



Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestone 2

Ampliación de Matemáticas I

2 de octubre de 2022

Autor:

Sergio de Ávila Cabral

Índice

1. Introducción	1
2. Descripción de módulos	1
2.1. Main.py	1
2.2. Physics.py, Temporal_Schemes.py, Cauchy_Problem.py	2
2.3. Plots.py	3
3. Resultados	3
3.1. Resultados Euler	3
3.2. Resultados Runge-Kutta orden 4	4
3.3. Resultados Crank-Nicolson	5
3.4. Resultados Euler inverso	5

1. Introducción

Para la programación de este *Milestone* se ha tenido en cuenta la metodología *Top-Down*. Para ello se han definido los siguientes módulos:

- Main.py: Actúa como *main*. En él se encuentran importados todos los módulos siguientes. A su vez, se definen las condiciones iniciales del problema, las iteraciones, diccionarios y la llamada al problema de Cauchy.
- Physics.py: Incluye la física del problema a resolver.
- Temporal_Schemes.py: Se incluyen los diferentes esquemas numéricos.
- Cauchy_Problem.py: En él se encuentra la definición del problema de Cauchy.

Para todas las simulaciones se han realizado un conjunto de tres N iteraciones y tres Δt pasos.

2. Descripción de módulos

2.1. Main.py

Como se ha descrito en la Sección 1, actúa como *main*. En él se pueden apreciar diferentes secciones entre ellas:

- Condiciones iniciales: inicializa las condiciones del problema y el número de iteraciones deseadas.
- Save Plots: se trata de un *boolean* que permite elegir si los *plots* se guardan o solo se muestran.
- Esquemas numéricos: Se indican los esquemas numéricos que se van a emplear.
- Inicialización de Diccionarios: Inicialización de los diccionarios que se emplearán.
- Tiempos: Definición de los tiempos para las diferentes simulaciones.

```
#Times for simulations
for i in range(len(N)):
    t_dic[str(i)] = linspace(0,20,N[i])
```

Puesto que se definen diferentes *linspace* con diferentes dimensiones, se introducen de forma dinámica en el diccionario, creando nuevas *Keys* que serán empleadas posteriormente.

- Simulaciones: Se realizan las diferentes simulaciones a través del problema de Cauchy.

```
#Simulations
for j in range(len(T_S_plot)):
    for x in t_dic: # x is a str and U is creating new keys
        U_dic[x] = C_P(Kepler, t_dic[x], U0, T_S[j])
        print(T_S_plot[j] + " calculado \n")
        Plot_CP(U_dic[x], t_dic[x], T_S_plot[j], Save)
```

Al igual que en el anterior diccionario, se definen nuevos *Keys* de forma dinámica. Como se puede apreciar, las *Keys* de ambos diccionarios están sincronizados. Esto se realiza para simplificar los *plots*. Del Mismo modo, se puede apreciar que se realiza la llamada a dos funciones:

- *C_P*: en la cual se introducen los diferentes esquemas numéricos, el problema físico, los tiempos y las condiciones iniciales.
- *Plot_CP*: a la cual introducen los resultados junto con el esquema introducido y el *boolean* para guardar o no las imágenes.

2.2. Physics.py, Temporal_Schemes.py, Cauchy_Problem.py

En los módulos que tratan los siguientes subapartados, se ha procedido a reducir el numero de variables empleadas como se indicación en el *Issue* del *Milestone 1*

Physics.py

Aplicando la metodología *Top-Down* explicada en clase, el problema físico se define en un módulo aparte y de manera más simplificada que en el *Milestone 1*

Temporal_Schemes.py

Todos los esquemas numéricos aplicados en el *Milestone 1* se introducen de forma más simplificada, introduciendo como *input* de las funciones la variable *F*, que será el problema físico que se querrá resolver. Ejemplo de ello es el esquema numérico *Runge-Kutta* de cuarto orden:

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

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

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

Cauchy_Problem.py

Para realizar el problema de Cauchy se definen el número de iteraciones *N*, el *array* *U* y el $\Delta t = t_{i+1} - t_i$.

```
def C_P(F, t, U0,T_S):

    N = len(t)-1 #Because of the divisions of the time's linspace
    U = zeros( (len(U0), N+1) )
    U[:,0] = U0
```

```

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

return U

```

2.3. Plots.py

En el módulo Plots se definen las funciones *Plot_CP*, grafica una única función; y *Plot_CP_all*, grafica todas las funciones relacionadas con un mismo esquema numérico. Estas funciones están concebidas para seguir la metodología *Top-Down* y simplificar el *main*. En estas funciones se definen los elementos básicos para graficar los resultados y se añade un condicional para guardar los archivos.

```

if not Save:
    plt.show()

else:
    ##Save Plots
    file = T_S + "_dt_" + str(dt) + ".png"
    path = os.path.join(os.getcwd(), 'Plots')

    if not os.path.exists(path):
        os.makedirs(path)

    plt.savefig(os.path.join(path, file))

```

Como se puede ver en estas líneas de código, se crea un condicional para el *boolean Save*. Si este es falso, solo enseña las gráficas, pero si es cierto comienza una serie de sentencias para guardar. Esta serie de sentencias crea, en el caso de no haber, una carpeta "*Plots*" en la misma localización que el *main*. De esta forma este modulo es utilizable para cualquier otro *main*.

3. Resultados

A continuación se presenta los resultados obtenidos para los diferentes esquemas numéricos.

3.1. Resultados Euler

Como se puede apreciar en la Figura 1, se aprecia una fuerte divergencia para $\Delta t \approx 0,01$, reduciéndose de forma considerable dicha divergencia para $\Delta t \approx 0,0001$.

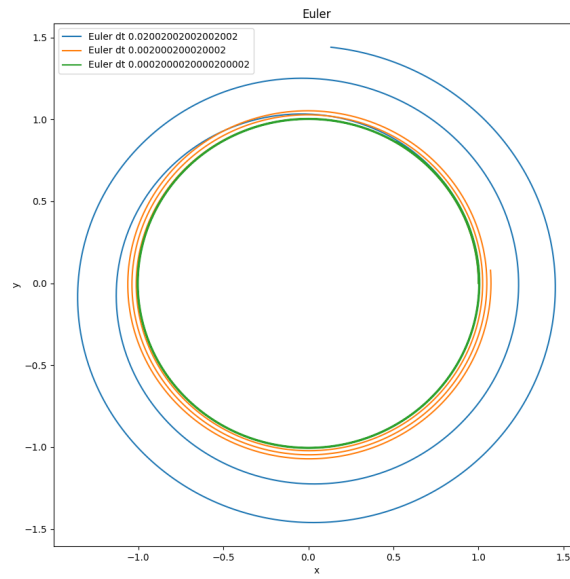


Figura 1: Resultados de Euler para diferentes Δt

3.2. Resultados Runge-Kutta orden 4

Como se puede apreciar en la Figura 2, este esquema numérico tiene una gran precisión para cualquiera de sus Δt con un tiempo de procesamiento muy pequeño.

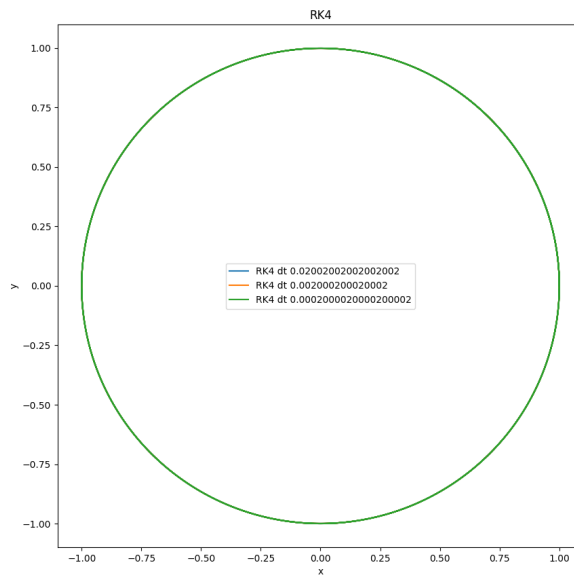


Figura 2: Resultado de RK4 para diferentes Δt

3.3. Resultados Crank-Nicolson

[] Este esquema produce resultados de gran precisión, como se puede observar en la Figura 3. En comparación con el anterior esquema, resulta de un procesamiento más lento

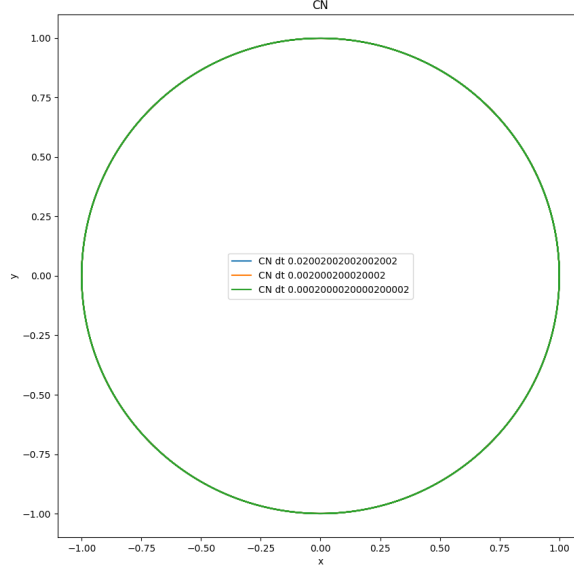


Figura 3: Resultados de Crank-Nicolson para diferentes Δt

En el caso de emplear como función una integración numérica de Newton, el tiempo de procesamiento se ve muy incrementado, de forma notable a partir de $\Delta t \approx 0,0001$, produciendo resultados análogos, como los mostrados en la Figura 4.

3.4. Resultados Euler inverso

Por último, se muestran los resultados del esquema numérico de Euler Inverso. Este esquema devuelve resultados análogos a los mostrados en la Sección 3.1, con divergencia en función de la Δt , pero de sentido inverso, como se puede apreciar en la espiral. Este resultado en la Figura 5.

Del mismo modo que en la Sección 3.3, se procede a realizar una integración numérica de Newton, obteniéndose resultados análogos, a excepción del resultado de $\Delta t \approx 0,01$. Como se puede apreciar en la Figura 6, hay una fuerte divergencia del resultado numérico con respecto al resultado exacto.

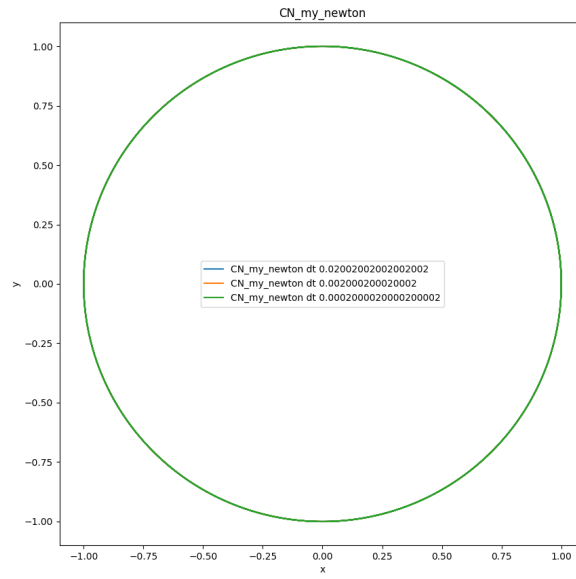


Figura 4: Resultados de Crank-Nicolson con función de Newton

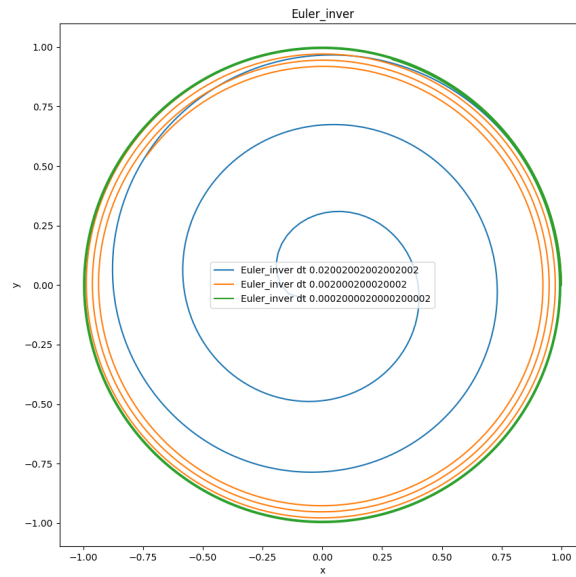


Figura 5: Resultados de Euler inverso para diferentes Δt

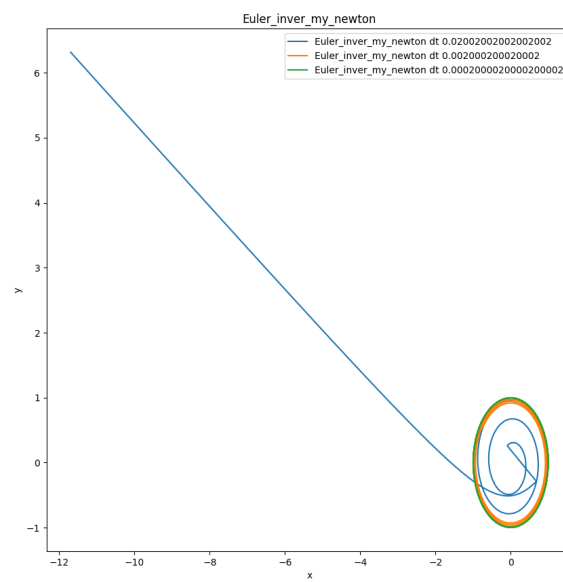


Figura 6: Resultados de Euler inverso con función de Newton