Harmonic Oscillator

Aoi Watanabe

March 14, 2020

1 Euler Method and Verlet Integration

1.0.1 Harmonic Oscillator

Hermonic Oscillator equation is simply written by the following equation.

$$\ddot{x} + x = 0$$

1.1 Euler Method

To solve this equation, let's try to use the euler mehod.

The euler method of first order differential equation becomes follows.

$$x_{n+1} = x_n + \Delta t f\left(x\right)$$

To bring this to the second order differential equation, let's take x as a vector. And the harmonic oscillator equation is given by followings.

$$\dot{X} = AX$$

Here,

$$X = \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$$
$$A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

So, let's simulate this in following code. here the initilal conditions are

$$x\left(0\right) = 1$$
$$\dot{x}\left(0\right) = 0$$

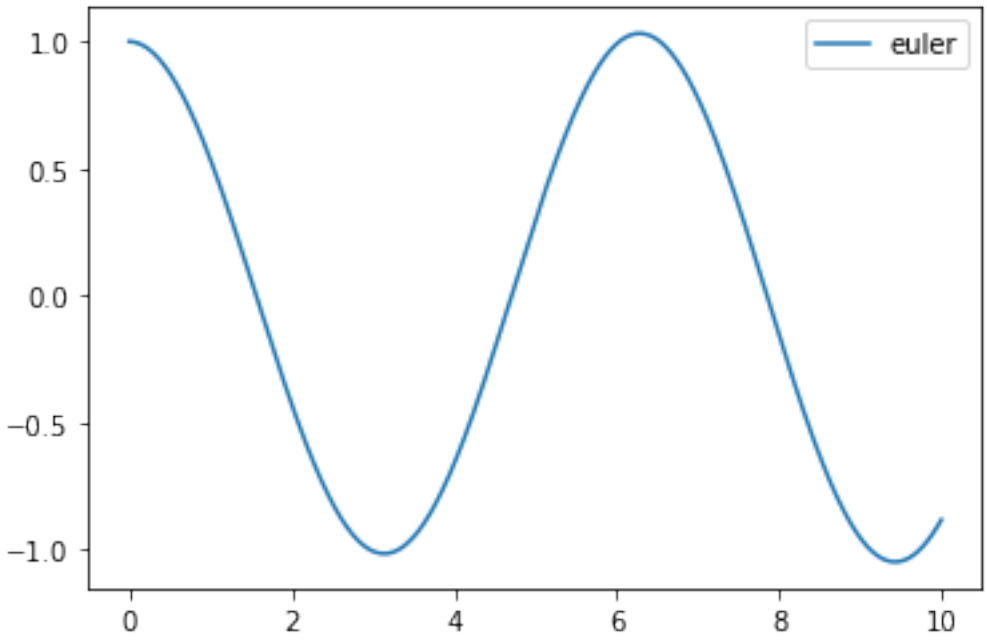
[11]:
```
import numpy as np
import matplotlib.pyplot as plt

def euler(dt, T):
    A = [[0,1], [-1,0]]
```

```
    X = [[1], [0]]

    X_list = []
    t_list = []
    for n in range(T):
        tmp = X + dt * np.dot(A,X)
        X = tmp

        # appending value to the list for data plot
        X_list.append(tmp[0][0])
        t_list.append(dt*n)



    plt.plot(t_list, X_list, label="euler")
```

The answer shapes cosine curve if the dt is sufficiantly small compare to the T.

[12]:
```
euler(0.01, 1000)
plt.legend()
```

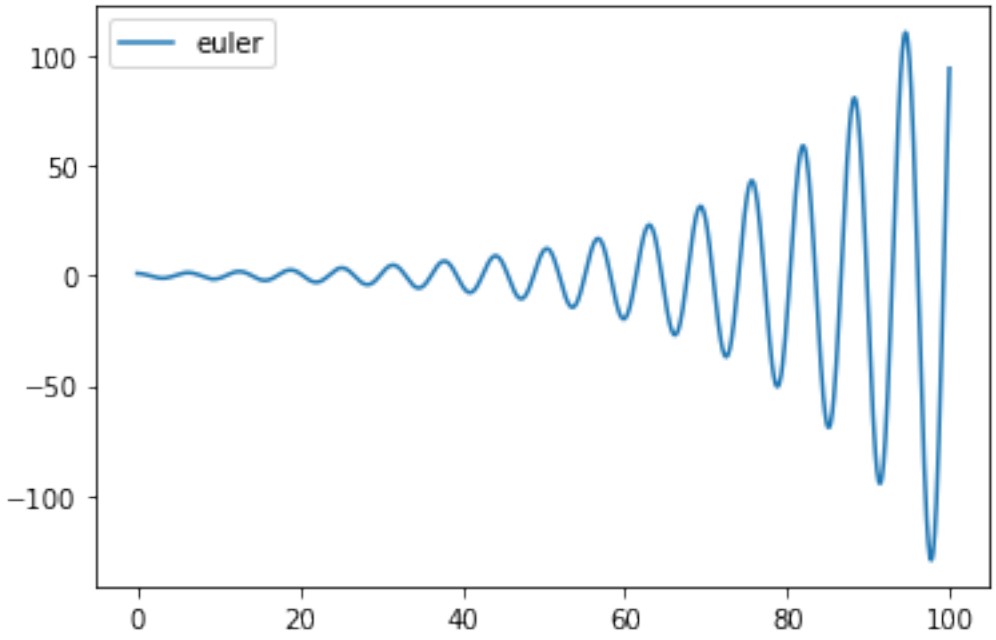
[12]: <matplotlib.legend.Legend at 0x11d946410>



However, when the observation time T was extended, the numerical answer starts exploding.

[13]:
```
euler(0.1, 1000)
plt.legend()
```

[13]:
```
<matplotlib.legend.Legend at 0x11d9ff190>
```



The answer does not fit anymore to the analytical solution. This is a big issue when we try to calculate some practical cases, for instance, crystal structure or polymer materials which requires long observation time for the reaction. So, Now let's take a look at where the phenomenon is coming from.

In the euler method, our model was followings.

$$\begin{aligned} X_{n+1} &= X_n + \Delta t A X_n \\ &= (I + \Delta t A)\, X_n \\ &= (I + \Delta t A)^n\, X_0 \end{aligned}$$

From the above expression, it is clear that the eigenvalues of the matrix $I + \Delta t A$ gives us the divergence causes.

The eigenvalues λ of $I + \Delta t A$ is calculated as follows,

$$\begin{vmatrix} 1-\lambda & \Delta t \\ -\Delta t & 1-\lambda \end{vmatrix} = \lambda^2 - 2\lambda + 1 + \Delta t^2$$

And eigenvalues are given by the complex number

$$\lambda = 1 \pm i\Delta t$$

So, now we can rewrite our models in different form by diagonalization as follows,

$$\begin{aligned} X_{n+1} &= P^{-1}D^nPX_n \\ &= P^{-1}\begin{pmatrix}(1+i\Delta t)^n & 0 \\ 0 & (1-i\Delta t)^n\end{pmatrix}PX_n \end{aligned}$$

This stand for that the modulus of the eigenvalues should be close enough to 1 so that to get the observation time independent oscillation numerical result. In here, the value of n is convertible to the observation time T. Now our modulus of the eigenvalues is

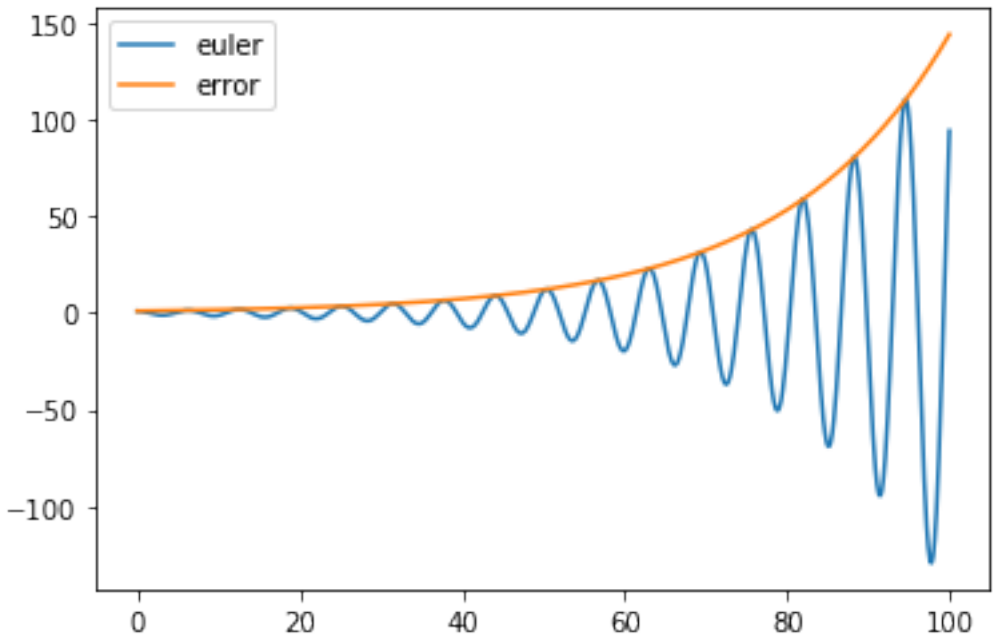
$$|1 + \pm i\Delta t| = \sqrt{1 + \Delta t^2}$$

Let's see the euler method plot with these error estimations.

[6]:
```
import numpy as np
import matplotlib.pyplot as plt

def euler_error(dt, T):
    A = [[0,1], [-1,0]]
    X = [[1], [0]]

    X_list = []
    t_list = []
    e_list = []
    for n in range(T):
        tmp = X + dt * np.dot(A,X)
        X = tmp

        # appending value to the list for data plot
        X_list.append(tmp[0][0])
        t_list.append(dt*n)

        # The error estimation
        e_list.append((1+dt**2)**(n/2))



    plt.plot(t_list, X_list, label="euler")
    plt.plot(t_list, e_list, label="error")
```

[7]:
```
euler_error(0.1, 1000)
plt.legend()
```

[7]:
```
<matplotlib.legend.Legend at 0x11950c8d0>
```



1.2 Verlet Integration (Leapfrog Algorithm)

The next thing we want to try is to reduce these effects of error. And the solution is called "verlet integration".

In the simple euler method that we solved before, the relation between distance x and velocity $\dot{x} = v$ was following.

$$v_{n+1} = v_n - dt \cdot x_n$$
$$x_{n+1} = x_n + dt \cdot v_n$$

Now, we modify to the calculation as follows in order to get next step of value with leapfrog algorithm.

$$v_{n+1} = v_n - dt \cdot x_n$$
$$x_{n+1} = x_n + dt \cdot v_{n+1}$$

Then, our model could be written as followings by simplifying above equations.

$$x_{n+1} = \left(1 - \Delta t^2\right) \cdot x_n + \Delta t \cdot v_n$$
$$v_{n+1} = -\Delta t \cdot x_n + v_n$$

And in matrix form,

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} 1 - \Delta t^2 & \Delta t \\ -\Delta t & 1 \end{pmatrix} \begin{pmatrix} x_n \\ v_n \end{pmatrix}$$

Next thing to do is that to evaluate the accuracy of the numerical method in time domain. As the same way before, calcurate eigonvalues as follows.

$$\lambda = \frac{2 - \Delta t^2 \pm i\Delta t\sqrt{4 - \Delta t^2}}{2}$$

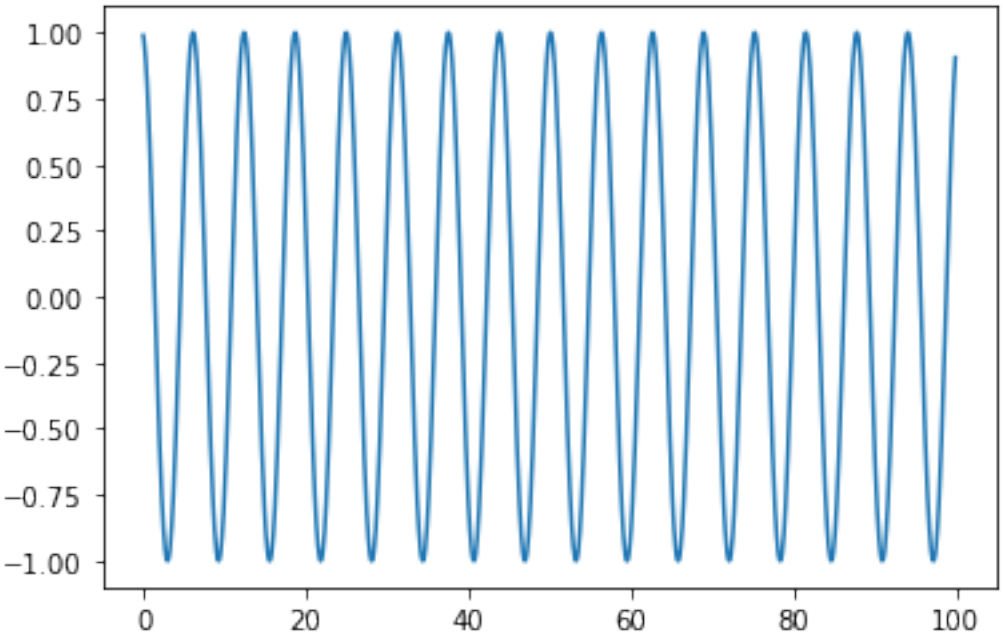
and it's modulus $|\lambda|$ is,

$$\begin{aligned} |\lambda| &= \frac{4 - 4\Delta t^2 + \Delta t^4 - \Delta t^4 + 4\Delta t^2}{4} \\ &= 1 \end{aligned}$$

Now we got $|\lambda| = 1$ one, which means that the numerical simulated oscillation is very stable compare to the previous methods. Lets's see it in python code.

[8]:
```
def verlet(dt, T):
    A = [[1.0-dt**2,dt], [-dt, 1.0]]
    X = [[1], [0]]

    X_list = []
    t_list = []
    for n in range(T):
        tmp =  np.dot(A,X)
        X = tmp

        # appending value to the list for data plot
        X_list.append(tmp[0][0])
        t_list.append(dt*n)


    plt.plot(t_list, X_list, label="verlet")
```

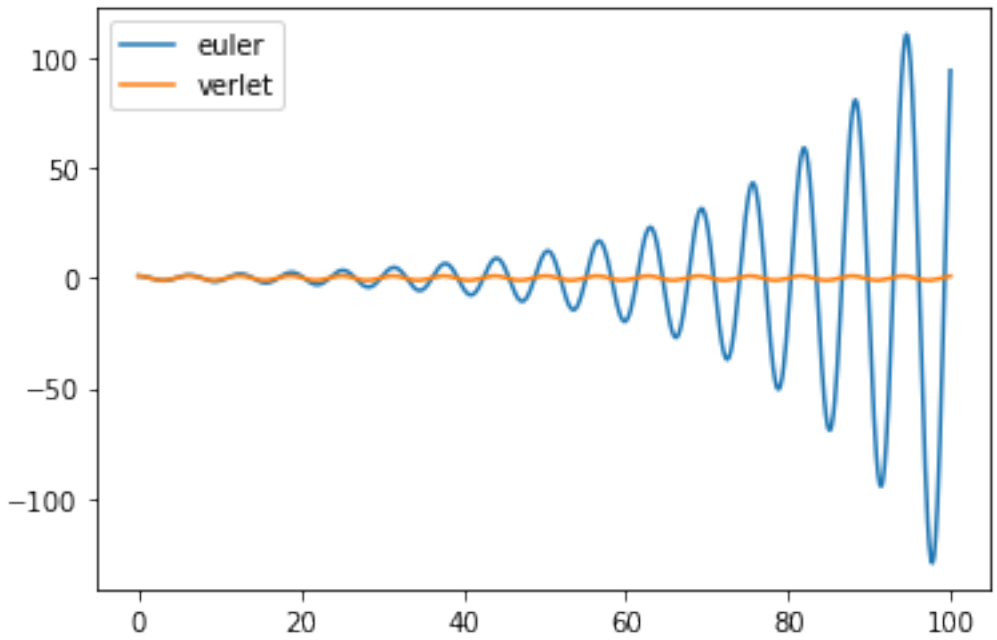
[9]:
```
verlet(0.1, 1000)
```



It seems very stable. Let's put these plot(euler method and verlet method) together.

[10]:
```
euler(0.1, 1000)
verlet(0.1, 1000)
plt.legend()
```

[10]:
```
<matplotlib.legend.Legend at 0x11d8f8b90>
```



Summary

- In the euler method, error of the calcuration is not only depends on step time dt, but also important to take account observation time T.
- Verlet method is valid for such case, and since in hamiltonian of the harmonic oscillator is expressed by the kinetic energy $\frac{1}{2}v^2$ and potential energy $\frac{1}{2}x^2$, to apply leapfrog algolism to these vareable is effective to get relatively accurate answer.