Universidad Politécnica de Madrid

Ampliación de Matemáticas I

Informe Hito 6

Autor:
Guillermo García del Río

Índice

Índice de figuras

Índice de tablas

1. Introducción

En este informe se presentan y comentan los resultados obtenidos en el hito 6 propuesto en la asignatura de Ampliación de Matemáticas I. En primer lugar se mostrará el enunciado que marca las pautas del proyecto, seguido de un análisis del problema a resolver, la exposición del código y los resultados obtenidos con él. Para terminar se ofrecerán las conclusiones obtenidas tanto de este Hito como del camino recorrido hasta él.

2. Enunciado del hito

Los objetivos de este hito quedan establecidos por los siguientes puntos:

1 Escribir un método de Runge-Kutta embebido de alto orden.

2 Escribir una función que simule el problema circular restringido de los tres cuerpos.

3 Determinar los puntos de Lagrange $F(U) = 0$.

4 Estabilidad de los puntos de Lagrange L1, L2, L3, L4, y L5.

5 Órbitas a alrededor de los puntos de Lagrange mediante diferente esquemas temporales.

3. Problema a resolver

3.1. Runge Kutta Embebido

El método numérico de Runge-Kutta embebido (ERK) es un método para solucionar ecuaciones diferenciales ordinarias (EDOs) de manera numérica. Está basado en el método de Runge-Kutta clásico con la ventaja de requerir menos cálculos para obtener una solución aproximada, la cual se obtiene mediante una interpolación polinómica de tercer orden, es decir, una interpolación cuadrática. Esta aproximación se realiza a partir de los valores de la EDO en tres puntos diferentes. El método consiste en aproximar la solución de la EDO en un punto dado, a partir de los valores de la EDO en los tres puntos anteriores.

Por otro lado, ERK es un método de paso variable, es decir, un método numérico que es capaz de ajustar el tamaño de paso en función de los requerimientos de precisión de la solución. Esto significa que el tamaño del paso puede ser ajustado en función de los errores que se estén cometiendo, lo cual permite obtener una solución mucho más precisa que los métodos de paso fijo. Para calcular dicho error se utilizan dos esquemas temporales (tipo Runge-Kutta) de orden q y $q - 1$.

La integración llevada a cabo por los dos esquemas es:

$$U_{n+1} = U_n + \Delta t_n \sum_{i=1}^{s} b_i k_i \qquad (\text{Orden} \quad q) \tag{3.1.1}$$

$$U^*_{n+1} = U_n + \Delta t_n \sum_{i=1}^{s} b^*_i k_i \qquad (\text{Orden} \quad q-1) \tag{3.1.2}$$

donde los k_i son los mismos para la solución de orden q y $q-1$:

$$k_i = F(U_n + \Delta t_n \sum_{j=1}^{s} a_{ij} k_j, t_n + c_i \Delta t_n) \tag{3.1.3}$$

Por lo tanto el error es:

$$e_{n+1} = U_{n+1} - U^*_{n+1} = \Delta t \sum_{i=1}^{s} (b_i - b^*_i) k_i \tag{3.1.4}$$

Que es $O(\Delta t^q)$.

Por último, los coeficientes a_i, b_i, b^*_i y c_i se obtienen de la matriz de Butcher, la cual cambia dependiendo de los métodos empleados.

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \\ & b^*_1 & b^*_2 & \dots & b^*_s \end{array} \tag{3.1.5}$$

Con todo esto, el paso de tiempo mínimo a utilizar se calcula como:

$$\Delta t_{min} = \Delta t \left(\frac{\epsilon}{\|U^*_{n+1} - U_{n+1}\|} \right)^{\frac{1}{q}} \tag{3.1.6}$$

Siendo ϵ el valor de la tolerancia.

En el código desarrollado se han incluido las siguientes matrices:

- Heun - Euler
- Bogacki - Shampine
- Fehlberg - RK12
- Dormand - Prince
- Cash - Karp
- Felberg

3.2. Problema circular restringido de los tres cuerpos

El problema circular restringido de los tres cuerpos (CRTBP) reduce el problema de los tres cuerpos general mediante dos asunciones principales, específicamente que dos partículas masivas se mueven en círculos alrededor de su centro de masas mientras que atraen a una tercera masa infinitesimal a la cual no se sienten atraídos. Siendo conocidas las órbitas y las masas de cada una de las partículas masivas, el problema se reduce a determinar el alcance del posible movimiento de la tercera partícula. Esta simplificación reduce significativamente el orden del problema sin dejar de proporcionar una buena aproximación para varios problemas dentro de la astrodinámica, como por ejemplo una nave en las proximidades de la Tierra y la Luna.

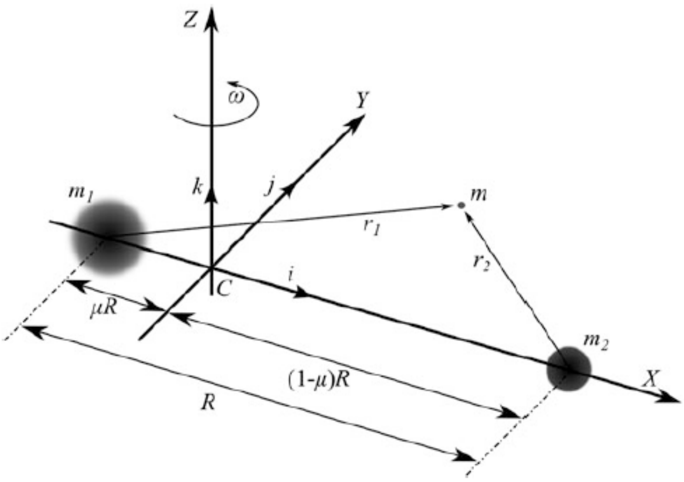

Figura 3.1: CRTBP usando un marco de referencia sinódico

Usando un marco de referencia sinódico el CRTBP se ilustra en la **Figura** 3.1. Se decide dar valor unidad a los parámetros de la distancia entre las dos partículas masivas, R, la masa total del sistema, M. De manera similar, la unidad de tiempo será tal para que la constante gravitacional G valga uno. La masa total del sistema es:

$$M = m_1 + m_2 \tag{3.2.1}$$

Mientras que el ratio de masas es:

$$\mu = \frac{m_2}{m_1 + m_2} \tag{3.2.2}$$

Cumpliéndose:

$$\mu \leq \frac{1}{2} \quad , \quad m_1 > m_2 \tag{3.2.3}$$

Con todo esto, las ecuaciones del movimiento de la partícula de masa infinitesimal se pueden derivar como:

$$\ddot{x} = 2\dot{y} + x - (1-\mu)\frac{(x+\mu)}{r_1^3} - \mu\frac{(x-(1-\mu))}{r_2^3}$$
$$\ddot{y} = -2\dot{x} + y - \left(\frac{1-\mu}{r_1^3} + \frac{\mu}{r_2^3}\right) y \tag{3.2.4}$$

Donde r_1 y r_2 vienen definidos como:

$$r_1 = \sqrt{(x+\mu)^2 + y^2} \qquad r_2 = \sqrt{(x-1+\mu)^2 + y^2} \tag{3.2.5}$$

Este sistema tendrá el siguiente potencial:

$$U = \frac{1}{2}\left(x^2 + y^2\right) + \frac{1-\mu}{r_1} + \frac{\mu}{r_2} \tag{3.2.6}$$

En la **Tabla** 3.1 se ofrecen distintos valores de μ vara posibles sistemas de cuerpos celestiales de interés.

Tabla 3.1: Valores de μ para diferentes sistemas

Sistema	**Valor de μ**
Tierra - Sol	$3{,}0039 \cdot 10^{-7}$
Tierra - Luna	$1{,}2151 \cdot 10^{-2}$
Júpiter - Sol	$7{,}1904 \cdot 10^{-4}$
Saturno - Sol	$2{,}8571 \cdot 10^{-4}$
Saturno - Titan	$2{,}366 \cdot 10^{-4}$

3.3. Puntos de Lagrange

Existen cinco puntos de equilibrio para el sistema de los tres cuerpos. Estos puntos cumplen las siguientes condiciones:

1. La fuerza resultante sobre cada masa pasa por el centro de masa del sistema.
2. Esta fuerza resultante es directamente proporcional a la distancia de cada masa al centro de masa.
3. Los vectores de velocidad iniciales son proporcionales en magnitud a las respectivas distancias de las partículas desde el centro de masa, y forman ángulos iguales con los radios vectores vectores a las partículas desde el centro de masa.

Matemáticamente esto significa que en el equilibrio el potencial de los tres cuerpos definido en la ecuación 3.2.6 ha de ser cero, es decir:

$$\frac{\partial U}{\partial x} = x - (1-\mu)\frac{(x+\mu)}{r_1^3} - \mu\frac{(x-(1-\mu))}{r_2^3} = 0 \tag{3.3.1}$$

$$\frac{\partial U}{\partial y} = \left(1 - \frac{1-\mu}{r_1^3} + \frac{\mu}{r_2^3}\right) y = 0 \tag{3.3.2}$$

Esto proporciona cinco puntos de equilibrio, conocidos como los puntos de Lagrange, los cuales se muestran en la **Figura** 3.2.

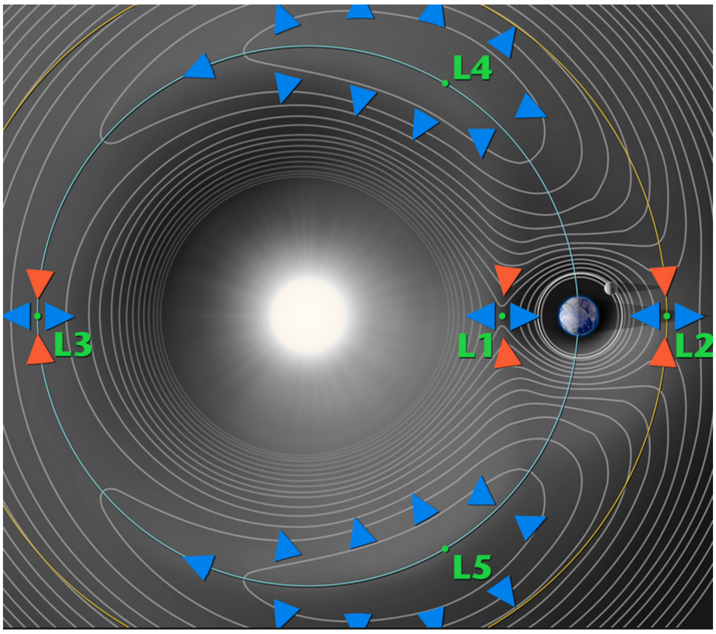

Figura 3.2: Representación de los puntos de Lagrange en un mapa de contorno del potencial

En cuanto a la estabilidad de los puntos de Lagrange, los puntos L1, L2 y L3 son normalmente inestables, lo cual se aprecia en la figura por ser puntos de silla. De todas formas, existen órbitas periódicas quasi-estables llamadas *órbitas halo* que siguen trayectorias *Lissajous*.

Por otro lado, los puntos L4 y L5, mostrados en la figura como cimas de colinas son estables. La razón de la estabilidad es un efecto de segundo orden: a medida que un cuerpo se aleja de la posición exacta de Lagrange, la aceleración de Coriolis (que depende de la velocidad de un objeto en órbita y no puede modelarse como un mapa de contorno) curva la trayectoria hacia una trayectoria alrededor (en lugar de alejarse) del punto. Dado que la fuente de estabilidad es la fuerza de Coriolis, las órbitas resultantes pueden ser estables, pero generalmente no son planas, sino "tridimensionales": se sitúan sobre una superficie alabeada que interseca el plano de la eclíptica. Las órbitas en forma de riñón que suelen mostrarse anidadas alrededor de L4 y L5 son las proyecciones de las órbitas sobre un plano (por ejemplo, la eclíptica) y no las órbitas tridimensionales completas.

4. Exposición del código

De cara a poder entender lo que se está haciendo se va a proceder a exponer el código desarrollado para solucionar el problema propuesto. Una visión general de la dependencia entre módulos y archivos se muestra en la figura 4.1:

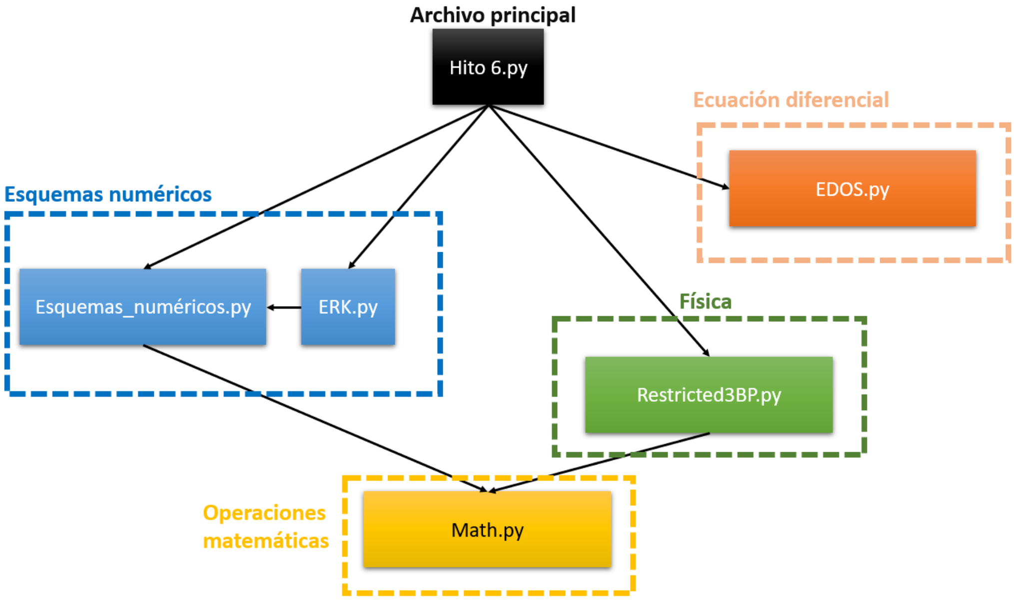

Figura 4.1: Dependencias entre archivos

Este diagrama hay que entenderlo como una ruta que define a quién llama cada archivo.

4.1. Código del problema circular restringido de los tres cuerpos

```python

def CR3BP(U, t, mu):

    x, y = U[0], U[1]        # The first two components are the position coordinates
    vx, vy = U[2], U[3]      # The last two components velocity

    r1 = sqrt((x + mu)**2 + y**2)         # Distance from the third body to the mass m1
    r2 = sqrt((x - 1 + mu)**2 + y**2)     # Distance from the third body to the mass m2

    dxdt = vx
    dydt = vy

    dvxdt = 2*vy + x - ((1 - mu)*(x + mu))/(r1**3) - mu*(x + mu - 1)/(r2**3)
    dvydt = -2*vx + y -((1 - mu)/(r1**3) + mu/(r2**3))*y


    return array([dxdt, dydt, dvxdt, dvydt])
```

Algoritmo 4.1: Problema circular restringido de los tres cuerpos

4.2. Código relacionado con los puntos de Lagrange

```python

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,:] = newton(F, U0[i,0:2])

    return LP

def Lagrange_Points_Stability(U0, mu):

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

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

    return values
```

Algoritmo 4.2: Cálculos relacionados con los puntos de Lagrange

4.3. Código del Runge-Kutta embebido

```

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

    ERK.__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_Tableau(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_Tableau(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_Tableau(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])
```

```python
        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
```

Algoritmo 4.3: Código del Runge-Kutta embebido

4.4. Código principal del Hito 6

```python

import matplotlib.pyplot as plt

from numpy import array, linspace, zeros, size, around
from random import random
from Numeric.Esquemas_numéricos import (RK4, Crank_Nicolson, Euler,
                                        Euler_inverso, leapfrog)
from Numeric.ERK import ERK
from Physics.Restricted3BP import CR3BP, Lagrange_Points_Calculation,
    Lagrange_Points_Stability
from Mathematics.EDOS import Cauchy_Problem


## Temporal variables ##

T = 1000                                # Integration duration [s]
n = int(1e6)                            # Number of Points
```

```
t = linspace(0,T,n)                           # Time array


######### Circular Restricted Three Body Problem Resolution #########

mu = 3.0039e-7  # Earth - Sun
#mu = 1.2151e-2 # Earth - Moon
#mu = 7.1904e-4 # Jupiter - Sun
#mu = 2.8571e-4 # Saturn - Sun
#mu = 2.366e-4  # Saturn - Titan

def F(U,t):

    return CR3BP(U, t, mu)

######### Lagrange Points #########

NL = 5     # Number of Lagrange Points

U0 = zeros([NL,4])  # Assigning the initial values for the system resolution

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, NL, mu)

######### Orbits around Lagrange points #########

U0LP = zeros(4)
U0stabLP = zeros(4)
eps = 1e-3*random()
Lagrange_Points_List = array([1,2,3,4,5])
for k in range(5):

    selectedLP = k + 1

    if selectedLP == 5:
        label = 'L2'
    elif selectedLP == 4:
        label = 'L1'
    elif selectedLP == 3:
        label = 'L3'
    elif selectedLP == 2:
        label = 'L5'
    elif selectedLP == 1:
        label = 'L4'

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

    U0stabLP[0:2] = LagrangePoints[selectedLP-1,:]
    U0stabLP[2:4] = 0

    eig = Lagrange_Points_Stability(U0stabLP, mu)
    print(around(eig.real,8))

    methods = [Euler,RK4, Crank_Nicolson, Euler_inverso, leapfrog, ERK]

    for j in range (size(methods)):

        U_LP = Cauchy_Problem(F, t, U0LP, methods[j])

        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(NL):
            ax1.plot(LagrangePoints[i,0], LagrangePoints[i,1] , 'o', color = "k")

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

        ax1.set_xlim(-2,2)
        ax1.set_ylim(-2,2)
        ax1.set_title("Vista del sistema orbital")
        ax2.set_title("Vista del punto de Lagrange")
        ax2.set_xlim(LagrangePoints[selectedLP - 1,0]-0.02,LagrangePoints[selectedLP - 1,0]+0.02)
        ax2.set_ylim(LagrangePoints[selectedLP - 1,1]-0.02,LagrangePoints[selectedLP - 1,1]+0.02)
        fig.suptitle(f"Tierra-Sol - CR3BP ({methods[j].__name__}) - Órbita alrededor de {label} con t = {T}s" )
        for ax in fig.get_axes():
            ax.set(xlabel='x', ylabel='y')
            ax.grid()

        manager = plt.get_current_fig_manager()
        manager.full_screen_toggle()
        figure = plt.gcf()
        figure.set_size_inches(16, 8)
        plt.savefig('Plots/Hito 6/ CR3BP ' + label +' '+ methods[j].__name__ +'.png', bbox_inches = 'tight')
        plt.close('all')
        #plt.show()
```

Algoritmo 4.4: Código principal del Hito 6

5. Resultados

Para la obtención de resultados se ha decidido llevar a cabo simulaciones con las características mostradas en la **Tabla** 5.1.

Tabla 5.1: Propiedades temporales de la integración

Propiedad	Valor	Unidades
Tiempo total	1000	s
Puntos de integración	10^6	-
Paso de integración	0,001	s

5.1. Órbita alrededor de L1

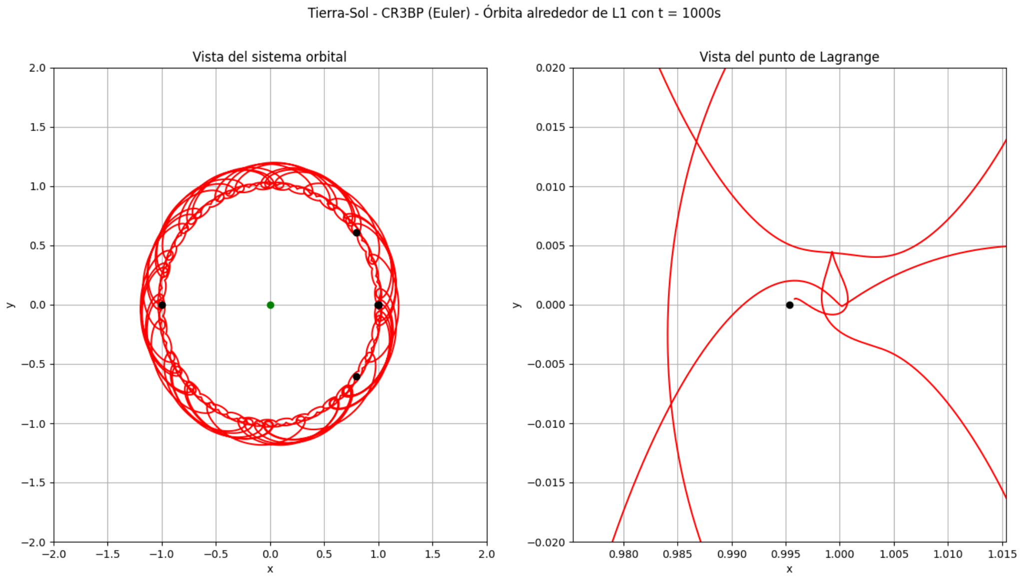

Figura 5.1: Órbita alrededor de L1 utilizando Euler

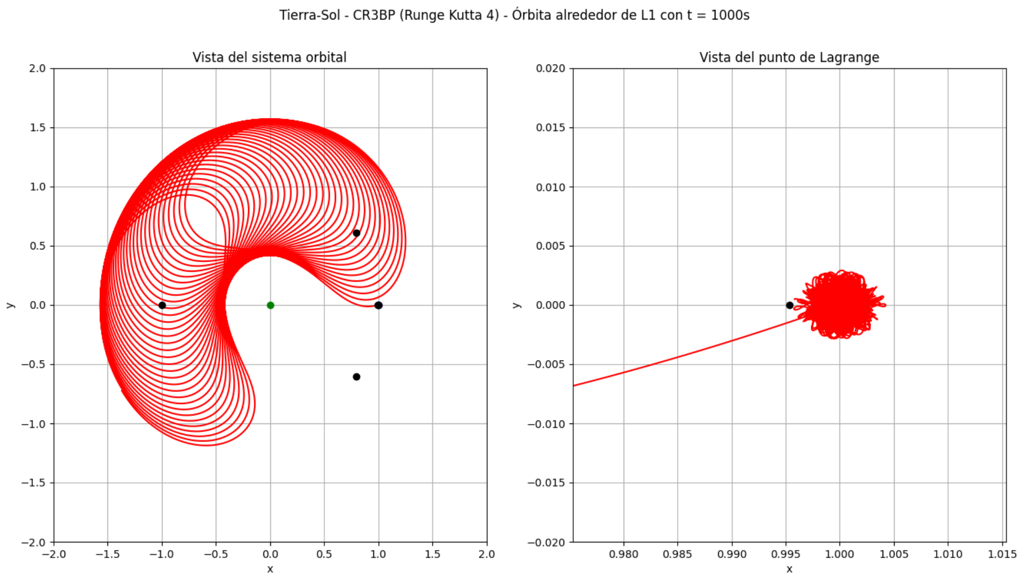

Figura 5.2: Órbita alrededor de L1 utilizando Runge-Kutta 4

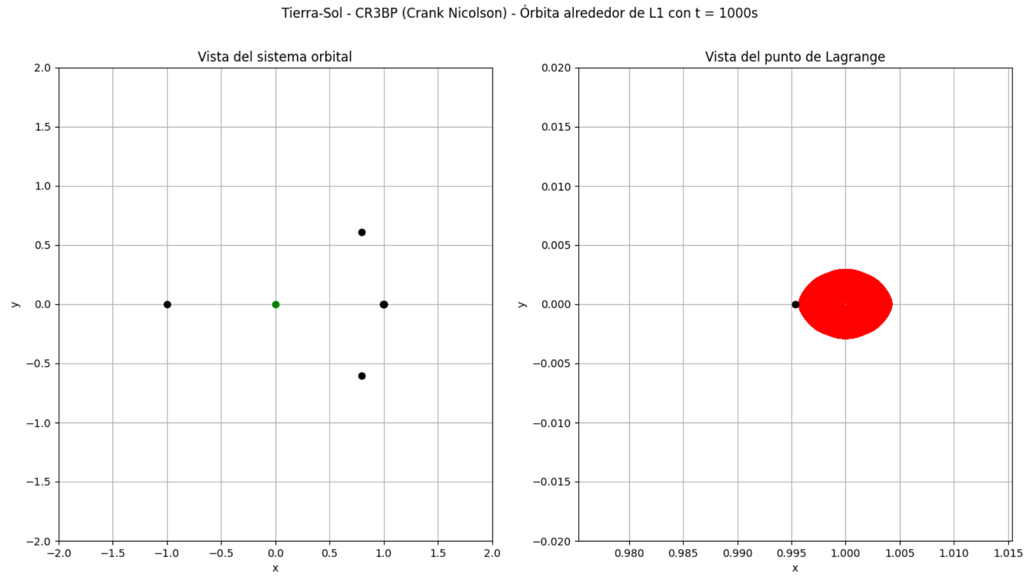

Figura 5.3: Órbita alrededor de L1 utilizando Crank Nicolson

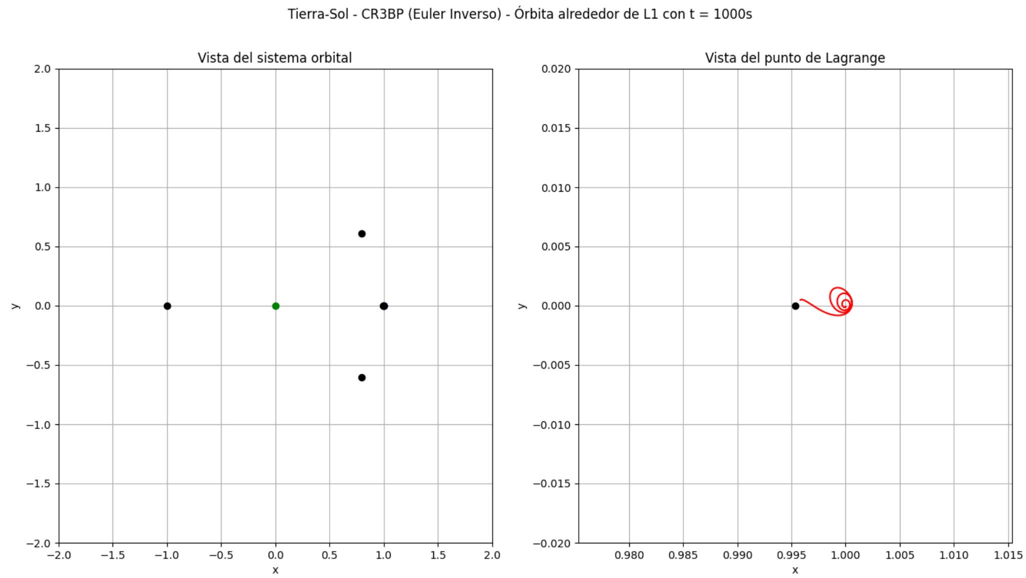

Figura 5.4: Órbita alrededor de L1 utilizando Euler inverso

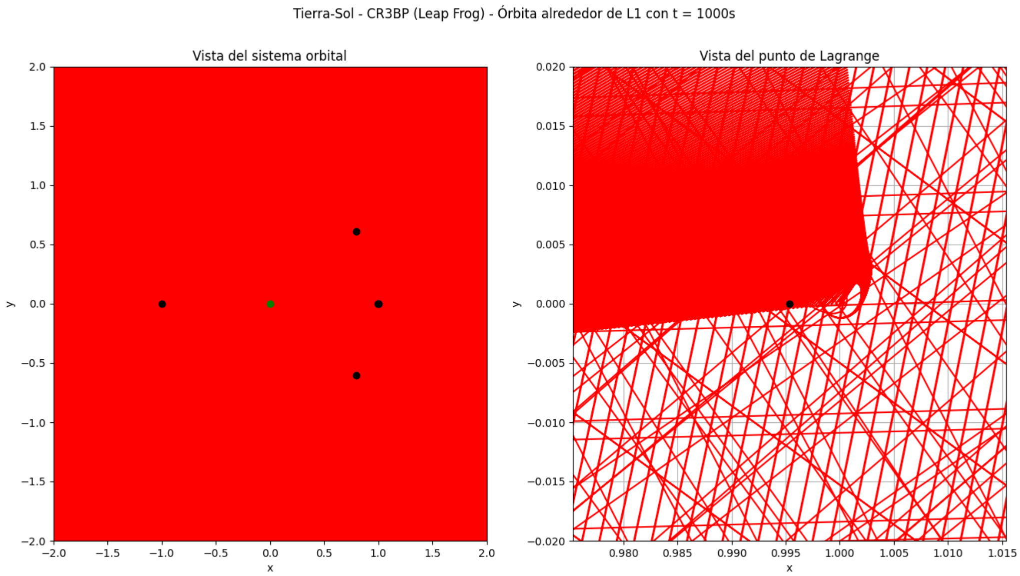

Figura 5.5: Órbita alrededor de L1 utilizando Leap Frog

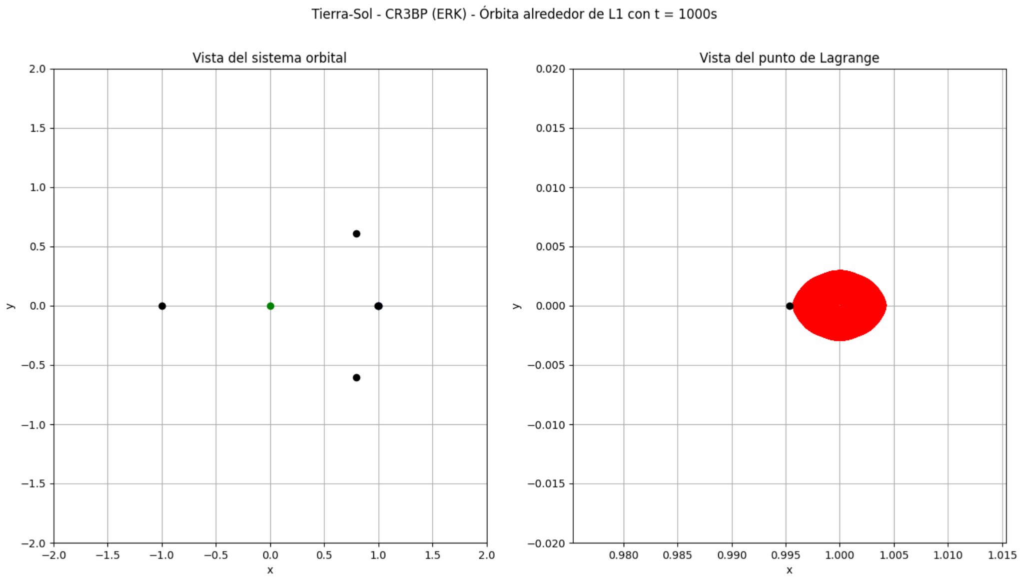

Figura 5.6: Órbita alrededor de L1 utilizando Runge-Kutta Embebido

5.2. Órbita alrededor de L2

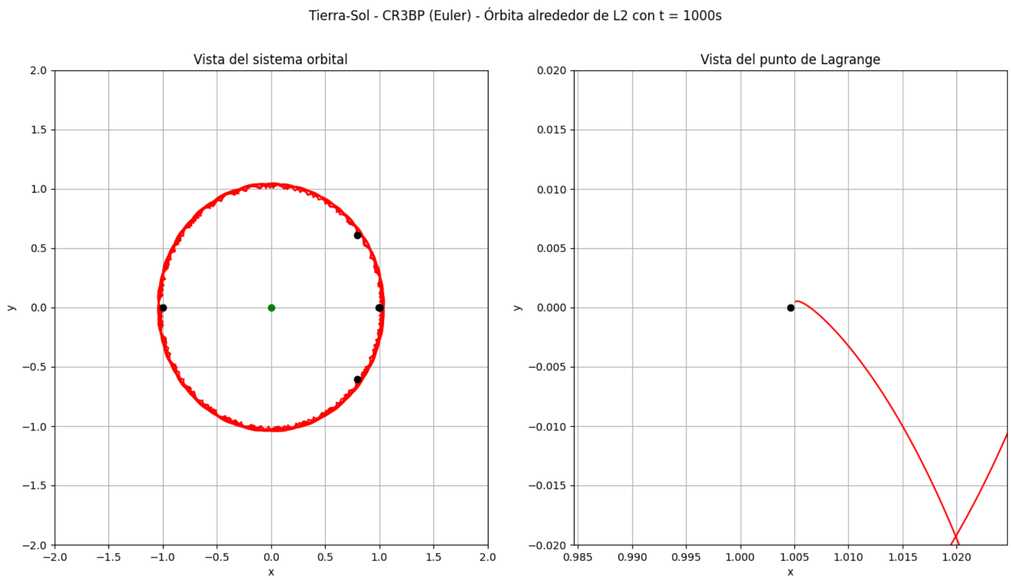

Figura 5.7: Órbita alrededor de L2 utilizando Euler

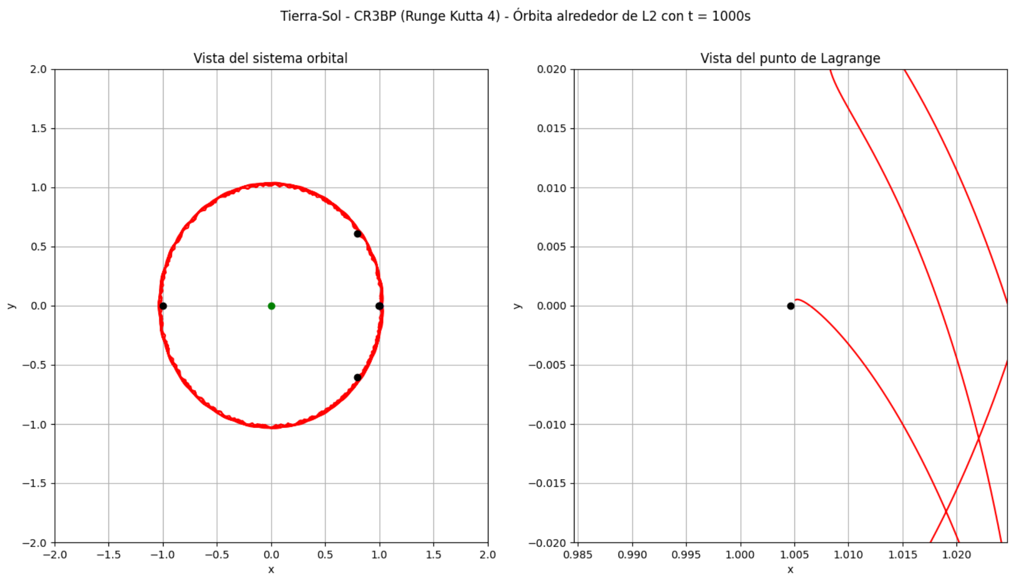

Figura 5.8: Órbita alrededor de L2 utilizando Runge-Kutta 4

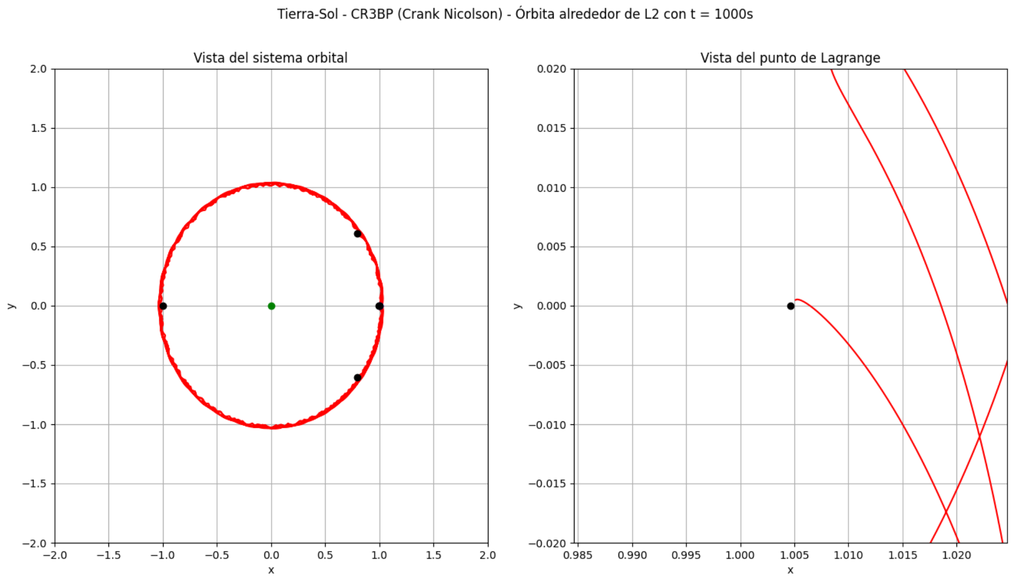

Figura 5.9: Órbita alrededor de L2 utilizando Crank Nicolson

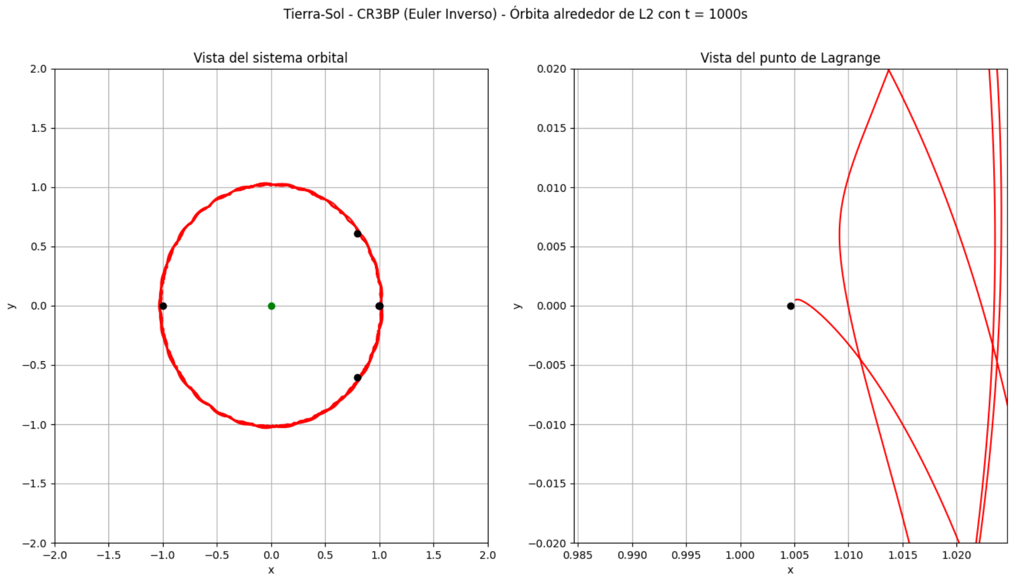

Figura 5.10: Órbita alrededor de L2 utilizando Euler inverso

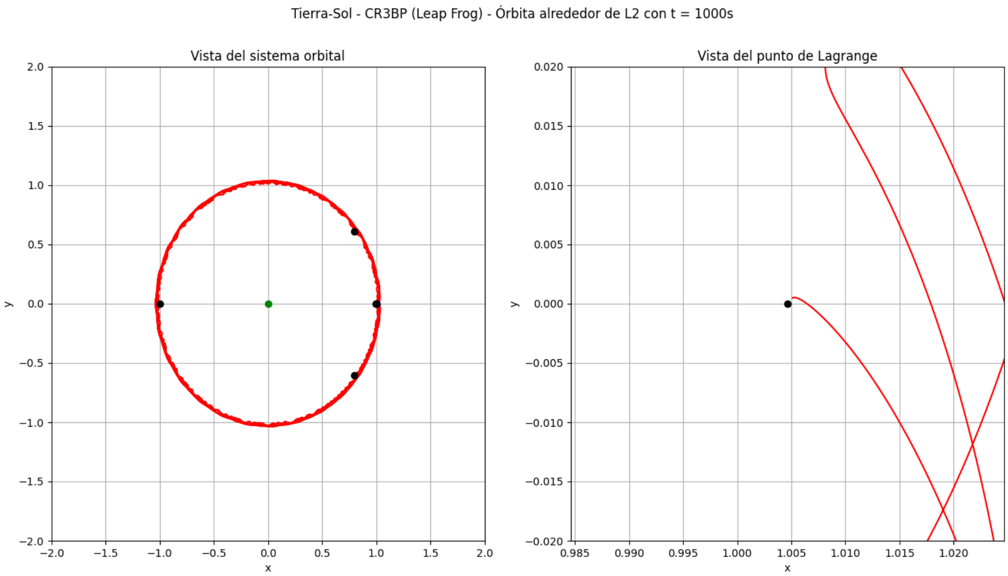

Figura 5.11: Órbita alrededor de L2 utilizando Leap Frog

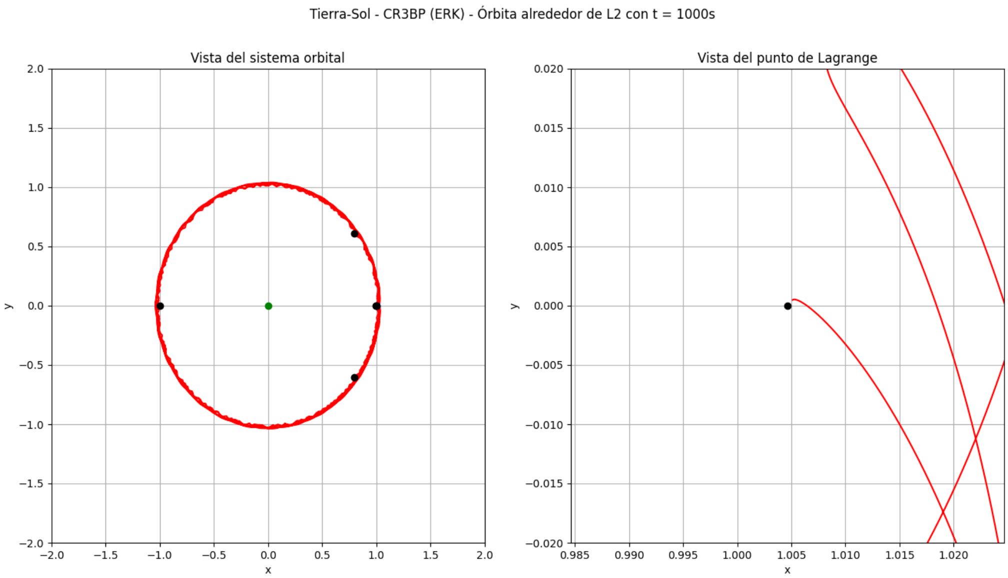

Figura 5.12: Órbita alrededor de L2 utilizando Runge-Kutta Embebido

5.3. Órbita alrededor de L3

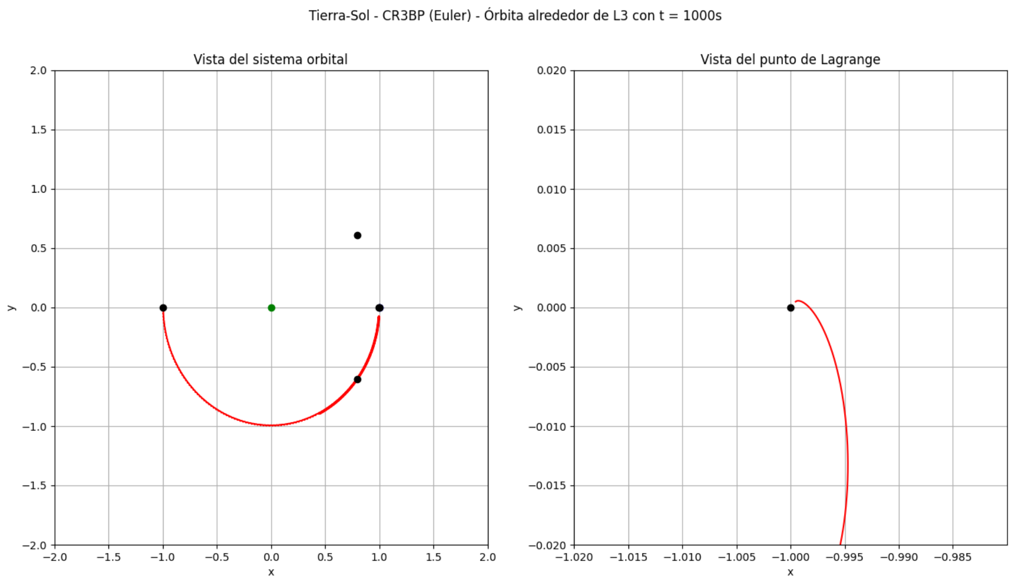

Figura 5.13: Órbita alrededor de L3 utilizando Euler

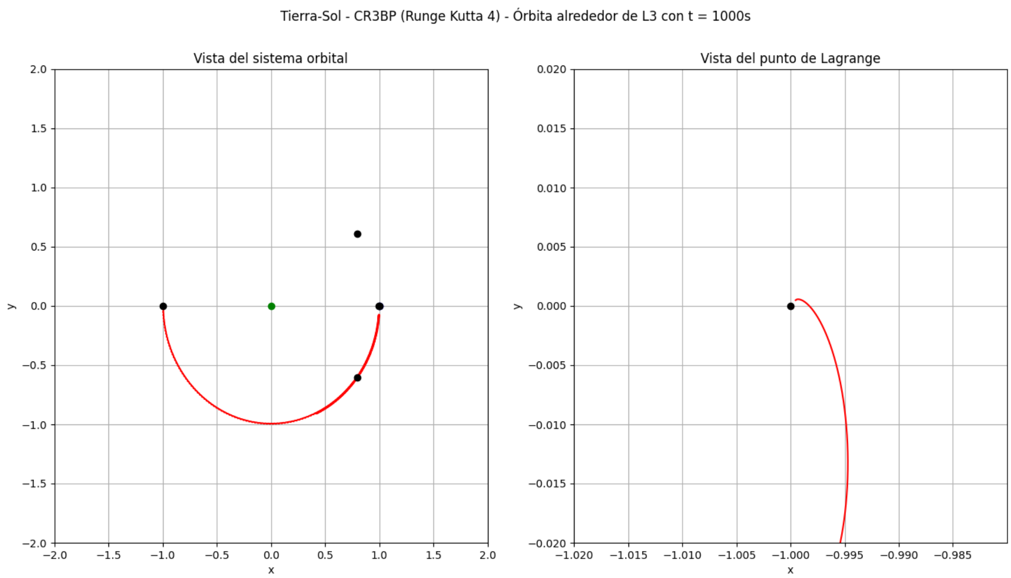

Figura 5.14: Órbita alrededor de L3 utilizando Runge-Kutta 4

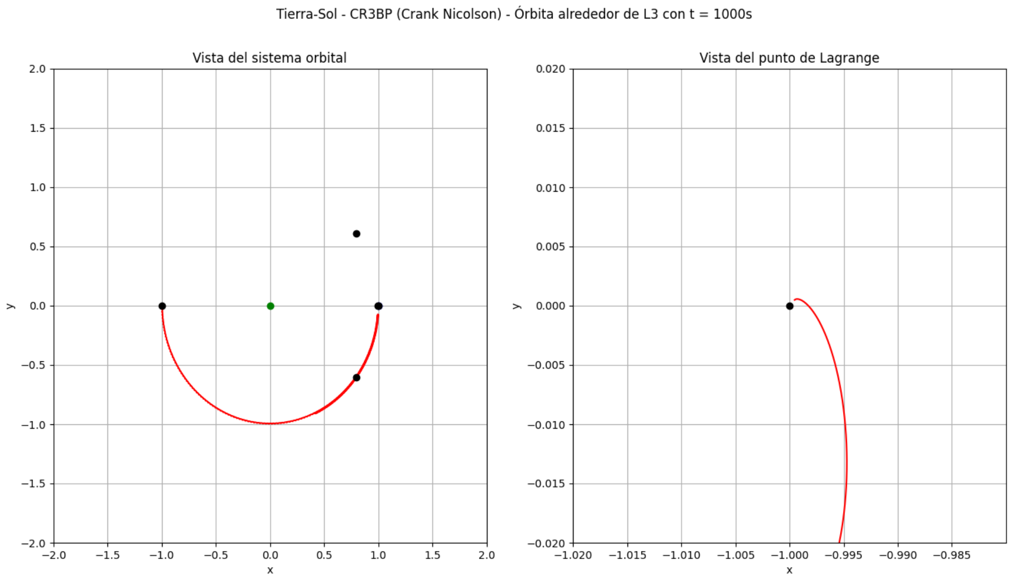

Figura 5.15: Órbita alrededor de L3 utilizando Crank Nicolson

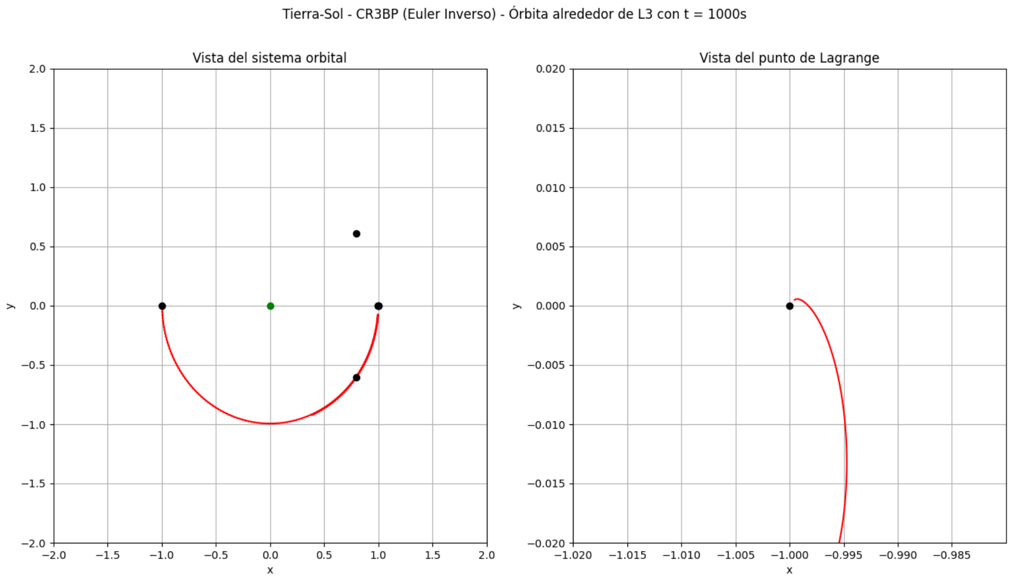

Figura 5.16: Órbita alrededor de L3 utilizando Euler inverso

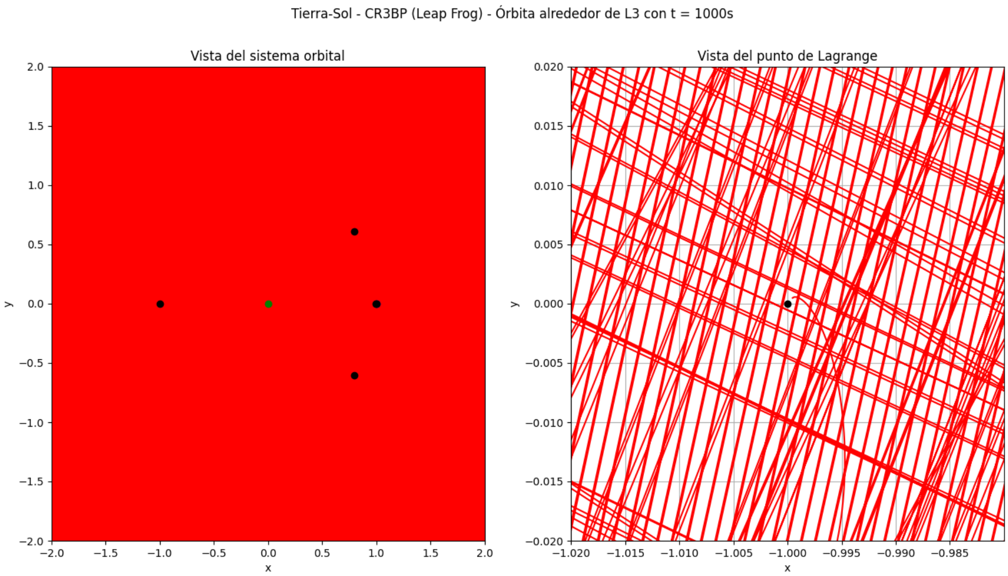

Figura 5.17: Órbita alrededor de L3 utilizando Leap Frog

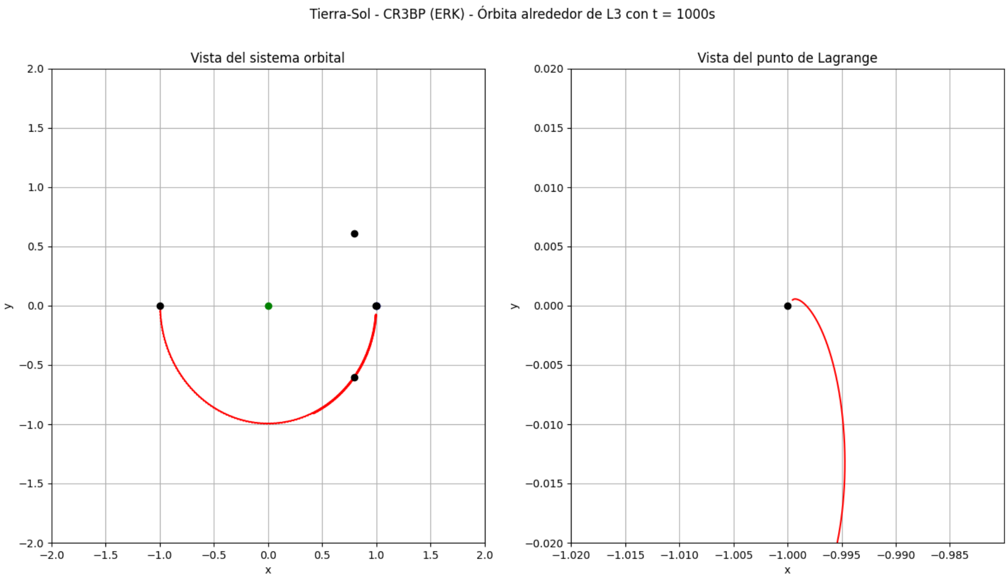

Figura 5.18: Órbita alrededor de L3 utilizando Runge-Kutta Embebido

5.4. Órbita alrededor de L4

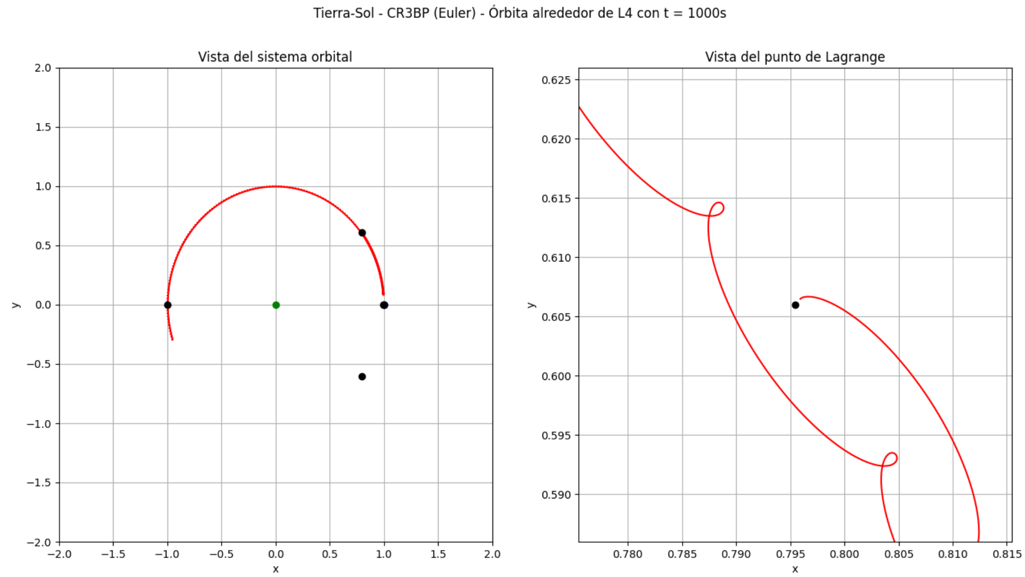

Figura 5.19: Órbita alrededor de L4 utilizando Euler

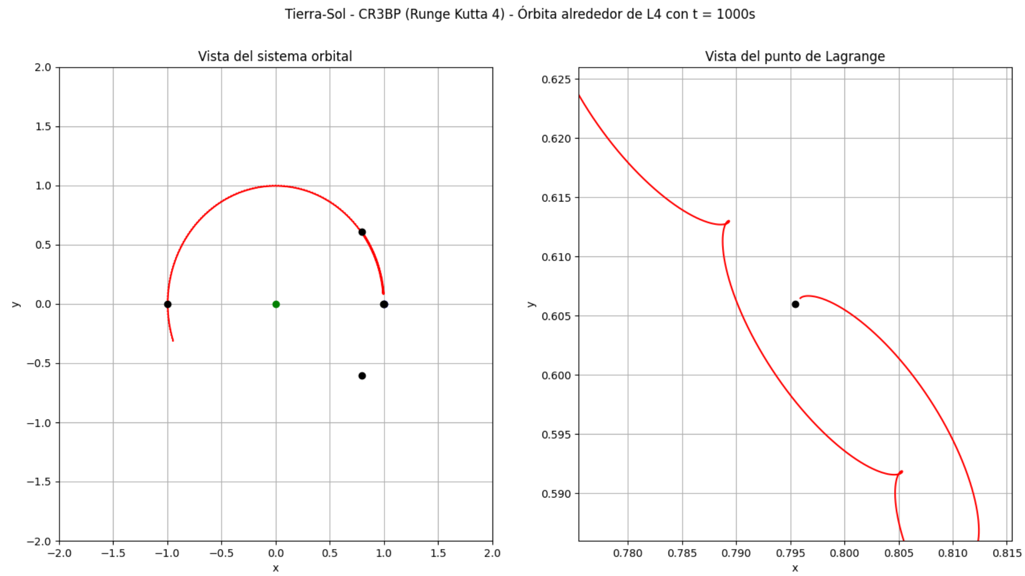

Figura 5.20: Órbita alrededor de L4 utilizando Runge-Kutta 4

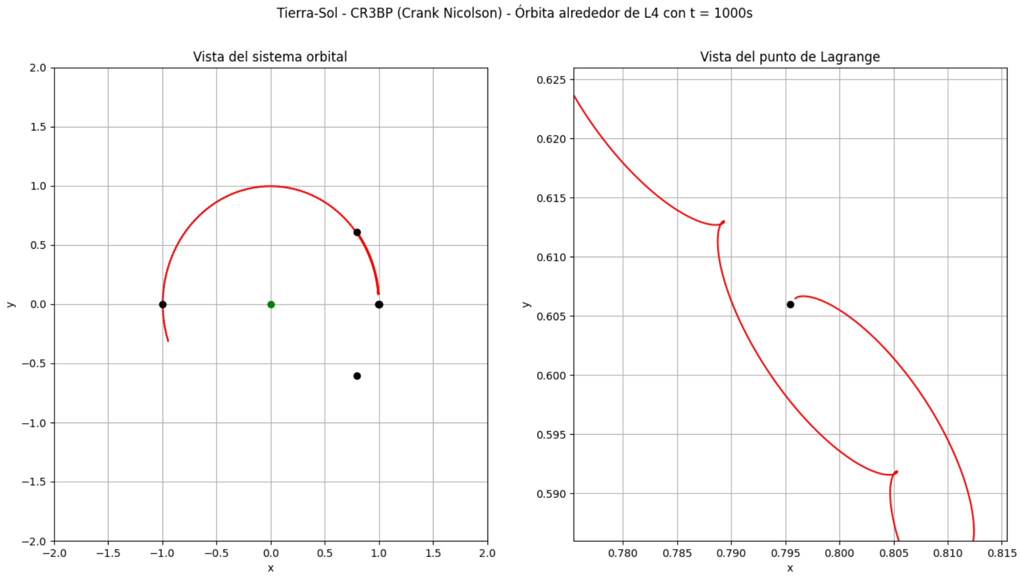

Figura 5.21: Órbita alrededor de L4 utilizando Crank Nicolson

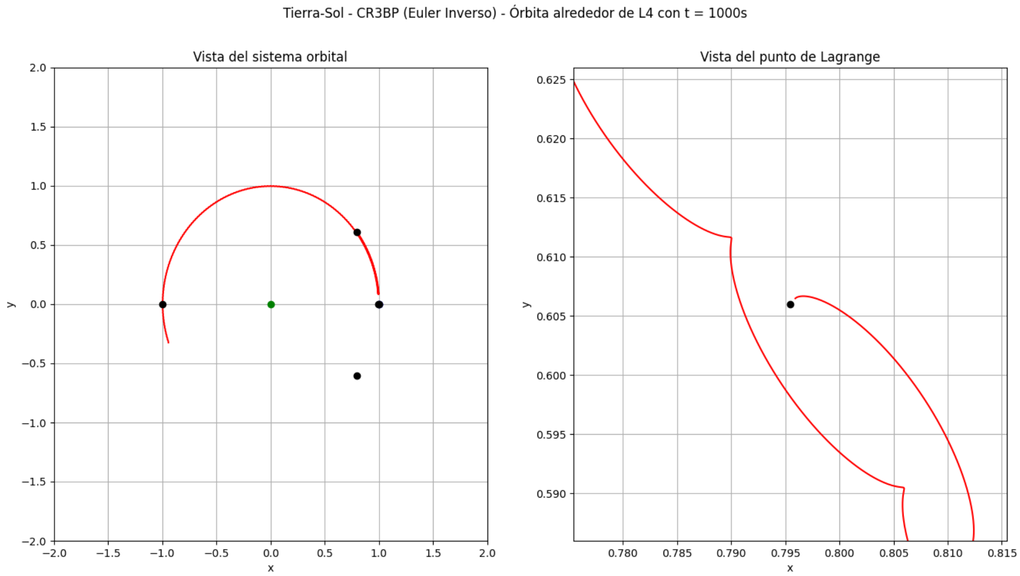

Figura 5.22: Órbita alrededor de L4 utilizando Euler inverso

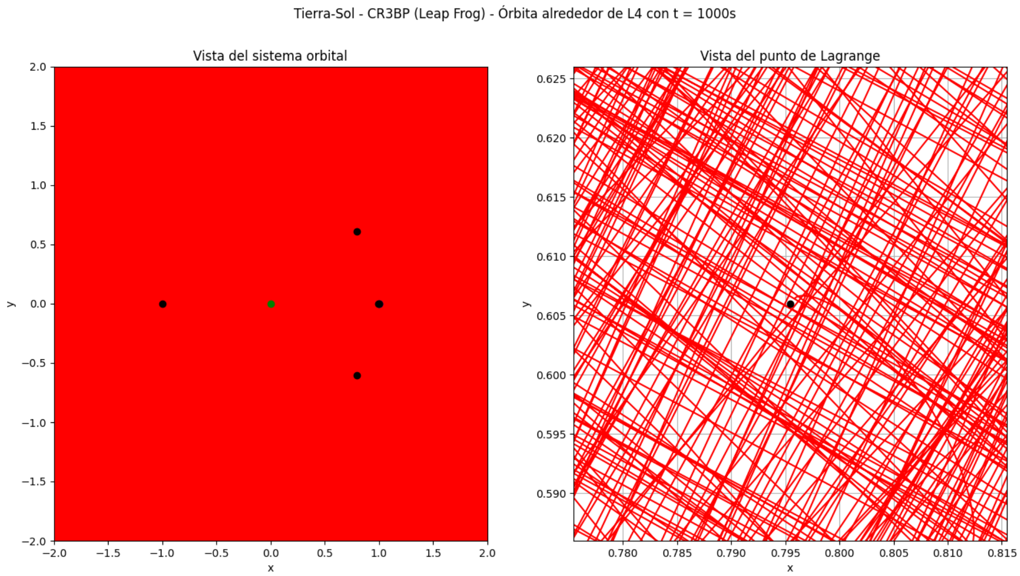

Figura 5.23: Órbita alrededor de L4 utilizando Leap Frog

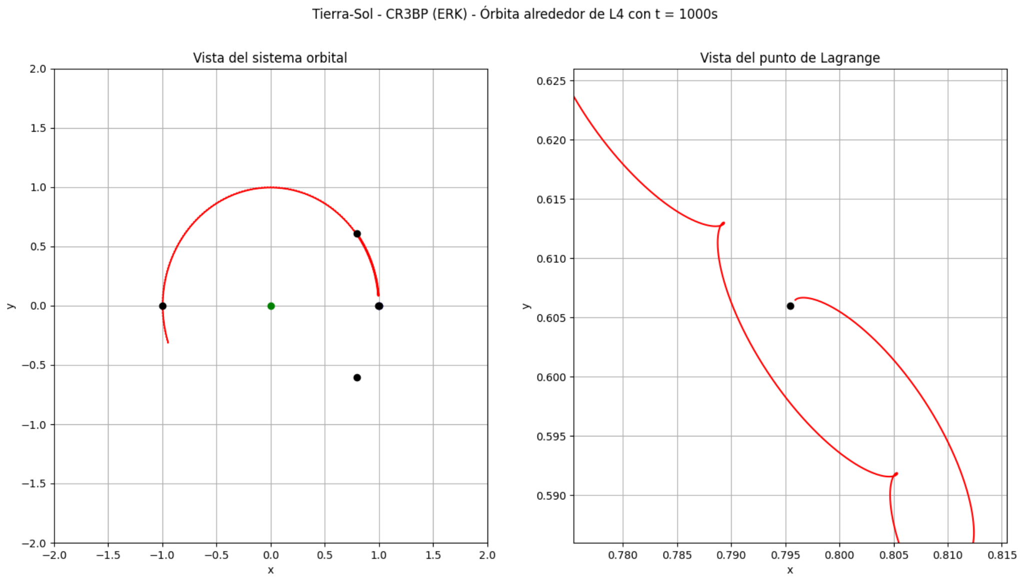

Figura 5.24: Órbita alrededor de L4 utilizando Runge-Kutta Embebido

5.5. Órbita alrededor de L5

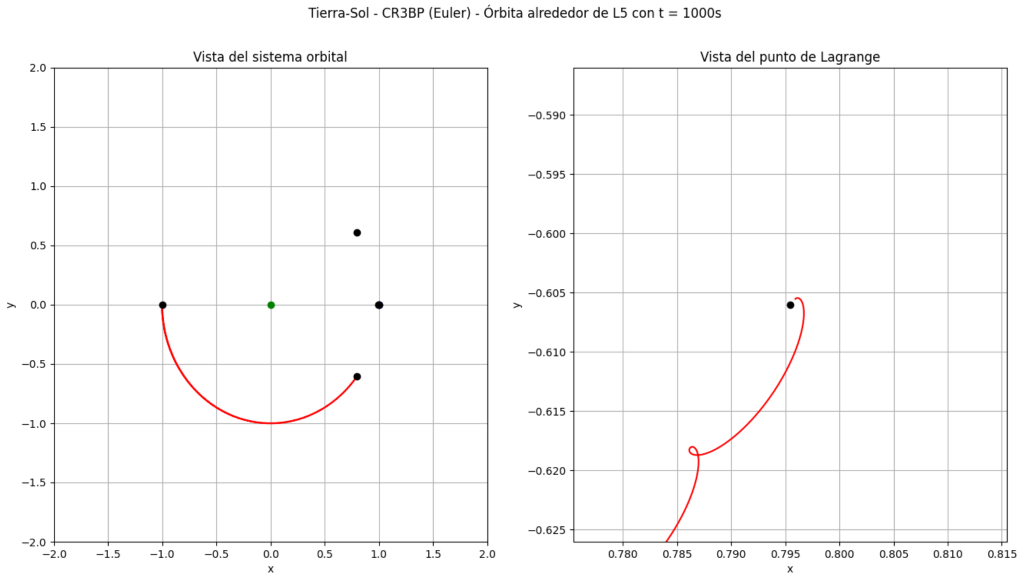

Figura 5.25: Órbita alrededor de L5 utilizando Euler

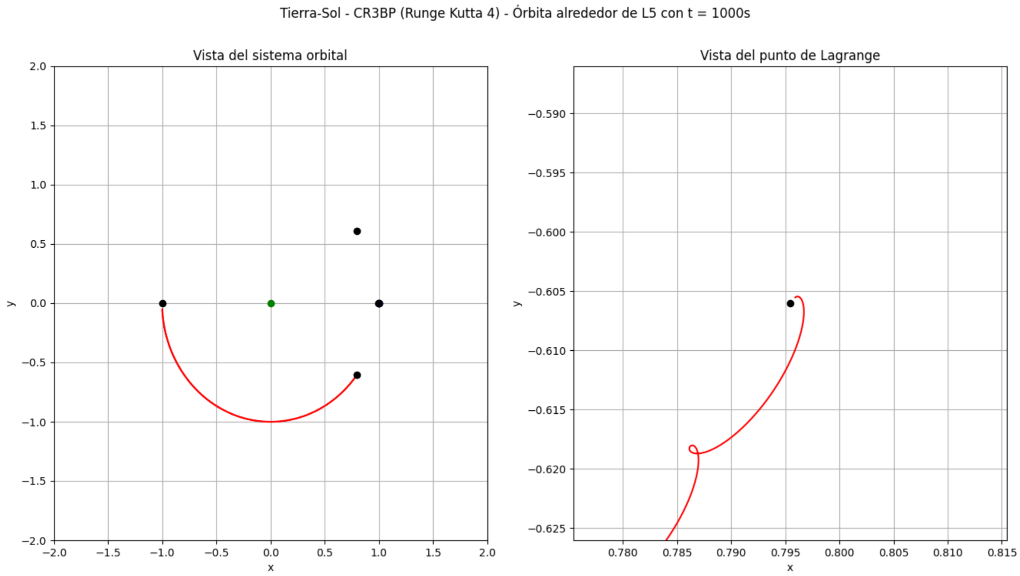

Figura 5.26: Órbita alrededor de L5 utilizando Runge-Kutta 4

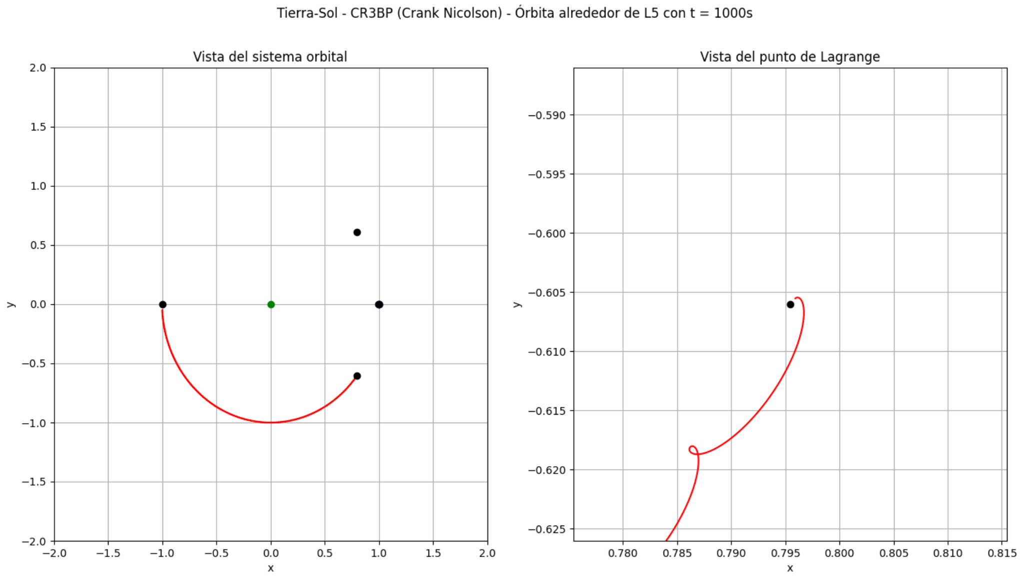

Figura 5.27: Órbita alrededor de L5 utilizando Crank Nicolson

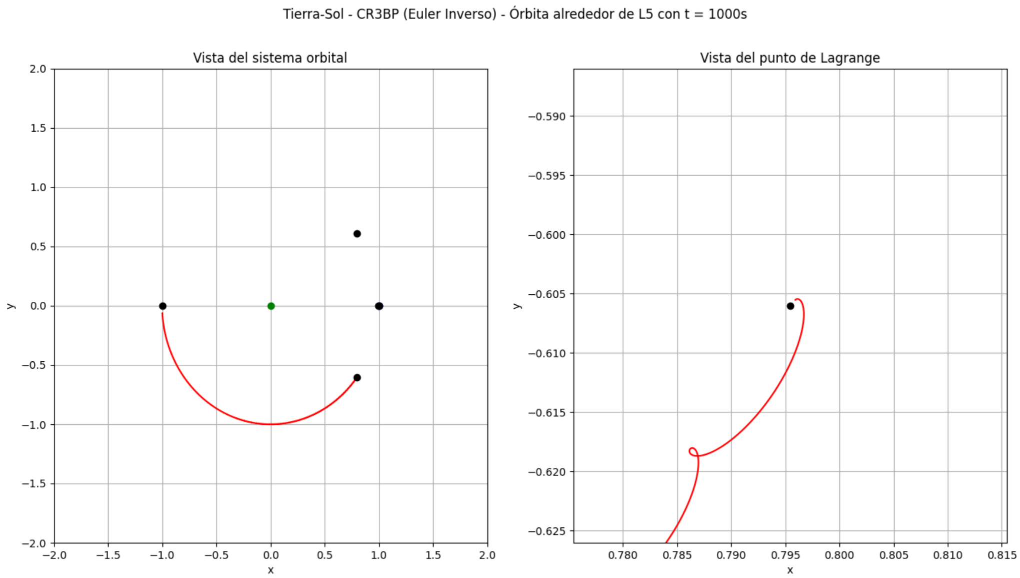

Figura 5.28: Órbita alrededor de L5 utilizando Euler inverso

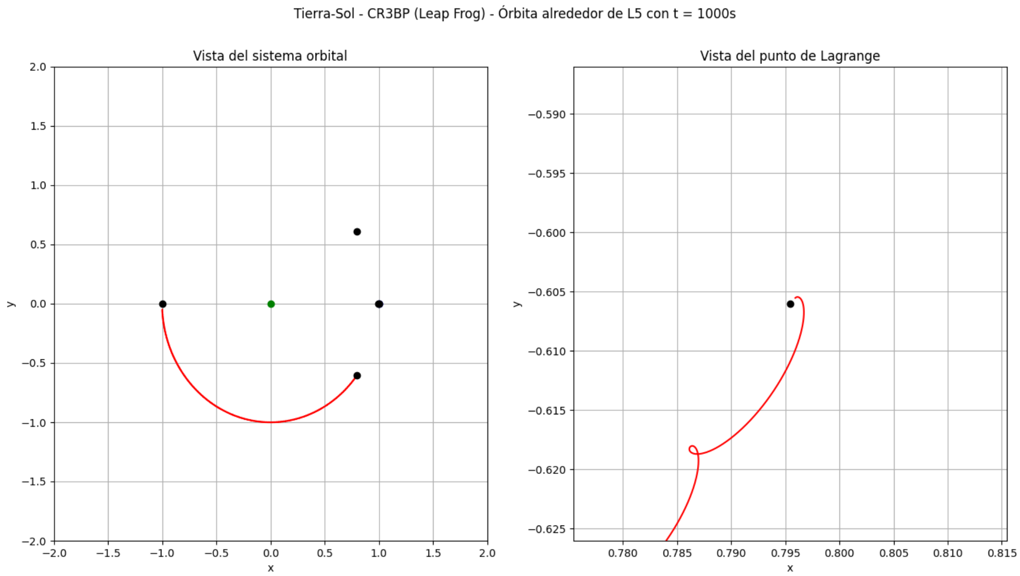

Figura 5.29: Órbita alrededor de L5 utilizando Leap Frog

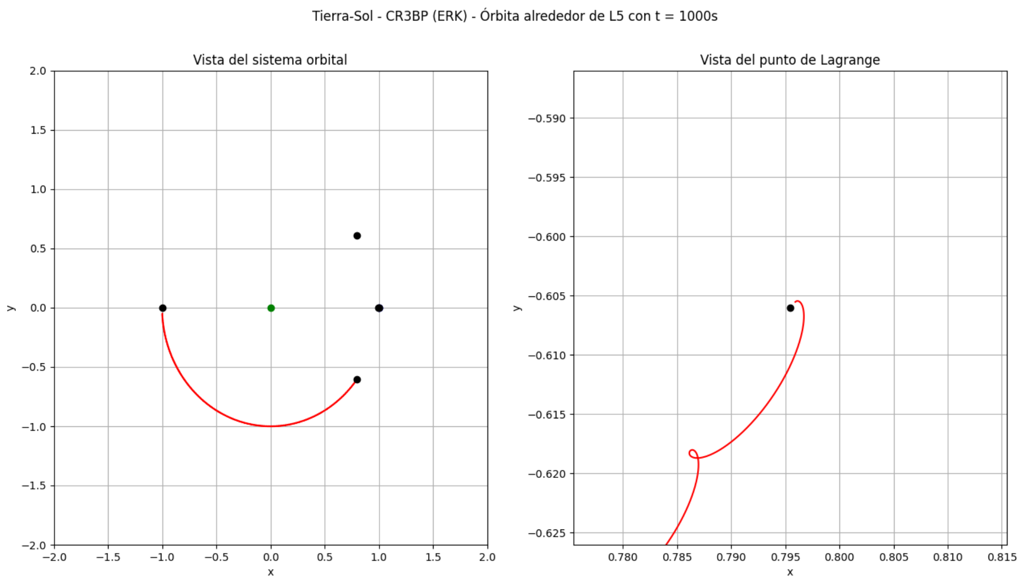

Figura 5.30: Órbita alrededor de L5 utilizando Runge-Kutta Embebido

6. Conclusión

Como se puede comprobar en la sección de exposición de los resultados [5] la integración de este problema depende enormemente de la perturbación a la que se someta a la partícula de masa infinitesimal, sobre todo teniendo en cuenta que esta no tendrá ningún tipo de mecanismo de corrección de órbita y podrá alejarse libremente del punto de equilibrio. Se ha de hacer especial mención a los resultados obtenidos por el esquema numérico de *Leap Frog* en algunos casos, los cuales son de lo más curiosos e inquietantes al mismo tiempo.

En cuanto al código desarrollado y expuesto en la sección [4] se puede comentar que las posibilidades de paralelización y aceleración del mismo son enormes, sobre todo teniendo en cuenta la estructura del archivo principal de bucles anidados en los que se simulan las órbitas alrededor de todos los puntos con todos los esquemas numéricos desarrollados hasta la fecha. Asimismo, las tareas de optimización del código quedan pendientes y se anotan en la libreta de cosas que hacer para cuando el alumno disponga del tiempo necesario para embarcarse en esta tarea con mayor compromiso y efectividad.

Por último y en cuanto al recorrido emprendido desde el primer Hito hasta este (el último que el alumno realiza de manera individual), se han de destacar el aprendizaje y los conocimientos adquiridos a lo largo de sus distintas etapas. El nivel de manejo y conocimiento en cuanto a programación en general y *Python* en particular ha mejorado considerablemente (sobre todo teniendo en cuenta el nivel del que se partía), lo cual es algo satisfactorio y cuya mención merece la pena. Por otro lado, el haber estudiado problemas físicos relacionados con el ámbito espacial y haber sido capaces de relacionar la física con la matemática que hay detrás (también de manera paralela en la parte de la asignatura impartida por Marta Cordero) ha hecho que el camino fuera mucho más ameno y entretenido.