In [ ]:

In [107]:
```
#Question 4

import numpy as np
import math
from tqdm import tqdm
import matplotlib.pyplot as plt
from numpy.linalg import multi_dot
```

In [108]:
```
def create_picture(positions):
    plt.cla()
    plt.gca().set_aspect('equal')
    plt.axis([0, L, 0, L])
    plt.setp(plt.gca(), xticks=[0, L], yticks=[0, L])
    for x,y in positions:
        atom = plt.Circle((x, y), Ratom, fc='r')
```

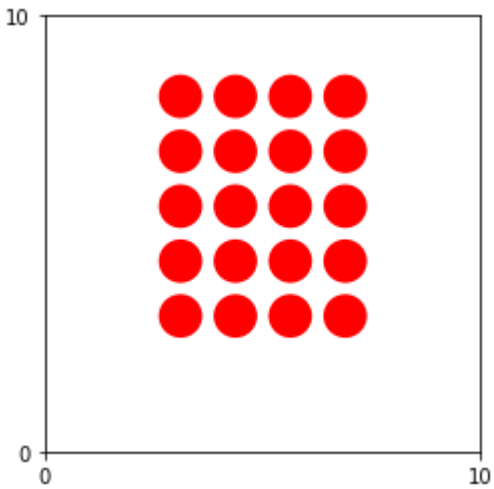
In [109]:
```
rcut = 3.0  ##### Cut-off distance.
rcutcube = rcut**3
g3 = 1./rcutcube
```

In [110]:
```
##### Function to compute acceleration for a pair of atoms.
## r12 is relative displacement of atoms.

def acceleration(r12):
    r12square= np.dot(r12,r12)
    f2=1./r12square
    f3 = 1./math.pow(r12square,1.5) #to get relevant potential
    acc = 24.*f2*f3*(f3-0.5)*r12
    return acc

def potentialenergy(pos):
    potential = 0.
    for i in range(Natoms-1):
        for j in range(i+1,Natoms):
            rij = pos[i] - pos[j] ## Relative position vector of the p
            for l in range(2):  ### Calculating the correct separation
                if abs(rij[l])>0.5*L: rij[l] -= L*np.sign(rij[l])
            rijsquare = np.dot(rij,rij)
            if rijsquare < rcutsquare: # Imposing interaction cut-off
                f2 = 1./rijsquare
                f3 = math.pow(f2,1.5)
                potential += 4.*f3*(f3-1.) - potcut
    return potential
```

In [ ]:

In [111]:
```
Natoms = 20 # No. of atoms
Ratom = 0.5  ### Radius of atom used to draw the atom
rho = 0.5 ### Number Density
L = 10 # Length of a side of the square containing the gas.
T0 = 110. # Natural temperature scale, T0 = epsilon/k.
T = T0   # Temperature in Kelvin
dt = 1E-2  # Time step
```

In [128]:
```
eqsep = math.pow(2.,1./3.)#eqlb sepration as calcualted
## Equilibrium separation of atoms
wall_spacing = (L-(4-1)*eqsep)/2.## Distance between the walls and the
poslist = []  # List for positions of atoms
vlist = []   # List for velocities

################  Initialize positions and velocities ################
x, y = wall_spacing, wall_spacing

for i in range(5):
    for j in range(4): #creating lattice of 4x5 atoms
        xi, yi = x + eqsep*j, y + eqsep*i
        poslist.append((xi,yi))

####### Vels. ################
v0 = 2*np.ones([Natoms])# setting velocities to the dimensionaless vel

for i in range(Natoms):#giving the particles random directions
    phi = 2*np.pi*np.random.random()
    vx = v0[i]*np.cos(phi)
    vy = v0[i]*np.sin(phi)
    vlist.append((vx,vy))
    #print(np.sqrt(vx**2+vy**2))
#########################################################################

pos = np.array(poslist)  ### Converts lists to numpy arrays
v = np.array(vlist) ## Scaled velocity
V_cm = np.sum(v, axis = 0)/Natoms ## Correcting for CM velocity.
V = np.array([V_cm,]*Natoms)
v -= V
```

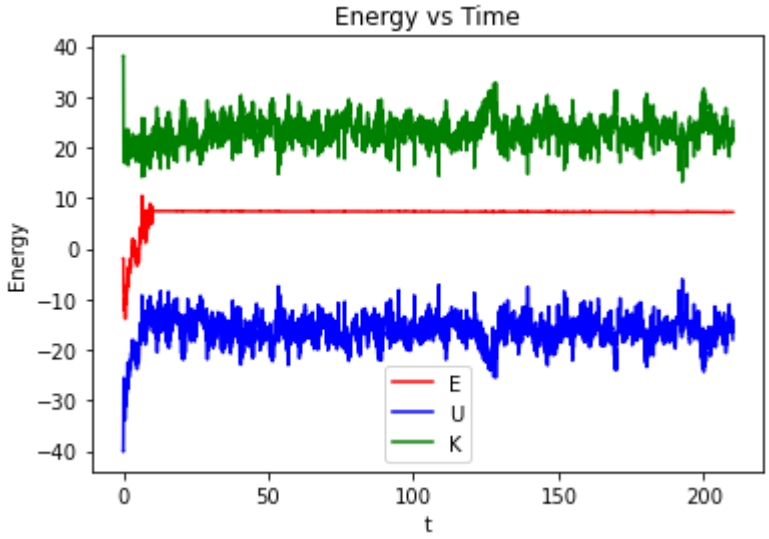


In [129]:
```
time = 0. # Initial time.
t_final = 200. # Time upto which simulation is carried out.
potential_energy = potentialenergy(pos)
kinetic_energy = 0.5*sum(np.square(v).sum(axis=1))
energy = kinetic_energy + potential_energy
Time_List = [time]
Energy_List = [energy]
PotentialEnergy_List = [potential_energy]
KineticEnergy_List = [kinetic_energy]
sart = 10 #steps to artificial evolution
nart = 100 # num of artificial evoltuion
T_art = sart * nart # time (divided by dt) for which there's artificia
iterations = int(t_final/dt+T_art) # Number of iterations of the Verle
```

In [130]:
```
##################################### The Verlet Algorithm Loop ######
for step in tqdm(range(iterations)):
    pos += v*dt/2. # Updating positions.
    for i in range(Natoms): ## Imposing periodic boundary conditions.
        for j in range(2):
            if pos[i][j] > L: pos[i][j] -= L
            elif pos[i][j] < 0: pos[i][j] += L
    accel = np.zeros((Natoms,2)) ## Initialising acceleration array.
    for i in range(Natoms-1):
        for j in range(i+1,Natoms):
            rij = pos[i] - pos[j]
            for l in range(2):  ### Calculating the correct separation
                if abs(rij[l])>0.5*L: rij[l] -= L*np.sign(rij[l])
            if np.dot(rij,rij) < rcutsquare:
                acc = acceleration(rij) # Computing acceleration for a
                accel[i] += acc
                accel[j] -= acc
    v += accel*dt  ## Updating velocities.
    pos += v*dt/2. ## Final updating of positions.
    for i in range(Natoms): ## Imposing periodic boundary conditions.
        for j in range(2):
            if pos[i][j] > L: pos[i][j] -= L
            elif pos[i][j] < 0: pos[i][j] += L
    potential_energy = potentialenergy(pos)
    kinetic_energy = 0.5*sum(np.square(v).sum(axis=1))
    energy = kinetic_energy + potential_energy
    Energy_List.append(energy)
    PotentialEnergy_List.append(potential_energy)
    KineticEnergy_List.append(kinetic_energy)
    time += dt
    if (step%sart) == 0 and (step>0) and (step<=sart*nart): # artifica
        ksf = (T/T0)*Natoms/kinetic_energy # KE scaling factor
        v *= ksf**.5
    if step == int(T_art+t_col*t_final/dt):
        v_ = np.copy((v[:,0]**2+v[:,1]**2)**.5)
        ind_coll = 1
    elif step > int(T_art+t_col*t_final/dt) and (step%10 == 0):
        v_ = np.concatenate((np.copy(v_),np.copy((v[:,0]**2+v[:,1]**2)
        ind_coll += 1
```

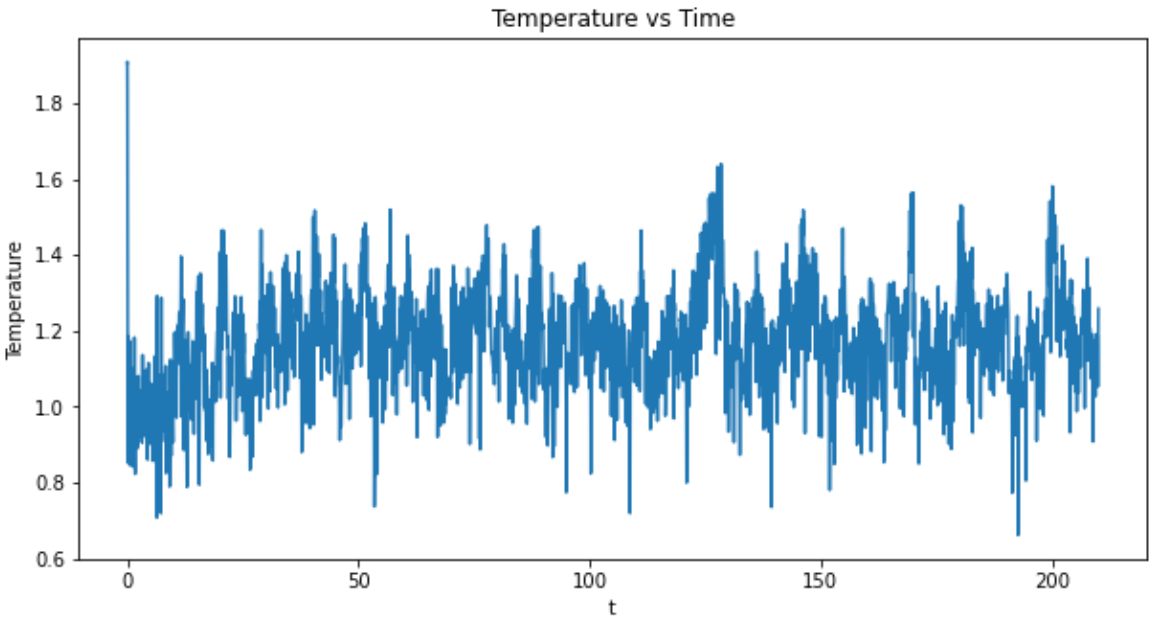
```
100%|████████████████████████████████████| 21000/21000 [00:34<00:00,
601.67it/s]
```

In [131]:
```
plt.xlabel('t') # Label for the x-axis
plt.ylabel('Energy') # Label for the y-axis
plt.title('Energy vs Time') # Title of plot
plt.plot(Time_List,Energy_List, color = 'r', label="E")
plt.plot(Time_List,PotentialEnergy_List, color = 'b', label="U")
plt.plot(Time_List,KineticEnergy_List, color = 'g', label="K")
```



In [132]:
```
#Question 4

plt.figure(figsize=[10,5])
plt.xlabel('t') # Label for the x-axis
plt.ylabel('Temperature') # Label for the y-axis
plt.title('Temperature vs Time') # Title of plot
T_list = np.array(KineticEnergy_List)/Natoms #Temperature is avg KE
```

Out[132]:
```
[<matplotlib.lines.Line2D at 0x10fd5b370>]
```



In [133]:
```
#Question 5
#calculating the theoretical maxwell distribution
Temp = sum(KineticEnergy_List[int(T_art+.25*t_final/dt):-1])/(len(Kine

def P_v(u,T_):
    return 1/T_ * u * np.exp(-u**2/(2*T_))
#Temp=T/T0
v_s = np.linspace(min(v_),max(v_),int((max(v_)-min(v_))/.01))
P_vs = P_v(v_s,Temp)
#P_vs = [P_v(i,T,T0) for i in v_s]
```

In [134]:
```
plt.figure(figsize=[10,5])
bin_num = 100
bin_width = (max(v_)-min(v_))/bin_num
plt.hist(v_,bins = bin_num,edgecolor="black")
plt.plot(v_s,P_vs*(len(v_)*bin_width),color='green')
plt.xlabel("speeds of the particle")
plt.ylabel('No of particles')
plt.title('Maxwell speed distribution of speeds')
```