Preparing True covariance matrix (Diagonal case)

June 14, 2023

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
############################## Importing packages␣
↪####################################

import numpy as np                      # to handle arrays and matrices
import pickle

from scipy.linalg import toeplitz       # to generate toeplitz matrix
from scipy.stats import chi2            # to have chi2 quantiles
from scipy.special import chdtri

import matplotlib.pyplot as plt         # to plot histograms ...
import pandas as pd                     # to handle and create dataframes

import time
import concurrent.futures
import random
import os
from scipy.linalg import toeplitz       # to generate toeplitz matrix

def scalar(A,B):
    """Takes two symmetric matrices A and B of sizes q
    and returns the modified frobenius scalar of A and B
    """
    return(np.trace(A.dot(np.transpose(B)))/A.shape[0])


def norm(A):
    """Takes a symmetric matrix A of sizes q
    and returns the norm of A
    This norm is associated to the modified frobenius scalar
    """
    return np.sqrt(scalar(A,A))

def cov_toep(s, q):
```

```
    """A function that takes the dependency thresholds $s$
    and the dimension $q$ and returns a qxq-toeplitz matrix.
    """
    row = np.array([])
    for k in range(q):
        row = np.append(row, float(s**k))
    return toeplitz(row, row)

# fonction qui reçoit S^2 et calcul alpha2
def alphaaa1(S_2):
    q = len(S_2)
    I_q = np.diag(np.ones(q))
    sigma2 = scalar(S_2,I_q)
    alpha2 = norm(S_2 - sigma2*I_q)**2
    return alpha2

# fonction qui reçoit la diagonale de S^2 et calcul alpha2
def alphaaa2(vec):
    q = len(vec)
    I_q = np.diag(np.ones(q))
    S_2 = np.diag(vec)
    sigma2 = scalar(S_2,I_q)
    alpha2 = norm(S_2 - sigma2*I_q)**2
    return alpha2
```

[3]:
```
q_list = list(pickle.load(open("q_list", "rb")))
n_list = list(pickle.load(open("n_list", "rb")))
```

[4]:
```
[alphaaa1(cov_toep(0.25,q)) for q in q_list]
```

[4]:
```
[0.13048888888888885,
 0.1319111111111111,
 0.13238518518518516,
 0.1326222222222222,
 0.13276444444444443,
 0.13285925925925923,
 0.1329269841269841,
 0.13297777777777772]
```

[5]:
```
[alphaaa1(cov_toep(0.6,q)) for q in q_list]
```

[5]:
```
[1.0898437500000002,
 1.107421875,
 1.1132812499999998,
 1.1162109374999998,
 1.1179687499999995,
 1.119140625,
```

```
 1.1199776785714288,
 1.1206054687499998]
```

[5]:
```
alpha_dep_99 = [alphaaa1(cov_toep(0.99,q)) for q in q_list]
alpha_dep_99
```

[5]:
```
[35.74129922798838,
 55.63558896635379,
 67.1216545130768,
 74.1974025217765,
 78.83310482926932,
 82.04261087020382,
 84.37246235469037,
 86.13181245597147]
```

[29]:
```
[alphaaa1(cov_toep(0.99999,q)) for q in q_list]
```

[29]:
```
[48.98334408090853,
 98.93337298370449,
 148.85011834535834,
 198.73360511617344,
 248.5838582265028,
 298.4009025867656,
 348.184763087464,
 397.9354645991996]
```

[42]:
```
[alphaaa1(cov_toep(0.99999999999999999999999999999999999999999999999,q)) for q in␣
 ↪q_list]
```

[42]:
```
[49.0,
 99.0,
 148.99999999999997,
 199.0,
 248.99999999999997,
 299.0,
 349.00000000000006,
 398.99999999999994]
```

[35]:
```
def sumk(n):
    return 9*sum([10**(-k) for k in range(1,n+1)])

sumk(4)
```

[35]:
```
0.9999
```

[39]:
```
%matplotlib notebook
for i in range(1,5):
```

```
    plt.plot(q_list, [alphaaa1(cov_toep(sumk(i),q)) for q in q_list],␣
 ↪label=sumk(i))
    plt.legend()

plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

Construction de la liste des valeurs de q ainsi que celle des valeurs de n

[2]:
```
q_list = np.arange(50,450,50)
n_list = np.arange(50,225,25)
for q in q_list[:3]:
    for n in n_list[:3]:
        if n <= q : print(q,n,round(q/n,2))
```

```
50 50 1.0
100 50 2.0
100 75 1.33
100 100 1.0
150 50 3.0
150 75 2.0
150 100 1.5
```

[3]:
```
# q et n pour des valeurs fixes de q/n, q/n = 1 and q/n=2
for q in q_list:
    for n in n_list:
        if n <= q and q/n==1 : print("q=",q, ";\t n=",n, ";\t q/n=",round(q/
 ↪n,2))

for q in q_list:
    for n in n_list:
        if n <= q and q/n==2 : print("q=",q, ";\t n=",n, ";\t q/n=",round(q/
 ↪n,2))
```

```
q= 50 ;  n= 50 ;         q/n= 1.0
q= 100 ;         n= 100 ;        q/n= 1.0
q= 150 ;         n= 150 ;        q/n= 1.0
q= 200 ;         n= 200 ;        q/n= 1.0
q= 100 ;         n= 50 ;         q/n= 2.0
q= 150 ;         n= 75 ;         q/n= 2.0
q= 200 ;         n= 100 ;        q/n= 2.0
q= 250 ;         n= 125 ;        q/n= 2.0
q= 300 ;         n= 150 ;        q/n= 2.0
q= 350 ;         n= 175 ;        q/n= 2.0
q= 400 ;         n= 200 ;        q/n= 2.0
```

0.0.1 Un premier jeu de valeurs propres, c'est-à-dire la diagonale de S^2 pour plusieurs valeurs de q

[43]:
```
valeur_propre_collection = np.array([np.random.lognormal(size=q, sigma=1) for q␣
↪in q_list])
print(*[np.std(i) for i in valeur_propre_collection], sep="\n")
```

```
1.4201941405336018
1.6856768290202366
3.033478093188223
1.9908952464491305
2.1203770954628824
1.9017642953156189
2.1699481792155133
1.8746815584901102
```

[69]:
```
np.exp(1/2)
```

[69]:
```
1.6487212707001282
```

[76]:
```
valeur_propre_collection = np.array([np.random.lognormal(size=q, sigma=1) for q␣
↪in q_list])
print(*[np.mean(i) for i in valeur_propre_collection], sep="\n")
print()
print(*[alphaaa2(i) for i in valeur_propre_collection], sep="\n")
```

```
1.598568621587769
1.502368149517207
1.5991760906576895
1.7347952045041808
1.5981867231776667
1.4872663014074528
1.6745850466917787
1.7107207193010003

2.739354565202441
1.5120971568434898
2.7456138677752504
4.589455220254421
3.1311126298432317
3.7137466861123993
3.5206768905095145
3.974672908424324
```

[80]:
```
[alphaaa1(np.exp(1/2)*cov_toep(0.6,q)) for q in np.append(q_list, 10000)]
```

[80]:
```
[2.9625024614846622,
 3.010284759250543,
 3.0262121918391705,
 3.0341759081334834,
 3.0389541379100713,
 3.0421396244277976,
 3.044414971940458,
 3.0461214825749545,
 3.0575892340387645]
```

[83]:
```
[alphaaa1(cov_toep(0.6,q)) for q in np.append(q_list, 10000)]
```

[83]:
```
[1.0898437500000002,
 1.107421875,
 1.1132812499999998,
 1.1162109374999998,
 1.1179687499999995,
 1.119140625,
 1.1199776785714288,
 1.1206054687499998,
 1.12482421875]
```

[97]:
```
print("sigma^2 = ", np.log(1+np.sqrt(1+4*1.11))-np.log(2))
print("sigma = ", np.sqrt(np.log(1+np.sqrt(1+4*1.11))-np.log(2)))
```

```
sigma^2 =  0.5105398103066511
sigma =  0.714520685709414
```

[102]:
```
[alphaaa2(i) for i in np.array([np.random.lognormal(size=q, sigma=0.71) for q␣
 ↪in q_list])]
```

[102]:
```
[0.7182440752076578,
 1.2519371640603563,
 1.1865661330154094,
 1.1694683615709787,
 0.9218263731726548,
 1.1710358700367298,
 1.2508285188659527,
 0.9810482882868599]
```

[84]:
```
[alphaaa1(cov_toep(0.99,q)) for q in np.append(q_list, 10000)]
```

[84]:
```
[35.74129922798838,
 55.63558896635379,
 67.1216545130768,
 74.1974025217765,
 78.83310482926932,
```

```
 82.04261087020382,
 84.37246235469037,
 86.13181245597147,
 98.00752506249833]
```

[85]:
```
[alphaaa1(cov_toep(0.999999,q)) for q in np.append(q_list, 10000)]
```

[85]:
```
[48.99833404081616,
 98.99333432995374,
 148.9850017840501,
 198.97333665305553,
 248.95833918690025,
 298.9400096354942,
 348.91834824872825,
 398.89335527647233,
 9932.665305424556]
```

[103]:
```
[np.log(1+np.sqrt(1+4*alpha))-np.log(2) for alpha in [alphaaa1(cov_toep(0.
 ↪999999,q)) for q in q_list]]
```

[103]:
```
[2.0172623322600325,
 2.34775871359193,
 2.54287503094484,
 2.6820244309608263,
 2.7903263474104416,
 2.879036076904896,
 2.9541832901858363,
 3.019379090874696]
```

[7]:
```
sigmas = [np.sqrt(np.log(1+np.sqrt(1+4*alpha))-np.log(2)) for alpha in␣
 ↪alpha_dep_99]
```

[111]:
```
sigs = [np.log(1+np.sqrt(1+4*alpha))-np.log(2) for alpha in␣
 ↪[alphaaa1(cov_toep(0.999999,q)) for q in q_list]]
[alphaaa2(i) for i in np.array([np.random.lognormal(size=q, sigma=np.
 ↪sqrt(sigs[list(q_list).index(q)])) for q in q_list])]
```

[111]:
```
[11.407622995763822,
 53.682832365992226,
 107.50405645000727,
 55.48862097602159,
 52.4482875562841,
 201.0256035906509,
 92.75975301310413,
 413.5518383735914]
```

L'écart-type des valeurs propre et α^2 de S^2

[77]:
```
for i, q in enumerate(q_list):
    I_q = np.diag(np.ones(q))
    vp = valeur_propre_collection[i]
    S_2 = np.diag(vp)
    sigma2 = scalar(S_2,I_q)
    alpha2 = norm(S_2 - sigma2*I_q)**2
    print("standard deviation is equal to =", np.std(vp), "\t;\talpha2 =",␣
 ↪alpha2)
```

```
standard deviation is equal to = 1.6550995635315844     ;       alpha2 =
2.739354565202441
standard deviation is equal to = 1.2296735976849669     ;       alpha2 =
1.5120971568434898
standard deviation is equal to = 1.656989398811969      ;       alpha2 =
2.7456138677752504
standard deviation is equal to = 2.142301384085447      ;       alpha2 =
4.589455220254421
standard deviation is equal to = 1.7694950211411253     ;       alpha2 =
3.1311126298432317
standard deviation is equal to = 1.9271083742520552     ;       alpha2 =
3.7137466861123993
standard deviation is equal to = 1.8763466871848375     ;       alpha2 =
3.5206768905095145
standard deviation is equal to = 1.9936581724117914     ;       alpha2 =
3.974672908424324
```

0.0.2 Générer plusieurs fois une matrice S^2 de même taille q, et ce pour plusieurs valeurs de q

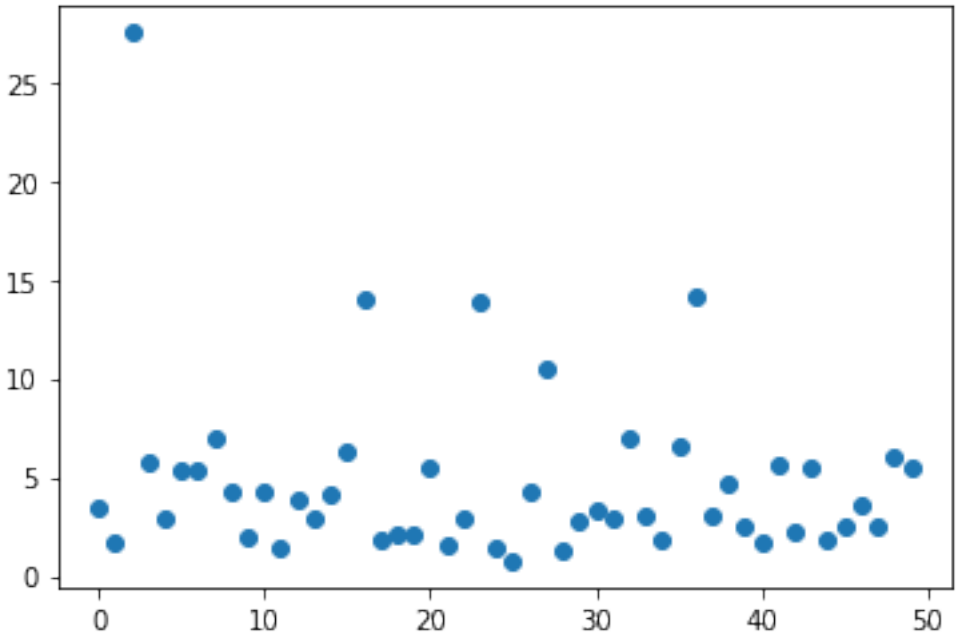
[8]:
```
S_2_dico = {}
for q in q_list[:3]:
    for j, k in enumerate(range(5)):
        vp = np.random.lognormal(size=q, sigma=sigmas[q_list.index(q)])
        if not j : S_2_dico[q] = np.array(vp)
        else : S_2_dico[q] = np.vstack((S_2_dico[q], vp))

print(S_2_dico.keys())
# Calcul des écart-types
print("écart-types des v.p. de chacune des matrices pour q = 50 : ", np.
 ↪std(S_2_dico[list(S_2_dico.keys())[0]], axis=1))
# Calcul des alpha2
print("valeurs de alpha2 de chaque matrice pour q = 50 : ", [alphaaa2(i) for i␣
 ↪in S_2_dico[list(S_2_dico.keys())[0]]])
```

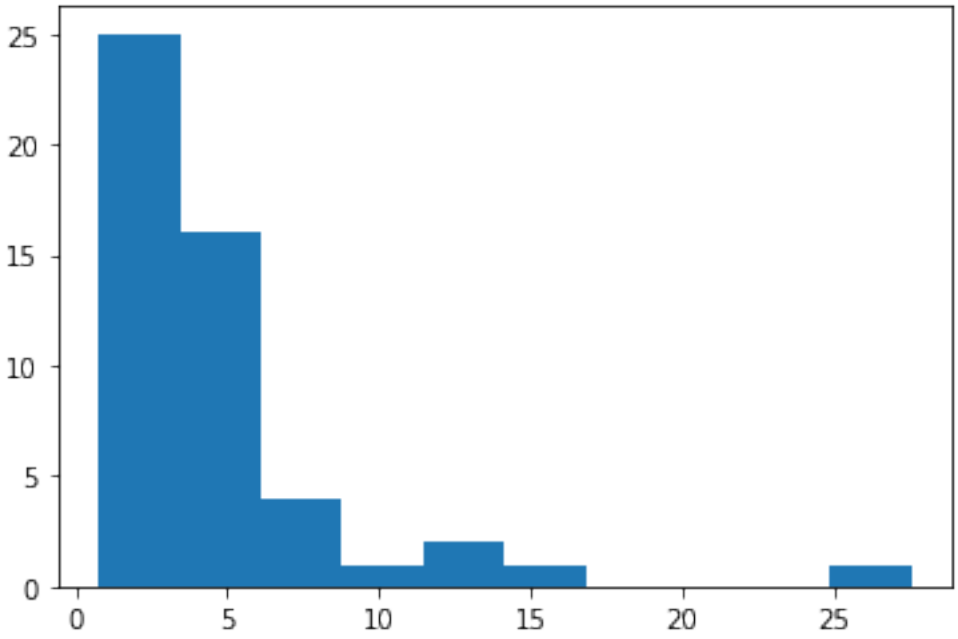
```
dict_keys([50, 100, 150])
écart-types des v.p. de chacune des matrices pour q = 50 :  [2.17970885
2.6945494  5.23377985 3.50609831 7.99837623]
```

valeurs de alpha2 de chaque matrice pour q = 50 : [4.751130677401268, 7.260596447975339, 27.392451556886925, 12.292725344630107, 63.974022240602714]

[81]:
```
%matplotlib inline

# En augmentant le nombre de matrices générées, pour avoir plus de chances de␣
 ↪tomber sur des matrices
# de différentes tailles avec plus ou moins la même valeur de alpha
S_2_dico = {}
for q in q_list[:3]:
    for j, k in enumerate(range(50)):
        vp = np.random.lognormal(size=q, sigma=1)
        if not j : S_2_dico[q] = np.array(vp)
        else : S_2_dico[q] = np.vstack((S_2_dico[q], vp))

print(S_2_dico.keys())
print(np.std(S_2_dico[list(S_2_dico.keys())[0]], axis=1))
print([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[0]]])

plt.plot([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[0]]], "o");plt.
 ↪show()
plt.hist([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[0]]])
plt.show()
```

```
dict_keys([50, 100, 150])
[1.85766833 1.31008252 5.24523567 2.38928045 1.70835806 2.31461636
 2.32493768 2.65176872 2.07145631 1.41798315 2.05121122 1.20174646
 1.97202626 1.72290065 2.01674991 2.51079824 3.7364525  1.37028044
 1.46597727 1.45403855 2.34092691 1.25949164 1.70328272 3.72944607
 1.16862211 0.8705149  2.05183486 3.24873603 1.16038475 1.68591855
 1.82057892 1.72006961 2.64685231 1.76224046 1.36691529 2.55271813
 3.76708526 1.74640932 2.15621444 1.60108441 1.29386386 2.365597
 1.51183815 2.33008273 1.37259019 1.56741065 1.90234254 1.60362104
 2.45729023 2.329759  ]
[3.4509316389134606, 1.7163162206844271, 27.51249725070785, 5.70866105309276,
2.9184872589075046, 5.357448875098937, 5.40533520032533, 7.0318773385404505,
4.290931249894241, 2.0106762008041508, 4.207467485331233, 1.4441945636581928,
3.8888875572390655, 2.9683866612744123, 4.067280212839588, 6.304107816802136,
13.961077277416322, 1.8776684757134527, 2.1490893655162644, 2.1142281127781466,
5.47993879653685, 1.5863191886263428, 2.901172017697523, 13.90876801096804,
1.3656776432118665, 0.7577961904537217, 4.21002630639968, 10.55428578841544,
1.3464927753831537, 2.8423213634896576, 3.3145076218154914, 2.9586394676461967,
7.005827142229907, 3.1054914269595115, 1.8684574088067905, 6.516369833486256,
14.190931329263806, 3.0499455088691145, 4.6492607102465575, 2.563471298638165,
1.674083697686798, 5.596049187428467, 2.285654584886062, 5.429285546066326,
1.8840038378809059, 2.456776138154511, 3.6189071426908184, 2.571600441224485,
6.038275250029967, 5.42777700229048]
```

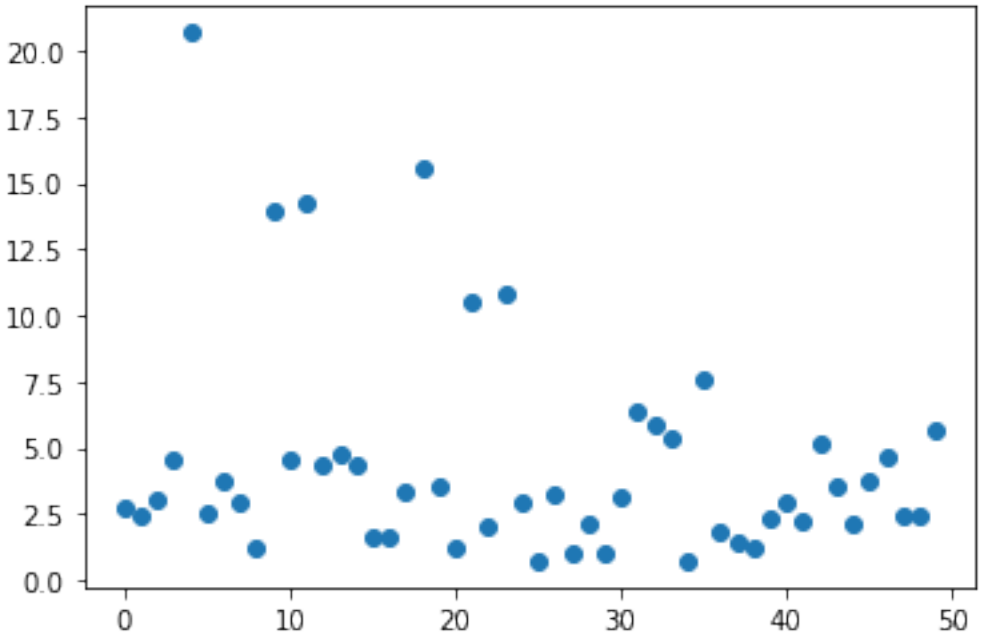


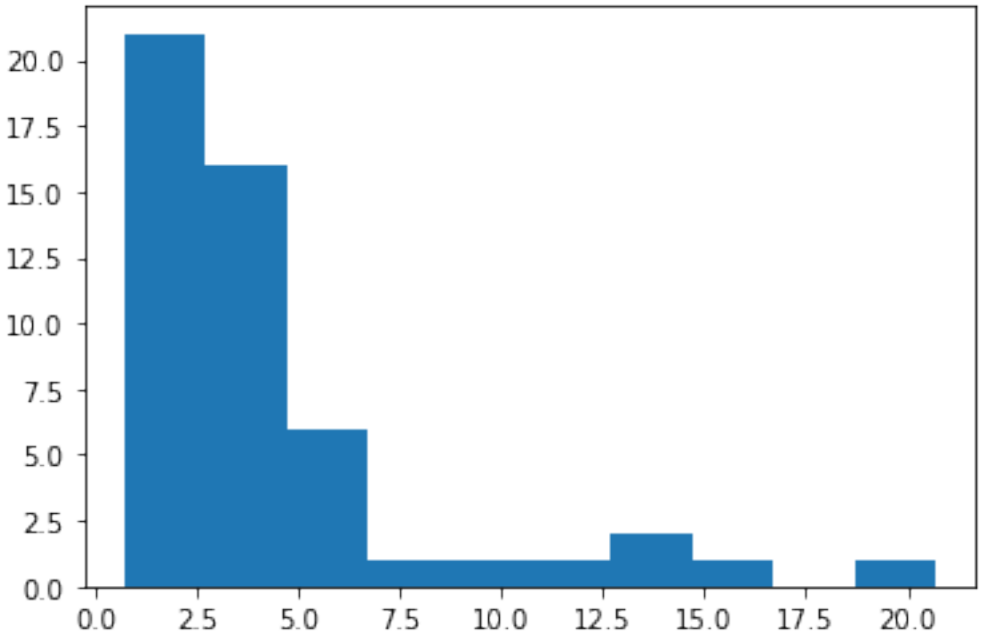


[82]:
```
%matplotlib inline

# En augmentant le nombre de matrices générées, pour avoir plus de chances de␣
 ↪tomber sur des matrices
```

```
# de différentes tailles avec plus ou moins la même valeur de alpha
S_2_dico = {}
for q in q_list[:3]:
    for j, k in enumerate(range(50)):
        vp = np.random.lognormal(size=q, sigma=1)
        if not j : S_2_dico[q] = np.array(vp)
        else : S_2_dico[q] = np.vstack((S_2_dico[q], vp))

print(S_2_dico.keys())
print(np.mean(S_2_dico[list(S_2_dico.keys())[0]], axis=1))
print([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[0]]])

plt.plot([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[0]]], "o");plt.
  ↪show()
plt.hist([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[0]]])
plt.show()
```

```
dict_keys([50, 100, 150])
[1.67190653 1.7060132  1.51060198 1.66222639 1.75651334 1.86100041
 1.63035288 1.61946407 1.34614844 2.34442116 1.85796231 2.66643435
 1.73139622 2.10870105 1.87890566 1.55152923 1.24930908 1.82889779
 1.85151948 1.92250207 1.03195214 1.56942259 1.63005224 2.13156413
 1.78064915 1.23449007 1.5560187  1.35835808 1.55965737 1.34747668
 1.6276308  2.12818907 2.10548301 1.81766911 1.03084293 1.88042407
 1.62747188 1.26928394 1.24430118 1.32425513 1.60705332 1.46514485
 1.66520887 1.3963944  1.53556077 1.59209917 1.79365571 1.29098683
 1.64173018 1.48769769]
[2.6973380255996626, 2.411402163190675, 3.0112255004907347, 4.52223579499691,
20.695162372678684, 2.511551959582993, 3.727079715468362, 2.946976597428291,
1.237885067528377, 13.962071717914078, 4.510470941196516, 14.269166045871538,
4.3749191780117505, 4.716738677735958, 4.356688689615363, 1.6099319348239516,
1.6018442691377361, 3.3009889974777162, 15.526804443878271, 3.518629615785457,
1.2049630922778414, 10.47001723981094, 2.0696994084078097, 10.838281363517842,
2.894535672643468, 0.7147745118527326, 3.2532931762910913, 1.02123382657475,
2.1610230930411394, 1.0528557195205293, 3.13850174370487, 6.36834390990056,
5.8270731989680735, 5.323122121050542, 0.7294644314699376, 7.538856431020331,
1.8205519379103448, 1.3986544266223035, 1.169879692870363, 2.365996222410213,
2.8818143504879727, 2.2711558541793746, 5.141706938431608, 3.537523358577128,
2.124368189036214, 3.699922801068106, 4.689245790235494, 2.4362063872501736,
2.3907808152918086, 5.609270642099803]
```





[11]:
```
# isoler un alpha2 qui a une valeur entre 1.8 et 2.4 compris (plus elle est␣
 ↪proche de 2 et mieux c'est)
isolated_values = np.array([alphaaa2(i) for i in S_2_dico[list(S_2_dico.
 ↪keys())[0]] if 1.9<=alphaaa2(i)<=2.1])
```

```
print(len(isolated_values))
print(isolated_values)
print(isolated_values[np.argmin(isolated_values-2)])
```

```
6
[2.03018378 1.96737679 1.92770153 2.07452294 1.97619214 2.02751925]
1.9277015292561674
```

---

1 Corresponding diagonals with same alpha as non diagonals

1.1 alpha = 1.1 and sigma = 1

[19]:
```
S_2_dico = {}
sig = np.sqrt(np.log(1+np.sqrt(1+4*1.1))-np.log(2))


# Generate 100 times a lognormal vector of size q
for q in q_list:
    for j, k in enumerate(range(1000)):
        vp = np.random.lognormal(size=q, sigma=sigmas[q_list.index(q)])
        if not j : S_2_dico[q] = np.array(vp)
        else : S_2_dico[q] = np.vstack((S_2_dico[q], vp))

isolated_values = {}

for q in q_list:
    # la collection des diagonales des matrices S_2, c-à-d les vecteurs␣
 ↪log-normaux générés
    collection = S_2_dico[q]
    # isoler des valeurs de alpha2 proche de 2
    isolated_values = np.array([alphaaa2(i) for i in collection if␣
 ↪alpha_dep_99[q_list.index(q)]-10<=alphaaa2(i)<=alpha_dep_99[q_list.
 ↪index(q)]+10])
    print("q= ", q)
    print("number of matches :", len(isolated_values))
    print("the best value matching is ", isolated_values[np.argmin(np.
 ↪abs(isolated_values-alpha_dep_99[q_list.index(q)]))])
    print("the index of matching value", np.where(np.array([alphaaa2(i) for i␣
 ↪in collection])==
```

```
                                                    isolated_values[np.argmin(np.
↪abs(isolated_values-sigmas[q_list.index(q)]))])[0][0])
    print()
```

```
q=  50
number of matches : 135
the best value matching is  35.81322639270601
the index of matching value 112

q=  100
number of matches : 90
the best value matching is  55.5057610930806
the index of matching value 633

q=  150
number of matches : 81
the best value matching is  67.16015515618975
the index of matching value 764

q=  200
number of matches : 83
the best value matching is  74.03142454338604
the index of matching value 459

q=  250
number of matches : 96
the best value matching is  78.71857289577437
the index of matching value 351

q=  300
number of matches : 77
the best value matching is  82.06276498069886
the index of matching value 377

q=  350
number of matches : 83
the best value matching is  84.38759586673962
the index of matching value 459

q=  400
number of matches : 83
the best value matching is  86.29469286163241
the index of matching value 902
```

[20]:
```
for q in q_list:
```

```
    # la collection des diagonales des matrices S_2, c-à-d les vecteurs␣
 ↪log-normaux générés
    collection = S_2_dico[q]
    # isoler des valeurs de alpha2 proche de 2
    isolated_values = np.array([alphaaa2(i) for i in collection if␣
 ↪alpha_dep_99[q_list.index(q)]-10<=alphaaa2(i)<=alpha_dep_99[q_list.
 ↪index(q)]+10])
    match_value = isolated_values[np.argmin(np.
 ↪abs(isolated_values-alpha_dep_99[q_list.index(q)]))]
    match_value_idx =  np.where(np.array([alphaaa2(i) for i in␣
 ↪collection])==match_value)[0][0]
    print(alphaaa2(S_2_dico[q][match_value_idx]))
```

```
35.81322639270601
55.5057610930806
67.16015515618975
74.03142454338604
78.71857289577437
82.06276498069886
84.38759586673962
86.29469286163241
```

[21]:
```
[alphaaa1(cov_toep(0.99,q)) for q in q_list]
```

[21]:
```
[35.74129922798838,
 55.63558896635379,
 67.1216545130768,
 74.1974025217765,
 78.83310482926932,
 82.04261087020382,
 84.37246235469037,
 86.13181245597147]
```

[23]:
```
vp_collection_d099 = []

for q in q_list:
    # la collection des diagonales des matrices S_2, c-à-d les vecteurs␣
 ↪log-normaux générés
    collection = S_2_dico[q]
    # isoler des valeurs de alpha2 proche de 2
    isolated_values = np.array([alphaaa2(i) for i in collection if␣
 ↪alpha_dep_99[q_list.index(q)]-10<=alphaaa2(i)<=alpha_dep_99[q_list.
 ↪index(q)]+10])
    match_value = isolated_values[np.argmin(np.
 ↪abs(isolated_values-alpha_dep_99[q_list.index(q)]))]
    match_value_idx =  np.where(np.array([alphaaa2(i) for i in␣
 ↪collection])==match_value)[0][0]
```

```
    vp_collection_d099.append(S_2_dico[q][match_value_idx])
```

[24]:
```
pickle.dump(vp_collection_d099, open("vp_collection_d099", "wb"))
```

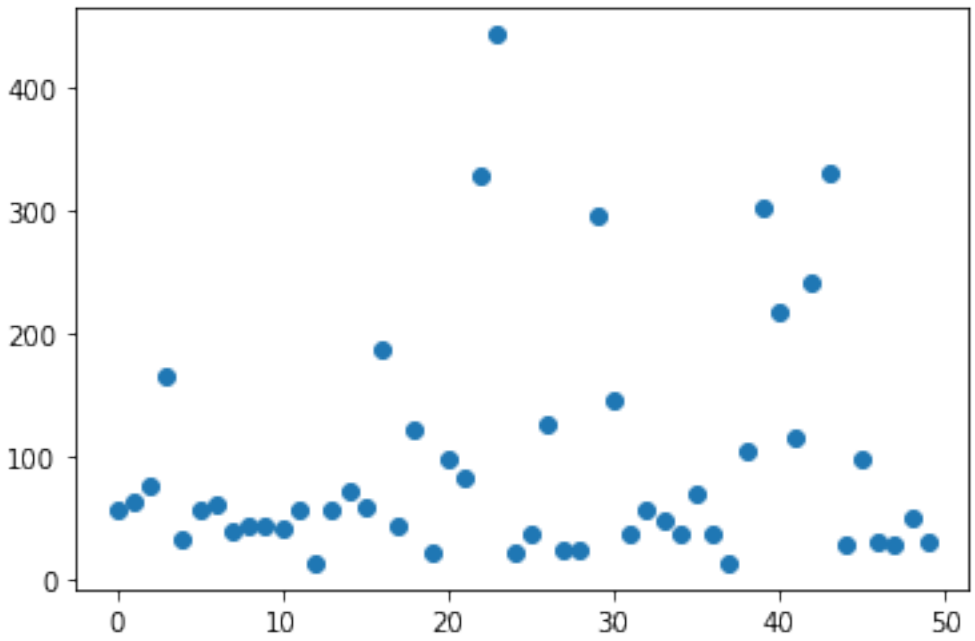
[ ]:
```

```

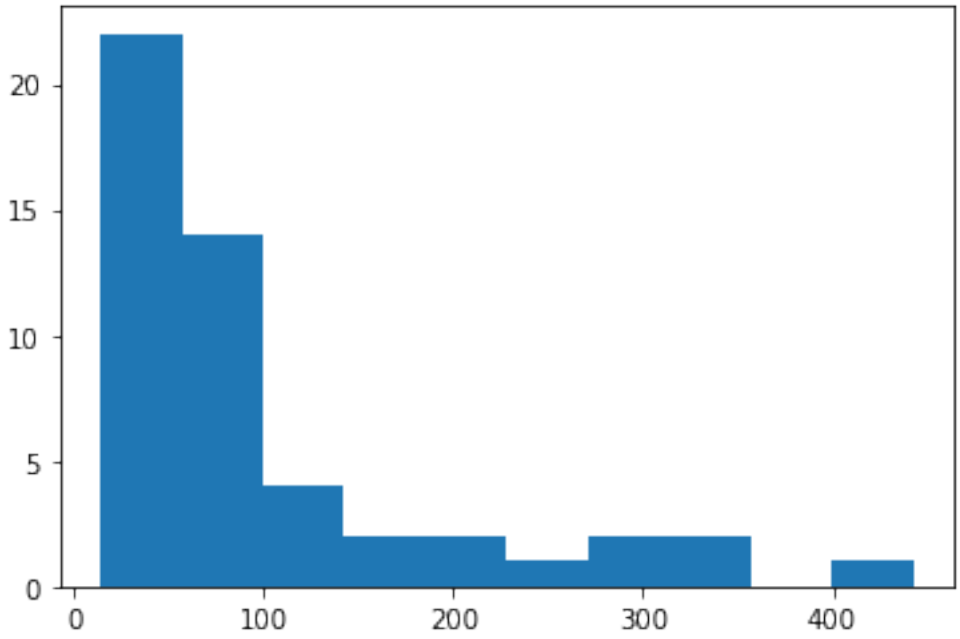
[ ]:
```

```

1.2 alpha = q-1 and sigma = 1

[139]:
```
%matplotlib inline

# En augmentant le nombre de matrices générées, pour avoir plus de chances de␣
 ↪tomber sur des matrices
# de différentes tailles avec plus ou moins la même valeur de alpha
S_2_dico = {}
for q in q_list[:4]:
    for j, k in enumerate(range(50)):
        vp = np.random.lognormal(size=q, sigma=np.sqrt(np.log(1+np.
 ↪sqrt(1+4*(q-1)))-np.log(2)))
        if not j : S_2_dico[q] = np.array(vp)
        else : S_2_dico[q] = np.vstack((S_2_dico[q], vp))

print(S_2_dico.keys())
# print(np.mean(S_2_dico[list(S_2_dico.keys())[0]], axis=1))
# print([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[0]]])

plt.plot([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[2]]], "o");plt.
 ↪show()
plt.hist([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[2]]])
plt.show()
```

```
dict_keys([50, 100, 150, 200])
```





[142]:
```
S_2_dico = {}
sig = np.sqrt(np.log(1+np.sqrt(1+4*1.1))-np.log(2))
```

```
# Generate 100 times a lognormal vector of size q
for q in q_list:
    for j, k in enumerate(range(1000)):
        vp = np.random.lognormal(size=q, sigma=np.sqrt(np.log(1+np.
 ↪sqrt(1+4*(q-1)))-np.log(2)))
        if not j : S_2_dico[q] = np.array(vp)
        else : S_2_dico[q] = np.vstack((S_2_dico[q], vp))

isolated_values = {}

for q in q_list:
    # la collection des diagonales des matrices S_2, c-à-d les vecteurs␣
 ↪log-normaux générés
    collection = S_2_dico[q]
    # isoler des valeurs de alpha2 proche de 2
    isolated_values = np.array([alphaaa2(i) for i in collection if q-2.
 ↪5<=alphaaa2(i)<=q+1])
    print("q= ", q)
    print("number of matches :", len(isolated_values))
    print("the best value matching is ", isolated_values[np.argmin(np.
 ↪abs(isolated_values-(q-1)))])
    print("the index of matching value", np.where(np.array([alphaaa2(i) for i␣
 ↪in collection])==
                                                  isolated_values[np.argmin(np.
 ↪abs(isolated_values-(q-1)))])[0][0])
    print()
```

```
q=  50
number of matches : 12
the best value matching is  49.039571398609425
the index of matching value 195

q=  100
number of matches : 9
the best value matching is  98.92339030289976
the index of matching value 635

q=  150
number of matches : 1
the best value matching is  148.28374992358252
the index of matching value 422

q=  200
number of matches : 2
the best value matching is  198.15543030436604
the index of matching value 39
```

```
q=  250
number of matches : 3
the best value matching is  248.65334620433111
the index of matching value 519

q=  300
number of matches : 4
the best value matching is  298.76240625400715
the index of matching value 854

q=  350
number of matches : 2
the best value matching is  347.65327277210804
the index of matching value 188

q=  400
number of matches : 3
the best value matching is  397.90154545111807
the index of matching value 888
```

[143]:
```
for q in q_list:
    # la collection des diagonales des matrices S_2, c-à-d les vecteurs␣
 ↪log-normaux générés
    collection = S_2_dico[q]
    # isoler des valeurs de alpha2 proche de 2
    isolated_values = np.array([alphaaa2(i) for i in collection if q-2.
 ↪5<=alphaaa2(i)<=q+1])
    match_value = isolated_values[np.argmin(np.abs(isolated_values-(q-1)))]
    match_value_idx =  np.where(np.array([alphaaa2(i) for i in␣
 ↪collection])==match_value)[0][0]
    print(alphaaa2(S_2_dico[q][match_value_idx]))
```

```
49.039571398609425
98.92339030289976
148.28374992358252
198.15543030436604
248.65334620433111
298.76240625400715
347.65327277210804
397.90154545111807
```

[144]:
```
vp_collection_d0999999 = []

for q in q_list:
    # la collection des diagonales des matrices S_2, c-à-d les vecteurs␣
 ↪log-normaux générés
```

```
    collection = S_2_dico[q]
    # isoler des valeurs de alpha2 proche de 2
    isolated_values = np.array([alphaaa2(i) for i in collection if q-2.
↪5<=alphaaa2(i)<=q+1])
    match_value = isolated_values[np.argmin(np.abs(isolated_values-(q-1)))]
    match_value_idx =  np.where(np.array([alphaaa2(i) for i in␣
↪collection])==match_value)[0][0]
    vp_collection_d0999999.append(S_2_dico[q][match_value_idx])
```

[145]:
```
pickle.dump(vp_collection_d0999999, open("vp_collection_d0999999", "wb"))
```

[147]:
```
teest = pickle.load(open("vp_collection_d0999999", "rb"))
print(teest[0])
print()
print(teest[-1][:60])
```

```
[3.18894520e-01 4.52994082e+01 1.28592317e+01 2.11195179e+00
 6.62357112e-01 5.82053682e+00 2.09626410e+00 4.00085785e-01
 2.93408094e-01 5.50537856e-02 5.57495114e-01 1.42318825e+00
 2.38894363e-01 7.46870598e-01 5.80652859e-01 4.62184826e+00
 5.47841374e-02 1.03344591e+00 3.54343430e-02 2.30767499e-01
 2.36480067e+00 4.55393557e+00 2.15870138e-01 2.87290090e+00
 1.02649741e+00 3.63394217e-01 2.22914488e-01 6.05129669e+00
 1.96964040e+00 4.55614851e-01 4.88561910e+00 6.04134953e+00
 1.51970862e+00 2.62885405e+00 1.46985040e+00 1.41892317e+00
 2.16914151e+01 3.38453119e+00 3.17823168e+00 5.20661610e+00
 5.92637439e-01 1.72292013e+00 1.20364968e+00 6.56391230e-01
 1.35797878e+00 6.04626546e-01 5.88333073e-01 1.99814061e+00
 2.86903237e-02 1.08931360e+00]

[5.30090161e-01 2.05928126e-02 1.53175755e+01 5.08827010e+00
 1.11098454e+00 6.11606243e-01 8.03896138e-02 4.89393945e+00
 5.54033762e-02 3.71208941e-01 1.03697852e+01 3.93740149e+00
 1.03993944e-01 2.02702807e+00 9.06405468e-01 1.07096914e+00
 1.27904042e+01 3.08298096e+00 1.11843784e+00 4.11752161e-02
 3.66789524e-01 1.30645751e+01 1.63053080e-01 1.96597245e-01
 2.80856477e+01 4.14573372e+00 5.61039077e-01 4.10376311e-01
 8.52219826e-01 1.14851139e+01 1.17459316e+00 3.28005873e-01
 2.70039224e+00 7.21305841e-02 7.31892012e-01 1.55917136e+00
 7.91070563e+00 2.78394447e+01 1.22230475e+00 5.07336520e-01
 2.97742639e+00 5.47318806e-01 4.15497662e-01 1.72709002e+00
 1.73390614e+00 2.67005628e+00 2.89026854e+00 1.85904307e+01
 8.84082232e-02 5.22422051e-01 4.80362722e-01 1.91427044e+00
 2.65167880e+00 4.55564530e-04 5.15994381e+00 9.75171671e-01
 6.33663989e+00 2.58121325e-01 2.12045871e+00 5.11944123e+00]
```

[148]:
```
[alphaaa2(i) for i in teest]
```

[148]:
```
[49.039571398609425,
 98.92339030289976,
 148.28374992358252,
 198.15543030436604,
 248.65334620433111,
 298.76240625400715,
 347.65327277210804,
 397.90154545111807]
```

Stocker S^2 Vérfier d'abord que c'est le bon α^2, c'est-à-dire que leur α^2 sont proches

[ ]:

[173]:
```
for q in q_list:
    # la collection des diagonales des matrices S_2, c-à-d les vecteurs␣
 ↪log-normaux générés
    collection = S_2_dico[q]
    # isoler des valeurs de alpha2 proche de 2
    isolated_values = np.array([alphaaa2(i) for i in collection if 1.
 ↪8<=alphaaa2(i)<=2.4])
    match_value = isolated_values[np.argmin(np.abs(isolated_values-2))]
    match_value_idx =  np.where(np.array([alphaaa2(i) for i in␣
 ↪collection])==match_value)[0][0]
    print(alphaaa2(S_2_dico[q][match_value_idx]))

for q in q_list:
    # la collection des diagonales des matrices S_2, c-à-d les vecteurs␣
 ↪log-normaux générés
    collection = S_2_dico[q]
    # isoler des valeurs de alpha2 proche de 5
    isolated_values = np.array([alphaaa2(i) for i in collection if 4.
 ↪5<=alphaaa2(i)<=5.5])
    match_value = isolated_values[np.argmin(np.abs(isolated_values-4))]
    match_value_idx =  np.where(np.array([alphaaa2(i) for i in␣
 ↪collection])==match_value)[0][0]
    print(alphaaa2(S_2_dico[q][match_value_idx]))
```

```
1.993066365613705
1.995667988032773
1.9956588156541615
2.0052626731249132
2.151484609828197
2.227792287784347
2.084701635151049
1.948271544049899
4.559315813319646
4.541632249194627
```

```
4.521032246785799
4.504829184413636
4.536351070409249
4.5618959138497
4.5385379118824325
4.525812841621547
```

Finally, retained values of S_2 for $\alpha^2 \simeq 2$ and $\sigma = 1$

[174]:
```
vp_2_collection = []
vp_4_collection = []

for q in q_list:
    # la collection des diagonales des matrices S_2, c-à-d les vecteurs␣
 ↪log-normaux générés
    collection = S_2_dico[q]
    # isoler des valeurs de alpha2 proche de 2
    isolated_values = np.array([alphaaa2(i) for i in collection if 1.
 ↪8<=alphaaa2(i)<=2.4])
    match_value = isolated_values[np.argmin(np.abs(isolated_values-2))]
    match_value_idx =  np.where(np.array([alphaaa2(i) for i in␣
 ↪collection])==match_value)[0][0]
    vp_2_collection.append(S_2_dico[q][match_value_idx])

for q in q_list:
    # la collection des diagonales des matrices S_2, c-à-d les vecteurs␣
 ↪log-normaux générés
    collection = S_2_dico[q]
    # isoler des valeurs de alpha2 proche de 5
    isolated_values = np.array([alphaaa2(i) for i in collection if 4.
 ↪5<=alphaaa2(i)<=5.5])
    match_value = isolated_values[np.argmin(np.abs(isolated_values-4))]
    match_value_idx =  np.where(np.array([alphaaa2(i) for i in␣
 ↪collection])==match_value)[0][0]
    vp_4_collection.append(S_2_dico[q][match_value_idx])
```

[175]:
```
print(vp_2_collection[0])
print()

print(vp_4_collection[-1])
```

```
[2.46711588 0.86018508 1.07495414 1.6800156  4.26761975 1.08234407
 1.95240009 0.69453705 0.98142713 1.63324193 1.17184247 0.61900364
 3.29864058 0.83575721 0.18153526 0.96019059 3.80345511 3.31687954
 0.46826784 4.30019118 0.3889938  3.63735567 0.19013967 0.29697951
 0.385916   0.27761594 0.40523776 0.86949066 6.22058018 0.64186503
 4.39385353 1.00334181 0.34837744 0.36899693 1.90420533 1.84400636
 0.35764925 1.51948023 0.52194299 3.07349145 0.71847765 1.91378266
```

```
 0.22104288 0.56835563 0.40411491 3.13814471 0.47570603 0.56686326
 0.27495282 0.09545525]

[ 2.88273611  8.99208315  1.11764201  0.87570785  0.79117313  1.24937197
  0.29820935  0.55315273  0.2633677   1.72642518  1.56511554  1.77229498
  0.95223441  0.59744304  0.70987093  2.7181974   0.30290939  0.86312335
  0.93986214  0.5957847   0.39863364  1.38265376  2.5862597   2.27304526
  0.30726557  1.82943198  3.23645525  0.21410839  0.5917582   1.18941942
  0.66050092  1.42400653  1.53105144  0.39830927  0.52776993  4.85207811
  0.63551118  0.23402299  0.82987269  0.52661759  0.98577458  0.43378664
  1.01854551  2.6298592   0.20004715  1.35174685  6.78224315  2.2702525
  0.77336281  3.43492525  0.56649981  0.29809672  0.61583065  0.21523625
  3.53101517  3.80192128  1.02901598  0.94074963  1.37061396  0.30923788
  0.22509737  0.39532704  3.12649286  0.73426481  1.04557792  1.12205736
  0.10854319  0.87600398  8.718843    0.10703236  0.46374219  4.95748809
  0.54182315  0.17912944  1.20183189  0.2944003   1.17943112  0.65042812
  1.14423032  0.09981442  4.13009194  2.35171432  0.17925835  2.4487281
  0.14813321  0.56395212  0.7379245   1.22156838  7.51469096  0.44945866
  1.26055772  0.66473819  0.66926182  0.79381452  2.22552157  1.21400991
  3.039677    0.54139352  0.91719545  1.20958314  1.27469419  0.23100284
  0.16521717  0.40144546  0.45663744  0.86251291  0.5914605   1.6403577
  0.3081625   2.98028583 12.26652303  0.7564454   0.25097451  0.11303424
 11.70001337  2.05562897  2.06058643  2.91027901  2.85092922  0.95168644
  0.22000572  4.08263566  0.40123175  1.24632736  0.71037083  0.72133807
  1.01819977  0.80482742  2.36837436  1.67429622  1.20351163  0.30238058
  0.50730632  1.6205946   0.21260827 10.54631594  0.40706071  0.92320389
  0.83569614  0.39917804  1.26931967  0.89817294  0.49180714  0.66092416
  4.43694083  1.29262094  0.69962895  0.99565392  0.41100401  0.82512633
  1.68922556  1.84474014  3.83514278  1.19294721  0.69388235  0.4183507
  0.88007442  4.77601505  1.93883074  0.2489309   1.91924729  1.077636
  1.42497962  1.40905579  0.12869575  3.488722    0.58564882  0.69447781
  0.17040678  0.36552987  2.85570906  0.9526082   0.99735797  7.41532526
  1.50650892  0.73403882  0.31991238  1.1189135   0.2158048   0.50900238
  0.43604764  0.99963355  3.17913906  1.34012965  0.75563829  1.64802363
  0.71482008  3.96382998  6.261696    0.47759094  0.57993613 13.08998129
  2.74937385  1.176841    1.65296279  1.93321923  0.74482626  1.15673494
  0.55620869  5.22303462  1.61381359  0.34362109  1.68799625  0.408803
  3.07297917  1.40725192  1.3334716   3.04064258  0.27322238  1.27249659
  6.51900291  1.65571191  5.42885649  0.44151442  0.84394204  8.62553657
  1.04413579  1.78012912  2.70347596  0.89121576  1.90658572  1.311706
  2.83367395  2.83389962  5.14726346  0.26908438  1.01927738  1.66585509
  0.30490322  1.44103858  0.77372354  3.03728292  0.98869916  1.39628229
  0.57850613  0.35585656  1.20451697  5.99525867  0.21216364  0.19095847
  1.78512116  1.13562561  0.70817978  3.79679662  3.15211242  0.16185976
  0.31058289  4.59783664  1.03874982  1.43308583  0.52290046  0.244939
  2.39060795  4.33731919  0.87586201  0.5053883   0.32100112  1.99929776
  0.49845753  0.77648932  0.22182652  0.45327423  3.56765378  1.82909001
  0.18672977  1.37373294  0.66533775  0.54823796  3.07922588  4.41007219
```

```
 0.94589203  0.25994898  4.18580921 10.3284467   2.00980659  3.03459873
 1.21717922  0.80083349  1.61011667  3.75113184  0.68466408  1.94671979
 0.78885099  0.55235835 11.3081546   2.39181298  1.61998329  2.02540591
 0.2282086   0.95016092  1.15172625  0.29245008  1.44646518  3.16392212
 2.08947203  1.36974648  2.21311226  3.66486673  0.72058682  0.77642258
12.58814254  1.35745982  2.19899071  0.73243479  1.68207898  1.49695407
 7.13187365  1.57409885  1.87819451  1.02505361  0.46930104  2.94879292
 0.99422807  1.70832886  1.31890595  0.46721899  1.58036752  5.29996999
 2.51820613  0.91037962  0.71870303  0.46851184  1.70777759  2.7700332
 0.29896338  3.43451461  0.8560999   1.38680206  3.51347656  0.72987143
 0.61424231  1.98154566  0.54425674  2.65427982  0.73761184  0.45786208
 0.17117716  0.33885936  3.09609082  0.39808511  0.78992548  0.56797088
 0.99493316  0.74052511  0.54104939  1.14439679  0.82147251  2.67022454
 1.47711046  0.8664686   1.28185419  2.59496444  0.20317185  0.96377224
 0.09638001  2.55597891  0.37870166  3.84725991  1.43893117  2.04936028
 0.77221187  0.25561036  0.95395199  1.73068778  0.57265335  1.04284491
 0.0711644   1.10684648  2.71052943  0.4401247   0.85613569  2.70279205
 0.61316393  0.6657534   1.05960226  7.09317728  0.75478516  0.54214715
 9.78059499  1.68959686  0.80017736  0.36489266  2.54585715  1.21144603
 2.10032465  3.06632289  0.85037003  0.5649282   0.8741009   1.82706857
 0.30016054  0.88188046  5.97802962  2.42903064  2.5427228   1.52056158
 0.4839625  13.18989223  1.36092643  4.5081549 ]
```

[176]:
```
pickle.dump(vp_2_collection, open("vp_2_collection_sigma1", "wb"))
pickle.dump(vp_4_collection, open("vp_4_collection_sigma1", "wb"))
```

[177]:
```
teeeest = pickle.load(open("vp_2_collection_sigma1", "rb"))
len(teeeest)
```

[177]: 8

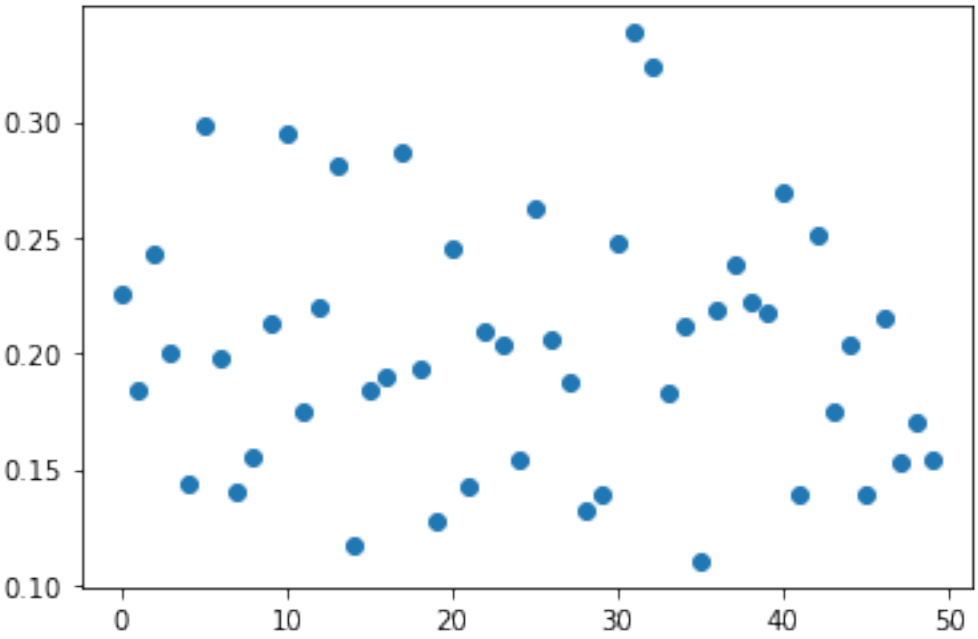
1.3 Now for $\sigma = 0.2$

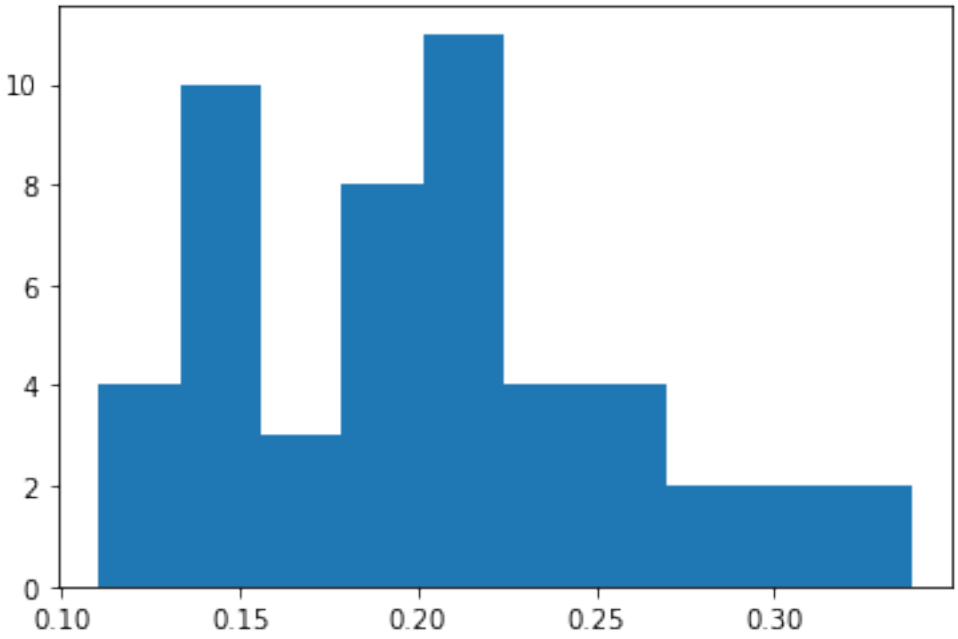
On regarde d'abord comment les valeurs de α^2 sont distribuées pour en sélectionner un même alpha pour toutes les valeurs de q.

[178]:
```
%matplotlib inline

# En augmentant le nombre de matrices, pour avoir plus de chances de tomber sur␣
 ↪des matrices de différentes
# tailles avec plus ou moins la même valeur de alpha
S_2_dico = {}
for q in q_list[:3]:
    for j, k in enumerate(range(50)):
        vp = np.random.lognormal(size=q, sigma=0.4)
        if not j : S_2_dico[q] = np.array(vp)
        else : S_2_dico[q] = np.vstack((S_2_dico[q], vp))
```

```
print(S_2_dico.keys())
print(np.std(S_2_dico[list(S_2_dico.keys())[0]], axis=1))
print([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[0]]])

plt.plot([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[0]]], "o");plt.
  ↪show()
plt.hist([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[0]]])
plt.show()
```

```
dict_keys([50, 100, 150])
[0.47491395 0.42914929 0.49278067 0.44766501 0.37855787 0.54653258
 0.4450773  0.37447784 0.39351694 0.46102555 0.54255136 0.41824437
 0.46836856 0.53040365 0.3418025  0.42846818 0.43592334 0.53527777
 0.44018192 0.35635718 0.495133   0.37808082 0.45752727 0.45087038
 0.39297111 0.51266559 0.45415233 0.43344942 0.36415766 0.37353635
 0.49746833 0.58171805 0.56843838 0.42704773 0.46069014 0.33211083
 0.46773067 0.48816468 0.47157041 0.46679367 0.51920569 0.37277395
 0.50072631 0.41761478 0.45145534 0.37263255 0.46373305 0.39069392
 0.41224395 0.39196912]
[0.2255432590994334, 0.1841691110286184, 0.24283278644182552,
0.20040396099781158, 0.14330606180352634, 0.2986978630133279,
0.19809380126286252, 0.14023365352135533, 0.15485558350853706,
0.21254455329041796, 0.2943619819408849, 0.17492835368419105,
0.21936910487079128, 0.28132803684421026, 0.11682894607034014,
0.1835849812505453, 0.19002916239796572, 0.2865222921117168,
0.19376012464407963, 0.12699044285896444, 0.24515668387456938,
0.1429451035878666, 0.20933119852764132, 0.20328410027533841,
0.15442629456822116, 0.2628260030603216, 0.20625434211492455,
0.1878784011285609, 0.13261080427692812, 0.1395294058806551,
0.24747473805694442, 0.3383958919055213, 0.323122195067223, 0.18236976201677899,
0.21223540774956937, 0.1102976052712034, 0.2187719815791158,
0.23830475590097974, 0.22237865034124069, 0.2178963259339448,
0.2695745444769828, 0.13896041574553322, 0.2507268354463331,
0.17440210697927228, 0.2038119195889201, 0.13885501893669713,
0.21504834594624625, 0.15264173907017844, 0.1699450730474996,
0.15363979377726683]
```





Une valeur entre 0.15 et 0.25 ferait l'affaire

[179]:
```
S_2_dico = {}

# Generate 100 times a lognormal vector of size q
```

```
for q in q_list:
    for j, k in enumerate(range(100)):
        vp = np.random.lognormal(size=q, sigma=0.4)
        if not j : S_2_dico[q] = np.array(vp)
        else : S_2_dico[q] = np.vstack((S_2_dico[q], vp))

isolated_values = {}

for q in q_list:
    # la collection des diagonales des matrices S_2, c-à-d les vecteurs␣
 ↪log-normaux générés
    collection = S_2_dico[q]
    # isoler des valeurs de alpha2 proche de 2
    isolated_values = np.array([alphaaa2(i) for i in collection if 0.
 ↪15<=alphaaa2(i)<=0.25])
    print("q= ", q)
    print("number of matches :", len(isolated_values))
    print("the best value matching is ", isolated_values[np.argmin(np.
 ↪abs(isolated_values-0.2))])
    print("the index of matching value", np.where(np.array([alphaaa2(i) for i␣
 ↪in collection])==
                                                  isolated_values[np.argmin(np.
 ↪abs(isolated_values-0.2))])[0][0])
    print()
```

```
q=  50
number of matches : 61
the best value matching is  0.1993876040192482
the index of matching value 92

q=  100
number of matches : 74
the best value matching is  0.20008224485626525
the index of matching value 24

q=  150
number of matches : 78
the best value matching is  0.19913827325065844
the index of matching value 14

q=  200
number of matches : 84
the best value matching is  0.19859731814079046
the index of matching value 13

q=  250
number of matches : 95
```

```
the best value matching is  0.19991755815079565
the index of matching value 72

q=  300
number of matches : 92
the best value matching is  0.19986192805084135
the index of matching value 93

q=  350
number of matches : 94
the best value matching is  0.20047692951573898
the index of matching value 64

q=  400
number of matches : 97
the best value matching is  0.20000250609983625
the index of matching value 15
```

2 Attention sigma4 signifie $\sigma = 0.4 = 2/5$

[180]:
```
vp_collection_sigma4 = []

for q in q_list:
    # la collection des diagonales des matrices S_2, c-à-d les vecteurs␣
 ↪log-normaux générés
    collection = S_2_dico[q]
    # isoler des valeurs de alpha2 proche de 2
    isolated_values = np.array([alphaaa2(i) for i in collection if 0.
 ↪15<=alphaaa2(i)<=0.25])
    match_value = isolated_values[np.argmin(np.abs(isolated_values-0.2))]
    match_value_idx =  np.where(np.array([alphaaa2(i) for i in␣
 ↪collection])==match_value)[0][0]
    vp_collection_sigma4.append(S_2_dico[q][match_value_idx])

pickle.dump(vp_collection_sigma4, open("vp_collection_sigma4", "wb"))
```

[181]:
```
[alphaaa2(i) for i in vp_collection_sigma4]
```

[181]:
```
[0.1993876040192482,
 0.20008224485626525,
 0.19913827325065844,
 0.19859731814079046,
 0.19991755815079565,
 0.19986192805084135,
 0.20047692951573898,
```

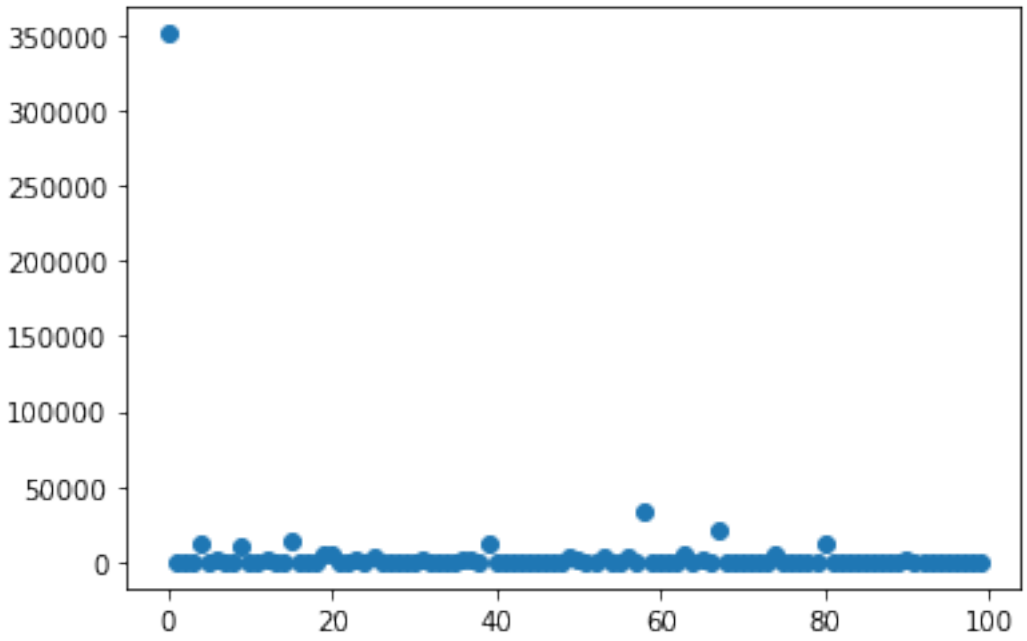
0.20000250609983625]

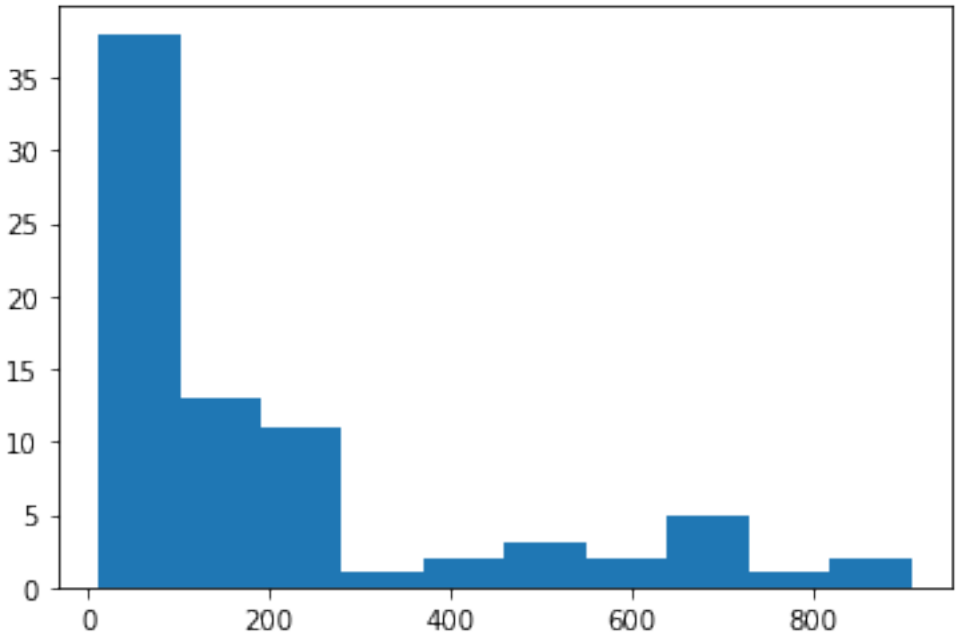
Maintenant pour $sigma = 2$

[182]:
```
%matplotlib inline

# En augmentant le nombre de matrices, pour avoir plus de chances de tomber sur␣
 ↪des matrices de différentes
# tailles avec plus ou moins la même valeur de alpha
S_2_dico = {}
for q in q_list[:3]:
    for j, k in enumerate(range(100)):
        vp = np.random.lognormal(size=q, sigma=2)
        if not j : S_2_dico[q] = np.array(vp)
        else : S_2_dico[q] = np.vstack((S_2_dico[q], vp))

print(S_2_dico.keys())
print(np.std(S_2_dico[list(S_2_dico.keys())[0]], axis=1))
print([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[0]]])

plt.plot([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[0]]], "o");plt.
 ↪show()
plt.hist([alphaaa2(i) for i in S_2_dico[list(S_2_dico.keys())[0]] if␣
 ↪alphaaa2(i)<=1000])
plt.show()
```

```
dict_keys([50, 100, 150])
[592.50754332   3.7547735    3.79223301   5.32471136 106.937549
  23.20899381  33.23686393  14.58707085   8.418266   106.68917594
   8.27431734   4.08041087  32.34818272   7.62083929  25.28880286
 121.88591791   9.45128529   5.04409599  12.55369088  66.77411501
  65.78064664  15.06018716  14.37751477  27.66071961   8.76433868
  58.39358861  12.75269307   8.08751467  25.26193859  14.26846786
  20.30896306  35.43061202   9.82533345  16.46011239   6.22097723
  16.35985668  46.96018652  30.11912253   6.15500898 110.24026285
   9.26578988  20.10950174   5.86225746  14.17706177   6.88054743
   3.45364399   8.46503354  12.39998798   5.77486568  52.02289303
  41.970219     8.17329262  12.5186808   64.82335451   7.66455047
   6.36513417  60.91575388   4.46658218 182.14937157  22.30358764
  25.81001308   4.2528063   23.51305988  71.35891123  14.92669968
  29.69875632   8.50735257 144.37206779   6.42711001   7.79539922
  12.2127071    8.44171253   8.19007605  26.11045615  74.72665621
   3.70388871   6.01724381  14.18523992  11.18558136   6.16191765
 109.60049395   4.24039127  23.1530037   25.87037811   7.50468467
   3.99937841  11.44957421  12.57932675  16.23416226  11.13186917
  32.1353462   12.83658062  13.24075806   6.89938553  14.72746441
  17.36951124  12.18069751   7.86128746  25.95504672  13.03053434]
```

[351065.18889074895, 14.098324041311612, 14.381031201320486, 28.35255109558771,
11435.639386989906, 538.6573935367207, 1104.6891240314253, 212.78263599407353,
70.86720236880217, 11382.580262594814, 68.4643275017636, 16.64975285458348,
1046.4049254081056, 58.077191431505405, 639.5235500824294, 14856.176985138783,
89.32679368721523, 25.44290436623556, 157.59515462739003, 4458.782435036714,
4327.093472963376, 226.80923738378635, 206.71293102911346, 765.1154093809894,
76.81363258491568, 3409.8111912317886, 162.63118055022352, 65.40789353622756,
638.1655414977145, 203.58917497786487, 412.4539807206373, 1255.3282682130205,
96.53717743033933, 270.93530005324624, 38.70055764005259, 267.64491074118547,
2205.259117993071, 907.1615420210312, 37.884135529519966, 12152.9155527971,
85.85486207223178, 404.3920601224414, 34.36606247540202, 200.9890804195913,
47.34193288493648, 11.927656842642081, 71.65679283171939, 153.75970199796643,
33.34907356840441, 2706.3813994644497, 1761.499283203608, 66.80271225182256,
156.71736906199283, 4202.067289748245, 58.74533393155939, 40.5149329579949,
3710.7290711576106, 19.95035635098118, 33178.393564213744, 497.450021679039,
666.1567749494311, 18.08636141483605, 552.8639850386551, 5092.094211816786,
222.8063632701045, 882.0161266668988, 72.37504771162827, 20843.293956517213,
41.30774305962563, 60.768248994273144, 149.15021472271795, 71.26251041175735,
67.07734574095579, 681.7559201170293, 5584.07314806519, 13.718791581373981,
36.20722302541595, 201.2210315978377, 125.1172303184506, 37.96922906490112,
12012.26827427674, 17.980918155871525, 536.0615804559468, 669.2764635374473,
56.32029206810314, 15.99502762868581, 131.09274965760497, 158.23946143378612,
263.54802413584434, 123.9185111423956, 1032.6804751746365, 164.7778020891593,
175.31767394485405, 47.60152062471548, 216.89820798354722, 301.69992084027183,
148.369391795565, 61.79984045978955, 673.6644503834657, 169.79482510795478]





[187]:
```
S_2_dico = {}

# Generate 100 times a lognormal vector of size q
for q in q_list:
    for j, k in enumerate(range(1000)):
        vp = np.random.lognormal(size=q, sigma=2)
        if not j : S_2_dico[q] = np.array(vp)
        else : S_2_dico[q] = np.vstack((S_2_dico[q], vp))

isolated_values = {}

for q in q_list:
    # la collection des diagonales des matrices S_2, c-à-d les vecteurs␣
 ↪log-normaux générés
    collection = S_2_dico[q]
    # isoler des valeurs de alpha2 proche de 2
    isolated_values = np.array([alphaaa2(i) for i in collection if␣
 ↪100<=alphaaa2(i)<=800])
    print("q= ", q)
    print("number of matches :", len(isolated_values))
    print("the best value matching is ", isolated_values[np.argmin(np.
 ↪abs(isolated_values-200))])
    print("the index of matching value", np.where(np.array([alphaaa2(i) for i␣
 ↪in collection])==
```

```
                                                    isolated_values[np.argmin(np.
↪abs(isolated_values-200))])[0][0])
    print()
```

```
q=  50
number of matches : 485
the best value matching is  200.2803785885067
the index of matching value 672

q=  100
number of matches : 544
the best value matching is  200.71140927608317
the index of matching value 5

q=  150
number of matches : 574
the best value matching is  199.88421563294
the index of matching value 85

q=  200
number of matches : 591
the best value matching is  202.33329233624087
the index of matching value 16

q=  250
number of matches : 611
the best value matching is  200.13370088922096
the index of matching value 63

q=  300
number of matches : 615
the best value matching is  199.96935471378177
the index of matching value 579

q=  350
number of matches : 583
the best value matching is  200.26778873916257
the index of matching value 250

q=  400
number of matches : 560
the best value matching is  199.75481902435996
the index of matching value 573
```

[188]:
```
vp_collection_sigma2 = []
```

```
for q in q_list:
    # la collection des diagonales des matrices S_2, c-à-d les vecteurs␣
 ↪log-normaux générés
    collection = S_2_dico[q]
    # isoler des valeurs de alpha2 proche de 2
    isolated_values = np.array([alphaaa2(i) for i in collection if␣
 ↪100<=alphaaa2(i)<=800])
    match_value = isolated_values[np.argmin(np.abs(isolated_values-200))]
    match_value_idx =  np.where(np.array([alphaaa2(i) for i in␣
 ↪collection])==match_value)[0][0]
    vp_collection_sigma2.append(S_2_dico[q][match_value_idx])

pickle.dump(vp_collection_sigma2, open("vp_collection_sigma2", "wb"))
```

[189]:
```
[alphaaa2(i) for i in vp_collection_sigma2]
```

[189]:
```
[200.2803785885067,
 200.71140927608317,
 199.88421563294,
 202.33329233624087,
 200.13370088922096,
 199.96935471378177,
 200.26778873916257,
 199.75481902435996]
```

[195]:
```
teest = pickle.load(open("vp_collection_sigma2", "rb"))
print([i.shape[0] for i in teest])
[alphaaa2(i) for i in teest]
```

```
[50, 100, 150, 200, 250, 300, 350, 400]
```

[195]:
```
[200.2803785885067,
 200.71140927608317,
 199.88421563294,
 202.33329233624087,
 200.13370088922096,
 199.96935471378177,
 200.26778873916257,
 199.75481902435996]
```

[197]:
```
pickle.dump(q_list, open("q_list", "wb"))
pickle.dump(n_list, open("n_list", "wb"))
```

[196]:
```
n_list
```

[196]:
```
array([ 50,  75, 100, 125, 150, 175, 200])
```