Importamos las bibliotecas correspondientes

In [8]:
```
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
from copy import *
%matplotlib notebook
#plt.rcParams["figure.figsize"] = (5,5)
```

In [9]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

Funciones de inicialización de posiciones y velocidades

In [10]:
```
def lattice_pos(npart,r):
    """
    Distribuye homogeneamente npart particulas en una celda
    de 1x1 entre 0 y 1, agregando un r/100% de aleatoriedad.
    Devuelve un np.array de npart de posiciones x 2 coordenadas

    """

    #Calculo cantidad de celdas por lado
    n = int(np.ceil(np.sqrt(npart)))

    #calculo el centro de la celda
    nf = 1.0/float(n)

    #Calculo las posiciones de todas las celdas disponibles agregando
un r% de randomizacion
    #r = 0.02
    celdas = [[i*nf +0.5*nf + r*np.random.rand(),j*nf + 0.5*nf+ r*np.r
andom.rand()]
                for i in range(n) for j in range(n)]

    #Mezclo las posiciones de las celdas
    np.random.shuffle(celdas)

    X = np.array(celdas[0:npart])
    return X
```

```
def vel_ini(npart,temp_red):
    """
    Asigna velocidades al azar de la distribucion uniforme en el rango
-0.5 a 0.5

    """
    V = np.random.rand(npart,2)-0.5*np.ones((npart,2), dtype=np.float)

    #Calculamos la velocidad del centro de masa
    Vx = sum(V[:,0])/npart
    Vy = sum(V[:,1])/npart

    #Calculamos la velocidad cuadratica media
    V2 = sum(sum(V**2))/npart

    #Factor de escala para temperatura
    fs = np.sqrt(2*temp_red/V2)

    #Elimino la velocidad del centro de masa y escalo
    V[:,0] = (V[:,0] - Vx)*fs
    V[:,1] = (V[:,1] - Vy)*fs

    return V

def init(npart,box,temp_red,dt_red,sigma):
    """
    inicializa posiciones y velocidades usando lattice_pos y vel_ini

    """
    r = 0.00
    X = lattice_pos(npart,r)

    #Escalo posiciones al tamanio de caja y lo reduzco con sigma
    X = (X*box)/sigma
    V = vel_ini(npart,temp_red)

    return X,V
```

Función para calcular las fuerzas usando Lennard-Jones en condiciones periódicas

In [11]:
```
def force(npart,X,rcut,box,epsilon):
    """
    Calcula las fuerzas de todas las particulas utilizando un potencia
l de Lennard-Jones.
    Las distancias ya estan reducidas por sigma y utiliza un radio de
corte rcut.
    Las fuerzas son devueltas en unidades reducidas y las energias en
kJ/mol

    """


    #inicializo las fuerzas a 0.0 y la energia a 0.0
    F = np.zeros((npart,2),dtype=np.float)
    Epot = 0.0

    #Calculo la energia de cutoff en unidades reducidas
    rci6 = 1.0/rcut**6
    rci12= rci6**2
    Ecut = 4.0*(rci12 -rci6)

    #Calculo la fuerza sobre todos los pares de particulas
    for i in range(npart-1):
        for j in range(i+1,npart):
            rx = X[i,0] - X[j,0]
            rx = rx - box*np.rint(rx/box)
            ry = X[i,1] - X[j,1]
            ry = ry - box*np.rint(ry/box)
            r2 = rx**2 + ry**2

            if r2 < rcut**2:
                r2i = 1/r2
                r6i = r2i**3
                ff = 48.0*r2i*r6i*(r6i-0.5)

                # F tiene unidades de reducidas
                F[i,0] = F[i,0] + ff*rx
                F[j,0] = F[j,0] - ff*rx

                F[i,1] = F[i,1] + ff*ry
                F[j,1] = F[j,1] - ff*ry

                Epot = Epot + 4.0*r6i*(r6i-1.0) - Ecut

    return [F,Epot*epsilon] #Fuerza en unidades reducidas y energia en
kJ/mol
```

Funciones para calcular posiciones y velocidades usando velocity Verlet en condiciones periódicas

In [12]:
```
def pos(npart,X,V,F,dt,box,m):
    """
    Calcula las nuevas posiciones usando el algoritmo de velocity Verl
et.
    Todas las distancias están reducidas. Las nuevas posiciones son de
vueltas en
    coordenadas reducidas.

    """


    Xnew = X + V*dt + (F/(2.0*m))*dt**2

    #Se fija si la partícula salió de la caja y la transporta a
    #la posición correspondiente de su imagen
    np.where(Xnew[:,0] > box, Xnew[:,0] - box, Xnew[:,0])
    np.where(Xnew[:,0] < 0.0, box + Xnew[:,0], Xnew[:,0])
    np.where(Xnew[:,1] > box, Xnew[:,1] - box, Xnew[:,1])
    np.where(Xnew[:,1] < 0.0, box + Xnew[:,1], Xnew[:,1])

    return Xnew

def vel(npart,V,F,Fnew,dt,m):
    """
    Calcula las nuevas velocidades usando el algoritmo de velocity Ver
let.
    Las nuevas velocidades son devueltas en coordenadas reducidas y la
energía
    cinética está en kJ/mol

    """
    Vnew = V + (0.5/m)*(Fnew + F)*dt

    V2 = sum(sum(Vnew**2))
    Ekin = 0.5*V2*epsilon

    return Vnew, Ekin
```

Función de inicialización de simulación que genera unidades reducidas y otros parámetros relevantes para simular, y función que genera una corrida (capaz de controlar la temperatura usando un termostato de Berendsen)

In [13]:
```
def inicializar(npart, box, temperatura, dt, sigma, epsilon, masa):
    """
    Inicializa una simulación en función de la cantidad de partículas
(npart),
    el tamaño de la caja en angstroms (box), una temperatura (K), el p
aso de simulación
```

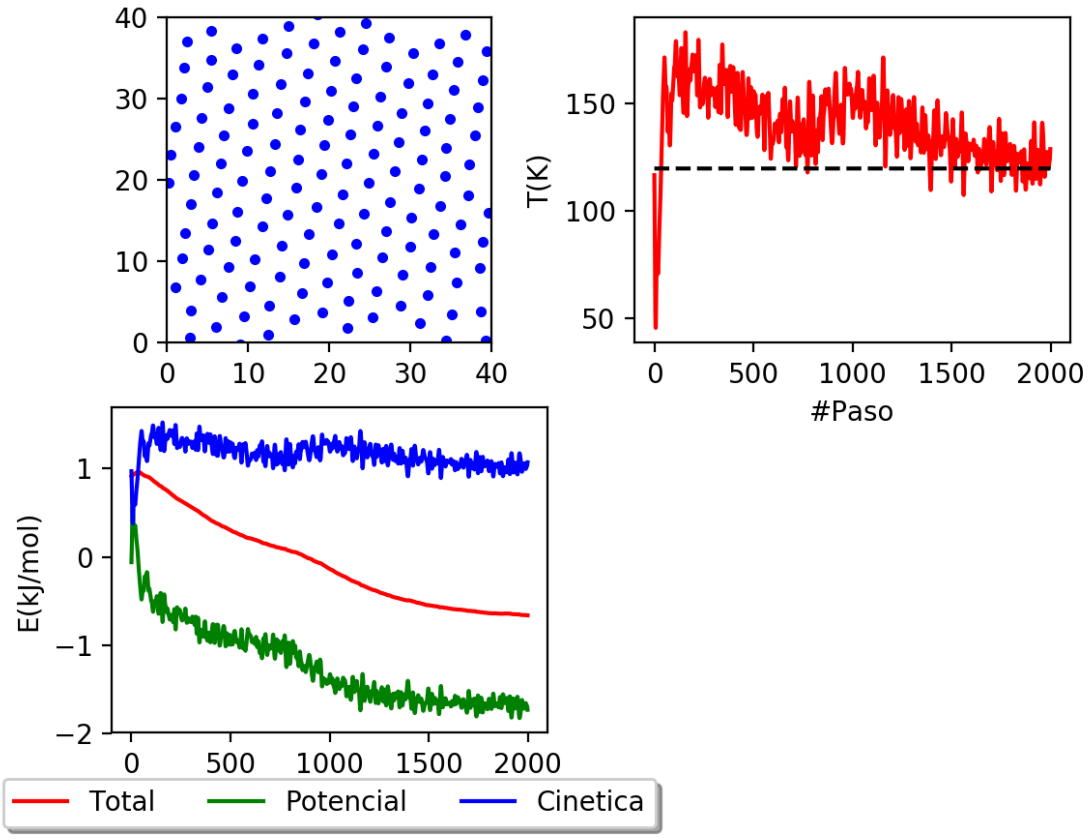
```
    (dt) en segundos, los parámetros de Lennard-Jones sigma y epsilon
(en A y kJ/mol)
    y la masa de la partícula (en g/mol). Devuelve coordenadas, veloci
dades y fuerzas iniciales en
    unidades reducidas y un diccionario con parámetros de la corrida (
unit_red)

    """
    NA = 6.02e23 # 1/mol
    kB = 1.381e-23 # J/K

    rcut = box/2.0 # A (unidades reales)

    #reduzco el tiempo
    t_unit = (sigma*1e-10) * np.sqrt(masa*1e-3/(1000*epsilon))
    dt_red = dt/t_unit

    #reduzco la temperatura
    temp_unit = 1.0/(kB/(1000*epsilon/NA))
    temp_red = temperatura/temp_unit

    #reduzco la masa
    m_unit = masa*1e-3/NA
    m = 1

    #Posiciones y Velocidad inicial
    X, V = init(npart,box,temp_red,dt_red,sigma)

    #Calculo fuerzas iniciales
    F,Epot = force(npart,X,rcut/sigma,box/sigma,epsilon)

    unit_red = {'t_unit': t_unit, 'T_unit': temp_unit, 'm_unit': m_uni
t, 'E_unit':epsilon, 'x_unit': sigma,
                'box': box/sigma, 'rcut':rcut/sigma, 'Temp0':temperatur
a, 'sigma':sigma, 'epsilon':epsilon,
                 'dt':dt, 'm':m}

    return X, V, F, unit_red

def simular(nsteps, X, V, F, unit_red, tau_ber, plot_freq):
    """
    Corre una simulación nsteps pasos a partir de las coordenadas, vel
ocidades
    y fuerzas iniciales. Si se provee un tau_ber > 0, utiliza un termo
stato de
    Berendsen para realizar una simulación a T constante (provista en
los parámetros
    de la corrida). plot_freq define cada cuantos pasos se grafica. Ad
emás de los gráficos
    de posición en función del paso, Temperatura y Energías, provee co
```

```
ordenadas, velocidades
    y fuerzas finales de la simulación.

    """
    sigma = unit_red['sigma']
    epsilon = unit_red['epsilon']
    dt = unit_red['dt']/unit_red['t_unit']
    box = unit_red['box']
    temp0 = unit_red['Temp0']
    m = unit_red['m']
    rcut = unit_red['rcut']
    temp_unit = unit_red['T_unit']

    if plot_freq < nsteps:
        fig = plt.figure()
        ax1 = fig.add_subplot(2,2,1)
        ax2 = fig.add_subplot(2,2,3)
        ax3 = fig.add_subplot(2,2,2)
        fig.show()
        fig.canvas.draw()

    #inicializo
    E = np.zeros((nsteps,2))
    T = np.zeros((nsteps,1))

    if tau_ber > 0:
        berendsen = True
    else:
        berendsen = False

    for i in range(nsteps):
        #Calculo Posicion
        Xnew = pos(npart,X,V,F,dt,box,m)

        #Calculo Fuerzas nuevas
        Fnew, Epot = force(npart,Xnew,rcut,box,epsilon)
        E[i,0] = Epot

        #Calculo velocidad
        Vnew, Ekin = vel(npart,V,F,Fnew,dt,m)
        E[i,1] = Ekin
        T_i = sum(sum(Vnew**2))/(2*npart)*temp_unit
        T[i] = T_i

        if berendsen:
            l = np.sqrt(1 + (dt/tau_ber)*(temp0/T_i - 1))
            Vnew = Vnew*l

        #Actualizo variables
        X = copy(Xnew)
```

```
        V = copy(Vnew)
        F = copy(Fnew)

        if (plot_freq < nsteps) and (i % plot_freq == 0):
            ax1.clear()
            ax1.plot(X[:,0]*sigma,X[:,1]*sigma,'b.')

            ax1.axis('equal')
            ax1.set_aspect('equal', 'box')
            ax1.set_xlim(0,box*sigma)
            ax1.set_ylim(0,box*sigma)

            ax2.clear()
            ax2.plot((E[0:i,0]+ E[0:i,1])/npart,'r', label = 'Total')
            ax2.plot(E[0:i,0]/npart,'g', label = 'Potencial')
            ax2.plot(E[0:i,1]/npart,'b', label = 'Cinetica')
            ax2.set_xlabel('#Paso')
            ax2.set_ylabel('E(kJ/mol)')
            ax2.legend(loc='upper center', bbox_to_anchor=(0.5, -0.1),
ncol=3, fancybox=True, shadow=True)

            ax3.clear()
            ax3.plot(T[0:i],'r')
            ax3.plot((i+1)*[temp0],'k--')
            ax3.set_xlabel('#Paso')
            ax3.set_ylabel('T(K)')

            fig.canvas.draw()

    return X, V, F
```

Generación de una corrida. Inicialización de parámetros y simulación. Termalización a 119.8K por 2000 pasos.

In [14]:
```
#Corrida
nsteps = 2000
npart  =  144
box = 40
temperatura = 119.8 #K
dt = 1e-14 # en segundos
sigma = 3.405 #A
epsilon = 0.996 #kJ/mol
masa = 39.95 #g/mol
tau_ber = 1
plot_freq = 1

#Inicializo con berendsen tau = 1 y corro 2000 pasos
X, V, F, unit_red = inicializar(npart, box, temperatura, dt, sigma, ep
silon, masa)
X, V, F = simular(nsteps, X, V, F, unit_red, tau_ber, plot_freq)
```



In [15]:
```
X
```

Out[15]:
```
array([[ 2.9807683 ,  2.03025418],
       [10.93873015,  6.42566827],
       [ 2.23639228,  2.71886996],
```

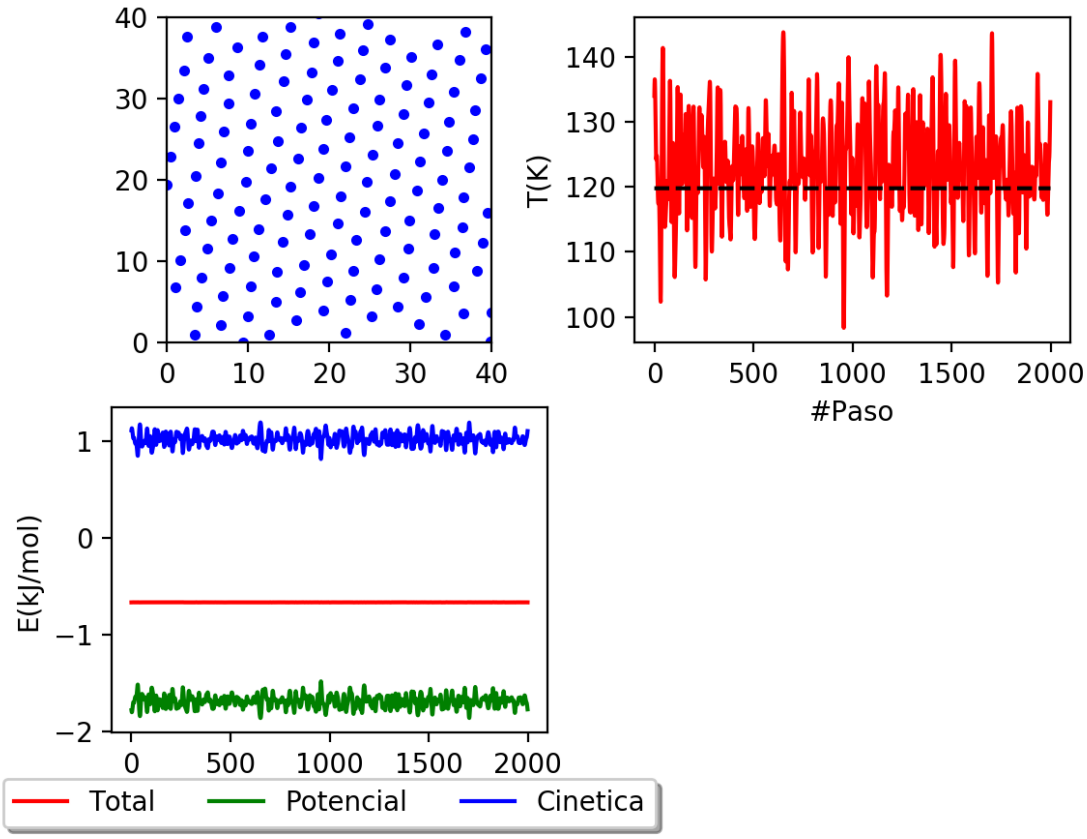
```
[11.35129716,  1.11282248],
[ 8.81860585,  4.52249217],
[ 9.58769799,  9.69885014],
[10.23068053,  8.07157928],
[ 5.60956161,  1.08017115],
[ 0.87080046,  1.15867348],
[ 3.68374123,  1.34233981],
[ 1.20412737,  2.28342109],
[ 6.6333089 ,  7.51120378],
[11.13593514,  7.47759662],
[ 7.55828121,  1.86325078],
[ 0.06986878,  5.76929339],
[ 6.49800576,  6.48214484],
[ 5.12071627,  3.91413683],
[ 0.94938343,  6.05246837],
[ 6.20468239,  4.31016004],
[ 3.10909319,  7.92521071],
[ 7.7102141 ,  8.88238601],
[10.89779212,  5.3188146 ],
[ 4.39665179, 11.42490602],
[ 0.30577519,  7.82095106],
[ 0.85963072,  5.01104122],
[10.41582744,  3.26259317],
[ 5.93214209,  9.09916455],
[ 2.23550411,  8.47657261],
[ 1.75283163,  0.58137961],
[ 8.90158996, 10.46461272],
[ 3.73113783,  6.20675485],
[ 3.11281762,  9.00661433],
[11.22533018,  8.49193754],
[ 9.84044161, 10.81369124],
[ 8.53080361,  2.47014031],
[ 9.42755022,  1.71524144],
[ 5.78383728,  2.17381313],
[ 3.1744556 ,  3.01397933],
[11.50850738,  0.08319544],
[ 7.91632322,  4.01391411],
[ 3.44220699, 10.99125931],
[ 9.20026601,  6.60649394],
[ 7.44883438,  6.82516323],
[ 9.33040678,  7.65974876],
[ 4.13555462,  3.49012748],
[ 6.86959072,  2.52452128],
[ 6.07789561, 10.17104027],
[ 4.6657209 ,  5.61847343],
[ 3.97859757,  8.30102637],
[ 2.64385806, -0.05317659],
[ 0.59838227,  9.92422227],
[10.48383486,  2.1625377 ],
[ 9.14317883,  0.69127978],
```

```
[10.0403702 ,  6.01766637],
[11.41658582,  9.47635798],
[10.79894851, 11.12818707],
[ 1.12065481,  7.05770454],
[ 3.65693469,  0.284898  ],
[ 5.95337473,  3.18337399],
[10.08433496,  7.02238151],
[ 6.8529278 ,  9.54664023],
[ 4.72480342,  6.61407264],
[ 5.42934494, 11.84831281],
[ 2.4574861 ,  3.69006756],
[ 1.6157459 ,  4.3006373 ],
[ 8.2339367 ,  6.1833352 ],
[ 0.50799738,  8.80616449],
[10.07218779,  0.06782995],
[ 7.93454325,  9.97947692],
[10.52368467, 10.15582178],
[ 9.87171484,  4.93731407],
[ 4.60732165,  0.83215526],
[ 2.51482729, 10.65182076],
[ 5.818607  ,  8.05044454],
[11.30058139,  2.70252034],
[ 7.42698512,  0.93083057],
[ 6.36178729,  5.36401224],
[ 4.94794712,  2.86372592],
[ 4.08085162,  2.38480419],
[ 8.06181121,  5.07933818],
[ 4.35772779,  4.6248957 ],
[ 1.94148966,  6.48611875],
[ 0.63199002,  3.97048487],
[ 6.74551818,  8.5484771 ],
[11.40400258,  3.64324545],
[ 7.2889696 ,  5.78531636],
[ 9.64013206,  3.92547572],
[ 1.23626585,  8.10086469],
[ 9.09455138,  5.56545009],
[ 3.88267801,  7.16719564],
[ 7.0927552 , 10.60268691],
[ 9.01618938, -0.26899414],
[ 9.4238739 ,  8.63878753],
[ 0.83993815,  0.17005136],
[ 6.85292502,  3.5649551 ],
[ 7.61928663,  7.8450179 ],
[ 2.72258915,  5.85427925],
[ 8.48131461,  8.29556676],
[ 8.1709056 , 12.0587765 ],
[ 9.51250544,  2.74126491],
[ 8.42963119,  1.34002788],
[ 0.29077802,  2.00044914],
[ 6.24434227, 11.22411963],
```

```
[ 7.10916979,  4.64905917],
[ 8.37837132,  7.23960825],
[ 7.20083571, 11.55803821],
[ 5.5917395 ,  6.08320297],
[ 5.39217977,  4.91759435],
[10.30237936,  1.03649417],
[ 3.31933701, 10.03066392],
[ 1.81640728,  5.43461236],
[ 5.6760668 ,  7.14519481],
[ 3.58402297,  5.20484411],
[ 4.12490231,  9.34780757],
[11.63598433,  4.67630439],
[ 1.59853477, 11.26036098],
[ 0.72951039, 10.88239534],
[ 2.03126125,  7.47817792],
[10.60937335,  4.27007125],
[ 8.73788764,  9.36737411],
[ 4.33311554, 10.4594578 ],
[ 8.0232439 , 11.0129346 ],
[ 5.08898591,  9.71825206],
[ 1.98320881,  1.6646361 ],
[ 8.78516892,  3.45473085],
[ 3.44976673,  4.19150788],
[11.56485092, 10.51478161],
[ 6.54416574,  1.51870497],
[ 1.44875405,  9.22129578],
[ 2.36345855,  9.68614141],
[ 4.82525056,  7.69362183],
[ 6.52733576,  0.53504944],
[ 2.78663205,  0.96337674],
[ 0.55248894,  3.04863054],
[ 1.48160354,  3.36178688],
[ 4.86463821,  1.7793041 ],
[ 7.78198423,  3.07006736],
[ 5.28667203, 10.81506513],
[ 4.97465135,  8.69856661],
[ 1.58161407, 10.20971413],
[ 2.88891491,  6.93288099],
[ 2.55091318,  4.71878916],
[ 0.12948334,  6.78536189],
[10.354288  ,  9.13908087]])
```

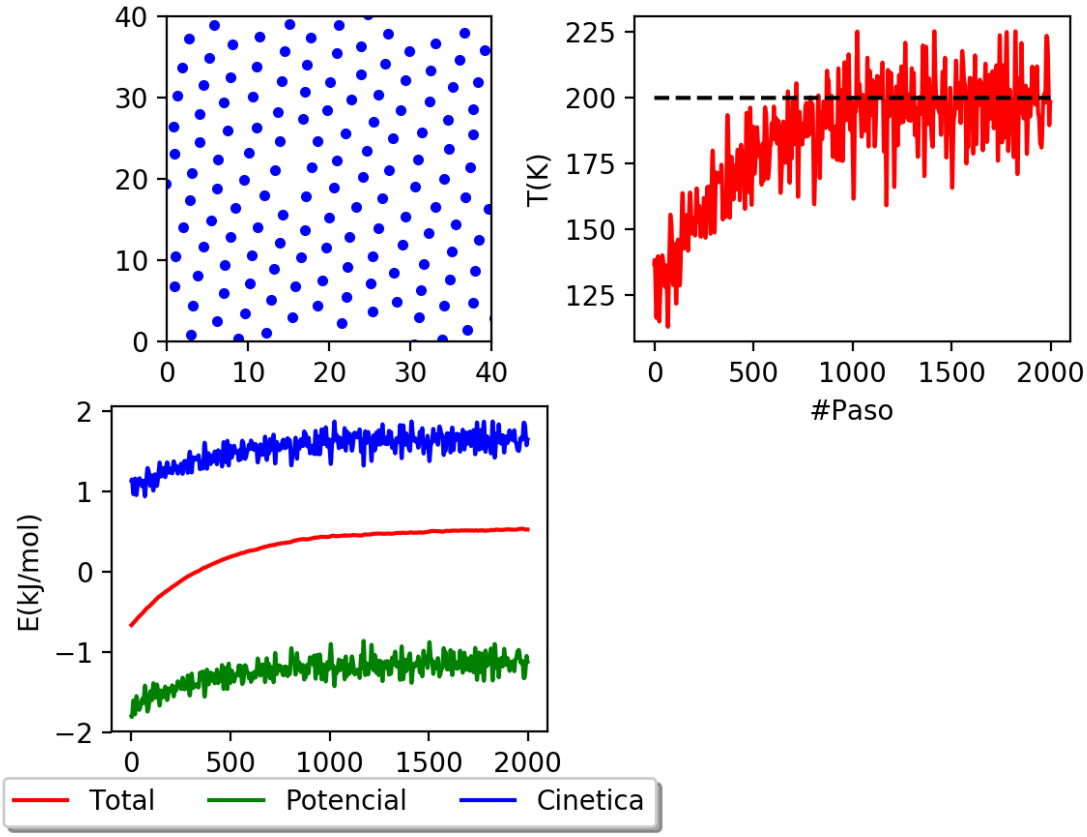
Testeo eliminar el termostato. Corrida de otros 2000 pasos.

In [16]:
```
#Pongo berendsen tau = 0 y corro 2000 pasos
nsteps = 2000
tau_ber = 0
X, V, F = simular(nsteps, X, V, F, unit_red, tau_ber, plot_freq)
V120 = copy(V)
```



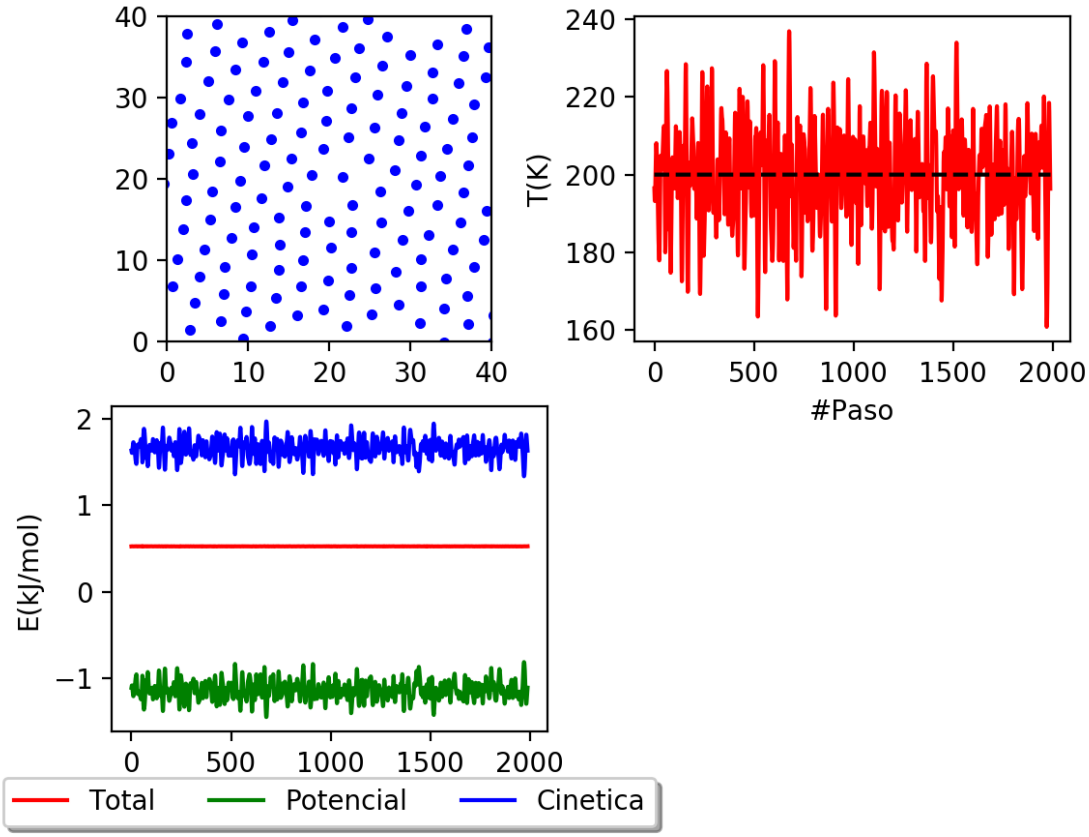
Cambio la T a 200K y pongo el termostato.

In [17]:
```
#Cambio la temperatura del sistema a T = 200K, pongo un berendsen y co
rro 2000 pasos
unit_red['Temp0'] = 200
tau_ber = 1.0
nsteps = 2000
X, V, F = simular(nsteps, X, V, F, unit_red, tau_ber, plot_freq)
```



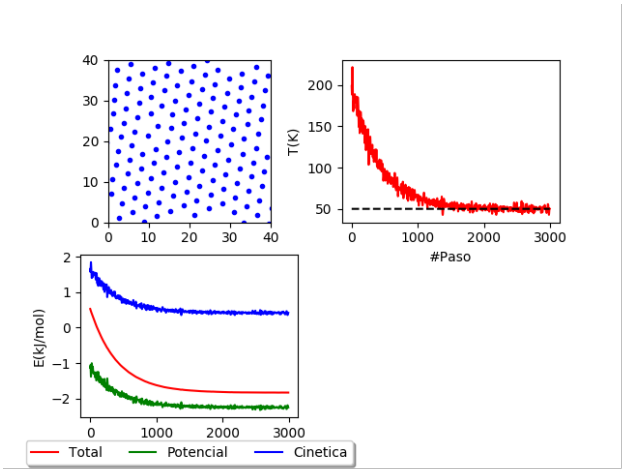
Elimino el termostato y chequeo que la T se mantiene.

In [18]:
```
#Pongo berendsen tau = 0 y corro 2000 pasos
nsteps = 2000
tau_ber = 0
plot_freq = 10
X, V, F = simular(nsteps, X, V, F, unit_red, tau_ber, plot_freq)
V200 = copy(V)
```



Enfrio el sistema a 50K usando el Berendsen.

In [19]:
```
#Cambio la temperatura del sistema a T = 50K, pongo un berendsen y cor
ro 1000 pasos
unit_red['Temp0'] = 50
tau_ber = 1.0
nsteps = 3000
X, V, F = simular(nsteps, X, V, F, unit_red, tau_ber, plot_freq)
```



Elimino el Berendsen y veo la dinámica.

In [20]:
```
#Pongo berendsen tau = 0 y corro 2000 pasos
nsteps = 2000
tau_ber = 0
plot_freq = 10
X, V, F = simular(nsteps, X, V, F, unit_red, tau_ber, plot_freq)
V50 = copy(V)
```

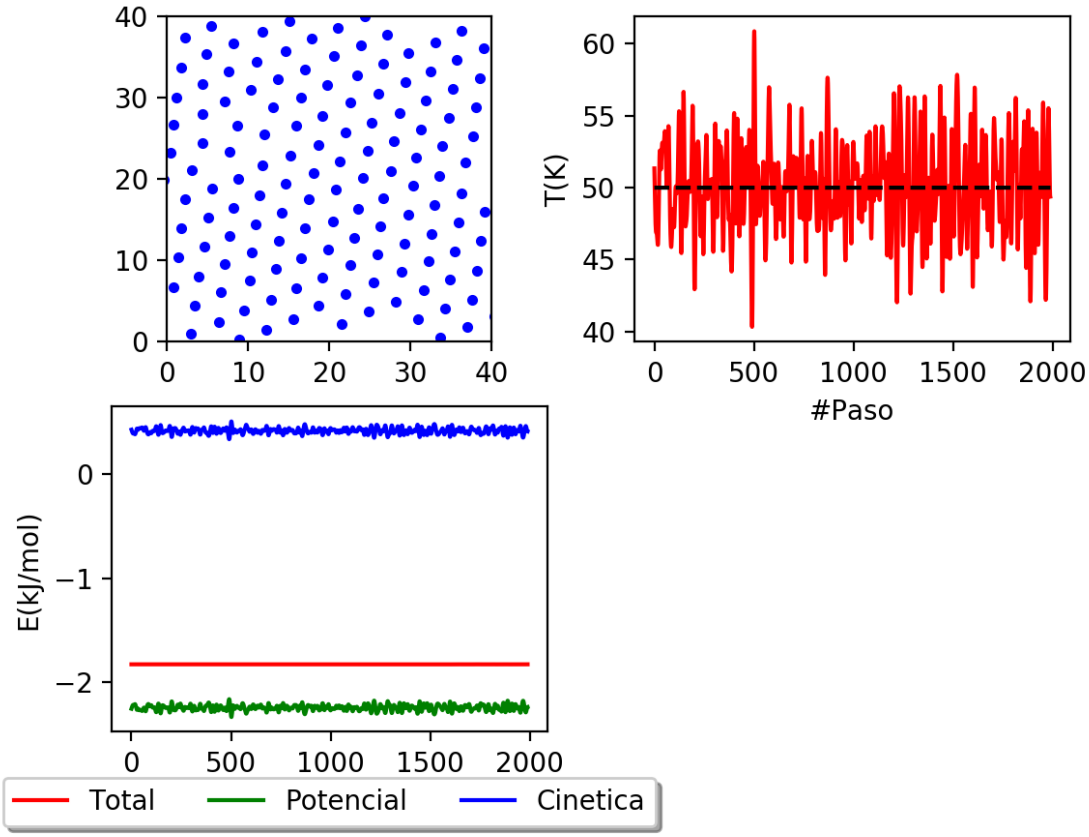


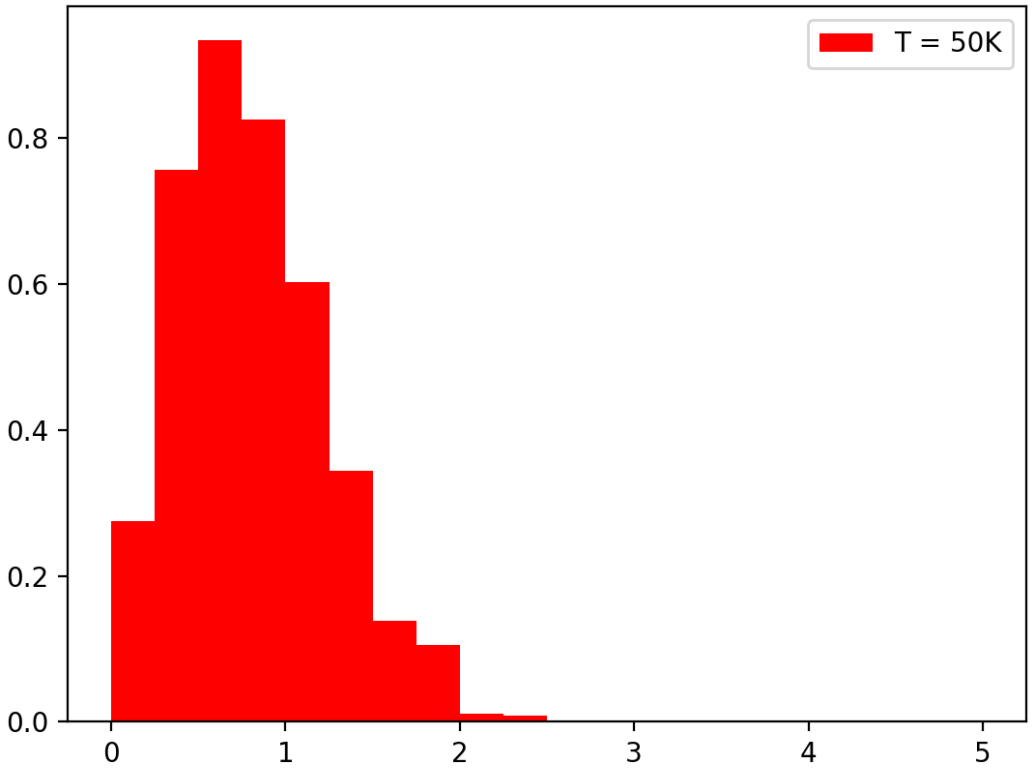
Grafico histogramas de las velocidades a las distintas temperaturas para los sistemas termalizados pero sin termostato.

In [21]:
```
fig = plt.figure()
ax1 = fig.add_subplot(3,1,1)
ax2 = fig.add_subplot(3,1,2)
ax3 = fig.add_subplot(3,1,3)
ax1.hist(np.sqrt(np.sum(V50**2, axis=1)), range = (0.0, 5.0), bins = 2
0, label= 'T = 50K', color = 'r', density=True)
ax1.legend()
ax2.hist(np.sqrt(np.sum(V120**2, axis=1)), range = (0.0, 5.0), bins =
20, label= 'T = 120K', color = 'g', density=True)
ax2.legend()
ax3.hist(np.sqrt(np.sum(V200**2, axis=1)), range = (0.0, 5.0), bins =
20, label= 'T = 200K', color = 'b')
ax3.legend()
plt.show()
```

Para aumentar la estádistica de los histogramas, termalizo y manteniendo el termostato realizo 10 simulaciones de 500 pasos, guardando las velocidades de las partículas. Asumo que en 500 pasos las velocidades están descorrelacionadas y por lo tanto es como muestrear independientemente las velocidades.

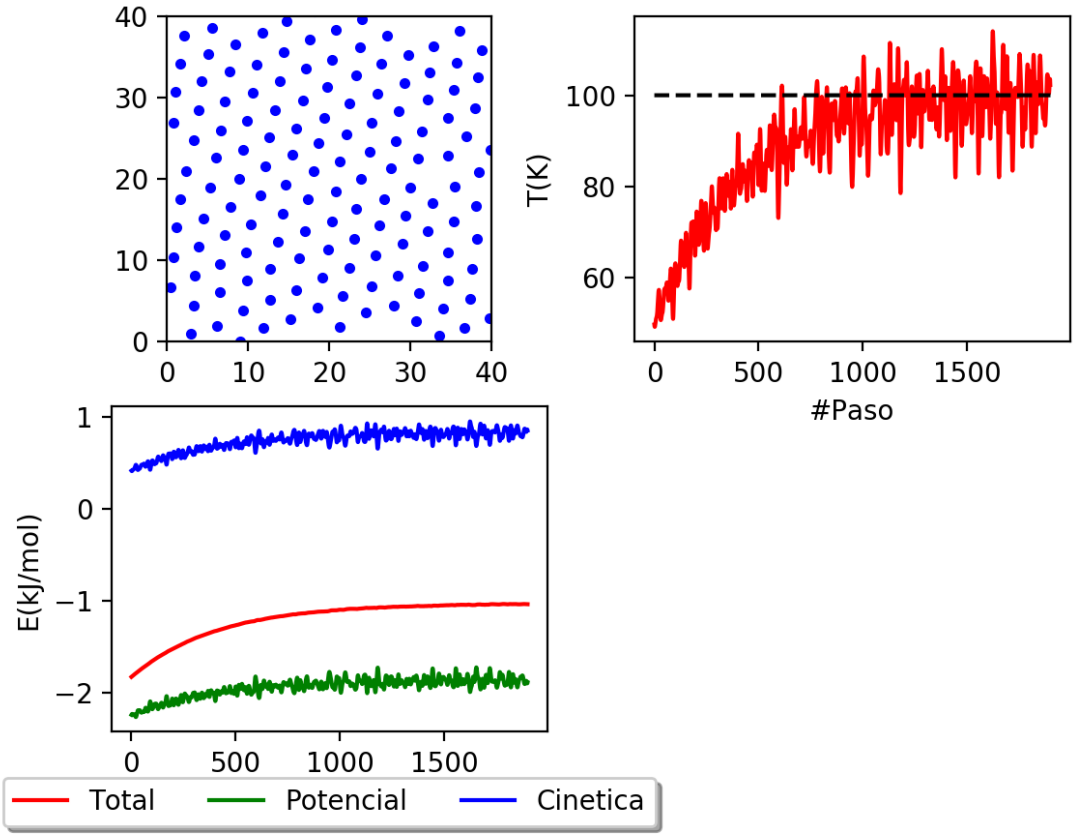
In [22]:
```
V50_x10 = []
nsim = 10
for i in range(nsim):
    print(i)
    nsteps = 500
    tau_ber = 1.0
    unit_red['Temp0'] = 50
    plot_freq = 1000
    X, V, F = simular(nsteps, X, V, F, unit_red, tau_ber, plot_freq)
    V50_x10.append(np.sqrt(np.sum(V**2, axis=1)))
V50_x10 = np.array(V50_x10)
V50_x10 = V50_x10.reshape(npart*nsim,1)
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
ax1.hist(V50_x10, range = (0.0, 5.0), bins = 20, label= 'T = 50K', col
or = 'r', density = True)
ax1.legend()
plt.show()
```

0
1
2
3
4
5
6
7
8
9

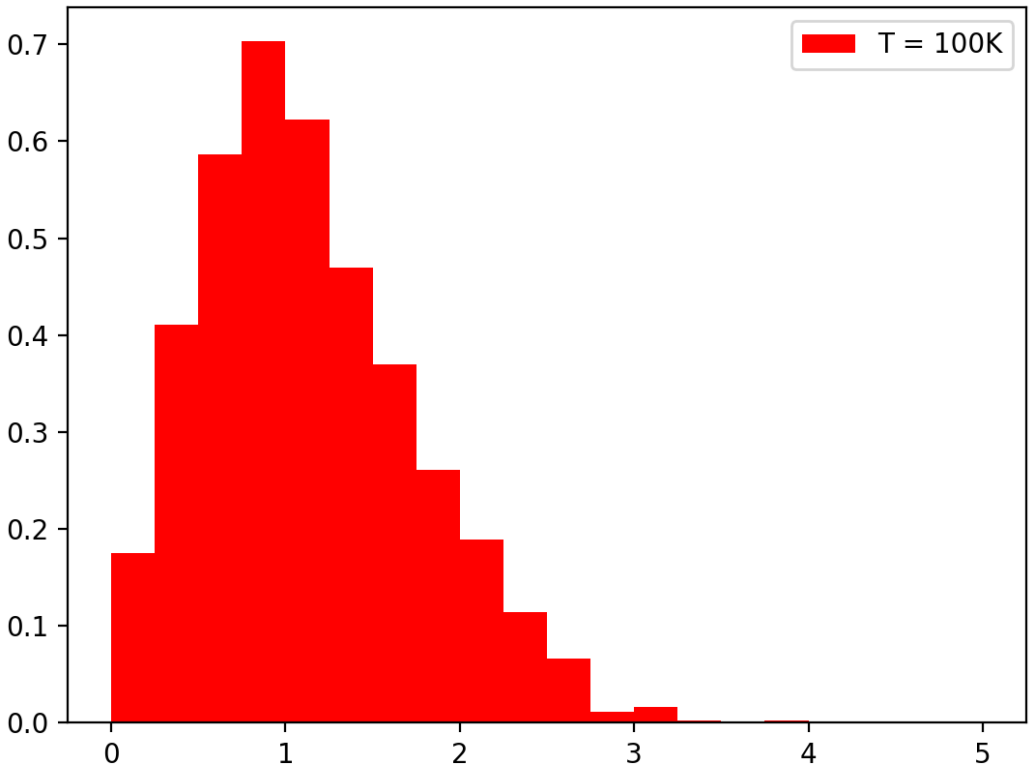


Repito a 100K

In [23]:
```
nsteps = 2000
tau_ber = 1.0
unit_red['Temp0'] = 100
plot_freq = 100
X, V, F = simular(nsteps, X, V, F, unit_red, tau_ber, plot_freq)

V100_x10 = []
nsim = 10
for i in range(nsim):
    print(i)
    nsteps = 500
    tau_ber = 1.0
    unit_red['Temp0'] = 100
    plot_freq = 1000
    X, V, F = simular(nsteps, X, V, F, unit_red, tau_ber, plot_freq)
    V100_x10.append(np.sqrt(np.sum(V**2, axis=1)))
V100_x10 = np.array(V100_x10)
V100_x10 = V100_x10.reshape(npart*nsim,1)
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
ax1.hist(V100_x10, range = (0.0, 5.0), bins = 20, label= 'T = 100K', c
olor = 'r', density = True)
ax1.legend()
plt.show()
```

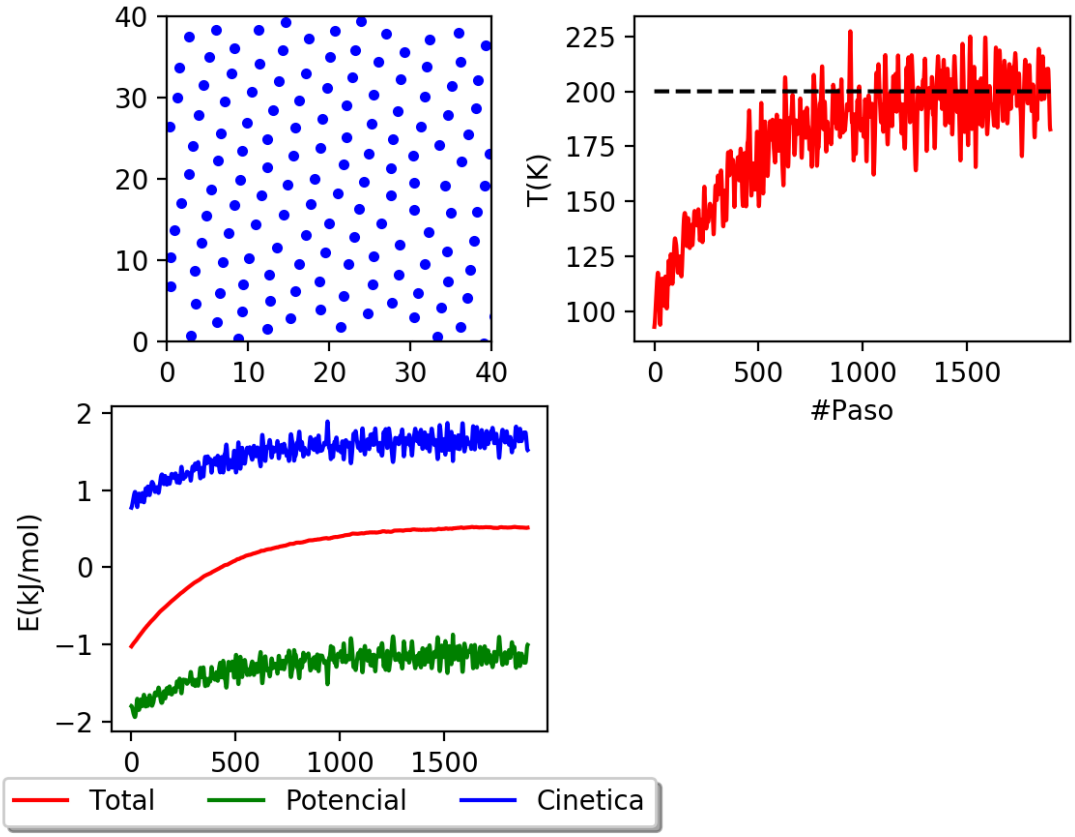


0
1
2
3
4
5
6
7
8
9

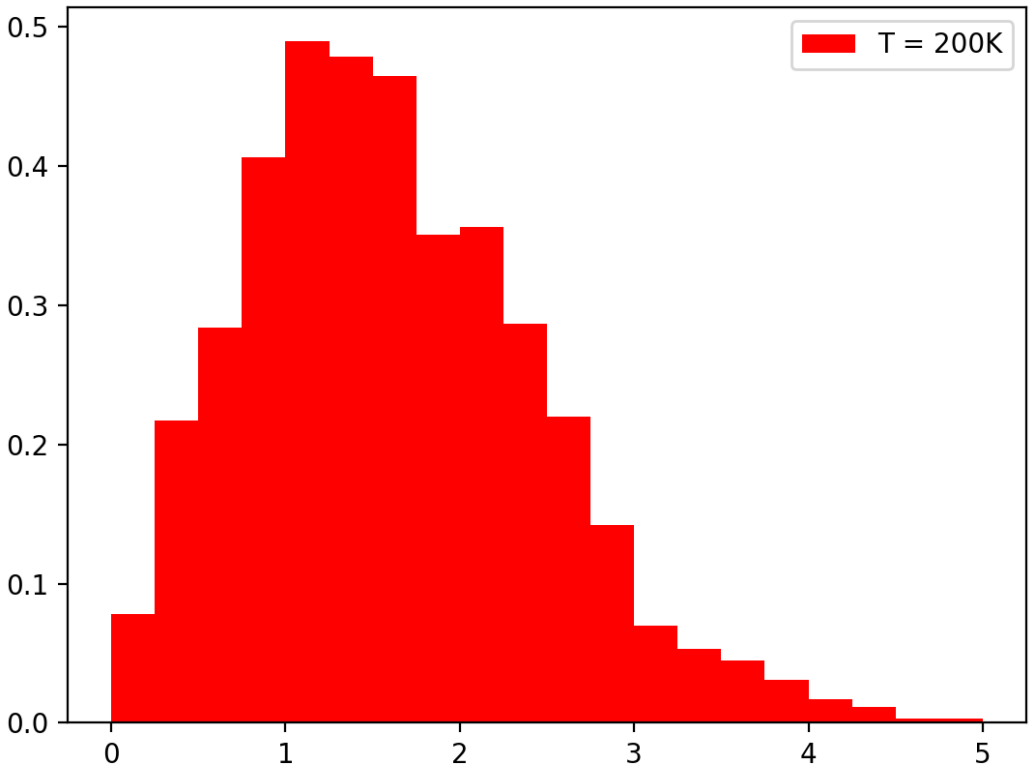


Repito a 200K

In [24]:
```
nsteps = 2000
tau_ber = 1.0
unit_red['Temp0'] = 200
plot_freq = 100
X, V, F = simular(nsteps, X, V, F, unit_red, tau_ber, plot_freq)

V200_x10 = []
nsim = 10
for i in range(nsim):
    print(i)
    nsteps = 500
    tau_ber = 1.0
    unit_red['Temp0'] = 200
    plot_freq = 1000
    X, V, F = simular(nsteps, X, V, F, unit_red, tau_ber, plot_freq)
    V200_x10.append(np.sqrt(np.sum(V**2, axis=1)))
V200_x10 = np.array(V200_x10)
V200_x10 = V200_x10.reshape(npart*nsim,1)
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
ax1.hist(V200_x10, range = (0.0, 5.0), bins = 20, label= 'T = 200K', c
olor = 'r', density = True)
ax1.legend()
plt.show()
```

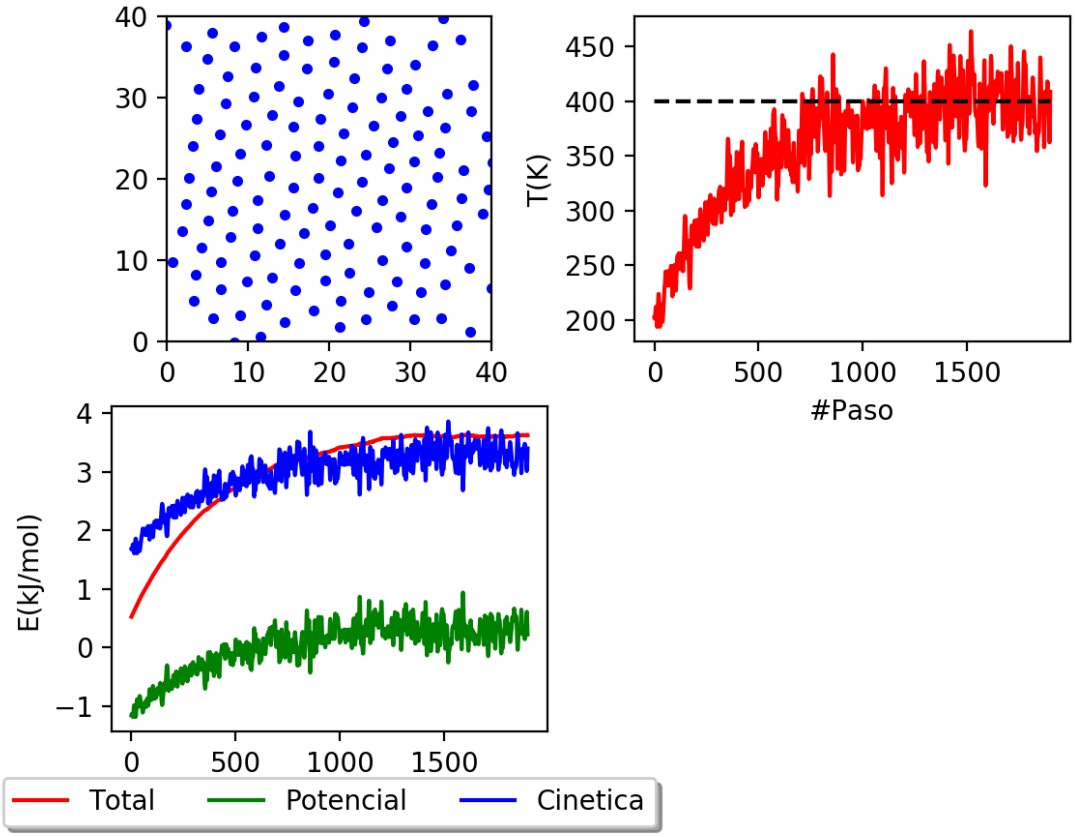


0
1
2
3
4
5
6
7
8
9



Repito a 400K!

In [25]:
```
nsteps = 2000
tau_ber = 1.0
unit_red['Temp0'] = 400
plot_freq = 100
X, V, F = simular(nsteps, X, V, F, unit_red, tau_ber, plot_freq)

V400_x10 = []
nsim = 10
for i in range(nsim):
    print(i)
    nsteps = 500
    tau_ber = 1.0
    unit_red['Temp0'] = 400
    plot_freq = 1000
    X, V, F = simular(nsteps, X, V, F, unit_red, tau_ber, plot_freq)
    V400_x10.append(np.sqrt(np.sum(V**2, axis=1)))
V400_x10 = np.array(V400_x10)
V400_x10 = V400_x10.reshape(npart*nsim,1)
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
ax1.hist(V400_x10, range = (0.0, 5.0), bins = 20, label= 'T = 400K', c
olor = 'r', density = True)
ax1.legend()
plt.show()
```



0
1
2
3
4
5
6
7
8
9

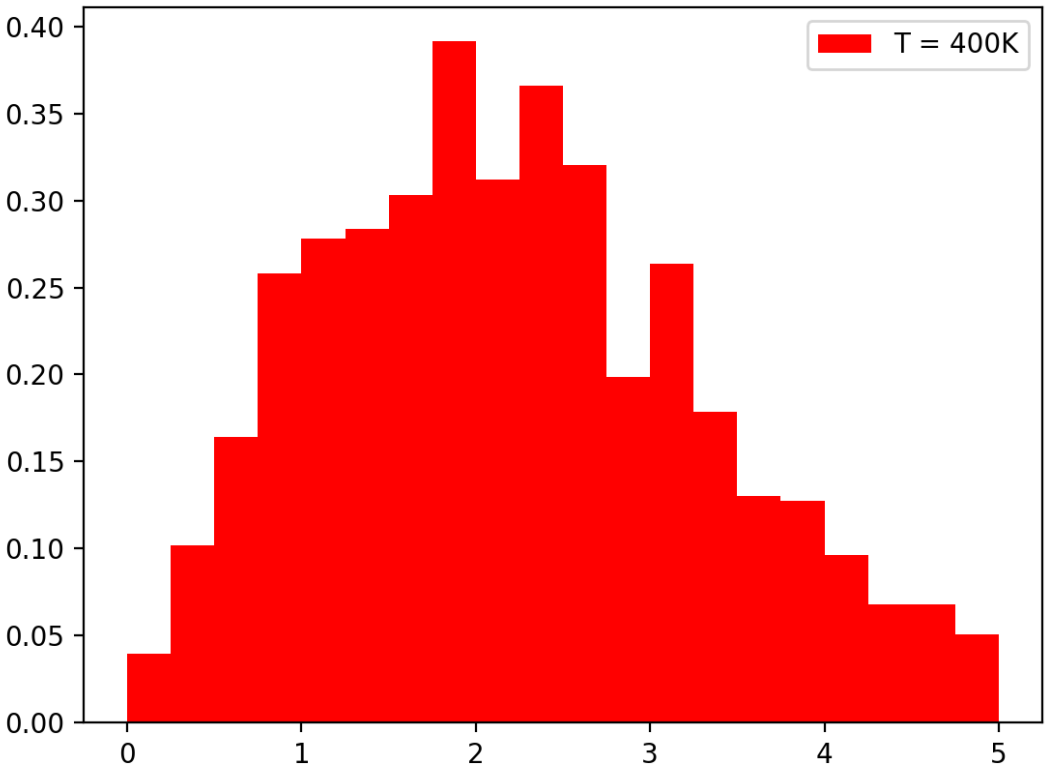
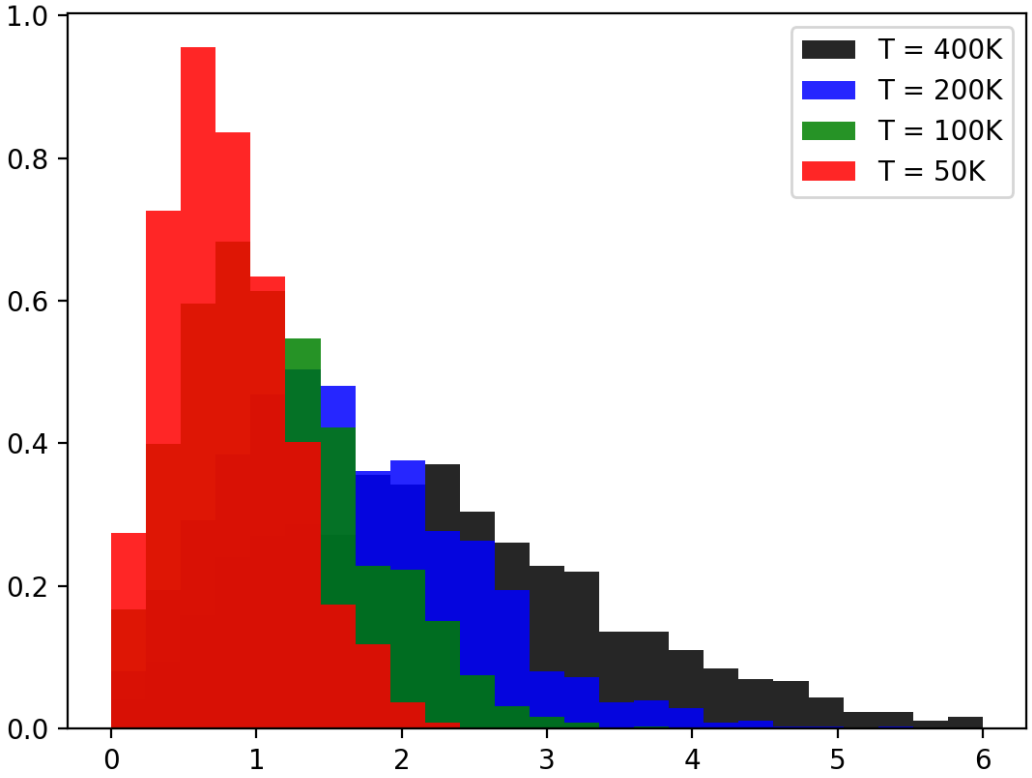


Grafico todos los histogramas juntos... Boltzman!

In [26]:
```
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
ax1.hist(V400_x10, range = (0.0, 6.0), bins = 25, label= 'T = 400K', c
olor = 'k', alpha = 0.85, density=True)
ax1.hist(V200_x10, range = (0.0, 6.0), bins = 25, label= 'T = 200K', c
olor = 'b', alpha = 0.85, density=True)
ax1.hist(V100_x10, range = (0.0, 6.0), bins = 25, label= 'T = 100K', c
olor = 'g', alpha = 0.85, density=True)
ax1.hist(V50_x10, range = (0.0, 6.0), bins = 25, label= 'T = 50K', col
or = 'r', alpha = 0.85, density=True)
ax1.legend()
plt.show()
```



In [ ]:
```

```