Homework

Computer Sciences for Physics and Chemistry

Presented by: ***Kanan Jafarli***
Group: ***Computer Science***

1 Diffusion in 1 dimension: the drunk walker

A drunk walker that moves along in a a straight street. At each step, he has a probability p to move forward and 1-p to move backward.

1.1 Scripts

You can get from here: link for source code

1.2 Libraries and Functions

1.2.1 Libraries or modules

- **random** for take random float number between 0 and 1 using its function *random()*
- **numpy** for calculate standard deviation and mean, for return spaced numbers over a specified interval using its function *linspace()*.
- **matplotlib.pyplot** for plot histogram using *hist()* and curve on histogram using *plot()*
- **scipy.stats** for normal distribution using *norm.pdf()*

```
import random as r
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as ss
```

1.2.2 Function: position

Calculate final position after N steps. Take initial position, probability of move forward and number of steps as parameters. First position of drunk walker is equal to initial position. At each step function takes random float number between 0 and 1. If probability bigger than this random float number, drunk walker moves 1 step forward, otherwise moves 1 step backward.

```
def position(initial_pos,p,nbsteps):
  pos=initial_pos
  for i in range(nbsteps):
    a=r.random()
    if p>a:
      pos=pos+1
    else:
      pos=pos-1
  return pos
```

1.2.3 Function: list_positions

Return positions x times as list. Take initial position, probability of move forward, number of steps and number of times as parameters. Each time list append final positions of drunk walker.

```
def list_positions(initial_pos,p,nbsteps,nbtimes):
  list_pos=[]
  for i in range(nbtimes):
    list_pos.append(position(initial_pos,p,nbsteps))
  return list_pos
```

1.2.4 Function: histogram

```
def histogram(initial_pos,p,nbsteps,nbtimes):
  plt.title(f'Calculate positions {nbtimes} times for {nbsteps} steps')
  plt.xlabel('Position')
  plt.ylabel('Probability density')
  plt.hist(list_positions(initial_pos,p,nbsteps,nbtimes),bins=20,density=True,edgecolor='black',color='salmon');
```

This function plots histogram. Take initial position, probability of move forward, number of steps and number of times as parameters.

1.2.5 Function: Gaussian

Calculate standard deviation and mean, plot normal curve to histogram. Take initial position, probability of move forward, number of steps and number of times as parameters.

```
def Gaussian(initial_pos,p,nbsteps,nbtimes):
  lpos=list_positions(initial_pos,p,nbsteps,nbtimes)
  std=np.std(lpos)
  mean=np.mean(lpos)
  print(f'sigma {nbsteps} is: {std}\nmean is: {mean}')

  histogram(initial_pos,p,nbsteps,nbtimes)
  xmin, xmax = plt.xlim()
  x = np.linspace(xmin, xmax, 1000)
  pdf = ss.norm.pdf(x, mean, std)
  plt.plot(x, pdf,'blue',linewidth=3)
  plt.show()
```

1.3 Testing functions and results

1.3.1 Get the final position for one time using *position* function

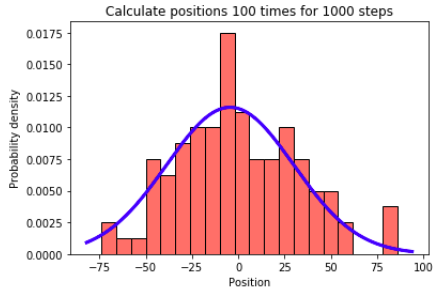
Initial position is 0, there are 1000 steps and if probability of move forward is

- 0.5 - the final position can be 20, -16, -72, 30 and etc. (approximately between -100 and 100)
- 0.75 - the final position can be 510, 468, 474, 538 and etc. (approximately between 400 and 600)

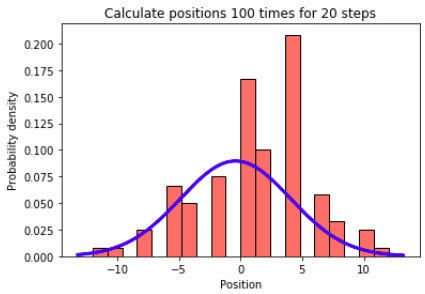
1.3.2 Plot histogram and obtain Gaussian distribution using *Gaussian* function

After running *position* function 100 times for different steps:

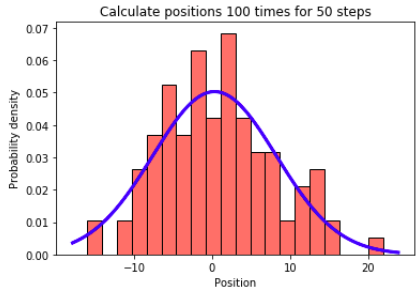
```
sigma 1000 is: 34.404650848395484
mean is: -4.4
```



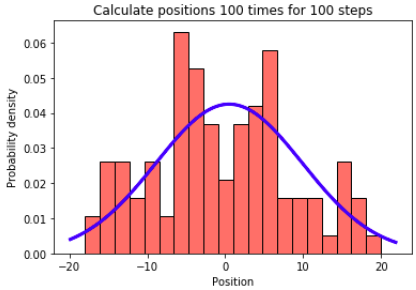

```
sigma 20 is: 4.454211490264018
mean is: -0.4
```



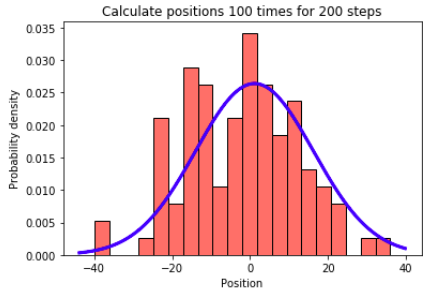
```
sigma 50 is: 7.9282785016673065
mean is: 0.32
```



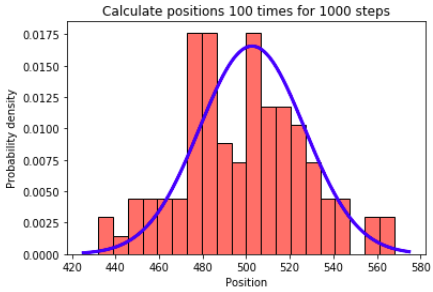

```
sigma 100 is: 9.382430388763884
mean is: 0.5
```




```
sigma 200 is: 15.113583294506965
mean is: 1.14
```




```
sigma 1000 is: 24.083554554923992
mean is: 502.68
```



Probability of last one is 0.75, others is 0.5.

1.4 Conclusion

When probability of move forward is bigger than probability of move backward, the final position of drunk walker always is at the right of initial position, otherwise the final position is at the left of initial position. If probability of move forward is equal to probability of move backward, the final position can be at the both sides of initial position.
Standard deviation increases when number of steps increases and also probability of move forward is more bigger.

2 Diffusion in 2 dimensions: diffusion of a dye in water

2.1 Scripts

You can get from here: link for source code

2.2 Libraries and Functions

2.2.1 Libraries or modules

- **random** for take random float number between 0 and 1 using its function *random()*
- **matplotlib.pyplot** for show positions of molecules as collection of points using *scatter()*

```
#libraries
import random
import matplotlib.pyplot as plt
```

2.2.2 Class: Molecule

Each molecule has id number, position, and point. The initial position of molecules are 0. *setPoint()* and *setPosition()* functions update points and positions of molecules. And with *getPoint(), getPosition(), getId()* we can get variables of molecules.

```
class Molecule:
  def __init__(self,id):
    self.id=id
    self.pos=0
    self.point=[0,0]
  def setPoint(self,r,c):
    self.point[0]=r
    self.point[1]=c
  def getPoint(self):
    return self.point
  def getPosition(self):
    return self.pos
  def setPosition(self,pos):
    self.pos=pos
  def getId(self):
    return self.id
```

2.2.3 Function: init_molecules

Initialize molecules with id at the center of lattice and return list of molecules. Take lattice, its row and column as parameters.

```
def init_molecules(lattice,row,col):
  molecules=[]
  id=1
  for i in range(row):
    for j in range(col):
      if i>=5 and i<col-5 and j>=5 and j<row-5:
        m=Molecule(id)
        lattice[i][j]=m.getId()
        m.setPoint(i,j)
        molecules.append(m)
        id+=1
  return molecules
```

2.2.4 Function: scatter

Represent the positions of the molecules in a scattered plot as collection of points. Take list of molecules and number of steps as parameters.

```
def scatter(molecules,nbsteps):
  x=[]
  y=[]
  for m in molecules:
    x.append(m.getId())
    y.append(m.getPosition())
  plt.title(f'For {nbsteps} steps')
  plt.xlabel('molecule')
  plt.ylabel('position')
  plt.scatter(x,y,c='red');
```

2.2.5 Function: final_position

```
def final_position(molecules,lattice,p,nbsteps,row,col):
  for step in range(nbsteps):
    for m in molecules:
      r=m.getPoint()[0]
      c=m.getPoint()[1]
      pos=m.getPosition()
      a=random.random()
      if p>a:
          m.setPosition(pos)
          m.setPoint(r,c)
      else:
        if r>0:
          m.setPoint(r-1,c)
          m.setPosition(pos+1)
          continue
        if c>0:
          m.setPoint(r,c-1)
          m.setPosition(pos+1)
          continue
        if r<row-1:
          m.setPoint(r+1,c)
          m.setPosition(pos+1)
          continue
        if c<col-1:
          m.setPoint(r,c+1)
          m.setPosition(pos+1)
          continue
```

Calculate final positions of molecules and molecules update its positions and points. Take lattice, its row and column, list of molecules, number of steps, and probability to stay in box as parameters.
At each step function takes random float number between 0 and 1 for each molecule. If probability bigger than this random float number, position and point of molecule does not change and molecule stay in its box. Else molecule move 1 step to down *(row-1)*, left *(column-1)*, up *(row+1)*, right *(column+1)* changing its point.

2.3 Testing functions and results

2.3.1 Description of lattice at initial position

Row and column of lattice is 20. There are 100 molecules at the center of lattice with ids.

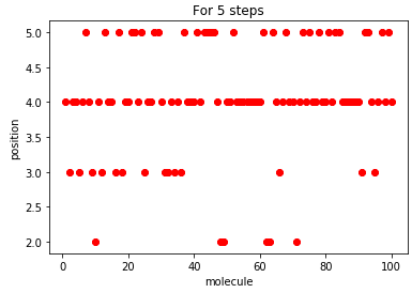
```
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   1   2   3   4   5   6   7   8   9   10  0   0   0   0   0
0   0   0   0   0   11  12  13  14  15  16  17  18  19  20  0   0   0   0   0
0   0   0   0   0   21  22  23  24  25  26  27  28  29  30  0   0   0   0   0
0   0   0   0   0   31  32  33  34  35  36  37  38  39  40  0   0   0   0   0
0   0   0   0   0   41  42  43  44  45  46  47  48  49  50  0   0   0   0   0
0   0   0   0   0   51  52  53  54  55  56  57  58  59  60  0   0   0   0   0
0   0   0   0   0   61  62  63  64  65  66  67  68  69  70  0   0   0   0   0
0   0   0   0   0   71  72  73  74  75  76  77  78  79  80  0   0   0   0   0
0   0   0   0   0   81  82  83  84  85  86  87  88  89  90  0   0   0   0   0
0   0   0   0   0   91  92  93  94  95  96  97  98  99  100 0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

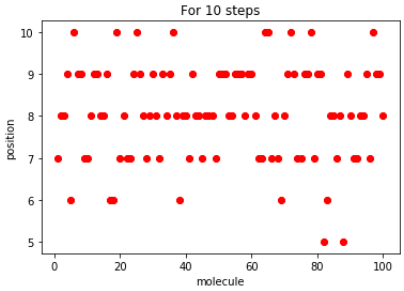
2.3.2 Final positions of molecules for different steps

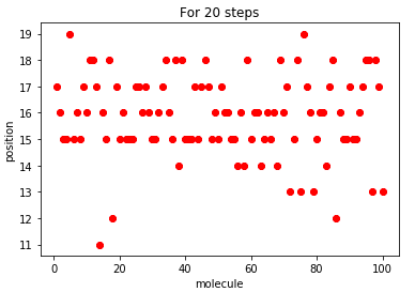
Probability to stay in box is 0.2. You can look from here: link for file_02
Probability to stay in box is 0.8. You can look from here: link for file_08

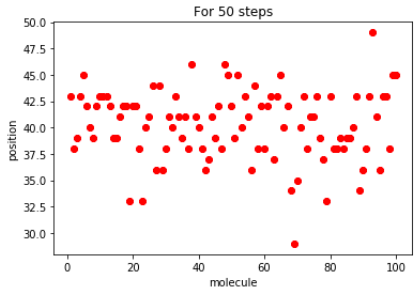
2.3.3 Represent the positions of the molecules in a scattered plot for different steps

Probability to stay in box is 0.2.

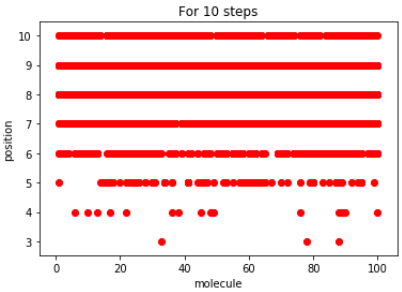


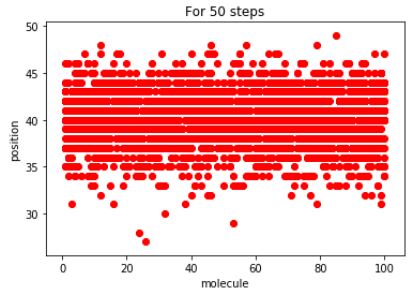






After running 25 times for 10 and 50 steps:





2.4 Conclusion

When probability to stay in box is bigger, molecule does not move and stay in its boxes, otherwise probability to stay in box is smaller, molecule moves 1 step to up, right, left or down. When number of steps increases, final positions of molecules also increase.