Peer Assessments (https://class.coursera.org/smac-001/human_grading/)
/ Homework Session 3: Two-dimensional liquids and solids
Help (https://class.coursera.org/smac-001/help/peergrading?url=https%3A%2F%2Fclass.coursera.org%2Fsmac-001%2Fhuman_grading%2Fview%2Fcourses%2F971628%2Fassessments%2F6%2Fresults%2Fmine)

Submission Phase

1. Do assignment ✓ (/smac-001/human_grading/view/courses/971628/assessments/6/submissions)

Evaluation Phase

2. Evaluate peers ✓ (/smac-001/human_grading/view/courses/971628/assessments/6/peerGradingSets)

Results Phase

3. See results ✓ (/smac-001/human_grading/view/courses/971628/assessments/6/results/mine)

Your effective grade is 21

Your unadjusted grade is 21, which is simply the grade you received from your peers.

See below for details.

A In this homework session 3 of **Statistical Mechanics: Algorithms and Computations**, we study the liquid-solid phase transition in hard disks. This transition was first observed in epochal numerical simulations of Alder and Wainwright (1962). It is the first phase transition identified in continous two-dimensional systems.

In our homework session, the box is a square of length 1.0, with periodic boundary conditions and the algorithm is made to work for N = k^2 disks. Before starting, please remind yourself of the following modulo operations in Python:

```
 1.2 % 1.0 = 0.2
-0.2 % 1.0 = 0.8
```

Note the second line: the modulo "%" conveniently folds negative floating point numbers into the real interval [0.0, 1.0] and, in our case, into the box.

A1 In this section we provide **preparation programs**:

PREPARATION PROGRAM 1:

```
import pylab

def show_conf(L, sigma, title, fname):
    pylab.axes()
    for [x, y] in L:
        for ix in range(-1, 2):
            for iy in range(-1, 2):
                cir = pylab.Circle((x + ix, y + iy), radius = sigma,  fc = 'r')
                pylab.gca().add_patch(cir)
    pylab.axis('scaled')
    pylab.title(title)
    pylab.axis([0.0, 1.0, 0.0, 1.0])
    pylab.savefig(fname)
    pylab.show()

L = [[0.9, 0.9]]
sigma = 0.4
show_conf(L, sigma, 'test graph', 'one_disk.png')
```

PREPARATION PROGRAM 2:

```
import os, random

filename = 'disk_configuration.txt'
if os.path.isfile(filename):
    f = open(filename, 'r')
    L = []
    for line in f:
        a, b = line.split()
        L.append([float(a), float(b)])
    f.close()
    print 'starting from file', filename
else:
    L = []
    for k in range(3):
        L.append([random.uniform(0.0, 1.0), random.uniform(0.0, 1.0)])
    print 'starting from scratch'
L[0][0] = 3.3
f = open(filename, 'w')
for a in L:
   f.write(str(a[0]) + ' ' + str(a[1]) + '\n')
f.close()
```

Download (cut-and-paste) and run **Preparation Program 1**. Explain in a few words each:

1. Which object does it draw?
2. What is the color of the object, and where is it specified?
3. How does it implement periodic boundary conditions?
4. What makes that you can see it on the computer screen?
5. Does this program create a file? What is this file's name? How would you change it?

1. It draws a circle using **pylab.Circle** function.

2. The color is red specified using fc parameter. [cir = pylab.Circle((x + ix, y + iy), radius = sigma, **fc = 'r'**)]

3. The following code segment does the implementation of periodic boundary.

```
    for [x, y] in L:
        for ix in range(-1, 2):
            for iy in range(-1, 2):
                cir = pylab.Circle((x + ix, y + iy), radius = sigma,  fc = 'r')
                pylab.gca().add_patch(cir)
```

For each (x,y), there are nine circles drawn with centers at (x-1, y-1), (x-1, y), (x-1, y+1), (x, y-1), (x, y), (x, y+1), (x+1, y-1), (x+1, y) and (x+1, y+1). The relevant portions of these nine circles (i.e. those lying inside the box which goes from 0 to 1 in x and y dimensions) are kept. This enables "wrapping up" of the disks across the edges and corners, thereby, implementing the periodic boundaries.

4. **pylab.show()** asks the function to show the graph on screen.

5. Yes. The file name is **"one_disk.png"**. One can edit the file name to adapt the file name.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

A1
Here, you are asked to evaluate your fellow student's understanding of a Python program in five questions.
You can give a total of 2 points.
Give 0 points if 0 or 1 questions are answered correctly
Give 1 points if 2 or 3 questions are answered correctly
Give 2 points if 4 or 5 questions are answered correctly

Take into account that students are not yet supposed to be Python experts, so do allow some imprecision in the answers.
An example of an answer that would receive full score is as follows: (NB: remarks in [ ] are optional)
1/ It draws a circle
2/ It simply draws the nine images of the disk at x+/- 1 ,y +/- 1.
[With x, y in the box, this is OK - this remark is optional]
3/ It is red, as specified in the fc = 'r'
4/ pylab.show() flashes the figure on the computer screen
5/ It creates a file 'one_disk.png' (this corresponds to the object fname)
(the answer: it creates a file filename is also OK). This can be changed
by changing the text.
POINTS - DESCRIPTION
0 points if 0 or 1 questions are answered correctly
1 points if 2 or 3 questions are answered correctly

2 points if 4 or 5 questions are answered correctly

Score from your peers: **2**

A2 Make sure you do not have a file "disk_configuration.txt" in your working directory. Then download (cut-and-paste) and run the **Preparation Program 2**. Run **Preparation Program 2** a second time. Explain in a few words each:

1. What does the "if os.path.isfile(filename)" condition test?
2. What is the difference between "f = open(filename, 'r')" and "f = open(filename, 'w')"?
3. What is the meaning of "a, b = line.split()"
4. What is the meaning of "f.write(str(a[0]) + ' ' + str(a[1]) + '\n')", and in particular of the "\n" ?
5. What is this program's potential use?

1. "if os.path.isfile(filename)" condition tests if there is a file with name "filename" exists.

2. "f = open(filename, 'r')" opens a file for reading and "f = open(filename, 'w')" opens a file for writing.

3. "a, b = line.split()" uses an in-built function of python "split" to split the line entries of line "line" into two and stores them as a and b.

4. "f.write(str(a[0]) + ' ' + str(a[1]) + '\n')" writes the values a[0] and a[1] after converting them to string in the file referenced by file handle f. The ' ' adds a space between the two values and and '\n' adds a new line character to ensure that the next set of entries will begin from the start of the next line.

5. The file name "disk_configuration.txt"suggests the possible use of this file to store the disk configurations and read during a simulation.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

A2
Here, you are asked to evaluate your fellow student's understanding of a Python program in five questions.

You can give a total of 2 points.
Give 0 points if 0 or 1 questions are answered correctly
Give 1 points if 2 or 3 questions are answered correctly
Give 2 points if 4 or 5 questions are answered correctly

Take into account that students are not yet supposed to be Python experts, so do allow some imprecision in the answers.

An example of an answer that would receive the full score is as follows: (NB: remarks in

[ ] are optional)

1/ It checks for the existence of the file filename
2/ 'r' stands for read access, 'w' stands for 'write' access [note that 'w' implies read access]
3/ line is a string, it is cut into two at the first space
4/ the floats a[0] and a[1] are transformed into a string with a space in the middle and a newline at the end
5/ This allows to start a new simulation from scratch, or to start from previous results, if available. Final results are written onto file.

POINTS - DESCRIPTION
0 points if 0 or 1 questions are answered correctly
1 points if 2 or 3 questions are answered correctly
2 points if 4 or 5 questions are answered correctly

Score from your peers: **2**

B In this section, you write **your own Markov-chain Monte Carlo simulation program for hard disks**, "my_markov_disks.py" (or any other name that you like). my_markov_disks.py is relatively unsophisticated, but it is correct. It supports input-output and computes a non-trivial order parameter.

B1 To construct my_markov_disks.py, start from markov_disks_box.py presented in lecture 2. Copy-and-paste the function dist() from direct_disks_multirun.py (presented by Alberto in Tutorial 2) into my_markov_disks.py. Modify the algorithm for periodic boundary conditions. Run this algorithm for four disks. Attention: there are no walls and no more wall collisions. Use the modulo operator discussed in this homework's introductory paragraph to ensure that, after each accepted move, x and y positions of each disk are folded back into the interval 0.0 <= x <

Upload your program.

```
import random, math

def dist(x,y):
    d_x = abs(x[0] - y[0]) % 1.0
    d_x = min(d_x, 1.0 - d_x)
    d_y = abs(x[1] - y[1]) % 1.0
    d_y = min(d_y, 1.0 - d_y)
    return  math.sqrt(d_x**2 + d_y**2)

L = [[0.25, 0.25], [0.75, 0.25], [0.25, 0.75], [0.75, 0.75]]
sigma = 0.15
delta = 0.1
n_steps = 1000
for steps in range(n_steps):
    a = random.choice(L)
    b = [a[0] + random.uniform(-delta, delta), a[1] + random.uniform(-delta, delta)]
    min_dist = min(dist(b, c) for c in L if c != a)
    if not (min_dist < 2.0 * sigma):
        a[:] = b
        a[0] = a[0] % 1
        a[1] = a[1] % 1
print L
```

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

B1
Here you are asked to check that a submitted Python program is OK.
There are two points to check
1/ Check that the correct dist() function was implemented.
2/ Check that the pair collision check in markov_dist_box.py:
min_dist = min((b[0] - c[0]) ** 2 + (b[1] - c[1]) ** 2 for c in L if c != a)
is replaced by something equivalent to
min_dist_sq = min(dist(b, c) for c in L if c != a)

You can give a total of 2 points.
Give 0 points if none of the aspects was adressed successfully
Give 1 points if one of the aspects was adressed successfully
Give 2 points if both aspects were adressed successfully.

Note that a program receiving 2 points should be able to run, even though
you are not obliged to download (cut and paste) and run it.

Below a program that (Staff hopes) would received full score

```
import math, random

def dist(x, y):
    d_x = abs(x[0] - y[0]) % 1.0
    d_x = min(d_x, 1.0 - d_x)
    d_y = abs(x[1] - y[1]) % 1.0
    d_y = min(d_y, 1.0 - d_y)
    return d_x**2 + d_y**2

N = 4
eta = 0.72
L = [[0.25, 0.25], [0.25, 0.75], [0.75, 0.25], [0.75, 0.75]]
sigma = math.sqrt(eta / N / math.pi)
sigma_sq = sigma ** 2
delta = 0.3 * sigma
n_steps = 10000
for steps in range(n_steps):
    a = random.choice(L)
    b = [a[0] + random.uniform(-delta, delta), a[1] + random.uniform(-de
lta, delta)]
    min_dist_sq = min(dist(b, c) for c in L if c != a)
    if  min_dist_sq > 4.0 * sigma ** 2:
        a[:] = [b[0] % 1.0, b[1] % 1.0]
    if steps % 100 == 0: print L
```

POINTS - DESCRIPTION

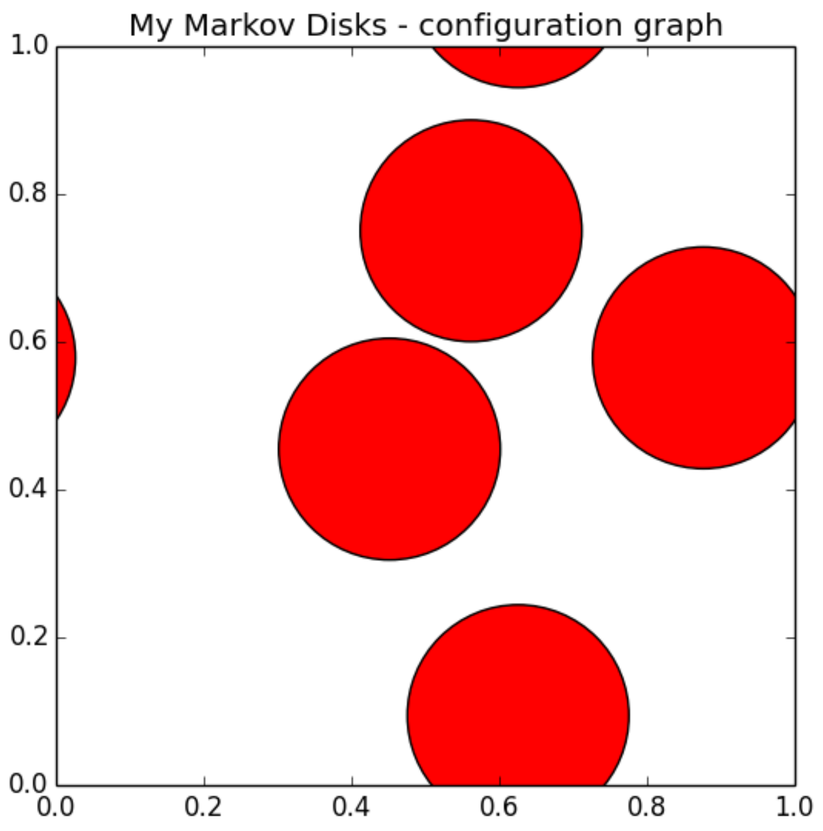
Give 0 points if none of the aspects was adressed successfully
Give 1 points if one of the aspects was adressed successfully
Give 2 points if both aspects were adressed successfully.

Score from your peers: **2**

B2 Further modify my_markov_disks.py by incorporating **Preparation program 1** so that you can plot the *final configuration* produced by it. Do not attempt to plot intermediate configurations. This often poses problems in Python.

Upload your program and one such configuration (as a graphics file) produced by my_markov_disks.py. Make sure it invokes periodic boundary conditions (at least one disk should be cut up into at least two pieces).

```
import random, math, pylab

def show_conf(L, sigma, title, fname):
    pylab.axes()
    for [x, y] in L:
        for ix in range(-1, 2):
            for iy in range(-1, 2):
                cir = pylab.Circle((x + ix, y + iy), radius = sigma,  fc = 'r')
                pylab.gca().add_patch(cir)
    pylab.axis('scaled')
    pylab.title(title)
    pylab.axis([0.0, 1.0, 0.0, 1.0])
    pylab.savefig(fname)
    pylab.show()

def dist(x,y):
    d_x = abs(x[0] - y[0]) % 1.0
    d_x = min(d_x, 1.0 - d_x)
    d_y = abs(x[1] - y[1]) % 1.0
    d_y = min(d_y, 1.0 - d_y)
    return  math.sqrt(d_x**2 + d_y**2)

L = [[0.25, 0.25], [0.75, 0.25], [0.25, 0.75], [0.75, 0.75]]
sigma = 0.15
delta = 0.1
n_steps = 1000
for steps in range(n_steps):
    a = random.choice(L)
    b = [a[0] + random.uniform(-delta, delta), a[1] + random.uniform(-delta, delta)]
    min_dist = min(dist(b, c) for c in L if c != a)
    if not (min_dist < 2.0 * sigma):
        a[:] = b
        a[0] = a[0] % 1
        a[1] = a[1] % 1
print L

show_conf(L, sigma, 'My Markov Disks - configuration graph', 'four_disks.png')
```



Evaluation/feedback on the above work

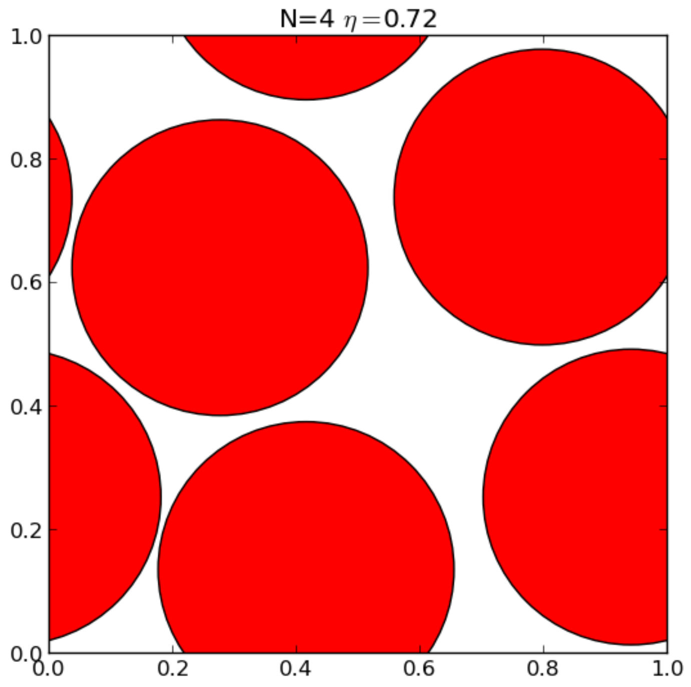
Note: this section can only be filled out during the evaluation phase.

B2
Here you are asked to check that a submitted Python program is OK and that the uploaded figure is ok.

There are three aspects to check:
1/ The function show_conf should have been copied
2/ It should be called at the end of the simulation, as in the below line...
show_conf(L, 'N='+str(N)+' $\eta =$'+str(eta), 'configuration_'+str(N)+'_'+str(eta)+'.png')
... also we did not ask for such a fancy filename
3/ You should check that the configuration actually shows ok, and that it handles the periodic boundary conditions

You can give a total of 2 points.
Give 0 points if none of the aspects was adressed successfully
Give 1 points if one or two of the aspects was adressed successfully
Give 2 points if all three aspects were adressed successfully.

Note that a program receiving full score should actually run, even though you are not obliged to download (cut and paste) and run it in order to check this.
Below a configuration and a program that (Staff hopes) would received full score


N=4 $\eta=0.72$
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

```
import math, random, pylab

def show_conf(L, title, fname):
    pylab.axes()
    for [x, y] in L:
        for delx in range(-1, 2):
            for dely in range(-1, 2):
                cir = pylab.Circle((x + delx, y + dely), radius = sigma,
  fc = 'r')
                pylab.gca().add_patch(cir)
    pylab.axis('scaled')
    pylab.title(title)
    pylab.axis([0.0, 1.0, 0.0, 1.0])
    pylab.savefig(fname)
    pylab.show()

def dist(x, y):
    d_x = abs(x[0] - y[0]) % 1.0
    d_x = min(d_x, 1.0 - d_x)
    d_y = abs(x[1] - y[1]) % 1.0
    d_y = min(d_y, 1.0 - d_y)
    return d_x**2 + d_y**2

N = 4
eta = 0.72
L = [[0.25, 0.25], [0.25, 0.75], [0.75, 0.25], [0.75, 0.75]]
sigma = math.sqrt(eta / N / math.pi)
sigma_sq = sigma ** 2
delta = 0.3 * sigma
n_steps = 1000
for steps in range(n_steps):
    a = random.choice(L)
    b = [a[0] + random.uniform(-delta, delta), a[1] + random.uniform(-de
lta, delta)]
    min_dist_sq = min(dist(b, c) for c in L if c != a)
    if  min_dist_sq > 4.0 * sigma ** 2:
        a[:] = [b[0] % 1.0, b[1] % 1.0]
show_conf(L, 'N='+str(N)+' $\eta =$'+str(eta), 'configuration_'+str(N)+'
_'+str(eta)+ '.png')
```

POINTS - DESCRIPTION

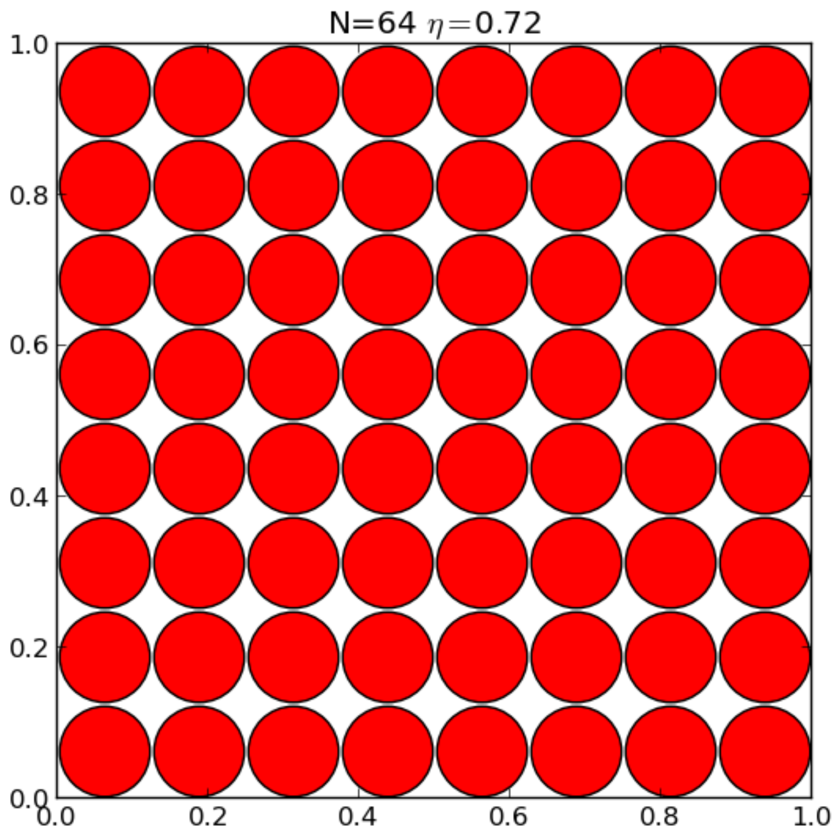
Give 0 points if none of the aspects was addressed successfully
Give 1 points if one or two of the aspects was addressed successfully
Give 2 points if all three aspects were addressed successfully.

Score from your peers: **2**

B3 Further modify my_markov_disks.py by incorporating **Preparation program 2**, so that it can obtain its initial configuration either from a file (if it exists) or otherwise from the program itself. To generate its own initial configuration, generalize the four-disk square-lattice initial configuration with...

```
L = [[0.25, 0.25], [0.25, 0.75], [0.75, 0.25], [0.75, 0.75]]
```

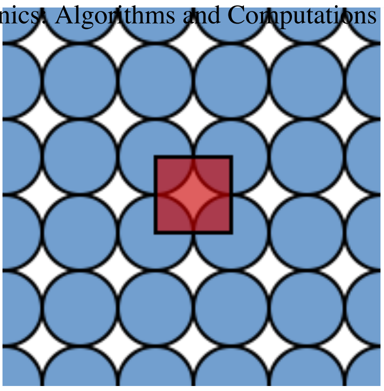
to variable N = k^2 disks. The initial condition for k=8 (N=64) is shown in the figure.
NOTE ADDED (WK 02-24-2014): As already indicated on the forum, choose a stepsize \propto sigma, for example 0.3 sigma, or 0.5 sigma. The value itself in uncritical.



Use k = int(math.sqrt(N) + 0.5). In your program, specify the density eta and the number of disks N rather than the radius sigma. The program should work for N equal to a square number, not for general N. The final configuration should then be saved to file, as indicated in Preparation program 2, ready for reuse at the next run of my_markov_disks.py.

- Indicate up to which density the square-lattice initial configuration is legal.
- Upload your program
- Upload the initial configuration (as a graphics file) for N = 256 at density eta= 0.72. (Note that you can do this easily by setting n_steps = 0.)

For the square lattice, there are $\sqrt{N}$ disks in each row/column and the maximum density will occur when $\sqrt{N}2\sigma = L$. This is shown below.



$\eta = \frac{N\pi\sigma^2}{L^2}$. Therefore, $\eta_{square packing} = \frac{N\pi\sigma^2}{(2\sqrt{N}\sigma)^2} = \frac{\pi}{4} \simeq 0.785$.

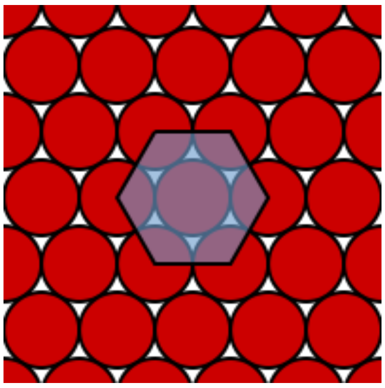
Alternatively, one can just focus on the shaded square region above and obtain density by dividing area occupied by four quarter circles to the area of shaded square. If radius is σ,

$$\eta_{square packing} = \frac{4\,\frac{\pi\sigma^2}{4}}{(2\sigma)^2} = \frac{\pi}{4} \simeq 0.785$$

Therefore, the square lattice initial configuration is only legal for $\eta \leq 0.785$.

EXTRA: JUST FOR MORE INSIGHTS

The **maximum possible density** will be for the hexagonal close packing of disks as shown below.



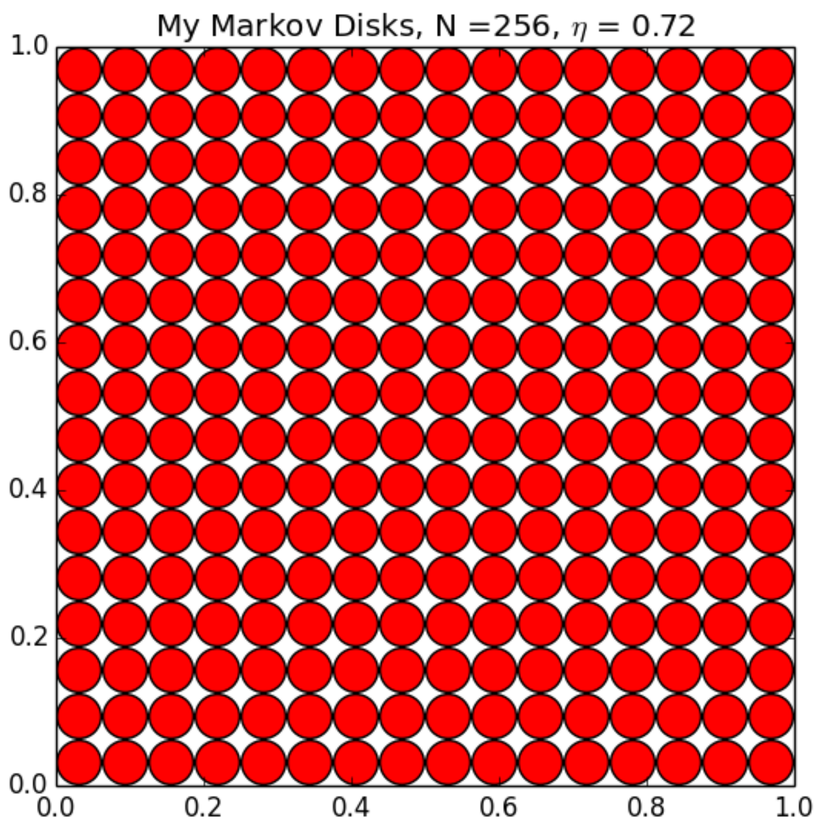
It is easily obtained by focusing on the shaded hexagonal region above. One needs to divide the area occupied by disks to the area of hexagonal region to get the density. If radius is σ,

$$\eta_{hexagonal packing} = \frac{\pi\sigma^2 + 6\,\frac{\pi\sigma^2}{3}}{6\,\frac{\sqrt{3}(2\sigma)^2}{4}} = \frac{\pi}{2\sqrt{3}} \simeq 0.907$$

Program code:

```
import os, random, math, pylab

def show_conf(L, sigma, title, fname):
    pylab.axes()
    for [x, y] in L:
        for ix in range(-1, 2):
            for iy in range(-1, 2):
                cir = pylab.Circle((x + ix, y + iy), radius = sigma,  fc = 'r')
                pylab.gca().add_patch(cir)
    pylab.axis('scaled')
    pylab.title(title)
    pylab.axis([0.0, 1.0, 0.0, 1.0])
    pylab.savefig(fname)
    pylab.show()

def dist(x,y):
    d_x = abs(x[0] - y[0]) % 1.0
    d_x = min(d_x, 1.0 - d_x)
    d_y = abs(x[1] - y[1]) % 1.0
    d_y = min(d_y, 1.0 - d_y)
    return  math.sqrt(d_x**2 + d_y**2)

N=256
eta=0.72
sqrt_N = int(math.sqrt(N))

sigma = math.sqrt(eta/(math.pi*N))
inter_disk_distance = (1.0-(sqrt_N*2.0*sigma))/sqrt_N

filename = 'N_disk_configuration.txt'
if os.path.isfile(filename):
    f = open(filename, 'r')
    L = []
    for line in f:
        a, b = line.split()
        L.append([float(a), float(b)])
    f.close()
    print 'starting from file', filename
else:
    L = []
    for k in range(sqrt_N):
                x = (2*k+1)*(sigma+inter_disk_distance/2.0)
                for l in range(sqrt_N):
                        y = (2*l+1)*(sigma+inter_disk_distance/2.0)
                        L.append([x, y])
    print 'starting from scratch'

delta = 0.1
n_steps = 0

for steps in range(n_steps):
    a = random.choice(L)
    b = [a[0] + random.uniform(-delta, delta), a[1] + random.uniform(-delta, delta)]
    min_dist = min(dist(b, c) for c in L if c != a)
    if not (min_dist < 2.0 * sigma):
        a[:] = b
        a[0] = a[0] % 1
        a[1] = a[1] % 1

# Write all disk positions in file to read during next run
f = open(filename, 'w')
for a in L:
   f.write(str(a[0]) + ' ' + str(a[1]) + '\n')
f.close()

title = 'My Markov Disks, N =' + str(N) +', $\eta$ = '+ str(eta)
show_conf(L, sigma, title, 'N_disks.png')
```

The initial configuration is shown below for N = 256 at $\eta = 0.72$



Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

B3
Here you are asked to check that a submitted Python program is OK, that the uploaded figure is correct and that the limiting density is correctly evaluated to pi/4 = 0.785 that the uploaded figure is ok.

There are thus five aspects to check:

1/ The initial configuration should have been written similar to:

```
L = [ [delxy + i * two_delxy, delxy + j * two_delxy] \
    for i in range(N_sqrt) for j in range(N_sqrt)]
```

2/ The radius sigma is computed through a formula similar to

```
sigma = math.sqrt(eta / N / math.pi)
```

3/ Preparation program 2 should have been correctly implemented

4/ The maximal density should have been evaluated to pi/4

5/ The initial (regular square) condition for 256 particles has been uploaded

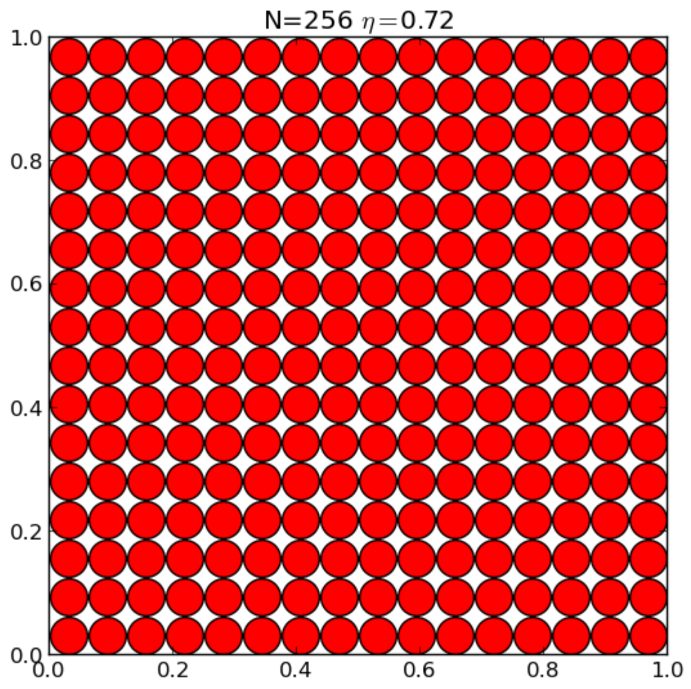
You can give a total of 3 points.
Give 0 points if none of the aspects was adressed correctly
Give 1 points if one of the aspects was adressed correctly
Give 2 points if two or three aspects were adressed correctly.
Give 3 points if four or five aspects were adressed correctly.

Note that a program receiving full score should actually run, even though you are not obliged to download (cut and paste) and run it in order to check this.
Below a program that (Staff hopes) would received full score. Note that the fancy filenames are not required, although they might provide inspiration.

```
import math, random, pylab, os

def show_conf(L, title, fname):
    pylab.axes()
    for [x, y] in L:
        for delx in range(-1, 2):
            for dely in range(-1, 2):
                cir = pylab.Circle((x + delx, y + dely), radius = sigma,
  fc = 'r')
                pylab.gca().add_patch(cir)
    pylab.axis('scaled')
    pylab.title(title)
    pylab.axis([0.0, 1.0, 0.0, 1.0])
    pylab.savefig(fname)
    pylab.show()

def dist(x, y):
    d_x = abs(x[0] - y[0]) % 1.0
    d_x = min(d_x, 1.0 - d_x)
    d_y = abs(x[1] - y[1]) % 1.0
    d_y = min(d_y, 1.0 - d_y)
    return d_x**2 + d_y**2

N =  64
eta = 0.72
filename = 'disk_configuration_' + '%.2f' % eta
if os.path.isfile(filename):
    f = open(filename, 'r')
    L = []
    for line in f:
        a, b = line.split()
        L.append([float(a), float(b)])
    f.close()
    print 'starting from file', filename
else:
    N_sqrt = int(math.sqrt(N) + 0.5)
    delxy = 1./ 2. / N_sqrt
    two_delxy = 2.0 * delxy
    L = [ [delxy + i * two_delxy, delxy + j * two_delxy] \
        for i in range(N_sqrt) for j in range(N_sqrt)]
    print 'starting from scratch'
sigma = math.sqrt(eta / N / math.pi)
sigma_sq = sigma ** 2
delta = 0.3 * sigma
n_steps = 10000
for steps in range(n_steps):
    a = random.choice(L)
    b = [a[0] + random.uniform(-delta, delta), a[1] + random.uniform(-de
lta, delta)]
    min_dist_sq = min(dist(b, c) for c in L if c != a)
    if  min_dist_sq > 4.0 * sigma ** 2:
```

```
    a[:] = [b[0] % 1.0, b[1] % 1.0]
show_conf(L, 'N='+str(N)+' $\eta =$'+str(eta), 'configuration_'+str(N)+'
_'+str(eta)+ '.png')
f = open(filename, 'w')
for a in L:
   f.write(str(a[0]) + ' ' + str(a[1]) + '\n')
f.close
```

Here a file with the initial condition



POINTS - DESCRIPTION

Give 0 points if none of the aspects was adressed correctly
Give 1 points if one of the aspects was adressed correctly
Give 2 points if two or three aspects were adressed correctly.
Give 3 points if four or five aspects were adressed correctly.

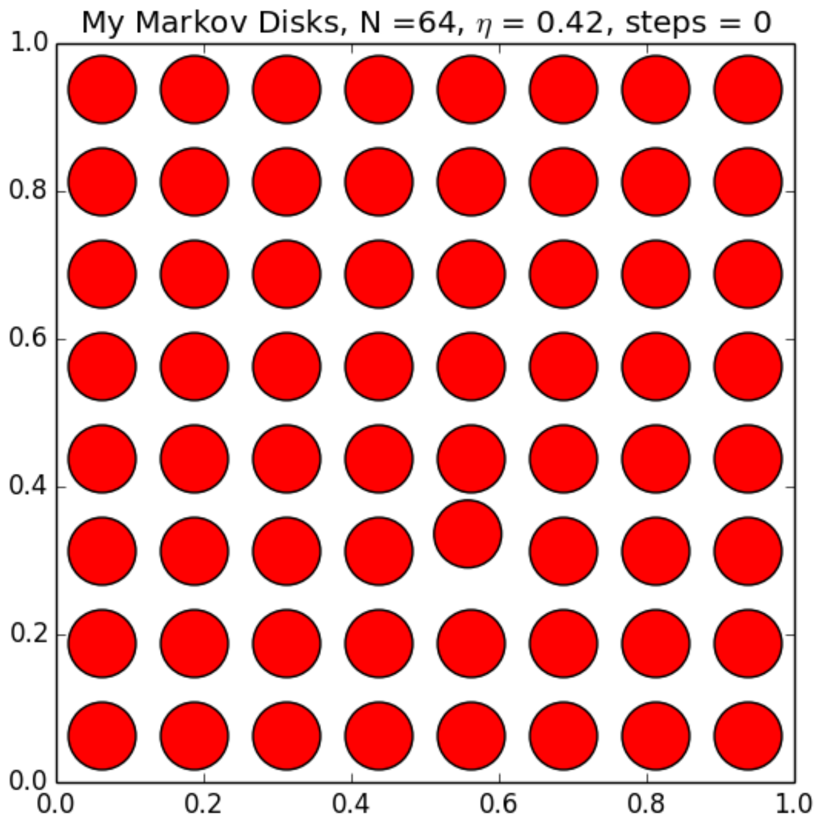
Score from your peers: **3**

B4 Run my_markov_disks.py for N = 64, n_steps = 10000, at eta = 0.42 from the square lattice initial condition, then repeatedly using as initial configuration the final configuration of the previous run. Explain what you see. Upload the initial configuration (as a graphics file) of the sequence of runs (set n_steps = 0 to plot it) and the final configuration (as a graphics file) you obtained from the entire sequence of runs.

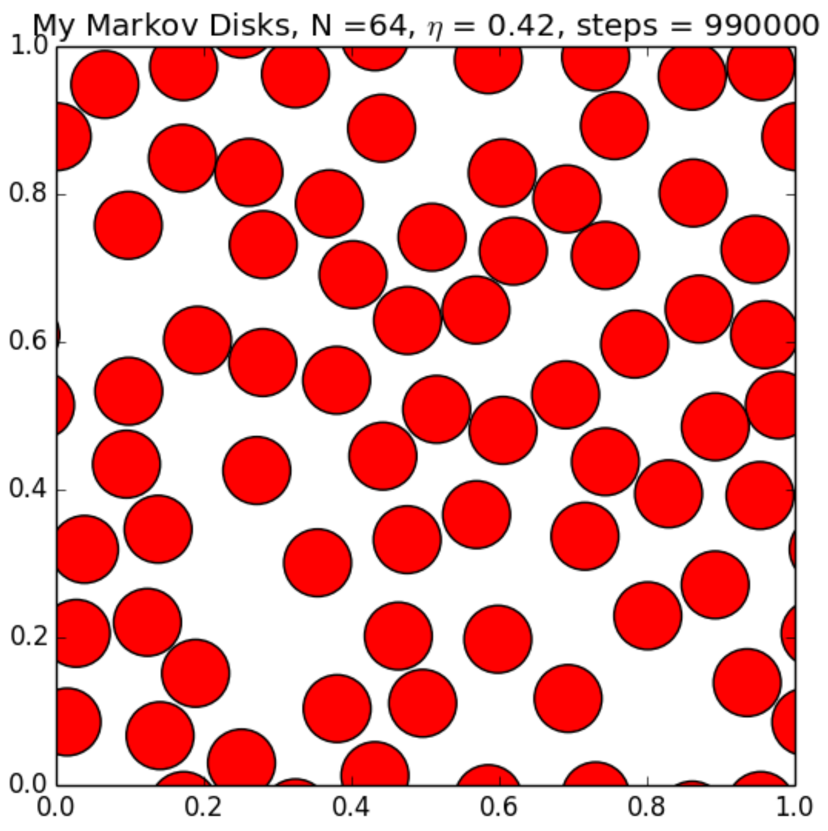
Below are the system snapshots for different n_steps. The simulation is run for n_steps=10 000 and the final structure is stored in a file. The simulation is continued with the stored state again for n_steps = 10 000. In total, 99 simulations were run resulting in total n_steps = 990 000.

The system gets randomized initially. However, slowly some sort of structure appears to be emerging. Some parts of the system are more aggregated than other parts. This happens due to the Asakura-Oosawa depletion interaction and causes unconnected local arrangements of disks at several parts of the system. At these places, the arrangement of disks resembles the hexagonal closed packing.

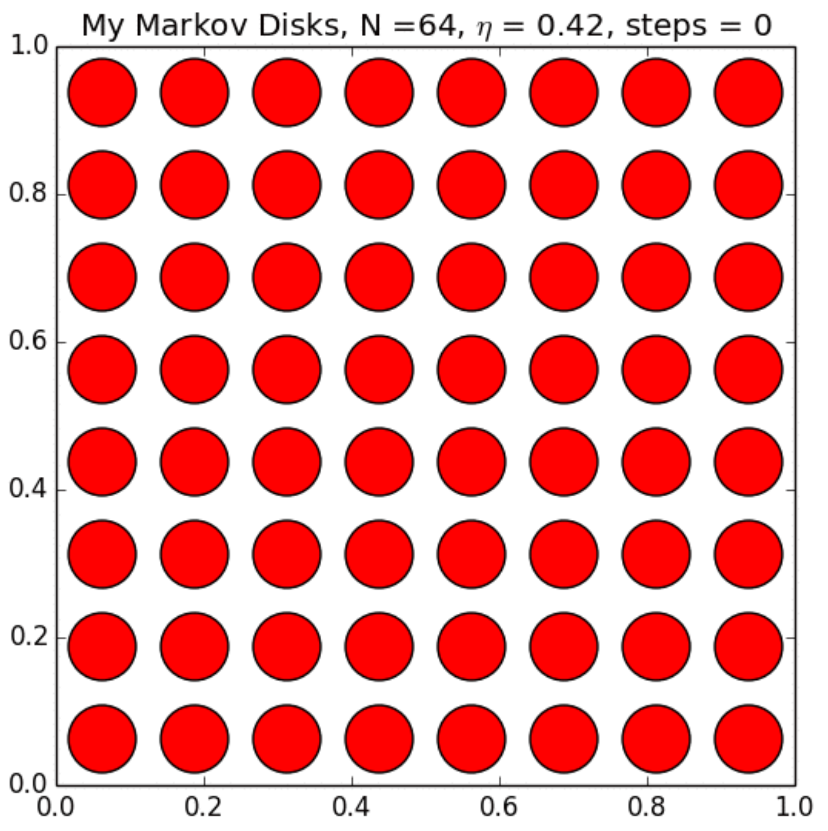
total n_steps = 0



total n_steps = 990 000



Movie of total n_steps = 0 to 990 000 (in a loop)



Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

B4

Here you are asked to check that the Python program of B3 is used correctly at low density and that some physical interpretation is provided.

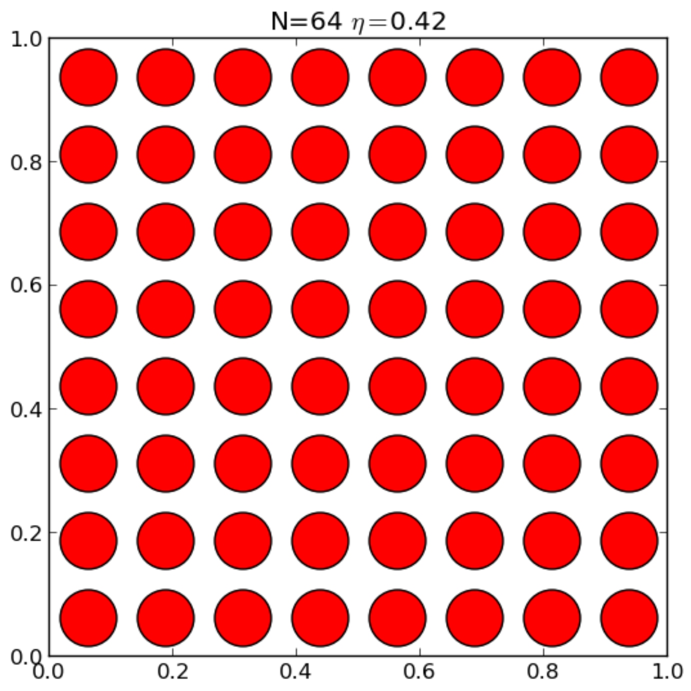
There are thus three aspects to check:
1/ The uploaded initial configuration should correspond to the square lattice at relatively small density (0.42).
2/ The final configuration should look liquid-like (like a gas, or a disordered system).
3/ There is some discussion that one sees a liquid (or a gas, or a disordered system).

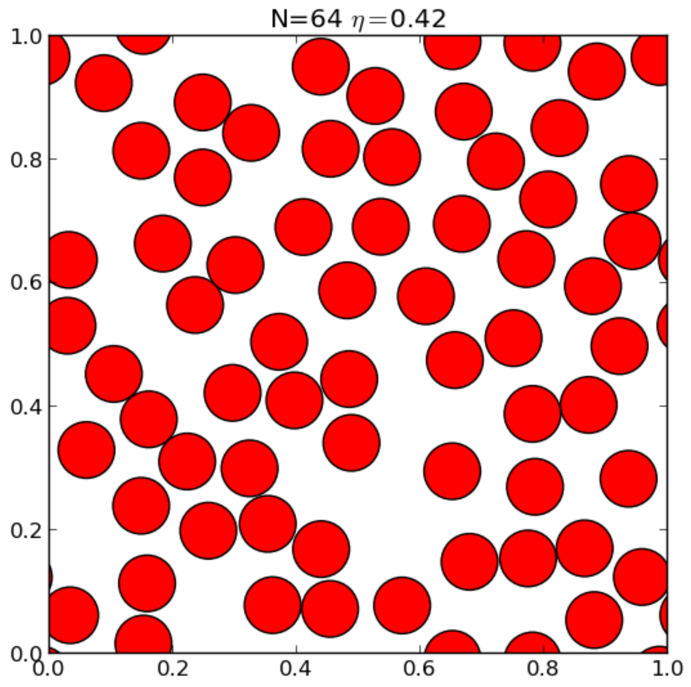
You can give a total of 2 points.

Give 0 points if none of the aspects was adressed correctly
Give 1 points if one or two of the aspects was adressed correctly
Give 2 points if all three of the aspects were adressed correctly

Below two files and an explanation that would received full score.

Explanation: there seems to be liquid (or gas-like) order in this system





POINTS - DESCRIPTION

Give 0 points if none of the aspects was adressed correctly
Give 1 points if one or two of the aspects was adressed correctly
Give 2 points if all three of the aspects were adressed correctly

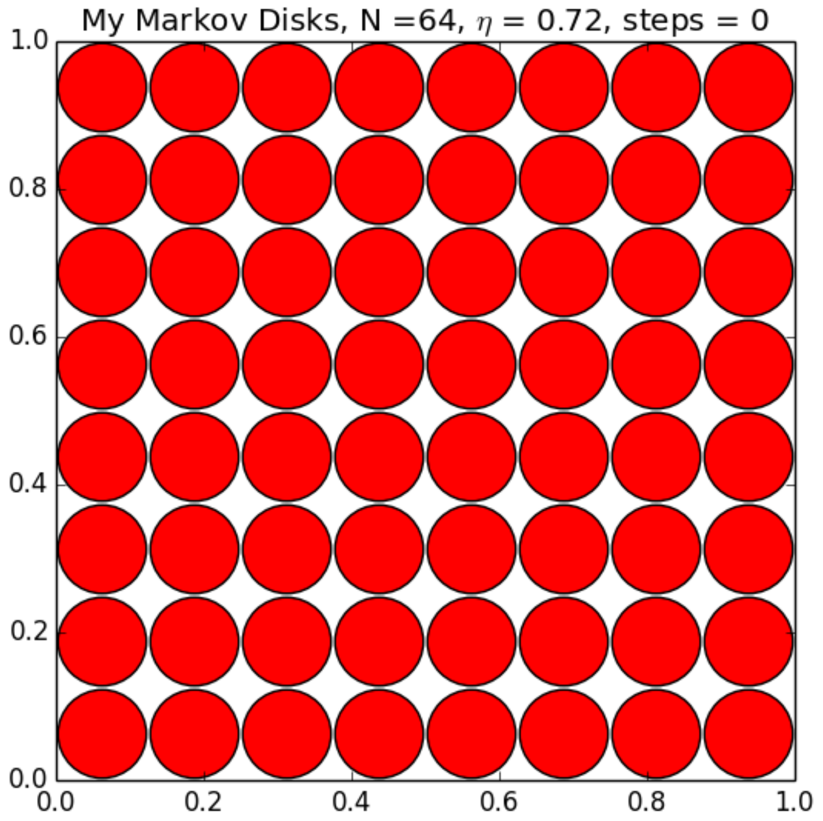
Score from your peers: **2**

B5 Run my_markov_disks.py for N = 64, n_steps = 10000, at eta = 0.72 from the square lattice initial condition, then at least ten times using as initial configuration the final configuration of the previous run. Explain what you see. Upload intermediate configurations (as graphics files) of the entire run and the final configuration you obtained from the entire sequence of runs. (You may join the graphics files into one big file).

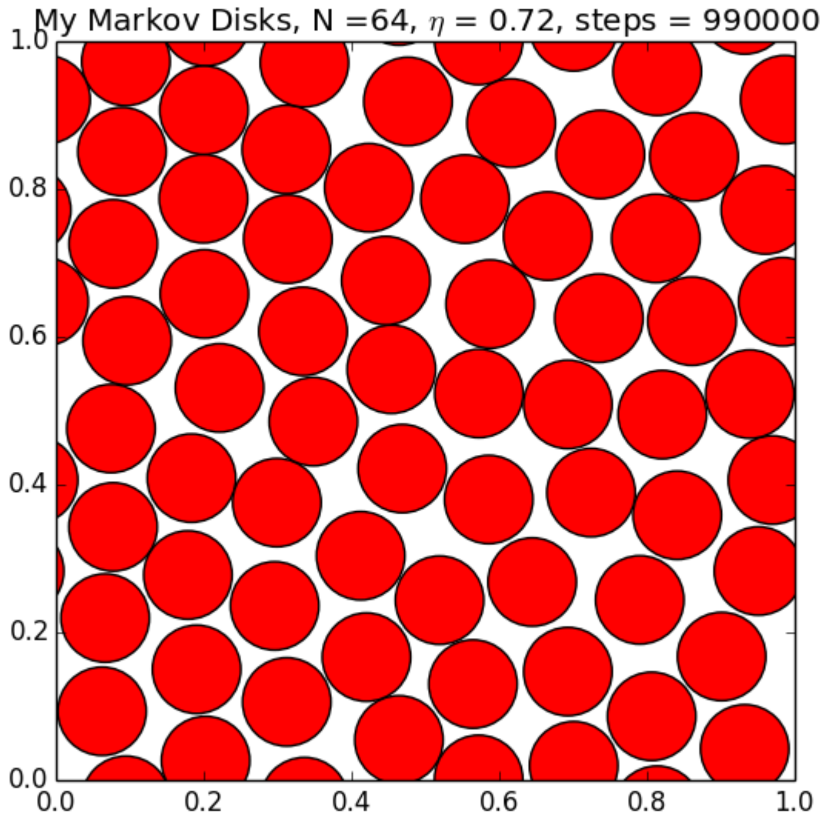
Below are the system snapshots for different n_steps. The simulation is run for n_steps=10 000 and the final structure is stored in a file. The simulation is continued with the stored state again for n_steps = 10 000. In this case, 99 simulations were ran resulting in total n_steps = 990 000. Below snapshots for 0 and 990 000 are provided followed by a slow movie in loop for the entire run.

At higher density of 0.72, the system looks packed. It's similar to phase transition from liquid phase (as in B4 above) to a more dense phase like a solid. The system is evolving very slowly due to lack of space to move around. After a long simulation, the system should converge to an arrangement where the entire system looks like hexagonal closed packing.

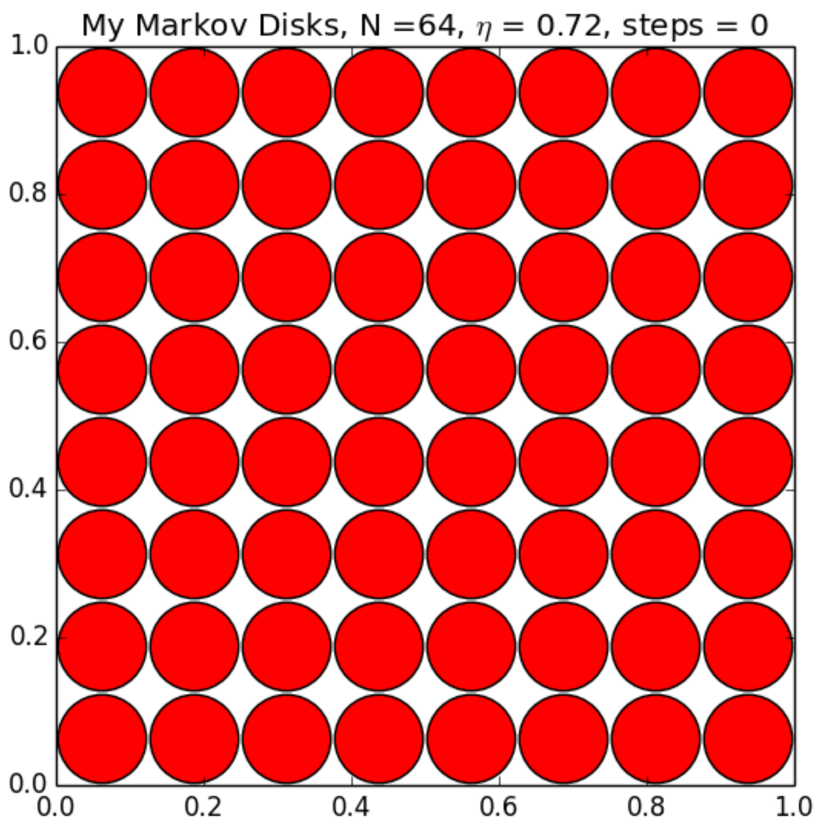
total n_steps = 0



total n_steps = 990 000



Movie of total n_steps = 0 to 990 000 (in a loop)



Evaluation/feedback on the above work

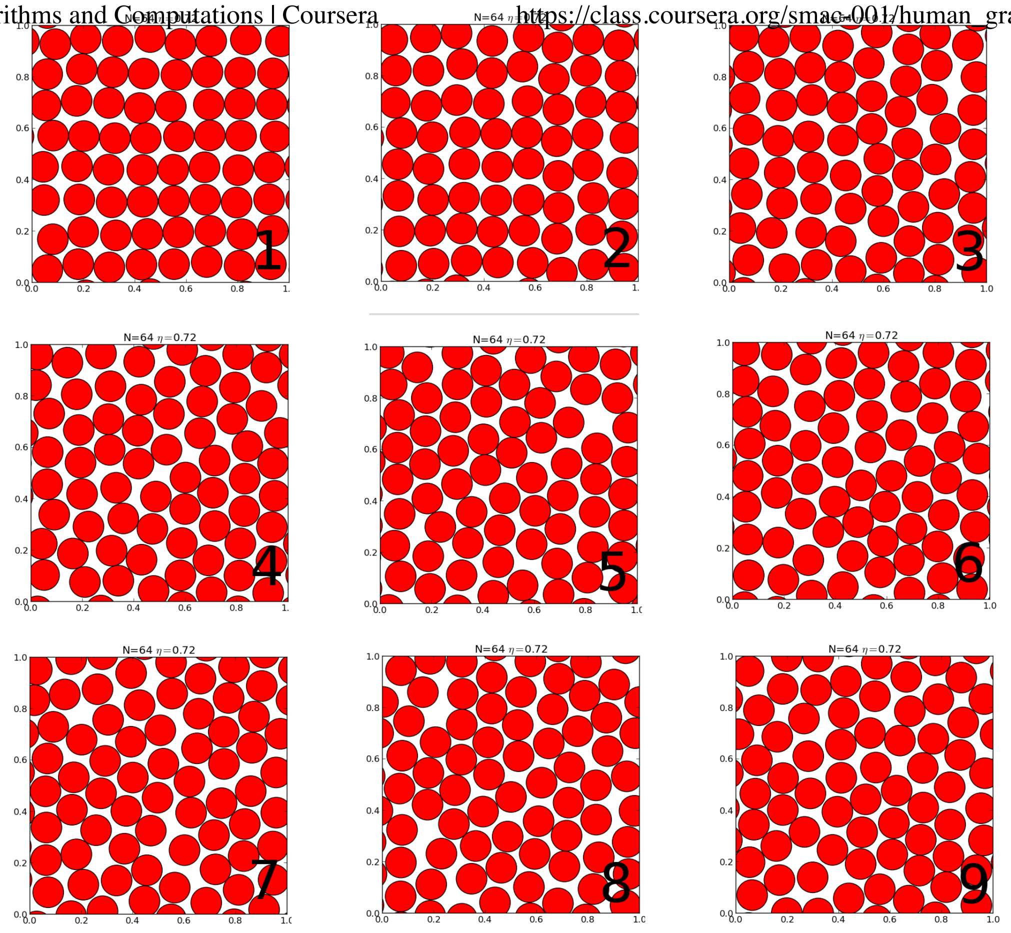
Note: this section can only be filled out during the evaluation phase.

B5 Here you are asked to check that the Python program of B3 is used correctly at high density and that some physical interpretation is provided.
There are thus two aspects to check:
1/ Some configurations have been uploaded. Initially, these configuration should carry some memory of the initial square configuration, but then
some hexagonal order should become predominant.
2/ It should be remarked that this system does not look like a liquid.

POINTS - DESCRIPTION

You can give a total of 2 points.
Give 0 points if none of the aspects was adressed correctly
Give 1 points if some work was done, but understanding was not reached
Give 2 points if the solution is satisfactory.

NB: Note that this is quite involved material, so don't be too harsh.



POINTS - DESCRIPTION

You can give a total of 2 points.

Give 0 points if none of the aspects was adressed correctly
Give 1 points if some work was done, but understanding was not reached
Give 2 points if the solution is satisfactory.

NB: Note that this is quite involved material, so don't be too harsh.

Score from your peers: **2**

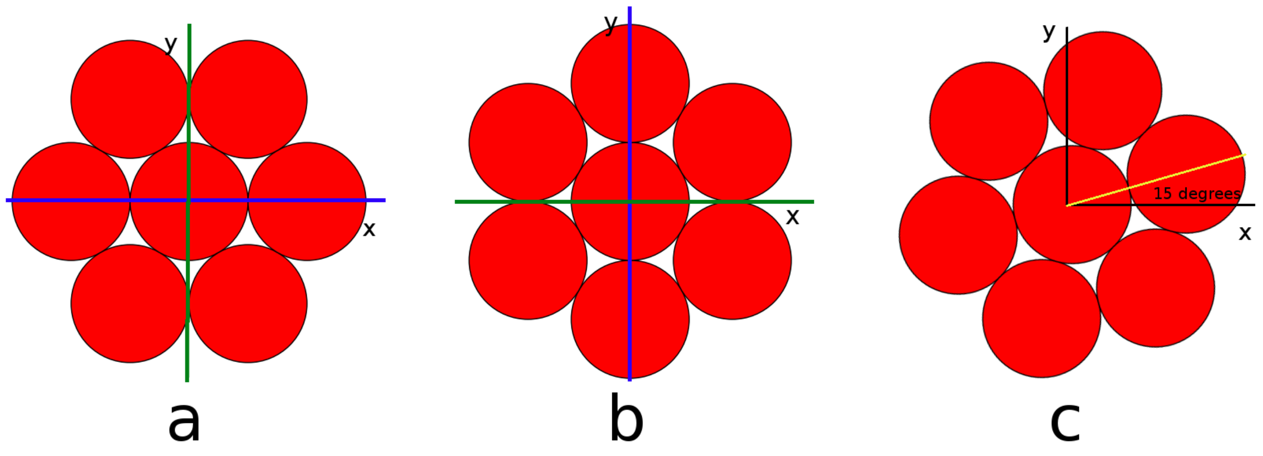
C This section is concerned with the quantitative interpretation of the findings of B4 and B5. It contains a programming assignment for which help is provided in form of a code snippet. Consider the local order parameter psi_6(i), of disk i:

$$\psi_6(i) = \frac{1}{\text{number of neighbors } j \text{ of } i} \sum_j \exp(6 i_{\text{imag}} \phi_{ij})$$

The local order parameter psi_6(i) is a complex number, i_imag is the complex number (0.0, 1.0) and phi_ij is the angle with respect to the x-axis, measured in radians. A "neighbor" j is a closeby other disk, we will take it to be a disk whose center is less than 2.8 sigma away from the center of disk i (there are more expert definition of neighborship, but this close proximity will be OK for our purposes).

NB: Note that the local order parameter is called psi (in small letters)

C1 Consider the disks a, b, and c, **the central disks** in the following figures:



1. What is the value of psi_6(a) (a complex number)?
2. What is the value of psi_6(b) (a complex number)?
3. What is the value of psi_6(c) (a complex number)?
4. Compute psi_6(a) + psi_6(b) (a complex number).

1. Angles (ϕ_{ij}) are 0, Pi/3, 2Pi/3, Pi, 4Pi/3, 5Pi/3 i.e. 0, 60, 120, 180, 240 and 300 degrees.
2. Angles (ϕ_{ij}) are Pi/6, Pi/2, 5Pi/6, 7Pi/6, 3Pi/2 and 11Pi/6 i.e. 30, 90, 150, 210, 270 and 330 degrees.
3. Angles (ϕ_{ij}) are Pi/12, 5Pi/12, 3Pi/4, 13Pi/12, 17Pi/12 and 7Pi/4 i.e. 15, 75, 135, 195, 255 and 315 degrees.
4. Sum of 1 and 2

1. psi_6(a) = (1/6) * [exp(i*0) + exp(i*2Pi) + exp(i*4Pi) + exp(i*6Pi) + exp (i*8Pi) + exp (i*10Pi)] = (1/6) * 6 = **1**

2. psi_6(b) = (1/6) * [exp(i*Pi)+exp(i*3Pi)+exp(i*5Pi) + exp (i*7Pi) + exp (i* 9Pi) + exp (i*11Pi)] = (1/6) * -6 = **-1**

3. psi_6(c) = (1/6) * [exp(i*Pi/2)+exp(i*5Pi/2)+exp(i*9Pi/2) + exp (i*13Pi/2) + exp (i*17Pi/2) + exp (i*21Pi/2)] = (1/6)* 6i = **i**

4. psi_6(a) + psi_6(b) = 1 + (-1) = **0**

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

C1
Here you are asked to check some basic understanding. There are four aspects:

1/ for configuration a, the angles are 0, 60, 120 degrees , etc and 6 times the angle makes 0 % 360, so the psi_6 = (1,0)
2/ for configuration b, the angles are 30, 90, 150 degrees and 6 times the angle makes 180 % 360, so the psi_6 = (-1,0)
3/ for configuration c, the angles are 15, 75, degrees etc, and 6 times the angles

makes 90 % 360, so psi_6 = (0,1)
4/ the sum adds to zero

You can give a total of 2 points.

Give 0 points if none of the aspects was adressed correctly
Give 1 points if one or two of the aspects was adressed correctly
Give 2 points if three or four of the aspects were adressed correctly

Below an answer that would received full score.

1/ psi_6 = (1,0) (equivalently: psi_6 = 1)
2/ psi_6 = (-1,0) (equivalently: psi_6 = -1)
2/ psi_6 = (0,1) (equivalently psi_6 = i)
4/ the sum is zero

Approximate values are accepted

POINTS - DESCRIPTION

Give 0 points if none of the aspects was adressed correctly
Give 1 points if one or two of the aspects was adressed correctly
Give 2 points if three or four of the aspects were adressed correctly

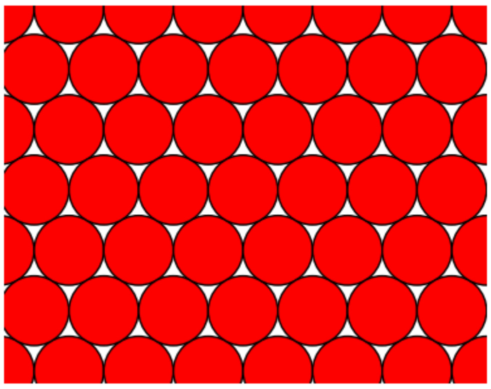
Score from your peers: **2**

C2 The global order parameter is defined as

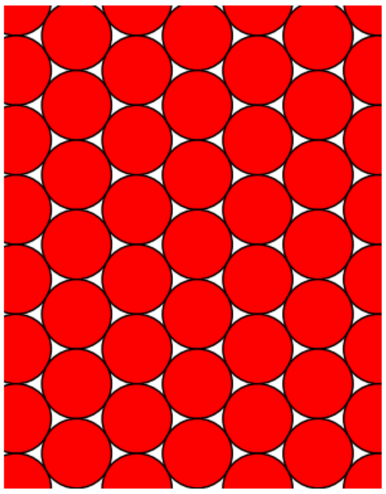
$$\Psi_6 = \frac{1}{N}\sum_i \psi_6(i)$$

(NB: Note the difference between Psi (capital letter) and psi.)

1. What is the value of Psi_6 for the below two configurations A and B? we suppose that they continue up to infinity.
2. What is the value of Psi_6 for a very large liquid configuration with eta = 0.42, (N very large, all correlation lengths much smaller than system size)?



A



B

1. Configuration A corresponds to C1.a and configuration B to C1.b. All the disks are arranged in same manner. So, ψ_6 is identical for all the disks. From C1, we can say of the value of ψ_6 is 1 and -1 for all disks of configurations A and B respectively. Therefore, $\Psi_6 = 1$ for Configuration A and $\Psi_6 = -1$ for configuration B.

2. As we noticed in B4 that for $\eta = 0.42$, there were some localized regions which were ordered and had hexagonal close packing. However, as expected, not all of these regions were aligned in the same direction as in configuration A or configuration B above. It's a mix of arrangements shown in configurations C1.A, C1.b, C1.c and several other hexagonal arrangements with different orientations.
So, for the given system, on an average, we expect approximately similar number of configurations having opposite signs for ψ_6 order parameter. Therefore, ψ_6 will cancel out in pairs and we'll have $\Psi_6 \approx 0$.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

C2
Here you are asked to check some basic understanding. There are the following aspects:

1/ For configuration A, the global order parameter is 1 (see discussion of C1)
2/ For configuration B, the global order parameter is -1 (see discussion of C1)
3/ For a liquid configuration, the global order parameter is close to zero. (as the individual contributions would average out).

You can give a total of 2 points:
Give 0 points if none of the aspects was adressed correctly
Give 1 points if one or two of the aspects was adressed correctly
Give 2 points if all three of the aspects were adressed correctly

Below an answer that would received full score.

1/ Psi_6 \sim 1;
  Psi_6 \sim -1
2/ Psi_6 \sim 0

POINTS - DESCRIPTION

Give 0 points if none of the aspects was adressed correctly
Give 1 points if one or two of the aspects was adressed correctly
Give 2 points if all three of the aspects were adressed correctly

Score from your peers: **2**

C3 It can be shown numerically that Psi_6 is finite for very large systems of hard disks at density eta > 0.72, as in our example. Incorporate the calculation of Psi_6 into my_markov_disks.py using the following code snippet. Note that you need the cmath library to perform complex arithmethic. Note also that 6j = (0.0, 6.0), in complex notation. The delx_dely function computes the distance vector, corrected for periodic boundary conditions.

Note added (02-22-2014): In checking for neighbors, we suppose in the below question that the **dist function** returns the **squared distance**

```
import cmath

def delx_dely(x, y):
    d_x = (x[0] - y[0]) % 1.0
    if d_x > 0.5: d_x -= 1.0
    d_y = (x[1] - y[1]) % 1.0
    if d_y > 0.5: d_y -= 1.0
    return d_x, d_y



def Psi_6(L, sigma_sq):
    sum_vector = 0j
    for i in range(N):
        vector  = 0j
        n_neighbor = 0
        for j in range(N):
            if dist(L[i], L[j]) < 2.8 **2 * sigma_sq and i != j:
                n_neighbor += 1
                dx, dy = delx_dely(L[j], L[i])
                angle = cmath.phase(complex(dx, dy))
                vector += cmath.exp(6.0j * angle)
        if n_neighbor > 0: vector /= n_neighbor
        sum_vector += vector
    return sum_vector / float(N)
```

Incorporate this snippet into my_markov_disks.py, and produce a plot of the mean absolute value of Psi_6 as a function of density eta for N = 64, up to density of eta = 0.72. For simplicity,

- you may start at eta = 0.72 (with a "well-thermalized" configuration)

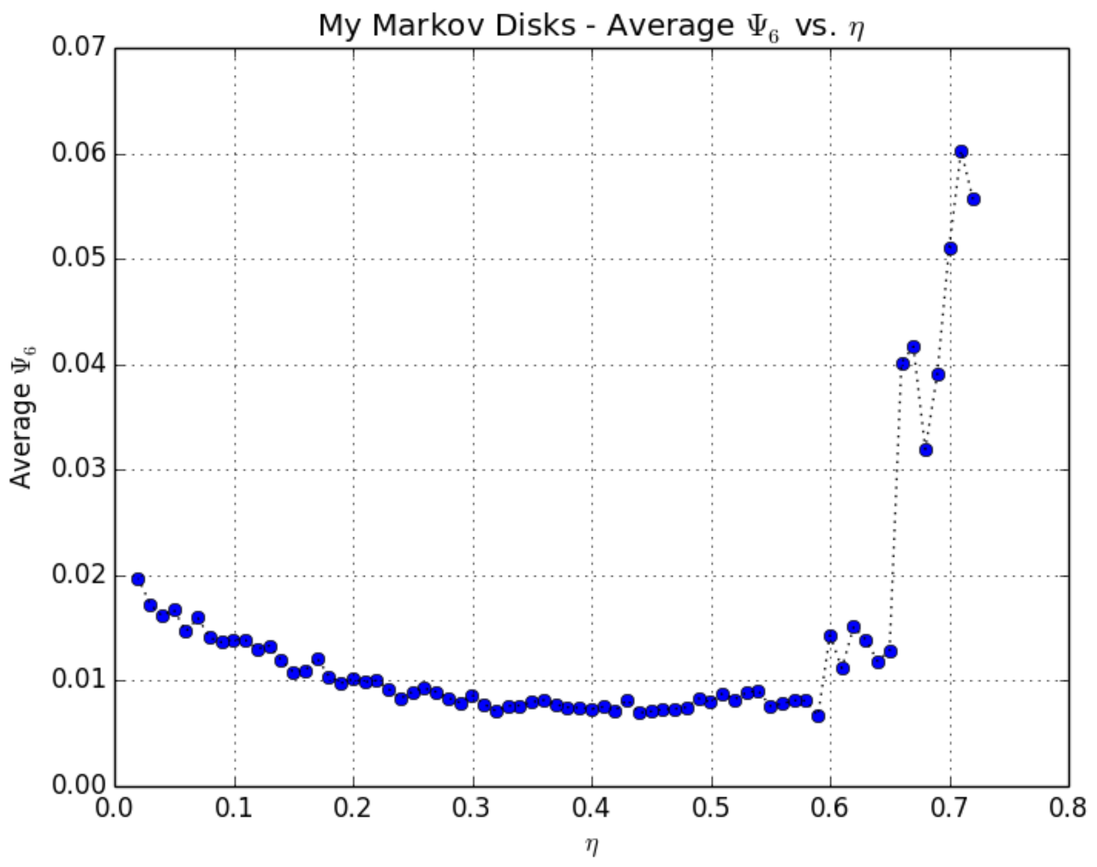
- compute |Psi_6| every 100 steps
- decrease the density by 0.01 after each 10,000 steps, using as initial
- configuration the final configuration at higher density

Your plot will be qualitative at high density, as we cannot reach large enough simulation times, even for 64 disks, but the procedure is OK. The finite values of |Psi_6| at small density is a finite-N effect.

- Upload the plot.
- Upload your final program.

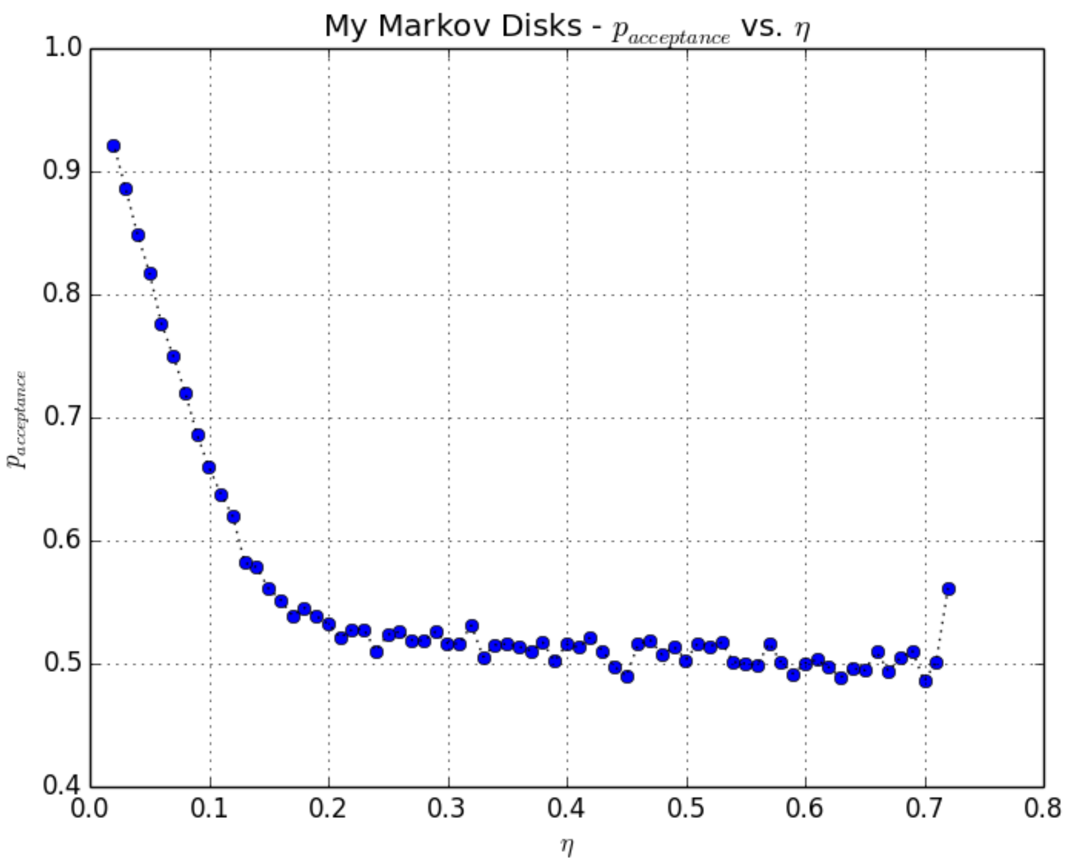
NB: Y**ou should realize that setting up your own versions of my_markov_disks.py constitutes a considerable achievement.**

NNB: The nature of this phase transition has been a mystery for several decades, and its detailed description is beyond the scope of this homework (in fact, there are two transitions at close-by densities). The seminal simulations of Metropolis et al (1953) used N = between 56 and 224 and n_steps = 20000. These simulation times are roughly 1000000 times smaller than the correlation times of the Markov chain for eta = 0.72, and this explains much of the confusion in the field. Mathematical estimates for the correlation time are unavailable. Likewise, the existence of the transition(s), while established beyond doubt by numerical simulations, is a mathematical mystery, unlike the two other transitions that we will treat in this course (Bose-Einstein condensation, ferromagnetic-paramagnetic transition).



```
import os, random, math, pylab, cmath, numpy

def delx_dely(x, y):
    d_x = (x[0] - y[0]) % 1.0
    if d_x > 0.5: d_x -= 1.0
    d_y = (x[1] - y[1]) % 1.0
    if d_y > 0.5: d_y -= 1.0
    return d_x, d_y

def Psi_6(L, sigma_sq):
    sum_vector = 0j
    for i in range(N):
        vector  = 0j
        n_neighbor = 0
        for j in range(N):
            if dist_sq(L[i], L[j]) < 2.8 **2 * sigma_sq and i != j:
                n_neighbor += 1
                dx, dy = delx_dely(L[j], L[i])
                angle = cmath.phase(complex(dx, dy))
                vector += cmath.exp(6.0j * angle)
        if n_neighbor > 0: vector /= n_neighbor
        sum_vector += vector
    return sum_vector / float(N)

def dist(x,y):
    d_x = abs(x[0] - y[0]) % 1.0
    d_x = min(d_x, 1.0 - d_x)
    d_y = abs(x[1] - y[1]) % 1.0
    d_y = min(d_y, 1.0 - d_y)
    return  math.sqrt(d_x**2 + d_y**2)

def dist_sq(x,y):
    d_x = abs(x[0] - y[0]) % 1.0
    d_x = min(d_x, 1.0 - d_x)
    d_y = abs(x[1] - y[1]) % 1.0
    d_y = min(d_y, 1.0 - d_y)
    return  (d_x**2 + d_y**2)

N=64
eta_start=0.72
eta_end=0.01

sqrt_N = int(math.sqrt(N))

sigma = math.sqrt(eta_start/(math.pi*N))
inter_disk_distance = (1.0-(sqrt_N*2.0*sigma))/sqrt_N

filename = 'N_C3_disk_configuration.txt'
if os.path.isfile(filename):
    f = open(filename, 'r')
    L = []
    for line in f:
        a, b = line.split()
        L.append([float(a), float(b)])
    f.close()
    print 'starting from file', filename
else:
    L = []
    for k in range(sqrt_N):
        x = (2*k+1)*(sigma+inter_disk_distance/2.0)
        for l in range(sqrt_N):
                y = (2*l+1)*(sigma+inter_disk_distance/2.0)
                L.append([x, y])
    print 'starting from scratch'

n_steps = 10000
Psi_6_list = []
eta_list = []
acc_prob_list = []
delta = 0.01

for eta in numpy.arange(eta_start, eta_end, -0.01):
        sigma = math.sqrt(eta/(math.pi*N))

        print 'sigma =', sigma
        print 'delta =', delta
        count_Psi_6 = 0
        total_Psi_6 = 0.0
        acc_steps = 0

        for steps in range(n_steps):
                a = random.choice(L)
                b = [a[0] + random.uniform(-delta, delta), a[1] + random.uniform(-delta
, delta)]
                min_dist = min(dist(b, c) for c in L if c != a)

                if not (min_dist < 2.0 * sigma):
                        a[:] = b
                        a[0] = a[0] % 1
                        a[1] = a[1] % 1
                        acc_steps += 1

                if steps > 0:
                        if (steps % 100) == 0:
                                count_Psi_6 +=1
                                total_Psi_6 += abs(Psi_6(L, sigma**sigma))

        delta += 0.001 * ((float(acc_steps)/steps) - 0.5)

        eta_list.append(eta)
        Psi_6_list.append(total_Psi_6/count_Psi_6)
        acc_prob_list.append(float(acc_steps)/steps)
        print 'eta = ', eta, ', Avg Psi_6 = ', total_Psi_6/count_Psi_6
        print 'Acc ratio = ', float(acc_steps)/steps, '\n'

pylab.plot(eta_list,Psi_6_list, 'bo')
pylab.plot(eta_list,Psi_6_list, ':k')
pylab.xlabel('$\eta$')
pylab.ylabel('Average $\Psi_6$')
pylab.title('My Markov Disks - Average $\Psi_6$ vs. $\eta$')
pylab.grid()
pylab.savefig('Psi_6_eta.png')
pylab.show()

pylab.plot(eta_list,acc_prob_list, 'bo')
pylab.plot(eta_list,acc_prob_list, ':k')
pylab.xlabel('$\eta$')
pylab.ylabel('$p_{acceptance}$')
pylab.title('My Markov Disks - $p_{acceptance}$ vs. $\eta$')
pylab.grid()
pylab.savefig('p_accept_eta.png')
pylab.show()

f = open(filename, 'w')
for a in L:
   f.write(str(a[0]) + ' ' + str(a[1]) + '\n')
f.close()
```

Below is a plot of the acceptance probabilities. Delta was dynamically adapted to generate acceptance probabilities around 0.5 for
the entire run. For densities, $\eta < 0.2$, then the acceptance probabilities start becoming large due to more empty space to move around for low densities resulting in majority of moves being accepted. The delta adaptation works well in the region of interest (densities from 0.72 to 0.42).



Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here you are asked to check that a submitted Python program is OK, and that the uploaded figure is correct

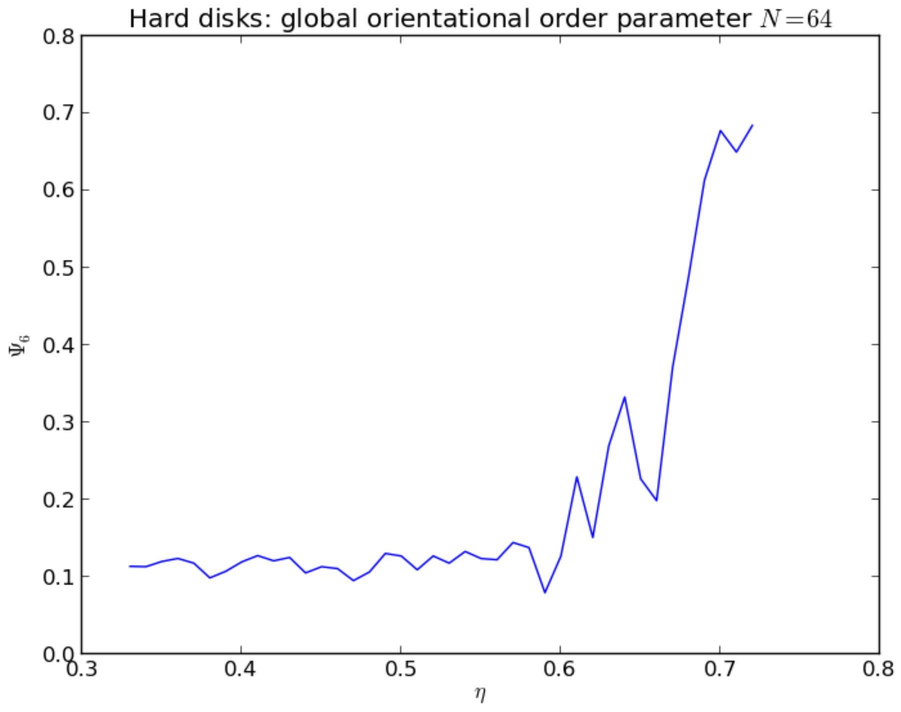
Two aspects should be checked

1/ The program should be OK, it may loop over densities (as the below program 0.72 -> 0.71 -> 0.70....), but does not have to do so.
2/ The order parameter should be obtained by averaging
3/ The output figure should show high order at eta \sim 0.72, and low order for smaller densities.

You can give a total of 2 points.

Give 0 points if none of the aspects was adressed correctly
Give 2 points if one or more of the above aspects were adressed correctly
(This is difficult material, thanks to all those who persevered).

Below a figure and a program that (Staff hopes) would received full score


Hard disks: global orientational order parameter $N=64$
Ψ_6
η
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.3
0.4
0.5
0.6
0.7
0.8

```
mport cmath, math, random, pylab, os

def show_conf(L, title, fname):
    pylab.axes()
    for [x, y] in L:
        for delx in range(-1, 2):
            for dely in range(-1, 2):
                cir = pylab.Circle((x + delx, y + dely), radius = sigma,
  fc = 'r')
                pylab.gca().add_patch(cir)
    pylab.axis('scaled')
    pylab.title(title)
    pylab.axis([0.0, 1.0, 0.0, 1.0])
    pylab.savefig(fname)
    pylab.show()

def dist(x, y):
    d_x = abs(x[0] - y[0]) % 1.0
    d_x = min(d_x, 1.0 - d_x)
    d_y = abs(x[1] - y[1]) % 1.0
    d_y = min(d_y, 1.0 - d_y)
    return d_x**2 + d_y**2


def delx_dely(x, y):
    d_x = (x[0] - y[0]) % 1.0
    if d_x > 0.5: d_x -= 1.0
    d_y = (x[1] - y[1]) % 1.0
    if d_y > 0.5: d_y -= 1.0
    return d_x, d_y

def Psi_6(L, sigma_sq):
    sum_vector = 0j
    for i in range(N):
        vector  = 0j
        n_neighbor = 0
        for j in range(N):
            if dist(L[i], L[j]) < 2.8 **2 * sigma_sq and i != j:
                n_neighbor += 1
                dx, dy = delx_dely(L[j], L[i])
                angle = cmath.phase(complex(dx, dy))
                vector += cmath.exp(6.0j * angle)
        if n_neighbor > 0: vector /= n_neighbor
        sum_vector += vector
    return sum_vector / float(N)

N =  64
filename = 'disk_configuration_0.72'
if os.path.isfile(filename):
    f = open(filename, 'r')
    L = []
    for line in f:
        a, b = line.split()
        L.append([float(a), float(b)])
    f.close()
    print 'starting from file', filename
else:
    N_sqrt = int(math.sqrt(N) + 0.5)
    delxy = 1./ 2. / N_sqrt
    two_delxy = 2.0 * delxy
    L = [ [delxy + i * two_delxy, delxy + j * two_delxy] \
        for i in range(N_sqrt) for j in range(N_sqrt)]
    print 'starting from scratch'
eta = 0.73
x_values = []
y_values = []
for iteration in range(40):
    eta -= 0.01
    print eta
    sigma = math.sqrt(eta / N / math.pi)
    sigma_sq = sigma ** 2
    delta = 0.3 * sigma
    n_steps = 10000
    average_Psi_6  = []
    for steps in range(n_steps):
        a = random.choice(L)
        b = [a[0] + random.uniform(-delta, delta), a[1] + random.uniform
(-delta, delta)]
        min_dist_sq = min(dist(b, c) for c in L if c != a)
        if  min_dist_sq > 4.0 * sigma ** 2:
            a[:] = [b[0] % 1.0, b[1] % 1.0]
        if steps % 100 == 0:
            average_Psi_6.append(abs(Psi_6(L, sigma_sq)))
    y_values.append(sum(average_Psi_6)/len(average_Psi_6))
    x_values.append(eta)

pylab.plot(x_values, y_values)
pylab.title('Hard disks: global orientational order parameter $N=64$')
pylab.xlabel('$\eta$')
pylab.ylabel('$\Psi_6$')
pylab.axis([0.3, 0.8, 0.0, 0.8])
```

Give 0 points if none of the aspects was adressed correctly
Give 2 points if one or more of the above aspects were adressed correctly

Score from your peers: **2**

Overall evaluation/feedback

Note: this section can only be filled out during the evaluation phase.

Here, you can provide feed-back to your fellow student.

peer 1 → Excellent, well done! I didn't get to complete C2 and C3 so I found your explanation very clear and concise. Also I liked that you elaborated on the derivation of pi/4 for the maximum density. I have calculated it the same way but the "extra insight" section was quite refreshing. Thanks.

peer 2 → except for the last question, I think you should get bonus marks !! amazing :D

peer 3 → *[This area was left blank by the evaluator.]*