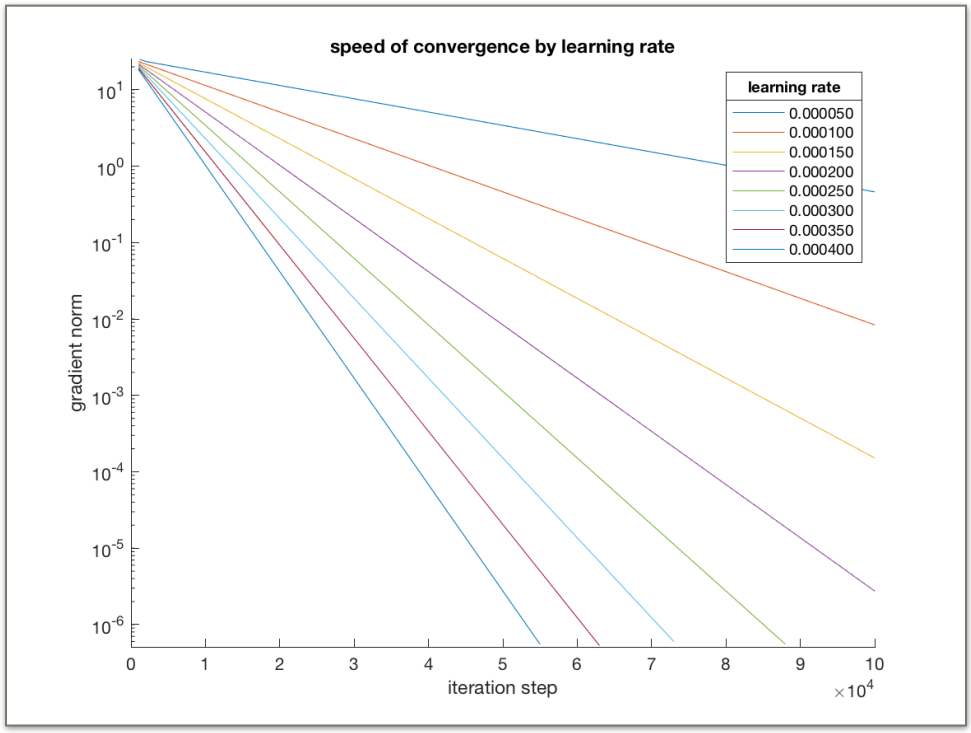
# Homework Set 5

**By Alexi Chryssanthou**

## 1.3 Analysis

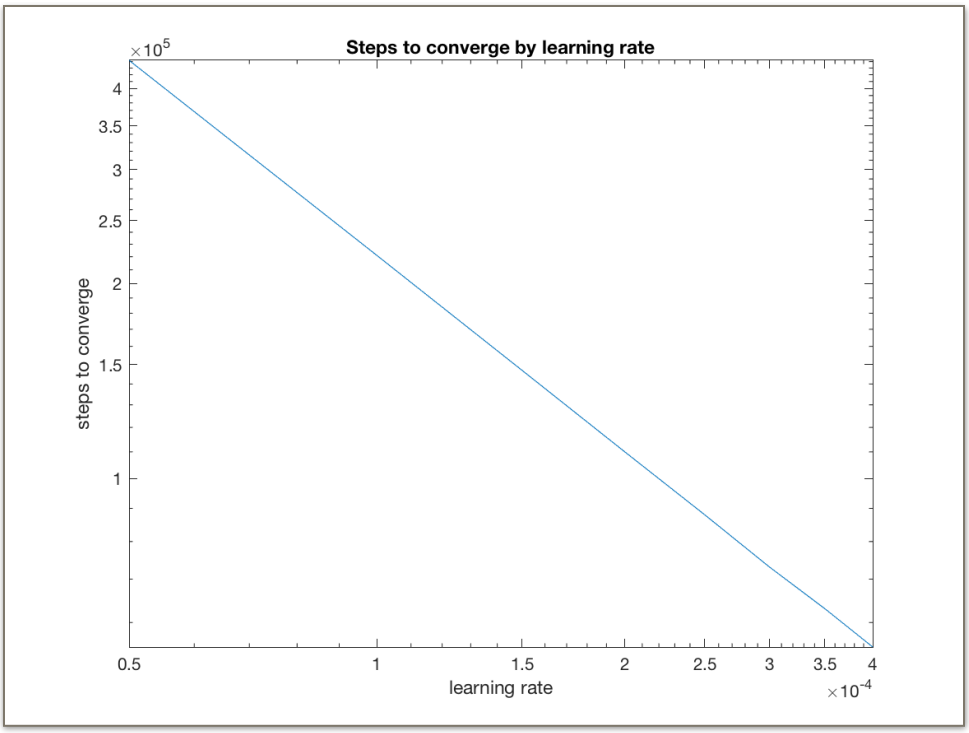
Regardless of learning rate, the resulting solution was the same, namely h = 3.308750, m = 6.726965, and b = 34.711305. The speed of convergence was proportional to the size of the learning rate; the higher the learning rate, the quicker it would converge.




If the learning rate was too high, then the process would diverge, i.e. blow up to infinity. The fastest learning rate I could find before the process started slowing down was 0.000428455, which converged on the solution in 51,635 steps. The slowest rate I tested was 0.00001 which converged in 2,212,640 steps (not plotted).

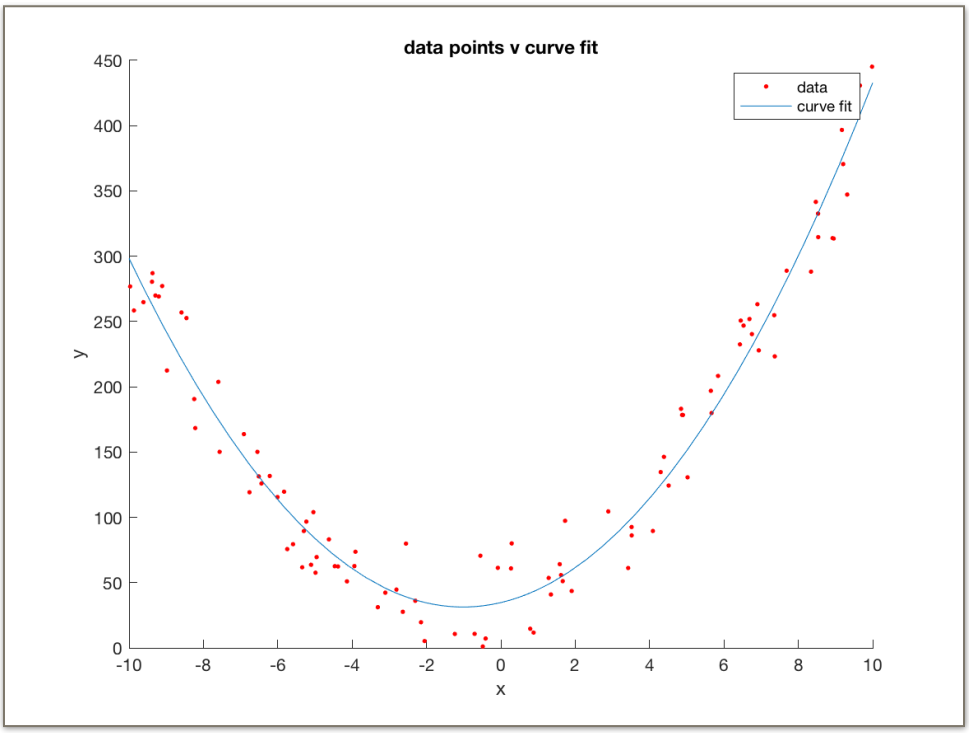
The process would diverge around a learning rate of 0.00043.

The gradient of the error function is dependent on the first derivative of the error function with respect to each coefficient. If we can't solve for the first derivative analytically, then we can use numerical differentiation for finding the first derivative, and then calculating the gradient from that numerical differentiation.




## 1.4 Analysis

The coefficients returned were h = 3.308750, m = 6.726965, and b = 34.711305, Giving us $f(x) = 3.30875x^2 + 6.726965x + 34.711305$ for our curve. The following graph plots the data points along with the found curve.



## 2.4 2D Heat Equation with Preconditioned Conjugate Gradient

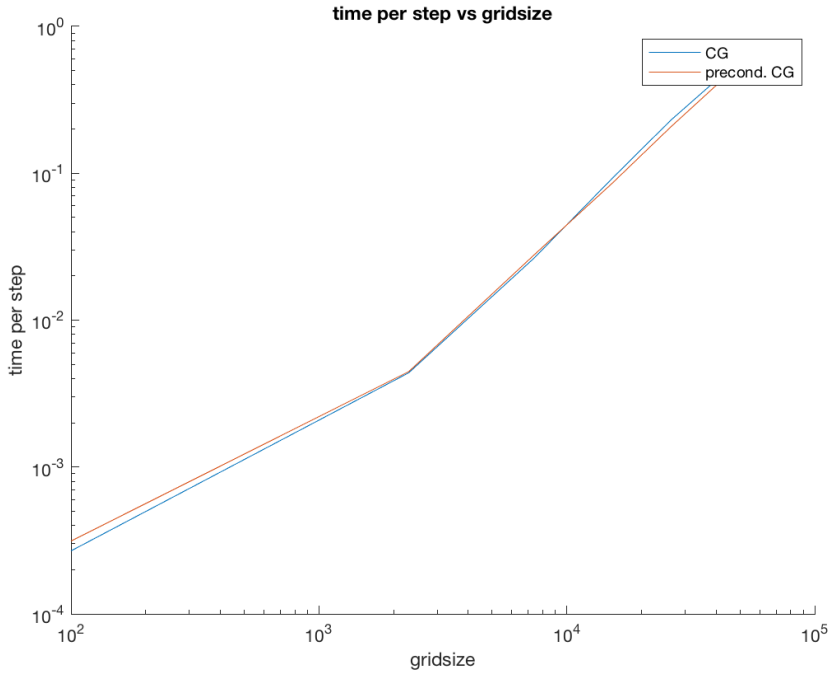
Testing the regular Conjugate Gradient and the preconditioned Conjugate Gradient for a variety of N's yielded similar results. The timings for a grid size of 10,000 over 200 time steps confirm the similarity:

| (in seconds) | CG | CG w/ Jacobi PC |
|---|---|---|
| **Time taken at 1:** | 0.29 | 0.29 |
| **Time taken at 66:** | 5.23 | 5.15 |
| **Time taken at 133:** | 7.42 | 7.23 |
| **Time taken at 200** | 8.72 | 8.56 |

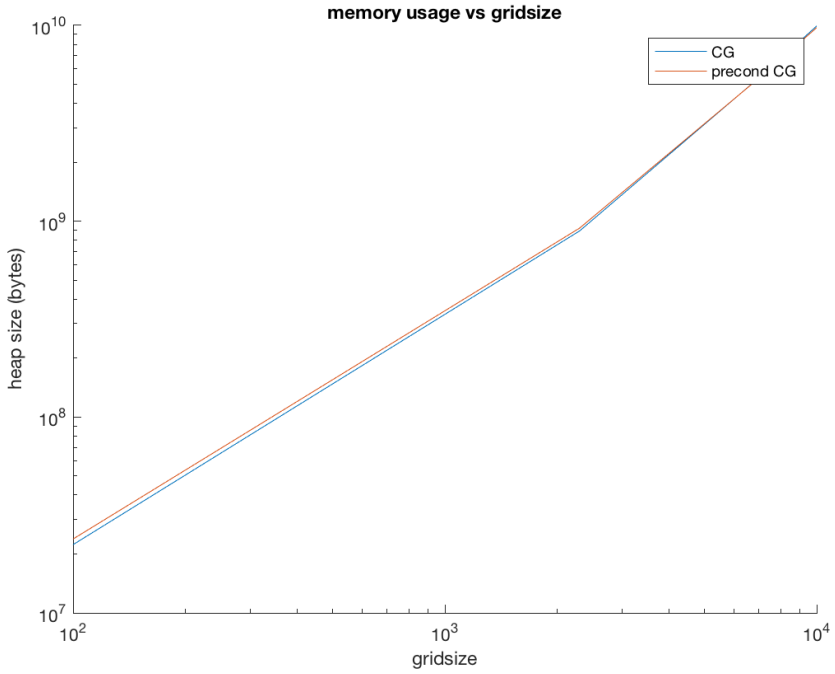
Using values: N 100, NT 200, L 4, T 1, a 1, dN: 0.040404, dT: 0.005000, Sx: 2.000000, Sy: 2.000000, C: 3.062812 for both simulations.

## 2.5 CG Performance Analysis

Plotting the time per step versus grid size for CG and preconditioned CG further illustrate the similarities:



Except for one extra array Z in the preconditioned CG, the memory usage between methods is the same, and grow at the same rate:



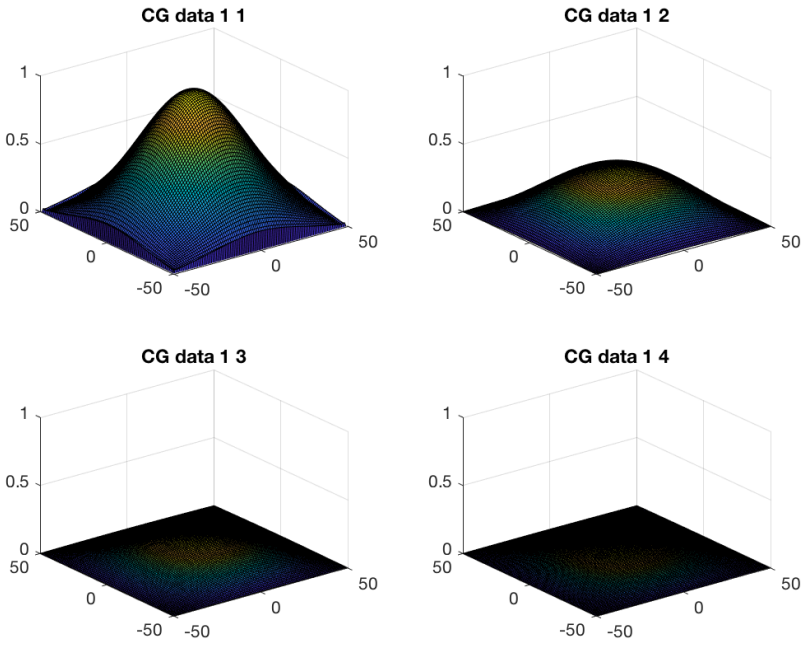

As for comparing to other methods, I wrote problem 3 in Matlab, problem 4 in python, and problem 5 in C++, so comparing the results across languages will be difficult.

Without question, C++ would perform all methods the most quickly. Method to method, it's easy to see that Conjugate gradient is the quickest iterative solver with the smallest memory consumption.

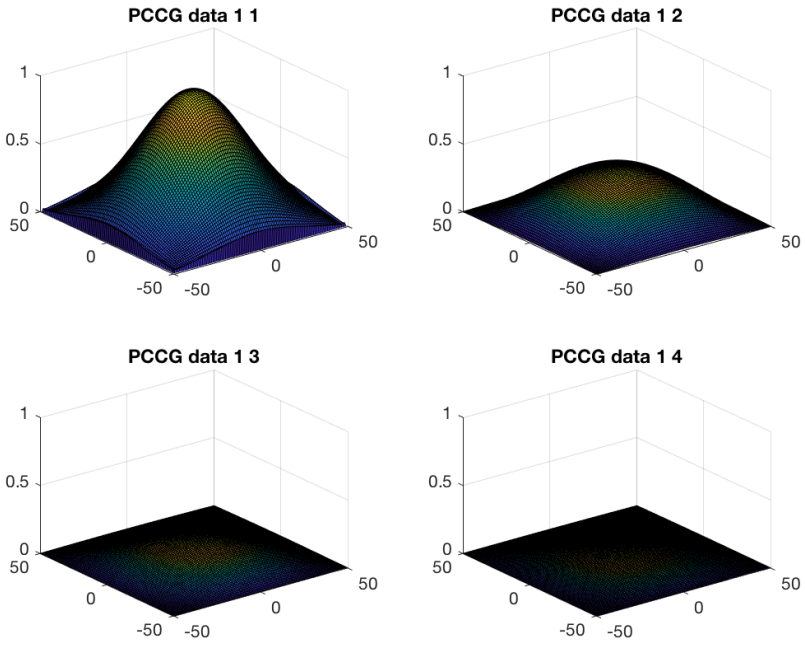
As for preconditioning, the Jacobi preconditioner did not make a significant difference, which is to be expected considering it simply scales the solution vector by a scalar (namely, 1/1+4C).

## 2.6 Plotting

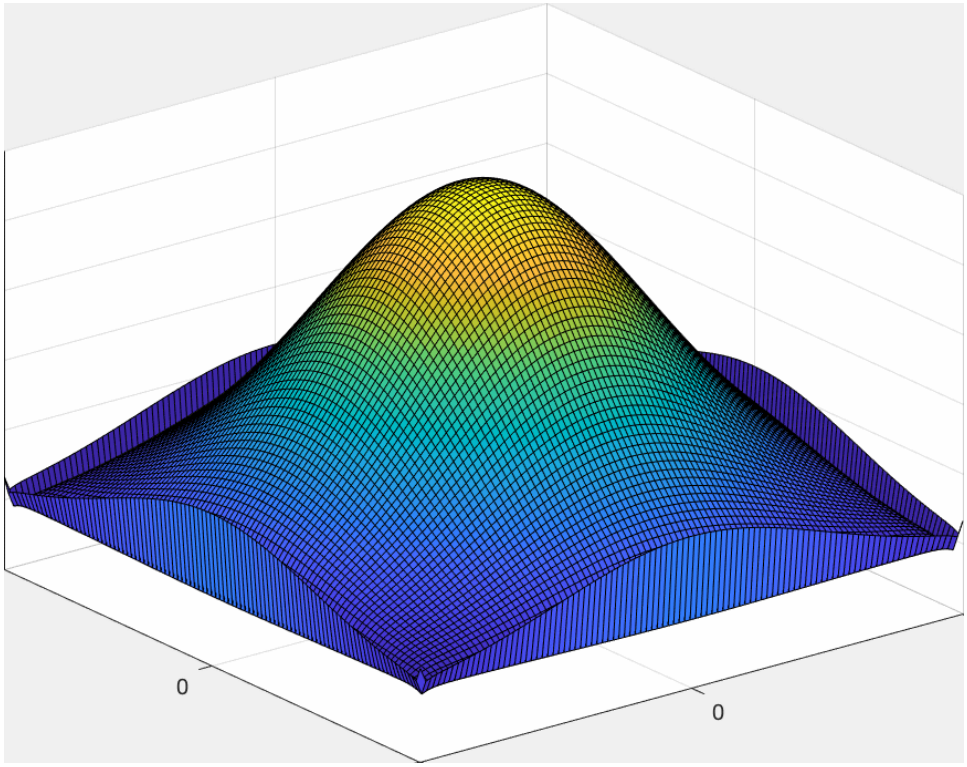
These four figures show Conjugate Gradient at work starting with the heat equation in its initial state:




These four figures show Jacobi preconditioned Conjugate Gradient at work, again, starting with the heat equation in its initial state:



A movie of the algorithm at work simulating the heat equation's state over time:




(reference: 2D_CN_CG.gif)