**Project 1**

Computational Physics I FYS3150/FYS4150

Hyejin Yun, Yisha Chen

**Abstract**

We study the limit and errors that occur when solving mathematical problems by computer programmed algorithms. We understand the method of algorithms solving a set of linear equations and each pros and cons of it. The algorithms we study are Tri-diagonal Matrix algorithm and LU decomposition. By increasing the number of linear equations we can see the limits of generating solutions through limited source of hardware or algorithm.

**Introduction**

The aim of this project is to get familiar with various vector and matrix operations, from dynamic memory allocation to the usage of programs in the library package of the course. Comparing the results of the solutions using several methods, we should be able to know which method is most efficient.

**Theoretical Models and Technicalities**

Due to installation problem, the code runs only partially. We could only get limited results therefore some of the tasks were not able to be managed precisely.

You can choose a flag for each solution. Flag ‘-t’ gives you the duration of time when each solution are run. Flag ‘-s’ executes the general solution, flag ‘-sc’ executes the optimized solution, flag ‘-sLU’ executes the LU decomposition, and flag ‘-err’ executes the calculation of relative error. Each solution generates a text file that writes every value of the solution. In case the error is calculated, we will receive a text file that contains every error of each solution. When you try to calculate the time or count the number of FLOPS (floating point operations per second) the results will be on the output window.

**Results and Discussion**

**(a) 1.** Show that you can rewrite this equation as a linear set of equations of the form

**Av** = **b**.

**2.** When , show .

1. The equation :

, where

Lets rearrange the elements.

The variables of unknown function v can be gathered as a set of vectors, and their coefficient can be written as a matrix to be multiplied to **v**. Since all of the coefficients of each equation are -1, -2, and -1, the element of the matrix will only contain -1 and -2 along the diagonal. Therefore, the matrix would be,

2.

Therefore,

**(b) 1.** Set up the general algorithm (assuming different values for the matrix elements) for solving this set of linear equations.

**2.** Find also the precise number of floating point operations needed to solve the above equations.

**3.** Then you should code the above algorithm and solve the problem for matrices of the size 10×10, 100×100 and 1000×1000.

**4.** Compare your results (make plots) with the closed-form solution for the different number of grid points in the interval x ∈(0,1).

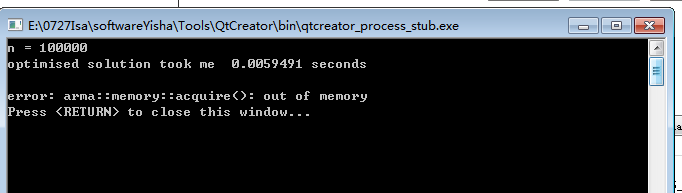
**(c) 1.** Specialize your algorithm to the special case and find the number of floating point operations for this specific tri-diagonal matrix.

**2.** Compare the CPU time with the general algorithm from the previous point for matrices up to n =106 grid points.

Numbers of FLOPS for specific tri-diagonal matrix and CPU time comparison between tri-diagonal matrix and the general algorithm

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| n | | 10 | 10^2 | 10^3 | 10^4 | 10^5 | 10^6 |
| FLOPS for optimized algorithm | | 27 | 297 | 2997 | 29997 | .. | ... |
| Time  (s) | Optimized algorithm  (tri-diagonal matrix) | 5.987e-06 | 1.3256e-05 | 7.2268e-05 | 0.00056138 | / | / |
| general algorithm | 0.00103399 | 0.0021548 | 0.133163 | 13.4013 | / | / |

Table1 time usage by solution



Picture 1 lack of memory

The number of FLOPS needed in optimized algorithm is given by 3\*(n-1).

When n is big enough, CPU time for optimized algorithm (tri-diagonal matrix algorithm) is much shorter than general algorithm (LU decomposition). We can see from the table (Table 1) that the time of general algorithm becomes larger than the optimized algorithm when n ≥103.

Due to lack of memory, the program could not generate results when n was set larger than 103.

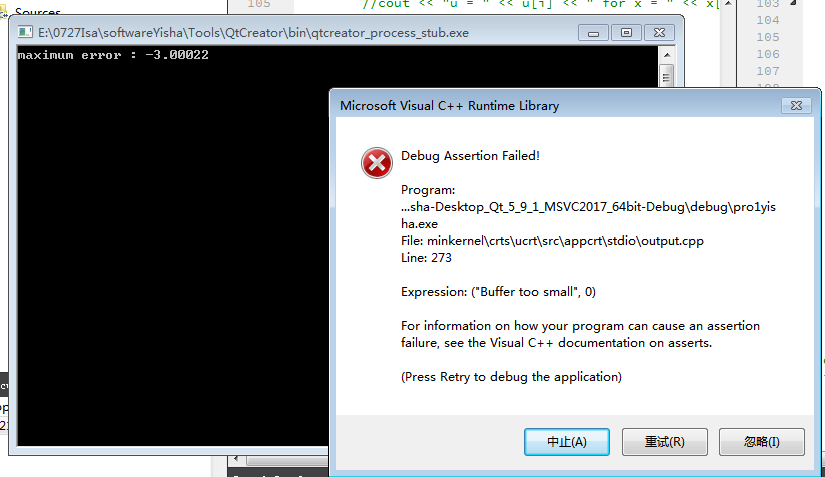
**(d) 1.** Compute the relative error

as function of log10(h) for the function values ui and vi. For each step length extract the max value of the relative error. Try to increase n to n=107. Make a table of the results and comment your results. You can use either the algorithm from b) or c).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| n | 10 | 100 | 1000 | 104 | 105 | 106 | 107 |
|  | -1.1797 | -3.08804 | -5.08005 | -7.07928 | -8.84297 | -6.07547 | -5.52523 |

Table 2 Max value of the relative error for different n

The max relative error is reduced with the increasing of n. The program did not make a file of errors when n is 1000 or larger due to assertion error.



Picture 2 error due to small buffer

**2.** Make a table of the results and comment the differences in execution time How many floating point operations does the LU decomposition use to solve the set of linear equations? Can you run the standard LU decomposition for a matrix of the size 105×105? Comment your results.

When you have n linear equations you need n3 floating point operations. Out of memory for LU decomposition when the matrix size is . In the case larger memory is available to use, the standard LU decomposition could be run, but with limited hardware, due to lack of memory, the operating system will kill the program. With LU decomposition, we could only get results until n was smaller than 1000. This differs to the size of memory that is able to use of each computer.

**Conclusion and Perspectives**

We found that the LU decomposition uses more memory compared to the optimized algorithm, while most of the values that are calculated as 0 in the case of generating the solution of a tri-diagonal matrix. Moreover, when the size of the matrix increases, due to the number of FLOPs, the time consumed by each algorithm has a different state of increase. The optimized algorithm is rather linear, while the LU decomposition increases more steeply. Therefore, in case of solving tri-diagonal matrices using optimized algorithm is more reasonable for both perspectives of hardware and software.

**Appendix with extra material**

Github address for full code :

<https://github.com/isabel2017/C.P.Projects-Yisha---Hyejin/blob/isabel2017-patch-1/project10922%20-%20finally.cpp>

**Bibliography**

*David Potter, Computational Physics, Imperial College, London, John Wiley & Sons, 1973, pg 82-87*