Project 1: Solution of a tridiagonal system of equations

Adam Powers

8 May 2016

Abstract

Quite often, one can use a system of linear equations to represent a differential equation. We used this technique and first use Gaussian elimination to solve for the general solution to the system. Gaussian elimination is extremely inefficient though, so for solving systems with constant diagonal elements in a tridiagonal matrix we wrote a more efficient algorithm. We then compared its results to a pre-written LU decomposition code.

1 Introduction

Scientists often encounter differential equations that are not able to be solved analytically and therefore need to be solved numerically. Often, computational science allows these differential equations to be solved more easily. The class we will be focused on is written as

$$\frac{d^2y}{dx^2} + k(x)y = f(x) \tag{1}$$

for a source function f(x) and a real equation k(x).

One example of this class of equation is found in classical electrostatics. The Electric field of a point charge can be found using Poisson's equation:

$$\nabla^2\Phi(r) = -4\pi\rho(r) \tag{2}$$

where $\rho(r)$ is the charge distribution. Assuming spherical symmetry, this can be written as a one-dimensional equation

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\Phi}{dr}\right) = -4\pi\rho(r)$$

which can be written as

$$\left(\frac{d^2\phi}{dr^2}\right) = -4\pi\rho(r)$$

by letting $\Phi(r) = \phi(r)/r$. This is now a linear second-order differential equation of the same form as (1) where k(r) = 0 and $f(r) = -4\pi r\rho(r)$. To makes things more relatable we let r → x and ϕ → u. We then define $f(x) = -4\pi\rho(r)$. Our equation then becomes

$$-u''(x) = f(x).$$

[1] http://arma.sourceforge.net

Sometimes equations of this form can be solved analytically, but more often they must be solved using numerical methods.

2 Algorithm

We will be solving the following equation in order to make the problem substantial:

$$-u''(x) = f(x). \tag{3}$$

This will be within the domain $x \in [0,1]$ with Dirichlet boundary conditions $u(0) = u(1) = 0$.

2.1 Transformation to linear equations

The second order finite difference relation is known to be

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

for some small step size h. Pluggin this back into (3) gives us the equation

$$f(x) = -\frac{u(x+h)+u(x-h)-2u(x)}{h^2}. \tag{5}$$

We then discretize the equation by creating pseudo-vectors of $u(x)$ and $f(x)$, named u_i and f_i, within our boundaries previously set. This, we can write as

$$-\frac{u_{i+1}+u_{i-1}-2u_i}{h^2} = f_i, \qquad i = 1, \dots, n. \tag{6}$$

If we continue thinking of u and f as vectors, we can interpret (6) as taking the (i+1)th element of u, the (i-1)th element of u, and so on. This creates an interpretation of the equations in terms of a system of linear equations

$$\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{pmatrix} \tag{7}$$

where $w_i = h^2 f_i$, and all elements not included in the matrix are taken to be zero. This matrix on the left is a *tridiagonal* matrix, meaning to only has elements in the primary diagonal and the diagonals above and below it.

Rewriting the equations in this way, allows us a much more manageable system of linear equations problem to solve numerically than trying to solve the earlier differential problem.

2.2 Gaussian elimination

One of the most basic and well-known methods for solving linear equations is Gaussian elimination. There are two steps to Gaussian elimination, a forward substitution and a backwards substitution.

If we start with a general matrix

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & w_1 \\ a_{21} & a_{22} & a_{23} & a_{24} & w_2 \\ a_{31} & a_{32} & a_{33} & a_{34} & w_3 \\ a_{41} & a_{42} & a_{43} & a_{44} & w_4 \end{pmatrix}$$

we place the solution vector **w** as the last column, the forward substitution step consists of adding multiples of each row to the rows below it until the matrix become upper-triangular. So, after one substitution, the original matrix becomes

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & w_1 \\ 0 & a'_{22} & a'_{23} & a'_{24} & w'_2 \\ 0 & a'_{32} & a'_{33} & a'_{34} & w'_3 \\ 0 & a'_{42} & a'_{43} & a'_{44} & w'_4 \end{pmatrix}, \text{ where } \begin{cases} a'_{ij} & = a_{ij} - \dfrac{a_{i1}}{a_{11}} a_{1j} \\ w'_i & = w_i - \dfrac{a_{i1}}{a_{11}} w_1 \end{cases}$$

After two substitutions,

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & w_1 \\ 0 & a'_{22} & a'_{23} & a'_{24} & w'_2 \\ 0 & 0 & a''_{33} & a''_{34} & w''_3 \\ 0 & 0 & a''_{43} & a''_{44} & w''_4 \end{pmatrix}, \text{ where } \begin{cases} a''_{ij} & = a'_{ij} - \dfrac{a'_{i2}}{a'_{22}} a'_{2j} \\ w''_i & = w'_i - \dfrac{a'_{i2}}{a'_{22}} w'_2 \end{cases}$$

and after a third substitution,

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & w_1 \\ 0 & a'_{22} & a'_{23} & a'_{24} & w'_2 \\ 0 & 0 & a''_{33} & a''_{34} & w''_3 \\ 0 & 0 & 0 & a'''_{44} & w'''_4 \end{pmatrix}, \text{ where } \begin{cases} a'''_{ij} & = a''_{ij} - \dfrac{a''_{i3}}{a''_{33}} a''_{3j} \\ w'''_i & = w''_i - \dfrac{a''_{i3}}{a''_{33}} w''_3 \end{cases}.$$

Once the forward substitution step is completed, we need to continue with the backwards substitution in order to fully diagonalize the matrix. The backwards substitution only affects the solution vector **w**. In the next steps, assume, after forward substitution the b_{ij} are the elements of the matrix, and the v_i are the elements of the solution vector.

After one backward substitution,

$$\begin{pmatrix} b_{11} & b_{12} & b_{13} & 0 & v_1' \\ 0 & b_{22} & b_{23} & 0 & v_2' \\ 0 & 0 & b_{33} & 0 & v_3' \\ 0 & 0 & 0 & b_{44} & v_4 \end{pmatrix}, \text{ where } v_i' = v_i - \frac{b_{i4}}{b_{44}} v_4.$$

After two,

$$\begin{pmatrix} b_{11} & b_{12} & 0 & 0 & v_1'' \\ 0 & b_{22} & 0 & 0 & v_2'' \\ 0 & 0 & b_{33} & 0 & v_3' \\ 0 & 0 & 0 & b_{44} & v_4 \end{pmatrix}, \text{ where } v_i'' = v_i' - \frac{b_{i3}}{b_{33}} v_3'.$$

Finally, after three,

$$\begin{pmatrix} b_{11} & 0 & 0 & 0 & v_1''' \\ 0 & b_{22} & 0 & 0 & v_2'' \\ 0 & 0 & b_{33} & 0 & v_3' \\ 0 & 0 & 0 & b_{44} & v_4 \end{pmatrix}, \text{ where } v_i''' = v_i'' - \frac{b_{i2}}{b_{22}} v_3''.$$

After all of the forward substitution is completed, the system is completely solved.

This is definitely not the most efficient algorithm for solving a system of equations computationally. The forward substitution step scales as $O(n^3)$ and the backward substitution scales as $O(n^2)$. Following will be a discussion of a more efficient method to solve our problem.

2.3 An algorithm for our problem

In (7) we showed that the problem for us to solve can be represented by a tridiagonal matrix. Most of the elements are zero, and the rest of the elements take one of two constant values. If we use the original Gaussian algorithm, there is wasted time computing the elements that are initially zeroes. Representing the equation as a dense matrix is a waste of memory and may be impossible as the step size gets increasingly smaller.

The matrix can be stored more efficiently if we look at the repeating elements. The upper and lower diagonal elements are all -1 and the Gaussian elimination process will not change any of these elements since all of the elements above the upper diagonal are all zeroes. The lower diagonal however will be changed to zeroes by the forward substitution step. We can simply represent the two diagonals by scalar constants. The primary diagonal, however, will change so this can be represented as a one-dimensional array or a vector. This reduces the memory needed to store the matrix from $O(n^2)$ to about $O(n)$.

If we reduce the number of elements, we also reduce the number of computations needed to solve the problem. If we keep the two constants for the upper and lower diagonals, and consider the

changing primary diagonal to be a vector, we can reduce the number of floating point operations from cubic dependency to linear ~ $O(8n)$.

3 Results

The algorithm explained above was run for matrices with n equal to 10, 100, and 1000 to solve the differential equation for the source function

$$f(x) = 100e^{-10x}. \tag{8}$$

The differential has an analytic solution of

$$u(x) = 1 - (1 - e^{-10})x - e^{-10x} \tag{9}$$

which allows us to compare how accurate our solution is.

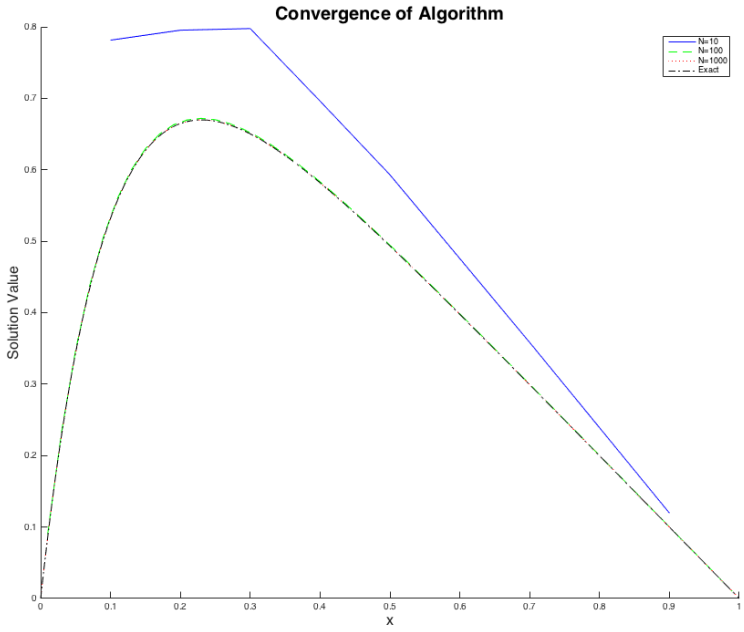

Figure 1: Convergence of the algorithm for varying values of n. The dotted and dashed line shows the exact values of the analytic solution. The results are converging on these values as n increases.

One can see how with a greater number of steps, the accuracy increases, but there becomes a point where the error does not increase due to the limitations on floating point arithmetic. Its decrease in error is ultimately limited by numerical precision.

n	Gaussian solver time (s)	Armadillo LU time (s)
10^1	4.00 e-6	9.1 e-5
10^2	5.00 e-6	6.88 e -4
10^3	3.50 e-5	1.15175
10^4	2.77 e-4	-
10^5	2.21 e-3	-

Table 1: Comparison of time taken by custom Gaussian elimination code and the Armadillo LU decomposition code. The Armadillo LU decomposition did not finish for the two largest values of n.

The running time of the code as measured for several values of n. A comparable code was found using the LU decomposition and linear equation solver functions built into the Armadillo linear algebra library[1] and timed for the same values of n. The timing comparison is shown in Table 1.

4 Conclusion

We explored the benefits of using a specialized Gaussian distribution algorithm over a more generic version. This decision boosted our performance from $O(n^3)$ FLOPs to O(8n) FLOPs. This is only available to be used with tridiagonal matrices with constant diagonal values. Even though this only applies to certain positions, the benefit of exploring this was to see how beneficial it can be to tailor code to certain situations after a couple light manipulations are done in order to cut away a lot of the unnecessary operations.

[1] http://arma.sourceforge.net