


Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestone 2: Orbit integration with funtions

Ampliación de matemáticas 1

2 de octubre de 2022

Autor:

Tomás Ley Oliver

Índice

1. Introducción

El problema de integración que se va a estudiar es el de una órbita Kepleriana mediante los métodos de Euler explícito e implícito, Runge-Kutta de orden 4 y Crank-Nicolson.

1.1. Discretización

Para realizar la integración de forma numérica, se ha utilizado inicialmente un tiempo total de integración, $t = 10$, y un número de pasos temporales $N = 1000$, resultando en un $\Delta t = 0{,}01$. En el apartado 2.3 se estudiarán las diferencias de variar el número de pasos N

2. Método de Integración

2.1. Métodos explícitos

Los esquemas explícitos se caracterizan por la cualidad de que para integrar en un instante de tiempo discretizado $n+1$, solo se necesita conocer la información de instantes previos $\leq n$.

Euler explícito

A continuación, en la figura 2.1, se presentan los resultados obtenidos integrando mediante el método de Euler explícito.

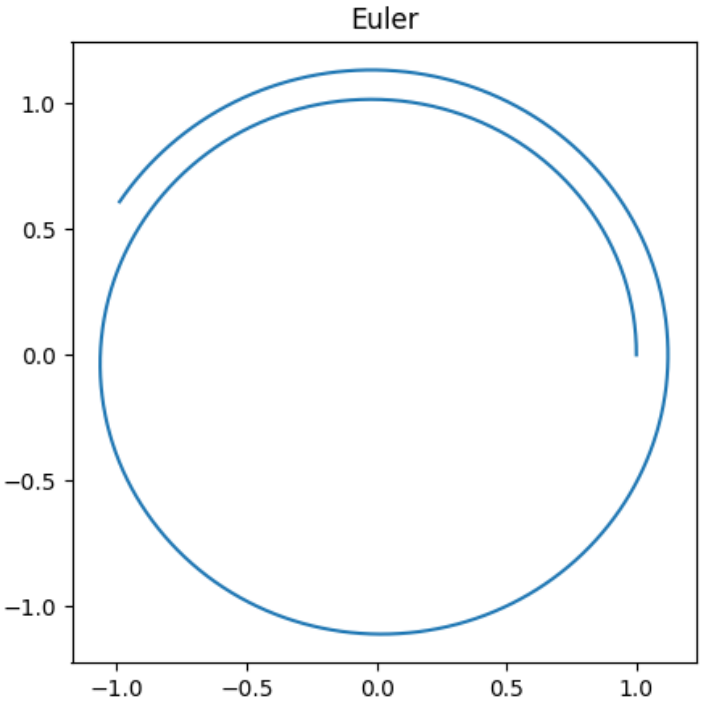


Figura 2.1: Integración de órbita kepleriana mediante Euler explícito

Se puede observar que el método de Euler explícito diverge de la solución real. Esto se debe a la sencillez del método, y el error de concepto que esto conlleva. Este método no tiene en cuenta la conservación de energía de las orbitas, por lo que el esquema divergirá independientemente del espaciado temporal utilizado.

Runge-Kutta de orden 4

A continuación, en la figura 2.2, se presentan los resultados obtenidos integrando mediante Runge-Kutta de orden 4.

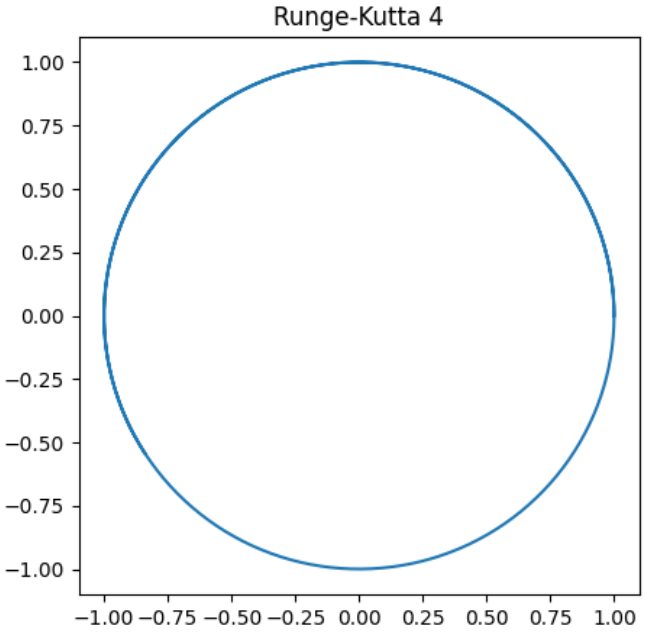


Figura 2.2: Integración de órbita kepleriana mediante Runge-Kutta de orden 4

Se observa como este método, a diferencia del Euler explícito, estudiado en el apartado 2.1, se asemeja a la realidad de las órbitas planetarias. Este esquema requiere de mayor potencia operativa, pero los resultados obtenidos son precisos.

2.2. Métodos implícitos

Los esquemas implícitos se caracterizan por ser lineales, ya que el cálculo en un instante $n+1$ depende de las variables en ese mismo instante, por lo que se requieren métodos iterativos para hallar la solución en un único instante. Esto incrementa mucho el coste computacional, y por ello estos métodos no se suelen utilizar en el cálculo de órbitas.

Euler inverso

A continuación, en la figura 2.3, se presentan los resultados obtenidos mediante el método de Euler inverso.

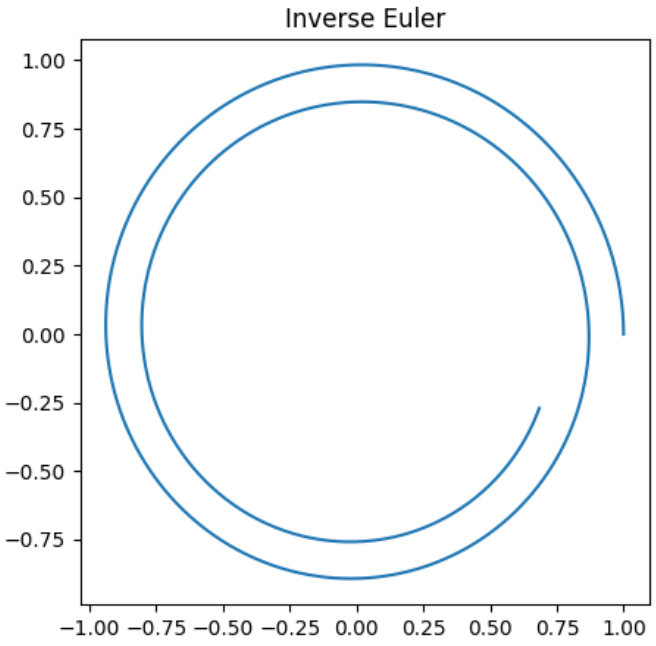


Figura 2.3: Integración de órbita kepleriana mediante Euler inverso

Al igual que en su forma explícita, el método de Euler inverso diverge de la solución real. De nuevo, se debe a que este método no tiene en cuenta la conservación de energía de las órbitas, teniendo en este caso una pérdida de energía.

Crank-Nicolson

A continuación, en la figura 2.4, se presentan los resultados obtenidos con el método Crank-Nicolson.

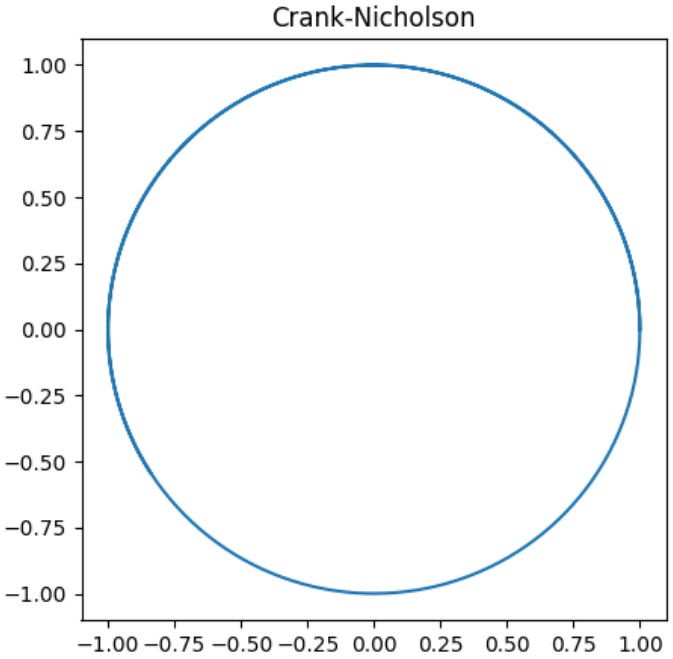


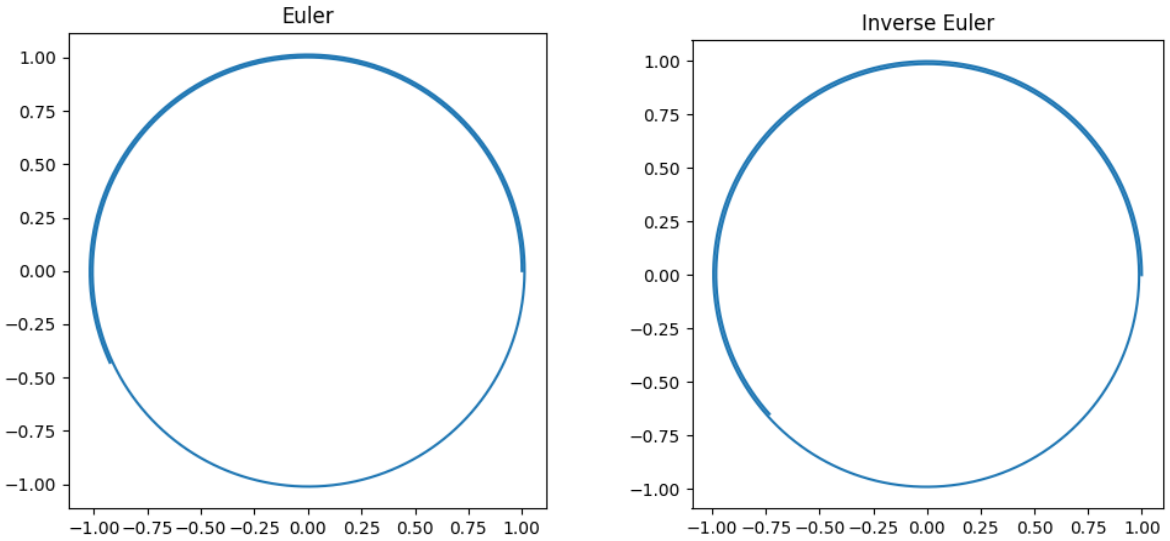
Figura 2.4: Integración de órbita kepleriana mediante Crank-Nicolson

El método Crank-Nicolson se puede considerar una combinación del método Euler explícito en *n* y Euler inverso en *n+1*. Esto da una convergencia de segundo orden en el tiempo, por lo cual el sistema converge.

2.3. Variación de la discretización temporal

Para observar las diferencias en los métodos con la discretización temporal, se va a variar el número de pasos *N*. Sólo se estudiarán los métodos de Euler, ya que son los que presentan diferencias en los resultados con la discretización.

El primer caso estudiado, es con $N = 10\ 000$. Esto resulta en una espiral más cerrada, pero que dándole un tiempo de integración lo suficientemente alto, terminará divergiendo del resultado real.



Figuras 2.5a y 2.5b: Variación en los métodos de Euler explícito (izquierda) e inverso (derecha) con la discretización temporal

Se ha observado que para $N < 500$, el cálculo del método de Euler inverso da problemas debido a fallos en la convergencia utilizando Newton-Raphson para anular el residuo.

3. Programación

3.1. Abstracción

Para llevar a cabo estos cálculos, se ha utilizado una metodología de programación Top-Down. Con esta metodología se diseñan los modelos para poder realizar las simulaciones con unas entradas y salidas lo más genéricas posibles. Esto implica una filosofía funcional, más que imperativa por su naturaleza compartimentalizada. En el caso de la órbita Kepleriana se ha diseñado el siguiente esquema funcional:

Cauchy → Kepler → Esquema temporal → Newton-Raphson

El método Newton-Raphson se utiliza para resolver los esquemas implícitos.

3.2. Módulos

El programa completo se ha dividido en cuatro módulos distintos, diferenciados por la función de cada uno en el problema:

Main

Este archivo incluye las condiciones iniciales del problema, la discretización temporal, y la definición del problema, además de la representación de los resultados con la librería *matplotlib*. En este archivo principal se importan todas las funciones definidas en el resto de módulos, que están diseñados para estar autocontenidos.

```
from esquemas_temporales import Euler, RK4, Euler_Inv, CN
from Cauchy_problem import Cauchy
from Orbit import Kepler
```

```
## Initial values for kepler orbit

U0 = [1, 0, 0, 1]

## Integration steps

N = 1000
t_fin = 10
```

```
temp_sch = [Euler, Euler_Inv, RK4, CN]


for scheme in temp_sch:

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

Cauchy_problem

Este módulo define el problema de Cauchy. Las entradas que utiliza están generalizadas para poder aplicarse a cualquier problema físico. *Phy* es la función que define el problema físico, *U0* las condiciones iniciales del problema, y *Esq_temp* el esquema temporal que se va a utilizar para resolverlo.

```
def Cauchy( Phy , t, U0, Esq_temp):

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

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

    U[0,:] = U0

    for n in range(N):

        dt = t[n+1] - t[n]  # Para casos donde dt =/ cte
        U[n+1,:] = Esq_temp(U[n,:], t[n], dt, Phy)

    return U
```

Orbit

Este módulo sólo incluye el problema de Kepler en un tiempo generalizado *t_n*

```python
def Kepler(U, t):      # U vector de estado en un tiempo tn

    d = (U[0]**2 + U[1]**2)**(3/2)

    f1 = U[2]
    f2 = U[3]
    f3 = -U[0] / d
    f4 = -U[1] / d
    Fn = [f1,f2,f3,f4]
    Fn = array(Fn)

    return Fn
```

Esquemas_temporales

En este módulo están definidos los cuatro esquemas temporales que se han estudiado en este hito. Para los implícitos, se ha recurrido a la librería *scipy* para importar el algoritmo Newton-Raphson.

```python
from scipy.optimize import newton
```

Este módulo está diseñado para recibir como entradas el vector de estado *Un* en *t_n*, el tiempo de integración *t_n*, el diferencial de tiempo Δt, y la función *F* del problema físico. A continuación, se muestran como ejemplo ambos esquemas de Euler:

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

    UE = U + dt * F(U, t)

    return UE
```

```python
def Euler_Inv(U, t, dt, F):

    def Residuo_EI(X):

        res = X - U - dt*F(X, t)

        return res

    UE_Inv = newton(func = Residuo_EI, x0 = U)

    return UE_Inv
```

3.3. Conclusiones

A diferencia de en el primer hito cuyo objetivo era resolver el problema de Cauchy, este se ha centrado en la abstracción del código a un modelo funcional Top-Down.

La metodología de programación Top-Down permite una resolución más general de los problemas a estudiar. Esto facilita la comprensión del código en su totalidad, y también su depuración. Gracias a esto, una vez se tienen diseñados los módulos, resolver un problema distinto, o utilizar un esquema diferente solo implicaría añadirlo al módulo respectivo y llamarlo desde el documento principal.