Summary and Trajectory

- Compexity estimates as a function of space dimentions, d:
  - banded solvers
  - nested dissection
  - fast diagonalization
  - simple iterative solvers (Jacobi, CG)

- Subsequently –
  - FEM
  - matrix and vector norms
  - projection methods (CG, GMRES, biCGSTAB,...)
  - preconditioners
  - multigrid

Today

- nested dissection (quick review and complexity)
- fast diagonalization method (FDM)
- Jacobi (and CG complexity)

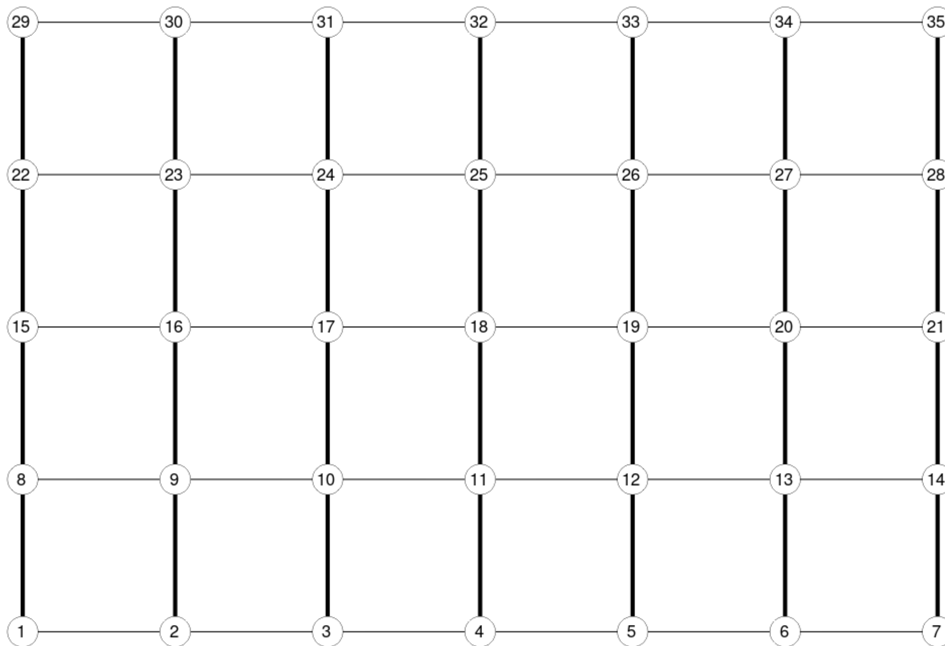
Nested Dissection Orderings

- So far, we've considered lexicographical orderings in which we enumerate nodes in the graph by advancing first in one direction, then the next, and so on.

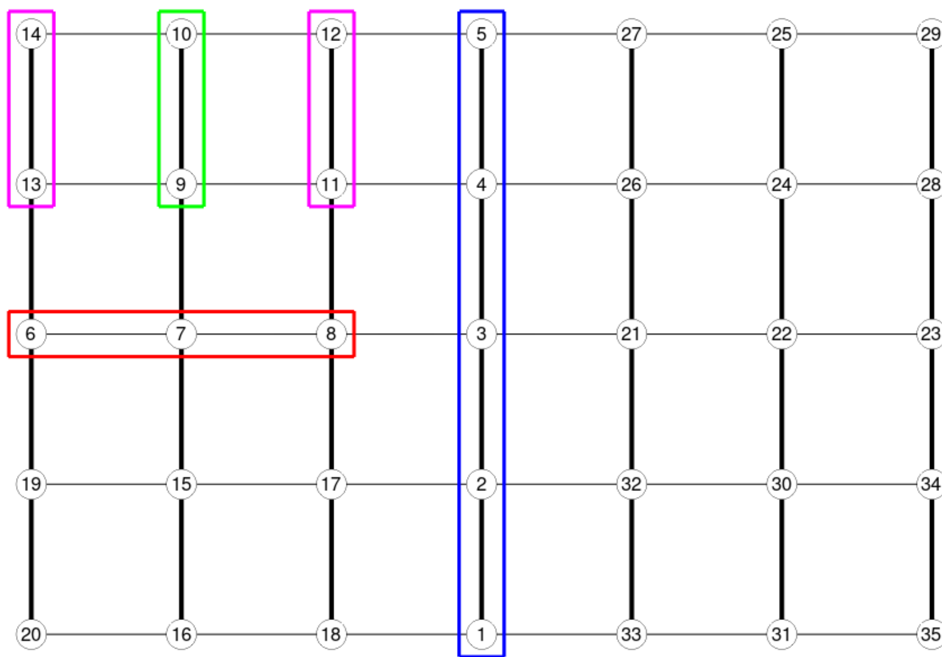
- Other orderings such as symmetric-approximate-minimal-degree (symamd in matlab) and *nested-dissection* [George'73,Martinsson'14,Schmitz&Ying'14] are designed to minimize fill in the LU factors.

- We consider nested-dissection here, which we illustrate via a 2D example.

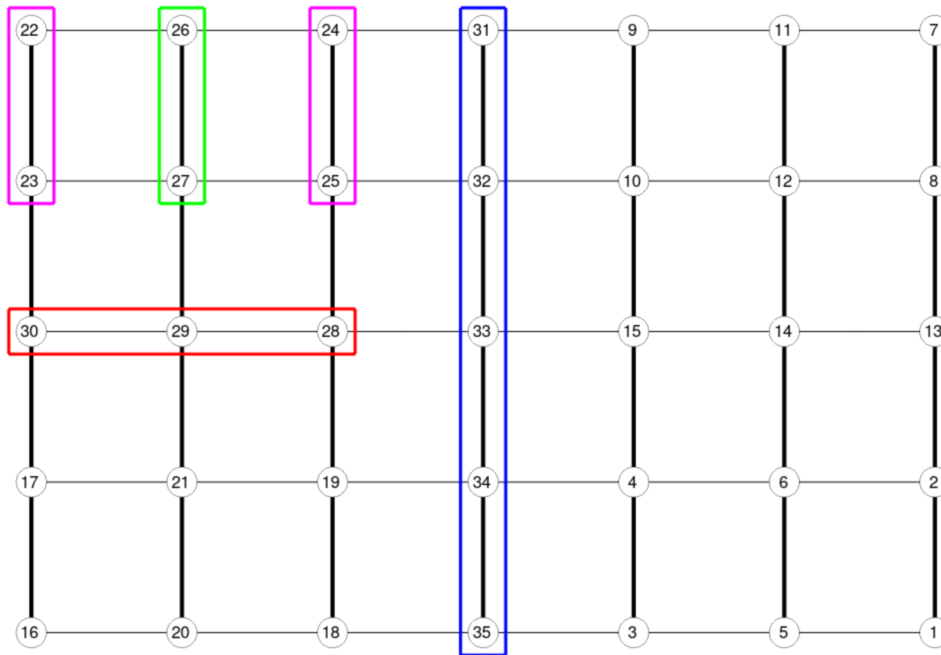
- Consider the graph below, shown initially with a lexicographical ordering.



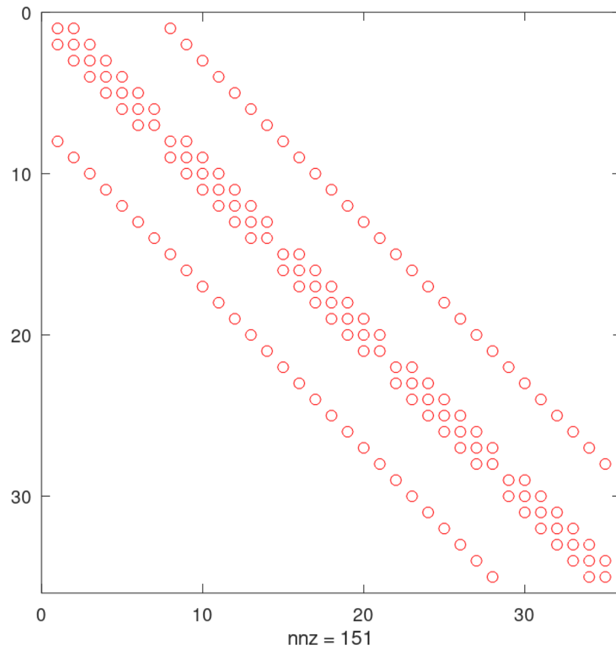
- To begin, we identify the smallest separator set that cuts the graph into two (nearly) equal pieces and enumerate the entries of that separator.
- Next, we do the same with one of the subgraphs.
- Following a depth-first traversal, we enumerate the remaining subgraphs in a recursive way.
- The first few separators are shown below.
- Here, 11–12 and 13–14 represent the leaves of the tree.

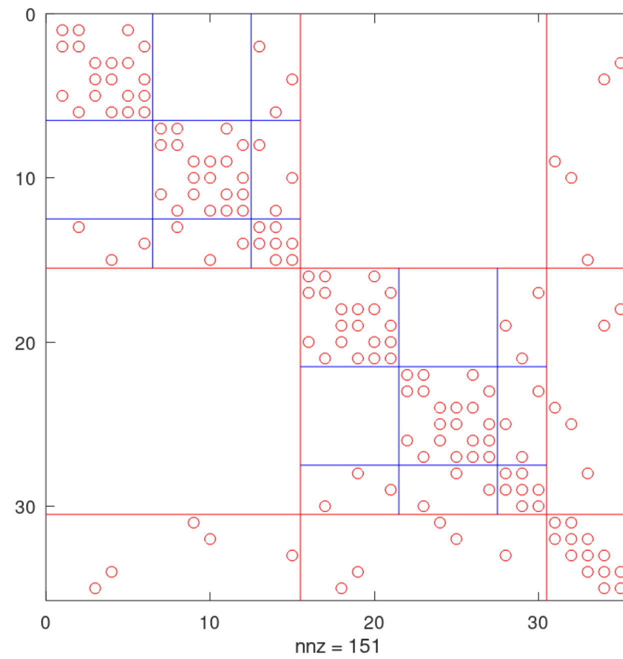


- Finally, we reverse the ordering so that the first separator is enumerated *last*: $k = (n+1) - k$ for $k = 1, \ldots, n$.



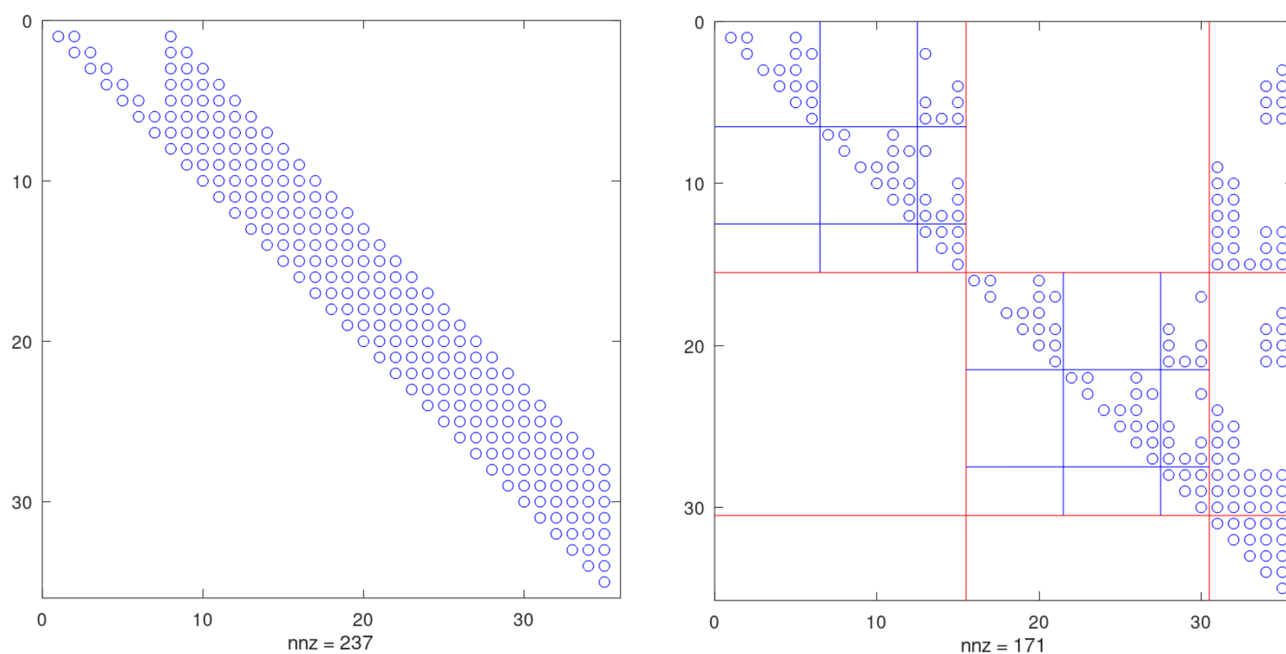
- We can see what this ordering does to the matrix A in the image below.





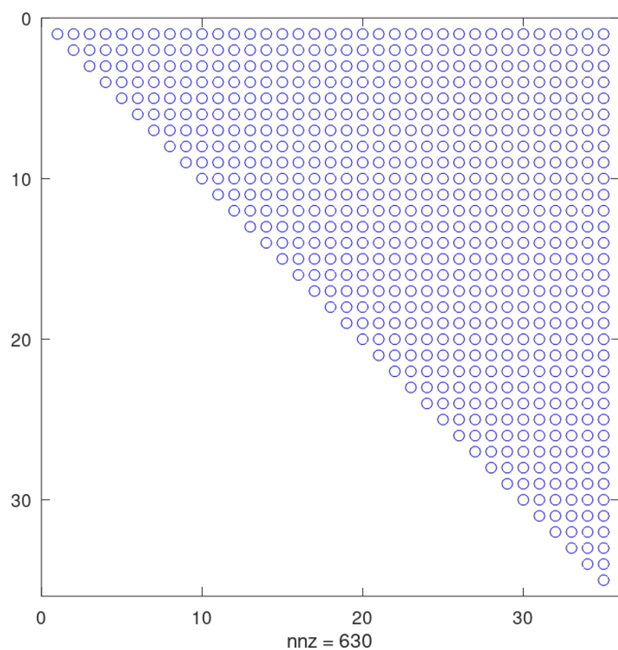
- On the left we have the pentadiagonal matrix from the lexicographical ordering and on the right that coming from the nested-dissection (ND) ordering.
- Notice the recursively smaller isolated principal submatrices of the ND-ordered matrix.
- It is clear that at least 4-way parallelism can be applied in factoring the four 6×6 blocks, and 2-way parallelism in the larger 15×15 blocks.
- Additionally, we see the 5×5 tridiagonal matrix in the lower right corner, which is associated with the principal separator that has 5 nodes.

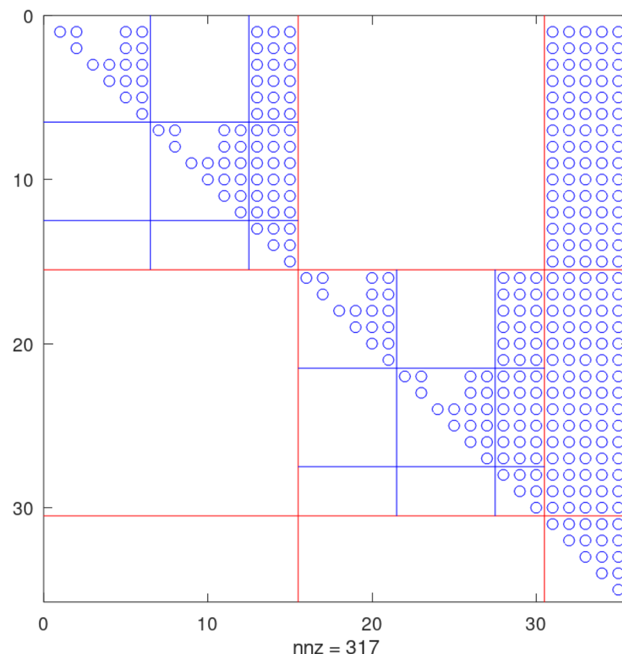
- Now look at the LU factors for these two matrices.



- Notice that the ND case has fewer nonzeros than the lexicographical case.

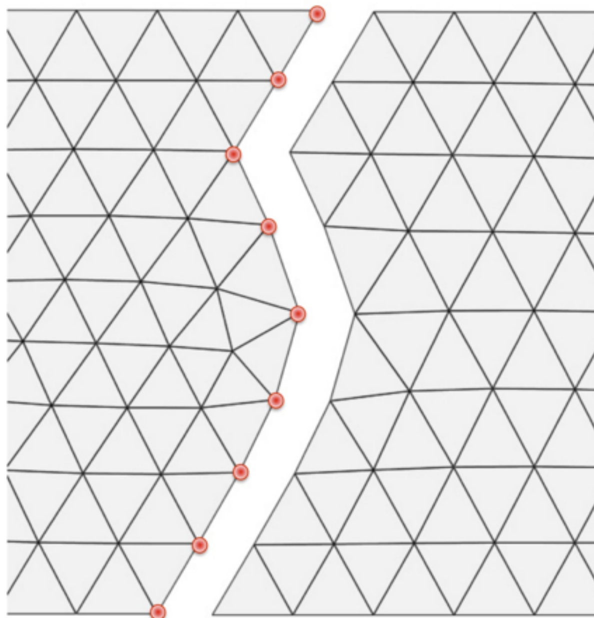
- The fill reduction is even more dramatic for the inverse factors, U^{-1}.





- Clearly, lexicographical leads to $\sim n^2/2$ nonzeros in U^{-1}.
- For ND, it is easy to show that U^{-1} has $< 3\,n^{\frac{3}{2}}$ nonzeros in the 2D case and $< 2.5\,n^{\frac{5}{3}}$ nonzeros in the 3D case.
- Generally, one would not compute U^{-1}, but it is a reasonable strategy for developing fast parallel coarse-grid solvers.

- ND can be extended to unstructured graphs, e.g., finite-elements.



- Recursive coordinate bisection (RCB) or (better) recursive spectral bisection (RSB) [Pothen& Simon'91] can be used to partition the graph.

- Typically, you would want to partition on the *dual-graph* having nodes at the element centers. **(Why?)**

- ND works for variable-coefficient problems—no change to the algorithm (at least, for the positive-definite case).

- Ordering of the subdomain interior DOFs can influence the fill.

- For the constant coefficient case, fast Poisson solvers are *much* faster, with $O(n \log n)$ complexity for the uniform grid case (`fishpack`), or $\sim 12n^{\frac{4}{3}}$ (all BLAS3) if the grid is based on a 3D tensor product of nonuniform one-dimensional grids.

- Three articles/notes: George'73, Martinsson '14, Schmitz & Ying '14.

- For an $N \times N$ grid in 2D or $N \times N \times N$ grid in 3D, we have the following complexities:

	Lexicographical	Nested-Dissection
2D Factor Cost	$\sim 2n^2$	$\sim (19.07\ldots)n^{\frac{3}{2}}$
2D Solve & Storage Cost	$\sim 2n^{\frac{3}{2}}$	$\sim \frac{31}{8} n \log_2 n$
3D Factor Cost	$\sim 2n^{\frac{7}{3}}$	$O(n^2)$
3D Solve & Storage Cost	$\sim 2n^{\frac{5}{3}}$	$O(n^{\frac{4}{3}})$

- Note that the constants in the lexicographical case are very small.

- The constants for the nested dissection are less clear. (Martinsson's notes do not account for the complexity of the side blocks, $L_{ii}^{-1} A_{i\Gamma}$.)

- The constants for the 2D nested dissection are from A. George's 1973 paper.

- Note that $20n^{\frac{3}{2}} < 2n^2$ for $n > 100$—meaning it starts to pay for 2D problems that are larger than 10×10.

- The costs, however, will be relatively (2-10×) higher for nested-dissection because indirect addressing will increase memory traffic and inhibit pipelined arithmetic. So ND may require a slightly larger value of N to be competitive. This hypothesis could be tested in matlab.

Poisson Equation in $\mathbb{R}^3$

We now extend the 1D and 2D concepts to the most important 3D case. The short story is that the 3D stiffness matrix has the beautifully symmetric form

$$\begin{aligned} A_{3D} &= (I_z \otimes A_{2D}) + (A_z \otimes I_{2D}) \\ &= (I_z \otimes I_y \otimes A_x) + (I_z \otimes A_y \otimes I_x) + (A_z \otimes I_y \otimes I_x). \end{aligned} \quad (1)$$

and the discrete system is, as before, $A_{3D}\underline{u} = \underline{f}$. This of course is the form that arises for a finite difference discretization of $-\nabla^2 u = f$ in $\Omega = [0,1]^3$, $u = 0$ on $\partial\Omega$, or, more explicitly,

$$-\left(\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2} + \frac{\delta^2 u}{\delta z^2} \right) = f(x_i, y_j, z_k), \quad (2)$$

with

$$\left. \frac{\delta^2 u}{\delta z^2} \right|_{ijk} := \frac{u_{ij,k+1} - 2u_{ijk} + u_{ij,k-1}}{\Delta z^2}, \quad (3)$$

and equivalent expressions for $\frac{\delta^2 u}{\delta x^2}$ and $\frac{\delta^2 u}{\delta y^2}$.

Note that, with (1), we really have not restricted ourselves to uniform mesh spacing in each direction. We could have different grid spacing Δx, Δy, and Δz and a different number of mesh points n_x, n_y, and n_z, in each of the space directions. Then, I_x would be the $n_x \times n_x$ identity matrix and A_x would be the corresponding stiffness matrix, as would also be the case for y and z. (Note also that we could even relax the condition of uniform spacing in *each* of the space directions.)

Quiz:

- Assume that we solve the Poisson problem on $\Omega = [0, L]^d$ with grid spacing $h = L/N$ in each of d space dimensions, $d = 2$ or 3. If the unknowns are ordered lexicographically, what is the *matrix bandwidth* of the system matrix A for the case $d = 2$?
- For $d = 3$?
- What is the number of unknowns, n for each case, $d = 2$ and 3?

Kronecker Product Matrix Properties

Kronecker products have several useful properties in the the solution of PDEs and (more recently) in machine learning, which is driving the development of specialized software and hardware for Kronecker product application and manipulation. (Unfortunately, much of this development is targeting 16-bit operations, which is not sufficient for most scientific computing applications.) There are two main properties of interest, the matrix-product rule and matrix-vector products.

For, completeness, we recall that the the Kronecker product of two matrices A and B is defined as the block matrix

$$C \;=\; A \otimes B \;:= \begin{pmatrix} a_{11}B & a_{12}B & \cdots & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & \cdots & a_{2n}B \\ \vdots & \vdots & & & \vdots \\ a_{m1}B & a_{m2}B & \cdots & \cdots & a_{mn}B \end{pmatrix}. \tag{4}$$

We note that if A and B are each $N \times N$ matrices, then C is a much larger $N^2 \times N^2$ matrix, with N^4 nonzeros in the case when A and B are full. We also note that $I \otimes B$ and $A \otimes I$ have a special structure.

$$I \otimes B \;=\; \begin{pmatrix} B & & & \\ & B & & \\ & & \ddots & \\ & & & B \end{pmatrix}, \tag{5}$$

$$A \otimes I \;=\; \begin{pmatrix} a_{11}I & a_{12}I & \cdots & a_{1n}I \\ a_{21}I & a_{22}I & \cdots & a_{2n}I \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}I & a_{m2}I & \cdots & a_{mn}I \end{pmatrix}. \tag{6}$$

Thus, $I \otimes B$ is block-diagonal, whereas $A \otimes I$ comprises diagonal blocks.

The most important property of Kronecker products for the purposes of PDEs is the matrix-product rule. Suppose we have C and F satisfying

$$C = A \otimes B, \qquad F = D \otimes E.$$

Then, straightforward application of (4) reveals that the matrix product CF is given by

$$CF = (A \otimes B)(D \otimes E) = AD \otimes BE, \tag{7}$$

under the assumption that the dimensions of (A,D) and (B,E) are such that the products AD and BE make sense.

The product rule (7) leads to several notable properties that we now describe. To simplify the exposition, we'll assume that A and B are $N \times N$ matrices unless otherwise noted.

- **Inverse.** The inverse of $C = A \otimes B$ is $C^{-1} = A^{-1} \otimes B^{-1}$. Thus, the inverse of an $N^2 \times N^2$ matrix can be found by inverting (or, more typically, factoring) two much smaller $N \times N$ matrices.
- **Eigenvalues.** Let $C = B_y \otimes B_x$ and suppose that there exists a diagonalization of B_x and B_y given, for B_x, by $S_x^{-1} B_x S_x = \Lambda_x$, with S_x the matrix of eigenvectors and Λ_x the diagonal matrix containing, with a similar form for B_y. Then

$$
\begin{aligned}
C \;=\; B_y \otimes B_x \;&=\; (S_y \Lambda_y S_y^{-1}) \otimes (S_x \Lambda_x S_x^{-1}) \qquad (8) \\
&=\; \underbrace{(S_y \otimes S_x)}_{S}\, \underbrace{(\Lambda_y \otimes \Lambda_x)}_{\Lambda}\, \underbrace{(S_y^{-1} \otimes S_x^{-1})}_{S^{-1}}. \qquad (9)
\end{aligned}
$$

  Thus, the $N^2 \times N^2$ eigenvalue problem $CS = S\Lambda$ is solved by solving two small $(N \times N)$ eigenvalue problems, which lead to $S = S_y \otimes S_x$ and $\Lambda = \Lambda_y \otimes \Lambda_x$.

- **Matrix-Vector Product.** Suppose $S = S_y \otimes S_x$ and we wish to evaluate the matrix-vector product $\underline{w} = S\underline{f}$, where $\underline{f}$ is assumed to have a natural *dual-subscript* ordering, $\{f_{ij}\}$, as in Fig. **??**. If we evaluate $\underline{w}$ by first forming S, then the storage and work rises sharply from $O(N^2)$ to $O(N^4)$. Instead, we evaluate the product in *factored form*,

$$\underline{w} \;=\; (S_y \otimes I_x)\,(I_y \otimes S_x)\underline{f}. \tag{10}$$

The first evaluation generates the vector $\underline{v} := (I_y \otimes S_x)\underline{f}$, which has entries

$$v_{ij} \;=\; \sum_{p=1}^{n_x} (S_x)_{ip} f_{pj}, \quad i \in [1,\ldots,n_x],\; j \in [1,\ldots,n_y]. \tag{11}$$

which can be seen by inspecting the figure preceding Eq. (**??**). Assuming S_x is full, the cost of computing $\underline{v}$ is $2n_x$ for each of the $n_y n_x$ entries, or $O(N^3)$.

The next step is evaluation of $\underline{w} = (S_y \otimes I_x)\underline{v}$.

$$w_{ij} \;=\; \sum_{q=1}^{n_y} (S_y)_{jq} v_{iq}, \tag{12}$$

which has cost $2n_y^2 n_x$.

Note that *significant performance gains* (up to an order-of-magnitude) can be realized by recognizing that the doubly-indexed vectors, $\underline{w}$, $\underline{v}$, and $\underline{f}$ can be viewed as corresponding $n_y \times n_x$ matrices, W, V, and F. In this case, the *matrix-vector product* evaluation (10) can be expressed in *matrix-matrix* product form:

$$W = S_x F S_y^T. \tag{13}$$

Or, in general, for any $\underline{v} = (A \otimes B)\underline{u}$, we have $V = B\,U A^T$. Matrix-matrix products are some of the most efficient operations possible in numerical computation because the require only $O(N^2)$ memory references for $O(N^3)$ operations, so the form (13) is generally very fast.

Matrix-vector products involving third-order tensors of the form $A = A_z \otimes A_y \otimes A_x$ can be evaluated with similar efficiencies. In partcular, $\underline{z} = A\underline{u}$ would be evaluated as

$$v_{ijk} = \sum_{p=1}^{n_x} (A_x)_{ip} u_{pjk} \qquad \underline{v} = (I_z \otimes I_y \otimes A_x)\underline{u} \tag{14}$$

$$w_{ijk} = \sum_{q=1}^{n_y} (A_y)_{jq} v_{iqk} \qquad \underline{w} = (I_z \otimes A_y \otimes I_x)\underline{v} \tag{15}$$

$$z_{ijk} = \sum_{r=1}^{n_z} (A_z)_{kr} w_{ijr} \qquad \underline{z} = (A_z \otimes I_y \otimes I_x)\underline{w}, \tag{16}$$

which has a total operation count of $O(N^4)$ and storage count of $O(N^3)$ for the input and output data. It is also possible, with minor effort, to recast (14)–(16) in terms of fast matrix-matrix products. (Recall, in 3D, $N^3 \approx n$.)

Fast Diagonalization Method (FDM)

- We will use the tools of the preceding section to develop fast solvers for the constant-coefficient 2D and 3D lexicographicallly-ordered cases.

- This idea originated in a 1964 paper by Birkhoff, Lynch, and Rice that predates the FFT by one year. In the present exposition we assume that A_* is the tridiagonal 1D Poisson operator of the preceding section, but the method carries through with the same complexity even if A_* is full (e.g., as might be the case for a high-order approximation to the Poisson operator).

- To start, we assume that the 1D matrices A_* have the similarity transform

$$A_* \;=\; S_* \Lambda_* S_*^{-1}, \tag{17}$$

for $*{=}x$, y, and z, where Λ_* is the diagonal matrix of eigenvalues and S_* the corresponding matrix of eigenvectors. (Always true if $A_* = A_*^T$.)

- For the 2D case, we have

$$\begin{aligned} A_{2D} &= (I_y \otimes A_x) \;+\; (A_y \otimes I_x) & (18)\\ &= (S_y S_y^{-1} \otimes S_x \Lambda_x S_x^{-1}) \;+\; (S_y \Lambda_y S_y^{-1} \otimes S_x S_x^{-1}) & (19)\\ &= (S_y \otimes S_x)\,(I_y \otimes \Lambda_x \;+\; \Lambda_y \otimes I_x)\,(S_y^{-1} \otimes S_x^{-1}), & (20) \end{aligned}$$

whose inverse is given by

$$\begin{aligned} A_{2D}^{-1} &= (S_y \otimes S_x)\, D^{-1}\, (S_y^{-1} \otimes S_x^{-1}), & (21)\\ & & (22) \end{aligned}$$

with the trivially inverted diagonal matrix

$$D \;:=\; (I_y \otimes \Lambda_x \;+\; \Lambda_y \otimes I_x). \tag{23}$$

- Aside from the preprocessing costs to find $[S_*, \Lambda_*]$, the total work is $\approx 8N^3$, assuming $N_x = N_y = N$.

- Let's look more closely at the steps of the FDM.

- Starting with known grid data, $\underline{f} = [f_{11}\; f21\; \cdots\; f_{n_x n_y}]$, we compute the solution to $A\underline{u} = \underline{f}$ as

$$\underline{u} \;=\; \underbrace{(S_y \otimes S_x)}_{S}\; \underbrace{D^{-1}}_{\Lambda^{-1}}\; \underbrace{(S_y^{-1} \otimes S_x^{-1})}_{S^{-1}}\; \underline{f}, \tag{24}$$

  where we have emphasized that $S := (S_y \otimes S_x)$ is the matrix of eigenvectors of $A := A_{2D}$ and Λ is the corresponding matrix of eigenvalues.

- Breaking the operation down further:

$$\begin{aligned} &i)\;\; \underline{\hat{f}} &=&\; (S_y^{-1} \otimes S_x^{-1})\, \underline{f} && \textit{Fourier transform of data} \\ &iii)\;\; \underline{\hat{u}} &=&\; D^{-1}\underline{\hat{f}} && \textit{divide by wavenumbers squared} \\ &iii)\;\; \underline{u} &=&\; (S_y \otimes S_x)\, \underline{\hat{u}} && \textit{combine eigenvectors to construct solution.} \end{aligned} \tag{25}$$

- The "Fourier" transform requires two tensor contractions of the form (13), which are implemented as

$$\hat{F} \;=\; S_x^{-1}\, F\, S_y^{-T}, \tag{26}$$

requiring $\sim 2(2N^3)$ floating point operations.

- Here we let capital F denote the "matrix form" of $\underline{f}$, etc.
- Division by the eigenvalues requires $\sim N^2 \ll N^3$ operations.

- Finally, the inverse transform is expressed as a second pair of tensor contractions,

$$U \;=\; S_x\, \hat{U}\, S_y, \tag{27}$$

such that the total complexity is $\sim 8N^3$.

- Here, we are neglecting the cost to find the eigenpairs as these are known in closed-form.

- For more general cases, we could call an eigenvalue solver to find (S_x, Λ_x) and (S_y, Λ_y) as part of a one-time setup.

- It is important to note that, for the uniformly-spaced grid case, the $O(N^3)$ complexity can be reduced to $O(N^2 \log N)$ by use of the FFT. This approach is used by `fishpack`, which s about 10–20× faster than multigrid for 2D problems.

- An interesting aside about the fast diagonalization method is that it allows us to easily solve systems of the form $g(A)\underline{u} = \underline{b}$.

- In 2D, the answer is

$$\underline{u} \;=\; (S_y \otimes S_x)\, G^{-1}\, (S_y^{-1} \otimes S_x^{-1})\underline{u}, \tag{28}$$

  where G is the diagonal matrix with entries $G_{ss} = f(\lambda_{x,k} + \lambda_{y,l})$, with $s = k + n_x(l-1)$ for $k = 1, \ldots, n_x$ and $l = 1, \ldots, n_y$.

- The 3D version of the FDM is similar to 2D:

$$A_{3D}^{-1} \;=\; (S_z \otimes S_y \otimes S_x)\, D^{-1}\, (S_z^{-1} \otimes S_y^{-1} \otimes S_x^{-1}), \tag{29}$$

$$\tag{30}$$

  with

$$D \;:=\; (I_z \otimes I_y \otimes \Lambda_x \,+\, I_z \otimes \Lambda_y \otimes I_x \,+\, \Lambda_z \otimes I_y \otimes I_x). \tag{31}$$

- The work is $\approx 12N^4$.

- The setup costs are nil. **(Why?)**

- Fast diagonalization is readily extended to the more general case that arises in finite element methods,

$$A_{3D} \;=\; (B_z \otimes B_y \otimes A_x) \,+\, (B_z \otimes A_y \otimes B_x) \,+\, (A_z \otimes B_y \otimes B_x), \tag{32}$$

  where B_* is the mass matrix in the "*" direction.

- The corresponding one-dimensional eigenpairs are solutions to the *generalized eigenproblem*, $A_* S_* \;=\; B_* S_* \Lambda_*$, with eigenvectors normalized to satisfy $S_*^T B_* S_* \;=\; I_*$.

- **Q:** What are the eigenvalues of A_{2D} and A_{3D} in the finite difference case on an N^d grid with $n = (N-1)^d$ unknowns?
- Recall that, for the 1D case,

$$\lambda_k \;=\; \frac{2}{h^2}\left(1 - \cos\frac{\pi k}{N}\right), \tag{33}$$

  for $k = 1, \ldots, N-1$ and $h = L/N$.
- *Hint:* Look at the entries of Λ.

Cost Analysis: Jacobi Iteration

- Recall our Jacobi iteration from Lec. 1, with $D := \mathrm{diag}(A)$.

- We can express this in two ways,

Analysis:

$$\begin{aligned} &\underline{x}_0 = 0 \\ &\textit{for } k = 1 : k_{\max} \\ &\quad \underline{x}_k \;=\; \underline{x}_{k-1} \;+\; D^{-1}(\underline{b} - A\underline{x}_{k-1}) \end{aligned}$$

Practice:

$$\begin{aligned} &\underline{x} = 0 \\ &\textit{for } k = 1 : k_{\max} \\ &\quad \underline{x} \;=\; \underline{x} \;+\; D^{-1}(\underline{b} - A\underline{x}) \end{aligned}$$

- Two main questions arise re. complexity:

  (1) How much work per iteration?

  (2) How many iterations?

- To address the first, consider more realistic loops:

$$
\begin{aligned}
&\underline{x} = 0, \;\; \underline{r} = \underline{b} \\
&\textit{for } k = 1 : k_{\max} \\
&\quad \alpha \;=\; \|\underline{r}\|_2 \;=\; \sqrt{\underline{r}^T\underline{r}} \\
&\quad \textit{if } \alpha < \textit{tol, break.} \\
&\quad \underline{x} \;=\; \underline{x} \;+\; D^{-1}\underline{r} \\
&\quad \underline{r} \;=\; \underline{b} \;-\; A\underline{x} \\
&\textit{end}
\end{aligned}
$$

- A better approach is:

$$
\begin{aligned}
&\underline{x} = 0, \;\; \underline{r} = \underline{b} \\
&\textit{for } k = 1 : k_{\max} \\
&\quad \underline{s} \;=\; D^{-1}\underline{r} \\
&\quad \alpha \;=\; \|\underline{s}\|_2 \;=\; \sqrt{\underline{s}^T\underline{s}} \\
&\quad \textit{if } \alpha < \textit{tol, break.} \\
&\quad \underline{x} \;=\; \underline{x} \;+\; \underline{s} \\
&\quad \underline{r} \;=\; \underline{r} \;-\; A\underline{s} \\
&\textit{end}
\end{aligned}
$$

- **Q:** *How many operations for each step, as a function of d?*

- Quick summary:

$$
\begin{aligned}
d = 1: &\quad 10n \cdot k \\
d = 2: &\quad 14n \cdot k \\
d = 3: &\quad 18n \cdot k
\end{aligned}
$$

- We see that the cost *per iteration* is only weakly dependent on d!

- What about the *number* of iterations?

- To analyze this question, we'll need some norms to measure the error.

- Let's start with the vector 2-norm,

$$\|\underline{x}\|_2 \;:=\; \sqrt{\underline{x}^T\underline{x}} = \left(\sum_j^n x_j^2\right)^{\frac{1}{2}}. \tag{34}$$

- With this vector norm, we have an associated *matrix norm*,

$$\|A\|_2 \;:=\; \max_{\underline{x}\in\mathbb{R}^n} \frac{\|A\underline{x}\|_2}{\|\underline{x}\|_2} \tag{35}$$

$$\;:=\; \max_{\|\underline{x}\|=1} \|A\underline{x}\|_2. \tag{36}$$

- We see that $\|A\|_2$ is identified with the *maximum stretching* (growth) of any input vector $\underline{x}$.

- For the case of $A = A^T$, we have $\|A\|_2 \;=\; \rho(A)$.

- Therefore, we know the 2-norm of A for our finite difference matrices!

- Which matrix do we need the 2-norm for to understand the error behavior of Jacobi iteration?