Open in Colab
(https://colab.research.google.com/github/schneider128k/quantum_computing_slides/blob/master/hw_1_problem_2.ipynb)

HW 1

Problem 2

Some definitions

Let

$$M \in \mathbb{R}^{n \times n}$$

be an arbitrary matrix.

Let

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n \in \mathbb{R}[x]$$

be an arbitrary polynomial of less or equal to n.

The above polynomial can be used to define a matrix function that takes matrices as input and outputs matrices as follows:

$$p(M) = a_0 I + a_1 M + \ldots + a_n M^n,$$

that is, each monomial x^k is substituted by the corresponding matrix power M^k.

We say that a polynomial $p(x)$ annihilates a matrix $M \in \mathbb{R}^{n \times n}$ iff $p(M) = \mathbf{0}$, where $\mathbf{0}$ is the zero matrix.

Task

The task is to write a function `annihilate_poly` that takes as input an arbitrary square numpy array M and outputs a vector whose cofficients are the coefficients of a (non-trivial) polynomial that annihilates M. One-trivial means that its is not the zero polynomial which maps every matrix to the zero matrix.

Hint

You can reduce the problem to finding a linear dependance relationship between the $n+1$ vectors

$$\mathrm{vec}(I), \mathrm{vec}(M), \mathrm{vec}(M^2), \ldots, \mathrm{vec}(M^n) \in \mathrm{R}^{n^2}.$$

The operation vec turns a square matrix $M \in \mathbb{R}^{n \times n}$ into a vector $v \in \mathbb{R}^{n^2}$ by first listing the entries of the first row, then those of the second row etc.

Update:

To solve this problem, you have to compute the null space of the matrix $A \in \mathbb{R}^{n^2 \times (n+1)}$ whose columns are the vectors $\mathrm{vec}(M^k)$ for $k \in \{0, \ldots, n\}$.

(This is not needed:

If you don't remeber how to compute the find a linear dependance relationship, check out this stackoverflow post: https://math.stackexchange.com/questions/2198960/finding-linear-dependence-relation (https://math.stackexchange.com/questions/2198960/finding-linear-dependence-relation)

You can use https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.solve.html (https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.solve.html) to solve the resulting matrix equation.)

Task

Write a function `annihilate_min_deg_poly` that computes a non-trivial polynomial that annihilates a given square matrix and has the smallest possible degree. Recall that a polynomial $p(x)$ has degree d if the coefficient $a_{d+1} = \ldots = a_n = 0$.

###
###
###

Solution

Import required libraries

In [99]:
```
import numpy as np
from scipy.linalg import null_space
np.set_printoptions(precision=8)
```

Define annihilate_poly function

In [100]:
```
def annihilate_poly(M):
    n = M.shape[1]
    I = np.eye(n, dtype=int)
    A =  I.flatten()
    A = np.reshape(A, (-1, 1))
    for x in range(1,n+1):
        temp = np.linalg.matrix_power(M,x)
        temp = np.transpose(temp.flatten())
        temp = np.reshape(temp, (-1, 1))
        A = np.concatenate((A,temp),axis=1)

    print("Matrix A=")
    print(A)
    Coef = null_space(A)
    return Coef
```

define annihilate_min_def_poly function

In [101]:
```
def annihilate_min_def_poly(M):
    n = M.shape[1]
    I = np.eye(n, dtype=int)
    A =  I.flatten()
    A = np.reshape(A, (-1, 1))
    x=1
    while True:
        temp = np.linalg.matrix_power(M,x)
        temp = np.transpose(temp.flatten())
        temp = np.reshape(temp, (-1, 1))
        A = np.concatenate((A,temp),axis=1)
        Coef = null_space(A)
        if(Coef.size!=0):
            print("Min Degree is")
            print(x)
            break
        x=x+1
#     print("Matrix A=")
#     print(A)
    return Coef
```

Function to test the polynomial

In [102]:
```
def Polytest(M,Coef):
    n = M.shape[1]
    I = np.eye(n, dtype=int)
    Val = I*Coef[0]
    for x in range(1,Coef.shape[0]):
        Val = Val + np.linalg.matrix_power(M,x)*Coef[x]

    return Val
```

We created 3 sample matrixes M1, M2, and M3 with sizes n=2, 3, and 4

In [103]:
```
S = 2
M1 = np.random.randint(10,size=(S,S))
S = 3
M2 = np.random.randint(10,size=(S,S))
S = 4
M3 = np.random.randint(10,size=(S,S))
```

##

Test Cases On annihilate_poly

##

Test for Matrix M1

In [104]:
```
print("M1 Matrix is:")
print(M1)
Coef = annihilate_poly(M1)
print("=============================================")
print("Coefficients:")
print(Coef)
print("=============================================")
Out = Polytest(M1,Coef)
print("P(M)=")
print(Out)
```

```
M1 Matrix is:
[[7 1]
 [8 4]]
Matrix A=
[[ 1  7 57]
 [ 0  1 11]
 [ 0  8 88]
 [ 1  4 24]]
=============================================
Coefficients:
[[-0.87537622]
 [ 0.48145692]
 [-0.04376881]]
=============================================
P(M)=
[[-2.66453526e-15 -4.99600361e-16]
 [-3.99680289e-15 -8.88178420e-16]]
```

Test for Matrix M2

In [105]:
```
print("M2 Matrix is:")
print(M2)
Coef = annihilate_poly(M2)
print("=============================================")
print("Coefficients:")
print(Coef)
print("=============================================")
Out = Polytest(M2,Coef)
print("P(M)=")
print(Out)
```

```
M2 Matrix is:
[[8 9 2]
 [1 0 3]
 [3 0 2]]
Matrix A=
[[  1   8  79 845]
 [  0   9  72 711]
 [  0   2  47 468]
 [  0   1  17 169]
 [  1   0   9 153]
 [  0   3   8  77]
 [  0   3  30 297]
 [  0   0  27 270]
 [  1   2  10 161]]
=============================================
Coefficients:
[[-0.9873929 ]
 [ 0.0156729 ]
 [-0.15672903]
 [ 0.0156729 ]]
=============================================
P(M)=
[[-5.32907052e-15 -7.10542736e-15 -5.32907052e-15]
 [-1.77635684e-15  0.00000000e+00 -2.22044605e-16]
 [-2.66453526e-15 -2.66453526e-15  4.44089210e-16]]
```

Test for Matrix M2

In [106]:
```
print("M3 Matrix is:")
print(M3)
Coef = annihilate_poly(M3)
print("=============================================")
print("Coefficients:")
print(Coef)
print("=============================================")
Out = Polytest(M3,Coef)
print("P(M)=")
print(Out)
```

```
M3 Matrix is:
[[3 5 2 4]
 [5 3 7 3]
 [4 9 1 3]
 [3 5 1 6]]
Matrix A=
[[     1      3     54    861 13954]
 [     0      5     68   1182 18558]
 [     0      2     47    688 11587]
 [     0      4     57    903 14472]
 [     0      5     67   1129 17800]
 [     1      3    112   1380 24703]
 [     0      7     41   1027 14080]
 [     0      3     68   1135 18547]
 [     0      4     70   1057 17371]
 [     0      9     71   1558 22453]
 [     1      1     75    776 14898]
 [     0      3     64   1102 17842]
 [     0      3     56    903 14703]
 [     0      5     69   1249 19498]
 [     0      1     48    709 12229]
 [     1      6     66    971 15312]]
=============================================
Coefficients:
[[-0.20642099]
 [-0.95271226]
 [ 0.21912382]
 [ 0.0412842 ]
 [-0.00317571]]
=============================================
P(M)=
[[-6.09645667e-12  7.95807864e-13  6.11066753e-13  6.46593890e-13]
 [ 7.67386155e-13 -5.27222710e-12  2.98427949e-13  9.16600129e-13]
 [ 8.52651283e-13  6.53699317e-13 -5.74118530e-12  8.59756710e-13]
 [ 7.10542736e-13  8.17124146e-13  6.96331881e-13 -6.13908924e-12]]
```

##

Test Cases On annihilate_min_def_poly

##

Test for Matrix M1

In [107]:
```
print("M1 Matrix is:")
print(M1)
Coef = annihilate_min_def_poly(M1)
print("=============================================")
print("Coefficients:")
print(Coef)
print("=============================================")
Out = Polytest(M1,Coef)
print("P(M)=")
print(Out)
```

```
M1 Matrix is:
[[7 1]
 [8 4]]
Min Degree is
2
=============================================
Coefficients:
[[-0.87537622]
 [ 0.48145692]
 [-0.04376881]]
=============================================
P(M)=
[[-2.66453526e-15 -4.99600361e-16]
 [-3.99680289e-15 -8.88178420e-16]]
```

Test for Matrix M2

In [108]:
```
print("M2 Matrix is:")
print(M2)
Coef = annihilate_min_def_poly(M2)
print("=============================================")
print("Coefficients:")
print(Coef)
print("=============================================")
Out = Polytest(M2,Coef)
print("P(M)=")
print(Out)
```

```
M2 Matrix is:
[[8 9 2]
 [1 0 3]
 [3 0 2]]
Min Degree is
3
=============================================
Coefficients:
[[-0.9873929 ]
 [ 0.0156729 ]
 [-0.15672903]
 [ 0.0156729 ]]
=============================================
P(M)=
[[-5.32907052e-15 -7.10542736e-15 -5.32907052e-15]
 [-1.77635684e-15  0.00000000e+00 -2.22044605e-16]
 [-2.66453526e-15 -2.66453526e-15  4.44089210e-16]]
```

Test for Matrix M3

In [109]:
```
print("M3 Matrix is:")
print(M3)
Coef = annihilate_min_def_poly(M3)
print("=============================================")
print("Coefficients:")
print(Coef)
print("=============================================")
Out = Polytest(M3,Coef)
print("P(M)=")
print(Out)
```

```
M3 Matrix is:
[[3 5 2 4]
 [5 3 7 3]
 [4 9 1 3]
 [3 5 1 6]]
Min Degree is
4
=============================================
Coefficients:
[[-0.20642099]
 [-0.95271226]
 [ 0.21912382]
 [ 0.0412842 ]
 [-0.00317571]]
=============================================
P(M)=
[[-6.09645667e-12  7.95807864e-13  6.11066753e-13  6.46593890e-13]
 [ 7.67386155e-13 -5.27222710e-12  2.98427949e-13  9.16600129e-13]
 [ 8.52651283e-13  6.53699317e-13 -5.74118530e-12  8.59756710e-13]
 [ 7.10542736e-13  8.17124146e-13  6.96331881e-13 -6.13908924e-12]]
```

##

For bigger matrices with size bigger than 7 annihilation needs polynomials of order 49 and is not very effective as for smaller matrices. As you can see in the following example the P(M) annihilates to numbers of order 1e-6 or 1e-7 while for matrices M of size 4 P(M) annihilated to numbers of order 1e-13

Test Case with M 7x7 Matrix.

In [110]:
```
S = 7
M6 = np.random.randint(10,size=(S,S))
print("M6 Matrix is:")
print(M6)
Coef = annihilate_min_def_poly(M6)
print("============================================")
print("Coefficients:")
print(Coef)
print("============================================")
Out = Polytest(M6,Coef)
print("P(M)=")
print(Out)
```

```
M6 Matrix is:
[[9 5 9 2 6 8 1]
 [4 7 2 8 3 4 7]
 [7 1 6 5 6 5 5]
 [0 8 4 0 5 4 4]
 [3 0 9 6 7 5 6]
 [2 7 9 7 2 9 6]
 [8 0 9 2 4 7 8]]
Min Degree is
49
=============================================
Coefficients:
[[ 9.75465385e-01]
 [-1.96017866e-01]
 [ 9.89339306e-02]
 [-1.59928555e-02]
 [-6.11107228e-04]
 [ 4.60436073e-04]
 [-1.19580377e-05]
 [ 1.57238023e-10]
 [ 4.98751429e-10]
 [ 9.17183274e-11]
 [ 7.19203586e-10]
 [-9.70418121e-10]
 [ 8.26199886e-10]
 [ 1.54031343e-10]
 [ 6.21436347e-10]
 [ 3.49476309e-10]
 [ 3.27648908e-10]
 [ 1.61931038e-10]
 [-5.08327880e-10]
 [ 3.48452683e-10]
 [ 1.13643658e-09]
 [ 5.52772550e-11]
 [-4.06716077e-10]
 [-7.59699373e-10]
 [ 2.96333562e-10]
 [-9.87264437e-11]
 [ 9.67531055e-11]
 [-8.22456547e-11]
 [-2.18844970e-10]
 [-4.03075143e-11]
 [ 5.40788858e-10]
 [-9.79834547e-11]
 [ 4.58642568e-10]
 [-3.65652788e-10]
 [-2.97700038e-10]
 [ 3.71099262e-10]
 [ 2.68315828e-10]
 [ 3.15387036e-10]
 [-3.91203375e-10]
 [ 5.32437983e-10]
 [ 2.69968214e-10]
 [-4.51919006e-10]
 [ 7.01375583e-10]
 [-1.02069464e-11]
 [-1.89863736e-10]
 [-3.80329684e-10]
 [ 9.04949032e-10]
 [ 1.10077614e-10]
 [-9.70261094e-11]
 [ 7.85955745e-11]]
=============================================
P(M)=
```