In [3]:
```python
import numpy as np
import sympy as smp
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from matplotlib import animation
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import PillowWriter
```

In [4]:
```python
# Using smp.symbols we defind the variables (in symbolic forms)

t, g = smp.symbols('t g')
m1, m2 = smp.symbols('m1 m2')
L1, L2 = smp.symbols('L1, L2')
```

In [5]:
```python
# We define the angles as fiunction of time.

the1, the2 = smp.symbols(r'\theta_1, \theta_2', cls=smp.Function) # Last part enables
```

In [6]:
```python
the1 = the1(t)
the2 = the2(t)
```

In [7]:
```python
the1 # Calling theta 1 for demonstration
```

Out[7]: $\theta_1(t)$

In [8]:
```python
# Using sympy function, we operate on the above variables.

# First order derivative with respect to time.
the1_d = smp.diff(the1, t)
the2_d = smp.diff(the2, t)
# Second order derivative with respect to time.
the1_dd = smp.diff(the1_d, t)
the2_dd = smp.diff(the2_d, t)
```

In [9]:
```python
# We define the spatial coordinates for the 2 masses

x1 = L1*smp.sin(the1)
y1 = -L1*smp.cos(the1)
x2 = L1*smp.sin(the1)+L2*smp.sin(the2)
y2 = -L1*smp.cos(the1)-L2*smp.cos(the2)
```

In [10]:
```python
# Kinetic enegies;
T1 = 1/2 * m1 * (smp.diff(x1, t)**2 + smp.diff(y1, t)**2)
T2 = 1/2 * m2 * (smp.diff(x2, t)**2 + smp.diff(y2, t)**2)
T = T1+T2
# Potential energies;
V1 = m1*g*y1
V2 = m2*g*y2
V = V1 + V2
# Lagrangian
L = T-V
```

In [11]:
```python
#Kinetic =
T
```

Out[11]:

$$0.5m_1\left(L_1^2\sin^2\left(\theta_1(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + L_1^2\cos^2\left(\theta_1(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2\right)$$

$$+ 0.5m_2\left(\left(L_1\sin\left(\theta_1(t)\right)\frac{d}{dt}\theta_1(t) + L_2\sin\left(\theta_2(t)\right)\frac{d}{dt}\theta_2(t)\right)^2 + \left(L_1\cos\left(\theta_1(t)\right)\frac{d}{dt}\theta_1(t) + L_2\cos\left(\theta_2(t)\right)\right.\right.$$

In [12]:
```python
#Potential=
V
```

Out[12]:

$$-L_1gm_1\cos\left(\theta_1(t)\right) + gm_2\left(-L_1\cos\left(\theta_1(t)\right) - L_2\cos\left(\theta_2(t)\right)\right)$$

In [13]:
```python
# Lagrangian =
L
```

Out[13]:

$$L_1gm_1\cos\left(\theta_1(t)\right) - gm_2\left(-L_1\cos\left(\theta_1(t)\right) - L_2\cos\left(\theta_2(t)\right)\right) + 0.5m_1\left(L_1^2\sin^2\left(\theta_1(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + L_1^2\right.$$

$$+ 0.5m_2\left(\left(L_1\sin\left(\theta_1(t)\right)\frac{d}{dt}\theta_1(t) + L_2\sin\left(\theta_2(t)\right)\frac{d}{dt}\theta_2(t)\right)^2 + \left(L_1\cos\left(\theta_1(t)\right)\frac{d}{dt}\theta_1(t) + L_2\cos\left(\theta_2(t)\right)\right.\right.$$

In [14]:
```python
# Lagrangian Eqautions ( Only LHS )

LE1 = smp.diff(L, the1) - smp.diff(smp.diff(L, the1_d), t).simplify()
LE2 = smp.diff(L, the2) - smp.diff(smp.diff(L, the2_d), t).simplify()
```

In [15]:
```python
# NOW WE HAVE 2 EQUATIONS with 2nd ORDER DIFFERENTIALS
```

In [16]:
```python
sols = smp.solve([LE1, LE2], (the1_dd, the2_dd),simplify=False, rational=False)
```

In [17]:
```python
sols[the1_dd]
```

Out[17]:

$$- \frac{1.0 L_{1} m_{2} \sin{\left(\theta_{1}(t) - \theta_{2}(t) \right)} \cos{\left(\theta_{1}(t) - \theta_{2}(t) \right)} \left(\frac{d}{dt} \theta_{1}(t)\right)^{2}}{1.0 L_{1} m_{1} - 1.0 L_{1} m_{2} \cos^{2}{\left(\theta_{1}(t) - \theta_{2}(t) \right)} + 1.0 L_{1} m_{2}} + \frac{1.0 L_{1} m_{2} \sin{\left(\theta_{1}(t) - \theta_{2}(t) \right)} \cos (\theta_{1}(t) - \theta_{2}(}{1.0 L_{1} m_{1} - 1.0 L_{1} m_{2} \cos^{2} (\theta_{1}(t) - \theta_{2}(t}$$
$$- \frac{1.0 L_{1} m_{2} \sin{\left(\theta_{1}(t) \right)} \cos{\left(\theta_{1}(t) - \theta_{2}(t) \right)} \cos{\left(\theta_{2}(t) \right)} \frac{d}{dt} \theta_{1}(t) \frac{d}{dt} \theta_{2}(t)}{1.0 L_{1} m_{1} - 1.0 L_{1} m_{2} \cos^{2}{\left(\theta_{1}(t) - \theta_{2}(t) \right)} + 1.0 L_{1} m_{2}} + \frac{1.0 L_{1} m_{2} \sin{\left(\theta_{2}(t) \right)} \cos (\theta_{1}(t) - \theta_{2}(}{1.0 L_{1} m_{1} - 1.0 L_{1} m_{2} \cos^{2} (\theta_{1}(}$$
$$+ \frac{1.0 L_{2} m_{2} \sin{\left(\theta_{1}(t) - \theta_{2}(t) \right)} \frac{d}{dt} \theta_{1}(t) \frac{d}{dt} \theta_{2}(t)}{1.0 L_{1} m_{1} - 1.0 L_{1} m_{2} \cos^{2}{\left(\theta_{1}(t) - \theta_{2}(t) \right)} + 1.0 L_{1} m_{2}} - \frac{1.0 L_{2} m_{2} \sin{\left(\theta_{1}(t) - \theta_{2}(t) \right)} \left(\frac{d}{dt} \theta_{2}(t)\right)^{2}}{1.0 L_{1} m_{1} - 1.0 L_{1} m_{2} \cos^{2}{\left(\theta_{1}(t) - \theta_{2}(t) \right)} + 1.(}$$
$$- \frac{1.0 L_{2} m_{2} \sin{\left(\theta_{1}(t) \right)} \cos{\left(\theta_{2}(t) \right)} \frac{d}{dt} \theta_{1}(t) \frac{d}{dt} \theta_{2}(t)}{1.0 L_{1} m_{1} - 1.0 L_{1} m_{2} \cos^{2}{\left(\theta_{1}(t) - \theta_{2}(t) \right)} + 1.0 L_{1} m_{2}} + \frac{1.0 L_{2} m_{2} \sin{\left(\theta_{2}(t) \right)} \cos{\left(\theta_{1}(t) \right)} \frac{d}{dt} \theta_{1}(t) \frac{d}{dt} \theta_{2}(}{1.0 L_{1} m_{1} - 1.0 L_{1} m_{2} \cos^{2}{\left(\theta_{1}(t) - \theta_{2}(t) \right)} + 1.(}$$
$$- \frac{1.0 g m_{1} \sin{\left(\theta_{1}(t) \right)}}{1.0 L_{1} m_{1} - 1.0 L_{1} m_{2} \cos^{2}{\left(\theta_{1}(t) - \theta_{2}(t) \right)} + 1.0 L_{1} m_{2}} - \frac{1.0 g m_{2} \sin{\left(\theta_{1}(t) \right)}}{1.0 L_{1} m_{1} - 1.0 L_{1} m_{2} \cos^{2}{\left(\theta_{1}(t) - \theta_{2}(t) \right)} + 1.(}$$
$$+ \frac{1.0 g m_{2} \sin{\left(\theta_{2}(t) \right)} \cos{\left(\theta_{1}(t) - \theta_{2}(t) \right)}}{1.0 L_{1} m_{1} - 1.0 L_{1} m_{2} \cos^{2}{\left(\theta_{1}(t) - \theta_{2}(t) \right)} + 1.0 L_{1} m_{2}}$$

In [18]:
```python
# We need to solve the above symbolic 2nd order ODE usinf NUMERICAL MEANS as following
```

In [19]:
```python
dz1dt_f = smp.lambdify((t,g,m1,m2,L1,L2,the1,the2,the1_d,the2_d), sols[the1_dd])
dz2dt_f = smp.lambdify((t,g,m1,m2,L1,L2,the1,the2,the1_d,the2_d), sols[the2_dd])
dthe1dt_f = smp.lambdify(the1_d, the1_d)
dthe2dt_f = smp.lambdify(the2_d, the2_d)
```

In [20]:
```python
dz1dt_f(2,9.8,1,1,1,1,4,4,4,4) #NUMERICAL FUNCTION/example
```

Out[20]: 7.416664454017697

In [ ]:
```python
# Defining S as the function of z1 , z2, theta1, theta2
```

In [21]:

```
def dSdt(S, t, g, m1, m2, L1, L2):
    the1, z1, the2, z2 = S
    return [
        dthe1dt_f(z1),
        dz1dt_f(t, g, m1, m2, L1, L2, the1, the2, z1, z2),
        dthe2dt_f(z2),
        dz2dt_f(t, g, m1, m2, L1, L2, the1, the2, z1, z2),
    ]
```

In [23]:

```
t = np.linspace(0, 100, 1001)
g = 9.81
m1=2
m2=1
L1 = 2
L2 = 1
ans = odeint(dSdt, y0=[1, -3, -1, 5], t=t, args=(g,m1,m2,L1,L2)) # USING inbuilt ODE t
```

In [24]:

```
t
```

Out[24]:

```
array([  0. ,   0.1,   0.2, ...,  99.8,  99.9, 100. ])
```

In [25]:

```
len(t[t<1]) #25 NUMERIC VALUES PER SECOND
```

Out[25]:

```
10
```

In [26]:

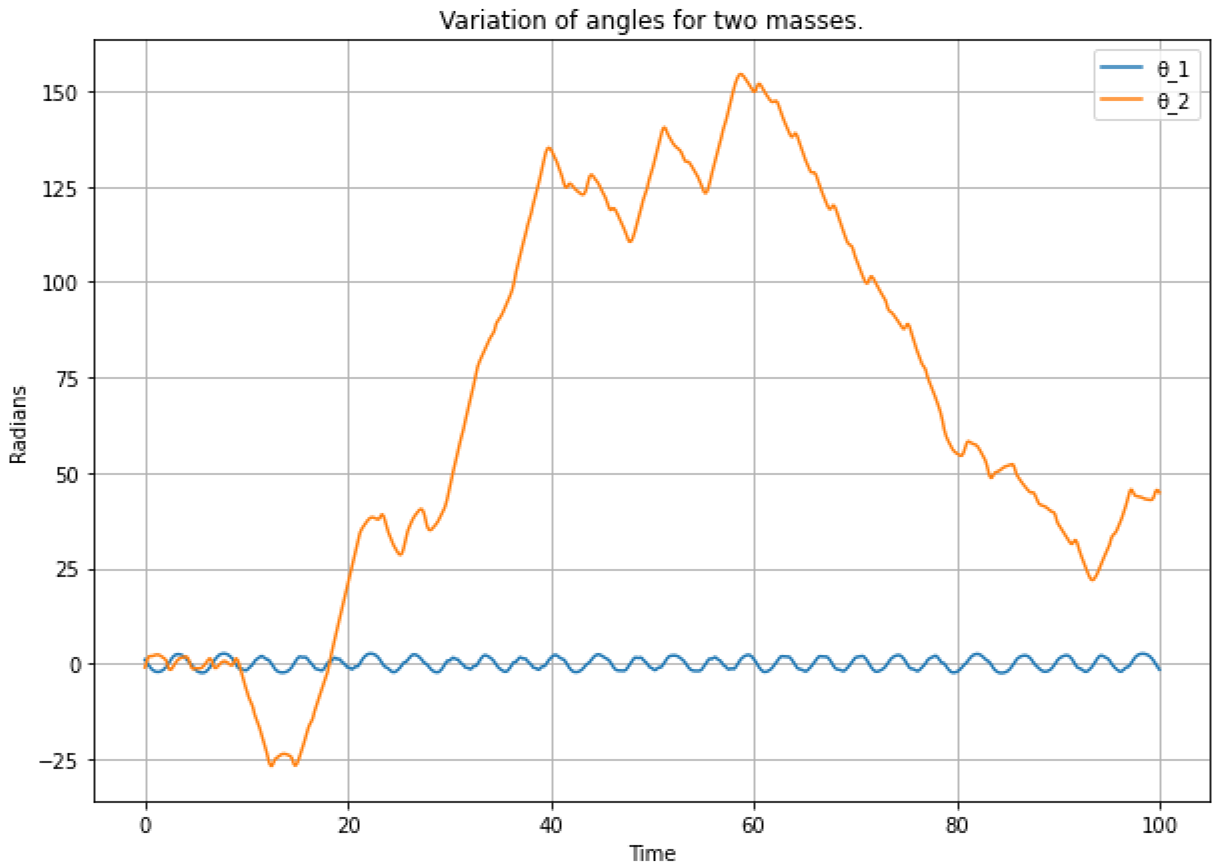
```
ans
```

Out[26]:

```
array([[ 1.        , -3.        , -1.        ,  5.        ],
       [ 0.65520407, -4.06290099, -0.36794454,  8.13329933],
       [ 0.17507665, -4.89789289,  0.65751614, 10.30400114],
       ...,
       [-0.86947179, -3.77828729, 45.43019457, -0.98994337],
       [-1.24083989, -3.60081172, 45.19649988, -3.52341054],
       [-1.57879455, -3.1018809 , 44.75729301, -5.10382174]])
```

In [34]:

```python
the1 = ans.T[0]
the2 = ans.T[2]
plt.rcParams["figure.figsize"]=(10,7)
plt.plot(t,the1)
plt.plot(t,the2)
plt.xlabel('Time')
plt.ylabel('Radians')
plt.legend(['θ_1','θ_2'])
plt.title('Variation of angles for two masses.')
plt.grid()
```



In [28]:
```python
# JUSTIFIES A CHAOTIC VALUE
```

In [29]:
```python
# RETURNS SPATIAL COORDINATES by above functions for the animations.
def get_x1y1x2y2(t, the1, the2, L1, L2):
    return (L1*np.sin(the1),
            -L1*np.cos(the1),
            L1*np.sin(the1) + L2*np.sin(the2),
            -L1*np.cos(the1) - L2*np.cos(the2))

x1, y1, x2, y2 = get_x1y1x2y2(t, ans.T[0], ans.T[2], L1, L2) #Required array
```

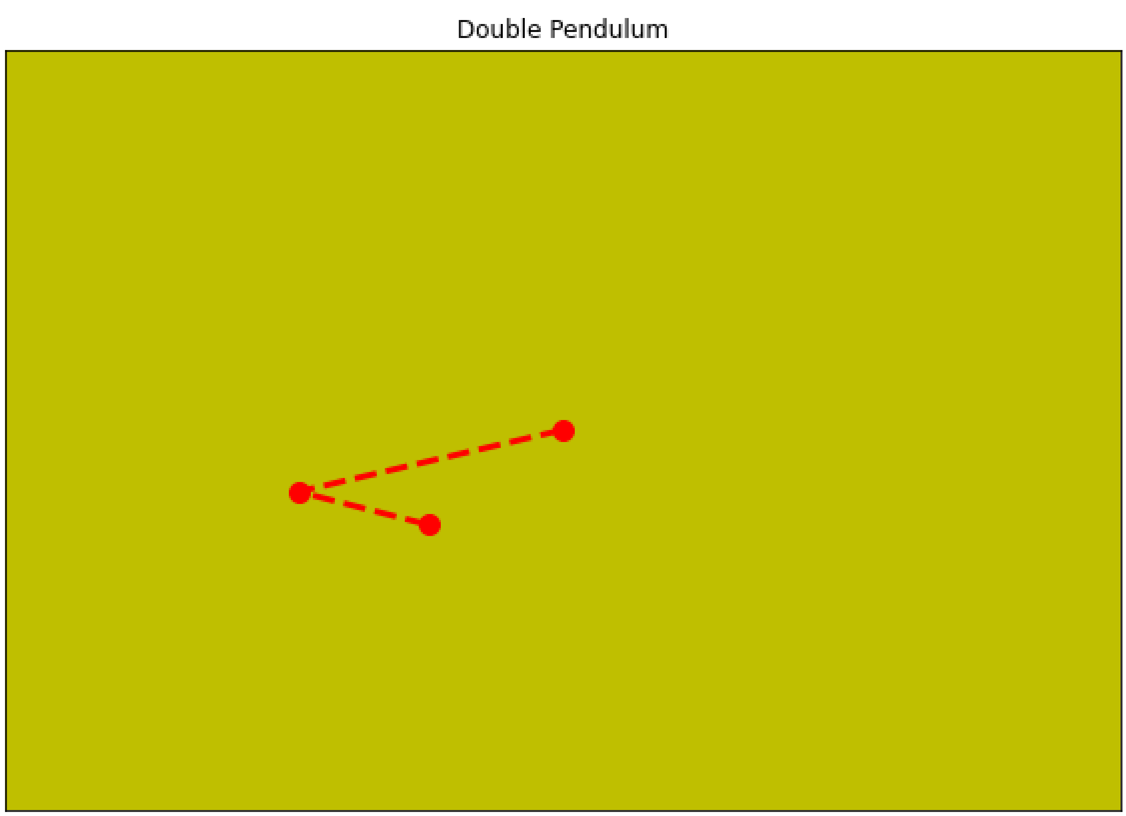
In [30]:
```python
x1
```

Out[30]:
```
array([ 1.68294197,  1.21864214,  0.34836724, ..., -1.52797645,
       -1.89211292, -1.99993603])
```

In [31]:
```python
#ANIMATION
```

In [32]:
```python
def animate(i):
    ln1.set_data([0, x1[i], x2[i]], [0, y1[i], y2[i]])   #Locations of 1st and 2nd BOB

fig, ax = plt.subplots(1,1, figsize=(10,7))
ax.set_title("Double Pendulum")
ax.set_facecolor('y')
ax.get_xaxis().set_ticks([])    # enable this to hide x axis ticks
ax.get_yaxis().set_ticks([])    # enable this to hide y axis ticks
ln1, = plt.plot([], [], 'ro--', lw=3, markersize=10)
ax.set_ylim(-4,4)
ax.set_xlim(-4,4)
ani = animation.FuncAnimation(fig, animate, frames=1000, interval=50)
ani.save('pen.gif',writer='pillow',fps=25)
```



In [ ]:
```python

```

In [ ]:

```
 
```

In [ ]:

```
 
```