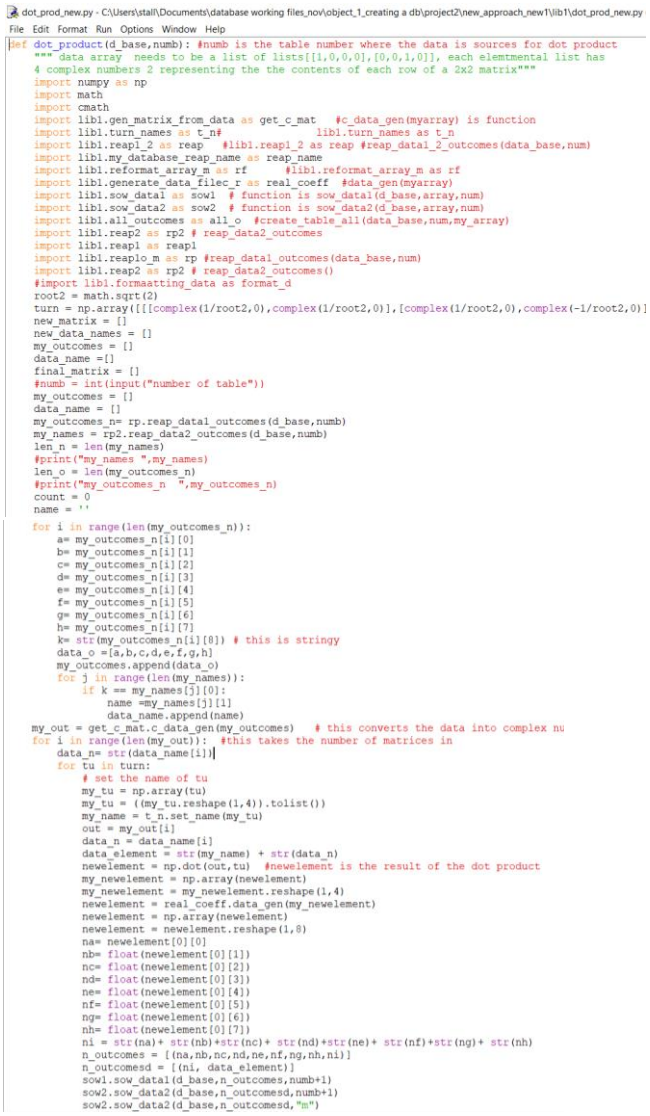
module dot_prod_new

It runs the function *dot_product(databasename,i*) where i represents the outcomes table number



```
dot_prod_new.py - C:\Users\stall\Documents\database working files_nov\object_1_creating a db\project2\new_approach_new1\lib1\dot_prod_new.py
File  Edit  Format  Run  Options  Window  Help
def dot_product(d_base,numb): #numb is the table number where the data is sources for dot product
    """ data array  needs to be a list of lists[[1,0,0,0],[0,0,1,0]], each elemtmental list has
    4 complex numbers 2 representing the the contents of each row of a 2x2 matrix"""
    import numpy as np
    import math
    import cmath
    import lib1.gen_matrix_from_data as get_c_mat   #c_data_gen(myarray) is function
    import lib1.turn_names as t_n#                  lib1.turn_names as t_n
    import lib1.reap1_2 as reap   #lib1.reap1_2 as reap #reap_data1_2_outcomes(data_base,num)
    import lib1.my_database_reap_name as reap_name
    import lib1.reformat_array_m as rf      #lib1.reformat_array_m as rf
    import lib1.generate_data_filec_r as real_coeff  #data_gen(myarray)
    import lib1.sow_data1 as sow1  # function is sow_data1(d_base,array,num)
    import lib1.sow_data2 as sow2  # function is sow_data2(d_base,array,num)
    import lib1.all_outcomes as all_o  #create_table_all(data_base,num,my_array)
    import lib1.reap2 as rp2 # reap_data2_outcomes
    import lib1.reap1 as reap1
    import lib1.reap1o_m as rp #reap_data1_outcomes(data_base,num)
    import lib1.reap2 as rp2 # reap_data2_outcomes()
    #import lib1.formaatting_data as format_d
    root2 = math.sqrt(2)
    turn = np.array([[[complex(1/root2,0),complex(1/root2,0)],[complex(1/root2,0),complex(-1/root2,0)]
    new_matrix = []
    new_data_names = []
    my_outcomes = []
    data_name =[]
    final_matrix = []
    #numb = int(input("number of table"))
    my_outcomes = []
    data_name = []
    my_outcomes_n= rp.reap_data1_outcomes(d_base,numb)
    my_names = rp2.reap_data2_outcomes(d_base,numb)
    len_n = len(my_names)
    #print("my_names ",my_names)
    len_o = len(my_outcomes_n)
    #print("my_outcomes_n  ",my_outcomes_n)
    count = 0
    name = ''
    for i in range(len(my_outcomes_n)):
        a= my_outcomes_n[i][0]
        b= my_outcomes_n[i][1]
        c= my_outcomes_n[i][2]
        d= my_outcomes_n[i][3]
        e= my_outcomes_n[i][4]
        f= my_outcomes_n[i][5]
        g= my_outcomes_n[i][6]
        h= my_outcomes_n[i][7]
        k= str(my_outcomes_n[i][8]) # this is stringy
        data_o =[a,b,c,d,e,f,g,h]
        my_outcomes.append(data_o)
        for j in range(len(my_names)):
            if k == my_names[j][0]:
                name =my_names[j][1]
                data_name.append(name)
    my_out = get_c_mat.c_data_gen(my_outcomes)   # this converts the data into complex nu
    for i in range(len(my_out)):  #this takes the number of matrices in
        data_n= str(data_name[i])
        for tu in turn:
            # set the name of tu
            my_tu = np.array(tu)
            my_tu = ((my_tu.reshape(1,4)).tolist())
            my_name = t_n.set_name(my_tu)
            out = my_out[i]
            data_n = data_name[i]
            data_element = str(my_name) + str(data_n)
            newelement = np.dot(out,tu)  #newelement is the result of the dot product
            my_newelement = np.array(newelement)
            my_newelement = my_newelement.reshape(1,4)
            newelement = real_coeff.data_gen(my_newelement)
            newelement = np.array(newelement)
            newelement = newelement.reshape(1,8)
            na= newelement[0][0]
            nb= float(newelement[0][1])
            nc= float(newelement[0][2])
            nd= float(newelement[0][3])
            ne= float(newelement[0][4])
            nf= float(newelement[0][5])
            ng= float(newelement[0][6])
            nh= float(newelement[0][7])
            ni = str(na)+ str(nb)+str(nc)+ str(nd)+str(ne)+ str(nf)+str(ng)+ str(nh)
            n_outcomes = [(na,nb,nc,nd,ne,nf,ng,nh,ni)]
            n_outcomesd = [(ni, data_element)]
            sow1.sow_data1(d_base,n_outcomes,numb+1)
            sow2.sow_data2(d_base,n_outcomesd,numb+1)
            sow2.sow_data2(d_base,n_outcomesd,"m")
```

1. gen_matrix_from_data

   This has function *c_data_gen(myarray)* this will take the list of nx 1x8 lists and reconstruct the complex numbers to output a list of 1x4 matrices which can be used in dot product module;

   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: 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]]])

2. turn_names

   This has function *set_name(my_array)*. It returns a string called name which is a text character h,s or t depending if the array matches the array elements of the h,s or t matrix.

3. generate_data_filec_r

   This function will take a list of 1x4 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 2x2 matrix, e.g. :

   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)]]. It uses the function *data_gen(myarray):* This is same as module used in Project 1.

4. sow_data1

   This has function is sow_data1(d_base,array,num) matrix will be a 1x 9 list of [('00001', '00001', '00001', '1','00004', '2', '3', '2','n')]. It runs the sql query:

   "INSERT OR IGNORE INTO outcomes1 VALUES(?,?,?,?,?,?,?,?,?);",my_matrix

5. sow_data2

   This has function is *sow_data2(data_base,my_list,num)*. It runs the sql query

   "INSERT OR IGNORE INTO outcomes1d VALUES(?,?);",my_list

   my_list will be a list of 1x2 elements

6. reap1o_m

   has function *reap_data1_outcomes(data_base,num).* It runs an sql query SELECT * FROM outcomesnum;

7. reap2

   This has function *reap_data2_outcomes(data_base,num):* returns a str array. It runs the sql query:

   "SELECT stringy, matrices FROM outcomes1d;"

Not used

1. my_database_reap_name

   This has function reap_data_name(data_base,num). It runs the sql query SELECT Matrix_name FROM tablename

2. reap1_2

   This has function *reap_data1_2_outcomes(data_base,num)*. It gets the data out of outcomes and outcomes d tables. It returns a mixed array, i.e. data from two tables. For the table specified by the num value it runs the sql query: SELECT * FROM outcomes1d LEFT JOIN outcomes1 ON outcomes1d.stringy = outcomes1.stringy

   this has not been used

3. reformat_array_m

   has been described in project 1