1. The chart below summarizes the errors for LU, Householder and Givens factorizations. The errors have been plotted as a function of *n*, e.g. a 2.0 on the x-axis corresponds to the 2x2 Hilbert matrix. All the errors are very small. Nonetheless, LU factorization yielded a consistent error 0 for every Hilbert matrix (2x2 – 20x20).

**SEE GRAPH.JPG (attachment)**

* 1. Why is it justified to use *LU* or *QR* factorizations as opposed to calculating an inverse matrix?
     1. LU can be used for any matrix; finds all solutions; is easy to program; and is fast.
     2. First, computing the LU and QR factorizations is not an overly expensive operation: the algorithms I used have running times of O(n2) for QR and O(n3) for LU. Once the factorizations have been computed, solving the system Ax = b is quite trivial. In the case of LU, forward substitution can be used with a lower-triangular matrix and backward substitution can be used with an upper-triangular system to quickly solve Ax = b in O(n2) time. In the case of QR, since Qt = Q-1, solving Ax = b is also trivial. Solving both systems using QR or LU is easy to implement. On the other hand, taking a matrix inverse is quite expensive and is not so straightforward to code. QR and LU decompose A into quickly solvable sub-problems. On the other hand, to use a matrix inversion algorithm like Gauss elimination requires separately solving each right-hand-side vector *bj*; overall Gauss elimination independently costs *O(rN3)*, where *r* is the number of rows. This is more expensive than LU and QR.
     3. Second, since cond(Q) = 1 while cond(A) is greater than or equal to 1, orthogonal transformations are the most stable ones, providing the highest degree of precision. This is because QR factorization decomposes the original problem Ax = b into two problems, Qy = b, which has no error amplification, and Rx = y, which has the minimal possible error amplification inherent in the original problem. However, since I used Python, I was forced to round down some of my calculations in the Givens and Householder functions; this is because after “killing off” an entry or series of entries, 0 would not result, but rather a very small number approximating 0. In fact, I rounded down my answers to 14 decimal places to force a 15-decimal-place number to round to 0. This was necessary to make the Givens rotations and Householder reflections work.
     4. Third, QR but not LU can be used to decompose not only square but rectangular matrices. Inverting a matrix requires that it be square. So, while this assignment did not ask us to work with rectangular matrices, generally speaking, QR can be used to solve rectangular matrices while algorithms that use matrix inversions might not work in this case.
  2. What is the benefit of using LU or QR-factorizations in this way?
     1. As mentioned above, QR is as well-conditioned as the original problem. The conditioning error is minimized as much as possible. Conditioning is a property of the matrix A that determines whether it is feasible for an algorithm to provide a numerical solution to a linear system involving A. Since any linear system on a computer has some error, it is important to reduce perturbation or imprecision as much as possible. LU and QR help to accomplish this. Unfortunately, LU is prone to instability, meaning that if both *L* and *U* do not have small condition numbers, then instability in the calculated answer will be inevitable. There are methods to combat LU-factorization’s instability however, by permuting the rows of the matrix *A* (not done in my code).