Solving a non-linear system of differential equations

Hanie Hatami (99100614) and Alireza Same (99100709)

1 Prerequisites

Hooke's spring: $\vec{\mathbf{F}} = -k\Delta x(\Delta\hat{\mathbf{x}})$. The force is given by latter where $\Delta\hat{\mathbf{x}}$ shows the direction of force. Hooke's spring is massless and obeys linearity to a good measurement.

2 Creating the model

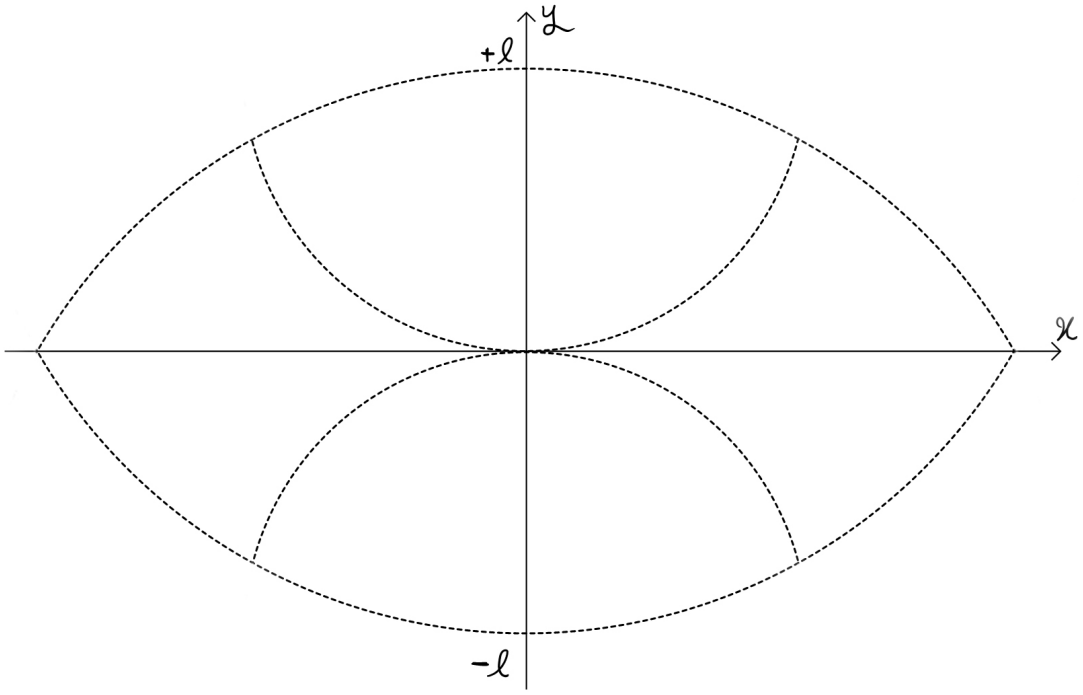

Figure 1: Regions

If we choose a point (x, y) within the boundary, we have:

$\vec{r}_1 \rightarrow$ is the vector connecting $(0, +\ell)$ to $(x, y) \Rightarrow \vec{r}_1 = (x, y - \ell)$

$\vec{r}_2 \rightarrow$ is the vector connecting $(0, -\ell)$ to $(x, y) \Rightarrow \vec{r}_2 = (x, y + \ell)$

ℓ'_1 , ℓ'_2 are lengths of springs 1 and 2 respectively:

$$\ell'_1 = \|\vec{r}_1\| = \sqrt{x^2 + (y - \ell)^2}$$
$$\ell'_2 = \|\vec{r}_2\| = \sqrt{x^2 + (y + \ell)^2}$$

Both strings have identical k and ℓ_0 , $\vec{\mathbf{F}} = -k\Delta\ell(\hat{z})$

$$\vec{\mathbf{F}}_1 = -k(\ell_1' - \ell_0)\hat{z}_1 \Rightarrow \ \vec{\mathbf{F}}_1 = k\left[\ell\frac{(x, y-\ell)}{\sqrt{x^2+(y-\ell)^2}} - (x, y-\ell)\right]$$

$$\vec{\mathbf{F}}_2 = -k(\ell_2' - \ell_0)\hat{z}_2 \Rightarrow \ \vec{\mathbf{F}}_2 = k\left[\ell\frac{(x, y+\ell)}{\sqrt{x^2+(y+\ell)^2}} - (x, y+\ell)\right]$$

Newton's law reads: $\vec{\mathbf{F}}_{net} = m\vec{\mathbf{a}} \Rightarrow m\vec{\mathbf{a}} = \vec{\mathbf{F}}_1 + \vec{\mathbf{F}}_2$

$$m\ddot{x} = k\left(\frac{x}{\sqrt{\left(\frac{x}{\ell}\right)^2+\left(\frac{y}{\ell}-1\right)^2}} - 2x + \frac{x}{\sqrt{\left(\frac{x}{\ell}\right)^2+\left(\frac{y}{\ell}+1\right)^2}}\right)$$

$$m\ddot{y} = k\left(\frac{y-\ell}{\sqrt{\left(\frac{x}{\ell}\right)^2+\left(\frac{y}{\ell}-1\right)^2}} - 2y + \frac{y+\ell}{\sqrt{\left(\frac{x}{\ell}\right)^2+\left(\frac{y}{\ell}+1\right)^2}}\right)$$

3 Equation of motion and Initial condition

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ \dfrac{k}{m}\left(\dfrac{x}{\sqrt{\left(\frac{x}{\ell}\right)^2+\left(\frac{y}{\ell}-1\right)^2}} - 2x + \dfrac{x}{\sqrt{\left(\frac{x}{\ell}\right)^2+\left(\frac{y}{\ell}+1\right)^2}}\right) \\ \dfrac{k}{m}\left(\dfrac{y-\ell}{\sqrt{\left(\frac{x}{\ell}\right)^2+\left(\frac{y}{\ell}-1\right)^2}} - 2y + \dfrac{y+\ell}{\sqrt{\left(\frac{x}{\ell}\right)^2+\left(\frac{y}{\ell}+1\right)^2}}\right) \end{pmatrix} \tag{1}$$

$$\text{IVC}: \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix}_{(t_0)} = \begin{pmatrix} x_0 \\ y_0 \\ v_{x0} \\ v_{y0} \end{pmatrix}$$

We can solve this equation Numerically with the help of Python's Libraries.

4 Euler's method

One powerful but slow method for solving system of ordinary differential equation is Euler's method: when we have a SODE with IVC :

$$\frac{\mathrm{d}}{\mathrm{d}t}(\vec{\mathbf{x}}) = \vec{\mathbf{f}}(t, \vec{\mathbf{x}})$$

In order to solve the equation numerically we should define these variables.
t_0 = initial time , h = small change in time , $\vec{\mathbf{x}}_0$ = initial conditions
then:

$$\vec{\mathbf{x}}_{n+1} = \vec{\mathbf{x}}_n + h\vec{\mathbf{f}}(t_n, \vec{\mathbf{x}}_n)$$

5 Python Program

```
# Libraries
import matplotlib.pyplot as plt

# Parameters
print("Please Enter the following parameters:")
m = float(input("mass of particle in kilograms:"))
k = float(input("coeffiecient of springs in N/m:"))
l = float(input("length in meter:"))
x0, y0 = map(float, input("Initial position (x0, y0) in meters:").split())
vx0, vy0 = map(float, input("Initial velocity (vx0, vy0) in m/s:").split())
t0 = 0
dt = float(input("Accuracy in time:"))
tf = float(input("Final time in seconds:"))
omega2 = k/m


# Functions
def vdotx(x, y):
    co1 = ( x**2 + (y - l)**2 )**(-0.5)
    co2 = ( x**2 + (y + l)**2 )**(-0.5)
    return -1 * omega2 * x * ( 2 - ( l*(co1 + co2) ) )

def vdoty(x, y):
    co1 = ( x**2 + (y - l)**2 )**(-0.5)
    co2 = ( x**2 + (y + l)**2 )**(-0.5)
    return -1 * omega2 * ( ((y+l)*(1-co2)) + ((y-l)*(1-co1)) )
```

```
n = int(tf // dt)     # number of steps
time = [0]
xc   = [x0]
yc   = [y0]
vx   = [vx0]
vy   = [vy0]
for i in range(n):
    time.append((i+1)*dt)
    xc.append(xc[i] + dt*(vx[i]))
    yc.append(yc[i] + dt*(vy[i]))
    vx.append(vx[i] + dt*float(vdotx(xc[i], yc[i])))
    vy.append(vy[i] + dt*float(vdoty(xc[i], yc[i])))

# The plot
plt.plot(xc, yc)
plt.show()
```

6 Some Examples

Main Setup: In a hypothetical world we can assume the mass is given by $m = 0.250$ kg and k is 5 N/m and the length is given by $\ell = 0.30$ m.

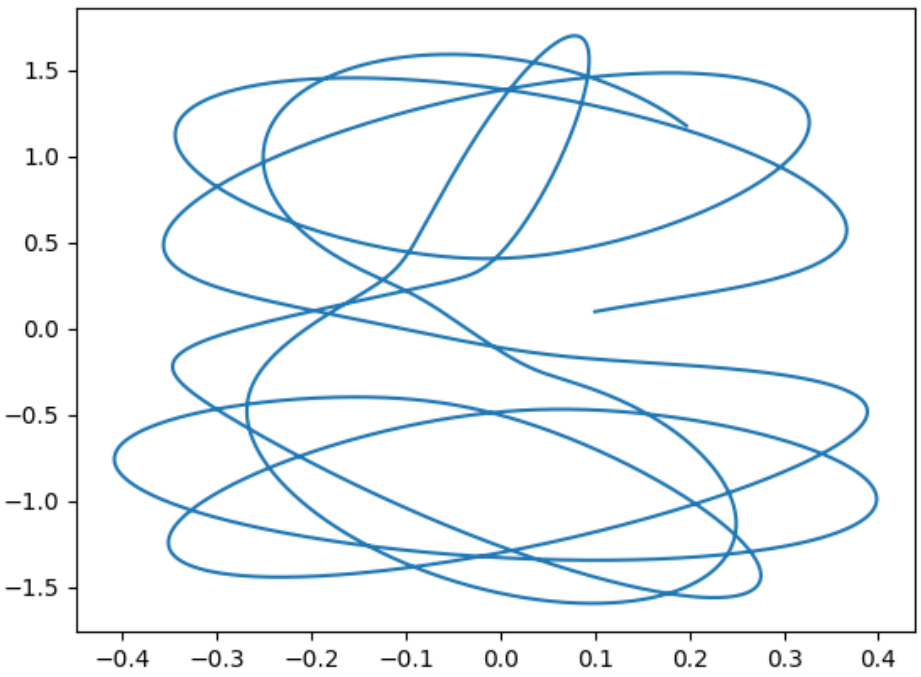

Figure 2: $x_0 = 0.1$, $y_0 = 0.1$, $v_{x0} = 1$, $v_{y0} = 1$

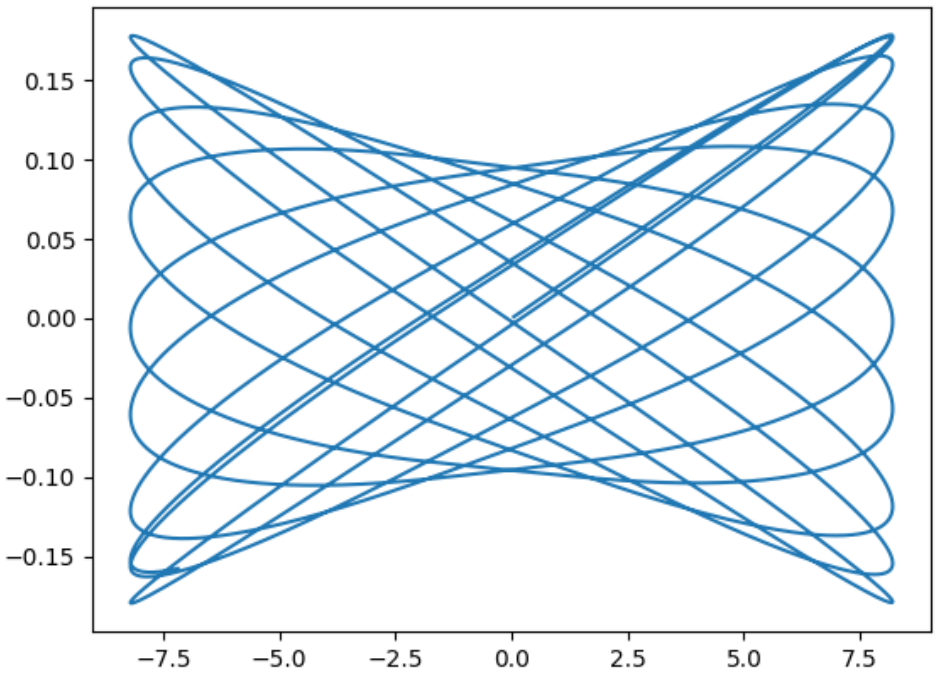

Figure 3: $x_0 = 0.05$, $y_0 = 0.001$, $v_{x0} = 50$, $v_{y0} = 1$

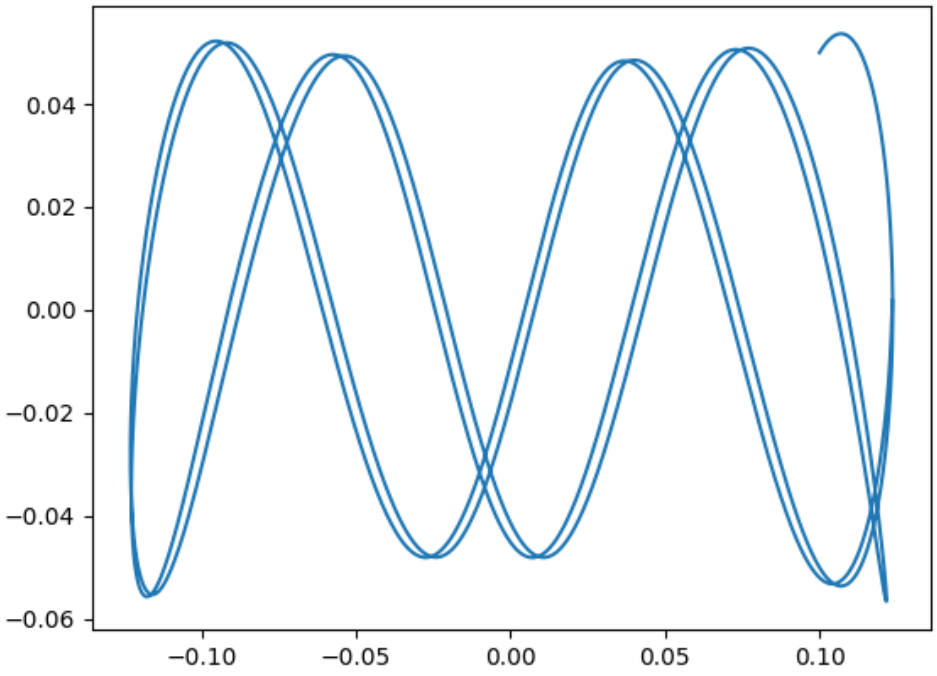

Figure 4: $x_0 = 0.1$, $y_0 = 0.005$, $v_{x0} = 0.1$, $v_{y0} = 0.1$

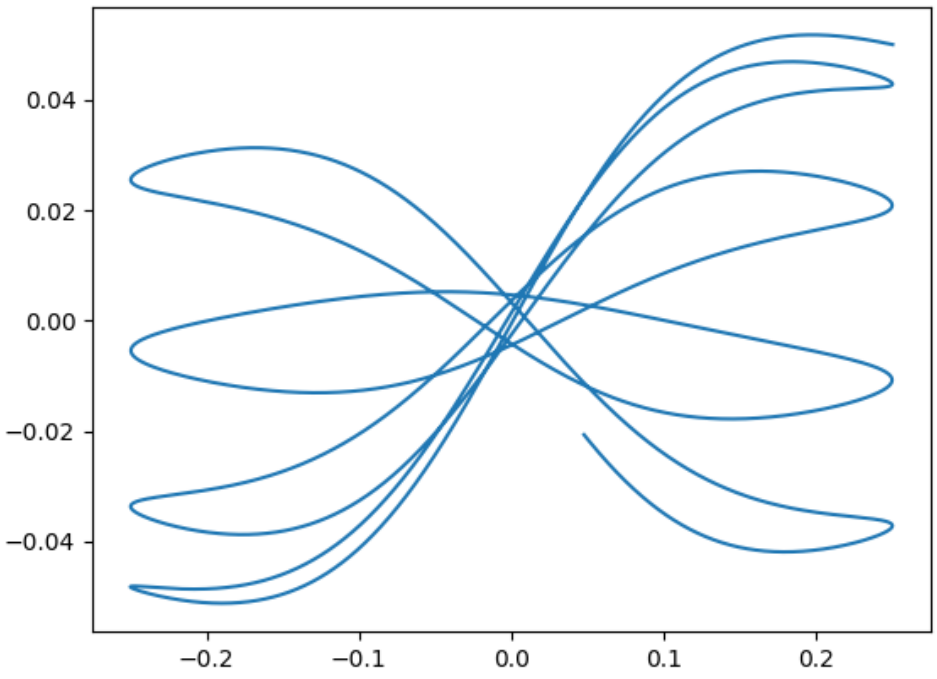

Figure 5: $x_0 = 0.25$, $y_0 = 0.05$, $v_{x0} = 0$, $v_{y0} = 0$

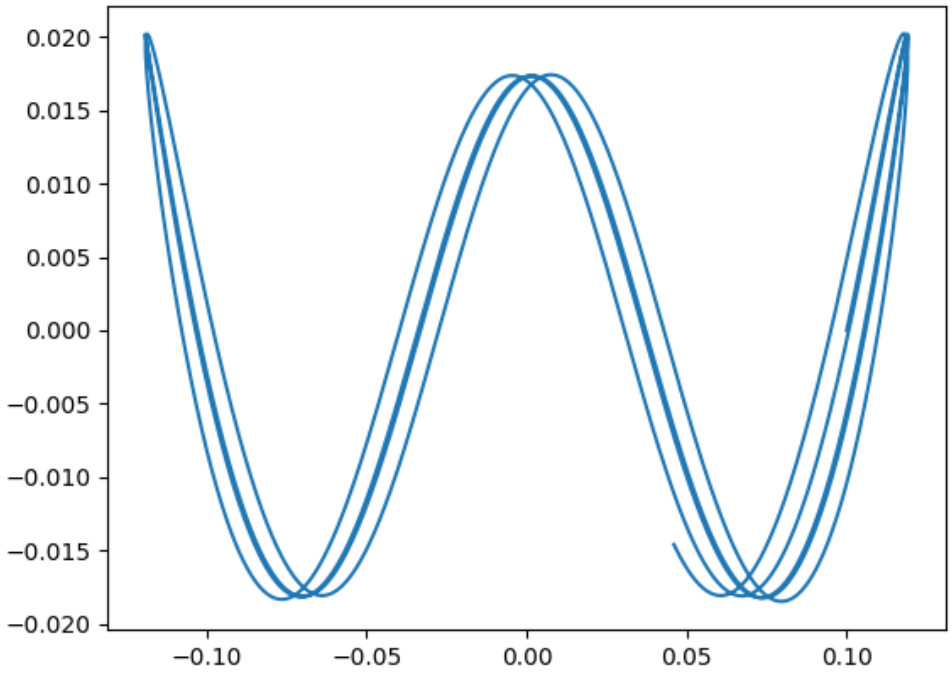

Figure 6: $x_0 = 0.1$, $y_0 = 0$, $v_{x0} = 0.1$, $v_{y0} = 0.1$

7 Chaos and non-linearity

As you may have seen in the results the answer of equation is highly dependent on the initial conditions and varies drastically by changing the initial value a bit. also this results in some odd behaviour in solving it numerically.
From Conservation of energy we know that the total energy cannot be exceeded from the initial energy. Also, this is reflected in the equation of motion. When viewed from an energy standpoint it obeys the conservation laws, but there are some instances where we can clearly see a gradual increment in energy over time: As a matter of fact, this approximation is very

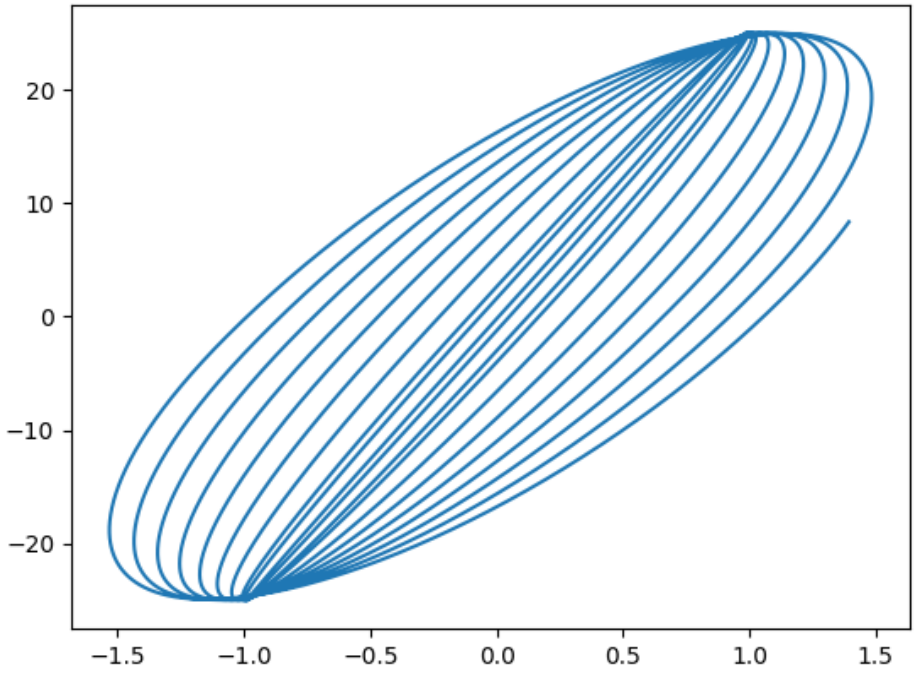

Figure 7: $x_0 = 1$, $y_0 = 20$, $v_{x0} = 1$, $v_{y0} = 1$

sensitive to dt where it changes the shape of the path drastically with change of discrete steps. this shows that this non-linear system of differential equations is chaotic and tolerates no approximation.