Exercise Sheet 2

Pulkit Kukreja, Xiongxiao Wang

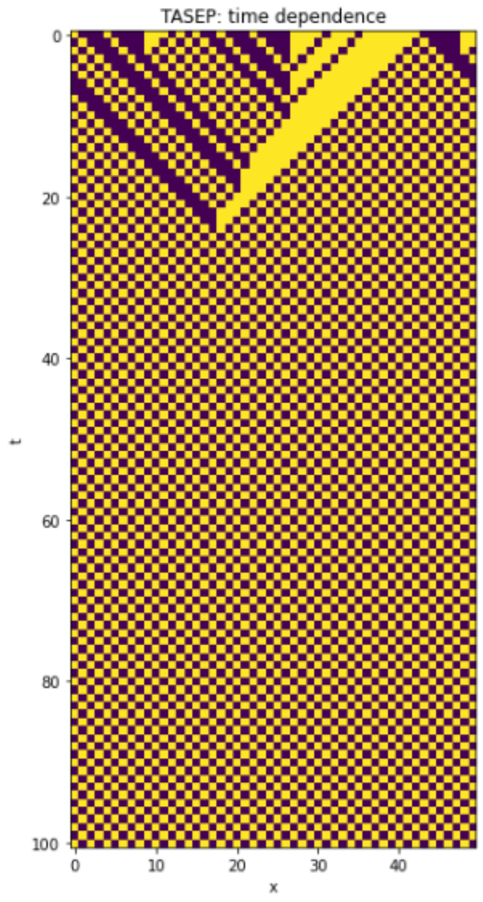
This Exercise has been done with Python 3.6

TASEP

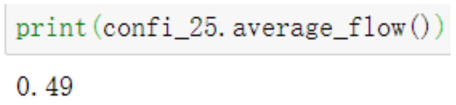
Exercise 1: TASEP (8 points) As shown in the lecture, the TASEP with parallel update corresponds to rule 184. Here we consider the TASEP with N = 50 sites and periodic boundary conditions.

A)The main loops of the code:

```
def TASEP(self):
#apply the rule of TASEP and get the whole configuration
        next_row = np.zeros(self.n_sites,dtype = int)
        for row_i in range(self.Nt):
            for i in range(self.n_sites):
                next_row [i]= self.Rule()[7-(self.confi[row_i,
(i-1+self.n_sites)
%self.n_sites]*4+self.confi[row_i,i]*2+self.confi[row_i,
(i+1+n_sites)%self.n_sites]*1)]
# apply the rule 184 with the periodic boundary condition
            self.confi = np.append(self.confi,[next_row],axis=0)
            if self.confi[row_i,-1]==1 and self.confi[row_i,0]==0:
                self.num_N_1 +=1
```



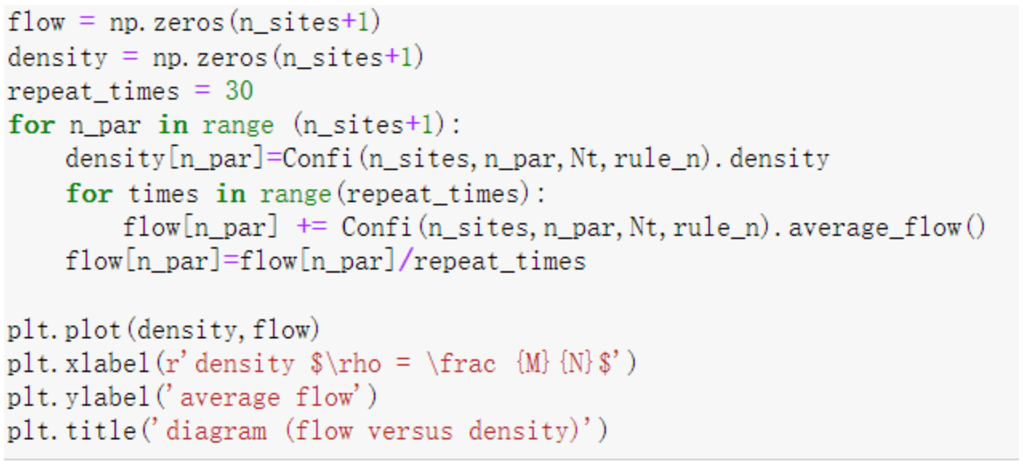
Choose M=25, N=50 ,Nt =100, we can get the figure above. The average flow for Nt=100 time steps is about 0.49



```
print(confi_25.average_flow())
```

```
0.49
```

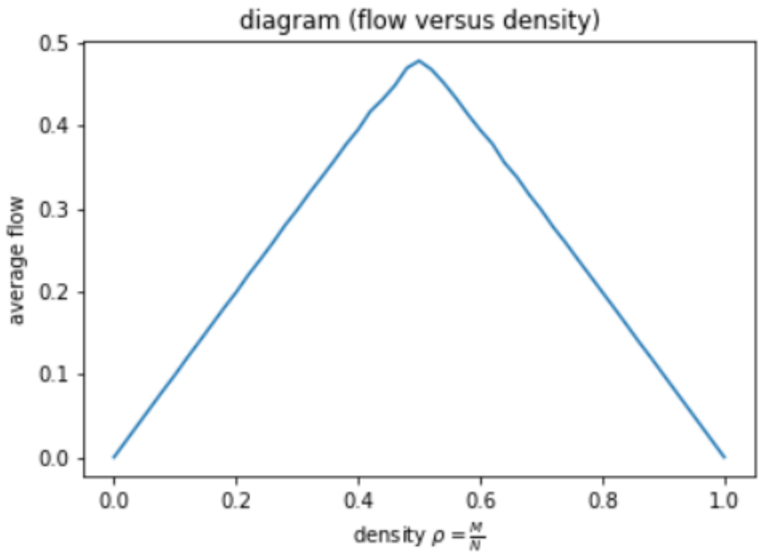
B) The main loops of the code:



```
flow = np.zeros(n_sites+1)
density = np.zeros(n_sites+1)
repeat_times = 30
for n_par in range (n_sites+1):
    density[n_par]=Confi(n_sites,n_par,Nt,rule_n).density
    for times in range(repeat_times):
        flow[n_par] += Confi(n_sites,n_par,Nt,rule_n).average_flow()
    flow[n_par]=flow[n_par]/repeat_times

plt.plot(density,flow)
plt.xlabel(r'density $\rho = \frac {M}{N}$')
plt.ylabel('average flow')
plt.title('diagram (flow versus density)')
```

To improve the quality of data, we choose repeat time as 30, and start the particle number from 0 to 50, and get the corresponding average flow of 30 times.

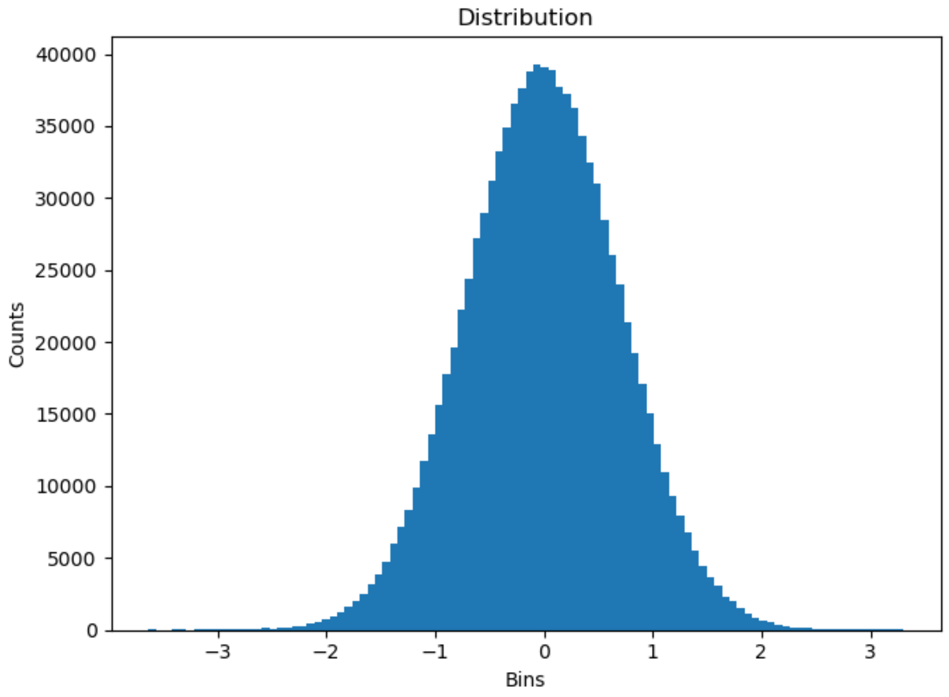


Problem 2: Metropolis Algorithm

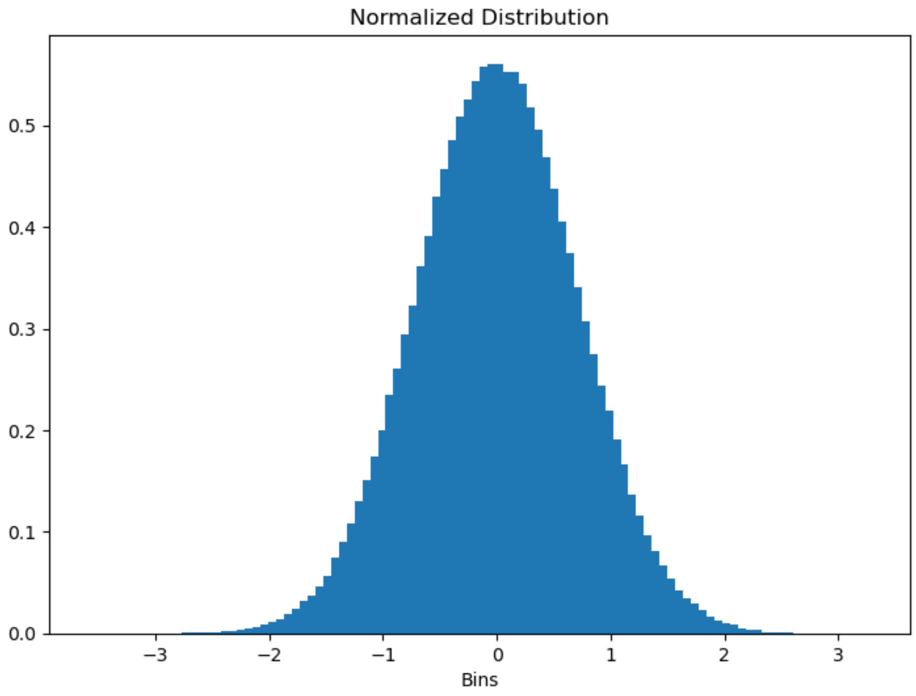
A) The main part of the code:

```
def target_dist(x):
    return (1/np.sqrt(np.pi))*np.exp(-x**2)
#h: stepsize, by default its one
def metropolis(x, h=1):
    delta = np.random.uniform(low=-1, high=1)
    new_x = x + h*delta
    alpha = target_dist(new_x)/target_dist(x)
    gamma = np.random.uniform(low=0, high=1)
    if alpha>=gamma:
        return new_x
    else:
        return x
#N: count of Markov chain numbers to be
generated
N = int(1e6)
ini = 1
arr = np.zeros(N)
arr[0] = ini
for i in range(1, N):
    arr[i] = metropolis(arr[i-1])
```

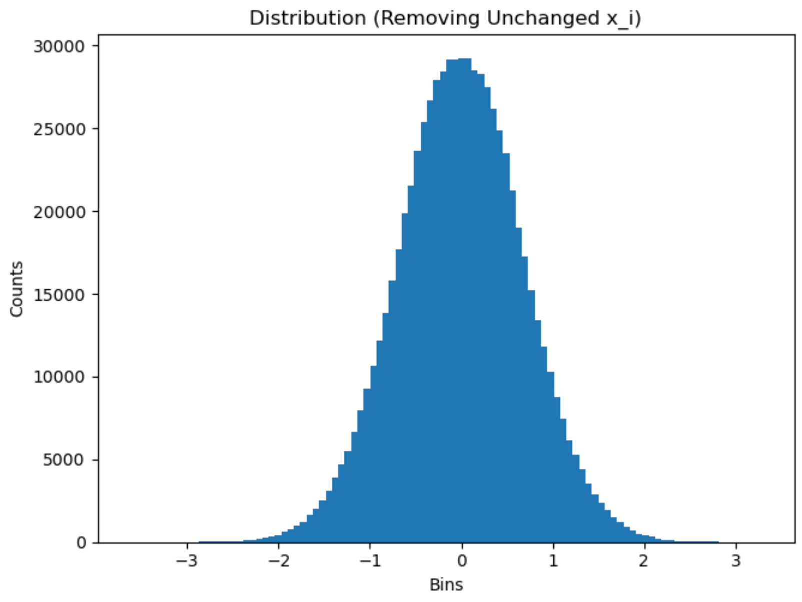
Distribution of Markov chain numbers as a histogram of 100 bins:

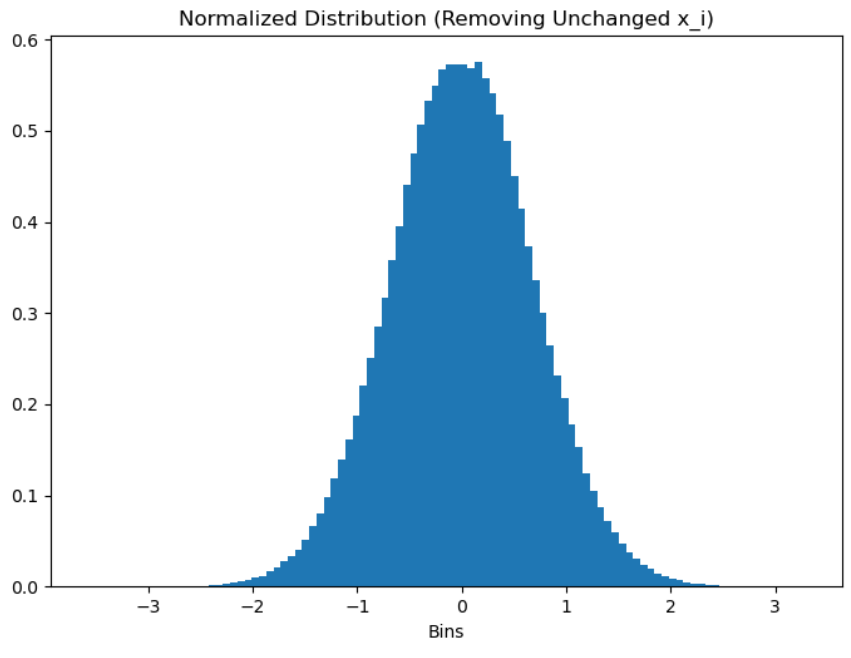


Normalized Distribution:



B) Now, we can remove the unchanged x_i by taking the unique elements of the Markov chain, so the unchanged x_i's are just counted once.
If we look at the distribution which is not normalised, the overall spectrum will just shift below, i.e. we'll have less counts than before. But the normalised distribution won't change at all, and the shape doesn't change.


Distribution (Removing Unchanged x_i)
Counts
Bins


Normalized Distribution (Removing Unchanged x_i)
Bins