


Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestone II

Ampliación de Matemáticas

Autor:

Miguel del Fresno Zarza

Índice

1. Introducción

Este informe presenta el trabajo realizado para completar el hito II, en le cual se pide obtener la órbita de kepler mediante los métodos de Euler, Euler inverso, Runge-Kutta de orden 4 y Crank Nicolson.

2. Explicación del código

Para el cálculo de la órbita de kepler se han desarrollado varias funciones que convergen en una función final para tener un código más organizado y entendible. Para ello se han creado una serie de archivos donde se encuentran las funciones.

- **Temporal_schemes.py**

  Este archivo recoge todas las funciones donde se desarrollan los métodos de Euler, Euler inverso, Runge-Kutta de orden 4 y Crank Nicolson.

  Dentro del código se desarrolla una función de Euler, Runge-Kutta de orden 4, Euler inverso y Crank Nicolson. De estas funciones cabde destacar que para el caso de Euler y Runge-Kutta de orden 4 se han desarrollado manualmente las fórmulas, mientras que para el caso de Euler Inverso y Crank Nicolson se ha recurrido a la función "newton"del módulo scipy.optimize para resolver los problemas.

```python
import numpy as np
from scipy.optimize import newton


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

    return U + dt * F(U, t)

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

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

        k = (k1 + 2*k2 + 2*k3 + k4)/6

        return U + k
```

```
20  def Euler_Inv(U, t, dt, F):
21
22      def Residual_EI(X):
23
24          return X - U - dt*F(X, t)
25
26      return newton(Residual_EI, U, tol = 1e-05)
27
28  def Crank_Nicolson(U, t, dt, F):
29
30      def Residual_CN(X):
31
32          return X - U - dt * (F(X, t+dt) + F(U, t)) / 2
33
34      return newton(Residual_CN, x0 = U, tol = 1e-05)
```

- **Kepler_problem.py**

  En este código se resuelve el problema de Kepler utilizando las fórmulas correspondientes. De esta manera, la función proporciona el vector F de velocidades y aceleraciones de la partícula en el instante n + 1.

```
1  import numpy as np
2
3
4  def Kepler(U, t):
5
6      x = U[0]; y = U[1]; dxdt = U[2]; dydt = U[3]
7      d = (x**2  + y**2)**(3/2)
8
9      return np.array( [ dxdt, dydt, -x/d, -y/d ] )
```

- **Cauchy_problem.py**

  El archivo de Cauchy establece un código en el que se resuelve el problema de la órbita con unas condiciones de entrada y se obtiene el vector de posición y velocidad en cada instante.

```python
import numpy as np


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

    N, Nv = len(t)-1, len(U0)    # Nº de intervalos y Nº de variables

    U = np.zeros((N+1, Nv))

    U[0,:] = U0

    for n in range(N):

        U[n+1,:] = Esquema_temp(U[n,:], t[n], t[n+1] - t[n], F)

    return U
```

- **Milestone_II.py**

  En este último archivo, se encuentra una serie de condiciones iniciales para poder resolver el problema. Seguidamente, se desarrolla un código en el que se resuelve el problema de Kepler con cada uno de los métodos utilizando las funciones de los anteriores archivos. De esta manera, se obtienen las gráficas de cada uno de los métodos.

```python
import numpy as np
from Cauchy_problem import Cauchy
from Kepler_problem import Kepler
from Temporal_schemes import Euler, RK4, Euler_Inv, Crank_Nicolson
import matplotlib.pyplot as plt

## Condiciones iniciales

U0 = [1, 0, 0, 1]

N = 1000
t_f = 10

t = np.linspace(0, t_f, N+1)


```

```
## Esquemas

esquema = [Euler, RK4, Euler_Inv, Crank_Nicolson]
titulo = ["Euler", "Runge-Kutta 4", "Inverse Euler", "Crank-Nicholson"]


def Simulation(t, U0, esquema, titulo):

    i = 0

    for scheme in esquema:

        U = Cauchy(Kepler, t, U0, scheme)

        plt.plot(U[:,0],U[:,1])
        plt.title(titulo[i])
        plt.show()

        i = i+1

Simulation(t, U0, esquema, titulo)
```

3. Resultados

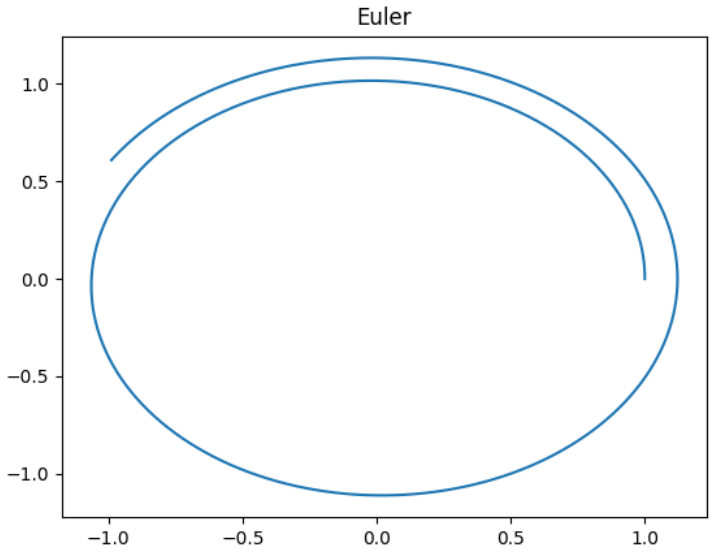
A continuación se muestran los resultados para cada uno de los métodos.

3.1. Métodos explícitos

Los métodos explícitos se caracterizan por obtener las soluciones en un instante n + 1 a partir de la información en el instante n.

Euler

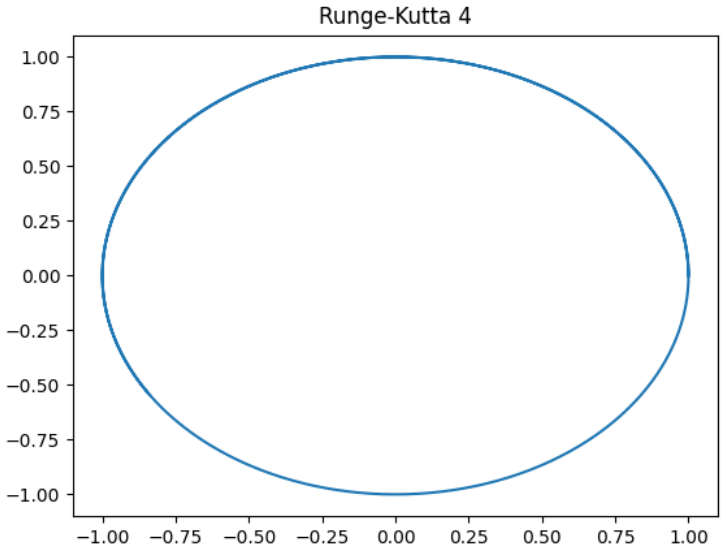
A continuación se muestran los resultados para Euler explícito.



Como se puede comprobar, el método de Euler explícito genera un cierto error con el paso del tiempo al no conservar la energía. Este error se minimiza haciendo más pequeño el incremento del tiempo para cada instante de n.

Runge-Kutta orden 4

A continuación se muestran los resultados para Runge Kutta de orden 4.



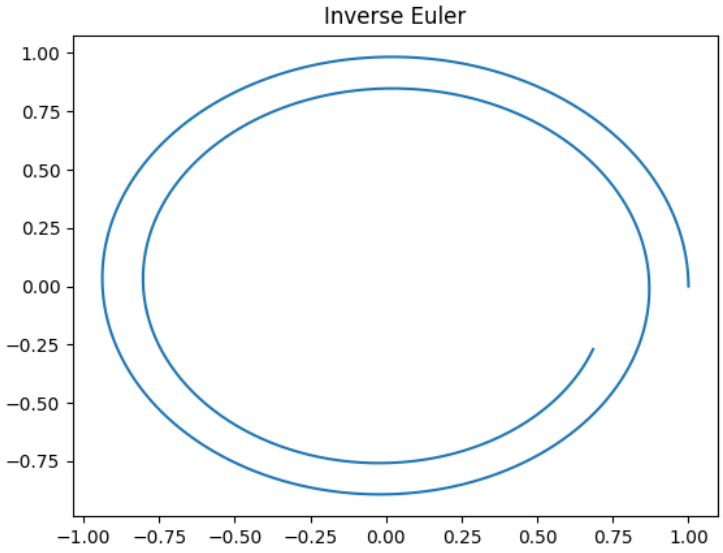
A diferencia del anterior método, este genera mucho menor error, aunque requiere de un coste computacional mayor.

3.2. Métodos implícitos

Los métodos implícitos se caracterizan por obtener las soluciones para los instantes n + 1 partiendo de una serie de variables en ese mismo instante del tiempo, por lo que requiere de un proceso iterativo que incrementa mucho el coste computacional.

Euler inverso

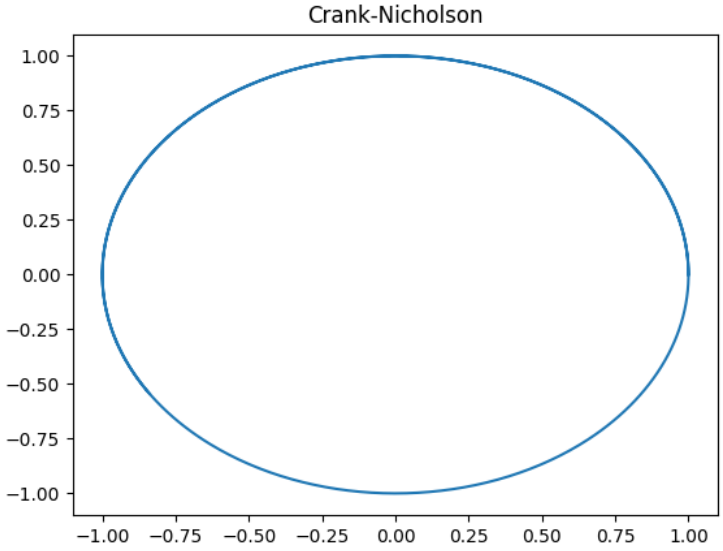
A continuación se muestran los resultados para Euler inverso.



De la misma manera que el método explícito, este también genera error al no conservar la energía.

Crank Nicolson

A continuación se muestran los resultados para Crank Nicolson.



Comparando con el resto de métodos, el de Crank Nicolson es el que menor error genera de todos.

4. Conclusiones

Mediante este ejercicio, se ha podido transformar un código que queda desarrollado en un único archivo a un solo archivo donde todo el código queda recogido mediante funciones. De esta manera todas las líneas de código quedan mucho más ordenadas y se puede localizar más fácilmente cualquier error.

Por otra parte, se ha desarrollado cada uno de los métodos para resolver el problema de Kepler y se ha podido ver las características generales de cada uno de los métodos.