Maxwell Kaplan, Daniel Ansher, Jacob Goodpasture

Professor Wang

Math 2605

Fall 2015

**Part 1: The Symmetric Pascal Matrix**

1. Why is it justified to use the LU or QR−factorizations as opposed of calculating an inverse martix?

The main reason why we use LU/QR factorizations as opposed to calculating the inverse matrix mainly has to do with the probability of the introduction of error into the calculations. Depending on the size of the matrix, the larger it is, the more probability it will be susceptible to error. For example, if you have a 4x4 matrix. You will be solving four linear systems and then multiply the inverses by your b matrix. The larger the system becomes, the more difficult it becomes to solve with regard to time complexity. It is more efficient to just use LU/QR factorizations and then apply forward or backward substitution.

1. What is the benefit of using LU or QR−factorizations in this way? (Your answer should consider the benefit in terms of conditioning error.)

Not only is using LU/QR factorizations more efficient than standard methods, but in general for Ax = b, cond of A describes the change that must be made to x to account for a change in b. Therefore using LU/QR factorizations also reduces error because cond of A = cond of QR = cond of Q \* cond of R and since cond of Q = 1, it becomes that cond of A is equal to cond of R. R is a much simpler matrix to use in computations.

For the plot part of the project, this can be ran using ‘java pascalplot’. The graph generates random matrices ranging from 1x1 to 4x4 dimensions. We can compare the algorithms that we have been using whether it be LU, QR Givens, or QR Householder. Below is our findings with regard to each of these algorithms:  
  


In the graph above, we can see the the general trend of errors through each type of algorithm. The graph is generated using JavaFX, where 5 iterations of each algorithm is performed on random matrices ranging from 1x1 to 4x4 dimensions. It is obvious to see that LU is the least error prone. Judging by these results and countless other trials to generate other graphs, the QR Householder algorithm generally proves to be the most error prone.



Trial 2: Householder still has the highest error approximation. QR Givens is second highest (aside from an outlier for LU)



Trial 3: Householder again holds true to being the most error prone.

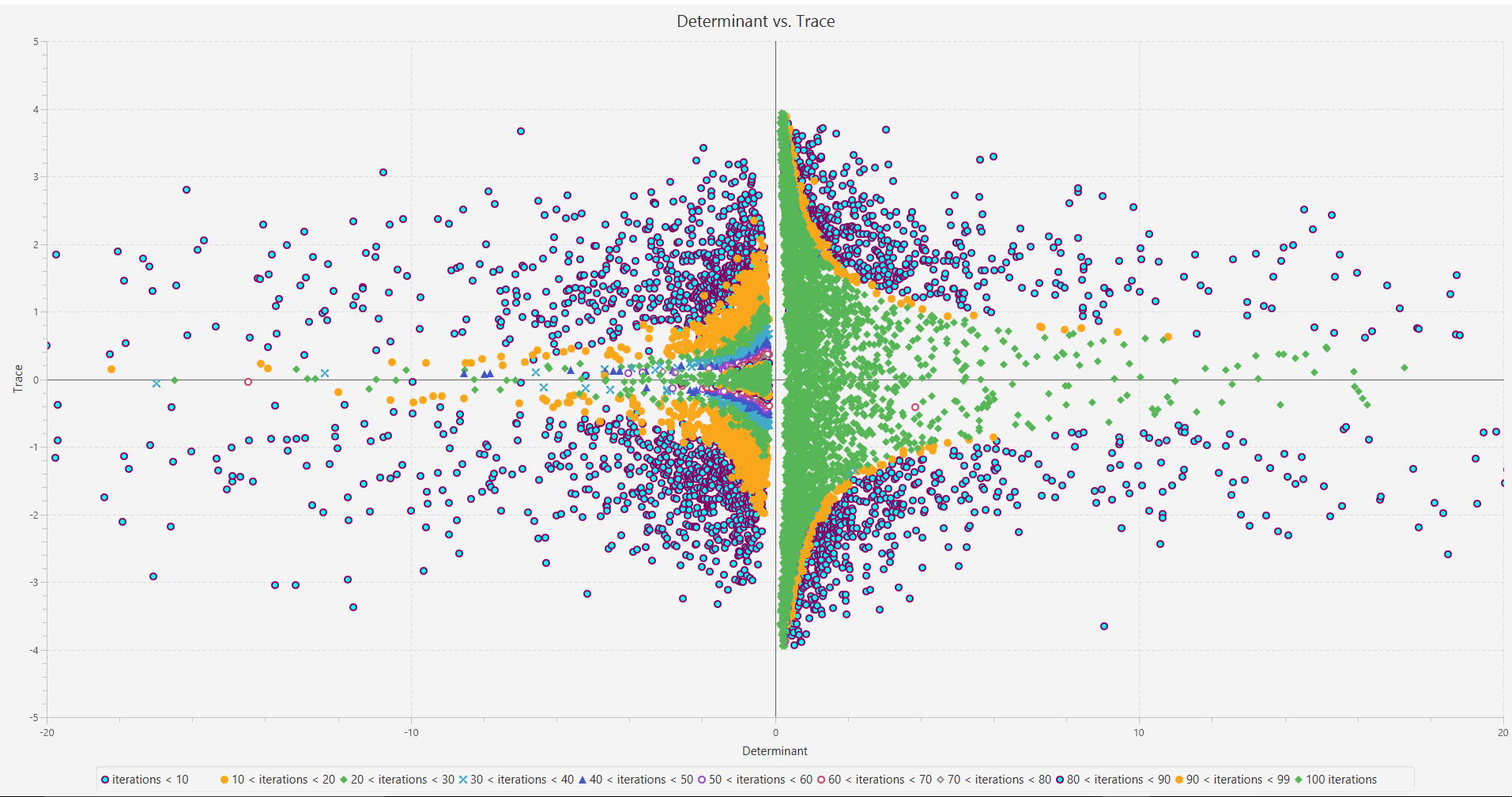
**Part 2: Iterative Methods**

This part of the project on the two iterative methods Jacobi and Gauss-Seidel. These methods are used to approximate solutions to the equations Ax=b for a given matrix. Part C of the outline stated that we were to find the solution to the about equation for 100 randomly generated vectors using both Jacobi iteration and Gauss-Seidel iterations. After finding these vectors an average vector was calculated for each type of iteration and then that average vector was compared to the exact answer and an approximate error for the vector was found. Lastly, the average number of iterations that each vector using each type of iteration was found and a ratio was made of the average number of Jacobi iterations to the average number of Gauss-Seidel iterations. What our group found after running these tests is that Jacobi iteration was found to take two and a half more times the number of iterations but was also significantly more accurate in how close it came to the exact vector after 100 iterations than when the Gauss-Seidel method of iteration was used.

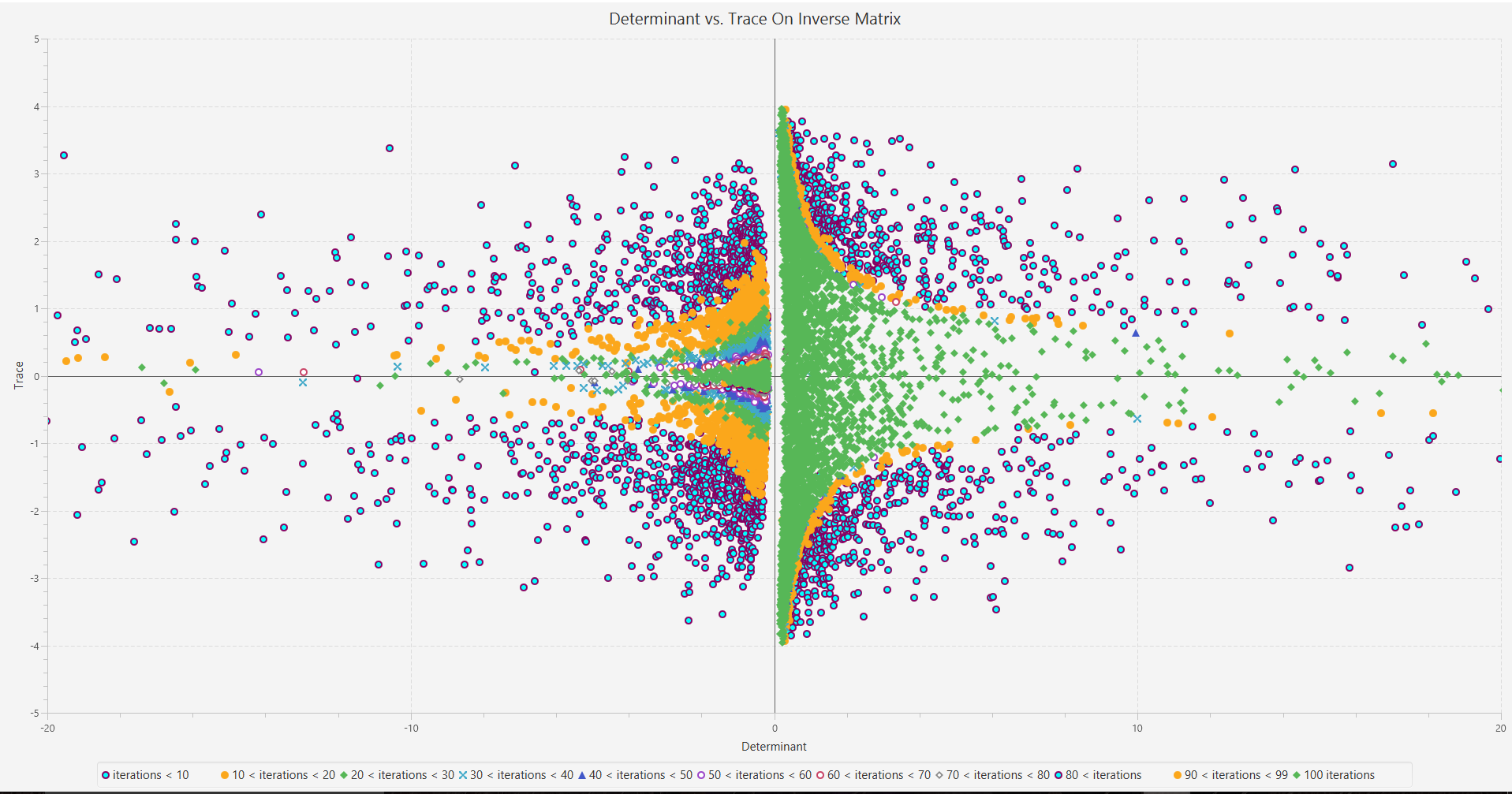
As shown in the graph on the next page there is a very noticeable distinction between the points plotted using Jacobi iteration and the points plotted using Gauss-Seidel iteration. Those points plotted using Jacobi iteration sit higher than every one of their Gauss-Seidel counterparts. This is because it takes on average a little over double the number of iterations to reach the allowable tolerance using Jacobi iteration. Also something to note about the points that use the Jacobi method is that the majority of the points sit inside the range of 0.4-1.2 meaning that when given a random 3x1 vector between [-1, 1] there is a high chance that the initial error to the given answer vector will lie somewhere in that range. This same observation can also be made about the Gauss-Seidel points, they also lie within approximately that same 0.4-1.2 range. Another difference that can be noticed between the two sets of data points is that the Jacobi points are generally clumped together and if you were to put a line of best fit through the points it would hit many of the points that were graphed. On the other hand, looking at the points plotted using Gauss-Seidel it seems to be that there are two distinct level at which the average number of iterations occurs. This implies that number of iterations varies more greatly in Gauss-Seidel iteration than in Jacobi Iteration

**Part 3: Convergence of the Power Method**

These graphs of the determinant vs. trace of many randomly generated 2\*2 matrices shows the distribution of iterations across them. Major observations that can be made from this is that the matrices which iterated 100 times, the maximum possible amount, are most highly concentrated with determinants from 0 to 1 and ranges from -4 to 4, especially around the x-axis. On the other hand, matrices which iterated less than 10 times are most commonly found to have a determinant from -1 to 0 with higher magnitudes trace values. As the magnitude of trace values approaches 0 on both side, the amount of iterations needed increases until reaching the 100 iteration cap right around the x-axis. Likewise, as the magnitude of the determinant gets smaller, the amount of iterations also increases. This causes the graph to look like an exponential function as it approaches the axes from each side, having more iterations as the points get closer to the origin. The reason for this graph’s shape has to do with the fact that a smaller trace means the diagonal values of the matrix are smaller. This causes the power method to run more and more times as the trace gets smaller and smaller.



Determinant vs. Trace with 10000 Matrices

Determinant vs. Trace with 10000 Inverse Matrix Iterations