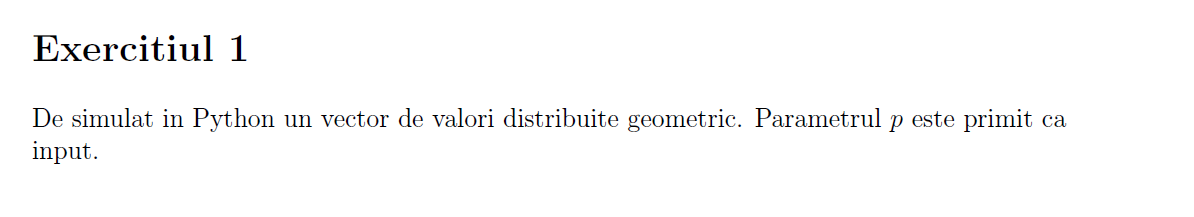
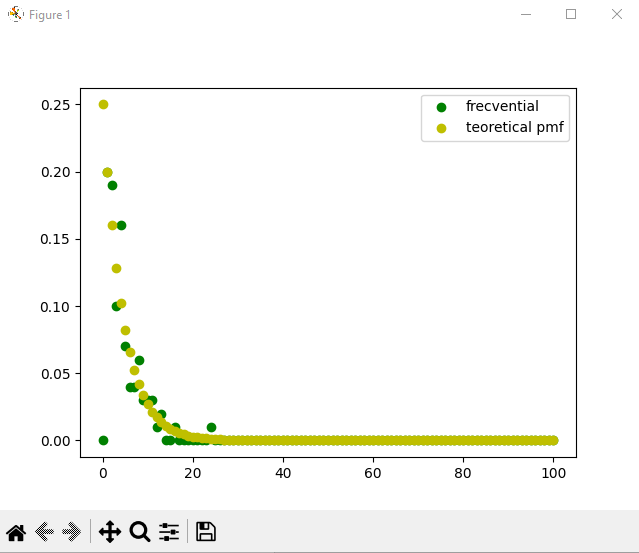
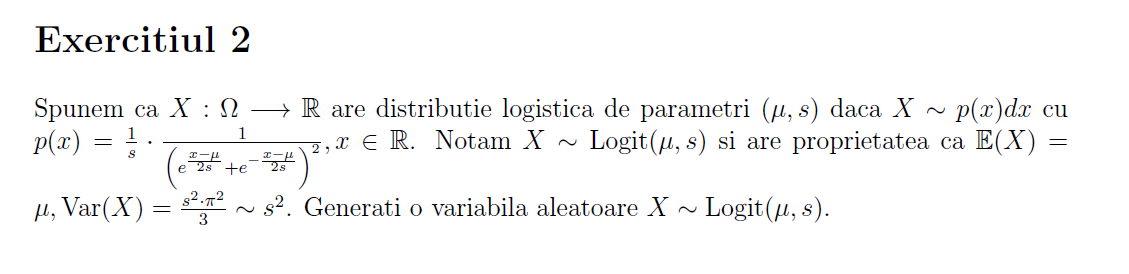
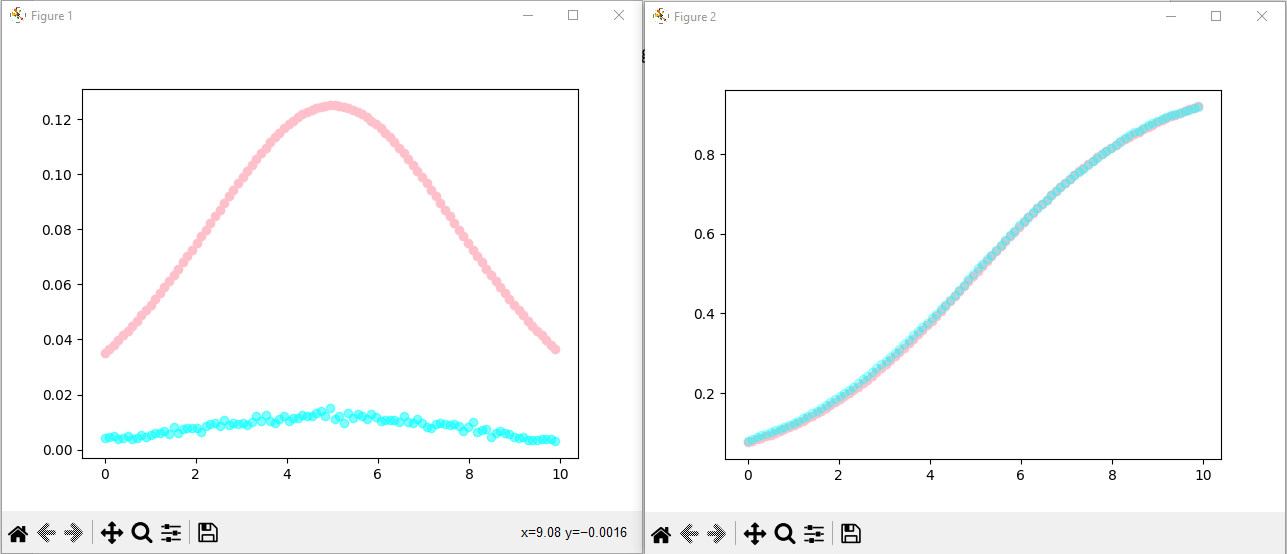
**Tema 3**

****

def probability\_to\_occur\_at(attempt, p):  
 return (1-p)\*\*(attempt-1) \* p  
  
def geometric(p, N=100):  
 V = [0 for \_ in range(N)]  
 val\_max = 1  
 for i in range(N):  
 nr = 1 \* (np.random.uniform(0, 1) < p)  
 cnt = 1  
 while nr != 1:  
 nr = 1 \* (np.random.uniform(0, 1) < p)  
 cnt += 1  
 V[i] = cnt  
 if val\_max < cnt:  
 val\_max = cnt  
 if val\_max < N + 1:  
 val\_max = N + 1  
  
 # vectorul calculat frecventionist  
 F = [0 for \_ in range(val\_max)]  
 for i in range(val\_max):  
 F[i] = V.count(i) / N  
 plt.scatter(range(N + 1), F, color='g', label='frecvential')  
  
 # compar cu valorile teoretice  
 T = [0 for \_ in range(val\_max)]  
 for i in range(val\_max):  
 T[i] = probability\_to\_occur\_at(i, p)  
 plt.scatter(range(N + 1), T, color='y', label='teoretical pmf')  
  
 plt.legend()  
 plt.show()  
  
def main():  
 p = float(input("Introduceti p = "))  
 geometric(p)

****Pentru p = 0.2 si N=100 a fost generat urmatorul grafic:

import numpy as np  
import matplotlib.pyplot as plt  
from math import e  
import math  
  
  
def count\_in\_interval(arr, left\_lim, right\_lim):  
 ctr = 0  
 for x in arr:  
 if x >= left\_lim and x <= right\_lim:  
 ctr += 1  
 return ctr  
  
  
def count\_smaller(arr, left\_lim):  
 ctr = 0  
 for x in arr:  
 if x <= left\_lim:  
 ctr += 1  
 return ctr  
  
  
def Logistic(u, s, N):  
 uniform\_array = []  
 for i in range(N):  
 uniform\_array.append(np.random.uniform(0, 1))  
  
 log\_x = np.log(uniform\_array)  
 uniform\_array = [1-x for x in uniform\_array]  
 log\_1\_x = np.log(uniform\_array)  
 log\_simulated = [u + s \* (log\_x[i] - log\_1\_x[i]) for i in range(N)]  
 return log\_simulated  
  
  
def see\_pdf(u, s, simulated\_values, bins\_number):  
 x\_bins = np.linspace(0, 10, bins\_number)  
  
 actual\_pdf = []  
 for i in range(len(x\_bins) - 1):  
 actual\_pdf.append(1/(s\*((math.pow(e, (x\_bins[i] - u)/(2\*s)) + math.pow(e, (-1)\*(x\_bins[i] - u)/(2\*s))))\*\*2))  
  
 distribution = []  
 for i in range(len(x\_bins) - 1):  
 ctr = count\_in\_interval(simulated\_values, x\_bins[i], x\_bins[i + 1])  
 distribution.append(ctr / len(simulated\_values))  
  
 fig, ax = plt.subplots(1, 1)  
 ax.scatter(x\_bins[:-1], actual\_pdf, color='pink', label='built-in pdf', alpha=1)  
 ax.scatter(x\_bins[:-1], distribution, color='cyan', label='simulated pdf', alpha=0.5)  
  
  
def see\_cdf(u, s, simulated\_values, bins\_number):  
 x\_bins = np.linspace(0, 10, bins\_number)  
  
 actual\_cdf = []  
 for i in range(len(x\_bins) - 1):  
 actual\_cdf.append(1/(1+(math.pow(e, (-1)\*(x\_bins[i] - u)/s))))  
  
 distribution = []  
 for i in range(len(x\_bins) - 1):  
 ctr = count\_smaller(simulated\_values, x\_bins[i])  
 distribution.append(ctr / len(simulated\_values))  
  
 fig, ax = plt.subplots(1, 1)  
 ax.scatter(x\_bins[:-1], actual\_cdf, color='pink', label='built-in pdf', alpha=1)  
 ax.scatter(x\_bins[:-1], distribution, color='cyan', label='simulated pdf', alpha=0.3)  
  
def main():  
 u = 5  
 s = 2  
 N = 10000  
 bins\_number = 100  
 simulated\_values = Logistic(u, s, N)  
 see\_pdf(u, s, simulated\_values, bins\_number)  
 see\_cdf(u, s, simulated\_values, bins\_number)  
 plt.show()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

 Pentru u = 5, s = 2 si N=10000 au fost generate urmatorele grafice: