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

Tong Li[1]

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

Abstract

test

1 Introduction

Poisson's equation is an important type of differential equations in physics, especially in electromagnetism. The 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

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

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

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

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

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

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

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

and a new right-hand side

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

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

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

which can be solved by backward substitution

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

This algorithm needs n floating point operations. The time complexity is $O(n)$.

2.2 Special method for this problem

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

$$\tilde{b}_i = 2 - \frac{1}{\tilde{b}_{i-1}} \tag{14}$$

which leads to

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

Therefore, $\tilde{\mathrm{b}}$ 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

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

The time complexity of this algorithm 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

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

The maximum relative errors of methods M1, M2 and M3 for different grid spacing h are shown in Fig. ??.

Table ?? shows the time used by algorithms M1, M2 and M3 for different grid spacing h.

4 Conclusions and outlook

conclusion

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: Linear algebra methods. `https://compphysics.github.io/ComputationalPhysicsMSU/doc/pub/linalg/html/linalg.html`. Accessed Feb 4, 2018.

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

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

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