Task 1

I have created the following code that implement the power method for finding the eigenvalue of largest magnitude. The code is documented here in the Software Manual.

```
import numpy as np
from matrixD import matrixD

n = int(input('Enter dimension of the diagonal dominant matrix: '))
mat = matrixD(n)
x = np.zeros(n)

for i in range(n):
    x[i] = i

# Reading tolerable error
tolerable_error = float(input('Enter tolerable error: '))

# Reading maximum number of steps
max_iteration = int(input('Enter maximum number of steps: '))

# Power Method Implementation
lambda_old = 1.0
condition = True
step = 1
while condition:
    x = np.matmul(mat, x)
    # Finding new Eigen value and Eigen vector
    lambda_new = max(abs(x))
    x = x / lambda_new

    # Displaying Eigen value and Eigen Vector
    print('\nSTEP %d' % (step))
    print('----------')
    print('Eigen Value = %0.4f' % (lambda_new))
    # print('Eigen Vector: ')
    # for i in range(n):
        # print('%0.3f\t' % (x[i]))

    # Checking maximum iteration
    step = step + 1
    if step > max_iteration:
        print('Not convergent in given maximum iteration!')
        break

    # Calculating error
    error = abs(lambda_new - lambda_old)
    print('errror=' + str(error))
    lambda_old = lambda_new
    condition = error > tolerable_error
```

with the following output:

```
Enter dimension of the diagonal dominant matrix: 100
The random diagonal matrix of size 100 is the following
 [[20.  0.  0. ...  0.  0.  0.]
 [ 0. 20.  0. ...  0.  0.  0.]
 [ 0.  0. 20. ...  0.  0.  0.]
 ...
 [ 0.  0.  0. ... 20.  0.  0.]
 [ 0.  0.  0. ...  0. 20.  0.]
 [ 0.  0.  0. ...  0.  0. 20.]]
Enter tolerable error: 5
Enter maximum number of steps: 50

STEP 1
----------
Eigen Value = 1980.0000
errror=1979.0

STEP 2
----------
Eigen Value = 20.0000
errror=1960.0

STEP 3
----------
Eigen Value = 20.0000
errror=0.0
```

Figure 1. Running the powerMethod.py From the IDE.

Task 2

I have created the following code that implement the power method for finding the eigenvalue of smallest magnitude. The code is documented here in the Software Manual.

```python
import numpy as np
from matrixD import matrixD

n = int(input('Enter dimension of the diagonal dominant matrix: '))
mat = matrixD(n)
x = np.ones(n)

for i in range(n):
    x[i] = i

# Reading tolerable error
tolerable_error = float(input('Enter tolerable error: '))

# Reading maximum number of steps
max_iteration = int(input('Enter maximum number of steps: '))

# Power Method Implementation
lambda_old = 1.0
condition = True
step = 1
while condition:
    x = np.matmul(mat, x)
    # Finding new Eigen value and Eigen vector
    lambda_new = max(abs(x))
    x = x / lambda_new

    # Displaying Eigen value and Eigen Vector
    print('\nSTEP %d' % (step))
    print('----------')
    print('Eigen Value = %0.4f' % (lambda_new))
    # print('Eigen Vector: ')
    # for i in range(n):
        # print('%0.3f\t' % (x[i]))

    # Checking maximum iteration
    step = step + 1
    if step > max_iteration:
        print('Not convergent in given maximum iteration!')
        break

    # Calculating error
    error = abs(lambda_new - lambda_old)
    print('errror=' + str(error))
    lambda_old = lambda_new
    condition = error > tolerable_error
```

with the following output:

```
Enter dimension of the diagonal dominant matrix: 100
The random diagonal matrix of size 100 is the following
 [[24.  0.  0. ...  0.  0.  0.]
 [ 0. 24.  0. ...  0.  0.  0.]
 [ 0.  0. 24. ...  0.  0.  0.]
 ...
 [ 0.  0.  0. ... 24.  0.  0.]
 [ 0.  0.  0. ...  0. 24.  0.]
 [ 0.  0.  0. ...  0.  0. 24.]]
Enter tolerable error: 5
Enter maximum number of steps: 50

STEP 1
----------
Eigen Value = 2376.0000
errror=2375.0

STEP 2
----------
Eigen Value = 24.0000
errror=2352.0

STEP 3
----------
Eigen Value = 24.0000
errror=0.0
```

Figure 2. Running the invPowerMethod.py From the IDE.

Task 3

I have created the following routine that compute the 1-matrix norm for a square matrix. The routine is documented here in the Software Manual.

```
import numpy as np

def mat1Norm(mat):
    tran = np.transpose(mat)
    colSums = []
    for i in range(len(tran)):
        sum = 0
        for j in range(len(tran[i])):
            sum += np.abs(tran[i][j])
        colSums.append(sum)
    return np.max(colSums)
```

I have also created the following testing routine for the 1-matrix norm

```
import numpy as np
from mat1Norm import mat1Norm

# testing matrix
A = [[1, 2, 3], [5, 5, 66], [9, 3, 11]]

print('The Matrix 1-Norm Result(Routine): ', mat1Norm(A))
print('The Matrix 1-Norm Result(Numpy): ', np.linalg.norm(A, ord=1))
```

with the following output:

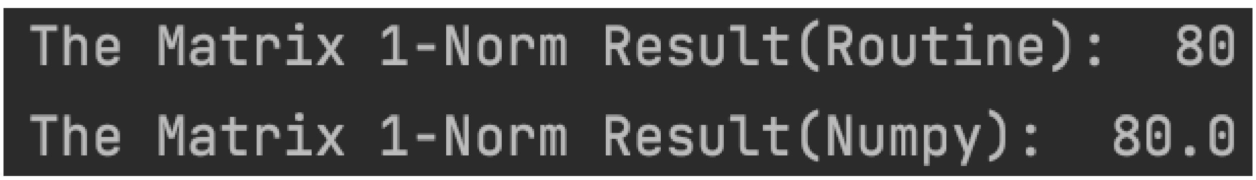

Figure 3. Testing the Routine with Numpy.

Task 4

I have created the following routine that compute the ∞-matrix norm for a square matrix. The routine is documented here in the Software Manual.

```
import numpy as np

def matInfNorm(mat):
    rowSums = []
    for i in range(len(mat)):
        sum = 0
        for j in range(len(mat[i])):
            sum += np.abs(mat[i][j])
        rowSums.append(sum)
    return np.max(rowSums)
```

I have also created the following testing routine for the ∞-matrix norm

```
import numpy as np
from matInfNorm import matInfNorm

# testing matrix
A = [[1, 2, 3], [5, 5, 66], [9, 3, 11]]

print('The Matrix 1-Norm Result(Routine): ', matInfNorm(A))
print('The Matrix 1-Norm Result(Numpy): ', np.linalg.norm(A, ord=np.inf))
```

with the following output:

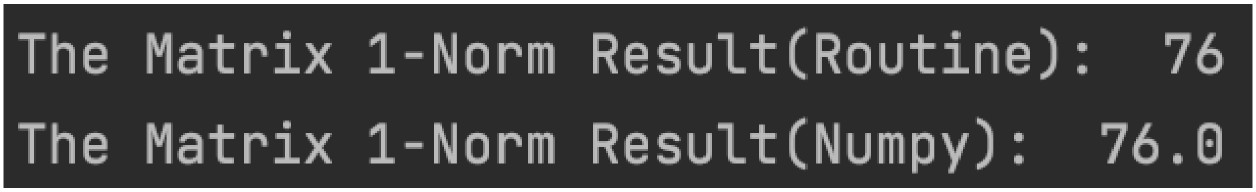

Figure 4. Testing the Routine with Numpy.

Task 5

Before jumping on to this task, I have realized my Power Method is not sufficient enough, so I have updated my power method and inverse method routine into the following, they are all documented in the Software Manual.

```
import numpy as np

def ScaMul(c, arr):
    ScalarProduct = c * arr
    return ScalarProduct

def newPowerMethod(mat, n, vec):
    eigenVector = vec
    for i in range(n):
        eigenVector = np.dot(mat, eigenVector)
        eigenVector = ScaMul((1/np.linalg.norm(eigenVector)), eigenVector)

    eigenValue = np.dot(eigenVector, np.dot(mat, eigenVector)) * (1/np.dot(eigenVector, eigenVector))

    return eigenValue
```

```
import numpy as np

def ScaMul(c, arr):
    ScalarProduct = c * arr
    return ScalarProduct

def newInvPowerMethod(mat, n, vec):
    eigenVector = vec
    invMat = np.linalg.inv(mat)

    for i in range(n):
        eigenVector = np.dot(invMat, eigenVector)
        eigenVector = ScaMul((1/np.linalg.norm(eigenVector, ord= 2)), eigenVector)

    eigenValue = np.dot(eigenVector, np.dot(mat, eigenVector)) * (1/np.dot(eigenVector, eigenVector))

    return eigenValue
```

Hence, the following is the routine for computing the condition number, it is documented here in the Software Manual.

```
import numpy as np
from matrixD import matrixD
from newPowerMethodRoutine import newPowerMethod
from newInversePowerRoutine import newInvPowerMethod

n = int(input('Enter dimension of the diagonal dominant matrix: '))
A = matrixD(n)
x = np.ones(n)

def conditionNumber(mat):
    lM = newPowerMethod(mat, n, x)
    lm = newInvPowerMethod(mat, n, x)
```

```python
    return np.abs(lM)/np.abs((lm))

print('Condition Number(Routine):', conditionNumber(A))
print('Condition Number(Numpy):', np.linalg.cond(A))
```

which has the following output:

```
Enter dimension of the diagonal dominant matrix: 200
The random diagonal matrix of size 200 is the following
 [[20.  0.  0. ...  0.  0.  0.]
 [ 0. 20.  0. ...  0.  0.  0.]
 [ 0.  0. 20. ...  0.  0.  0.]
 ...
 [ 0.  0.  0. ... 20.  0.  0.]
 [ 0.  0.  0. ...  0. 20.  0.]
 [ 0.  0.  0. ...  0.  0. 20.]]
Condition Number(Routine): 1.0000000000000002
Condition Number(Numpy): 1.0
```

Figure 5. Testing the Routine with Numpy.

Task 6

Please click here to access the Software Manual.