Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestone 2

Ampliación de Matemáticas 1

5 de octubre de 2022

Autores:

- Jaime Jiménez-Alfaro Piédrola

Índice

1. Introducción

En este informe se va a integrar una órbita de Kepler con diferentes esquemas numéricos. Se va a explicar el código que se utiliza para ello y se van a comentar los resultados que se obtienen.

2. Código

El código que se ha utilizado para resolver este problema se ha separado en diferentes módulos.

2.1. Módulo: Kepler

En este módulo se encuentra el problema que se quiere resolver. Se quiere integrar una órbita de Kepler conociendo las condiciones iniciales.

El código es:

```
# Kepler(orbita)
def Kepler(U, t):

    x = U[0]; y = U[1]; dx_dt = U[2]; dy_dt = U[3]
    d = (x**2 + y**2)**1.5

    return array( [ dx_dt, dy_dt, -x/d, -y/d] )
# ------------
```

Figura 1: Código del módulo Kepler

2.2. Módulo: EDO

En este módulo se encuentra el problema de Cauchy.

Las ecuaciones del Problema de Cauchy son:

$$\frac{d\vec{U}}{dt} = \vec{F}(U,t) \tag{1}$$

$$\vec{U}(0) = \vec{U}_0 \tag{2}$$

El código es:

```python
from numpy import zeros, float64

#----- MODULO EDO(Ecuaciones diferenciales ordinarias)-----#



def Cauchy(F,t,U0,E_Temporal):

    n, nv = len(t)-1,len(U0)

    U = zeros((nv,n+1), dtype=float64)

    U[:,0] = U0

    for i in range(n):

        U[:,i+1] = E_Temporal(U[:,i],t[i+1] - t[i],t[i],F)

    return U
```

Figura 2: Código del módulo EDO

2.3. Módulo: Matemáticas

En este módulo se encuentran algunas funciones matemáticas.

Para los esquemas numéricos implícitos es necesario usar el método Newton-Raphson, y para dicho método es necesario desarrollar un código de la matriz Jacobiana.

El código es:

```python
from numpy import array, zeros, dot
from numpy.linalg import inv, norm

#----- MODULO MATEMATICAS -----#


# Jacobiano
def Jacobiano(F, U):

    dx = 1e-3
    dim = len(U)
    Jab = zeros( (dim,dim) )
    x = zeros( dim )

    for i in range(dim):

        x[i] = dx
        Jab[:,i] = ( F(U + x) - F(U - x) )/( 2*dx )

    return Jab
# ------------

# Metodo de Newton
def Newton(F, U0):

    dim = len(U0)
    dx = array(zeros(dim))
    b  = array(zeros(dim))

    eps = 1
    it = 0
    it_max = 10000

    while ( eps > 1e-8 ) and ( it <= it_max ):

        it = it + 1
        Jab = Jacobiano(F,U0)
        b = F(U0)
        dx = dot( inv(Jab),b )
        U0  = U0 - dx
        eps = norm( dx )

    return U0
# ------------
```

Figura 3: Código del módulo Matemáticas

2.4. Módulo: Esquemas numéricos

En este módulo se encuentran los cuatro esquemas numéricos que se han utilizado para la integración del problema de Kepler.

2.4.1. Euler

El método de Euler se puede expresar como:

$$U^{n+1} = U^n + \Delta t_n \cdot F^n \tag{3}$$

El código es:

```
# Euler
def Euler(U, dt, t, F):

    return U + dt*F(U,t)
# -----------
```

Figura 4: Código del esquema numérico Euler

2.4.2. Runge-Kutta 4

El método de Runge-Kutta de orden 4 se puede expresar como:

$$U^{n+1} = U^n + \frac{\Delta t_n}{6} \left(k_1 + 2k_2 + 2k_3 + k_4\right) \tag{4}$$

Siendo:

$$k_1 = F \left(U^n \ ; \ t_n\right) \tag{5}$$

$$k_2 = F \left(U^n + \Delta t_n \frac{k_1}{2} \ ; \ t_n + \frac{\Delta t_n}{2}\right) \tag{6}$$

$$k_3 = F \left(U^n + \Delta t_n \frac{k_2}{2} \ ; \ t_n + \frac{\Delta t_n}{2}\right) \tag{7}$$

$$k_4 = F \left(U^n + \Delta t_n k_3 \ ; \ t_n + \Delta t_n\right) \tag{8}$$

El código es:

```
# Runge-Kutta orden 4
def RK4(U, dt, t, F):

    k1 = F( U, t )
    k2 = F( U + (k1 * dt)/2, t + dt/2 )
    k3 = F( U + (k2 * dt)/2, t + dt/2 )
    k4 = F( U + (k3 * dt)   , t + dt   )

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

Figura 5: Código del esquema numérico Runge-Kutta 4

2.4.3. Crank-Nicolson

El método de Cranck-Nicolson se puede expresar como:

$$U^{n+1} = U^n + \frac{\Delta t_n}{2} \left(F^{n+1} + F^n \right) \tag{9}$$

El código es:

```
# Crank-Nicolson
def Crank_Nicolson(U, dt, t, F):

    def CN(x):
        return x - U - dt/2*( F(U, t) + F(x,t) )

    return Newton( CN, U )
# ------------
```

Figura 6: Código del esquema numérico Crank-Nicolson

2.4.4. Euler inverso

El método de Euler inverso se puede expresar como:

$$U^{n+1} = U^n + \Delta t_n \cdot F^{n+1} \tag{10}$$

El código es:

```
# Euler Inverso
def Inverse_Euler(U, dt, t, F):

    def IE(x):
        return x - U - dt*F( x, t )

    return Newton( IE, U )
# ------------
```

Figura 7: Código del esquema numérico Euler inverso

2.5. Módulo: Principal Hito 2

Este módulo será el módulo principal, y desde aquí se llamará a los diferentes módulos y funciones y se harán las gráficas. También se podrá variar las variables temporales.

El código es:

```python
from numpy import array, zeros, linspace
import matplotlib.pyplot as plt

from Esquemas_Numericos import Euler, RK4, Crank_Nicolson, Inverse_Euler
from EDO import Cauchy
from Kepler import Kepler

#----- MODULO PRINCIPAL HITO 2 -----#


# Variables
T = 20          # Duracion
dt = 0.005      # Paso integracion


n = int(T/dt)
t  = linspace(0,T,n)
U0 = array( [1,0,0,1] )
U = U0

E_Temporal = [ Euler, RK4, Crank_Nicolson, Inverse_Euler ]
E_Temporal_Plot = ['EULER','RUNGE-KUTTA 4','CRANK-NICOLSON','EULER INVERSO']

# Graficas
for i in range (4):

    U = Cauchy( Kepler, t, U0, E_Temporal[i] )

    print( U[:, len(t)-1] )
    plt.title(f'{E_Temporal_Plot[i]}')
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.plot( U[0,:], U[1,:] )

    plt.savefig('Plots/' + E_Temporal_Plot[i]+ ' ' + str(dt)+'.png')
    plt.show()
```

Figura 8: Código del módulo Hito2

3. Resultados

Utilizando los cuatro métodos numéricos, se han realizado varias simulaciones variando el paso de integración Δt, con un tiempo de simulación de 25 s.

3.1. Euler

Los resultados obtenidos con el esquema numérico de Euler son:

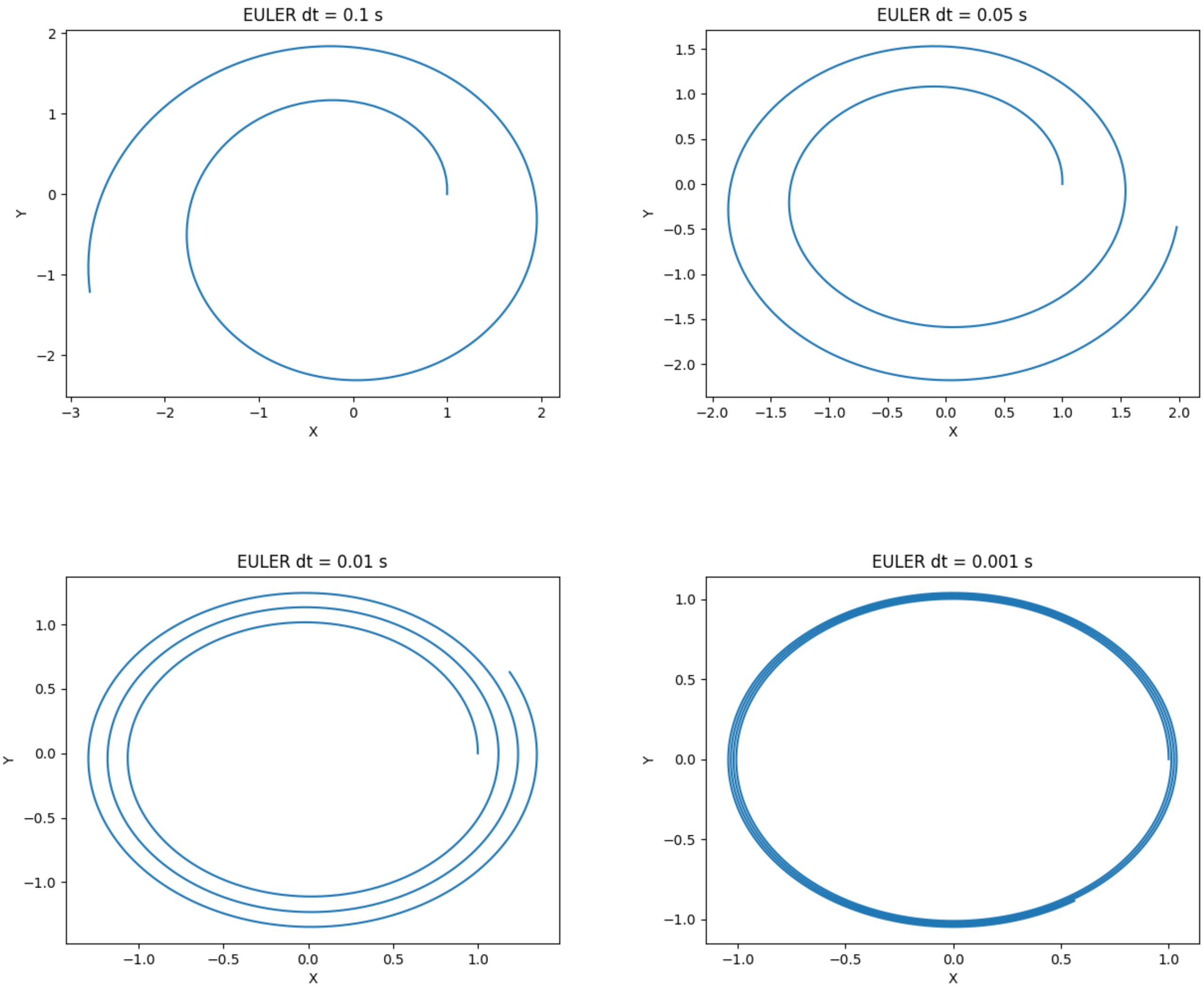

Figura 9: Resultados con Euler

En los resultados obtenidos se observa que cuanto más pequeño se hace el paso de integración Δt, la solución es más precisa.

3.2. Runge-Kutta orden 4

Los resultados obtenidos con el esquema numérico de Runge-Kutta de orden 4 son:

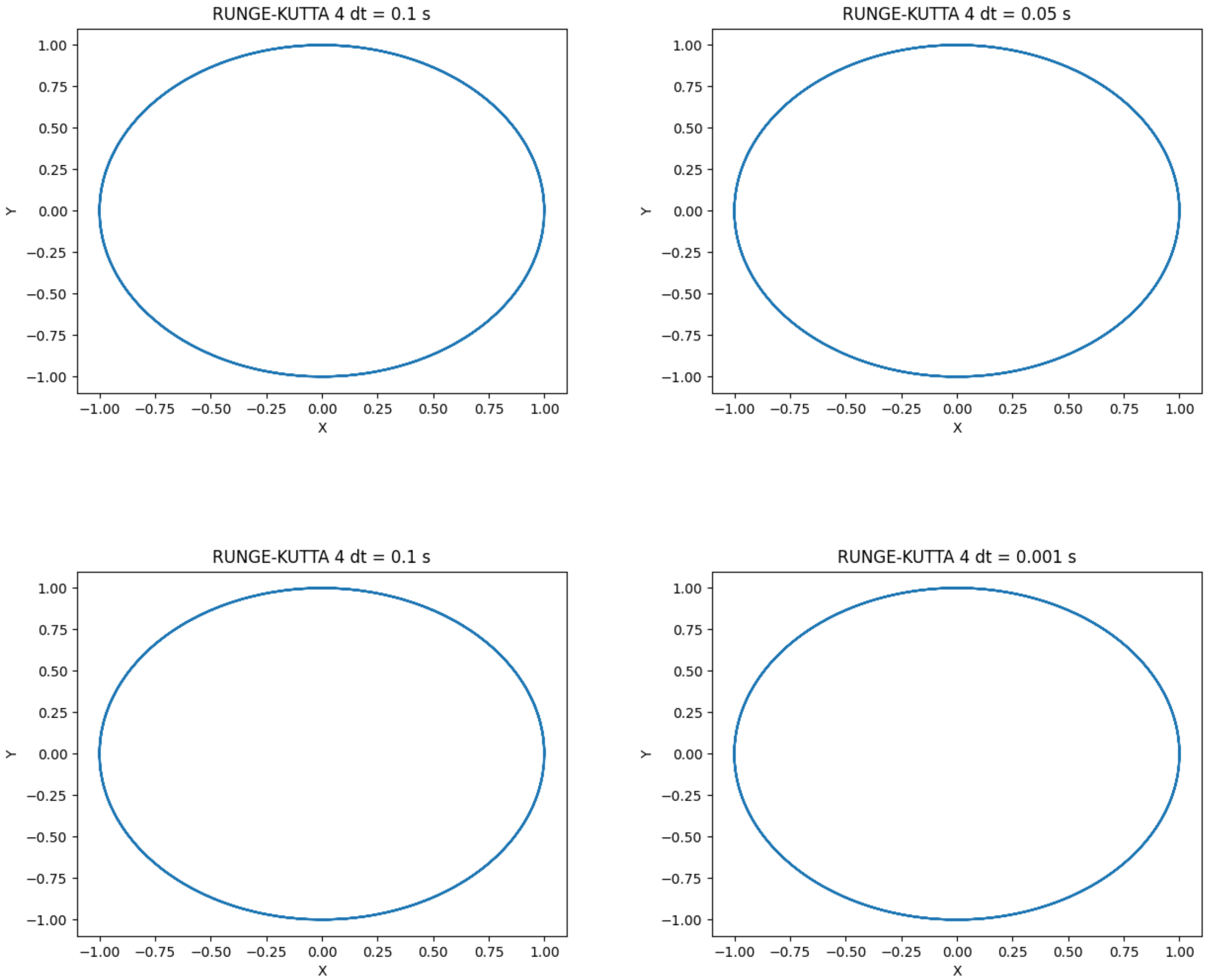

Figura 10: Resultados con Runge-Kutta orden 4

En los resultados obtenidos se observa que este esquema numérico tiene mucha precisión y aunque se varié el valor del paso de integración Δt el error es el mismo.

3.3. Crank-Nicolson

Los resultados obtenidos con el esquema numérico de Crank-Nicolson son:

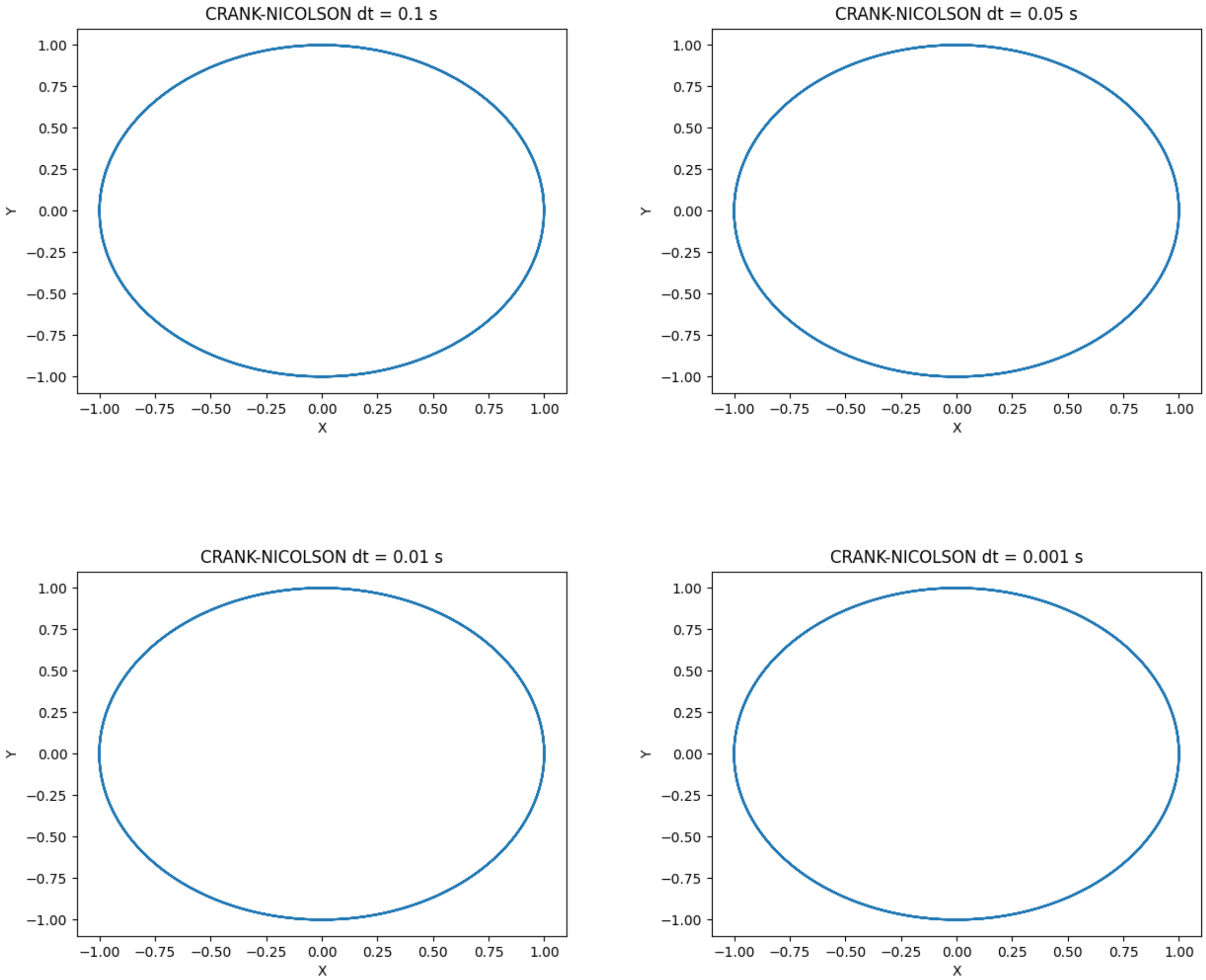

Figura 11: Resultados con Crank-Nicolson

En los resultados obtenidos se observa que este esquema numérico, al igual que el Runge-Kutta de orden 4, tiene mucha precisión y aunque se varié el valor del paso de integración Δt el error es el mismo.

3.4. Euler inverso

Los resultados obtenidos con el esquema numérico de Euler inverso son:

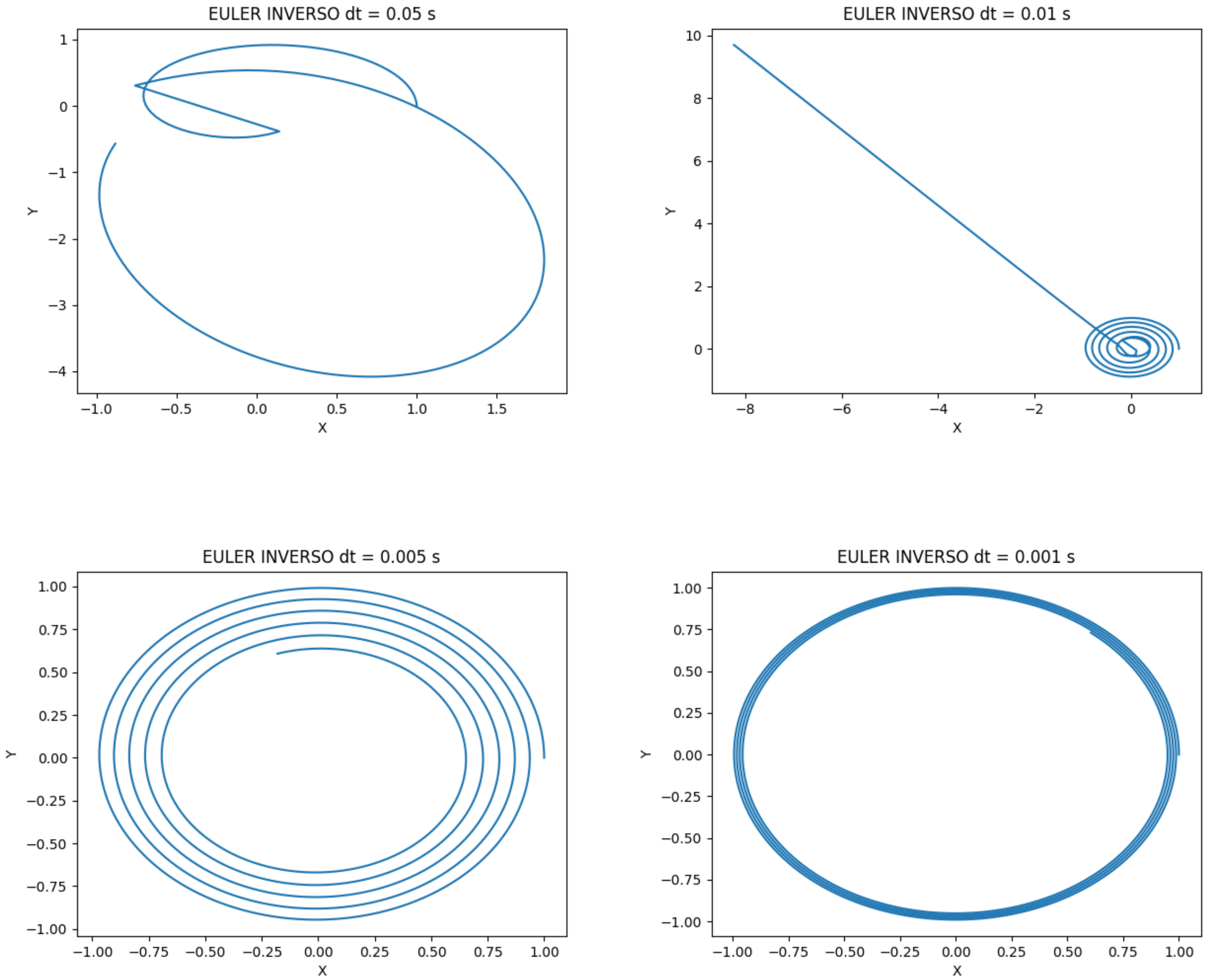

Figura 12: Resultados con Euler inverso

En los resultados obtenidos se observa que en este esquema numérico, al utilizar un valor de paso de integración Δt superior a 0.005 s, la solución no se asemeja a la que debería dar. Al disminuir el valor de paso de integración, ocurre igual que en el Euler, cuanto más pequeño se hace el paso de integración Δt, la solución es más precisa.