Comprehensive Analysis of Computational Decompositions of Matrices

Brian Wang
Wei Wang
Karl Nicodemus

The purpose of Matrix decompositions is to create an easier way to compute solutions to large linear system of equations. With LU decomposition, solving with forward and backward substitution with lower and upper triangular matrices make it really exceptionally easy to solve for matrices. With QR decomposition, having an orthogonal matrix makes it easy to solve for Qy = b because the inverse of Q is Q^t. You can then use backward substitution to solve for the solution vector since R is an upper triangular matrix. But why is it not preferable to compute the inverse to solve for linear systems?

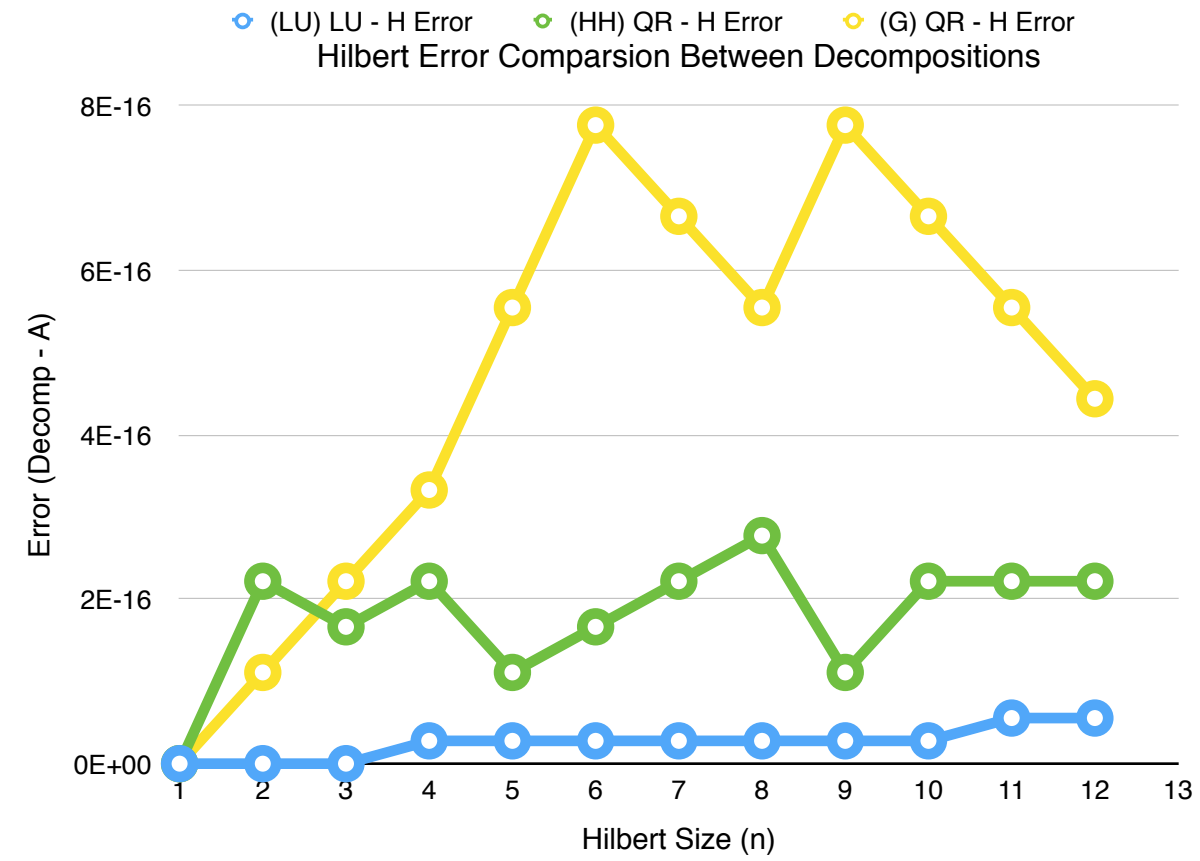
i) We want to avoid using inverses because calculating inverses require row reducing matrices to echelon form, which is computationally difficult, requires more transformations, has more error, and is time inefficient. It requires more transformations cause not only do you have to transform lower triangular area, you have to transform the upper triangular area, comparing literally every single value in the matrix with the pivots. In contrast, LU, Householder, and Givens only row reduces the lower triangular area, which halves the number of transformations used, and thus more time efficient. Although it may be appropriate to row reduce with a piece of paper and pen, it is not efficient with matrices of larger size in computation. It is justified to program with LU and QR factorizations because it produces less error with fewer transformations, is easier to program, and is more time efficient.

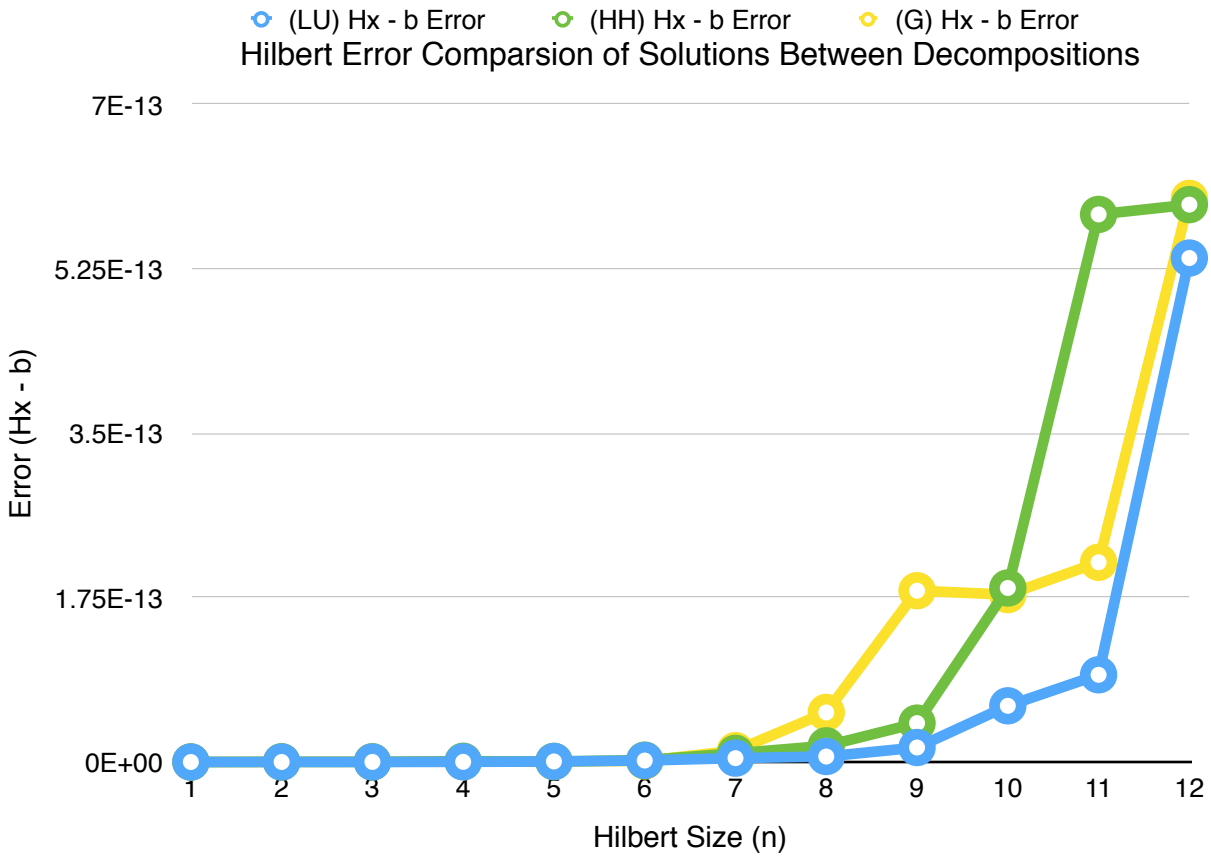
ii) Using LU and QR factorization is not only easier to program, but the error for computing solutions for large matrices is relatively small, no bigger than 10^-13 for matrices of size 10-15. In large-scale real-world projects, such as creating visual physics engines that require vector calculus and solving for linear system of equations, accuracy can be sacrificed for time, lag reduction, and code efficiency. Since finding the inverse requires more transformations, it has the potential to produce more errors. Compared to the error of finding the inverse matrix, since LU and QR factorizations require fewer transformations, they have less error than computing the transformations for row reducing echelon form.

How large these errors are is dependent on how large the matrix size is. To observe the relationship between matrix size n and errors from decompositions and solutions of Ax = b, we tested the LU, HouseHolder QR, and Givens QR decompositions with Hilbert matrices of given inputted size n, and found solutions of the equation Hx = b where b = 0.1^(n/3) * (1, 1, 1, …, 1)^t. Below are the results of the errors ||LU - H||, ||QR-H|| (HouseHolder HH), ||QR - H||(Givens G), and each respective decomposition solution error ||Hx-b||.

Error Table of Decompositions of the Hilbert Matrix

	(LU) LU - H Error	(HH) QR - H Error	(G) QR - H Error	(LU) Hx - b Error	(HH) Hx - b Error	(G) Hx - b Error
n = 1	0	0	0	0	0	0
n = 2	Approx. 0	2.2204460 * 10^-16	1.1102230 * 10^-16	2.7755576 * 10^-17	1.9428902 * 10^-16	5.5511151 * 10^-17
n = 3	Approx. 0	1.6653345 * 10^-16	2.2204460 * 10^-16	8.3266727 * 10^-17	8.3266727 * 10^-17	3.6082248 * 10^-16
n = 4	2.7755576 * 10^-17	2.2204460 * 10^-16	3.3306690 * 10^-16	1.5959456 * 10^-16	8.2572837 * 10^-16	2.8449465 * 10^-16
n = 5	2.7755576 * 10^-17	1.1102230 * 10^-16	5.5511151 * 10^-16	5.7939764 * 10^-16	5.7939764 * 10^-16	5.3082538 * 10^-16
n = 6	2.7755576 * 10^-17	1.6653345 * 10^-16	7.7715611 * 10^-16	1.5612511 * 10^-15	1.9914625 * 10^-15	1.1032841 * 10^-15
n = 7	2.7755576 * 10^-17	2.2204460 * 10^-16	6.6613381 * 10^-16	4.0939474 * 10^-15	9.2287288 * 10^-15	1.2087553 * 10^-14
n = 8	2.7755576 * 10^-17	2.7755575 * 10^-16	5.5511151 * 10^-16	5.8473191 * 10^-15	1.7245319 * 10^-14	5.2772456 * 10^-14
n = 9	2.7755576 * 10^-17	1.1102230 * 10^-16	7.7715611 * 10^-16	1.5432750 * 10^-14	4.1410668 * 10^-14	1.8241029 * 10^-13
n = 10	2.7755576 * 10^-17	2.2204460 * 10^-16	6.6613381 * 10^-16	5.9828498 * 10^-14	1.8530874 * 10^-13	1.7820331 * 10^-13
n = 11	5.5511151 * 10^-17	2.2204460 * 10^-16	5.5511151 * 10^-16	9.2851643 * 10^-14	5.8312613 * 10^-13	2.1268173 * 10^-13
n = 12	5.5511151 * 10^-17	2.2204460 * 10^-16	4.4408921 * 10^-16	5.3669284 * 10^-13	5.9353626 * 10^-13	6.0017553 * 10^-13


(LU) LU - H Error
(HH) QR - H Error
(G) QR - H Error
Hilbert Error Comparsion Between Decompositions
8E-16
6E-16
4E-16
2E-16
0E+00
Error (Decomp - A)
1
2
3
4
5
6
7
8
9
10
11
12
13
Hilbert Size (n)


(LU) Hx - b Error
(HH) Hx - b Error
(G) Hx - b Error
Hilbert Error Comparsion of Solutions Between Decompositions
7E-13
5.25E-13
3.5E-13
1.75E-13
0E+00
Error (Hx - b)
1
2
3
4
5
6
7
8
9
10
11
12
Hilbert Size (n)

Data Analaysis

From the data above, we can see that generally for the error ||Decomp - H|| (with low correlation), LU produces the lowest errors, Householder produces relatively low errors, and Givens produces the largest errors. Givens probably produces the largest errors because it's designed for matrices that mostly consist of zeros, so that it can skip over the zeros. However, since the Hilbert matrix is fully filled with non-zero doubles, It ended up producing a large amount of error.

This is the same case for solution errors ||Hx - b||, where LU produces the least error, Householder produces the next least, and Givens produces the most. We can see a general pattern. The errors became exponentially larger in respect to matrix size. Towards the end, the errors shot up as it went past n = 10, a sign of an exponential relationship between solution error and matrix size.

It seems as though LU is the most preferable and most accurate decomposition. However, on larger scale matrices, LU may be inefficient due to the number of transformations it makes. It has to go through every single pivot and do comparisons for every single value in the lower triangular area. Compare to Givens, although Givens does comparisons for every single value in the lower triangular area, Givens has the potential to skip over values of 0, which means in terms of time, it may be more efficient than LU, albeit more inaccurate. As for householder, it seems to be the balance between the two. It doesn't have to do comparisons for every lower triangular, but instead uses the first vector of every householder matrix iteration. From what our data tell us, it is a balance between accuracy and time efficiency.

Granted that these numbers may not be entirely correct, especially for the errors for ||Decomp - H||. Errors in coding inefficiencies can result in different error values. When we were programming these algorithms, changing the code just a little bit (like separating functions into parts), the error value changed. Our algorithm may also not be the most optimal algorithm out there as Java is not an linear algebra friendly language.

Conclusion

We now understand the purpose of matrix decompositions, and their respective specializations: LU is more accurate but not time efficient, Givens is less accurate but more time efficient with matrices with 0s, and Householder is the balance between accuracy and time-efficiency. Knowing these different specializations and ways to solve systems of equations will greatly help in developing large scale algorithm systems that require vector calculus, such as physics engines, simulations, and modeling. One thing is for sure, computing decompositions is a lot better, more efficient, and more accurate than computing the inverse.