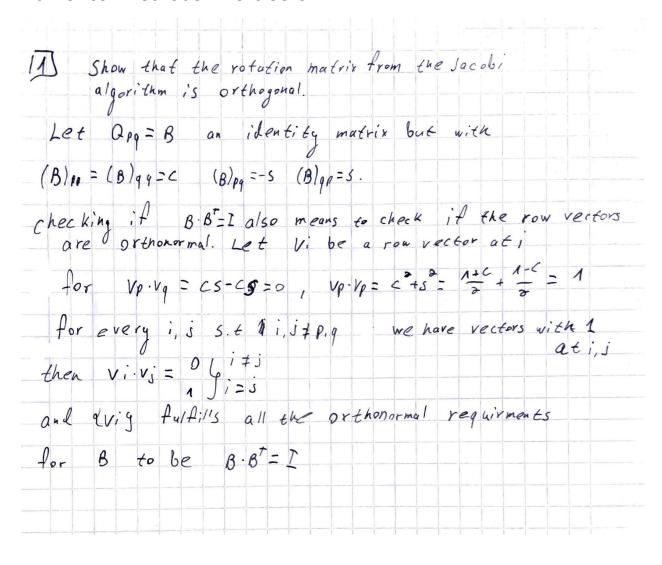
Numerical Methods Exercise 3



2

Calculates the eigenvalues of a given matrix using the jacobi algorithm.

Run code at jacobi.py using python(3), (it uses numpy and numpy.linalg) use test_jacobi() to print eigenvalues of 4 matrices using jacobi algo and numpy's linalg package.

<u>B</u>

The output of test_jacobi for 3 predefined matrices, and one randomly generated 10 x 10 are:

```
[ 3 -10
          311
the jacobi eigenvalues are: [13.04378227 2.29769964 -8.34148191]
while the result from numpy's linalg is: [-8.34148191 2.29769964
13.04378227]
For the matrix
 [[ 1 5 200 100]
     2 0
 [ 5
              01
 [200
       0
          3
              0]
 [100
          0
              511
the jacobi eigenvalues are: [ 225.86701533 4.60007698
2.00057659 -221.46766891]
while the result from numpy's linal gis: [-221.46766891
             4.60007698 2.00057659]
225.86701533
For the matrix
 [[ 1 0 0 100]
 0 ]
       2 0 01
 [ 0 0 3 10]
     0 10
              511
the jacobi eigenvalues are: [103.52855059 2.
                                                   2.98020586
-97.508756451
while the result from numpy's linalg is: [103.52855059 -97.50875645
2.98020586
          2.
                     1
For the matrix
 [[-56. -34.5 -47. 64.5 -90.5 7.
                                   14. -88.
                                              16.5 -59. ]
 [-34.5 	 46.
            46. -83.5 4.5 -61.
                                  -4.5
                                          9.5 32.
 [-47.
       46. -35. -18.5 22. 48. -54. -66.5 -56.5
                                                   43.51
 [ 64.5 -83.5 -18.5 70.
                        1.5 29. 68.5 54. 27.5 -29.5]
                  1.5 -89. -25.5 -5.5 -22. -21.5 0.]
 [-90.5 4.5 22.
 [ 7. -61. 48.
                  29. -25.5 37.
                                   7.5 -52. -38.5 -56. 1
 [ 14. -4.5 -54. 68.5 -5.5 7.5 13.
                                         53. 64.5 21. 1
 [-88.
       9.5 -66.5 54. -22. -52.
                                   53.
                                         90.
                                             -15.
 [ 16.5 32. -56.5 27.5 -21.5 -38.5 64.5 -15. -31. -39.5]
       4.5 43.5 -29.5 0. -56.
                                  21.
                                        1.5 -39.5 -73. ]]
the jacobi eigenvalues are: [ 247.74302641 200.44069249
97.25290829 13.62744394 -3.64339941
 -27.441335 -51.69562761 -113.21591146 -164.50171125 -226.5660864
while the result from numpy's linalg is: [ 247.74302641 200.44069249
-226.5660864 97.25290829 -164.50171125
-113.21591146 -51.69562761 -27.441335 13.62744394
-3.64339941]
```

And we can clearly see that the jacobi method gives exact results compared to the package which actually uses lapack's _geev behind the scenes.

I don't see any difference, the 1st problem did state that it fits for small symmetric matrices, hence I used small ones.

3

Run using python(3), prerequisites are again numpy.

Matrices are generated randomly for values between [-100, 100]

В

Calculated using the err_propag_stats(N, dq) func.

```
For N=20, dq=1
The error propag mean: 0.00025601637591712547, standard deviation: 5.4375039899289913e-05

For N=12, dq=0.1
The error propag mean: 0.0005409561519609541, standard deviation: 0.000196768770406304

For N=10, dq=0.001
The error propag mean: 0.0007649821110136015, standard deviation: 0.0002642929156775915

For N=9, dq=2
The error propag mean: 0.0008423355300222206, standard deviation: 0.000297719902599775
```

I don't see any out of the ordinary error propagations, trying more extreme values I get

```
For N=30, dq=20 The error propag mean: 0.00014197612235776312, standard deviation: 2.47973311674375e-05
```

I do see that the mean of the propagation factor grows smaller as we introduce a bigger dq