Numerical Methods in Scientific Computing 2021
Answers Ex03

Jake Muff 9/02/21

Problem 1

To show that the matrix Q is orthogonal I used

$$Q_{pq}^T \cdot Q_{PQ} = I$$

Where I is the identity matrix. For this I used to traditional notation for the jacobi matrix where

$$c = \cos(\theta)$$

$$s = \sin(\theta)$$

Note that I also used this throughout the exercise. For the 9x9 version of the matrix we have

$$Q_{pq}^T \cdot Q_{pq} = \begin{pmatrix} 1 &&&&&&&& \\ & 1 &&&&&&& \\ && s^2+c^2 &&&&&& \\ &&& 1 &&&&& \\ &&&& 1 &&&& \\ &&&&& 1 &&& \\ &&&&&& s^2+c^2 && \\ &&&&&&& 1 & \\ &&&&&&&& 1 \end{pmatrix}$$

And we know that

$$\cos^2(\theta) + \sin^2(\theta) = 1$$

Using the notation in the exercise we get the same answer

$$\left(\sqrt{\frac{1+C}{2}}\right)^2 + \left(\sqrt{\frac{1-C}{2}}\right)^2 = 1$$

Thus, the matrix Q_{pq} is orthogonal.

Problem 2

The function jacobi(Q,N) is implemented in the jacobi.cpp code. The test matrix used was

$$\begin{pmatrix} 1.0 & 2.0 & 3.0 \\ 4.0 & 5.0 & 6.0 \\ 7.0 & 8.0 & 9.0 \end{pmatrix}$$

The eigenvalues calculated by Maple were

$$0, 16.117, -1.1169$$

From numpy

$$-9.75918483E-16, 16.1168440, -1.11684397$$

The eigenvalues gathered from my jacobi function were

$$0.1919, -1.2943, 16.1024$$

Which are reasonably close. Other matrix eigenvalue calculates such as NumPy show similarly close results. I would argue that my implementation of the jacobi method for calculating eigenvalues works reasonably well. I believe the reason in the difference is due to different algorithm's used and my test matrix may not be the best use fo the jacobi method.

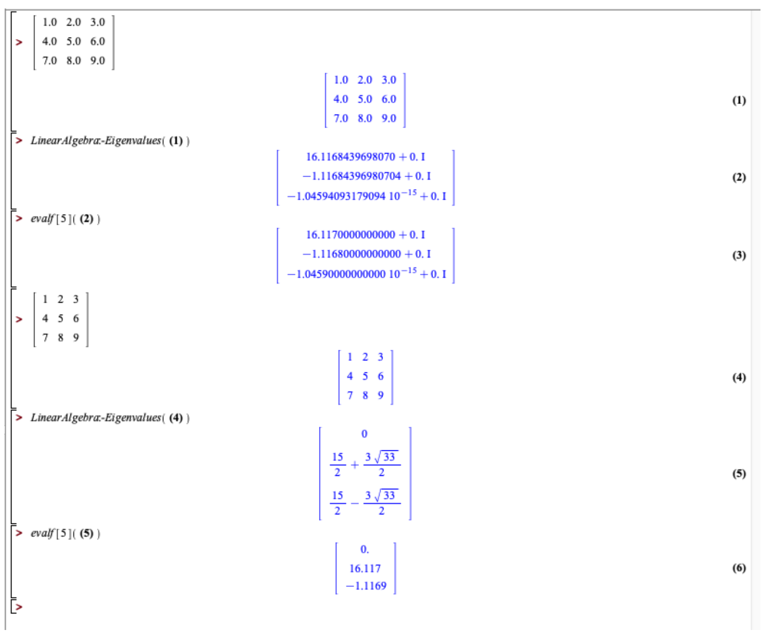


Figure 1: Output from Maple 2020 for the matrix above

Problem 3

The function err_propag(N,dq) is implemented in the jacobi.cpp code. For this to work I created a copy of the above jacobi function which returned the final eigenvalue matrix from which I could extract the eigenvalues themselves to be used in the err_propag(N,dq) function.

```
In [10]:
import numpy as np
from numpy import linalg as LA



A = [[1.0, 2.0, 3.0],
    [4.0, 5.0, 6.0],
    [7.0,8.0,9.0]]

w, v = LA.eig(A)
print(w)

[ 1.61168440e+01 -1.11684397e+00 -9.75918483e-16]
```

Figure 2: Output from Numpy for the matrix above

```
Hello World


Initial matrix a[][]:
      1.0000      2.0000      3.0000
      4.0000      5.0000      6.0000
      7.0000      8.0000      9.0000
Final Matrix with diagonal values as eigenvalues

      0.1919     -0.0000      0.0000
     -0.0000     -1.2943     -0.0000
      0.0000     -0.0000     16.1024
jake@Jakes-MBP Exercises 3 %
```

Figure 3: Output from jacobi.cpp using jacobi function for the matrix above

For part B I was not quite sure how to show my results.

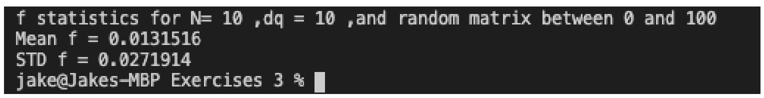
Some key things to point out:

- The random matrix is generated using an implementation of the Mersenne Twister PRNG from rng.h in the zip file. Originally I did random doubles between 0 and 1 but increased this to 100. The difference I found was that the values for f actualyl decreased.
- Matrix Memory Allocation was done via a function found in Numerical Recipes.
- The random pertubation is doen via ran() % n picking a random integer between 0 and n
- As mentioned before the eigenvalues are pulled from a modified variation of the funcion done in the previous question.
- f was quite often filled with zeros. As of writing this I am still unsure why there are so many zeros, I can only think this is due to writing to file roundig very small numbers.

- Various runs for f are found in .txt files in the zip. They are denoted by N then dq so f_3_1 is f for 3x3 with $dq = 1$

```
f statistics for N= 10 ,dq = 1 ,and random matrix between 0 and 100
Mean f = 0.0126296
STD f = 0.0505347
jake@Jakes-MBP Exercises 3 %
```

Figure 4: Output from jacobi.cpp using err_propag(N,dq) function for a random matrix filled with real values from 0 to 100. This run was done for a 10x10 matrix with $dq = 1$ for 100 times



```
f statistics for N= 10 ,dq = 10 ,and random matrix between 0 and 100
Mean f = 0.0131516
STD f = 0.0271914
jake@Jakes-MBP Exercises 3 %
```

Figure 5: Output from jacobi.cpp using err_propag(N,dq) function for a random matrix filled with real values from 0 to 100. This run was done for a 10x10 matrix with $dq = 10$ for 100 times

```
f statistics for N= 3 ,dq = 1 ,and random matrix between 0 and 100
Mean f = 0.0320719
STD f = 0.194322
jake@Jakes-MBP Exercises 3 %
```

Figure 6: Output from jacobi.cpp using err_propag(N,dq) function for a random matrix filled with real values from 0 to 100. This run was done for a 3x3 matrix with $dq = 1$ for 100 times

```
f statistics for N= 3 ,dq = 10 ,and random matrix between 0 and 100
Mean f = 0.00904044
STD f = 0.0227381
jake@Jakes-MBP Exercises 3 %
```

Figure 7: Output from jacobi.cpp using err_propag(N,dq) function for a random matrix filled with real values from 0 to 100. This run was done for a 3x3 matrix with $dq = 10$ for 100 times

```
f statistics for N= 5 ,dq = 10 ,and random matrix between 0 and 100
Mean f = 0.0165649
STD f = 0.0396646
jake@Jakes-MBP Exercises 3 %
```

Figure 8: Output from jacobi.cpp using err_propag(N,dq) function for a random matrix filled with real values from 0 to 100. This run was done for a 5x5 matrix with $dq = 10$ for 100 times