = $[h_{ij}]$

1. $h_{ii} > 0$ *for* $1 \leq i \leq N$, *and* $h_{NN} \geq 0$
   - All diagonal entries (except the last one) are strictly positive.
   - the last diagonal entries can be non-negative (≥ 0)
   - this ensures that the matrix has positive pivots - a key property for LU factorization.

2. $h_{ij} \leq 0 \; for \;\; i, j = 1, \ldots, N \; and \; i \neq j$

- All off-diagonal entries are nonpositive (either zero or negative).
- This ensures that the matrix has an M-matrix structure — positive diagonals and negative off-diagonals.
- Intuitively: each variable's "self-effect" (diagonal) is positive, while its "influence" on others (off-diagonals) is opposing.

3. $\sum_{j=i+1}^{N} h_{ij} < 0, for \; 1 \leq i < N$

- The third condition is simply a requirement that there be at least one nonzero element to the right of the diagonal element, in each row except the last.
- So, for every row except the last:
  - there must be at least one connection to a column to the right of the diagonal,
  - meaning the matrix cannot be strictly lower triangular — it must have some nonzero above the diagonal.

- The row-sum for the i-th row is defined by $rs(h_{i,*}) = \sum_{j=1}^{N} h_{ij}$

A given row of an M matrix H is diagonally dominant M if its row-sum is nonnegative. matrix H is said to be diagonally dominant if all its rows are diagonally dominant.

Theorem 4.1

The underlying asumption is that an ILUT strategy is used with modification 2.

Theorem 4.1: Existence Result for ILUT

This theorem's proof relies on using a modified ILUT strategy (called Modification 2). This modification states that during the factorization, the algorithm is not allowed to drop (i.e., set to zero) any element being generated in the position (i, j_i), where a_{i,j_i} was the off-diagonal element with the largest magnitude in the original row i.

Theorem Statement: If the matrix A is a diagonally dominant $\widehat{M}$-matrix, then the rows $u_{i,*}^k$ (which are the rows being computed at each step k of the factorization) satisfy the following three relations for k=0, 1, ..., i-1:

1. $u_{ij}^{k+1} \leq 0 \; for \; j \neq i$ (Off-diagonals remain non-positive).
2. $rs(u_{i,*}^{k+1}) \geq rs(u_{i,*}^k) \geq 0$ (Row-sums remain non-negative, i.e., diagonal dominance is preserved).
3. $u_{ii}^{k+1} > 0 \; for \; i < N \; and \; u_{NN}^{k+1} \geq 0$ (Diagonal elements remain positive).

Proof

The proof is done by induction on k. It is noted that the relations are all true for k=0, since the initial row $u_{i,*}^1$ is just the original matrix row $a_{i,*}$, which is a diagonally dominant $\widehat{M}$-matrix by assumption.

The core of the induction step uses the update formula from the algorithm:

$$u_{i,*}^{k+1} = u_{i,*}^k - l_{ik} u_{k,*} - r_{i*}^k \qquad \text{where } r_{i*}^k \text{ is the row of elements being dropped.}$$

1. **proof of** $u_{ij}^{k+1} \leq 0$ ($for\ j \neq i$)

- By the induction hypothesis, we assume the properties hold for step k. This means $l_{ik} = u_{ik}^{k}/u_{kk} \leq 0$ ($since\ u_{ik}^{k} \leq 0\ and\ u_{kk} > 0$). *we also know* $u_{k,j} \leq 0$ ($for\ j \neq k$).
- There are two cases for an element u_{ij}^{k+1} :
  - Case 1: **The element is dropped.** If the element is dropped, it is replaced by zero. Thus, $u_{ij}^{k+1} = 0$, which satisfy $u_{ij}^{k+1} \leq 0$.
  - Case 2: **The element is not dropped.** The update is $u_{i,j}^{k+1} = u_{i,j}^{k} - l_{ik}u_{k,j}$. since $u_{ij}^{k} \leq 0$, $l_{ik} \leq 0$, *and* $u_{k,j} \leq 0$, the term $l_{ik}u_{k,j}$ is non-negative. Subtracting it ($l_{ik} - u_{k,j}$) makes it non positive. therefore, u_{ij}^{k+1} is the sum of two non-positive value, must be ≤ 0.
- In both cases, the relation holds. This also implies that any dropped element r_{ij}^{k} must be ≤ 0 .

2. **proof of** $rs(u_{i,*}^{k+1}) \geq rs(u_{i,*}^{k}) \geq 0$

- we take the row sum of the updated formula $rs(u_{i,*}^{k+1}) = rs\,(u_{i,*}^{k}) - l_{ik}rs\,(u_{k,*}) - rs(r_{i*}^{k})$.
- lets analyze the terms:
  - by the induction hypothesis, $rs(u_{i,*}^{k}) \geq 0\ and\ (u_{k,*}) \geq 0$.
  - we also know $l_{ik} \leq 0$.

- as established in the previous step, all dropped elements are non-positive ($r_{ij}^k \leq 0$), so their sum $rs(r_{i*}^k)$ must also be ≤ 0.

- now look at the equation again:
  - the term $-l_{ik}rs(u_{k,*})$ *must be* ≥ 0 *since* $l_{ik} \leq 0$ *and* $rs(u_{k,*}) \geq 0$
  - the term $-rs(r_{i*}^k)$ *must be* ≥ 0 (*since* $rs(r_{i*}^k) \leq 0$).

- this means $rs(u_{i,*}^{k+1})$ is equal to $rs(u_{i,*}^k)$ (which is ≥ 0) plus two non-negative terms. therefore, $rs(u_{i,*}^{k+1}) \geq rs(u_{i,*}^k)$. *this proves the relation.*

3. **proof of** $u_{ii}^{k+1} > 0$ (***for*** $i < N$)
   - from the relation we just proved (2), we know the row sum $rs(u_{i,*}^{k+1}) \geq 0$.
   - the row sum can be written as $u_{ii}^{k+1} + \sum_{j \neq i} u_{ij}^{k+1}$.
   - therefore, $u_{ii}^{k+1} = rs(u_{i,*}^{k+1}) - \sum_{j \neq i} u_{ij}^{k+1}$.
   - from relation (1), we know all u_{ij}^{k+1} ($for\ j \neq i$) are ≤ 0. this means $-\sum_{j \neq i} u_{ij}^{k+1}$ is the same as $\sum_{j \neq i} |u_{ij}^{k+1}|$.
   - so, $u_{ii}^{k+1} \geq \sum_{j \neq i} |u_{ij}^{k+1}|$.
   - this sum must be greater than and equal to any single element in it, so $u_{ii}^{k+1} \geq |u_{i,j_i}^{k+1}|$, *where* j_i is the position of the element from the modification 2 assumption.

- because this element is never dropped, its magnitude can only increase or stay the same during the updates. So, $|u_{ii}^{k+1}| \geq |u_{ii}^{1}| = |a_{i,j_i}|$.
- by the definition of an $\widehat{M}$ Matrix (condition 3), $|a_{i,j_i}|$ must be positive for i < N.
- therefore, $u_{ii}^{k+1} \geq |a_{i,j_i}| > 0$, which proves $u_{ii}^{k+1} > 0$. (for the final row i = N, the proof just shows $u_{NN}^{k+1} \geq 0$).

computational details:

- **ILU with fill-in matrix**

Calculation	Error Norm	Time Taken
ILU(0)	$2.17589193 * 10^{8}$	0.45 ms
ILU(3)	$1.0085219 * 10^{7}$	10 ms
ILU(4)	$3.5056 * 10^{4}$	9.86 ms
ILU(5)	$1.095 * 10^{-6}$	9.83 ms
ILU(6)	$1.095 * 10^{-6}$	9.8 ms

- **ILU(tau, p)**

Calculation of ILU(tau, p = 48) for different tau	Error Norm	Time Taken
10^{-5}	$2.982 * 10^{6}$	2.47 ms
10^{-6}	$1.429 * 10^{5}$	2.65 ms
10^{-7}	$6.727 * 10^{3}$	2.56 ms
10^{-8}	$8.2 * 10^{1}$	2.57 ms

References

- ILUT : A dual incomplete LU factorization, Y Saad 1994
- Iterative Methods for Sparse Linear Systems