Corey Schulz
CS 3200 – Spring 2020
April 1, 2020

Homework 4 Report
Iterative Methods

Question 1

Considering a system of equations, Ax = b, where A is defined by

$$\begin{pmatrix} 9, -4, 1, 0, 0, 0, 0, 0 \\ -4, 6, -4, 1, 0, 0, 0, 0 \\ 1, -4, 6, -4, 1, 0, 0, 0 \\ 0, 1, -4, 6, -4, 1, 0, 0 \\ 0, 0, 1, -4, 6, -4, 1, 0 \\ 0, 0, 0, 1, -4, 6, -4, 1 \\ 0, 0, 0, 0, 1, -4, 5, 2 \\ 0, 0, 0, 0, 0, 1, -2, 1 \end{pmatrix}$$

and

$$b = 1/N^4[1,1,1,1,\cdots,1]^T, N = 8$$

This system can be solved using iterative solver methods! In my solution, I did this in `question_1.m`, with the Jacobi and Gauss-Seidel algorithms defined in `jacobi.m` and `gauss-seidel.m` files respectively, using a convergence parameter in both cases to check for completion each iteration. I also keep track of the iterations using an incrementing variable in each of these algorithms.

From testing with a tolerance of 1.0e-5, it seems that the Gauss method works better than the Jacobi method in terms of accuracy – Gauss got the same solution as the Matlab solver, in 101 iterations. However, the Jacobi solution was off a bit from the Matlab solution, but only took five iterations.

So, answering which method is better is a matter of what you're looking for in the method. If you're going for accuracy – which seems to be the case most of the time – then you should use Gauss. If you're going for a fast completion time and just care about getting in the right ballpark in terms of accuracy, then the Jacobi iterative solving

method may be the one to go with. These results can be seen by running the question 1 script!

Question 2

Extending the linear system in the previous question, where A is now an arbitrary N, defined by:

$$\begin{pmatrix} 9, -4, 1, 0, \ldots & ,0 \\ -4, 6, -4, 1, 0, \ldots & ,0 \\ 1, -4, 6, -4, 1, 0 & \\ 0, 1, & 1, 0, \ldots, 0 \\ \ddots\ddots & \ddots\ddots\ddots \\ & 0, 1, -4, 6, -4, 1 \\ & 0, 1, -4, 5, 2 \\ 0, \ldots\ldots, & 0, 1, -2, 1 \end{pmatrix}$$

And,

$$b = 1/n^4[1,1,1,1,\cdots,1]^T$$

I setup a script to solve the linear system using the Gauss method under different combinations of N and tolerance.
The possible N values were in {16, 32, 64} and the possible tolerance values were in {`1.0e-5, 1.0e-7, 1.0e-9, 1.0e-11, 1.0e-13, 1.0e-15`}.

Included on the next page is a table that shows how the number of iterations varies with tolerance and the size of N.

N Value	Tolerance Value	Gauss-Seidel Iterations
16	1.0e-5	330
16	1.0e-7	3514
16	1.0e-9	6697
16	1.0e-11	9881
16	1.0e-13	13064
16	1.0e-15	16248
32	1.0e-5	1
32	1.0e-7	32697
32	1.0e-9	88273
32	1.0e-11	143850
32	1.0e-13	199427
32	1.0e-15	254982
64	1.0e-5	1
64	1.0e-7	60747
64	1.0e-9	991439
64	1.0e-11	1922155
64	1.0e-13	2852869
64	1.0e-15	3783218

As you can see, the number of iterations explodes as the value of N gets larger and the iterations get smaller.

However, interestingly, the value goes up linearly as the tolerance goes down. In the case of N = 16, as the tolerance goes down by a rate of 1.0e-2 each step, the number of iterations grows by about 3,000 each step.

Similarly, for N = 32, as the tolerance goes down by a rate of 1.0e-2 each step, the number of iterations grows by about 50,000 each step.

And also, for N = 64, as the tolerance goes down by a rate of 1.0e-2 each step, the number of iterations grows by about 900,000 each step.

So it *is* possible to predict how many iterations a given solution will take with this method, given enough data points. Of course, as N gets large and tolerance gets small, the computation time jumps drastically. All the cases for N = {16, 32} completed very quickly, but you might have to wait a while for the N = 64 scenarios.

The data in this question can be obtained by running `question_2.m`.

Question 3

For this question, I modified the Gauss-Seidel algorithm, now in `gauss_seidel_relax.m`, using a new method to update x:

$$x^{k,new} = \omega x^{k} + (1-\omega)x^{k-1}$$

With and an omega value of 0.5. I used the same sets of N values as in the previous question and ran the tests again. The results are reported in the table below.

N Value	Tolerance Value	Gauss-Seidel Iterations
16	1.0e-5	2
16	1.0e-7	8217
16	1.0e-9	17698
16	1.0e-11	27179
16	1.0e-13	36660
16	1.0e-15	46141
32	1.0e-5	1
32	1.0e-7	58284
32	1.0e-9	224717
32	1.0e-11	391150
32	1.0e-13	557583
32	1.0e-15	724009
64	1.0e-5	1
64	1.0e-7	1
64	1.0e-9	2307572
64	1.0e-11	5098499
64	1.0e-13	7889427
64	1.0e-15	10679691

These results are rather interesting. Using the omega value when updating x every iteration, when the tolerance is sufficiently large, can cause the iterations in some cases to drop down to 1, significantly reducing computation time. However, if the tolerance is smaller than allowing the solution to immediately converge, the number of iterations required to converge – and so too the computation time – rise drastically, quickly overtaking a solution where omega is not used when updating x.

Here, again, the number of computations rise linearly with a given N value as the tolerance value grows smaller. The jumps here are bigger than in question 2, however.

It can be a challenge to find a good omega value here that works with a certain solution. If the omega value doesn't work for the solution, it can potentially cause increased computation required to converge. However, if a good value of omega is chosen, it can greatly reduce converging time. It's a tradeoff as are so many other optimization problems.

These results can be seen by running `question_3.m`. Note that the computation time might take up to a couple of minutes when calculating the N = 64 values because they just take so many iterations here.

Overall, in the general case, omega cannot reduce computation time or cause less iterations, but it can in some cases, so it might be worth it to find the right omega value depending on the problem you're trying to solve.