Daniel Perry
CS6210 / Fall 2005
Homework 2, Problem 3 Results

3)
a)Overview

In this exercise we used two different methods of decomposition to solve problems of the form Ax=b. To test for accuracy, x was chosen to be x=[1,1,....,1], and multiplied by A to obtain b. Then assuming we didn't know what x really was, we solved for x. Then we compare those results to the original vector x=[1,1,...,1].

We used the Hilbert matrix, as well as a matrix of our own. I attempted to create an more elaborate SPD matrix, by using diagonal matrices, rearranging columns, and multiplying them, but I couldn't every arrive at a matrix I felt was appropriate. So I ended up using a modified form of the example matrix given in Heath's book in the section where he discusses Cholesky decomposition. To see the specific matrices, you can look at the data section at the end of the document, where they are listed (along with the data results).

b) Data Analysis:

I have attached to the end of this document the results from my implementation. I will try to summarize what I understand the results to indicate.

Cholesky LU Decomposition:

The results from using the Cholesky LU decomposition, along with the Forward and Backward Substitution Algorithms, to solve Ax=b where A is a symmetric positive-definite matrix, initially indicate some limitations for the method, at least in my implementation. For the hilbert matrices of sizes <= 6x6, the method seemed to give very good results, meaning minimal error from in finding x to be [1,1,...,1]. However in instances where the hilbert matrix was 8x8 or 10x10, the errors seemed to "blow up" even with iterative refinement. For example, in an 8x8 hilbert matrix system, we would get answers like -15.8604 or 12.7855 instead of 1. This is a pretty bad error. And in fact, with such a bad original result, iterative refinement made the result worse, resulting in numbers like 54.8025 or 42.8685, obviously very far from the correct result of 1. Because these errors are occurring in elements towards the bottom of the array, it leads one to believe the errors are probably "creeping" in as we rescale the remaining rows of the matrix. Because elements in a Hilbert matrix are also much smaller towards the end of the matrix (especially as the matrix gets large or >= 8x8), these smaller numbers probably contribute to the errors as well. Because the last rows are last, their results rely on the most floating point arithmetic which can introduce more error. The outrageous errors after 3 iterations of refinement are results of those initial errors, which seem to characterize newton approximation-like refinement (if the original "guess" is off by a lot, the refinements don't really help).

Using Cholesky LU decomposition, along with FSA and BSA gave decent results on my matrix as well. Because the matrix I chose was sufficiently simple, the errors aren't quite as noticeable as in those by Hilbert cases. The iterative refinement seems to recover from the errors sufficiently, even in the 10x10 case. In other words, probably more because of the selection of the matrix rather than the algorithm, I could get the expected result of [1,1,....,1] or pretty close to it.

Iterative Refinement:

The iterative refinement worked as expected. It would get the answers closer to the correct answer with each refinement, except in the cases, as noted above, where the original answers were so far off, that the refining didn't help. Normally the refinement would not get exactly 1 (the correct result), but get really close. Probably because of precision error. But they would get closer.

Householder QR Decomposition:

Surprisingly to me, the Householder QR decomposition had similar behavior for the Hilbert matrix. For hilbert matrices <= 6x6 in size seemed to find the correct answer, or at least came extremely close. However for matrices >= 8x8 in size, we got extremely large errors. The refinement again seemed to make the result worse, than the original result. Again, very similar to Cholesky, the outrageous errors seem to occur towards the end of the vector, giving the indication, that the errors are probably due to accumulation of errors, **as well as** the fact that a Hilbert matrix's elements in the ending columns are much smaller than the first column, contributes to these errors as well. This makes sense, as the errors are larger in the larger Hilbert matrix, and the larger the Hilbert matrix, the smaller the small elements will be.

The Householder did seem to do better than Cholesky, because, while the Cholesky method seemed to give REALLY bad results for a 10x10 hilbert matrix (resulting in nan's), the Householder method could actually obtain a result. This results was pretty good for the first elements, but very bad for the ending elements. This indicates that the Householder has better stability than Cholesky (performs better with bad data).

Householder also had similar results for the simple matrix. It would arrive at the correct result with little error, and the iterative refinement would only reduce that error.

Out of curiosity I tried it with double precision throughout the program, instead of single (float) precision, and it would achieve the desired on the first attempt (no iteration required).

Iterative Refinement:

Very similar to the Cholesky iterative refinement, the iterative refinement in Householder method would improve results when the initial result was close to 1, but would only make it worse if it was very erroneous.

c) Method Comments:

I implemented the methods in C++ using the g++ compiler. I used float precision, except for calculations for the residual vector in iterative refinement, where double precision was used for the arithmetic with float size storage. I am putting my code on my cs web space as indicated below.

Cholesky LU Decomposition:

This decomposition was much easier to implement than the householder decomposition. This was due to the excellent algorithm outline by Heath in his book. The followed his outline mostly.

I had some major delays in my implementation for this algorithm because of a small error in my Backward Substitution Algorithm. I had an "off-by-one" error in that code, and it had me puzzled for several days. I supposed a result from translating from 1-based vectors in the book to 0-based vectors in C++. Thanks to some direction by Mike, I was able to find it, and once that was fixed it worked.

Householder QR Decomposition:

This was bit more tricky to implement. I think mainly because I didn't have a very good grasp of the algorithm while trying to implement it. Again, Mike helped me out and I was able to put it all together finally.

One important aspect of this implementation, is that you perform the householder transformation on iteratively smaller matrices inside the original A matrix. By writing code that only performs arithmetic on these “inner” matrices, instead of the entire matrix A, you save a lot of computation time. Luckily the general algorithm outlined in Heath's book, does this. The inner loop of calculation only performs it on columns and rows greater than the current column (which is the next inner matrix).

Another special implementation note is on the solution to Qy=b (which then is used to solve Rx=y). Naturally one would want to multiply out all the H-matrices and use FSA to solve that matrix. However, we are storing Q as independent vectors v, which are used to construct each H, so a better method to storing H, is to recalculate each H and apply them in reverse order to b (essentially solving Qy=b, by doing y=Q^(-1)b). It is important to note, that because the special properties of the H matrix, you don't have to apply that transformation to every element of b, but to the “inner-vectors” of b (very similar to the idea of inner matrices of A). This saves several arithmetic operations each iteration, until the last H is only affecting the last element of b. This is not specifically pointed out anywhere, but is rather implied with the definition of this property in the formation of R by R=(H...H)*A.

Iterative Refinement:

This was implemented mainly following the notes in class, just re-using the decomposed matrices and adding the difference to the original x. It's the same idea using in newton-approximation.

Resulting Data:
(listing is titled by decomposition method and matrix used. Each line under each matrix size are the results from an iteration refinement – each has 3 iterative refinements).
(note: I will post this document, as well as my code at www.cs.utah.edu/~dperry/classes/cs6210/)

Cholesky Decomposition – Hilbert Matrix:

hilbert 4x4
```
x = 1.00001, 0.999925, 1.00019, 0.999874,
error_x = 6.4373e-06, 7.53403e-05, 0.000188351, 0.000125587,
x = 0.999992, 1.00009, 0.999802, 1.00012,
error_x = 7.86781e-06, 8.55923e-05, 0.000197887, 0.000124693,
x = 1.00002, 0.999764, 1.00057, 0.999624,
error_x = 2.07424e-05, 0.000236273, 0.000574589, 0.000375986,
x = 1.00002, 0.999764, 1.00057, 0.999624,
error_x = 2.07424e-05, 0.000236273, 0.000574589, 0.000375986,
```
hilbert 6x6
```
x = 1.00042, 0.988545, 1.07478, 0.81042, 1.20524, 0.920347,
error_x = 0.000423908, 0.0114546, 0.0747792, 0.18958, 0.205239, 0.0796529,
x = 1.00073, 0.980222, 1.12959, 0.669786, 1.35932, 0.859895,
error_x = 0.000734329, 0.0197781, 0.129587, 0.330214, 0.359319, 0.140105,
x = 1.00094, 0.974408, 1.16786, 0.57287, 1.46374, 0.819614,
error_x = 0.000941157, 0.0255917, 0.167861, 0.42713, 0.463741, 0.180386,
x = 1.00058, 0.984332, 1.10222, 0.741134, 1.27997, 0.891434,
error_x = 0.000578046, 0.0156677, 0.102216, 0.258866, 0.279973, 0.108566,
```
hilbert 8x8
```
x = 1.00135, 0.930109, 1.88254, -3.63658, 13.1719, -15.8604, 12.7855, -2.27506,
error_x = 0.00134683, 0.069891, 0.882544, 4.63658, 12.1719, 16.8604, 11.7855, 3.27506,
x = 1.00135, 0.911925, 2.26299, -6.20669, 21.0596, -28.0312, 21.9959, -4.99552,
error_x = 0.0013479, 0.0880749, 1.26299, 7.20669, 20.0596, 29.0312, 20.9959, 5.99552,
x = 1.00311, 0.910543, 1.48579, 1.07144, -4.78424, 15.4889, -12.9811, 5.80918,
error_x = 0.00311422, 0.0894566, 0.485787, 0.0714417, 5.78424, 14.4889, 13.9811, 4.80918,
x = 0.9991, 0.955274, 2.3646, -9.37429, 34.9828, -53.8025, 43.8685, -11.9993,
error_x = 0.000899553, 0.0447257, 1.3646, 10.3743, 33.9828, 54.8025, 42.8685, 12.9993,
```
hilbert 10x10
```
x = nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
error_x = nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
x = nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
error_x = nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
x = nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
error_x = nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
x = nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
error_x = nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
```

Daniel Perry
CS6210 / Fall 2005
Homework 2, Problem 3 Results

Cholesky Decomposition -Simple SPD Matrix:
simple matrix 4x4:
```
3.03, -1, -1, -1,
-1, 6.03, -1, -1,
-1, -1, 9.03, -1,
-1, -1, -1, 12.03,
x = 1, 1, 1, 1,
error_x = 5.96046e-08, 1.19209e-07, 0, 0,
x = 1, 1, 1, 1,
error_x = 1.19209e-07, 0, 0, 0,
x = 1, 1, 1, 1,
error_x = 5.96046e-08, 5.96046e-08, 0, 0,
x = 1, 1, 1, 1,
error_x = 1.19209e-07, 0, 0, 0,
```
simple matrix 6x6:
```
5.03, -1, -1, -1, -1, -1,
-1, 8.03, -1, -1, -1, -1,
-1, -1, 11.03, -1, -1, -1,
-1, -1, -1, 14.03, -1, -1,
-1, -1, -1, -1, 17.03, -1,
-1, -1, -1, -1, -1, 20.03,
x = 1, 1, 1, 1, 1, 1,
error_x = 0, 5.96046e-08, 5.96046e-08, 5.96046e-08, 0, 1.19209e-07,
x = 1, 1, 1, 1, 1, 1,
error_x = 0, 0, 0, 0, 0, 0,
x = 1, 1, 1, 1, 1, 1,
error_x = 0, 0, 0, 0, 0, 0,
x = 1, 1, 1, 1, 1, 1,
error_x = 0, 0, 0, 0, 0, 0,
```
simple matrix 8x8:
```
7.03, -1, -1, -1, -1, -1, -1, -1,
-1, 10.03, -1, -1, -1, -1, -1, -1,
-1, -1, 13.03, -1, -1, -1, -1, -1,
-1, -1, -1, 16.03, -1, -1, -1, -1,
-1, -1, -1, -1, 19.03, -1, -1, -1,
-1, -1, -1, -1, -1, 22.03, -1, -1,
-1, -1, -1, -1, -1, -1, 25.03, -1,
-1, -1, -1, -1, -1, -1, -1, 28.03,
x = 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 1.78814e-07, 1.78814e-07, 0, 1.19209e-07, 1.19209e-07, 0, 1.19209e-07, 1.19209e-07,
x = 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 1.19209e-07, 1.19209e-07, 1.19209e-07, 1.19209e-07, 0, 1.19209e-07, 0, 1.19209e-07,
x = 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 1.19209e-07, 5.96046e-08, 5.96046e-08, 5.96046e-08, 5.96046e-08, 0, 5.96046e-08, 5.96046e-08,
x = 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 1.19209e-07, 1.19209e-07, 1.19209e-07, 0, 0, 0, 0, 0,
```

Daniel Perry
CS6210 / Fall 2005
Homework 2, Problem 3 Results

simple matrix 10x10:

```
9.03, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, 12.03, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, 15.03, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, 18.03, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, 21.03, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, 24.03, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, 27.03, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, 30.03, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, 33.03, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, 36.03,
x = 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 1.19209e-07, 0, 5.96046e-08, 1.19209e-07, 0, 0, 0, 1.19209e-07, 1.19209e-07, 0,
x = 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 0, 0, 0, 5.96046e-08, 0, 0, 0, 0, 0, 0,
x = 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
x = 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Daniel Perry
CS6210 / Fall 2005
Homework 2, Problem 3 Results

Householder Decomposition – Hilbert Matrix:
hilbert 4x4
```
x = 1, 0.999993, 1.00003, 0.999974,
error_x = 0, 7.33137e-06, 3.18289e-05, 2.64049e-05,
x = 0.99999, 1.0001, 0.999774, 1.00014,
error_x = 9.77516e-06, 9.98974e-05, 0.000225663, 0.000140429,
x = 1, 0.999939, 1.00016, 0.99989,
error_x = 4.52995e-06, 6.10352e-05, 0.000160575, 0.000109911,
x = 1, 0.999939, 1.00016, 0.99989,
error_x = 4.52995e-06, 6.10352e-05, 0.000160575, 0.000109911,
```
hilbert 6x6
```
x = 1.00054, 0.98526, 1.09735, 0.750963, 1.27154, 0.894007,
error_x = 0.000537753, 0.0147402, 0.0973479, 0.249037, 0.271541, 0.105993,
x = 1.00085, 0.976426, 1.15706, 0.5957, 1.44295, 0.826443,
error_x = 0.000847459, 0.0235741, 0.157056, 0.4043, 0.442945, 0.173557,
x = 1.00112, 0.969104, 1.20418, 0.4779, 1.56884, 0.77815,
error_x = 0.00112104, 0.0308964, 0.204181, 0.5221, 0.568836, 0.22185,
x = 0.999377, 1.01736, 0.884188, 1.29845, 0.672755, 1.1283,
error_x = 0.000623465, 0.0173638, 0.115812, 0.298446, 0.327245, 0.128301,
```
hilbert 8x8
```
x = 1.00275, 0.871656, 2.4774, -6.13986, 18.3505, -21.3575, 15.5981, -2.80302,
error_x = 0.0027529, 0.128344, 1.4774, 7.13986, 17.3505, 22.3575, 14.5981, 3.80302,
x = 1.00361, 0.828659, 2.96292, -8.33909, 23.2032, -26.8725, 18.6724, -3.4585,
error_x = 0.00361085, 0.171341, 1.96292, 9.33909, 22.2032, 27.8725, 17.6724, 4.4585,
x = 1.00365, 0.798128, 3.60654, -12.7866, 37.1701, -48.8807, 35.6418, -8.55438,
error_x = 0.00365365, 0.201872, 2.60654, 13.7866, 36.1701, 49.8807, 34.6418, 9.55438,
x = 1.00627, 0.669563, 5.14628, -20.4847, 56.4627, -74.4675, 52.8117, -13.1461,
error_x = 0.00627267, 0.330437, 4.14628, 21.4847, 55.4627, 75.4675, 51.8117, 14.1461,
```
hilbert 10x10
```
x = 0.980285, 1.89056, -8.05856, 32.7468, -24.1469, -78.6503, 168.385, -67.5268, -55.2754, 39.6815,
error_x = 0.019715, 0.890561, 9.05856, 31.7468, 25.1469, 79.6503, 167.385, 68.5268, 56.2754, 38.6815,
x = 1.9322, -41.5287, 443.036, -1634.7, 1769.33, 2546.1, -6729.72, 3015.73, 2258.89, -1620.24,
error_x = 0.9322, 42.5287, 442.036, 1635.7, 1768.33, 2545.1, 6730.72, 3014.73, 2257.89, 1621.24,
x = -37.1552, 1739.7, -18047, 66624.5, -71263.9, -106140, 277234, -123916, -92630.1, 66492.6,
error_x = 38.1552, 1738.7, 18048, 66623.5, 71264.9, 106141, 277234, 123917, 92631.1, 66491.6,
x = 313.426, -14433.3, 153139, -592306, 769183, 500211, -1.92714e+06, 948829, 652441, -490598,
error_x = 312.426, 14434.3, 153138, 592308, 769182, 500210, 1.92714e+06, 948828, 652440, 490599,
```

Daniel Perry
CS6210 / Fall 2005
Homework 2, Problem 3 Results

Householder Decomposition – Simple SPD Matrix:
simple 4x4:

```
3.03, -1, -1, -1,
-1, 6.03, -1, -1,
-1, -1, 9.03, -1,
-1, -1, -1, 12.03,
x = 1, 1, 1, 1,
error_x = 3.57628e-07, 0, 1.19209e-07, 0,
x = 1, 1, 1, 1,
error_x = 0, 0, 0, 5.96046e-08,
x = 1, 1, 1, 1,
error_x = 0, 0, 0, 0,
x = 1, 1, 1, 1,
error_x = 0, 0, 0, 0,
```

simple 6x6:

```
5.03, -1, -1, -1, -1, -1,
-1, 8.03, -1, -1, -1, -1,
-1, -1, 11.03, -1, -1, -1,
-1, -1, -1, 14.03, -1, -1,
-1, -1, -1, -1, 17.03, -1,
-1, -1, -1, -1, -1, 20.03,
x = 0.999999, 1, 1, 1, 1, 1,
error_x = 5.36442e-07, 1.19209e-07, 3.57628e-07, 5.96046e-08, 2.38419e-07, 1.19209e-07,
x = 1, 1, 1, 1, 1, 1,
error_x = 1.19209e-07, 1.19209e-07, 0, 0, 0, 0,
x = 1, 1, 1, 1, 1, 1,
error_x = 0, 5.96046e-08, 5.96046e-08, 0, 0, 0,
x = 1, 1, 1, 1, 1, 1,
error_x = 0, 0, 0, 0, 0, 0,
```

simple 8x8:

```
7.03, -1, -1, -1, -1, -1, -1, -1,
-1, 10.03, -1, -1, -1, -1, -1, -1,
-1, -1, 13.03, -1, -1, -1, -1, -1,
-1, -1, -1, 16.03, -1, -1, -1, -1,
-1, -1, -1, -1, 19.03, -1, -1, -1,
-1, -1, -1, -1, -1, 22.03, -1, -1,
-1, -1, -1, -1, -1, -1, 25.03, -1,
-1, -1, -1, -1, -1, -1, -1, 28.03,
x = 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 2.38419e-07, 4.76837e-07, 2.38419e-07, 1.19209e-07, 1.19209e-07, 1.19209e-07, 1.19209e-07, 0,
x = 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 5.96046e-08, 5.96046e-08, 5.96046e-08, 5.96046e-08, 0, 0, 0, 5.96046e-08,
x = 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 1.19209e-07, 1.19209e-07, 0, 0, 0, 0, 0, 0,
x = 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 5.96046e-08, 0, 5.96046e-08, 0, 0, 0, 0, 0,
```

Daniel Perry
CS6210 / Fall 2005
Homework 2, Problem 3 Results

simple 10x10:

```
9.03, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, 12.03, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, 15.03, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, 18.03, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, 21.03, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, 24.03, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, 27.03, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, 30.03, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, 33.03, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, 36.03,
x = 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 8.34465e-07, 1.07288e-06, 2.38419e-07, 0, 5.96046e-07, 5.96046e-07, 2.38419e-07, 2.38419e-07, 2.38419e-07,
4.76837e-07,
x = 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 1.78814e-07, 1.19209e-07, 5.96046e-08, 1.19209e-07, 1.78814e-07, 5.96046e-08, 1.19209e-07, 5.96046e-08,
1.19209e-07, 5.96046e-08,
x = 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 2.38419e-07, 1.19209e-07, 1.19209e-07, 1.19209e-07, 1.19209e-07, 1.19209e-07, 0, 1.19209e-07, 0, 0,
x = 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
error_x = 1.19209e-07, 5.96046e-08, 0, 5.96046e-08, 5.96046e-08, 0, 5.96046e-08, 5.96046e-08, 5.96046e-08, 5.96046e-
08,
```