Project 1

Colin and Spencer[1]

[1] *Department of Physics and Astronomy, Michigan State University*

We present two solution methods to a second order differentiable function subject to Dirichlet boundary conditions. Floating point operations are tabulated with their corresponding matrix sizes and a relationship is established between computational intensity and numerical accuracy.

INTRODUCTION

This is project 1 of our computational physics course. The basic goal of the project is to find a numerical solution to the second order differential equation $-\frac{d^2u}{dx^2} = f(x)$ specifically for

$$f(x) = 100e^{-10x} \tag{1}$$

with $u(0) = u(1) = 0$. We solve this by making a matrix on the idea that

$$\frac{u(x+h) + u(x-h) - 2u(x)}{h^2} = \frac{d^2u}{dx^2} + \sigma(h^2) \tag{2}$$

. This equation gives us a matrix which can take a set of points of our unknown $u(x)$ and turn that into a column vector $(u(x_1), u(x_2), u(x_3), ..., u(x_n)^{\mathrm{T}}$. That column vector transformed by a matrix A which would have the diagonal be all 2 thus corresponding to $u(x_{\mathrm{i}})$ and the lower and upper diagonal -1, which corresponds to $u(x_{\mathrm{i-1}})$ and $u(x_{\mathrm{i+1}})$ respectively. This will result in the known vector $h^2(f(x_1), f(x_2), f(x_3), ..., f(x_n))^{\mathrm{T}}$.

THEORY, ALGORITHMS AND METHODS

The Gaussian method

So we split the matrix into three arrays since it is tridiagonal. One has a length of n (actual diagonal), the other two (upper and lower) have a length $n-1$ for an $n \times n$ matrix. What we do here is equivalent to a basic Gaussian reduced row echelon form of a general tridiagonal matrix. First thing we do is get rid of the lower diagonal which is represented by the array $c[i]$. We do this by taking the highest row with a lower tridiagonal element and subtracting it from a product of the row above such that it will eliminate that lower tridiagonal element. Since this is essentially an augmented matrix this also acts on the inhomogeneous part $f[i]$ of the equation. Then we get rid of the upper tridiagonal, which is rather analogues. The main difference is you start at the bottom row and work your way up. since it is unnecessary, to change the diagonal and the arrays for the tridiagonals become obsolete, it only calculates the change of the inhomogeneous part.

the LU decomposition method

The LU decomposition method is a more general method to do a reduced row echelon form. one can take a generic matrix A and split it into two matrices L and U such that $LU = A$. The L matrix is a lower triangle matrix and the U matrix is an upper triangle matrix. Since they are lower and upper triangle matrices, it is easy to write an algorithm to reduce row echelon these matrices. We solve $Ly = f$ first then solve $Ux = y$ where x is the solution to the differential equation and f is the inhomogeneous part. The L matrix is generally set up where there are only 1's in the diagonal. Using the 1's we can easily cancel all other columns that are off diagonal by eliminating them with the row with zeros to the left of the 1 in the diagonal. Do the same elimination on the f array and you should get y as a result. Now we have to now solve $Ux = y$ which can be done by doing elimination the opposite way as the L. This will give us the result $x = U^{-1}y$ which is our solution.

Relative Error

n conjunction with floating point operations we must consider the relative accuracy of our solutions. We compare our solution, $u(x_i)$, to the exact value from equation 1. The logarithmic error

$$\log(\frac{|u(x_i) - f(x_i)|}{f(x_i)}) \tag{3}$$

is examined to determine the power law relationship between numerical error and step size. A the slope of a log-log plot for error and step size will be representative of the power law relationship we are looking for.

$$\tilde{y} = \tilde{a} + nx \tag{4}$$

RESULTS AND DISCUSSIONS

Floating point operations

The different solution methods, Gaussian elimination and LU decomposition, both yield relationships of floating point operations to matrix size that agree with an

n	FLOPS_G	TIME_G (s)	FLOPS_LUD	TIME_LUD (s)
120	1191	0.001	4636839	0.0
130	1291	0.0	5892509	0.001
140	1391	0.0	7356579	0.001
150	1491	0.0	9045049	0.001
160	1591	0.0	10973919	0.001
170	1691	0.001	13159189	0.001
180	1791	0.001	15616859	0.001
190	1891	0.0	18362929	0.001
200	1991	0.0	21413399	0.001

TABLE I. FLOPS for numerical solution with $A_{n\times n}$, X_G for Gaussian Elimination and X_{LUD} for LU decomposition. it becomes clear that the LU Decomposition method requires a high number of operations to perform while the Gaussian elimination method is streamlined and requires a number of operations linear with respect to matrix size.

expected behavior of Gaussian FLOPS $\propto n$ and LU decomposition FLOPS $\propto n^3$. Naturally, this change in order on the dependence of FLOPS to matrix size results in a longer calculation time for the solutions produced by the LU method. As seen in table 1, the time required for calculation in larger matrices is consistently larger or equal to that of the Gaussian method. To examine the limitations imposed by either solution method we used the matrix sizes: $10, 100, 1000, 10000$. As seen for the $n = 200$ entry in table 1, the LU decomposition method needed to perform $\approx 2 \times 10^7$ operations. Solving with the previously mentioned matrix sizes resulted in a non-responsive program at the beginning of the LU Decomposition with $A_{n\times n} = A_{10^4\times 10^4}$.

n	FLOPS_G	TIME_G (s)	FLOPS_LUD	TIME_LUD (s)
10	91	0.0	2869	0.0
100	991	0.0	2686699	0.0
1000	9991	0.0	1626300297	0.03
10000	99991	0.01	–	–

TABLE II. A comparison between the Gaussian elimination method and the LU decomposition method by orders of magnitude in the matrix size, $A_{n\times n}$. The LU decomposition program crashed during the process of solving with a matrix, $A_{10^4\times 10^4}$

From table 2 we see that the relationship between FLOPS and matrix size for the Gaussian method is linear while the FLOPS for the LU decomposition method is a higher order polynomial. In figure 1 a plot of FLOPS vs matrix size for matrices 23 matrices between the sizes of 10 and 1000. Performing a fit for a $3rd$ order polynomial; using the python extension numpy.polyfit we found the polynomial to be, $P(x) = \frac{8}{3}x^3 + 2x^2 + \frac{1}{3}x - 1$. In terms of computational intensity the Gaussian elimination method is the clear winner, as expected. With an algorithm designed to handle this exact problem we would expect a lower computational load. The LU decomposition method is designed to be a more general approach to solving a system of equations, $A_{n\times n}$.

Numerical Error

Numerical error of the Gaussian method and the LU decomposition method was nearly identical. In figure 2 we see the solutions for both processes and aside from the initial point, both plots demonstrate nearly equivalent calculations which converge toward the exact solution at the limit of $x \to 1$. Notice in the bottom plot for the LU Decomposition method our initial starting point was incorrect resulting in fewer data points that the Gaussian. Despite not including the same number of points the solutions produced both exhibit the same behavior. Following in figure 3 is a log-log plot displaying the power law dependence of relative numerical error and matrix size. Referring back to equation 2 the error excluded in our second order differential is proportional to h^2. In the construction of our solution method our step size was inversely proportional to the matrix size. The results of our error analysis therefore exhibit the behavior we expect, for increases in matrix size the error decreases. The slope of error vs matrix size on a log-log plot indicates the power-law relationship. We find that the error decreases as, m^n where m is the matrix size and n is the slope of the log-log plot.

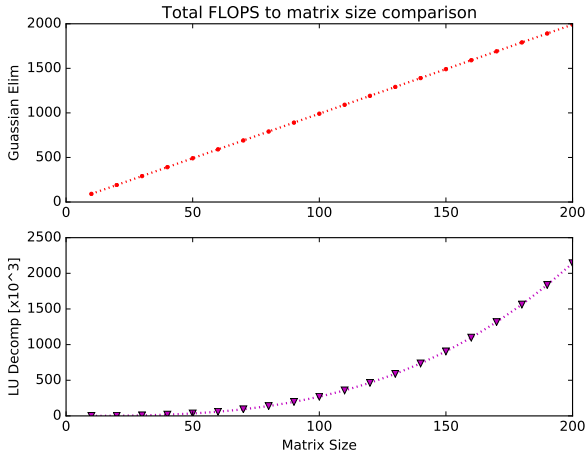

FIG. 1. We see how the Gaussian elimination method, seen in the top plot, has a linear relationship with matrix size while the LU decomposition method, in the lower plot, has a cubic relationship with matrix size. The LU decomposition has a characteristic polynomial of, $\frac{8}{3}x^3+2x^2\frac{1}{3}x-1$. The data points come from evaluation of matrices sized $10 \to 200$

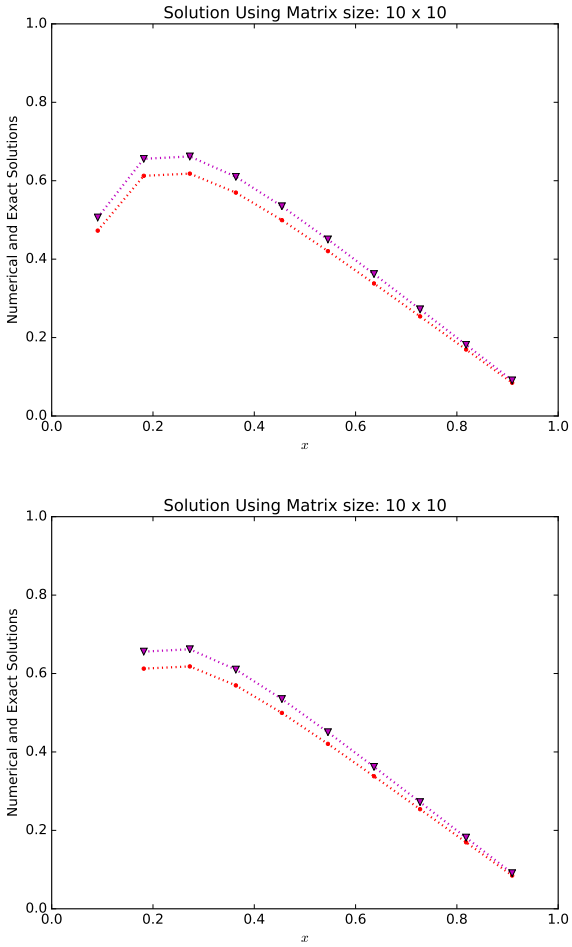

FIG. 2. Exact and numerical solutions for $n = 10$ mesh points.

CONCLUSIONS

if you look at the log-log plot above of the error, the slope of the error is about -2 for large n. since the plot is log(error) to essentially $log(n)$, then the error to n relationship is proportional to n^{-2} for large n. this is what we expected since the algorithm has an error that is proportional to $h^2 = n^{-2}$. for the second part of d; so the number of flops for the Gaussian method is $10n - 9$ in our program. The number flops for large n is on the order of n^3 for an LU decomposition. calculated to be $FLOPS(n) = \frac{8}{3}n^3 + 2n^2 + \frac{1}{3}n - 1$. the time it takes to run the 1000×1000 LU decomposition was around .0108 seconds. because the program would have a million times more flops to solve a $100,000 \times 100,000$ LU decomposition, it would take 3 hours to finish the program.

[1] G. A. Miller, A. K. Opper, and E. J. Stephenson, Annu. Rev. Nucl. Sci. **56**, 253 (2006).

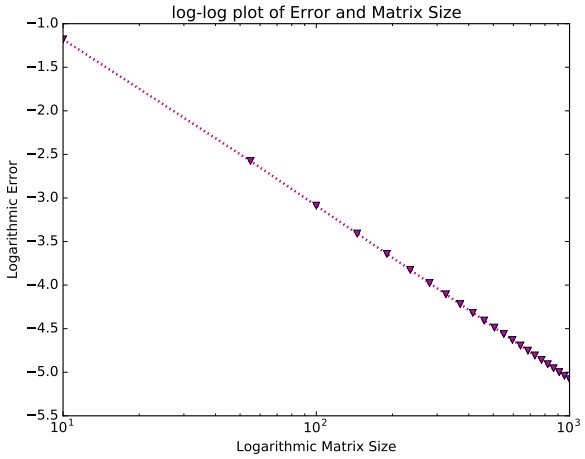

FIG. 3. Log-log plot of the relative error and matrix size. Larger matrix sizes result in a finer mesh grid to evaluate across.