Zhang_Jeffrey_code2

February 22, 2022

1 AMSC 460 HW 2 Part 2

1.1 Jeffrey Zhang

The Hilbert matrix $H_n = (h_{ij}), 1 \leq i, j \leq n$, is defined by

$$h_{ij} = \frac{1}{i+j-1}.$$

This matrix is nonsingular and has an explicit inverse. Let $\mathbf{x}_n = (1, 1, \cdots, 1)$ and $\mathbf{b}_n = H_n\mathbf{x}_n$. This problem examines the quality of the computed solution $\mathbf{x}_n^*$.

(a) For $n = 5, 10$: set $\mathbf{x}_n = (1, 1, \cdots, 1)$, multiply $H_n\mathbf{x}_n$ to obtain $\mathbf{b}_n$, and then solve $H_n\mathbf{x}_n^* = \mathbf{b}_n$ for $\mathbf{x}_n^*$ by any direct method, e.g. by Gaussian elimination.

[91]:
```
import numpy as np
import sys
import math
from scipy.linalg import hilbert
```

I imported the built in hilbert function for convenience. I will make a Hilbert Matrix function on my own and test its validity with the results shown.

[131]:
```
def HilbertMatrix(n):
    Hn = np.zeros(shape=(n,n))
    for i in range(n):
        for j in range(n):
            Hn[i][j] = 1/(i + j + 1)
    return Hn
```

[132]:
```
H5 = hilbert(5)
H5
```

[132]:
```
array([[1.        , 0.5       , 0.33333333, 0.25      , 0.2       ],
       [0.5       , 0.33333333, 0.25      , 0.2       , 0.16666667],
       [0.33333333, 0.25      , 0.2       , 0.16666667, 0.14285714],
       [0.25      , 0.2       , 0.16666667, 0.14285714, 0.125     ],
       [0.2       , 0.16666667, 0.14285714, 0.125     , 0.11111111]])
```

[133]:
```
HilbertMatrix(5)
```

[133]:
```
array([[1.        , 0.5       , 0.33333333, 0.25      , 0.2       ],
       [0.5       , 0.33333333, 0.25      , 0.2       , 0.16666667],
       [0.33333333, 0.25      , 0.2       , 0.16666667, 0.14285714],
       [0.25      , 0.2       , 0.16666667, 0.14285714, 0.125     ],
       [0.2       , 0.16666667, 0.14285714, 0.125     , 0.11111111]])
```

[93]:
```
H10 = hilbert(10)
H10
```

[93]:
```
array([[1.        , 0.5       , 0.33333333, 0.25      , 0.2       ,
        0.16666667, 0.14285714, 0.125     , 0.11111111, 0.1       ],
       [0.5       , 0.33333333, 0.25      , 0.2       , 0.16666667,
        0.14285714, 0.125     , 0.11111111, 0.1       , 0.09090909],
       [0.33333333, 0.25      , 0.2       , 0.16666667, 0.14285714,
        0.125     , 0.11111111, 0.1       , 0.09090909, 0.08333333],
       [0.25      , 0.2       , 0.16666667, 0.14285714, 0.125     ,
        0.11111111, 0.1       , 0.09090909, 0.08333333, 0.07692308],
       [0.2       , 0.16666667, 0.14285714, 0.125     , 0.11111111,
        0.1       , 0.09090909, 0.08333333, 0.07692308, 0.07142857],
       [0.16666667, 0.14285714, 0.125     , 0.11111111, 0.1       ,
        0.09090909, 0.08333333, 0.07692308, 0.07142857, 0.06666667],
       [0.14285714, 0.125     , 0.11111111, 0.1       , 0.09090909,
        0.08333333, 0.07692308, 0.07142857, 0.06666667, 0.0625    ],
       [0.125     , 0.11111111, 0.1       , 0.09090909, 0.08333333,
        0.07692308, 0.07142857, 0.06666667, 0.0625    , 0.05882353],
       [0.11111111, 0.1       , 0.09090909, 0.08333333, 0.07692308,
        0.07142857, 0.06666667, 0.0625    , 0.05882353, 0.05555556],
       [0.1       , 0.09090909, 0.08333333, 0.07692308, 0.07142857,
        0.06666667, 0.0625    , 0.05882353, 0.05555556, 0.05263158]])
```

[134]:
```
HilbertMatrix(10)
```

[134]:
```
array([[1.        , 0.5       , 0.33333333, 0.25      , 0.2       ,
        0.16666667, 0.14285714, 0.125     , 0.11111111, 0.1       ],
       [0.5       , 0.33333333, 0.25      , 0.2       , 0.16666667,
        0.14285714, 0.125     , 0.11111111, 0.1       , 0.09090909],
       [0.33333333, 0.25      , 0.2       , 0.16666667, 0.14285714,
        0.125     , 0.11111111, 0.1       , 0.09090909, 0.08333333],
       [0.25      , 0.2       , 0.16666667, 0.14285714, 0.125     ,
        0.11111111, 0.1       , 0.09090909, 0.08333333, 0.07692308],
       [0.2       , 0.16666667, 0.14285714, 0.125     , 0.11111111,
        0.1       , 0.09090909, 0.08333333, 0.07692308, 0.07142857],
       [0.16666667, 0.14285714, 0.125     , 0.11111111, 0.1       ,
        0.09090909, 0.08333333, 0.07692308, 0.07142857, 0.06666667],
       [0.14285714, 0.125     , 0.11111111, 0.1       , 0.09090909,
        0.08333333, 0.07692308, 0.07142857, 0.06666667, 0.0625    ],
       [0.125     , 0.11111111, 0.1       , 0.09090909, 0.08333333,
```

```
 0.07692308, 0.07142857, 0.06666667, 0.0625    , 0.05882353],
[0.11111111, 0.1       , 0.09090909, 0.08333333, 0.07692308,
 0.07142857, 0.06666667, 0.0625    , 0.05882353, 0.05555556],
[0.1       , 0.09090909, 0.08333333, 0.07692308, 0.07142857,
 0.06666667, 0.0625    , 0.05882353, 0.05555556, 0.05263158]])
```

Multiplying $H_n\mathbf{x}_n$ to obtain $\mathbf{b}_n$

[94]:
```python
x5 = np.ones(5)
b5 = np.matmul(H5, x5)
b5
```

[94]:
```
array([2.28333333, 1.45      , 1.09285714, 0.88452381, 0.74563492])
```

[95]:
```python
x10 = np.ones(10)
b10 = np.matmul(H10, x10)
b10
```

[95]:
```
array([2.92896825, 2.01987734, 1.60321068, 1.34680042, 1.16822899,
       1.03489566, 0.93072899, 0.84669538, 0.77725094, 0.7187714 ])
```

Solving $H_n\mathbf{x}_n^* = \mathbf{b}_n$ for $\mathbf{x}_n^*$ by any direct method, e.g. by Gaussian elimination:

[96]:
```python
def GaussianElimination(Hn, bn, n):
    xstar = np.zeros(n)
    # Applying Gauss Elimination
    for i in range(n):
        if Hn[i][i] == 0.0:
            sys.exit('Divide by zero detected!')

        for j in range(i+1, n):
            ratio = Hn[j][i]/Hn[i][i]

            for k in range(n):
                Hn[j][k] = Hn[j][k] - ratio * Hn[i][k]
    # Back Substitution
    xstar[n-1] = Hn[n-2][n-1]/Hn[n-2][n-2]

    for i in range(n-2,-1,-1):
        xstar[i] = Hn[i][n-1]

        for j in range(i+1,n):
            xstar[i] = xstar[i] - Hn[i][j]*xstar[j]

        xstar[i] = xstar[i]/Hn[i][i]

    return xstar
```

[97]:
```
x5_star = GaussianElimination(H5, b5, 5)
x5_star
```

[97]:
```
array([ 0.01428571, -0.28571429,  1.28571429, -2.        ,  2.        ])
```

[98]:
```
x10_star = GaussianElimination(H10, b10, 10)
x10_star
```

[98]:
```
array([-7.19847978e-05,  6.47867058e-03, -1.42531411e-01,  1.33029806e+00,
       -6.48522249e+00,  1.81586682e+01, -3.02645105e+01,  2.96469207e+01,
       -1.57499511e+01,  4.49999389e+00])
```

(b) For $n = 5, 10$: compute the error $\mathbf{e}_n = \mathbf{x}_n - \mathbf{x}_n^*$, the residual $\mathbf{r}_n = \mathbf{b}_n - H_n\mathbf{x}_n^*$, and their norms $\|\cdot\|_1, \|\cdot\|_2, \|\cdot\|_\infty$. What could be the reason for such behaviour?

1.1.1 Norm functions

[99]:
```
def norm1(vector):
    norm = 0
    for i in range(len(vector)):
        norm += abs(vector[i])
    return norm
```

[100]:
```
def norm2(vector):
    norm = 0
    for i in range(len(vector)):
        norm += abs(vector[i])**2
    return math.sqrt(norm)
```

[102]:
```
def normInfinity(vector):
    norm = 0
    for i in range(len(vector)):
        if abs(vector[i]) > norm:
            norm = vector[i]
    return norm
```

1.1.2 H_5

$e_5 = x_5 - x_5^*$

[107]:
```
e5 = x5 - x5_star
e5
```

[107]:
```
array([ 0.98571429,  1.28571429, -0.28571429,  3.        , -1.        ])
```

$\|e_5\|_1$

[108]:
```
norm1(e5)
```

[108]: 6.557142857142295

$\|e_5\|_2$

```
[109]: norm2(e5)
```

[109]: 3.5645934593738042

$\|e_5\|_\infty$

```
[110]: normInfinity(e5)
```

[110]: 2.9999999999997815

The answer above should be 3 however, due to roundoff errors with doubles, the function returns 2.9999999999997815.

$r_5 = b_5 - H_5 x_5^*$

```
[90]: r5 = b5 - np.matmul(H5,x5_star)
      r5
```

[90]: array([2.08333333, 1.38333333, 1.08333333, 0.88380952, 0.74558957])

$\|r_5\|_1$

```
[111]: norm1(r5)
```

[111]: 6.179399092970522

$\|r_5\|_2$

```
[112]: norm2(r5)
```

[112]: 2.9604937223406

$\|r_5\|_\infty$

```
[115]: normInfinity(r5)
```

[115]: 2.083333333333333

1.1.3 H_{10}

$e_{10} = x_{10} - x_{10}^*$

```
[116]: e10 = x10 - x10_star
       e10
```

[116]:
```
array([  1.00007198,   0.99352133,   1.14253141,  -0.33029806,
          7.48522249, -17.15866816,  31.26451049, -28.64692068,
         16.7499511 ,  -3.49999389])
```

$\|e_{10}\|_1$

[121]:
```
norm1(e10)
```

[121]:
```
108.27168959197715
```

$\|e_{10}\|_2$

[122]:
```
norm2(e10)
```

[122]:
```
49.44468274549276
```

$\|e_{10}\|_\infty$

[123]:
```
normInfinity(e10)
```

[123]:
```
31.264510489672187
```

$r_{10} = b_{10} - H_{10}x^*_{10}$

[124]:
```
r10 = b10 - np.matmul(H10,x10_star)
r10
```

[124]:
```
array([2.82896825, 1.97896825, 1.59411977, 1.34533189, 1.16804917,
       1.03487901, 0.93072786, 0.84669533, 0.77725093, 0.7187714 ])
```

$\|r_{10}\|_1$

[125]:
```
norm1(r10)
```

[125]:
```
13.223761870980642
```

$\|r_{10}\|_2$

[126]:
```
norm2(r10)
```

[126]:
```
4.6270643572049135
```

$\|r_{10}\|_\infty$

[128]:
```
normInfinity(r10)
```

[128]:
```
2.8289682539682537
```

1.1.4 Analysis

As we can see, the norms of the error and residual for n = 5 are less than that for n = 10. This is most likely due to the fact that as more computations are needed to solve a system of equations, round off errors start to accumulate since the computer doesn't always keep track of all the digits in each calculation.