Poisson Solvers

William McLean

April 21, 2004

Return to Math3301/Math5315 Common Material.

1 Introduction

Many problems in applied mathematics lead to a partial differential equation of the form

$$-a\nabla^2 u + \boldsymbol{b}\cdot\nabla u + cu = f \quad \text{in } \Omega. \tag{1}$$

Here, Ω is an open subset of $\mathbb{R}^d$ for $d = 1$, 2 or 3, the *coefficients* a, $\boldsymbol{b}$ and c together with the *source term* f are given functions on Ω and we want to determine the unknown function $u : \Omega \to \mathbb{R}$. The definition of the vector differentiation operator ∇ means that, on the left hand side,

$$\boldsymbol{b}\cdot\nabla u = \sum_{i=1}^{d} b_i \frac{\partial u}{\partial x_i}$$

and $\nabla^2 u$, the *Laplacian* of u, is given by

$$\nabla^2 u = \nabla\cdot(\nabla u) = \operatorname{div}\operatorname{grad} u = \sum_{i=1}^{d} \frac{\partial^2 u}{\partial x_i^2}.$$

In addition to satisfying (1) the solution must obey some *boundary conditions* on $\partial\Omega$, the boundary of Ω. The simplest is a *Dirichlet* boundary condition:

$$u = g \quad \text{on } \partial\Omega, \tag{2}$$

for a given function g. A *Neumann* boundary condition takes the form

$$\frac{\partial u}{\partial n} = g \quad \text{on } \partial\Omega, \tag{3}$$

where $\boldsymbol{n}$ is the outward unit normal to Ω.

We will assume that (1) is *uniformly elliptic*. This means that there exist positive constants $a_{\min}$ and $a_{\max}$ such that the coefficient a satisfies

$$a_{\min} \leq a(x) \leq a_{\max} \quad \text{for all } x \in \Omega.$$

The simplest example is *Poisson's equation*, which arises when a is a positive constant, $\boldsymbol{b} = \mathbf{0}$ and $c = 0$:

$$-a\nabla^2 u = f \quad \text{in } \Omega. \tag{4}$$

An elliptic PDE like (1) together with suitable boundary conditions like (2) or (3) constitutes an *elliptic boundary value problem*. The main types of numerical methods for solving such problems are as follows.

Finite difference methods are the simplest to describe and the easiest to implement provided the domain Ω has a reasonably simple shape;

Finite element methods are capable of handling very general domains;

Spectral methods can achieve very high accuracy but are practical only for very simple domains.

We will discuss only the first of these methods.

2 Central Difference Approximations in 1D

In the 1-dimensional case ($d = 1$), the set Ω is just an open interval and the PDE (1) reduces to an ODE:

$$-au'' + bu' + cu = f \quad \text{in } \Omega. \tag{5}$$

We assume $\Omega = (0, L)$ and then divide this interval into N subintervals, each of length $h = L/N$, by defining grid points

$$x_i = ih \quad \text{for } i = 0,\ 1,\ 2,\ \ldots,\ N.$$

The next Lemma shows that the finite-difference approximations

$$\begin{aligned} u'(x) &\approx \frac{u(x+h) - u(x-h)}{2h}, \\ u''(x) &\approx \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}, \end{aligned} \tag{6}$$

are accurate to $O(h^2)$.

Lemma 1. *If u is sufficiently differentiable in a neighbourhood of x then, as $h \to 0$,*

$$\frac{u(x+h) - u(x-h)}{2h} = u'(x) + \frac{u'''(x)}{6}\, h^2 + O(h^4), \tag{7}$$

$$\frac{u(x-h) - 2u(x) + u(x+h)}{h^2} = u''(x) + \frac{u^{(4)}(x)}{12}\, h^2 + O(h^4). \tag{8}$$

Proof. Make the Taylor expansion

$$u(x+h) = u(x) + u'(x)h + \frac{u''(x)}{2}\, h^2 + \frac{u'''(x)}{3!}\, h^3 + \frac{u^{(4)}(x)}{4!}\, h^4 + \frac{u^{(5)}(x)}{5!}\, h^5 + O(h^6)$$

and then replace h with $-h$ to obtain

$$u(x-h) = u(x) - u'(x)h + \frac{u''(x)}{2}\, h^2 - \frac{u'''(x)}{3!}\, h^3 + \frac{u^{(4)}(x)}{4!}\, h^4 - \frac{u^{(5)}(x)}{5!}\, h^5 + O(h^6).$$

Subtracting these two expansions gives

$$u(x+h) - u(x-h) = 2\left[u'(x)h + \frac{u'''(x)}{3!}\, h^3 + O(h^5)\right]$$

from which (7) follows, whereas by adding them we see that

$$u(x+h) + u(x-h) = 2\left[u(x) + \frac{u''(x)}{2}\, h^2 + \frac{u^{(4)}(x)}{4!}\, h^4 + O(h^6)\right]$$

and so (8) holds. □

Applying the approximations (6) to the ODE (5) we arrive at a finite difference equation

$$a_i\, \frac{-u_{i-1} + 2u_i - u_{i-1}}{h^2} + b_i\, \frac{u_{i+1} - u_{i-1}}{2h} + c_i u_i = f_i \quad \text{for } 1 \le i \le N-1. \tag{9}$$

Here,

$$a_i = a(x_i), \quad b_i = b(x_i), \quad c_i = c(x_i), \quad f_i = f(x_i)$$

and, we hope,

$$u_i \approx u(x_i).$$

At this stage we have $N+1$ unknowns u_0, u_1, …, u_N but only $N-1$ equations (9). To arrive at a square linear system we need two more equations, which will come from the boundary conditions at $x = 0$ and at $x = 1$. For simplicity, suppose that we have a Dirichlet problem, i.e.,

$$u(0) = g(0) \quad \text{and} \quad u(L) = g(L).$$

We therefore set

$$u_0 = g_0 \quad \text{and} \quad u_N = g_N,$$

leaving an $(N-1) \times (N-1)$ system

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f} + \boldsymbol{g}$$

whose solution is the vector of unknowns

$$\boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix}.$$

The coefficient matrix has the form

$$K = AD^{(2)} + BD^{(1)} + C \tag{10}$$

with diagonal matrices

$$A = \operatorname{diag}(a_1, a_2, \ldots, a_{N-1}), \qquad B = \operatorname{diag}(b_1, b_2, \ldots, b_{N-1}),$$
$$C = \operatorname{diag}(c_1, c_2, \ldots, c_{N-1}),$$

and tri-diagonal matrices

$$D^{(1)} = \frac{1}{2h} \begin{bmatrix} 0 & 1 & & & & \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 0 & 1 \\ & & & & -1 & 0 \end{bmatrix},$$

$$D^{(2)} = \frac{1}{h^2} \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}.$$

The vectors on the right hand side are

$$\boldsymbol{f} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{N-2} \\ f_{N-1} \end{bmatrix} \quad \text{and} \quad \boldsymbol{g} = \frac{1}{2h} \begin{bmatrix} b_1 g_0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ -b_{N-1} g_N \end{bmatrix}.$$

3 The Five-Point Discrete Laplacian

In this section we will treat a simple, two-dimensional problem ($d = 2$). Consider a rectangular domain in $\mathbb{R}^2$,

$$\Omega = (0, L_1) \times (0, L_2).$$

We choose positive integers N_1 and N_2, define step sizes in the horizontal and vertical directions,

$$h_1 = \frac{L_1}{N_1} \quad \text{and} \quad h_2 = \frac{L_2}{N_2},$$

and introduce the grid points

$$\boldsymbol{x}_{ij} = (ih_1, jh_2) \quad \text{for } 0 \le i \le N_1 \text{ and } 0 \le j \le N_2.$$

Let us solve Poisson's equation (4) subject to a Dirichlet boundary condition (2). We write

$$u_{ij} \approx u(\boldsymbol{x}_{ij}), \quad f_{ij} = f(\boldsymbol{x}_{ij}), \quad g_{ij} = g(\boldsymbol{x}_{ij}),$$

and note that, in view of the second approximation in (6),

$$\begin{aligned} \frac{\partial^2 u}{\partial x_1^2} &\approx \frac{u(x_1 - h_1, x_2) - 2u(x_1, x_2) + u(x_1 + h_1, x_2)}{h_1^2}, \\ \frac{\partial^2 u}{\partial x_2^2} &\approx \frac{u(x_1, x_2 - h_2) - 2u(x_1, x_2) + u(x_1, x_2 + h_2)}{h_2^2}, \end{aligned}$$

so $(\nabla^2 u)(\boldsymbol{x}_{ij})$ may be approximated by the *discrete Laplacian*

$$\nabla_{\boldsymbol{h}}^2 u_{ij} = \frac{u_{i-1,j} - 2u_{ij} + u_{i+1,j}}{h_1^2} + \frac{u_{i,j-1} - 2u_{ij} + u_{i,j+1}}{h_2^2},$$

where $\boldsymbol{h} = (h_1, h_2)$. The finite difference *stencil* for $\nabla_{\boldsymbol{h}}^2$ consists of five points arranged in a star shaped like a + sign.

Denote the set of *interior grid points* by

$$\Omega_{\boldsymbol{h}} = \{\, (ih_1, jh_2) : 1 \le i \le N_1 - 1, 1 \le j \le N_2 - 1 \,\}$$

and the set of *boundary grid points* by

$$\begin{aligned} \partial\Omega_{\boldsymbol{h}} = \{\, (ih_1, jh_2) : i = 0, 0 \le j \le N_2 \,\} \\ \cup \{\, (ih_1, jh_2) : i = N_1, 0 \le j \le N_2 \,\} \\ \cup \{\, (ih_1, jh_2) : 1 \le i \le N_1 - 1, j = 0 \,\} \\ \cup \{\, (ih_1, jh_2) : 1 \le i \le N_1 - 1, j = N_2 \,\}. \end{aligned}$$

Our discrete approximation to (4) and (2) may then be written as

$$\begin{aligned} -a\nabla^2_{\boldsymbol{h}} u_{ij} &= f_{ij} \qquad \text{for } \boldsymbol{x}_{ij} \in \Omega_{\boldsymbol{h}}, \\ u_{ij} &= g_{ij} \qquad \text{for } \boldsymbol{x}_{ij} \in \partial\Omega_{\boldsymbol{h}}. \end{aligned} \tag{11}$$

This discrete problem is equivalent to an $M \times M$ linear system, where

$$M = (N_1 - 1)(N_2 - 1).$$

In fact, if we put

$$\boldsymbol{u}_j = \begin{bmatrix} u_{1j} \\ u_{2j} \\ \vdots \\ u_{N_1-1,j} \end{bmatrix} \quad \text{and} \quad \boldsymbol{f}_j = \begin{bmatrix} f_{1j} \\ f_{2j} \\ \vdots \\ f_{N_1-1,j} \end{bmatrix} \quad \text{for } 1 \le j \le N_2 - 1,$$

then

$$\begin{aligned} aD^{(2)}_{h_1}\boldsymbol{u}_1 + a\frac{2\boldsymbol{u}_1 - \boldsymbol{u}_2}{h_2^2} &= \boldsymbol{f}_1 + \boldsymbol{g}_1, \\ aD^{(2)}_{h_1}\boldsymbol{u}_j + a\frac{-\boldsymbol{u}_{j-1} + 2\boldsymbol{u}_j - \boldsymbol{u}_{j+1}}{h_2^2} &= \boldsymbol{f}_j + \boldsymbol{g}_j, \qquad 2 \le j \le N_2 - 2, \\ aD^{(2)}_{h_1}\boldsymbol{u}_{N_2-1} + a\frac{-\boldsymbol{u}_{N_2-2} + 2\boldsymbol{u}_{N_2-1}}{h_2^2} &= \boldsymbol{f}_{N_2-1} + \boldsymbol{g}_{N_2-1}, \end{aligned}$$

where

$$\boldsymbol{g}_j = \frac{a}{h_1^2}\begin{bmatrix} g_{0j} \\ 0 \\ \vdots \\ 0 \\ g_{N_1,j} \end{bmatrix} \quad \text{for } 2 \le j \le N_2 - 2.$$

with

$$\boldsymbol{g}_1 = \frac{a}{h_1^2}\begin{bmatrix} g_{01} \\ 0 \\ \vdots \\ 0 \\ g_{N_1,1} \end{bmatrix} + \frac{a}{h_2^2}\begin{bmatrix} g_{10} \\ g_{20} \\ \vdots \\ g_{N_1-2,0} \\ g_{N_1-1,0} \end{bmatrix}$$

and

$$\boldsymbol{g}_{N_2-1} = \frac{a}{h_1^2}\begin{bmatrix} g_{0,N_2-1} \\ 0 \\ \vdots \\ 0 \\ g_{N_1,N_2-1} \end{bmatrix} + \frac{a}{h_2^2}\begin{bmatrix} g_{1,N_2} \\ g_{2,N_2} \\ \vdots \\ g_{N_1-2,N_2} \\ g_{N_1-1,N_2} \end{bmatrix}.$$

Thus, if we define the M-dimensional vectors

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{u}_{N_2-1} \end{bmatrix}, \quad \boldsymbol{f} = \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \vdots \\ \boldsymbol{f}_{N_2-1} \end{bmatrix}, \quad \boldsymbol{g} = \begin{bmatrix} \boldsymbol{g}_1 \\ \boldsymbol{g}_2 \\ \vdots \\ \boldsymbol{g}_{N_2-1} \end{bmatrix},$$

then

$$K\boldsymbol{u} = \boldsymbol{f} + \boldsymbol{g}, \tag{12}$$

where, with I_1 denoting the $(N_1 - 1) \times (N_1 - 1)$ identity matrix,

$$K = a \begin{bmatrix} D_{h_1}^{(2)} & & & & \\ & D_{h_1}^{(2)} & & & \\ & & \ddots & & \\ & & & D_{h_1}^{(2)} & \\ & & & & D_{h_1}^{(2)} \end{bmatrix} + \frac{a}{h_2^2} \begin{bmatrix} 2I_1 & -I_1 & & & \\ -I_1 & 2I_1 & -I_1 & & \\ & \ddots & \ddots & \ddots & \\ & & -I_1 & 2I_1 & -I_1 \\ & & & -I_1 & 2I_1 \end{bmatrix}$$

The $M \times M$ matrix K can be expressed succinctly using the *Kronecker product* $\otimes$. For any matrices $A = [a_{ij}] \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$ we define

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix} \in \mathbb{R}^{(mp) \times (nq)}.$$

Thus, if I_2 denotes the $(N_2 - 1) \times (N_2 - 1)$ identity matrix, then

$$K = aI_2 \otimes D_{h_1}^{(2)} + aD_{h_2}^{(1)} \otimes I_2. \tag{13}$$

4 Matrices with Special Structure

When solving linear systems it is important to take advantage of any special structure present in the coefficient matrix. Lapack provides many routines that exploit such structure.

4.1 Bandwidth

We say that the matrix $A = [a_{ij}] \in \mathbb{R}^{n \times n}$ has *upper bandwidth* q if

$$a_{ij} = 0 \quad \text{whenever } j > i + q.$$

Likewise, A has *lower bandwidth* p if

$$a_{ij} = 0 \quad \text{whenever } i > j + p,$$

When $p = q$ we refer to their common value simply as a bandwidth of A. For instance, the matrix (10) arising from the 1D elliptic equation (5) has bandwidth 1, whereas for the 2D Poisson equation (4) we obtain a matrix (13) with bandwidth $N_1 - 1$.

Theorem 2. *Suppose that $A \in \mathbb{R}^{n\times n}$ has LU-factorization $A = LU$.*

1. *If A has upper bandwidth q then so does U.*
2. *If A has lower bandwidth p then so does L.*

Proof. We use induction on n. The case $n = 1$ is trivial, so let $n > 1$ and assume the theorem holds for any matrix in $\mathbb{R}^{(n-1)\times(n-1)}$. Write

$$\boldsymbol{v} = \begin{bmatrix} a_{21} \\ \vdots \\ a_{n1} \end{bmatrix}, \quad \boldsymbol{w} = \begin{bmatrix} a_{12} \\ \vdots \\ a_{1n} \end{bmatrix}, \quad B = \begin{bmatrix} a_{22} & \cdots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

so that

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & \boldsymbol{w}^T \\ \boldsymbol{v} & \boldsymbol{B} \end{bmatrix}$$

then

$$\boldsymbol{A} = \begin{bmatrix} 1 & \boldsymbol{0}^T \\ \boldsymbol{v}/a_{11} & I \end{bmatrix} \begin{bmatrix} 1 & \boldsymbol{0}^T \\ \boldsymbol{0} & B - \boldsymbol{v}\boldsymbol{w}^T/a_{11} \end{bmatrix} \begin{bmatrix} a_{11} & \boldsymbol{w}^T \\ \boldsymbol{0} & I \end{bmatrix}.$$

If A has upper bandwidth q and lower bandwidth p, then so does the matrix $B - \boldsymbol{v}\boldsymbol{w}^T/a_{11} \in \mathbb{R}^{(n-1)\times(n-1)}$. But

$$L = \begin{bmatrix} 1 & \boldsymbol{0}^T \\ \boldsymbol{v}/a_{11} & L_1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} a_{11} & \boldsymbol{w}^T \\ \boldsymbol{0} & U_1 \end{bmatrix}$$

with $B - \boldsymbol{v}\boldsymbol{w}^T/a_{11} = L_1U_1$. Hence, the induction hypothesis implies that L_1 had lower bandwidth p and U_1 has upper bandwidth q. It follows that the same is true for L and U, so the induction goes through. □

For A as in Theorem 2 we see that

$$(LU)_{ij} = \sum_{k=1}^{n} \ell_{ik} u_{kj}$$

$$= \begin{cases} \displaystyle\sum_{k=\max(1,i-p,j-q)}^{j} \ell_{ik} u_{kj} & \text{if } 1 \le j < i \le n \text{ and } i \le j+p, \\ u_{ij} + \displaystyle\sum_{k=\max(1,i-p,j-q)}^{i-1} \ell_{ik} u_{kj} & \text{if } 1 \le i \le j \le n \text{ and } j \le i+q, \\ 0 & \text{otherwise,} \end{cases}$$

so

$$\begin{aligned} \ell_{ij} &= \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=\max(1,i-p,j-q)}^{j-1} \ell_{ik} u_{kj} \right) \quad \text{for } 1 \le j < i \le n \text{ and } i \le j+p, \\ u_{ij} &= a_{ij} - \sum_{k=\max(1,i-p,j-q)}^{i-1} \ell_{ik} u_{kj} \qquad\qquad \text{for } 1 \le i \le j \le n \text{ and } j \le i+q. \end{aligned} \tag{14}$$

Suppose for simplicity that $p = q$. The number of flops to compute L is then

$$\sum_{j=1}^{n-1} \sum_{i=j+1}^{\min(n,j+p)} [j - \max(1, i-p)] \le \sum_{j=1}^{n} \sum_{i=j+1}^{j+p} (j - i + p) \approx \frac{1}{2}\, np^2,$$

and similarly we need about $\frac{1}{2}np^2$ flops to compute U, so the LU-factorization costs $O(np^2)$ flops. In particular, if $p \ll n$ then this cost is much less that the $O(n^3/3)$ cost of factoring a full (i.e., non-banded) $n \times n$ matrix.

In general, the band LU-factorization must incorporate partial pivoting to avoid numerical instability, so that we have $PA = LU$. In this case, it can be shown that if A has lower bandwidth p and upper bandwidth q, then U has upper bandwidth $p + q$ but L might have lower bandwidth as large as $n - 1$.

4.2 Symmetry and Positivity

Recall that a matrix $A = [a_{ij}] \in \mathbb{R}^{n \times n}$ is *symmetric* if $A^T = A$, i.e., if

$$a_{ji} = a_{ij} \quad \text{for all } i,\, j.$$

In this case, A is said to be *positive-definite* if

$$\boldsymbol{x}^T A \boldsymbol{x} > 0 \quad \text{for all non-zero } \boldsymbol{x} \in \mathbb{R}^m.$$

Theorem 3. *A matrix $A \in \mathbb{R}^{n\times n}$ is symmetric and positive-definite iff there exists a non-singular, lower-triangular matrix L such that $A = LL^T$.*

Proof. Omitted. □

We call $A = LL^T$ the *Cholesky factorization* of A. Given the Cholesky factorization, we can solve a linear system $A\boldsymbol{x} = \boldsymbol{b}$ by forward elimination and back substitution just as we would using an LU-factorization since L^T is upper triangular.

The Cholesky factor L is unique:

$$(LL^T)_{ij} = \sum_{k=1}^{n} \ell_{ik}\ell_{jk} = \sum_{k=1}^{\min(i,j)} \ell_{ik}\ell_{jk}$$

so $A = LL^T$ iff

$$\ell_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} \ell_{ik}^2} \qquad \text{for } 1 \le i \le n,$$

$$\ell_{ij} = \frac{1}{\ell_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik}\ell_{jk}\right) \quad \text{for } 1 \le j < i \le n.$$

These formulae may be used to compute L; of course, if the matrix A is not positive-definite then the algorithm will fail because one of the ℓ_{ii} is the square root of a negative number. It is easy to verify that the Cholesky factorization costs about $n^3/6$ flops, i.e., about half the cost of the LU-factorization.

If A is positive-definite then

$$\ell_{ij}^2 \le \sum_{k=1}^{i} \ell_{ik}^2 = a_{ii}$$

so

$$|\ell_{ij}| \le \sqrt{a_{ii}} \quad \text{for } 1 \le j \le i \le n.$$

Thus, we cannot encounter the sort of numerical instability that may happen in an LU-factorization without pivoting. It can also be shown that if $A\boldsymbol{x} = \boldsymbol{b}$ is solved numerically via the Cholesky factorization of A then the computed solution $\hat{\boldsymbol{x}}$ is the exact solution of a perturbed linear system

$$(A + E)\hat{\boldsymbol{x}} = \boldsymbol{b}$$

with

$$\|E\|_2 \le C_n\epsilon\|A\|_2.$$

Furthermore, no square roots of negative numbers will arise provided

$$\mathrm{cond}_2(A) \leq \frac{C_n}{\epsilon}.$$

In both cases, the constant C_n is of moderate size.

If A is not only symmetric and positive-definite but also has bandwidth p, then the Cholesky factor L has bandwidth p and can be computed in about $np^2/2$ flops.

4.3 Diagonal Dominance

The matrix $A = [a_{ij}] \in \mathbb{R}^{n\times n}$ is *row-diagnally dominant* if

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}| \quad \text{for } 1 \leq i \leq n.$$

The following result shows that in this case, pivoting is not required.

Theorem 4. *If $A \in \mathbb{R}^{n\times n}$ is row-diagonally dominant then it has an LU-factorization $A = LU$ with $|\ell_{ij}| \leq 1$ for all $i > j$.*

Proof. Omitted. □

5 Direct Solution of a Poisson Problem

We return to the problem discussed in Section 3. The FORTRAN module `poisson_solvers.f95` contains several subroutines that set up and solve the linear system (12).

The subroutine `direct_solve` constructs a full $M \times M$ matrix together with the vector of right hand sides and then calls the Lapack routine `sposv` to compute the solution. This approach is very inefficient, both in memory usage and CPU time, because the bandwidth $N_1 - 1$ of the coefficient matrix K is much smaller than $M = (N_1 - 1)(N_2 - 1)$.

A better approach is used by `band_solve` which constructs K in *band storage mode*: the ij-entry is stored in position $(i-j+1, j)$ of an $N_1 \times M$ array. Thus, the main diagonal of K is stored in the first row of the array, the leading subdiagonal in the second row, and so on until the $(N_1 - 1)$-th diagonal is stored in the row N_1.

The program `test_poisson.f95` solves the problem in three ways: using `direct_solve`, `band_solve` and a routine `fast_solve`. The third routine

will be explained later. The data f and g are chosen so that the exact solution is known, and the program prints the maximum error in the finite difference solution together with the CPU time taken by each routine. The two routines produce nearly identical results but, as expected, the second is much faster. Also, using `direct_solve` you will soon run out of memory if you try to increase the grid resolution. Use the makefile `poisson.mk` to compile and link `test_poison`.

If, for simplicity, we assume that $L_1 = L_2 = L$ and $N_1 = N_2 = N$ then the cost of solving the full $M \times M$ system is $O(M^3/3)$. The band solver reduces this to $O(MN_1^2) = O(M^2)$. It can be shown that

$$u(ih, jh) = u_{ij} + O(h^2), \qquad h = h_1 = h_2 = L/N,$$

and since $h = O(N_1^{-1}) = O(M^{-1/2})$ we see that the error is $O(M^{-1})$. Thus, every time we double the value of N we expect the error to be reduced to about $\frac{1}{4}$ of its previous value but the number of unknowns M grows by a factor of 4 so that the solution cost grows by a factor of $4^3 = 48$ if we do not exploit the band structure. Even if we use a band solver the cost grows by a factor of $4^2 = 16$.

6 A Fast Poisson Solver

There exist a number of algorithms that reduce the cost of solving Poisson's equation from $O(M^2)$ for a band Cholesky solver to $O(M)$ or something that is only a bit worse than $O(M)$. In this section, we will see how the *fast Fourier transform* or *FFT* may be exploited to obtain a reasonably simple Poisson solver with $O(M \log M)$ complexity. For simplicity, we will assume that the Dirichlet data is identically zero, so that

$$\begin{aligned} -a\nabla^2 u &= f \qquad \text{in } \Omega, \\ u &= 0 \qquad \text{on } \partial\Omega. \end{aligned} \tag{15}$$

6.1 The Fast Fourier Transform

The *discrete Fourier transform* of a sequence of n complex numbers a_0, a_1, $\ldots$, a_{N-1} is the sequence b_0, b_1, $\ldots$, b_{N-1} defined by

$$b_p = \sum_{j=0}^{N-1} a_j W^{-pj} \quad \text{for } 0 \le p \le N-1, \text{ where } W = e^{-i2\pi/n}.$$

Notice that W is an nth root of unity, i.e., $W^n = 1$.

Lemma 5. *The original sequence a_j can be reconstructed from its discrete Fourier transform using the inversion formula*

$$a_j = \frac{1}{N} \sum_{p=0}^{N-1} b_p W^{pj} \quad \textit{for } 0 \le j \le N-1.$$

To compute a discrete Fourier transform in the obvious way by evaluating N sums, each with N terms, requires $O(N^2)$ flops but this cost can be reduced to $O(N \log_2 N)$ by using an algorithm called the *fast Fourier transform* or *FFT*. We will use an open-source library called FFTW (Fastest Fourier Transform in the West) that provides efficient implementations of a variety of transforms. The library is coded in C but also has a FORTRAN 77 interface. The FFTW home page is

`www.fftw.org`

and on the lab PCs you will find `libfftw3.a` in `/usr/local/numeric/lib`

The programs `test_fourier.c` and `test_fourier.f95` use both methods to reconstruct a random sequence, printing out the CPU time and the maximum error in the reconstructed sequence for each case. A rule is provided in `poisson.mk` to compile and link the each program.

The FFT may be used to evaluate various other types of sums. In the next section, we will need to compute a *discrete sine transform*

$$b_p = 2 \sum_{j=1}^{N-1} a_j \sin\left(\frac{\pi p j}{N}\right) \quad \text{for } p = 1,\, 2,\, \ldots,\, N-1.$$

Since

$$\sin\left(\frac{\pi p j}{N}\right) = \frac{e^{\pi p j/N} - e^{-\pi p j/N}}{2i} = \frac{W^{pj/2} - W^{-pj/2}}{2i}$$

where

$$W = e^{i2\pi/N'} \quad \text{and} \quad N' = 2N,$$

we see that

$$-ib_p = \sum_{j=1}^{N-1} a_j W^{-pj} - \sum_{j=1}^{N-1} a_j W^{pj}$$

The substitution $k = N' - j$ gives $W^{pj} = W^{pN'-pk} = W^{-pk}$ so

$$\sum_{j=1}^{N-1} a_j W^{pj} = \sum_{k=N+1}^{N'-1} a_{N'-k} W^{-pk}$$

and

$$b_p = \sum_{j=0}^{N'-1} c_j W^{-pj}$$

where

$$c_j = \begin{cases} 0 & \text{if } j = 0 \text{ or } j = N, \\ ia_j & \text{if } 1 \le j \le N-1, \\ -ia_{N'-j} & \text{if } N+1 \le j \le N'-1, \end{cases}$$

The FFTW library implements a *fast sine transform* in this way.

6.2 Fourier Expansion in the Space Variable

We expand the solution u into a Fourier sine series in x:

$$u(x,y) = \sum_{p=1}^{\infty} \hat{u}(p,y) \sin\left(\frac{\pi p}{L_1} x\right) \quad \text{for } 0 < x < L_1, \tag{16}$$

where the Fourier coefficients are given by

$$\hat{u}(p,y) = \frac{2}{L_1} \int_0^{L_1} u(x,y) \sin\left(\frac{\pi p}{L_1} x\right) dx \quad \text{for } p = 1,\, 2,\, 3,\, \ldots.$$

We also expand the source term in the same way:

$$f(x,y) = \sum_{p=1}^{\infty} \hat{f}(p,y) \sin\left(\frac{\pi p}{L_1} x\right), \tag{17}$$

$$\hat{f}(p,y) = \frac{2}{L_1} \int_0^{L_1} f(x,y) \sin\left(\frac{\pi p}{L_1} x\right) dx. \tag{18}$$

Since

$$-\frac{\partial^2}{\partial x^2} \sin\left(\frac{\pi p}{L_1} x\right) = \left(\frac{\pi p}{L_1}\right)^2 \sin\left(\frac{\pi p}{L_1} x\right)$$

we see that (15) is equivalent to the sequence of *ordinary* differential equations

$$a\left(\frac{\pi p}{L_1}\right)^2 \hat{u}(p,y) - a\,\frac{\partial^2 \hat{u}}{\partial y^2} = \hat{f}(p,y) \quad \text{for } 0 < y < L_2, \tag{19}$$

with boundary conditions

$$\hat{u}(p,0) = 0 \quad \text{and} \quad \hat{u}(p,L_2) = 0,$$

for $p = 1,\, 2,\, 3,\, \ldots$.

We apply the trapezoidal rule with step-size h_1 to the integral (18) with $y = kh_2$ and obtain the approximation

$$\hat{f}(p, kh_2) \approx \hat{f}_{pk} = \frac{2}{N_1} \sum_{j=1}^{N_1-1} f(jh_1, kh_2) \sin\left(\frac{\pi p}{L_1} jh_1\right)$$

Next, we generate approximations

$$\hat{u}_{pk} \approx \hat{u}(p, kh_2)$$

by applying a central difference approximation in (19):

$$a\left(\frac{\pi p}{L_1}\right)^2 \hat{u}_{pk} + a\,\frac{-\hat{u}_{p,k-1} + 2\hat{u}_{pk} - \hat{u}_{p,k-1}}{h_2^2} = \hat{f}_{pk} \quad \text{for } k = 1,\, 2,\, \ldots,\, N_2 - 1,$$

with boundary conditions $\hat{u}_{p0} = 0 = \hat{u}_{pN_2}$. Thus, we can compute $\hat{u}_{pk}$ by solving an $(N_2-1)\times(N_2-1)$ symmetric, positive-definite, tri-diagonal linear system for each p.

Having obtained $\hat{u}_{pk}$ we truncate the Fourier series in (16) to obtain an approximation to the solution of (15),

$$u(jh_1, kh_2) \approx u_{jk} = \sum_{p=1}^{N_1-1} \hat{u}_{pk} \sin\left(\frac{\pi p}{L_1} jh_1\right).$$

The cost of computing u_{jk} in this way is

1. $O(N_2 N_1 \log_2 N_1)$ flops using FFTs to compute $\hat{f}(p, kh_2)$ for $0 \le p \le N_1 - 1$ and $1 \le k \le N_2 - 1$;

2. $O(N_2 N_1)$ flops using a band Cholesky solver to find $\hat{u}_{pk}$ for $1 \le p \le N_1 - 1$ and $0 \le k \le N_2 - 1$;

3. $O(N_2 N_1 \log_2 N_1)$ flops using FFTs to compute u_{jk} for $0 \le j \le N_1 - 1$ and $1 \le k \le N_2 - 1$.

Assuming that $N_1 = O(N_2)$ so that $\log_2 N_1 = O(\log_2 \sqrt{M}) = O(\log_2 M)$ it follows that the total cost is $O(M \log_2 M)$. Here, we have not counted the work to evaluate f at each interior grid point, but this will be $O(M)$ provided it costs $O(1)$ flops to evaluate f at each individual point.

The routine `fast_solve` from `poisson_solvers.f95` implements the algorithm described above.

7 Tutorial Exercises

7.1 Comparison of Computational Costs

Modify the module `Dirichlet_problem` in `test_poisson.f95` so that the exact solution is

$$u(x, y) = x(L_1 - x) \sin\left(\frac{\pi y}{L_2}\right).$$

Obviously, you must also modify `f` and `g` accordingly.

1. Verify that the maximum error is $O(h_1^2 + h_2^2)$ for `fast_solve` and $O(h_2^2)$ for `band_solve` and `direct_solve`.

2. Modify the program `test\_poisson` so that you can estimate the constants in the asymptotic complexity bounds $O(M \log_2 M)$, $O(M^2)$ and $O(M^3)$ for `fast_solve`, `band_solve` and `direct_solve`, respectively. For example, in the case of `fast_solve` the constant should be approximately equal to the CPU time divided by $M \log_2 M$.

3. Determine the minimum error achievable in under 60 seconds using `fast_solve`. Estimate how long it would take to achieve the same accuracy using `band_solve` and `direct_solve`. How many Megabytes of RAM would you need in each case?