TMA4205 Numerical Linear Algebra

The Poisson problem in $\mathbb{R}^2$: diagonalization methods

September 3, 2007

©Einar M. Rønquist
Department of Mathematical Sciences
NTNU, N-7491 Trondheim, Norway

1 A direct method based on diagonalization

We consider here the numerical solution of the Poisson problem based on finite differences. In particular, we focus on a particular method for solving the linear system of equations in one and two space dimensions. The method is based on *diagonalization*, and we first explain the basic approach in the context of the one-dimensional Poisson problem:

$$\begin{aligned} -u_{xx} &= f \quad \text{in } \Omega = (0,1), \\ u(0) = u(1) &= 0. \end{aligned}$$

Assume that we use a uniform finite difference *grid* given by:

$$x_i = x_0 + ih, \quad i = 0, 1, \ldots, n.$$

The corresponding system of algebraic equations can be written as:

$$\frac{1}{h^2} \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_{n-1} \end{bmatrix},$$

where u_i is an approximation to $u(x_i) = u(ih), \quad i = 1, \ldots, n-1$, $f_i = f(x_i)$, and $u_0 = u_n = 0$ due to the specified boundary conditions. Let us write this system as

$$\frac{1}{h^2} \underline{T}\, \underline{u} = \underline{f}$$

where

$$\underline{T} = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix}, \qquad \underline{u} = \begin{bmatrix} u_1 \\ \vdots \\ \vdots \\ \vdots \\ u_{n-1} \end{bmatrix}, \qquad \underline{f} = \begin{bmatrix} f_1 \\ \vdots \\ \vdots \\ \vdots \\ f_{n-1} \end{bmatrix},$$

and h is the grid size or mesh size. Since $\underline{T}$ is symmetric positive definite, it can be *diagonalized*. (Recall that a real, symmetric matrix is a *normal* matrix.)

1.1 Diagonalization of $\underline{T}$

Diagonalization of $\underline{T}$ means that we wish to find the eigenvalues λ_j and the eigenvectors $\underline{q}_j$ of $\underline{T}$,

$$\underline{T}\,\underline{q}_j = \lambda_j\,\underline{q}_j, \quad j = 1, \ldots, n-1,$$

where

$$\begin{aligned} \lambda_j &> 0 \qquad \text{(positive eigenvalues)}, \\ \underline{q}_k^T \underline{q}_j &= \delta_{jk} \qquad \text{(orthonormal eigenvectors)}. \end{aligned}$$

We collect all the eigenvectors $\underline{q}_j$ into the (orthonormal) matrix $\underline{Q}$,

$$\underline{Q} = [\underline{q}_1, \underline{q}_2, \ldots, \underline{q}_{n-1}].$$

Then

$$\underline{T}\,\underline{Q} = \underline{Q}\,\underline{\Lambda}$$

where

$$\underline{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_{n-1}) = \begin{bmatrix} \lambda_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \lambda_{n-1} \end{bmatrix}.$$

Since

$$\underline{Q}^T\underline{Q} = \underline{I} = \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \qquad \Rightarrow \quad \underline{Q}^T = \underline{Q}^{-1},$$

and

$$\underline{T} = \underline{Q}\,\underline{\Lambda}\,\underline{Q}^T \tag{1}$$

or

$$\underline{Q}^T\,\underline{T}\,\underline{Q} = \underline{\Lambda} \qquad \text{(diagonal)}.$$

The finite difference approximation can thus be computed as follows:

$$\underline{g} \equiv h^2 \underline{f} \qquad \underline{g} : \mathcal{O}(n) \text{ operations}$$

$$\begin{aligned} \underline{T}\,\underline{u} &= \underline{g} \\ \underline{Q}\,\underline{\Lambda}\,\underline{Q}^T\underline{u} &= \underline{g} \\ \underline{\Lambda}\,\underbrace{\underline{Q}^T\,\underline{u}}_{\widetilde{\underline{u}}} &= \underbrace{\underline{Q}^T\underline{g}}_{\widetilde{\underline{g}}} \end{aligned} \qquad \widetilde{\underline{g}} : \mathcal{O}(n^2) \text{ operations}$$

$$\begin{aligned} \underline{\Lambda}\widetilde{\underline{u}} &= \widetilde{\underline{g}} \\ \widetilde{\underline{u}} &= \underline{\Lambda}^{-1}\widetilde{\underline{g}} \\ \underline{Q}^T\underline{u} &= \widetilde{\underline{u}} \\ \underline{u} &= \underline{Q}\,\widetilde{\underline{u}} \end{aligned} \qquad \begin{aligned} &\\ \widetilde{\underline{u}} &: \mathcal{O}(n) \text{ operations} \\ &\\ \underline{u} &: \mathcal{O}(n^2) \text{ operations} \end{aligned}$$

Note that the transformations

$$\widetilde{\underline{g}} = \underline{Q}^T\underline{g} \text{ and } \underline{u} = \underline{Q}\,\widetilde{\underline{u}}$$

are *matrix-vector products operations*. In summary, we can compute $\underline{u}$ in

$$\mathcal{O}(n) + \mathcal{O}(n^2) + \mathcal{O}(n) + \mathcal{O}(n^2) \sim \mathcal{O}(n^2) \text{ floating-point operations.}$$

Hence, we can solve our finite difference system in $(n-1)$ unknowns in $\mathcal{O}(n^2)$ operations. This is *not* competitive with a direct solution algorithm based upon LU-factorization (Gaussian elimination) of a tridiagonal matrix, which can be done in $\mathcal{O}(n)$ operations (since the bandwidth is equal to one).

Let us also compare the memory requirement:

$\mathcal{O}(n^2)$ for the diagonalization approach (we need to store $\underline{Q}$);
$\mathcal{O}(n)$ for a tridiagonal direct solver.

Again, the diagonalization approach is *not* competitive.

So, why bother? The answer is that the diagonalization approach becomes more interesting in $\mathbb{R}^2$. In addition, it turns out that it is possible to use the Fast Fourier Transform (FFT) to lower the computational complexity.

2 The Poisson problem in $\mathbb{R}^2$

The two-dimensional Poisson problem on the unit square is given by

$$\begin{aligned} -\nabla^2 u &= f \qquad \text{in } \Omega = (0,1)\times(0,1), \\ u &= 0 \qquad \text{on } \partial\Omega, \end{aligned} \tag{2}$$

where $\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$.

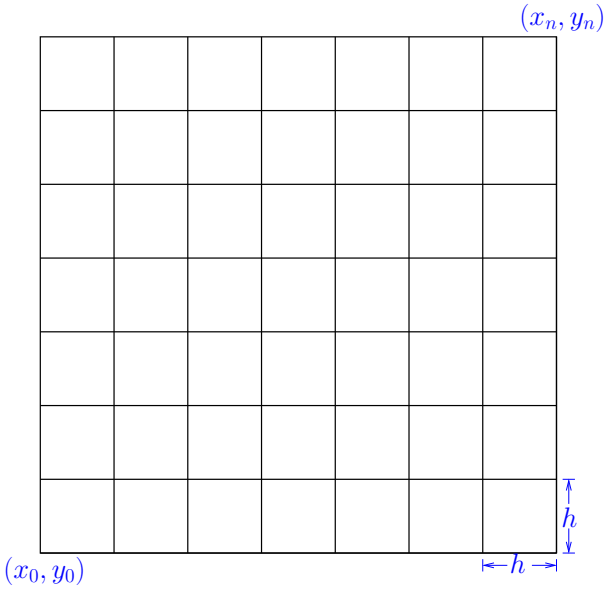

Figure 1: A uniform finite difference grid.

Again, using the notation $u_{i,j} \simeq u(x_i, y_j) = u(ih, jh)$ and $f_{i,j} = f(x_i, y_j)$, and discretizing (2) using the 5-point stencil (see Figure 1), the discrete equations read:

$$-\frac{(u_{i+1,j} - 2u_{i,j} + u_{i-1,j})}{h^2} - \frac{(u_{i,j+1} - 2u_{i,j} + u_{i,j-1})}{h^2} = f_{i,j} \qquad 1 \le i, j \le n-1. \tag{3}$$

2.1 Diagonalization

Let

$$\underline{U} = \begin{bmatrix} u_{1,1} & \cdots & \cdots & u_{1,n-1} \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ u_{n-1,1} & \cdots & \cdots & u_{n-1,n-1} \end{bmatrix}$$

and

$$\underline{T} = \begin{bmatrix} 2 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & & \ddots & & \\ & & -1 & 2 & -1 \\ 0 & & & -1 & 2 \end{bmatrix}$$

Then,

$$\begin{aligned} (\underline{T}\,\underline{U})_{ij} &= 2u_{i,j} - u_{i+1,j}, && i = 1, \\ (\underline{T}\,\underline{U})_{ij} &= -u_{i-1,j} + 2u_{i,j} - u_{i+1,j}, && 2 \leq i \leq n-2, \\ (\underline{T}\,\underline{U})_{ij} &= -u_{i-1,j} + 2u_{i,j}, && i = n-1. \end{aligned}$$

and thus,

$$\boxed{\frac{1}{h^2}(\underline{T}\,\underline{U})_{ij} \simeq -\left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j}.} \tag{4}$$

Similarly,

$$\boxed{\frac{1}{h^2}(\underline{U}\,\underline{T})_{ij} \simeq -\left(\frac{\partial^2 u}{\partial y^2}\right)_{i,j}.} \tag{5}$$

Our finite difference system (3) can thus be expressed as

$$\frac{1}{h^2}(\underline{T}\,\underline{U} + \underline{U}\,\underline{T})_{ij} = f_{i,j} \qquad \text{for} \qquad \begin{matrix} 1 \leq i \leq n-1, \\ 1 \leq j \leq n-1, \end{matrix}$$

or

$$\underline{T}\,\underline{U} + \underline{U}\,\underline{T} = \underline{G} \tag{6}$$

where

$$\underline{G} = h^2 \begin{bmatrix} f_{1,1} & \cdots & \cdots & f_{1,n-1} \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ f_{n-1,1} & \cdots & \cdots & f_{n-1,n-1} \end{bmatrix}.$$

Combining (1) and (6) we get

$$\underline{Q}\,\underline{\Lambda}\,\underline{Q}^T\underline{U} + \underline{U}\,\underline{Q}\,\underline{\Lambda}\,\underline{Q}^T = \underline{G}. \tag{7}$$

Multiplying (7) from the right with $\underline{Q}$ and from the left with $\underline{Q}^T$, and using the fact that $\underline{Q}^T\underline{Q} = \underline{I}$, we get:

$$\underline{\Lambda}\underbrace{\underline{Q}^T\,\underline{U}\,\underline{Q}}_{\equiv\widetilde{\underline{U}}} + \underbrace{\underline{Q}^T\,\underline{U}\,\underline{Q}}_{\equiv\widetilde{\underline{U}}}\underline{\Lambda} = \underbrace{\underline{Q}^T\,\underline{G}\,\underline{Q}}_{\equiv\widetilde{\underline{G}}}.$$

Hence, (6) may be solved in three steps:

Step 1): Compute

$$\boxed{\widetilde{\underline{G}} = \underline{Q}^T\,\underline{G}\,\underline{Q}} \quad - \quad \text{matrix-matrix products.}$$

Step 2): Solve

$$\underline{\Lambda}\,\widetilde{\underline{U}} + \widetilde{\underline{U}}\,\underline{\Lambda} = \widetilde{\underline{G}}$$

or

$$\begin{aligned}
\lambda_i\,\widetilde{u}_{i,j} + \widetilde{u}_{i,j}\,\lambda_j &= \widetilde{g}_{i,j}, \qquad 1 \le i,j \le n-1 \\
(\lambda_i + \lambda_j)\,\widetilde{u}_{i,j} &= \widetilde{g}_{i,j}, \qquad 1 \le i,j \le n-1 \\
\boxed{\widetilde{u}_{i,j} = \frac{\widetilde{g}_{i,j}}{\lambda_i + \lambda_j}} &\qquad 1 \le i,j \le n-1.
\end{aligned}$$

Step 3): Compute

$$\boxed{\underline{U} = \underline{Q}\,\widetilde{\underline{U}}\,\underline{Q}^T} \quad - \quad \text{matrix-matrix products.}$$

Here,

$$\underline{U}, \widetilde{\underline{U}}, \widetilde{\underline{G}}, \underline{Q}, \underline{Q}^T \in \mathbb{R}^{(n-1)\times(n-1)}.$$

2.1.1 Computational cost

The number of degrees-of-freedom (or unknowns), N, is

$$N = (n-1)^2 \sim \mathcal{O}(n^2) \quad (n \gg 1).$$

Step 1)

$$\underline{\widetilde{G}} = \overbrace{\underbrace{\underline{Q}^T \underline{G}}_{\mathcal{O}(n^3)} \underline{Q}}^{\mathcal{O}(n^3)} \qquad \longrightarrow \qquad \mathcal{O}(n^3) \text{ operations.}$$

Step 2)

$$\widetilde{u}_{i,j} = \frac{\widetilde{g}_{i,j}}{\lambda_i + \lambda_j} \qquad \longrightarrow \qquad \mathcal{O}(n^2) \text{ operations.}$$

Step 3)

$$\underline{U} = \overbrace{\underline{Q} \underbrace{\underline{\widetilde{U}}\, \underline{Q}^T}_{\mathcal{O}(n^3)}}^{\mathcal{O}(n^3)} \qquad \longrightarrow \qquad \mathcal{O}(n^3) \text{ operations.}$$

In summary, we can compute the discrete solution, $\underline{U}$, in $\mathcal{O}(n^3) = \mathcal{O}(N^{3/2})$ operations. Note: this method is an example of a *direct method*.

2.1.2 Comparison with other direct methods

Computational cost		
Method	Operations ($\mathcal{N}_{op}$)	Memory requirement ($\mathcal{M}$)
Diagonalization	$\mathcal{O}(N^{3/2}) = \mathcal{O}(n^3)$	$\mathcal{O}(N) = \mathcal{O}(n^2)$
Banded LU	$\mathcal{O}(Nb^2) = \mathcal{O}(n^4)$	$\mathcal{O}(Nb) = \mathcal{O}(n^3)$
Full LU	$\mathcal{O}(N^3) = \mathcal{O}(n^6)$	$\mathcal{O}(N^2) = \mathcal{O}(n^4)$

Table 1: Computational cost and memory requirement for direct methods. For the banded solver, we have used a bandwidth $b \sim \mathcal{O}(n)$.

We conclude that the diagonalization method is much more attractive in $\mathbb{R}^2$ than in $\mathbb{R}^1$. The number of floating-point operations per degree-of-freedom is $\mathcal{O}(n)$, while the memory requirement is close to optimal (i.e., scalable).

2.1.3 The matrices $\underline{Q}$ and $\underline{\Lambda}$.

The computational cost associated with the diagonalization approach tacitly assumes that we know the eigenvector matrix $\underline{Q}$ and the corresponding eigenvalues. Let us therefore derive explicit expressions for these. To this end, consider first the continuous eigenvalue problem

$$\begin{aligned} -u_{xx} &= \lambda u \qquad\qquad \text{in } \Omega = (0,1), \\ u(0) = u(1) &= 0, \end{aligned}$$

with solutions

$$\begin{aligned} u_j^*(x) &= \sin(j\pi x), \\ \lambda_j^* &= j^2\pi^2, \end{aligned} \qquad\qquad j = 1, 2, \ldots, \infty.$$

Consider now the discrete eigenvalue problem

$$\underline{T}\,\underline{\widetilde{q}}_j = \lambda_j\,\underline{\widetilde{q}}_j.$$

Try eigenvector solutions which correspond to the continuous eigenfunctions $u_j^*(x)$ sampled at the grid points $x_i, i = 1, \ldots, n-1$, i.e.,

$$\begin{aligned} (\underline{\widetilde{q}}_j)_i &= u_j^*(x_i) \\ &= \sin(j\pi x_i) \\ &= \sin(j\pi(ih)), \qquad \left(h = \frac{1}{n}\right) \\ &= \sin\left(\frac{ij\pi}{n}\right) \end{aligned}$$

Operating on $\underline{\widetilde{q}}_j$ with $\underline{T}$ gives

$$(\underline{T}\underline{\widetilde{q}}_j)_i = \underbrace{2\left(1 - \cos\left(\frac{j\pi}{n}\right)\right)}_{\lambda_j}\underbrace{\sin\left(\frac{ij\pi}{n}\right)}_{(\underline{\widetilde{q}}_j)_i}.$$

Hence, our try was successful: operating on $\underline{\widetilde{q}}_j$ with $\underline{T}$ gives a multiple of $\underline{\widetilde{q}}_j$.

In order to proceed, set $\underline{q}_j = \alpha\, \underline{\widetilde{q}}_j$, and choose α such that $\underline{q}_j$ is normalized:

$$\begin{aligned} &\underline{q}_j^T \underline{q}_j = 1, \\ &\qquad \Downarrow \\ &(\underline{q}_j)_i = \sqrt{\frac{2}{n}} \sin\left(\frac{ij\pi}{n}\right), \qquad\qquad 1 \le i, j \le n-1, \\ &\lambda_j = 2\left(1 - \cos\left(\frac{j\pi}{n}\right)\right). \end{aligned}$$

For $j \ll n$, we observe that

$$\lambda_j \simeq 2\left(1 - \left(1 - \frac{1}{2}\frac{j^2\pi^2}{n^2} + \ldots\right)\right) \simeq \frac{j^2\pi^2}{n^2}.$$

Since $h = \frac{1}{n}$, we have

$$\lambda_j \simeq h^2 j^2 \pi^2 = h^2 \lambda_j^* \qquad \text{for } j \ll n.$$

Since the approximation of the one-dimensional Laplace operator on our finite difference grid is equal to $\frac{1}{h^2}\,\underline{T}$ (and not $\underline{T}$), this is the same as saying that the first, lowest eigenvalues (and eigenvectors) for the continuous case are well approximated by our finite difference formulation.

Note that in this case

$$Q_{ij} = (\underline{q}_j)_i = \sqrt{\frac{2}{n}} \sin\left(\frac{ij\pi}{n}\right), \qquad 1 \le i, j \le n-1,$$

and that indeed

$$\underline{Q}^T = \underline{Q}.$$

From the comparison of computational cost shown earlier (see Table 1), the diagonalization approach to solving the discrete Poisson problem appears promising.

Questions:

1. Is the memory requirement optimal?
2. Can the matrix-matrix multiplications be done fast?
3. Can we do better?
4. Can the diagonalization method be extended to three space dimensions?
5. Can the diagonalization method be used on general domains?

The answer to the first question is yes: we need to store $\mathcal{O}(n^2)$ floating point numbers for $\mathcal{O}(n^2)$ unknowns. Note that we store the unknowns as a multi-dimensional array instead of as a long vector. Also note that we never form the global system matrix in the two-dimensional case.

The answer to the second question is yes. Matrix-matrix multiplication is one of the fastest floating-point tasks on modern microprocessors. The best performance is normally achieved using the appropriate BLAS (Basic Linear Algebra Subroutines) library function since this library is optimized for each particular microprocessor. Matrix-matrix multiplication is one of the operations which comes closest to maximum theoretical performance (i.e., the maximum number of floating point operations per second). The reason for this is that the number of operations per memory reference is high: $\mathcal{O}(n^3)$ floating point operations and $\mathcal{O}(n^2)$ memory references. Note that bringing data to and from memory is often the bottleneck when it comes to performance.

The answer to the third question is yes. The matrix-vector multiplication $\underline{w} = \underline{Q}\,\underline{v}$ (or $\underline{w} = \underline{Q}^T\,\underline{v}$ can alternatively be done via the Discrete Sine Transform (DST). This is, of course, related to the fact that the columns of $\underline{Q}$ represent the continuous eigenfunctions $u_j^*(x) = \sin(j\pi x)$, $j = 1, \ldots, n-1$, sampled at the internal grid points. However, the DST is again related to the Discrete Fourier Transform (DFT), which can be performed most efficiently using the Fast Fourier Transform (FFT). Hence, instead of obtaining $\underline{w} = \underline{Q}\,\underline{v}$ via matrix-vector multiplication in $\mathcal{O}(n^2)$ operations, we can alternatively obtain $\underline{w}$ from $\underline{v}$ via FFT in $\mathcal{O}(n \log n)$ operations. The solution of the Poisson problem in two space dimensions (i.e., $\mathcal{O}(n^2)$ unknowns), can therefore be reduced from $\mathcal{O}(n^3)$ operations to $\mathcal{O}(n^2 \log n)$ operations. This is close to an optimal solver: $\mathcal{O}(\log n)$ operations per unknown and $\mathcal{O}(1)$ storage requirement per unknown.

The answer to the fourth question is yes, assuming that the domain is an *undeformed* box and we use a simple, structured grid. This is also related to the fifth question: the fast solution method presented here is only applicable on simple domains. However, the approach is also attractive as a preconditioner for problems which do not fall into this category (more on this later). The solver presented here is an example of a class of solvers called *tensor-product solvers*.