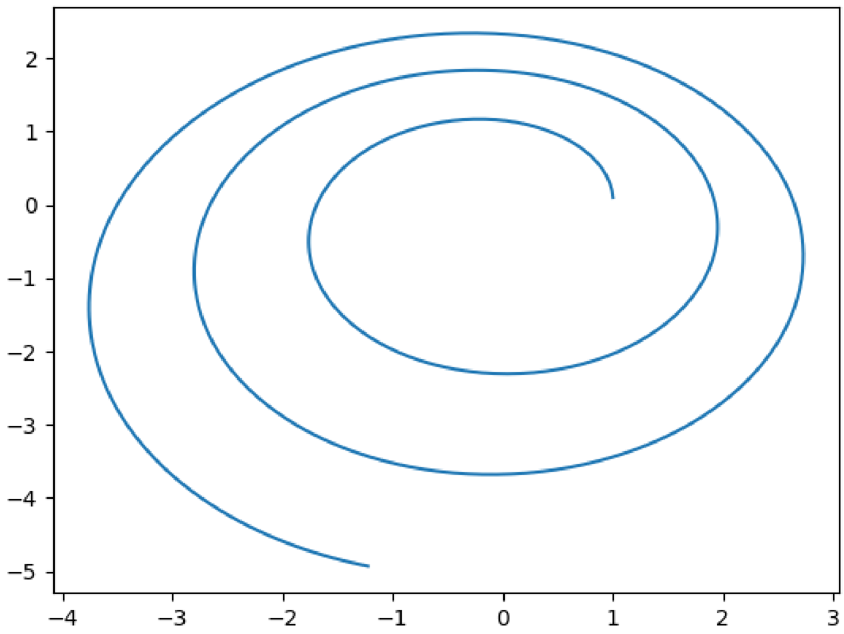
MILESTONE I - Miguel del Fresno Zarza

Código para Kepler con método de Euler

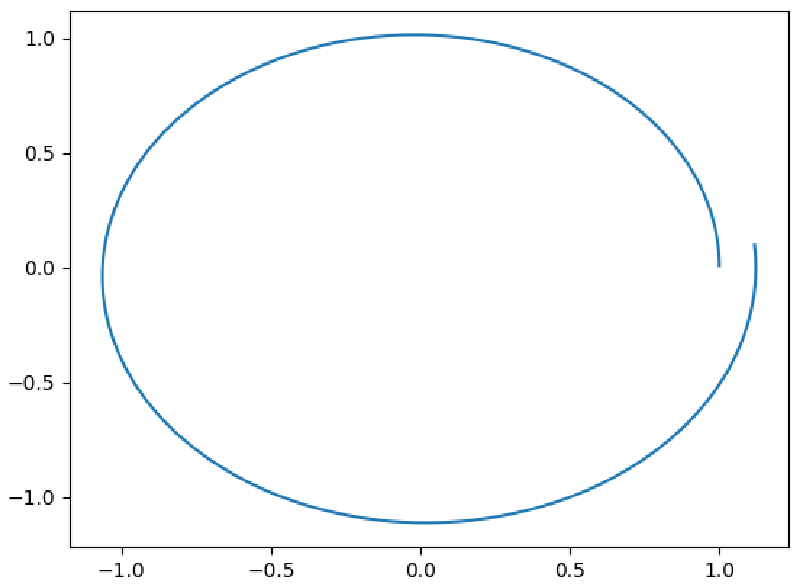
```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as sp

def kepler_abstraction_euler():

    #condiciones iniciales de Cauchy
    U0 = np.array( [ 1, 0, 0, 1] )
    N = 150
    dt = 0.1

    #Función de cálculo de Euler para Kepler
    U = kepler_euler( U0, dt, N)

def kepler_euler( U0, dt, N):

    nu = len(U0)
    U = np.zeros(nu)
    U = U0
    x = np.zeros(N)
    y = np.zeros(N)
    x[0] = U0[0]
    y[0] = U0[1]
    for i in range(N):

        #Ecuaciones del problema de Euler
        F = F_euler(U,i)
        U = U_euler(U,F,dt,i)
        x[i] = U[0]
        y[i] = U[1]

    #Gráfica de la órbita
    plt.plot( x, y)
    plt.show()

def U_euler(U,F,dt,i):

    return U + dt * F

def F_euler(U,i):

    x = U[0]; y = U[1]; dx_dt = U[2]; dy_dt = U[3]
    d = ( x**2 + y**2 )**1.5
    return np.array([ float(dx_dt), float(dy_dt), float(-x/d), float(-y/d)])
```

En Kepler_euler se establecen las ecuaciones para resolver el problema de Euler, donde se llama a otras funciones para simplificar el código.

Cuanto más pequeño es el incremento del tiempo, menor es el error.



dt = 0.1



dt = 0.01

Como se puede apreciar, el error disminuye al hacer el incremento de tiempo más pequeño.

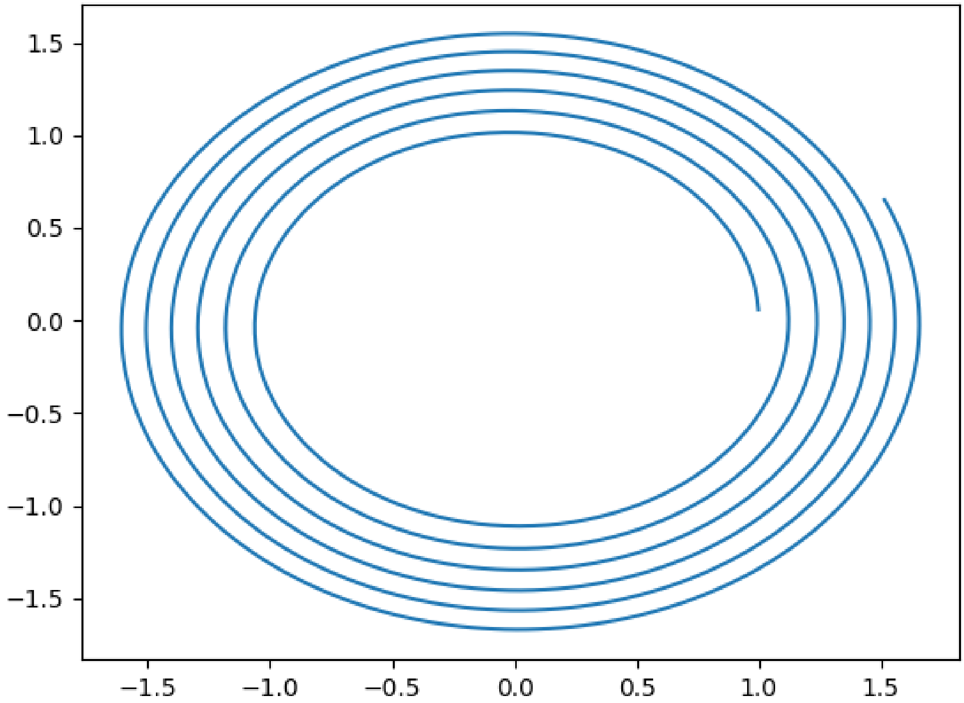
Código para Kepler con método de Euler

```
def kepler_abstraction_rk4():

    #condiciones iniciales de Cauchy
    U0 = np.array([ 1, 0, 0, 1])
    N = 1000
    dt = 0.01

    #Función de cálculo de Runge Kutta para Kepler
    kepler_rk4(U0,N,dt)

def kepler_rk4(U0,N,dt):

    k1 = np.zeros(4)
    k2 = np.zeros(4)
    k3 = np.zeros(4)
    k4 = np.zeros(4)
    x = np.zeros(N)
    y = np.zeros(N)
    x[0] = U0[0]
    y[0] = U0[1]

    nu = len(U0)
    U = np.zeros(nu)
    U = U0

    for i in range(N):

        #Ecuaciones del problema de Runge Kutta de 4º Orden
        k1 = rk4_k1(U,dt,i)
        k2 = rk4_k(U,dt,k1,i)
        k3 = rk4_k(U,dt,k2,i)
        k4 = rk4_k(U,dt,k3,i)

        U = F_rk(U,k1,k2,k3,k4,dt,i)

        x[i] = U[0]
        y[i] = U[1]

    #Gráfica de la órbita
    plt.plot( x, y)
    plt.show()
```

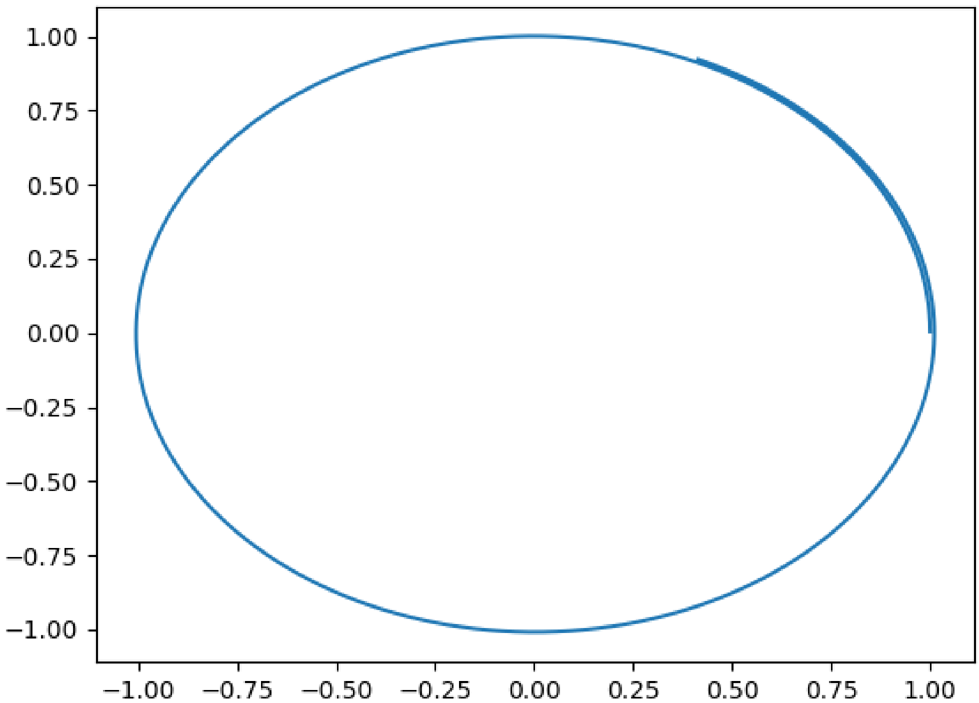
```python

def rk4_k1(U,dt,i):

    x = U[0]
    y = U[1]
    dx = U[2]
    dy = U[3]
    d = ( x**2 + y**2)**1.5

    return np.array([ dx, dy, -x/d, -y/d])

def rk4_k(U,dt,k,i):
    a = i-1
    k_ = U + k * dt / 2
    x = k_[0]
    y = k_[1]
    dx = k_[2]
    dy = k_[3]
    d = ( x**2 + y**2)**1.5

    return np.array([ dx, dy, -x/d, -y/d])

def F_rk(U,k1,k2,k3,k4,dt,i):

    return U + dt*( k1 + 2 * k2 + 2 * k3 + k4 )/6

kepler_abstraction_rk4()

```

En la función Kepler_rk4 se establecen las ecuaciones para resolver el problema. Al igual que con Euler, se llama a otras funciones para simplificar el código y realizar la abstracción.

También, como cabe esperar, el error disminuye al hacer el incremento de tiempo más pequeño.


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

dt = 0.1


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

dt = 0.01

Por último, se puede comprobar que el método de Runge Kutta, pese a ser más complejo, tiene un error más bajo que se puede apreciar a simple vista.