DEPARTMENT OF PHYSICS
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS

PH5270 Numerical Methods and Prog. **Week5: Assignment 4** **12 Mar 2020**
Time: **Max. pts: 10**

Solution Set

1 See Assignment sheet for detailed question [1 pts]

Solution is $x_1 = 5/4$, $x_2 = -3/4$, $x_3 = -1/2$

2 Gaussian Elimination [2 pts]

2.1 Write therafter a program which implements Gaussian elimination (with pivoting) and solve the above system of linear equations.

Please check file named `Question2/Question2_Pivoting.cpp`

2.2 How many floating point operations are involved in the solution via Gaussian elimination without pivoting?

FLOPs or Floating Point Operations are operations which require floating-point calculations such as addition, subtraction, multiplication, division. Pivoting doesn't really introduce floating point operations. But it increases number of operations for CPU to perform. Number of floating point operations are 28 for 3×3 matrix. This includes operations made for backward substitution. Checkout the program for `Question2/Question2_NoPivoting.cpp` where we have calculated that number. In general $\frac{n(n+1)(4n-1)}{6} - n$ FLOPs for Gaussian Elimination. And n^2 FLOPs for Back Substitution.

2.3 Can you estimate the number of floating point operations with pivoting?

Please take a look at and run file named `Question2/Question2_Pivoting.cpp` for the answer. Which is same. i.e. 28. This includes operations made for backward substitution.

Pivoting helps in reducing spread of errors because of multiplication or division by very big number or very smaller number.

3 Poisson Equation [1 pts]

For detailed question, check assigment sheet. Question is same as given in the quiz. You will have to derive or show the matrix form of the Poisson equation. No coding needed here.

4 Method for solving Tridiagonal Matrix [2 pts]

For detailed question, check assigment sheet.

4.1 Show that they behave like O(n) with n the dimensionality of the problem

Take a look and run the file `Question4/Question4.cpp`. Run it using `make`. You will notice that number of floating point operations needed for are $8(n-1)$ which is order of n, where n is dimension of matrix.

4.2 Compare this with standard Gaussian elimination.

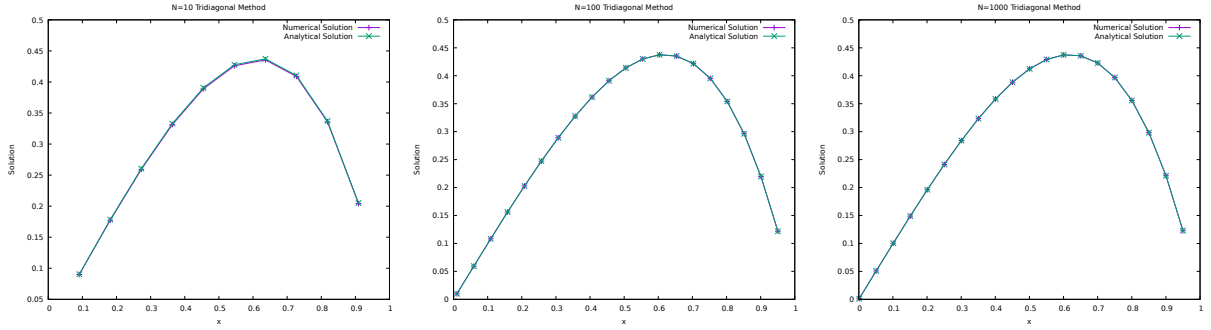
From the previous question you can notice that for the gauss elimination number of FLOPs(floating point operations) are $O(n^3)$

4.3 Solve the problem for matrices of the size 10, 100 and 1000.

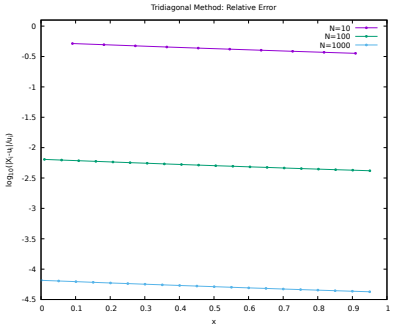
Again checkout `Question4/Question4.cpp`

4.4 Compare your results(make plots) with the analytic results for matrices of the size 10, 100 and 1000.

After you run `Question4/Question4.cpp`, you can produce plots using `gnuplot gnuplot.sh`. Plots are



4.5 Compute also the maximal relative error in the data



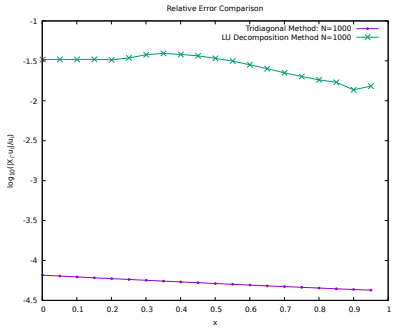
5 LU Decomposition [2 pts]

For this Question find the code in folder `Question5`. Run it by using `make`.

5.1 Question: Compare your results with those from the LU decomposition codes for the matrix of size 1000 $\times$ 1000.

We can compare number of floating point operations, relative error it introduces, etc.

- LU decomposition costs $\frac{2}{3}n^3 + O(n^2)$. To be precise $\frac{4n^3+3n^2-7n}{6}$
- While Tridiagonal Solver costs $O(n)$
- Forward substitution and backward substitution costs $O(n^2)$ each.
- Relative error for LU decomposition are calculated and shown for $n = 1000$. You can compare it with previously calculated using Tridiagonal Method. See below figure:



- From above points you can conclude that Tridiagonal solver much faster than LU decomposition. Also relative error is better compared to LU Decomposition method.

5.2 Question: Compare the time usage between LU decomposition and your tridiagonal solver:

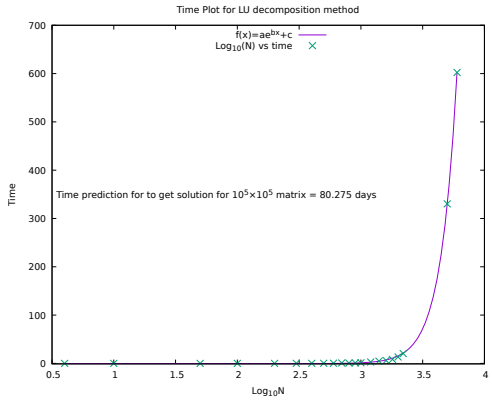
Time of computation for LUDecomposition method for n=1000 is: 0.001265 seconds
Time of computation for Tridiagonal method n=1000 is : 1.20647 seconds
You can clearly notice the difference. In fact for larger n you will notice that time taken by LUDecomposition method increases by large margin.

5.3 Can you run the standard LU decomposition code for a matrix of the size $10^5 \times 10^5$? Comment your results.

If you try to run the LU Decomposition code for such large number, it will take forever to complete your code. See, only for FLOPs this code requires to run at least n^3 iterations(i.e. more than 10^{15} operations). There are other operations such as setting size to the matrix etc which also takes CPU time.

However we can predict how much time it will take to find the solution for such large matrices. First we will find computation time for running a code for small n (ex: $n = 100, 200, .., 5000.$). We will store that data. Then we will fit appropriate funtion to predict how much time it will take to get solution for $10^5 \times 10^5$ matrix.

Below plot is of time taken for computation verses log of size of the matrix. We will can easily fit exponential function for these data points. *Fitted exponential function is predicting that it will take* 80 *days for that computation.*



However run the tridiagonal solver for $10^5 \times 10^5$ matrix and you get a solution in 0.009892 of seconds. Reason is simple We are required to do only 799992 FLOPs for getting solution from this method. So we can conclude that LUDecomposition is efficient but take toll on computational resource.

6 Using GSL Libraries [2 pts]

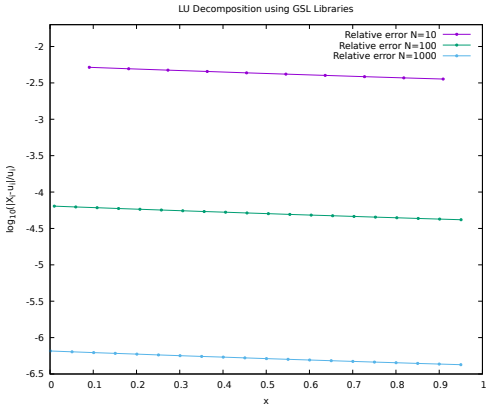
For detailed question go through assigment sheet.

6.1 Question: Obtain $u(x)$ numerically using the gsl library functions:

Detailed code is located in `Question6` directory. Run it using `make`.

6.2 Question: Compare the relative error between the analytical and the numerical solutions:

See below plot,



A Number of Floating Point Operations for Gaussian Elimination

2.1. **Operations count for Gauss elimination.** We consider the number of floating point operations ("flops") required to solve the system $Ax = b$. Gaussian Elimination first uses row operations to transform the problem into an equivalent problem of the form $Ux = b$, where U is upper triangular. Then back substitution is used to solve for x. First we look at how many floating point operations are required to reduce into upper triangular matrix.

First a multiplier is computed for each row. Then in each row the algorithm performs n multiplies and n adds. This gives a total of $(n-1) + (n-1)n$ multiplies (counting in the computing of the multiplier in each of the $(n-1)$ rows) and $(n-1)n$ adds.

In total this is $2n^2 - n - 1$ floating point operations to do a single pivot on the n by n system.

Then this has to be done recursively on the lower right subsystem, which is an $(n-1)$ by $(n-1)$ system. This requires $2(n-1)^2 - (n-1) - 1$ operations. Then this has to be done on the next subsystem, requiring $2(n-2)^2 - (n-2) - 1$ operations, and so on.

In total, then, we use I_n total floating point operations, with

$$I_n = \sum_{k=1}^{n} (2k^2 - k - 1) = \frac{n(n+1)(4n-1)}{6} - n \approx \frac{2}{3} n^3.$$

Counts for back substitution: To find x_n we just requires one division. Then to solve for x_{n-1} we requires 3 flops. Similarly, solving for x_{n-2} requires 5 flops. Thus in total back substitution requires B_n total floating point operations with

$$B_n = \sum_{k=1}^{n} (2k - 1) = n(n-1) - n = n(n-2) \approx n^2.$$