Gauss elimination

Alaina Ross
(Dated: February 1, 2017)

I. INTRODUCTION

Often in science problems can be written as a linear second order differential equation such as:

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

One specific example of this is the Poisson equation in electromagnetism, which is used to find the electrostatic potential, Φ, from a spherically symmetric charge distribution, $\rho(r)$ as:

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

using the substitution $\Phi = \phi/r$ and simplifying gives:

$$-\frac{d^2\phi}{dr^2} = 4\pi r\rho(r). \tag{3}$$

In general terms, the equation we aim to solve is $-u''(x) = f(x)$. Since this is a second order differential equation, it requires two boundary conditions. Here we use Dirchilet conditions, namely $u(0) = u(1) = 0$.

Turning these continuous functions in discrete ones we can approximate the second derivative as:

$$-\frac{u_{i+1} + u_{i-1} - 2u_i}{h^2} = f_i \quad i = 1, 2...n \tag{4}$$

where h is the step size defined as $h = 1/(n+1)$ and n is the number of grid points. This set of equations then can be rearranged as a matrix equation $\hat{A}\vec{v} = \vec{b}$ where:

$$\hat{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,n} \\ a_{n,1} & a_{n,2} & a_{n,n} \end{bmatrix} \quad \vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_n \end{bmatrix} \quad \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_n \end{bmatrix} \tag{5}$$

To solve this matrix equation the first goal is to get the matrix in row echelon form (shown below), where the only nonzero elements form the upper right triangle of the matrix. This step is called forward elimination and is done analytically via elementary row operations i.e. addition, interchanging rows, and multiplication by a constant.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,n} \\ 0 & \tilde{a}_{2,2} & \tilde{a}_{2,n} \\ 0 & 0 & \tilde{a}_{n,n} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_n \end{bmatrix} = \begin{bmatrix} \tilde{b}_1 \\ \tilde{b}_2 \\ \tilde{b}_n \end{bmatrix} \tag{6}$$

Algorithmically, this can be calculated two ways: the simpler way which only uses addition and multiplication by a constant but will fail if $a_{1,1} = 0$, or the more sophisticated way where rows are interchanged to ensure a solution even when $a_{1,1} = 0$ [**?**]. In this work we will use the former method and simply ensure that for all matrices $a_{1,1} \neq 0$.

We start with the first row and column as what is called the pivot; this is the column that will be updated to be zeros based on the first pivot element (the first element in the row/the corner piece). Then, all values in the rest of the matrix are updated via the same operations that were used in the corresponding element of the pivot column. In the next iteration, the pivot moves to the next row/column and the old pivot becomes unused data for the rest of the iteration process, meaning that each iteration the subspace to update becomes smaller. This process can be viewed graphically in Fig. 1.

Once the elimination process is complete the next step is called backwards substitution. First, the solution for x_n is obtained trivially from Eq. 6 as $\tilde{b}_n/\tilde{a}_{n,n}$. Next, this solution for x_n is substituted into the $n-1$ equation to solve for x_{n-1}. This is repeated until the whole vector $\vec{x}$ has been solved for. Algorithmically this corresponds to:

$$x_i = \frac{1}{\tilde{a}_{i,i}}\left(\tilde{b}_i + \sum_{j=i+1}^{n} \tilde{a}_{i,j}x_j\right) \tag{7}$$

In this work we implement the general gauss elimination algorithm as described above for matrix sizes of $n = 10, 100, 1000$, and compare the performance to that of a general tridiagonal matrix and the specific tridiagonal matrix that results from Eq. 4 both shown below:

$$\hat{A} = \begin{bmatrix} d_1 & c_1 & 0 & 0 \\ a_1 & d_2 & c_2 & 0 \\ 0 & a_{n-2} & d_{n-1} & c_{n-1} \\ 0 & 0 & a_{n-1} & d_n \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & 0 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \tag{8}$$

II. RESULTS

III. CONCLUSIONS

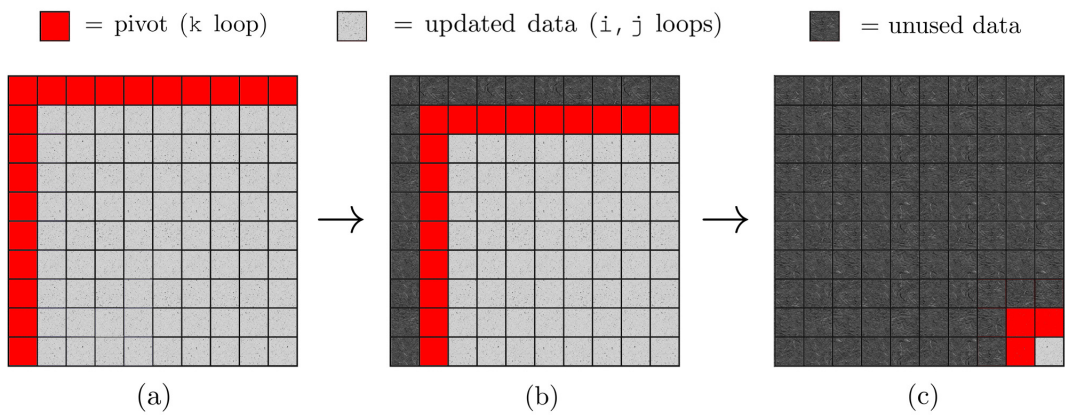

FIG. 1: Graphical representation of forward elimination algorithm based on [**?**] for (a) the first iteration, (b) the second iteration, and (c) the final iteration.

$log_{10}(h)$	$max(\epsilon_i)$
-1	-1.1797
-2	-3.08804
-3	-5.08005
-4	-7.07929
-5	-9.00487
-6	-6.77135
-7	-5.96231

TABLE I: Relative error as a function of step size

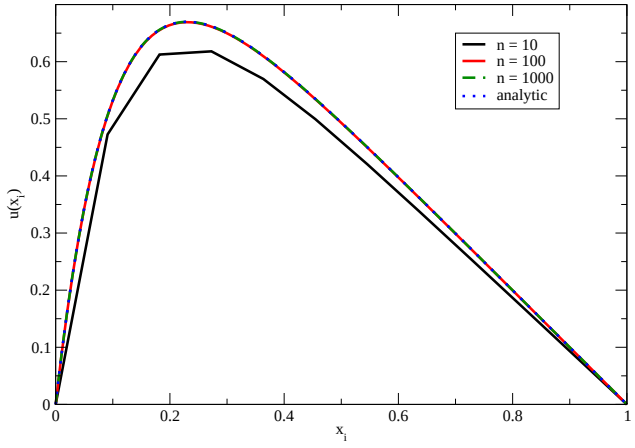

FIG. 2: Solution to diff eqn as a function of matrix size