Palmer Edholm
Dr. Joe Koebbe
MATH 4610
November 29, 2021

Tasksheet 8

1. See my software manual entry to see my method on generating a random, diagonally dominant matrix. See my other software manual entry to see my method for solving square linear systems of equations with an example on a diagonally dominant coefficient matrix.

2. See my software manual entry to see my method for LU factorization. See my software manual entry to see my method for forward substitution.

3. See my software manual entry to see my method for generating Hilbert matrices. The following code generates an array of Hilbert matrices of size $n = 4, 5 \ldots, 10$, multiplies them into a vector of ones, uses LU factorization to reduce them to lower triangular, and uses those vectors to solve the system via forward substitution and see how closely it approximates the original vector of ones.

```
1  import numpy as np
2  from frwrdsub import frwrdsub
3  from more_matgen import hilbert
4
5
6  H = []
7  b = []
8  forw = []
9  for i in range(4, 11):
10     b1 = np.ones(i, np.float)
11     H.append(np.asarray(hilbert(i), np.float))
12     b.append(np.dot(H[i-4], b1))
13     L, U = lu(H[i-4])
14     forw.append(frwrdsub(L, b[i-4]))
15 for sol in forw:
16     print(sol)
```

As a note, I'm using the lower triangular matrices and forward substitution because those results contained less garbage. I get the following result from the above code.

```
[2.08333333e+00 2.41666667e-01 1.38888889e-02 3.57142857e-04]
[2.28333333e+00 3.08333333e-01 2.34126984e-02 1.07142857e-03
 2.26757370e-05]
[2.45000000e+00 3.67857143e-01 3.33333333e-02 2.06349206e-03
 7.93650794e-05 1.43154905e-06]
[2.59285714e+00 4.21428571e-01 4.32539683e-02 3.25396825e-03
 1.72129458e-04 5.72619620e-06 9.00974922e-08]
[2.71785714e+00 4.70039683e-01 5.29761905e-02 4.57972583e-03
 2.98392084e-04 1.38199543e-05 4.05438716e-07 5.65997032e-09]
```

```
[2.82896825e+00 5.14484127e-01 6.24037999e-02 5.99386724e-03
 4.53792240e-04 2.61533000e-05 1.07816666e-06 2.82998527e-08
 3.55136719e-10]
[2.92896825e+00 5.55393218e-01 7.14947090e-02 7.46239871e-03
 6.33612419e-04 4.28033166e-05 2.21339507e-06 8.22360425e-08
 1.95325000e-09 2.22674809e-11]
```

As we can see, Hilbert matrices are ill-conditioned.

4. See my software manual entry to see my method that implements scaled partial pivoting to perform LU factorization.

5. The following code is essentially the same as that from question 3 except using the scaled partial pivoting method.

```
1  from lu import hilbert
2  from frwrdsub import frwrdsub
3
4
5  H = []
6  b = []
7  forw = []
8  for i in range(4, 11):
9      b1 = np.ones(i, np.float)
10     H.append(np.asarray(hilbert(i), np.float))
11     b.append(np.dot(H[i-4], b1))
12     L, U = sc_part_pivot(H[i-4])
13     forw.append(frwrdsub(L, b[i-4]))
14 for sol in forw:
15     print(sol)
```

We get the following results.

```
[ 2.08333333  0.7625     -0.93888889 -0.4065873 ]
[ 2.28333333  0.99333333 -1.29047619 -0.43       -0.4081746 ]
[ 2.45        1.18452381 -1.66547619 -0.60972222  0.08126323 -0.55783367]
[ 2.59285714  1.34744898 -2.06349206 -0.69615079  0.01879563 -0.73934589
  0.03618392]
[ 2.71785714  1.48923611 -2.02996032  0.31957973 -0.77878015 -0.60323956
 -0.94577425 -0.11186604]
[ 2.82896825  1.61463845 -2.45055916  0.40167824 -0.88768774 -0.14338744
 -0.82146058 -0.58472028 -0.21474248]
[ 2.92896825  1.72698052 -2.89103535  0.49444352 -0.9480699  -0.54897389
 -1.27151033 -0.31164935 -0.40719707 -0.67725278]
```

Even though we're still not recovering our original vectors, these vectors are closer than those in question 3.

6. In [1], the author starts off by talking about some of the properties of Hilbert matrices: they are symmetric positive definite matrices. In addition, they also fit the definition of Hankel matrices (constant along the antidiagonals). One main reason why Hilbert matrices are so ill-conditioned is because they're obtained when least squares polynomial approximation is done using the monomial basis (which is an ill-conditioned polynomial basis). A Hilbert matrix is not a good test matrix not only because it is ill-conditioned, but also because it's too special. In addition to its definiteness, it's a special case of a Cauchy matrix, all of which mean that certain elements can be calculated more accurately than for a general positive definite matrix.

References

[1] https://nhigham.com/2020/06/30/what-is-the-hilbert-matrix/