MA3227 Numerical Analysis II
Lecture 4: Sparse LU Factorisation

Simon Etter

National University
of Singapore

Semester II, AY 2020/2021

Sparse LU Factorisation

Problem statement

> Solve $Ax = b$ via LU factorisation in less than $O(n^3)$ operations, assuming $A \in \mathbb{R}^{n \times n}$ is sufficiently sparse.

Why sparse LU factorisation?

LU factorisation is often the best algorithm for solving linear systems due to a number of reasons:

- ▶ It is the fastest known algorithm for solving dense linear systems.
- ▶ It can detect whether A is close to singular and if so warn the user that the numerical solution to $Ax = b$ is likely to be inaccurate.
- ▶ Unlike most other algorithms in numerical analysis, LU factorisation does not introduce an effort parameter which would require us to compromise between accuracy and performance.

The "only" drawback of LU factorisation is that the $O(d^3)$ scaling of the standard algorithm stops us from solving many of the linear systems that we would like to solve (e.g. the discrete Poisson equation from Lecture 3). Exploiting sparsity allows us to overcome this drawback (to some extent).

Sparse LU Factorisation

Let us begin our discussion by recapitulating the main ideas of the standard, dense algorithms.

Thm: LU factorisation

Any matrix $A \in \mathbb{R}^{n \times n}$ can be written as $PA = LU$, where

- $P \in \mathbb{R}^{n \times n}$ is a permutation matrix,
- $L \in \mathbb{R}^{n \times n}$ is lower-triangular with unit diagonal, and
- $U \in \mathbb{R}^{n \times n}$ is upper-triangular.

The matrices L and U are unique for fixed P if A is invertible.

Proof. See any linear algebra textbook.

I will discuss the definition and meaning of the permutation factor P on slide 11. For now, let us ignore this factor and instead address the following questions:

- How do we compute $A = LU$?
- How do we use the $A = LU$ to solve $Ax = b$.

Sparse LU Factorisation

Computing the LU factorisation

Main ideas:

- ▶ Use the top left entry to eliminate all entries below it.
  This top left entry is called pivot.
- ▶ Use the L-factor to keep track of the elimination factors.

Example.

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & 1 & -2 \\ -8 & 2 & 3 \\ 12 & 7 & -5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & 1 & -2 \\ 0 & 4 & -1 \\ 0 & 4 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & 1 & -2 \\ 0 & 4 & -1 \\ 0 & 4 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & 1 & 1 \end{pmatrix} \begin{pmatrix} 4 & 1 & -2 \\ 0 & 4 & -1 \\ 0 & 0 & 2 \end{pmatrix} = LU$$

Sparse LU Factorisation

Solving linear systems

The purpose of the LU factorisation is to reduce an arbitrary linear systems to two triangular system,

$$Ax = LUx = b \qquad \Longleftrightarrow \qquad Ly = b, \quad Ux = y$$

Doing so is useful because triangular systems are easy to solve.

Example

$$\begin{pmatrix} 4 & 1 & -2 \\ & 2 & -1 \\ & & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \\ 6 \end{pmatrix}$$

Third eq.:	$3x_3 = 6$	$\Longrightarrow$	$x_3 = 2$
Second eq.:	$2x_2 - x_3 = 4$	$\Longrightarrow$	$x_2 = 3$
First eq.:	$4x_1 + x_2 - 2x_3 = 3$	$\Longrightarrow$	$x_1 = 1$

Sparse LU Factorisation

Terminology: Back substitution

The above algorithm is known as *back substitution*.

Remark. Some authors distinguish between back substitution for solving upper-triangular systems and forward substitution for solving lower-triangular systems. I will call both of these algorithms back substitution because they are fundamentally the same algorithm.

Sparse LU Factorisation

The runtimes of sparse LU factorisation and back substitution are easily derived from the above algorithm sketches.

Thm: Runtime of sparse LU factorisation

The runtime of computing the LU factorisation $LU = A$ is

$$O\left(\sum_{k=1}^{n} \text{nnz}(L[:, k]) \, \text{nnz}(U[k, :])\right).$$

: is a shorthand notation for $\{1, \ldots, n\}$.

Thm: Runtime of sparse back substitution

The runtime of solving $Ly = b$ and $Ux = y$ via back substitution is

$$O\left(\text{nnz}(L) + \text{nnz}(U)\right).$$

Proofs of these results will be provided on the following slides.

Sparse LU Factorisation

Proof of LU runtime.

Let us denote by $U^{(k)}$ the state of the U-factor just before eliminating the lower-triangular entries in the kth column (see slide 4).

The LU factorisation can then be formulated as follows.

```
1: for k = 1, ..., n do
2:     for i = k + 1, ..., n such that U^(k)[i, k] ≠ 0 do
3:         L[i, k] = U^(k)[i, k]/U[k, k]
4:         for j = k + 1, ..., n such that U[k, j] ≠ 0 do
5:             U^(k+1)[i, j] = U^(k)[i, j] − L[i, k] U[k, j]
6:         end for
7:     end for
8: end for
```

This immediately yields a runtime of

$$O\left(\sum_{k=1}^{n} \text{nnz}(L[:, k])\, \text{nnz}(U[k, :]) \right).$$

Sparse LU Factorisation

Proof of back substitution runtime.

Back substitution applied to $Ux = y$ proceeds as follows.

```
1: for i = n, ..., 1 do
2:     x[i] = y[i]
3:     for j = i + 1, ..., n such that U[i, j] ≠ 0 do
4:         x[i] = x[i] − U[i, j] x[j]
5:     end for
6:     x[i] = x[i]/U[i, i]
7: end for
```

We observe that line 4 is executed exactly once for each nonzero entry of U, and the runtimes of all other lines are negligible in the big-O sense. The runtime of solving $Ux = y$ is thus $O\big(\mathrm{nnz}(U)\big)$, and likewise we conclude that the runtime of $Ly = b$ is $O\big(\mathrm{nnz}(L)\big)$.

Sparse LU Factorisation

Remarks on the LU and back substitution runtime estimates

The above runtime estimates deserve a few more remarks.

▶ If L and U are dense, then the runtime of the LU factorisation is

$$O\left(\sum_{k=1}^{n} \text{nnz}(L[:,k])\, \text{nnz}(U[k,:])\right) = O\left(\sum_{k=1}^{n} k^2\right) = O(n^3),$$

and the runtime of back substitution is

$$O\big(\text{nnz}(L) + \text{nnz}(U)\big) = O(n^2).$$

The above results thus include the well-known runtimes for dense factorisation and back substitution as a special case.

▶ The runtime of LU factorisation is at least as large as that of back substitution since $\text{nnz}(L[:,k]), \text{nnz}(U[k,:]) \geq 1$ and thus

$$\sum_{k=1}^{n} \text{nnz}\big(L[:,k]\big)\, \text{nnz}\big(U[k,:]\big) \geq \begin{cases} \sum_{k=1}^{n} \text{nnz}\big(L[:,k]\big) = \text{nnz}(L), \\ \sum_{k=1}^{n} \text{nnz}\big(U[k,:]\big) = \text{nnz}(U). \end{cases}$$

This point explains why we will be talking about LU factorisation rather than back substitution for most of this lecture.

Sparse LU Factorisation

Why the LU theorem involves a permutation matrix

Let us now return to the question of why the LU theorem from slide 3 involves a permutation factor P.

It turns out that this factor is needed to swap rows in case the top-left entry is zero and hence cannot be used as a pivot element.

Example.

$$\begin{pmatrix} 0 & 3 \\ 1 & 2 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix}.$$

In the language of linear algebra, "swapping rows" corresponds to replacing

$$Ax = b \qquad \text{with} \qquad (PA)\, x = Pb$$

where P is a permutation matrix defined as follows.

Sparse LU Factorisation

Def: Permutations and permutation matrices

A *permutation* is a bijective map $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$.

The *permutation matrix* associated with a permutation π is a matrix $P \in \mathbb{R}^{n \times n}$ such that $(Pb)[i] = b[\pi(i)]$ for all $b \in \mathbb{R}^n$.

Permutations are conveniently written as a vector $p = [\pi(1), \ldots, \pi(n)]$.

Example

i	1	2	3
$\pi(i)$	1	3	2

$\longleftrightarrow \quad P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad \longleftrightarrow \quad p = [1, 3, 2].$

Useful facts

▶ A matrix P is a permutation matrix if and only if all its entries are zero except for exactly one 1 in each row and column.

▶ We have $(PA)[i, j] = A[\pi(i), j]$; hence $A \mapsto PA$ indeed corresponds to swapping rows.

Sparse LU Factorisation

Why the LU theorem involves a permutation matrix (continued)

The example on slide 11 demonstrates that swapping rows is sometimes necessary to make the LU factorisation work. One can further show that it is also sufficient, i.e. the row-permuted LU factorisation $LU = PA$ exists for all A. This is one of the key points of the LU theorem on slide 3.

Nevertheless, I will usually omit the P-factor in the following, i.e. I will say "let $LU = A$ be the LU factorisation of A" rather than "let $LU = PA$ be the LU factorisation of A". The intention in doing so is that we can always replace A with its row-permuted copy PA and thereby avoid having to carry an extra factor P around.

Outlook

We are now ready to start looking into how we can exploit sparsity in the LU algorithm. I will begin this journey on the next slide with an example which illustrates some of the complications arising in sparse LU factorisations.

Sparse LU Factorisation

Example

$$
\begin{pmatrix}
1 & & 1 & \\
& 1 & 1 & \\
& & 1 & \\
1 & 1 & \square & 1
\end{pmatrix}
=
\begin{pmatrix}
1 & & & \\
\square & 1 & & \\
\square & \square & 1 & \\
1 & 1 & -2 & 1
\end{pmatrix}
\begin{pmatrix}
1 & \square & 1 & \square \\
& 1 & 1 & \square \\
& & 1 & \square \\
& & & 1
\end{pmatrix},
$$

$$
\begin{pmatrix}
1 & & 1 & \\
& -1 & 1 & \\
& & 1 & \\
1 & 1 & \square & 1
\end{pmatrix}
=
\begin{pmatrix}
1 & & & \\
\square & 1 & & \\
\square & \square & 1 & \\
1 & 1 & \square & 1
\end{pmatrix}
\begin{pmatrix}
1 & \square & 1 & \square \\
-1 & 1 & & \square \\
& & 1 & \square \\
& & & 1
\end{pmatrix}.
$$

Observations:

- $A[i,j] = 0$ does not imply $L[i,j] = 0$ or $U[i,j] = 0$.
- Some entries of L, U are zero regardless of the values we assign to the nonzero entries of A. Others may be zero or nonzero depending on the values in A.

Sparse LU Factorisation

The above example prompts us to introduce some terminology.

Terminology: Sparsity pattern

The set of all indices $(i, j) \in \{1, \ldots, n\}^2$ such that $A[i, j] \neq 0$ is called the *sparsity pattern* or *structure* of a matrix $A \in \mathbb{R}^{n \times n}$.

Sparsity patterns are conveniently described by drawing a matrix where each zero entry is left empty and each nonzero entry is marked with a bullet ($\bullet$), see the middle matrix below.

In all the examples considered in this lecture, the diagonal will always be nonzero. I use this fact to write the numbers 1 to n on the diagonal rather than bullets, see the right matrix below. Doing so makes the sparsity patterns easier to read and talk about.

Example

$$
\begin{pmatrix} 3 & 0 & 2 \\ 5 & 1 & 0 \\ 0 & 0 & 4 \end{pmatrix} = \left(\begin{array}{cc|c} \bullet & & \bullet \\ \hline \bullet & \bullet & \\ \hline & & \bullet \end{array} \right) = \left(\begin{array}{cc|c} 1 & & \bullet \\ \hline \bullet & 2 & \\ \hline & & 3 \end{array} \right).
$$

Sparse LU Factorisation

Terminology: Structural nonzero

Let B be a matrix derived from some other matrix A (e.g. $B = A^2$, $B = A^{-1}$, or B is one of the LU factors).

An entry $B[i,j]$ is called *structurally nonzero* if $B[i,j]$ could be nonzero given the sparsity pattern of A, and *structurally zero* otherwise.

I will occasionally use $B[i,j] \neq 0$ as a shorthand notation for saying that $B[i,j]$ is structurally nonzero (but not necessarily actually nonzero).

Example

Consider

$$A = \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix}, \qquad A^2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

▶ $A^2[2,1] = 0 \times 1 + -1 \times 0$ is structurally zero because the only way to make this number nonzero is to assign a nonzero value to $A[2,1] = 0$ and doing so would change the sparsity pattern of A.

▶ $A^2[1,2] = 1 \times 1 + 1 \times -1$ is structurally nonzero because we could make this number nonzero e.g. by changing $A[1,1] = 1$ to $A[1,1] = 2$ and doing so would not change the sparsity pattern of A.

Sparse LU Factorisation

Terminology: Derived sparsity pattern

Let B be a matrix derived from some other matrix A (e.g. $B = A^2$, $B = A^{-1}$, or B is one of the LU factors).

The set of all indices $(i, j) \in \{1, \ldots, n\}^2$ such that $B[i, j]$ is structurally nonzero is called the *sparsity pattern* or *structure* of the derived matrix B.

Example

Consider

$$A = \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix}, \qquad A^2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

The sparsity pattern of A^2 is

$$A^2 = \left(\begin{array}{c|c} 1 & \bullet \\ \hline & 2 \end{array} \right) \qquad \text{and not} \qquad A^2 = \left(\begin{array}{c|c} 1 & \square \\ \hline & 2 \end{array} \right)$$

because $A[1, 2]$ is structurally nonzero.

Sparse LU Factorisation

Resolving the ambiguity of "structure" for derived matrices

The above slides introduced two definitions for structure, namely one for matrices and another for derived matrices.

This strictly speaking means that "structure of a derived matrix B" can have two distinct meanings depending on whether we interpret B as a derived matrix or simply as a matrix.

Let us therefore establish the convention that the "derived" interpretation takes precedence whenever it is applicable.

Why study structure rather than values?

The fundamental idea behind the above notion of "structure" is that we only want to distinguish between whether an entry is guaranteed to be zero or could potentially be nonzero. Reasons for doing so include:

▶ Reasoning about structure is easier than reasoning about values.

▶ structure(B) provides a worst-case estimate for how much memory will be needed to store the derived matrix B.

▶ Cancellation (i.e. a structurally nonzero entry being zero) is unlikely.

Sparse LU Factorisation

The final term required to talk about sparsity is the following.

Terminology: Fill-in entry

Let B be a matrix derived from some other matrix A (e.g. $B = A^2$, $B = A^{-1}$, or B is one of the LU factors).

An index $(i, j) \in \text{structure}(B) \setminus \text{structure}(A)$ is called a *fill-in entry* of B.

Example

Consider

$$A = \begin{pmatrix} 1 & & \\ \bullet & 2 & \\ \boxed{} & \bullet & 3 \end{pmatrix}, \qquad A^2 = \begin{pmatrix} 1 & & \\ \bullet & 2 & \\ \bullet & \bullet & 3 \end{pmatrix}.$$

$A^2[3, 1]$ is a fill-in entry because $A[3, 1] = 0$ but $A^2[3, 1] \neq 0$.

This shows that matrix powers can create fill-in.

Example 2

$L[4, 3]$ in the LU factorisation from slide 14 is a fill-in entry.

This shows that the LU factorisation can create fill-in.

Sparse LU Factorisation

The importance of understanding fill-in

The fact that matrix powers and LU factorisations can create fill-in makes it hard to predict the runtime of such operations.

Example. `lu_benchmark()` shows that the runtime of computing an LU factorisation can differ by three orders of magnitude even if the input matrices have exactly the same size and number of nonzero entries.

"Order of magnitude" = power of 10.

This discrepancy arises because the factorisations of such matrices can have very different sparsity patterns, see `lu_structures()`.

The above example shows that it would be useful if we could estimate the amount of fill-in incurred by a sparse matrix operation before actually running it so we can decide ahead of time whether the computations will be feasible and sufficiently fast.

The following slides will show that we can do so by relating sparse matrices to graphs and then deducing the sparsity patterns of matrix powers, inverses and LU factorisations based on these graphs.

Sparse LU Factorisation

Let me begin by describing how we can translate matrices into graphs.
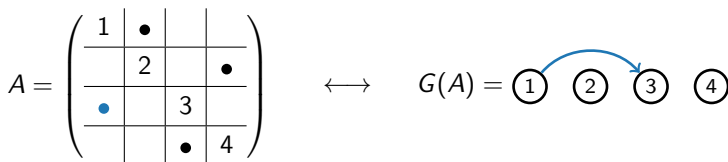
Def: Graph of a sparse matrix
The *graph* $G(A) = \big(V(A), E(A)\big)$ of a sparse matrix $A \in \mathbb{R}^{n \times n}$ is given by

$$V(A) = \{1, \ldots, n\}, \qquad E(A) = \big\{j \to i \mid A[i,j] \neq 0\big\}.$$

$V(A)$ denotes the set of vertices, $E(A)$ denotes the set of edges.
Note the transpose in $E(A)$: entry $A[i,j]$ corresponds to the edge $j \to i$.

Example

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \textcircled{1} \quad \textcircled{2} \quad \textcircled{3} \quad \textcircled{4}$$

Note that the picture on the right omits the diagonal edges $i \to i$ for
better readability.

Sparse LU Factorisation

Let me begin by describing how we can translate matrices into graphs.
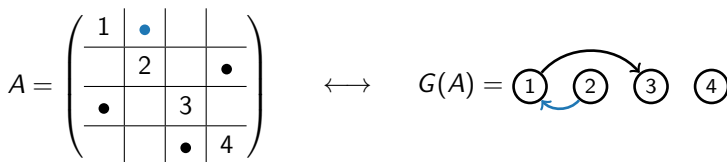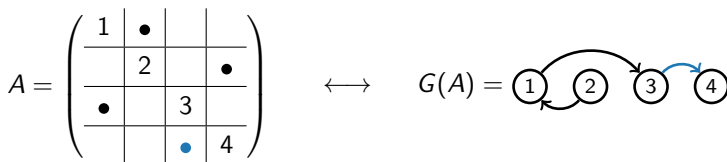
Def: Graph of a sparse matrix

The *graph* $G(A) = \big(V(A), E(A)\big)$ of a sparse matrix $A \in \mathbb{R}^{n \times n}$ is given by

$$V(A) = \{1, \ldots, n\}, \qquad E(A) = \big\{j \to i \mid A[i,j] \neq 0\big\}.$$

$V(A)$ denotes the set of vertices, $E(A)$ denotes the set of edges.
Note the transpose in $E(A)$: entry $A[i,j]$ corresponds to the edge $j \to i$.

Example

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \textcircled{1} \quad \textcircled{2} \quad \textcircled{3} \quad \textcircled{4}$$

Note that the picture on the right omits the diagonal edges $i \to i$ for better readability.

Sparse LU Factorisation

Let me begin by describing how we can translate matrices into graphs.
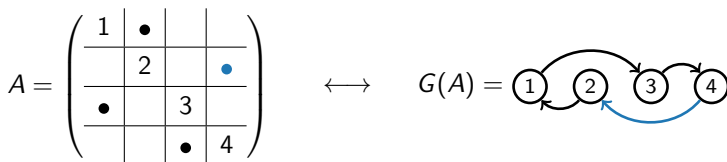
Def: Graph of a sparse matrix

The *graph* $G(A) = \big(V(A), E(A)\big)$ of a sparse matrix $A \in \mathbb{R}^{n \times n}$ is given by

$$V(A) = \{1, \ldots, n\}, \qquad E(A) = \big\{j \to i \mid A[i,j] \neq 0\big\}.$$

$V(A)$ denotes the set of vertices, $E(A)$ denotes the set of edges.

Note the transpose in $E(A)$: entry $A[i,j]$ corresponds to the edge $j \to i$.

Example

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \text{①} \; \text{②} \; \text{③} \; \text{④}$$

Note that the picture on the right omits the diagonal edges $i \to i$ for better readability.

Sparse LU Factorisation

Let me begin by describing how we can translate matrices into graphs.
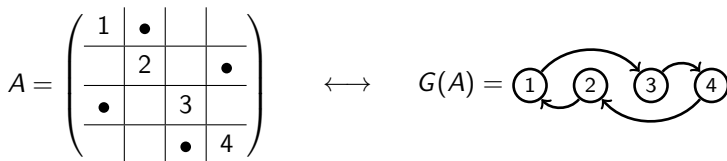
Def: Graph of a sparse matrix

The *graph* $G(A) = \big(V(A), E(A)\big)$ of a sparse matrix $A \in \mathbb{R}^{n \times n}$ is given by

$$V(A) = \{1, \ldots, n\}, \qquad E(A) = \big\{j \to i \mid A[i,j] \neq 0\big\}.$$

$V(A)$ denotes the set of vertices, $E(A)$ denotes the set of edges.
Note the transpose in $E(A)$: entry $A[i,j]$ corresponds to the edge $j \to i$.

Example

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \text{①②③④}$$

Note that the picture on the right omits the diagonal edges $i \to i$ for better readability.

Sparse LU Factorisation

Let me begin by describing how we can translate matrices into graphs.
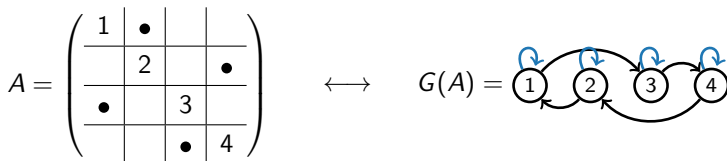
Def: Graph of a sparse matrix

The *graph* $G(A) = \big(V(A), E(A)\big)$ of a sparse matrix $A \in \mathbb{R}^{n \times n}$ is given by

$$V(A) = \{1, \ldots, n\}, \qquad E(A) = \big\{ j \to i \mid A[i,j] \neq 0 \big\}.$$

$V(A)$ denotes the set of vertices, $E(A)$ denotes the set of edges.
Note the transpose in $E(A)$: entry $A[i,j]$ corresponds to the edge $j \to i$.

Example

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \;①\;②\;③\;④$$

Note that the picture on the right omits the diagonal edges $i \to i$ for better readability.

Sparse LU Factorisation

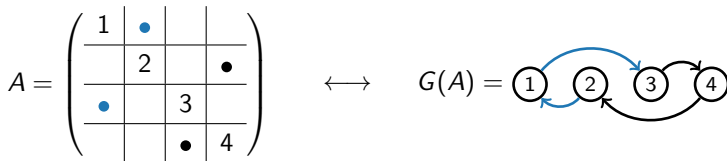Let me begin by describing how we can translate matrices into graphs.

Def: Graph of a sparse matrix

The *graph* $G(A) = \big(V(A), E(A)\big)$ of a sparse matrix $A \in \mathbb{R}^{n \times n}$ is given by

$$V(A) = \{1, \dots, n\}, \qquad E(A) = \big\{ j \to i \mid A[i,j] \neq 0 \big\}.$$

$V(A)$ denotes the set of vertices, $E(A)$ denotes the set of edges.
Note the transpose in $E(A)$: entry $A[i,j]$ corresponds to the edge $j \to i$.

Example

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \text{①②③④}$$

Note that the picture on the right omits the diagonal edges $i \to i$ for
better readability.

Sparse LU Factorisation

Let me begin by describing how we can translate matrices into graphs.

Def: Graph of a sparse matrix
The *graph* $G(A) = \big(V(A), E(A)\big)$ of a sparse matrix $A \in \mathbb{R}^{n \times n}$ is given by

$$V(A) = \{1, \ldots, n\}, \qquad E(A) = \big\{j \to i \mid A[i,j] \neq 0\big\}.$$

$V(A)$ denotes the set of vertices, $E(A)$ denotes the set of edges.
Note the transpose in $E(A)$: entry $A[i,j]$ corresponds to the edge $j \to i$.

Example

$$A = \begin{pmatrix} 1 & \bullet & & \\ \hline & 2 & & \bullet \\ \hline \bullet & & 3 & \\ \hline & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \text{①②③④}$$

Note that the picture on the right omits the diagonal edges $i \to i$ for better readability.

Sparse LU Factorisation

The following definition will be important to deduce the sparsity patterns of derived matrices from the graph of the original matrix.

Def: Paths

A *path* in a graph $G = (V, E)$ is an ordered sequence of vertices $k_0, \ldots, k_p \in V$ such that $k_{q-1} \to k_q \in E$ for all $q \in \{1, \ldots, p\}$.
The number of edges p is called the *length* of the path.

Example

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \begin{array}{cccc} \text{①} & \text{②} & \text{③} & \text{④} \end{array}$$

$2 \to 1 \to 3$ is a path of length 2.

Sparse LU Factorisation

We now have the tools to formulate our first fill-in-describing result.

Path theorem for matrix powers

$$A^p[i,j] \neq 0 \qquad \Longleftrightarrow \qquad \exists \text{ path } j \to i \text{ of length } p \text{ in } G(A).$$

Proof. We have

$$A^2[i,j] = \sum_k A[i,k]\,A[k,j].$$

Each term in this sum is nonzero iff $j \to k \to i$ is a path in $G(A)$.

Generalising this proof to arbitrary powers p, we observe that
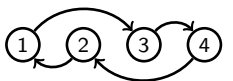
$$A^p[i,j] = \sum_{k_{p-1}} \ldots \sum_{k_1} A[i,k_{p-1}] \ldots A[k_a,k_{a-1}] \ldots A[k_1,j].$$

is nonzero iff $j \to k_1 \to \ldots \to k_{p-1} \to i$ is a path in $G(A)$.

Sparse LU Factorisation

Example

Recall from slide 21 the matrix

$$A = \begin{pmatrix} 1 & \bullet & & \\ \hline & 2 & & \bullet \\ \hline \bullet & & 3 & \\ \hline & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \begin{array}{c} \text{①} \; \text{②} \; \text{③} \; \text{④} \end{array} .$$
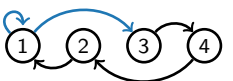
We observe:

- $A^2[1,1] \neq 0$ because $1 \to 1 \to 1$ is a path of length 2 in $G(A)$.

- $A^2[2,1] = 0$ because the only path connecting 1 to 2, namely $1 \to 3 \to 4 \to 2$, has length 3.

  Note that strictly speaking there are more paths from 1 to 2, namely we can extend the above path with diagonal edges. Such paths will not create additional fill-in, however, because adding diagonal edges only makes a path longer.

- $A^2[3,1] \neq 0$ because $1 \to 1 \to 3$ is a path of length 2 in $G(A)$.

- $A^2[4,1] \neq 0$ because $1 \to 3 \to 2$ is a path of length 2 in $G(A)$.

Sparse LU Factorisation

Example

Recall from slide 21 the matrix

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \text{①} \, \text{②} \, \text{③} \, \text{④} \,.$$

We observe:

- $A^2[1,1] \neq 0$ because $1 \to 1 \to 1$ is a path of length 2 in $G(A)$.

- $A^2[2,1] = 0$ because the only path connecting 1 to 2,
  namely $1 \to 3 \to 4 \to 2$, has length 3.

  Note that strictly speaking there are more paths from 1 to 2, namely we can extend the above path with diagonal edges. Such paths will not create additional fill-in, however, because adding diagonal edges only makes a path longer.

- $A^2[3,1] \neq 0$ because $1 \to 1 \to 3$ is a path of length 2 in $G(A)$.

- $A^2[4,1] \neq 0$ because $1 \to 3 \to 2$ is a path of length 2 in $G(A)$.

Sparse LU Factorisation

Example

Recall from slide 21 the matrix

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \text{①} \; \text{②} \; \text{③} \; \text{④} \; .$$

We observe:

▶ $A^2[1,1] \neq 0$ because $1 \to 1 \to 1$ is a path of length 2 in $G(A)$.

▶ $A^2[2,1] = 0$ because the only path connecting 1 to 2,
namely $1 \to 3 \to 4 \to 2$, has length 3.

   Note that strictly speaking there are more paths from 1 to 2, namely we can
   extend the above path with diagonal edges. Such paths will not create additional
   fill-in, however, because adding diagonal edges only makes a path longer.

▶ $A^2[3,1] \neq 0$ because $1 \to 1 \to 3$ is a path of length 2 in $G(A)$.

▶ $A^2[4,1] \neq 0$ because $1 \to 3 \to 2$ is a path of length 2 in $G(A)$.

Sparse LU Factorisation

Example

Recall from slide 21 the matrix

$$
A = \begin{pmatrix} 1 & \bullet & & \\ \hline & 2 & & \bullet \\ \hline \bullet & & 3 & \\ \hline & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \underset{1}{\textcircled{1}} \quad \underset{2}{\textcircled{2}} \quad \underset{3}{\textcircled{3}} \quad \underset{4}{\textcircled{4}} \;.
$$

We observe:

▶ $A^2[1,1] \neq 0$ because $1 \to 1 \to 1$ is a path of length 2 in $G(A)$.

▶ $A^2[2,1] = 0$ because the only path connecting 1 to 2, namely $1 \to 3 \to 4 \to 2$, has length 3.

  Note that strictly speaking there are more paths from 1 to 2, namely we can extend the above path with diagonal edges. Such paths will not create additional fill-in, however, because adding diagonal edges only makes a path longer.

▶ $A^2[3,1] \neq 0$ because $1 \to 1 \to 3$ is a path of length 2 in $G(A)$.

▶ $A^2[4,1] \neq 0$ because $1 \to 3 \to 2$ is a path of length 2 in $G(A)$.

Sparse LU Factorisation

Example

Recall from slide 21 the matrix

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ \square & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \text{①} \ \text{②} \ \text{③} \ \text{④} \ .$$

We observe:

- $A^2[1,1] \neq 0$ because $1 \to 1 \to 1$ is a path of length 2 in $G(A)$.

- $A^2[2,1] = 0$ because the only path connecting 1 to 2,
  namely $1 \to 3 \to 4 \to 2$, has length 3.

  Note that strictly speaking there are more paths from 1 to 2, namely we can
  extend the above path with diagonal edges. Such paths will not create additional
  fill-in, however, because adding diagonal edges only makes a path longer.

- $A^2[3,1] \neq 0$ because $1 \to 1 \to 3$ is a path of length 2 in $G(A)$.

- $A^2[4,1] \neq 0$ because $1 \to 3 \to 2$ is a path of length 2 in $G(A)$.

Sparse LU Factorisation

Example (continued)

Continuing the above analysis for all the remaining entries, we conclude

$$
A^2 = \begin{pmatrix}
1 & \bullet & & \textcolor{red}{\bullet} \\
& 2 & \textcolor{red}{\bullet} & \bullet \\
\bullet & \textcolor{red}{\bullet} & 3 & \\
\textcolor{red}{\bullet} & & \bullet & 4
\end{pmatrix} \quad .
$$

The $\bullet$ indicate nonzero entries which were nonzero already in A.
The $\textcolor{red}{\bullet}$ indicate fill-in.

Sparse LU Factorisation

Path theorem for inverses

$$A^{-1}[i,j] \neq 0 \quad \Longleftrightarrow \quad \exists \text{ path } j \to i \text{ in } G(A).$$

Proof (not examinable).
Let $p(x) = \sum_{k=0}^{n-1} c_k x^k$ be the unique polynomial which interpolates $\frac{1}{x}$ in all the n eigenvalues of A. We then have

$$A^{-1} = p(A) = \sum_{k=0}^{n-1} c_k A^k,$$

which shows that $A^{-1}[i,j] \neq 0$ if and only if there is a path from j to i of any arbitrary length $k \in \{0, \dots, n-1\}$.

(Do not worry in case the claim $A^{-1} = p(A)$ is not clear to you. We will discuss matrix polynomials and their relation to inverses in Lecture 5.)

Sparse LU Factorisation

Example

Recall from slide 21 the matrix

$$A = \begin{pmatrix} 1 & \bullet & & \\ \hline & 2 & & \bullet \\ \hline \bullet & & 3 & \\ \hline & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \text{①②③④} .$$

We observe that any vertex i is reachable from any other vertex j in $G(A)$; hence we have

$$A^{-1} = \begin{pmatrix} 1 & \bullet & \bullet & \bullet \\ \hline \bullet & 2 & \bullet & \bullet \\ \hline \bullet & \bullet & 3 & \bullet \\ \hline \bullet & \bullet & \bullet & 4 \end{pmatrix} .$$

The $\bullet$ indicate nonzero entries which were nonzero already in A.
The $\bullet$ indicate fill-in.

Sparse LU Factorisation

Corollaries of path theorem for inverses

The path theorem for inverses provides simple proofs for a number of useful results which would be much more tedious to show otherwise.

▶ The inverse of a tridiagonal matrix

$$
A = \begin{pmatrix}
1 & \bullet & & \\
\bullet & 2 & \bullet & \\
& \bullet & 3 & \bullet \\
& & \bullet & 4
\end{pmatrix}
\qquad \longleftrightarrow \qquad
G(A) = \text{①②③④}
$$

is dense because any pair of vertices i, j is connected.

▶ The inverse of an upper-triangular matrix

$$
A = \begin{pmatrix}
1 & \bullet & \bullet & \bullet \\
& 2 & \bullet & \bullet \\
& & 3 & \bullet \\
& & & 4
\end{pmatrix}
\qquad \longleftrightarrow \qquad
G(A) = \text{①②③④}
$$

is upper triangular because all edges in $G(A)$ go only from right to left and hence paths j to i exist only if $i \leq j$.

Sparse LU Factorisation

We now know how to predict fill-in in matrix powers and inverses.
Let us therefore move on to predicting fill-in in the LU factorisation next.
Doing so requires a new notion of path defined as follows.

Def: Fill path
Path $j \to k_1 \to \ldots \to k_p \to i$ in $G(A)$ such that $k_1, \ldots, k_p < \min\{i, j\}$.

We then have the following result.

Fill path theorem
Let $LU = A$ be the LU factorisation of A. Then,

$$(L + U)[i, j] \neq 0 \quad \Longleftrightarrow \quad \exists \text{ fill path } j \to i \text{ in } G(A).$$

Proof (not examinable). A proof of this result is provided on slide 33, but I will not discuss it in class. I recommend you ignore this proof unless you are truly interested.

Sparse LU Factorisation

Describing the sparsity pattern of an LU factorisation

The above result describes the sparsity pattern of $L + U$ rather than the two sparsity patterns of L and U. Doing so is convenient because it means we have to think about only one rather than two matrices, and it does not discard any information because we have

$$(L + U)[i, j] \neq 0 \quad \Longleftrightarrow \quad \begin{cases} L[i, j] \neq 0 & \text{if } i \geq j, \\ U[i, j] \neq 0 & \text{if } i \leq j. \end{cases}$$

Note that we do not have to worry about cancellation in the diagonal entries

$$(L + U)[i, i] = L[i, i] + U[i, i]$$

because cancellation does not change whether an entry is structurally nonzero.

Sparse LU Factorisation

Example

Recall from slide 21 the matrix

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \text{(1) (2) (3) (4)} \, .$$

We observe:

- ▶ $U[1,1] \neq 0$ because $1 \rightarrow 1$ is a fill path in $G(A)$.
- ▶ $L[2,1] = 0$ because $1 \rightarrow 3 \rightarrow 4 \rightarrow 2$ is not a fill path $(3, 4 > 1)$.
- ▶ $L[3,1] \neq 0$ because $1 \rightarrow 3$ is a fill path in $G(A)$.
- ▶ $L[4,1] = 0$ because $1 \rightarrow 3 \rightarrow 4$ is not a fill path $(3 > 1)$.
- ▶ $U[1,2], U[2,2] \neq 0$ because $2 \rightarrow 1$ and $2 \rightarrow 2$ are fill paths in $G(A)$.
- ▶ $L[3,2] \neq 0$ because $2 \rightarrow 1 \rightarrow 3$ is a fill path in $G(A)$.
- ▶ The only other fill-in entry is $U[3,4]$ because $4 \rightarrow 2 \rightarrow 1 \rightarrow 3$ is a fill path in $G(A)$.

Sparse LU Factorisation

Example

Recall from slide 21 the matrix

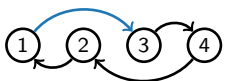$$A = \begin{pmatrix} 1 & \bullet & & \\ \hline & 2 & & \bullet \\ \hline \bullet & & 3 & \\ \hline & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \;$$.

We observe:

- $U[1,1] \neq 0$ because $1 \to 1$ is a fill path in $G(A)$.
- $L[2,1] = 0$ because $1 \to 3 \to 4 \to 2$ is not a fill path ($3, 4 > 1$).
- $L[3,1] \neq 0$ because $1 \to 3$ is a fill path in $G(A)$.
- $L[4,1] = 0$ because $1 \to 3 \to 4$ is not a fill path ($3 > 1$).
- $U[1,2], U[2,2] \neq 0$ because $2 \to 1$ and $2 \to 2$ are fill paths in $G(A)$.
- $L[3,2] \neq 0$ because $2 \to 1 \to 3$ is a fill path in $G(A)$.
- The only other fill-in entry is $U[3,4]$ because $4 \to 2 \to 1 \to 3$ is a fill path in $G(A)$.

Sparse LU Factorisation

Example

Recall from slide 21 the matrix

$$A = \begin{pmatrix} 1 & \bullet & & \\ \blacksquare & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \text{①} \; \text{②} \; \text{③} \; \text{④} .$$

We observe:

- $U[1,1] \neq 0$ because $1 \to 1$ is a fill path in $G(A)$.
- $L[2,1] = 0$ because $1 \to 3 \to 4 \to 2$ is not a fill path $(3, 4 > 1)$.
- $L[3,1] \neq 0$ because $1 \to 3$ is a fill path in $G(A)$.
- $L[4,1] = 0$ because $1 \to 3 \to 4$ is not a fill path $(3 > 1)$.
- $U[1,2], U[2,2] \neq 0$ because $2 \to 1$ and $2 \to 2$ are fill paths in $G(A)$.
- $L[3,2] \neq 0$ because $2 \to 1 \to 3$ is a fill path in $G(A)$.
- The only other fill-in entry is $U[3,4]$ because $4 \to 2 \to 1 \to 3$ is a fill path in $G(A)$.

Sparse LU Factorisation

Example

Recall from slide 21 the matrix

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \text{①} \; \text{②} \; \text{③} \; \text{④} \; .$$

We observe:

- $U[1,1] \neq 0$ because $1 \to 1$ is a fill path in $G(A)$.
- $L[2,1] = 0$ because $1 \to 3 \to 4 \to 2$ is not a fill path $(3, 4 > 1)$.
- $L[3,1] \neq 0$ because $1 \to 3$ is a fill path in $G(A)$.
- $L[4,1] = 0$ because $1 \to 3 \to 4$ is not a fill path $(3 > 1)$.
- $U[1,2], U[2,2] \neq 0$ because $2 \to 1$ and $2 \to 2$ are fill paths in $G(A)$.
- $L[3,2] \neq 0$ because $2 \to 1 \to 3$ is a fill path in $G(A)$.
- The only other fill-in entry is $U[3,4]$ because $4 \to 2 \to 1 \to 3$ is a fill path in $G(A)$.

Sparse LU Factorisation

Example

Recall from slide 21 the matrix

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \text{①②③④} .$$

We observe:

▶ $U[1,1] \neq 0$ because $1 \to 1$ is a fill path in $G(A)$.

▶ $L[2,1] = 0$ because $1 \to 3 \to 4 \to 2$ is not a fill path $(3, 4 > 1)$.

▶ $L[3,1] \neq 0$ because $1 \to 3$ is a fill path in $G(A)$.

▶ $L[4,1] = 0$ because $1 \to 3 \to 4$ is not a fill path $(3 > 1)$.

▶ $U[1,2], U[2,2] \neq 0$ because $2 \to 1$ and $2 \to 2$ are fill paths in $G(A)$.

▶ $L[3,2] \neq 0$ because $2 \to 1 \to 3$ is a fill path in $G(A)$.

▶ The only other fill-in entry is $U[3,4]$ because $4 \to 2 \to 1 \to 3$ is a fill path in $G(A)$.

Sparse LU Factorisation

Example

Recall from slide 21 the matrix

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \text{①} \quad \text{②} \quad \text{③} \quad \text{④} \; .$$

We observe:

- $U[1,1] \neq 0$ because $1 \rightarrow 1$ is a fill path in $G(A)$.
- $L[2,1] = 0$ because $1 \rightarrow 3 \rightarrow 4 \rightarrow 2$ is not a fill path ($3, 4 > 1$).
- $L[3,1] \neq 0$ because $1 \rightarrow 3$ is a fill path in $G(A)$.
- $L[4,1] = 0$ because $1 \rightarrow 3 \rightarrow 4$ is not a fill path ($3 > 1$).
- $U[1,2], U[2,2] \neq 0$ because $2 \rightarrow 1$ and $2 \rightarrow 2$ are fill paths in $G(A)$.
- $L[3,2] \neq 0$ because $2 \rightarrow 1 \rightarrow 3$ is a fill path in $G(A)$.
- The only other fill-in entry is $U[3,4]$ because $4 \rightarrow 2 \rightarrow 1 \rightarrow 3$ is a fill path in $G(A)$.

Sparse LU Factorisation

Example

Recall from slide 21 the matrix

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \text{①} \; \text{②} \; \text{③} \; \text{④} \; .$$

We observe:

- $U[1,1] \neq 0$ because $1 \to 1$ is a fill path in $G(A)$.
- $L[2,1] = 0$ because $1 \to 3 \to 4 \to 2$ is not a fill path $(3, 4 > 1)$.
- $L[3,1] \neq 0$ because $1 \to 3$ is a fill path in $G(A)$.
- $L[4,1] = 0$ because $1 \to 3 \to 4$ is not a fill path $(3 > 1)$.
- $U[1,2], U[2,2] \neq 0$ because $2 \to 1$ and $2 \to 2$ are fill paths in $G(A)$.
- $L[3,2] \neq 0$ because $2 \to 1 \to 3$ is a fill path in $G(A)$.
- The only other fill-in entry is $U[3,4]$ because $4 \to 2 \to 1 \to 3$ is a fill path in $G(A)$.

Sparse LU Factorisation

Example

Recall from slide 21 the matrix

$$A = \begin{pmatrix} 1 & \bullet & & \\ & 2 & & \bullet \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = \textcircled{1} \quad \textcircled{2} \quad \textcircled{3} \quad \textcircled{4} \ .$$

We observe:

- $U[1,1] \neq 0$ because $1 \to 1$ is a fill path in $G(A)$.
- $L[2,1] = 0$ because $1 \to 3 \to 4 \to 2$ is not a fill path $(3, 4 > 1)$.
- $L[3,1] \neq 0$ because $1 \to 3$ is a fill path in $G(A)$.
- $L[4,1] = 0$ because $1 \to 3 \to 4$ is not a fill path $(3 > 1)$.
- $U[1,2], U[2,2] \neq 0$ because $2 \to 1$ and $2 \to 2$ are fill paths in $G(A)$.
- $L[3,2] \neq 0$ because $2 \to 1 \to 3$ is a fill path in $G(A)$.
- The only other fill-in entry is $U[3,4]$ because $4 \to 2 \to 1 \to 3$ is a fill path in $G(A)$.

Sparse LU Factorisation

Example (continued)

We thus conclude that the sparsity pattern of $L + U$ is given by

$$
L + U = \begin{pmatrix}
1 & \bullet & & \\
& 2 & & \bullet \\
\bullet & \bullet & 3 & \bullet \\
& & \bullet & 4
\end{pmatrix}.
$$

The $\bullet$ indicate nonzero entries which were nonzero already in A.
The $\bullet$ indicate fill-in.

Sparse LU Factorisation

The proof of the fill path theorem is based on the following result.

Lemma (not examinable, ignore unless you are interested)

Let $LU = A$ be the LU factorisation of a matrix $A \in \mathbb{R}^{n \times n}$, let $i, j \in \{1, \ldots, n\}$ and set $\ell = \{1, \ldots, \min\{i,j\} - 1\}$. We then have,

$$U[i,j] = A[i,j] - A[i,\ell]\, A[\ell,\ell]^{-1}\, A[\ell,j] \qquad \text{for } i \leq j,$$
$$L[i,j]\, U[j,j] = A[i,j] - A[i,\ell]\, A[\ell,\ell]^{-1}\, A[\ell,j] \qquad \text{for } i \geq j.$$

Proof. Consider the block LU factorisation

$$\begin{pmatrix} A[\ell,\ell] & A[\ell,\bar{r}] \\ A[\bar{r},\ell] & A[\bar{r},\bar{r}] \end{pmatrix} = \begin{pmatrix} I & \\ A[\bar{r},\ell]\, A[\ell,\ell]^{-1} & I \end{pmatrix} \begin{pmatrix} A[\ell,\ell] & A[\ell,\bar{r}] \\ & A[\bar{r},\bar{r}] - A[\bar{r},\ell]\, A[\ell,\ell]^{-1}\, A[\ell,\bar{r}] \end{pmatrix} \qquad (1)$$

where $\bar{r} = \{\min\{i,j\}, \ldots, n\}$, The full factorisation is then given by

$$\begin{pmatrix} A[\ell,\ell] & A[\ell,\bar{r}] \\ A[\bar{r},\ell] & A[\bar{r},\bar{r}] \end{pmatrix} = \begin{pmatrix} L_1 & \\ A[\bar{r},\ell]\, A[\ell,\ell]^{-1} L_1 & L_2 \end{pmatrix} \begin{pmatrix} U_1 & L_1^{-1} A[\ell,\bar{r}] \\ & U_2 \end{pmatrix}$$

where $\quad L_1 U_1 = A[\ell,\ell] \quad$ and $\quad L_2 U_2 = A[\bar{r},\bar{r}] - A[\bar{r},\ell]\, A[\ell,\ell]^{-1}\, A[\ell,\bar{r}]$ are the LU factorisations of the top-left and bottom-right blocks of the U-factor in (1). The claim follows by noting that $L[i,j] = L_2[i,j]$ and $U[i,j] = U_2[i,j]$ have the given form.

Sparse LU Factorisation

Proof of fill path theorem (not examinable, ignore unless you are interested).

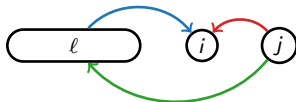According to the lemma on the previous slide, we have

$$U[i,j] = A[i,j] - A[i,\ell]\,A[\ell,\ell]^{-1}\,A[\ell,j] \qquad \text{for } i \leq j,$$

$$L[i,j]\,U[j,j] = A[i,j] - A[i,\ell]\,A[\ell,\ell]^{-1}\,A[\ell,j] \qquad \text{for } i \geq j.$$

The first term makes $U[i,j]$ / $L[i,j]\,U[j,j]$ nonzero if there is a fill path $j \to i$ of length 1, and the second term makes $U[i,j]$ / $L[i,j]\,U[j,j]$ nonzero if there is a fill path $j \to i$ of length > 1.

It then remains to observe that

$$L[i,j]\,U[j,j] \neq 0 \quad \Longleftrightarrow \quad L[i,j]$$

since $U[j,j]$ is a pivot entry and hence we necessarily have $U[j,j] \neq 0$.

Sparse LU Factorisation

Example: LU factorisation of arrow matrices

Our motivation for looking into the above (fill) path theorems was so we would be able to predict the runtime and memory requirements of a sparse matrix operation before actually running it.

Let us therefore next revisit the arrow matrix example from slide 20 to verify that these theorems indeed fulfill their purpose. Specifically, let us try to understand the LU sparsity patterns shown in `lu_structures()` using the fill-path theorem, and then let us translate these sparsity patterns into runtime estimates using the formula from slide 7.

Sparse LU Factorisation

Example: LU factorisation of arrow matrices (continued)

Application of the fill path theorem requires us to first determine the matrix graphs. In the $n = 4$ case, these graphs are given by

$$A_4^{\nwarrow} = \begin{pmatrix} 1 & \bullet & \bullet & \bullet \\ \bullet & 2 & & \\ \bullet & & 3 & \\ \bullet & & & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A_4^{\nwarrow}) = \text{①②③④},$$

$$A_4^{\searrow} = \begin{pmatrix} 1 & & & \bullet \\ & 2 & & \bullet \\ & & 3 & \bullet \\ \bullet & \bullet & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A_4^{\searrow}) = \text{①②③④}.$$
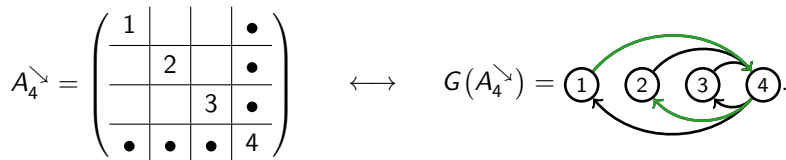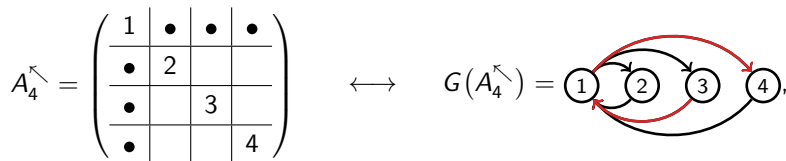
We observe:

▶ All paths $j \to 1 \to i$ in $G(A_n^{\nwarrow})$ are fill paths.
▶ No path $j \to 4 \to i$ in $G(A_n^{\searrow})$ is a fill path.

Sparse LU Factorisation

Example: LU factorisation of arrow matrices (continued)

We therefore conclude:

► The LU factorisation of $A_n^{\nwarrow}$ is completely filled in, i.e.

$$A_4^{\nwarrow}: \quad (L + U) = \begin{pmatrix} 1 & \bullet & \bullet & \bullet \\ \bullet & 2 & \bullet & \bullet \\ \bullet & \bullet & 3 & \bullet \\ \bullet & \bullet & \bullet & 4 \end{pmatrix}.$$

► The LU factorisation of $A_n^{\searrow}$ contains no fill-in at all, i.e.

$$A_4^{\searrow}: \quad (L + U) = \begin{pmatrix} 1 & & & \bullet \\ & 2 & & \bullet \\ & & 3 & \bullet \\ \bullet & \bullet & \bullet & 4 \end{pmatrix}.$$

This agrees with our observations in `lu_structure()`.

Sparse LU Factorisation

Example: LU factorisation of arrow matrices (continued)

Inserting these patterns into the formula from slide 7, we conclude:

► The runtime of factoring $A_n^{\nwarrow}$ is

$$O\left(\sum_{k=1}^{n} \text{nnz}(L[:,k])\, \text{nnz}(U[k,:])\right) = O\left(\sum_{k=1}^{n} k \times k\right) = O(n^3).$$

► The runtime of factoring $A_n^{\searrow}$ is

$$O\left(\sum_{k=1}^{n} \text{nnz}(L[:,k])\, \text{nnz}(U[k,:])\right) = O\left(\sum_{k=1}^{n-1} 2 \times 2 + 1\right) = O(n).$$

LU factorisation of $A_{1000}^{\nwarrow}$ should hence be about $1000^2 = 1$ million times slower than LU factorisation of $A_{1000}^{\searrow}$.

This agrees with our observations in `lu_benchmark()` in the sense that LU factorisation of $A_{1000}^{\nwarrow}$ is indeed much slower, but the empirically observed slowdown is "only" about 1000x rather than 1 million x.

The difference of a factor 1000 is likely due to hardware details.

Sparse LU Factorisation

Fill-reducing permutations

The above example demonstrates that the arrangement of the nonzero entries of a sparse matrix can have a drastic impact on the runtime of its LU factorisation.

It may further seem to indicate that $A_n^{\nwarrow} x = b$ is intrinsically harder to solve than $A_n^{\searrow} x = b$, but I will show next that this is not true.

Claim

The LU factorisation can be used to solve $A_n^{\nwarrow} x = b$ in $O(n)$ operations.

Proof. See next slide.

Sparse LU Factorisation

Proof of claim from slide 39.

A linear system $Ax = b$ ultimately represents a collection of linear equations

$$A[1,1]\,x[1]\ +\ \ldots\ +\ A[1,n]\,x[n]\ =\ b[1],$$
$$\vdots \qquad \ddots \qquad \vdots \quad = \quad \vdots$$
$$A[n,1]\,x[1]\ +\ \ldots\ +\ A[n,n]\,x[n]\ =\ b[n],$$

in terms of some unknowns $x[i]$, and we are free to enumerate these equations and unknowns in whichever way is suitable for us.

In the language of linear algebra, this means that we are free to pick two permutation matrices P, Q and solve

$$(PAQ^T)\,(Qx) = Pb \qquad \text{instead of} \qquad Ax = b.$$

Fact: Permutation matrices satisfy $Q^{-1} = Q^T$.

The next slide will show that we can use this freedom to reduce

$$A_n^{\nwarrow}\, x = b \qquad \text{to} \qquad A_n^{\searrow}\, (Qx) = (Pb)$$

if we choose the correct $P = Q$.

Sparse LU Factorisation

Proof of claim from slide 39 (continued).

Let $P \in \mathbb{R}^{n \times n}$ be the permutation matrix which swaps the first and last entry, i.e. the permutation associated with the permutation vector

$$p = [\, n, \; 2, \ldots, n-1, \; 1\,].$$

We then have

$$PA_4^{\nwarrow} \;=\; P \begin{pmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & & \\ \bullet & & \bullet & \\ \bullet & & & \bullet \end{pmatrix} \;=\; \begin{pmatrix} \bullet & & & \bullet \\ \bullet & \bullet & & \\ \bullet & & \bullet & \\ \bullet & \bullet & \bullet & \bullet \end{pmatrix},$$

$$\left(PA_4^{\nwarrow}\right)P^T \;=\; \begin{pmatrix} \bullet & & & \bullet \\ \bullet & \bullet & & \\ \bullet & & \bullet & \\ \bullet & \bullet & \bullet & \bullet \end{pmatrix} P^T \;=\; \begin{pmatrix} \bullet & & & \bullet \\ & \bullet & & \bullet \\ & & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{pmatrix}.$$

Fact: $A \mapsto AP^T$ corresponds to swapping columns.

Sparse LU Factorisation

Proof of claim from slide 39 (conclusion).

The sparsity pattern of $PA_n^{\nwarrow}P^T$ is thus the same as that of $A_n^{\searrow}$ and hence the LU factorisation of $PA_n^{\nwarrow}P^T$ does not create any fill-in.

This fact can be used to solve

$$A_n^{\nwarrow} x = b \qquad \Longleftrightarrow \qquad (PA_n^{\nwarrow}P^T)(Px) = Pb$$

as follows.

- Compute the factorisation $LU = PA_n^{\nwarrow}P^T$.
- Solve $Ly = Pb$ and $U(Px) = y$.
- Compute $x = P^T(Px)$.

Clearly, each of these steps can be done in $O(n)$ operations due to the sparsity of the LU factors established above.

Sparse LU Factorisation

The above example shows that matrix permutations of the form $A \mapsto PAP^T$ are very useful for accelerating LU factorisations.

The following result describes the effect of these permutations on the matrix graph.

Vertex renumbering theorem

Let P be the matrix associated with the permutation $\pi(i)$. The map $A \mapsto PAP^T$ then corresponds to applying $\pi^{-1}(i)$ to the vertices in $G(A)$.

Proof (not examinable). In mathematical terms, the above theorem says that

$$V(PAP^T) = \{\pi^{-1}(v) \mid v \in V(A)\},$$
$$E(PAP^T) = \{\pi^{-1}(j) \to \pi^{-1}(i) \mid (j \to i) \in E(A)\}.$$

Sparse LU Factorisation

Proof (not examinable, continued).

To verify the claim regarding $V(PAP^T)$, I observe that

$$\{\pi^{-1}(v) \mid v \in V(A)\} = \{\pi^{-1}(v) \mid v \in \{1, \ldots, n\}\}$$
$$= \{1, \ldots, n\} = V(PAP^T),$$

where for the first and third equality I used the definition of the vertex set associated with a sparse matrix, and for the second equality I used that $\pi^{-1}(i)$ is bijective.

The claim regarding $E(PAP^T)$ follows from

$$\big(\pi^{-1}(j) \rightarrow \pi^{-1}(i)\big) \in E(PAP^T)$$
$$\iff (PAP^T)[\pi^{-1}(i), \pi^{-1}(j)] \neq 0$$
$$\iff A[\pi(\pi^{-1}(i)), \pi(\pi^{-1}(j))] \neq 0$$
$$\iff A[i, j] \neq 0$$
$$\iff (j \rightarrow i) \in E(A).$$

Sparse LU Factorisation

Example

Let P be the matrix associated with permutation $p = [4, 2, 1, 3]$, and let

$$A = \begin{pmatrix} 1 & & & \bullet \\ \bullet & 2 & & \\ \bullet & & 3 & \\ & & & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(A) = $$

We then have (see `matrix()`)

$$PAP^T = \begin{pmatrix} 1 & & & \\ & 2 & \bullet & \\ \bullet & & 3 & \\ & & \bullet & 4 \end{pmatrix} \qquad \longleftrightarrow \qquad G(PAP^T) = $$

Sparse LU Factorisation

Example (continued)

We observe:

- ▶ The graphs of A and PAP^T are the same except that the vertices are assigned different numbers.
- ▶ If we go through the vertices in the order of $G(A)$ and write down the corresponding numbers in $G(PAP^T)$, we obtain the vector

$$q = [3, 2, 4, 1].$$

  This vector represents the inverse permutation to

$$p = [4, 2, 1, 3],$$

  i.e. we have $p[q[i]] = i$ (see `permutation()`).

Sparse LU Factorisation

The vertex numbering theorem also works in reverse.

Reverse vertex numbering theorem
Let A, B be two matrices such that $G(A)$ and $G(B)$ are the same up to vertex numbering. Then, there exists a permutation matrix P such that

$$\text{structure}(B) = \text{structure}(PAP^T).$$

Proof (not examinable).
Two graphs being the same up to vertex numbering means that there exists a bijection $\pi(i)$ between $V(A)$ and $V(B)$ such that

$$(j \to i) \in E(A) \quad \Longleftrightarrow \quad \big(\pi(j) \to \pi(i)\big) \in E(B).$$

This bijection defines a permutation matrix P, and according to the forward vertex numbering theorem this P satisfies $G(B) = G(PAP^T)$. The claim then follows by observing that

$$G(B) = G(PAP^T) \quad \Longleftrightarrow \quad \text{structure}(B) = \text{structure}(PAP^T).$$

Sparse LU Factorisation

Matrix permutations vs. vertex renumberings

The practical consequence of the (reverse) vertex numbering theorems is that we can think of $A \mapsto PAP^T$ *exclusively* in terms of assigning new numbers to the vertices in $G(A)$.

This massively simplifies thinking about such permutations.

Example

Instead of explicitly working out the shape of $PA_n^{\nwarrow} P^T$ on slide 41, we could simply have observed that

$$p = [\, n, \ 2, \ldots, n-1, \ 1 \,]$$

transforms

into.

Sparse LU Factorisation

Avoiding fill-in is not always possible

The above arrow matrix example was extreme in that we managed to
replace the fully filled-in LU factorisation of $A_n^{\nwarrow}$ with the fill-in-free LU
factorisation of $PA_n^{\nwarrow}P^T$. This raises the question whether eliminating all
fill-in is always possible. Unfortunately, the answer is no.

Claim

Consider the matrix

$$A = \begin{pmatrix} 1 & \bullet & \bullet & \\ \bullet & 2 & & \bullet \\ \bullet & & 3 & \bullet \\ & \bullet & \bullet & 4 \end{pmatrix}.$$

LU factorisation of PAQ^T leads to fill-in in exactly two entries for any
two permutation matrices P and Q such that the unpivoted LU
factorisation exists.

Sparse LU Factorisation

Proof (not examinable).

We can verify the claim by going through all $(4!)^2 = 576$ different choices of P and Q and verify that each such choice leads to either a zero pivot entry or exactly two fill-in entries.

You can do so either with the help of a computer (see `check_fillin()`), or you can use the symmetries of A to reduce the number of nontrivial permutations to a manageable number.

Avoiding all fill-in is not always possible (conclusion)

We hence conclude that instead of looking for a permutation PAQ^T which avoids all fill-in, we in general have to look for permutations which keep the amount of fill-in as small as possible.

This in turn raises the question whether we can determine such fill-minimising permutations in a reasonable amount of time.

Unfortunately, the answer is once again no.

Sparse LU Factorisation

Theorem

Let A be a symmetric matrix. The problem of finding the permutation matrix P which minimises the fill-in in the LU factorisation of PAP^T is NP-complete.

Proof. See M. Yannakakis (1981), *Computing the minimum fill-in is NP-complete.*

Determining fill-minimising permutations is difficult

NP-completeness is a concept from theoretical computer science which in this context means that we generally cannot do better than computing the fill-in for all possible permutations P and then pick the one P which leads to the least fill-in.

There are $n!$ different permutations on the set $\{1, \ldots, n\}$, and the factorial function grows *very* rapidly with n. Finding the best possible permutation is hence not possible except for very small n.

Sparse LU Factorisation

Example

Consider a 20×20 matrix A and assume that computing the amount of fill-in in the LU factorisation of PAP^T for a particular permutation P takes 1 nanoseconds. Going through all 20! permutations P would then take

$$20! \times 1\,\text{nanosecond} \approx 77\,\text{years!}$$

Determining fill-minimising permutations is difficult (conclusion)

NP-completeness of finding the fill-minimising permutation P thus means that we have to give up on finding the best possible permutation P and instead rely on heuristics which deliver decent but perhaps not best possible performance.

We will see an example of such a fill-reducing heuristic later in this lecture.

Sparse LU Factorisation

Summary of fill-reducing permutations

The discussion between slides 39 and now can be summarised as follows.

- Instead of solving $Ax = b$, we can solve $(PAQ^T)(Qx) = Pb$.

- Doing so can lead to computational benefits if the LU factorisation of PAQ^T is more sparse than that of A.

- Finding the best possible permutations P and Q is difficult, but we can use heuristics to find P and Q which are often good enough.

Sparse LU Factorisation

Solving $-\Delta_n^{(d)} u_n = f$ via LU factorisation

In the remainder of this lecture, I will demonstrate how we can use the abstract theory of sparse LU factorisations developed above to determine the runtime and memory requirements of solving the discretised Poisson equation $-\Delta_n^{(d)} u_n = f$ via LU factorisation.

To put these requirements into perspective, let me introduce the variable $N = n^d$ to denote the number of rows and columns in $\Delta_n^{(d)}$, and let me point out the following.

Thm: Optimal runtime and memory for solving $-\Delta_n^{(d)} u_n = f$

No algorithm can solve $-\Delta_n^{(d)} u_n = f$ using less than $O(N)$ runtime and memory.

Proof. Any algorithm for solving $-\Delta_n^{(d)} u_n = f$ must at the very least read all of $f \in \mathbb{R}^N$ and write into all of $u \in \mathbb{R}^N$.

These operations require at least $O(N)$ runtime and memory.

Sparse LU Factorisation

It turns out that LU factorisation has optimal runtime and memory in the one-dimensional case.

Thm: LU factorisation for solving $-\Delta_n^{(1)} u_n = f$

Solving $-\Delta_n^{(1)} u_n = f$ via LU factorisation requires $O(n) = O(N)$ runtime and memory.

Proof. Recall from the previous lecture that $\Delta_n^{(1)}$ is of the form

$$\Delta_n^{(1)} = (n+1)^2 \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & 1 & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & -2 \end{pmatrix}.$$

The graph associated with this matrix is

$$G\left(\Delta_n^{(1)}\right) = \begin{array}{ccccccc} \textcircled{1} & \textcircled{2} & \textcircled{3} & \cdots & \textcircled{n} \end{array}.$$

Sparse LU Factorisation

Proof (continued).

It is easily seen that the only path between any pair of vertices j, i in this graph is the path consisting of all vertices between j and i.

Example. The only path from $j = 2$ to $i = 5$ in $G(\Delta_6^{(1)})$ is given by

Since the intermediate vertices lie between j and i, these paths are fill paths only if they contain at most a single edge; hence the LU factorisation of $\Delta_n^{(1)}$ creates no fill-in and its sparsity pattern is given by

$$L + U = \begin{pmatrix} 1 & \bullet & & & & \\ \bullet & 2 & \bullet & & & \\ & \bullet & \ddots & \ddots & & \\ & & \ddots & \ddots & \bullet & \\ & & & \bullet & n \end{pmatrix}.$$

This shows that the LU factorisation of $\Delta_n^{(1)}$ requires only $O(n)$ memory, and inserting this sparsity pattern into the runtime estimates from slide 7 shows that we can solve $-\Delta_n^{(1)} u_n = f$ in $O(n)$ runtime.

Sparse LU Factorisation

Unfortunately, LU factorisation is no longer true once we move to higher dimensions.

Thm: LU factorisation for solving $-\Delta_n^{(d)} u_n = f$ without fill-reducing permutations

The runtime and memory requirements for solving $-\Delta_n^{(d)} u_n = f$ via LU factorisation without fill-reducing permutations are as follows.

	Runtime	Memory
$d = 2$	$O(N^2)$	$O(N^{3/2})$
$d = 3$	$O(N^{7/3})$	$O(N^{5/3})$

Sparse LU Factorisation

Proof for $d = 2$. As before, the proof consists in drawing the graph of $\Delta_n^{(2)}$ and studying the fill-path-connectedness of pairs of vertices (i, j).

However, considering a general pair of vertices (i, j) in $G(\Delta_n^{(2)})$ for arbitrary n is too abstract to allow for a reasonable discussion.

Instead, I will discuss fill-path-connectedness for just the particular choice $n = 5$ and $j = 13$ and leave it up to you to convince yourself that repeating the presented arguments for different n and (i, j) yields the claimed generalisations.

Sparse LU Factorisation

Proof (continued).

The graph of $\Delta_5^{(2)}$ is given by

$$G\big(\Delta_5^{(2)}\big) = \quad$$
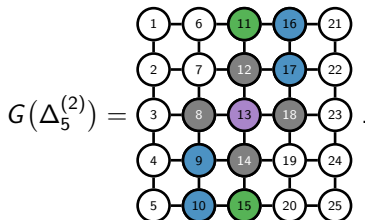
For better readability, I replaced each pair of directed edges ⟶⟷ with a single undirected edge O—O in this picture.

Furthermore, I marked in **black** all neighbours of vertex 13, and I marked in blue and green all vertices which are fill-path-connected to vertex 13 but which are not neighbours of vertex 13.

The next slide will show that this colouring is correct.

Sparse LU Factorisation

Proof (continued).
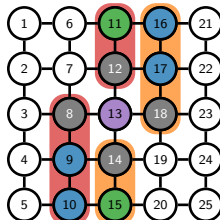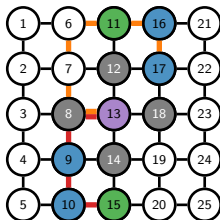
Proof that coloured vertices are fill-path-connected to 13: (left graph)

Observe that all subpaths of

$13 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 15$ and $13 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 11 \rightarrow 16 \rightarrow 17$

of the form $13 \rightarrow$ (blue or green vertex) are fill paths.

*Proof that white vertices are not fill-path-connected to* 13: (right graph)

Any path from 13 to a vertex ≤ 7 has to go through at least one vertex $v \in \{8, \ldots, 12\}$ and is therefore not a fill path since $v > 7$.

Any path from 13 to a vertex ≥ 19 has to go through at least one vertex $v \in \{14, \ldots, 18\}$ and is therefore not a fill path since $v > 13$.

Sparse LU Factorisation
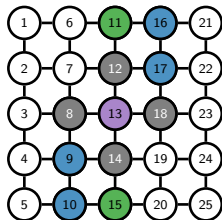
Proof (continued).
We conclude from the graph

$$G\big(\Delta_5^{(2)}\big) = \quad .$$
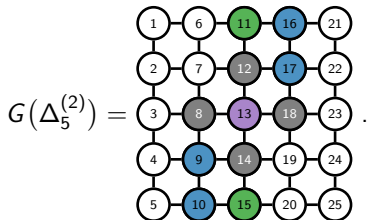
that the sparsity pattern of row 13 of $L + U$ is given by

$$(L + U)[13, :] = (\quad \bullet \, \bullet \, \bullet \, \bullet \, \bullet \, 13 \, \bullet \, \bullet \, \bullet \, \bullet \, \bullet \quad),$$
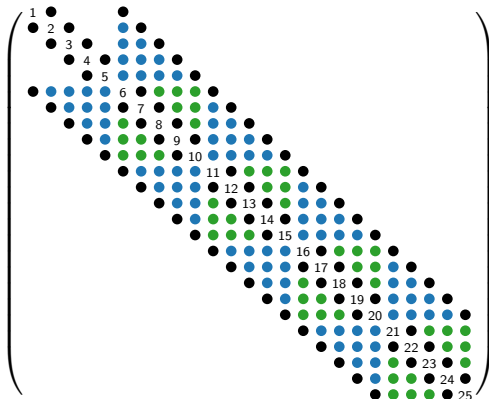
and the sparsity pattern of all of $L + U$ is as shown on the next slide.

Sparse LU Factorisation

Proof (continued).

$$L + U =$$

Colour scheme:

$$(L + U)[i,j] = \begin{cases} \bullet & \iff & i \text{ and } j \text{ are in the same column of } G(\Delta_5^{(2)}). \\ \bullet & \iff & i \text{ and } j \text{ are in neighbouring columns of } G(\Delta_5^{(2)}). \end{cases}$$

Sparse LU Factorisation

Proof (continued).

Generalising the above to arbitrary n, we conclude that

$$\text{nnz}(L[:,k]) = \text{nnz}(U[k,:]) = O(n). \qquad (2)$$

The runtime of computing $LU = \Delta_n^{(2)}$ is thus given by

$$O\left(\sum_{k=1}^{n^2} \text{nnz}(L[:,k]) \, \text{nnz}(U[k,:])\right) = O(n^2) \, O(n) \, O(n) = O(n^4) = O(N^2),$$

and the memory required for storing L (and equally U) is given by

$$\text{nnz}(L) = \sum_{k=1}^{n^2} \text{nnz}(L[:,k]) = O(n^2) \, O(n) = O(n^3) = O(N^{3/2}),$$

as claimed on slide 57.

The *proof for $d = 3$* is analogous. I omit the details.

Sparse LU Factorisation

The nested dissection order

The above runtimes for solving $-\Delta_n^{(d)} u_n = f$ are of course better than the $O(N^3)$ runtime which would result if we did not exploit sparsity, but they are also quite far from the optimal $O(N)$ runtime.

Fortunately, it turns out that we can do better than the above by using the $A \mapsto PAP^T$ trick discussed on slide 39 with a permutation $\pi(i)$ constructed as follows.

Algorithm Nested dissection order

1: Partition the vertices into three sets V_1, V_2, V_{sep} such that there are no edges between V_1 and V_2 (sep stands for *separator*).

2: Arrange the vertices in the order V_1, V_2, V_{sep}, where V_1 and V_2 are ordered recursively according to the nested dissection algorithm and V_{sep} is ordered arbitrarily.

Sparse LU Factorisation

The nested dissection order (continued)

The goal of the nested dissection order is to obtain a matrix of the form

$$PAP^T = \begin{pmatrix} \blacksquare & & \blacksquare \\ & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare \end{pmatrix} \quad \longleftrightarrow \quad G(PAP^T) = \boxed{V_1} \; \boxed{V_2} \; \boxed{V_{\mathrm{sep}}}.$$

This shape has two consequences:

▶ $L[V_2, V_1] = U[V_1, V_2] = 0$, because any path from V_1 to V_2 must pass through V_{sep} and is therefore not a fill-path.

▶ $L[V_{\mathrm{sep}}, V_1 \cup V_2]$, $U[V_1 \cup V_2, V_{\mathrm{sep}}]$ and $(L + U)[V_{\mathrm{sep}}, V_{\mathrm{sep}}]$ are likely to be fairly dense because all of V_1 and V_3 contributes towards the fill paths in these blocks.

For the nested dissection order to be most effective, we should hence choose V_{sep} according to the following criteria:

▶ $|V_{\mathrm{sep}}|$ should be as small as possible since this minimises the likely fill-in in $L[V_{\mathrm{sep}}, V_1 \cup V_2]$, $U[V_1 \cup V_2, V_{\mathrm{sep}}]$ and $(L + U)[V_{\mathrm{sep}}, V_{\mathrm{sep}}]$.

▶ $|V_1|$ and $|V_2|$ should be of roughly equal size since this maximises the area of the blocks $L[V_2, V_1]$ and $U[V_1, V_2]$ which are guaranteed to contain no fill-in.

Sparse LU Factorisation

Separators for $G(\Delta_n^{(2)})$

The above criteria lead us to choose separators in $G(\Delta_n^{(2)})$ as follows.
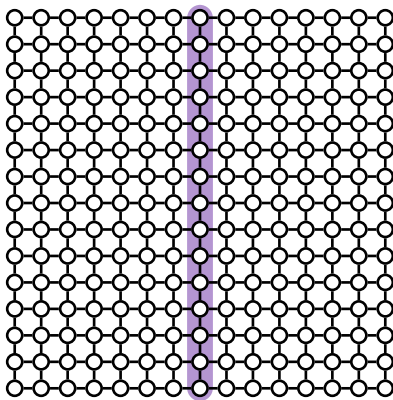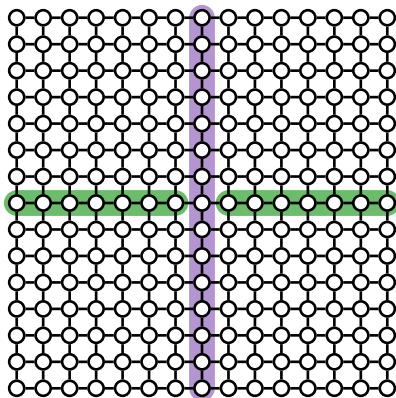

Top-level separator $V_{\text{sep}}^{(1)}$

Note how each separator $V_{\text{sep}}^{(k)}$ is the smallest vertex set splitting each of the remaining white blocks into two blocks of equal size.

Sparse LU Factorisation

Separators for $G(\Delta_n^{(2)})$

The above criteria lead us to choose separators in $G(\Delta_n^{(2)})$ as follows.

Separators $V_{\text{sep}}^{(2)}$ after one level of recursion

Note how each separator $V_{\text{sep}}^{(k)}$ is the smallest vertex set splitting each of the remaining white blocks into two blocks of equal size.

Sparse LU Factorisation

Separators for $G(\Delta_n^{(2)})$

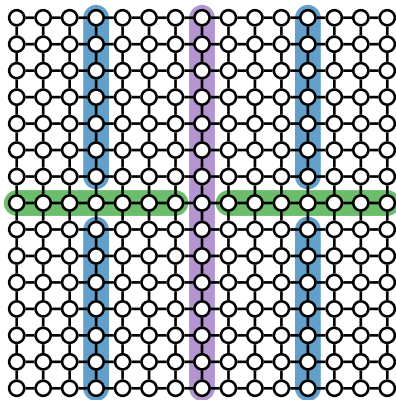The above criteria lead us to choose separators in $G(\Delta_n^{(2)})$ as follows.

Separators $V_{\text{sep}}^{(3)}$ after two levels of recursion

Note how each separator $V_{\text{sep}}^{(k)}$ is the smallest vertex set splitting each of the remaining white blocks into two blocks of equal size.

Sparse LU Factorisation

Separators for $G(\Delta_n^{(2)})$

The above criteria lead us to choose separators in $G(\Delta_n^{(2)})$ as follows.
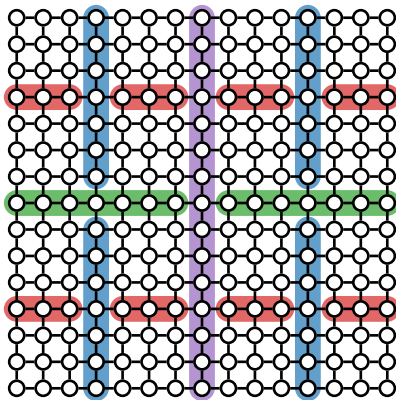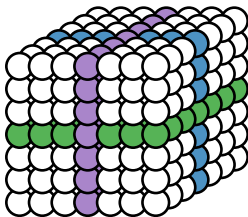

Separators $V_{\text{sep}}^{(4)}$ after three levels of recursion

Note how each separator $V_{\text{sep}}^{(k)}$ is the smallest vertex set splitting each of the remaining white blocks into two blocks of equal size.

Sparse LU Factorisation

Separators for $G(\Delta_n^{(3)})$

In three dimensions, the optimal separator sets are two-dimensional slices through a three-dimensional cube.

Separators $V_{\text{sep}}^{(1)}$, $V_{\text{sep}}^{(2)}$ and $V_{\text{sep}}^{(3)}$.

Sparse LU Factorisation

Nested dissection ordering allows us to significantly reduce the runtime and memory requirements of solving $-\Delta_n^{(d)} u_n = f$ relative to the requirements for the unpermuted case discussed on slide 57.

Thm: LU factorisation for solving $-\Delta_n^{(d)} u_n = f$ with nested dissection permutation

The runtime and memory requirements for solving $-\Delta_n^{(d)} u_n = f$ via LU factorisation with a nested dissection permutation are as follows.

	Runtime	Memory
$d = 2$	$O(N^{3/2})$	$O(N \log(N))$
$d = 3$	$O(N^2)$	$O(N^{4/3})$

Sparse LU Factorisation

Partial proof. For the top-level separator $V_{\text{sep}}^{(1)}$, we have

$$|V_{\text{sep}}^{(1)}| = n^{d-1} \qquad \text{and} \qquad (L+U)[V_{\text{sep}}, V_{\text{sep}}] \text{ is dense.}$$

The runtime required for factorising just this bottom-right corner of $P\Delta_n^{(d)} P^T$ is hence

$$O\big(|V_{\text{sep}}^{(1)}|^3\big) = O\big(n^{3(d-1)}\big) = \begin{cases} O(n^3) = O(N^{3/2}) & \text{if } d = 2, \\ O(n^6) = O(N^2) & \text{if } d = 3. \end{cases}$$

This explains why the overall runtimes cannot be lower than what is reported in the above table. Showing that these lower bounds on the runtime are also achievable is beyond the scope of this module.

Sparse LU Factorisation

It turns out that nested dissection ordering is in fact optimal in the following sense.

Thm: Optimality of nested dissection

The runtime and memory requirements for computing the LU factorisation of $P\Delta_n^{(d)}P^T$ for any permutation P are at least as large as those reported on slide 68.

Proof. See Hoffmann, Martin, Rose (1973), *Complexity bounds for regular finite difference and finite element grids*.

Sparse LU Factorisation

Numerical demonstration

See `nested_dissection()`, `runtimes()` and `sparsity_pattern()`.

Conclusion

We have seen in this lecture that LU factorisation can solve $-\Delta_n^{(d)} u_n = f$ in an optimal $O(N)$ runtime for $d = 1$, but also that the runtime changes to $O(N^\alpha)$ with increasingly larger $\alpha > 1$ as we go from $d = 1$ to $d = 2$ and $d = 3$.

This is unfortunate because larger d also mean an exponentially growing system size $N = n^d$. LU factorisation is thus the furthest from optimality precisely in the case $d = 3$ where achieving optimal performance would be the most important.

This failure of the LU factorisation motivates us to look at a completely different class of algorithms for solving $-\Delta_n^{(d)} u_n = f$ in the next lecture.

Sparse LU Factorisation

Summary

▶ Path and fill path theorems:

$$A^p[i,j] \neq 0 \quad \Longleftrightarrow \quad \exists \text{ path } j \to i \text{ of length } p \text{ in } G(A)$$

$$A^{-1}[i,j] \neq 0 \quad \Longleftrightarrow \quad \exists \text{ path } j \to i \text{ in } G(A)$$

$$(L+U)[i,j] \neq 0 \quad \Longleftrightarrow \quad \exists \text{ fill path } j \to i \text{ in } G(A)$$

▶ Cost of sparse LU factorisation with nested dissection order for partial differential equations in d dimensions:

	Runtime	Memory
$d = 1$	$O(N)$	$O(N)$
$d = 2$	$O(N^{3/2})$	$O(N \log(N))$
$d = 3$	$O(N^2)$	$O(N^{4/3})$

$N = O(n^d)$ denotes the number of unknowns.
n denotes the number of grid points in each direction.