|  |  |  |  |
| --- | --- | --- | --- |
| **paf_kiet_logo** | **COLLEGE OF COMPUTING AND INFORMATION SCIENCES** | | |
| **Final Assessment of Lab Exam (Summer 2021 Semester)** | | |
| **Class Id** | 107285 | **Course Title** | Numerical Computing LAB |
| **Program** | BSCS | **Campus / Shift** | MAIN MORNING |
| **Date** | 16-07-2021 | **Total Marks** | 20 |
| **Duration** | 2.5 hours | **Faculty Name** | Aziz Mehmoud Farooqi |
| **Student Id** | 10619 | **Student Name** | Muhammad Umar Khan |
| **Code** | A1 |  |  |

**Instructions:**

* Fill out your Student ID and Student Name in above header.
* Do not remove or change any part question paper.
* Write down your answers with title “Answer for Question# 00”.
* Handwritten text or image should be on A4 size page with clear visibility of contents.
* In case of CHEATING, COPIED material or any unfair means would result in negative marking or ZERO.
* **Caution:** Duration to perform Final Assessment is **02 hours & 30 mins only**. **If you failed to upload answer sheet on LMS (in PDF format) within 2.5 hours limit, you would be considered as ABSENT/FAILED.**



**Instructions:** Attempt all two questions.

**[10 Marks]**

**Question 1:**

Write a Dynamic Python code for solving any set of equations problem (3 unknown variables or 4 etc.) using **Gaussian Elimination Method**.

You have to define two returnable functions named ***funForwardEDynamically***(a) and ***funBackSubDynamically***(a) that takes only one array matrix and returns an another array.

After creating the functions, first you will call the ***funForwardEDynamically*** then ***funBackSubDynamically*** in the main function...

Solve the following two set of equations by Gaussian elimination one by one:

*3 variables & 3 equations:*

-x -5y -5z = 2

4x -5y +4z = 19

x +5y -z = -20

*4 variables & 4 equations:*

x + z +2w = 6

y -2z = -3

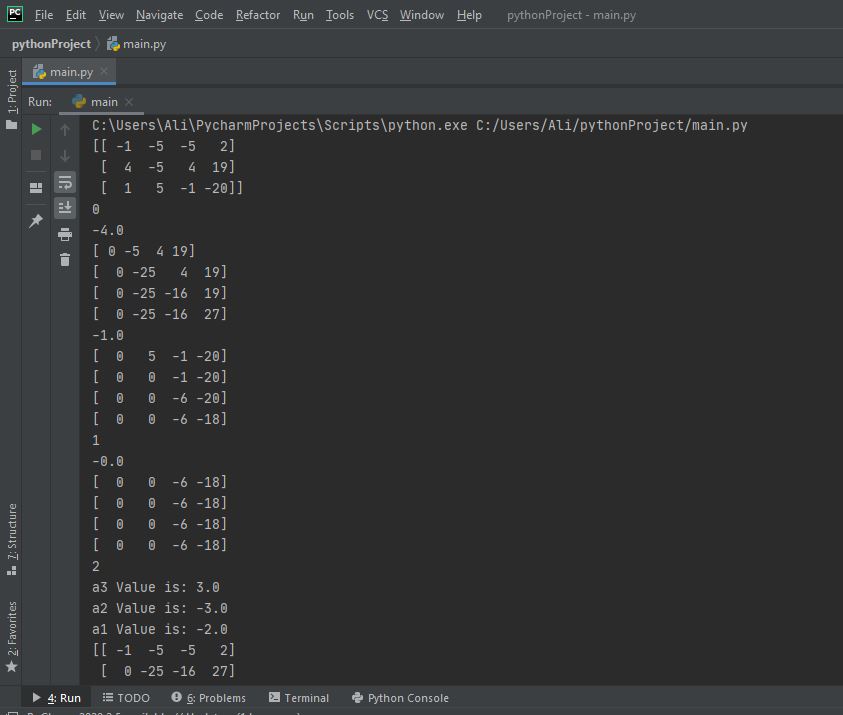
x +2y - z = -2

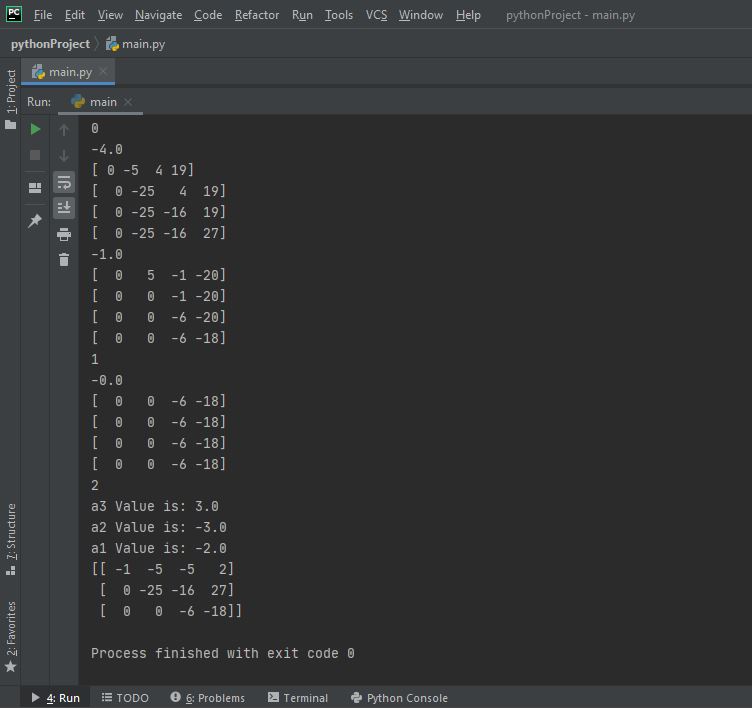
2x + y +3z -2w = 0

**CODE: (3 VARIABLES 3 EQUATIONS):**

def funDynamic(a):  
 n = len(a)  
 c = np.zeros(n)  
 for i in range(n):  
 for j in range(i + 1, n):  
 ratioIJ = a[j][i] / a[i][i]  
 for k in range(n + 1):  
 a[j][k] = a[j][k] - ((ratioIJ) \* a[i][k])  
 return a  
  
  
def BackDub(a):  
 n = len(a) - 1  
 kvalues = np.zeros(n + 1)  
 for i in range(n, -1, -1):  
 constant = a[i][n + 1]  
 if (i == n):  
 kvalues[i] = constant / a[i][i]  
 else:  
 mid = 0  
 for j in range(n, -1, -1):  
 if (a[i][j] != 0. and a[i][j] != a[j][j]):  
 a\_kvalues = kvalues[j]  
 a\_cellvalues = a[i][j] \* (-1)  
 mid = mid + (a\_kvalues \* a\_cellvalues)  
 kvalues[i] = (constant + mid) / a[i][i]  
 return kvalues  
  
  
import numpy as np  
  
  
def funForwardEDynamically(a):  
 a = np.array([[-1, -5, -5, 2], [4, -5, 4, 19], [1, 5, -1, -20]])  
 return a  
  
  
a = np.array([[-1, -5, -5, 2], [4, -5, 4, 19], [1, 5, -1, -20]])  
print(a)  
n = 3  
for i in range(n):  
 print(i)  
 for j in range(i + 1, n):  
 ratioIJ = a[j][i] / a[i][i]  
 print(ratioIJ)  
 for k in range(n + 1):  
 a[j][k] = a[j][k] - ((ratioIJ) \* a[i][k])  
 print(a[j])  
  
a3 = a[2][3] / a[2][2]  
  
a2 = ((a[1][3]) + (-a[1][2] \* a3)) / (a[1][1])  
  
a1 = ((a[0][3]) + (-a[0][2] \* a3) + (-a[0][1] \* a2)) / a[0][0]  
  
print('a3 Value is:', a3)  
  
print('a2 Value is:', a2)  
  
print('a1 Value is:', a1)  
  
import numpy as np  
  
  
equationTHREE = np.array([[-1,-5,-5,2],[4,-5,4,19],[1,5,-1,-20]])  
  
  
FORWARD = funDynamic(equationTHREE)  
  
  
print(FORWARD)  
backword = BackDub(FORWARD)  
res = backword  
res

**OUTPUT:**

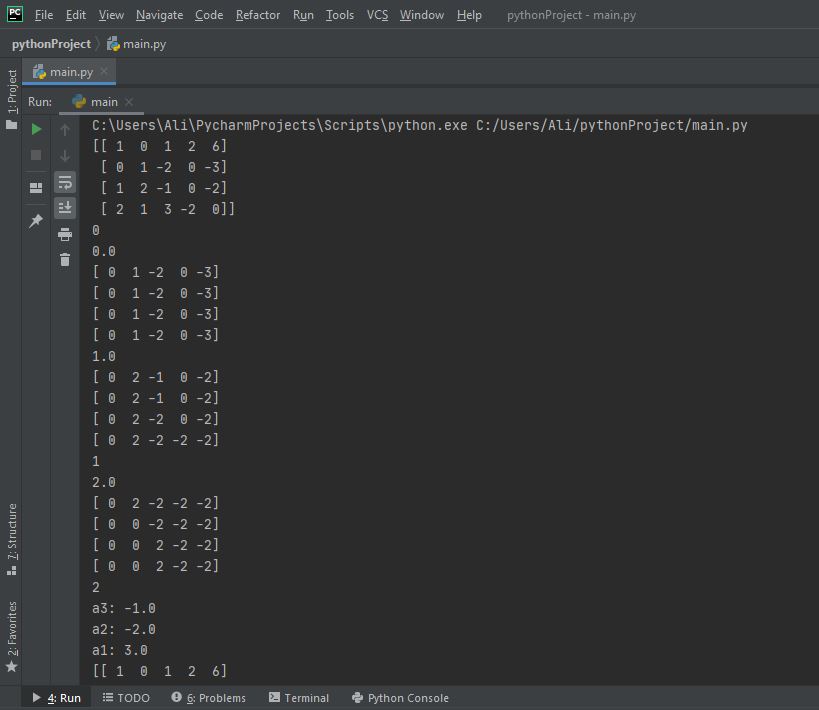
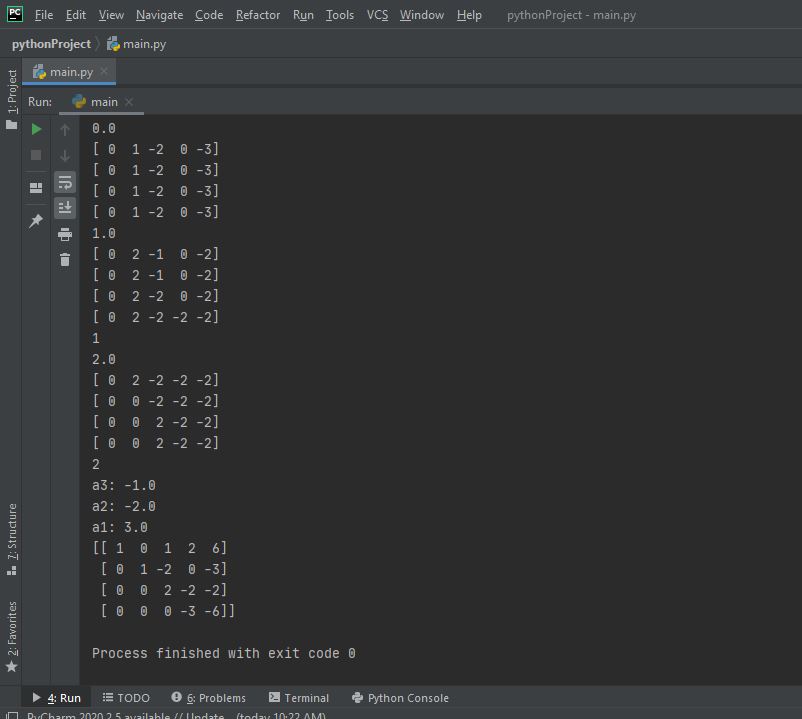




**4X4 CODE:**

import numpy as np  
  
  
def funDynamic(a):  
 n = len(a)  
 c = np.zeros(n)  
 for i in range(n):  
 for j in range(i + 1, n):  
 ratio = a[j][i] / a[i][i]  
 for k in range(n + 1):  
 a[j][k] = a[j][k] - ((ratio) \* a[i][k])  
 return a  
  
  
def BackDub(a):  
 n = len(a) - 1  
 kvalues = np.zeros(n + 1)  
 for i in range(n, -1, -1):  
 cons = a[i][n + 1]  
 if (i == n):  
 kvalues[i] = cons / a[i][i]  
 else:  
 mid = 0  
 for j in range(n, -1, -1):  
 if (a[i][j] != 0. and a[i][j] != a[j][j]):  
 a\_kvalues = kvalues[j]  
 a\_cellvalues = a[i][j] \* (-1)  
 mid = mid + (a\_kvalues \* a\_cellvalues)  
 kvalues[i] = (cons + mid) / a[i][i]  
 return kvalues  
  
  
import numpy as np  
  
  
def funForwardEDynamically(a):  
 a = np.array([[1, 0, 1, 2, 6], [0, 1, -2, 0, -3], [1, 2, -1, 0, -2], [2, 1, 3, -2, 0]])  
 return a  
  
  
a = np.array([[1, 0, 1, 2, 6], [0, 1, -2, 0, -3], [1, 2, -1, 0, -2], [2, 1, 3, -2, 0]])  
print(a)  
n = 3  
for i in range(n):  
 print(i)  
 for j in range(i + 1, n):  
 ratio = a[j][i] / a[i][i]  
 print(ratio)  
 for k in range(n + 1):  
 a[j][k] = a[j][k] - ((ratio) \* a[i][k])  
 print(a[j])  
  
a3 = a[2][3] / a[2][2]  
  
a2 = ((a[1][3]) + (-a[1][2] \* a3)) / (a[1][1])  
  
a1 = ((a[0][3]) + (-a[0][2] \* a3) + (-a[0][1] \* a2)) / a[0][0]  
  
print('a3:', a3)  
  
print('a2:', a2)  
  
print('a1:', a1)  
import numpy as np  
  
  
eq3 = np.array([[1,0,1,2,6],[0,1,-2,0,-3],[1,2,-1,0,-2],[2,1,3,-2,0]])  
  
  
forword = funDynamic(eq3)  
  
  
print(forword)  
backword = BackDub(forword)  
res = backword  
res

**OUTPUT:**

****

**[10 Marks]**

**Question 2:**

Write a Static Python code for solving the particular set of equation problem (4 unknown variables and 4 equations) using **Jacobi Method**.

Solve the following set of equations by Jacobi:

*4 variables & 4 equations:*

10x - y + 2z = 6

- x +11y - z +3w = 25

2x - y +10z - w = -11

+ 3y - z +8w = 15

**CODE:**

import numpy as np  
  
  
Main\_Matrix = [[10, -1, 2, 0, 6], [-1, 11, -1, 3, 25], [2, -1, 10, -1, -11], [0, 3, -1, 8, 15]]  
GM = np.array(Main\_Matrix, dtype=float)  
Matrix = np.array(Main\_Matrix, dtype=float)  
Ta = []  
Ta = [row[-1] for row in Matrix]  
T = np.array(Ta, dtype=float)  
matr = np.delete(Matrix, -1, axis=1)  
print(matr)  
print(T)  
  
  
def JacobiMethod(x0, y0, z0, w0, e):  
 count = 0  
 condition = True  
 eq1 = lambda x, y, z, w: (T[0] + matr[0][1] \* y - matr[0][2] \* z - matr[0][3] \* w) / matr[0][0]  
 eq2 = lambda x, y, z, w: (T[1] + matr[0][1] \* y - matr[0][2] \* z - matr[1][3] \* w) / matr[1][1]  
 eq3 = lambda x, y, z, w: (T[2] + matr[0][1] \* y - matr[0][2] \* z - matr[2][3] \* w) / matr[2][2]  
 eq4 = lambda x, y, z, w: (T[3] + matr[0][1] \* y - matr[0][2] \* z - matr[3][2] \* w) / matr[3][3]  
  
 while (condition):  
 x1 = eq1(x0, y0, z0, w0)  
 y1 = eq2(x0, y0, z0, w0)  
 z1 = eq3(x0, y0, z0, w0)  
 w1 = eq4(x0, y0, z0, w0)  
 print('%d\t%0.6f\t%0.6f\t%0.6f\n\t%0.6f\n' % (count, x1, y1, z1, w1))  
 e1 = abs(x0 - x1)  
 e2 = abs(y0 - y1)  
 e3 = abs(z0 - z1)  
 e4 = abs(w0 - w1)  
 count += 1  
 x0 = x1  
 y0 = y1  
 z0 = z1  
 w0 = w1  
 condition = e1 > e and e2 > e and e3 > e and e4 > e  
  
 print('\nSolution is: x=%0.6f, y=%0.6f and z = %0.6f\n w = %0.6f\n' % (x1, y1, z1, w1))  
  
  
def VC():  
 if (abs(GM[0][0]) > abs(GM[0][1]) + abs(GM[0][2]) + abs(GM[0][3]) and abs(GM[1][1]) > abs(  
 GM[1][0]) + abs(GM[1][2]) + abs(GM[1][3]) and abs(GM[3][3]) > abs(GM[3][1]) + abs(  
 GM[3][0]) + abs(GM[3][2])):  
 JacobiMethod(0, 0, 0, 0, 0.000001)  
 else:  
 print("These are Incorrect Values")  
  
  
VC()

OUTPUT:

