Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestone 6

Ampliación de Matemáticas I

Javier Garrido Castillo

2 diciembre de 2022

Índice

1. Introducción

En este hito 6, primeramente se desarrollará un nuevo esquema temporal, el Runge-Kutta Embebido, que será implementado al resto de esquemas numéricos ya empleados en hitos anteriores. Además, se va a analizar el problema restringido de los tres cuerpos y elaborar una función para ello. Adicionalmente, se establecerán los puntos de Lagrange y se estudiará su estabilidad. Por último, se dibujarán distintas órbitas alrededor de los puntos de Lagrange empleando los distintos esquemas temporales.

2. Problema

2.1. Runge-Kutta Embebido

Con este esquema temporal se puede controlar el paso de tiempo que se emplea para integrar la función. Para ello, se realiza una aproximación local del error de truncación para así variar el paso de tiempo y dedicar más o menos recursos computacionales a ciertas regiones de la solución. De esta forma, es posible obtener una solución de mayor precisión que los esquemas de paso temporal fijo. Para el cálculo de dicho error, se emplearán dos esquemas temporales de tipo Runge-Kutta de orden q y orden $q-1$. De esta forma, las soluciones para cada uno de ellos son respectivamente:

$$U_{n+1} = U_n + \Delta t_n \sum_{i=1}^{s} b_i k_i \tag{1}$$

$$U^*_{n+1} = U_n + \Delta t_n \sum_{i=1}^{s} b^*_i k_i \tag{2}$$

con,

$$U_i = F(U_n + \Delta t_n \sum_{j=1}^{s} a_{ij} k_j,\ t_n + c_i \Delta t_n) \tag{3}$$

Donde, a es la matriz de Butcher. Esta matriz variará en función del método empleado para la resolución del problema. Con todo ello, el paso del tiempo mínimo se obtiene de la siguiente forma:

$$\Delta t_{min} = \Delta t \left(\frac{\epsilon}{||U^*_{n+1} - U_{n+1}||} \right)^{1/q} \tag{4}$$

2.2. Problema restringido de los tres cuerpos

Se trata de un caso particular del problema de los N cuerpos estudiado en el hito 5. Este caso consiste en establecer una masa despreciable para uno de los tres cuerpos y este se mueve bajo la influencia de los dos cuerpos principales. Para ello, es preciso definir el siguiente parámetro que relaciona las dos masas de los cuerpos principales:

$$\mu = \frac{m_2}{m_1 + m_2} \tag{5}$$

En este caso, se establecerá el sistema Tierra-Luna con un $\mu = 1.2151 \times 10^2$. Para estas condiciones, el movimiento del tercer cuerpo (el de masa despreciable) se rige por las siguientes expresiones:

$$\ddot{x} = 2\dot{y} + x - \frac{(1-\mu)(x+\mu)}{r_1^3} - \frac{\mu(x-1+\mu)}{r_2^3} \tag{6}$$

$$\ddot{y} = -2\dot{x} + y - \left(\frac{1-\mu}{r_1^3} - \frac{\mu}{r_2^3}\right) y \tag{7}$$

Donde,

$$r_1 = \sqrt{(x+\mu)^2 + y^2}; \qquad r_2 = \sqrt{(x-1+\mu)^2 + y^2} \tag{8}$$

2.3. Puntos de Lagrange y estabilidad

Estos son los puntos donde, el tercer cuerpo definido anteriormente afectado unicamente por la gravedad de los dos principales, tiene aceleración nula. Por tanto, en estos puntos, la función F es cero. Adicionalmente, para estudiar la estabilidad de dichos puntos, se debe calcular los autovalores de la matriz jacobiana del sistema para cada uno de ellos y en caso de que el autovalor tenga parte real nula, se puede asegurar que es estable. En el sistema estudiado de Tierra-Luna, se obtienen 5 puntos de Lagrange: L1, L2, L3, L4 y L5. Tan solo dos de ellos, L4 y L5 son estables, con órbitas periódicas.

3. Resultados

3.1. Puntos de Lagrange

Se han obtenido las siguientes coordenadas para los puntos de Lagrange:

- $L_1 = (0.83691309, 0)$
- $L_2 = (1.15568376, 0)$
- $L_3 = (-1.00506282, 0)$
- $L_4 = (0.487849, 0.8660254)$
- $L_5 = (0.487849, -0.8660254)$

3.2. Estabilidad puntos de Lagrange

Los autovalores del Jacobiano para cada punto de Lagrange son:

- $L_1 = (2.93206148, -2.93206148, 0, 0)$
- $L_2 = (2.1586705, -2.1586705, 0, 0)$
- $L_3 = (0, 0, -0.17787838, 0.17787838)$
- $L_4 = (-3.6 \times 10^{-7}, -3.6 \times 10^{-7}, 3.6 \times 10^{-7}, 3.6 \times 10^{-7})$
- $L_5 = (3.6 \times 10^{-7}, 3.6 \times 10^{-7}, -3.6 \times 10^{-7}, -3.6 \times 10^{-7})$

3.3. Órbitas alrededor de los puntos de Lagrange

Se representarán las órbitas para un tiempo de 500 segundos y 10^6 puntos de integración.

3.3.1. Runge-Kutta Embebido

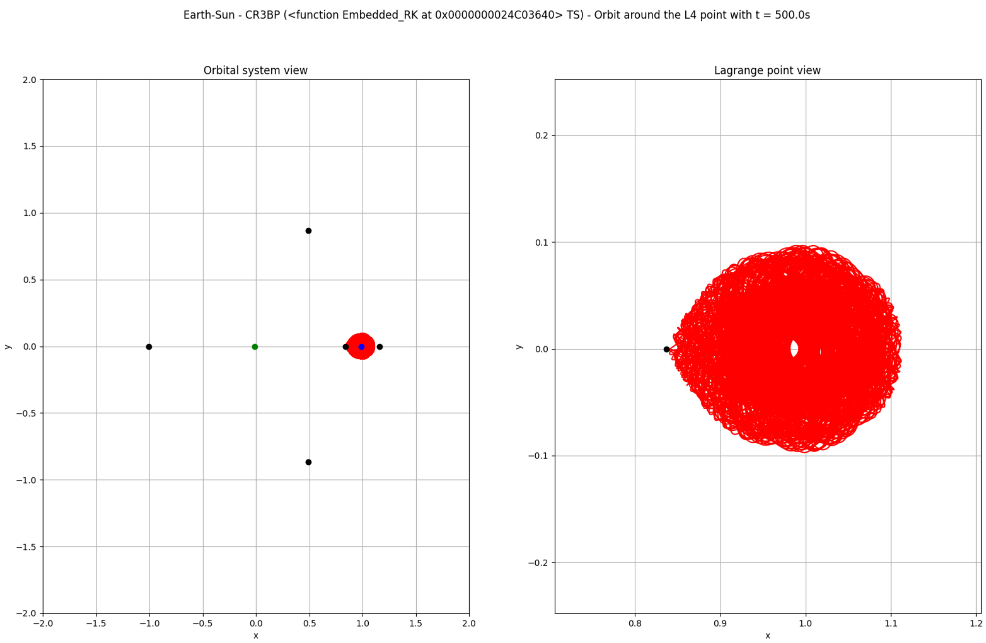

Figura 1: Órbita alrededor de L1 con Runge-Kutta Embebido

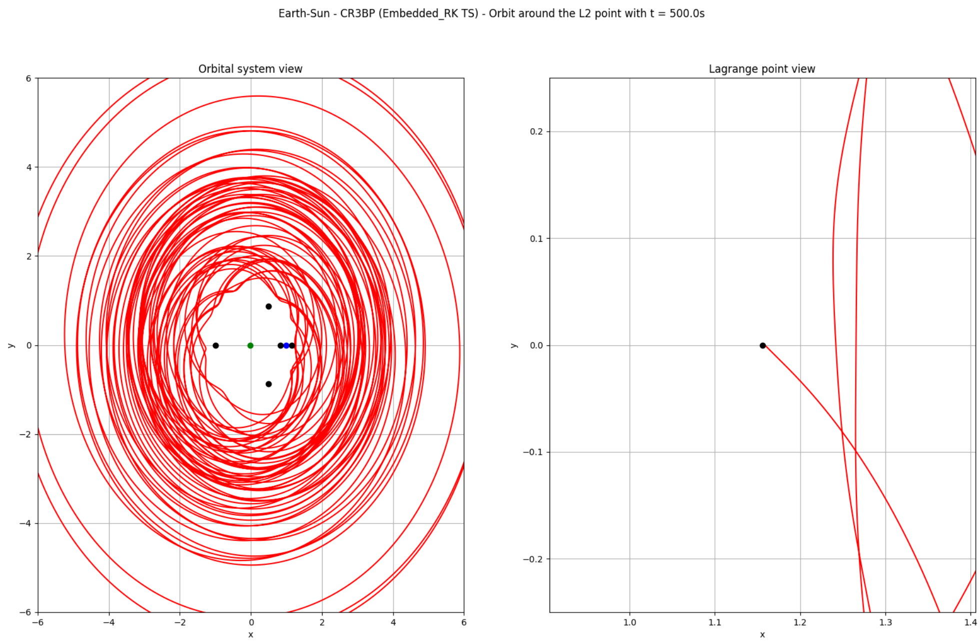

Figura 2: Órbita alrededor de L2 con Runge-Kutta Embebido

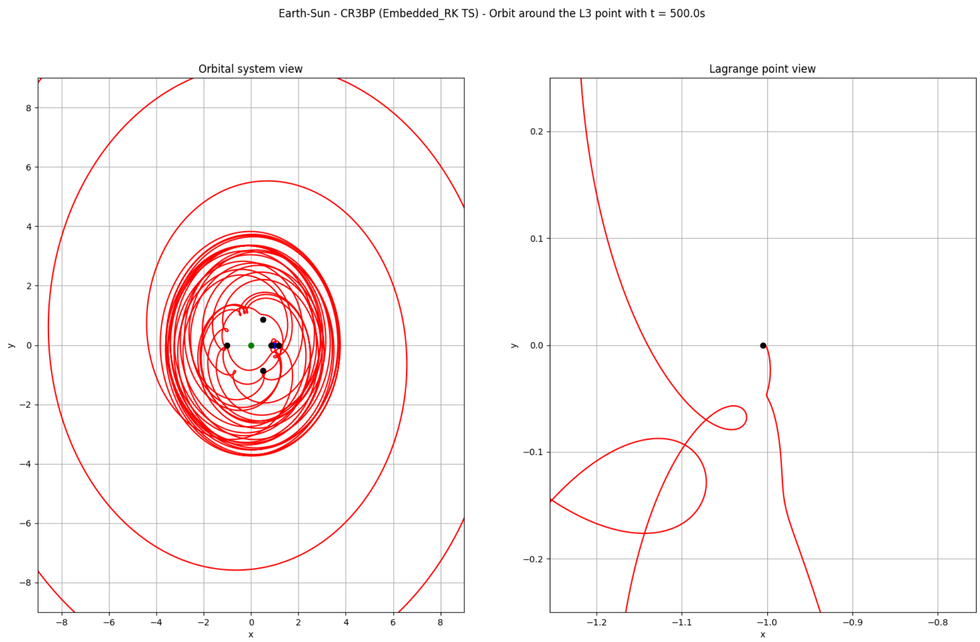

Figura 3: Órbita alrededor de L3 con Runge-Kutta Embebido

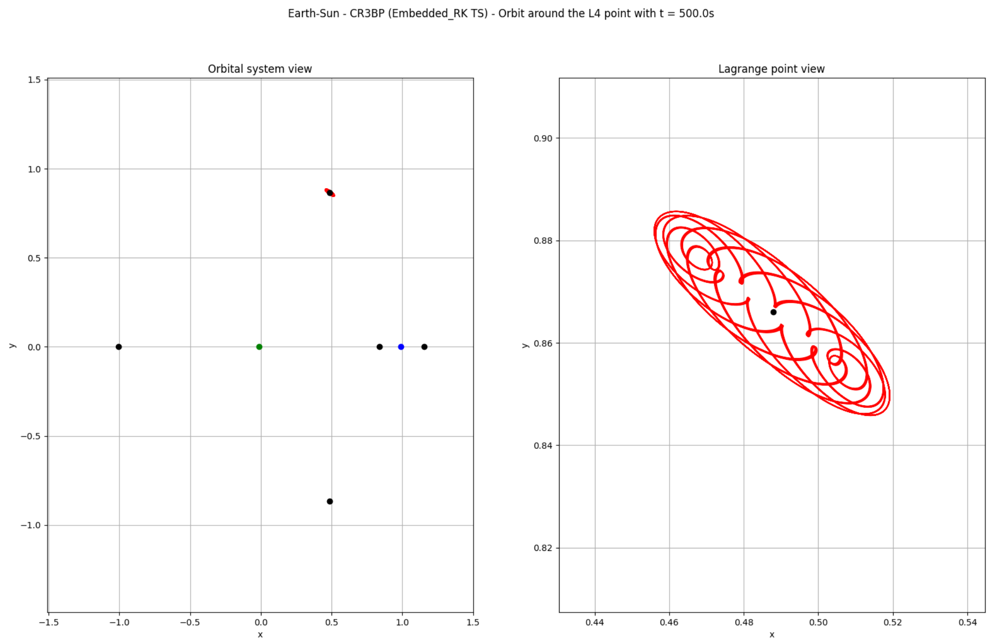

Figura 4: Órbita alrededor de L4 con Runge-Kutta Embebido

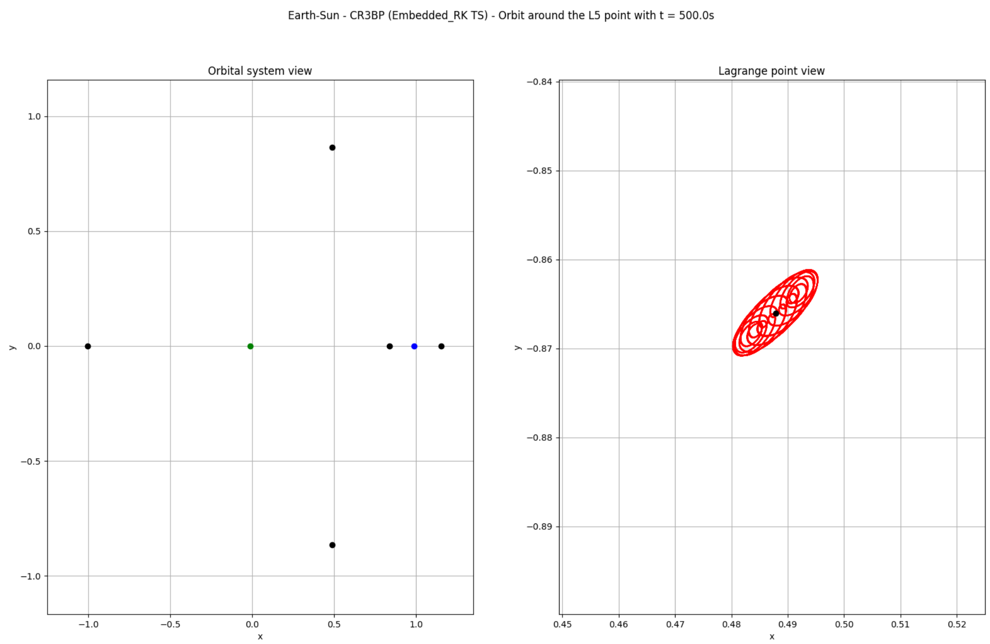

Figura 5: Órbita alrededor de L5 con Runge-Kutta Embebido

3.3.2. Otros esquemas numéricos

Se ha tomado el punto L2 para mostrar la diferencia de emplear otros esquemas temporales y observas las diferencias con el Runge-Kutta Embebido empleado anteriormente.

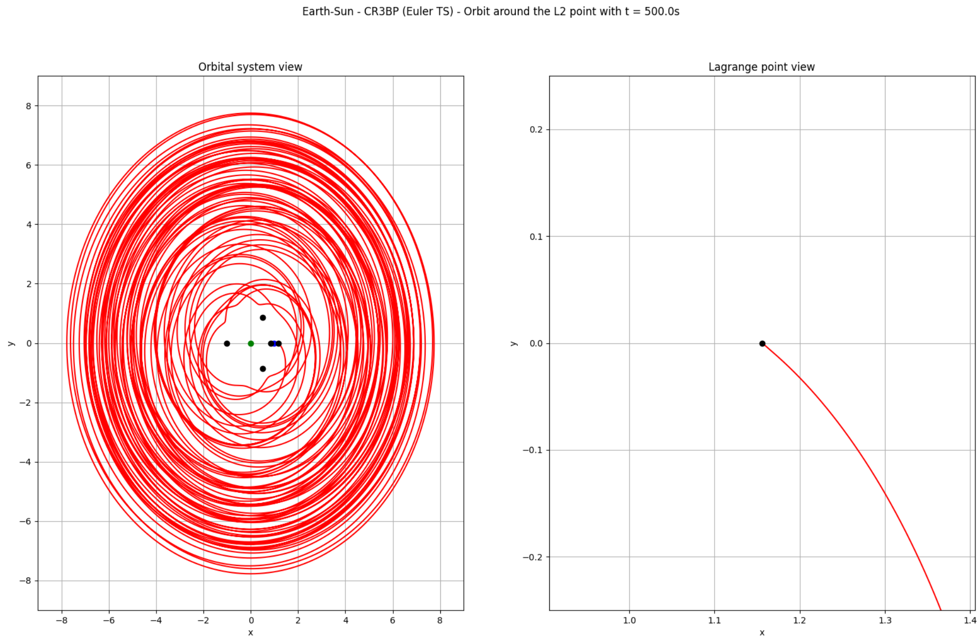

Figura 6: Órbita alrededor de L2 con Euler

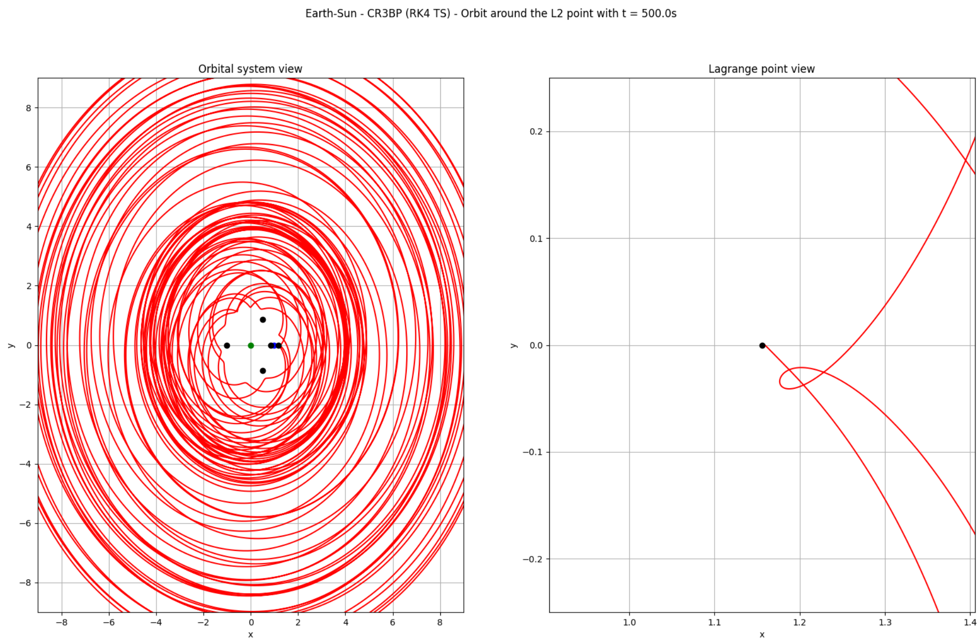

Figura 7: Órbita alrededor de L2 con Runge-Kutta orden 4

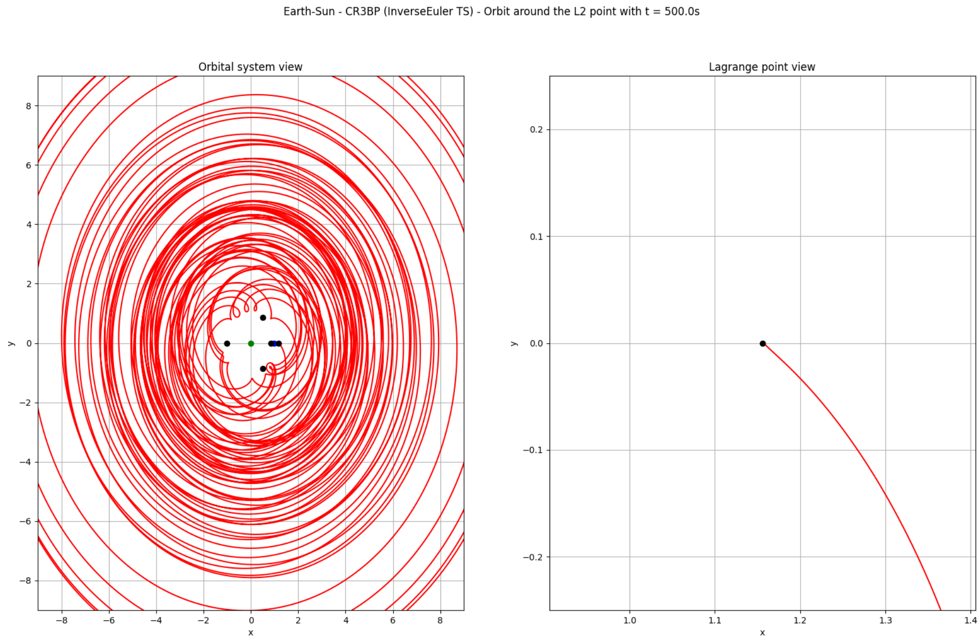

Figura 8: Órbita alrededor de L2 con Euler Inverso

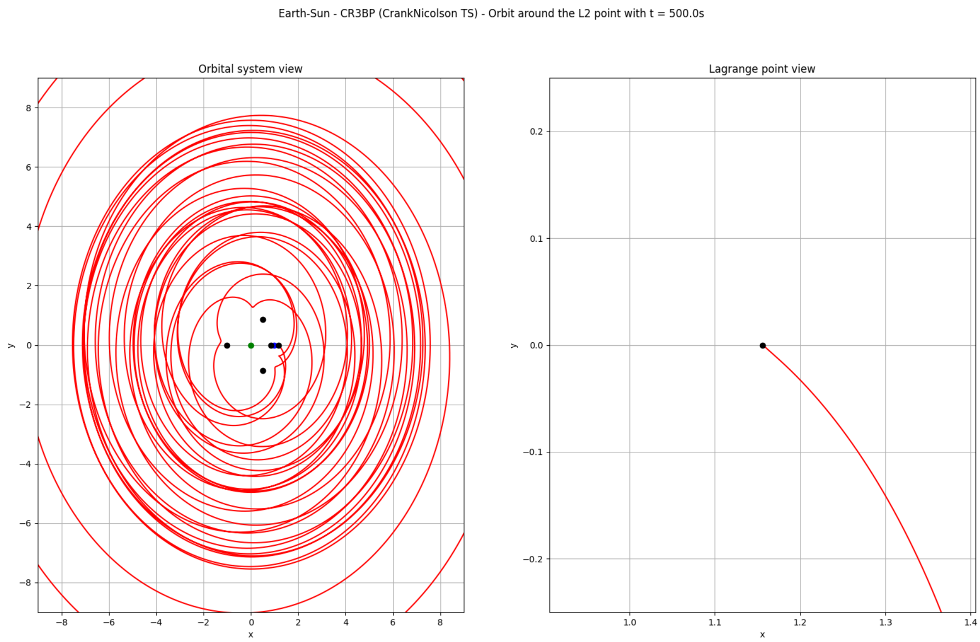

Figura 9: Órbita alrededor de L2 con Crank-Nicolson

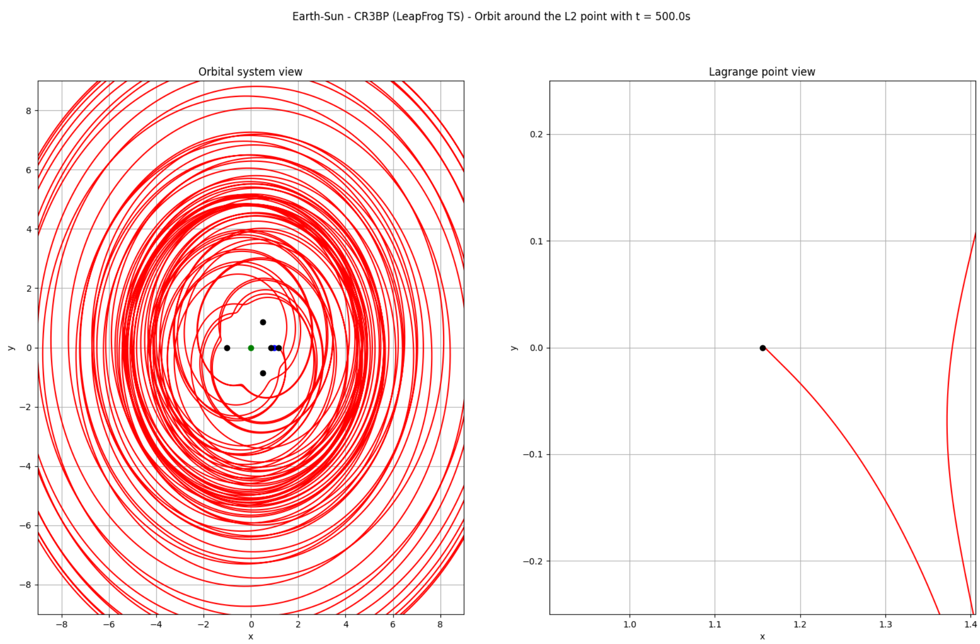

Figura 10: Órbita alrededor de L2 con Leap-Frog

4. Código

Empleando de nuevo la estructura Top-Down seguida en el resto de hitos de la asignatura, se emplea nuevamente la carpeta *Functions* donde se ordenarán las distintas funciones empleadas en este hito. Se emplean de nuevo los archivos de *Temporal_Schemes*, *Physics* y *Cauchy_Problem*. Adicionalmente, se ha definido un nuevo archivo denominado $Embedded_RK$ donde se definirá el esquema temporal de Runge-Kutta embebido. Además, en *Physics*, se implementará la función del *CR3BP* y tanto el cálculo de los puntos de Lagrange (*Lagrange_Points_Calculation*) como la función para determinar su estabilidad (*LP_Stability*).

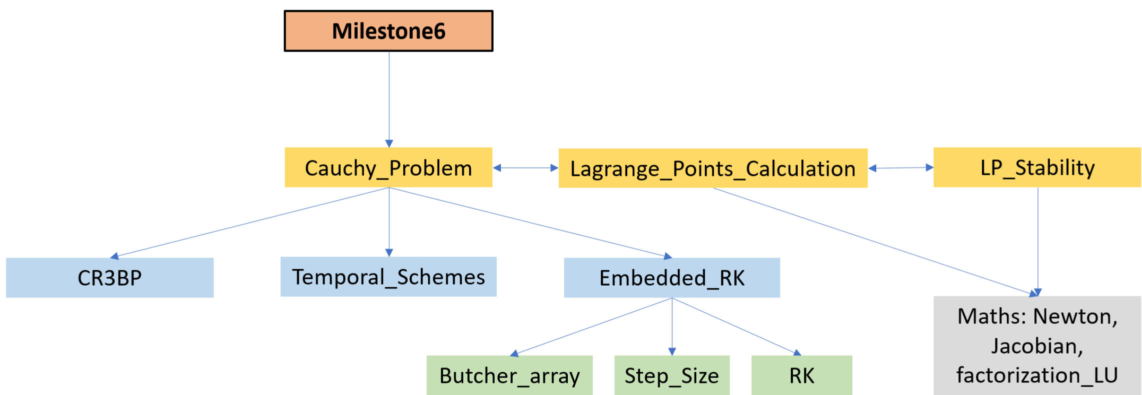

Figura 11: Estructura Top-Down del problema

4.1. Embedded_RK

```
from numpy import zeros, matmul, size
from numpy.linalg import norm

def Embedded_RK(U, dt, t, F):

    Embedded_RK.__name__ == "Embedded Runge-Kutta"
    RK_Method = "Dormand-Prince"
    tol = 1e-6

    V1 = RK_Scheme(RK_Method, "First", U, t, dt, F)
    V2 = RK_Scheme(RK_Method, "Second", U, t, dt, F)

    a, b, bs, c, q, Ne = Butcher_array(RK_Method)

    h = min(dt, Step_Size(V1-V2, tol, dt,  min(q)))

    N_n = int(dt/h)+1
    n_dt = dt/N_n

    V1 = U
    V2 = U

    for i in range(N_n):
        time = t + i*dt/int(N_n)
        V1 = V2
        V2 = RK_Scheme(RK_Method, "First", V1, time, n_dt, F)

    U2 = V2

    ierr = 0

    return U2

def RK_Scheme(name, tag, U1, t, dt, F):
    a, b, bs, c, q, N = Butcher_array(name)
    k = zeros([N, len(U1)])

    k[0,:] = F(U1, t + c[0]*dt)

    if tag=="First":

        for i in range(1,N):
            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 tag == "Second":

        for i in range(1,N):
            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 Step_Size(dU, tolerance, q, dt):

    normT = norm(dU)

    if normT > tolerance:
        step_size = dt*(tolerance/normT)**(1/(q+1))

    else:
        step_size = dt

    return step_size

def Butcher_array(Name: str):
    """This function generates the Butcher Tableau depending on the
   method selected
    Args:
        Name (str): Name of the method, options are: Heun-Euler, Bogacki
   -Shampine, Fehlberg-RK12, Dormand-Prince, Cash-Karp and Felberg.

    """
    if Name == "Heun-Euler":
        q = [2,1]          # Heun's order = 2, Eulers order = 1
        N = 2

        a = zeros([N,N-1])
        b = zeros([N])
        bst = zeros([N])
        c = zeros ([N])

        c = [0., 1.]

        a[0,:] = [  0. ]
        a[1,:] = [  1. , 0.]

        b[:]   = [0.5 , 0.5 ]
        bst[:] = [ 1.,    0.  ]

    elif  Name=="Bogacki-Shampine":
        q = [3,2]
        N = 4

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

        c[:] = [ 0., 1./2, 3./4, 1. ]
```

```
        a[0,:] = [  0., 0., 0.            ]
        a[1,:] = [ 1./2, 0., 0.           ]
        a[2,:] = [ 0.,  3./4, 0.      ]
        a[3,:] = [ 2./9,  1./3, 4./9  ]

        b[:]  = [ 2./9, 1./3, 4./9, 0. ]
        bs[:] = [ 7./24,  1./4, 1./3, 1./8 ]

    elif Name == 'Fehlberg-RK12':

        q = [2,1]
        N = 3

        a = zeros([N,N-1])
        b = zeros([N])
        bst = zeros([N])
        c = zeros([N])

        c[:] = [ 0., 0.5, 1. ]

        a[0,:] = [  0., 0. ]
        a[1,:] = [  1./2, 0. ]
        a[2,:] = [  1./256,  255./256 ]

        b[:]  = [ 1./256, 255./256, 0. ]
        bst[:] = [ 1./512,  255./256, 1./512 ]

    elif Name=="Dormand-Prince":
        q = [5,4]
        N = 7

        a = zeros([N,N-1])
        b = zeros([N])
        bst = zeros([N])
        c = zeros([N])

        c[:] = [ 0., 1./5, 3./10, 4./5, 8./9, 1., 1. ]

        a[0,:] = [          0.,           0.,           0.,          0.,
             0.,     0. ]
        a[1,:] = [      1./5  ,           0.,           0.,          0.,
             0.,     0. ]
        a[2,:]=  [      3./40 ,        9./40,           0.,          0.,
             0.,     0. ]
        a[3,:] = [     44./45 ,      -56./15,        32./9,          0.,
             0.,     0. ]
        a[4,:] = [ 19372./6561, -25360./2187,  64448./6561,  -212./729,
             0.,     0. ]
        a[5,:] = [  9017./3168,    -355./33 ,  46732./5247,    49./176,
    -5103./18656,     0. ]
        a[6,:]=  [    35./384 ,           0.,    500./1113,   125./192,
    -2187./6784 , 11./84 ]

        b[:]  = [ 35./384   , 0.,   500./1113, 125./192,  -2187./6784
    ,  11./84   ,      0.]
```

```
        bst[:] = [5179./57600, 0., 7571./16695,  393./640,
    -92097./339200, 187./2100, 1./40 ]

    elif Name =="Cash-Karp":
        q = [5,4]
        N = 6

        a = zeros([N,N-1])
        b = zeros([N])
        bst = zeros([N])
        c = zeros([N])

        c[:] = [ 0., 1./5, 3./10, 3./5, 1., 7./8 ]

        a[1,:] = [ 0.,          0.,        0.,          0.,            0.
    ]
        a[2,:] = [ 1./5,        0.,        0.,          0.,            0.
    ]
        a[3,:] = [ 3./40,       9./40,     0.,          0.,            0.
    ]
        a[4,:] = [ 3./10,      -9./10,     6./5,        0.,            0.
    ]
        a[5,:] = [ -11./54,     5./2,     -70./27,     35./27,         0.
    ]
        a[6,:] = [ 1631./55296, 175./512, 575./13824, 44275./110592,
    253./4096 ]

        b[:]  = [    37./378, 0.,      250./621,      125./594,
    0., 512./1771]
        bst[:] = [2825./27648, 0., 18575./48384, 13525./55296,
    277./14336,     1./4 ]

    elif Name == "Felberg":

        q = [5,4]
        N = 6

        a = zeros([N,N-1])
        b = zeros([N])
        bst = zeros([N])
        c = zeros([N])

        c[:] = [ 0., 1./4, 3./8, 12./13, 1., 0.5 ]

        a[1,:] = [ 0.,          0.,        0.,          0.,            0.
    ]
        a[2,:] = [ 1./4,        0.,        0.,          0.,            0.
    ]
        a[3,:] = [ 3./32,       9./32,     0.,          0.,            0.
    ]
        a[4,:] = [ 1932./2197, -7200./2197 , 7296./2197 ,  0.,         0.
    ]
        a[5,:] = [ 439./216,  -8 , 3680./513,      -845./4104,         0.
    ]
        a[6,:] = [ -8./27, 2.  , -3544./2565 , 1859./4104, -11./40 ]

```

```
        b[:]  = [ 16./135 , 0., 6656./12825, 28561./56430 , -9./50 , 2./55]
        bst[:] = [25./216 , 0., 1408./2565, 2197./4104 , -1./5,  0 ]



    return a, b, bst, c, q, N
```

Listing 1: Embedded_RK

4.2. CR3BP

```
    def CR3BP(U,t,mu):
    r_x, r_y = U[0], U[1] # Posicion
    v_x, v_y = U[2], U[3] # Velocidad

    r1 = sqrt((r_x + mu)**2 + r_y**2)
    r2 = sqrt((r_x - 1 + mu)**2 + r_y**2)

    dvdt_x = 2*v_y + r_x - ((1 - mu)*(r_x + mu))/(r1**3) - mu*(r_x - 1 + mu)/(r2**3)
    dvdt_y = -2*v_x + r_y -((1 - mu)/(r1**3) + mu/(r2**3))*r_y

    return array([v_x, v_y, dvdt_x, dvdt_y])
```

Listing 2: CR3BP

4.3. Lagrange_Points_Calculation

```
    def Lagrange_Points_Calculation(U0, NL, mu):

    LP = zeros([5,2])

    def F(Y):

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

    for i in range(NL):
        LP[i,:] = fsolve(F, U0[i,0:2])

    return LP
```

Listing 3: Lagrange_Points_Calculation

4.4. LP_Stability

```
    def LP_Stability(U0, mu):

    def F(Y):
        return CR3BP(Y, 0 , mu)

    A = Jacobian(F, U0)
    values, vectors = eig(A)

    return values
```

Listing 4: LP_Stability

4.5. Código principal: Milestone6

```
from Functions.Temporal_Schemes import Euler, Crank_Nicolson, RK4,
    Inverse_Euler, LeapFrog
from Functions.Cauchy_Problem import Cauchy_problem
from Functions.N_Body import F_NBody
from Functions.Physics import Kepler, CR3BP, Lagrange_Points_Calculation
    , LP_Stability
from Functions.EmbRK import Embedded_RK

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from numpy import array, linspace, zeros, size, linspace, reshape,
    around
from numpy.random import random

# Time definition
N = int(1e6) # time steps
t0 = 0 # timepo inicial
tf = 500 # tiempo final
#dt = [0.1, 0.01, 0.001]
#N = int(tf/dt[j])
t = linspace(t0, tf, N)

### CR3BP SOLUTION ###

#mu = 3.0039e-7   # Earth - Sun
mu = 1.2151e-2 # Earth - Moon

def F(U, t):
    return CR3BP(U, t, mu)

### LAGRANGE POINTS ###

NLP = 5 # Number of Lagrange Points

U0 = zeros([NLP,4])  # Initial values
U0[0,:] = array([0.8, 0.6, 0, 0])
U0[1,:] = array([0.8, -0.6, 0, 0])
U0[2,:] = array([-0.1, 0, 0, 0])
U0[3,:] = array([0.1, 0, 0, 0])
```

```
U0[4,:] = array([1.01, 0, 0, 0])

LagrangePoints = Lagrange_Points_Calculation(U0, NLP, mu)
print(LagrangePoints)
### ORBITS AROUND LP ###
U0LP = zeros(4)
U0SLP = zeros(4)

eps = 1e-3*random()
print('Choose Lagrange Point: 1, 2, 3, 4, 5')
sel_LP_ = int(input())  # Selected Lagrange Point
if sel_LP_ == 1:
    sel_LP = 4
elif sel_LP_ == 2:
    sel_LP = 5
elif sel_LP_ == 3:
    sel_LP = 3
elif sel_LP_ == 4:
    sel_LP = 1
elif sel_LP_ == 5:
    sel_LP = 2



U0LP[0:2] = LagrangePoints[sel_LP-1,:] + eps
U0LP[2:4] = eps

U0SLP[0:2] = LagrangePoints[sel_LP-1,:]
U0SLP[2:4] = 0

Temp_Schemes = [Euler, RK4, Crank_Nicolson, Inverse_Euler, LeapFrog,
   Embedded_RK]
T_S_list = ['Euler', 'RK4', 'CrankNicolson', 'InverseEuler', 'LeapFrog',
    'Embedded_RK']
colors = ['b','r','g','m','y','c']

### STABILITY OF LAGRANGE POINTS ###
print('Choose temporal scheme: Euler[0], RK4[1], Crank_Nicolson[2],
   Inverse_Euler[3], LeapFrog[4], Embedded_RK[5]')
k = int(input())
TS = Temp_Schemes[k]
T_S = T_S_list[k]
U_LP = Cauchy_problem(TS, F, t, U0LP)
eig = LP_Stability(U0SLP, mu)
print(around(eig.real,8))

### PLOT ###
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.plot(U_LP[:,0], U_LP[:,1],'-',color = "r")
ax1.plot(-mu, 0, 'o', color = "g")
ax1.plot(1-mu, 0, 'o', color = "b")

for i in range(NLP):
    ax1.plot(LagrangePoints[i,0], LagrangePoints[i,1] , 'o', color = "k"
   )

```

```
ax1.set_xlim(-9,9)
ax1.set_ylim(-9,9)
ax1.set_title("Orbital system view")

ax2.plot(U_LP[:,0], U_LP[:,1],'-',color = "r")
ax2.plot(LagrangePoints[sel_LP-1,0], LagrangePoints[sel_LP-1,1] , 'o',
    color = "k")


ax2.set_title("Lagrange point view")
ax2.set_xlim(LagrangePoints[sel_LP-1,0]-0.25,LagrangePoints[sel_LP
    -1,0]+0.25)
ax2.set_ylim(LagrangePoints[sel_LP-1,1]-0.25,LagrangePoints[sel_LP
    -1,1]+0.25)
fig.suptitle(f"Earth-Sun - CR3BP ({T_S} TS) - Orbit around the L{sel_LP_
    } point with t = " + str(t[N-1])+'s')

for ax in fig.get_axes():
    ax.set(xlabel='x', ylabel='y')
    ax.grid()

plt.show()
```

Listing 5: Milestone6 code

5. Conclusión

En cuanto al hito en cuestión, se ha presentado un nuevo esquema temporal como es el embebido el cuál a pesar de emplear mayores recursos computacionales, obtiene grandes resultados. Se puede observar así la mayor precisión de la órbita dibujada en comparación con el resto de esquemas representados. Adicionalmente, se podría implementar el sistema de Tierra-Sol para el cálculo de los apartados de este hito.

Este ha sido el último de los hitos individuales realizados en esta asignatura, y querría destacar el notable aprendizaje que he notado durante el desarrollo del cuatrimestre. Y no solo en el ámbito de la programación con el lenguaje Python si no también en el cálculo numérico. Al no haber cursado la especialidad de CTA, la gran mayoría de los conceptos presentados en la asignatura eran nuevos para mí. Este camino se inició presentando algunos de los esquemas temporales y ya en los últimos hitos ha sido posible ver su utilidad física en el ámbito espacial, lo cuál ha permitido que sea más satisfactorio el aprendizaje.