Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milstone 2

Ampliación de Matemáticas I

5 de octubre de 2023

Autor

Domínguez Gómez, Sandra

Índice

1. Introducción **1**

2. Descripción de los módulos **2**

2.1. *Milstone_2.py* . . . 2

2.2. Problema de Cauchy . . . 4

2.3. Esquemas numéricos temporales . . . 5

2.4. Función de Kepler . . . 6

3. Resultados y análisis **8**

3.1. Métodos explícitos . . . 8

3.1.1. Euler Explícito . . . 8

3.1.2. Runge Kutta 4 . . . 9

3.2. Métodos implícitos . . . 9

3.2.1. Crank Nicolson . . . 10

3.2.2. Euler Implícito . . . 11

Índice de figuras

2.1. Importación de librerías y funciones. 2

2.2. Condicones inciales y declaración de variables. 2

2.3. Vector para distintas Δt. 3

2.4. Bucles para obtener soluciones de los esquemas temporales. 3

2.5. Plotear las soluciones para los distintos esquemas temporales. 4

2.6. Problema de Cauchy. 5

2.7. Esquemas temporales. 6

2.8. Función de Kepler. 7

3.1. Euler Explícito. 8

3.2. Runge Kutta 4. 9

3.3. Crank Nicolson. 10

3.4. Euler Implícito. 11

1. Introducción

El objetivo de este apartado es explicar cómo se encuentra estructurado el código en el repositorio *GitHub* para poder ejecutarlo de manera correcta.

El código se encuentra estructurado de la siguiente manera:

- Milstone2.py: Este archivo es considerado como p*main*. Está ubicado en la carpeta "*sources*"que contiene todos los módulos necesario a implementar. En dicho código se definen las condiciones iniciales y se hace uso de las herramientas de Python para realizar el problema numérico. También contiene el código necesario que permitirá representar las soluciones de los esquemas numéricos empleados.
- Funciones.py:La función que podrá resolver el problema temporal.
- Problema_Cauchy.py: Es el problema que podrá resolver los esquemas temporales.
- Esquemas_Temporales.py: Esta función engloba los diferentes esquemas numéricos temporales solicitados para este trabajo: Euler explícito, Euler Implícito, Crank Nicolson y Runge Kutta 4.

A continuación, se detallará el desarrollo del código y los resultados de las simulaciones.

2. Descripción de los módulos

2.1. *Milstone_ 2.py*

Este archivo contiene la información principal del código completo. Para empezar, es necesario llamar a todas las librerías de Python necesarias para emplear el lenguaje matemático. Principalmente serán *numpy* que permite definir matrices y vectores y *matplotlib.pyplot* para representar gráficamente las soluciones con los diferentes esquemas numéricos.

También se debe llamar a todos las funciones que se encuentran mencionadas en la Sección 1.

```
from numpy import array, linspace
from Methods.Esquemas_Temporales import *
from Methods.Problema_Cauchy import P_C
from Problems.Funciones import F_Kepler
import matplotlib.pyplot as plt
```

Figura 2.1: Importación de librerías y funciones.

Para comenzar, es necesario definir los datos y/o parámetros iniciales. Para ello, se ha creado una lista: N, que son las divisiones del tiempo total, T. Es creada con el objetivo de poder ver como se comporta el modelo al variar los Δt.

Por otro lado, se va a emplear el instrumento diccionario que es una herramienta de Python con el objetivo de poder almacenar matrices con diferentes dimensiones al cual hay que darle una palabra clave (*key*, de tipo *string*) que permita llamarla cuando se necesite utilizar. En este caso, los diccionarios que se han creado son: t para almacenar los diferentes saltos de tiempo desde 0 hasta T (t_1, t_2, t_3, ..., t_n) y que se generará creando un bucle para que considere la lista N, U_Euler, U_RK4, U_CN, y U_In_Euler, que almacenarán los diferentes casos.

```
## Datos iniciales
N = [1000, 10000, 100000]
U_0 = array([1, 0, 0, 1])
#Establecer un tiempo fijo, T, que queremos operar.
T = 20

#Inicializar el diccionario.
t = {}
U_Euler = {}
U_RK4 = {}
U_CN = {}
U_In_Euler = {}
```

Figura 2.2: Condicones inciales y declaración de variables.

```
for i in N:
    t[str(i)] = linspace(0,T,i+1)
```

Figura 2.3: Vector para distintas Δt.

Tras haber inicializado y creado las variables necesarias, es el momento de resolver el problema . Para ello se han creado 4 bucles que puedan resolver la Función de Kepler en cada esquema numérico solicitado.

```
for key in t:

    U_Euler[str(key)] = P_C( U_0, t[key], F_Kepler, Euler)

print('Euler done')

for key in t:
    U_RK4[str(key)] = P_C( U_0, t[key], F_Kepler, RK4)

print('RK4 done')


for key in t:
    U_CN[str(key)] = P_C( U_0, t[key], F_Kepler, CN)

print('CN done')


for key in t:
    U_In_Euler[str(key)] = P_C( U_0, t[key], F_Kepler, In_Euler)
```

Figura 2.4: Bucles para obtener soluciones de los esquemas temporales.

Estos bucles funcionan de la siguiente manera: se define el esquema temporal deseado donde va a llamar al problema de Cauchy (P_C). Esta función tomará como datos (para poder realizar las operaciones): el vector U_0, la t[key] (que permite obtener las soluciones para diferentes vectores t que fueron creados previamente), la función a utilizar en este caso que es la función de Kepler y, por último, el esquema numérico correspondiente.

Por último, sería la representación gráfica de las soluciones.

```
ET_plots = ['Euler_plt', 'RK4_plt', 'CN_plt', 'Inv_Euler_plt'] #  Lista de los nombre de los futuros objetos de plotear que voy a crear
U_plots = [U_Euler, U_RK4, U_CN, U_In_Euler] # lista con los resultados
Titles = ['Euler Explícito', 'Runge Kutta 4', 'Crank Nicolson', 'Inversa de Euler'] # lista de los títulos

i = 0
for pintar in ET_plots:
    fig, pintar = plt.subplots(figsize = (4,4)) # Creo el objeto desde la lista
    # U = U_plots[]
    for key in t:
        dt = t[key][2] - t[key][1] # Se hace concatenacion. Es de la key en la que estés la posicon 2 - la 1.
        pintar.plot(U_plots[i][key][0, : ], U_plots[i][key][1, : ], label = "dt = " + str(dt))

    pintar.set_xlabel('x')
    pintar.set_ylabel('y')
    pintar.set_title(Titles[i])
    pintar.legend()
    i += 1

plt.show()
```

Figura 2.5: Plotear las soluciones para los distintos esquemas temporales.

2.2. Problema de Cauchy

El problema de Cauchy se encuentra dentro de la carpeta *Methods*. Dicha función tiene como parámetros de entreada: el vector de condiciones iniciales (U_0), el vector t (definido previamente en Milstone_2.py), la función a emplear F (que será la función de Kepler) y el esquema numérico temporal (ET).

Se observa que el código se ha escrito de la manera más genérica posible con el objetivo de poderlo aplicar a diferentes casos.

La función comienza con la inicialización de variables. Es necesario crear una N, dentro de la propia función que considere los diferentes t_n; definir el valor inicial con U; y, por último, crear un Δt que será la diferencia entre dos instantes temporales. A continuación, se realiza la resolución del problema de Cauchy con el esquema numérico temporal correspondiente.

```python
from numpy import zeros, array

# P_C= Problema de Cauchy

def P_C( U_0, t, F, ET ):
    # U_0 vector inicializador, t= tiempo a estudiar,
    # F= es la funcion, ET= esquema temporal

    ## Inicializacion de variables
    N = len(t) - 1 # Se define aquí con la t (tiempo), donde t-1.


    U = zeros( (len(U_0), N+1)) # La longitud de U_0 para que sea generalista
    U[:, 0] = U_0 # definimos el valor inicial

    dt = t[2] - t[1]
    for i in range (N):

        U[:, i + 1] = ET(dt, U[:,i], F)

    return U
```

Figura 2.6: Problema de Cauchy.

2.3. Esquemas numéricos temporales

Los esquemas temporales han sido creados en un *script* aparte. En la Figura 2.7 se muestran. Para los esquemas implícitos, es necesario realizar un fsolve, que se ha importado de *mathplot.py*.

```python
from scipy.optimize import fsolve

#EULER
def Euler(dt,U,F):

    return U + dt * F(U)

# RUNGE KUTTA 4

def RK4(dt,U,F):

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


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

# CRANCK NICOLSON
def CN( dt, U,F):

    def CN_res(x):

        return x - U_temp -dt/2 * F(x)

    U_temp = U + dt/2 * F(U)

    return  fsolve(CN_res,U)

#EULER IMPLICITO
def In_Euler(dt, U, F):

    def Euler_res(x):

        return x - U - dt*F(x)

    return fsolve(Euler_res, U)
```

Figura 2.7: Esquemas temporales.

2.4. Función de Kepler

En la imagen Figura 2.8 se puede observar el código de la función de Kepler.

```python
from numpy import array


##FUNCION DE KEPLER

def F_Kepler(U):

    x, y, vx, vy = U[0], U[1], U[2], U[3]
    mr = (x**2 + y**2 )**1.5

    return array( [vx, vy, -x/mr, -y/mr])
```

Figura 2.8: Función de Kepler.

La ventaja de crear módulos por separado es que el código queda más simplificado y genérico.

3. Resultados y análisis

3.1. Métodos explícitos

3.1.1. Euler Explícito

Como se puede observar en la Figura 3.1, cuando el Δt aumenta, el problema diverge, mientras que cuando disminuye el problema converge. El motivo de este resultado es debido al error que se comete a la hora de aplicar el esquema temporal.

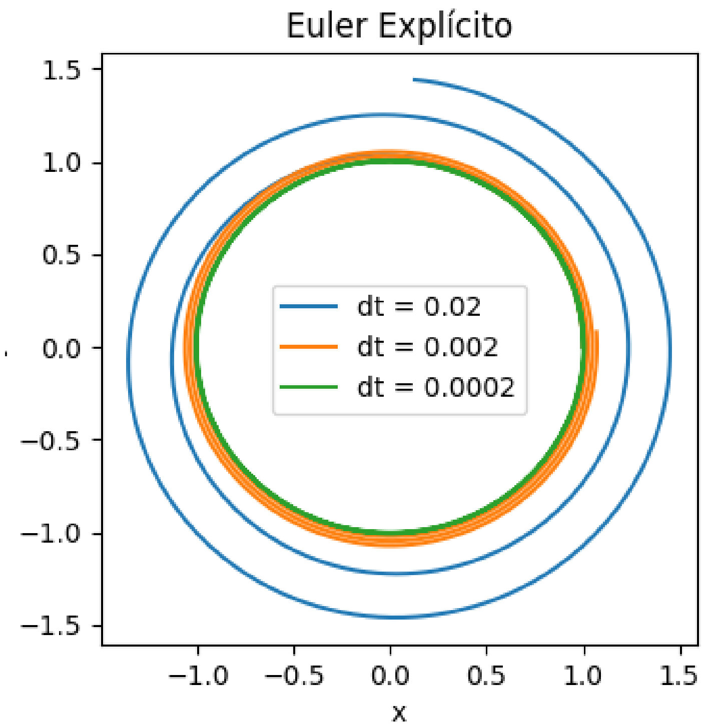

Figura 3.1: Euler Explícito.

3.1.2. Runge Kutta 4

Como se aprecia en la Figura 3.2, este esquema numérico presenta una gran precision para cualquiera de sus Δt con un tiempo de procesamiento muy pequeño.

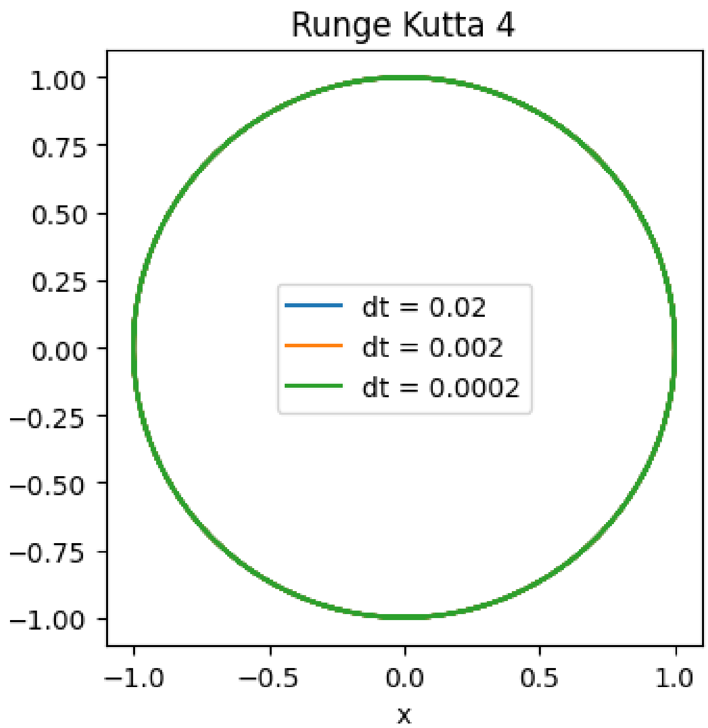

Figura 3.2: Runge Kutta 4.

3.2. Métodos implícitos

Estos resultados, tienen como inconveniente que los tiempos de resolución son más largo en comparación con los esquemas temporales explícitos debido a que hay que resolver ecuaciones implícitas y el coste computacional es mucho mayor.

3.2.1. Crank Nicolson

Este esquema produce resultados de gran precisión, como se puede observar en la Figura 3.3.

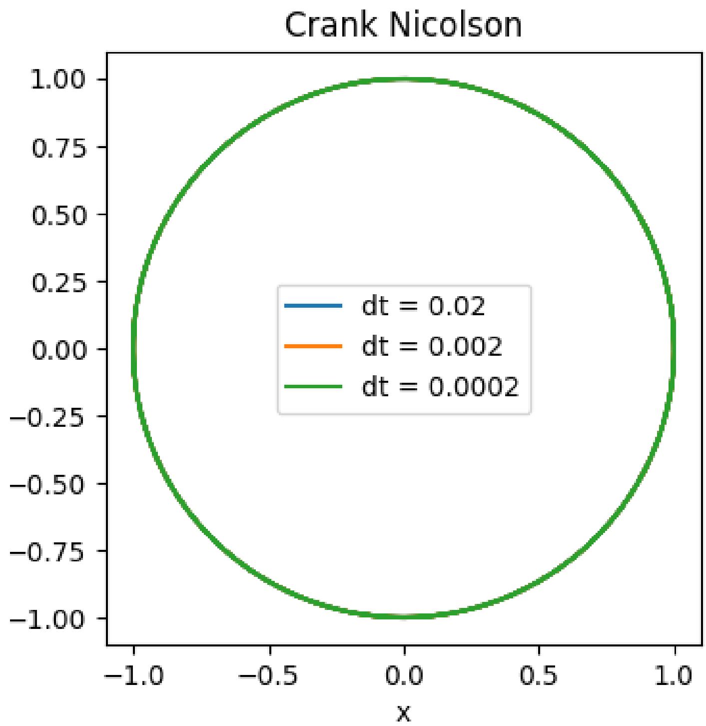

Figura 3.3: Crank Nicolson.

3.2.2. Euler Implícito

Por último, la Figura 3.4 representa los resultados del esquema numérico de Euler Implícito. Los resultados son muy similares a los que se obtienen de Euler Explícito, divergencia en función de Δt, pero sentido inverso si se observa la espiral.

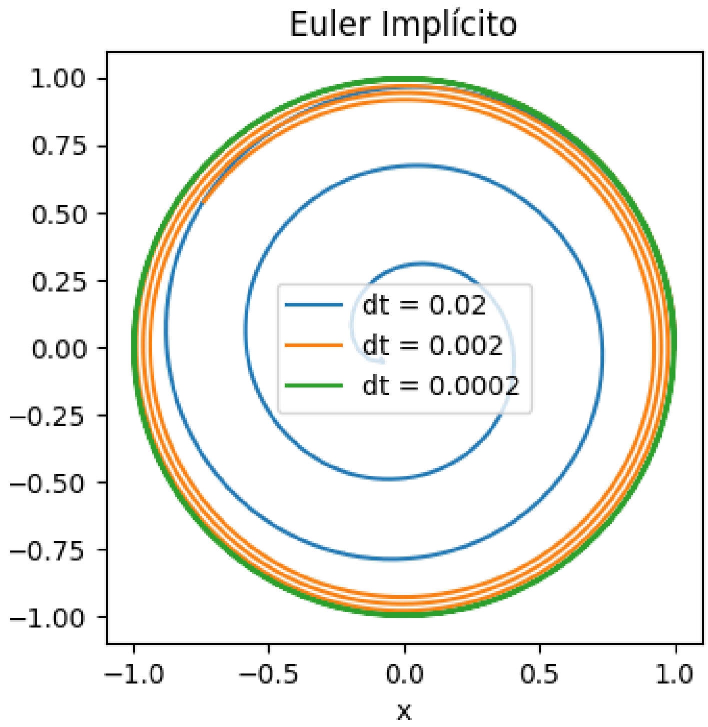

Figura 3.4: Euler Implícito.