


Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestones 5 & 6

Ampliación de Matemáticas I

11 de diciembre de 2022

Autor:

- Sergio Cavia Fraile

Índice

1. Introducción

El objetivo del milestone 5 será simular el problema de los N cuerpos y comentar los resultados de distintas condiciones iniciales.

El objetivo del milestone 6 será estudiar el problema circular restringido de los 3 cuerpos mediante distintos esquemas temporales que incluyen al Runge-Kutta embebido. Se calculará la posición de los puntos de Lagrange, su estabilidad y se comparará la órbita de un cuerpo que orbita alrededor de dichos puntos en función del esquema temporal escogido.

2. Ecuaciones

2.1. El problema de los N cuerpos

La fórmula que rige el problema de los N cuerpos es la siguiente:

$$m_i \frac{\delta^2 r_i}{\delta t^2} = \sum_{j=1, j\neq i}^{N} \frac{G m_i m_j (r_j - r_i)}{\parallel r_j - r_i \parallel^3}$$

Por simplicidad tomaremos todas las masas y la constante gravitacional iguales a la unidad, de tal forma que la fórmula se convierte en:

$$\frac{\delta^2 r_i}{\delta t^2} = \sum_{j=1, j\neq i}^{N} \frac{(r_j - r_i)}{\parallel r_j - r_i \parallel^3}$$

2.2. Problema circular restringido de los 3 cuerpos

El problema circular restringido de los tres cuerpos (CR3BP) es un caso particular del problema de los N cuerpos donde uno de los cuerpos tiene una masa despreciable (por ejemplo, un satélite) y el movimiento de los otros cuerpos es circular alrededor de su baricentro.

El cuerpo principal se encontrará en la posición $(\mu, 0)$ tomando el baricentro como origen del coordenadas, el cuerpo secundario se situará en $(1 - \mu, 0)$, donde μ es:

$$\mu = \frac{m_2}{m_1 + m_2}$$

Para todos los cálculos en este informe se utilizará el sistema Tierra-Luna, cuya $\mu = 1{,}2151 \times 10^2$.

En estas condiciones, el movimiento del tercer cuerpo se rige por las siguientes ecuaciones:

$$\ddot{x} = 2\dot{y} + x - \frac{(1-\mu)(x+\mu)}{p_1^3} - \frac{\mu(x-1+\mu)}{p_2^3}$$

$$\ddot{y} = -2\dot{x} + y - \frac{(1-\mu)y}{p_1^3} - \frac{\mu y}{p_2^3}$$

Donde:

$$p_1 = \sqrt{(x+\mu)^2 + y^2}$$

$$p_2 = \sqrt{(x-1+\mu)^2 + y^2}$$

2.3. Puntos de Lagrange

Los puntos de Lagrange, también denominados puntos L o puntos de libración, son las cinco posiciones en un sistema orbital donde un objeto pequeño, solo afectado por la gravedad, puede estar teóricamente estacionario respecto a dos objetos más grandes.

Por lo tanto, los puntos que buscamos son aquellos en los que la aceleración del tercer cuerpo es nula.

2.4. Estabilidad de los puntos de Lagrange

Para estudiar la estabilidad debemos calcular los autovalores de la matriz jacobiana del sistema en cada punto de Lagrange. Si dichos autovalores tienen parte real nula podemos decir que el punto es estable.

2.5. Runge-Kutta embebido

Los métodos embebidos están diseñados para producir una estimación del error de truncamiento local de un solo paso de Runge-Kutta y, como resultado, permiten controlar el error con un tamaño de paso adaptativo. Esto se hace teniendo dos métodos en el cuadro, uno con orden p y otro con orden p-1.

El más sencillo de los métodos de Runge-Kutta embebidos es el Heun-Euler. Su tabla de Butcher es la siguiente:

0		
1	1	
	1/2	1/2
	1	0

Y combina los métodos de Heun (orden 2) y Euler (orden 1). La estimación del error se utiliza para controlar el tamaño del paso.

3. Código

3.1. Problema de los N cuerpos

El problema de los N cuerpos se ha implementado mediante las siguientes líneas de código:

```
from numpy import reshape, zeros
from numpy.linalg import norm

def NBody(U, t, Nb, Nc):
    F =  zeros(len(U))
    Us  = reshape(U,(Nb, Nc, 2))
    dUs = reshape(F,(Nb, Nc, 2))

    r = reshape(Us[:,:,0],(Nb, Nc))
    v = reshape(Us[:,:,1],(Nb, Nc))
    dr = reshape(dUs[:,:,0], (Nb,Nc))
    dv = reshape(dUs[:,:,1], (Nb,Nc))

    for i in range(Nb):
        dr[i,:] = v[i,:]
        for j in range(Nb):
            if j != i:
                dv[i,:] = dv[i,:] + (r[j,:]-r[i,:])/norm(r[j,:]-r[i,:])**3

    return F
```

Donde simplemente utilizamos la función reshape para facilitarnos el trabajo y creamos dos bucles anidados en los que se dan valores a la velocidad y aceleración según las fórmulas que ya comentamos antes.

3.2. Código final N cuerpos

El código que realmente integra el problema de los N cuerpos es el siguiente:

Primero defino una función que permite introducir las condiciones iniciales (así como el número de cuerpos y de coordenadas por cuerpo) de una forma cómoda. Para ello, sitúo la función en el propio programa principal para que no sea necesario buscar el módulo que contiene las condiciones iniciales cada vez que se quieran cambiar.

```
def Init():

    #Función para dar las condiciones iniciales.
    #Definir nº coordenadas, cuerpos y velocidad y posición inicial de cada cuerpo

    Nc = 3
    Nb = 3

    U0 =  zeros(2*Nc*Nb)
    Uord  = reshape(U0,(Nb, Nc, 2))
    r0 = reshape(Uord[:,:,0],(Nb, Nc))
    v0 = reshape(Uord[:,:,1],(Nb, Nc))

    #Cuerpo 1
    r0[0,:] = [0.5, -0.5, 0]
    v0[0,:] = [0.1, 0.1, 0.3]
    #Cuerpo 2
    r0[1,:] = [0, 0, 0]
    v0[1,:] = [-0.1, -0.1, 0.5]
    #Cuerpo 3
    r0[2,:] = [-0.4, 0.5, 0]
    v0[2,:] = [-0.35, -0.3, 0.3]

    return U0, Nc, Nb
```

Posteriormente se definen los tiempos y pasos en los que se integrará y se llama a la función de condiciones iniciales. Tras esto se llama a Cauchy, creando primero una función de paja que permita que Cauchy lea la función del problema de los N cuerpos pese a contar con más argumentos de entrada de los que Cauchy tenía en cuenta.

```
#Defino tiempo final, numero divisiones y condiciones iniciales
tf = 4
N = 2000
t = linspace(0, tf, N+1)
U0, Nc, Nb = Init()

# Llamo a Cauchy para calcular la función del problema de los N cuerpos pese a que tiene más inputs que (U, t)
def F(U, t):
   return NBody(U, t, Nb, Nc)
U = Cauchy(F, t, U0, RK4)
```

Finalmente se grafican los resultados en 3D y en el plano XY con las siguientes líneas de código:

```
#Grafico 3D
r   = reshape(U,(N+1, Nb, Nc, 2))[:,:,:,0]
fig = plt.figure()
ax1 = fig.add_subplot(111,projection='3d')
col = ["blue","red","green","purple","yellow","orange","black"]
for i in range(Nb):
    ax1.plot_wireframe(r[:,i,0].reshape((-1, 1)), r[:,i,1].reshape((-1, 1)), r[:,i,2].reshape((-1, 1)), color = col[i])
plt.title("N-body problem")
ax1.set_xlabel("x")
ax1.set_ylabel("y")
ax1.set_zlabel("z")
plt.grid()
plt.show()
#Grafico 2D
for i in range(Nb):
   plt.plot(r[:, i, 0], r[:, i, 1], color = col[i])
plt.axis('equal')
plt.title("N-body problem X-Y Plane")
plt.xlabel("x")
plt.ylabel("y")
plt.grid()
plt.show()
```

3.3. Problema circular restringido de los 3 cuerpos

En esta función simplemente hemos introducido las ecuaciones descritas en el apartado 2.2:

```
def CR3BP(U, mu):
    r = U[0:2]
    drdt = U[2:4]

    p1 = sqrt((r[0]+mu)**2 + r[1]**2)
    p2 = sqrt((r[0]-1+mu)**2 + r[1]**2)

    dvdt_1 = -(1-mu)*(r[0]+mu)/(p1**3) - mu*(r[0]-1+mu)/(p2**3)
    dvdt_2 = -(1-mu)*r[1]/(p1**3) - mu*r[1]/(p2**3)

    return array([drdt[0], drdt[1], 2*drdt[1] + r[0] + dvdt_1, -2*drdt[0] + r[1] + dvdt_2])
```

3.4. Puntos de Lagrange

En esta función hemos buscado los puntos donde la aceleración del tercer cuerpo es nula utilizando la función fsolve:

```
def Lpoints(U_0, Np, mu):
    LP = zeros([5,2])

    def F(Y):
        X = zeros(4)
        X[0:2] = Y
        X[2:4] = 0
        FX = CR3BP(X, mu)
        return FX[2:4]

    for i in range(Np):
        LP[i,:] = fsolve(F, U_0[i,0:2])

    return LP
```

3.5. Estabilidad de los puntos de Lagrange

En esta función calculamos los autovalores de la matriz jacobiana del sistema en un punto dado:

```
def Stability(U_0, mu):

    def F(Y):
        return CR3BP(Y, mu)

    A = Jacobian(F, U_0)
    values, vectors = eig(A)

    return values

def Jacobian(F, U):
    N = size(U)
    Jac = zeros([N,N])
    t = 1e-10

    for i in range(N):
        xj = zeros(N)
        xj[i] = t
        Jac[:,i] = (F(U + xj) - F(U - xj))/(2*t)
    return Jac
```

3.6. Runge-Kutta embebido

En este módulo definimos el método de Runge-Kutta embebido por el método de Heun-Euler definiendo los dos métodos, el paso de tiempo variable y la tabla de Butcher:

```
def RKEmb(U, dt, t, F):

    tol = 1e-9
    orders, Ns, a, b, bs, c = ButcherArray()
    est1 = RK(1, U, t, dt, F)
    est2 = RK(2, U, t, dt, F)
    h = min(dt, StepSize(est1-est2, tol, dt,  min(orders)))
    N_n = int(dt/h)+1
    n_dt = dt/N_n
    est1 = U
    est2 = U

    for i in range(N_n):
        time = t + i*dt/int(N_n)
        est1 = est2
        est2 = RK(1, est1, time, n_dt, F)

    final_state = est2
    ierr = 0

    return final_state


def RK(order, U1, t, dt, F):
    orders, Ns, a, b, bs, c = ButcherArray()
    k = zeros([Ns, len(U1)])
    k[0,:] = F(U1, t + c[0]*dt)

    if order == 1:
        for i in range(1,Ns):
            Up = U1
            for j in range(i):
                Up = Up + dt*a[i,j]*k[j,:]
            k[i,:] = F(Up, t + c[i]*dt)
        U2 = U1 + dt*matmul(b,k)

    elif order == 2:
        for i in range(1,Ns):
            Up = U1
            for j in range(i):
                Up = Up + dt*a[i,j]*k[j,:]
            k[i,:] = F(Up, t + c[i]*dt)
        U2 = U1 + dt*matmul(bs,k)

    return U2
```

```
def StepSize(dU, tol, dt, orders):
    error = norm(dU)

    if error > tol:
        step_size = dt*(tol/error)**(1/(orders+1))
    else:
        step_size = dt

    return step_size

def ButcherArray(): #Heun-Euler
    orders = [2,1]
    Ns = 2

    a = zeros([Ns,Ns-1])
    b = zeros([Ns])
    bs = zeros([Ns])
    c = zeros([Ns])

    c = [ 0., 1. ]
    a[0,:] = [  0. ]
    a[1,:] = [  1. ]
    b[:]  = [ 1./2, 1./2 ]
    bs[:] = [ 1.,    0.  ]
    return orders, Ns, a, b, bs, c
```

3.7. Código final N cuerpos

Por último, debajo de estas líneas incluyo el código general en el que llamo a las funciones anteriores para obtener los resultados:

```
from numpy import array, linspace, zeros, around
import matplotlib.pyplot as plt
from random import random

from ODEs.CauchyCode import Cauchy
from ODEs.SchemesCode import Euler, RK4, CN, EulerInv, LF
from ODEs.RKEmbCode import RKEmb
from Functions.Res3BodyCode import CR3BP, Lpoints, Stability

#Defino tiempo final, numero divisiones
N = int(1e5)
t = linspace(0, 100, N)
mu = 1.2151e-2 #Tierra-Luna
#mu = 3.0039e-7 #Sol-Tierra

# Creo una función que Cauchy pueda llamar ya que la nuestra tiene más inputs que (U, t)
def F(U,t):
   return CR3BP(U, mu)

#Calculo los puntos de Lagrange iniciando desde puntos cercanos
U0LP = array([[0.1, 0, 0, 0],[1.01, 0, 0, 0],[-0.1, 0, 0, 0],[0.8, 0.6, 0, 0],[0.8, -0.6, 0, 0]])
LagPoints = Lpoints(U0LP, 5, mu)
print(LagPoints)
```

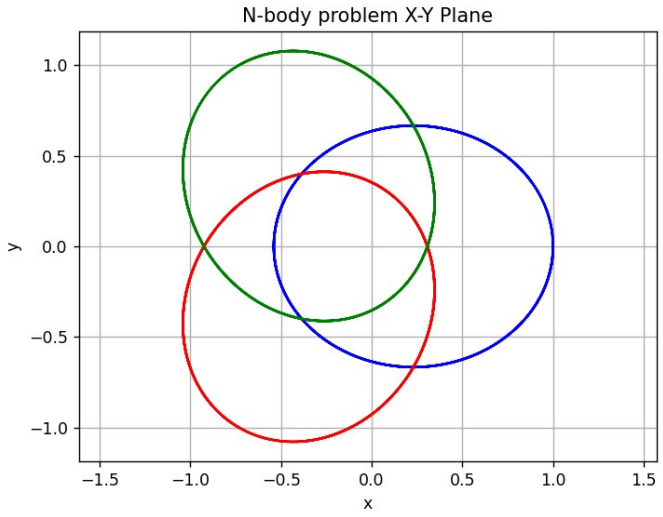
```
#Genero una condiciones iniciales cercanas a un punto de lagrange
sel_LG = 4

U0 = zeros(4)
U0[0:2] = LagPoints[sel_LG-1,:]
ran = 1e-4*random()
U0 = U0 + ran

#Integro el problema circular restringido de los 3 cuerpos mediante un esquema temporal
U = Cauchy(F, t, U0, RKEmb)

#Estudio la estabilidad del punto
U0S = zeros(4)
U0S[0:2] = LagPoints[sel_LG-1,:]
eingvalues = Stability(U0S, mu)
print(around(eingvalues.real,8))


#Grafico
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.plot(U[:,0], U[:,1],'-',color = "r")
ax1.plot(-mu, 0, 'o', color = "g")
ax1.plot(1-mu, 0, 'o', color = "b")
for i in range(5):
    ax1.plot(LagPoints[i,0], LagPoints[i,1] , 'o', color = "k")
ax2.plot(U[:,0], U[:,1],'-',color = "r")
ax2.plot(LagPoints[sel_LG-1,0], LagPoints[sel_LG-1,1] , 'o', color = "k")
ax1.set_title("Orbital view")
ax2.set_title("Close-up")
fig.suptitle("Orbit around L2 with Embedded RK")
for ax in fig.get_axes():
    ax.set(xlabel='x', ylabel='y')
    ax.grid()

plt.show()
```

4. Resultados

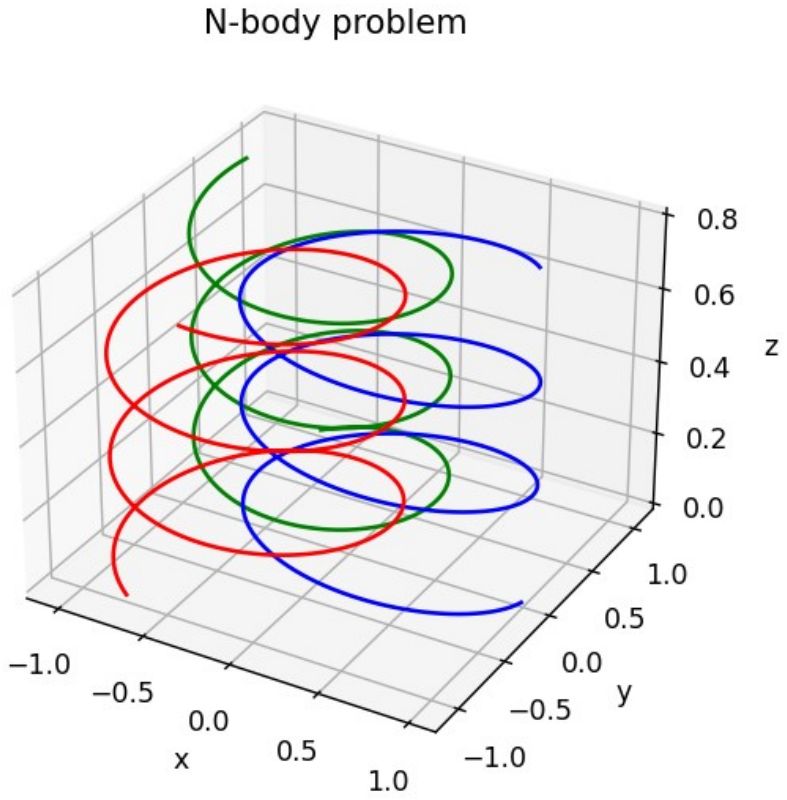
4.1. Problema de los N cuerpos

Se ha probado el programa con diversas condiciones iniciales:

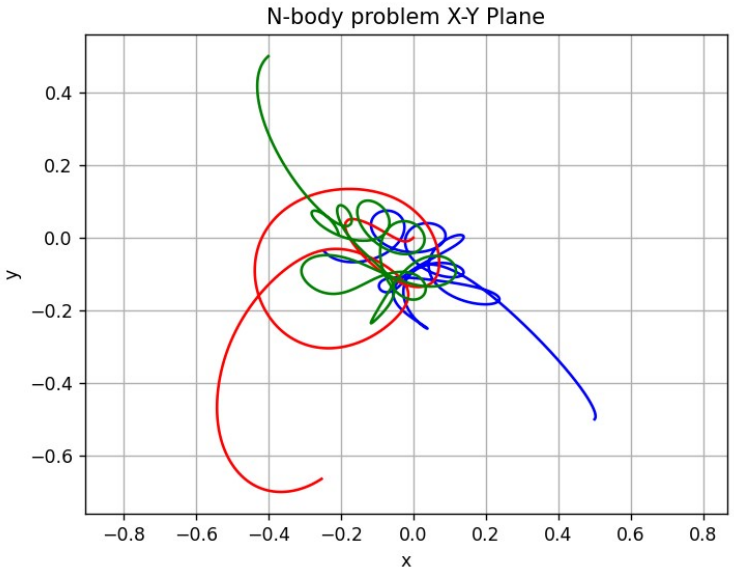
3 cuerpos estables



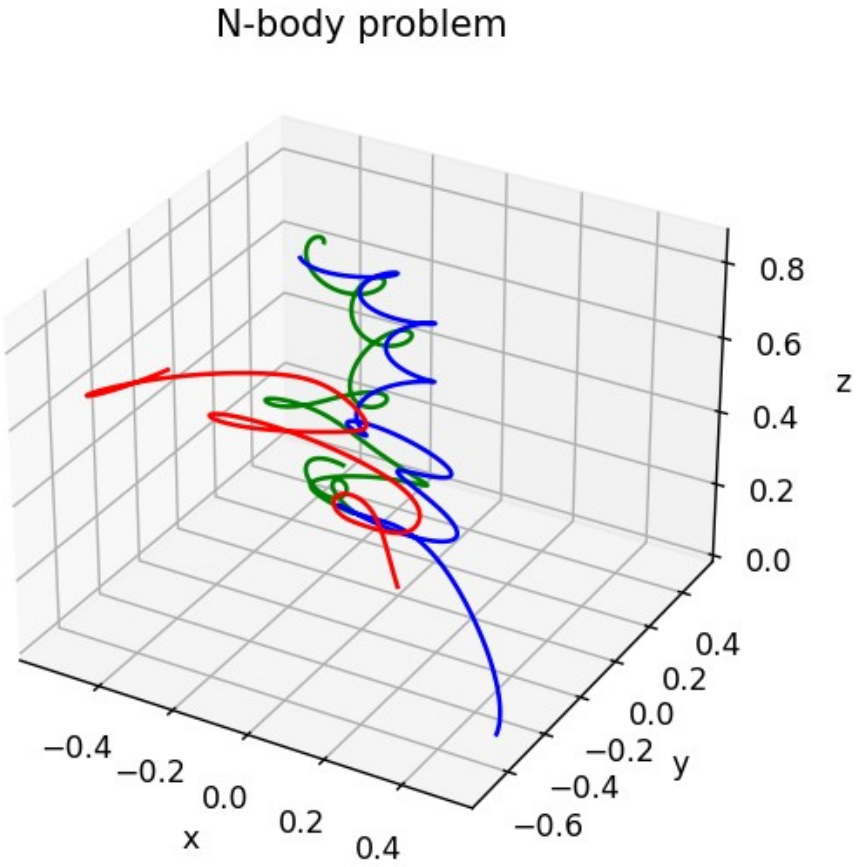

```
vini = 0.5
#Cuerpo 1
r0[0,:] = [1, 0, 0]
v0[0,:] = [0, -vini, 0.05]
#Cuerpo 2
r0[1,:] = [-sqrt(3)+1, -1, 0]
v0[1,:] = [-vini*sqrt(3)/2, vini/2, 0.05]
#Cuerpo 3
r0[2,:] = [-sqrt(3)+1, 1, 0]
v0[2,:] = [vini*sqrt(3)/2, vini/2, 0.05]
```



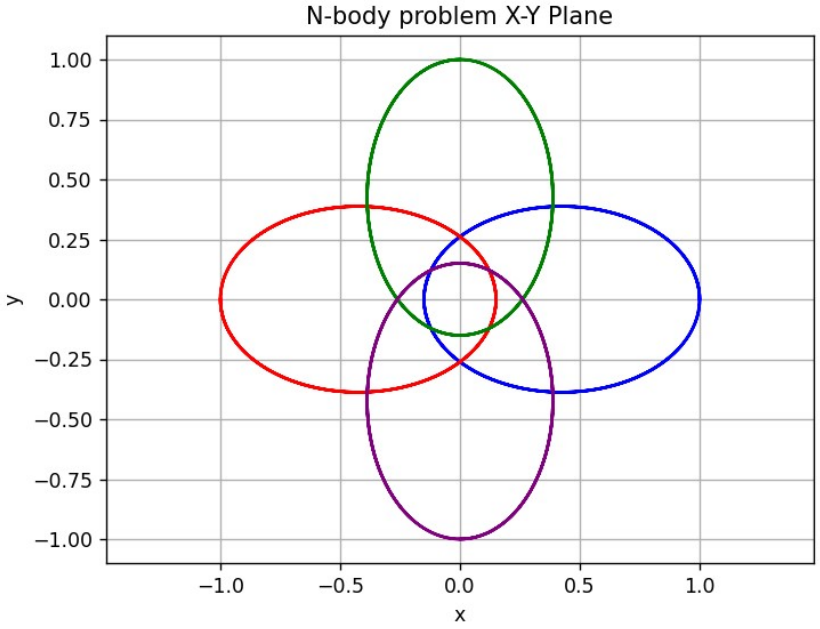
3 cuerpos inestables



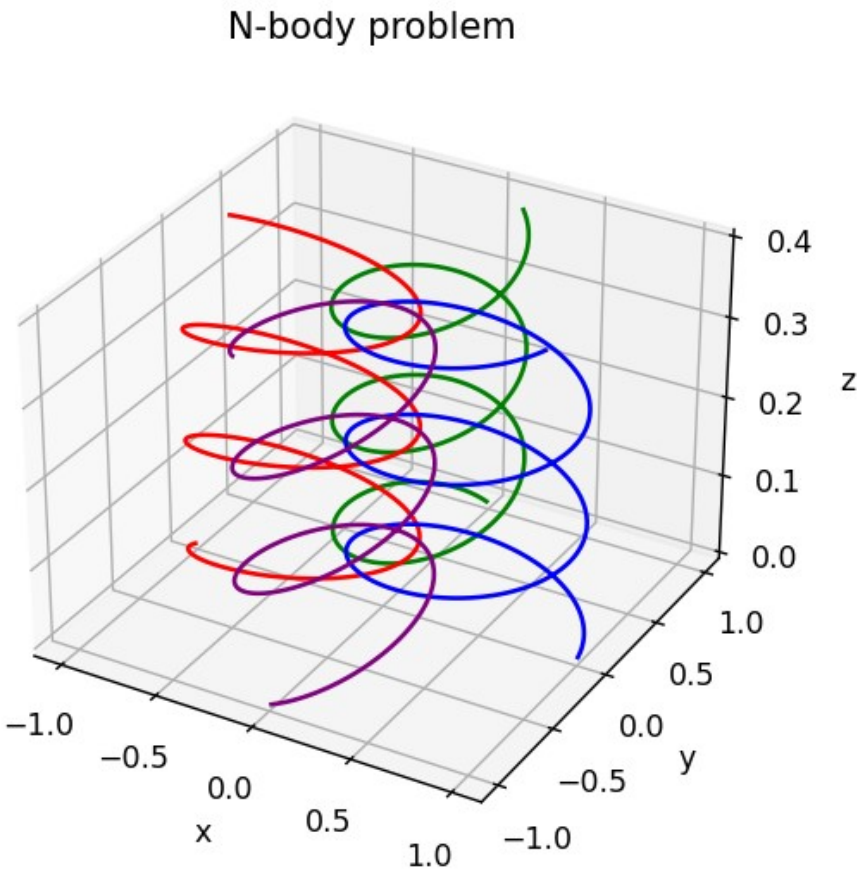

```
#Cuerpo 1
r0[0,:] = [0.5, -0.5, 0]
v0[0,:] = [0.1, 0.1, 0.3]
#Cuerpo 2
r0[1,:] = [0, 0, 0]
v0[1,:] = [-0.1, -0.1, 0.5]
#Cuerpo 3
r0[2,:] = [-0.4, 0.5, 0]
v0[2,:] = [-0.35, -0.3, 0.3]
```



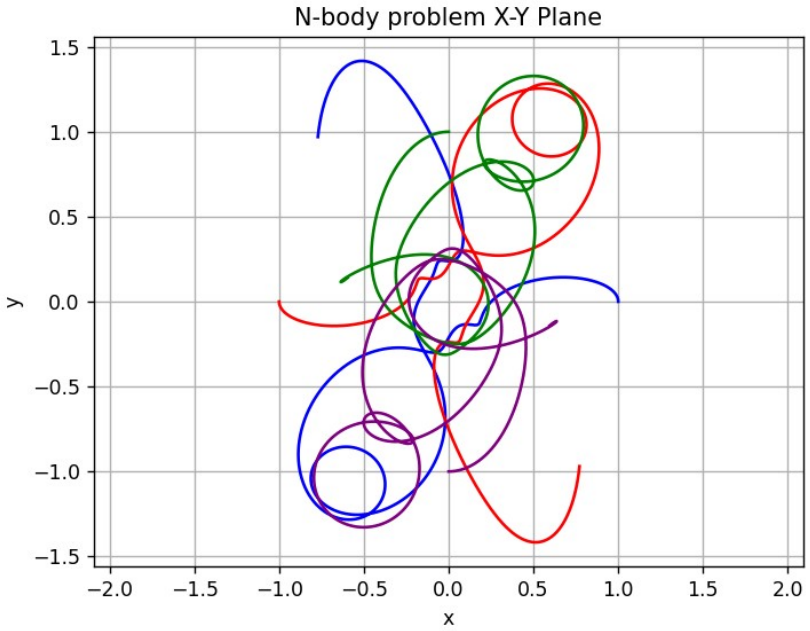
4 cuerpos estables



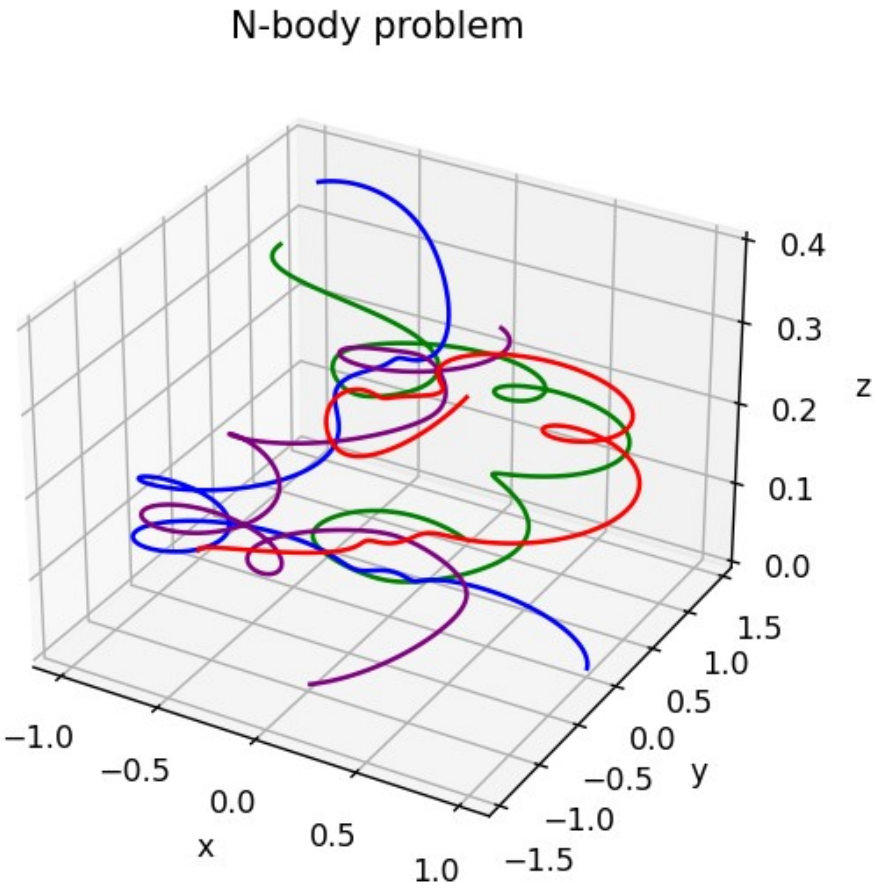

```
#Cuerpo 1
r0[0,:] = [1, 0, 0]
v0[0,:] = [0, 0.5, 0.05]
#Cuerpo 2
r0[1,:] = [-1, 0, 0]
v0[1,:] = [0, -0.5, 0.05]
#Cuerpo 3
r0[2,:] = [0, 1, 0]
v0[2,:] = [-0.5, 0, 0.05]
#Cuerpo 4
r0[3,:] = [0, -1, 0]
v0[3,:] = [0.5, 0, 0.05]
```



4 cuerpos inestables




```
#Cuerpo 1
r0[0,:] = [1, 0, 0]
v0[0,:] = [0, 0.25, 0.05]
#Cuerpo 2
r0[1,:] = [-1, 0, 0]
v0[1,:] = [0, -0.25, 0.05]
#Cuerpo 3
r0[2,:] = [0, 1, 0]
v0[2,:] = [-0.5, 0, 0.05]
#Cuerpo 4
r0[3,:] = [0, -1, 0]
v0[3,:] = [0.5, 0, 0.05]
```



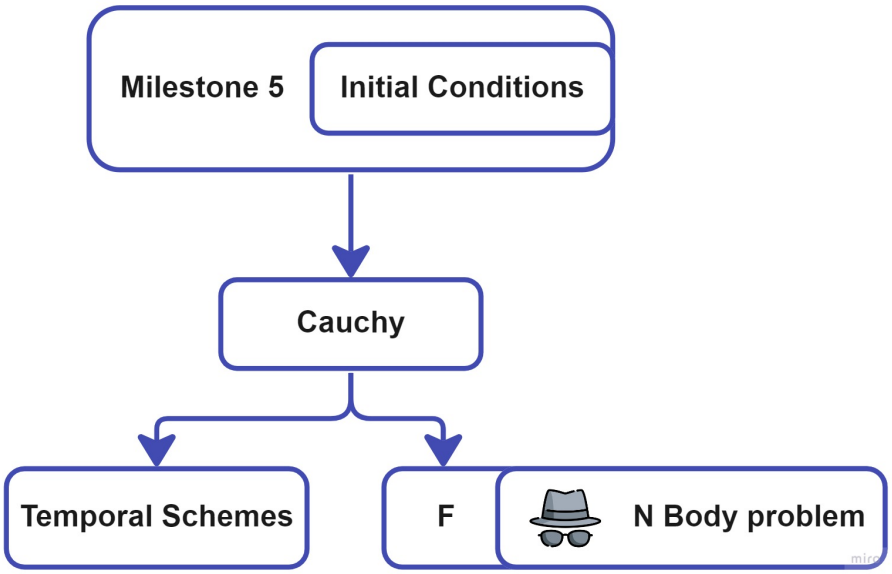
Como se puede observar, los resultados tienen sentido y son coherentes entre sí.

Para una mejor visualización de los resultados se han animado y guardado en formato de vídeo en el GitHub del alumno:

`https://github.com/jahrTeaching/milestones-SCaviaF/tree/main/doc/NBodyAnimations`

Creo necesario señalar que el esquema temporal que se ha utilizado es el Runge-Kutta de cuarto orden y que el paso elegido ha sido 0.002 unidades temporales.

Finalmente, la estructura del programa es la siguiente:



Donde se puede ver cómo el problema principal incluye la función de condiciones iniciales y cómo Cauchy llama a una F de paja que realmente está llamando a la función del problema de los N cuerpos.

4.2. Puntos de Lagrange

Los puntos de Lagrange se han calculado mediante el código descrito en el apartado 3.4 partiendo de unas condiciones iniciales cercanas a cada uno de los puntos que conocemos de forma aproximada. Los puntos calculados son los siguientes:

$$L_1 = (0{,}83691309, 0)$$
$$L_2 = (1{,}15568376, 0)$$
$$L_3 = (-1{,}00506282, 0)$$
$$L_4 = (0{,}487849, 0{,}8660254)$$
$$L_5 = (0{,}487849, -0{,}8660254)$$

4.3. Estabilidad de los puntos de Lagrange

Los autovalores del Jacobiano para cada punto son los siguientes:

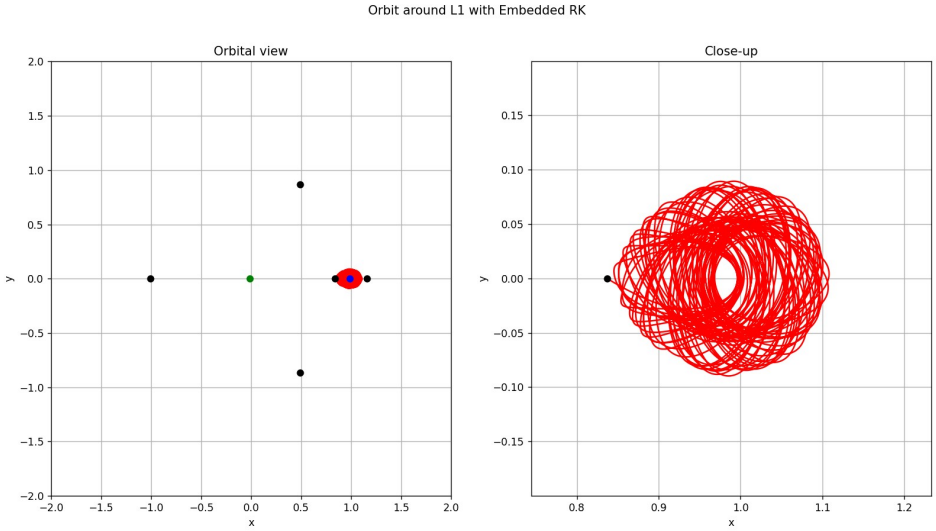
$$L_1 : (2{,}93206148, -2{,}93206148, 0, 0)$$
$$L_2 : (-2{,}15867061, 2{,}15867061, 0, 0)$$
$$L_3 : (0, 0, -0{,}17787832, 0{,}17787832)$$
$$L_4 : (3{,}4 \times 10^{-7}, 3{,}4 \times 10^{-7}, -3{,}4 \times 10^{-7}, -3{,}4 \times 10^{-7})$$
$$L_5 : (-3{,}4 \times 10^{-7}, -3{,}4 \times 10^{-7}, 3{,}4 \times 10^{-7}, 3{,}4 \times 10^{-7})$$

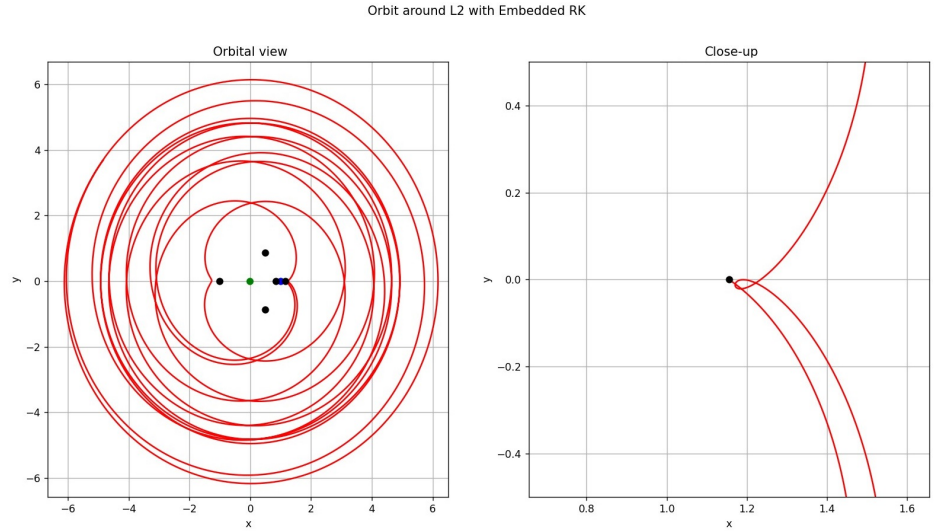
Como podemos ver, la parte real de todos los autovalores solo es nula para los puntos L_4 y L_5 (realmente es despreciable al ser del orden de 10^{-7} por errores de cálculo), por lo que estos serán los dos únicos estables.

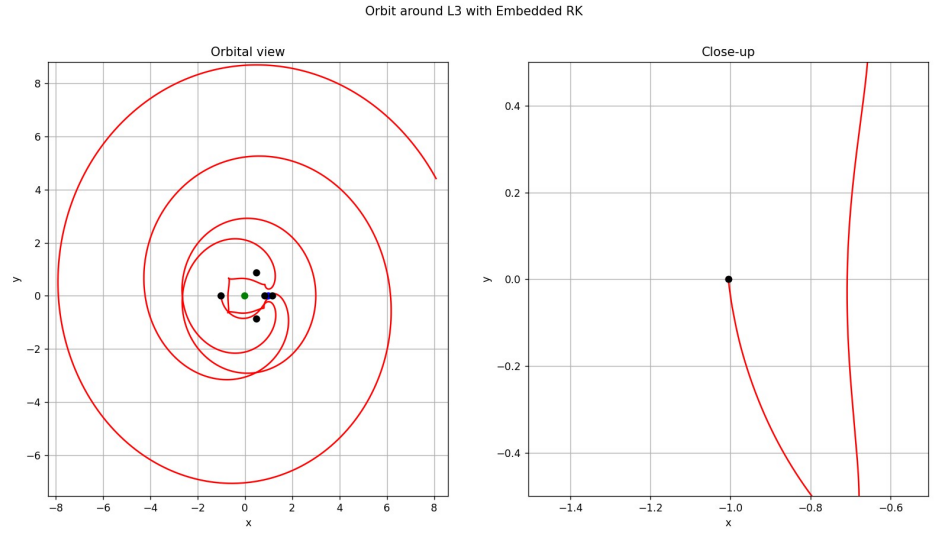
4.4. Órbitas alrededor de los puntos de Lagrange

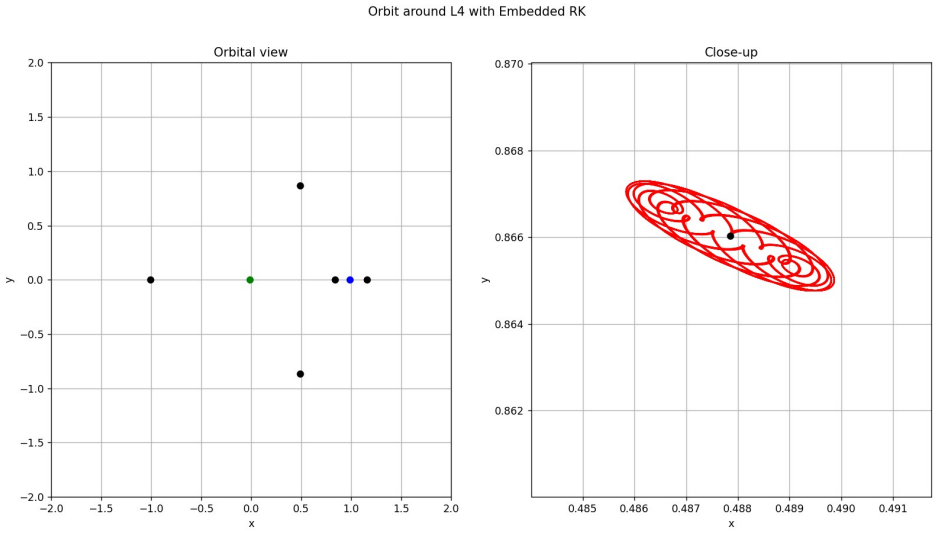
Runge-Kutta embebido

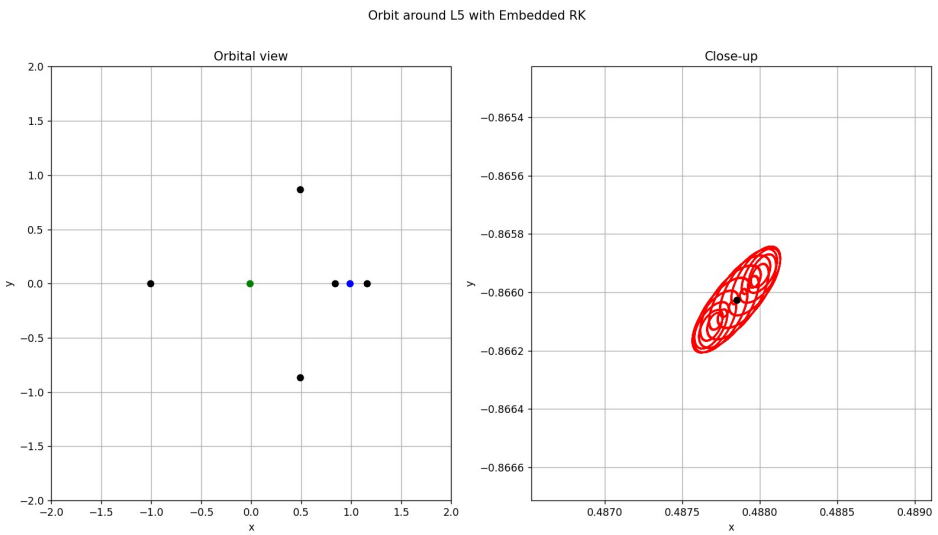
En las siguientes gráficas se muestran las órbitas que describe el tercer cuerpo al situarlo en un punto cercano a cada punto de Lagrange utilizando el Runge-Kutta embebido:





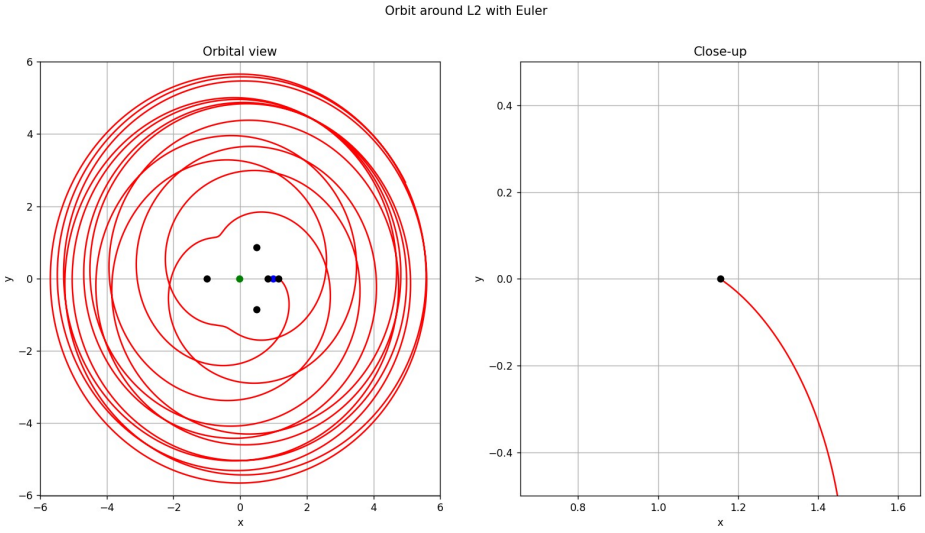

Orbit around L3 with Embedded RK
Orbital view
Close-up
x
y

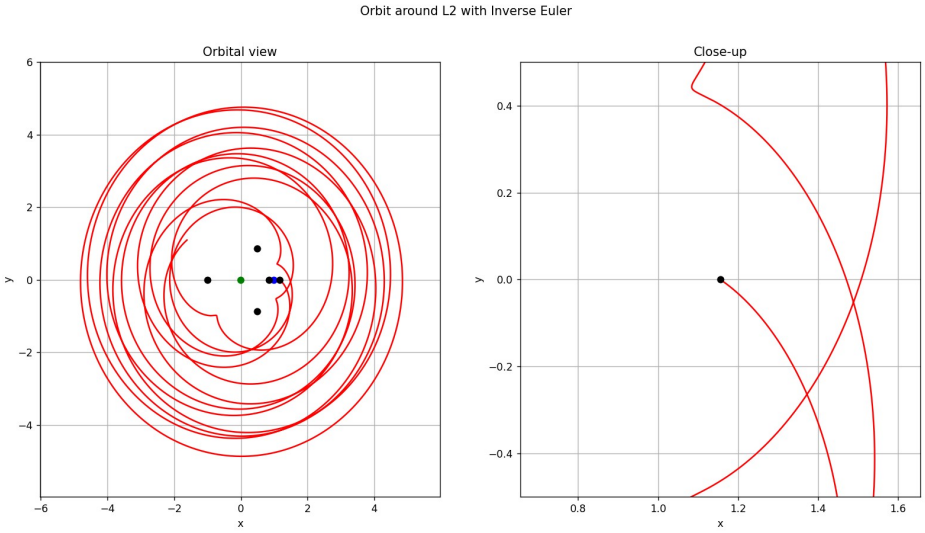

Orbit around L4 with Embedded RK
Orbital view
Close-up
x
y

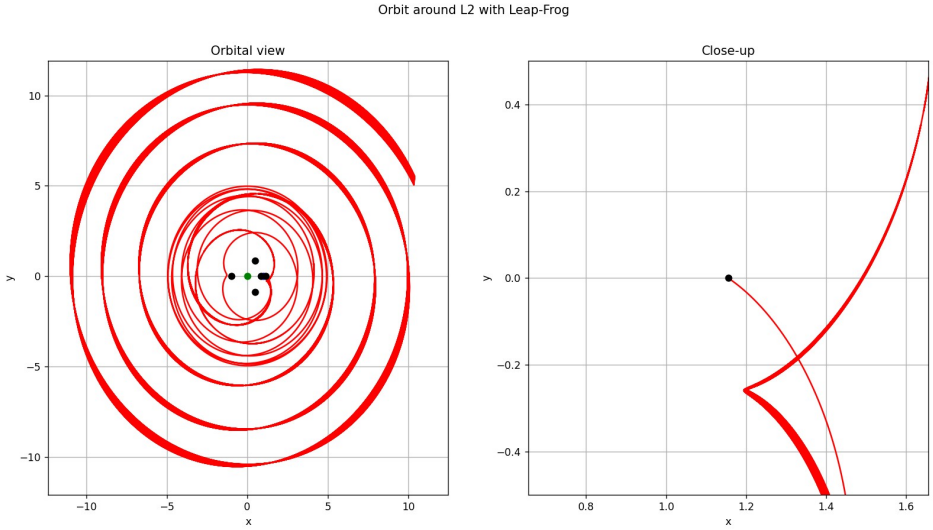

Orbit around L5 with Embedded RK
Orbital view
Close-up
x
y

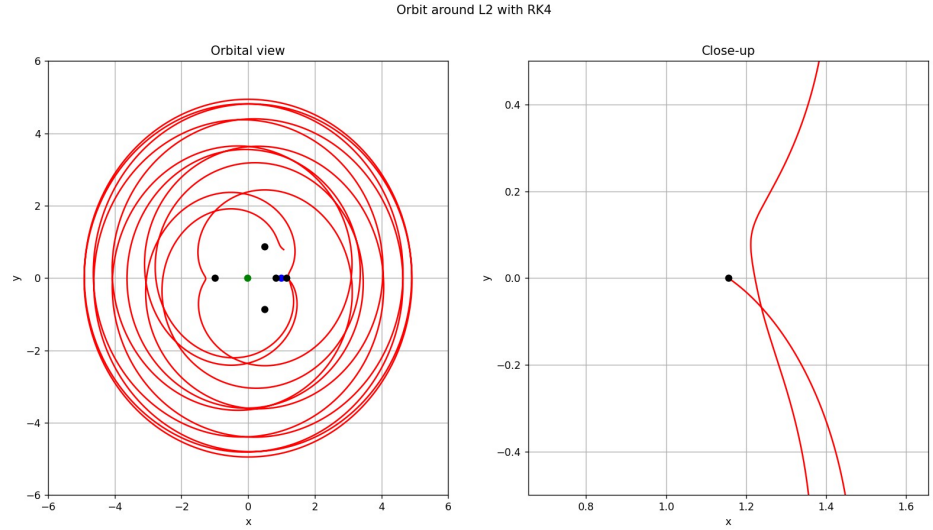
Otros esquemas temporales

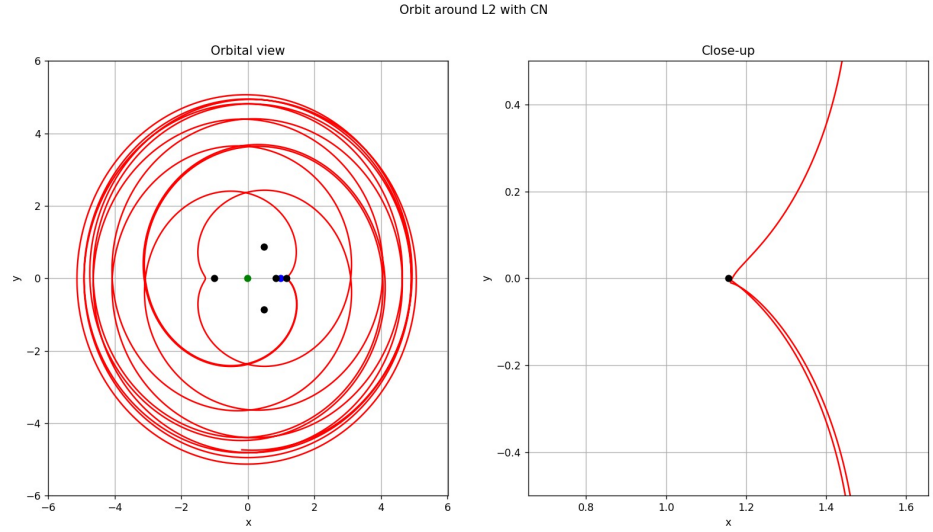
En las siguientes gráficas se muestran las órbitas que describe el tercer cuerpo al situarlo en un punto cercano a L_2 utilizando los otros esquemas temporales para poder comparar entre ellos:






Orbit around L2 with Leap-Frog
Orbital view
Close-up
x
y


Orbit around L2 with RK4
Orbital view
Close-up
x
y

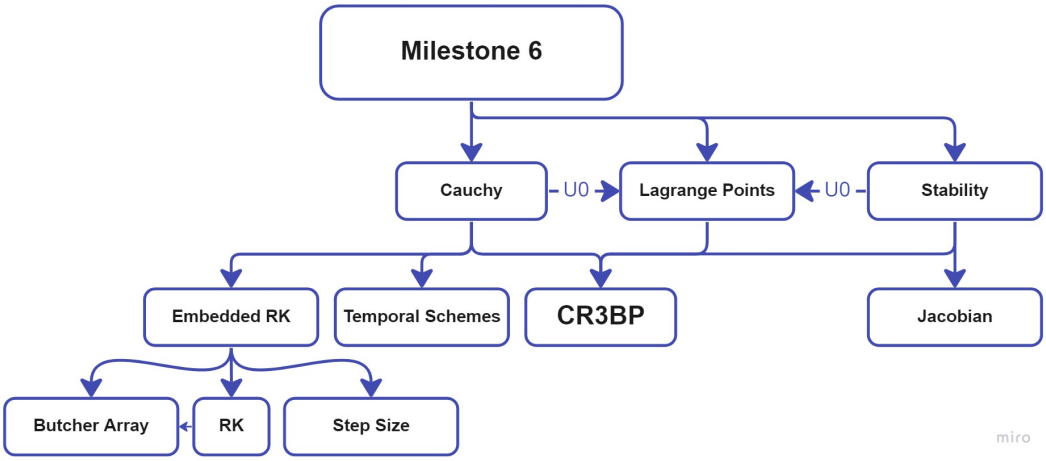

Orbit around L2 with CN
Orbital view
Close-up
x
y

4.5. Conclusiones del CR3BP

En las gráficas se han confirmado los resultados antes calculados, tanto la posición de los puntos de Lagrange como la estabilidad de los puntos L_4 y L_5.

Por otro lado hemos podido ver como todos los esquemas temporales (salvo el Leap-Frog en el que la energía del tercer cuerpo varía más) dan lugar a resultados muy similares. Hay que tener en consideración que el punto de inicio de la órbita alrededor de L_2 así como su velocidad se ha randmizado, por lo que algunas de las diferencias entre la integración mediante distintos esquemas temporales será explicada por esta randomización y no tanto por el esquema temporal utilizado en sí.

La estructura del código es la siguiente:



Al igual que para el Milestone 5 se podría haber implementado una función que realizara animaciones de las órbitas alrededor de los puntos de Lagrange.