Application and Comparison of Different Numerical Methods for One-dimensional Poisson's Equation

Tong Li[1]

[1]Department of Physics and Astronomy, Michigan State University

Abstract

Poisson's equation is an important type of differential equations in science and engineering, and thus it is valuable to study numerical methods for it. In this report we numerically solve a one-dimensional Poisson's equation (with Dirichlet boundary conditions) on an equidistant grid by converting it into a set of linear equations $\mathbf{A}\mathbf{v} = \bar{\mathbf{f}}$ where $\mathbf{A}$ is a tri-diagonal matrix. Three numerical methods are discussed and compared in this report: Gaussian elimination for a general tri-diagonal matrix (M1), a special method that makes use of the fact that matrix $\mathbf{A}$ has identical elements along the diagonal and identical (but different) values for the non-diagonal elements (M2), and LU decomposition (M3). The relative errors of these methods are consistent with the approximation error introduced in discretization, and M2 is the most efficient algorithm as it uses the property of this problem to do precalculation.

1 Introduction

Poisson's equation is one important type of differential equations in science and engineering, especially in electromagnetism. Poisson's equation has the following form:

$$\nabla^2 u\left(\vec{r}\right) = f\left(\vec{r}\right) \ , \tag{1}$$

where $f\left(\vec{r}\right)$ is known and $u\left(\vec{r}\right)$ is the function to be solved. In electrostatics, $u\left(\vec{r}\right)$ is the electric potential and $f\left(\vec{r}\right) = -4\pi\rho\left(\vec{r}\right)$ where $\rho\left(\vec{r}\right)$ is charge density. In physics, Poisson's equation usually comes with different kinds of boundary value conditions, such as Dirichlet (values of $u\left(\vec{r}\right)$ are specified on boundaries), or Neumann (normal derivatives $\frac{\partial u}{\partial n}$ are specified on boundaries) boundary conditions.

Poisson's equation can be analytically solved by separation of variables and eigenfunction method, which usually gives $u\left(\vec{r}\right)$ as an infinite series. However, when the inhomogeneous term $f(x)$ or boundary conditions become very complicated, it is impossible to find an analytical solution. Therefore, it is important to study the numerical methods to solve Poisson's equations.

For simplicity, we will focus on one-dimensional Poisson's equation with Dirichlet boundary conditions in this report. In section 2 we will describe the three numerical methods used to solve the Poisson's equation. Section 3 will discuss the numerical error and speed of these methods. Conclusions and perspective will be given in section 4.

2 Numerical methods

The one-dimentional Poisson's equation can be written as

$$-u''(x) = f(x) \ . \tag{2}$$

This equation will be solved in the region $x \in [0, 1]$, with boundary condition $u(0) = u(1) = 0$.

First, the discretized approximation to function u is defined as v_i on $(n+1)$ equidistant points $x_i \in [0, 1]$, where $x_0 = 0$ and $x_n = 1$. Thus, the step length is $h = 1/n$ and $x_i = ih$.

By Taylor expansion, we have

$$u(x+h) = u(x) + u'(x)h + \frac{1}{2}u''(x)h^2 + O(h^2) \ , \tag{3}$$

$$u(x-h) = u(x) - u'(x)h + \frac{1}{2}u''(x)h^2 + O(h^2) \ . \tag{4}$$

By adding the two equations above, we can approximate the second derivative as

$$u''(x_i) = \frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} + O(h^2) \tag{5}$$

where the approximation error is $O(h^2)$.

So the problem can be converted to a set of linear equations

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i \qquad \text{for } i = 1, 2, \ldots, n-1 \tag{6}$$

where $f_i = f(x_i)$. This set of equations can be rewritten as

$$\mathbf{A}\mathbf{v} = \overline{\mathbf{f}} \ , \tag{7}$$

where $\mathbf{A}$ is a $(n-1) \times (n-1)$ tridiagonal matrix

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & \ldots & \ldots & 0 \\ -1 & 2 & -1 & 0 & \ldots & \ldots \\ 0 & -1 & 2 & -1 & 0 & \ldots \\ & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & & -1 & 2 & -1 \\ 0 & \ldots & & 0 & -1 & 2 \end{bmatrix} \tag{8}$$

and $\overline{f}_i = h^2 f_i$. Now we will discuss the three numerical methods to solve Eq. 7.

2.1 Gaussian elimination for a general tri-diagonal matrix

In this subsection we will solve $\mathbf{Av} = \overline{\mathbf{f}}$ for any tri-diagonal matrix $\mathbf{A}$ using Gaussian elimination. The equation is written as

$$\mathbf{Au} = \begin{bmatrix} b_1 & c_1 & 0 & \dots & \dots & 0 \\ a_2 & b_2 & c_2 & 0 & \dots & \dots \\ 0 & a_3 & b_3 & c_3 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & a_{n-2} & b_{n-2} & c_{n-2} \\ 0 & \dots & & 0 & b_{n-1} & c_{n-1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} \overline{f}_1 \\ \overline{f}_2 \\ \overline{f}_3 \\ \dots \\ \overline{f}_{n-2} \\ \overline{f}_{n-1} \end{bmatrix} = \overline{\mathbf{f}} \ . \qquad (9)$$

In the i-th step of Gaussian elimination, the product of $-\frac{a_i}{\tilde{b}_{i-1}}$ (where $\tilde{b}_{i-1}$ is the new diagonal element obtained in the previous step) and row i is added to row $(i+1)$, giving a new diagonal element

$$\tilde{b}_i = b_i - \frac{a_i c_{i-1}}{\tilde{b}_{i-1}} \quad i = 2, 3, \dots, n-1 \qquad (10)$$

and a new right-hand side

$$\tilde{f}_i = \overline{f}_i - \frac{a_i \tilde{f}_{i-1}}{\tilde{b}_{i-1}} \quad i = 2, 3, \dots, n-1 \ . \qquad (11)$$

And we always have $\tilde{b}_1 = b_1$ and $\tilde{f}_1 = \overline{f}_1$. Then the equation becomes

$$\tilde{\mathbf{A}}\mathbf{u} = \begin{bmatrix} \tilde{b}_1 & c_1 & 0 & \dots & \dots & 0 \\ 0 & \tilde{b}_2 & c_2 & 0 & \dots & \dots \\ 0 & 0 & \tilde{b}_3 & c_3 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & 0 & \tilde{b}_{n-2} & c_{n-2} \\ 0 & \dots & & 0 & \tilde{b}_{n-1} & c_{n-1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} \tilde{f}_1 \\ \tilde{f}_2 \\ \tilde{f}_3 \\ \dots \\ \tilde{f}_{n-2} \\ \tilde{f}_{n-1} \end{bmatrix} = \tilde{\mathbf{f}} \qquad (12)$$

which can be solved by backward substitution

$$\begin{aligned} u_{n-1} &= \frac{\tilde{f}_{n-1}}{\tilde{b}_{n-1}} \\ u_i &= \frac{\tilde{f}_i - c_i u_{i+1}}{\tilde{b}_i} \quad i = n-2, n-3, \dots, 1 \ . \end{aligned} \qquad (13)$$

This algorithm needs $[8(n-2)+1]$ floating-point operations (FLOPs), and its time complexity is $O(n)$.

2.2 A special method for this problem

By Eq. 8 and 9, we can see that $a_i = c_i = -1$ and $b_i = 2$ for any i. Thus, Eq. 10 becomes

$$\tilde{b}_i = 2 - \frac{1}{\tilde{b}_{i-1}} \qquad (14)$$

which yields

$$\tilde{b}_i = \frac{i+1}{i} \ . \tag{15}$$

Therefore, $\tilde{\mathbf{b}}$ can be calculated beforehand, which eliminates redundant calculations when we change function $f(x)$.

Eq. 11 and 15 yield

$$\tilde{f}_i = \overline{f}_i + \frac{(i-1)\tilde{f}_{i-1}}{i} \ . \tag{16}$$

and then the backward substitution (Eq. 13) can be written as

$$\begin{aligned} u_{n-1} &= \frac{n-1}{n}\tilde{f}_{n-1} \\ u_i &= \frac{i}{i+1}\left(\tilde{f}_i + u_{i+1}\right) \quad i = n-2, n-3, \ldots, 1 \ . \end{aligned} \tag{17}$$

Since $\tilde{\mathbf{b}}$ can be precalcuated, we do not consider Eq. 15 when counting FLOPs. So this algorithm needs $[4(n-2)+1]$ FLOPs, and its time complexity is also $O(n)$.

2.3 LU decomposition

For a general (not necessarily tri-diagonal) matrix $\mathbf{A}$, equation $\mathbf{Av} = \overline{\mathbf{f}}$ is usually solved by LU decomposition. First, matrix $\mathbf{A}$ is decomposed as $\mathbf{A} = \mathbf{LU}$, where $\mathbf{L}$ and $\mathbf{U}$ are upper- and lower-triangular matrices, respectively. Then the equation $\mathbf{Av} = \overline{\mathbf{f}}$ becomes

$$\mathbf{Lw} = \overline{\mathbf{f}} \tag{18}$$

$$\mathbf{Uv} = \mathbf{w} \tag{19}$$

Eq. 18 and 19 can be easily solved by forward and backward substitution, respectively. The time complexity of LU decomposition is $O(n^3)$, which is much larger than the methods discussed in section 2.1 and 2.2. In this work we use the LU decomposition algorithm in Armadillo (a C++ linear algebra library).

3 Results and discussion

In this section we will discuss the results of the three numerical methods described in section 2. The three methods discussed in section 2.1, 2.2 and 2.3 are denoted as M1, M2 and M3, respectively. For benchmark, we choose $f(x) = 100e^{-10x}$, which gives an analytical solution of Eq. 2:

$$u(x) = 1 - \left(1 - e^{-10}\right)x - e^{-10x} \ . \tag{20}$$

The relative error at point x_i is defined as

$$\epsilon_i = \left|\frac{v_i - u(x_i)}{u(x_i)}\right| \ . \tag{21}$$

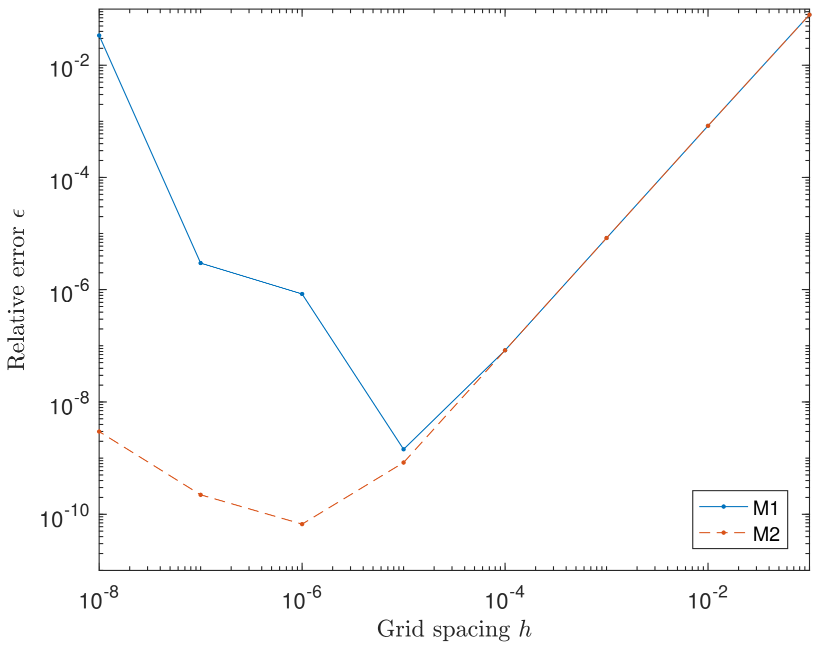

Figure 1: Maximum relative errors of M1 and M2 for different grid spacing h.

The maximum relative errors of methods M1 and M2 for different grid spacing h are shown in Figure 1. These errors are small enough to confirm the correctness of our program. In the region of $h = 10^{-1} \sim 10^{-4}$, M1 and M2 have the same relative errors ϵ. The logarithm of ϵ decreases linearly as $\log h$ decreases, with a slope of 2. So in this region the relative error is $O(h^2)$, the same as the approximation error given in Eq. 5. When $h < 10^{-4}$, the relative error begins to fall down slowly and even increase as h decreases, which indicates that the round-off error becomes important for very small h. In Eq. 5, both $(v_{i+1}-v_i)$ and $(v_{i-1}-v_i)$ give an $\epsilon_M = O(10^{-15})$ round-off error (for double precision), and thus the round-off error should be $O(\frac{10^{-15}}{h^2})$ for $u''(x_i)$. Therefore, when $h = 10^{-4} \sim 10^{-5}$, $O(\frac{10^{-15}}{h^2}) \approx O(h^2)$, i.e. the round-off and approximation error become comparable. So we should not use grid spacing $h < 10^{-4}$ when the round-off error contribute significantly to our loss of precision.

Because of memory and time limitation, M3 cannot go to $h < 10^{-3}$. For $h \geq 10^{-3}$, M3 gives the same relative errors as M1 and M2, which are not shown in Figure 1.

Table 1 shows the time used by algorithms M1 and M2 for different number of grid points n, with no optimization flag applied. In order to measure the time accurately, n is chosen to be large enough (10^6, 10^7 and 10^8), and each value given in Table 1 is the average over three measurements. For M2, the

Table 1: Time used by M1 and M2 for different number of grid points n. Each value given in this table is the average over three measurements. For M2 the time of precalculating Eq. 15 is not included.

n	h	Time (sec)	
		M1	M2
10^6	10^{-6}	0.0469	0.0313
10^7	10^{-7}	0.3698	0.1875
10^8	10^{-8}	7.0729	2.4323

time of precalculating Eq. 15 is not included. As n increases, the time of both M1 and M2 rises almost linearly, corresponding to their O(n) time complexity. In addition, M2 is much faster than M1 because Eq. 15 is precalculated in M2. For the same n, M2's FLOPs is about half of M1's, and the time of M2 is also roughly half of M1's. Therefore, from the perspective of efficiency, M2 is the best option to solve a one-dimensional Poisson's equation with Dirichlet boundary conditions, especially when we want to change the inhomogeneous term $f(x)$ many times.

4 Conclusions and outlook

In this report we discuss three numerical methods to solve a one-dimensional Poisson's equation with Dirichlet boundary conditions. First, this problem is discretized on equidistant grid points with spacing h, and converted into a set of linear equations $\mathbf{A}\mathbf{v} = \overline{\mathbf{f}}$ where $\mathbf{A}$ is a tri-diagonal matrix. Then we describe three numerical methods to solve this equation: Gaussian elimination for a general tri-diagonal matrix (M1, section 2.1), a special method that makes use of the fact that matrix $\mathbf{A}$ has identical elements along the diagonal and identical (but different) values for the non-diagonal elements (M2, section 2.2), and LU decomposition (M3, section 2.3).

In section 3 we apply the three methods to solve $-u''(x) = 100e^{-10x}$ whose analytical solution is known. The small relative errors extracted from our program confirm the correctness of these algorithms. When $h = 10^{-1} \sim 10^{-4}$ the relative error $\epsilon \sim h^2$, which is consistent with the approximation error of the second derivative (Eq. 5). When $h < 10^{-4}$ round-off error becomes more and more dominant as h decreases. Thus the optimal grid spacing h is 10^{-4}. Comparing the time used by these methods, we find that M2 is the most efficient one because it fully utilizes the property of this problem to precalculate. The advantage of M2 will be more significant if we want to change the inhomogeneous term many times.

Although the Poisson's equation we usually see is two- or three-dimensional, the algorithms for one-dimensional Poisson's equation are still useful because many high-dimensional problems can be reduced to one dimension by methods like separation of variables. In addition, this project provides numerical algorithms to solve linear equation $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is a tri-diagonal matrix. This type of problem often appears in science and engineering, so it is important to understand all the numerical methods discussed in this report.

Acknowledgments

I am grateful for the sincere guidance from Prof. Morten Hjorth-Jensen, and I also want to thank Mengzhi Chen for helpful discussions.

References

[1] Morten Hjorth-Jensen. Computational physics lectures: Introduction to programming (C++ and Fortran). `https://compphysics.github.io/ComputationalPhysicsMSU/doc/pub/languages/html/languages.html`. Accessed Feb 5, 2018.

[2] Morten Hjorth-Jensen. Computational physics lectures: Linear algebra methods. `https://compphysics.github.io/ComputationalPhysicsMSU/doc/pub/linalg/html/linalg.html`. Accessed Feb 5, 2018.

[3] Kenneth Franklin Riley, Michael Paul Hobson, and Stephen John Bence. *Mathematical methods for physics and engineering: a comprehensive guide*. Cambridge University Press, 2006.

[4] Conrad Sanderson and Ryan Curtin. Armadillo: a template-based C++ library for linear algebra. *Journal of Open Source Software*, 2016.

[5] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*. Siam, 1997.