MA3227 Numerical Analysis II

Lecture 4: Sparse LU Factorisation

Simon Etter

National University of Singapore

2019/2020

Sparse LU Factorisation

Introduction

We have seen in Lectures 2 and 3 how partial differential equations give rise to very large but sparse linear systems.

So far, we solved these systems using Julia's backslash operator \.
In this lecture, we will take a closer look at how \ works.

Recap from Numerical Analysis I

LU factorisation (aka Gaussian elimination) is the method of choice for solving dense linear systems. Julia's \ uses LU factorisation.

LU factorisation of a dense $N \times N$ matrix requires $\mathcal{O}\big(N^3\big)$ floating-point operations. This is way too much for the applications we have in mind.

Example

Consider the 2D Poisson equation on an $n \times n$ grid with $n = 1000$.
LU factorisation of $\Delta_n^{(2)}$ requires $\mathcal{O}(n^6) = \mathcal{O}\big(10^{18}\big)$ operations.
Assuming 10^9 ops/sec (1GHz), this will take approximately 32 years!

Question: Can we compute $A = LU$ more efficiently if A is sparse?

Sparse LU Factorisation

Recap LU factorisation

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

Question: Are L and U sparse if A is sparse?

Sparse LU Factorisation

Example

$$A_1 := \begin{pmatrix} 1 & & 1 & \\ & 1 & 1 & \\ & & 1 & \\ 1 & 1 & & 1 \end{pmatrix} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ 1 & 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} 1 & & 1 & \\ & 1 & 1 & \\ & & 1 & \\ & & & 1 \end{pmatrix} =: L_1 U_1,$$

$$A_2 := \begin{pmatrix} 1 & & 1 & \\ & -1 & 1 & \\ & & 1 & \\ 1 & 1 & & 1 \end{pmatrix} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ 1 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & & 1 & \\ & -1 & 1 & \\ & & 1 & \\ & & & 1 \end{pmatrix} =: L_2 U_2.$$

Observations:

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

Sparse LU Factorisation

Terminology

Let $A \in \mathbb{R}^{N \times N}$ be a matrix and B some quantity derived from A.

Examples: $B = A^2$, $B = A^{-1}$, B is one of the LU factors.

- *Sparsity pattern/structure of A:* list of all (i, j) such that $A[i, j] \neq 0$.
- *Structural nonzero in B:* (i, j) such that $B[i, j]$ could be nonzero for the given sparsity pattern of A. See examples below.
- *Sparsity pattern/structure of B:* list of all (i, j) such that $B[i, j]$ is structurally nonzero.
- *Fill-in entry in B:* (i, j) such that $A[i, j] = 0$ but $B[i, j]$ is structurally nonzero.

Sparse LU Factorisation

Example 1

On page 4, $L_1[4,3]$ and $L_2[4,3]$ are both structural nonzeros even though $L_2[4,3] = 0$.

Example 2

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

- $B[1,2] = 1 \times 1 + 1 \times (-1) = 0$ is structurally nonzero.
- $B[2,1] = 0 \times 1 + (-1) \times 0 = 0$ is structurally zero.

Sparse LU Factorisation

Remark

A common theme underlying the terminology is that for derived quantities B, we always talk about structural zeros/nonzeros and not about actual zeros/nonzeros.

Reasons for looking at structure rather than actual values:

- It is much easier.
- It provides a worst-case estimate.
- Cancellation (i.e. a structurally nonzero entry being zero) is rare.

Sparse LU Factorisation

Outlook

The following slides will present statements regarding the sparsity patterns of polynomials, the inverse and the LU factorisation of A.

All of these statements are meant in the structural sense.

Remark

For LU factorisation, we usually talk about the sparsity pattern of $L + U$ to avoid having to provide two almost identical statements for the lower and upper triangles.

Note that in the structural sense, we have

$$\text{structure}(L + U) = \text{structure}(L) \cup \text{structure}(U).$$

Sparse LU Factorisation

Graph of a sparse matrix $A \in \mathbb{R}^{n \times n}$

Graph $G(A) := \big(V(A), E(A)\big)$ given by

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

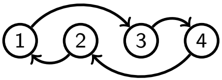
Note transpose in $E(A)$: entry $A[i,j]$ corresponds to edge $j \to i$.

Path in $G = (V, E)$

Ordered sequence $k_0, \dots, k_p \in V$ such that $k_{q-1} \to k_q \in E$ for all q.
Number of edges p is called the length of the path.

Example (Numbers and $\bullet$ indicate nonzeros.)

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

$G(A) =$ 

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

Note that I do not draw edges $i \to i$ to keep the graph readable.

Sparse LU Factorisation

Path theorem for matrix powers

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

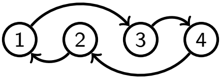
Proof (not examinable).

$$A^p[i,j] = \sum_{k_{p-1}} \dots \sum_{k_1} A[i, k_{p-1}] \, \dots \, A[k_a, k_{a-1}] \, \dots \, A[k_1, j].$$

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

Example (Numbers and $\bullet$ indicate nonzeros in A. ${\color{red}\bullet}$ indicates fill-in.)

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

$G(A) =$ 

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

Note that nonzeros of A^q with $q < p$ are also nonzeros of A^p since we can extend paths of length q to paths of length p by adding edges $i \to i$.

Sparse LU Factorisation

Path theorem for inverse

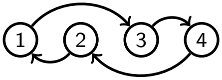
$$A^{-1}[i,j] \neq 0 \quad \Longleftrightarrow \quad \exists \text{ path } j \to i.$$

Proof (not examinable).

- $A^{-1} = p(A)$ for polynomial $p(x)$ interpolating $\frac{1}{x}$ on eigenvalues of A.
- Hence entry (i,j) of $A^{-1} = \sum_{p=0}^{n-1} c_p \, A^p$ is nonzero if there is a path $j \to i$ of arbitrary length.

Example (Numbers and $\bullet$ indicate nonzeros in A. ${\color{red}\bullet}$ indicates fill-in.)

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

$G(A) =$ 

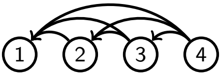
$A^{-1}[2,1] \neq 0$ because $1 \to 3 \to 4 \to 2$ is a path in $G(A)$.

Sparse LU Factorisation

Corollaries of path theorem

- If $G(A)$ is connected (there exists a path between any pair of vertices), then A^{-1} is dense (all entries of A^{-1} are structurally nonzero).
- If $G(A)$ is disconnected, i.e. $A = \begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} \end{pmatrix}$, then $A^{-1} = \begin{pmatrix} A_{11}^{-1} & 0 \\ 0 & A_{22}^{-1} \end{pmatrix}$.
- Inverse of upper/lower triangular matrix is upper/lower triangular.

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

$G(A) =$ 

Sparse LU Factorisation

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

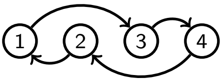
Fill path theorem

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

Proof (not examinable). See next two slides.

Example (Numbers and $\bullet$ indicate nonzeros in A. $\color{red}{\bullet}$ indicates fill-in.)

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

$G(A) =$ 

$L[3, 2] \neq 0$ because $2 \to 1 \to 3$ is a fill path in $G(A)$.
$L[4, 1] = 0$ because $1 \to 3 \to 4$ is not a fill path in $G(A)$.

Sparse LU Factorisation

Lemma for fill path theorem (not examinable)

Let $i, j \in \{1, \dots, n\}$ and set $\ell := \big\{1, \dots, \min\{i,j\} - 1\big\}$. Then,

$$\begin{aligned} U[i,j] &= A[i,j] - A[i,\ell]\, A[\ell,\ell]^{-1}\, A[\ell,j] && \text{for } i \le j, \\ L[i,j]\, U[j,j] &= A[i,j] - A[i,\ell]\, A[\ell,\ell]^{-1}\, A[\ell,j] && \text{for } i \ge j. \end{aligned}$$

Proof. Block LU factorisation with $\bar r := \big\{\min\{i,j\}, \dots, n\big\}$:

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

Let $\quad L_1 U_1 = A[\ell,\ell], \quad L_2 U_2 = A[\bar r,\bar r] - A[\bar r,\ell]\, A[\ell,\ell]^{-1}\, A[\ell,\bar r].$

Full factorisation is then given by

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

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

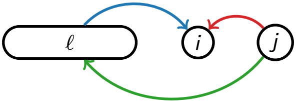
Fill path theorem (repeated from earlier slide)

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

Proof (not examinable). According to lemma on previous slide:

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

1st term makes $U[i,j]$ nonzero if there is a fill path $j \to i$ of length 1.
2nd term makes $U[i,j]$ nonzero if there is a fill path $j \to i$ of length > 1.
Same arguments apply for $L[i,j]\, U[j,j]$.



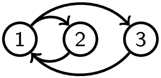
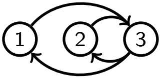
Sparse LU Factorisation

Observation

Whether or not a path is a fill-path depends on the numbers we assign to the vertices. Sometimes we can reduce the number of fill paths by reordering the vertices.

Example

Consider the two graphs

$G_1 =$ , $G_2 =$ 

which are equivalent up to reordering of the vertices.

We observe:

- $3 \to 1 \to 2$ and $2 \to 1 \to 3$ are fill paths in G_1.
- G_2 has no fill paths.

Question: What does "renumber the vertices" mean in terms of matrices?

Sparse LU Factorisation

Example (continued)

Example matrix corresponding to G_1 and its LU factorisation:

$$A_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 0 \\ 1 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 1 & & \\ 1 & 1 & \\ 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ & 1 & -1 \\ & & 1 \end{pmatrix}.$$

Swapping rows and columns 1 and 3 yields a matrix corresponding to G_2:

$$A_2 = \begin{pmatrix} 3 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & & \\ 0 & 1 & \\ \frac{1}{3} & \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 3 & 0 & 1 \\ & 2 & 1 \\ & & \frac{1}{6} \end{pmatrix}.$$

Furthermore, it is clear that

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 0 \\ 1 & 0 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \quad \Longleftrightarrow \quad \begin{pmatrix} 3 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_3 \\ x_2 \\ x_1 \end{pmatrix} = \begin{pmatrix} b_3 \\ b_2 \\ b_1 \end{pmatrix}.$$

Example (continued)

We conclude from the above that sometimes it is possible to transform a linear system into an equivalent linear system with less fill-in simply by swapping rows and columns.

The key to describing this swapping of rows and columns more rigorously are permutations.

Sparse LU Factorisation

Permutations

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

There are several ways to represent a permutation:

- As a function $\pi(i)$ as above.
- As a matrix $P \in \mathbb{N}^{n\times n}$ with $P[i,j] = \begin{cases} 1 & \text{if } j = \pi(i), \\ 0 & \text{otherwise.} \end{cases}$
- As a vector $p = [\pi(1), \dots, \pi(n)]$.

Example

$$\pi(1) = 2, \qquad \pi(2) = 4, \qquad \pi(3) = 3, \qquad \pi(4) = 1,$$

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \qquad p = [2, 4, 3, 1].$$

Sparse LU Factorisation

Properties of permutation matrices

- P is invertible, and it holds $P^{-1} = P^T$
- $(PA)[i,j] = A[\pi(i), j]$, i.e. PA permutes the rows of A.
- $(AP^T)[i,j] = A[i, \pi(j)]$, i.e. AP^T permutes the columns of A.

Proof. We show the first claim by showing that $PP^T = I$. To this end, we compute

$$(PP^T)[i,j] = \sum_{k=1}^{n} P[i,k]\, P[j,k].$$

Each term in this sum is nonzero if and only

$$P[i,k], P[j,k] \neq 0 \quad \Longleftrightarrow \quad k = \pi(i), k = \pi(j).$$

If $i \neq j$, the last condition cannot be satisfied and hence $P[i,j] = 0$.
If $i = j$, then $k = \pi(i) = \pi(j)$ is the only term satisfying this condition and hence $P[i,i] = 1$. Thus, $PP^T = I$ as claimed.

The other two claims can be shown by similar arguments.

Sparse LU Factorisation

Vertex renumbering as applying permutation matrices

In terms of the linear system, renumbering the vertices corresponds to the transformation

$$Ax = b \qquad \iff \qquad (PAP^T)(Px) = (Pb) \tag{1}$$

with P some permutation matrix.

Example

Recall from page 17 the two linear systems

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 0 \\ 1 & 0 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \quad \iff \quad \begin{pmatrix} 3 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_3 \\ x_2 \\ x_1 \end{pmatrix} = \begin{pmatrix} b_3 \\ b_2 \\ b_1 \end{pmatrix}.$$

These linear systems are related by the transformation (1) with

$$P = \begin{pmatrix} & & 1 \\ & 1 & \\ 1 & & \end{pmatrix}.$$

Sparse LU Factorisation

Corollary of fill path theorem (and summary of the above)
Given a linear system

$$Ax = b,$$

it is sometimes possible to reduce the amount of fill-in by replacing the original problem with

$$(PAP^T)(Px) = Pb$$

where P is a suitable permutation matrix.

P is fully specified if we specify the corresponding permutation π on the vertices.

Sparse LU Factorisation

Exercise

Consider the matrix

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

- Determine the sparsity pattern of the LU factorisation of A.
- Find a permutation of the rows and columns of A such that there is no fill-in.

Sparse LU Factorisation

Outlook

Our aim for the remainder of this lecture is to use the fill path theorem to analyse the costs of computing the LU factorisation of $\Delta_n^{(1)}$ and $\Delta_n^{(2)}$.

The general outline of this analysis is as follows.

- Determine the graph $G\big(\Delta_n^{(d)}\big)$.
- Determine the number of fill-path-neighbours n_{fpn} per vertex j, i.e. the number of vertices i which are reachable from j by fill paths

The memory and runtime requirements are then given by

$$\text{memory} = \mathcal{O}\big(N\, n_{\text{fpn}}\big) \qquad \text{and} \qquad \text{runtime} = \mathcal{O}\big(N\, n_{\text{fpn}}^2\big)$$

where $N = n^d$ denotes the number of rows/columns.

Proof. Memory requirements are obvious.

Runtime: In each column, we have $\mathcal{O}\big(n_{\text{fpn}}\big)$ entries to eliminate.
Each elimination operates on rows with $\mathcal{O}(n_{\text{fpn}})$ nonzeroes and hence requires $\mathcal{O}(n_{\text{fpn}})$ operations. Estimate follows by multiplying

(# columns)$\times$(# eliminations per column)$\times$(# operations per elimination).

Sparse LU Factorisation

LU factorisation of $\Delta_n^{(1)}$

$$G\left(\Delta_5^{(1)}\right) = \begin{tikzcd} 1 \arrow[r, bend left] & 2 \arrow[l, bend left] \arrow[r, bend left] & 3 \arrow[l, bend left] \arrow[r, bend left] & 4 \arrow[l, bend left] \arrow[r, bend left] & 5 \arrow[l, bend left] \end{tikzcd}$$

Observation: $G\left(\Delta_n^{(1)}\right)$ does not contain any fill-paths of length > 1;

Conclusion: $n_{\mathrm{fpn}} = \mathcal{O}(1), \quad \text{memory} = \mathcal{O}(N), \quad \text{runtime} = \mathcal{O}(N).$

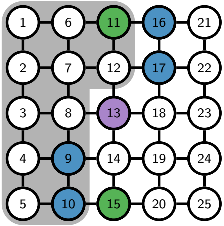
Discussion

- $\mathcal{O}(N)$ scaling is much better than $\mathcal{O}(N^3)$ scaling for dense LU.
- $\mathcal{O}(N)$ scaling is optimal: no algorithm operating on $\mathcal{O}(N)$ data can do better than $\mathcal{O}(N)$ memory and runtime.

Sparse LU Factorisation

LU factorisation of $\Delta_n^{(2)}$

$G\left(\Delta_5^{(2)}\right) =$



Marked in gray are all vertices numbered less than 13.

Highlighted in blue and green are the vertices which are fill-path-connected to vertex 13.

Row 13 in $L + U$ has therefore the sparsity pattern

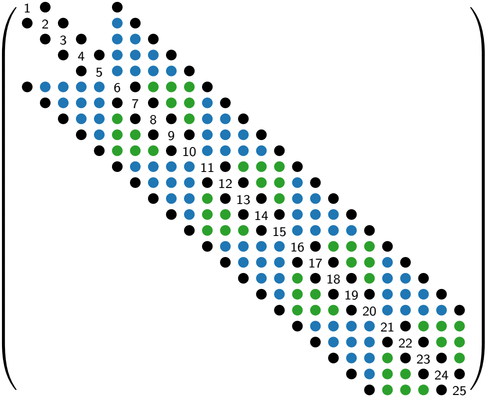
$$\left(\quad \bullet\ \bullet\ \bullet\ \bullet\ \bullet\ 13\ \bullet\ \bullet\ \bullet\ \bullet\ \bullet \quad \right)$$

and the full sparsity pattern of $L + U$ is as shown on the next slide.

Sparse LU Factorisation

LU factorisation of $\Delta_n^{(2)}$ (continued)

$$L + U =$$



Conclusion:

$$\begin{aligned} n_{\text{fpn}} &= \mathcal{O}(n) &&= \mathcal{O}\big(N^{1/2}\big), \\ \text{memory} &= \mathcal{O}(Nn) &&= \mathcal{O}\big(N^{3/2}\big), \\ \text{runtime} &= \mathcal{O}(Nn^2) &&= \mathcal{O}\big(N^2\big). \end{aligned}$$

Sparse LU Factorisation

Discussion

Sparse LU applied to $\Delta_n^{(2)}$ performs better than dense LU ($\mathcal{O}(N^2)$ vs. $\mathcal{O}(N^3)$ runtime), but it is still quite expensive.

Corollary of fill path theorem: fill-in depends on order of vertices.

It turns out that we can compute the LU factorisation of $\Delta_n^{(2)}$ more economically if we reorder the vertices.

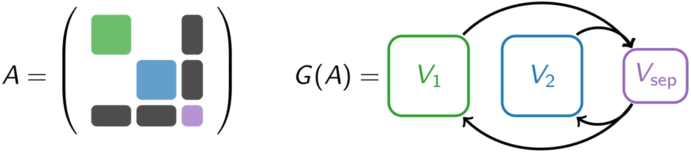
The next slide introduces the nested dissection order. This order is asymptotically optimal, i.e. no vertex order leads to less fill-in in the big-$\mathcal{O}$ sense than the nested dissection order.

Sparse LU Factorisation

Algorithm 1 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 (subscript 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.

After reordering:

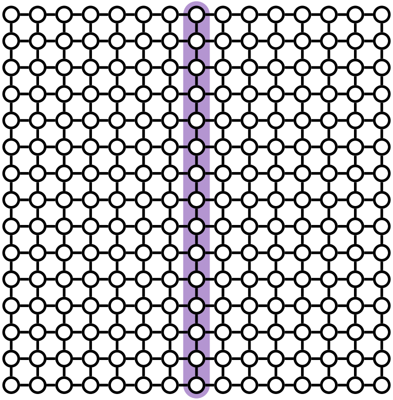


Observation

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

Sparse LU Factorisation

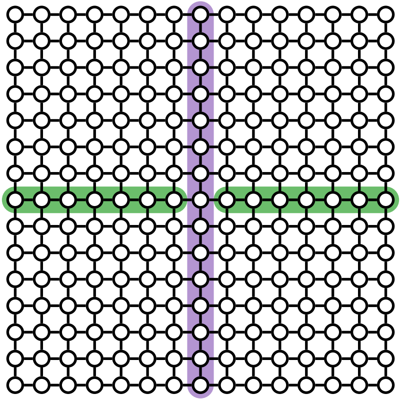
Separators for 2D mesh



The highlighted vertices indicate separator sets at various levels of the nested dissection recursion.

Sparse LU Factorisation

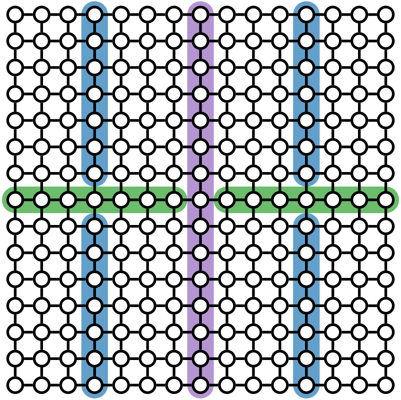
Separators for 2D mesh



The highlighted vertices indicate separator sets at various levels of the nested dissection recursion.

Sparse LU Factorisation

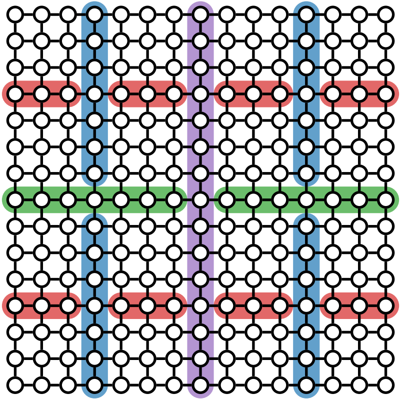
Separators for 2D mesh



The highlighted vertices indicate separator sets at various levels of the nested dissection recursion.

Sparse LU Factorisation

Separators for 2D mesh



The highlighted vertices indicate separator sets at various levels of the nested dissection recursion.

Sparse LU Factorisation

LU factorisation of $\Delta_n^{(2)}$ with nested dissection

It can be shown that the requirements of LU factorisation of $\Delta_n^{(d)}$ with nested dissection ordering are as follows.

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

A rigorous proof of this result is beyond our scope.
Instead, I will present an argument why the runtimes cannot be lower.

Observation: Diagonal blocks associated with the V_{sep} are dense.
Factorising these blocks hence requires $\mathcal{O}\big(|V_{\text{sep}}|^3\big)$ operations.

In 2D: largest V_{sep} satisfies $|V_{\text{sep}}| = \mathcal{O}(n) = \mathcal{O}\big(N^{1/2}\big)$.
In 3D: largest V_{sep} satisfies $|V_{\text{sep}}| = \mathcal{O}(n^2) = \mathcal{O}\big(N^{2/3}\big)$.
Recall: $n = \#$ mesh points per dimension, $N = n^d =$ total $\#$ mesh points.

Taking these $|V_{\text{sep}}|$ to the third power yields the numbers in the table.

Sparse LU Factorisation

Discussion

LU factorisation is popular for solving linear systems because it is very easy to use: all we have to do is provide A and b and hit enter.

Unfortunately, this lecture showed that the simplicity of LU comes at the price of superlinear memory and runtime requirements for partial differential equations in dimensions $d > 1$.

In the next few weeks, we will look at algorithms which are more complicated to use but which perform better than LU when used correctly.

Sparse LU Factorisation

Discussion (continued)

Some further practical complications (not examinable):

- Computing a nested dissection order can be quite expensive.

  Solution: use Approximate Minimum Degree (AMD) order instead. This order sorts the vertices in increasing order of degree (number of neighbours). Because even computing degrees is expensive, we use approximate degrees instead.

- LU factorisation requires pivoting (interchanging of rows) to be numerically stable. This may require striking a balance between minimising fill-in and maintaining stability.

  Solution: some matrices do not require pivoting, namely diagonally dominant and symmetric positive definite ones.

Julia's \ automatically takes care of these complications.
It internally uses the `SuiteSparse` package which is also used by Matlab.

Sparse LU Factorisation

Summary

$$
\begin{aligned}
A^p[i,j] \neq 0 \quad &\iff \quad \exists \text{ path } j \to i \text{ of length } p \\
A^{-1}[i,j] \neq 0 \quad &\iff \quad \exists \text{ path } j \to i \\
(L+U)[i,j] \neq 0 \quad &\iff \quad \exists \text{ fill path } j \to i
\end{aligned}
$$

Cost of sparse LU factorisation for partial differential equations:

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