Sarah Mahl

Intro to Numerical Methods

1/30/21

HW3

2. (a) To investigate the convergence of the Jacobi and Gauss-Seidel methods for *Ax = b*, I set the iterations for my code at *N* = 20. With *A = (a\*I + R)* and the initial guess *x* random matrices, with the tolerance set at 1e-08, I ran the code five times. Jacobi’s method converged within the tolerance within a range of 5 to 7 iterations, while Gauss-Seidel did not converge within 20 iterations.

(b) To observe the convergence as *N* changes, I set *N* to be 5, 10, 50, and 100. For *N* = 5, both the Jacobi and Gauss-Seidel methods did not converge within the tolerance. For *N* = 10, 50, and 100, the Jacobi method converged to within the tolerance after 6, 6, and 8 iterations respectively, which is consistent with the previous trend observed described in part (a). The Gauss-Seidel method, however, did not converge within the tolerance for any values of *N*. This is also consistent with the previously observed trend.

Based on the theorems we have discussed in class, this leads me to believe that the randomly generated matrix *A* is not diagonally dominant or positive definite. If *A* were diagonally dominant, both Jacobi and Gauss-Seidel iterations would converge to the exact solution for the system *Ax = b*, which is not the case. If *A* were positive definite, the Gauss-Seidel iterations would converge to the exact solution, which is also not true based on the output from my code. Because only the Jacobi method converges, though, it can most likely be found that at least one of the eigenvalues of , where *A = N - P*, is less than 1. This would mean that the error for the iterations converges to 0 as the approximations converge to the exact solution of the system.

I would like to mention, however, that when I tested my Gauss-Seidel code with the matrices *A* = [[10,3,1],[2,-10,3],[1,3,10]] and *b* = [14,-5,14] with an initial guess of *x* = [0,0,0] and a tolerance of 1e-08--the example we looked at in class--I received the exact solution to the system, but my output printed that the iterations did not converge to within the tolerance given. Thus, it is possible that the Gauss-Seidel method for the randomly generated matrix *A* of size 22x22 with a random initial guess *x* of size 22 converged within the tolerance of 1e-08. If it did converge, this would mean that *A* is both diagonally dominant and positive defiant. It would also mean that at least one of the eigenvalues of *M* for both methods are below 1.

(c) The trends observed in part (b) occurred when *a* = 10 for A = (a\*I + R). To observe the changes that happen when we change *a*, I set *a* to 5, 25, 50, and 100 and kept N = 20 constant. As observed, as the values of *a* increased, the Jacobi iterations decreased. At *a* = 5, Jacobi converged within 9 iterations, but at *a* = 25, the Jacobi method converged within 5 iterations. At both *a* = 50 and *a* = 100, Jacobi converged within 4 iterations. To see if this trend would continue, I continued to increase *a* to 150 and 200, but the method converged at 4 iterations both times. I then set the value for *a* to be 1 and 2. At *a* = 1, the method did not converge within 20 iterations, but at *a =* 2, the method converged within 14 iterations. This is consistent with the trend described when I first set *a* to 5, 25, 50, and 100. The trend observed for the Gauss-Seidel iterations, however, remained the same and did not converge to the solution, regardless of the changing values for *a*.

3. (a,b) For all methods to solve the system *Wx = p*, I set the initial guess *x*  to be a random matrix with the same length as the given matrix *p*. The results from the Gauss-Seidel method and the “backslash” operator ended up being the same. The solution found with the Jacobi differed from the solutions found with the Gauss-Seidel iterations and the backslash operator. I will discuss the results of the LU decomposition and the inverse of *W* further on.

(c) (i) The Jacobi method for *Wx = p* converged after 5 iterations, and the time it took to run was about 78.75 seconds.

(ii) Gauss-Seidel, to contrast, did not converge in the set *N* = 20 iterations, but solving for the solution of the system only took about 2.6 seconds.

(iii) I was unable to properly download the SciPy library needed for the built-in LU decomposition function in Python, so instead, I found the LU decomposition of *W* using Julia to be able to look at the results given. This process took a couple minutes, and while I did not calculate the exact number for the time, it was clear that the calculation took awhile.

(iv) Solving for the inverse of W only took about 0.18 seconds.

(v) The “backslash” operator ran just as quickly, solving the system over about 0.071 seconds.

(d) The Jacobi method, while accurate if coded correctly, takes much longer to converge to the solution of a linear system than the Gauss-Seidel method. Given small systems, the speed of the method is less of an issue, especially since the operations of both Jacobi and Gauss-Seidel are order but when given such large systems as with *W* and *p*, this becomes something to consider. Both methods are useful because they will converge to a solution regardless of the initial guess (as shown with our random matrices), but with systems as large as our given *Wx = p,* it’s impractical to spend over a minute solving a single linear system. If we have to solve multiple systems of that size, the operation becomes even more time consuming. Similarly, because the inverse of a matrix is an order operation, solving a linear system using the inverse of that matrix also takes a long time. Though it did not take long with our *Wx = p* system, compounding that operation with others makes the time it takes to solve that system longer than necessary. Again, considering solving multiple systems, this would be impractical. The LU Decomposition method also took a very long time. For very large systems, this is not ideal, and ends up taking away from the time needed to solve the system. It can be helpful because it is an example of a “direct” method, and is thus very straightforward in solving a linear system, but that depends on the system being solved. The “backslash” operator, however, was the fastest method of all. Because this is a built-in method in Python, the code behind the function is optimized to be as fast as possible. This makes it the most practical method when solving for the exact solution of a linear system, though other methods such as the Gauss-Seidel allow for more of an analysis on the convergence towards that solution.