

## Project 2b and 2c modules

### 1. my\_database\_reap\_all\_eigen

This module has a function *reap\_data\_outcomes(data\_base)* and it returns  $n \times 1 \times 8$  number array. It runs the sql query:

```
"SELECT * FROM master_list;"
```

### 2. generate\_matrix\_from\_datafn

It uses the function *c\_data\_gen(myarray)* which takes the list of  $n \times 1 \times 8$  lists and reconstruct the complex numbers to output a list of  $1 \times 4$  matrices which can be used in dot product module, e.g.:

input data  $[[[1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 2.0]], [(1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 2.0)]]$  .

example of output data:  $\text{array}([[[1.+0.j, 1.+1.j],$

$[0.+0.j, 1.+2.j]],$

$[[1.+1.j, 1.+2.j],$

$[1.+1.j, 1.+2.j]])]$

### 3. generate\_data\_filec\_r

It uses the function *data\_gen(myarray)* function will take a list of  $1 \times 4$  matrices of complex numbers and return a list of lists where each element has 8 numbers  $r$ , and  $im$  coefficients of the numbers in the  $2 \times 2$  matrix, e.g. :

$\text{my\_list} = [[1.0, (1+1j), 0, (1+2j)], [(1+1j), (1+2j), (1+1j), (1+2j)]]$

will produce an output which has the form

$[[[1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 2.0]], [(1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 2.0)]]$ .

### 4. root\_eigen1

This has a function *r\_e(my\_array1, my\_array2)* which uses *gen\_matrix\_from\_datafn* as *make\_matrix* and *generate\_data\_filec\_r* as *make\_data*. The first of these is outlined above, the second is outlined in project 1 (and above).

It reformats *my\_array1* and *my\_array2* as lists of matrices ( in this case each list will have 1 matrix) called first (this is the input matrix) and next (this is a matrix taken from the database) . It takes the difference first – next and assigns the result to the variable *c* . It then reformats this as a numpy matrix and calls it *x*. It uses the numpy library to take the Hermitian conjugate (*z*).

the dot  
product  
of *w*  
and *z*  
is

```
getH (numpy.matrix.getH.html#numpy.matrix.getH)(self)  
getI (numpy.matrix.getI.html#numpy.matrix.getI)(self)  
getT (numpy.matrix.getT.html#numpy.matrix.getT)(self)
```

```
(numpy.ndarray.html#numpy.ndarray).  
Returns the (complex) conjugate transpose of self.  
Returns the (multiplicative) inverse of invertible se.  
Returns the transpose of the matrix.
```

performed and the eigenvalues calculated using the numpy library

it then returns the square root of the maximum eigenvalue.

root\_eigen1.py - C:\Users\stall\Documents\future-Learn\using\_database\root\_eigen1.py (3.8.2)

File Edit Format Run Options Window Help

```
#this takes 2 arrays of 1x8 shape representing  
#two complex 2x2 matrices
```

```
def r_e(my_array1,my_array2):
```

```
    import numpy as np  
    import cmath  
    import math  
    import lib.gen_matrix_from_datafn as make_matrix  
    import lib.generate_data_filec_r as make_data  
    #
```

```
    first = make_matrix.c_data_gen(my_array1) # this will be input matrix
```

```
    next_ = make_matrix.c_data_gen(my_array2) # this will be turn[i]
```

```
    #print("first  ",first)
```

```
    #print("next   ",next_)
```

```
    c = first - next_
```

```
    #print("c     ",c)
```

```
    x = np.matrix(c)
```

```
    #print("x     ",x)
```

```
    z = x.getH()
```

```
    #print("z     ",z)
```

```
    w = np.dot(x,z)
```

```
    #print("w     ",w)
```

```
    eigenvalues, eigenvectors = np.linalg.eigh(w)
```

```
    #print("eigenvalues  ",eigenvalues)
```

```
    my_eigenvalue = max(eigenvalues)
```

```
    my_eigen = math.sqrt(my_eigenvalue)
```

```
    return my_eigen
```

Testing of modules is included in

## 5. enter\_matrix\_fn1

This has a procedure called  
my\_matrix()

enter\_matrix\_fn1.py - C:\Users\stall\Documents\database working files\_nov\working files nov9\lib\enter\_matrix\_fn1.py (3.8.2)

File Edit Format Run Options Window Help

```
for n in range(4):  
    print(n+1,"value  ")  
    q1 = input("is the number in complex form? if so Enter as complex , e.g 1.0+1.0j, enter y or  n\n")  
    if q1 == "y":  
        a = complex(input("enter your value \n "))  
        print("a  ",a)  
        number.append(a)  
    elif q1 == "n":  
        q3 = input("is your value in exponential (exp (i*n*pi)) form, n can be fractional and negative ?  y/n \n ")  
        if q3 == "y":  
            nn = float(input("enter your value of n \n "))  
            ag = nn*(math.pi)  
            a = complex(math.cos(ag),math.sin(ag))  
            print("a  ",a)  
            number.append(a)  
        else:  
            print(" you have entered an invalid character\n")
```

```
#entries = list(map(complex,number)) #input().split()  
#print("entries  ",entries)
```

```
# For printing the matrix  
matrix = np.array(number).reshape(R, C)
```

```
a = matrix[0][0].real  
b = matrix[0][0].imag  
c = matrix[0][1].real  
d = matrix[0][1].imag  
e = matrix[1][0].real  
f = matrix[1][0].imag  
g = matrix[1][1].real  
h = matrix[1][1].imag
```

```
data = [(a,b,c,d,e,f,g,h)]  
print("matrix  :",matrix)  
print("\n"," ", "\n")  
return data
```

## 6. gen\_names

This has a function *g\_names(data\_base,string)* which runs `SELECT matrices from master_listd`  
where string = ? using the command `cur.execute(sql_update_query, (string, ))`

## 7. calculate\_phi\_2\_test

This module has *phi\_22(my\_array)* which return phi (solution 2)

## 8. calculate\_phi\_1\_test

This module has *phi\_21(my\_array)* which return phi (solution 1)

## 9. gen\_v\_matrix 1

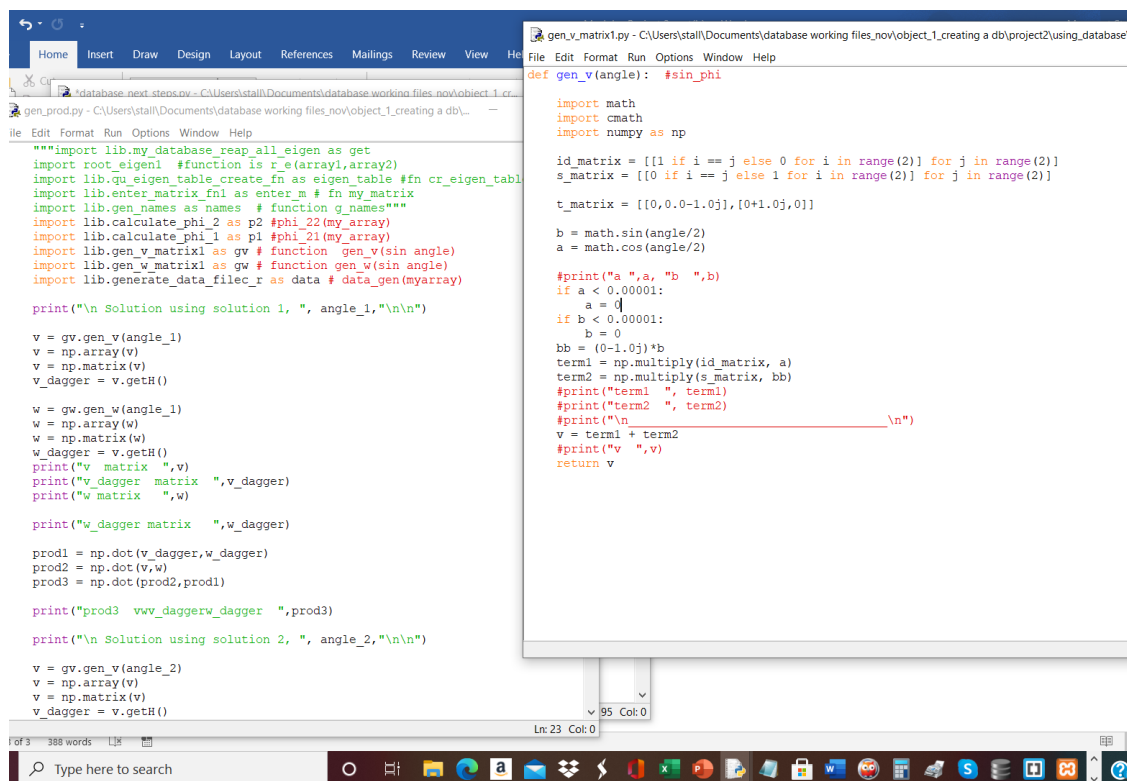
This module has function *gen\_v( angle)*

## 10. gen\_w\_matrix 1

This module has function *gen\_w(angle)*

## 11. gen\_prod

This module has *gen\_prod(angle\_1,angle\_2)*



The screenshot shows a Python IDE with two files open. The left file, `gen_prod.py`, contains a docstring and several imports, followed by a function `gen_v` and a function `gen_w`. The right file, `gen_v_matrix1.py`, contains a function `gen_v` that calculates a vector `v` based on an angle and a parameter `a`. The code in `gen_v` uses `math`, `cmath`, and `numpy` to calculate the vector components and the resulting matrix `v`.

```
"""import lib.my_database_reap_all_eigen as get
import root_eigen1 #function is r_e(array1,array2)
import lib.qu_eigen_table_create_fn as eigen_table #fn cr_eigen_tabl
import lib.enter_matrix_fn1 as enter_m # fn my_matrix
import lib.gen_names as names # function g_names"""
import lib.calculate_phi_2 as p2 #phi_22(my_array)
import lib.calculate_phi_1 as p1 #phi_21(my_array)
import lib.gen_v_matrix1 as gv # function gen_v(sin angle)
import lib.gen_w_matrix1 as gw # function gen_w(sin angle)
import lib.generate_data_filec_r as data # data_gen(myarray)

print("\n Solution using solution 1, ", angle_1,"\n\n")

v = gv.gen_v(angle_1)
v = np.array(v)
v = np.matrix(v)
v_dagger = v.getH()

w = gw.gen_w(angle_1)
w = np.array(w)
w = np.matrix(w)
w_dagger = v.getH()
print("v matrix ",v)
print("v_dagger matrix ",v_dagger)
print("w matrix ",w)

print("w_dagger matrix ",w_dagger)

prod1 = np.dot(v_dagger,w_dagger)
prod2 = np.dot(v,w)
prod3 = np.dot(prod2,prod1)

print("prod3 vw_daggerw_dagger ",prod3)

print("\n Solution using solution 2, ", angle_2,"\n\n")

v = gv.gen_v(angle_2)
v = np.array(v)
v = np.matrix(v)
v_dagger = v.getH()
```

```
def gen_v(angle): #sin_phi

import math
import cmath
import numpy as np

id_matrix = [[1 if i == j else 0 for i in range(2)] for j in range(2)]
s_matrix = [[0 if i == j else 1 for i in range(2)] for j in range(2)]

t_matrix = [[0,0.0-1.0j],[0+1.0j,0]]

b = math.sin(angle/2)
a = math.cos(angle/2)

#print("a ",a, "b ",b)
if a < 0.00001:
    a = 0
if b < 0.00001:
    b = 0
bb = (0-1.0j)*b
term1 = np.multiply(id_matrix, a)
term2 = np.multiply(s_matrix, bb)
#print("term1 ", term1)
#print("term2 ", term2)
#print("\n _____ \n")
v = term1 + term2
#print("v ",v)
return v
```

Not used

## 1. qu\_eigen\_table\_create\_fn #fn cr\_eigen\_table(data\_base)