Thermostats_Ex

January 23, 2021

1. Perform a simulation of a Lennard Jones crystal with 864 particles. Temperature should be controlled using the Langevin thermostat at temperature T = 2. Try with different values of the friction (e.g. 0.01, 0.1, 1.0,10.0) and look at the difference in the behavior of the potential energy.

1.1) First, to generate a crystal with 864 particles, we need to remember the number of particles will be:

```
# particles = 4*n^3
```

Notice that, for n=6, the number of atoms is equal to 864 particles.

[75]:
```
%%bash
python3.7 ./lattice.py 6 > start.xyz
head start.xyz
```

```
864
10.0776 10.0776 10.0776
Ar 0.0 0.0 0.0
Ar 0.8398 0.0 0.8398
Ar 0.8398 0.8398 0.0
Ar 0.0 0.8398 0.8398
Ar 0.0 0.0 1.6796
Ar 0.8398 0.0 2.5194
Ar 0.8398 0.8398 1.6796
Ar 0.0 0.8398 2.5194
```

The first row is the number of particles: 864

The second row is the size of the box: 10.0776 10.0776 10.0776

The next rows are the coordinates of the Ar atoms.

1.2) Now, we need to open files to save the results for the different values of the friction:

We make a loop in bash in order to compute the energies and coordinates of velocity and momentum of each friction:

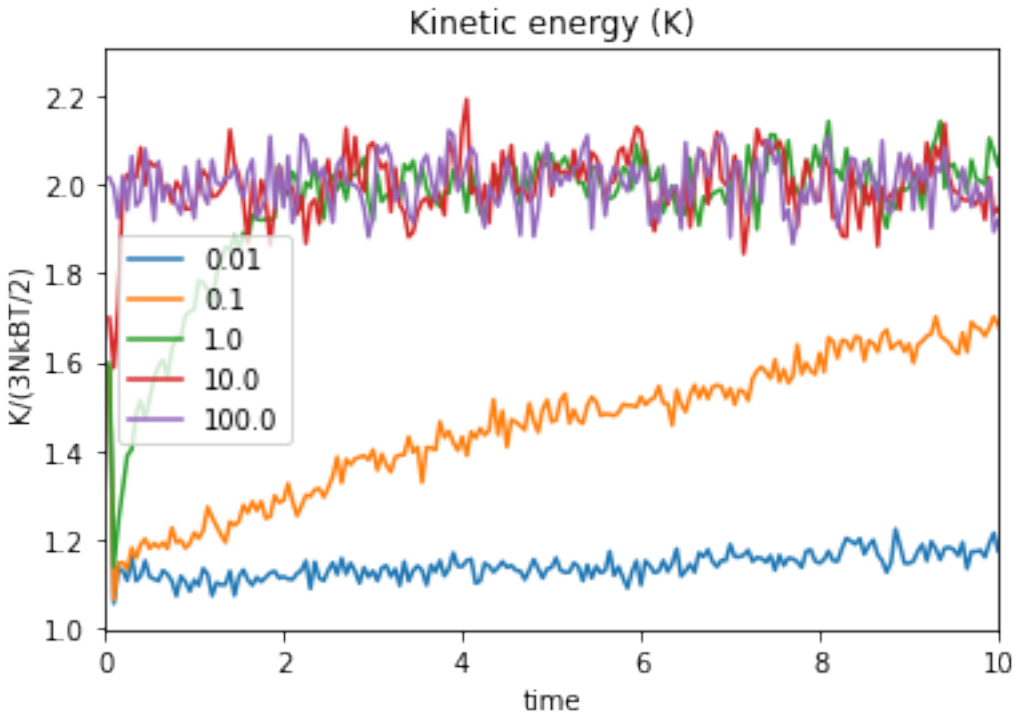
[249]:
```
energies[0.1].shape
#There are 2000 entries because we run simplemd.py for 20000 steps (nstep␣
↪20000) but we save the energy every 10 steps (nstat   10 energies-FRICTION.
↪dat)
```

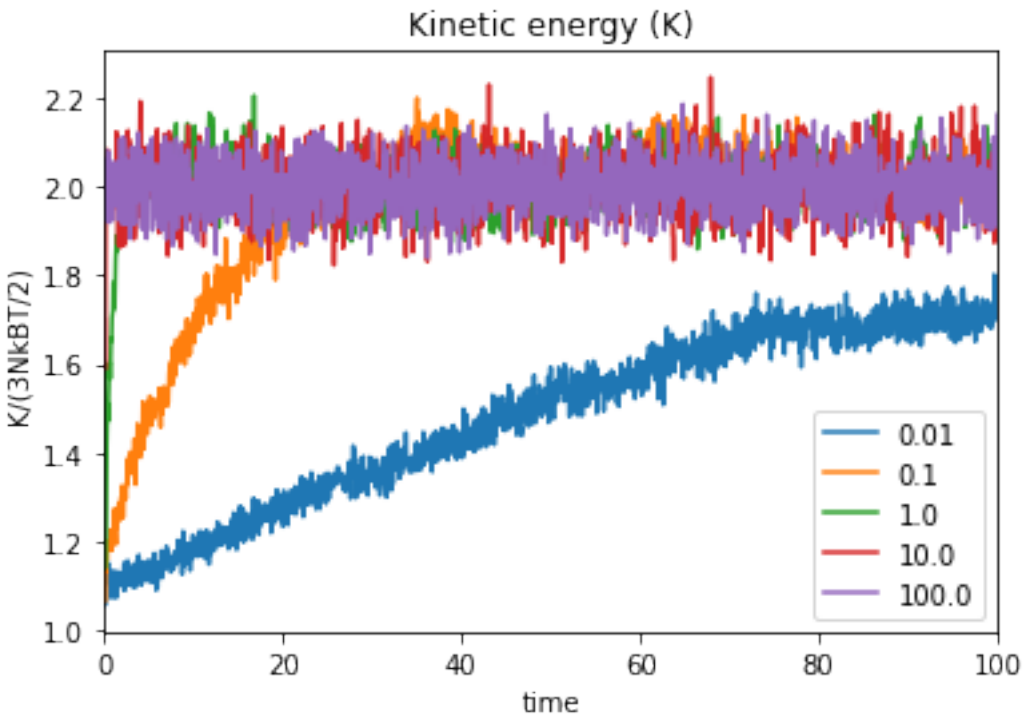
[249]: (6, 2000)

1.3) The kinetic and potential energies are plotted for the different values of the friction γ:

[250]:
```
for friction in energies.keys():
    plt.plot(energies[friction][1],energies[friction][2],label=str(friction))
    #The array energies[friction] has 6 columns: the first one is the time, the␣
 ↪second one is the kinetic energy and the third one is the potential energy
    plt.title("Kinetic energy (K)")
    plt.xlabel("time")
    plt.ylabel("K/(3NkBT/2)")
plt.legend()
plt.xlim((0,10))
plt.show()

for friction in energies.keys():
    plt.plot(energies[friction][1],energies[friction][2],label=str(friction))
    plt.title("Kinetic energy (K)")
    plt.xlabel("time")
    plt.ylabel("K/(3NkBT/2)")
plt.legend()
plt.xlim((0,100))
plt.show()
```

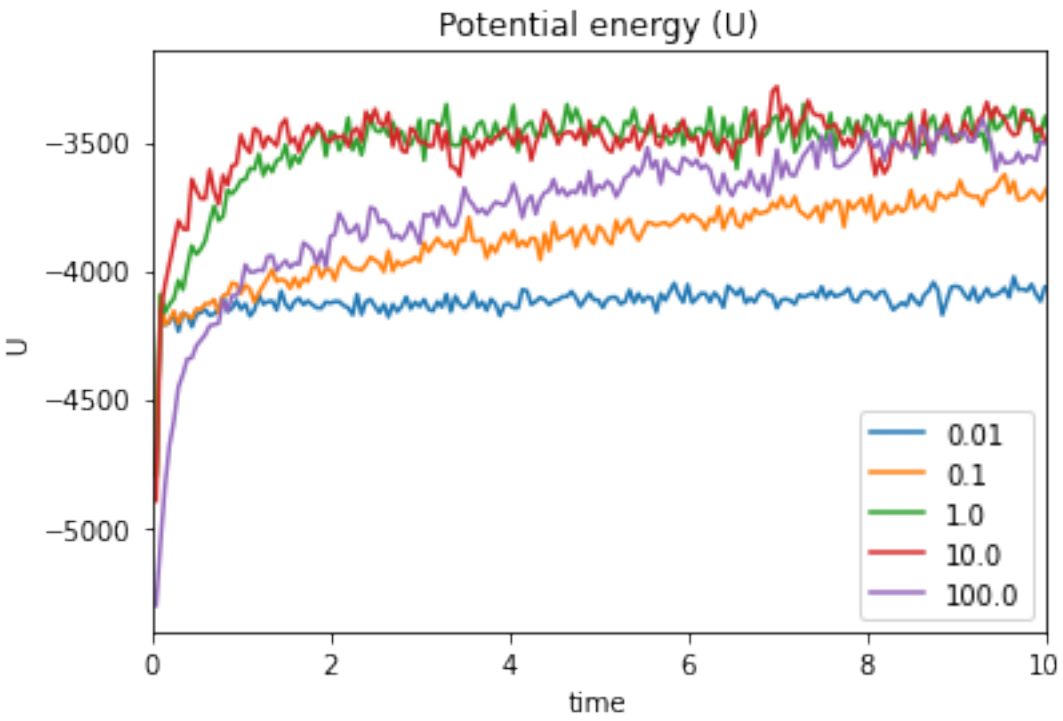


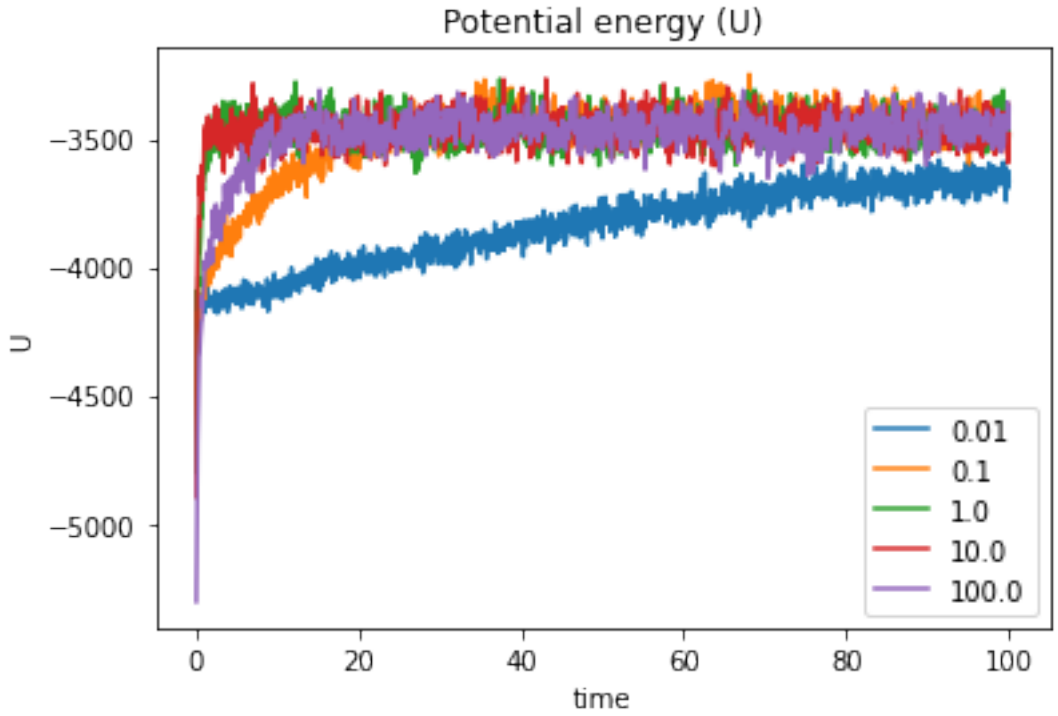


OBS: The higher the friction, the faster the kinetic energy will reach the equilibrium; in this case, at a temperature T=2.

[9]:
```
for friction in energies.keys():
    plt.plot(energies[friction][1],energies[friction][3],label=str(friction))
    plt.title("Potential energy (U)")
    plt.xlabel("time")
    plt.ylabel("U")
plt.legend()
plt.xlim((0,10))
plt.show()

for friction in energies.keys():
    plt.plot(energies[friction][1],energies[friction][3],label=str(friction))
    plt.title("Potential energy (U)")
    plt.xlabel("time")
    plt.ylabel("U")
plt.legend()
#plt.xlim((0,10))
plt.show()
```


Potential energy (U)
−3500
−4000
−4500
−5000
U
0
2
4
6
8
10
time
0.01
0.1
1.0
10.0
100.0


Potential energy (U)
−3500
−4000
−4500
−5000
U
0
20
40
60
80
100
time
0.01
0.1
1.0
10.0
100.0

OBS: At the beginning of the simulation, the system is in a minimun potential energy while the kinetic energy initializes from a value different from 0. When the simulation starts, U increases very fast because the system tries to reach the equipartion energy. After this very short transient, U starts to equilibrate and, as happened for K, for higher values of the friction γ, the equilibrium is reached faster. However, it's possible to see that this behaviour doesn't occure for $\gamma = 100$ (purple plot) at difference for the kinetic energy plot where it was the faster reaching T=2.

In conclusion, there is an optimal value for the friction γ that is between 1 and 10 (the read and green plots are almost equivalent); after γ=10, the systems equilibrates slower. The reason of this is that, for local thermostats, when the friction is too large, the thermostat will affect the dynamics of the particles more than the collisions between the particles. The difussion and relaxation of the system don't reach their lowest values because the particles interact more with the thermal bath of the thermostat than between them.

OBS: The smaller the system, the fewer differences between the global and the local thermostats. As the global thermostats can be seen as a reescaling of the momentum vector, the scale used is the minimum one that makes possible to achieve a new module of the p vector, modifing the momentum as little as possible, to obtain the same value of the kinetic energy. In this sense, the global thermostats affect less the dynamics of the particles than the local ones.

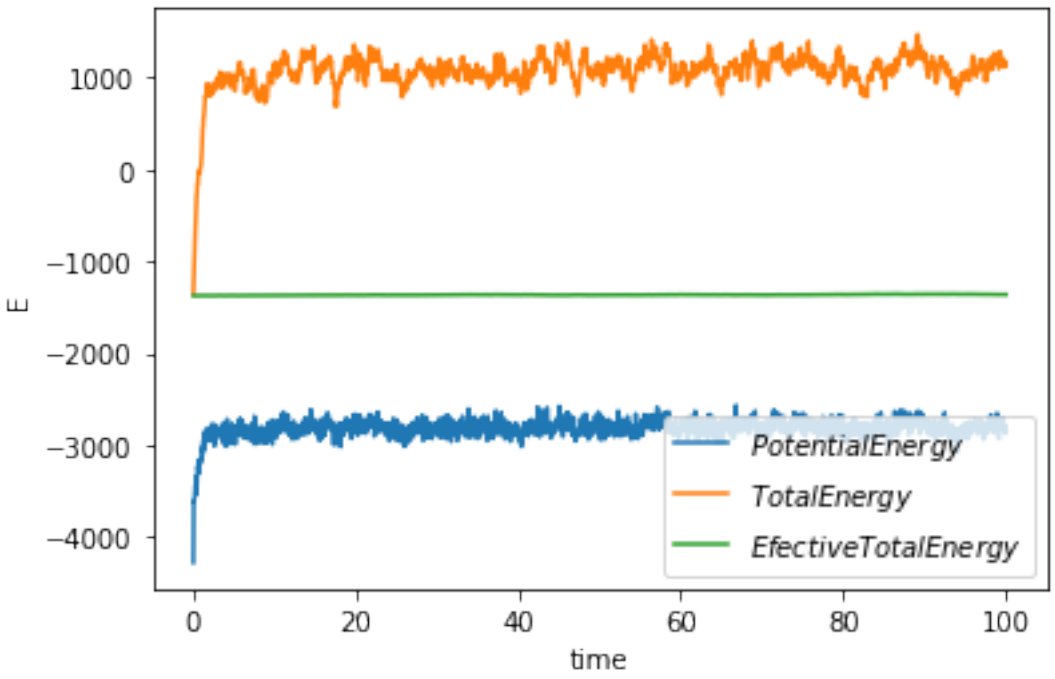
2. Compute the average potential energy for different values of the temperature (ranging from 0.1 to 3 in Lennard Jones units) using a fixed friction (e.g. = 1). Also compute the specific heat using energy flucutations and look at how the specific heat depends on the temperature. Remember that you should discard the initial equilibration.

2.1) Average potential energy for different values of the temperature.

NOTE: We can prove the energy conservation. As the orange and blue plots show that just on average, once the stationary state is reached, the total and the potential energy are constant, the correct form to prove it is shown in the green plot: the **efective total energy** is the conserved quantity that tells how much the detailed balance is violated. This quantity depends on the time step, the smaller Δt, the less fluctuations there will be.

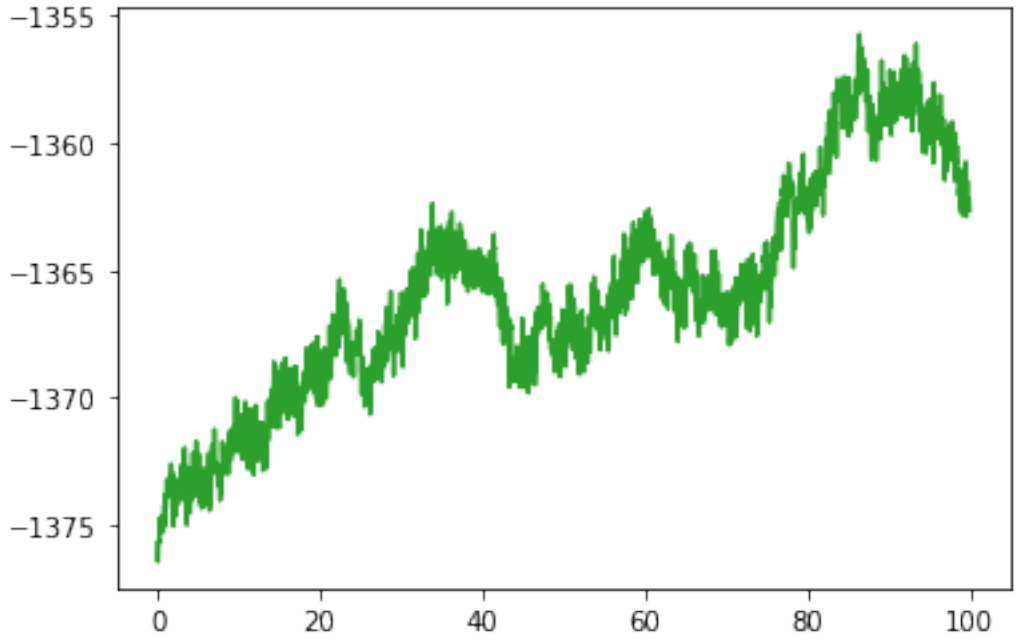
[12]:
```
ene = np.loadtxt("energies-t-3.0.dat").T;

plt.plot(ene[1],ene[3], label=r'$Potential Energy$')
plt.plot(ene[1],ene[4], label=r'$Total Energy$')
plt.plot(ene[1],ene[5], label=r'$Efective Total Energy$')
plt.xlabel("time")
plt.ylabel("E")
plt.legend()
 #plt.xlim((0,10))
plt.show()
```

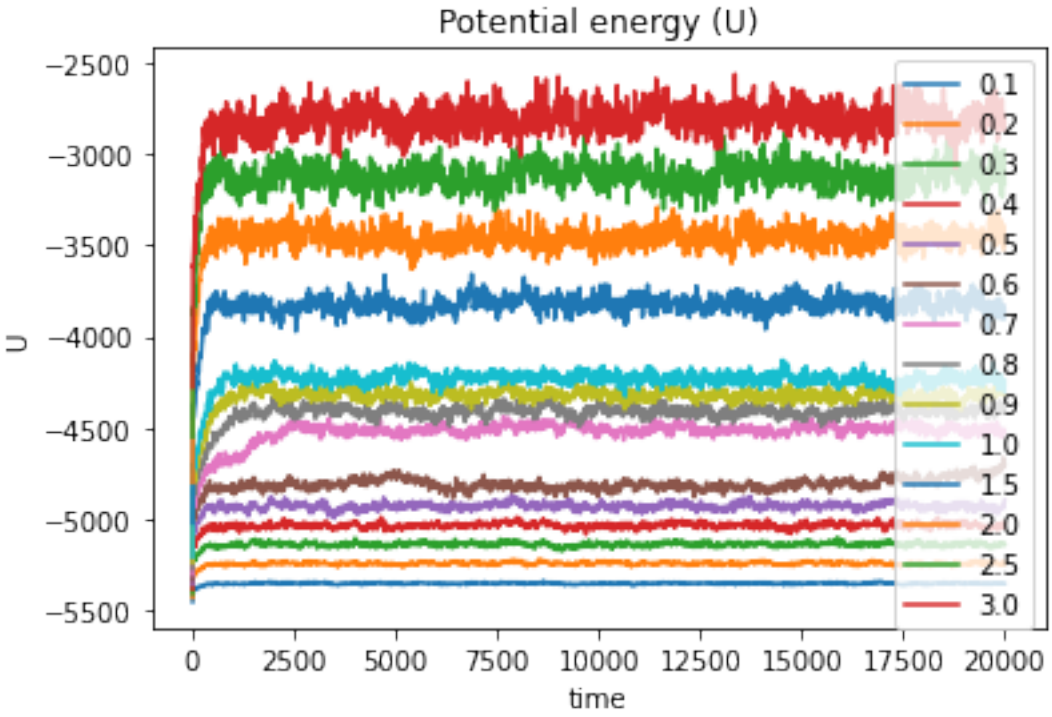


[13]:
```
plt.plot(ene[1],ene[5], color='tab:green' ,label=r'$Efective Total Energy$')
#Close up of the Efective Total Energy, the fluctuations are proportional to␣
 ↪the time step
```

[13]: [<matplotlib.lines.Line2D at 0x10d982070>]



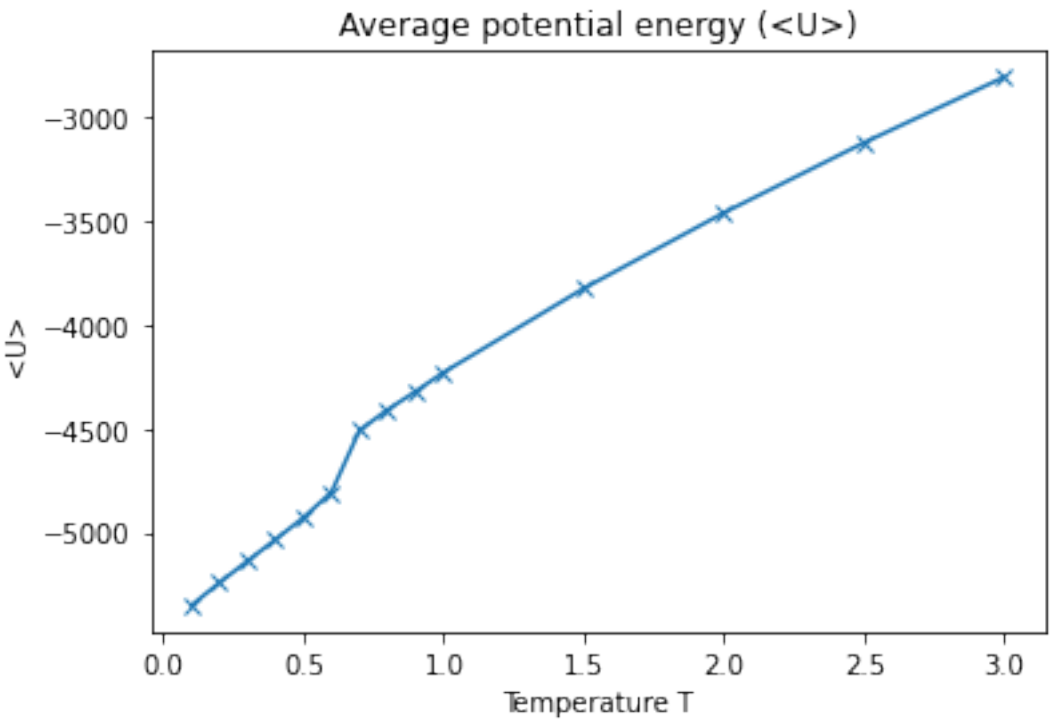
[15]:
```
for temperature in[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.5,2.0,2.5,3.0]:
    plt.
↪plot(energies_T[temperature][0],energies_T[temperature][3],label=str(temperature))
    plt.legend()
    plt.title("Potential energy (U)")
    plt.xlabel("time")
    plt.ylabel("U")
```



OBS: The potential energy U is proportional to the temperature. As after the inital transient U is stable, it's better to average U discarding the first period of the simulation -Bussi discard the first 10% = 200 time steps but for this simulation doesn't change a lot the result-.

[28]:
```
plt.plot(T,U,"x-")
plt.title("Average potential energy (<U>)")
plt.xlabel("Temperature T")
plt.ylabel("<U>")
```

[28]:
```
Text(0, 0.5, '<U>')
```



OBS: The potential energy U increases with the temperature. The jump that is seen between T = 5 and 10 is a PHASE TRANSITION and the difference of $< U >$ between the first and second slope is the extra energy that the system needs to increase the temperature above the melting point, i.e. the latent heat.

2.2) Specific heat using energy flucutations and its dependence on the temperature. To do it, it's used the heat capacity computing the derivative of U with respect to T.

Remember that:

The specific heat capacity of a substance, usually denoted by c, is the heat capacity C of a sample of the substance, divided by the mass M of the sample:

$$c = \frac{C}{M} = \frac{1}{M} \cdot \frac{\mathrm{d}Q}{\mathrm{d}T}$$

where $\mathrm{d}Q$ represents the amount of heat needed to uniformly raise the temperature of the sample by a small increment $\mathrm{d}T$. Its SI unit is (J/K).

The heat capacity of an object should be considered a function $c(p, T)$.

The first way to do it is using energy flucutations in dependence on the temperature. To do it, we should divide $Var(U)$ over T^2 where T is the thermostat temperature.

Note the instantaneous temperature is called "temperature" here and represents the kinetic energy expressed in temperature units (Kelvins):

$$K = \frac{3}{2} N K_B$$

where N = number of atoms.

The derivation of this last expression is given by:

$$< U >= \frac{\int dq e^{\beta U(q)} U(q)}{\int dq e^{\beta U(q)}}$$

$$\Rightarrow \frac{\partial < U >}{\partial \beta} = \frac{-\int dq U(q) e^{\beta U(q)} U(q)}{\int dq e^{\beta U(q)}} + \frac{\int dq e^{\beta U(q)} U(q)}{(\int dq e^{\beta U(q)})^2} \int dq e^{\beta U(q)} U(q)$$

$$\Rightarrow \frac{\partial < U >}{\partial \beta} = - < U^2 > + < U >^2$$

As

$$\beta = \frac{1}{K_B T} \Rightarrow \frac{\partial \beta}{\partial T} = \frac{1}{K_B T^2}$$

then:

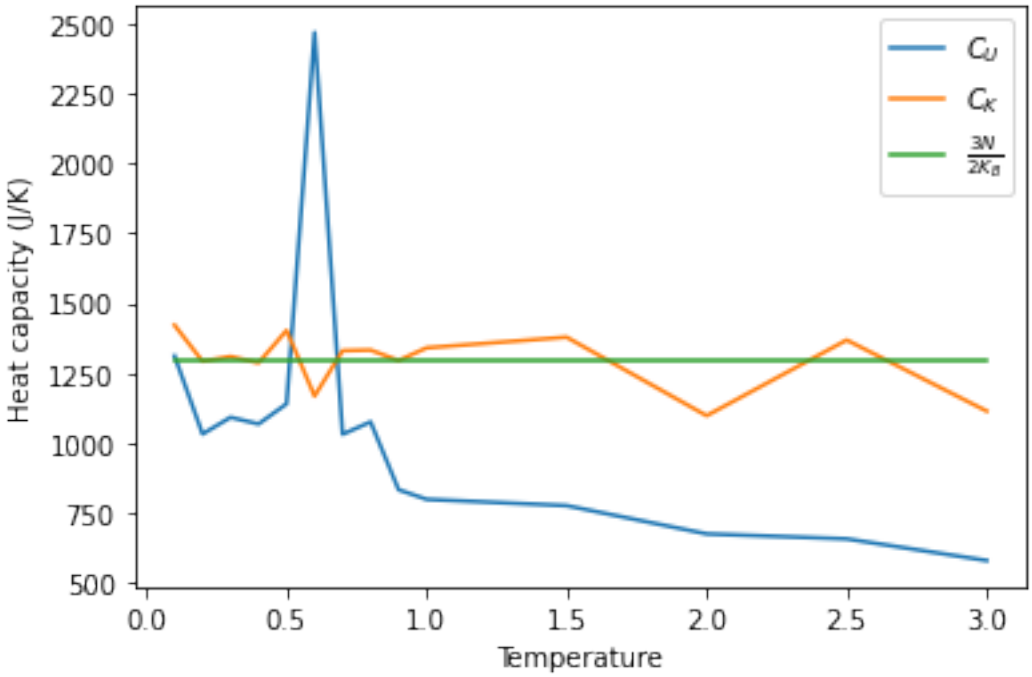
$$\frac{\partial < U >}{\partial T} = \frac{\partial < U >}{\partial \beta} \frac{\partial \beta}{\partial T} = \frac{< U^2 > - < U >^2}{K_B T^2}$$

$$\Rightarrow C_U = \frac{\sigma(U)^2}{T^2}$$

(This expression is equal for C_K and C_U)

[106]:
```
    plt.plot(T,C_U, label=r'$C_U$')
    plt.plot(T,C_K, label=r'$C_K$')
    plt.plot(T, 0*np.array(T)+3/2*864, label=r'$\frac {3N}{2K_B}$')
    # Note that the contribution of K to C is equal to 3N/2K_B \aprox 3/2*864
↪(K_B=1)
    plt.legend()
    plt.xlabel("Temperature")
    plt.ylabel("Heat capacity (J/K)")
```

[106]: Text(0, 0.5, 'Heat capacity (J/K)')



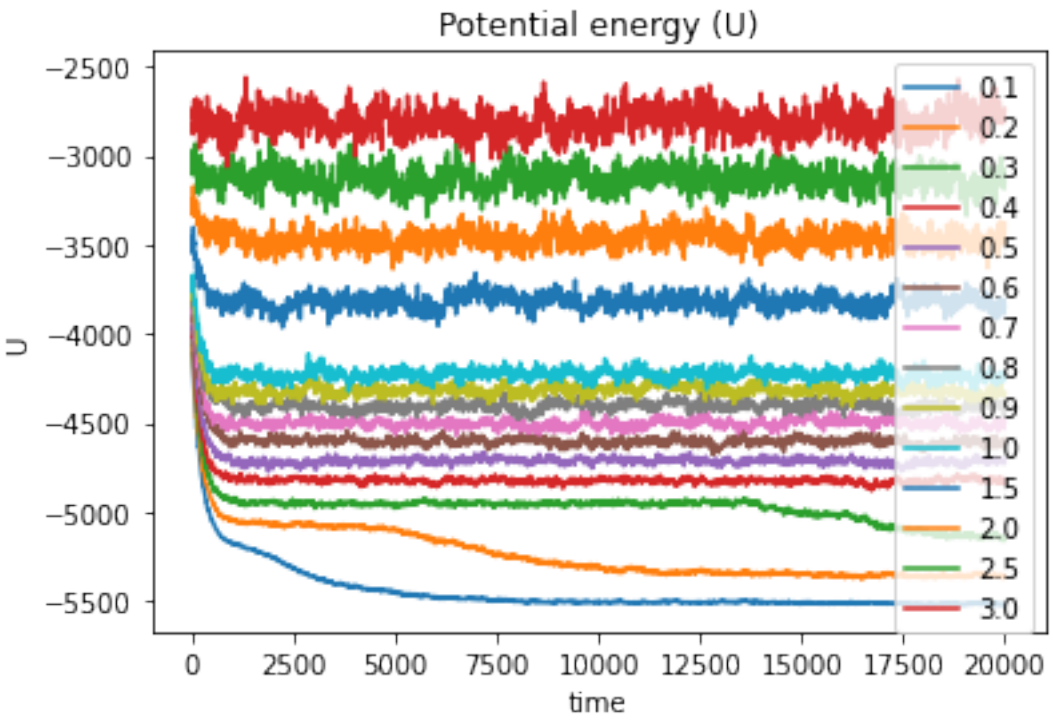
NOTE: The difference between the theorical (green) and numerical (orange) value of C_K is due to an statistical error: the trajectories are not long enough.

OBS: The meaning of the peak between T = 5 and 10 is a PHASE TRANSITION and corresponds to the change of energy that you need to add to the system. The blue plot here (C_U) is the derivative of the blue plot in the previous graph ($< U >$).

3. Repeat the same calculation but starting from a structure that has been equilibrated for a long time at temperature T=3. (hint: simplemd writes the final coordinates on a file, you should restart from that configuration). Are the value of average and fluctuations of U equal or difeerent with respect to previous points? For which values of T do you observe more diference?

3.1) We'll do the same previous calculation but starting from a structure that has been equilibrated for a long time at temperature T=3.

[29]:
```
for temperature in[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.5,2.0,2.5,3.0]:
    plt.
↪plot(energies_T_hot[temperature][0],energies_T_hot[temperature][3],label=str(temperature))
    plt.legend()
    plt.title("Potential energy (U)")
    plt.xlabel("time")
    plt.ylabel("U")
```



OBS: U starts high because the system was heated. The system reachs the equilibrium according to the temperature of each simulation. For the highest temperature (red plot), the system is already in a stationary state.

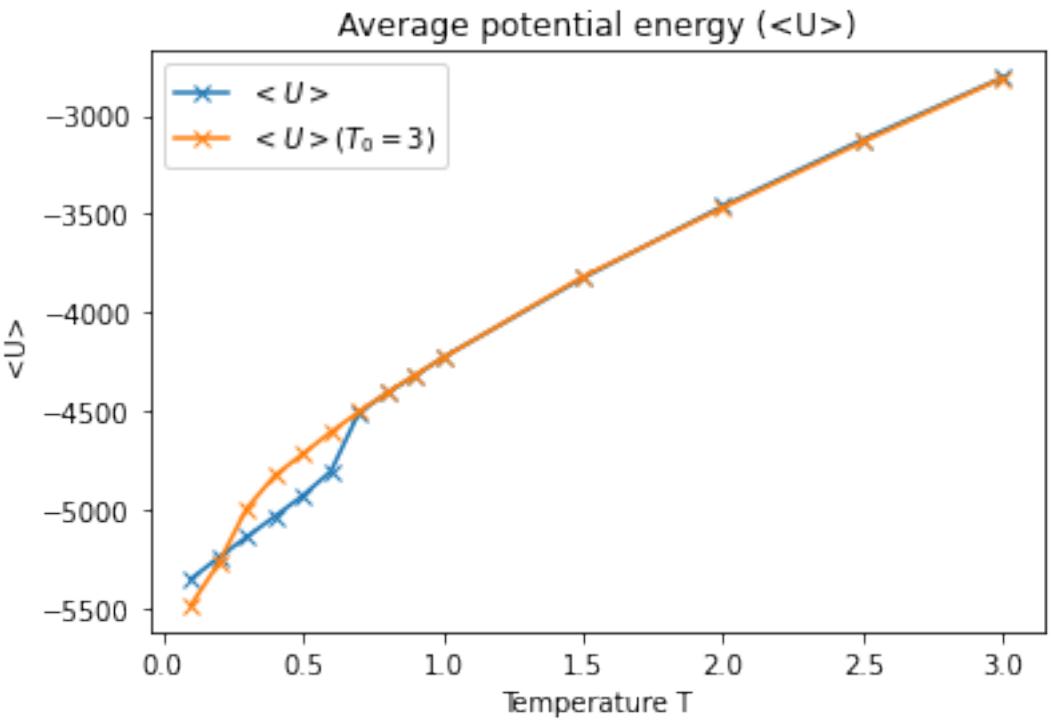
OBS: The melting point is around 0.7 K. Under this temperature, the system becomes to a super cooler liquid; after a time, the potential energy suddendly decreases (see plots of T=0.1,0.2,0.3) because the system finds a local minumum. In general, if the system is originally in a global minimum and if the system is heated and cooled it down again, it's not a fact that we'll find again a local minimun. The more complex is a system (more degrees offreedom), the less likely will be find a global minimum.

However, in this example, we find a global minimum (see plot of T=0.1) after heating and cooling the system.

3.2) To analize the value of average and fluctuations of U, we will compute $< U > vsT$.

[34]:
```
plt.plot(T,U,"x-",label=r'$<U>$')
plt.plot(T,U_hot,"x-",label=r'$<U> (T_0=3)$')
plt.legend()
plt.title("Average potential energy (<U>)")
plt.xlabel("Temperature T")
plt.ylabel("<U>")
```

[34]:
```
Text(0, 0.5, '<U>')
```



OBS: In both cases there is a transition from a solid to a liquid state, the differences is that the blue plot is cooled faster than the orange one where, instead of a jump, the relaxation time is slower. In the end, the final solid state of the heated system (orange) will be amorfous glass while for the other case (blue) the state will be a crystal.

OBS: There is a competition between two factors:

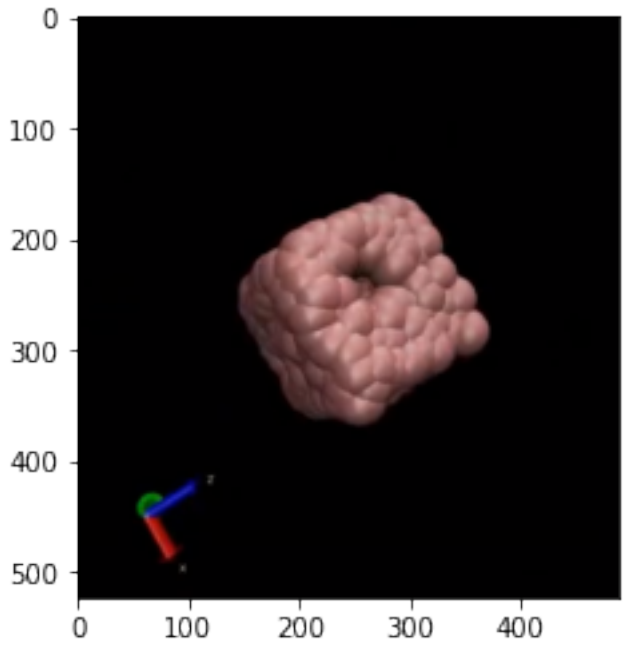
1) Box size: The highest the box, the less likely the particles can form an organized arragment (crystal) and the highest the hysteresis will be. If the box is smaller, the particles could find easier a minimun energy.

2) Temperature: If the system is heated at a highest initial temperature, the minimum energy of the orange plot will be highest that the blue one.

The final arragment of the heated system is shown here. The system has an empthy space because, in order to recover the fixed density -that is the density given by the box size and the number of atoms-, the system inserts a defect. In this way, the particles are more closer in the solid state but the density is preserved.

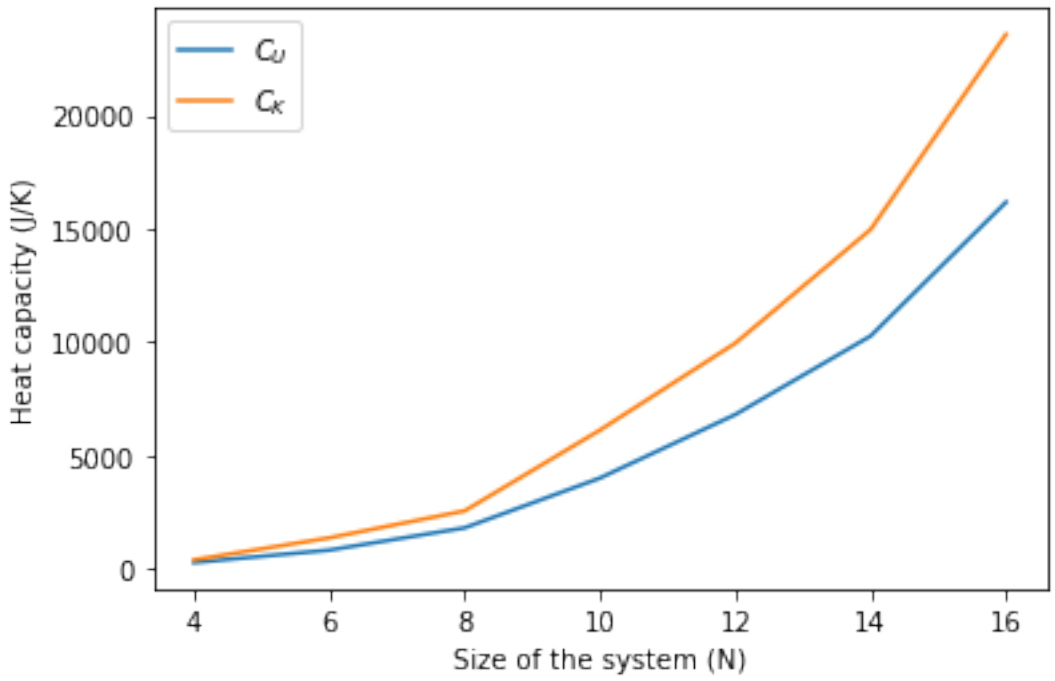
[4]:

[4]: `<matplotlib.image.AxesImage at 0x10bb92400>`



4. At fixed T=1, look at how specific heat depends on system size (try e.g. 256, 500, and 864 particles).

[82]:
```
plt.plot(S,C_U_S, label=r'$C_U$')
plt.plot(S,C_K_S, label=r'$C_K$')
plt.legend()
plt.xlabel("Size of the system (N)")
plt.ylabel("Heat capacity (J/K)")
plt.savefig('heat_capacity.png')
```



CONCLUSION: The heat capacity is proportional to the number of particles; for large enough systems, this behaviour is linear. The specific heat capacity of a substance is constant.

5. Modify the routine thermostat() so as to implement velocity rescaling. Compute average and fluctuations at fixed T and compare with results obtained with Langevin thermostat

CODE: We need to scale the kinetic energy to obtain the coorrect . To do it, into simplemd.py we modify the code of the thermostat (I saved this new program as simplemd_vr.py):

def thermostat(self,masses,dt,friction,temperature,velocities,engint,random):

Kbar = 3/2 * len(masses) * (K_B=1) * temperature —> Compute the Kinetic Energy

engkin = 0.5 * np.sum(masses * np.sum(velocities^2,axis=1)) —> Sum the three components of the velocity, multiply by the masses, sum over all the particles and then is divided by 2 (this because $K = \frac{3}{2}NK_B$))

velocities *= np.sqrt(Kbar/engkin) —> The velocities are rescaling in a way that the new kinetic energy is equal to Kbar

return velocities,0.0

Note: If you thermostat respect Detailed Balance, this extra column tell you how much Detailed Balance is satisfied but how this thermostat doesn't respect DB, we don't need to use this last lines.

```
c1=np.exp(-friction*dt)
c2=np.sqrt((1.0-c1*c1)*temperature)/np.sqrt(masses)
```

```
    engint+=0.5*np.sum(masses*np.sum(velocities**2,axis=1))---> This are the sum of contributi
    velocities=c1*velocities+c2[:,np.newaxis]*random.Gaussian(shape=velocities.shape)
    engint-=0.5*np.sum(masses*np.sum(velocities**2,axis=1))
    return velocities,engint
```

CODE: And, to save the results only just 10 steps, we change (lines 375-377):

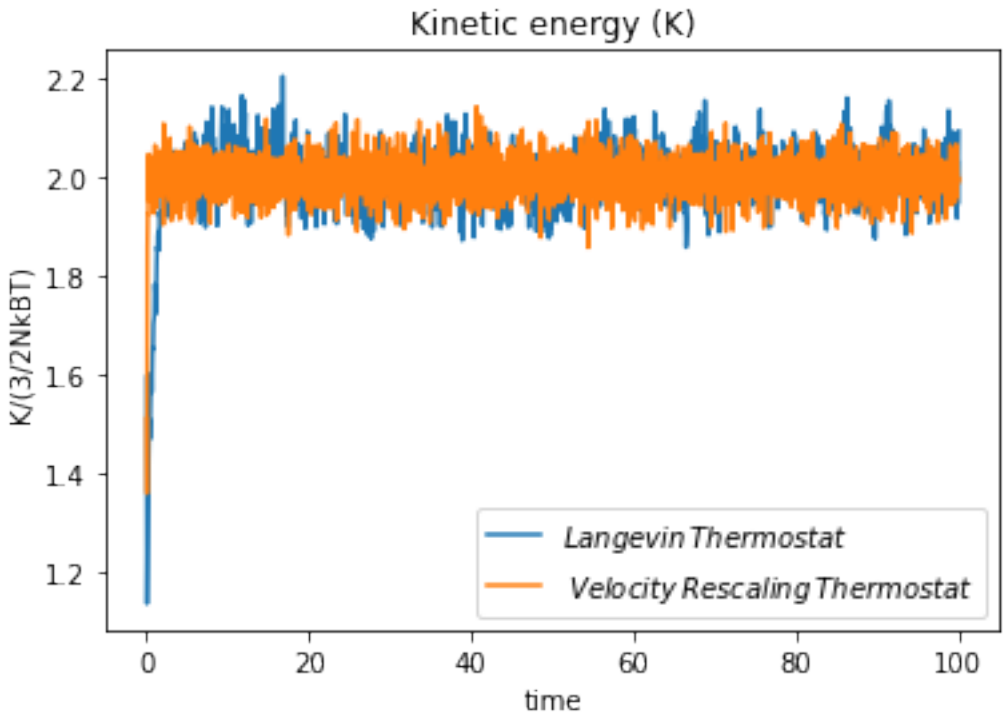
if self.friction>0.0 **and istep%10==0**:

velocities,engint = self.thermostat(

masses,0.5*self.tstep,self.friction,self.temperature,velocities,engint,random)

[269]:
```
plt.plot(ene[1],ene[2], label=r'$Langevin \: Thermostat$')
plt.plot(ene_vr[1],ene_vr[2], label=r'$\: Velocity  \:Rescaling \:Thermostat $')
plt.title("Kinetic energy (K)")
plt.xlabel("time")
plt.ylabel("K/(3/2NkBT)")
plt.legend()
```

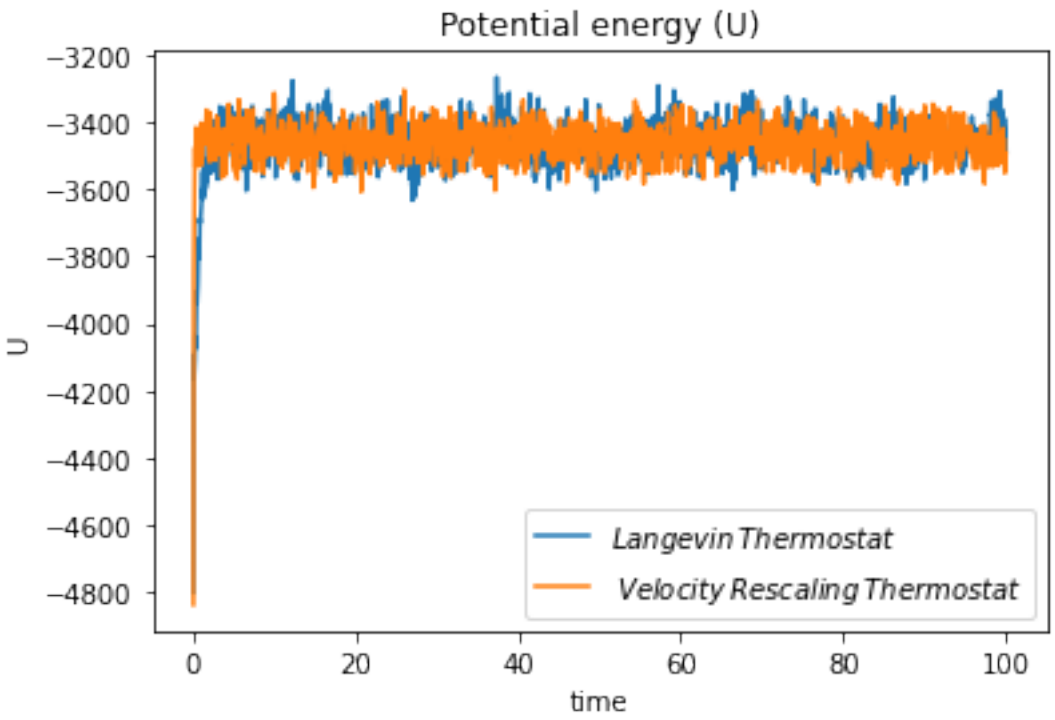
[269]:
```
<matplotlib.legend.Legend at 0x12253ddf0>
```



[268]:
```
plt.plot(ene[1],ene[3], label=r'$Langevin\: Thermostat$')
plt.plot(ene_vr[1],ene_vr[3], label=r'$\: Velocity  \:Rescaling \:Thermostat $')
plt.title("Potential energy (U)")
plt.xlabel("time")
```

```
plt.ylabel("U")
plt.legend()
```

[268]: <matplotlib.legend.Legend at 0x122374460>



OBS: In both cases, the velocity reescaling and the Langevin thermostat, reach same potential energy. Observing the STD(U) it's possible to see that even when both are very simular however, if we compute the statistical errors, the Langevin thermostat presents the smallest ones.

[263]:
```
np.average(ene[3,200:]),np.average(ene_vr[3,200:]) #Average of Potential energy␣
↪(U)
```

[263]: (-3459.262955084444, -3460.8273290438888)

[264]:
```
np.std(ene[3,200:]),np.std(ene_vr[3,200:]) #STD of Potential energy (U)
```

[264]: (51.95132440471794, 46.00017857517875)

OBS: On the other hand, even when both reach the same temperature T=2, the fluctuations in the instantanuos temperature tell us that the

[265]:
```
np.average(ene[2,200:]),np.average(ene_vr[2,200:]) #Average of Kinetic energy␣
↪(K)
```

[265]: (2.0045982355555556, 2.0002418877777774)

[266]:
```
np.std(ene[2,200:]),np.std(ene_vr[2,200:])  #Fluctuations in the instantanuos␣
↪temperature
```

[266]: (0.05112428877511334, 0.0393431667357972)

OBS: Finally, we know that the instantanuos temperature mus be equal to the squart root of 2 over the number of degrees of freedom (= Number of particles multiplied by 3) by the temperature:

[267]:
```
np.sqrt(2/(864*3)) * 2  #Instantanuos temperature
```

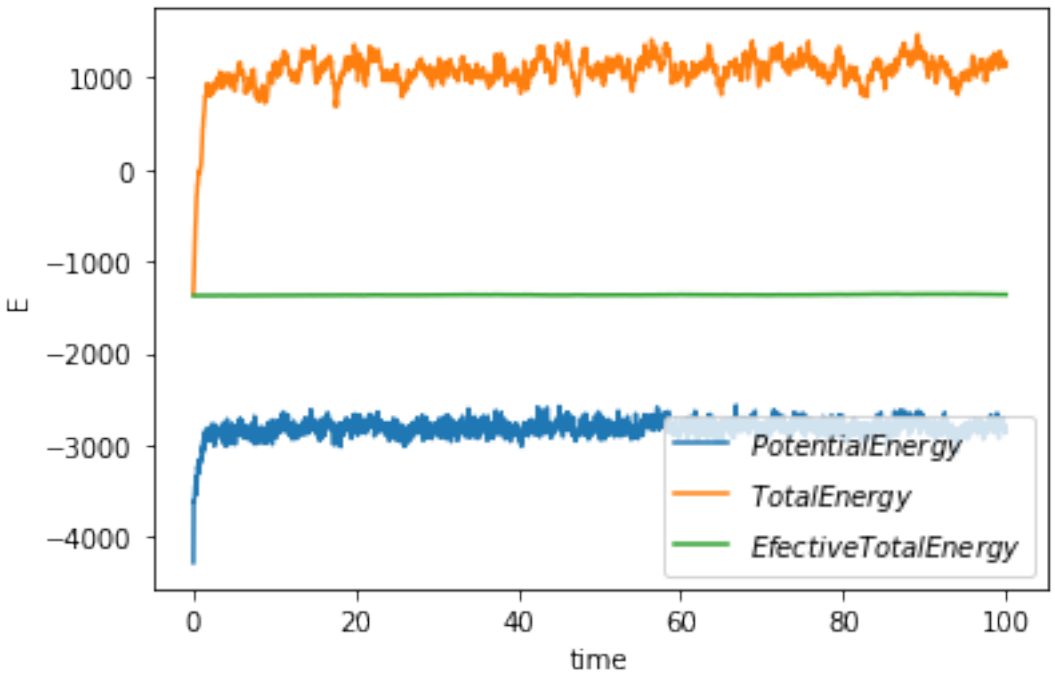
[267]: 0.05555555555555555

OBS: If we run a longer trajectory, the STD of the Langevin thermostat will be more similar to this value while the STD of the Velocity Rescaling thermostat will be smaller.

FinalNote: The time step choosen, for microcanonical ensembles, must the Energy conservation given a confidence interval (the criterio used to choose this interval is that the system doesn't explote). In this case the conservation energy is observed in this plot:

[6]:
```
ene = np.loadtxt("energies-t-3.0.dat").T;

plt.plot(ene[1],ene[3], label=r'$Potential Energy$')
plt.plot(ene[1],ene[4], label=r'$Total Energy$')
plt.plot(ene[1],ene[5], label=r'$Efective Total Energy$')
plt.xlabel("time")
plt.ylabel("E")
plt.legend()
 #plt.xlim((0,10))
plt.show()
```


1000
0
−1000
−2000
−3000
−4000
E
0
20
40
60
80
100
time
PotentialEnergy
TotalEnergy
EfectiveTotalEnergy