EXERCISE 1

Write a molecular dynamics code to integrate the motion of a particle of mass m in one dimension subject to a harmonic spring potential with the following form:

$$U(d) = \frac{1}{2} * k * d^2$$

with $k = 1$.
Use the Verlet algorithm and the following conditions:
• $q(0) = 2$
• $q(-\Delta t) = q(0)$
Consider different possible values of Δt. Which is the maximum value of Δt for which the simulation is stable? How do you expect the result to depend on k?

In [11]:

```
import numpy as np
import math
import matplotlib.pyplot as plt
def force(x,*,k=1.0):                          #function for the harmonic oscillator for
ce
    return -k*x
def energy(x,*,k=1.0):
    return 0.5*k*x**2                          #energy

def run(*,dt=0.01,q=2,qold=2,maxt=10,k=1.0):   #initial conditions are estabilished in t
he method.
    time=[]                                    #methods (like this run) permit to use th
e algorithm for other initial conditions
    traj=[]
    nsteps=int(maxt/dt)                        #not a fixed nsteps, but a fixed max time
    for istep in range(nsteps):                #for loop i=1...nsteps
        qn=2*q-qold+force(q,k=k)*dt**2         #new position with Verlet, q_new
        qold=q                                 #update position
        q=qn
        traj.append(q)                         #add current position to the list  (traje
ctory)
        time.append(dt*(istep+1))              #+1 because I already moved the system fr
om the initial conditions
    return np.array(time),np.array(traj)       #np.arrays are easier to use, I return a
 tuple
```

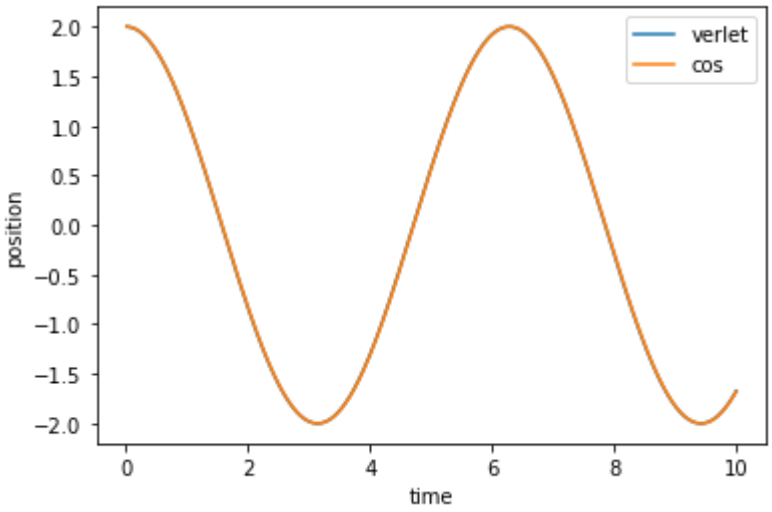
In [12]:

```
time,traj=run()                                #use the method with standard initial con
ditions
```

Let's compare the trajectory obtained with $x = 2cos(t)$ (assuming that the initial velocity is zero, because q and qold are initially equal)
Amplitude is given by the initial condition on q.

In [13]:

```
plt.plot(time,traj,label="verlet")            #plot the trajectory in function of time
plt.plot(time,2*np.cos(time),label="cos")
plt.xlabel('time')
plt.ylabel('position')
plt.legend()
plt.show()
```



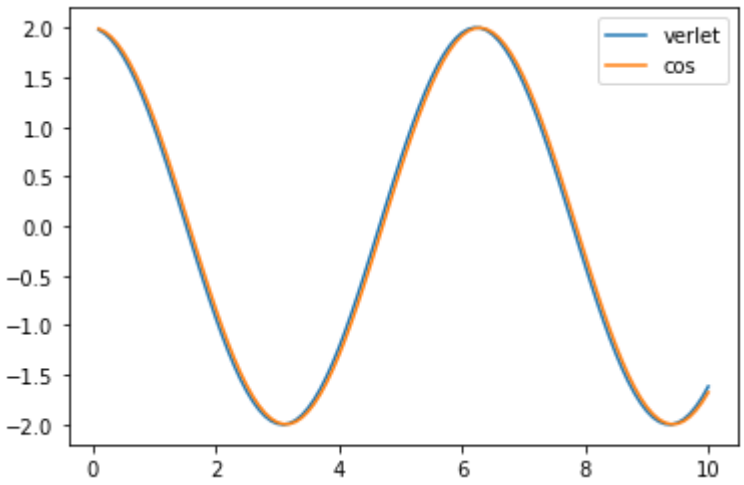
Consider different possible values of Δt. Which is the maximum value of Δt for which the simulation is stable?

In [14]:

```
time,traj=run(dt=0.1) #try to increase dt to see when it exploses.
```

In [15]:

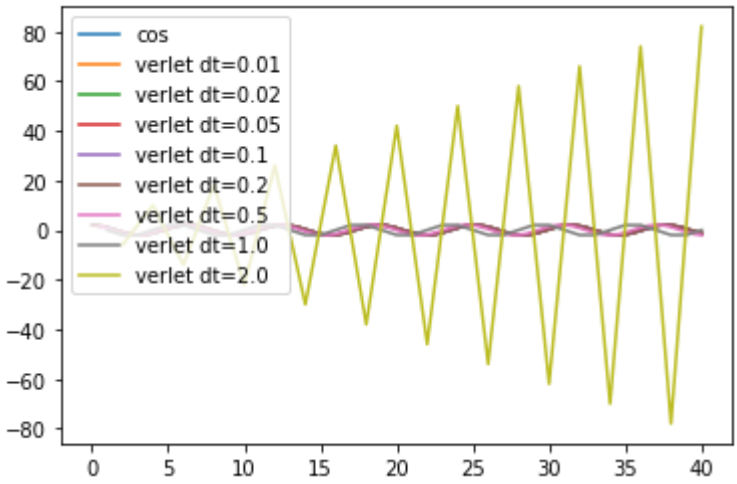
```
plt.plot(time,traj,label="verlet")
plt.plot(time,2*np.cos(time),label="cos")
plt.legend()    #SMALL DIFFERENCE: reason is given by the initial conditions, that are
 not given properly (qold can only be calculated if we know the exact solution, which i
s possible in this case but not in general)
plt.show()
```



Let's try with many different Δt to see at which point the simulation becomes unstable

In [16]:

```
time=np.linspace(0,40,1000)
plt.plot(time,2*np.cos(time),label="cos")        #plot the same cosine as before, analy
tical solution
for dt in (0.01,0.02,0.05,0.1,0.2,0.5,1.0,2.0):
    time,traj=run(dt=dt,maxt=40)                 #all with the same time range, so they
are comparable
    plt.plot(time,traj,label="verlet dt="+str(dt))
plt.legend()
plt.show()
```

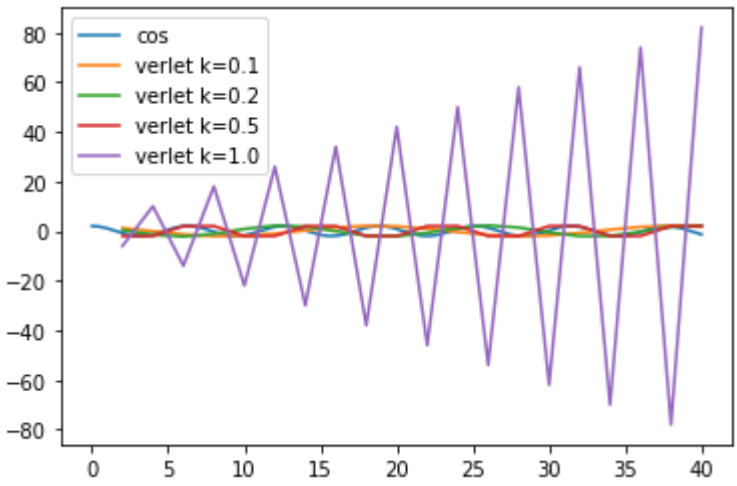


Trajectory is stable until $\Delta t = 1.0$: if $\Delta t = 2.0$ the simulation explodes, between them there is a sort of a threshold for Δt. With $\Delta t = 1.9$ it is still stable but it does not look like a cosine, it is discrete because time is discrete (jumps are given by the way we plot functions). We can solve this issue using a lot of points to plot the cosine, so it becomes "less discrete". Maximum value for the timestep is 2. It can be proven theoretically. Numbers very close to 2 leads to an oscillation which is not correct but still not exploding. Mathematically the limit value is 2, but we should use a timestep which is way smaller to obtain interesting results.

How do you expect the result to depend on k?

In [17]:

```
time=np.linspace(0,40,1000)
plt.plot(time,2*np.cos(time),label="cos")
dt=2.0                                          #dt=2.0 and k=1.0 is previous situation
for k in (0.1,0.2,0.5,1.0):                     #Loop on k
    time,traj=run(k=k,maxt=40,dt=dt)
    plt.plot(time,traj,label="verlet k="+str(k))
plt.legend()
plt.show()
```



$$\frac{T}{\Delta t} = \frac{2\pi\sqrt{\frac{m}{k}}}{\Delta t}$$

The theoretical rule we proved imposes a limit on Δt depending on the period:

$$\Delta t < \frac{T}{\pi}$$

Bigger k leads to smaller period, therefore the threshold for the timestep will be smaller. Changing the mass, period is proportional to the square root of the mass, therefore if we take twice the mass we will obtain a maximum value for Δt which is sqrt(2) times before. With smaller k (fixed $\Delta t = 2.0$) the simulation does not explode.

==========

Particle subject to a Lennard-Jones potential

$$U(d) = 4\epsilon\left(\left(\frac{\sigma}{d}\right)^{12} - \left(\frac{\sigma}{d}\right)^{6}\right)$$

with $\epsilon = 1$ and $d = 1$. Find a value of Δt such that the simulation is stable.

Can you relate it with the value obtained for the harmonic potential?

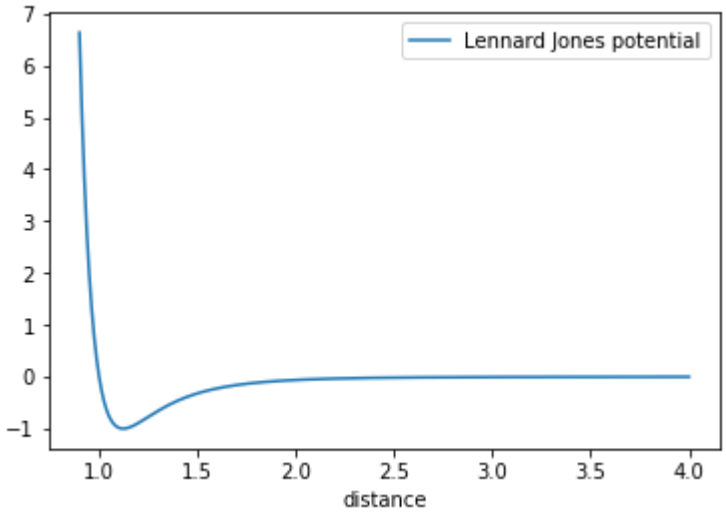
In [18]:

```
def force(x):
    return -4.0*(-12/x**13+6/x**7)     #minus the derivative of the potential over x
def energy(x):
    return 4.0*(1/x**12-1/x**6)        #Lennard Jones potential
```

Lennard Jones potential is composed by repulsive and attractive interactions.

In [19]:

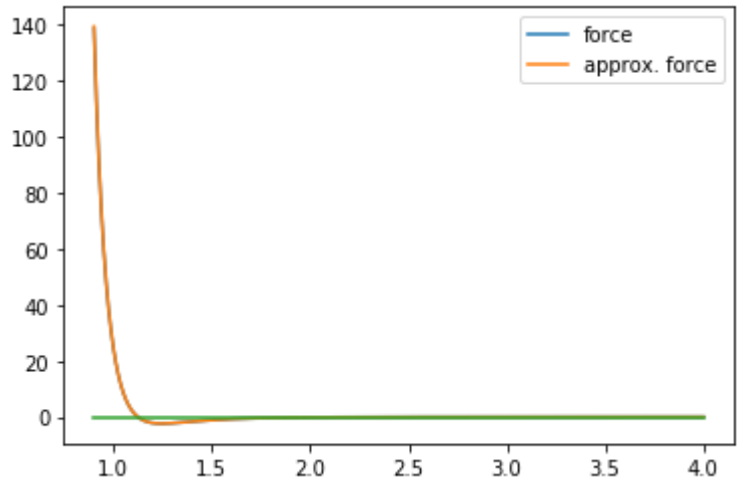
```
x=np.linspace(0.9,4,500)
plt.plot(x,energy(x),label='Lennard Jones potential')
plt.xlabel('distance')
plt.legend()
plt.show()
```



In [21]:

```
x=np.linspace(0.9,4,500)
plt.plot(x,force(x),label='force')                                    #plot of the
 force
plt.plot(x,-(energy(x+0.01)-energy(x-0.01))/0.02,label='approx. force')   #derivative d
one "numerically"
plt.plot(x,0*x)                                                       #oriz. line,
 x -axis
plt.legend()
plt.show()

#this was done to check if the derivative was correct.
#approx. force is given by the difference of energy divided by delta t
# approx. doesn't work well for low distances because the curvature is too large.
```



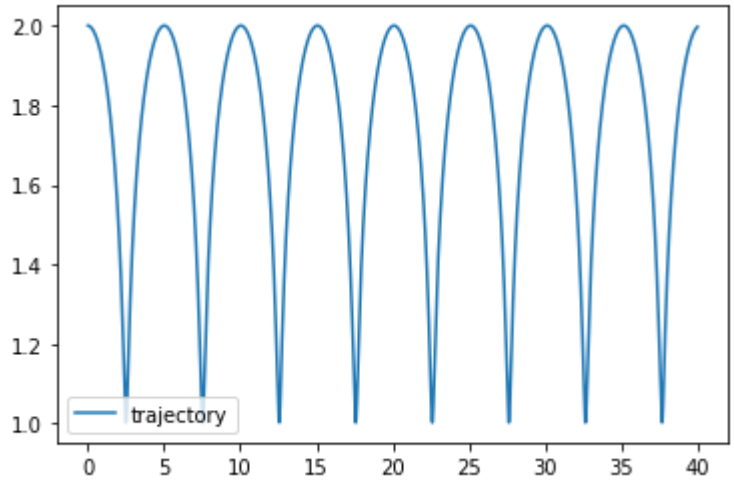
In [22]:

```
def run(*,dt=0.01,q=2,qold=2,maxt=10): #same as before
    time=[]
    traj=[]
    nsteps=int(maxt/dt)
    for istep in range(nsteps):
        qn=2*q-qold+force(q)*dt**2
        qold=q
        q=qn
        traj.append(q)
        time.append(dt*(istep+1))
    return np.array(time),np.array(traj)
```

We can notice that initially the time derivative is zero.
This potential energy is asymmetric, while before it was perfectly symmetric. With this timestep the simulation is stable, as we can see from the periodicity we obtain.

In [13]:

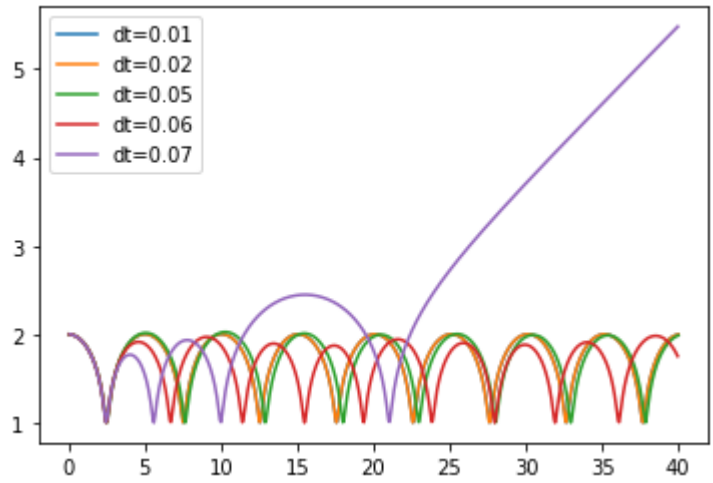
```
time,traj=run(maxt=40)
plt.plot(time,traj,label='trajectory')
plt.legend()
plt.show()
```



Now we can try to change Δt and see when it is exploiding. We do not have an analytical solution to compare with, but we can notice that for $\Delta t = 0.7$ it explodes.
In this case, system bounced back some times and after that it explodes. One consideration we could do is the following: at our initial condition $x = 2$ it is sufficient to have a small violation of the energy conservation for the system to escape the well.
Otherwise, it fluctuates back and forth into a bound state.

In [23]:

```
for dt in (0.01,0.02,0.05,0.06,0.07):
    time,traj=run(maxt=40,dt=dt)
    plt.plot(time,traj,label="dt="+str(dt))
plt.legend()
plt.show()
```

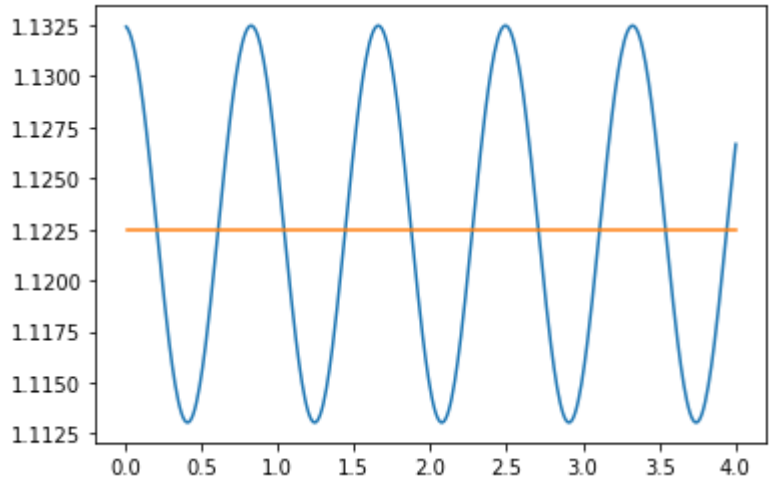


For harm. oscillator, we have an analytical solution and so we are sure that if the simulation is stable it will be so also for an infinite long simulation. This is not true for Lennard Jones, where we do not have an exact threshold, therefore the explosion of a trajectory depends on the length of the simulation. There could be a very slow drift, so you are not sure you will notice it. A good decision is always a very short timestep.

Can you relate it with the value obtained for the harmonic potential?

In [24]:

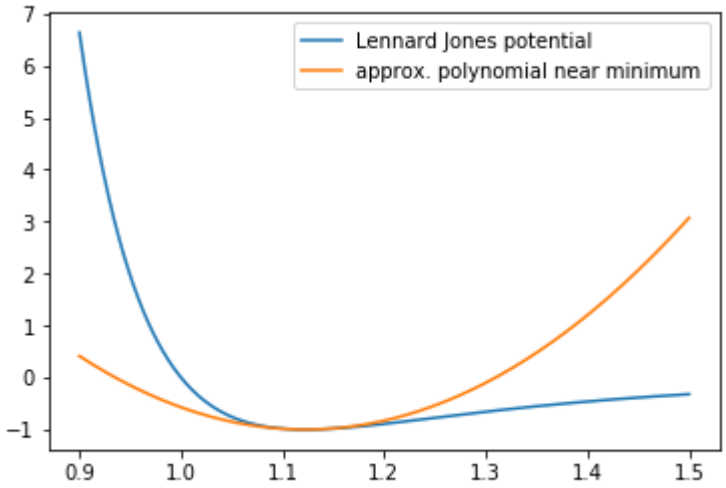
```
time,traj=run(q=2**(1/6)+0.01,qold=2**(1/6)+0.01,maxt=4)
plt.plot(time,traj)
plt.plot(time,0*time+2**(1/6)) #array of zeros
plt.show()
```



To relate the value of k to ϵ and d, we assume our particle to be very close to the minimum of Lennard Jones potential, where we can approximate it as an harmonic oscillator. I should initialize the particle in a position closer to the minimum, which can be found analytically, $q_{min} = 2^{1/6}$. Therefore I initialize
$q = 2^{1/6} + 0.01 = q_{old}$
In this regime, the system will behave like an harmonical oscillator.

In [17]:

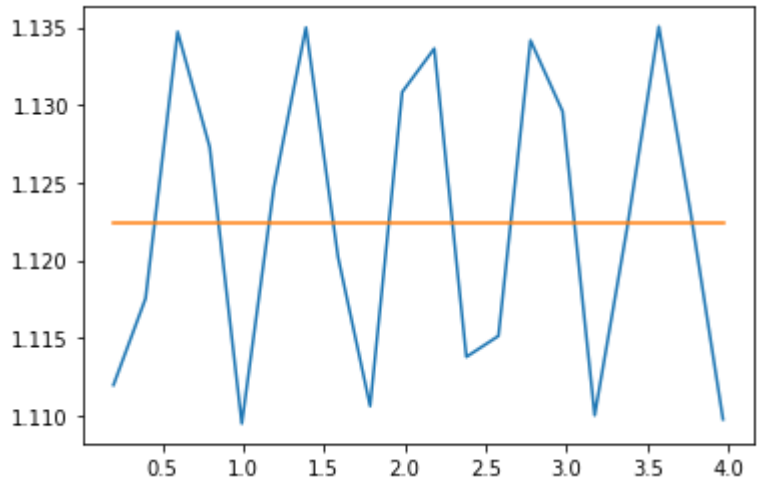
```
x=np.linspace(0.9,1.5,500)
plt.plot(x,energy(x),label='Lennard Jones potential')
x0=2**(1/6)
plt.plot(x,energy(x0)+0.5*57.14643787085514*(x-x0)**2,label='approx. polynomial near mi
nimum')
plt.legend()
plt.show()
```



Let's calculate the second derivative to find the best parabolic function which approximates the Lennard Jones potential in this regime.
Let's calculate it in $x = q_{min}$: we obtain this derivative equal to 57.14643787085514
I can divide the timestep by the sqrt of this value to obtain the timestep we would have with an harmonic oscillator in this case.

In [25]:

```
time,traj=run(dt=1.5/np.sqrt(57.14643787085514),q=2**(1/6)+0.01,qold=2**(1/6)+0.01,maxt
=4)
plt.plot(time,traj)
plt.plot(time,0*time+2**(1/6))
plt.show()
```



Therefore, the system will explode with a timestep equal to $dt = 2/57.14643787085514$