FYS3150 Homework 1

Christopher Parker
Computational Physics Department
University of Oslo
chris@chris-parker.net

I. Introduction

I seek to solve the Poisson equation $u'' = -f$ discretely in one dimension, on the interval $(0, 1)$ for known f and Dirichlet boundary conditions. This entails solving the system of equations $Av = b$. While b is related to f by discretization and a scaling factor of the reciprocal square of lattice spacing, v is simply the discretization of u. v is second order accurate and exactly u in the limit of vanishing lattice spacing, or equivalently, in the limit of an infinity of lattice points n.

For this project, both u & f were specified, so that error bounds could be calculated.

I developed and implemented an algorithm which improved upon the accepted algorithm, by exploiting the strong properties present in A. I compared my algorithm to the accepted implementation in Armadillo and the analytic solution.

The relevant code may be found at:
github.com/csp256/FYS3150_Homework1

II. Optimizations

The LU decomposition of a symmetric, tridiagonal, Toeplitz matrix A has four nonzero diagonals. The main diagonal of the L matrix is entirely 1, and the first off diagonal in the U matrix is entirely a (the off-diagonal elements in A). The first off diagonal in the L matrix (l) and the main diagonal of the U matrix (d) share a recurrence relation:

$$d_1 = b \tag{1}$$

$$l_i = \frac{a}{d_{i-1}} \tag{2}$$

$$d_i = b - a l_i \tag{3}$$

This recurrence relation has a closed form solution. For $b = 2$ & $a = -1$ successive terms are given by $d_i = \frac{i+1}{i}$ and $l_i = \frac{-1}{d_{i-1}}$. This permits A to be fully factored in $2n$ embarassingly parallel floating point division operations. However, it is not necessary to fully factor the matrix.

The Thomas algorithm is a well known, fast method for solving tridiagonal systems of equations. The matrix may optionally be LU decomposed prior to applying the algorithm. This causes the sole presence of l_i to be of the form $\frac{-a}{l_i}$, which is exactly d_{i-1}. It therefor suffices to calculate d_i and apply the Thomas algorithm with this replacement.

This is a significant improvement over LU decomposition's & Gaussian eliminations oft-cited complexity of $O\left(\frac{2}{3}n^3\right)$ (though faster algorithms exist for LU decomposition). The standard extention of Gaussian elimination to tridiagonal systems is $O(n)$, but roughly half as fast.

d_i can be computed in n FLOPs (embarassingly parallel), and the entire solution can be found in a cumulative $4n$ FLOPs. The corresponding implementation is $poissonSolver()$ in $matrixSolvers.cpp$. A fast ($6n$ FLOPs) implementation for arbitrary a and b is $tridiagonalSymmetricToeplitzSolver()$ in the same file. The author suspects the general case can be further improved by manipulating the closed form of the relevant recurrence relation. The author is certain that algorithms faster than the general case exist with less strict requirements than are present in $poissonSolver()$, such as when $|a| = 1$.

As an aside, in exploring this optimization the author learned that all symmetric, tridiagonal, Toeplitz matrices have identical eigenvectors and eigenvalues. The resulting inquiry to this has introduced the author to random matrix theory.

III. Summary of Results

A. Domain

My method works with $n = 10^8$ but fails with $n = 10^{8.5}$, solely due to memory constraints. In comparison, Armadillo works with $n = 10^4$ but exceeds available memory before $n = 10^{4.5}$ (due to storing all n^2 matrix elements). I could not get Armadillo's experimental sparse matrix capabilities to work at all.

B. Numerical Stability

My method agrees well with the solution provided by Armadillo's $solve()$ function, in the range which Armadillo was capable of operating. My method was noticably less accurate for $n = 10^{3.5}$ and a full order of magnitude less accurate for $n = 10^4$. Optimal stability for both methods is also achieved in that interval.

In my method the final term x_n has a $3n$ step dependency, which encourages the propogation of numerical errors when $n >> 1$. A natural further investigation would be to investigate stability with $doubles$ instead of the present $floats$, or by using an arbitrary precision arithmetic library.

My method assumes a benign (positive definite) matrix for stability, but is optimized for a positive semidefinite matrix.

C. *Speed*

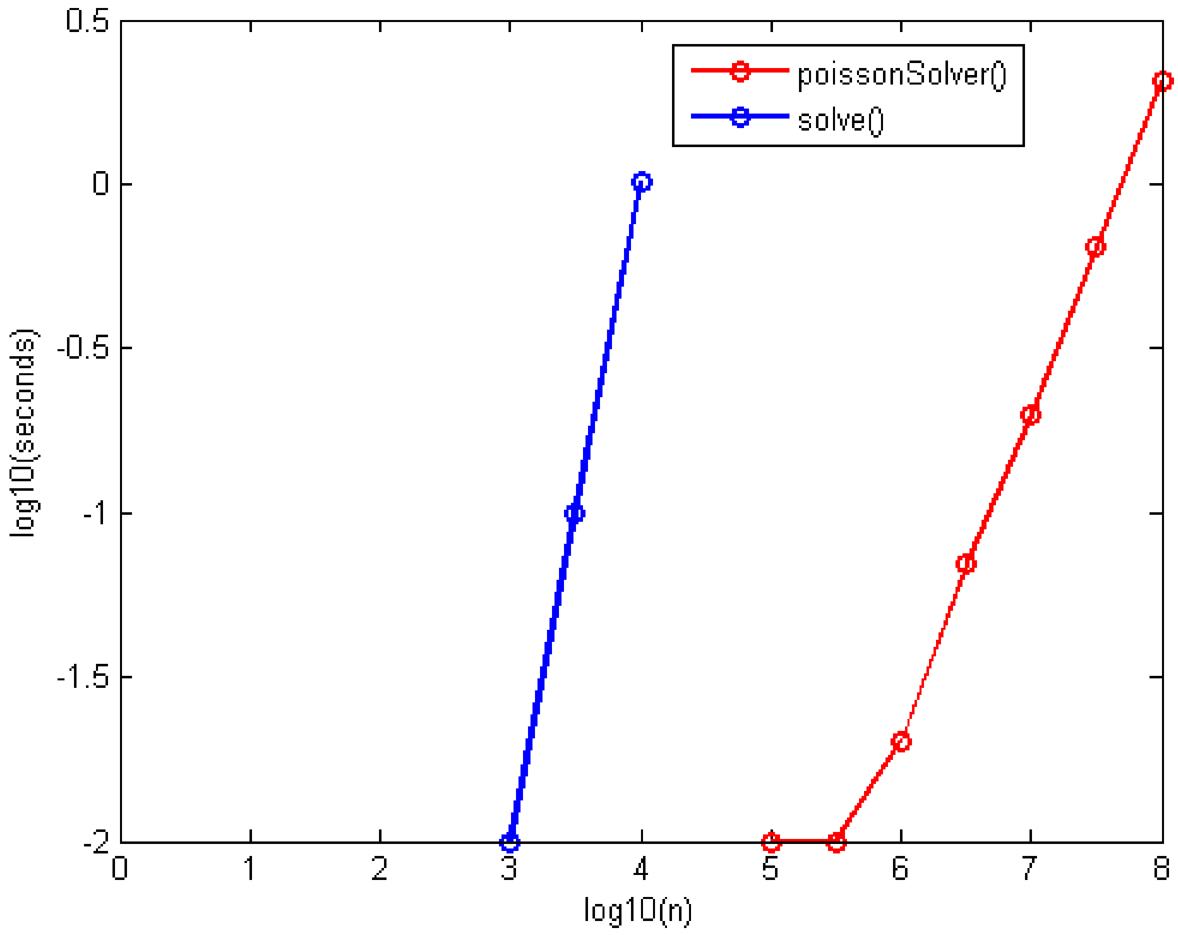
My method is faster than Armadillo for every measurable case. The timing function was not capable of resolving time periods less than $10^{-2}s$. QTcreator gave impenetrable errors when trying to use two other standard (more precise) timing functions. My $poissonSolver()$ is linear in time while Armadillo's $solve()$ is quadratic at best.

IV. Conclusion

My implementation is fast and scales strongly, but suffers minor numerical instability. This might be remedied by switching to double precision.

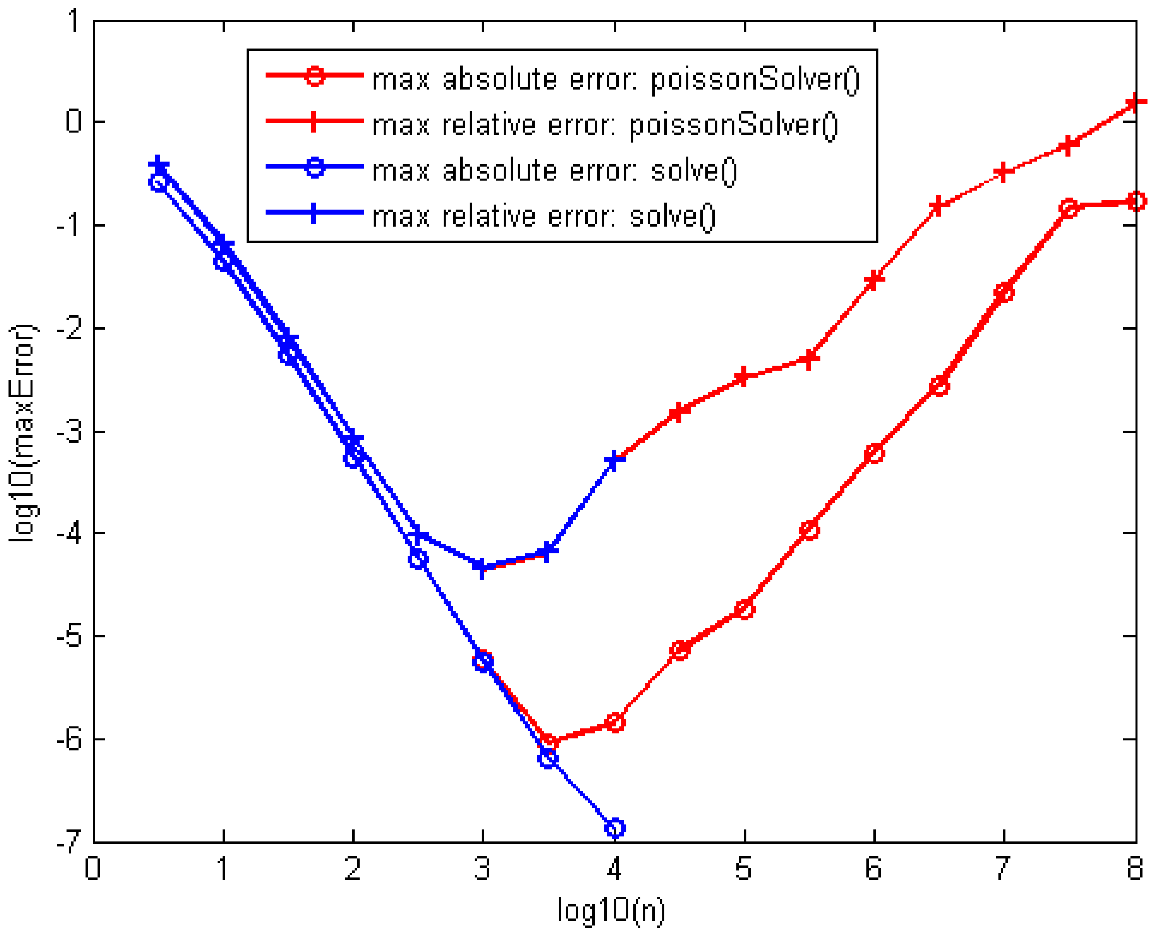
Optimizing the more general solver algorithm $tridiagonalSymmetricToeplitzSolver()$ was left undone, but is likely to yield good results.

Figure 1. Performance



$n = 10^k$	My method [seconds]	Armadillo's $solve()$ [seconds]
0.5	0	0
1	0	0
1.5	0	0
2	0	0
2.5	0	0
3	0	0.01
3.5	0	0.1
4	0	1.01
4.5	0	-
5	0.01	-
5.5	0.01	-
6	0.02	-
6.5	0.07	-
7	0.2	-
7.5	0.64	-
8	2.06	-

Figure 2. Maximum Error Margins



$n = 10^k$	$poissonSolver()$ $log_{10}\left(MaxError\right)$	$log_{10}\left(MaxRelativeError\right)$	Armadillo's $solve()$ $log_{10}\left(MaxError\right)$	$log_{10}\left(MaxRelativeError\right)$
0.5	-0.582973	-0.407701	-0.582973	-0.407701
1	-1.35891	-1.1797	-1.35892	-1.1797
1.5	-2.26612	-2.09156	-2.26613	-2.09157
2	-3.26211	-3.08724	-3.26207	-3.08735
2.5	-4.25952	-4.01674	-4.25493	-4.0126
3	-5.23795	-4.34564	-5.24985	-4.34328
3.5	-6.04863	-4.18512	-6.20424	-4.17909
4	-5.84451	-3.30048	-6.87417	-3.30161
4.5	-5.14554	-2.82121	-	-
5	-4.73758	-2.4849	-	-
5.5	-3.96584	-2.30661	-	-
6	-3.21302	-1.53862	-	-
6.5	-2.56504	-0.820207	-	-
7	-1.64842	-0.485974	-	-
7.5	-0.835712	-0.224452	-	-
8	-0.770185	0.183236	-	-