An Overview on the Effect of Floating Point Errors in Matrix Decompositions

Olabanji Shonibare, James Wright

April 27, 2012

Abstract

Matrix Decompositions are useful for solving systems of linear equations. Soft errors in floating point computations caused by neutron and alpha particle strikes are a problem when accuracy of the computation is of utmost importance. While redundant computations can be used to detect and overcome such phenomena, this paper looks at the effects of a floating point error on several matrix decompositions and on the solutions to linear equations solved with these erroneous matrix decompositions.

1 Introduction

In situations when the accuracy of the solution to the linear system is of utmost importance, considerations for phenomena which can cause rare soft errors in the memory of a computer, computations are done several times to ensure the accuracy of a result unaffected by such phenomena. In this paper we will look at the possible effects to the solution of a linear system when this phenomena is not accounted for and such a phenomena does occur.

A soft error, which results by a bit changing in the active memory of a computer, can drastically change the results of a numerical computation. However, instead of trying to model the change of a random bit in active memory, we will be looking at what happens if the accuracy of a 64 bit floating point number in the course of the matrix decomposition is degraded down to that of a 32-bit floating point number.

Specifically we will look at the effects of such phenomena on several matrix decompositions commonly used in the solving of linear systems, specifically the LU, QR, QR via Gram-Schmidt, and Cholesky Decompositions. We will not be looking at sparse versions of these decompoitions but expect our results to carry over quite similarly when the same errors happen in the sparse decompositions.

2 Code

To look at the effect of a floating point error in a matrix decomposition, we had to write our own code for the matrix decompositions: LU, QR, QR via Gram-Schmidt, and Cholesky [2]. For various reasons we choose to implement the decompositions in Mathematica due to our skill level with the software and its ease of use in rapid prototyping. Specifically we implemented: the LU Decomposition without pivoting, the QR Decomposition with Householder reflectors, QR Decomposition via a Double Orthogonalized Gram-Schmidt Procedure, and a recursive Outer-Product form of the Cholesky Decomposition.

These functions were then modified to degrade the accuracy of a number in the middle of the matrix decomposition down to that of a single from a double. The function to degrade the accuracy of a double into that of a single simulates that of a missed-cast from a double into a single using the default rounding scheme of "Round to Nearest, Tie to Even" which is the default floating point rounding mode according to the IEEE standard [1].

Here below we show display our code which degrades the accuracy of a double value in Mathematica to that of a single, it does so by first turning the number into a base to digit and then rounding at the appropriate place for the equivalent loss of accuracy via the default floating point rounding scheme:

```
Code for the Degrading the Accuracy of a double to that of a single;
Simulates the default rounding for a cast from a double to a single:
(* This Implements the "Round to Nearest, Tie to Even" Rounding Scheme *)
Degrade2[number_]:=Module[{n,chop=23,i,m},
If[number!=0,
m=RealDigits[number,2];
n=m[[1]];
If[n[[chop+1]]==1 && Total[n[[chop+2;;-1]]]>=1,
n[[chop]]=n[[chop]]+1;
Do[
If[n[[i+1]]==2,
n[[i]]=n[[i]]+1;
n[[i+1]]=0;
];
,{i,chop-1,1,-1}];
If[n[[1]]==2,n[[1]]=1];
];
```

We also submit an example of our LU Decomposition code which degrades the accuracy of the 50th entry on the diagonal of a matrix (LUloss1):

LUloss1 is a function which degrades the accuracy of a double in the position (50,50) in the matrix A in the middle of the LU decomposition.

Code for the LU decomposition with a loss of accuracy on the diagonal at (50,50):

```
LUloss1[A_] := Module[{LU = A, n = Length[A], d, e, r, i, j},
  Do[
   If[r == 50, LU[[r, r]] = Degrade2[LU[[50, 50]]];];
   d = 1/LU[[r, r]];
   Do[
    e = d LU[[i, r]];
    LU[[i, r]] = e;
    Do[
     LU[[i, j]] -= e* LU[[r, j]],
     {j, r + 1, n}
     ],
    {i, r + 1, n, 1}],
   {r, 1, n - 1, 1}];
  {LowerTriangularize[LU, -1] +
    SparseArray[Band[{1, 1}] -> 1, {n, n}],
   UpperTriangularize[LU]}
  ]
```

These errors occur both on and off the diagonal. The case where the error occurs on a row/column which the algorithm has already operated on results in an error in x equivalent to that of the loss of accuracy due to the down-cast, this was expected, and is hence omitted from this presentation.

In all we have may different modified versions of each of the matrix decompositions so that we can observe the effect of a floating point error in varying places in the matrix decomposition as well as in the matrix. Each decomposition has an error occur when the algorithm reaches the 50th row/column, as each of these algorithms processes the matrix by row/column. A table of these functions, and the specific location of the error they induce is below:

Table of decomposition functions which simulate a floating point error in the process of the decomposition.	
LUloss1	A loss of precision at (50,50) before any operations have been done to the 50th row.
LUloss2	A loss of precision at (50,60) before any operations have been done to the 50th row.
LUloss3	A loss of precision at (55,50) after the constant for the row operation has been found, but not yet used.
LUloss4	A loss of precision at (55,50) after the constant for the row operation has been found has already been used.
QRloss1	A loss of precision at (50,50) in R before the householder is constructed from the 50th column.
QRloss2	A loss of precision at (50,60) in R before the householder is constructed from the 50th column.
QRloss3	A loss of precision at (50,50) in the householder after it is constructed but before it's used.
QRloss4	A loss of precision at (50,50) in the householder after it is constructed and used on R, but before it is accumulated into Q.
GSO1	A loss of precision at (50,50) in A before the 50th column has been orthogonalized.
GSO2	A loss of precision in the one of the coefficients for the first orthogonalization of the 50th column, before it is orthogonalized.
GSO3	A loss of precision in the **all** of the coefficients for the first orthogonalization of the 50th column, before it is orthogonalized.
GSO4	A loss of precision in the one of the coefficients for the second orthogonalization of the 50th column, before it is orthogonalized (for the second time).
GSO5	A loss of precision in the **all** of the coefficients for the second orthogonalization of the 50th column, before it is orthogonalized (for the second time).
GSO6	A loss of precision in (50,50) after the column has been orthogonalized twice and scaled.
GSO7	A loss of precision in (50,50) after the column has been orthogonalized twice and before it has been scaled.
OutCholloss1	A loss of precision at (50,50) before the Cholesky has operated on the sub-matrix starting at the corner (50,50).
OutCholloss2	A loss of precision at (50,70) before the Cholesky has operated on the sub-matrix starting at the corner (50,50).

Scenarios which have been omitted were ones where the error was quite predictable. You'll notice that all of the cases are when the error happens on a part of the matrix which the algorithms have already operated on are not present, that is because this amounts to a single floating point error in the

resulting decompositions, which brings the error of the solutions to the system $A.x = b$ to increase to what you would expect of single arithmetic.

3 Results

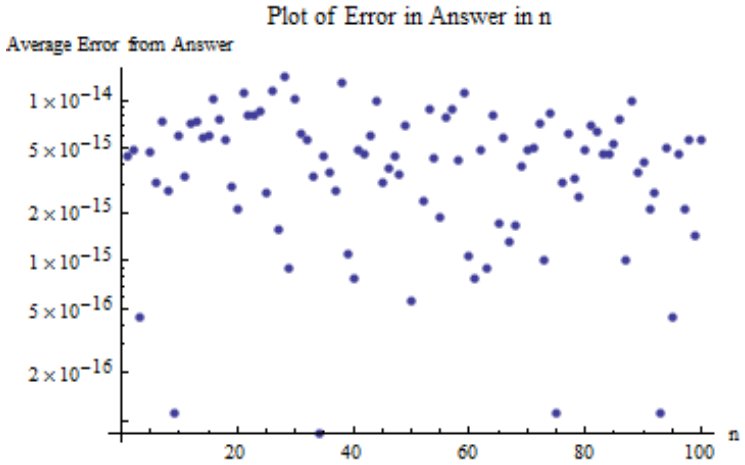
Of principal interest to us are the accuracy of the solutions to the system $Ax = b$, when A is decomposed with a floating point error in the middle of the algorithm. The following simulations were done with a square matrix of dimension 100, which was positive definite and symmetric, and had eigen-values equally spaced between 1 and 100. Below you will find a table of average relative accuracies for each of the functions we listed under section 3.

Function	Average Relative Error of the Answer (50 reps.)
LUloss1	$1.66855 * 10^-8$
LUloss2	$7.80196 * 10^-10$
LUloss3	$2.18903 * 10^-9$
LUloss4	$2.15568 * 10^-9$
QRloss1	$2.81688 * 10^-8$
QRloss2	$2.04667 * 10^-9$
QRloss3	$1.52377 * 10^-14$
QRloss4	$4.21604 * 10^-9$
GSO1	$1.28923 * 10^-7$
GSO2	$3.21932 * 10^-15$
GSO3	$3.5297 * 10^-15$
GSO4	$3.82953 * 10^-15$
GSO5	$3.48716 * 10^-15$
GSO6	$8.40512 * 10^-9$
GSO7	$3.22206 * 10^-7$
OutCholloss1	$1.76103 * 10^-8$
OutCholloss2	$2.1098 * 10^-9$

We expected that the size of the errors in the answer might vary according to their position in the answer vector, as it seemed the errors in the decompositions grew larger and larger towards the bottom right of the matrices, almost as if the errors were multiplying, however upon further inspection most of the errors were evenly distributed amongst the values of the solution to the decomposed system x, except for the single case we observed where the error was not evenly distributed throughout the answer in the case of QRloss4. You can see below the error in the answers by term:

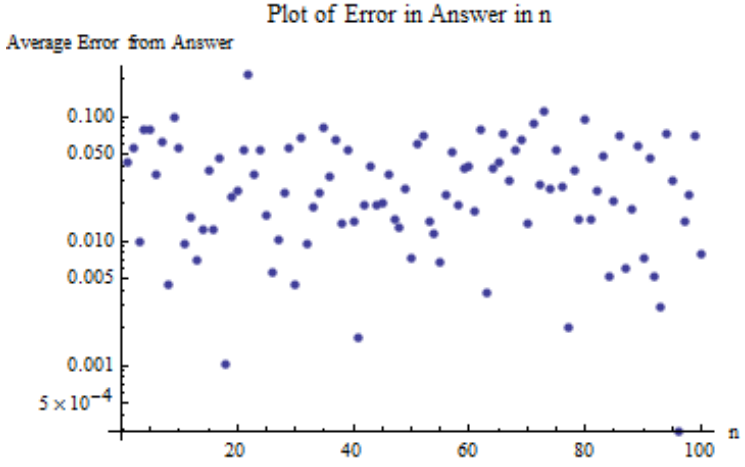
Plot error in Solution via QRloss1

Figure 1: A loss of precision at (50,50) in R before the householder is constructed from the 50th column. Notice that the Error is evenly distributed throughout the solution.



Plot error in Solution via QRloss2

Figure 2: A loss of precision at (50,60) in R before the householder is constructed from the 50th column.



Plot error in Solution via QRloss3

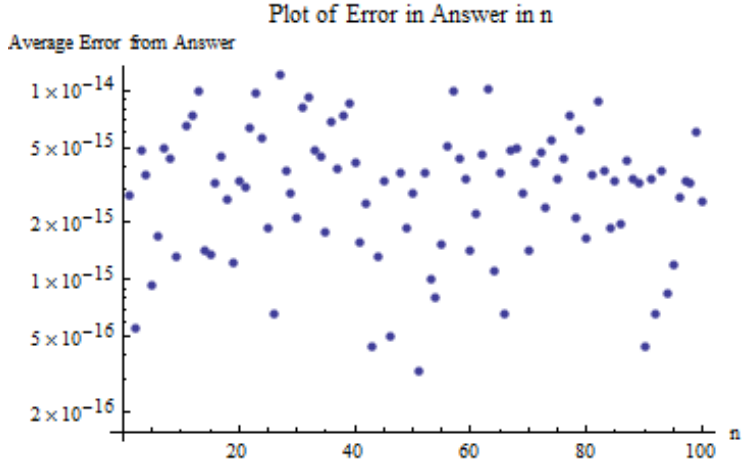
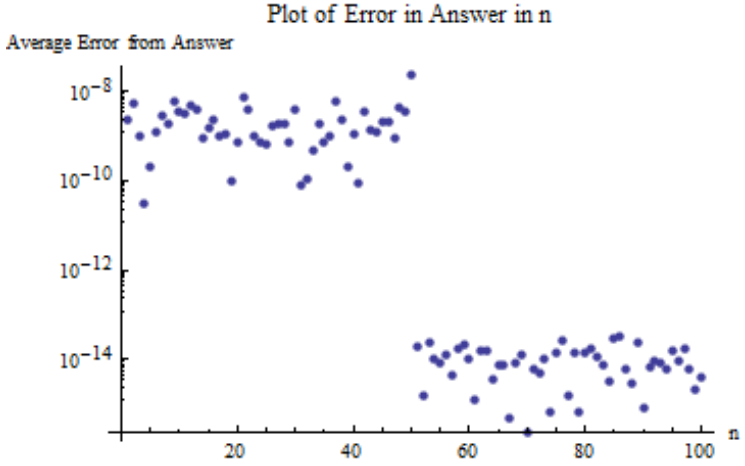


Figure 3: A loss of precision at (50,50) in the householder after it is constructed but before it's used.

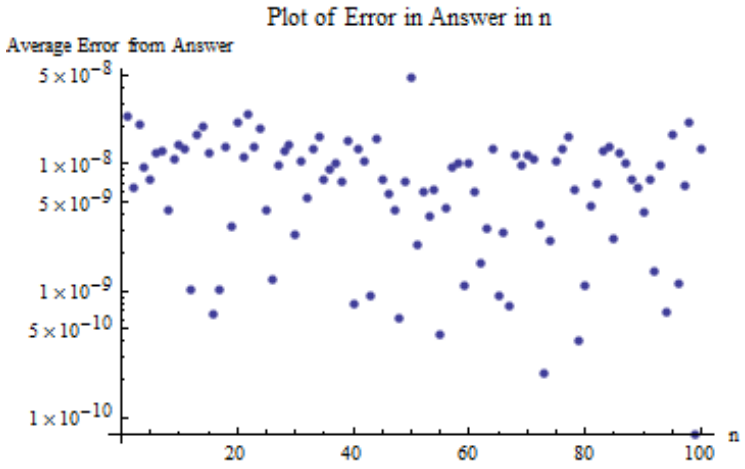
Plot error in Solution via QRloss4

Figure 4: A loss of precision at (50,50) in the householder after it is constructed and used on R, but before it is accumulated into Q. Notice that the error is much greater in the beginning of the answer, this is because that erroneousus Householder was accumulated into the other householders which acted on the first 50 columns. Then the remaining householders act to smooth out this error, hence the error has only increased slightly in compairison.



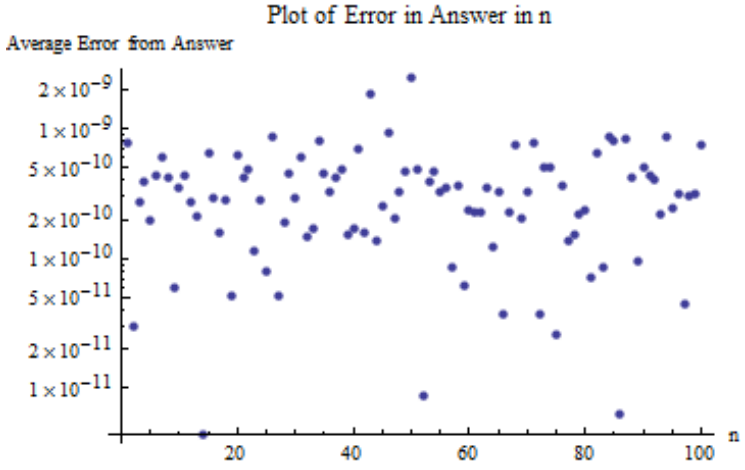
Plot error in Solution via LUloss1

Figure 5: A loss of precision at (50,50) before any operations have been done to the 50th row.



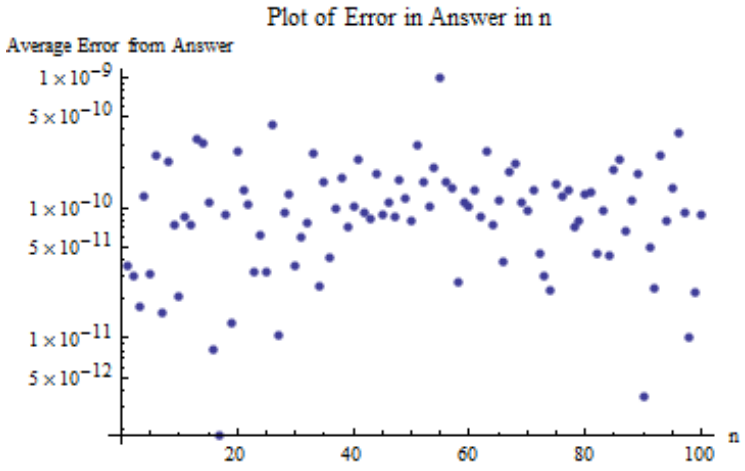
Plot error in Solution via LUloss2

Figure 6: A loss of precision at (50,60) before any operations have been done to the 50th row.



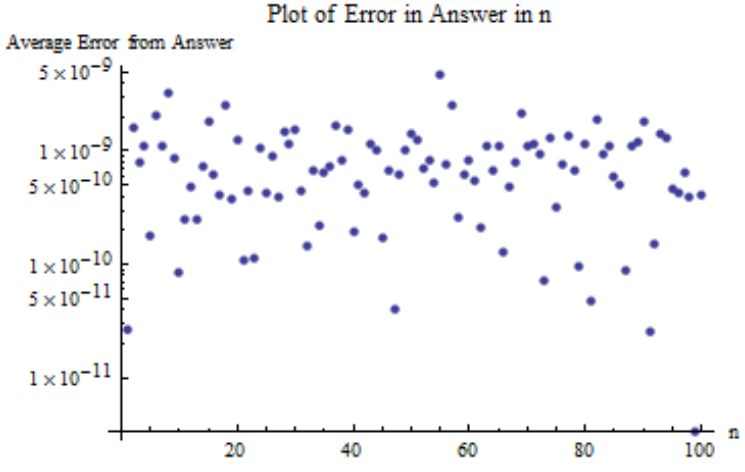
Plot error in Solution via LUloss3

Figure 7: A loss of precision at (55,50) after the constant for the row operation has been found, but not yet used.



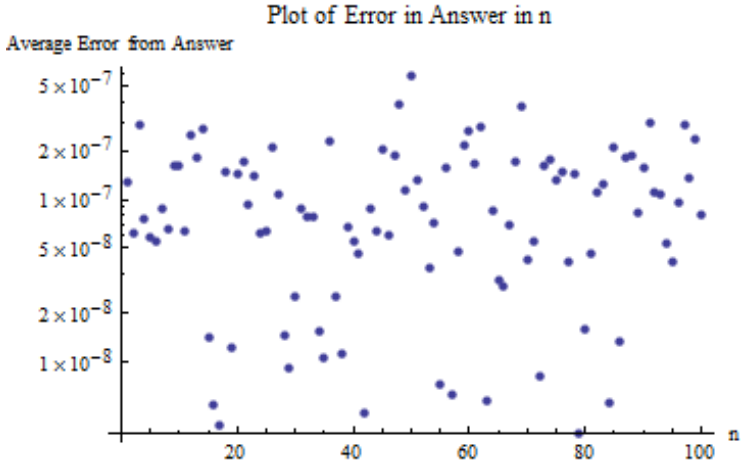
Plot error in Solution via LUloss4

Figure 8: A loss of precision at (55,50) after the constant for the row operation has been found has already been used.



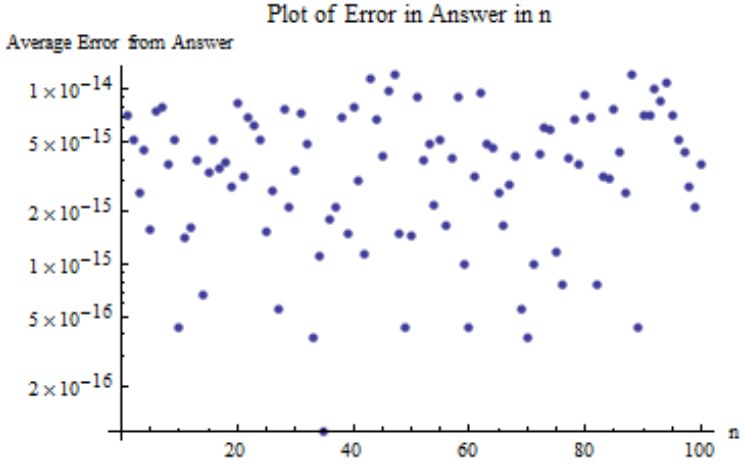
Plot error in Solution via GSO1

Figure 9: A loss of precision at (50,50) in A before the 50th column has been orthogonalized.



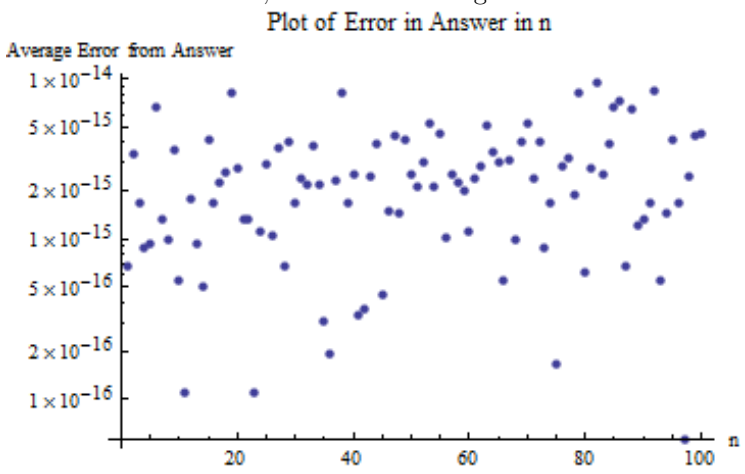
Plot error in Solution via GSO2

Figure 10: A loss of precision in the one of the coefficients for the first orthogonalization of the 50th column, before it is orthogonalized.



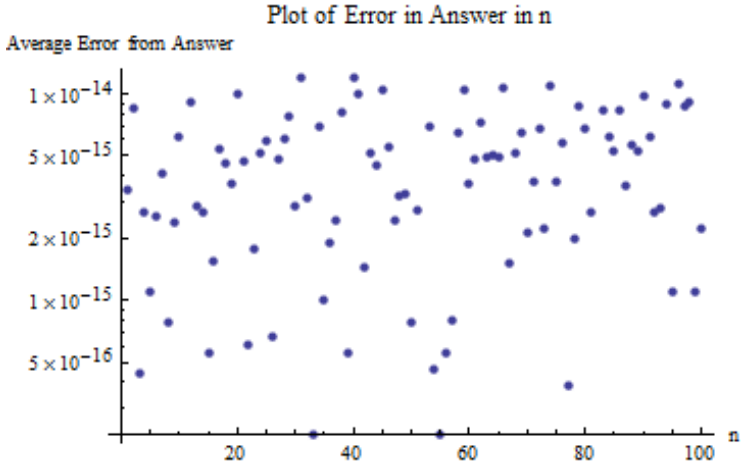
Plot error in Solution via GSO3

Figure 11: A loss of precision in the **all** of the coefficients for the first orthogonalization of the 50th column, before it is orthogonalized.



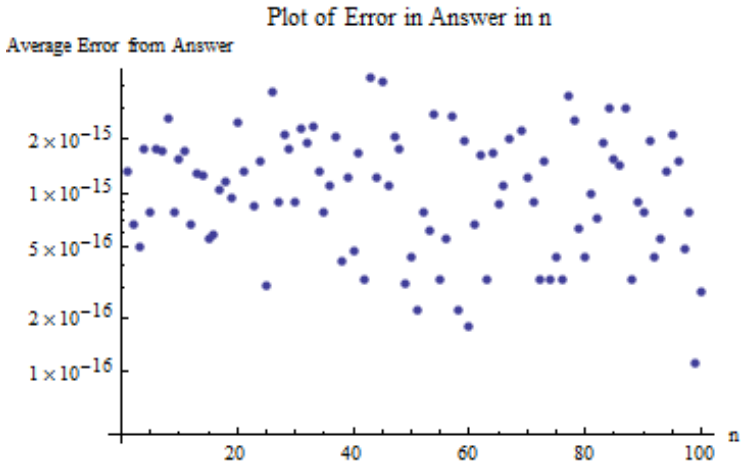
Plot error in Solution via GSO4

Figure 12: A loss of precision in the one of the coefficients for the second orthogonalization of the 50th column, before it is orthogonalized (for the second time)



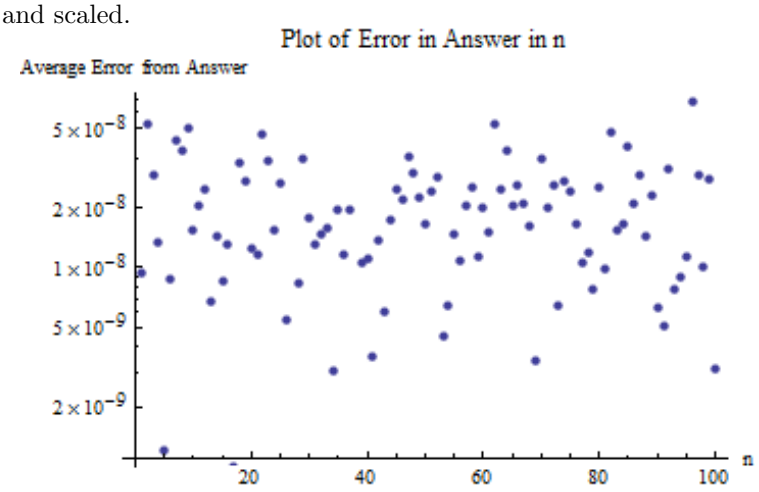
Plot error in Solution via GSO5

Figure 13: A loss of precision in the **all** of the coefficients for the second orthogonalization of the 50th column, before it is orthogonalized (for the second time)



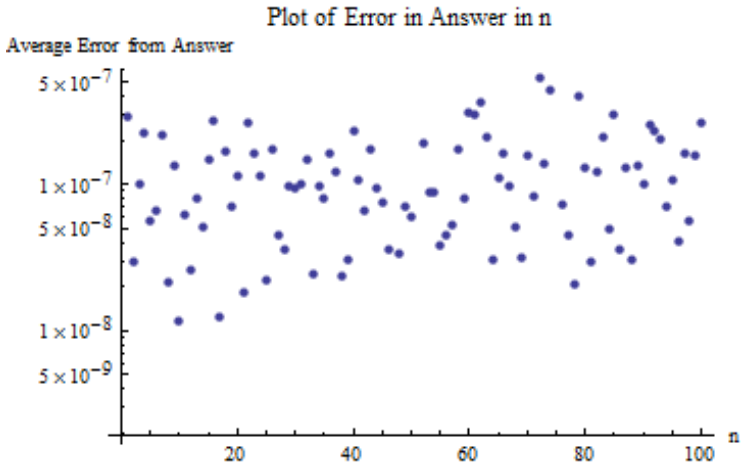
Plot error in Solution via GSO6

Figure 14: A loss of precision in (50,50) after the column has been orthogonalized twice and scaled.



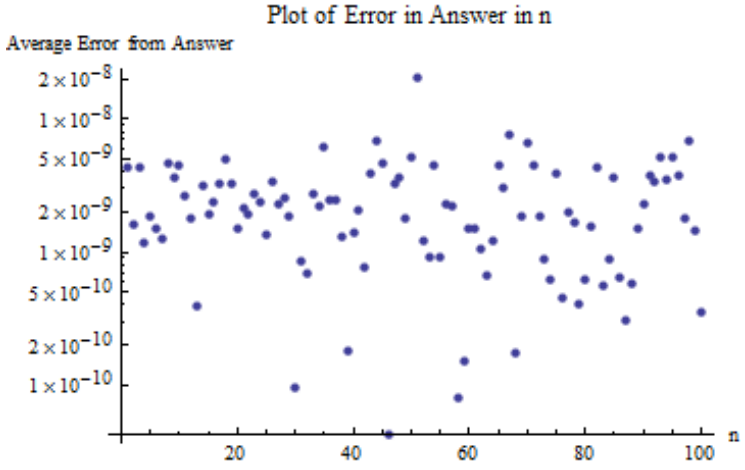
Plot error in Solution via GSO7

Figure 15: A loss of precision in (50,50) after the column has been orthogonalized twice and before it has been scaled.



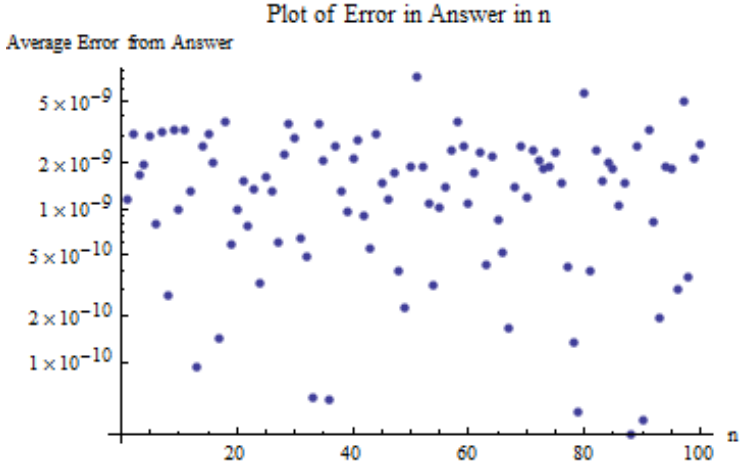
Plot error in Solution via Cholloss1

Figure 16: A loss of precision at (50,50) before the Cholesky has operated on the sub-matrix starting at the corner (50,50)



Plot error in Solution via Cholloss2

Figure 17: A loss of precision at (50,70) before the Cholesky has operated on the sub-matrix starting at the corner (50,50)



As you can see from out results, there did not appear to be any accretion of error in any of the decompositions, and the errors were, but for a single case, quite evenly distributed through the solutions. The QR via the Gram-Schmidt proceedure has in the worst case an error one order of magnitude larger than the other three decompoitions. In addition we noticed some strange behavior in the distribution of the entries of the Q in the correct QR decomposition which we will now address.

4 The Distribution of Q

It was noted that the entries of the Q in the QR decomposition appeared to look normally distributed, especially the values on the last column. These values were found to be non-normal through repeated simulation, which is expected since the numbers in each column must all be less than one and cannot be iid because their is a dependence between them due to the requirement of orthogonality.

I was then hypothized to us that they may instead follow some non-centered Student T distribution, deriving exactly which was quite beyond our scope, but the constraint of othogonality still imposed a weak cross-correlation between the entries of -0.007974. Thus even if they do individually follow the same distribution they are not independent, meaning that this process is not useful for the generation of an i.i.d. sample from that distribution.

Because the scaling done in the Gram-Schmidt process involves the sum of the column, which is not independent from the individual entries of the column, we know that the resultant number violates an assumption required for the Student T distribution, namely independence of the χ^2 and the normal random variable it divides. So we have that it cannot exactly follow a non-cental T.

5 Future Work

In the future we would like to see what differences there are between the solutions observed under the effect of a floating point error and the solutions which would be found if the entire computation was done in single floating point arithmetic.

It also interests the author to analyticly find out the exact distribution of a set of random variables after the Gram-Schmidt process is applied once. This would let us determine the exact distribution of the final column of Q, and allow us to see why this process may be making things look semi-normal.

6 References

1. IEEE Computer Society (August 29, 2008), IEEE Standard for Floating-Point Arithmetic, IEEE, doi:10.1109/IEEESTD.2008.4610935, IEEE Std 754-2008

2. Watkins, D. S. (2010). *Fundamentals of Matrix Computations.* (3 ed.). Wiley.