Robert Matthew Keezer

Calculus 3

Programming Project

3/31/2015

**Intro**

For this project, I decided to use the Java programming language. Since precision and rounding error is important in this assignment, I chose to use the Java primitive “double”. Double is a double floating point value with 64 bits of precision instead of 32 bits of a float. Despite this increase in precision, it still has inaccuracy in LU decomposition and QR factorization algorithms. Theoretically, the 64 bit double should be able to reach as low as 2^-63, however, due to using several multiplication and addition methods to find error of the methods used in this project, the lowest error achieved (besides 0.0) was 2^-55 or 2.7755576e-17.

**Part One**

In the first part of this assignment, I implemented LU decomposition, QR householder factorization, and QR givens factorization. The Hilbert matrix was then used to measure the precision of the algorithms on the computer. Due to repeating decimal fractions in the Hilbert matrix such as (1/3) or (0.3333...), accuracy was lost in the multiplication and division parts of the LU decomposition and QR factorization. We are able to create six graphs using the infinity norm with these three equations:

|  |  |  |
| --- | --- | --- |
| ||LU −H||∞ | ||QR−H||∞ | ||Hxsol−b||∞ |

The first three graphs use the first two equations and show the errors of the L, U, Q, and R Matrices due to the floating point error:

|  |  |
| --- | --- |
|  |  |

As the graphs show for the QR Householder and Givens, the error for both similarly began at 1e-16 error but the householder quicker became worse whereas givens remained constant. This agrees with the methods because Givens is a slower method but has better accuracy. LU on the other hand was very accurate. There was 0.0 error in the first two iterations, meaning we can’t detect the error. Afterwards there was a constant error of 2^-55, which is the lowest error possible in our calculations.

The next three graphs were created using the third algorithm to calculate the X solution error for each method:

|  |  |
| --- | --- |
|  |  |

These three graphs turned out to be very similar. The X solution error for all of these methods began very small close to 1e-17. As n increased and the size of the Hilbert matrix increased, the error grew to as high as 1e-12. However, the error receded in all three of these graphs as the matrix grew more because there may be an increase of repeating number fractions in the low numbers, but the ratio of repeating to not repeating fractions declines later on.

**Why is it justified to use the LU or QR-factorizations as opposed of calculating an inverse matrix?**

If we wanted to calculate an inverse matrix directly by using Gaussian elimination, we would be using division for the majority of the calculations. Division in the computer is one of the most intensive operations causing the program to be very slow and it would be useless in operations involving thousands of numbers. The division operations in Gaussian elimination also bring up the possibility of dividing by zero which needs to be checked for. Also, these operations will cause a large error to form due to the extensive use of division. With LU or QR factorizations, they minimally use division and instead replace it with quicker multiplication and back substitution. This means they are faster and have a smaller error than Gaussian elimination.

**What is the benefit of using LU or QR-factorizations in this way?**

As stated in the previous question, using LU or QR factorizations in this way allows for much quicker computations and more reliable accuracies. The use of division to find the inverse causes many numerical complications. When dividing, the computer is required to do many computations in the background that cause repeating number fractions to lose their original accuracy because they computer can only carry decimals to so many places. Using multiplication and back substitution is much faster and more reliable.