Computational Physics - Exercise 3

October 2015

1 Task 1

1.1 Code

I created the class Random walker. Each walker has at each time x, and y position and can take a random step in any direction in 2D

```
class Randomwalker(object):

    def __init__(self, x_pos, y_pos):
        self.x_pos = x_pos
        self.y_pos = y_pos

    def take_one_step_randomly(self):
        rnd_x, rnd_y = get_random_unit_vector()
        self.x_pos = self.x_pos+rnd_x
        self.y_pos = self.y_pos+rnd_y
        return self.x_pos, self.y_pos

    def walk_X_steps(self, x):
        for i in range(0,x):
            self.take_one_step_randomly()
        return self.x_pos, self.y_pos


def get_random_angle():
    rnd = random.random()
    rnd_angle = rnd*2.0*math.pi
    return rnd_angle

def get_random_unit_vector():
    rnd_angle = get_random_angle()
    y = math.sin(rnd_angle)
    x = math.cos(rnd_angle)
    return x,y
```

1.2 Results

I plotted $\langle R^2 \rangle$ as a function of N. The theory from the lectures predicts that $\langle R^2 \rangle \propto$ N. Which implies the relation $\langle R^2 \rangle = k \times N$ where k is a constant.

On figure 1 we see the linear relation between $\langle R^2 \rangle$ and N - as predicted by the model. In This figure M=350000 (number of random walks) such that the error is below 1

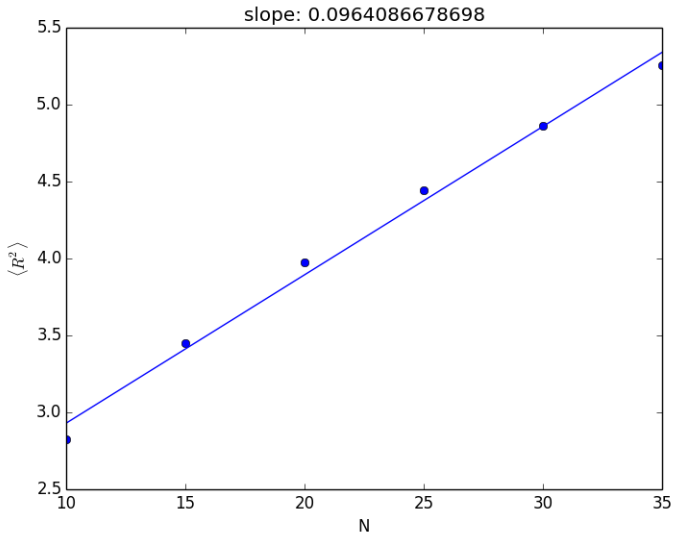

Figure 1: result of M=350000 random walks

2 Task 2

2.1 Code

I created the class RandomChain which represents a 3 dimensional chain. The chain has at each time a list of x,y and z components corresponding to the coordinates of each sphere. There are methods to add spheres and check if next random sphere overlaps with a previous ones in the chain. The class uses the helper methods get random theta, get random phi and get random unit vector to create the location of the next random sphere.

```
class RandomChain(object):

    def __init__(self, x_pos, y_pos, z_pos):
        self.x_pos_list = []
        self.y_pos_list = []
        self.z_pos_list = []
```

```
        self.x_pos_list.append(x_pos)
        self.y_pos_list.append(y_pos)
        self.z_pos_list.append(z_pos)

    def is_there_a_overlapping_sphere(self, new_x, new_y, new_z):
        number_of_spheres = len(x_pos_list)
        for i in range(0,number_of_spheres):
            old_x = self.x_pos_list[i]
            old_y = self.y_pos_list[i]
            old_z = self.z_pos_list[i]
        dis = sqrt((old_x-new_x)**2+(old_x-new_x)**2+(old_x-new_x)**2)
        if dis<1:
            return True
        else:
            return False


    def add_one_sphere(self):
        last_z = self.z_pos_list[-1]
        last_x = self.x_pos_list[-1]
        last_y = self.y_pos_list[-1]
        overlap = True
        while overlap:
            new_x, new_y, new_z = get_random_unit_vector(last_x, last_y, last_z)
            overlap = self.is_there_a_overlapping_sphere(new_x, new_y, new_z)
        self.x_pos_list.append(new_x)
        self.y_pos_list.append(new_y)
        self.z_pos_list.append(new_z)

    def add_X_spheres(self, X):
        for i in range(0,X):
            self.add_one_sphere()
        return self.x_pos_list[-1], self.y_pos_list[-1], self.z_pos_list[-1]



    def is_there_a_overlapping_sphere(self, new_x, new_y, new_z):
        number_of_spheres = len(self.x_pos_list)
        overlap = False
        for i in range(0,number_of_spheres):
            old_x = self.x_pos_list[i]
            old_y = self.y_pos_list[i]
            old_z = self.z_pos_list[i]
            dis = math.sqrt((old_x-new_x)**2+(old_x-new_x)**2+(old_x-new_x)**2)
            if dis<1:
               overlap = True
               break
        return overlap
```

```
def get_random_theta():
    rnd = random.random()
    rnd_theta = rnd*math.pi
    return rnd_theta


def get_random_phi():
    rnd = random.random()
    rnd_phi = rnd*2.0*math.pi
    return rnd_phi

def get_random_unit_vector(last_x, last_y, last_z):
    rnd_phi = get_random_phi()
    rnd_theta = get_random_theta()
    x = math.sin(rnd_theta)*math.cos(rnd_phi)+last_x
    y = math.sin(rnd_theta)*math.sin(rnd_phi)+last_y
    z = math.cos(rnd_theta)+last_z
    return x,y,z
```

2.2 Results

I plotted $\langle R^2 \rangle$ as a function of N. The theory from the lectures predicts that $\langle R^2 \rangle \propto N^{\alpha}$. Which implies the relation $log(\langle R^2 \rangle) = log(k) + \alpha \times log(N)$ where k is a constant.

On figure 1 we see the linear relation between $log(\langle R^2 \rangle)$ and log(N) - as predicted by the model. In This figure M=350000 (number of random chains).

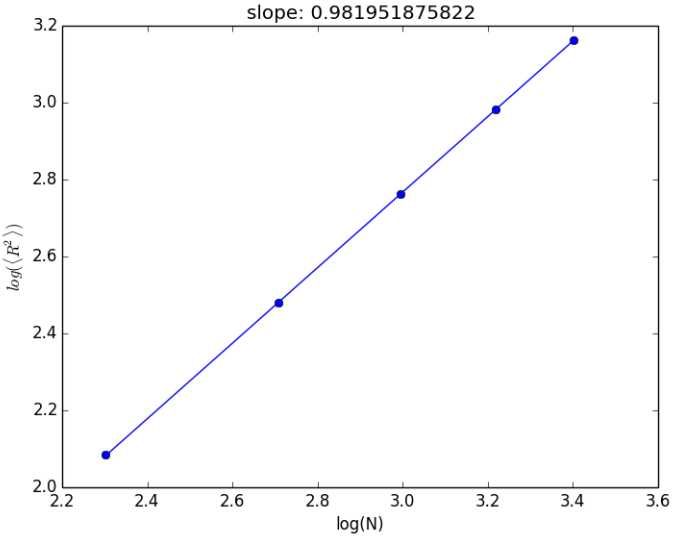

Figure 2: result of M=350000 random chains