Computational Physics

Project 1

UiO : Universitetet i Oslo

Hunter Wilhelm Phillips (*hunterp*)

September 10, 2018

GitHub Repository at

https://github.com/robolux/Computational_Physics

Abstract

This project explores the numerical solving of a second order differential equation in terms of a matrix. The use of general gaussian elimination, specialized tridiagonal solver, and LU decomposition allows for varying approaches to solving the problem. An error analysis was also performed to determine the costs of increased computational time with respect to a decrease in error. To verify the correlated computation time, a timing sequence was executed to compare with the recorded error analysis. The results show that the specialized tridiagonal algorithm has the optimal attributes of low computation time and low error.

1. Discussion of Problem Statement

In this project, the goal is to find a numerical solution to the following second order derivative with Dirichlet boundary conditions

$$-u''(x) = f(x) \quad , \quad x \in (0,1) \quad , \quad u(0) = u(1) = 0 \tag{1}$$

where f(x) is a known function, characterized as the source term

This problem in this particular situation the one-dimensional Poisson equation and can be solved analytically. To develop the ODE into a set of linear equations we can translate the structure into a tridiagonal matrix. This matrix can be reduced through the use of the algorithms developed for this project. These solutions are produced through gaussian elimination, a specialized tridiagonal scheme, and the widely popular LU decomposition. To compare the algorithms and provide data analysis, the performance with respect to computational time and error of the algorithms are produced to form a conclusion.

2. Discussion of Methods

2.1 Analytical Solution

We know that the source term is characterized as

$$f(x) = 100e^{-10x} \tag{2}$$

This source term has an analytical solution that exists as shown below

$$u(x) = 1 - (1 - e^{-10})x - e^{-10x} \tag{3}$$

To verify that the analytical solution satisfies the initial conditions proposed in this case, (3) should be coupled with (2) to form a solution to (1).

$$\frac{d^2}{dx^2}u(x) = \frac{d^2}{dx^2}(1 - (1 - e^{-10})x - e^{-10x})$$
$$\frac{d^2}{dx^2}u(x) = \frac{d}{dx}\left((1 - e^{-10}) + 10e^{-10x}\right)$$
$$\frac{d^2}{dx^2}u(x) = -100e^{-10x}$$
$$\frac{d^2}{dx^2}u(x) = -f(x)$$

It can be shown that (1) is satisfied by (3) and (2), verifying the validity of our analytical solution. This allows us to compare all following numerical calculations with an exact solution.

2.2 Numerical Solution

To start the process of solving the differential equation shown in (1) we discretized the problem into a numerical equation.

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

where $f_i = f(x_i)$

By defining the following three quantities using the a, b, and c values given by the prompt.

$$A = \begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ 0 & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & -1 & 2 & -1 \\ 0 & \dots & \dots & \dots & -1 & 2 \end{bmatrix} \quad v = \begin{bmatrix} u_1 \\ u_2 \\ \dots \\ u_i \\ \dots \\ u_n \end{bmatrix} \quad d = \begin{bmatrix} d_1 \\ d_2 \\ \dots \\ d_i \\ \dots \\ d_n \end{bmatrix} \tag{5}$$

Through the coupling of (4) being discretized and the definition of (5) it is trivial to rewrite the numerical equation as follows.

$$Av = s \tag{6}$$

The project can proceed forward with (6) laying the groundwork for the final numerical progression step.

The final step is to characterize a general tridiagonal matrix as seen below.

$$M = \begin{bmatrix} b_1 & c_1 & 0 & \dots & \dots & 0 \\ a_2 & b_2 & c_2 & 0 & \dots & \dots \\ 0 & a_3 & b_3 & c_3 & 0 & \dots \\ 0 & 0 & \dots & \dots & \dots & 0 \\ 0 & \dots & 0 & \dots & \dots & c_{n-1} \\ 0 & \dots & \dots & 0 & a_n & b_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_i \\ \dots \\ v_n \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ \dots \\ s_i \\ \dots \\ s_n \end{bmatrix} \quad (7)$$

The final matrix system showcased in (7) provides the conditions for successful algorithm development.

3. Discussion of Algorithms

3.1 General Gaussian Elimination

The first algorithm proposed is using standard gaussian elimination through Av = s to solve any tridiagonal matrix. The elements a, b, and c are represented as arrays since most elements are zero in a tridiagonal matrix.

```
22        # forward substitution
23        for m in range(1, n+2):
24            rf   =  a[m]/b[m-1]
25            b[m] -= c[m-1]*rf
26            s[m] -= s[m-1]*rf
27
28        # backward substitution
29        v[n+1] = s[n+1] / b[n+1]
30
31        for m in range(n, -1, -1):
32            v[m] = (s[m] - c[m]*v[m+1]) / b[m]
```

It can be seen that this algorithm requires approximately $8n^3$ floating point operations. It can also be observed that the space requirements are on the order of 5n.

3.2 Specialization for Tridiagonal Matrix

Since our matrix contains the prescribed values a = -1, b = 2, and c = -1 we can use the correlating row indexes to optimize the solving process within the algorithm. This is shown in the fact that $b_i = \frac{i+1}{i}$ and allows us to precalculate b_i. When coupled with the ability to simplify the respective s_i and v_i a new algorithm can be devised to take advantage of these characteristics.

```
49        # forward substitution
50        for m in range(2, n+1):
51            s[m] += (s[m-1])/(b[m-1])
52
53        # backward substitution
54        v[n] = (s[n])/(b[n])
55
56        for m in range(n-1,0,-1):
57            v[m] = (s[m]+v[m+1])/b[m]
```

It can be seen that this algorithm requires approximately $4n^3$ floating point operations. This is half of the operations required by the general gaussian elimination. It appears that our optimization and exploitation of the matrix features provided a huge benefit!

3.3 LU Decomposition

Using premade LU decomposition functions built into *numpy.linalg* the following script was programmed.

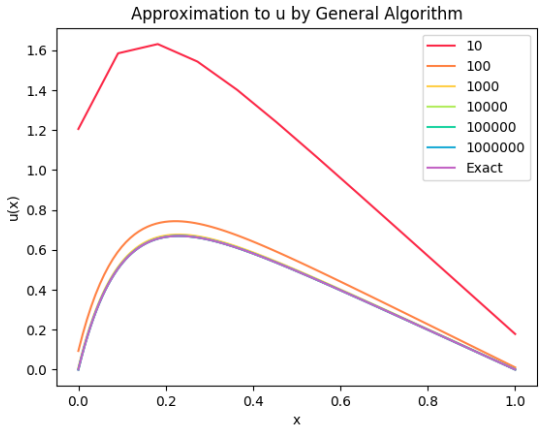
```
70        lu, pivot = scp.lu_factor(A, overwrite_a=True, check_finite=False)
71
72
73
74        x = nmp.linspace(0, 1, n+2)
75        h = x[1]-x[0]
76        s = f_function(x)[1:-1]*h**2
77
78        # solve the lu equation system
79        v_i = scp.lu_solve((lu, pivot), s, overwrite_b=True)
```

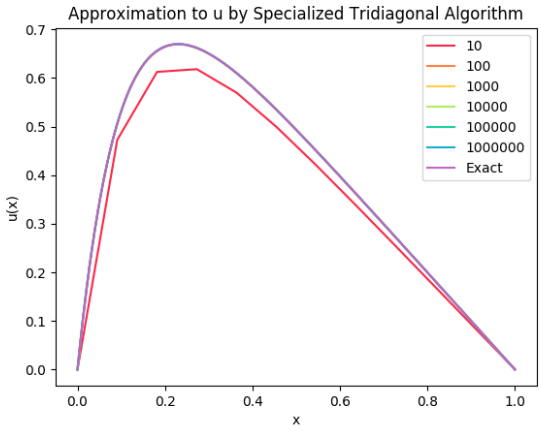
It can be noted that *lu_factor* only takes a full matrix as an argument, causing a massive waste of resources due to the large number of zeros in the tridiagonal matrix.

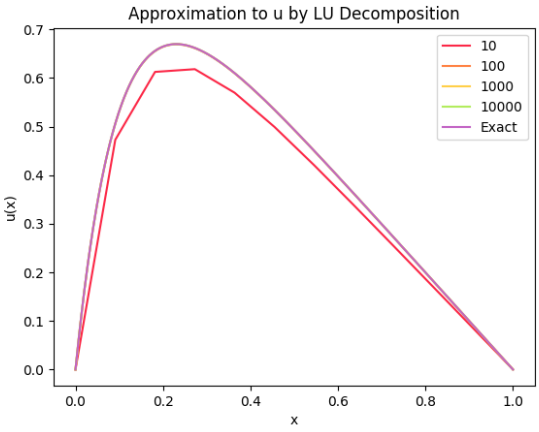
4. Discussion of Results

4.1 Numerical vs Analytical Solution

To compare the output of the algorithms versus the exact know solution plots were created to display the convergence with varying n values.







It can be noted that the LU Decomposition could only be run to matrix size 10000 x 10000 due to the terminal locking upon the computational load of a 100000 x 100000 or larger matrix. By analyzing the graphs, it can be seen the quick convergence to the exact solution. This validates the stability and junction reliability of our numerical solutions.

4.2 Relative Error

Computing the relative error of the specialized tridiagonal matrix can be done through the use of (8)

$$\varepsilon_i = log_{10}\left(\left|\frac{v_i - u_i}{u_i}\right|\right) \; with \; i = 1, \ldots \; , n \tag{8}$$

The results can be seen below in *Table 1*.

N	ε
10	-1.179698
100	-3.088037
1000	-5.080052
10000	-7.079268
100000	-9.079101
1000000	-10.162831

Table 1: Relative Error for Tridiagonal Special Algorithm N Range

When trying to increase n to n = 10^7 the program produces the following error: *divide by zero encountered in log10*. This stems from the fact that with floating point round off error we have essentially created a matrix that is trying to divide elements by zero. To solve this, we would need to make zero a really small number, however this can quickly lead down a rabbit hole of problems including round off error growing out of control and is not recommended.

4.3 Computation Time

One of the most important and troublesome facets of computing is the amount of computing time needed to solve the solution. To properly compare the algorithms a benchmark was run pitting the three algorithms used in this project against each other to compare run times.

The efficiency of each program can be seen below in *Table 2*.

N	General (s)	Specialized (s)	LU (s)
10	0.000135	0.000069	0.000172
100	0.000250	0.000155	0.000384
1000	0.001889	0.001130	0.025779
10000	0.028790	0.012470	14.001648
100000	0.196613	0.118152	Not Tested
1000000	1.920405	1.172803	Not Tested

Table 2: Computation Time Benchmarks

A few notes about the system running the programs before analyzing the results should be helpful in gauging performance between machines as seen in *Table 3*.

Operating System	Mac OS X
Processor	i7 - 2.2 GHz
Memory	8 GB 1600 MHz DDR3
Hard drive	128 GB SSD

Table 3: System Specifications

With the test conditions being established we can now dive into the data recorded. It can be observed that the general algorithm takes approximately twice as long to compute as the specialized tridiagonal algorithm. The LU factorization and its large amount of computational time come as no surprise due to it running in $\Theta(n^3)$ time. These results correlate with our expected findings and validate the program is running properly.

5. Conclusion

Through this project, the benefits of developing specialized algorithms instead of always using built in functions have been discovered. The case Av = s to solve the single-dimension Poisson's equations provided a perfect example to experiment with. The tridiagonal matrix is a unique case in that it has the ability to transcend across a multitude of algorithm solution possibilities. When comparing the specialized algorithm with LU decomposition, their results were almost equivalent, but LU decomposition was much slower in executing the result. If we take into account that they are both *reasonably* fast for small n values, it can come down to how much of a repeated task the algorithm needs to complete. This can come down to how much time the programmer has, since it was much quicker to use the LU built in functions that program a custom algorithm. For large n value datasets, there is no doubt that specialized algorithms are the way to go. This project culminates in fresh perspectives and the learning of new skillsets pertaining to Computational Physics.

6. Future Work and Thoughts

A few ideas I had while working on this project came to me during the process. I would like to try and build three lu functions that use no pivoting, partial pivoting, and full pivoting to solve the tridiagonal matrix. I think it would be intriguing to see the results and possibly find ways to manipulate it to optimize the lu pivot types for situations like this.

I also dove a little deeper into specialized tridiagonal solvers as this project peaked my curiosity into what else was out there. Article [1] was very well optimized to run in parallel which would greatly benefit a problem like this due to its inherent symmetry. Article [2] could allow possible multiplexing across parallel cores to solve the linear system by splicing the right-hand side. If I get time in the future I hope to implement these ideas and compare them to the results achieved in this project.

7. References

[1] Bar-On, Ilan, et al. "A Fast Parallel Cholesky Decomposition Algorithm for Tridiagonal Symmetric Matrices." SIAM Journal on Matrix Analysis & Applications, vol. 18, no. 2, Apr. 1997, p. 403.

[2] Terekhov, Andrew V. "Parallel Dichotomy Algorithm for Solving Tridiagonal System of Linear Equations with Multiple Right-Hand Sides." Parallel Computing, vol. 36, no. 8, Aug. 2010, pp. 423-438