Hand-in 2

Kriek van der Meulen
Numerical Algorithms

November 22, 2017

1 Excercise 1

1.1

An easy way to solve this is to calculate when the determinant of A is equal to zero. This gives us:

$$\begin{aligned}
0 &= 1*((2*-\frac{3}{2})-(\alpha)+1*((2*-\frac{3}{2})-0)+\alpha*((2*\alpha)-0) \\
0 &= -3-\alpha-3+2\alpha^2 \\
0 &= 2\alpha^2-\alpha-6 \\
0 &= \alpha^2-\frac{1}{2}\alpha-3 \\
0 &= \alpha*(\alpha-\frac{1}{2}-3 \\
\alpha &= 2 \text{ or } \alpha=-\frac{3}{2}
\end{aligned}$$

So for $\alpha = 2$ or $\alpha = -\frac{3}{2}$ the given matrix A is singular. This is because a singular matrix has a zero determinant.

1.2

We have:

$$\begin{aligned}
2x+y+z &= 3 && (1) \\
2x-y+3z &= 5 && (2) \\
-2x+\alpha y+3z &= 1 && (3)
\end{aligned}$$

Adding equation (1) to equation (2) gives us x in terms of a scalar and z and vice versa:

$$\begin{aligned}
2x+y+z+(2x-y+3z) &= 3+5 \\
4x+4z &= 8 \\
x = 2-z \text{ and } z &= 2-x && (4)
\end{aligned}$$

Subtracting equation (2) to equation (1) gives us y in terms of a scalar and z and vice versa:

$$\begin{aligned} 2x + y + z - (2x - y + 3z) &= 3 - 5 \\ 2y - 2z &= -2 \\ y = z - 1 \text{ and } z &= 1 - y \end{aligned} \tag{5}$$

Using equations (5),(4) and (3) we can finally solve for α to see which α gives us infinity solutions:

$$\begin{aligned} -2x + \alpha y + 3z &= 1 \\ -4 + 2z + \alpha y + 3z - 1 &= 0 \\ 5 - 5 - 5y + \alpha y &= 0 \\ \alpha &= 5 \end{aligned}$$

So for $\alpha = 5$ we have infinite solutions.

2 Excercise 2

2.1

For p norms above 1 we get the following values:

P = 1	P = 2	P = 3	P = 10	P = 100	P = inf
34.0898	4.6168	2.4713	1.1579	0.9893	0.9867

Table 1: Values found for Pnorms

for p norms between 0 and 1 we find the following values:

P = 0.5	P = 0.1	P = 0.01	P = 0.001
47.3593	67.9118	75.1104	75.9102

Table 2: Values found for Pnorms

When we compare the values for the Pnorms in Table 2 with the non-zero entries of vector x, 76 entries which are non-zero, we find that as P approaches zero the value of the norm approaches the amount of non-zero entries in the vector.

2.2

First let us cite the 3 properties we need to proof:

1. $\|A\| > 0$ if $A \neq 0$
2. $\|y * A\| = |y| * \|A\|$
3. $\|A + B\| \leq \|A\| + \|B\|$

Now the p norm of a matrix is write-able as:

$$\|A\|_p = (\sum_{k=0}^{n} |x_k|^p)^{\frac{1}{p}} \tag{6}$$

Since we take the absolute value of the elements in the matrix A when we calculate the norm (as seen in (6)). as long as $A \neq 0$ we will always have a value for the norm that is larger than zero. Thus our first property is easily satisfied.

For the second property we need to realize that we multiply the matrix A by a scalar, this scalar can be easily removed from the sum in the norm since it is the same for every indice so we will have:

$$\begin{aligned}
\|y * A\|_p &= (\sum_{k=0}^{n} |y * x_k|^p)^{\frac{1}{p}} \\
&= (|y|^p)^{\frac{1}{p}} * (\sum_{k=0}^{n} |x_k|^p)^{\frac{1}{p}} \\
&= |y| * (\sum_{k=0}^{n} |x_k|^p)^{\frac{1}{p}}
\end{aligned}$$

Thus proving property 2.

Now for the final property we have the norm of the sum of the two matrices A and B. This one is easy once you realize that:

$$|x + y| \leq |x| + |y| \tag{7}$$

In equation (7) $|x + y|$ can be smaller than $|x| + |y|$ once for example, x is a positive number and y is a negative number. It is however the same once both x and y are either positive or negative numbers. Using this fact it is clear to see why the norm of

$\|A + B\|$ is smaller or equal to $\|A\| + \|B\|$

$$\begin{aligned}
\|A + B\|_p &= (\sum_{k=0}^{n} |x_k + y_k|^p)^{\frac{1}{p}} \\
\|A\| + \|B\| &= (\sum_{k=0}^{n} |x_k|^p)^{\frac{1}{p}} + (\sum_{k=0}^{n} |y_k|^p)^{\frac{1}{p}} \\
\|A + B\| &\leq \|A\| + \|B\| \\
(\sum_{k=0}^{n} |x_k + y_k|^p)^{\frac{1}{p}} &\leq (\sum_{k=0}^{n} |x_k|^p)^{\frac{1}{p}} + (\sum_{k=0}^{n} |y_k|^p)^{\frac{1}{p}}
\end{aligned}$$

3 Excercise 3

3.1

We need to calculate A^{-1} using LU decomposition.
We know that matrix $A = L \cdot U$ and that $A \cdot A^{-1} = I$ where I is the identity matrix. This can be rewritten as:

$$A \cdot A^{-1} = I$$
$$L \cdot U \cdot A^{-1} = I$$

We can further write this down as:

$$L \cdot (U \cdot A^{-1}) = I$$
$$U \cdot A^{-1} = x$$
$$L \cdot x = I$$

Now, we know what I is and we found L and U using the inbuilt scipy package. so we can solve for A^{-1} . This is done by first solving for x using L and I and then solving for A^{-1} using U and x

3.2

After filling in matrix A in our code we get the following:

$$A^{-1} = \begin{bmatrix} 0.8 & -0.6 & 0.4 & -0.2 \\ -0.6 & 1.2 & -0.8 & 0.4 \\ 0.4 & -0.8 & 1.2 & -0.6 \\ -0.2 & 0.4 & -0.6 & 0.8 \end{bmatrix}$$

3.3

The computational complexity of our algorithm is $\mathcal{O}(n)$ Cause when solving for L and U we go only once over the rows of a matrix with shape $n * n$

4 Excercise 4

We consider the following equation:

$$H \cdot x = b$$

Where H is the Hilbert matrix we generate and x is a vector of ones. First, we solve the above equation to find our b after which we backsolve the equation to find our value for $\hat{x}$. Now we can calculate the residual using $r = b - H\hat{x}$ and the relative error in percentage using $\Delta x_{rel} = \frac{(\hat{x}-x)\cdot 100}{x}$ of these vectors we then calculate the p-norm and plot the solutions in Figure 1. The condition number we characterize as a function of n also in Figure 1 we also plotted the number of digits lost which is just the log10 of the condition number.

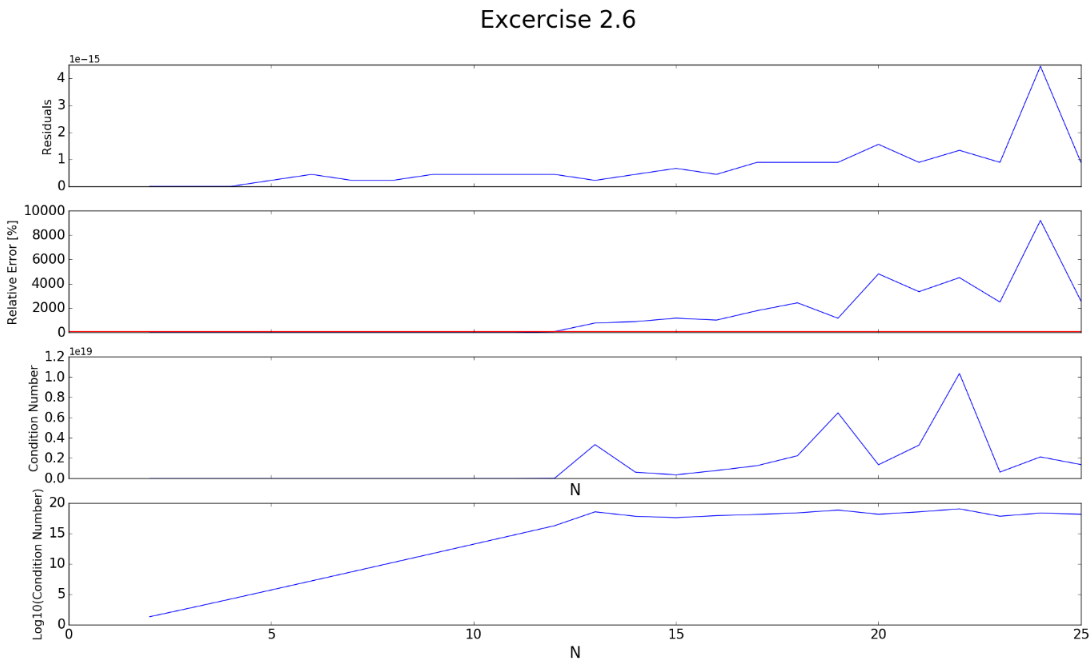

Figure 1: Plot depicting all the values asked for, for different Hilbert matrices of order n, On the x-axis we have the order of n and on the y axis we have the residuals, the relative error,the condition number and the log10 of the condition number

From Figure 1 we can see that around n = 12 our error exceeds 100 percent. We can also see our condition number blow up around that point and since the number of digits lost relate to our condition number by log10 we can also see the number of digits lost blow up.

```python
# EXCERCISE 2

import numpy as np
import scipy as sp
import matplotlib.pyplot as plt

# Generate vector with random entries
x = np.random.rand(100)
# set random entries in the vector to 0
for i in np.nditer(x,op_flags=["readwrite"]):
    if (np.random.randint(4)) == 1:
        i[...] = 0
    else:
        continue
# define pnorms
pnorm = [1,2,3,10,100,np.inf]
pnormvalues = []
# apply pnorms to the vector using np.linalg.norm
for i in range(len(pnorm)):

    pnormvalues.append(np.linalg.norm(x,ord=pnorm[i]))

print '%0.4f & %0.4f & %0.4f & %0.4f & %0.4f & %0.4f' %(pnormvalues[0],pnormvalues[1],pnormvalues[2],pnormvalues[3],pnormvalues[4],pnormvalues[5])

# Set new pnorm values
pnormvaluesp = []
pnormp = [0.5,0.1,0.01,0.001]
# apply pnorms to vector
for i in range(len(pnormp)):

    pnormvaluesp.append(sum(x**pnormp[i]))

print '%0.4f & %0.4f & %0.4f & %0.4f ' %(pnormvaluesp[0],pnormvaluesp[1],pnormvaluesp[2],pnormvaluesp[3])

# find out how many entries in vector x are non zero
indicies = np.nonzero(x)
print np.shape(indicies)[1]
```

```python
# EXCERCISE 3

from scipy import linalg as sl
import numpy as np

# Define the matrix A
A = np.zeros((4,4))
for i in range(np.shape(A)[0]):
    for j in range(np.shape(A)[0]):
        if i == j:
            A[i][j] = 2
        if i == (j-1):
            A[i][j] = 1
        if j == (i-1):
            A[i][j] = 1

# get the L and U
P = sl.lu(A)
U = P[2]
L = P[1]
P = P[0]

# Define identity matrix
b = np.identity(np.shape(L)[0])
# Solve L*d = b
d = sl.solve_triangular(L,b,lower=True)

# Solve U*x = d
Ainv = sl.solve_triangular(U,d)
print Ainv

```

```python
# EXCERCISE 4

import numpy as np
import matplotlib.pyplot as plt

# Create Hilbert matrix
def CreateMatrix(n):
    Matrix = np.empty((n,n))
    for i in range(n):
        for j in range(n):
            Matrix[i][j] = 1./((i+1)+(j+1)-1)
    return Matrix

# Generate vector of ones
def GenX(n):
    x = np.ones((n))
    return x

# get solution and xhat for the matrix and x
def SolveMatrix(n):
    x = GenX(n)
    M = CreateMatrix(n)
    b = np.dot(M,x)
    xhat = np.linalg.solve(M,b)
    return b,xhat,M,x

# calculate the residual and the relative error and take the inf norms of them.
# Also calculate the norm of the condition number
def NormResidual(n):
    b,xhat,M,x = SolveMatrix(n)
    residual = b-np.dot(M,xhat)
    r = np.linalg.norm(residual,ord=np.inf)
    relerror = (((xhat-x)*100)/x)
    errorx = np.linalg.norm(relerror,ord =np.inf)
    cond = np.linalg.cond(M)
    return r,errorx,cond

# define lists and amount of arrays of size nxn
nsize = 26
xlist = []
Reslist = []
Errlist = []
Condloglist = []
Condlist = []
for i in range(2,nsize):
```

```python
    r,errorx,cond = NormResidual(i)
    xlist.append(i)
    Reslist.append(r)
    Errlist.append(errorx)
    Condlist.append(cond)
    # take the log10 of the condition number in order to find the amount of digits that we lose
    Condloglist.append(np.log10(cond))

# plot everything
fig, (ax1,ax2,ax3,ax4) = plt.subplots(4,1, sharex=True)
plt.suptitle("Excercise 2.6",fontsize = 30)
ax1.plot(xlist,Reslist)
ax1.tick_params(axis='both', labelsize=17)
ax1.set_ylabel('Residuals', fontsize = 15)
ax2.plot(xlist,Errlist)
ax2.set_ylabel("Relative Error [%]", fontsize = 15)
ax2.tick_params(axis='both', labelsize=17)
ax2.axhline(100,color='r')
ax3.plot(xlist,Condlist)
ax3.set_ylabel("Condition Number",fontsize = 15)
ax3.set_xlabel("N",fontsize = 20)
ax3.tick_params(axis='both', labelsize=17)
ax4.set_ylabel("Log10(Condition Number)",fontsize = 15)
ax4.set_xlabel("N",fontsize = 20)
ax4.tick_params(axis='both', labelsize=17)
ax4.plot(xlist,Condloglist)
plt.show()
```