

Improvement of Correlation amongst elliptic points in Elliptic Curve Cryptography based on Metaheuristic algorithm and Implementation on Python

Md Asim Kamal Farooqui (M.Tech student) at Center of Development of Advanced Computing ,Noida

Abstract:-Elliptic curve cryptography has paved the way for making a communication system robust and highly secure with extremely small key size. It offers equivalent safety in contrast to RSA with only short key size. The present project focuses on amalgamating the ECC (Elliptic curve cryptography) with a global search metaheuristic named PSO (Particle Swarm Optimization) and then calculating the ECC curve points. Further the correlation among the points is calculated and is compared with other correlation work involving encryption without PSO. As the correlation factor tends to minus one value, a sense of randomization in key is induced which makes the system less prone to any adversary's attack.

Introduction

1.1 Overview

Elliptic curve cryptography (ECC) give a quicker option in contrast to open key cryptography. Smaller key sizes are needed with ECC to give an ideal dimension of security, which implies more rapidly key trade, client validation, signature age and check, notwithstanding littler key stockpiling needs. The well-being of ECC has not been certified nevertheless rather it relies on the trouble of reckoning "Elliptic curve discrete logarithm Problem" in the elliptic curve gathering. Metaheuristics are structured as well as assume significant job by means of complex enhancement issues.

The present elliptic curve cryptographic approach is mugged up with a population-based metaheuristic algorithm named as Particle Swarm Optimization or simply PSO. PSO belongs to a class of stochastic algorithm. It was first developed by James Kennedy and Russel Eberhart in 1995. It is basically used in combinatorial problem optimization and is an iteration-based method. The method tries to remove some pitfalls in the the candidate solution in regard to some level of quality based on the context in which the method needs to be applied. In the present scenarios, the whole algorithm has been amalgamated with elliptic curve points generated and the main goal is to minimize the correlation coefficient amongst elliptic curve points by applying PSO on the elliptic curve equation treating the same as an objective function or fitness function. The entire approach has been implemented on Spyder (Python 3.7) as platform.

The entire approach of Particle Swarm Optimization can be depicted with the given steps: -

1. First step of Particle swarm optimization deals with initialisation of position, p_i of each particle as PSO algorithm keeps multiple solution of a problem at a time depending on the context
2. The PSO algorithm is an iterative approach. In each iteration, the solution of the problem is evaluated using an objective function which in turn gives its fitness value
3. The solutions are basically the particles in the fitness graph.
4. The particle move in the search space to determine the maximum or optimal value returned by the objective function

Each particle in PSO adheres to these parameters: -

1. Position
2. Velocity
3. Individual best position

Apart from these, the particle also maintains its global best in the record

In nutshell entire PSO algorithm can be summed up in three steps.

1. Calculate fitness of each particle
2. Update particle individual and global best.
3. Update particle velocity and position after each iteration.

These three steps keep on running until some stopping condition is achieved

Each particle's velocity is updated using this equation

$$v_i(t+1) = wv_i(t) + c_1r_1[x^*_i(t) - x_i(t)] + c_2r_2[g(t) - x_i(t)] \dots\dots\dots 1.1$$

Where, $v_i(t)$ is the velocity of particle at time t

$x_i(t)$ is the position of particle at time t

$x^*_i(t)$ is the particle individual best solution as of time t

$g(t)$ is best solution of swarm as of time t

i is the particle index

w is the inertial coefficient

c_1 and c_2 are the acceleration coefficient where $0 < c_1, c_2 < 2$

r_1 and r_2 are random values where, $0 < r_1, r_2 < 1$

Each particle position is updated using this equation

$$x_i(t+1) = x_i(t) + v_i(t+1) \dots\dots\dots 1.2$$

After applying Particle Swarm Optimization and seeing g_{best} and p_{best} on elliptic curve, a set of points is obtained as X values and Y values.

X values = $\{x_i\}$

Y values = $\{y_i\}$

The correlation between X values and Y values for each large prime number is computed using general Karl Pearson correlation coefficient formulae as given below: -

$$r = r_{xy} = \frac{Cov(x, y)}{S_x \times S_y}$$

OR

$$r = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sqrt{\sum(x - \bar{x})^2} \sqrt{\sum(y - \bar{y})^2}}$$

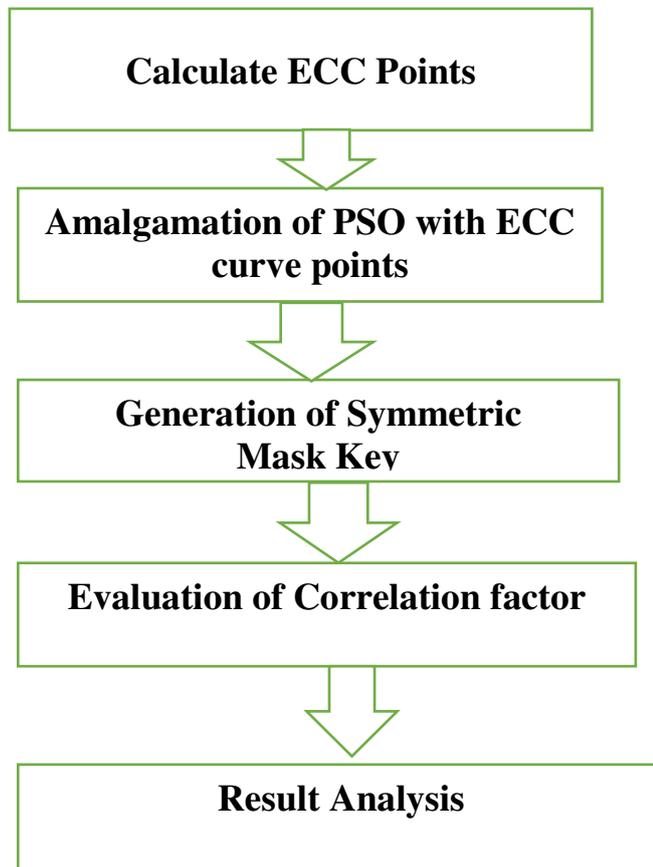
Where, \bar{x} = mean of X variable

\bar{y} = mean of Y variable

It is observed from the calculated correlation coefficient, r valued considering different prime numbers that a level of improvement has been made when ECC generated were clubbed with PSO algorithms in comparison to alone ECC points correlation coefficient. This is the major Autocorrelation test for checking randomness amongst the elliptic curve points. Lesser the correlation value, more will be the degree of randomness in the key and lesser it will susceptible of any adversaries' attack.

Flowchart

Proposed Approach is as under



STEPS OF PROPOSED APPROACH: -

1. First and foremost step of proposed approach focuses on calculation of elliptic curve points based on following cubic equation with different prime number set.

$$y^2=x^3+a.x+b \text{ mod } p \dots\dots\dots 3.1$$

Where a=1.b=1 and p= {251,457,593,929}

2.On generating the elliptic curve points, it is amalgamated with global search metaheuristic named Particle Swarm Optimization [11] Algorithm to induce a sense of improvement.

3.Third step generates a symmetric mask key which is used for encryption and decryption. Here symmetric term is a misnomer as the cryptographic algorithm is using the symmetric property of elliptic curve. The key is used for signing as well as verifying purpose.

4.. Fourth step deals with calculation of correlation [8] factors of generated elliptic points and points obtained after applying PSO (Particle Swarm Optimization)

5.Last step deals with result analysis check from by comparing the correlation values of elliptic points generated and the points obtained after PSO has been applied to them and further comparison of results with earlier work of genetic algorithm based DES,Data Encryption Standard Cryptosystem

IMPLEMENTATION

4.1 Minimum Hardware and Software prerequisites: -

Processor: Intel i3 2.10 GHz

Working System: Windows 10.

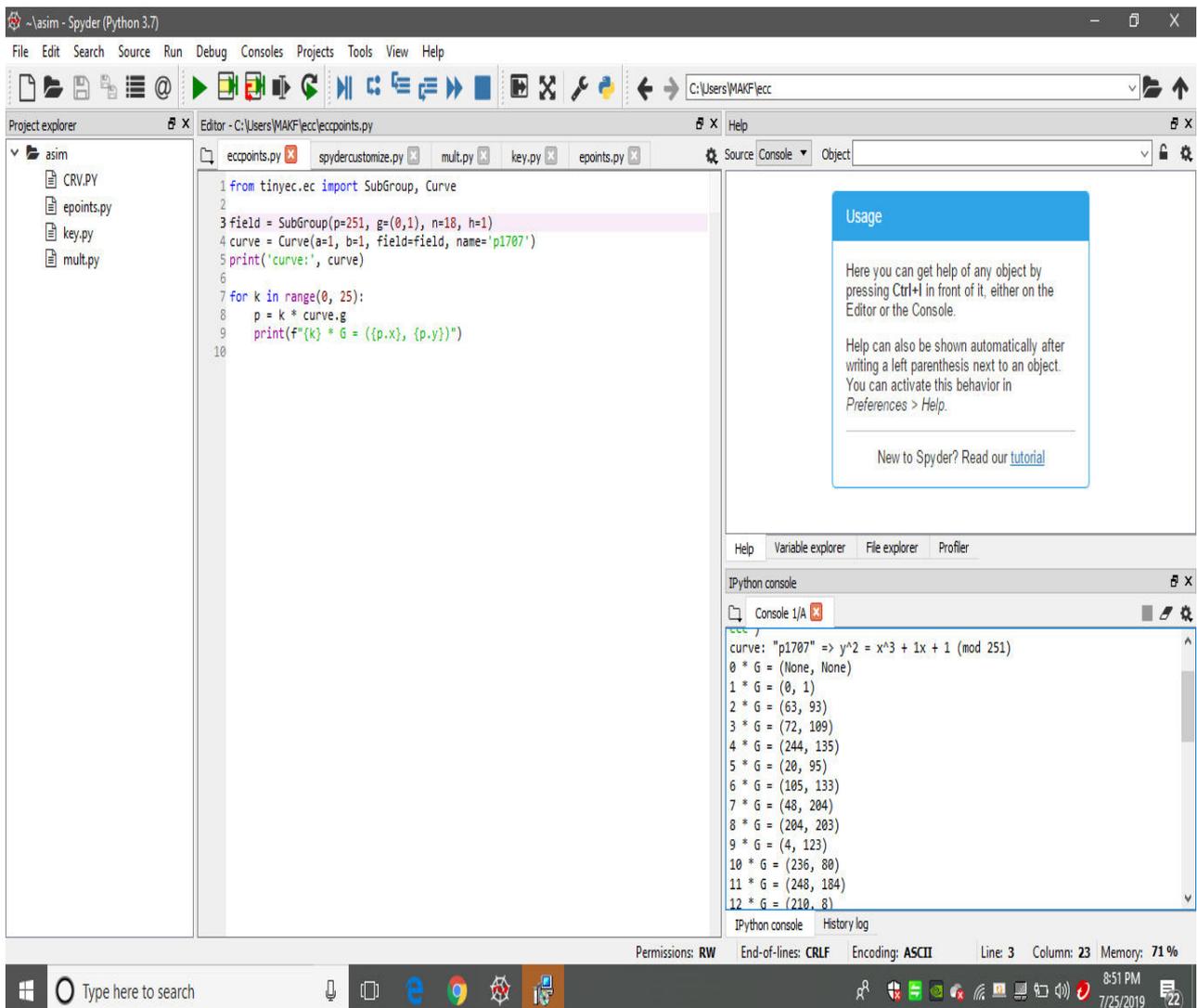
Hard Disk: 320 GB

RAM: 4 GB

Programming: Anaconda (Spyder) version 3.7

4.2 Python code for computing ECC Points with different prime numbers,Pon Spyder(Python 3.7)

a) P=251



```
1 from tinyec.ec import SubGroup, Curve
2
3 field = SubGroup(p=251, g=(0,1), n=18, h=1)
4 curve = Curve(a=1, b=1, field=field, name='p1707')
5 print('curve:', curve)
6
7 for k in range(0, 25):
8     p = k * curve.g
9     print(f"{k} * G = ({p.x}, {p.y})")
10
```

Usage

Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.

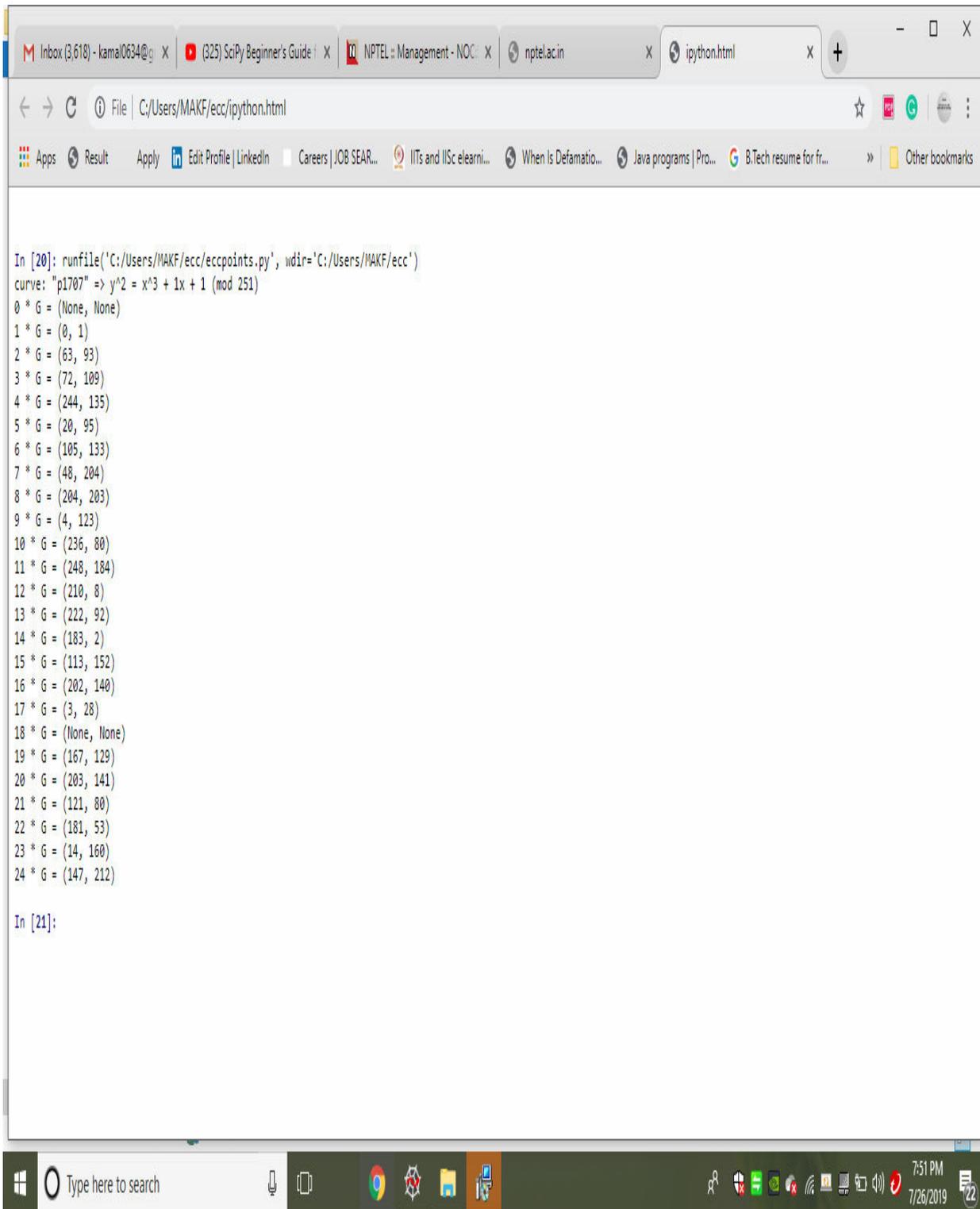
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.

New to Spyder? Read our [tutorial](#)

IPython console

```
curve: "p1707" => y^2 = x^3 + 1x + 1 (mod 251)
0 * G = (None, None)
1 * G = (0, 1)
2 * G = (63, 93)
3 * G = (72, 109)
4 * G = (244, 135)
5 * G = (20, 95)
6 * G = (105, 133)
7 * G = (48, 204)
8 * G = (204, 203)
9 * G = (4, 123)
10 * G = (236, 80)
11 * G = (248, 184)
12 * G = (210, 8)
```

Fig.4.1.1.1 Python code for computing ECC Points with prime numbers,P=251 on Spyder(Python 3.7)



```
In [20]: runfile('C:/Users/MAKF/ecc/eccpoints.py', wdir='C:/Users/MAKF/ecc')
curve: "p1707" => y^2 = x^3 + 1x + 1 (mod 251)
0 * G = (None, None)
1 * G = (0, 1)
2 * G = (63, 93)
3 * G = (72, 109)
4 * G = (244, 135)
5 * G = (20, 95)
6 * G = (105, 133)
7 * G = (48, 204)
8 * G = (204, 203)
9 * G = (4, 123)
10 * G = (236, 80)
11 * G = (248, 184)
12 * G = (210, 8)
13 * G = (222, 92)
14 * G = (183, 2)
15 * G = (113, 152)
16 * G = (202, 140)
17 * G = (3, 28)
18 * G = (None, None)
19 * G = (167, 129)
20 * G = (203, 141)
21 * G = (121, 80)
22 * G = (181, 53)
23 * G = (14, 160)
24 * G = (147, 212)

In [21]:
```

Fig. 4.1.1.2 ECC points output with P= 251

a) P=457

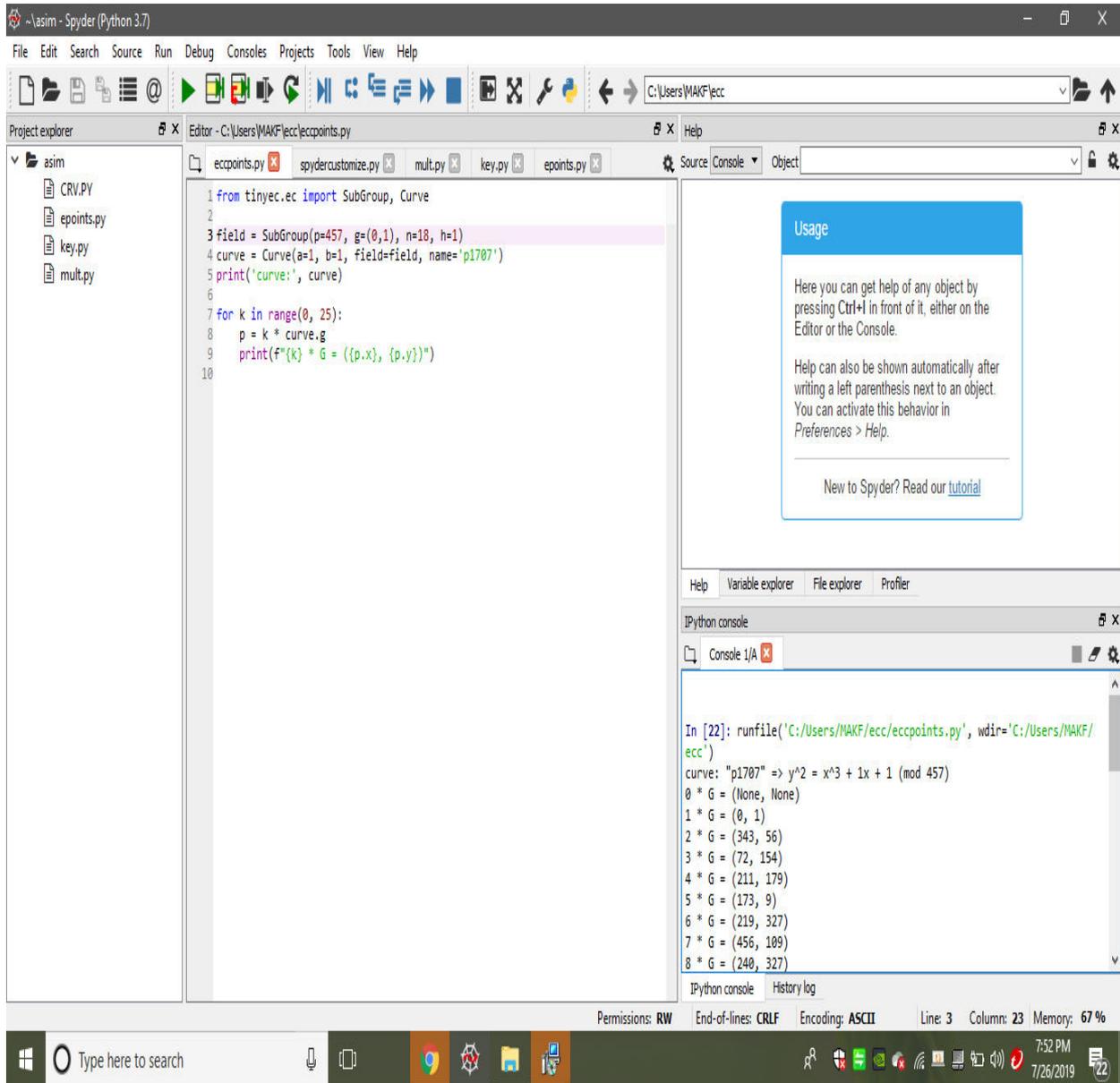
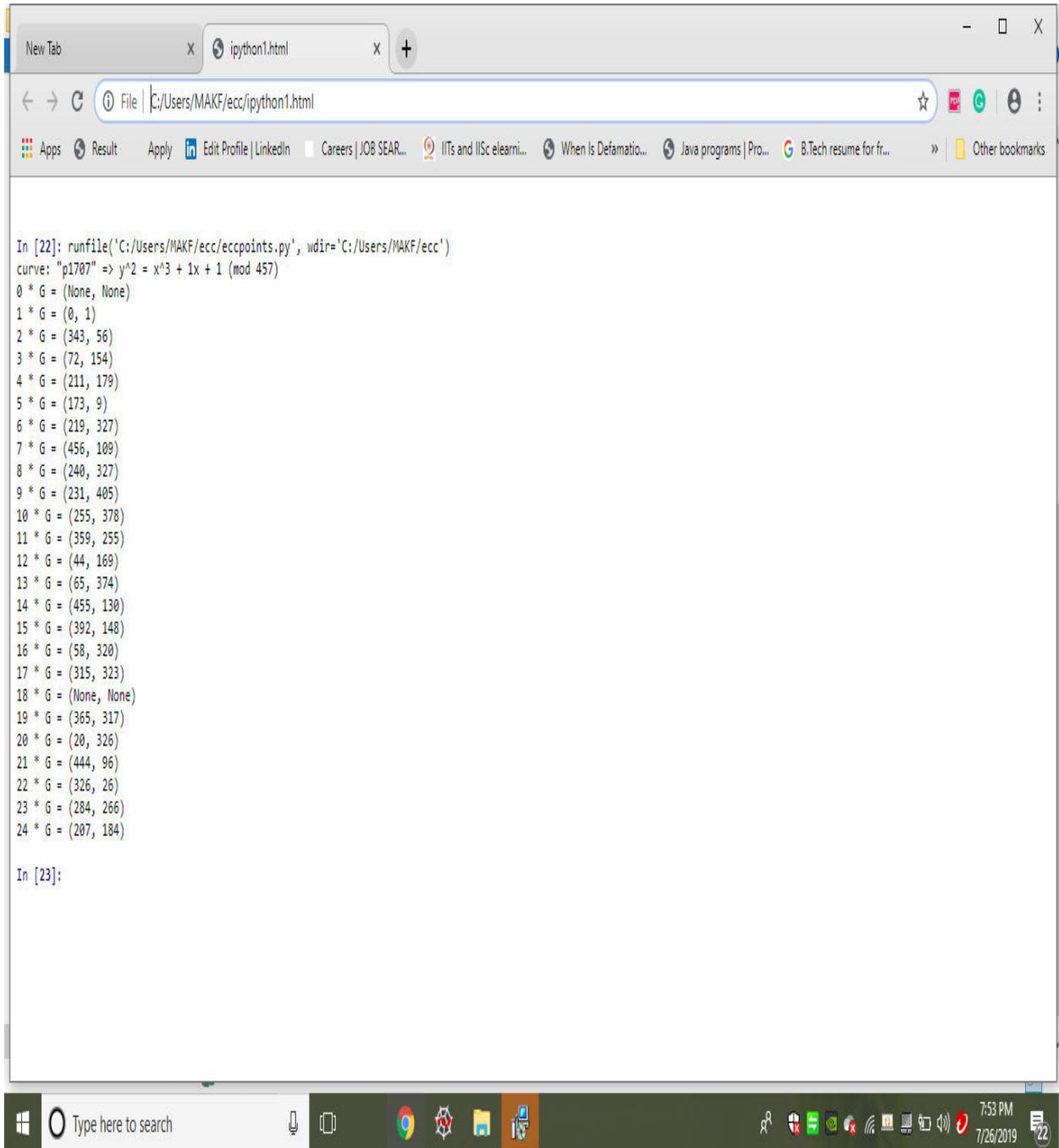


Fig.4.1.1.3 Python code for computing ECC Points with primenumber,P=457 on Spyder(Python 3.7)



```
In [22]: runfile('C:/Users/MAKF/ecc/eccpoints.py', wdir='C:/Users/MAKF/ecc')
curve: "p1707" => y^2 = x^3 + 1x + 1 (mod 457)
0 * G = (None, None)
1 * G = (0, 1)
2 * G = (343, 56)
3 * G = (72, 154)
4 * G = (211, 179)
5 * G = (173, 9)
6 * G = (219, 327)
7 * G = (456, 109)
8 * G = (240, 327)
9 * G = (231, 405)
10 * G = (255, 378)
11 * G = (359, 255)
12 * G = (44, 169)
13 * G = (65, 374)
14 * G = (455, 130)
15 * G = (392, 148)
16 * G = (58, 320)
17 * G = (315, 323)
18 * G = (None, None)
19 * G = (365, 317)
20 * G = (20, 326)
21 * G = (444, 96)
22 * G = (326, 26)
23 * G = (284, 266)
24 * G = (207, 184)

In [23]:
```

Fig. 4.1.1.4 ECC points output with P= 457

c) P=593

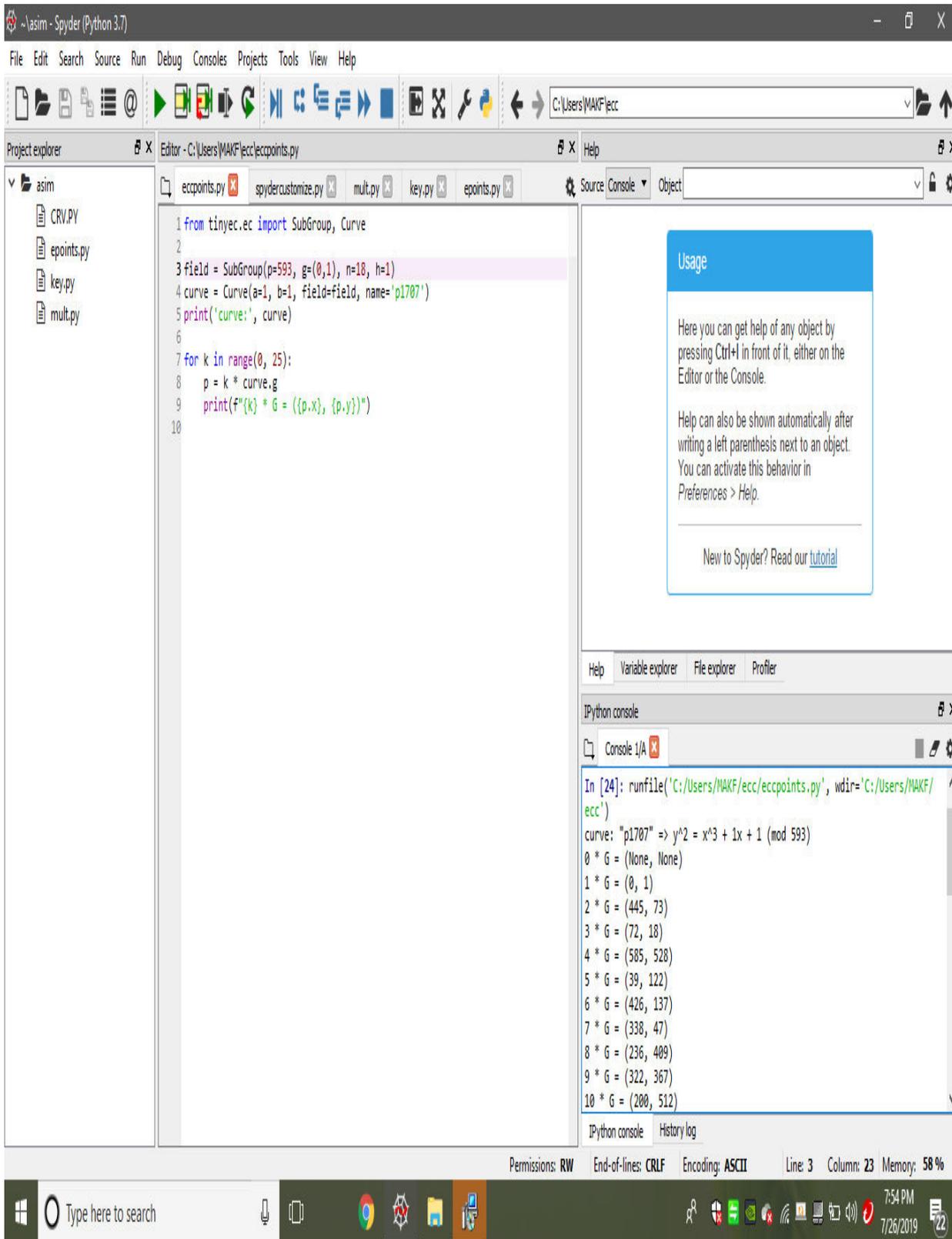
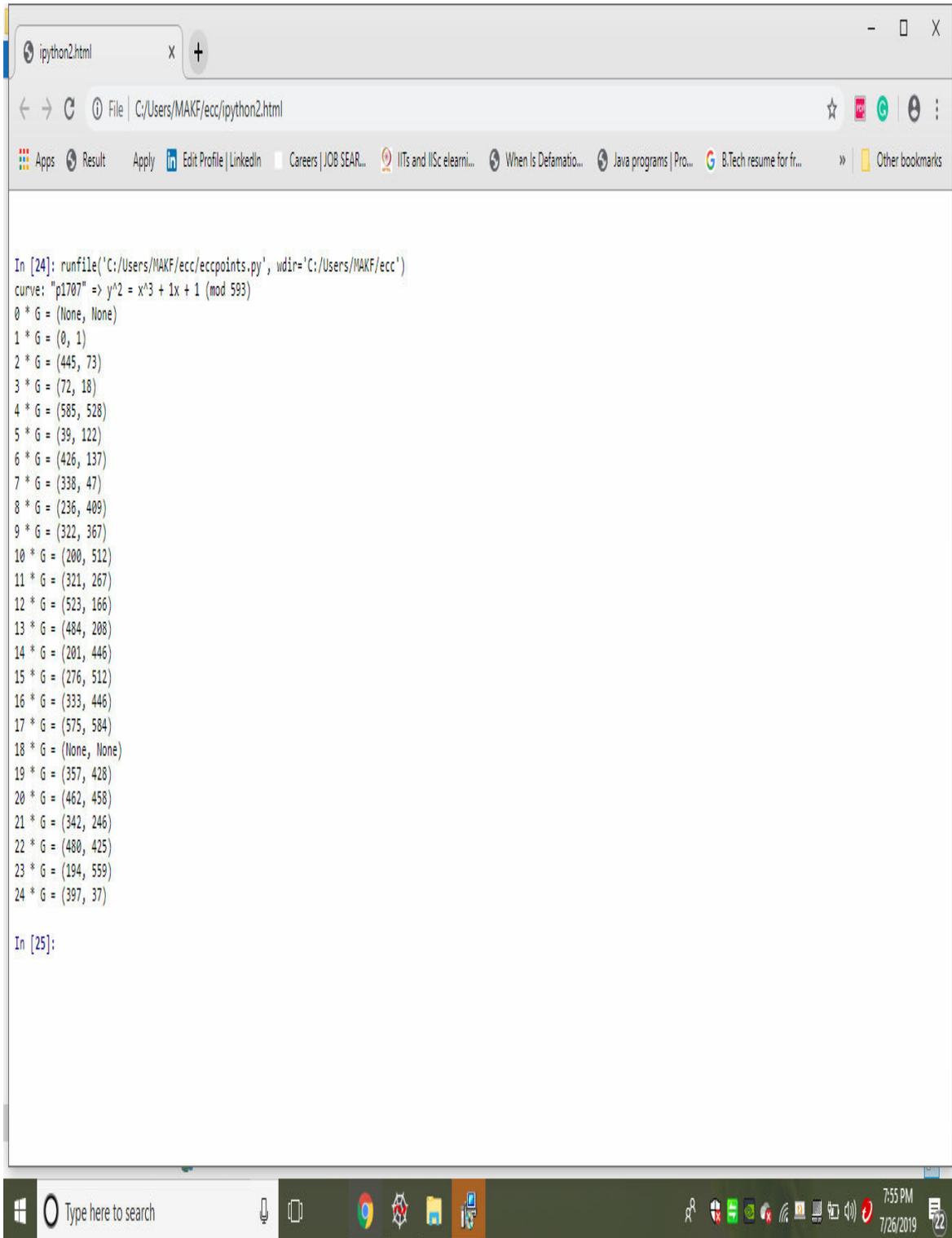


Fig.4.1.1.5 Python code for computing ECC Points with primenumber,P=593 on Spyder(Python 3.7)



```
In [24]: runfile('C:/Users/MAKF/ecc/eccpoints.py', wdir='C:/Users/MAKF/ecc')
curve: "p1707" => y^2 = x^3 + 1x + 1 (mod 593)
0 * G = (None, None)
1 * G = (0, 1)
2 * G = (445, 73)
3 * G = (72, 18)
4 * G = (585, 528)
5 * G = (39, 122)
6 * G = (426, 137)
7 * G = (338, 47)
8 * G = (236, 409)
9 * G = (322, 367)
10 * G = (200, 512)
11 * G = (321, 267)
12 * G = (523, 166)
13 * G = (484, 208)
14 * G = (201, 446)
15 * G = (276, 512)
16 * G = (333, 446)
17 * G = (575, 584)
18 * G = (None, None)
19 * G = (357, 428)
20 * G = (462, 458)
21 * G = (342, 246)
22 * G = (480, 425)
23 * G = (194, 559)
24 * G = (397, 37)

In [25]:
```

Fig. 4.1.1.6 ECC points output with P= 593

d) P=929

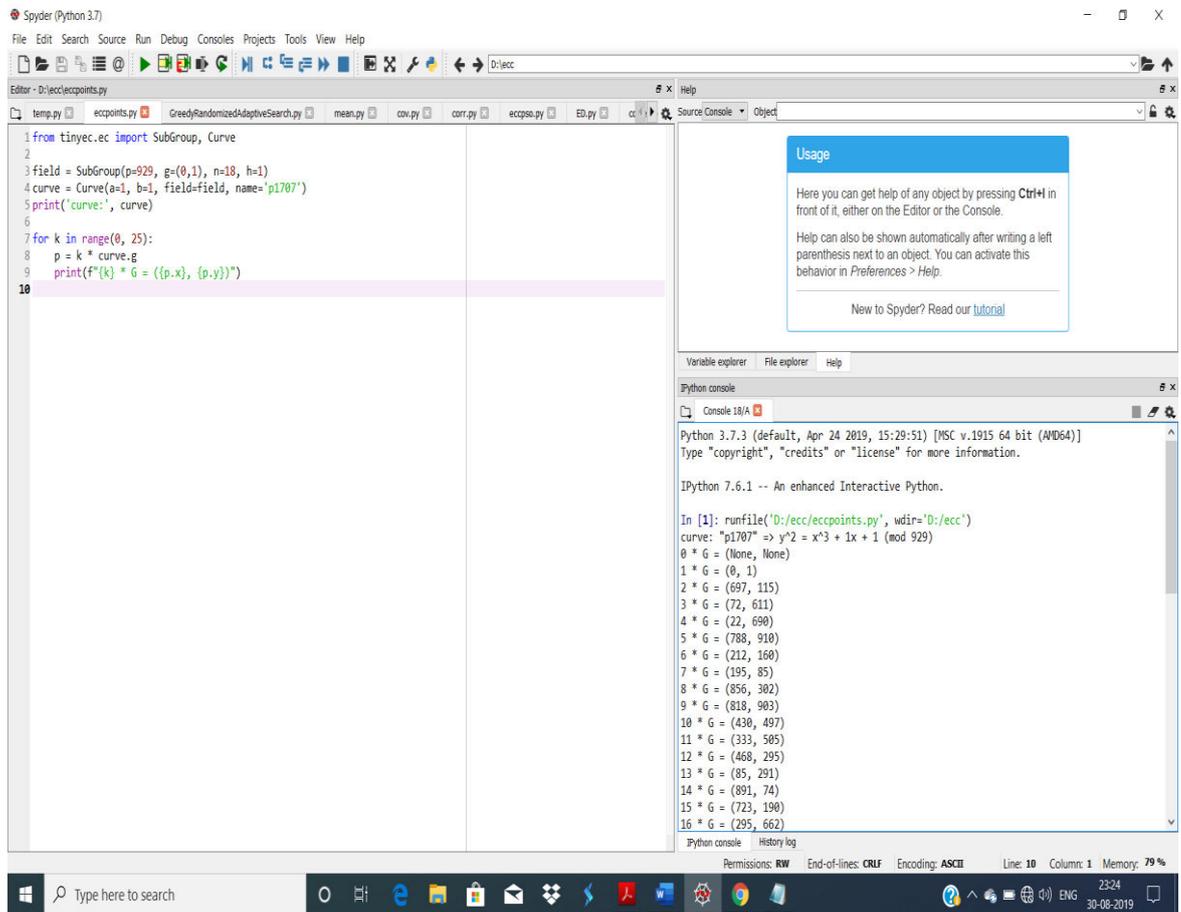
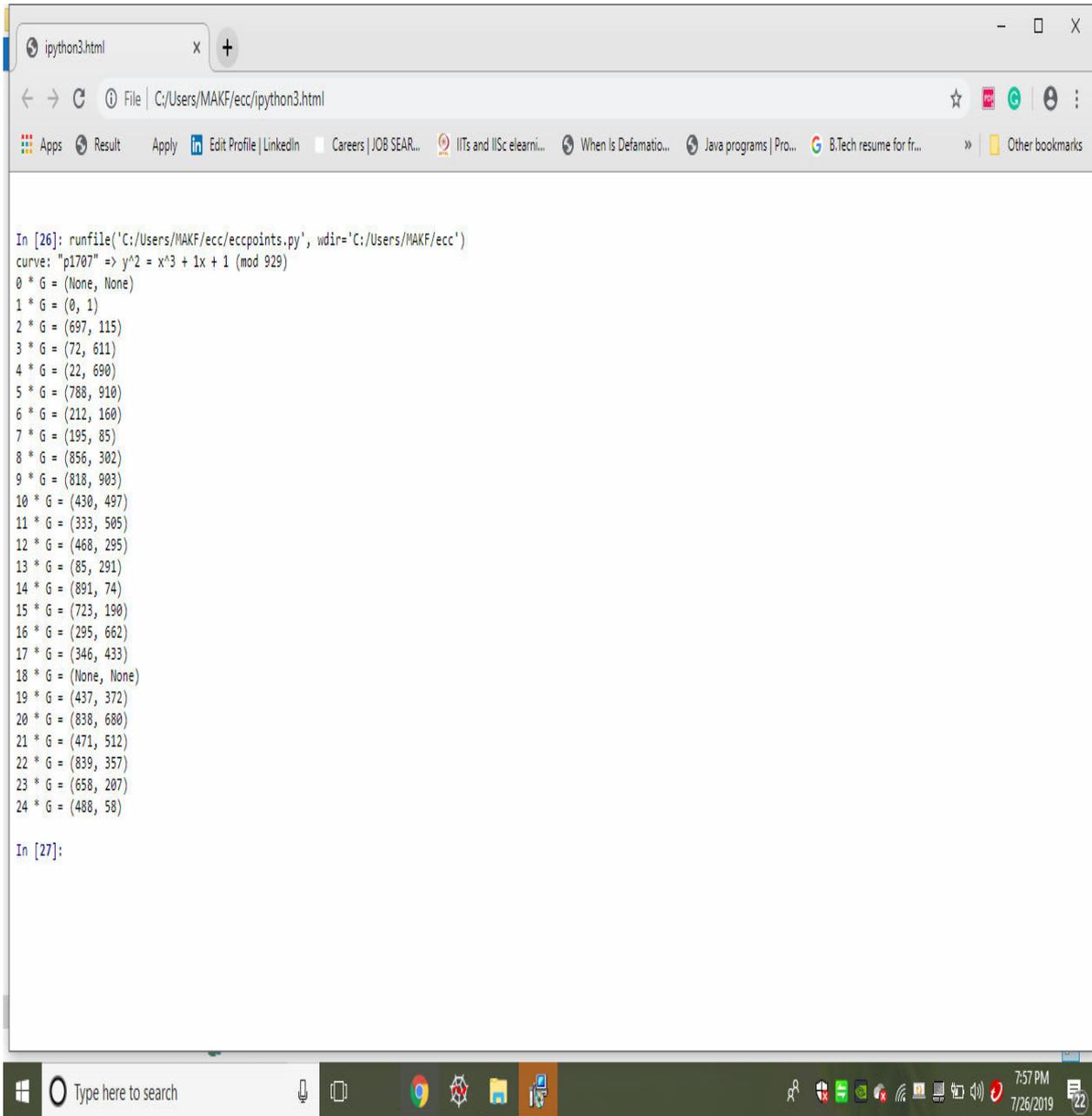


Fig.4.1.1.7 Python code for computing ECC Points with primenumber,P=929 on Spyder(Python 3.7)



```
In [26]: runfile('C:/Users/MAKF/ecc/eccpoints.py', wdir='C:/Users/MAKF/ecc')
curve: "p1707" => y^2 = x^3 + 1x + 1 (mod 929)
0 * G = (None, None)
1 * G = (0, 1)
2 * G = (697, 115)
3 * G = (72, 611)
4 * G = (22, 690)
5 * G = (788, 910)
6 * G = (212, 160)
7 * G = (195, 85)
8 * G = (856, 302)
9 * G = (818, 903)
10 * G = (430, 497)
11 * G = (333, 505)
12 * G = (468, 295)
13 * G = (85, 291)
14 * G = (891, 74)
15 * G = (723, 190)
16 * G = (295, 662)
17 * G = (346, 433)
18 * G = (None, None)
19 * G = (437, 372)
20 * G = (838, 680)
21 * G = (471, 512)
22 * G = (839, 357)
23 * G = (658, 207)
24 * G = (488, 58)

In [27]:
```

Fig.4.1.1.8 ECC points output with P= 929

4.3 Python code for amalgamation of ECC points with PSO in Spyder(Python 3.7)

a) P=251

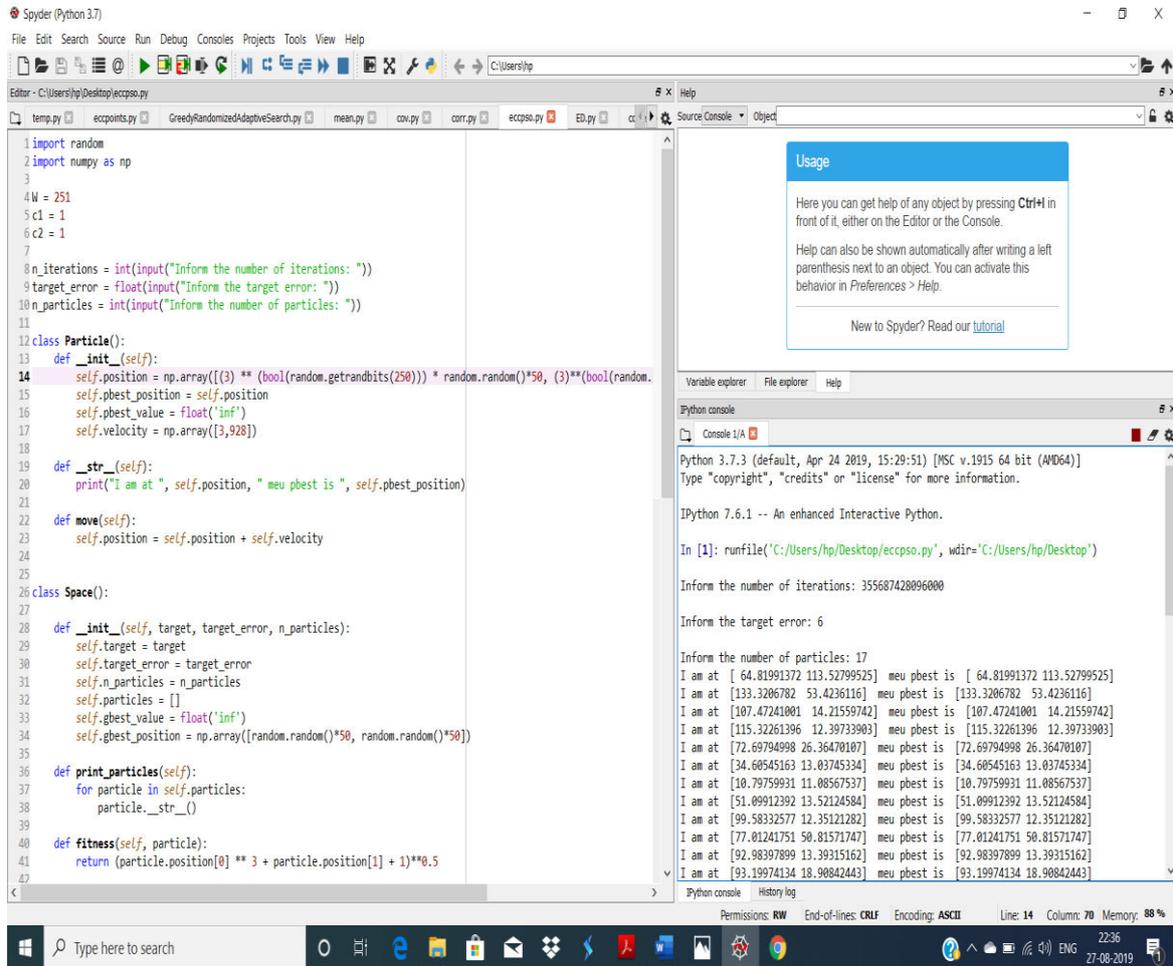


Fig.4.1.1.9 Python code for computing ECC Points with PSO with prime number,P=251 on Spyder(Python 3.7)

```
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

IPython 7.6.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/hp/Desktop/eccps0.py', wdir='C:/Users/hp/Desktop')

Inform the number of iterations: 355687428096000

Inform the target error: 6

Inform the number of particles: 17
i am at [ 64.81991372 113.52799525] meu pbest is [ 64.81991372 113.52799525]
i am at [133.3206782 53.4236116] meu pbest is [133.3206782 53.4236116]
i am at [107.47241001 14.21559742] meu pbest is [107.47241001 14.21559742]
i am at [115.32261396 12.39733903] meu pbest is [115.32261396 12.39733903]
i am at [72.69794998 26.36470107] meu pbest is [72.69794998 26.36470107]
i am at [34.60545163 13.03745334] meu pbest is [34.60545163 13.03745334]
i am at [10.79759931 11.08567537] meu pbest is [10.79759931 11.08567537]
i am at [51.09912392 13.52124584] meu pbest is [51.09912392 13.52124584]
i am at [99.58332577 12.35121282] meu pbest is [99.58332577 12.35121282]
i am at [77.01241751 50.81571747] meu pbest is [77.01241751 50.81571747]
i am at [92.98397899 13.39315162] meu pbest is [92.98397899 13.39315162]
i am at [93.19974134 18.90842443] meu pbest is [93.19974134 18.90842443]
i am at [142.84404672 56.59705039] meu pbest is [142.84404672 56.59705039]
i am at [ 65.78232477 126.65532477] meu pbest is [ 65.78232477 126.65532477]
i am at [ 2.10659649 143.25315481] meu pbest is [ 2.10659649 143.25315481]
i am at [105.6241311 39.42455596] meu pbest is [105.6241311 39.42455596]
i am at [84.35197363 26.14328493] meu pbest is [84.35197363 26.14328493]
C:/Users/hp/Desktop/eccps0.py:41: RuntimeWarning: overflow encountered in double_scalars
  return (particle.position[0]**3 + particle.position[1]**0.5)
C:/Users/hp/Desktop/eccps0.py:61: RuntimeWarning: overflow encountered in multiply
  new_velocity = (w*particle.velocity) + (c1*random.random()) * (particle.pbest_position - particle.position) + \
C:/Users/hp/Desktop/eccps0.py:61: RuntimeWarning: invalid value encountered in add
  new_velocity = (w*particle.velocity) + (c1*random.random()) * (particle.pbest_position - particle.position) + \
```

Fig.4.1.2.0 PSO output for P=251

b)P=457

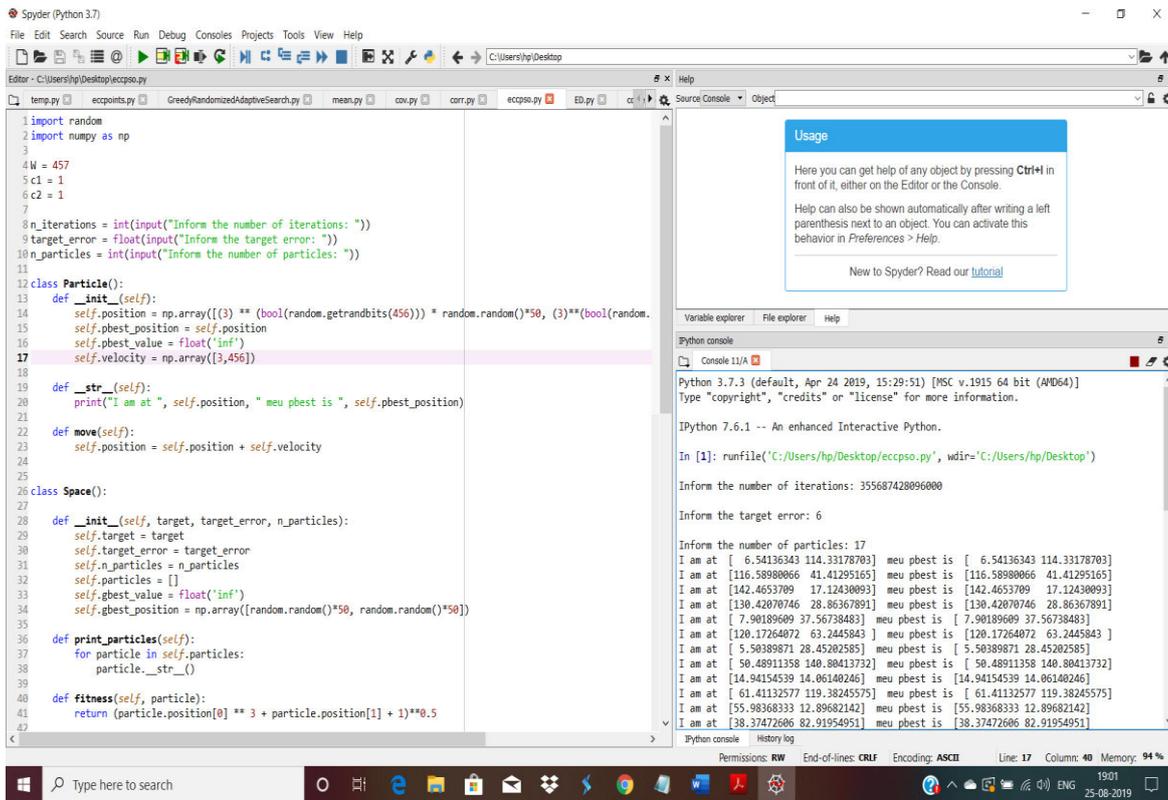


Fig.4.1.2.1 Python code for computing ECC Points with PSO with prime number,P=457 on Spyder(Python 3.7)

```
MTEch | Project | Algebra | 192.168 | Sci-Hub | 192.168 | Practica | anh.cs.l | 17 | 251.htrn | 457.htrn | 457.htrn | + | - | [ ] | X
File | D:\metaheuristicps/457.html
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

Python 7.6.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/hp/Desktop/eccpso.py', wdir='C:/Users/hp/Desktop')

Inform the number of iterations: 355687428096000

Inform the target error: 6

Inform the number of particles: 17
I am at [ 6.54136343 114.33178703] meu pbest is [ 6.54136343 114.33178703]
I am at [116.58980066 41.41295165] meu pbest is [116.58980066 41.41295165]
I am at [142.4653709 17.12430093] meu pbest is [142.4653709 17.12430093]
I am at [130.42070746 28.86367891] meu pbest is [130.42070746 28.86367891]
I am at [ 7.90189609 37.56738483] meu pbest is [ 7.90189609 37.56738483]
I am at [120.17264072 63.2445843 ] meu pbest is [120.17264072 63.2445843 ]
I am at [ 5.50389871 28.45202585] meu pbest is [ 5.50389871 28.45202585]
I am at [ 50.48911358 140.80413732] meu pbest is [ 50.48911358 140.80413732]
I am at [14.94154539 14.06140246] meu pbest is [14.94154539 14.06140246]
I am at [ 61.41132577 119.38245575] meu pbest is [ 61.41132577 119.38245575]
I am at [55.98368333 12.89682142] meu pbest is [55.98368333 12.89682142]
I am at [38.37472606 82.91954951] meu pbest is [38.37472606 82.91954951]
I am at [73.00308689 74.33681003] meu pbest is [73.00308689 74.33681003]
I am at [103.60302569 106.55737896] meu pbest is [103.60302569 106.55737896]
I am at [77.46542836 30.90924108] meu pbest is [77.46542836 30.90924108]
I am at [ 13.27190352 140.45797161] meu pbest is [ 13.27190352 140.45797161]
I am at [90.02998776 99.70156727] meu pbest is [90.02998776 99.70156727]
C:/Users/hp/Desktop/eccpso.py:41: RuntimeWarning: overflow encountered in double_scalars
    return (particle.position[0] ** 3 + particle.position[1] + 1)**0.5
C:/Users/hp/Desktop/eccpso.py:61: RuntimeWarning: overflow encountered in multiply
    new_velocity = (W*particle.velocity) + (c1*random.random()) * (particle.pbest_position - particle.position) + \
C:/Users/hp/Desktop/eccpso.py:61: RuntimeWarning: invalid value encountered in add
    new_velocity = (W*particle.velocity) + (c1*random.random()) * (particle.pbest_position - particle.position) + \
```

Fig.4.1.2.2 PSO output for P=457

c)P=593

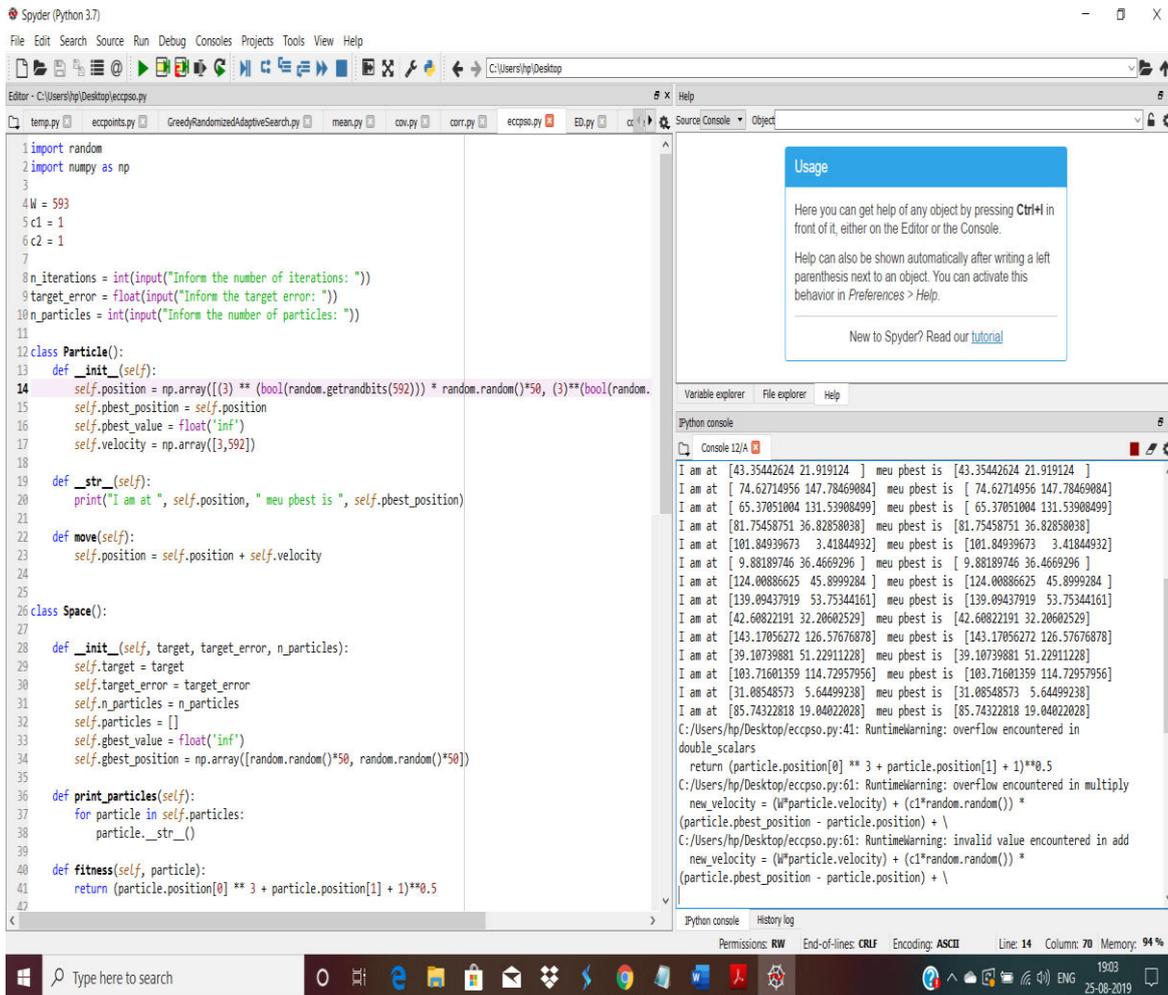


Fig.4.1.2.3 Python code for computing ECC Points with PSO with prime number,P=593 on Spyder(Python 3.7)

```
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

IPython 7.6.1 -- An enhanced Interactive Python.

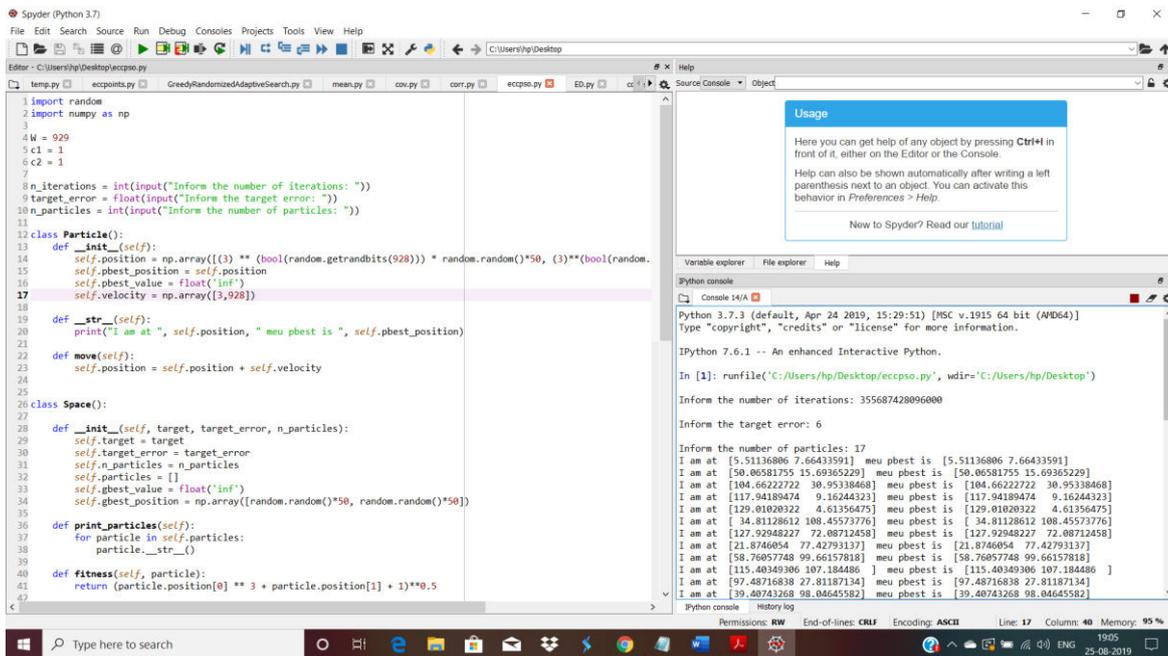
In [1]: runfile('C:/Users/hp/Desktop/eccps0.py', wdir='C:/Users/hp/Desktop')

Inform the number of iterations: 355687428096000
Inform the target error: 6

Inform the number of particles: 17
I am at [ 33.66601707 136.92770293 ] meu pbest is [ 33.66601707 136.92770293 ]
I am at [ 71.31911887 6.44930963 ] meu pbest is [ 71.31911887 6.44930963 ]
I am at [ 83.941363 5.88514816 ] meu pbest is [ 83.941363 5.88514816 ]
I am at [ 43.35442624 21.919124 ] meu pbest is [ 43.35442624 21.919124 ]
I am at [ 74.62714956 147.78469884 ] meu pbest is [ 74.62714956 147.78469884 ]
I am at [ 65.37051004 131.53908499 ] meu pbest is [ 65.37051004 131.53908499 ]
I am at [ 81.75458751 36.82858038 ] meu pbest is [ 81.75458751 36.82858038 ]
I am at [ 101.84939673 3.41844932 ] meu pbest is [ 101.84939673 3.41844932 ]
I am at [ 9.88189746 36.4669296 ] meu pbest is [ 9.88189746 36.4669296 ]
I am at [ 124.00886625 45.8999284 ] meu pbest is [ 124.00886625 45.8999284 ]
I am at [ 139.09437919 53.75344161 ] meu pbest is [ 139.09437919 53.75344161 ]
I am at [ 42.60822191 32.20602529 ] meu pbest is [ 42.60822191 32.20602529 ]
I am at [ 143.17056272 126.57676878 ] meu pbest is [ 143.17056272 126.57676878 ]
I am at [ 39.10739881 51.22911228 ] meu pbest is [ 39.10739881 51.22911228 ]
I am at [ 103.71601359 114.72957956 ] meu pbest is [ 103.71601359 114.72957956 ]
I am at [ 31.08548573 5.64499238 ] meu pbest is [ 31.08548573 5.64499238 ]
I am at [ 85.74322818 19.04022028 ] meu pbest is [ 85.74322818 19.04022028 ]
C:/Users/hp/Desktop/eccps0.py:41: RuntimeWarning: overflow encountered in double_scalars
  return (particle.position[0] ** 3 + particle.position[1] + 1)**0.5
C:/Users/hp/Desktop/eccps0.py:61: RuntimeWarning: overflow encountered in multiply
  new_velocity = (W*particle.velocity) + (c1*random.random()) * (particle.pbest_position - particle.position) + \
C:/Users/hp/Desktop/eccps0.py:61: RuntimeWarning: invalid value encountered in add
  new_velocity = (W*particle.velocity) + (c1*random.random()) * (particle.pbest_position - particle.position) + \
```

Fig.4.1.2.4 PSO output with P=593

d) P=929



```
1 import random
2 import numpy as np
3
4 N = 929
5 c1 = 1
6 c2 = 1
7
8 n_iterations = int(input("Inform the number of iterations: "))
9 target_error = float(input("Inform the target error: "))
10 n_particles = int(input("Inform the number of particles: "))
11
12 class Particle():
13     def __init__(self):
14         self.position = np.array([3] * (bool(random.getrandbits(928))) * random.random()*50, (3)**(bool(random.
15         self.pbest_position = self.position
16         self.pbest_value = float('inf')
17         self.velocity = np.array([3,928])
18
19     def __str__(self):
20         print("I am at ", self.position, " meu pbest is ", self.pbest_position)
21
22     def move(self):
23         self.position = self.position + self.velocity
24
25
26 class Space():
27
28     def __init__(self, target, target_error, n_particles):
29         self.target = target
30         self.target_error = target_error
31         self.n_particles = n_particles
32         self.particles = []
33         self.gbest_value = float('inf')
34         self.gbest_position = np.array([random.random()*50, random.random()*50])
35
36     def print_particles(self):
37         for particle in self.particles:
38             particle.__str__()
39
40     def fitness(self, particle):
41         return (particle.position[0] ** 3 + particle.position[1] + 1)**0.5
42
```

Usage

Here you can get help of any object by pressing **Ctrl+H** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in **Preferences > Help**.

New to Spyder? Read our [tutorial](#)

Python console

Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

Python 7.6.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/hp/Desktop/eccps.py', wdir='C:/Users/hp/Desktop')

Inform the number of iterations: 355687428096000

Inform the target error: 6

Inform the number of particles: 17

I am at [5.51136806 7.66433591] meu pbest is [5.51136806 7.66433591]
I am at [50.06581755 15.69365229] meu pbest is [50.06581755 15.69365229]
I am at [104.66222722 30.95338468] meu pbest is [104.66222722 30.95338468]
I am at [117.94188474 9.16244323] meu pbest is [117.94188474 9.16244323]
I am at [129.01020322 4.61356475] meu pbest is [129.01020322 4.61356475]
I am at [34.81128612 108.45573776] meu pbest is [34.81128612 108.45573776]
I am at [127.92948227 72.08712458] meu pbest is [127.92948227 72.08712458]
I am at [21.8746054 77.42791131] meu pbest is [21.8746054 77.42791131]
I am at [58.76057748 99.66157818] meu pbest is [58.76057748 99.66157818]
I am at [115.40349306 107.184486] meu pbest is [115.40349306 107.184486]
I am at [97.48716838 27.81187134] meu pbest is [97.48716838 27.81187134]
I am at [39.48743268 98.04645582] meu pbest is [39.48743268 98.04645582]

Fig.4.1.2.5 Python code for computing ECC Points with PSO with prime number,P=929 on Spyder(Python 3.7)

```
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

IPython 7.6.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/hp/Desktop/eccps0.py', wdir='C:/Users/hp/Desktop')

Inform the number of iterations: 355687428096000

Inform the target error: 6

Inform the number of particles: 17
I am at [5.51136806 7.66433591] meubest is [5.51136806 7.66433591]
I am at [50.06581755 15.69365229] meubest is [50.06581755 15.69365229]
I am at [104.66222722 30.95338468] meubest is [104.66222722 30.95338468]
I am at [117.94189474 9.16244323] meubest is [117.94189474 9.16244323]
I am at [129.01020322 4.61356475] meubest is [129.01020322 4.61356475]
I am at [34.81128612 108.45573776] meubest is [34.81128612 108.45573776]
I am at [127.92948227 72.08712458] meubest is [127.92948227 72.08712458]
I am at [21.8746054 77.42793137] meubest is [21.8746054 77.42793137]
I am at [58.76057748 99.66157818] meubest is [58.76057748 99.66157818]
I am at [115.40349306 107.184486 ] meubest is [115.40349306 107.184486 ]
I am at [97.48716838 27.81187134] meubest is [97.48716838 27.81187134]
I am at [39.40743268 98.04645582] meubest is [39.40743268 98.04645582]
I am at [15.76689235 47.70358666] meubest is [15.76689235 47.70358666]
I am at [55.24800684 29.29279562] meubest is [55.24800684 29.29279562]
I am at [143.29994672 24.72522159] meubest is [143.29994672 24.72522159]
I am at [101.89320987 127.21196668] meubest is [101.89320987 127.21196668]
I am at [66.16833069 15.88015323] meubest is [66.16833069 15.88015323]
C:/Users/hp/Desktop/eccps0.py:41: RuntimeWarning: overflow encountered in double_scalars
return (particle.position[0] ** 3 + particle.position[1] + 1)**0.5
C:/Users/hp/Desktop/eccps0.py:61: RuntimeWarning: overflow encountered in multiply
new_velocity = (w*particle.velocity) + (c1*random.random()) * (particle.pbest_position - particle.position) + \
C:/Users/hp/Desktop/eccps0.py:61: RuntimeWarning: invalid value encountered in add
new_velocity = (w*particle.velocity) + (c1*random.random()) * (particle.pbest_position - particle.position) + \
```

Fig.4.1.2.6 PSO output with P=929

4.4 CORRELATION OUTPUT:-

4.4.1 WITHOUT PSO

a) P=251

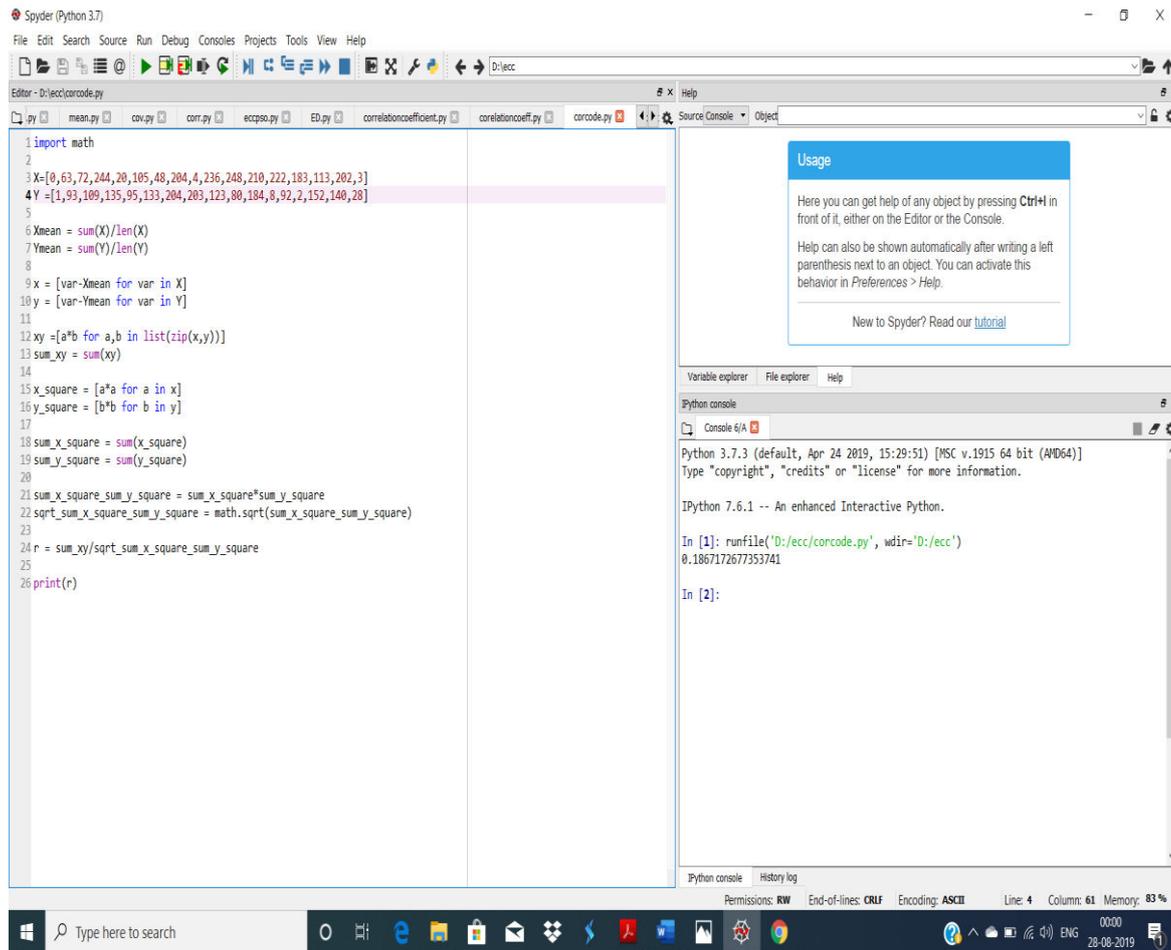
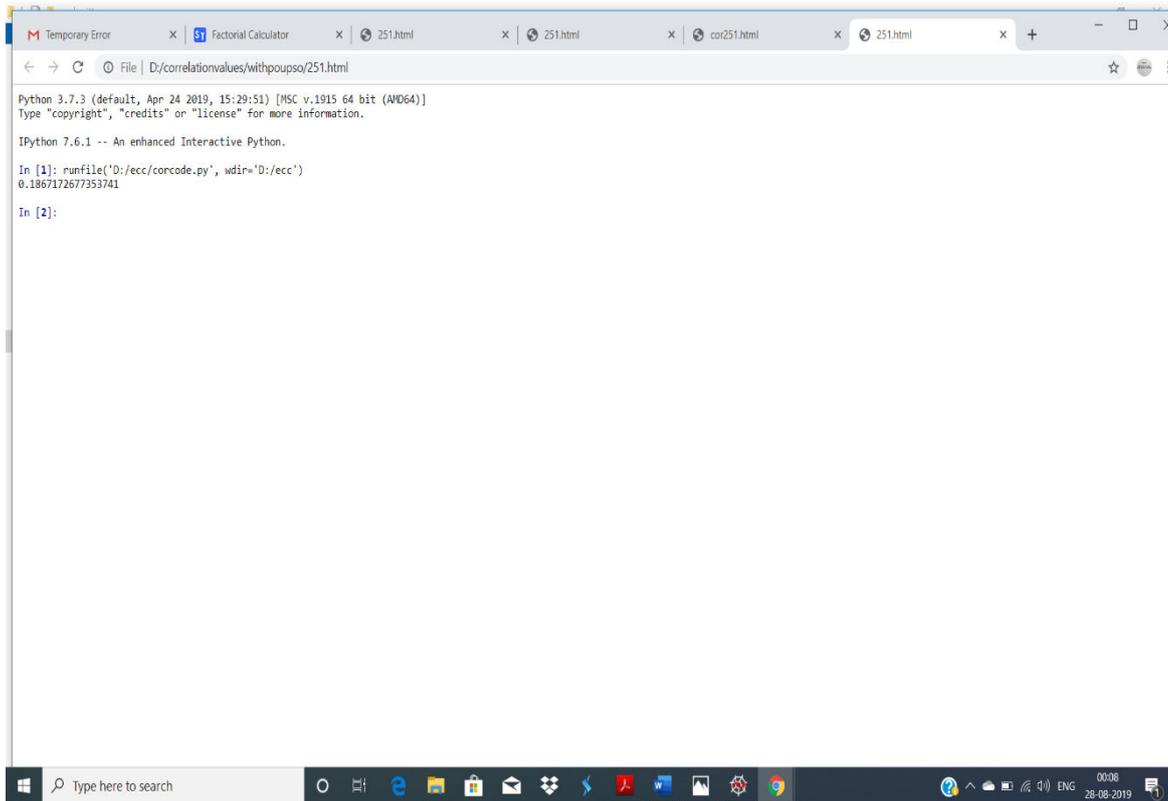


Fig.4.1.2.7 Python code for calculating correlation coefficient with P=251



```
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

Python 7.6.1 -- An enhanced Interactive Python.

In [1]: runfile('D:/ecc/corcode.py', wdir='D:/ecc')
0.1867172677353741

In [2]:
```

Fig.4.1.2.8 Correlation Coefficient output with P=251

b)P=457

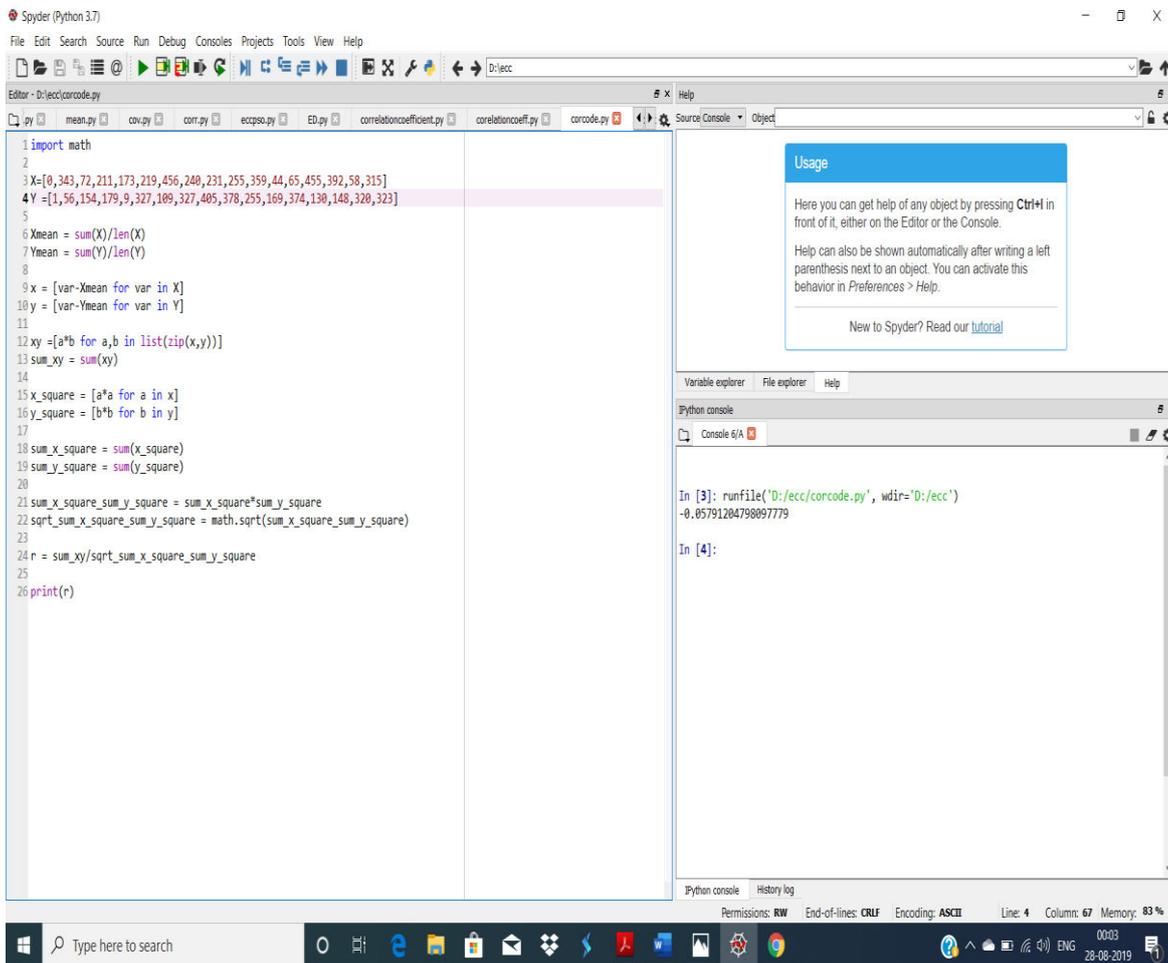
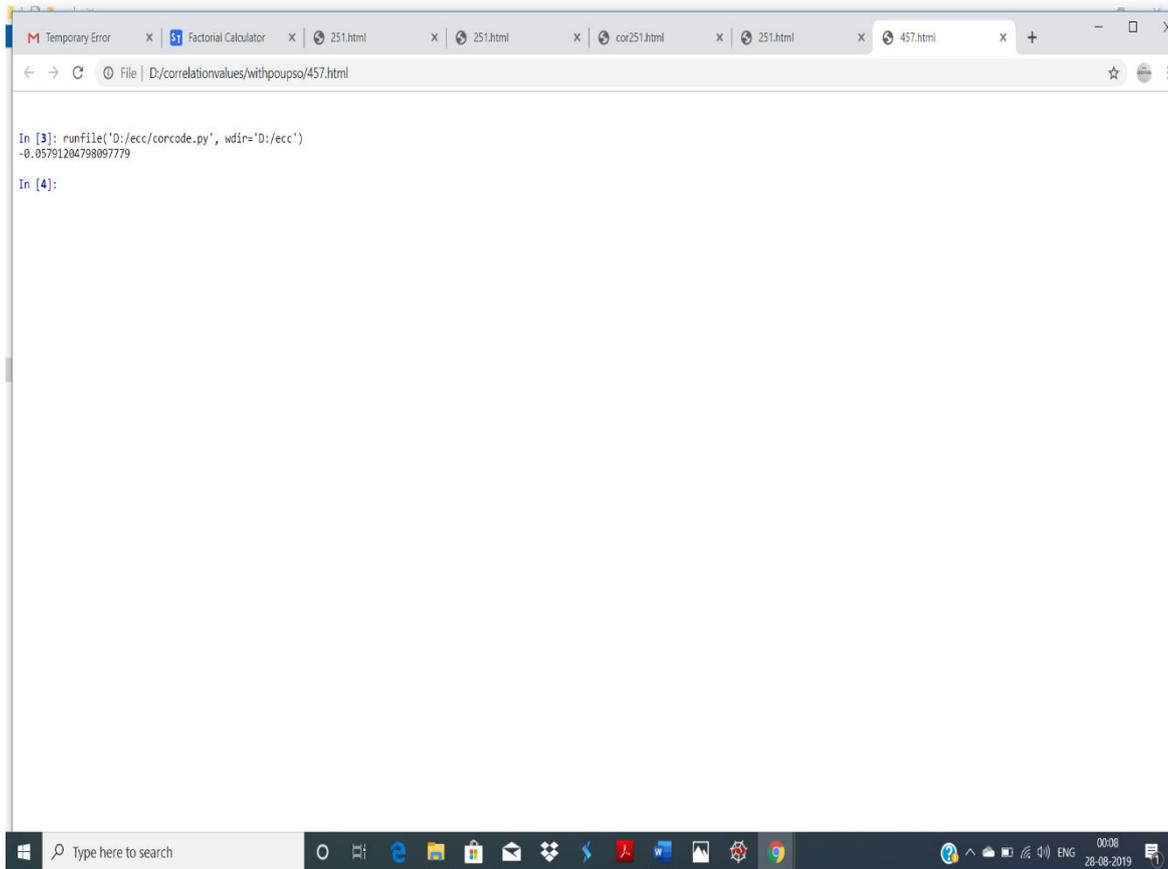


Fig.4.1.2.9 Python code for calculating correlation coefficient with P=457



```
In [3]: runfile('D:/ecc/corcode.py', wdir='D:/ecc')
-0.05791204798097779

In [4]:
```

Fig.4.1.3.0 Correlation Coefficient output with P=457

c)P=593

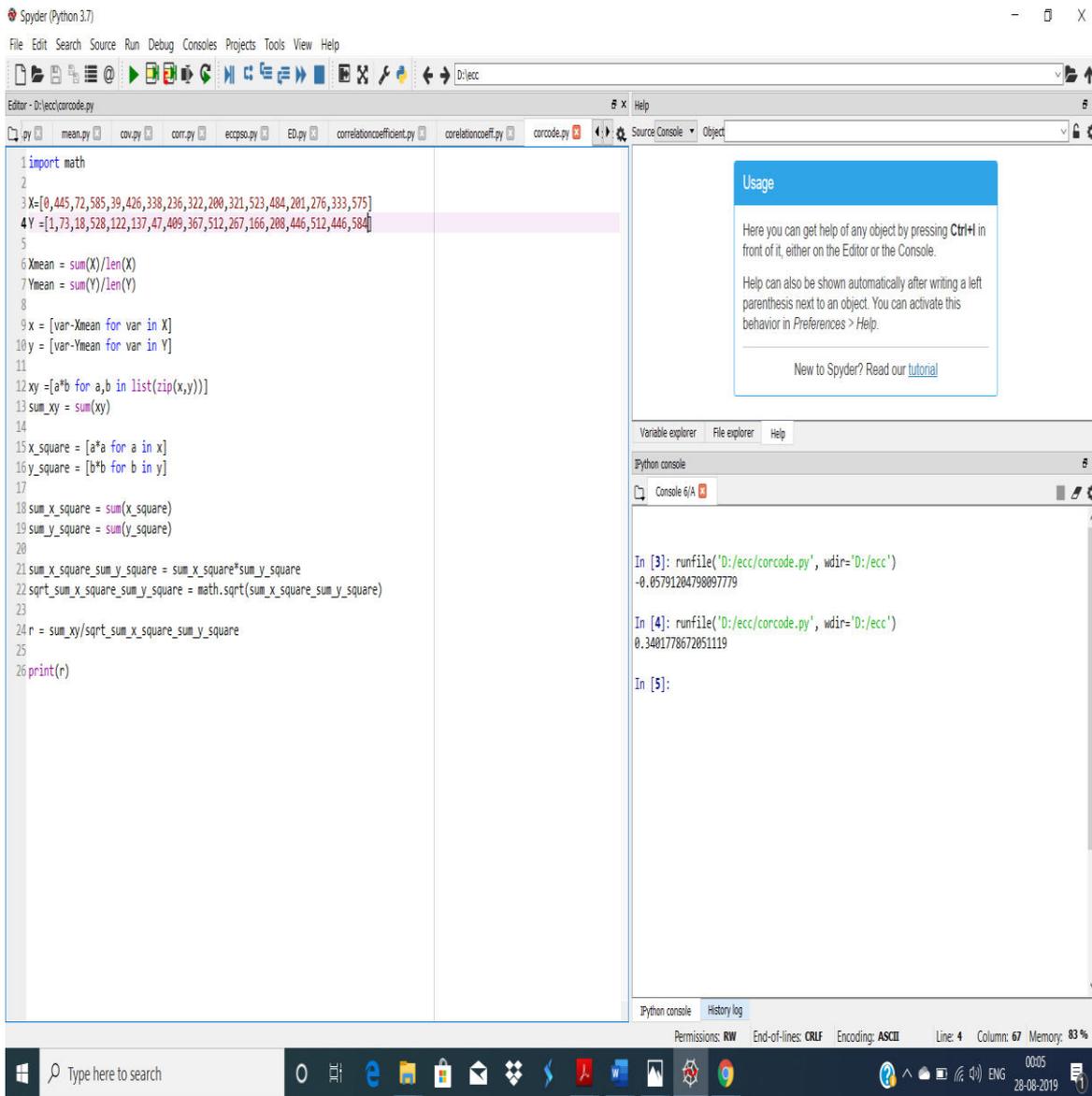


Fig.4.1.3.1 Python code for calculating correlation coefficient with P=593

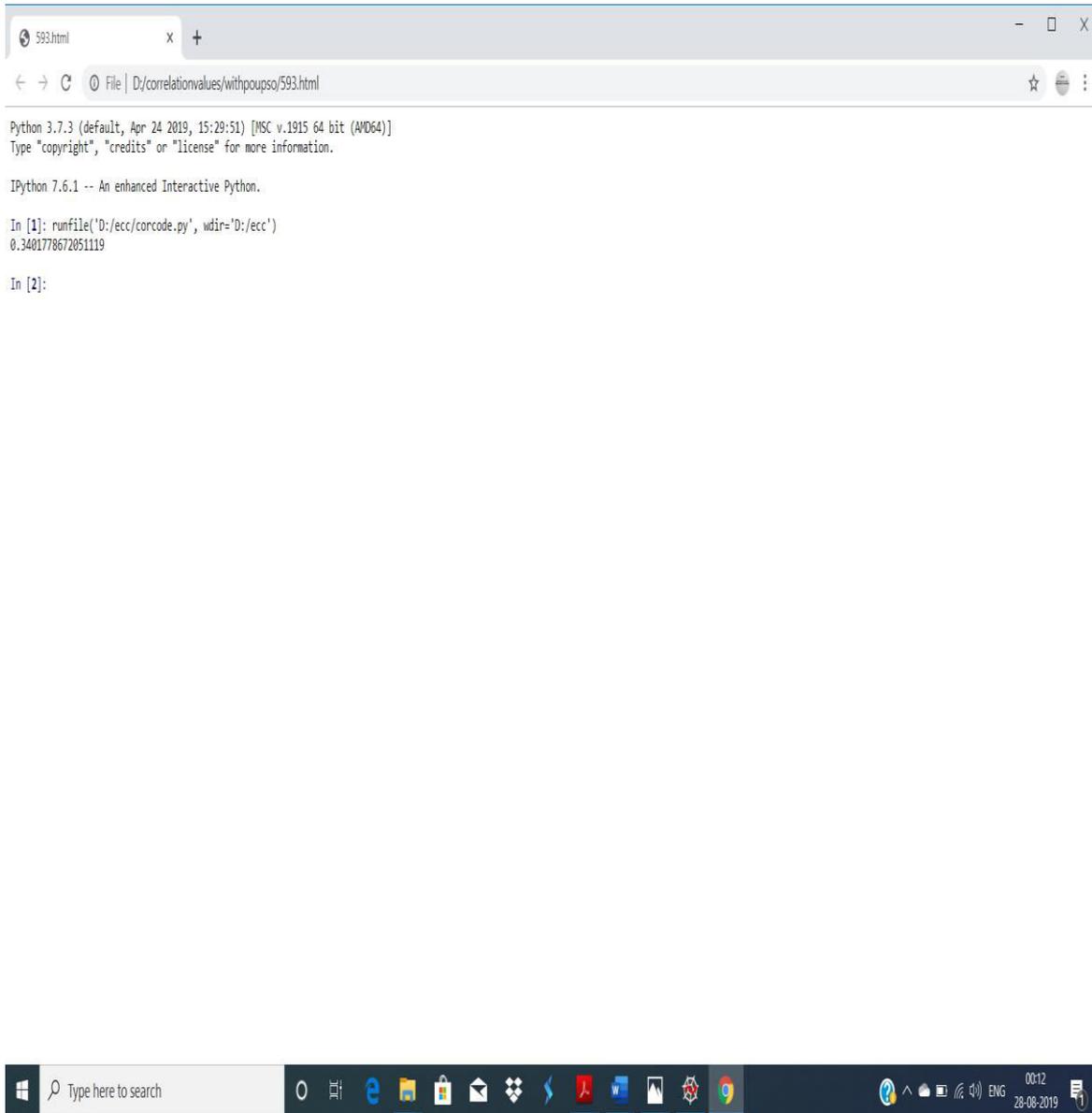


Fig.4.1.3.2 Correlation Coefficient Output with P=593

d)929

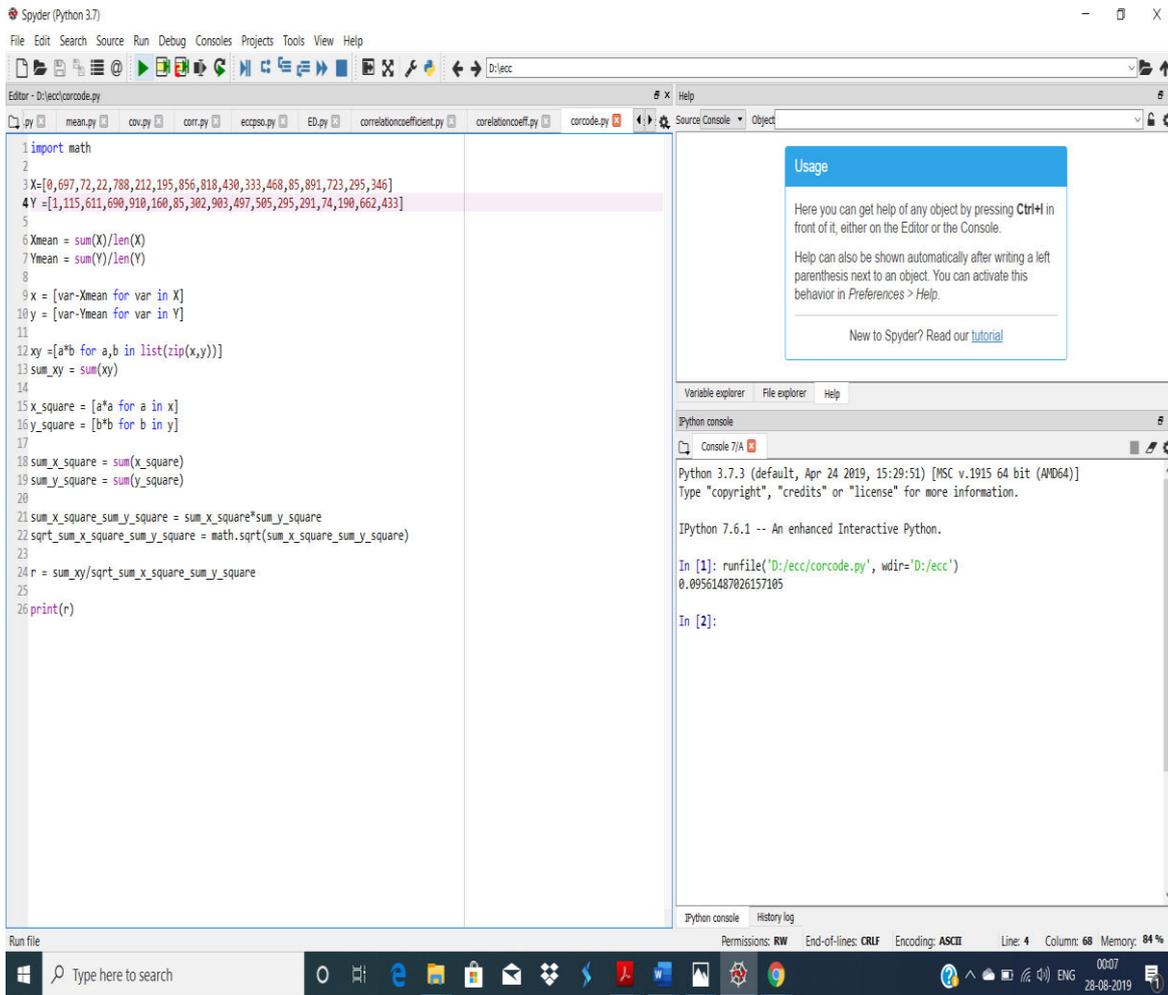


Fig.2.1.3.3 Python code for calculating correlation coefficient with P=929

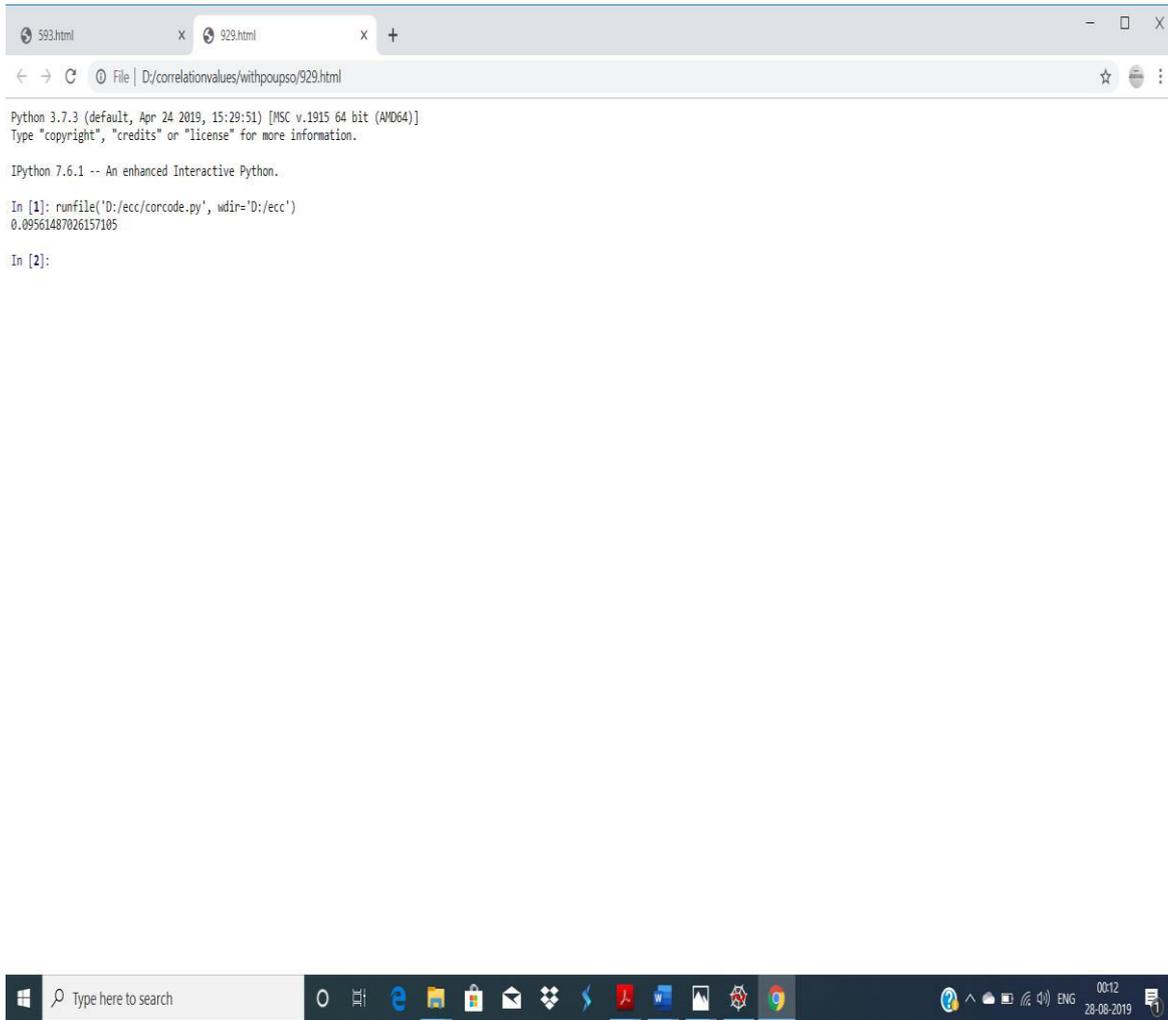


Fig.4.1.3.4 Correlation Coefficient output with P=929

4.4.2 WITH PSO

a)P=251

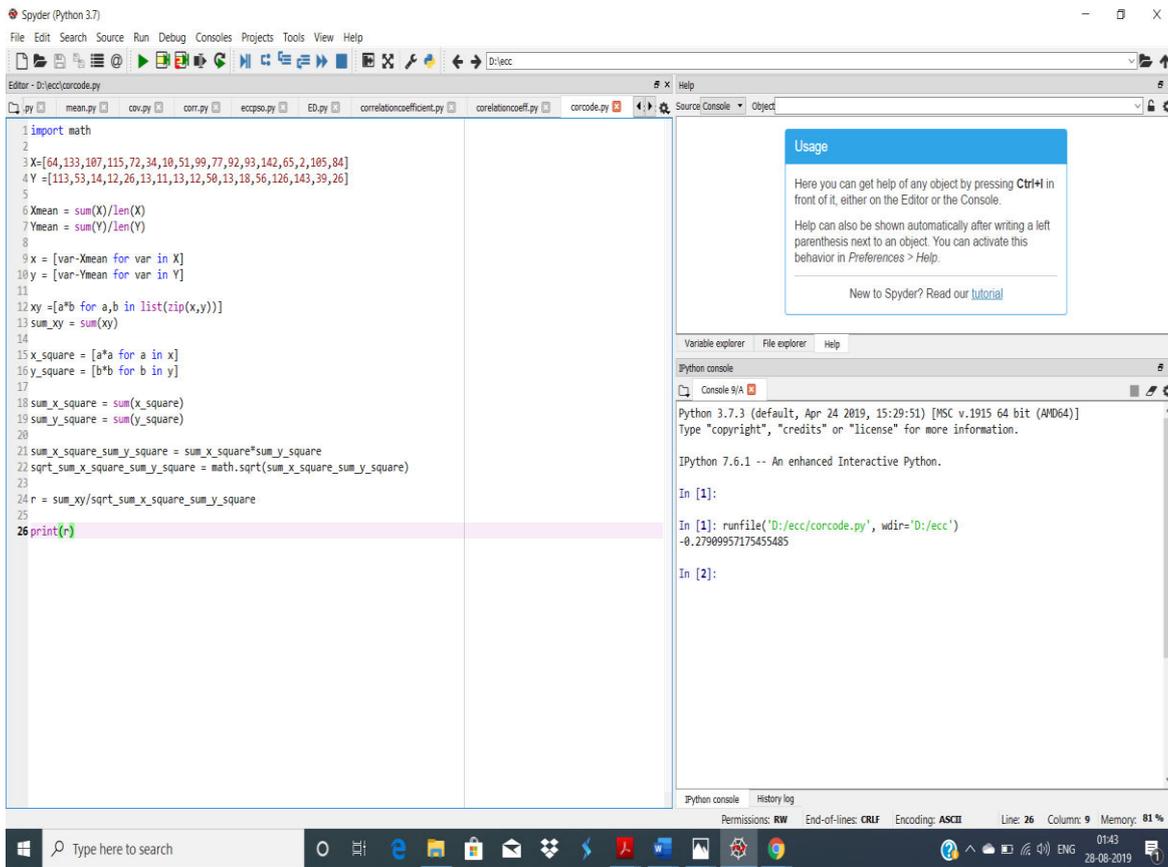


Fig.4.1.3.5 Python code for calculating correlation coefficient of obtained PSO points with P=251

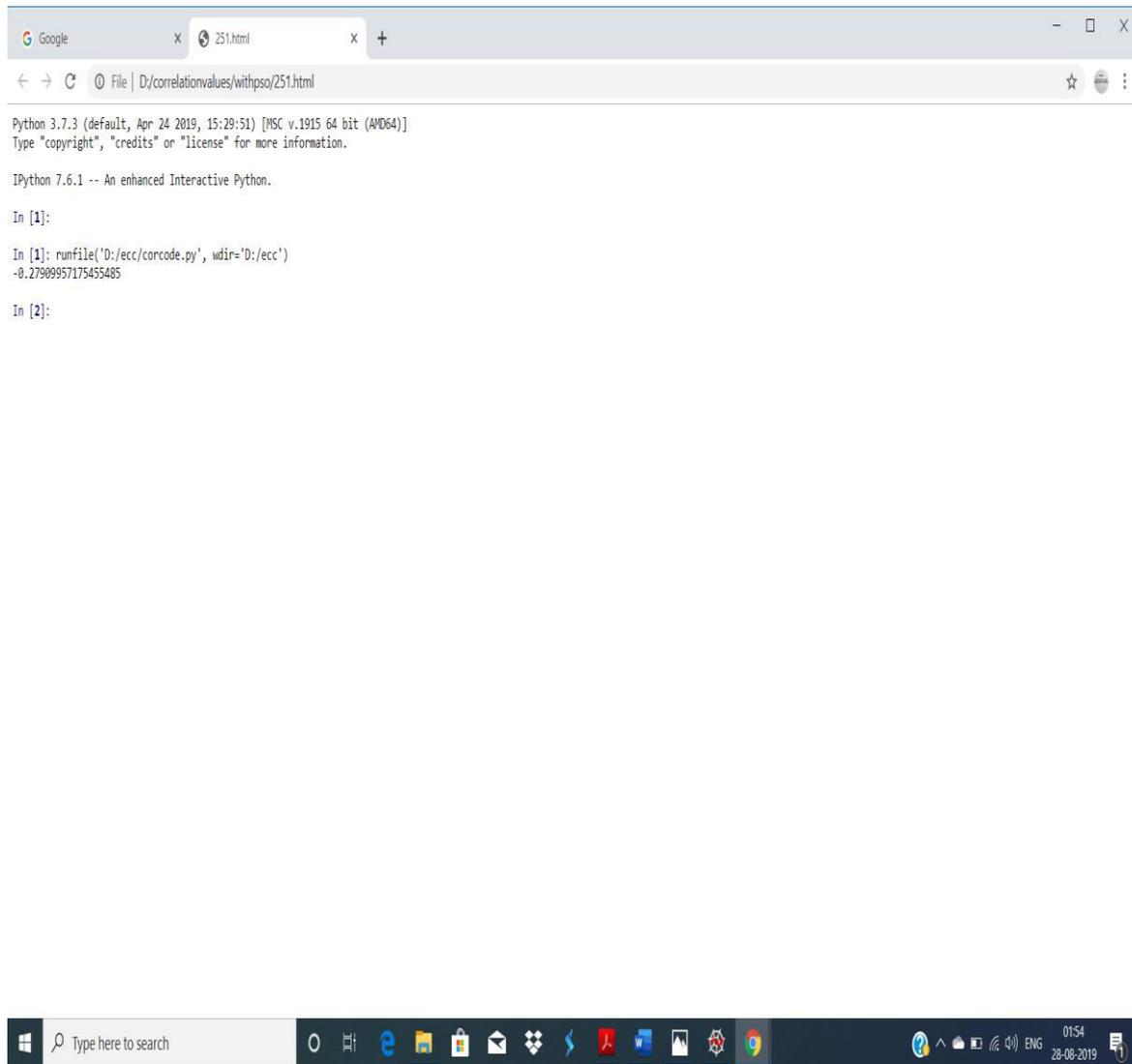


Fig.4.1.3.6 Correlation Coefficient of PSO applied elliptic points with P=251

b)P=457

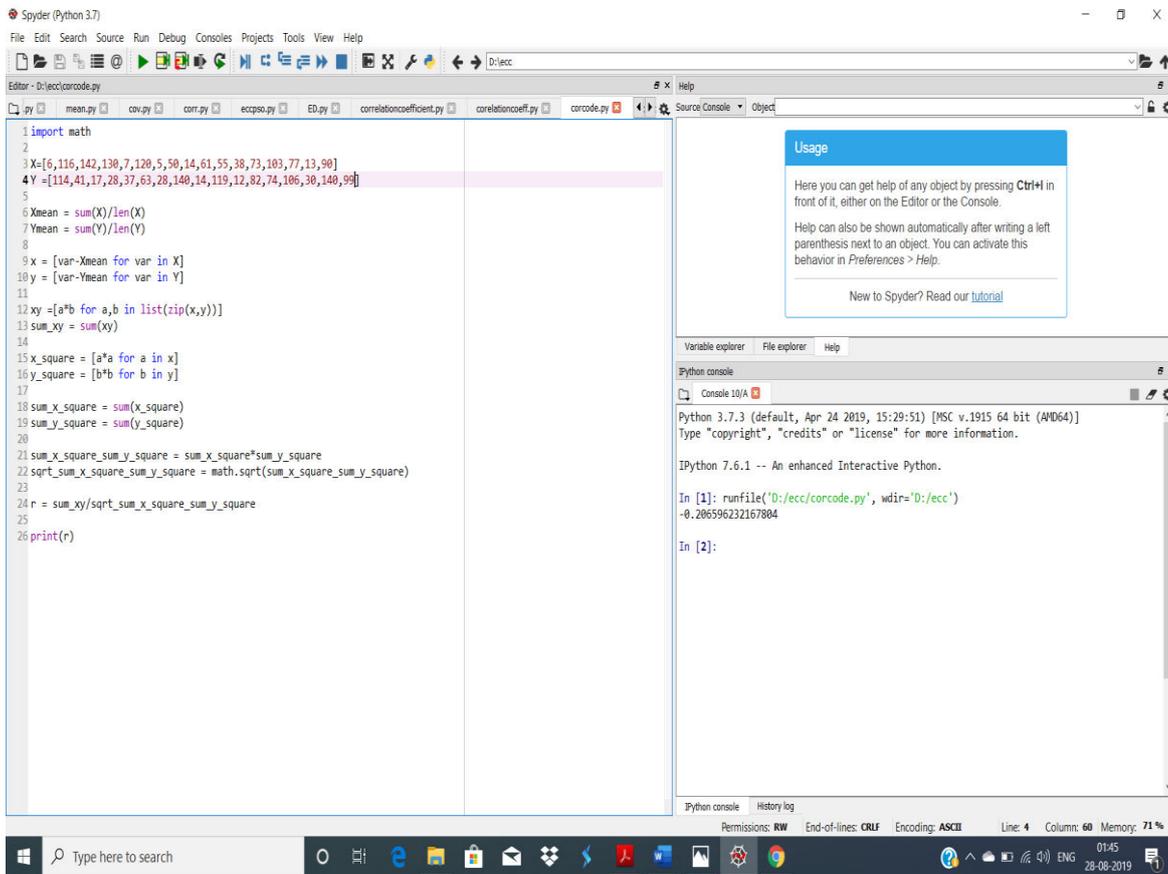


Fig.4.1.3.7 Python code for calculating correlation coefficient of obtained PSO points with P=251

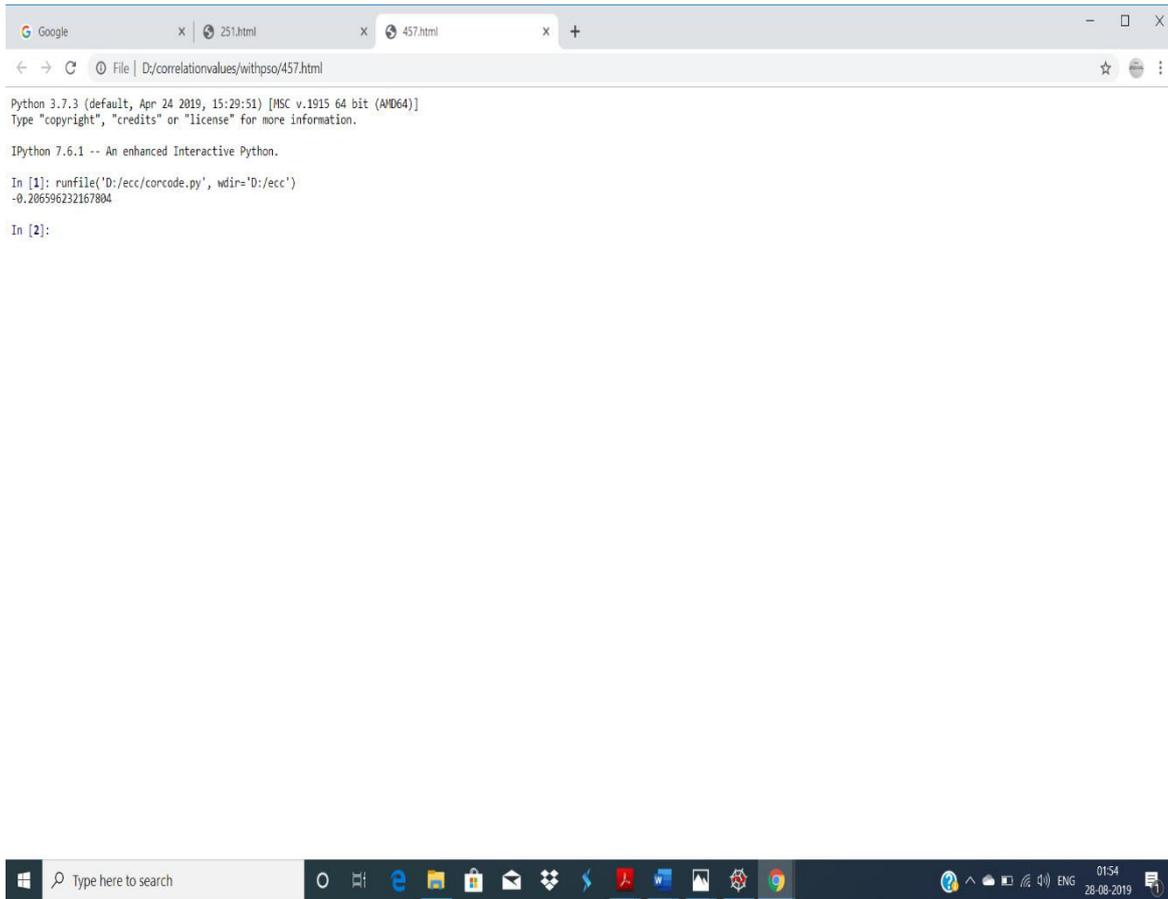


Fig.4.1.3.8 Correlation Coefficient of PSO applied elliptic points with P=457

b) P=593

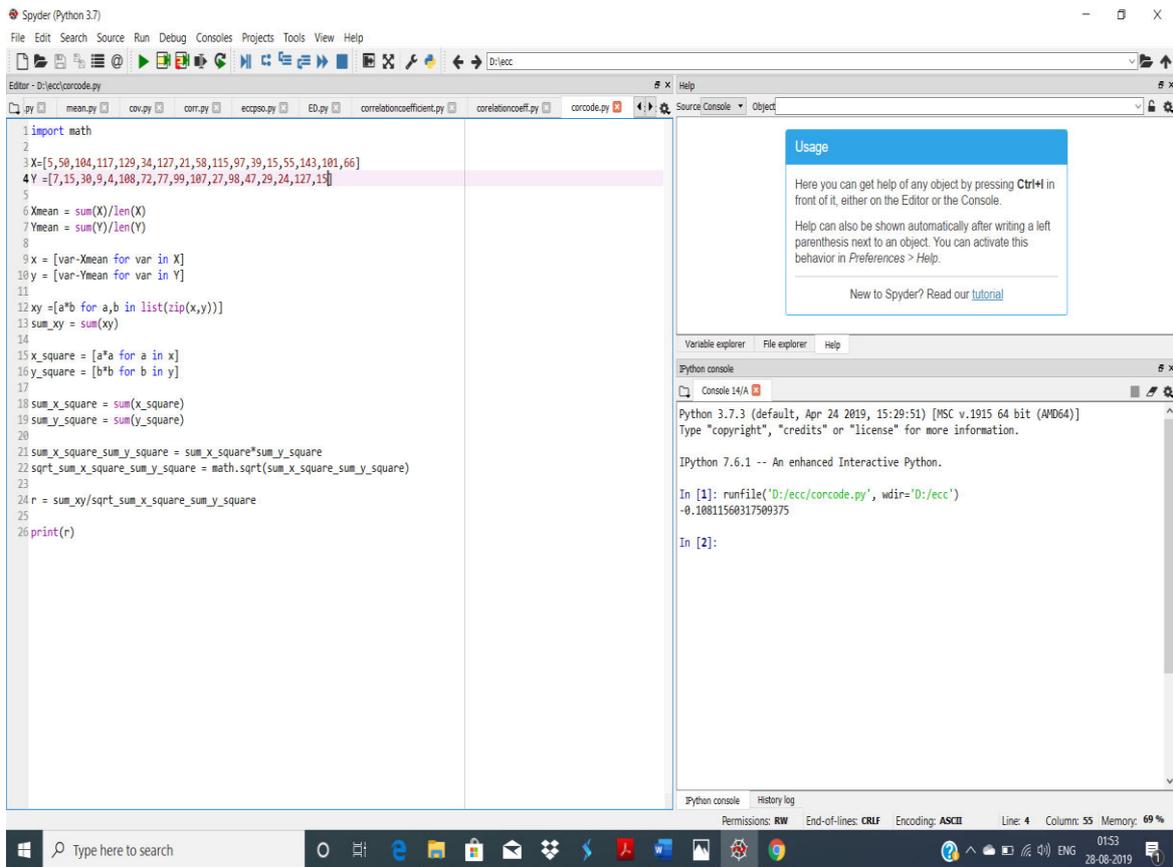


Fig.4.1.3.9 Python code for calculating correlation coefficient of obtained PSO points with P=593

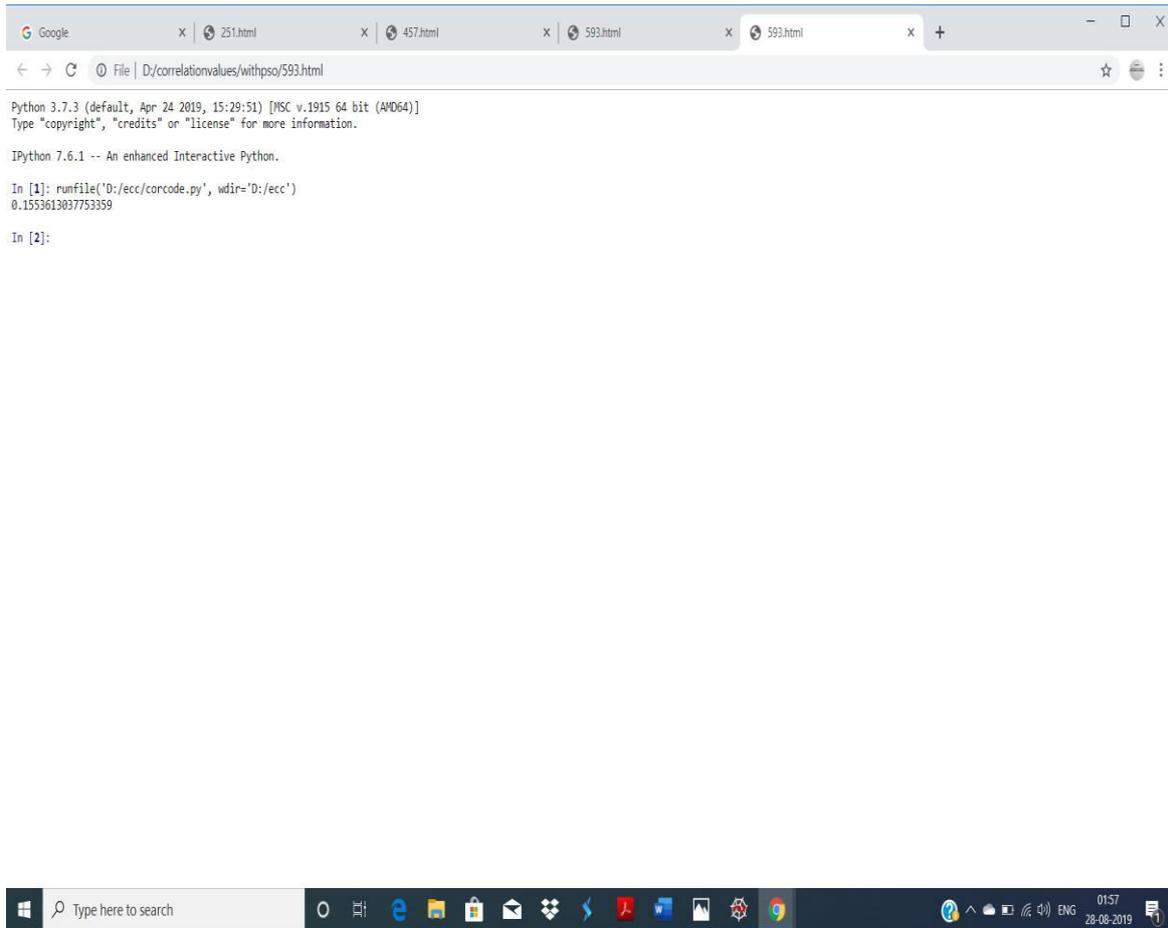


Fig.4.1.4.0 Correlation Coefficient of PSO applied elliptic points with P=593

c) $P=929$

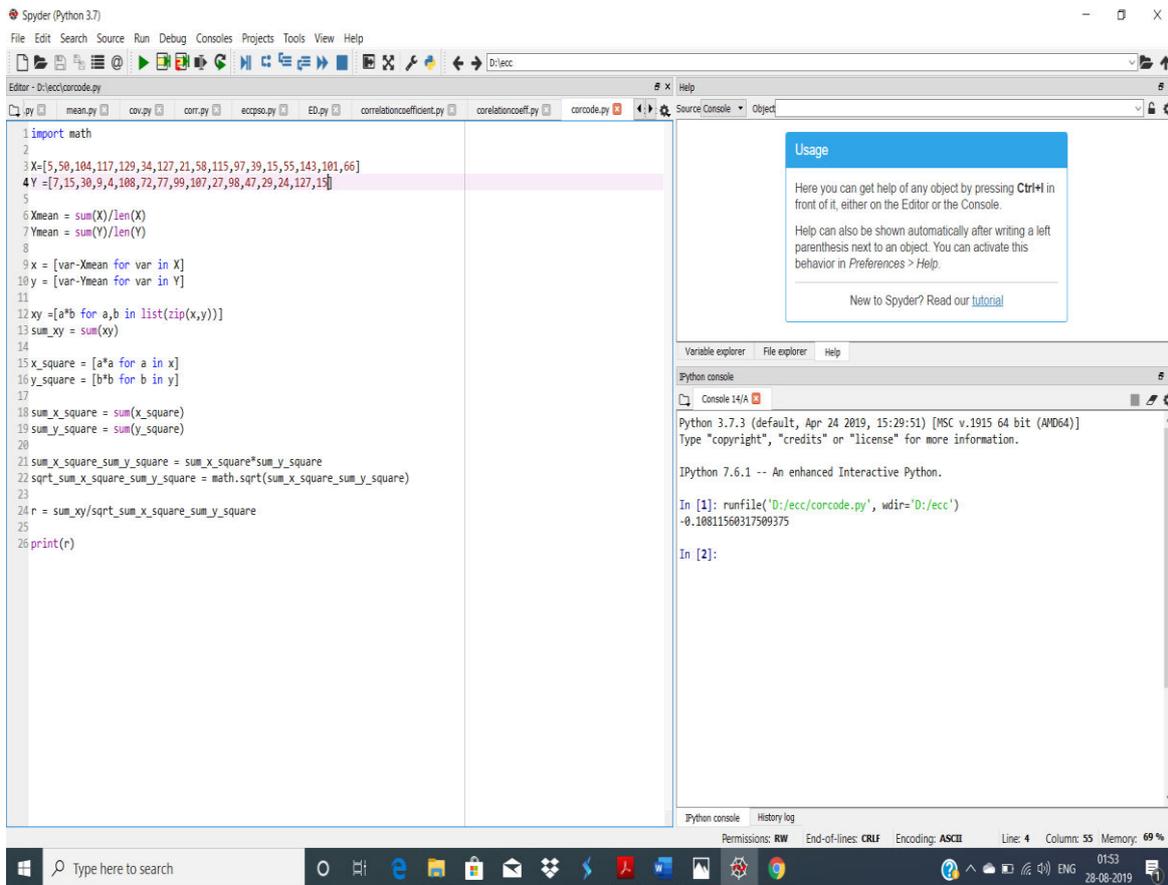


Fig.4.1.4.1 Python code for calculating correlation coefficient of obtained PSO points with $P=929$

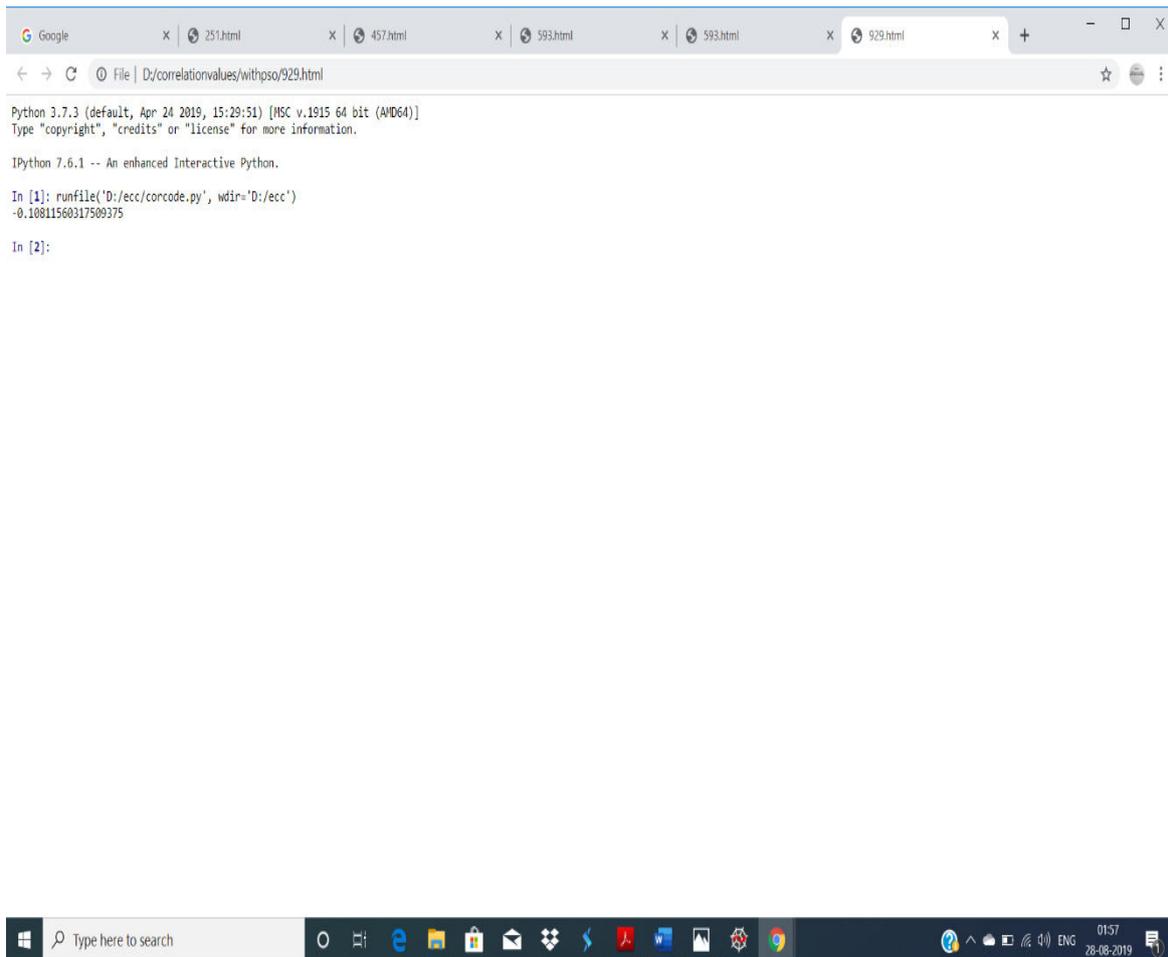


Fig.4.1.4.2 Correlation Coefficient of PSO applied elliptic points with P=929

RESULT ANALYSIS

5.1 TABLE SHOWING IMPROVEMENT OF CORRELATION AMONGST ELLIPTIC POINTS AFTER APPLYING METAHEURISTIC ALGORITHM VIZ.PSO

S No.	Prime number	Correlation coefficient between X and Y Coordinates of elliptic points	Correlation coefficient after applying Metaheuristics viz.PSO on elliptic points
1	251	0.1867172677353741	-0.27909957175455485
2	457	-0.05791204798097779	-0.206596232167804
3	593	0.3401778672051119	-0.06280917543035339
4	929	0.09561487026157105	-0.10811560317509375

TABLE 5.1 SHOWING IMPROVEMENT OF CORRELATION AMONGST ELLIPTIC POINTS AFTER APPLYING METAHEURISTIC ALGORITHM VIZ.PSO

5.2 GRAPH SHOWING CORRELATION VALUES VERSUS PRIME NUMBERS

1.CORRELATION VERSUS PRIME NUMBER WITHOUT PSO

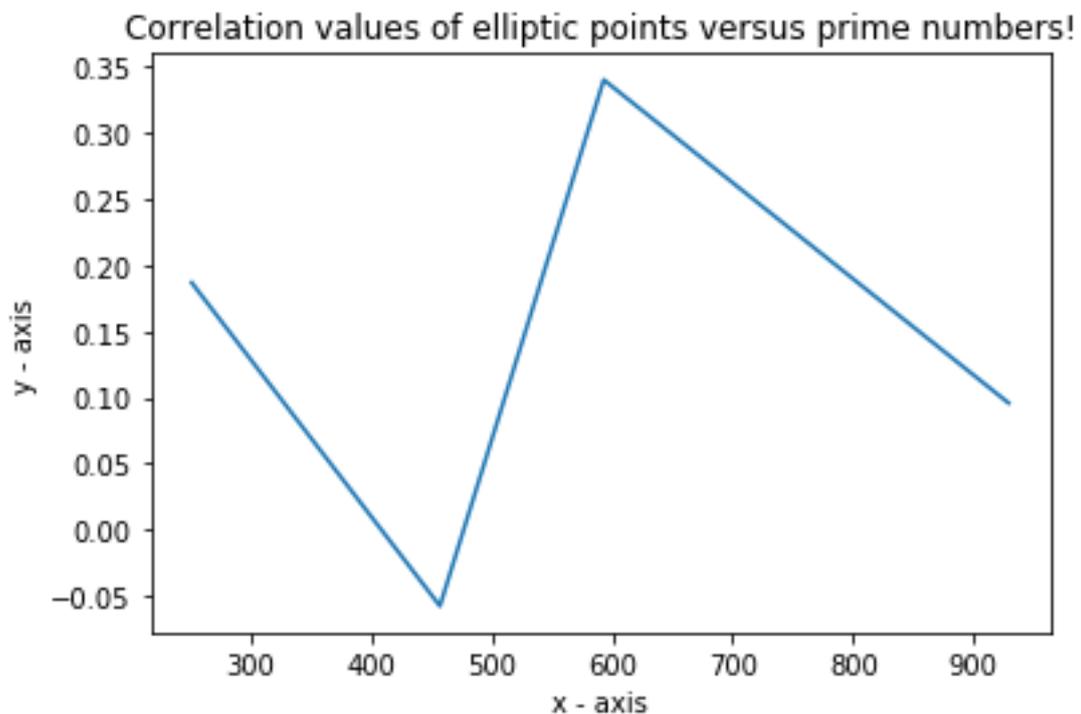


Fig.5.1.1.1 GRAPH SHOWING CORRELATION VERSUS PRIME NUMBER WITHOUT PSO

2.CORRELATION VALUES VERSUS PRIME NUMBERS AFTER APPLYING PSO

Correlation values obtained after applying PSO on elliptic points versus prime numbers!

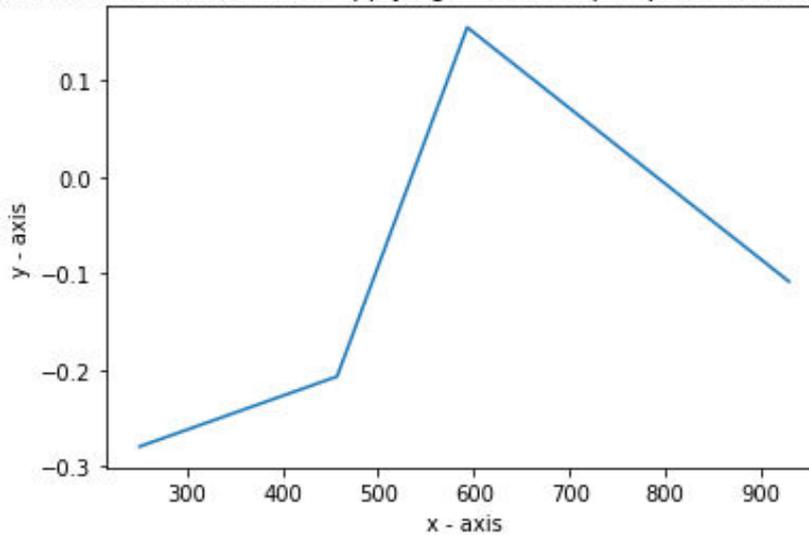


Fig.5.1.1.2 GRAPH SHOWING CORRELATION VALUES VERSUS PRIME NUMBERS AFTER APPLYING PSO

The above obtained correlation values are compared and contrasted with the earlier work of “Selection of Fittest Key Using Genetic Algorithm and Autocorrelation in Cryptography”[12] based on the following results.

Table 5.2: - Table showing autocorrelation values of different keys when Genetic Algorithm is applied on DES to find the fittest key

	Autocorrelation values
Iteration 1 Crossover set	-0.0077539468
Iteration 2 Population Generation set	0.0535485376
Iteration 3 Crossover set	-0.0111042183

It is observed that the values of correlation are much better when Elliptic Curve Cryptography is amalgamated with a metaheuristic algorithm named Particle Swarm Optimization as almost all correlation values are negative and less than Table 5.2 values

Conclusion and Future Scope

6.1 Conclusion

- Proximity of correlation value to -1 guarantees randomness in the key
- The key may be less susceptible to any adversaries attack.
- From Fig.5.1.1.2 graph, it is inferred that with large prime numbers, correlation values decrease increasing randomness in the key when PSO is applied on elliptic curve points

6.2 Future Scope

- The venture can be conveyed on asset obliged gadgets like versatile phones, IoT and so on
- The symmetric cover key offers greater security when contrasted with RSA with little key size

References

- [1]D. Hankerson, A. Menezes, and S.A. Vanstone, Guide to Elliptic Curve Cryptography, Springer-Verlag, 2004
- [2]T.A. Feo and M.G.C. Resende (1995) Greedy randomized adaptive search procedures. Journal of Global Optimization, 6:109–133, 1995.
- [3]Hansen, P.; Mladenovic, N.; Perez, J.A.M. (2010). "Variable neighborhood search: methods and applications". Annals of Operations Research. 175: 367–407doi:10.1007/s10479-009-0657-6.
- [4]Mattern, Friedemann; Floerkemeier, Christian (2010). "From the Internet of Computer to the Internet of Things" (PDF). Informatik-Spektrum. 33 (2): 107–121. Bibcode:2009InfSp..32..496H. doi:10.1007/s00287-010-0417-7. Retrieved 3 February 2014.
- [5]Glover, F.; Kochenberger, G.A. (2003). Handbook of metaheuristics. 57. Springer, International Series in Operations Research & Management Science. ISBN 978-1-4020-7263-5.
- [6]Du, Wenliang Kevin; Deng, Jing; Han, Yunghsiang S.; and Varshney, Pramod K., "A Pairwise Key Pre-Distribution Scheme for Wireless Sensor Networks" (2000). Electrical Engineering and Computer Science. Paper 36. <http://surface.syr.edu/eecs/36>
- [7]Bitcoin Exchanges Linked To Silk Road? So the FBI Thinks, retrieved from<https://silkroaddrugs.org/category/bitcoin/>
- [8]Croxtton, Frederick Emory; Cowden, Dudley Johnstone; Klein, Sidney (1968) Applied General Statistics, Pitman. ISBN 9780273403159 (page 625)
- [9]Dargie, W. and Poellabauer, C. (2010). Fundamentals of wireless sensor networks: theory and practice.

John Wiley and Sons. pp. 168–183, 191–192. ISBN 978-0-470-99765-9

[10]Savage, David G., LA Times, Supreme Court rules in favor of California police chief who read employee's texts

[11] Zhang, Y. (2015). "A Comprehensive Survey on Particle Swarm OptimizationAlgorithm and Its Applications". Mathematical Problems in Engineering. 2015: 931256.

[12]Sania Jawaid, AnamSaiyeda,,NabaSuroor"Selection of Fittest Key Using Genetic Algorithm and Autocorrelation in Cryptography"(2015)DOI:10.12691/jcsa-3-2-5

[13] Abdul wahab, Hala&Kadhem, Suhad&Kadhim, Estabraq. (2012). planned Approach for Elliptic curve hidden writing supported Metaheuristic Algorithms. International Journal of research. 2. 1-5. 10.15373/22778179/OCT2013/33.

[14]E. Jovanov, A. Milenkovic, C. Otto, P. C.D. Groen, "A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation", Journal of Neuro Engineering and Rehabilitation, vol. 2, no. 6, 2005.

[15]Tebogo Kgogo, Bassey Isong, Adnan M. Abu-Mahfouz, "Software defined wireless sensor networks security challenges", AFRICON 2017 IEEE, pp. 1508-1513, 2017.

[16]A. R. Elshazly et al., "Synchronized Double Watermark Audio Watermarking Scheme Based on a Transform Domain for Stereo Signals", IEEE Fourth International Ja-pan-Egypt Conference on Electronics Communications and Computers, 2016.

[17] R.N. Akram, K. Markantonakis, Smart Cards: State-of-the-Art to Future Directions, IEEE, pp. 154-162, 2013.