Solving Two-Dimensional Poisson Equation with Multigrid Preconditioned Conjugate Gradient Method

Hongyi Zhang
Yuanpei College, Peking University
hongyizhang@pku.edu.cn

January 25, 2018

1 Problem formulation

Given a function f, consider the following anisotropic Poisson equation with homogeneous Dirichlet boundary conditions:

$$\begin{cases} -u_{xx} - \epsilon u_{yy} = f, \quad u : \Omega \to \mathbb{R}, \\ u|_{\partial\Omega} = 0, \end{cases}$$

where $u(x, y)$ is defined on $\Omega = (0, 1) \times (0, 1)$. Take an $N \times N$ mesh on Ω with stride $h = 1/N$ in both x and y directions. Discretizing the equation with five-point finite difference scheme yields

$$-\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} - \epsilon \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2} = f_{i,j}, \quad 1 \le i, j \le N-1, \tag{*}$$

and $u_{i,j} = 0$ on the boundary (namely when $\{i, j\} \cap \{0, N\} \neq \emptyset$). If we vectorize the uniform grid in the row-major order, i.e. let $u_i = (u_{i,1}, u_{i,2}, \cdots, u_{i,N-1})^T$ and $f_i = (f_{i,1}, f_{i,2}, \cdots, f_{i,N-1})^T$ for $1 \le i \le N-1$, then the discrete equation (*) can be rewritten as $A_h u_h = f_h$, which expands to

$$\begin{bmatrix} A_{N-1} & -I_{N-1}/h^2 & & & \\ -I_{N-1}/h^2 & A_{N-1} & -I_{N-1}/h^2 & & \\ & \ddots & \ddots & \ddots & \\ & & -I_{N-1}/h^2 & A_{N-1} & -I_{N-1}/h^2 \\ & & & -I_{N-1}/h^2 & A_{N-1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-2} \\ f_{N-1} \end{bmatrix},$$

where $A_{N-1} = \text{tridiag}(-\epsilon, 2(1+\epsilon), -\epsilon)/h^2 \in \mathbb{R}^{N-1 \times N-1}$ is a symmetric tridiagonal matrix. The coefficient matrix can also be expressed in a more compact form:

$$A_h = T_{N-1} \otimes I_{N-1} + \epsilon \, I_{N-1} \otimes T_{N-1},$$

where $T_{N-1} = \text{tridiag}(-1, 2, -1)/h^2 \in \mathbb{R}^{N-1\times N-1}$ and $\otimes$ denotes the Kronecker product. The Kronecker product of matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$ is defined as

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

and this notation will greatly reduce the descriptive complexity of our equations.

For this project, we fix the true solution $u(x, y) = \sin(\pi x)\sin(\pi y)$ which is symmetric with respect to x and y, and the corresponding function $f(x, y) = \pi^2(1 + \epsilon)u(x, y)$. Based on this setting, we explore a series of methods to solve the discrete equation, including Gaussian elimination, conjugate gradient methods, discrete sine transform as well as multigrid method. Multiple comparisons are made to illustrate the strengths and weaknesses of each method.

2 Classical methods

2.1 Gaussian elimination

Gaussian elimination makes use of elementary row operations to solve systems of linear equations. At time step k, the algorithm eliminates $A(k + 1 : n, k)$ by subtracting a multiple of the k-th row from each row below, and the process can be represented as LU decomposition.

For the discrete Poisson equation, we take two advantages to simplify the computation. Firstly, note that the coefficient matrix A_h is diagonally dominant, hence there is no need for partial pivoting and the numerical stability is guaranteed in nature. Secondly, considering that A_h has (both lower and upper) bandwidth of $b = N - 1$, no effort is required for eliminating the rows below $k + b$. Combining these two features, the complexity of the whole process is reduced from $O(N^6)$ to $O(N^4)$ (note that the matrix is of size $O(N^2)$). Detailed implementation of LU decomposition is shown in Algorithm 1, and this band property can also be exploited when solving triangular systems.

Algorithm 1 LU decomposition for banded matrices

```
Input: A square matrix $A \in \mathbb{R}^{n\times n}$ with bandwidth $band$
Output: LU decomposition $A = LU$
  for $k = 1 : n - 1$ do
      $m = \min(k + band, n)$
      $A(k + 1 : m, k) = A(k + 1 : m, k)/A(k, k)$
      $A(k + 1 : m, k + 1 : m) = A(k + 1 : m, k + 1 : m) - A(k + 1 : m, k) \cdot A(k, k + 1 : m)$
  $L = \text{speye}(n) + \text{tril}(A, -1)$
  $U = \text{triu}(A)$
```

2.2 Iterative methods

Classical iterative schemes are based on a fixed point iteration for solving linear systems. If we decompose $A = M - N$, then the equation $Au = f$ is equivalent to $Mu = Nu + f$, leading to a general iterative approach

$$u^{(k+1)} = M^{-1}Nu^{(k)} + M^{-1}f = u^{(k)} + M^{-1}r^{(k)},$$

where $r^{(k)} = f - Au^{(k)}$ is the residual. Separate $A = L + D + U$ into lower triangular, diagonal and upper triangular parts, then $M = D$ for the Jacobi method, $M = D + L$ for the Gauss-Seidel method, and $M = \omega^{-1}D + L$ for the SOR method.

These methods can also be written in a component-wise fashion, for example, Gauss-Seidel method iterates by

$$u_i^{(k+1)} = a_{ii}^{-1}\left(f_i - \sum_{j=1}^{i-1} a_{ij}u_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}u_j^{(k)}\right), \quad i = 1, 2, \cdots, n.$$

For our particular problem, note that $A_h = [A_{ij}]$ is a symmetric block tridiagonal matrix, we can take advantage of its special properties by applying symmetric line Gauss-Seidel iterative scheme, which performs the update

$$u_i^{(k+1)} = A_{ii}^{-1}\left(f_i - \sum_{j=1}^{i-1} A_{ij}u_j^{(k+1)} - \sum_{j=i+1}^{n} A_{ij}u_j^{(k)}\right)$$

for each line $u_i \in \mathbb{R}^{N-1}$, first in the increasing order of index i ($M = D + L$) followed by the decreasing order ($M = D + U$). In this way, one iteration updates twice the whole vector u_h, and thus the number of iterations needed for convergence is cut down. On the computational complexity, however, the line Gauss-Seidel method does not actually give noticeable improvement on this problem.

2.3 Conjugate gradient method

Consider the equation $Ax = b$ where $A \in \mathbb{S}_+^n$ (the set of all symmetric positive definite matrices). The solution x_* is the unique minimizer of the quadratic function

$$\phi(x) = \frac{1}{2}x^T Ax - b^T x.$$

The algorithm searches for the global minimum of $\phi(x)$ by choosing a best descent direction p_k in the subspace spanned by the previous (negative) gradient r_{k-1} and previous direction p_{k-1} at time step k, and it updates x_k by going along the direction p_k by a factor of $\alpha_k = (r_k, p_k)/(p_k, p_k)_A$. The residuals $\{r_k\}$ are conjugate and the directions $\{p_k\}$ are A-conjugate, which exactly form a basis of the Krylov subspace

$$\mathcal{K}(A, r_0, k) = \text{span}\{r_0, Ar_0, \cdots, A^{k-1}r_0\}.$$

In the affine subspace $x_0 + \mathcal{K}(A, r_0, k)$, the vector x_k is the closest to x_* in the sense of A-norm. Therefore, the conjugate gradient method is able to produce the accurate solution within n iterations in theory. It can be used either as a direct method or as an iterative method.

Preconditioning is often necessary to ensure fast convergence of the conjugate gradient method. If we decide on $M \in \mathbb{S}^n_+$ as a preconditioner, then there exists a unique $C \in \mathbb{S}^n_+$ such that $M = C^2$. Let

$$\tilde{A} = C^{-1}AC^{-1}, \quad \tilde{x} = Cx, \quad \tilde{b} = C^{-1}b,$$

then the original equation is equivalent to the "tilde problem" $\tilde{A}\tilde{x} = \tilde{b}$. Applying the conjugate gradient method to this new equation gives the procedure described in Algorithm 2. The preconditioner M should capture the essence of A to make $\tilde{A}$ look like identity matrix I, otherwise the overhead spent on dealing with M can exceed the cost saved from less number of iterations.

Algorithm 2 Preconditioned conjugate gradient method

```
Input: An equation $Ax = b$ where $A \in \mathbb{S}^n_+$
Output: Solution $x_*$
  Initialize $k = 0,\ r_0 = b - Ax_0,\ p_0 = 0$
  Solve $Mz_0 = r_0$
  while not converged do
     $k = k + 1$
     $\beta_k = \frac{r_{k-1}^T z_{k-1}}{r_{k-2}^T z_{k-2}}$
     $p_k = z_{k-1} + \beta_k p_{k-1}$
     $\alpha_k = \frac{r_{k-1}^T z_{k-1}}{p_k^T A p_k}$
     $x_k = x_{k-1} + \alpha_k p_k$
     $r_k = r_{k-1} - \alpha_k A p_k$
     Solve $Mz_k = r_k$
  return $x_k$
```

2.4 Discrete sine transform

Discrete sine transform (DST) bears resemblance to fast Fourier transform (FFT), and is widely employed in solving partial differential equations. To apply this method, note that the discrete Poisson equation (*) can also be written in a matrix form:

$$T_{N-1}U + \epsilon\, UT_{N-1} = F,$$

where $T_{N-1} = \text{tridiag}(-1, 2, -1)$, $U = [u_{ij}]$ and $F = [h^2 f_{ij}] \in \mathbb{R}^{N-1 \times N-1}$. The eigenvalues of T_{N-1} are given by

$$\lambda_j = 4 \sin^2 \left(\frac{j\pi}{2N} \right), \quad 1 \leq j \leq N-1,$$

and thus T_{N-1} has eigendecomposition

$$T_{N-1} = VDV^{-1}, \quad D = \text{diag}(\lambda_1, \cdots, \lambda_{N-1}),$$

where $V_{kj} = \sin(\frac{kj\pi}{N})$ is exactly the DST matrix and $V^{-1} = \frac{2}{N}V$ is the inverse DST matrix. By substituting this decomposition into the original equation, we obtain

$$D\tilde{U} + \epsilon\, \tilde{U}D = \tilde{F},$$

where $\tilde{U} = V^{-1}UV$ and $\tilde{F} = V^{-1}FV$. Since D is diagonal, this matrix equation is extremely easy to solve:

$$\tilde{u}_{ij} = \frac{\tilde{f}_{ij}}{\lambda_i + \epsilon\lambda_j}.$$

Using the DST technique, all these operations can be done within $O(N^2 \log N)$ time with desirable numerical properties. The whole process is described in Algorithm 3.

Algorithm 3 Discrete sine transform method

```
Input: Grid size $N$
Output: Solution $U$
  $\tilde{F} = V^{-1}(VF)^T$ where $F_{ij} = h^2 f(ih, jh)$
  $d = [\lambda_1, \cdots, \lambda_{N-1}]^T$ where $\lambda_j = 4\sin^2\left(\frac{j\pi}{2N}\right)$
  $\tilde{U} = \tilde{F}/(d\mathbf{1}^T + \epsilon\mathbf{1}d^T)$ (element-wise division)
  $U = V(V^{-1}\tilde{U}^T)^T$
```

3 Multigrid method

3.1 Motivation

For a detailed investigation into classical iterations, consider the one-dimensional discrete Poisson equation $Au = 0$ and an initial iterate

$$u^{(0)} = \left(\sin\frac{k\pi}{N}, \sin\frac{2k\pi}{N}, \cdots, \sin\frac{(N-1)k\pi}{N} \right)^T,$$

with varying k representing different frequencies. The lower part of the spectrum $(1 \leq k < N/2)$ are called low frequencies or smooth modes, while the upper part $(N/2 \leq k < N)$ are called high frequencies or oscillating modes. Application of a typical iterative method (e.g. Gauss-Seidel method) gives the following important observations:

- On a fixed grid size N, there is an effective damping of the high frequency errors whereas almost no damping occurs to the low frequency errors.
- For a fixed wave number k, the error is reduced better on a coarser grid than on a finer grid.

These properties motivate the idea of multigrid method. On a fixed fine grid, higher frequency errors can be efficiently smoothed out by the iterative method. In order to reduce those lower frequency errors, it is reasonable to restrict to a coarser grid, where the previously smooth modes become more oscillating, and therefore can again be damped by iterative schemes. Later the error is prolonged back to the fine grid for further smoothing, and this process loops until convergence.

3.2 Algorithm in detail

Denote by Ω^h the uniform rectangular grid on Ω with stride h. Given an iterative smoother R to solve a two-dimensional discrete Poisson equation $A_h u_h = f_h$ on Ω^h, we first perform the iteration v_1 times to damp the oscillating modes. Next, the residual $r_h = f_h - A_h u_h^{(v_1)}$ is restricted to a coarser grid Ω^{2h} by a restriction operator I_h^{2h} for further smoothing. Restriction is done by

$$u_{i,j}^{2h} = \frac{1}{16}\left[u_{2i\pm1,2j\pm1}^{h} + 2\left(u_{2i\pm1,2j}^{h} + u_{2i,2j\pm1}^{h}\right) + 4u_{2i,2j}^{h}\right],$$

thus the restriction operator can be expressed as $I_h^{2h} = (J_h \otimes J_h)/16$, where

$$J_h = \begin{bmatrix} 1 & 2 & 1 & & & & \\ & & 1 & 2 & 1 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & 1 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{N/2-1\times N-1}.$$

The coarser grid damps those error modes which turn to be more oscillating, but is not able to tackle the ones which are still fairly smooth. Hence the restriction process is recursively called to successively switch to coarser and coarser grids, and stops at a grid coarse enough for the equation to be directly solved. Then the error should be prolonged back to the finer grids by a prolongation operator I_{2h}^{h}. Prolongation is done by

$$\begin{aligned} &u_{2i,2j}^{h} = u_{i,j}^{2h}, \quad u_{2i+1,2j}^{h} = \frac{1}{2}(u_{i,j}^{2h} + u_{i+1,j}^{2h}), \quad u_{2i,2j+1}^{h} = \frac{1}{2}(u_{i,j}^{2h} + u_{i,j+1}^{2h}), \\ &u_{2i+1,2j+1}^{h} = \frac{1}{4}(u_{i,j}^{2h} + u_{i+1,j}^{2h} + u_{i,j+1}^{2h} + u_{i+1,j+1}^{2h}), \end{aligned}$$

thus the prolongation operator can be expressed as $I_{2h}^{h} = (J_h \otimes J_h)^T/4 = 4(I_h^{2h})^T$. After correct the iterate $u_h^{(v_1)}$ with the prolonged solution, several steps of post-smoothing continues to damp the error, completing the overall procedure. Algorithm 4 concludes the description above, and this process is usually referred to as a V-cycle.

Algorithm 4 Multigrid scheme: V-cycle

```
Input: An equation $A_h u_h = f_h$ on $\Omega^h$
Output: Solution $u_h$
  if grid is sufficiently coarse then
      Solve $A_h u_h = f_h$ directly, return ;
  Pre-smoothing: $u_h^{(k)} = R(u_h^{(k-1)})$, $1 \leq k \leq v_1$;
  Residual: $r_h = f_h - A_h u_h^{(v_1)}$
  Restriction: $r_{2h} = I_h^{2h} r_h$
  Solve $A_{2h} u_{2h} = r_{2h}$ by recursion
  Prolongation and correction: $u_h^{(v_1+1)} = u_h^{(v_1)} + I_{2h}^{h} u_{2h}$
  Post-smoothing: $u_h^{(k)} = R(u_h^{(k-1)})$, $v_1 + 2 \leq k \leq v_1 + v_2 + 1$;
```

4 Experiment results

4.1 Direct solution with Gaussian elimination

We first solve the equation directly with Gaussian elimination. Fix the grid size $N = 64$ and select $\epsilon = 1, 10^{-1}, 10^{-3}, 10^{-5}$. We run two different implementations on the equation: with or without utilizing the band property of the matrix, and the results are shown in Table 1.

To estimate the error of our numerical solution $\tilde{u}_h$, we use piecewise constant function to extend its domain from discrete grid points to the whole region Ω, and correspondingly define a discrete l^2 norm

$$\|u - \tilde{u}_h\|_{l^2} = \left(\int_\Omega |u - \tilde{u}_h|^2 \mathrm{d}x \mathrm{d}y \right)^{1/2} = \left(\sum_{i,j=1}^{N-1} h^2 |u - \tilde{u}_h|_{ij}^2 \right)^{1/2} = h\|u - \tilde{u}_h\|_2$$

to measure the global error. We have also estimated the forward error of computation, namely the error between the true solution u_h and the numerical solution $\tilde{u}_h$ to the discrete equation. Displayed in Table 1 are exactly the global error and the forward error.

As can be seen from Table 1, the global error for different parameters are basically the same, since the grid size sets the upper limit of the accuracy of the discretization process. The forward error is of fairly small magnitude, implying that the diagonal dominance makes A_h a well-conditioned matrix, and also stabilizes the solving routine. Performance improvements when taking advantage of the

Table 1: Numerical results of Gaussian elimination

ϵ	Banded			Non-banded
	$\|u - \tilde{u}_h\|_{l^2}$	$\|u_h - \tilde{u}_h\|_2$	Avg. Time	Avg. Time
1	1.0041e-04	1.3771e-12	10.6134 sec	12.1842 sec
10^{-3}	1.0041e-04	1.5040e-12	10.5733 sec	12.5042 sec
10^{-3}	1.0041e-04	2.6208e-12	10.1394 sec	12.4213 sec
10^{-5}	1.0041e-04	1.7012e-12	10.7254 sec	12.3975 sec

Table 2: Runtime of CG and PCG methods (in seconds)

	$N = 32$		$N = 64$		$N = 128$	
	CG	PCG	CG	PCG	CG	PCG
$\epsilon = 10^{-3}$	0.0015	0.1962	0.0143	1.2270	0.0254	10.3213
$\epsilon = 10^{-5}$	0.0032	0.1671	0.0100	0.9876	0.0377	12.9569

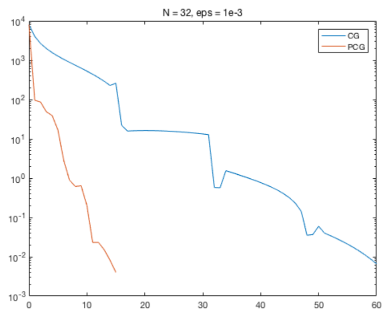
band property can be seen from the difference of average runtime — banded Gaussian elimination saves about 16.7% of time than non-banded one on the $63^2 \times 63^2$ equation. Such disparity will be more notable when N grows larger.

4.2 Conjugate gradient method

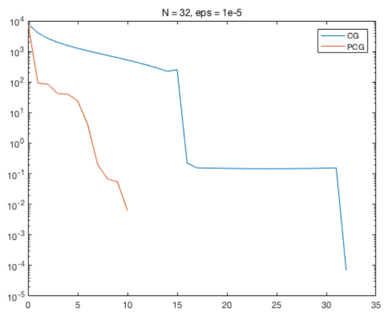
In this section, we will compare the conjugate gradient (CG) method with preconditioned conjugate gradient (PCG) method to see how precondition can help with faster convergence. We use multigrid V-cycle scheme as the preconditioner (see the next section for detailed settings). Runtime for different grid sizes and diffusion parameters are shown in Table 2 (measured in seconds). It can be seen that PCG actually takes more time than CG to achieve a desirable result, which is largely due to the high computational cost of multigrid scheme. Therefore, the tradeoff between the extra cost on the preconditioner and the number of iterations has to be considered carefully when choosing a proper M for precondition.

Figure 1 shows the advantage of the preconditioner — the four panels each depict the relationship between the residual $r_k = f - Au_k$ (semilog) and the iteration index k for different parameters. Figure 1 strongly proves that precondition can greatly help with cutting down the number of iterations needed for convergence, and this improvement becomes much more significant when N grows larger.

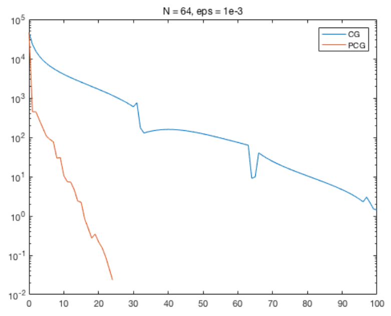
One interesting phenomenon about the plot is, the residual curve of CG method seems to thrash every $N/2$ iterations, and this thrash may or may not help reduce the error. In panel (d), we set the maximum number of iterations to 100, which is below the "thrash period" $N/2 = 128$, hence the



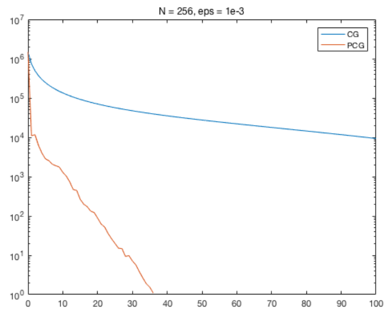
(a) $N = 32,\ \epsilon = 10^{-3}$



(b) $N = 32,\ \epsilon = 10^{-5}$



(c) $N = 64,\ \epsilon = 10^{-3}$



(d) $N = 256,\ \epsilon = 10^{-3}$

Figure 1: Convergence of CG and PCG methods

residual curve shows better smoothness than others. Besides, the thrash is far more salient when ϵ gets smaller as is shown in panel (b). More investigation is needed on this weird phenomenon.

4.3 Multigrid preconditioned conjugate gradient method

We now try the multigrid preconditioned conjugate gradient method (MGCG) on the equation $A_h u_h = f_h$, i.e. solving the residual equation $A_h z_k = r_k$ in Algorithm 2 with multigrid V-cycle scheme. Iteration starts at $u^{(0)} = (1, 1, \cdots, 1)^T$. The procedure terminates when the current residual r satisfies $\|r\|_2/\|r_0\|_2 < 10^{-6}$, where r_0 is the initial residual. Multigrid scheme adopts the symmetric line Gauss-Seidel method as the smoother, performs only one pre-smoothing step as well as one post-smoothing step ($v_1 = v_2 = 1$), and solves the equation directly when $N \leq 4$.

Results of MGCG method in different settings, including CPU runtime and the number of iterations, are displayed in Table 3. As the matrix size grows larger and larger, the time spent on solving the equation soon becomes unbearably long. The effect of parameter ϵ is non-negligible: smaller ϵ can make the system more singular and the problem more difficult to deal with, but this impact is not monotonically increasing as ϵ tends to 0. A conjecture could be: there exists an ϵ_* which makes

Table 3: Numerical results of MGCG (row-major order)

	$\epsilon = 1$		$\epsilon = 10^{-1}$		$\epsilon = 10^{-3}$		$\epsilon = 10^{-5}$		$\epsilon = 10^{-7}$	
	Time	Iters.	Time	Iters.	Time	Iters.	Time	Iters.	Time	Iters.
$N = 32$	0.050	4	0.076	7	0.224	22	0.159	17	0.138	15
$N = 64$	0.175	4	0.308	8	1.525	34	1.134	29	1.005	25
$N = 128$	1.064	4	2.075	8	10.39	47	12.11	56	8.575	38
$N = 256$	9.931	4	16.94	8	110.1	51	203.2	99	131.3	66
$N = 512$	94.62	4	150.9	7	993.1	53	3392.	182	2416.	125
$N = 1024$	776.8	4	1296.	7						

Table 4: Numerical results of MGCG (column-major order)

	$\epsilon = 1$		$\epsilon = 10^{-1}$		$\epsilon = 10^{-3}$		$\epsilon = 10^{-5}$		$\epsilon = 10^{-7}$	
	Time	Iters.	Time	Iters.	Time	Iters.	Time	Iters.	Time	Iters.
$N = 32$	0.076	4	0.048	3	0.021	1	0.029	1	0.025	1
$N = 64$	0.173	4	0.143	3	0.109	2	0.071	1	0.072	1
$N = 128$	1.007	4	0.816	3	0.584	2	0.398	1	0.402	1
$N = 256$	9.295	4	7.456	3	5.773	2	3.977	1	3.938	1
$N = 512$	86.86	4	71.69	3	57.24	2	41.78	1	37.74	1
$N = 1024$	766.5	4	649.6	3	480.4	2	318.6	1	319.2	1

the problem the hardest, and this ϵ_* gets smaller when N goes larger.

On the other hand, if we rearrange the discrete grid points in a column-major order, i.e. let $u_i = (u_{1,i}, u_{2,i}, \cdots, u_{N-1,i})^T$ and $f_i = (f_{1,i}, f_{2,i}, \cdots, f_{N-1,i})^T$, then the equation can be similarly formulated as $B_h u_h = f_h$ where

$$B_h = \epsilon\, T_{N-1} \otimes I_{N-1} + I_{N-1} \otimes T_{N-1},$$

and $T_{N-1} = \text{tridiag}(-1, 2, -1)/h^2$. The parameter settings are repeated for this new equation, and the corresponding results are shown in Table 4. Contrary to what we have seen from Table 3, the smaller the ϵ, the less iterations are needed for convergence — even in some cases, simply iterating once is enough for the desired numerical precision. But note that the time needed for each iteration is increased for small ϵ's despite the reduced total time cost, while this is not the case for the row-major ordered equation.

The discrete l^2 error $\|u - \tilde{u}_h\|_{l^2}$ is sketched in Figure 2. We compute the error for N from 2^5 to 2^{10} and $\epsilon = 1, 10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}$. From the figure, we can observe more carefully the effect of ϵ. For coarser grids, ϵ does not have a noticeable impact, but finer grids bring to light completely the singularity of the system caused by ϵ.

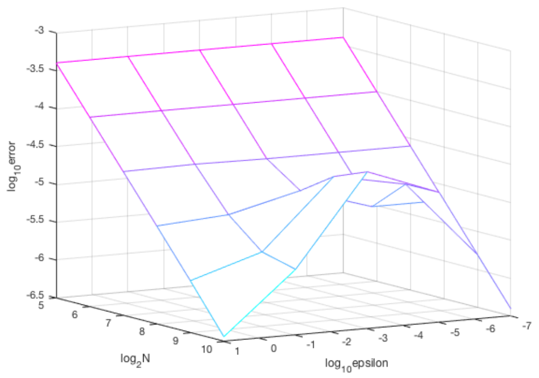


Figure 2: Discrete l^2 error of MGCG with respect to grid size and ϵ

Table 5: Comparison of using line GS and point GS as smoothers for MGCG

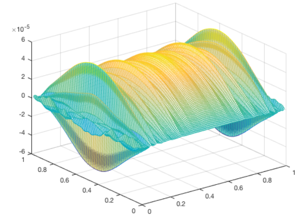
		$\epsilon = 10^{-1}$		$\epsilon = 10^{-3}$		$\epsilon = 10^{-5}$		$\epsilon = 10^{-7}$	
		Time	Iters.	Time	Iters.	Time	Iters.	Time	Iters.
$N = 128$	Line GS	2.075	8	10.39	47	12.11	56	8.575	38
	Point GS	9.870	11	60.56	69	72.36	90	56.11	73
$N = 256$	Line GS	16.94	8	110.1	51	203.2	99	131.3	66
	Point GS	37.54	11	264.4	81	605.7	177	464.7	142

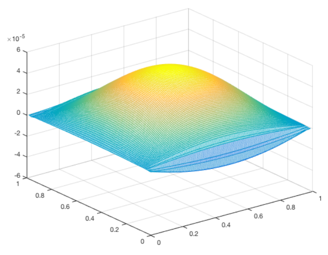
We have also changed the smoother for the multigrid from line Gauss-Seidel method to the common point Gauss-Seidel method to compare their performance, as is shown in Table 5. Line GS method saves about half of the iterations, and thus surpasses the point GS method greatly on time consumption. The combination of $N = 256$ and $\epsilon = 10^{-5}$ is the most difficult among the options in the table whether using line GS or point GS, which serves as another evidence of the ill-condition of the equation under this setting.

4.4 Discrete sine transform method

Simply out of curiosity, we also implement the DST-based method to compare with the MGCG method. Fix $N = 128$ and $\epsilon = 10^{-3}$, three different routines are applied to solve the discrete Poisson equation, and the error surface is plotted for each routine in Figure 3.

Surface plot gives us more detailed information about the numerical schemes. MGCG method solving row-major order equation produces a rather rough surface and has large variance on the y boundary. In contrast, MGCG solving column-major order equation enjoys a better condition, producing a smoother surface and less variance on the x boundary. DST method has superiority


(a) MGCG (row-major)


(b) MGCG (column-major)

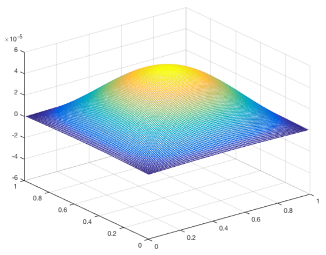

(c) DST method

Figure 3: Error surface of different methods ($N = 128,\ \epsilon = 10^{-3}$)

over MGCG method in the sense of both efficiency and stability. It takes only 0.012 seconds to finish the solving procedure, much less than 10.392 seconds for row-major MGCG and 0.584 seconds for column-major MGCG. Moreover, it achieves an equally accurate result as MGCG method but is incredibly stable on the boundary. Therefore, it is reasonable to adopt DST method for this particular problem, but its scope of application is much more limited than MGCG method.

5 Conclusion

For this project, we explore a total of five numerical methods to solve the two-dimensional Poisson equation with homogeneous Dirichlet boundary conditions, including Gaussian elimination, classical iterations, (preconditioned) conjugate gradient method, multigrid method and DST-based method. We describe these methods in detail and implement all of them for solving the equation. A series of comparisons between these methods are presented to highlight their advantages and disadvantages. Apart from a delicate design of the numerical algorithm, a proper formulation of the problem also helps to ensure fast convergence and desirable precision of the solution.

References

[1] G. Golub and C. V. Loan. Matrix Computations, 4th Edition. The Johns Hopkins University Press. 1983.

[2] V. John. Multigrid Methods. 2013. https://www.wias-berlin.de/people/john/LEHRE/MULTIGRID/multigrid.pdf.