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:

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

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:

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

where $h = \frac{1}{n+1}$. Using the Dirichlet boundary conditions $v_0 = v_{n+1} = 0$, 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 $d = h^2 f_i$. 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

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

3 Results

Let's start with the solution to equation (her trengs ref). 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 similar that we can not distinguish them in the plot, even after zooming in with a factor of 100.

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.

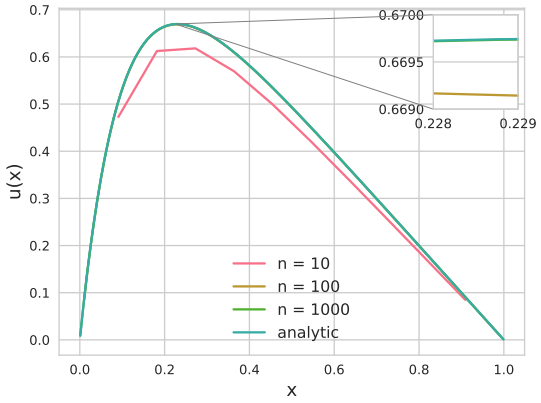

Figure 1: The numeric solution using different numbers of steps with the analytic solution. The window shows a section of the plot zoomed in with a factor of 100.

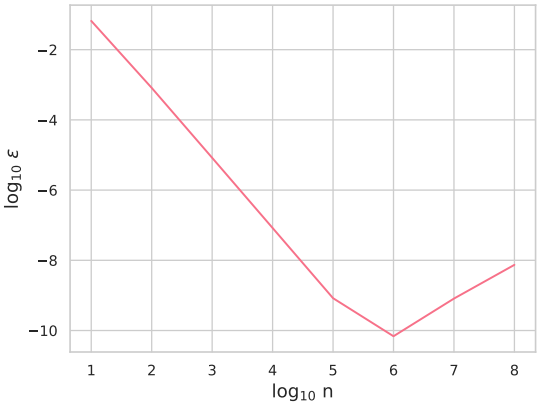

Figure 2: The maximum error in our specialised matrix solver as a function of the number of steps/matrix size in a logarithmic scale.

$\mathbf{n}$	$\mathbf{t_g}/\mathbf{t_s}$	$\mathbf{t_{LU}}/\mathbf{t_s}$
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 ($\mathbf{n}$). The LU decomposition chrased for $\mathbf{n}$ greater than 10^4.

4 Discussion

References