

# PHY 905 Project 1

Jialin Liu  
Thanaphong Phongpreecha

February 13, 2016

# 1 Introduction

Several differential equations can be written in a form of a non-homogeneous second-derivative differential equation

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

This equation is versatile and is encountered in many fields. For example in engineering, it could be used for modeling mechanical and electrical oscillators with external force. [?]

The goal of this project is to use computational calculations, if possible, with dynamic memory allocation to solve the one-dimensional Poisson's equation in electromagnetism with Dirichlet boundary conditions. Detailed derivation can be found in the problem statement, but essentially, the 3D Poisson's equation is simplified to 1D (radial dimension) based on its spherical symmetry. After simplifying to 1D, the differential equation is discretized by rewriting to a set of linear equations of,

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i \quad \text{for } i = 1, \dots, n$$

The equation is solved by transforming to a tridiagonal matrix with  $f(x) = 100 \cdot e^{-10x}$ , where  $x \in (0, 1)$ . These matrices can then be solved using conventional techniques, such as Gaussian Elimination and LU Decomposition [?]. The results will be compared with exact solution given as  $u(x) = 1 - (1 - e^{10})x - e^{-10x}$ . The analyses of results generated in this report include, in an order of,

1. the contribution of the number of steps  $h$  to the accuracy of the estimation as compared to exact results
2. the relative errors of different step length
3. the differences and limits in using either LU decomposition or Gaussian Elimination approach to solve the matrix
4. the analytic solutions to this particular problem, and
5. the differences in computing time by MATLAB, Python, and Fortran using the same laptop.

## 2 Theoretical models and technicalities

### 2.1 Gaussian Elimination for Tridiagonal Matrix

To solve ordinary differential equation, Central Euler method is commonly used to achieve  $o(h^2)$  accuracy as shown in Eq.1 for second order derivative:

$$\frac{x_{i+1} + x_{i-1} - 2 * x_i}{h^2} = f''_i, \quad (1)$$

where  $h$  is the step length,  $x(i)$  is the  $i^{th}$  points and  $f$  is the function value at the  $i^{th}$  point.

This method can be implemented to solve many ODEs involving first and second order derivatives in the following form numerically:

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

where  $k^2$  is a real function.

For example, for the Poisson's equation under spherical symmetrical field using polar coordinations, the original equation can be simplified as following form:

$$\frac{1}{r^2} \frac{d}{dr} \left( r^2 \frac{d\Phi}{dr} \right) = -4\pi\rho(r), \quad (3)$$

where  $\phi$  is the electrostatic potential,  $\rho(r)$  is the local charge distribution and  $r$  is the radial distance

If we substitute  $\Phi(r) = \phi(r)/r$ , we can have

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

by letting  $\phi \rightarrow u$  and  $r \rightarrow x$ , this equation can be transformed into a simple form:

$$-u''(x) = f(x). \quad (5)$$

After discretizing this function on the region of  $(0,1)$ , at each point  $x_i$ , Central Euler method can be used to calculate the second order derivative. At the  $i^{th}$  point, the function value of Poisson Equation can be approximate using:

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i \quad \text{for } i = 1, \dots, n, \quad (6)$$

We rewrite this equation as a linear equation system into the form of

$$\mathbf{A}\mathbf{v} = \tilde{\mathbf{b}}, \quad (7)$$

where  $\mathbf{A}$  is an  $n \times n$  tridiagonal matrix which we rewrite as

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{pmatrix} \quad (8)$$

and  $\tilde{b}_i = h^2 f_i$ .

In this special matrix, the  $o(n^3)$  Gaussian Elimination (GE) methods can be simplified to Tridiagonal Gaussian Elimination (TGE) method which cost only  $o(8n)$  floating point calculations.

In the standard GE method, when we eliminate the first row,  $A_{21}$  will be eliminated and so will all  $A_{i,i-1}$  elements. Then let's consider elimination of the  $i^{th}$  row. The element on diagonal is  $A_{i,i}$ . The element on the left of diagonal,  $A_{i,i+1}$  will remain 0 after elimination because the elements above it are all 0. After forward substitution, all  $A_{i,i-1}$  elements are 0 and all  $A_{i,i+1}$  elements are -1. All elements on diagonal,  $d_i$ , will be substitute to a new value,  $\tilde{d}_i$ .

If we add  $N$  extra points in  $(0, 1)$ , the matrix  $A$  will be a  $N$  by  $N$  matrix. After forward substitution, the  $N^{th}$  row have only one element,  $A_{N,N}$  so the corresponding function value,  $u_N$ , can be solved easily. For the  $i^{th}$  row, if we expand the matrix operation, we will have the following equation:

$$\tilde{A}_{i,i} * u_i - u_{i+1} = \tilde{f}_i, \quad (9)$$

where  $u_i$  is the approximated function value at  $i^{th}$  point. Since  $u_{i+1}$  is already solved from previous step, the  $u_i$  can also solved from this equation. In this TGE method, only  $8N$  FLOPS are needed to solve for  $N$  points. Assume

$$f(x) = 100e^{-10x}, \quad (10)$$

and we can get the closed form solution:

$$u(x) = 1 - (1 - e^{-10})x - e^{-10x} \quad (11)$$

So, the relative error in each point can be calculated as

$$\epsilon_i = \log_{10} \left( \left| \frac{v_i - u_i}{u_i} \right| \right), \quad (12)$$

where  $u_i$  is the exact solution and  $v_i$  is the numerical solution.

## 2.2 LU decomposition

In LU decomposition method, a matrix  $A$  can be decomposed into the product of two special matrices,  $L$  and  $U$  which is shown below:

$$A = L * U, \quad (13)$$

where  $L$  is a lower triangle matrix with all diagonal elements as 1 and  $U$  is an upper triangle matrix. LU decomposition required  $o(N^3)$  floating points calculation which is same as standard GE method. However, after LU decomposition, the determinate can be easily calculated as product of diagonal elements in  $U$  matrix.

## 2.3 Implement TGE in Codes

To compare the calculation speed of different methods and efficiency of different programming languages, TGE and LU method are implemented in Matlab and Python to solve the Poisson Equation under Dirichlet boundary conditions on the region of  $(0, 1)$  using 10, 100, 1000 and 10000 points. All calculations are done in a Core i7-4770@3.40GHz CPU 16.0GB RAM desktop machine.

### 2.3.1 Matlab

MATLAB is a high-level language and interactive environment for numerical computation, visualization, and programming. MATLAB can analyze data, develop algorithms, and create models and applications. The language, tools, and built-in math functions can be used to explore multiple approaches and reach a solution faster than with spreadsheets or traditional programming languages, such as C/C++ or Java. Also, MATLAB can be used for a range of applications, including signal processing and communications, image and video processing, control systems, test and measurement, computational finance, and computational biology. More than a million engineers and scientists in industry and academia use MATLAB, the language of technical computing.

Matlab is specifically expertise on matrix operation, the first part of Matlab program simply called the native slash operator to solve Poisson equation:

```
Matlab_cal_u = (f/A);
```

The TGE method is implemented using FOR loop to ensure the using of atomic operation in Matlab. All variables are initialized at the beginning of this program based on different number of data points. In the backward substitution, the last row is calculated separately and then other rows are calculated in a FOR loop.

```
% Forward Substitute  
for i = 2:size(A,1)
```

```

        ratio = -(A(i,i-1)/A(i-1,i-1));
        A(i,:) = A(i,:)+A(i-1,:)*ratio;
        f(i) = f(i)+f(i-1)*ratio;
    end

% Backward Substitue
cal_u(N-2) = f(N-2)/A(N-2,N-2);
for i = (size(A,1)-1):-1:1
    cal_u(i) = (f(i)-cal_u(i+1)*A(i,i+1))/(A(i,i));
end

```

For the LU decomposition, a native `lu()` function is used to calculate the two matrices and the slash operator is used on each matrix to solve function value:

```

[L,U] = lu(fullMatrix);
LU_results = (f0/L)/U;

```

One thing to be notice is, in Matlab, the efficiency of FOR loop is relatively low comparing with matrix operation. To assign value for each element in the matrix, extra floating point calculations are needed.

### 2.3.2 Python

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. Comparing with Matlab, similar data structure and algorithms are used in Python code. Also numpy package and scipy.linalg module are used for mathematical operations and matplotlib is used for graph plotting. The core code for TGE is shown below:

```

# Forward Substitution
for i in xrange(1, n-1):
    d[0, i] = d[0, i] - e[0, i-1] / d[0, i-1] * e[0, i-1]
    f[i] = f[i] - e[0, i-1] / d[0, i-1] * f[i-1]
# Backward Substitution
u[0, n-2] = f[n-2] / d[0, n-2]
for i in range(n-3, -1, -1):
    u[0, i] = (f[i] - e[0, i] * u[0, i+1]) / d[0, i]

```

And for the LU decomposition,

```

LU = la.lu_factor(A)
X = la.lu_solve(LU, f)

```

## 3 Results and Discussion

### 3.1 Unit Test

The unit tests are done under Matlab Unit Testing Framework. The TGE solver developed in Matlab code is sealed in a subroutine TGE(N, d, e1, e2, f). Three test cases are used in this unit test:

1. A matrix is a 3 by 3 unity
2. A matrix is a 2 by 2 matrix
3. 10 points Poisson's Equation

By executing RunTest.m, this subroutine passed all three test cases. So this subroutine will be used as standard benchmark to test Python code.

```
>> RunTest
Running TGETest
...
Done TGETest
-----

ans =

1x3 TestResult array with properties:

Name
Passed
Failed
Incomplete
Duration

Totals:
3 Passed , 0 Failed , 0 Incomplete .
0.023027 seconds testing time.
```

### 3.2 Running Time Analysis

Figure 1 showed the running time for different methods in Matlab and Python. When we use less than 1000 points, the running time fluctuate a lot. This may due to the influence of data storage structure in the RAM. If a variable is stored in a continuous region, the I/O speed will be faster. And the actual calculation time contribute less in the total time. When we use 10000 points, the TGE method in Python is the fastest method. In the 10000 points test, it only cost

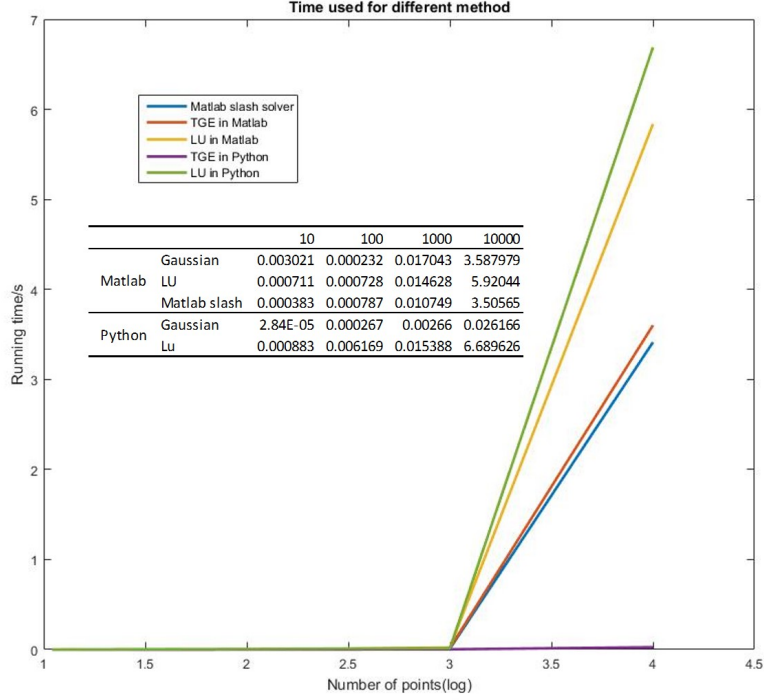


Figure 1: The time used for different methods.

3.9% of the Lu decomposition method in Python. This is due to the different calculation complexity of the two methods. TGE methods need  $8N$  FLOPS but LU method need  $N^3$  FLOPS. Similarly, the LU method in Matlab cost almost same time as LU in Python.

However, the TGE in Matlab cost only half of the LU method. This is because the FOR loop, Matrix indexing and value assignment for one element in matrix in Matlab is very slow. From the report of native code analyzer shown in Figure 2, we can see 24.2% running time are wasted in line 42 which is doing the value assignment in a matrix. The native slash operator in matlab can have as good performance as TGE method. This indicating slash operator can recognize character of a matrix and call proper solver.

## 4 Conclusion

In this work, we developed a numerical solver for Poisson's equation using TGE method and reduce the calculation complexity from  $o(N^3)$  to  $o(8N)$  comparing with standard GE and LU decomposition method. The solver passed all









Lines where the most time was spent					
Line Number	Code	Calls	Total Time	% Time	Time Plot
<a href="#">66</a>	<code>[L,U] = lu(fullMatrix);</code>	1	5.897 s	40.6%	
<a href="#">35</a>	<code>Matlab_cal_u = (f/A);</code>	1	3.552 s	24.5%	
<a href="#">42</a>	<code>A(i,:) = A(i,:)+A(i-1,:)*ratio...</code>	9998	3.512 s	24.2%	
<a href="#">63</a>	<code>fullMatrix = 2*diag(ones(N-2,1...</code>	1	0.571 s	3.9%	
<a href="#">2</a>	<code>clc</code>	1	0.202 s	1.4%	
All other lines			0.786 s	5.4%	
Totals			14.519 s	100%	

Figure 2: Report of Matlab Code Analyzer.

unit tests and can give correct results. The more points used in the calculation, the more accurate result one can get. But when step length is too small, the rounding error and overflow error will happen and decrease the calculation accuracy.

## References