Hyperuniformity

We propose to study by numerical simulation the hyperuniform states which appears in various situations (biology, astrophysics, granular media). The hyperuniformity has a very recent reseach activity, because unusual physical properties seem to be related to the disordered structure of these systems. The goal of this homework is to study by numerical simulation some properties characterizing the state of the matter.

The hyperuniformity corresponds to the situation when the variance of the number of particles in a finite volume of the system scales less than the number of particles with the linear size L' of the volume.

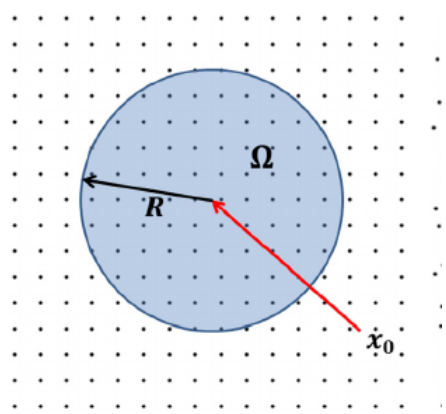


FIGURE 1 – Observation window of radius R

For this homework, you can choose the preferred language for your codes. The simulations are not very demanding for this first homework and you can use Python by using massively numpy and by suppressing loops in many situations. The typical execution time in Python should be close to 2mns on a Laptop.

1 Lattice

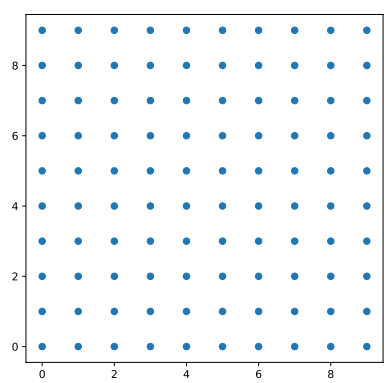
One first considers a square lattice of linear size L, where point particles are located at the lattice nodes. One set the lattice step as a unit length, which gives a total number of particles equal to L^2. To avoid boundary effects, one must implement periodic boundary conditions

1. Write a simple code displaying point particles on a square lattice of linear size $L = 10$.

Solution:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 5 12:41:36 2021

@author: viot
"""
import numpy as np
import matplotlib.pyplot as plt

class square:
    def __init__(self,L):
        self.L=L
        self.N=self.L*self.L
        X=np.arange(0,self.L)
        Y=np.arange(0,self.L)
        self.x,self.y=np.meshgrid(X,Y,sparse=False, indexing='ij')
    def plot_lattice(self):
        plt.figure(figsize=(6,6))
        plt.scatter(self.x,self.y)

Square=square(10)
Square.plot_lattice()
plt.savefig("lattice.pdf")
plt.show()
```

with the figure



2. To measure the hyperuniformity of the system, one introduces a circular window of observation of radius. x_0 denotes the center of the window. (see Fig.1) The mean

variance of the number of particles for a window of radius R is given by

$$\sigma^2(R) = \langle N^2(R)\rangle - \langle N(R)\rangle^2$$

where the brackets denote the average of over uniform random positions x_0 and $N(R)$ the number of particles of the observation window. In your code, you must include the periodic boundary conditions. Write a code for calculating this quantity as a function of R. Justify why the maximum distance for measuring the variance is $L/2$. For convenience, one chooses the minimum distance of observation equal to 0.01. The number of random points is equal to 10000 and the number of bins for calculating the rescaled variance is equal to 200 and the different distances R in the interval $[0.01, L/2]$ must be choosen with a geometric sequence.

Hint : Finally, you should obtain a figure similar to Fig.2.

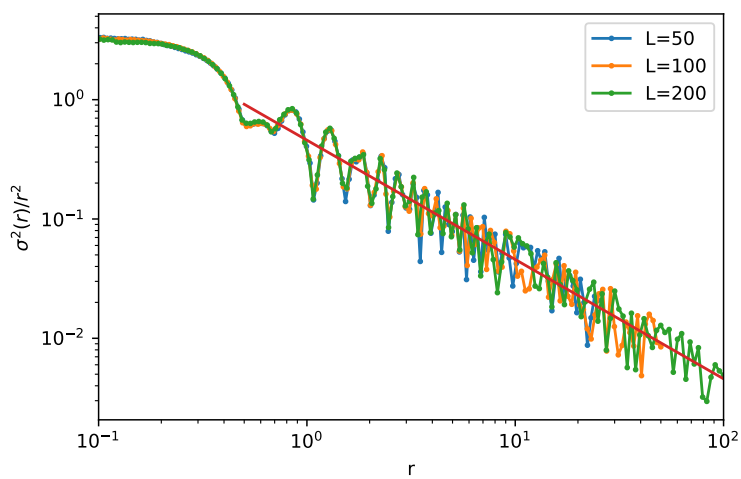


FIGURE 2 – Rescaled variance $\frac{\sigma^2(r)}{r^2}$ as a function of the linear size r of the window.

Solution:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 5 12:41:36 2021

@author: viot
"""

import numpy as np
```

```
import matplotlib.pyplot as plt


import time


class square:

    def __init__(self,L,rmin=0.1,nbin=100,npoint=10000):
        self.L=L
        self.N=self.L*self.L
        self.rmin=rmin
        self.nbin=nbin
        self.r=np.geomspace(self.rmin,self.L/2,self.nbin)
        self.q=self.r[1]/self.r[0]
        self.npoint=npoint
        X=np.arange(0,self.L)
        Y=np.arange(0,self.L)
        self.x,self.y=np.meshgrid(X,Y,sparse=False, indexing='ij')
    def plot_lattice(self):
        plt.scatter(self.x,self.y)
        plt.show()
    def calcul(self):
        n=np.zeros(self.nbin)
        nn=np.zeros(self.nbin)
        x2=self.x-self.x0
        y2=self.y-self.y0
        Ls2=self.L/2
        x2=x2+self.L*(x2<-Ls2)
        x2=x2-self.L*(x2>Ls2)
        y2=y2+self.L*(y2<-Ls2)
        y2=y2-self.L*(y2>Ls2)
        dist=x2*x2+y2*y2
        rmax2=self.N/4
        rmin2=self.rmin*self.rmin
        dist=np.sort(dist)
        dist=dist[dist>rmin2]
        dist=dist[dist<rmax2]
        indm=np.int64(0.5*np.log(dist/rmin2)/np.log(self.q))
        for i in np.arange(self.nbin):
            nn[i]=np.count_nonzero(indm==i)
        n=n+nn
        n=np.cumsum(n)
        return n
```

```
    def var_placement(self):
        posaleat=np.random.uniform(0,self.L,size=(self.npoint,2))
        sumn=np.zeros(self.nbin)
        sumn2=np.zeros(self.nbin)
        for i,j in posaleat:
            self.x0=i
            self.y0=j
            tmp=self.calcul()
            sumn=sumn+tmp
            sumn2=sumn2+tmp*tmp
        sumn/=self.npoint
        sumn2/=self.npoint
        variance_s=(sumn2-sumn*sumn)
        plt.loglog(self.r,variance_s/self.r/self.r,'-o',ms='2',subs
            ↪ =[1,2,3,4,5,6,7,8,9,10],label='L='+str(self.L))
        plt.xlim(0.1,100)
        return(self.r,variance_s)



def compute(i):
    Square=square(i,0.01,200,10000)
    start=time.time()
    r,var=Square.var_placement()
    end=time.time()
    print("elapsed time",end-start)

for i in [50,100,200]:
    compute(i)
x=np.linspace(0.5,100,10)
plt.loglog(x,0.457648/x)
x2=np.linspace(0.01,1/2,200)
plt.loglog(x2,np.pi-np.pi**2*x2**2)

plt.legend()
plt.xlabel('r')
plt.ylabel(r'$\sigma^2(r)/r^2$')
plt.savefig("hyperbis.pdf")
plt.show()
```

The computer time for these three curves are on my laptop

```
elapsed time 6.758406639099121
elapsed time 11.615439176559448
elapsed time 32.40011286735535
```

It is possible to obtain more efficient code, but the objective is reached. Less than 1mn !

Due to the finite size of the simulation box and with the periodic boundary conditions, the maximum of the distance for measurement is $L/2$.

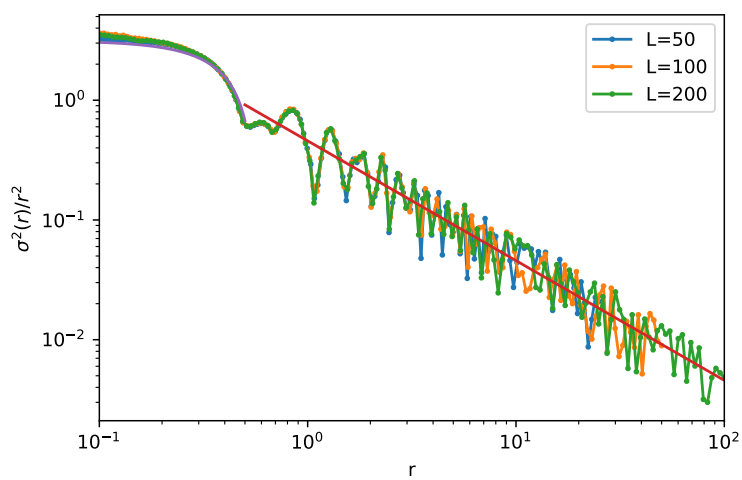
3. Fig.2 shows the rescaled variance versus r for 3 different system sizes. Comment the results : why do $\sigma^2(r)/r^2$ goes to a constant at short distance ? What are the reasons of the oscillations of the curve ? The straight line corresponds to the asymptotic behavior of variance at long distance.

$$\sigma^2(r)/r^2 = \frac{0.457648}{r}$$

.

Why do the amplitudes of the oscillations decrease at long distance ?

Solution: For distance less than 1/2, the circular windows captures zero or one particule. The variance goes as $\pi r^2 - \pi^2 r^4$ and the reduced variance goes to $\pi - \pi^2 r^2$. However, fluctuations increase at short distance and the asymptotic value is difficult to observe. At long distance the discrete symmetry of the lattice (which is responsible of the oscillations) has less impact for the calculation of the variance because the number of particles in the window is large. One adds the analytical expression on the following figure :



2 Shuffled lattice

We obtained that the ordered system displays hyperuniformity as expected . In this second part of the homework, one considers particles which undergoes a single stochastic displacement from the initial lattice configuration.

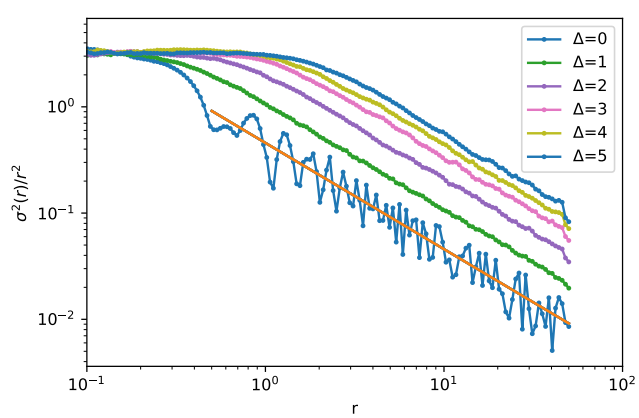
1. Starting from the lattice configuration move all particles by a displacement in all directions by selecting random numbers from a centered uniform distribution with the range $[-\Delta/2, \Delta/2]$
   Modify the previous code for simulating this new system and plot rescaled variance as a function of the distance for different systems when $\Delta = 1, 2, 3, 4, 5$. ($L = 100$). Save the program and generate the pdf files for the different figures. What do you oberve ? Give a physical interpretation of the observation.

Solution: One adds a member function to the square class for moving the particles and some ligne of the main code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 5 12:41:36 2021

@author: viot
"""

import numpy as np
import matplotlib.pyplot as plt
#from pybinding.repository import graphene
import time

class square:

    def __init__(self,L,rmin=0.1,nbin=100,npoint=10000):
        self.L=L
        self.N=self.L*self.L
        self.rmin=rmin
        self.nbin=nbin
        self.r=np.geomspace(self.rmin,self.L/2,self.nbin)
        self.q=self.r[1]/self.r[0]
        self.npoint=npoint
        X=np.arange(0,self.L)
        Y=np.arange(0,self.L)
        self.x,self.y=np.meshgrid(X,Y,sparse=False, indexing='ij')
    def plot_lattice(self):
        plt.scatter(self.x,self.y)
```

```
        plt.show()
    def calcul(self):
        n=np.zeros(self.nbin)
        nn=np.zeros(self.nbin)
        x2=self.x-self.x0
        y2=self.y-self.y0
        Ls2=self.L/2
        x2=x2+self.L*(x2<-Ls2)
        x2=x2-self.L*(x2>Ls2)
        y2=y2+self.L*(y2<-Ls2)
        y2=y2-self.L*(y2>Ls2)
        dist=x2*x2+y2*y2
        rmax2=self.N/4
        rmin2=self.rmin*self.rmin
        dist=np.sort(dist)
        dist=dist[dist>rmin2]
        dist=dist[dist<rmax2]
        indm=np.int64(0.5*np.log(dist/rmin2)/np.log(self.q))
        for i in np.arange(self.nbin):
            nn[i]=np.count_nonzero(indm==i)
        n=n+nn
        n=np.cumsum(n)
        return n
    def var_placement(self,delta):
        posaleat=np.random.uniform(0,self.L,size=(self.npoint,2))
        sumn=np.zeros(self.nbin)
        sumn2=np.zeros(self.nbin)
        for i,j in posaleat:
            self.x0=i
            self.y0=j
            tmp=self.calcul()
            sumn=sumn+tmp
            sumn2=sumn2+tmp*tmp
        sumn/=self.npoint
        sumn2/=self.npoint
        variance_s=(sumn2-sumn*sumn)
        plt.loglog(self.r,variance_s/self.r/self.r,'-o',ms='2',subs
            ↪ =[1,2,3,4,5,6,7,8,9,10],label='$\Delta$='+str(delta))
        plt.xlim(0.1,100)
        return(self.r,variance_s)
    def shuffled(self,delta):
        self.x=self.x+delta*np.random.uniform(-0.5,0.5,size=(self.L,self.L
            ↪ ))
        self.y=self.y+delta*np.random.uniform(-0.5,0.5,size=(self.L,self.L
```

```
            ↪ ))
        self.x=self.x+self.L*(self.x<0)
        self.x=self.x-self.L*(self.x>self.L)
        self.y=self.y+self.L*(self.y<0)
        self.y=self.y-self.L*(self.y>self.L)
        # plt.scatter(self.x,self.y,marker="x")
        # plt.savefig("lattice"+str(self.L))
        # plt.clf()
#L=[10,50,100]
L=100
for i in np.arange(6):
    Square=square(L,0.01,200,10000)
#Carre.plot()
    Square.shuffled(i)
    start=time.time()
    r,var=Square.var_placement(i)
    end=time.time()
    print("elapsed time",end-start)
    print(r[np.argmax(var/r/r)])
    x=np.linspace(0.5,L/2,10)
    plt.loglog(x,0.457648/x)
plt.legend()
plt.xlabel('r')
plt.ylabel(r'$\sigma^2(r)/r^2$')
plt.savefig("hypershuffled.pdf")
plt.show()
```

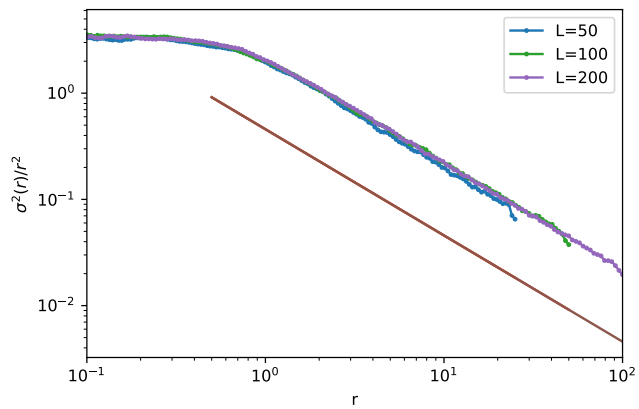


As expected, moving the particles leads to loose hyperuniformity at short distances, but a single motion is not enough to destroy hyperuniformity at long distances even if the amplitude is less pronounced.

2. Modify the previous code in order to plot for $\Delta = 2$ the results for $L = 50, 100, 200$. Comment your results.

Solution:

Few changes in the main code



For a given value of Δ, the different system sizes give almost the same curve. The oscillations are very small because the lattice structure is lost in part.