



Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

# Análisis del Hito 2

**Ampliación de Matemáticas I**

**5 de octubre de 2022**

**Autor:**

- Rafael Rivero de Nicolás

# Índice

# 1. Introducción

Se recogen en este documento los resultados obtenidos en el Problema de Cauchy dado por la ecuación diferencial ordinaria de segundo orden

$$\frac{\mathrm{d}^2\vec{\mathbf{r}}}{\mathrm{d}t^2} = -\frac{\vec{\mathbf{r}}}{\|\vec{\mathbf{r}}\|^3}, \tag{1}$$

y las condiciones iniciales

$$\vec{\mathbf{r}}(t=0) = \vec{\mathbf{r}}_o = (1,0), \qquad \left.\frac{\mathrm{d}\vec{\mathbf{r}}}{\mathrm{d}t}\right|_{t=0} = \dot{\vec{\mathbf{r}}}_o = (0,1),$$

tras implementar distintos esquemas numéricos en Python. Para ello se han empleado distintas mallas temporales, abarcando todas ellas un tiempo adimensional $t \in [0, 20]$.

# 2. Planteamiento del Código

Para resolver el problema se escribe el problema de Cauchy como

$$\frac{\mathrm{d}U}{\mathrm{d}t} = F(U,t), \tag{2}$$

la cual se ha implementado en Python como se recoge en el Extracto de código 1.

```
def Kepler_Orbits_2N(X, t):
    '''
    This function only depends on the physics of the problem, it musts be
    an input argument

    Parameters
    ----------
    X : Array
        State vector of the system in instant t.
    t : Float
        Time instant in which F is being evaluated.

    Returns
    -------
    Array
        First derivate of the tate vector. dU/dt = F(U,t).

    '''
    F1 = X[2]

    F2 = X[3]

    F3 = -X[0]/(X[0]**2 + X[1]**2)**(3/2)

    F4 = -X[1]/(X[0]**2 + X[1]**2)**(3/2)

    return np.array([F1, F2, F3, F4])
```

Extracto de código 1: Función del problema de Kepler implementada en Python.

Esta función es la que caracteriza la ecuación diferencial ordinaria que se pretende resolver, por lo que debe ser un input del programa.

Otros parámetros que deben ser introducidos en el programa, como las condiciones iniciales, se recogen en el Extracto de código 2, siendo `scheme` el esquema numérico empelado, `Delta_t` la lista que engloba todos los $\Delta t$ con los que se van a simular y el tiempo final que marcará la finalización de la simulación `tf`.

```
# %% Initialitation

Temoral_schemes_available = {0:"Euler",
                             1:"Inverse Euler",
                             2:"RK4",
                             3:"Crank-Nicolson"}

scheme = Temoral_schemes_available[2]

r_0 = np.array([1, 0]) # Initial position

v_0 = np.array([0, 1]) # Initial velocity

U_0 = np.hstack((r_0,v_0))

print('Initial State Vector: U_0 = ', U_0, '\n\n\n')

tf = 20

Delta_t = [0.2, 0.1, 0.01, 0.001]   # dt for different simulations
```

Extracto de código 2: Incialización del código Python para el Hito 2.

Tras la inicialización de las variables necesarias que establecen todos los argumentos de entrada para la función `Cauchy_Problem` se procede con el siguiente bucle, recogido en el Extracto de código 3, en el que se llama a esta función, que a su vez llama al esquema numérico previamente seleccionado.

```
# %%

U = {}

for dt in Delta_t:

    N = int( tf/dt )

    time_domain = np.linspace( 0, tf, N+1 )

    print('Temporal partition used dt = ', str(dt))

    U[scheme+'__dt=' + str(dt)] = mth.Cauchy_Problem( Kepler_Orbits_2N,
   U_0, time_domain, Temporal_scheme = scheme )

    print('\n\n\n')
```

Extracto de código 3: Bucle para la simulación del Hito 2.

### 2.1. Módulos necesarios

Para el funcionamiento del programa se emplean dos módulos desarrollados completamente con la intención de servir como librerías, tanto de funciones puramente matemáticas como de esquemas numéricos. Se consigue así un código compartimentado que permite la aplicación de un estilo de programación más funcional.

El módulo `LB_Math_Functions`, donde se encuentra el solver del problema de Cauchy, hará uso del módulo `LB_Temporal_Schemes`.

```
import numpy as np
# import LB_Temporal_Schemes as ts # Users module
import LB_Math_Functions as mth # Users module
```

Extracto de código 4: Módulos necesarios para el funcionamiento del Hito 2.

## 3. Resultados

Para la resolución de los sistemas implícitos de ecuaciones resultantes de la aplicación de los esquemas numéricos de Euler inverso y Crank-Nicolson se ha empleado una función propia basada en el cálculo del Jacobiano, tal y como se ha procedido en [2] .

### 3.1. Esquema: Euler

Como se aprecia en la Figura 1, independientemente del $\Delta t$ empleado las órbitas resultantes de la aplicación de este esquema numérico para el Problema de Cauchy presentado son espirales en las que $\|\vec{\mathbf{r}}\| \to \infty$ y $\left\|\dot{\vec{\mathbf{r}}}\right\| \to \infty$ para $t \to \infty$.

### 3.2. Esquema: Euler inverso

Como se aprecia en la Figura 2, pasa lo contrario que en el esquema Euler, ya que independientemente del $\Delta t$ empleado las órbitas resultantes de la aplicación de este esquema numérico para el Problema de Cauchy presentado son espirales en las que la energía del sistema va disminuyendo, haciendo que el objeto que orbita el cuerpo central acabe colapsando contra él.

Cabe mencionar que una vez el objeto está muy próximo al cuerpo central, el algoritmo da unos resultados erróneos pues se produce un cambio repentino y se generan trayectorias rectas debidas a que el sistema de ecuaciones implícito con $\Delta t$ aun relativamente pequeños no es capaz de reproducir la colisión.

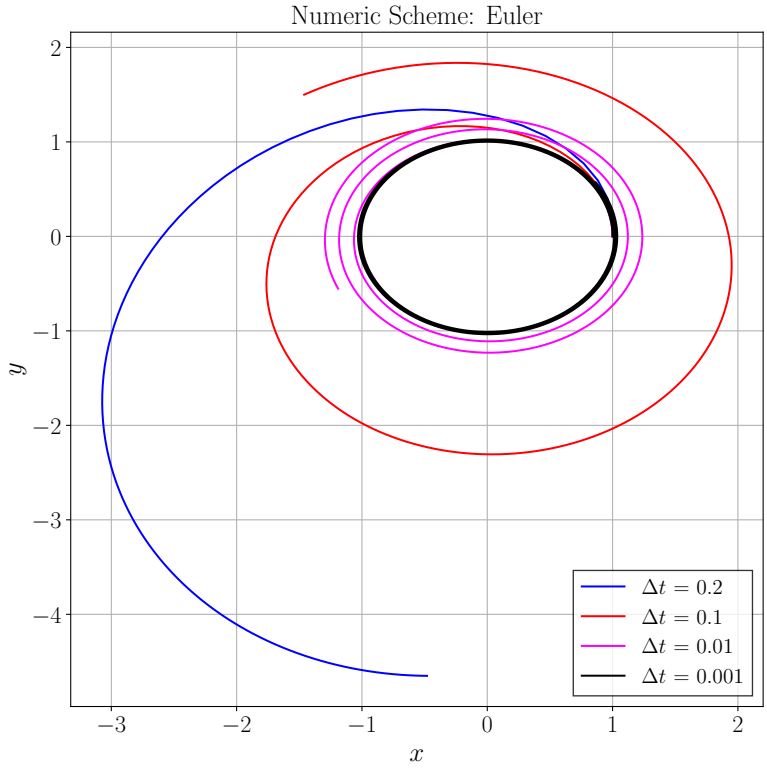


Figura 1: Órbitas calculadas mediante un esquema Euler.

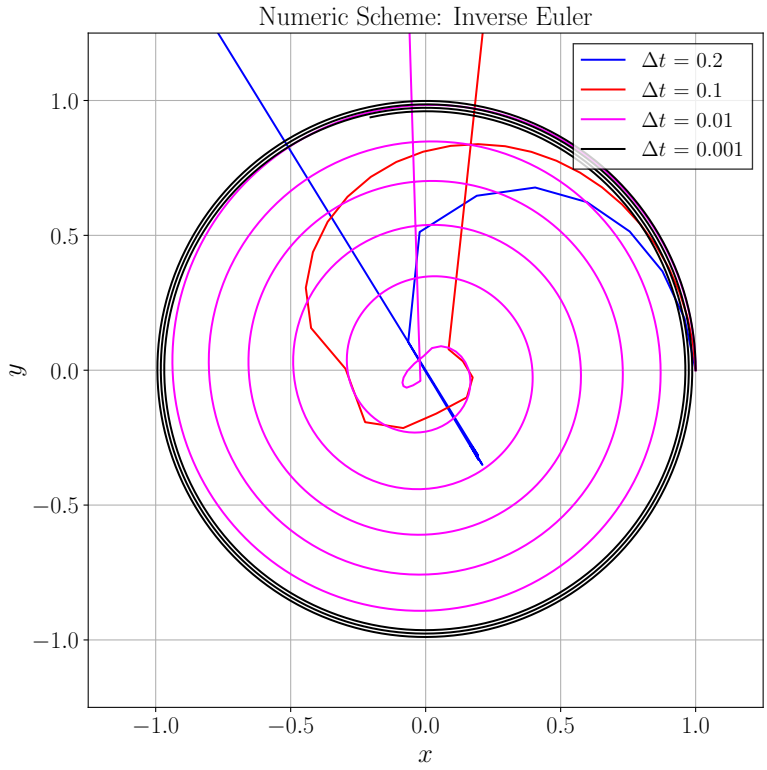


Figura 2: Órbitas calculadas mediante un esquema Euler inverso.

## 3.3. Esquema: Runge Kutta dde 4º Orden

En la Figura 3 se recogen las órbitas simuladas con este este esquema para distintos $\Delta t$. Incluso para el más pequeño de ellos, de un 1 % del valor del tiempo final de simulación, la órbita simulada prácticamente no difiere de la analítica, representada como una circunferencia de color negro de radio unidad en torno al $(0, 0)$.

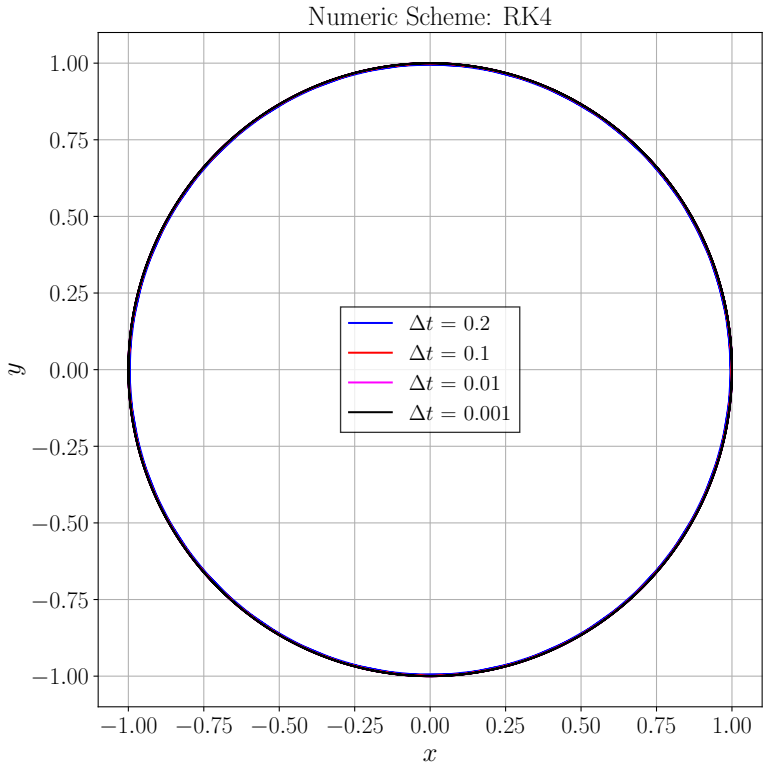


Figura 3: Órbitas calculadas mediante un esquema Runge Kutta de 4º orden.

Como hemos visto en clase, el orden de este esquema es mayor que el de Euler, por lo que su velocidad de convergencia es mucho mayor, justificando esto la solución tan precisa que se consigue.

## 3.4. Esquema: Crank-Nicolson

Observando la Figura 4 se llega a la misma conclusión que con el esquema anterior, las órbitas con este esquema son muy fieles a la solución analítica incluso con $\Delta t$ en el límite de lo que se podría considerar pequeño frente al tiempo máximo de simulación $t = 20$.

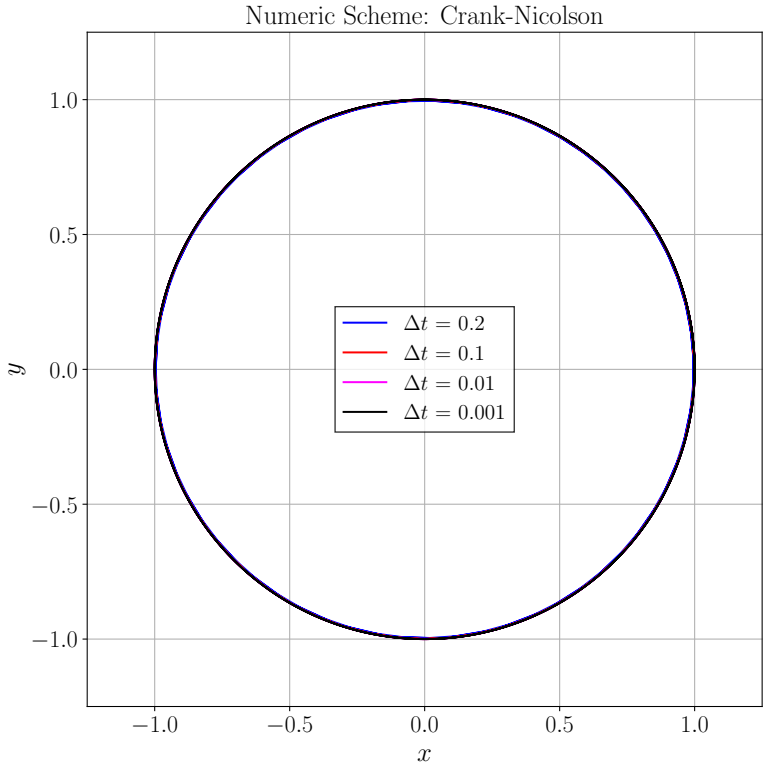


Figura 4: Órbitas calculadas mediante un esquema Crank-Nicolson.

# 4. Órbitas elípticas

Como curiosidad, para encontrar diferencias apreciables entre los esquemas Runge Kutta de 4º orden y de Crank-Nicolson se ha analizado también el problema de Cauchy dado por 1 y las condiciones iniciales

$$\vec{\mathbf{r}}(t=0) = \vec{\mathbf{r}}_o = (1{,}9, 0), \qquad \left.\frac{\mathrm{d}\vec{\mathbf{r}}}{\mathrm{d}t}\right|_{t=0} = \dot{\vec{\mathbf{r}}}_o = (0, 1),$$

para un tiempo adimensional de simulación $t = 500$, ya que la órbita resultante será elíptica y de gran excentricidad.

Se observa en las Figuras presentadas a continuación que el esquema Runge Kutta de 4º orden reúne unos resultados muy similares independientemente del $\Delta t$ empleado, mientras que el esquema Crank-Nicolson es menos robusto, además de computacionalmente más exigente.

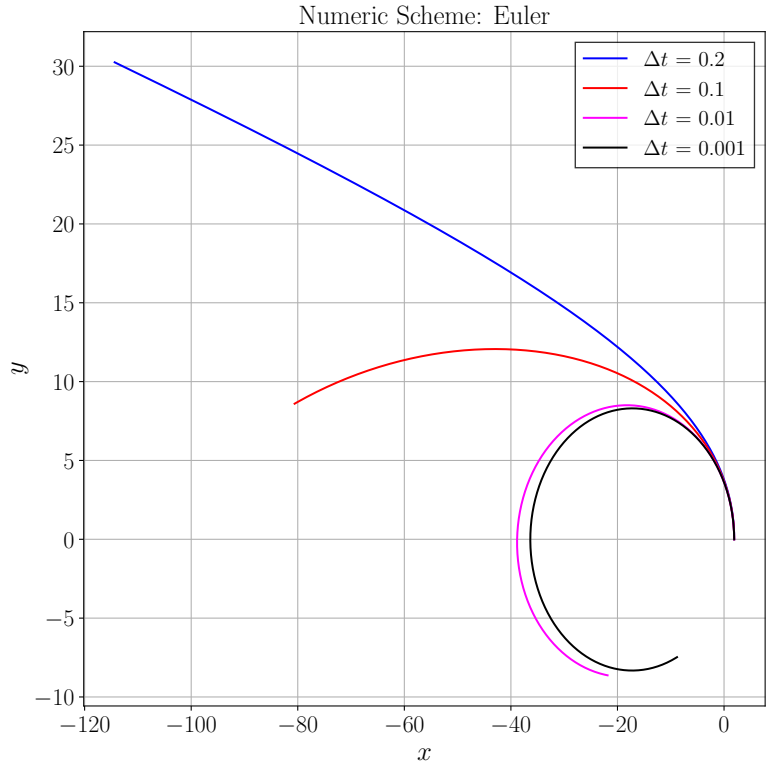


Figura 5: Órbitas calculadas mediante un esquema Euler.

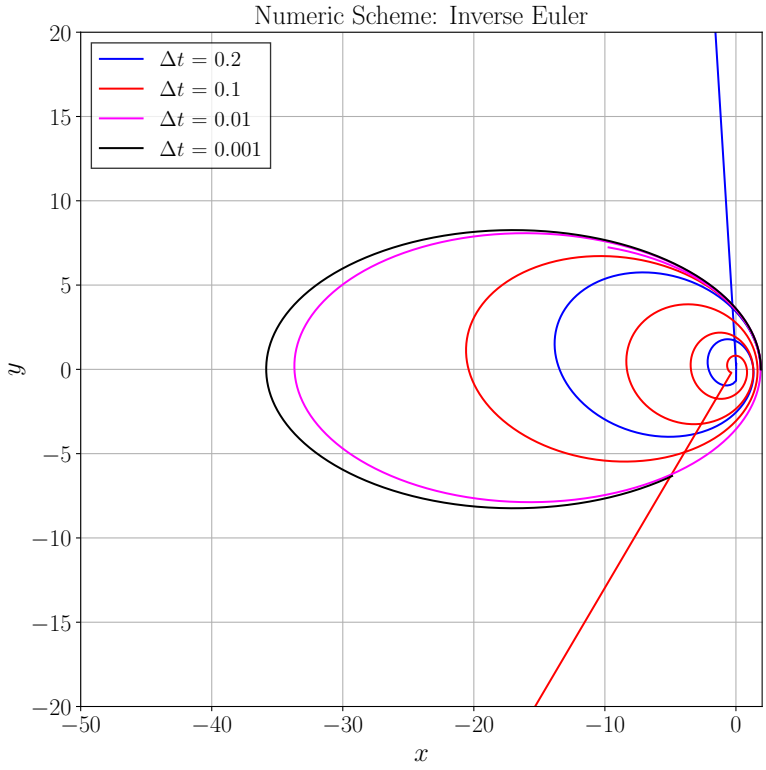


Figura 6: Órbitas calculadas mediante un esquema Euler inverso.

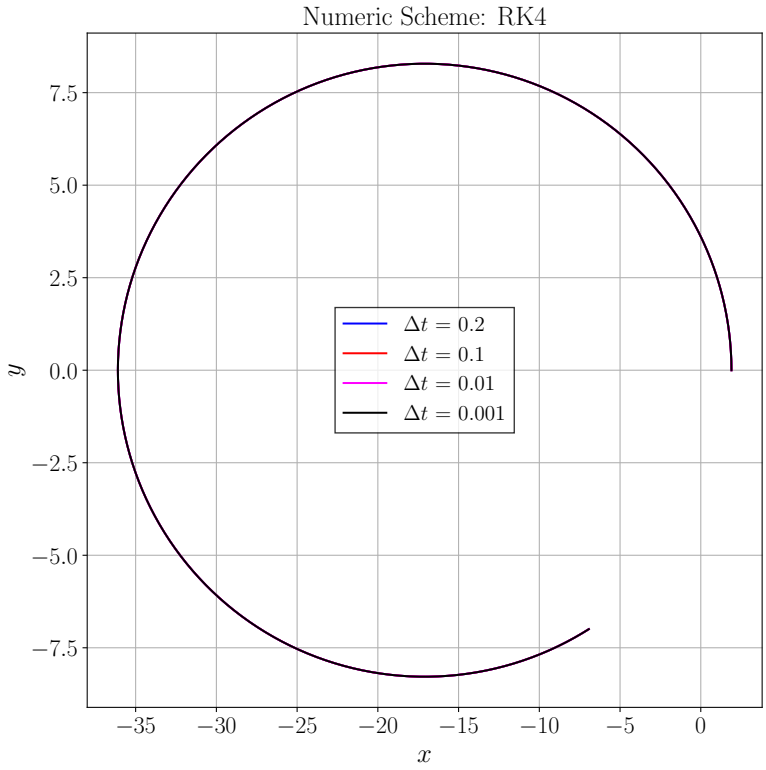


Figura 7: Órbitas calculadas mediante un esquema Runge Kutta de 4º orden.

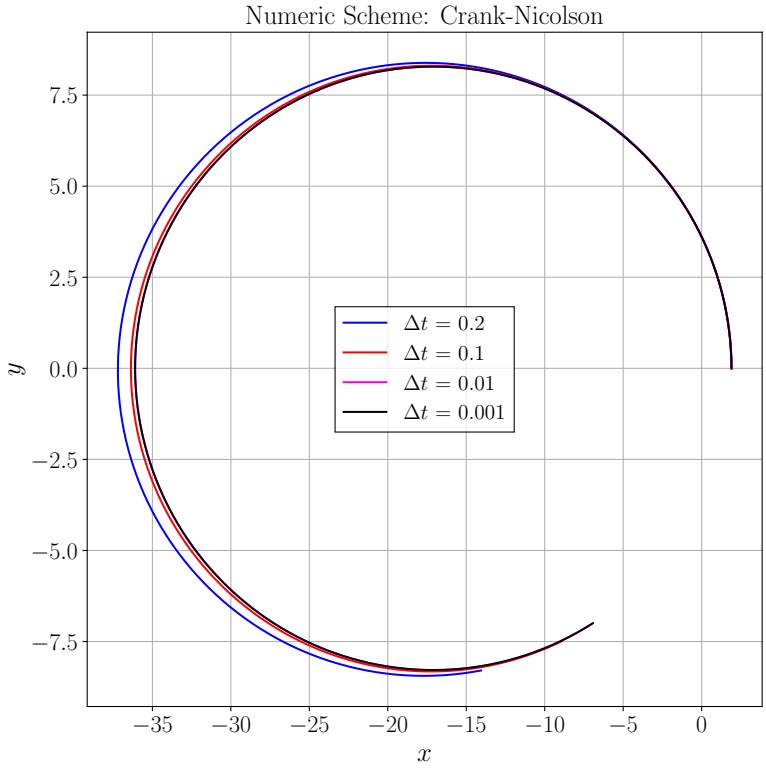


Figura 8: Órbitas calculadas mediante un esquema Crank-Nicolson.

# Referencias

[1] Hernández, Juan Antonio, *Cálculo Numérico en Ecuaciones Diferenciales Ordinarias*, 2018.

[2] Hernández, Juan Antonio & Escoto, F. Javier *How to learn Applied Mathematics through modern FORTRAN*, 2017.