Milestone 2

Ampliación de Matemáticas 1

5 de octubre de 2022

Javier Garrido Castillo

I. Introducción

En este primer hito se procede a resolver el problema de Cauchy para una órbita Kepleriana a partir de los esquemas de Euler Explícito, Crank Nicolson, Runge Kutta de orden 4 y Euler Ímplicito. El problema a resolver es el siguiente:

$$\ddot{\vec{r}} = -\frac{\vec{r}}{\|\vec{r}\|^3}$$

$$\vec{r}(0) = (1, 0)$$

$$\dot{\vec{r}}(0) = (0, 1)$$

Para ello, se ha tomado inicialmente un número de pasos temporales de N = 1000 y un diferencial de tiempo entre pasos de $\Delta t = 0.01$, aplicando un t_f = 10. Posteriormente, estos valores se pueden modificar fácilmente en el código desarrollado.

Adicionalmente, en este segundo hito se implementa una metodología Top-Down. Se comentará en el apartado de "Código" como se ha llevado a cabo este método.

II. Resultados

El problema de Cauchy viene definido de la siguiente forma:

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

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

De esta forma,

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dfrac{-x}{(x^2+y^2)^{3/2}} \\ \dfrac{-y}{(x^2+y^2)^{3/2}} \end{pmatrix}$$

A. Esquema Euler explícito

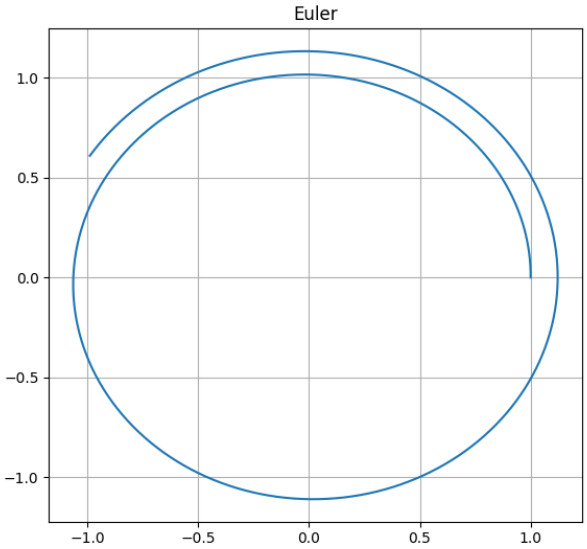


Figura 1. Resultados Euler para deltat = 0.01

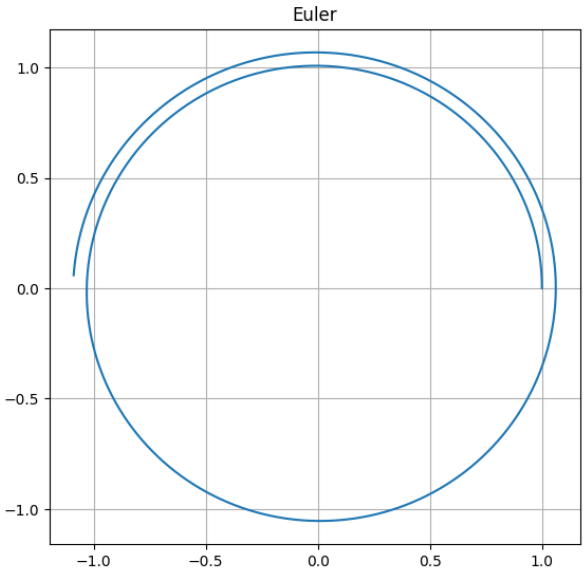


Figura 2. Resultados Euler para deltat = 0.005

Como se puede observar en las figuras anteriores, se produce un error en los resultados obtenidos. Esta divergencia aumenta cuando el diferencial de tiempo aumenta, obteniendo resultados más exactos para un diferencial de tiempo menor.

B. Esquema Runge Kutta de orden 4

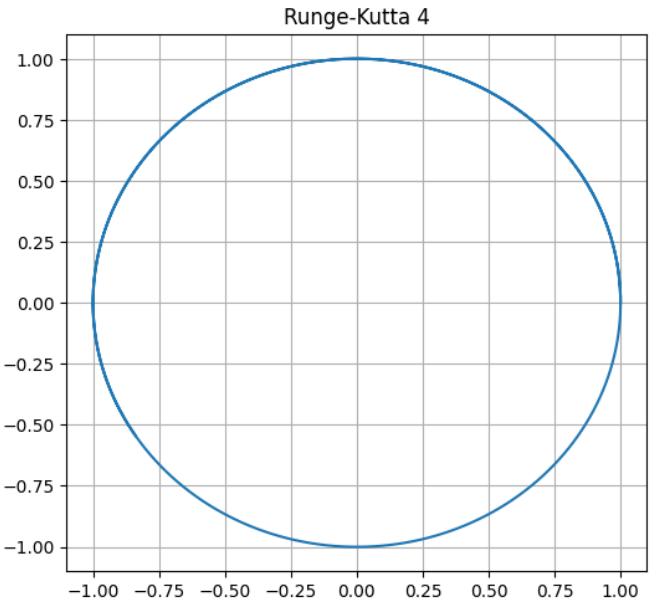


Figura 3. Resultados RK4 para ambos deltat

En este caso, con este esquema se observa una solución mucho más exacta que el apartado anterior. Los resultados son igual de precisos para los distintos diferenciales de tiempo tomados. Se observa así que, a mayor orden del esquema, mayor precisión, no influyendo en este caso el paso temporal establecido para la resolución.

C. Crank-Nicolson

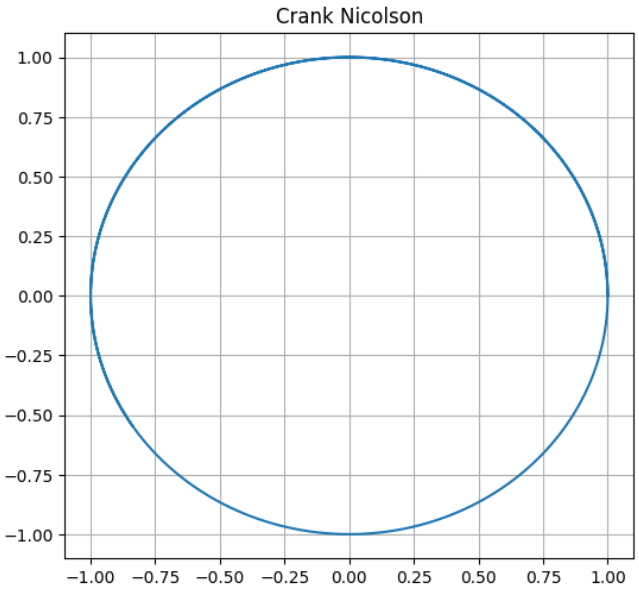


Figura 4. Resultados CN para ambos deltat

Con este esquema temporal, al igual que con el RK4 se obtiene una solución estable para distintos pasos de tiempo.

D. Euler Inverso

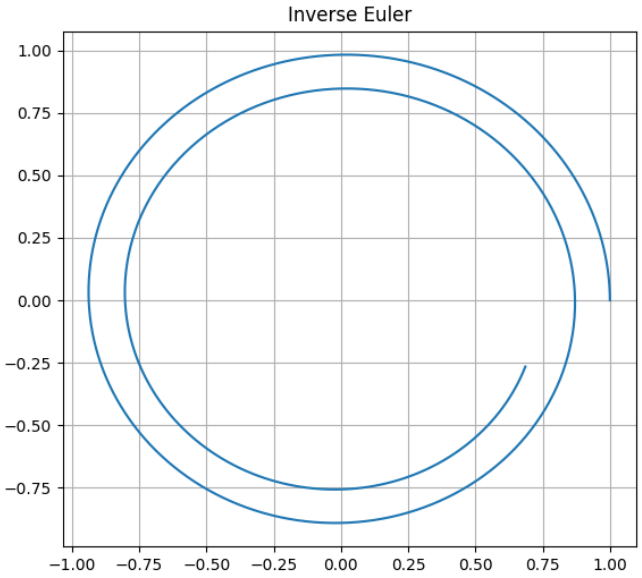


Figura 5. Resultados Euler Inverso para deltat = 0.01

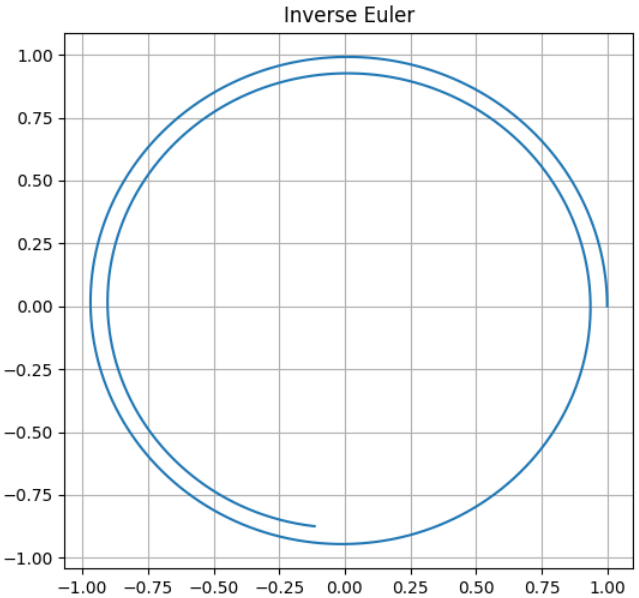
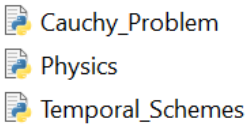


Figura 6. Resultados Euler Inverso para deltat = 0.005

Sucede algo parecido al Euler Explícito, la solución no es estable para ningún *time step,* pero si que se vuelve más exacta cuando este disminuye.

III. Código

En este segundo hito, se ha implementado la estructura Top-Down, organizando las funciones creadas en distintos archivos en base a su funcionalidad. Se ha creado así una carpeta de "Functions" donde se encuentran los distintos archivos que reúnen las funciones empleadas en este hito:



De esta forma, desde el código principal "Milestone2", se importan las funciones de los distintos archivos que se emplean en este hito.

En "Cauchy_Problem", se implementa la siguiente función para resolver un sistema de ecuaciones diferenciales partiendo de unas condiciones iniciales de la solución. Como *inputs,* se deben introducir: el esquema temporal escogido para resolver el sistema, la función F de Kepler, el tiempo y la condición inicial.

```
import numpy as np

def Cauchy_problem(Temporal_Scheme, F, t, U0):

    N = len(t) - 1
    N0 = len(U0)
    U = np.zeros((N+1, N0))

    U[0,:] = U0

    for i in range(N):
        #print(i)
        U[i+1,:] = Temporal_Scheme(U[i,:], t[i+1] - t[i], t[i], F)
        #print(U[i+1,:])

    return U
```

En "Physics", se encuentra la función de Kepler que permitirá resolver el problema planteado en este hito. Como *inputs,* se deben introducir: el vector de soluciones y el tiempo.

```
import numpy as np

def Kepler(U,t):

    F0 = U[2] # dxdt
    F1 = U[3] # dydt
    F2 = -U[0]/(U[0]**2 + U[1]**2)**(3/2) # -x/(x^2+y^2)3/2
    F3 = -U[1]/(U[0]**2 + U[1]**2)**(3/2) # -y/(x^2+y^2)3/2

    return np.array([F0, F1, F2, F3])
```

Por último, en "Temporal_Schemes", se hallan los distintos esquemas temporales con los que se resuelve el problema de Cauchy: Euler, Crank Nicolson, Runge-Kutta de orden 4 y Euler Inverso. Como *inputs,* se deben introducir: el vector estado en cada paso del tiempo, el deltat, la función dU/dt y el tiempo.

```python
from scipy.optimize import newton
from Functions.Physics import Kepler

"""

Functions for the temporal schemes
    Inputs:
        U = vector value at tn
        dt = time step
        F = function dU/dt
        t = tn

        Returns:
            U = vector value at (tn+dt)

"""

def Euler(U, dt, t, F_E):

    return (U + dt*F_E(U,t))

def Crank_Nicolson(U, dt, t, F ):
    def f_CN(X):

         return  X - (U + dt/2*F(U, t)) - dt/2*F(X, t+dt)

    return newton(f_CN, U)

def RK4(U, dt, t, F):

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

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

def Inverse_Euler(U, dt, t, F):
    def f_I(X):
        return X - U - dt*F(X,t)

    return newton(f_I, U)
```

Finalmente, el código para este segundo hito, “Milestone2”, se encarga de importar las distintas funciones empleadas, a partir de los códigos mostrados anteriormente, y definir las condiciones del problema: Nº pasos, tiempo final, deltat y condición inicial de la solución. Finalmente, representa las distintas soluciones para cada uno de los esquemas temporales obteniendo los resultados previamente expuestos.

```python
from Functions.Cauchy_Problem import Cauchy_problem
from Functions.Physics import Kepler
from Functions.Temporal_Schemes import Euler, Crank_Nicolson, RK4, Inverse_Euler
import matplotlib.pyplot as plt
import numpy as np


N = 2000 # Nº pasos
tf = 10 # valor final
dt = tf/N

# CONDICIONES INICIALES
r0 = np.array([1, 0])
v0 = np.array([0, 1])

U0 = np.hstack((r0,v0)) # Initial condition
print(U0)

t = np.linspace(0, tf, N)
print(t[0])



U_Euler = Cauchy_problem(Euler, Kepler, t, U0)

plt.figure(1)
plt.plot(U_Euler[:,0],U_Euler[:,1])
plt.title('Euler')
plt.grid(True)
plt.show()



U_Crank_Nicolson = Cauchy_problem(Crank_Nicolson, Kepler, t, U0)

plt.figure(2)
plt.plot(U_Crank_Nicolson[:,0],U_Crank_Nicolson[:,1])
plt.title('Crank Nicolson')
plt.grid(True)
plt.show()



U_RK4 = Cauchy_problem(RK4, Kepler, t, U0)

plt.figure(3)
plt.plot(U_RK4[:,0],U_RK4[:,1])
plt.title('Runge-Kutta 4')
plt.grid(True)
plt.show()


U_InverseEuler = Cauchy_problem(Inverse_Euler, Kepler, t, U0)

plt.figure(4)
plt.plot(U_InverseEuler[:,0],U_InverseEuler[:,1])
plt.title('Inverse Euler')
plt.grid(True)
plt.show()
```

Se obtiene así la siguiente estructura Top-Down seguida en este ejercicio: