ECS231 Handout **Fast Solvers**

In this part, we use Poisson's equation as a model problem to show structure of problems arising in practice, and how to exploit the structure to develop fast algorithms.

Part I. Model problem.

1. Poisson's equation is a differential equation of elliptic type with broad utility in physical models that include gravitation, temperature distribution, electromagnetism, elasticity and inviscid fluid mechanics. It is also used as a physical model for data clustering using spectral methods.

2. One-dimensional model Poisson's equation takes the form

$$-\frac{d^2v(x)}{dx^2} = f(x), \qquad 0 < x < 1 \tag{1}$$

with Dirichlet boundary conditions:

$$v(0) = v(1) = 0, \tag{2}$$

where $f(x)$ is a given function and $v(x)$ is the unknown function to be computed.

3. Let us *discretize* Poisson's equation by trying to compute an approximate solution $N+2$ evenly spaced point x_i between 0 and 1:

$$0 = x_0 < x_1 < x_2 < \cdots < x_N < x_{N+1} = 1$$

and

$$x_i = x_0 + ih = ih, \qquad h = \frac{1}{N+1}.$$

The points x_0 and x_{N+1} are called boundary points and are known. x_i for $i = 1, 2, \ldots, N$ are called interior points and are unknown. h is called "mesh" size.

Denoting $v_i = v(x_i)$ and $f_i = f(x_i)$ and using the 3-point centered difference approximation, at $x = x_i$, we have

$$-\frac{d^2v(x)}{dx^2} = \frac{-v_{i-1} + 2v_i - v_{i+1}}{h^2} + \tau_i,$$

where the truncation error $\tau_i = O(h^2)$ (assuming $v(x)$ is smooth enough). Therefore at $x = x_i$, $0 < i < N+1$, we have

$$-v_{i-1} + 2v_i - v_{i+1} = h^2 f_i + h^2\tau_i$$

and $v_0 = v_{N+1} = 0$.

4. In matrix notation, let

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix}, \quad \bar{\tau} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \vdots \\ \tau_N \end{bmatrix} \quad \text{and} \quad T_N = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix},$$

then we have

$$T_N\, v = h^2 f + h^2 \bar{\tau} \tag{3}$$

To solve this equation, let us ignore $\bar{\tau}$, since it is expected to be small compared to f, then we have the linear system of equations

$$T_N\, \hat{v} = h^2 f, \tag{4}$$

where $\hat{v}$ is an approximation of v.

5. The tridiagonal matrix $T_N = \text{tridiag}(-1, 2, -1)$ has the following explicit eigenvalue decomposition

$$T_N = Z_N \Lambda_N Z_N^T,$$

where $Z_N = [z_1, z_2, \ldots, z_N]$ and $\Lambda_N = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N)$,

- The eigenvalues of T_N are

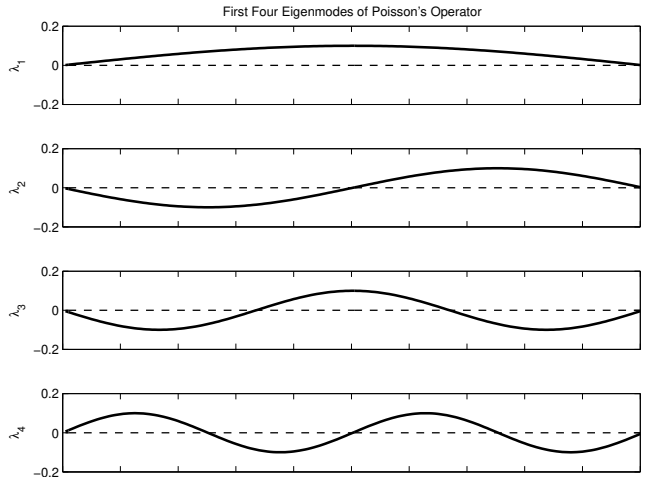
$$\lambda_j = 2(1 - \cos\frac{\pi j}{N+1}) \quad \text{for} \quad j = 1, 2, \ldots, N$$

- Since $\lambda_j > 0$ for all j, T_N is symmetric positive definite.
- z_j are the eigenvectors for $j = 1, 2, \ldots, N$. The kth entry of z_j is given by

$$z_j(k) = \sqrt{\frac{2}{N+1}} \sin(\frac{\pi k j}{N+1}) \quad \text{for} \quad k = 1, 2, \ldots, N.$$

- Z is orthogonal.

The following plot shows the eigenvectors corresponding to the first four eigenvaluesgskip



The smallest eigenvalue λ_1 is

$$\lambda_1 = 2(1 - \cos\frac{\pi}{N+1}) \approx 2\left(1 - \left(1 - \frac{\pi^2}{2(N+1)^2}\right)\right) = \frac{\pi^2}{(N+1)^2} \quad \text{for large } N,$$

and the largest eigenvalue of T_N λ_N is

$$\lambda_N = 2(1 - \cos\frac{N\pi}{N+1}) \approx 4 \quad \text{for large } N,$$

Therefore, the condition number of T_N is

$$\text{cond}(T_N) = \|T_N\|_2 \|T_N^{-1}\|_2 = \frac{\lambda_N}{\lambda_1} \approx \frac{4(N+1)^2}{\pi^2} = \mathcal{O}(h^{-2}) \quad \text{for large } N.$$

6. Let us bound the error $v - \hat{v}$. Subtracting equation (4) from equation (3), we have

$$v - \hat{v} = h^2 T_N^{-1}\ \bar{\tau}.$$

By taking norm and assuming that v is sufficient smooth (the required derivatives are bounded), we have

$$\|v - \hat{v}\|_2 \leq h^2 \|T_N^{-1}\|_2 \|\bar{\tau}\|_2 \approx h^2 \frac{(N+1)^2}{\pi^2} \|\bar{\tau}\|_2 = O(\|\bar{\tau}\|_2) = O(h^2).$$

This indicates that when we solve algebraic system (4), it is not necessary to impose the accuracy more than $O(h^2)$.

In the rest of this note, we will not distinguish between v and its approximation $\hat{v}$ and so will simplify notation by letting

$$T_N\ v = h^2 f.$$

7. Two-dimensional model Poisson's equation takes the form

$$\begin{aligned} -\nabla^2 v(x,y) &= f(x,y) \quad \text{for } (x,y) \in \Omega, \\ v(x,y) &= \phi(x,y) \quad \text{for } (x,y) \in \partial\Omega, \end{aligned}$$

where ∇^2 is the Laplace operator (sometimes, also denoted as Δ): $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$, the domain Ω is the unit sqaure $\Omega = (0,1) \times (0,1)$, the unit square and $\partial\Omega$ is its boundary. f and ϕ are given functions.

Example. consider Poisson's equation

$$-\Delta v(x,y) = f(x,y), \quad (x,y) \in (0,1) \times (0,1)$$

with the boundary condition

$$v(x,y) = \begin{cases} 0 & x = 0 \\ 0 & x = 1 \\ \sin(2\pi x) & y = 0 \\ \sin(2\pi x) & y = 1 \end{cases}$$

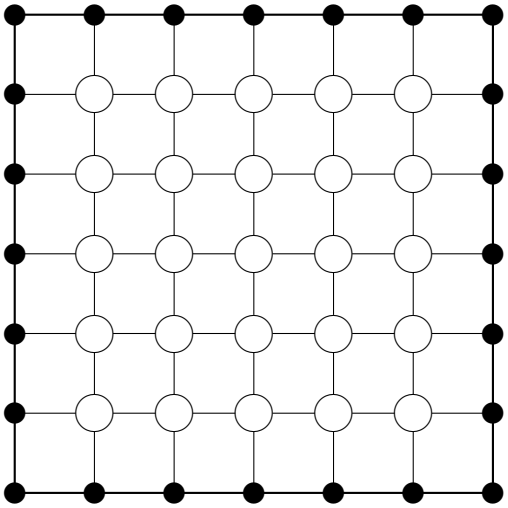
If the right-hand-side function is

$$f(x,y) = 4\pi \sin(2\pi x)(\pi \cos(2\pi y^2)(1 + 4y^2) + \sin(2\pi y^2)),$$

then the analytical solution $v(x,y)$ is

$$v(x,y) = \sin(2\pi x) \cos(2\pi y^2).$$

8. To discretize the differential equation, the domain Ω is covered with a grid of mesh size $h = 1/(N+1)$ as follows.



This is an example grid with $N = 5$. The values $v(x, y)$ at the boundary grid points ● is given by $\phi(x, y)$, and the values $v(x, y)$ at interior grid points ◯ are to be sought.

Each grid point (x_i, y_j) have the representation

$$x_i = ih \quad \text{and} \quad y_j = jh \quad \text{for } i,\, j = 0, 1, \ldots, N+1.$$

Those points with one of i and j being $i = 0$ or $N + 1$ are the *boundary grid points*; all other points are the *interior grid points*. We seek approximations to $v(x_i, y_j)$ for all the interior grid points. Write

$$v_{ij} = v(x_i, y_j), \quad f_{ij} = f(x_i, y_j), \quad \text{and} \qquad \phi_{ij} = \phi(x_i, y_j).$$

To this end, we do approximately at each interior grid point:

$$\begin{aligned} -\frac{\partial^2 v}{\partial x^2}\bigg|_{\text{at } (x_i, y_j)} &\approx \frac{-v_{i-1\,j} + 2v_{ij} - v_{i+1\,j}}{h^2}, \\ -\frac{\partial^2 v}{\partial y^2}\bigg|_{\text{at } (x_i, y_j)} &\approx \frac{-v_{i\,j-1} + 2v_{ij} - v_{i\,j+1}}{h^2}. \end{aligned}$$

Adding these approximations we have

$$-\frac{\partial^2 v}{\partial x^2} - \frac{\partial^2 v}{\partial y^2}\bigg|_{\text{at } (x_i, y_j)} = \frac{-v_{i-1\,j} - v_{i\,j-1} + 4v_{ij} - v_{i+1\,j} - v_{i\,j+1}}{h^2} + \tau_{ij}$$

where τ_{ij} is a truncation error. By Taylor expansion, it is easy to show that it is at the order of h^2, $O(h^2)$. Ignoring the truncation errors, we arrive at the linear equations in the unknowns v_{ij},

$$-v_{i-1\,j} - v_{i\,j-1} + 4v_{ij} - v_{i+1\,j} - v_{i\,j+1} = h^2 f_{ij}, \tag{5}$$

for $1 \le i, j \le N$. The left-hand side of which is 4 times the v at the point subtracting the v at the four neighbor points. This is called *5-point centered difference* or *5-point stencil*.

Notice that the boundary points

$$v_{0j} = \phi_{0j}, \quad v_{0\,N+1} = \phi_{0\,N+1}, \quad v_{i0} = \phi_{i0}, \quad v_{i\,N+1} = \phi_{i\,N+1}$$

are known and the unknowns are for $0 < i, j < N + 1$; so there are N^2 of them. By collecting all v_{ij} to form an $N \times N$ matrix V whose (i, j)th entry is v_{ij}:

$$V = (v_{ij})$$

and define an $N \times N$ matrix $\widetilde{F}$ by

$$h^2(\widetilde{F})_{ij} = \begin{cases} h^2 f_{ij}, & \text{for } 2 \le i, j \le N-1, \\ h^2 f_{ij} + \phi_{i\,j-1}, & \text{for } 2 \le i \le N-1 \text{ and } j = 1, \\ h^2 f_{ij} + \phi_{i\,j+1}, & \text{for } 2 \le i \le N-1 \text{ and } j = N, \\ h^2 f_{ij} + \phi_{i-1\,j}, & \text{for } i = 1 \text{ and } 2 \le j \le N-1, \\ h^2 f_{ij} + \phi_{i+1\,j}, & \text{for } i = N \text{ and } 2 \le j \le N-1, \\ h^2 f_{ij} + \phi_{i\,j-1} + \phi_{i-1\,j}, & \text{for } (i,j) = (1,1), \\ h^2 f_{ij} + \phi_{i\,j-1} + \phi_{i+1\,j}, & \text{for } (i,j) = (N,1), \\ h^2 f_{ij} + \phi_{i-1\,j} + \phi_{i\,j+1}, & \text{for } (i,j) = (1,N), \\ h^2 f_{ij} + \phi_{i\,j+1} + \phi_{i+1\,j}, & \text{for } (i,j) = (N,N). \end{cases}$$

then it can be verified that the equation (5) becomes

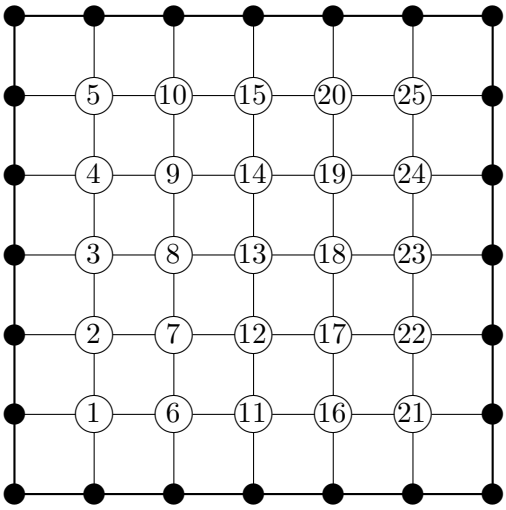
$$T_N \cdot V + V \cdot T_N = h^2 \widetilde{F}, \tag{6}$$

where $T_N = \text{tridiag}(-1, 2, -1)$. Note that care should be taken for the grid points that are neighbors of boundary grid points.

9. Lexicographic (natural) ordering: the system (6) is not in the familiar form "$Ax = b$" of linear system of equations because all the unknowns are compactly stored into a matrix. To reorganize equations (5) in a way that leads to the $Ax = b$ form, we need to arrange v_{ij} into a column vector. A natural way would be arranging one column of V on top of another, i.e., defining a N^2-dimensional vector v as (in MATLAB-like notation)

$$v = [V(:,1); V(:,2); \ldots; V(:,N)] \equiv \text{vec}(V).$$

Such an ordering of v_{ij} is best described by the following picture in the case of $N = 5$.



Define also N^2-dimensional vector $\widetilde{f}$ from the matrix $\widetilde{F}$ analogously. The system (6) becomes

$$Av = h^2 \widetilde{f}, \tag{7}$$

where

$$A = \begin{pmatrix} T_N + 2I_N & -I_N & & & & \\ -I_N & T_N + 2I_N & -I_N & & & \\ & \ddots & \ddots & \ddots & & \\ & & -I_N & T_N + 2I_N & -I_N & \\ & & & -I_N & T_N + 2I_N & \end{pmatrix}.$$

In fact, using the notion of the Kronecker product $\otimes$, the matrix A be written as

$$A = I_N \otimes T_N + T_N \otimes I_N \equiv T_{N\times N}.$$

10. Kronecker product

    (a) Let $A = (a_{ij})$ be $m \times n$ and $B = (b_{ij})$ be $p \times q$, then the *Kronecker product* of A and B are defined as

    $$A \otimes B = (a_{ij}B) = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix}.$$

    Note that $A \otimes B$ is a $(mp) \times (nq)$ matrix.

    (b) Kronecker product has the following basic properties:

    - $(A \otimes B)^T = A^T \otimes B^T$
    - If A and B are invertible, $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.
    - Assume AC and BD are well defined, then $(A \otimes B) \cdot (C \otimes D) = (AC) \otimes (BD)$

    (c) Let $\mathrm{vec}(X)$ be defined to be a column vector of length $m \cdot n$ made of the columns of an $m \times n$ matrix X stacked atop one another from left to right, i.e.,

    $$\mathrm{vec}(X) = \mathrm{vec}([x_1, x_2, \ldots, x_n]) = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix},$$

    then we have

    - $\mathrm{vec}(AX) = (I_n \otimes A) \cdot \mathrm{vec}(X)$
    - $\mathrm{vec}(XB) = (B^T \otimes I_m) \cdot \mathrm{vec}(X)$

11. Using the Kronecker product and the eigenvalue decomposition of the tridiagonal matrix T_N, we immediately derive the eigenvalue decomposition of the matrix $T_{N\times N}$:

    Let $T_N = Z_N \Lambda_N Z_N^T$ be the eigendecomposition of the tridiagonal matrix T_N. Then the eigendecomposition of $T_{N\times N}$ is given by

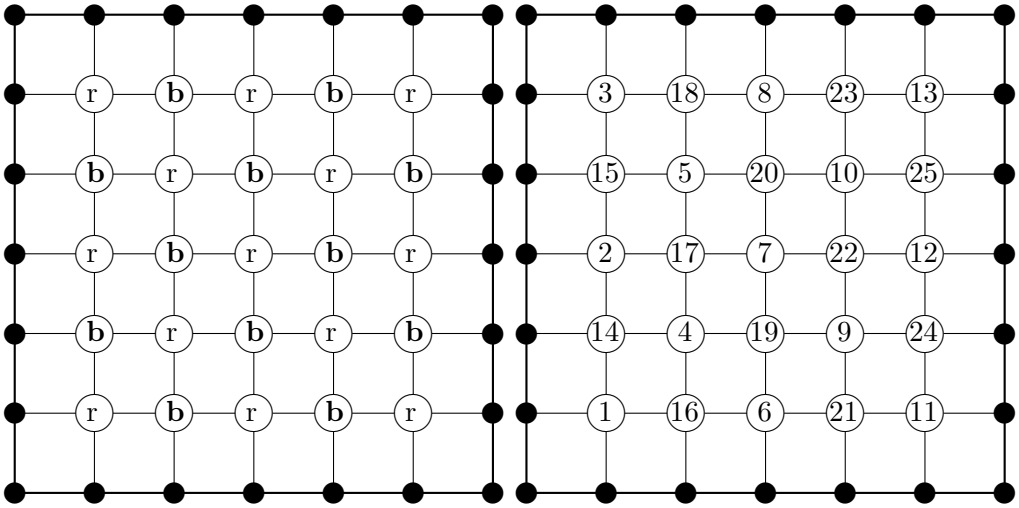
    $$\begin{aligned} T_{N\times N} &= I_N \otimes T_N + T_N \otimes I_N \\ &= (Z_N \otimes Z_N)(I_N \otimes \Lambda_N + \Lambda_N \otimes I_N)(Z_N \otimes Z_N)^T. \end{aligned}$$

    By the eigenvalue decomposition of $T_{N\times N}$, we know that eigenvalues λ_{ij} of the Poisson matrix $T_{N\times N}$ are given by

    $$\lambda_{i\,j} \stackrel{\text{def}}{=} \lambda_i + \lambda_j = 2(2 - \cos i\pi h - \cos j\pi h) \tag{8}$$

    $i, j = 1, 2, \ldots, N$, where λ_i and λ_j are the eigenvalues of T_N. Note that $h = 1/(N+1)$.

12. Red-black ordering: first color all nodes by either *red* or *black* in such a way that no neighbor nodes share the same color; and then enumerate all nodes with one color and then all nodes with the other. Such an ordering of v_{ij} is best described by the following picture in the case of $N = 5$.



Let $v_{\rm rb}$ and $\widetilde{f}_{\rm rb}$ be the N^2-dimensional vectors obtained from V and $\widetilde{F}$ with this red-black ordering. The system (6) becomes

$$A_{\rm rb} v_{\rm rb} = h^2 \widetilde{f}_{\rm rb}, \quad A_{\rm rb} = \begin{pmatrix} D_r & B \\ B^T & D_b \end{pmatrix}, \tag{9}$$

both D_r and D_b are diagonal matrices with all diagonal entries being 4. B is a sparse matrix with nonzero entries -1 (the details of the structure of B is not important for us now).

Notice that $A_{\rm rb}$ is consistently ordered and has eigenvalues given by (8).

13. Three-dimensional model Poisson's equation takes the form

$$\begin{aligned} -\nabla^2 v(x,y,z) &= f(x,y,z) \quad \text{for } (x,y,z) \in \Omega, \\ v(x,y,z) &= \phi(x,y,z) \quad \text{for } (x,y,z) \in \partial\Omega, \end{aligned}$$

where the Laplace operator $\nabla^2 \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$, the domain Ω is the unit cubic $\Omega = (0,1) \times (0,1) \times (0,1)$, and is the boundary of Ω, f and ϕ are given functions.

14. Using a 7-point centered finite difference on a cubic grid of mesh size $h = 1/(N+1)$, with natural ordering, it leads to the linear system of equations $Av = b$, where the coefficient matrix

$$A = T_{N\times N\times N} = T_N \otimes I_N \otimes I_N + I_N \otimes T_N \otimes I_N + I_N \otimes I_N \otimes T_N$$

It can be shown that A's eigenvalues are all possible triple sum of the eigenvalues of T_N and the eigenvector matrix is $Z_N \otimes Z_N \otimes Z_N$.

Part II. Block cyclic reduction

1. Block cyclic reduction (BCR) is a fast method for the Poisson model problem. Recall 2-D Poisson's model problem is given by

$$\underbrace{(I_N \otimes T_N + T_N \otimes I_N)}_{T_{N\times N}} \mathrm{vec}(V) = \mathrm{vec}(h^2 F).$$

Write it as the standard form of the linear system of equations, we have

$$\begin{bmatrix} A^{(0)} & -I & & \\ -I & A^{(0)} & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & A^{(0)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix},$$

where $A^{(0)} = T_N + 2I$ and I is an $N \times N$ identity matrix. x_i and b_i are N-vectors.

For simplicity we assume that N is **odd**. We use block Gaussian elimination to combine three consecutive sets of equations

$$\begin{array}{rl} & [\ -x_{j-2} \quad +A^{(0)}x_{j-1} \quad -x_j \qquad\qquad\qquad\qquad = b_{j-1}\ ] \\ +A & [\qquad\qquad -x_{j-1} \quad +A^{(0)}x_j \quad -x_{j+1} \qquad\qquad = b_j\ ] \\ + & [\qquad\qquad\qquad\qquad -x_j \quad +A^{(0)}x_{j+1} \quad -x_{j+2} \quad = b_{j+1}\ ] \end{array}$$

Thus eliminating x_{j-1} and x_{j+1}

$$-x_{j-2} + ((A^{(0)})^2 - 2I)x_j - x_{j+2} = b_{j-1} + A^{(0)}b_j + b_{j+1}.$$

Doing this for every set of three consecutive equations yields two sets of equations:

- one for the x_j with j even

$$\begin{bmatrix} A^{(1)} & -I & & \\ -I & A^{(1)} & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & A^{(1)} \end{bmatrix} \begin{bmatrix} x_2 \\ x_4 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} b_1 + Ab_2 + b_3 \\ b_3 + Ab_4 + b_5 \\ \vdots \\ b_{N-2} + Ab_{N-1} + b_N \end{bmatrix}, \tag{10}$$

where

$$A^{(1)} = \left(A^{(0)}\right)^2 - 2I.$$

- one set of equations for the x_j with j odd,

$$\begin{bmatrix} A & & & \\ & A & & \\ & & \ddots & \\ & & & A \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 + x_2 \\ b_3 + x_2 + x_4 \\ \vdots \\ b_N + x_{N-1} \end{bmatrix}. \tag{11}$$

This set of equations can be solved directly after solving the equation (10) for x_j with j even.

Note that equation (10) has the same form as the original problem, so we may repeat this process recursively. For example, at the next step we get

$$\begin{bmatrix} A^{(2)} & -I & & \\ -I & A^{(2)} & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & A^{(2)} \end{bmatrix} \begin{bmatrix} x_4 \\ x_8 \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \\ \vdots \end{bmatrix}, \tag{12}$$

where

$$A^{(2)} = \left(A^{(1)}\right)^2 - 2I,$$

and

$$\begin{bmatrix} A^{(1)} & & & \\ & A^{(1)} & & \\ & & \ddots & \\ & & & A^{(1)} \end{bmatrix} \begin{bmatrix} x_2 \\ x_6 \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \\ \vdots \end{bmatrix}. \tag{13}$$

We repeat this until only one equation is left, which we solve another way.

2. In summary, assume that $N = 2^{k+1} - 1$, the BCR algorithm consists of the following three steps:

   (a) Block reduction (see equations (10) and (12))
   (b) Solve $A^{(k)}x^{(k)} = b^{(k)}$
   (c) Back solve (see equations (11) and (13))

   Complexlity: $O(N^2 \log_2 N)$

3. The simple BCR approach has two drawbacks:

   (a) It is numerically unstable because $A^{(r)}$ grows quickly:

   $$\|A^{(r)}\| \sim \|A^{(r-1)}\|^2 \approx 4^{2^r},$$

   so in computing $b_j^{(r+1)}$, the $b_{2j\pm1}^{(r)}$ are lost in roundoff.

   (b) $A^{(r)}$ has bandwidth $2^r + 1$ if $A^{(0)} = A$ is tridiagonal, so it can be dense and thus more expensive to multiply or solve.

4. Here we described a simple but numerically unstable version of the BCR algorithm. A stable implementation are described in [B. Buzbee, G. Golub and C. Nielson , On the direct method for solving Poisson's equation, SIAM J. Numer. Anal. Vol. 7, pp.627–656, 1970.]

   Fastest algorithms on vector and parallel computers are often a hybrid of block cyclic reduction and FFT. BCR has also be extendned to solve many other types of structured matrix computation problems. A survey of the cyclic reduction can be found in [W. Gander and G.H. Golub, Cyclic Reduction - History and Applications, Proceedings of the Workshop on Scientific Computing: 10-12 March, 1997, edited by F. T. Luk, R. Plemmons. Springer Verlag, New York, 1997. Also appeared as Technical Report SCCM 97-02, Stanford University, 1997.]

Part III. FFT (Fast Fourier Transform) method

1. Let us learn how to solve the 2D Poisson's model problem using the matrix-matrix multiplications involving the eigenvector matrix of T_N. A straightforward implementation of the matrix-matrix would cost $O(N^3)$. We will show how this multiplication can be implemented using the fast Fourier transform (FFT) in only $O(N^2 \log_2 N)$ operation. Note that if $N = 2^{20} = 1,048,576$, then $\log_2 N = 20$.

2. Recall that in the matrix equation form, 2D Poisson's equation is

$$T_N \cdot V + V \cdot T_N = h^2 F.$$

Let $T_N = Z \Lambda Z^T$ be the eigenvalue decomposition of T_N. Then the previous equation becomes

$$\Lambda \cdot \widetilde{V} + \widetilde{V} \cdot \Lambda = h^2 \widetilde{F}.$$

where $\widetilde{V} = Z^T V Z$ and $\widetilde{F} = Z^T F Z$. It is easy to see that the (j,k) entry of this equation is

$$\lambda_j \tilde{v}_{jk} + \tilde{v}_{jk} \lambda_k = h^2 \tilde{f}_{jk},$$

which can be solved for $\tilde{v}_{jk}$ to get

$$\tilde{v}_{jk} = \frac{h^2 \tilde{f}_{jk}}{\lambda_j + \lambda_k}.$$

This yields the first version of the algorithm to solve the Poisson's equation via matrix-matrix multiplications:

   (a) Compute $\widetilde{F} = Z^T F Z$

   (b) For all j and k, compute $\tilde{v}_{jk} = \frac{h^2 \tilde{f}_{jk}}{\lambda_j + \lambda_k}$

   (c) Compute $V = Z \widetilde{V} Z^T$

   The cost of steps (a) and (b) is four matrix-matrix multiplications by Z and $Z^T (= Z)$, which is $8N^3$ using a conventional algorithm. The cost of step (b) is $3N^2$. Therefore the total cost is about $8N^3$.

   In the following, we show how multiplication by Z is essentially the same as computing a *discrete Fourier transform*, which can be done in $O(N^2 \log_2 N)$ operation.

3. Using the Kronecker product, we have

$$\begin{aligned} v = \mathrm{vec}(V) &= (T_{N\times N})^{-1} \cdot \mathrm{vec}(h^2 F) \\ &= \left((Z_N \otimes Z_N)(I_N \otimes \Lambda_N + \Lambda_N \otimes I_N)(Z_N \otimes Z_N)^T\right)^{-1} \cdot \mathrm{vec}(h^2 F) \\ &= (Z_N \otimes Z_N)\left(I_N \otimes \Lambda_N + \Lambda_N \otimes I_N\right)^{-1} (Z_N^T \otimes Z_N^T) \cdot \mathrm{vec}(h^2 F) \end{aligned}$$

   It is easy to see that doing the indicated matrix-vector multiplications from right to left is mathematically the same as the algorithm described in Item 1. This also shows how to extend the algorithm to Poisson's equation in higher dimension.

4. The Discrete Fourier Transform (DFT) of an N-vector a is the vector

$$b = \Phi a,$$

where $\Phi = (\phi_{jk})$ is N-by-N matrix defined as follows:

$$\phi_{jk} = \omega^{j \times k}, \quad \text{for} \quad j, k = 0, 1, \ldots, N-1$$

where ω is the principal Nth root of unity equation $\omega^N = 1$, i.e.,

$$\omega = \exp\left(\frac{-2\pi \mathtt{i}}{N}\right) = \cos\frac{2\pi}{N} - \mathtt{i}\sin\frac{2\pi}{N},$$

and $\mathtt{i} = \sqrt{-1}$.

The Inverse Discrete Fourier Transform (IDFT) of b is the vector

$$a = \Phi^{-1} b.$$

By the definitions, we see that both the DFT and IDFT are just matrix-vector multiplications and they can be straightforwardly implemented in $2N^2$ operations.

5. Properties of DFT:

   (a) $\frac{1}{\sqrt{N}}\Phi$ is a complex symmetric and unitary matrix, i.e.,

   $$\Phi^{-1} = \frac{1}{N}\Phi^H = \frac{1}{N}\bar{\Phi}.$$

   (Exercise: verify that $\Phi^H = (\bar{\Phi})^T = \bar{\Phi}$ and $\frac{1}{N}\Phi \cdot \Phi^H = I$.)

   (b) Let $a = [a_0, a_1, \ldots, a_{N-1}]$, then the kth component of the DFT $b = \Phi a$ is

   $$b_k = \sum_{j=0}^{N-1} a_j \omega^{kj}.$$

   Therefore, b_k can be viewed as the value of the polynomial $p_a(x) = \sum_{j=0}^{N-1} a_j x^j$ at $x = \omega^k$:

   $$b_k = (\Phi a)_k = p_a(\omega^k).$$

   In other words,

   > *the DFT is polynomial evaluation at the points* $\omega^0, \omega^1, \ldots, \omega^{N-1}$.

   Conversely,

   > the IDFT is polynomial interpolation, producing the coefficients of a polynomial given its values at $\omega^0, \omega^1, \ldots, \omega^{N-1}$.

   (c) If $a = [a_0, \ldots, a_{N-1}, 0, \ldots, 0]^T$ and $b = [b_0, \ldots, b_{N-1}, 0, \ldots, 0]^T$ are $2N$-vectors, then the *discrete convolution* of a and b, denoted as $a * b$, is defined as

   $$a * b \equiv c = [c_0, c_1, \ldots, c_{2N-1}]^T,$$

   where $c_k = \sum_{j=0}^{k} a_j b_{k-j}$.

To illustrate the use of the discrete convolution, consider the polynomial multiplication. Let $p_a(x) = \sum_{k=0}^{N-1} a_k x^k$ and $p_b(x) = \sum_{k=0}^{N-1} b_k x^k$ be degree-$(N-1)$ polynomials. Then their product

$$p_a(x) \cdot p_b(x) = \sum_{k=0}^{2N-1} c_k x^k \equiv p_c(x),$$

where the coefficients c_k are given by the discrete convolution.

One purpose of the Fourier transform is to convert the convolution into multiplication.

Theorem 1. *Let $a = [a_0, \ldots, a_{N-1}, 0, \ldots, 0]^T$ and $b = [b_0, \ldots, b_{N-1}, 0, \ldots, 0]^T$ be vectors of dimension $2N$, and let $c = a * b = [c_0, \ldots, c_{2N-1}]^T$. Then*

$$(\Phi c)_k = (\Phi a)_k \cdot (\Phi b)_k.$$

PROOF. Recall the property (b), if $\widetilde{a} = \Phi a$, then the kth entries of $\widetilde{a}$ is $\widetilde{a}_k = \sum_{j=0}^{2N-1} a_j \omega^{kj}$, the value of the polynomial $p_a(x) = \sum_{j=0}^{N-1} a_j x^j$ at $x = \omega^k$, i.e.,

$$\widetilde{a}_k = p_a(\omega^k).$$

Similarly, $\widetilde{b} = \Phi b$ means that $\widetilde{b}_k = \sum_{j=0}^{N-1} b_j \omega^{kj} = p_b(\omega^k)$, and $\widetilde{c} = \Phi c$ means that $\widetilde{c}_k = \sum_{j=0}^{2N-1} c_j \omega^{kj} = p_c(\omega^k)$. Therefore

$$(\Phi a)_k \cdot (\Phi b)_k = \widetilde{a}_k \cdot \widetilde{b}_k = p_a(\omega^k) \cdot p_b(\omega^k) = p_c(\omega^k) = \widetilde{c}_k = (\Phi c)_k.$$

6. Fast Fourier Transform (FFT) is a fast way to multiply a given vector a by the Fourier matrix Φ. Instead of $2N^2$, it will require only about $\frac{3N}{2} \cdot \log_2 N$ operations.

   We now derive the FFT via its interpretation as polynomial evaluation. Recall that the goal is to evaluate $p_a(x) = \sum_{k=0}^{N-1} a_k x^k$ at $x = \omega^j$ for $0 \le j \le N-1$. For simplicity we will assume $N = 2^m$. The FFT algorithm is based on the following two critical observations:

   (a) By writing

$$\begin{aligned} p_a(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1} \\ &= (a_0 + a_2 x^2 + a_4 x^4 + \cdots) + (a_1 x + a_3 x^3 + a_5 x^5 + \cdots) \\ &= (a_0 + a_2 x^2 + a_4 x^4 + \cdots) + x(a_1 + a_3 x^2 + a_5 x^4 + \cdots) \\ &= p_{a_{\text{even}}}(x^2) + x p_{a_{\text{odd}}}(x^2), \end{aligned}$$

   we see that the evaluation of $p_a(x)$ is *divided* into evaluating two polynomials $p_{a_{\text{even}}}$ and $p_{a_{\text{odd}}}$ of degree $\frac{N}{2} - 1$ at $(\omega^j)^2$, $0 \le j \le N-1$.

   (b) Since $\omega^N = 1$,

$$\omega^{2j} = \omega^{2(j+\frac{N}{2})}.$$

   Therefore, there are really just $\frac{N}{2}$ points ω^{2j} for $j = 0, 1, \ldots, \frac{N}{2} - 1$.

   By these two observations, we see that evaluating a polynomial of degree $N - 1 = 2^m - 1$ at all N points ω^j $(0 \le j \le N-1)$ is the same as evaluating two polynomials of degree $\frac{N}{2} - 1$ at all $\frac{N}{2}$ points,[1], and then combining the results with N multiplications and additions. This can be done *recursively* as shown by the following pseudo-code:

[1] those are the $\frac{N}{2}$th roots of the unity.

function $\widetilde{a} = \mathrm{FFT}(a, N)$
    if $N = 1$
        return $\widetilde{a} = a$
    else
        $\widetilde{a}_{\mathrm{even}} = \mathrm{FFT}(a_{\mathrm{even}}, N/2)$
        $\widetilde{a}_{\mathrm{odd}} = \mathrm{FFT}(a_{\mathrm{odd}}, N/2)$
        $\omega = e^{-2\pi i/N}$
        $z = [\omega^0, \omega^1, \ldots, \omega^{N/2-1}]$
        return $\widetilde{a} = [\widetilde{a}_{\mathrm{even}} + z. * \widetilde{a}_{\mathrm{odd}},\ \widetilde{a}_{\mathrm{even}} - z. * \widetilde{a}_{\mathrm{odd}}]$
    end if

where $.*$ means componentwise multiplication of arrays (as in MATLAB), and have used the fact that $\omega^{j+N/2} = -\omega^j$.

7. Matlab script of textbook FFT.

```
function y = ffttx(x)
%FFTTX  Textbook Fast Finite Fourier Transform.
%    FFTTX(X) computes the same finite Fourier transform as FFT(X).
%    The code uses a recursive divide and conquer algorithm for
%    even order and straight matrix-vector multiplication otherwise.
%    If length(X) is m*p where m is odd and p is a power of 2, the
%    computational complexity of this approach is O(m^2)*O(p*log2(p)).
% by Cleve Moler

x = x(:);
n = length(x);
omega = exp(-2*pi*i/n);

if rem(n,2) == 0
   % Recursive divide and conquer
   u = ffttx(x(1:2:n-1));
   v = ffttx(x(2:2:n));
   k = (0:n/2-1)';
   w = omega .^ k;
   y = [u+w.*v; u-w.*v];
else
   % The Fourier matrix.
   j = 0:n-1;
   k = j';
   F = omega .^ (k*j);
   y = F*x;
end
```

8. Let the cost of this algorithm be denoted $C(N)$. Then we see that

$$C(N) = 2 \cdot C\left(\frac{N}{2}\right) + \frac{3N}{2},$$

assuming that the powers if ω are precomputed and stored in tables. This recurrence can be solved as the following:

$$\begin{aligned} C(N) &= 2 \cdot C\left(\frac{N}{2}\right) + \frac{3N}{2} \\ &= 2^2 \cdot C\left(\frac{N}{4}\right) + 2 \cdot \frac{2N}{2} \\ &= 2^3 \cdot C\left(\frac{N}{8}\right) + 3 \cdot \frac{2N}{2} \\ &= \cdots \\ &= (\log_2 N) \cdot \frac{3N}{2}. \end{aligned}$$

Note that $C(1) = 0$.

In conclusion, to compute the FFTs of columns (or rows) of an N-by-N matrix the total costs $N \cdot (\log_2 N) \cdot \frac{3N}{2} = \frac{3}{2} N^2 \log_2 N$, which is the complexity of the FFT method for solving the 2D Poisson's model problem.

9. We have seen that to solve the discrete Poisson's model problem by the eigenvalue decomposition of T_N requires the ability to multiply by the N-by-N matrix Z, whose the (j, k) entry is

$$z_{jk} = \sqrt{\frac{2}{N+1}} \sin\left(\frac{\pi(k+1)(j+1)}{N+1}\right),$$

where for the convenient of notation, we number rows and columns from 0 to $N-1$ starting now.

Now consider the $(2N+2)$-by-$(2N+2)$ DFT matrix Φ, whose j, k entry is

$$\exp\left(\frac{-2\pi \mathtt{i} jk}{2N+2}\right) = \exp\left(\frac{-\pi \mathtt{i} jk}{N+1}\right) = \cos\frac{\pi jk}{N+1} - \mathtt{i} \sin\frac{\pi jk}{N+1}.$$

Thus the N-by-N matrix Z consists of $-\sqrt{\frac{2}{N+1}}$ times the imaginary part of the second through $(N+1)$st rows and columns of Φ. So if we can multiply efficiently by Φ using the FFT, then we can multiply efficiently by Z. In practice, we can modify the FFT to multiply by Z directly. This is called the *Fast Sine Transform (FST)*.

10. Classical references:

- C. Van Loan, Computational Framework for the Fast Fourier Transform, SIAM Press, 1992
- A. Edelman, P. McCorquodale, and S. Toledo. The future fast fourier transform? SIAM Journal on Scientific Computing, Vol.20, pp.1094-1114, 1999.