Leap-flog Algolism

Aoi Watanabe

March 19, 2020

1 Leap-frog Algorithm

We want to solve the following equation.

$$\begin{aligned}\frac{d^2}{dt^2}x\left(t\right) &= a\left(x\right)\\ &= -\frac{d}{dx}\Phi\left(x\right)\end{aligned}$$

Here, a is the acceleration and $\Phi\left(x\right)$ is a potential of a point mass.

This equation represents hamiltonian system with the energy

$$E = \frac{1}{2}v^2 + \Phi\left(x\right)$$

which is conserved because of

$$\begin{aligned}\frac{dE}{dt} &= v\frac{dv}{dt} + \frac{d\Phi}{dx}\frac{dx}{dt}\\ &= v\left(\frac{d^2}{dt^2}x + \frac{d\Phi}{dx}\right)\\ &= 0\end{aligned}$$

Here, v stands for the velocity and $v = \frac{dx}{dt}$.

We use a leap-frog algorithm. From the energy conservation equation, we get

$$\frac{d^2}{dt^2}x + \frac{d\Phi}{dx} = 0$$

Therefore,

$$\begin{aligned}v_{n+\frac{1}{2}} &= v_n + a_n\frac{dt}{2}\\ x_{n+\frac{1}{2}} &= x_n + v_{n+\frac{1}{2}}\frac{dt}{2}\end{aligned}$$

and then the other way round

$$
\begin{aligned}
x_{n+1} &= x_{n+\frac{1}{2}} + v_{n+\frac{1}{2}}\frac{dt}{2} \\
v_{n+1} &= v_{n+\frac{1}{2}} + a_{n+1}\frac{dt}{2}
\end{aligned}
$$

From the first half-step

$$x_{n+\frac{1}{2}} = x_n + v_n\frac{dt}{2} + a_n\frac{dt^2}{4}$$

and therefore, from the second half-step

$$
\begin{aligned}
x_{n+1} &= x_n + v_n\frac{dt}{2} + a_n\frac{dt^2}{4} + v_{n+\frac{1}{2}}\frac{dt}{2} \\
&= x_n + v_n\frac{dt}{2} + a_n\frac{dt^2}{4} + v_n\frac{dt}{2} + a_n\frac{dt^2}{4} \\
&= x_n + v_n dt + \frac{1}{2}a_n dt^2
\end{aligned}
$$

and

$$
\begin{aligned}
v_{n+1} &= v_{n+\frac{1}{2}} + a_{n+1}\frac{dt}{2} \\
&= v_n + \frac{a_n + a_{n+1}}{2}dt
\end{aligned}
$$

Hence, we get the Verlet algorithm

$$
\begin{aligned}
x_{n+1} &= x_n + v_n dt + \frac{1}{2}a_n dt^2 \\
v_{n+1} &= v_n + \frac{a_n + a_{n+1}}{2}dt
\end{aligned}
$$

1.0.1 Exercise 1

Harmonic Oscillator Show that for the harmonic oscillator $a_n = -x_n$, the modified energy

$$\hat{E}_n = \frac{1}{2}v_n^2 + \frac{1}{2}x_n^2\left(1 - \frac{dt^2}{4}\right)$$

is conserved, that is,

$$\hat{E}_n = \hat{E}_{n+1}$$

Solution

$$\hat{E}_{n+1} = \frac{1}{2}v_{n+1}^2 + \frac{1}{2}x_{n+1}^2\left(1 - \frac{dt^2}{4}\right)$$

and by using the following to the above,

$$\begin{aligned} x_{n+1} &= x_n + v_n dt + \frac{1}{2} a_n dt^2 \\ &= v_n dt + x_n \left(1 - \frac{dt^2}{2}\right) \end{aligned}$$

$$\begin{aligned} v_{n+1} &= v_n + \frac{a_n + a_{n+1}}{2} dt \\ &= v_n \left(1 - \frac{dt^2}{2}\right) - x_n \left(1 - \frac{dt^2}{4}\right) dt \end{aligned}$$

We get

$$\begin{aligned} \hat{E}_{n+1} &= \frac{1}{2} v_{n+1}^2 + \frac{1}{2} x_{n+1}^2 \left(1 - \frac{dt^2}{4}\right) \\ &= \frac{1}{2} v_n^2 \left(1 - \frac{dt^2}{2}\right)^2 + \frac{1}{2} x_n^2 \left(1 - \frac{dt^2}{4}\right)^2 dt^2 - x_n v_n \left(1 - \frac{dt^2}{2}\right)\left(1 - \frac{dt^2}{4}\right) dt \\ &+ \frac{1}{2} v_n^2 \left(1 - \frac{dt^2}{4}\right)^2 dt^2 + \frac{1}{2} x_n^2 \left(1 - \frac{dt^2}{2}\right)^2 \left(1 - \frac{dt^2}{4}\right) + x_n v_n \left(1 - \frac{dt^2}{2}\right)\left(1 - \frac{dt^2}{4}\right) dt \\ &= \frac{1}{2} v_n^2 + \frac{1}{2} x_n^2 \left(1 - \frac{dt^2}{4}\right) \end{aligned}$$

1.0.2 Exercise 2

Time Reversal Show that the algorithm is time-reversal invariant, that is, that by interchanging $dt \to -dt$ and $n \leftrightarrow n+1$ we get the same algorithm, backwatds in time:

$$\begin{aligned} x_n &= x_{n+1} - v_{n+1} dt + \frac{1}{2} a_{n+1} dt^2 \\ v_n &= v_{n+1} - \frac{a_{n+1} + a_n}{2} dt \end{aligned}$$

and that it therefore has the accuracy in 4th order $\mathcal{O}\left(dt^4\right)$ to x.

Solution For the velocity, it is obviously

$$v_{n+1} = v_n + \frac{a_n + a_{n+1}}{2} dt$$

Therefore,

$$\begin{aligned} x_{n+1} &= x_n + v_{n+1} dt - \frac{1}{2} a_{n+1} dt^2 \\ &= x_n + v_n dt + \frac{a_n + a_{n+1}}{2} dt^2 - \frac{1}{2} a_{n+1} dt^2 \\ &= x_n + v_n dt + \frac{1}{2} a_n dt^2 \end{aligned}$$

1.0.3 Exercise 3

Verlet without velocity as 4th order algorithm Use the Taylor expansions

$$x(t+dt) = x(t) + v(t)\,dt + \frac{1}{2!}a(t)\,dt^2 + \frac{1}{3!}b(t)\,dt^3 + \mathcal{O}\left(dt^4\right)$$
$$x(t-dt) = x(t) - v(t)\,dt + \frac{1}{2!}a(t)\,dt^2 - \frac{1}{3!}b(t)\,dt^3 + \mathcal{O}\left(dt^4\right)$$

and show that there is an algorithm with error in 4th order only connecting x_{n-1}, x_n and x_{n+1}. Compare with the algorithm above.

Solution By subtracting those above,

$$x(t+dt) - x(t-dt) = 2x(t) + a(t)\,dt^2 + \mathcal{O}\left(dt^4\right)$$
$$x_{n+1} = 2x_n + x_{n-1} + a_n dt^2 + \mathcal{O}\left(dt^4\right)$$

Since the algorithm from exercise 3 is time-reversal symmetric, the third-order error term in $x(t)$ must be absent.

1.0.4 Exercise 4

The nonliner oscillator Solve

$$\frac{d^2x}{dt^2} = -x^p$$

for odd $p = 11$, or

$$\frac{d^2x}{dt^2} = -\mathrm{sign}(x)\,|x|^p$$

for any $p > 0$ and observe the oscillations, and how good the energy is conserved.

Initial conditions: $x_0 = 1$ and $v_0 = 0$.

Codes We compare verlet algorithm and 2nd order Runge-Kutta algorithm below.

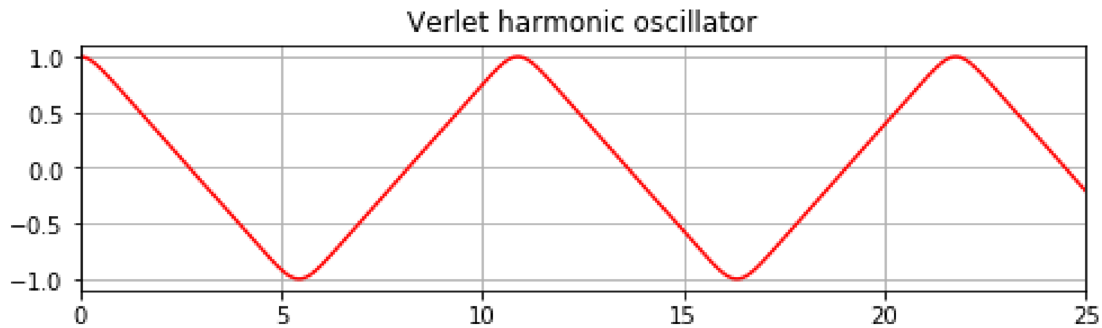
[4]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.ticker as ptick
import numpy as np

dt  = 0.01
T = 25

p = 11
x_0 = 1.0
v_0 = 0.0

def energy(x, v):
    global dt
```

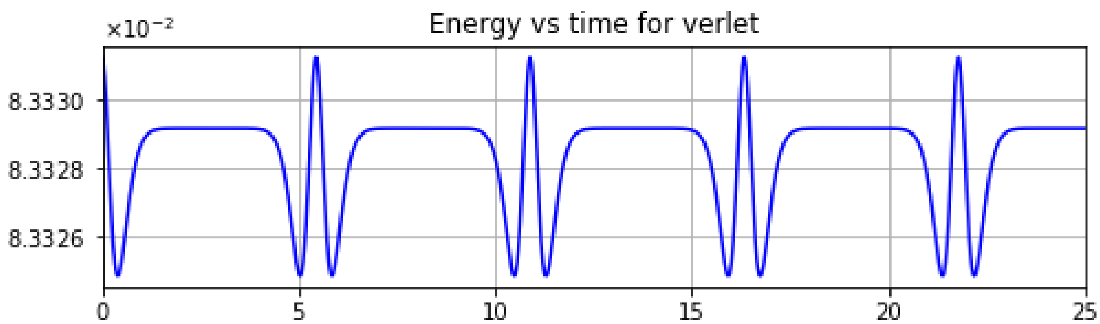
```
    return 0.5 * v**2 + np.abs(x)**(p+1) *( 1- dt**2*0.25) / (1+p)

def runge_kutta():
    global dt, T, p, x_0, v_0

    x = x_0
    v = v_0
    E = energy(x,v)

    x_list = [x]
    v_list = [v]
    t_list = [0]
    E_list = [E]

    for t in np.arange(dt, T, dt):
        x1 = x + 0.5*dt * v
        v1 = v - 0.5*dt * np.sign(x)*np.abs(x)**p

        x2 = x + v1*dt
        v2 = v - dt * np.sign(x1)*np.abs(x1)**p

        x_list.append(x2)
        v_list.append(v2)
        t_list.append(t)
        E_list.append(energy(x2, v2))

        x = x2
        v = v2


    fig1, ax1 = plt.subplots(figsize=(8, 2))
    ax1.yaxis.set_major_formatter(ptick.ScalarFormatter(useMathText=True))
    ax1.ticklabel_format(style='sci',axis='y',scilimits=(0,0))

    plt.figure(1)
    plt.title("2nd order Runge-Kutta harmonic oscillator")
    plt.plot(t_list, x_list, label="x", color="red")
    plt.xlim([0,T])
    plt.grid()


    fig2, ax2 = plt.subplots(figsize=(8, 2))
    ax2.yaxis.set_major_formatter(ptick.ScalarFormatter(useMathText=True))
    ax2.ticklabel_format(style='sci',axis='y',scilimits=(0,0))

    plt.figure(2)
    plt.title("Energy vs time for Runge-Kutta")
```

```
    plt.plot(t_list, E_list, label="E", color="blue")
    plt.xlim([0,T])
    plt.grid()


def leapflog():
        global dt, T, p, x_0, v_0

        x = x_0
        v = v_0
        E = energy(x,v)

        x_list = [x]
        v_list = [v]
        t_list = [0]
        E_list = [E]

        for t in np.arange(dt, T, dt):
            x_next = x + v*dt - 0.5* np.sign(x) * np.abs(x)**p * dt**2
            v_next = v - 0.5 * (np.sign(x) * np.abs(x)**p + np.sign(x_next) *␣
 ↪np.abs(x_next)**p) * dt

            x_list.append(x_next)
            v_list.append(v_next)
            t_list.append(t)
            E_list.append(energy(x_next, v_next))

            x = x_next
            v = v_next


        fig1, ax1 = plt.subplots(figsize=(8, 2))
        ax1.yaxis.set_major_formatter(ptick.ScalarFormatter(useMathText=True))
        ax1.ticklabel_format(style='sci',axis='y',scilimits=(0,0))

        plt.figure(1)
        plt.title("Verlet harmonic oscillator")
        plt.plot(t_list, x_list, label="x", color="red")
        plt.xlim([0,T])
        plt.grid()


        fig2, ax2 = plt.subplots(figsize=(8, 2))
        ax2.yaxis.set_major_formatter(ptick.ScalarFormatter(useMathText=True))
        ax2.yaxis.get_major_formatter().set_useOffset(False)
        ax2.ticklabel_format(style='sci',axis='y',scilimits=(0,0))
```

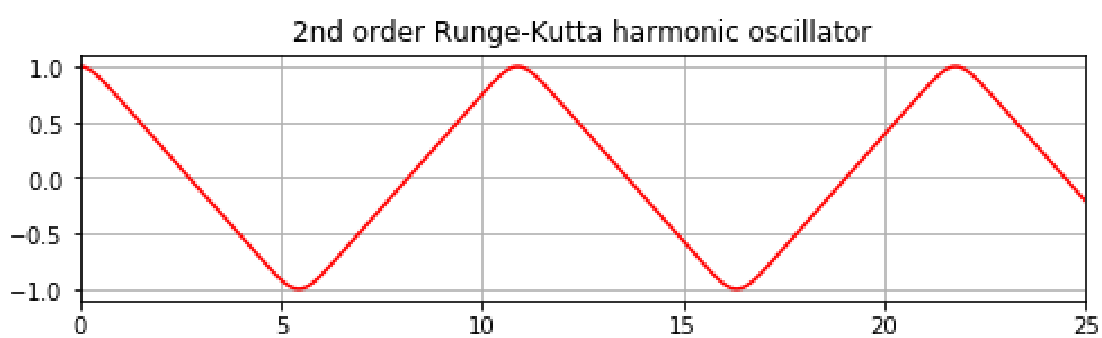
```
        plt.figure(2)
        plt.title("Energy vs time for verlet")
        plt.plot(t_list, E_list, label="E", color="blue")
        plt.xlim([0,T])
        plt.grid()
```

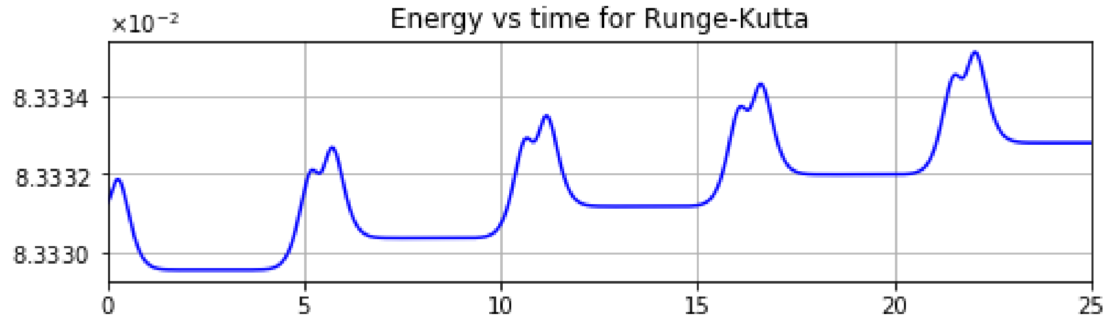
[5]:
```
leapflog()
plt.show()
```





[6]:
```
runge_kutta()
plt.show()
```





Results In the case of the 2nd order Runge-Kutta algorithm, the energy is increasing which is not appropriate behavior of the harmonic oscillator. On the other hand, the verlet algorithm keeps its integrity.