Two methods of solving Poisson's equation

Anna Lina P. Sjur and Jan-Adrian H. Kallmyr

September 6, 2018

Abstract

State problem. Briefly describe method and data. Summarize main results.

1 Introduction

Physics is a field concerned with the behaviour of nature, and nature is everchanging. It is therefore no surprise that differential equations appear everywhere in physics. From global climate dynamics to statistical mechanics, what we find is that differential equations, often many and coupled, are required to explain or model the phenomena. For such large models, efficiency is important, as we would, for example, like to have timely weather forecasts. One way to make a model more efficient is by using an efficient algorithm for solving differential equations.

In this report, we compare two different numerical methods of solving linear second-order differential equations with the Dirichlet boundary conditions. To do this, we will solve the one-dimensional Poisson's equation:

$$\frac{\mathrm{d}^2\phi}{\mathrm{d}r^2} = -4\pi r\rho(r). \tag{1}$$

In the methods section, we develop an approximation for the 2nd derivative to the 2nd order. We will then solve eq. 1 numerically, using gaussian elimination and lower-upper decomposition. As for the former, we will further specialise it to solve eq. 1 more efficiently. Next, in the results section we present the comparison between our numerical solutions and the analytical solution, as well as the error. We then compare the efficiency of all three algorithms, and finally, in the discussion section we will consider the advantages and disadvantages of each algorithm, and discuss their uses.

2 Methods

We would like to solve eq. 1 numerically. Generalising the equation, we get

$$-\frac{\mathrm{d}^2u}{\mathrm{d}x^2} = f(x), \tag{2}$$

where we have assumed that $\rho \propto \frac{1}{r}e^{-r}$ and let $r \to x$, $\phi \to u$. Summing the backward and forward Taylor expansions of $u(x)$ and discretising the equation for n integration points, we get:

$$\frac{v_{i+1} - 2v_i + v_{i-1}}{h^2} + \mathcal{O}(h^2) = f_i, \tag{3}$$

where $h = \frac{1}{n+1}$. Using the Dirichlet boundary conditions $v_0 = v_{n+1} = 0$ and only considering $x \in (0,1)$, we can rewrite the equation as a set of linear equations, represented as the following matrix equation:

$$A\mathbf{v} = \mathbf{d}, \tag{4}$$

where

$$A = \begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \ddots & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -1 & 2 \end{bmatrix},$$

is a tridiagonal matrix and $d_i = h^2 f_i$. Having the equation in this form, we can use linear algebra to solve the set of linear equations, and obtain the 2nd derivative of $u(x)$.

2.1 Gaussian elimination

The fact that A is tridiagonal, means that we can develop a general algorithm to solve the equations by the method of Gaussian elimination. If we generalise our matrix

$$A = \begin{bmatrix} b_1 & c_1 & 0 & \dots & \dots & 0 \\ a_1 & b_2 & c_2 & 0 & \dots & 0 \\ 0 & a_2 & b_3 & c_3 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \ddots & a_{n-2} & b_{n-1} & c_{n-1} \\ 0 & \dots & \dots & 0 & a_{n-1} & b_n \end{bmatrix},$$

and row reduce A to become upper-triangular (see appendix A.) This yields us the following relations:

$$\widetilde{b_i} = b_i - \frac{a_{i-1}}{\widetilde{b}_{i-1}} c_{i-1}, \quad \widetilde{d_i} = d_i - \frac{a_{i-1}}{\widetilde{b}_{i-1}} \tilde{d}_{i-1}, \quad (5)$$

where $\widetilde{b_1} \equiv b_1$ and $\widetilde{d_1} \equiv d_1$. Performing the matrix-vector multiplication in eq. 4, we get the following relations from the second-to-last and last row:

$$v_i = \frac{d_i - c_i v_{i+1}}{\widetilde{b_i}}, \quad v_n = \frac{\widetilde{d_n}}{\widetilde{b_n}}. \quad (6)$$

With these expressions, we can construct our algorithms as follows:

```
initialise $\mathbf{v}$, $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$;
for $i = 2, \dots, n$ do
    $b_i - = \frac{a_{i-1}}{b_{i-1}} c_{i-1}$;
    $d_i - = \frac{a_{i-1}}{b_{i-1}} c_{i-1}$;
$v_n = \frac{d_n}{b_n}$;
for $i = n-1, \dots, 1$ do
    $v_i = \frac{d_i - c_i v_{i+1}}{b_i}$;
```

2.2 LU-decomposition

3 Results

Let's start with the solution to equation 4. Our general matrix solver gave the results shown in figure 1 for n=10, n=100 and n=1000. The analytic solution is also shown. For n=1000 the numerical and analytical solutions are so close to each other that we can not distinguish them in the plot, even after zooming in with a factor of 100. Both the specialized algorithm and the LU decomposition algorithm gave the same result as shown in figure 1.

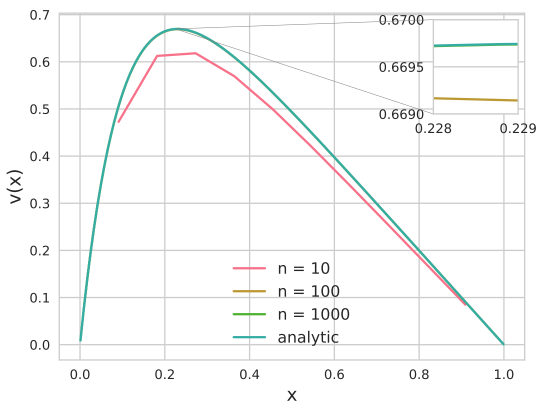

Figure 1: The numeric solution using different numbers of steps and the analytic solution. The window shows a section of the plot zoomed in with a factor of 100.

We wanted to get an idea about the efficiency of the different algorithms and how they compare. Table 1 shows the ratio between the CPU time for the general algorithm and the special algorithm, and the ratio between the LU decomposition algorithm and the special algorithm for different values of n. We see that the special algorithm is between 40% to 100% faster than the general algorithm, where the difference appears to decrease as n increases. On the other hand, the ratio between the LU decomposition algorithm and the special algorithm increases rapidly with n. For n equal to 10^4 the special algorithm is more than a million times faster than the LU decomposition. We were not able to run the program for n greater than 10^4.

n	$\mathbf{t_g/t_s}$	$\mathbf{t_{LU}/t_s}$
10	2.08	3.70
10^2	1.89	$1.00 \cdot 10^2$
10^3	1.48	$1.05 \cdot 10^4$
10^4	1.43	$1.18 \cdot 10^6$
10^5	1.39	-
10^6	1.41	-
10^7	1.39	-

Table 1: Ratio between CPU time for the general algorithm ($\mathbf{t_g}$), the special algorithm ($\mathbf{t_g}$) and the LU decomposition algorithm ($\mathbf{t_{LU}}$) for different matrix sizes (**n**). The LU decomposition crashed for **n** greater than 10^4.

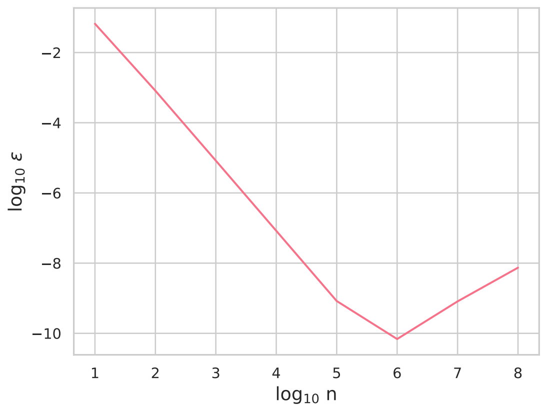

Figure 2: The maximum error in our specialized matrix solver as a function of the number of steps/matrix size on a logarithmic scale.

Lastly we wanted to assert the error in our numerical solutions. Figure 2 shows the maximum error in our specialized algorithm as a function of n with a logarithmic scale. We see that the error decreases with n until n = 10^6, and then increases.

4 Discussion

References

A

Derivation of Gaussian elimination algorithm terms

Considering our generalised matrix

$$A = \begin{bmatrix} b_1 & c_1 & 0 & \dots & \dots & 0 \\ a_1 & b_2 & c_2 & 0 & \dots & 0 \\ 0 & a_2 & b_3 & c_3 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \ddots & a_{n-2} & b_{n-1} & c_{n-1} \\ 0 & \dots & \dots & 0 & a_{n-1} & b_n \end{bmatrix},$$

we will obtain our algorithm from row reduction by subtracting the first row times $\frac{a_1}{b_1}$ from the second row $(\mathrm{I} - \frac{a_1}{b_1}\mathrm{II})$), and we see that the new diagonal element is given by

$$\widetilde{b_2} \equiv b_2 - \frac{a_1}{b_1}c_1.$$

Considering eq. 4 then, the corresponding change on the left hand side is given by

$$\widetilde{d_2} \equiv d_2 - \frac{a_1}{b_1}d_1.$$

Performing the same operation on the next row $(\mathrm{III} - \frac{a_2}{\widetilde{b_2}}\mathrm{II})$, we get

$$\widetilde{b_3} \equiv b_3 - \frac{a_2}{\widetilde{b_2}}c_2, \quad \widetilde{d_3} \equiv d_3 - \frac{a_2}{\widetilde{b_2}}\widetilde{d_2}.$$

Letting $\widetilde{b_1} \equiv b_1$, $\widetilde{d_1} \equiv d_1$, and generalising, we get:

$$\widetilde{b_i} = b_i - \frac{a_{i-1}}{\widetilde{b}_{i-1}}c_{i-1}, \quad \widetilde{d_i} = d_i - \frac{a_{i-1}}{\widetilde{b}_{i-1}}\widetilde{d_{i-1}}. \tag{4.7}$$