Generator (dependent case - s0.99)

June 14, 2023

[11]:
```
import warnings
warnings.filterwarnings('ignore')
```

[12]:
```
############################## 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

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

import time
import concurrent.futures
import random
import os

from itertools import product


######################## For printing with colors ##########################
class color:
   PURPLE = '\033[95m'
   BLACK = '\033[1;90m'
   CYAN = '\033[96m'
   DARKCYAN = '\033[36m'
   BLUE = '\033[94m'
   GREEN = '\033[1;92m'
   YELLOW = '\033[93m'
```

```
    RED = '\033[1;91m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'
    END = '\033[0m'
    BBCKGRND = '\033[0;100m'
    RBCKGRND = '\033[0;101m'


print(color.BLUE + color.BOLD + '******************** Starting the program !␣
 ↪********************' + color.END )
```

```
******************** Starting the program ! ********************
```

0.1 2.1 Toeplitz Matrix

Creating a function that gets two parameters s and q such that : - s (dependency threshold or intraparameter of dispersion) - and q (the dimension)

then returns a symetric toeplitz matrix S^2 that can be used as a covariance matrix for generating normal vectors

[13]:
```
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)
```

[14]:
```
print(cov_toep(0.1, 4))
print()
print(cov_toep(0.99, 4))
```

```
[[1.    0.1   0.01  0.001]
 [0.1   1.    0.1   0.01 ]
 [0.01  0.1   1.    0.1  ]
 [0.001 0.01  0.1   1.   ]]

[[1.       0.99     0.9801   0.970299]
 [0.99     1.       0.99     0.9801  ]
 [0.9801   0.99     1.       0.99    ]
 [0.970299 0.9801   0.99     1.      ]]
```

[15]:
```
# importing q_list, n_list, S2 diagonals, for different sigma (sigma means here␣
 ↪the std of underluying normal
# distribution that generates lognormal vectors), having the same alpha
q_list = list(pickle.load(open("q_list", "rb")))
```

```
n_list = list(pickle.load(open("n_list", "rb")))
print(q_list)
```

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

[16]:
```
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 alphaaa1(S_2):
    """Pour une matrice de covariance non diagonale - cas de dependence
    arg = matrice qxq"""
    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

def alphaaa2(vec):
    """Pour une matrice de covariance diagonale - cas de dependence
    arg = un vecteur qui represente la diagonale"""
    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
```

[17]:
```
######################### Preparing true covariance matrices for different␣
 ↪values of q ##########################

valeur_propre_collection = [cov_toep(0.99,q) for q in q_list]
print(*valeur_propre_collection[q_list.index(50)][0][:10])
print([alphaaa1(cov_toep(0.99,q)) for q in q_list])
```

```
1.0 0.99 0.9801 0.970299 0.96059601 0.9509900498999999 0.941480149401
0.9320653479069899 0.9227446944279201 0.9135172474836408
```

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

[18]:
```
# list(product(n_list, range(K)))[:12]
```

[19]:
```
############################ Choosing dimension␣
↪#######################################

# K = int(input("Number of Monte Carlo iterations is : "))
print()
print("list of q values : ", q_list, "\n")
print("list of n values : ", n_list, "\n")


######################### Generator function ################################

def generator(n, q):
    """
    Creating a function that gets into paramaters :
       the number of observations
       the dimension
       dependency threshold
     ²  covariance parameter
    and returns a matrix of n observations (n rows), where each row represents
    a q-vector normally distributed with a mean 0 and covariance matrix S²␣
 ↪defined
    by a toeplitz matrix with a threshold s
    """

    # defining eigenvalues, lognormal variables
    valeur_propre = valeur_propre_collection[q_list.index(q)]

    # defining a mean vector and a covariance matrix
    mean = np.zeros(q) ; cov = valeur_propre

    # z_intermediate = q rows and n columns we still need to transpose
    z_int = np.random.multivariate_normal(mean, cov, n).T

    # z sample matrix, having n rows and q columns
    z = np.transpose(z_int)

    # Returning the matrix of n-observations of dimension q
    return z


########################### Statistics functions ######################
```

```
def sn2(z):
    return np.cov(np.transpose(z))

def zbar(z):
    """takes a n x q sample matrix and return the vector mean of each column
    """
    return z.mean(axis=0)

def max_p(M):
    """Largest eigenvalue of a given matrix M"""
    val_p = np.linalg.eigvals(M)
    return max(val_p.real)

def min_p(M):
    """Smallest eigenvalue of a given matrix M that is not null"""
    val_p = np.linalg.eigvals(M)
    val_p = val_p[val_p>=10**-6]
    return min(val_p.real)


def product_vect(z, i):
    return np.matmul(z[i].reshape(z[i].shape[0], 1), np.transpose(z[i].
 ↪reshape(z[i].shape[0], 1)))


################################ Algebra functions␣
 ↪#####################################
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))
```

```
list of q values :  [50, 100, 150, 200, 250, 300, 350, 400]

list of n values :  [50, 75, 100, 125, 150, 175, 200]
```

1 Time comparision

[20]:
```
def monte_carlo(k):
    # random.seed(k)
    np.random.seed(int(os.getpid() * time.time()) % 123456789)
    z = generator(n,q)
    sn_2 = sn2(z)
    # beta = (norm(sn_2 - s_2))**2
    # calculate empirical mean
    z_bar = zbar(z)
    #empirical covariance matrix
    sn_2 = sn2(z)

    # Calculating empirical eigenvalues and eigenvectors (of Sn²)
    emp_val_p, emp_vec_p = np.linalg.eigh(sn_2)

    # Calculating true (theoretical) eigenvalues and eigenvectors (of S²)
    # val_p, vec_p = np.linalg.eigh(s_2)

    # Identity matrix of size q
    I_q = np.diag(np.ones(q))

    # sigma_n ( ² )
    sigma_n = scalar(sn_2, I_q)

    # delta_n ( ²)
    delta_n = norm(sn_2 - sigma_n*I_q)**2

    # intermediate beta_n
    beta_bar_n = (1/n**2)*0
    for i in range(n):
        beta_bar_n += (1/n**2)*norm(product_vect(z, i) - sn_2)**2

    # beta_n ( ²)
    beta_n = min(beta_bar_n, delta_n)

    # alpha_n ( ² )
    alpha_n = delta_n - beta_n

    # rho_n ( *² )
    rho_n = (beta_n/alpha_n)*sigma_n

    rho_1_n = (beta_n/delta_n)*sigma_n

    rho_2_n = alpha_n/delta_n

    Sigma_n_hat_ast = sn_2 + rho_n*I_q
```

```
    Sigma_n_hat = rho_1_n*I_q + rho_2_n*sn_2

    self_norm_sum = n*z_bar.dot(np.linalg.inv(Sigma_n_hat).dot(np.
 ↪transpose(z_bar)))
    self_norm_sum_ast = n*z_bar.dot(np.linalg.inv(Sigma_n_hat_ast).dot(np.
 ↪transpose(z_bar)))
    return np.array([self_norm_sum, self_norm_sum_ast, beta_n, sigma_n,␣
 ↪alpha_n, rho_n, max_p(sn_2),
                     min_p(sn_2)], dtype=np.int)



t1 = time.perf_counter()
dico = {}
for n in n_list[:3]:
    for q in q_list[:3]:
        if n <= q :
            dico[(n,q)] = np.stack(list(map(monte_carlo, range(10))))

print(dico[list(dico.keys())[0]])

t2 = time.perf_counter()

print(f'Finished in {t2-t1} seconds')
```

```
[[17 16  2  1 42  0 47  0]
 [32 30  0  0 11  0 24  0]
 [33 31  0  0 22  0 34  0]
 [36 34  1  0 20  0 32  0]
 [20 19  2  1 46  0 49  0]
 [12 11  1  0 29  0 40  0]
 [22 21  1  0 22  0 35  0]
 [24 23  2  1 45  0 49  0]
 [15 14  2  1 64  0 58  0]
 [24 23  1  0 27  0 37  0]]
Finished in 1.3481029690010473 seconds
```

[21]:
```
len(dico.keys())
```

[21]: 7

[22]:
```
t1 = time.perf_counter()
data = {}
for q in q_list[:3]:
    for n in n_list[:3]:
        if n <= q :
            with concurrent.futures.ProcessPoolExecutor() as executor:
```

```
                f1 = np.stack(list(executor.map(monte_carlo, range(10))))
                #f2 = f1.result()
                data[(n,q)] = f1
                # if q/n%1==0 : print(str(i)+","+str(j)+")",
                                #color.BLUE + color.BOLD + f"for n = %d \t␣
 ↪and q = %d"%(n,q) + color.END,"\n",
                                #f1, "\n\n")

print(data[list(data.keys())[0]])

t2 = time.perf_counter()

print(f'Finished in {t2-t1} seconds')
```

```
[[23 22  1  0 26  0 37  0]
 [12 12  1  1 35  0 43  0]
 [19 18  2  1 61  0 57  0]
 [20 19  2  1 38  0 46  0]
 [ 9  8  3  1 58  0 55  0]
 [14 13  1  0 26  0 37  0]
 [16 15  1  0 24  0 36  0]
 [15 14  1  0 24  0 36  0]
 [11 11  2  1 45  0 49  0]
 [25 24  0  0 25  0 36  0]]
Finished in 1.5542298139989725 seconds
```

[23]:
```
data[list(data.keys())[1]]
```

[23]:
```
array([[36, 34,  1,  0, 34,  0, 58,  0],
       [40, 38,  2,  1, 55,  0, 75,  0],
       [31, 29,  2,  0, 37,  0, 62,  0],
       [45, 43,  1,  0, 24,  0, 46,  0],
       [49, 47,  2,  1, 62,  0, 80,  0],
       [46, 43,  3,  0, 50,  0, 73,  0],
       [36, 33,  2,  0, 33,  0, 58,  0],
       [39, 37,  3,  1, 65,  0, 81,  0],
       [57, 54,  4,  1, 95,  0, 99,  0],
       [35, 32,  1,  0, 19,  0, 43,  0]])
```

2 Generating step

[24]:
```
K = int(input("Number of Monte Carlo iterations is : "))
```

```
Number of Monte Carlo iterations is : 999
```

[25]:
```
t_init = time.perf_counter()
dico = {}
for n in n_list:
    for q in q_list:
        t1 = time.perf_counter()
        if n <= q :
            dico[(n,q)] = np.stack(list(map(monte_carlo, range(10))))
            t2 = time.perf_counter()
            if q==n or q==2*n :
                print(f"q = {q}  and  n = {n} --> ",f'Finished in {round(t2-t1,
 ↪4)} seconds')

print(dico[list(dico.keys())[0]][:10])

t_fin = time.perf_counter()
print()
print(f'All loops finished in {round(t_fin-t_init, 3)} seconds')
```

```
q = 50  and  n = 50 -->  Finished in 0.0503 seconds
q = 100  and  n = 50 -->  Finished in 0.0863 seconds
q = 150  and  n = 75 -->  Finished in 0.2668 seconds
q = 100  and  n = 100 -->  Finished in 0.1252 seconds
q = 200  and  n = 100 -->  Finished in 0.5247 seconds
q = 250  and  n = 125 -->  Finished in 1.1458 seconds
q = 150  and  n = 150 -->  Finished in 0.4823 seconds
q = 300  and  n = 150 -->  Finished in 2.7629 seconds
q = 350  and  n = 175 -->  Finished in 3.5474 seconds
q = 200  and  n = 200 -->  Finished in 1.0641 seconds
q = 400  and  n = 200 -->  Finished in 5.6463 seconds
[[12 11  2  1 39  0 46  0]
 [23 23  0  0 25  0 36  0]
 [25 24  1  0 31  0 41  0]
 [23 22  1  0 20  0 33  0]
 [16 15  2  1 47  0 50  0]
 [31 30  1  0 33  0 42  0]
 [10  9  1  0 23  0 36  0]
 [13 12  1  0 30  0 40  0]
 [19 18  2  1 38  0 45  0]
 [16 15  0  0 14  0 27  0]]

All loops finished in 77.982 seconds
```

[26]:
```
def monte_carlo(k):
    # random.seed(k)
    np.random.seed(int(os.getpid() * time.time()) % 123456789)
    z = generator(n,q)
    sn_2 = sn2(z)
```

```
    s_2 = valeur_propre_collection[q_list.index(q)]
    beta = (norm(sn_2 - s_2))**2
    # calculate empirical mean
    z_bar = zbar(z)
    #empirical covariance matrix
    sn_2 = sn2(z)

    # Calculating empirical eigenvalues and eigenvectors (of Sn²)
    emp_val_p, emp_vec_p = np.linalg.eigh(sn_2)

    # Calculating true (theoretical) eigenvalues and eigenvectors (of S²)
    # val_p, vec_p = np.linalg.eigh(s_2)

    # Identity matrix of size q
    I_q = np.diag(np.ones(q))

    # sigma_n ( ² )
    sigma_n = scalar(sn_2, I_q)

    # delta_n ( ²)
    delta_n = norm(sn_2 - sigma_n*I_q)**2

    # intermediate beta_n
    beta_bar_n = (1/n**2)*0
    for i in range(n):
        beta_bar_n += (1/n**2)*norm(product_vect(z, i) - sn_2)**2

    # beta_n ( ²)
    beta_n = min(beta_bar_n, delta_n)

    # alpha_n ( ² )
    alpha_n = delta_n - beta_n

    # rho_n ( *² )
    rho_n = (beta_n/alpha_n)*sigma_n

    rho_1_n = (beta_n/delta_n)*sigma_n

    rho_2_n = alpha_n/delta_n

    Sigma_n_hat_ast = sn_2 + rho_n*I_q
    Sigma_n_hat = rho_1_n*I_q + rho_2_n*sn_2

    self_norm_sum = n*z_bar.dot(np.linalg.inv(Sigma_n_hat).dot(np.
↪transpose(z_bar)))
    self_norm_sum_ast = n*z_bar.dot(np.linalg.inv(Sigma_n_hat_ast).dot(np.
↪transpose(z_bar)))
```

```
    return np.array([self_norm_sum, self_norm_sum_ast, beta_n, sigma_n,␣
 ↪alpha_n, rho_n, max_p(sn_2),
                     min_p(sn_2), beta])
```

[27]:
```
print(color.BLUE + color.BOLD + '******************** Starting the extraction !␣
 ↪********************' + color.END )

K = int(input("Number of Monte Carlo iterations is : "))
t_init = time.perf_counter()
dico = {}
for n in n_list:
    for q in q_list:
        t1 = time.perf_counter()
        if n <= q :
            dico[(n,q)] = np.stack(list(map(monte_carlo, range(K))))
            t2 = time.perf_counter()
            if q==n or q==2*n :
                print(f"q = {q}  and  n = {n} --> ",f'Finished in {round(t2-t1,␣
 ↪4)} seconds')

print(dico[list(dico.keys())[0]][:10])

t_fin = time.perf_counter()
print()
print(f'All loops finished in {round(t_fin-t_init, 3)} seconds')
```

```
******************** Starting the extraction ! ********************
Number of Monte Carlo iterations is : 999
q = 50  and  n = 50 -->  Finished in 3.7521 seconds
q = 100  and  n = 50 -->  Finished in 8.4912 seconds
q = 150  and  n = 75 -->  Finished in 28.0677 seconds
q = 100  and  n = 100 -->  Finished in 30.1414 seconds
q = 200  and  n = 100 -->  Finished in 130.4157 seconds
q = 250  and  n = 125 -->  Finished in 217.6728 seconds
q = 150  and  n = 150 -->  Finished in 71.7104 seconds
q = 300  and  n = 150 -->  Finished in 376.5496 seconds
q = 350  and  n = 175 -->  Finished in 594.7796 seconds
q = 200  and  n = 200 -->  Finished in 180.3045 seconds
q = 400  and  n = 200 -->  Finished in 1054.9941 seconds
[[3.21952376e+01 3.10150524e+01 7.11801162e-01 7.74766911e-01
  1.87060062e+01 2.94814394e-02 3.13576569e+01 1.04305480e-05
  2.66646422e+00]
 [1.58018970e+01 1.50726312e+01 1.55567635e+00 9.63912677e-01
  3.21530736e+01 4.66374125e-02 4.14331898e+01 1.23171019e-05
  4.33843232e-01]
 [2.38948749e+01 2.26396453e+01 1.24945839e+00 8.43273886e-01
  2.25355540e+01 4.67543699e-02 3.47204240e+01 1.91892136e-06
```

```
 1.45305188e+00]
[1.48164040e+01 1.38989302e+01 3.62768241e+00 1.22649215e+00
 5.49562317e+01 8.09612282e-02 5.46689243e+01 7.49985260e-06
 3.30060712e+00]
[1.40688199e+01 1.30875111e+01 5.01252764e+00 1.38977875e+00
 6.68510356e+01 1.04206380e-01 6.04238849e+01 5.24114757e-05
 6.98224785e+00]
[2.14795707e+01 2.05093405e+01 2.47552942e+00 1.19821392e+00
 5.23293052e+01 5.66836078e-02 5.28495461e+01 7.34089613e-06
 2.28001743e+00]
[1.74796520e+01 1.68958670e+01 1.11538497e+00 9.74024052e-01
 3.22814009e+01 3.36544190e-02 4.11613570e+01 4.31547260e-06
 6.55340018e-01]
[1.73714222e+01 1.66508228e+01 1.33758807e+00 9.78928856e-01
 3.09075196e+01 4.23652101e-02 4.04156895e+01 9.64826294e-06
 6.69483702e-01]
[2.63460685e+01 2.49666493e+01 1.72078569e+00 9.39745335e-01
 3.11451747e+01 5.19213759e-02 4.09484789e+01 1.03666823e-05
 1.99555910e-01]
[3.21045671e+01 3.05140447e+01 2.70927656e+00 1.20189590e+00
 5.19772522e+01 6.26479519e-02 5.27637989e+01 1.44505065e-05
 2.22342701e+00]]

All loops finished in 12819.036 seconds
```

[28]:
```
pickle.dump(dico, open("data_d_s0.99", "wb"))
```

[29]:
```
teeest = pickle.load(open("data_d_s0.99", "rb"))
len(teeest[list(teeest.keys())[0]][:,0])
```

[29]: 999

[30]:
```
print(teeest[list(teeest.keys())[0]][:,0])
```

```
[32.19523757 15.80189695 23.89487488 14.81640401 14.06881987 21.47957073
 17.47965199 17.37142225 26.34606849 32.10456712 18.90710812 35.21434059
 16.41328806 18.70887049 26.99265713 25.4879795  27.45100905 18.65066424
 20.04724445 17.95743187 12.70408048 27.96108477 37.59509942 29.13596961
 32.80142467 20.31098177 38.14106264 18.16441563 25.29244995 22.93025313
 20.70963424 17.91635523 20.71352662 14.98602769 19.55511269 26.57521169
 29.54276634 16.3618215  36.7957392  31.12048128 21.58148172 17.45545827
 18.44907662 28.34065285 19.09601013 16.71445689 14.318893   18.21714379
 10.24198793 30.07984393 36.61293907 15.84908024 12.40630342 14.21289174
 24.68740781 17.59144989 24.31415299 13.22510019 18.81581884 36.5785202
 16.41400052 14.28758412  9.55724017 28.57306457 20.71965413 18.24806039
 24.42370481 28.95239037 23.00849872 20.42853623 14.27368697  9.97168807
 20.06070935 16.55865031 25.3904365  17.80131985  9.34655483 18.0944154
 21.99211982 12.05998174 17.51636642 28.77100146 16.23402093 27.30643612
```

15.16221753 24.58433238 24.53671158 22.08373828 16.34617536 18.28717495
19.97954536 12.25860582 32.31104936 11.96916653 13.58592018 20.72554146
13.46930618 21.81002787 25.78203811 26.50447317 30.59474952 44.95519194
14.09714471 24.78587802 19.29329001 13.14365089 15.9773684 15.25759351
21.03730467 11.08534256 11.56063534 25.58498073 24.63642576 32.65976871
18.66434821 16.62865518 18.01302295 21.21631731 15.70765423 17.17877074
26.84019579 16.48285308 24.35000913 21.46558245 7.74558607 17.54680568
22.14067614 20.22379356 16.17508331 15.63574384 27.06920418 26.01939326
12.18189573 28.21009437 31.08350156 22.56461371 21.39959698 14.05110296
12.38472658 13.55983531 14.94223725 19.57064303 16.2921108 27.56572954
21.20914937 15.41160369 18.10213129 26.58358335 28.20631465 17.57277008
13.41243238 24.03662873 21.0269898 18.19623509 24.49952284 22.98200912
31.53124624 20.28912817 27.96059447 18.70498148 22.27786549 30.82603081
24.13881961 22.47105248 28.23628547 12.74203473 17.88253028 14.52211938
21.47429562 43.21531755 22.83275824 18.85683271 12.0963194 18.80422742
17.16823709 18.69559487 10.28063417 13.36757335 20.27443267 19.38327505
14.17610973 22.5320225 17.90599958 25.74824091 19.98611758 35.36280504
21.63150552 18.24275747 23.72107551 16.9460321 22.36058992 24.28578919
30.56770796 19.43873739 26.09271346 18.14309283 33.92486024 31.84478255
17.47857671 18.85259735 19.07230278 8.94818082 23.32362414 11.47078581
32.44448232 17.92586846 18.27083057 22.61913998 29.08638566 17.86962722
46.2900509 14.0474269 27.78721842 13.07082725 14.18044519 16.72834111
46.71366638 20.20175497 17.90651521 22.94336293 25.66408989 21.98732126
14.74656019 23.92981327 12.14360716 19.37589942 16.12418826 11.17969519
21.47311948 28.82076134 13.98759929 14.40889509 17.33371829 20.52443305
25.74560957 24.09235984 26.99946132 22.01364513 22.40363501 16.4867399
16.70344334 26.02854224 29.43709705 24.63292527 27.88383319 33.23897867
34.65687195 26.60615249 23.57686974 26.52203218 18.92108736 31.96603166
20.2780884 26.4614222 14.87488025 17.92995086 19.4515709 22.10347566
34.02286498 16.88198313 17.28588671 22.97867193 13.4143453 38.51269316
 9.40338137 15.01976859 28.19550143 14.99595313 25.11045411 16.21879668
24.31652794 19.49395827 19.67153039 23.55176008 12.47724734 16.06840292
14.91613812 36.35629001 15.00262902 16.23662878 22.48239515 32.00219969
16.96827751 30.41115497 16.29358316 16.68062716 12.45710803 17.73082988
24.82343914 10.75241574 22.00869158 18.41430104 18.90016904 26.10379224
10.18449928 32.06035358 22.05576906 20.35704817 21.65341056 18.66267535
17.91920401 27.51218084 28.95795127 14.25337613 15.05040741 25.02315169
14.26208877 27.65953607 13.195796   17.61924935 21.26251624 21.30472344
25.68955632 13.65521268 19.33173889 23.05893985 22.70115924 16.52381744
11.24118571 16.028321   10.54114345 20.57003401 21.33092152 13.64033695
26.20275991 20.88932242 23.49789363 21.34629554 28.20689142 28.04625813
13.32036804 16.14191324 23.157915   33.03788569 22.9940731 10.02671955
 7.20189032 18.5214538 23.94634564 21.38817143 20.69338078 21.18402688
24.38219488 12.56818704 13.24910711 23.11899399 26.06496094 19.81996185
21.3322929 19.3665599 17.39950671 16.96650609 15.30822261 24.0509664
17.8113926 12.36470512 23.74015811 8.52833785 20.45267642 23.30923112
34.46625637 16.79756479 13.08999215 15.86339327 20.73478597 18.65970069
17.97579946 16.38895575 13.41441672 18.62293647 16.59413042 13.37294092

28.44264573 13.74992879 40.71569865 19.40869833 26.90861471 24.18460321
11.22579002 24.50239226 31.10875993 16.55212288 27.77508785 28.65992126
21.61160971 22.70598722 16.17567624 18.26069201 17.94152559 20.59131116
39.08906745 8.6875946 28.70711311 9.49036513 17.37075148 31.94548957
24.00991462 25.38762916 12.5214531 11.13730962 18.32981072 32.37299812
15.66796146 26.59828964 35.88953446 20.67243698 13.50148465 36.84039762
19.56817729 20.85041288 25.98431215 36.79505   17.13690225 18.0956179
12.04699742 17.26597146 20.96487484 25.58525211 17.64471758 14.82360898
26.31320353 27.77683922 19.76229362 23.96539364 27.32897978 31.35129921
12.54961868 12.83079028 18.94512717 23.22761022 20.42574236 28.41742317
17.12921244 21.65181597 16.58118469 26.52555703 23.51429208 27.46446481
13.89370884 19.08390502 19.94324591 21.57933327 16.10852463 14.74741375
18.2135284 17.0873142 24.55623251 16.73127791 19.08102785 23.14100958
16.61324072 19.08878497 12.20600911 17.17456563 21.11136063 24.0311683
15.69837001 20.22403574 30.00951278 16.09718948 33.08193105 15.42790884
13.28959425 15.9728505 16.72234763 21.37923284 27.33196312 16.1908917
20.81945756 19.25185058 25.05123704 23.09349414 33.28910406 8.39524149
15.98040448 15.45634118 19.20084667 12.93507051 22.11190771 24.87566637
22.8022404 28.04145553 17.23615655 25.28628766 14.23912362 40.69813341
25.33681824 17.09291136 35.15101292 18.58733543 33.17672604 29.25358264
30.70199631 39.11964701 22.23621265 16.13884388 19.33683257 11.77899971
12.38032483 17.40198807 19.47164159 28.59170959 15.82146176 19.83532297
24.97795005 26.80801439 21.10351181 19.73476957 32.43096837 10.85244187
14.83930507 27.29761937 18.9811558 27.31997402 21.26214697 19.66152278
11.78195072 13.01758517 19.04426009 19.82636213 30.41539603 7.31586126
13.55402567 13.11670168 18.65379254 16.90602599 18.48115866 18.48248831
16.85065655 22.0226538 16.37358173 18.42204227 31.62317614 7.53614908
12.01841436 15.91356855 16.64989056 32.18760533 31.00399831 31.35248441
24.22454411 8.5316864 15.90062499 8.39984968 23.44716896 20.76587854
15.33942498 15.92961351 15.11699204 19.35557047 13.3606418 19.34726245
27.36609478 15.6682998 28.35329195 25.53564064 28.66025014 32.01977781
21.84978422 20.16740811 17.0057583 23.09185081 13.95321638 21.93899024
20.56990567 10.9976719 11.67838003 24.21368757 16.96044815 26.24501856
15.48696342 23.04001557 40.59202931 21.32231122 18.28606537 31.73931727
31.87526179 18.30788022 15.22691454 22.33873236 33.26134875 10.78207257
10.27338913 14.93084241 9.70747967 32.42529818 39.31833097 17.52937135
26.26572159 15.77409195 20.3948767 17.70827439 11.58823727 21.09949342
11.77181807 24.16907532 21.74167716 31.2536833 19.74530256 16.35492983
19.4954017 20.4038971 25.4297575 21.36600506 16.0357119 22.12720603
21.00071308 17.07594792 17.95475667 27.97011219 15.64969239 17.8950925
27.43035931 15.66689823 27.77154016 12.97653646 20.38923481 7.7814188
14.17015543 24.84664638 17.44977111 10.99599239 15.55976363 10.68350102
19.67803211 21.74849652 25.31119323 32.92832174 15.633299   29.90760744
16.71904694 18.40883489 18.87512172 27.65049371 21.9043545 15.24033027
20.49041452 16.19435019 11.23703012 17.12408885 19.33961848 20.33159345
12.80075797 20.80253889 33.04138194 19.00960764 14.11063989 31.7277385
18.16387809 17.82722061 10.57586274 15.82342172 25.57438414 32.36393542
22.60873505 27.25565662 30.47035391 22.89455926 15.07276718 39.55771593

13.62016505 13.10871354 30.84338717 36.38271429 21.76783125 12.9741421
15.39932355 26.60628553 14.36282432 33.45614984 11.25271977 29.91731154
 9.64982404 25.26574768 17.44790939 15.72897922 11.99999872 18.4320264
20.018824   22.49162524 14.20995822 10.98934403 11.96812884 22.29318502
20.9722146 22.40102466 21.40699889 32.18070147 32.30168936 17.90427593
18.81770332 11.07799493 18.44397271 15.62199383 22.02331434 34.40366823
30.6698688 14.59305403 20.09559971 21.00359305 19.41716846 10.99791693
12.66389947 24.09814908 23.28192142 18.73238441 24.09732637 20.83168965
28.76735162 17.10993423 20.68394468 47.70784036 21.32462412 13.5374993
18.99231852 16.08054365 33.09816589 12.50770638 20.42348309 17.90880479
20.78742586 19.69141043 24.5692579 21.3550679 28.29747324 11.01314887
22.06312693 22.18944643 19.56012514 19.66939399 32.05432438 22.30177731
29.05813578 17.9080122 33.54617601 26.12267639 36.45815201 29.92125603
21.29085651 17.79670927 16.60707916 17.96374821 16.65218532 13.76477207
13.34710803 26.94717848 18.94761058 32.24040491 8.62276081 24.65304951
11.64679846 27.93125303 18.33900066 15.07698701 20.20894599 19.76942215
15.83555057 15.83271433 21.23249286 22.04948217 16.96134834 12.30665287
13.61042334 29.00187898 23.47633336 15.2284743 24.2027681 27.01380534
25.77377578 15.10544687 12.56628416 21.50175936 21.93644476 25.50589544
20.54873214 16.74601351 20.70699198 10.02194653 33.63953549 14.41204684
24.50031955 24.22820336 20.83685754 16.1627584 12.37926358 24.83190719
13.37764848 28.63928884 16.42170372 14.37876926 18.80237465 18.28194992
15.42988623 26.80934936 16.8284846 24.22752669 16.80037983 16.53616319
28.00594814 12.9386479 25.87189928 20.38097048 15.23330511 13.807863
39.53871184 38.20828285 18.70809386 16.85970095 27.01118789 19.05340922
12.97318717 24.00288551 20.06378309 12.13590611 23.78656365 17.32443698
27.67513848 34.87305801 21.08301112 26.27700148 15.86818655 17.51676105
22.11422675 23.42167231 13.09513339 21.9936216 27.20859823 13.33835061
32.09509114 18.50953066 13.65717235 17.94175824 17.26116276 23.6874589
29.27104374 44.44419763 13.34918897 10.1439381 16.64318267 12.10951693
18.44410525 31.91430842 6.09744408 14.7839464 10.34149261 17.43192943
20.71820811 16.6890982 15.88904503 20.78425425 12.00090244 11.06250884
19.83105756 22.18983309 18.23866678 13.91887957 20.87057167 37.22594466
24.61167044 21.9769213 23.06031406 9.39642285 20.66446031 16.76493473
21.89935572 24.75781933 14.16747468 13.07690286 16.86342177 19.22087445
16.43453147 17.03689849 27.68860833 13.12398381 15.43589659 32.72350139
18.59825242 30.10363041 29.74356809 15.42184243 14.69491452 24.55604626
13.86897775 26.67952268 21.26846841 20.55317095 26.10445446 34.60679721
23.50471602 13.63922473 20.52351311 25.10742135 17.37389889 17.16079895
32.51644953 21.19285569 23.2771468 18.82148436 22.6188909 18.65586217
10.43440607 21.30243041 14.84506326 15.9868196 15.74553909 17.68295988
34.31309188 25.88664647 28.11619516 13.86251628 18.86744642 17.13502405
17.74721701 25.17821245 25.53286527 12.7051435 18.00228706 16.76226074
16.82097429 26.3295297 16.14803274 13.44825412 24.03832361 29.0073573
15.50292282 14.67169802 18.81460777 12.79017351 24.06778857 13.01712087
16.55136746 15.49830354 14.70990553 23.99099714 18.68301874 22.26120631
11.67484746 21.37917157 26.32910516 10.05572546 36.43636476 22.23876511
15.44458882 18.51952737 26.54970351 18.26873111 19.29380834 14.62835735

```
 17.2950305  34.95381781 16.76664645 17.20008327 19.15615649 11.85782589
26.06730724 17.50602599 18.75779767 27.31743291 15.35596598 35.15757091
15.96276035 23.07993902 16.19269361 19.00796293 11.02722154 18.46448869
 9.29670949 21.97097378 17.58806229 43.55398337 15.48990345 17.47382929
 9.70029417 23.97663808 19.85078991 10.46356389 12.06027822 26.48645368
22.67437951 17.68534876 19.06162158 24.37593086 14.94798138 16.68501431
30.02983244 16.62407604 17.18179953 15.47742466 20.04641304 24.21275406
19.01395772 22.27773142 15.58000302 21.11940014 15.77065429 27.27129941
14.62897508 15.62619175 22.64970433]
```

[ ]:

[ ]:

[ ]: