



Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

## Milestones 4, 5 y 6

Ampliación de Matemáticas I

4 de diciembre de 2022

**Autor:**

Sergio de Ávila Cabral

# Índice

<b>1. Milestone 4</b>	<b>1</b>
1.1. Introducción . . . . .	1
1.2. Esquema de dependencia funcional . . . . .	1
1.3. Descripción de módulos nuevos o modificados . . . . .	2
1.4. Comentarios y mejoras . . . . .	7
<b>2. Milestone 5</b>	<b>13</b>
2.1. Introducción . . . . .	13
2.2. Esquema de dependencia funcional . . . . .	13
2.3. Scripts . . . . .	14
2.4. Resultados . . . . .	17
2.5. Comentarios y posibles mejoras . . . . .	20
<b>3. Milestone 6</b>	<b>22</b>
3.1. Introducción . . . . .	22
3.2. Esquema de dependencia funcional . . . . .	22
3.3. Scripts . . . . .	23
3.4. Resultados . . . . .	29
3.5. Comentarios y posibles mejoras . . . . .	36
<b>Referencias</b>	<b>37</b>

## Índice de figuras

1.1.	Relaciones de dependencia del Milestone 4 . . . . .	2
1.2.	Resultados Euler . . . . .	8
1.3.	Resultados Runge-Kutta orden 4 . . . . .	9
1.4.	Resultados Crank-Nicolson . . . . .	10
1.5.	Resultados Euler inverso . . . . .	11
1.6.	Resultados LeapFrog . . . . .	12
2.1.	Esquema de dependencia del <i>Milestone 5</i> . . . . .	13
2.2.	Órbitas estables . . . . .	18
2.3.	Órbitas inestables . . . . .	19
2.4.	Movimiento helicoidal de dos cuerpos . . . . .	20
3.1.	Esquema de dependencia funcional del Milestone 6 . . . . .	23
3.2.	Órbitas entorno a L1 . . . . .	30
3.2.	Órbitas entorno a L1 . . . . .	31
3.3.	Órbitas entorno a L2 . . . . .	31
3.3.	Órbitas entorno a L2 . . . . .	32
3.4.	Órbitas entorno a L3 . . . . .	33
3.4.	Órbitas entorno a L3 . . . . .	34
3.5.	Órbitas entorno a L4 . . . . .	34
3.5.	Órbitas entorno a L4 . . . . .	35
3.6.	Órbitas entorno a L5 . . . . .	35
3.6.	Órbitas entorno a L5 . . . . .	36

## Índice de tablas

3.1. Puntos de Lagrange Sistema Tierra-Luna . . . . .	29
---	----

## 1. Milestone 4

### 1.1. Introducción

Para la realización el *Milestone 4* se define como objetivo principal la integración del oscilador lineal:

$$\ddot{x} + x = 0 \quad (1.1.1)$$

Dicha integración será realizada mediante los esquemas numéricos ya empleados hasta ahora (Euler, Runge-Kutta de orden 4, Crank-Nicolson y Euler inverso) junto con un nuevo esquema, *LeapFrog*. El segundo objetivo es el cálculo de las regiones de estabilidad de dichos esquemas.

Para la realización de estos cálculos se han empleado los módulos ya existentes:

- Physics.py: Incluye la física del problema a resolver. Se encuentra en *Sources/Problems*.
- Temporal\_Schemes.py: Se incluyen los diferentes esquemas numéricos. Se localiza en *Sources/ODES*.
- Cauchy\_Problem.py: En el se encuentra la definición del problema de Cauchy, cuya localización coincide con la de Temporal\_Schemes.py.
- Jab\_Newt.py: En el se encuentra la integración numérica de Newton junto con el Jacobiano. Se localiza en el mismo path que las dos anteriores.
- Plots.py: se encarga de graficar las soluciones de los problemas de Cauchy. Se localiza en *Sources/Graf*.

Y se ha definido los siguientes nuevos módulos:

- Milestone\_4.py: Actúa como *main* y en él se encuentran importados todos los módulos. A su vez, se definen: las condiciones iniciales del problema, las iteraciones, diccionarios y las llamadas a las funciones que integran el problema objetivo y, por último, llama a las nuevas funciones que calculan la región de estabilidad de los diferentes esquemas numéricos.
- R\_S.py: En el se encuentran las dos nuevas funciones que permiten el segundo objetivo principal de este *Milestone*: la función *Stability\_Region*, que calcula la región de estabilidad; y la función *Stability\_Region*, que entrega a la función anterior el polinomio de estabilidad de cada esquema numérico.

### 1.2. Esquema de dependencia funcional

Como ya se ha mencionado en anteriores informes, se pretende cumplir con la filosofía de programación *Top-Down*. Esto requiere una buena jerarquización de los scripts empleados para la realización del *Milestone 4*.

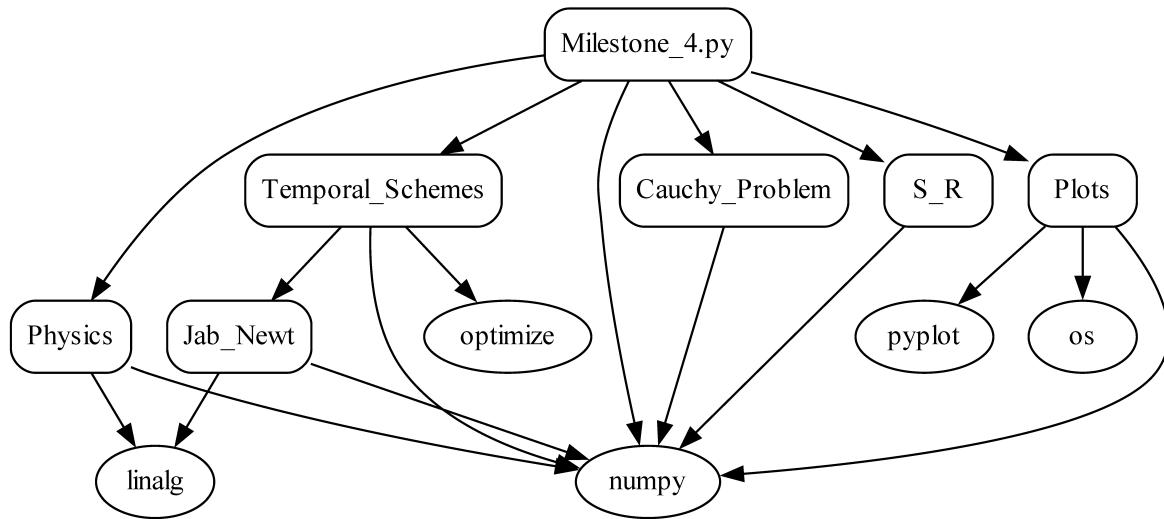


Figura 1.1: Relaciones de dependencia del Milestone 4

Como se puede apreciar en la Figura 1.1, todo el diagrama de dependencia fluye hacia abajo, lo que indica una buena estructuración según la filosofía *Top-Down*. Se puede apreciar que todos los módulos dependen de *numpy* (*linalg*) es un módulo interno de *numpy*.

### 1.3. Descripción de módulos nuevos o modificados

A continuación se hará una descripción de los nuevos módulos y de las modificaciones realizadas a ciertos módulos ya existentes.

#### Milestone\_4.py

Como se ha descrito en la Sección 1.1, actual como *main*. Este módulo sigue la misma estructura que los *main* de anteriores *Milestones*. En él se pueden apreciar diferentes secciones a lo largo del mismo:

En primer lugar, se exponen las librerías empleadas para el desarrollo de este módulo:

```

1 from numpy import array, save, zeros, linspace, shape
2 from ODES.Cauchy_Problem import C_P
3 from Problems.Physics import Linear_Oscilator
4 from ODES.Temporal_Schemes import Euler, RK4, Crank_Nicolson,
   Inverse_Euler, LeapFrog
5 from Stability_Region.S_R import Stability_Region
6 import Graf.PLOTS as plt

```

Listing 1.1: Librerías

Tras las librerías empleadas se encuentra la inicialización de las condiciones iniciales. En este caso, se definen como condiciones iniciales  $r_0 = 1$  y  $v_0 = 0$  para este oscilador libre.

```

1 #Initial Conditions
2 r0 = [1]
3 v0 = [0]
4 U0 = r0 + v0
5 T = 100
6 dt = [0.1, 0.01, 0.001]
```

Listing 1.2: Condiciones iniciales

A continuación, se mantiene el *boolean* que permite elegir si los *plots* se guardan o solo se muestran. Esta función está resultando muy útil puesto que, dada la correcta implementación de las funciones de los anteriores *Milestones*, con la comprobación del correcto funcionamiento de una esquema numérico en las nuevas funciones, se puede graficar todas las figuras si necesidad de controlar el proceso de los cálculos.

```

1 #Save Plots
2 Save = False # True Save the plots / False show the plots
3
```

Listing 1.3: selección guardado

Tras ello se definen las listas que contienen los diferentes esquemas numéricos, donde se puede observar el nuevo esquema numérico, *LeapFrog*.

```

1 #Temporal_Schemes to use
2 T_S = [Euler, RK4, Crank_Nicolson, Inverse_Euler, LeapFrog]
3 T_S_plot = ["Euler", "RK4", "CN", 'Euler_inver', "LeapFrog"]
```

Listing 1.4: Esquemas temporales

Se inicializan los diccionarios que se emplearán para el almacenamiento de los datos junto con la lista de los  $N$  que se emplearán. En este caso se inicializa también un diccionario para las regiones de estabilidad.

```

1 #Initiation of Dictionaries
2 t_dic = { }
3 U_dic = { }
4 S_R = { }
5 N = zeros(len(dt))
6
```

Listing 1.5: Inicialización de diccionarios

Se calculan las iteraciones necesarias junto con los tiempos que se emplearán en los cálculos de forma análoga al anterior *Milestone*.

```

1 #Iterations and times
2 for i in range(len(dt)):
```

```

3     N[i] = int(T/dt[i])
4     t_dic[str(i)] = linspace(0,T, int(N[i]))
5

```

Listing 1.6: definición tiempos e iteraciones

Por último, para la realización de los diferentes cálculos encomendados se emplea, al igual que en el *Milestone 2*, un doble bucle que recorre todos los quemas numéricos y todos os  $\Delta t$  deseados. Tras ello, se implementa un segundo bucle que calcula, mediante la función *Stability\_Region*, las diferentes regínes de estabilidad. Tras ello, se grafica mediante una nueva función en el módulo *Plots.py*.

```

1 # # #Simulations
2 for j in range ( len(T_S_plot) ):
3     for x in t_dic: # x is a str and U is creating new keys
4         U_dic[x]= C_P(Linear_Oscilator, t_dic[x], U0, T_S[j])
5         plt.Plot_CP(U_dic[x], t_dic[x], T, dt[int(x)], T_S_plot[j]
6 ], Save) # x return the value of the key
7     plt.Plot_CP_all(U_dic,t_dic,T, dt, T_S_plot[j], Save)
8     print(T_S_plot[j] + " calculado \n")
9
10
11 #Stability Region
12 for i in range( len(T_S_plot) ):
13     S_R[i] = Stability_Region(T_S_plot[i])
14     plt.Plot_SR(S_R[i], dt, T_S_plot[i], Save)

```

Listing 1.7: Integración numérica y región de estabilidad

## Physics.py

Se define el problema objetivo de este *Milestone* como un oscilador lineal. Este oscilador lineal tiene la siguiente forma:

$$\ddot{x} + x = 0 \quad (1.3.1)$$

Se define el problema de Cauchy para la integración numérica:

$$\frac{dU}{dt} = F(U, t) \quad (1.3.2)$$

Puesto que el vector posición se define como  $\vec{r}(x)$  y la aceleración como  $\dot{\vec{r}}(\dot{x})$ :

$$U = \begin{pmatrix} x \\ \dot{x} \end{pmatrix} \quad (1.3.3)$$

Se obtiene que  $F(U, t)$  como:

$$F(U, t) = \begin{pmatrix} \dot{x} \\ -x \end{pmatrix} \quad (1.3.4)$$

Por lo tanto, la programación de este problema resulta de la siguiente forma:

```

1 def Linear_Oscilator(U,t):
2
3     return array([U[1], -U[0]])

```

Listing 1.8: Definición problema Oscilador lineal

## R\_S.py

Como se ha descrito en la Sección 1.1, se implementa este módulo para el cálculo de las regiones de estabilidad. En un primer lugar, se definen los módulos que se van a emplear.

```

1 from numpy import sqrt, linspace, zeros, absolute, array

```

Listing 1.9: librerías para R\_S.py

## Función Stability\_Region

Se sigue de la función *Stability\_Region*, en la cual se definen dos vectores de  $N$  componentes, uno para números reales y otro para imaginario. Una vez definidos estas dos series de números, en una primera iteración se intento realizar la obtención de  $\omega$  mediante un simple bucle, lo cual presenta dos problemas: El primero de ellos que se impide el empleo de la función que se emplea para graficar dicha región, pues requiere un *array*  $2 - D$  y no  $1 - D$ . Derivado de este problema se deduce un segundo: dado que se realiza en un simple bucle, no se garantizan todas las combinaciones posibles para  $\omega$ .

Tras la obtención de  $\omega$ , se busca obtener todos los  $|r|$  que derivan de dichas  $\omega$  a través de los diferentes esquemas numéricos.

```

1 def Stability_Region(T_S):
2     N = 100
3     R = linspace(-5,5,N)
4     I = linspace(-5,5,N)
5     w = zeros([N, N], dtype = complex)
6     for i in range(N):
7         for j in range(N):
8             w[j,i] = complex(R[i], I[j])
9
10    if T_S.__name__ == "LeapFrog":
11        return absolute( T_S(1,1,1,0, lambda U, t: w*U) )
12    else:
13        return absolute( T_S(1,1,0, lambda U, t: w*U) )
14

```

Listing 1.10: Función Stability\_Region

## Plots.py

Para graficar las regiones de estabilidad es necesario la implementación de una nueva función que empleará una función de la librería *matplotlib* que permite graficar el contorno deseado de una serie de datos, llamado en esta función "*levels*".

Como se puede ver a continuación, se implementa la función *Plot\_SR*. Como comentario, se ve que se hace necesario separar el caso del *LeapFrog*, dado que el valor devuelto por la función *Stability\_Region* no es un *array*. El resto de la función es simple, se emplea la función *contour* y *contourf* para que grafique aquellos valores cuyo  $|r| = 1$  y resalte el área donde ello ocurre, *levels* = [0, 1]

```

1 def Plot_SR(r, dt, T_S, Save):
2     ax = plt.figure(figsize = (10, 10))
3     if T_S == 'LeapFrog':
4         Im = linspace(-1,1,100)
5         Re = zeros(100)
6
7         ax = plt.plot(Re, Im, color = '#0013ff')
8
9     else:
10        N = len(r)
11        x = linspace(-5,5,N)
12        y = linspace(-5,5,N)
13        ax = plt.contour(x,y, r, levels = [0, 1], colors = ['#0013ff'])
14        ax = plt.contourf(x,y, r, levels = [0, 1], colors =[#626262'])
15
16        colors = ['r','orange','g']
17
18    for i in range(len(dt)):
19        plt.plot([0,0], [dt[i],-dt[i]], 'o', color = colors[i], label = "Oscillator's Roots by dt " + str(dt[i]))
20    plt.ylabel("Im")
21    plt.xlabel("Re")
22    plt.legend()
23    plt.grid()
24
25    file = 'Stability Region of ' + T_S + ".png"
26    Save_plot(Save, file)

```

Listing 1.11: Función para graficar las regiones de estabilidad

subsectionResultados Por último, se exponen los resultados obtenidos mediante el código expuesto en este informe. Todos estos resultados han sido obtenidos para un  $T = 30$  s.

## Euler

En primer lugar se expone los resultados de la integración mediante el esquema numérico de Euler, Figura 1.2a. Se puede observar que para  $\Delta t = 0,1$  se produce una amplificación del valor de  $x$  con respecto del tiempo. Esto es debido al error que arrastra el método para ese  $\Delta t$ . Esto es comprobable, pues como se puede apreciar en la región de estabilidad, Figura 1.2b, cuanto menor es  $\Delta t$  más estable es la solución del método.

Del mismo modo, se puede comprobar que se produce la amplificación por el error del método, pues en el problema de Cauchy, al igual que ocurría con el problema de Kepler, mantiene la Energía constante. Como se aprecia, el valor de  $x$  aumenta progresivamente, no conservándose la energía.

### Runge-Kutta orden 4

Como se puede observar en la Figura 1.3a, a diferencia del resultado anterior, este error se encuentra acotado dada la estabilidad del método, observable en la Figura 1.3b. Se observa que se encuentra en la región de estabilidad, pero no indefinidamente.

### Crank-Nicolson

Al igual que en RK4, se obtiene resultados con un error muy acotado (Figura 1.4a). Pero a diferencia de este, como se puede observar en la 1.4b, la región de estabilidad es todo el semiplano negativo de los números reales. Esto le concede grandes propiedades de estabilidad. Las raíces de este oscilador se mantienen en todo momento sobre la frontera de la región de estabilidad.

### Euler Inverso

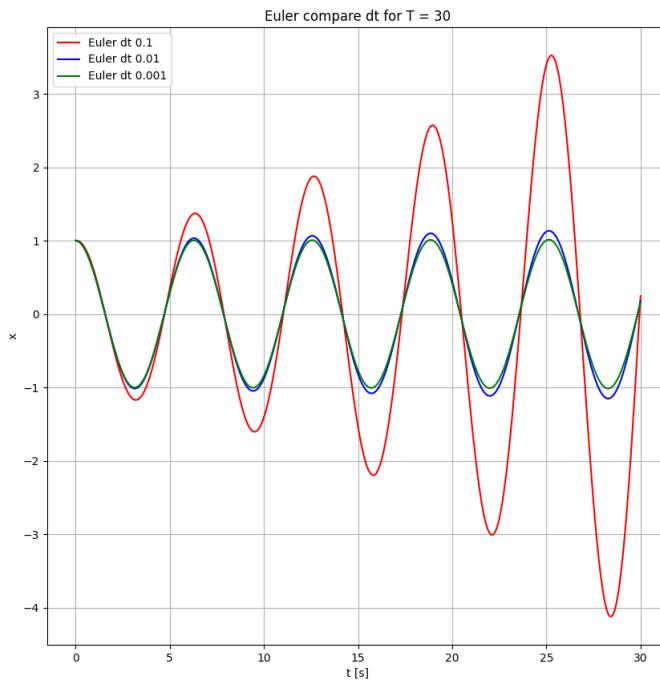
Al igual que en el caso de integración de Euler, la región de estabilidad para Euler Inverso es una circunferencia, pero en este caso en el semieje positivo de los números reales. Como se puede observar en la Figura 1.5a, a medida que se disminuye  $\Delta t$ , disminuye el error del método. Como se visualiza en la Figura 1.5b, cuanto menor  $\Delta t$ , más cerca de la frontera se encuentran las raíces del oscilador lineal, otorgando mayor precisión y estabilidad a los resultados.

### LeapFrog

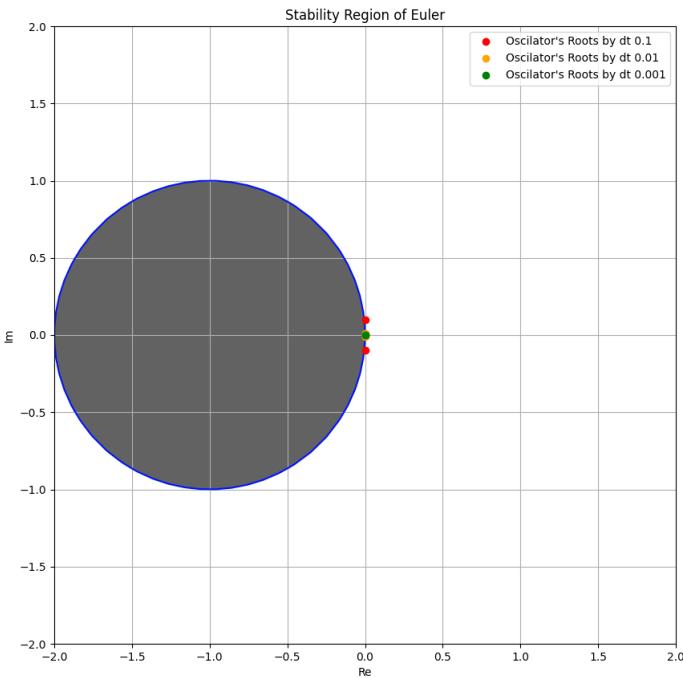
Por último, se encuentra el nuevo método implementado, *LeapFrog*. Como se aprecia en la Figura 1.6a, los resultados tienen una buena estabilidad a partir de  $\Delta t = 0,01$ . Como se puede apreciar en la Figura 1.6b, su región de estabilidad es una recta desde  $(0, -1)$  hasta  $(0, 1)$ . Dado este hecho, se puede deducir que este método no presenta tanta estabilidad como en RK4 o en CN.

## 1.4. Comentarios y mejoras

Como posible mejora a este hito, se podrían aplicar *decorator* que permitan plotear las figuras de una función en concreto. La implementación de un procesamiento en paralelo no se hace necesario, pues el tiempo computacional no es muy elevado para el esfuerzo que requeriría dicha programación.

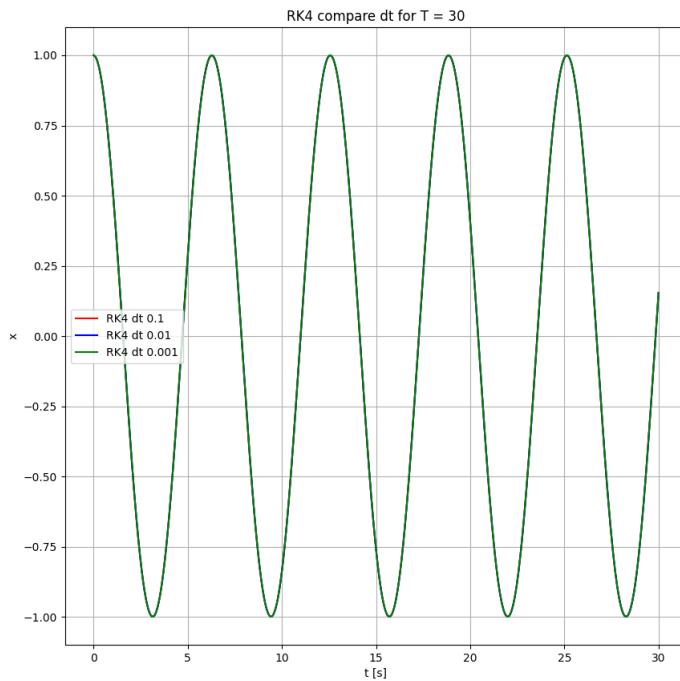


(a) Integración Euler para diferentes  $\Delta t$

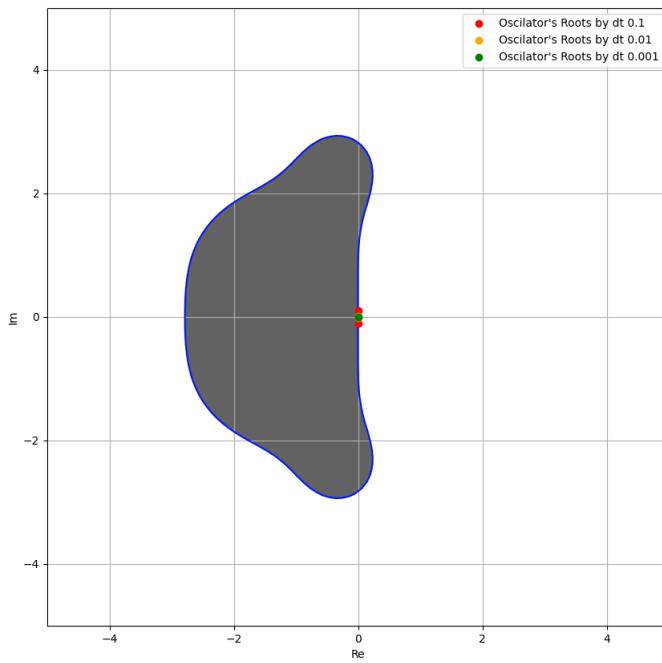


(b) Región de estabilidad de Euler

Figura 1.2: Resultados Euler

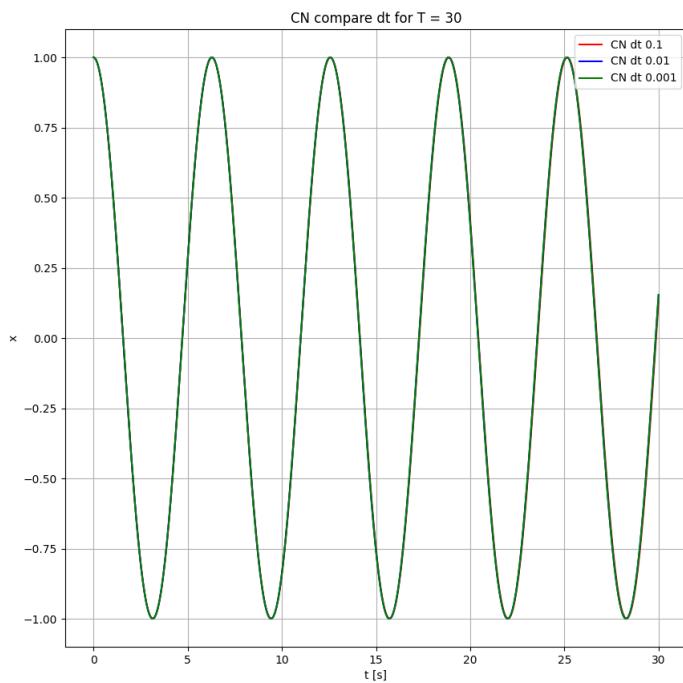


(a) Integración RK4 para diferentes  $\Delta t$

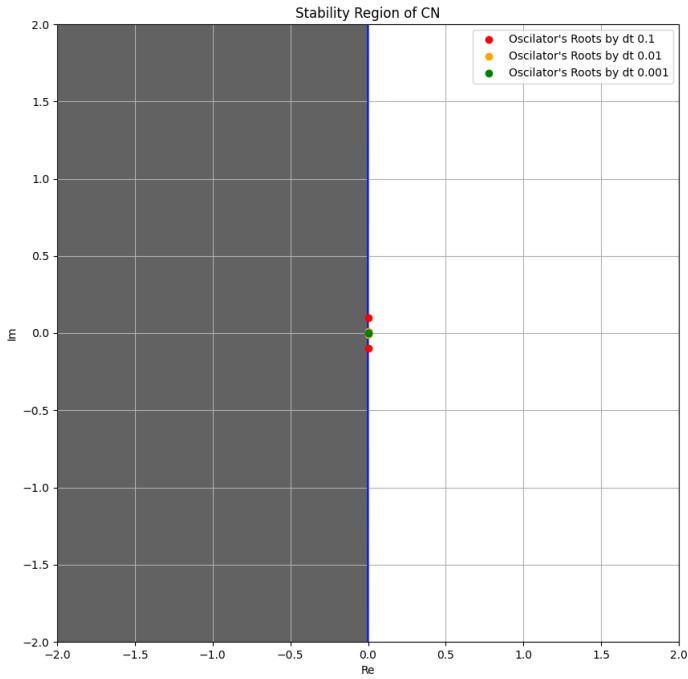


(b) Región de estabilidad de Rk4

Figura 1.3: Resultados Runge-Kutta orden 4

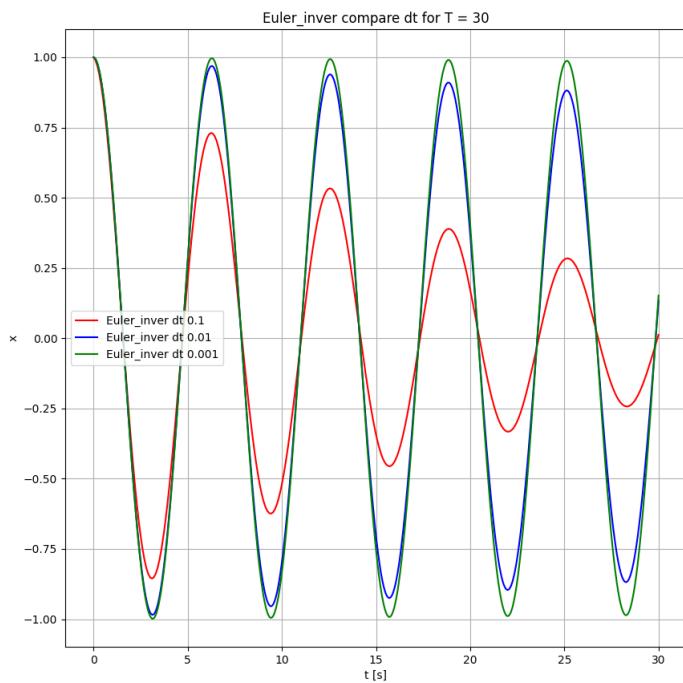


(a) Integración CN para diferentes  $\Delta t$

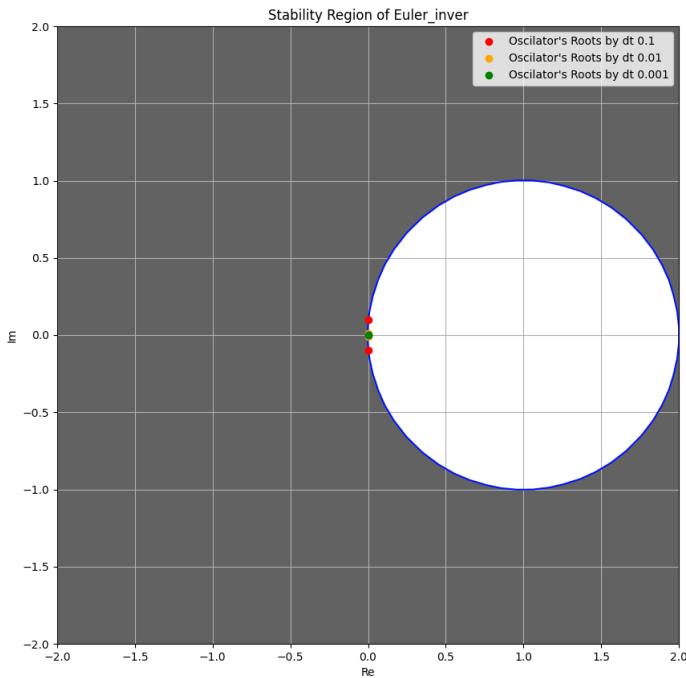


(b) Región de estabilidad de CN

Figura 1.4: Resultados Crank-Nicolson

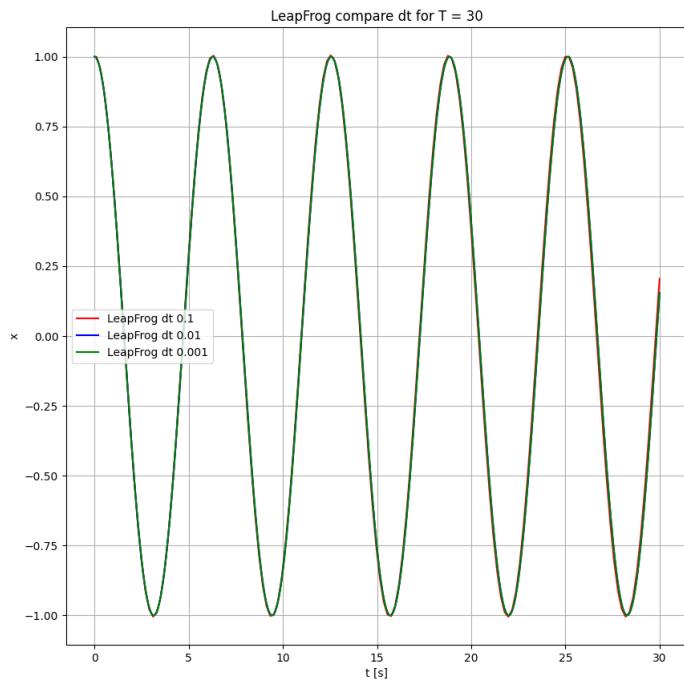


(a) Integración Euler inverso para diferentes  $\Delta t$

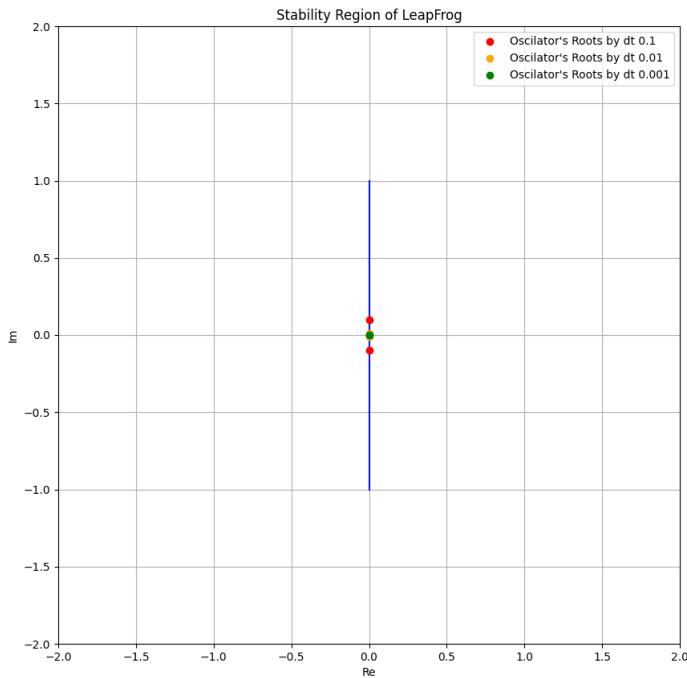


(b) Región de estabilidad de Euler inverso

Figura 1.5: Resultados Euler inverso



(a) Integración LeapFrog para diferentes  $\Delta t$



(b) Región de estabilidad de LeapFrog

Figura 1.6: Resultados LeapFrog

## 2. Milestone 5

### 2.1. Introducción

El problema de los N-cuerpos trata de determinar los movimientos individuales de diferentes cuerpos astronómicos, los cuales interactúan entre sí según las leyes de la gravitación universal. La resolución de este problema concluye en la siguiente expresión:

$$m_i \frac{d^2 r_i}{dt^2} = \sum_{\substack{j=1 \\ j \neq i}}^n \frac{G m_i m_j (r_j - r_i)}{\|r_j - r_i\|^3} \quad (2.1.1)$$

A lo largo de la Sección 2 se describirá la programación realizada para conseguir resolver el problema de los N-cuerpos computacionalmente, partiendo de la estructura del software y analizando individualmente cada módulo programado.

### 2.2. Esquema de dependencia funcional

En la Figura 2.1 se expone el diagrama de dependencia funcional del *Milestone 5*. En el se puede observar que todas las dependencias discurren unidireccionalmente, cumpliendo con la programación *Top-Down*.

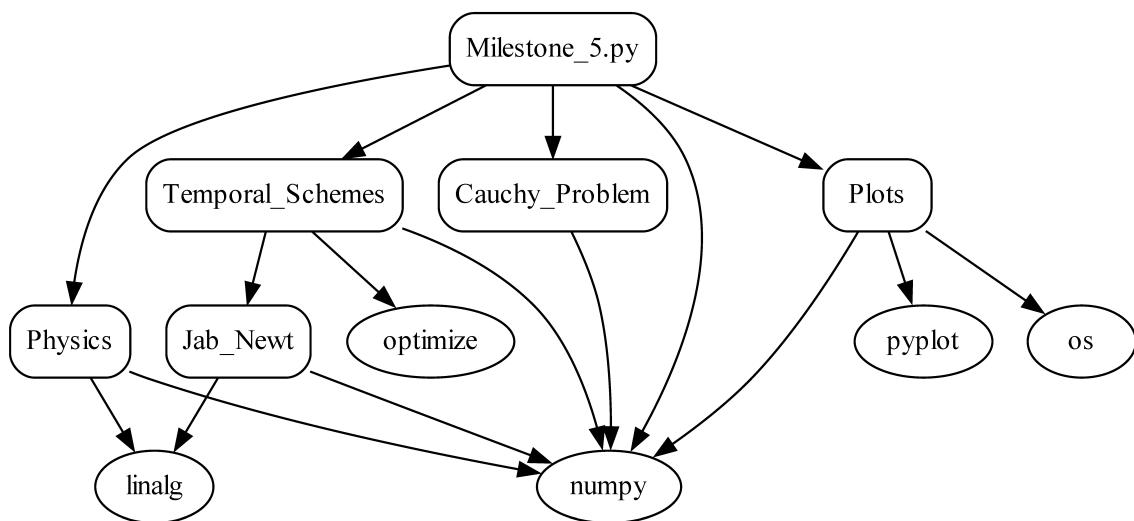


Figura 2.1: Esquema de dependencia del *Milestone 5*

## 2.3. Scripts

En primer lugar se encuentra el *main* de este software. Como se puede apreciar en el Listing 2.1, se definen las librerías y módulos introducidos. Cabe destacar que se siguen empleando los mismos módulos implementados desde el *Milestone 2*, habiéndose realizado pequeñas modificaciones a lo largo de los diferentes hitos. Tras ello, se inicializan las condiciones iniciales y se ensamblan en un vector  $U_0$ . Tras definir el número de iteraciones y el tiempo de simulación, se realiza la simulación. A diferencia de otros *Milestones* ya no se introduce un *string* con el nombre del esquema temporal, se ha definido el nombre de las respectivas funciones de los esquemas temporales en las propias funciones. Del mismo modo en la línea 33 del código, se puede ver cómo se introduce el nombre asignado al esquema temporal *Runge-Kutta orden 4*.

```

1 from numpy import array, save, zeros, linspace, shape, reshape, transpose
2 from ODES.Cauchy_Problem import C_P
3 from Problems.Physics import N_B, split, join
4 from ODES.Temporal_Schemes import Euler, RK4, Crank_Nicolson,
   Inverse_Euler, LeapFrog
5 import Graf.Plots as plt
6
7 #Initial Conditions
8 (Nb, Nc) = (4,3) # Number of bodies and coordinates
9
10 r0 = zeros( (Nb, Nc) ) # Position Matrix
11 v0 = zeros( (Nb, Nc) ) # Velocity Matrix
12
13 r0[0,:] = [1,0,1]; r0[1,:] = [-1,0,-1]; r0[2, :] = [ 0, 1, 2 ]; r0[3, :]
   = [ -2, -1, 0 ] # Position
14 v0[0,:] = [0, 0.4, 0]; v0[1,:] = [0, -0.4, 0]; v0[2, :] = [ -0.4, 0., 0.
   ]; v0[3, :] = [ 0.4, 0., 0. ] # Velocity
15
16 # Join the initial conditions in a vector
17 U0 = join(r0, v0, Nc, Nb)
18
19 #Save Plots
20 Save = True # True Save the plots / False show the plots
21
22 #Iterations and times
23 N = 10000
24 T = 10
25 t = linspace(0,T,N+1) # Temporal Mesh
26
27 # Simulations
28 U = transpose( C_P(N_B, t, U0, RK4) )
29
30 # Separate the solutions in tensors
31 r = split(U)
32
33 plt.Plot_NBodies(r, t, T, 15/1000, RK4.__name__, Save) # x plot de move
   of the N_bodies

```

Listing 2.1: Script principal Milestone 5

Dentro de *Physics* se han definido las tres funciones empleadas en el *main*. Tal y como se puede apreciar en el Listing 2.2, las funciones *join* y *split* introducen realizan un *reshape*, tanto para ensamblar como desempaquetar la información de los diferentes *arrays*. Estas funciones se realizaron para conseguir mayor limpieza en el *main*.

```

1 from numpy import array, reshape, zeros, shape
2 from numpy.linalg import norm
3
4 #-----
5 # Kepler -----
6 #
7
8 def Kepler(U, t):
9
10    x = U[0]; y = U[1]; x_dot = U[2]; y_dot = U[3]
11    d = (x**2 + y**2)**1.5
12
13    return array( [ x_dot, y_dot, -x/d, -y/d ] )
14
15
16 #-----
17 # Linear Oscilator -----
18 #
19
20 def Linear_Oscilator(U,t):
21
22    return array([U[1], -U[0]])
23
24 #-----
25 # N_Bodies -----
26 #
27
28 def split(U):
29    Nc = 3
30    Nb = int(shape(U)[1] / (2*Nc))
31    N = shape(U)[0] - 1
32
33    Us = reshape( U, (N+1, Nb, Nc, 2) )
34
35    return reshape( Us[:, :, :, 0], (N+1, Nb, Nc) )
36
37 def join(r0, v0, Nc, Nb):
38    U0 = zeros(2*Nc*Nb)
39    U1 = reshape( U0, (Nb, Nc, 2) )
40    U1[:, :, 0] = r0
41    U1[:, :, 1] = v0
42    return U0
43
44 def N_B(U,t): # N-Bodies problem
45
46    Nc = 3
47    Nb = int(len(U) / (2*Nc))
48
49    Us = reshape( U, (Nb, Nc, 2) ) # Tensor Nb, Nc, (Position, Velocity)
50
51    r = reshape( Us[:, :, 0], (Nb, Nc) ) # Matrix Position

```

```

52     v = reshape( Us[:, :, 1], (Nb, Nc) ) # Matrix Velocity
53
54     F = zeros( len(U) )
55     Fs = reshape( F, (Nb, Nc, 2) ) # Tensor Nb, Nc, (Position, Velocity)
56
57     drdt = reshape( Fs[:, :, 0], (Nb, Nc) ) # Matrix Velocity
58     dvdt = reshape( Fs[:, :, 1], (Nb, Nc) ) # Matrix Acceleration
59
60     dvdt[:, :] = 0
61
62     for i in range(Nb):
63         drdt[i, :] = v[i, :]
64         for j in range(Nb):
65             if j != i:
66                 d = r[j, :] - r[i, :]
67                 dvdt[i, :] += d[:] / (norm(d)**3)
68
69     return F

```

Listing 2.2: Definición del problema de los N cuerpos

Por último, se encuentra en el módulo *Plots* la función *Plot\_NBodies*, la cual permite el graficado de las diferentes figuras que se podrán observar. Como se puede ver en el Listing 1.3, dicha función recoge y plotea las diferentes órbitas. Del mismo modo que los anteriores hitos, se apoya en la función *Save\_plot*, la cual permite elegir entre guardar o visualizar las diferentes figuras. Como ya se ha mencionado, debido a la filosofía de realizar una programación para que puedan realizarse todos los cálculos seguidos, esta función resulta muy útil para no tener que estar comprobando el proceso de cálculo del ordenador.

```

1 def Plot_NBodies(r, t, T, dt, T_S, Save):
2     fig, ax = plt.subplots( figsize = (10, 10) )
3     ax = plt.axes(projection = '3d')
4     N = shape(r)[1]
5     for i in range(N):
6         ax.plot(r[:, i, 0], r[:, i, 1], r[:, i, 2], label = "Orbit " + str(i+1))
7     ax.set_xlabel('x')
8     ax.set_ylabel('y')
9     ax.set_zlabel('z')
10    ax.set_title(T_S + "; T = " + str(T) + "; dt = " + str(dt))
11    ax.legend()
12    ax.grid()
13
14    file = T_S + "_dt_" + str(dt) + ".png"
15
16    Save_plot(Save, file)
17    def Save_plot(Save, file):
18
19        if not Save:
20            plt.show()
21
22    else:
23        ##Save Plots
24
25        path = os.path.join(os.getcwd(), 'Plots')
26

```

```

27     if not os.path.exists(path):
28         os.makedirs(path)
29
30     plt.savefig(os.path.join(path, file))

```

Listing 2.3: Plot de las figuras

Como se puede observar, todas las funciones descritas son de primera clase, pues en algunas de ellas algunos argumentos son otras funciones o devuelven funciones; y puras puesto que no son globales y no modifican argumentos de entrada.

## 2.4. Resultados

Se han realizado dos demostraciones del problema de los N cuerpos. la primera de ella se empleó para comprobar el correcto funcionamiento del código desarrollado.

Como se puede ver en las Figura 2.2, se obtiene órbitas estables alrededor del punto (0, 0, 0) para las siguientes condiciones iniciales:

$$\begin{aligned}
 \vec{r}_1 &= 1, 0, 0; \quad \vec{v}_1 = 0, 0, 4, 0 \\
 \vec{r}_2 &= -1, 0, 0; \quad \vec{v}_2 = 0, -0, 4, 0 \\
 \vec{r}_3 &= 0, 1, 0; \quad \vec{v}_3 = -0, 4, 0, 0 \\
 \vec{r}_4 &= 0, -1, 0; \quad \vec{v}_4 = 0, 4, 0, 0
 \end{aligned}$$

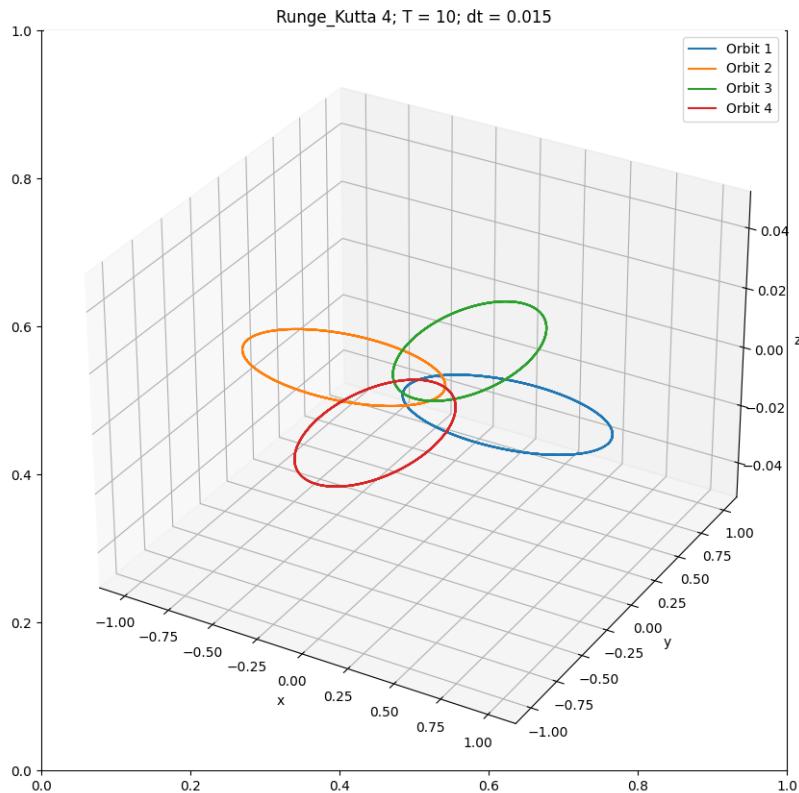
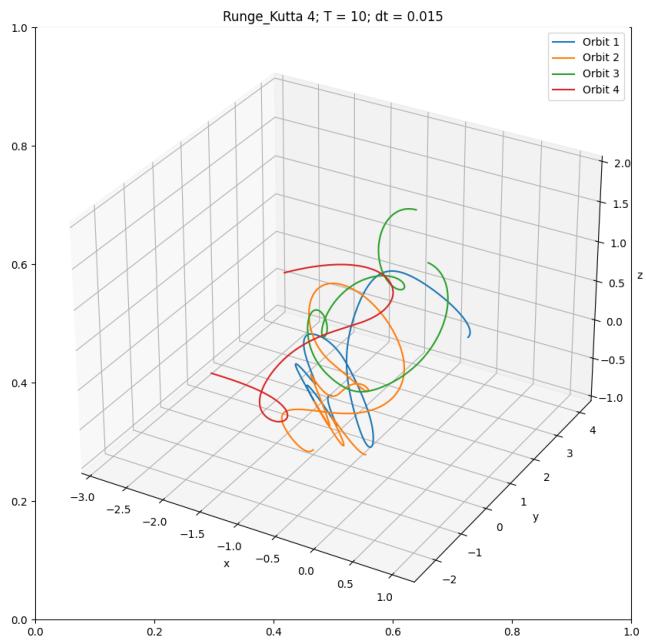


Figura 2.2: Órbitas estables

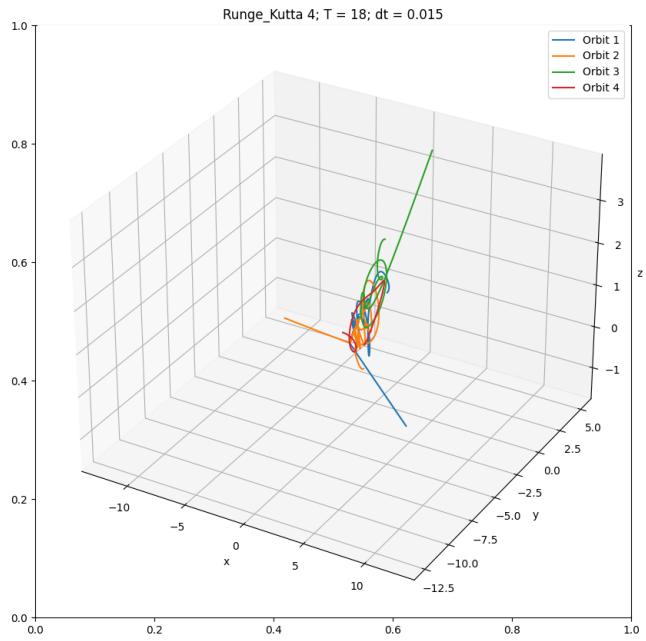
Comparada la buena programación del código se introducen otros cuatro cuerpos con valores iniciales diferentes.

$$\begin{aligned}\vec{r}_1 &= 1, 0, 1; \vec{v}_1 = 0, 0, 4, 0 \\ \vec{r}_2 &= -1, 0, -1; \vec{v}_2 = 0, -0, 4, 0 \\ \vec{r}_3 &= 0, 1, 2; \vec{v}_3 = -0, 4, 0, 0 \\ \vec{r}_4 &= -2, -1, 0; \vec{v}_4 = 0, 4, 0, 0\end{aligned}$$

En la Figura 2.3 se puede observar las órbitas de cuatro cuerpos. Dichas órbitas son inestables, pues los movimientos de los mismos son erráticos. Si se simula en un tiempo cercano a  $T = 18$  s, como se muestra en la Figura 2.3b, las órbitas se proyectan de forma lineal, divirgiendo el resultado.



(a) Simulación de  $T = 10$  s



(b) Simulación de  $T = 18$  s

Figura 2.3: Órbitas inestables

Por último, se simula el movimiento de dos cuerpos con las siguientes condiciones iniciales:

$$\vec{r}_1 = -0,5, -0,5, 0; \vec{v}_1 = 0, 0,5, 0,5$$

$$\vec{r}_2 = 0, 0,5, 0,5; \vec{v}_2 = 0, -0,5, 0,5$$

Dadas las condiciones iniciales, es de esperar un movimiento helicoidal hacia el arriba. Tal y como se puede visualizar en la Figura 2.4, dicha suposición es correcta. Los dos cuerpos realizan un movimiento estable alrededor de un punto intermedio a ellos.

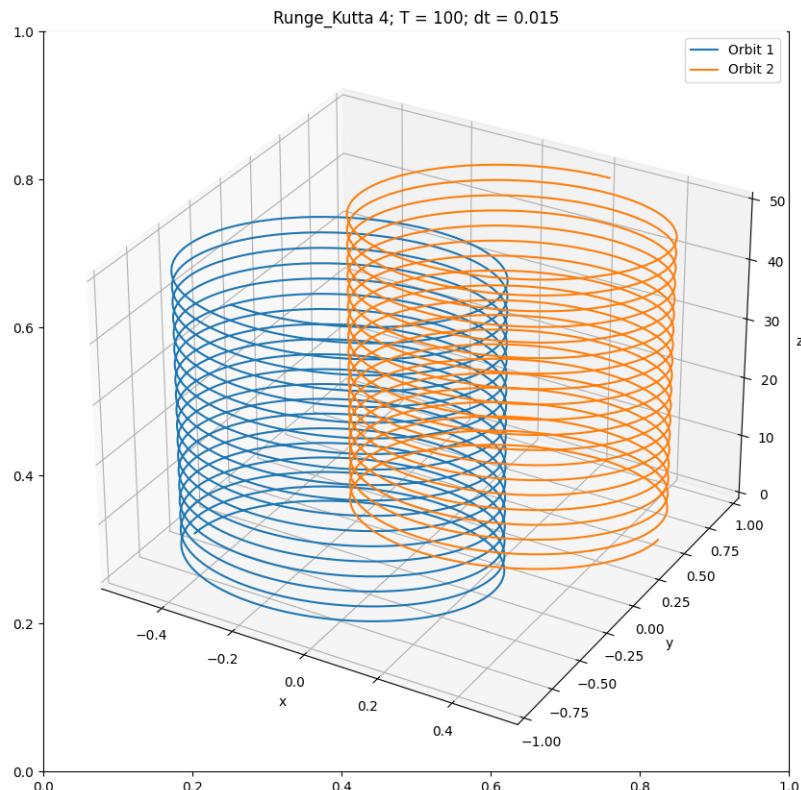


Figura 2.4: Movimiento helicoidal de dos cuerpos

## 2.5. Comentarios y posibles mejoras

Resulta muy complicado introducir valores que conlleven a una correcta integración del problema, teniendo que variar el tiempo de integración o las condiciones iniciales para conseguir unos resultados correctos, pues en el caso de la Figura 2.2, a pesar de ser órbitas estables, si la integración supera a 10 segundos, los resultados divergen.

A pesar del correcto funcionamiento de este software y de algunas mejoras implementadas, se pueden introducir mejoras, como puede ser un *decorator* que, a través de la programación de una función que contenta la simulación, se realice la graficación de las diferentes figuras.

## 3. Milestone 6

### 3.1. Introducción

El Objetivo principal de este hito es la programación de un Runge-Kutta embedido, el cual se define mediante de la siguiente forma:

$$U^{n+1} = u^n + \Delta t_n \sum_{i=1}^s b_i k_i \quad (3.1.1)$$

$$k_i = F \left( U^n + \Delta t_n \sum_{j=1}^s a_{ij} j_j, t_n + a \Delta t_n \right)$$

A su vez, se define el problema de los tres cuerpos restringido circular descrito por las ecuaciones de Arenstorf, las cuales determinan el problema para el sistema tierra luna. Este problema se define mediante las siguientes expresiones:

$$\dot{x} = v_x \quad (3.1.2)$$

$$\dot{y} = v_y \quad (3.1.3)$$

$$\dot{v}_x = x + 2v_y - \frac{(1-\mu)(x+\mu)}{\sqrt{[(x+\mu)^2 + y^2]^3}} - \frac{\mu(x+\mu-1)}{\sqrt{[(x+\mu-1)^2 + y^2]^3}} \quad (3.1.4)$$

$$\dot{v}_y = y - 2v_x - \frac{(1-\mu)y}{\sqrt{[(x+\mu)^2 + y^2]^3}} - \frac{\mu y}{\sqrt{[(x+\mu-1)^2 + y^2]^3}} \quad (3.1.5)$$

Del mismo modo, se deben de determinar lo puntos de Langrange y calcular su estabilidad. Por último, se comprarán los resultados obtenido al calcular órbitas alrededor de los puntos de Lagrange mediante diferentes esquemas numéricos.

### 3.2. Esquema de dependencia funcional

Al igual que los anteriores hitos, la programación realizada para este hito cumple la filosofía de programación *Top-Down*, pues como se puede ver en la Figura 3.1, todo el flujo de dependencia discurre en diferentes niveles y hacia abajo. Del mismo modo, se puede ver que entre os módulos principales (*Cauchy\_Problem*, *Temporal\_Schemes*, *NB3\_restrict*, *ERK* y *Plots*), siendo estos llamados desde el *main*. A su vez, estos módulos principales si que requieren de mismas módulos secundarios, como puede ser del módulo *optimize* de *Scipy*, pues es requerido para todos los esquemas temporales y para el problema restringido. Por último, se puede apreciar como todos los módulos son dependientes de la librería *Numpy* (*Linalg* es parte de esta librería).

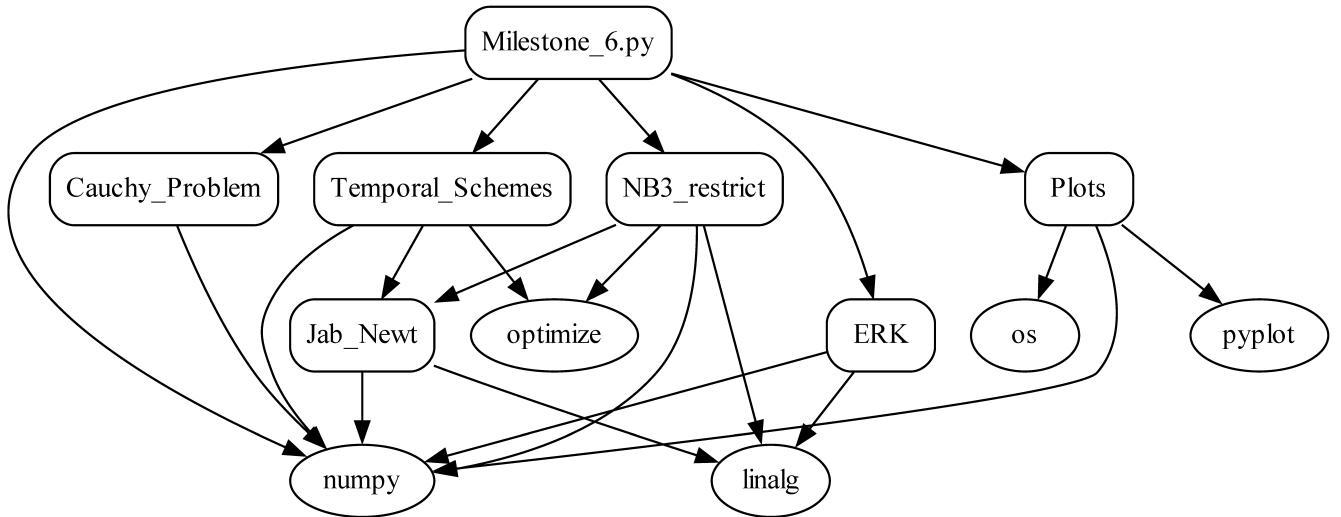


Figura 3.1: Esquema de dependencia funcional del Milestone 6

### 3.3. Scripts

Este módulo es el *main* del software desarrollado para este hito. En el Listing 3.1 se puede apreciar en primer lugar la importación de los diferentes módulos expuestos en la Figura 3.1. Tras la importación se encuentra el ya recurrente parámetro para guardar las figuras ploteadas y tras este la lista de esquemas numéricos. Como ya se ha mencionado, la programación de todos estos hitos está orientada a realizar scripts que, tras realizar una prueba para verificar su correcto desempeño, se introduzcan los diferentes parámetros y se realicen todos los cálculos sucesivamente, pues en casos de cálculos con esquemas numéricos como Crank-Nicolson o Euler inverso, el tiempo de computación es bastante elevado.

Posteriormente se introduce la distancia tierra-luna, se inicializan los parámetros que definen la malla de integración y las condiciones iniciales de los puntos de Lagrange. Es de extrema importancia introducir unas condiciones iniciales correctas para un buen cálculo de los puntos de Lagrange. Se calculan dichos puntos y se verifica su estabilidad. Tras ello, se calculan las diferentes órbitas a través del problema de Cauchy y se plotean. Para poder obtener órbitas desde esos puntos se requiere una pequeña perturbación que posibilite el comienzo de las órbitas, pues aunque sean algunos puntos inestables, no dejan de ser posiciones de equilibrio, por lo que si no se perturba no se podrá obtener ninguna órbita.

Cabe destacar que, aunque las condiciones iniciales se estén introduciendo en filas a diferencia de otros hitos, la función `C_P` ya redimensiona oportunamente dicho vector para introducirlo de forma apropiada en el cálculo. Se devuelven los resultados introducidos en columnas, dato a tener en cuenta para un procesamiento adecuado de los datos. De igual modo, se pueden observar una función wrappeadas. Tal y como se comentó en clase, esto se

debe a la necesidad de introducir en el problema de Chauchy una función con dos variables. Por ello, se transforma el problema restringido en un problema de dos variables, U y t.

Por último, destacar que el esquema temporal de Runge-Kutta embebido se ha decidido programarlo en un módulo diferente al resto de parámetros esquemas temporales por requerir de diferentes funciones bastante extensas.

```

1 from numpy import array, save, zeros, linspace, shape, reshape, around
2 from ODES.Cauchy_Problem import C_P
3 from Problems.NB3_restrict import RBP, Lagrange_points, stb_Lp
4 from ODES.Temporal_Schemes import Euler, RK4, Crank_Nicolson,
   Inverse_Euler, LeapFrog
5 from ODES.ERK import ERK_sheme
6 import Graf.Plots as plt
7
8 # Save plot
9 Save = False
10
11 # Temporal_Scheme
12
13 T_S = [Euler, RK4, LeapFrog, ERK_sheme, Crank_Nicolson, Inverse_Euler,
   ERK_sheme]
14
15 # Data of the system
16 mu_earth_Moon = 1.2151e-2
17
18 # Initialitation of the integration mesh
19 N = 10000
20 t = linspace(0, 100, N)
21
22 # Initial Conditions for interpolation
23 U0 = zeros( [5,4] ) # Number of Lagrange points and coordiantes
24
25 U0[0,:] = [0.8, 0.6, 0 , 0]
26 U0[1,:] = [0.8, -0.6, 0, 0]
27 U0[2,:] = [-0.1, 0, 0, 0]
28 U0[3,:] = [0.1, 0, 0, 0]
29 U0[4,:] = [1.01, 0, 0, 0]
30
31 L_p = Lagrange_Points(U0, shape(U0)[0], mu_earth_Moon)
32 print("\n" + str(L_p) + "\n")
33
34 #Stability of Lagrande Points
35 L_p_stb = zeros( 4 )
36
37 for i in range( shape(U0)[0] ):
38     L_p_stb[:2] = L_p[i, :]
39     stb = around( stb_Lp(L_p_stb, mu_earth_Moon), 5 )
40     print(str(stb) + "\n")
41
42 #Lagrange Point's Orbits
43
44 U_DLPO = zeros( [shape(U0)[0], 4] )
45
46 eps = 1e-4 # perturbation to initiate movement
47

```

```

48 for i in range( shape(U0)[0] ):
49     U_OLPO[i, :2] = L_p[i, :] + eps
50     U_OLPO[i, 2:] = eps
51
52 def F(U,t):
53     return RBP(U, t, mu_earth_Moon)
54
55 for j in range( len(T_S) ):
56     for i in range( shape(U0)[0] ):
57         U_LP = C_P(F, t, U_OLPO[i,:], T_S[j]) # introduce lines and
58     return in rows
      plt.Plot_LPO(U_LP, L_p, mu_earth_Moon, T_S[j], i, Save)

```

Listing 3.1: Script principal Milestone 6

Continuando con el esquema temporal de Runge-Kutta embebido (Listing 3.2), se sigue la programación que se puede obtener en [2]. Como modificación de este código, solo se ha dispuesto de una matriz de Butcher, por lo que en dicha función no se introduce ningún parámetro.

```

1 from numpy.linalg import norm
2 from numpy import zeros, matmul, linspace
3
4 def ERK_sheme(U1 , dt, t, F):
5     ERK_sheme.__name__ = "ERK"
6
7     tolerance = 1e-10
8
9     V1 = RK_scheme("First", U1, t, dt, F)
10    V2 = RK_scheme("Second", U1, t, dt, F)
11
12    (a, b, bs, c, q, Ne) = Butcher_array()
13
14    h = min( dt, Step_size(V1 - V2, tolerance, min(q), dt) )
15
16    N = int( dt / h ) + 1
17    h = dt / N
18
19    V1 = U1; V2 = U1
20
21    for i in range(N):
22        time = t + i * dt / N
23        V1 = V2
24        V2 = RK_scheme("First", V1, time, h, F)
25
26    U2 = V2
27
28    ierr = 0
29
30    return U2
31
32 def RK_scheme(tag, U1, t, dt, F):
33
34     (a, b, bs, c, q, Ne) = Butcher_array()
35
36     N = len(U1)
37     k = zeros( [Ne, N] )

```

```

38
39     k[0,:] = F( U1, t + c[0]*dt )
40
41
42     if tag == "First":
43
44         for i in range(1,Ne):
45             Up = U1
46
47             for j in range(i):
48                 Up = Up + dt * a[i,j]*k[j,:]
49
50             k[i,:] = F( Up, t + c[i]*dt )
51
52         U2 = U1 + dt * matmul(b,k)
53
54     elif tag == "Second":
55
56         for i in range(1,Ne):
57             Up = U1
58
59             for j in range(i):
60                 Up = Up + dt * a[i,j]*k[j,:]
61
62             k[i,:] = F(Up, t + c[i] * dt)
63
64         U2 = U1 + dt * matmul(bs,k)
65
66     return U2
67
68 def Step_size(dU, tolerance, q, dt):
69
70     if( norm(dU) > tolerance ):
71         size = dt *( tolerance / norm(dU) )** ( 1 / ( q + 1 ) ) #
Step_size
72     else:
73         size = dt # Step_size
74
75     return size
76
77 def Butcher_array():
78     q = [5,4]
79     Ne = 7
80
81     a = zeros( [Ne, Ne-1] )
82     b = zeros( [Ne] )
83     bs = zeros( [Ne] )
84     c = zeros( [Ne] )
85
86     c[:] = [ 0., 1./5, 3./10, 4./5, 8./9, 1., 1. ]
87
88     a[0,:] = [ 0., 0., 0., 0., 0., 0., 0. ]
89     a[1,:] = [ 0., 1./5, 0., 0., 0., 0., 0. ]

```

```

90     a[2,:] = [      3./40 ,      9./40 ,          0. ,          0. ,
91           0. ,      0. ]
92     a[3,:] = [      44./45 ,    -56./15 ,      32./9 ,          0. ,
93           0. ,      0. ]
94     a[4,:] = [ 19372./6561 , -25360./2187 , 64448./6561 , -212./729 ,
95           0. ,      0. ]
96     a[5,:] = [ 9017./3168 ,    -355./33 , 46732./5247 ,   49./176 ,
97   -5103./18656 ,          0. ]
98     a[6,:] = [      35./384 ,          0. , 500./1113 , 125./192 ,
99   -2187./6784 , 11./84 ]
100
101    b[:] = [ 35./384 , 0. , 500./1113 , 125./192 , -2187./6784 ,
102   11./84 , 0.]
103    bs[:] = [5179./57600, 0., 7571./16695, 393./640, -92097./339200,
104  187./2100, 1./40 ]
105
106    return (a, b, bs, c, q, Ne)

```

Listing 3.2: Runge-Kutta embedido

Continuando con el problema restringido definido mediante las Expresiones 3.1.2, se realiza el código que se puede observar en el Listing 3.3. Del mismo modo que en caso anterior, esta programación se apoya en [2]. Se puede observar que el cálculo de la estabilidad de los puntos de Lagrange se realiza mediante el análisis de la matriz Jacobiana, tal como se explico en la parte teórica de la asignatura. Aquellos sistemas con autovalores negativos serán estables.

```

1 from numpy import sqrt, zeros, array
2 from numpy.linalg import eig
3 from scipy.optimize import newton, fsolve
4 from ODES.Jab_Newt import Newton, Jacobiano
5
6 def RBP(U, t, mu): # Arenstorf_equations
7
8     r = U[:2] #Position
9     drdt = U[2:] #Velocity
10
11    p1 = sqrt( ( r[0] + mu )**2 + r[1]**2 )
12    p2 = sqrt( (r[0] - 1 + mu)**2 + r[1]**2 )
13
14    dvdt_1 = - ( 1 - mu ) * ( r[0] + mu ) / ( p1**3 ) - mu*( r[0] - 1 +
15    mu ) / ( p2**3 )
16    dvdt_2 = - ( 1 - mu ) * r[1] / ( p1**3 ) - mu*r[1] / ( p2**3 )
17
18    return array( [ drdt[0], drdt[1], 2*drdt[1] + r[0] + dvdt_1, -2*drdt
19 [0] + r[1] + dvdt_2 ] )
20
21 def Lagrange_points(U0, N_PL, mu):
22
23     LP = zeros( [5,2] )
24
25     def F(Y):
26         X = zeros(4)
27         X[:2] = Y
28         X[2:] = 0
29
30         r = X[:2]
31         drdt = X[2:]
32
33         p1 = sqrt( ( r[0] + mu )**2 + r[1]**2 )
34         p2 = sqrt( (r[0] - 1 + mu)**2 + r[1]**2 )
35
36         dvdt_1 = - ( 1 - mu ) * ( r[0] + mu ) / ( p1**3 ) - mu*( r[0] - 1 +
37         mu ) / ( p2**3 )
38         dvdt_2 = - ( 1 - mu ) * r[1] / ( p1**3 ) - mu*r[1] / ( p2**3 )
39
40         return array( [ drdt[0], drdt[1], 2*drdt[1] + r[0] + dvdt_1, -2*drdt
41 [0] + r[1] + dvdt_2 ] )
42
43     for i in range(N_PL):
44         if i == 0:
45             X0 = U0
46             Y0 = X0
47             X1 = F(Y0)
48             Y1 = X1
49             X2 = F(Y1)
50             Y2 = X2
51             X3 = F(Y2)
52             Y3 = X3
53             X4 = F(Y3)
54             Y4 = X4
55             X5 = F(Y4)
56             Y5 = X5
57
58             X = Y0
59             Y = Y1
60             X1 = Y1
61             Y1 = X2
62             X2 = Y2
63             Y2 = X3
64             X3 = Y3
65             Y3 = X4
66             X4 = Y4
67             Y4 = X5
68             X5 = Y5
69
70             X = X1
71             Y = Y1
72             X1 = Y1
73             Y1 = X2
74             X2 = Y2
75             Y2 = X3
76             X3 = Y3
77             Y3 = X4
78             X4 = Y4
79             Y4 = X5
80
81             X = X1
82             Y = Y1
83             X1 = Y1
84             Y1 = X2
85             X2 = Y2
86             Y2 = X3
87             X3 = Y3
88             Y3 = X4
89             X4 = Y4
90             Y4 = X5
91
92             X = X1
93             Y = Y1
94             X1 = Y1
95             Y1 = X2
96             X2 = Y2
97             Y2 = X3
98             X3 = Y3
99             Y3 = X4
100            X4 = Y4
101            Y4 = X5
102
103            X = X1
104            Y = Y1
105            X1 = Y1
106            Y1 = X2
107            X2 = Y2
108            Y2 = X3
109            X3 = Y3
110            Y3 = X4
111            X4 = Y4
112            Y4 = X5
113
114            X = X1
115            Y = Y1
116            X1 = Y1
117            Y1 = X2
118            X2 = Y2
119            Y2 = X3
120            X3 = Y3
121            Y3 = X4
122            X4 = Y4
123            Y4 = X5
124
125            X = X1
126            Y = Y1
127            X1 = Y1
128            Y1 = X2
129            X2 = Y2
130            Y2 = X3
131            X3 = Y3
132            Y3 = X4
133            X4 = Y4
134            Y4 = X5
135
136            X = X1
137            Y = Y1
138            X1 = Y1
139            Y1 = X2
140            X2 = Y2
141            Y2 = X3
142            X3 = Y3
143            Y3 = X4
144            X4 = Y4
145            Y4 = X5
146
147            X = X1
148            Y = Y1
149            X1 = Y1
150            Y1 = X2
151            X2 = Y2
152            Y2 = X3
153            X3 = Y3
154            Y3 = X4
155            X4 = Y4
156            Y4 = X5
157
158            X = X1
159            Y = Y1
160            X1 = Y1
161            Y1 = X2
162            X2 = Y2
163            Y2 = X3
164            X3 = Y3
165            Y3 = X4
166            X4 = Y4
167            Y4 = X5
168
169            X = X1
170            Y = Y1
171            X1 = Y1
172            Y1 = X2
173            X2 = Y2
174            Y2 = X3
175            X3 = Y3
176            Y3 = X4
177            X4 = Y4
178            Y4 = X5
179
180            X = X1
181            Y = Y1
182            X1 = Y1
183            Y1 = X2
184            X2 = Y2
185            Y2 = X3
186            X3 = Y3
187            Y3 = X4
188            X4 = Y4
189            Y4 = X5
190
191            X = X1
192            Y = Y1
193            X1 = Y1
194            Y1 = X2
195            X2 = Y2
196            Y2 = X3
197            X3 = Y3
198            Y3 = X4
199            X4 = Y4
200            Y4 = X5
201
202            X = X1
203            Y = Y1
204            X1 = Y1
205            Y1 = X2
206            X2 = Y2
207            Y2 = X3
208            X3 = Y3
209            Y3 = X4
210            X4 = Y4
211            Y4 = X5
212
213            X = X1
214            Y = Y1
215            X1 = Y1
216            Y1 = X2
217            X2 = Y2
218            Y2 = X3
219            X3 = Y3
220            Y3 = X4
221            X4 = Y4
222            Y4 = X5
223
224            X = X1
225            Y = Y1
226            X1 = Y1
227            Y1 = X2
228            X2 = Y2
229            Y2 = X3
230            X3 = Y3
231            Y3 = X4
232            X4 = Y4
233            Y4 = X5
234
235            X = X1
236            Y = Y1
237            X1 = Y1
238            Y1 = X2
239            X2 = Y2
240            Y2 = X3
241            X3 = Y3
242            Y3 = X4
243            X4 = Y4
244            Y4 = X5
245
246            X = X1
247            Y = Y1
248            X1 = Y1
249            Y1 = X2
250            X2 = Y2
251            Y2 = X3
252            X3 = Y3
253            Y3 = X4
254            X4 = Y4
255            Y4 = X5
256
257            X = X1
258            Y = Y1
259            X1 = Y1
260            Y1 = X2
261            X2 = Y2
262            Y2 = X3
263            X3 = Y3
264            Y3 = X4
265            X4 = Y4
266            Y4 = X5
267
268            X = X1
269            Y = Y1
270            X1 = Y1
271            Y1 = X2
272            X2 = Y2
273            Y2 = X3
274            X3 = Y3
275            Y3 = X4
276            X4 = Y4
277            Y4 = X5
278
279            X = X1
280            Y = Y1
281            X1 = Y1
282            Y1 = X2
283            X2 = Y2
284            Y2 = X3
285            X3 = Y3
286            Y3 = X4
287            X4 = Y4
288            Y4 = X5
289
290            X = X1
291            Y = Y1
292            X1 = Y1
293            Y1 = X2
294            X2 = Y2
295            Y2 = X3
296            X3 = Y3
297            Y3 = X4
298            X4 = Y4
299            Y4 = X5
300
301            X = X1
302            Y = Y1
303            X1 = Y1
304            Y1 = X2
305            X2 = Y2
306            Y2 = X3
307            X3 = Y3
308            Y3 = X4
309            X4 = Y4
310            Y4 = X5
311
312            X = X1
313            Y = Y1
314            X1 = Y1
315            Y1 = X2
316            X2 = Y2
317            Y2 = X3
318            X3 = Y3
319            Y3 = X4
320            X4 = Y4
321            Y4 = X5
322
323            X = X1
324            Y = Y1
325            X1 = Y1
326            Y1 = X2
327            X2 = Y2
328            Y2 = X3
329            X3 = Y3
330            Y3 = X4
331            X4 = Y4
332            Y4 = X5
333
334            X = X1
335            Y = Y1
336            X1 = Y1
337            Y1 = X2
338            X2 = Y2
339            Y2 = X3
340            X3 = Y3
341            Y3 = X4
342            X4 = Y4
343            Y4 = X5
344
345            X = X1
346            Y = Y1
347            X1 = Y1
348            Y1 = X2
349            X2 = Y2
350            Y2 = X3
351            X3 = Y3
352            Y3 = X4
353            X4 = Y4
354            Y4 = X5
355
356            X = X1
357            Y = Y1
358            X1 = Y1
359            Y1 = X2
360            X2 = Y2
361            Y2 = X3
362            X3 = Y3
363            Y3 = X4
364            X4 = Y4
365            Y4 = X5
366
367            X = X1
368            Y = Y1
369            X1 = Y1
370            Y1 = X2
371            X2 = Y2
372            Y2 = X3
373            X3 = Y3
374            Y3 = X4
375            X4 = Y4
376            Y4 = X5
377
378            X = X1
379            Y = Y1
380            X1 = Y1
381            Y1 = X2
382            X2 = Y2
383            Y2 = X3
384            X3 = Y3
385            Y3 = X4
386            X4 = Y4
387            Y4 = X5
388
389            X = X1
390            Y = Y1
391            X1 = Y1
392            Y1 = X2
393            X2 = Y2
394            Y2 = X3
395            X3 = Y3
396            Y3 = X4
397            X4 = Y4
398            Y4 = X5
399
400            X = X1
401            Y = Y1
402            X1 = Y1
403            Y1 = X2
404            X2 = Y2
405            Y2 = X3
406            X3 = Y3
407            Y3 = X4
408            X4 = Y4
409            Y4 = X5
410
411            X = X1
412            Y = Y1
413            X1 = Y1
414            Y1 = X2
415            X2 = Y2
416            Y2 = X3
417            X3 = Y3
418            Y3 = X4
419            X4 = Y4
420            Y4 = X5
421
422            X = X1
423            Y = Y1
424            X1 = Y1
425            Y1 = X2
426            X2 = Y2
427            Y2 = X3
428            X3 = Y3
429            Y3 = X4
430            X4 = Y4
431            Y4 = X5
432
433            X = X1
434            Y = Y1
435            X1 = Y1
436            Y1 = X2
437            X2 = Y2
438            Y2 = X3
439            X3 = Y3
440            Y3 = X4
441            X4 = Y4
442            Y4 = X5
443
444            X = X1
445            Y = Y1
446            X1 = Y1
447            Y1 = X2
448            X2 = Y2
449            Y2 = X3
450            X3 = Y3
451            Y3 = X4
452            X4 = Y4
453            Y4 = X5
454
455            X = X1
456            Y = Y1
457            X1 = Y1
458            Y1 = X2
459            X2 = Y2
460            Y2 = X3
461            X3 = Y3
462            Y3 = X4
463            X4 = Y4
464            Y4 = X5
465
466            X = X1
467            Y = Y1
468            X1 = Y1
469            Y1 = X2
470            X2 = Y2
471            Y2 = X3
472            X3 = Y3
473            Y3 = X4
474            X4 = Y4
475            Y4 = X5
476
477            X = X1
478            Y = Y1
479            X1 = Y1
480            Y1 = X2
481            X2 = Y2
482            Y2 = X3
483            X3 = Y3
484            Y3 = X4
485            X4 = Y4
486            Y4 = X5
487
488            X = X1
489            Y = Y1
490            X1 = Y1
491            Y1 = X2
492            X2 = Y2
493            Y2 = X3
494            X3 = Y3
495            Y3 = X4
496            X4 = Y4
497            Y4 = X5
498
499            X = X1
500            Y = Y1
501            X1 = Y1
502            Y1 = X2
503            X2 = Y2
504            Y2 = X3
505            X3 = Y3
506            Y3 = X4
507            X4 = Y4
508            Y4 = X5
509
510            X = X1
511            Y = Y1
512            X1 = Y1
513            Y1 = X2
514            X2 = Y2
515            Y2 = X3
516            X3 = Y3
517            Y3 = X4
518            X4 = Y4
519            Y4 = X5
520
521            X = X1
522            Y = Y1
523            X1 = Y1
524            Y1 = X2
525            X2 = Y2
526            Y2 = X3
527            X3 = Y3
528            Y3 = X4
529            X4 = Y4
530            Y4 = X5
531
532            X = X1
533            Y = Y1
534            X1 = Y1
535            Y1 = X2
536            X2 = Y2
537            Y2 = X3
538            X3 = Y3
539            Y3 = X4
540            X4 = Y4
541            Y4 = X5
542
543            X = X1
544            Y = Y1
545            X1 = Y1
546            Y1 = X2
547            X2 = Y2
548            Y2 = X3
549            X3 = Y3
550            Y3 = X4
551            X4 = Y4
552            Y4 = X5
553
554            X = X1
555            Y = Y1
556            X1 = Y1
557            Y1 = X2
558            X2 = Y2
559            Y2 = X3
560            X3 = Y3
561            Y3 = X4
562            X4 = Y4
563            Y4 = X5
564
565            X = X1
566            Y = Y1
567            X1 = Y1
568            Y1 = X2
569            X2 = Y2
570            Y2 = X3
571            X3 = Y3
572            Y3 = X4
573            X4 = Y4
574            Y4 = X5
575
576            X = X1
577            Y = Y1
578            X1 = Y1
579            Y1 = X2
580            X2 = Y2
581            Y2 = X3
582            X3 = Y3
583            Y3 = X4
584            X4 = Y4
585            Y4 = X5
586
587            X = X1
588            Y = Y1
589            X1 = Y1
590            Y1 = X2
591            X2 = Y2
592            Y2 = X3
593            X3 = Y3
594            Y3 = X4
595            X4 = Y4
596            Y4 = X5
597
598            X = X1
599            Y = Y1
600            X1 = Y1
601            Y1 = X2
602            X2 = Y2
603            Y2 = X3
604            X3 = Y3
605            Y3 = X4
606            X4 = Y4
607            Y4 = X5
608
609            X = X1
610            Y = Y1
611            X1 = Y1
612            Y1 = X2
613            X2 = Y2
614            Y2 = X3
615            X3 = Y3
616            Y3 = X4
617            X4 = Y4
618            Y4 = X5
619
620            X = X1
621            Y = Y1
622            X1 = Y1
623            Y1 = X2
624            X2 = Y2
625            Y2 = X3
626            X3 = Y3
627            Y3 = X4
628            X4 = Y4
629            Y4 = X5
630
631            X = X1
632            Y = Y1
633            X1 = Y1
634            Y1 = X2
635            X2 = Y2
636            Y2 = X3
637            X3 = Y3
638            Y3 = X4
639            X4 = Y4
640            Y4 = X5
641
642            X = X1
643            Y = Y1
644            X1 = Y1
645            Y1 = X2
646            X2 = Y2
647            Y2 = X3
648            X3 = Y3
649            Y3 = X4
650            X4 = Y4
651            Y4 = X5
652
653            X = X1
654            Y = Y1
655            X1 = Y1
656            Y1 = X2
657            X2 = Y2
658            Y2 = X3
659            X3 = Y3
660            Y3 = X4
661            X4 = Y4
662            Y4 = X5
663
664            X = X1
665            Y = Y1
666            X1 = Y1
667            Y1 = X2
668            X2 = Y2
669            Y2 = X3
670            X3 = Y3
671            Y3 = X4
672            X4 = Y4
673            Y4 = X5
674
675            X = X1
676            Y = Y1
677            X1 = Y1
678            Y1 = X2
679            X2 = Y2
680            Y2 = X3
681            X3 = Y3
682            Y3 = X4
683            X4 = Y4
684            Y4 = X5
685
686            X = X1
687            Y = Y1
688            X1 = Y1
689            Y1 = X2
690            X2 = Y2
691            Y2 = X3
692            X3 = Y3
693            Y3 = X4
694            X4 = Y4
695            Y4 = X5
696
697            X = X1
698            Y = Y1
699            X1 = Y1
700            Y1 = X2
701            X2 = Y2
702            Y2 = X3
703            X3 = Y3
704            Y3 = X4
705            X4 = Y4
706            Y4 = X5
707
708            X = X1
709            Y = Y1
710            X1 = Y1
711            Y1 = X2
712            X2 = Y2
713            Y2 = X3
714            X3 = Y3
715            Y3 = X4
716            X4 = Y4
717            Y4 = X5
718
719            X = X1
720            Y = Y1
721            X1 = Y1
722            Y1 = X2
723            X2 = Y2
724            Y2 = X3
725            X3 = Y3
726            Y3 = X4
727            X4 = Y4
728            Y4 = X5
729
730            X = X1
731            Y = Y1
732            X1 = Y1
733            Y1 = X2
734            X2 = Y2
735            Y2 = X3
736            X3 = Y3
737            Y3 = X4
738            X4 = Y4
739            Y4 = X5
740
741            X = X1
742            Y = Y1
743            X1 = Y1
744            Y1 = X2
745            X2 = Y2
746            Y2 = X3
747            X3 = Y3
748            Y3 = X4
749            X4 = Y4
750            Y4 = X5
751
752            X = X1
753            Y = Y1
754            X1 = Y1
755            Y1 = X2
756            X2 = Y2
757            Y2 = X3
758            X3 = Y3
759            Y3 = X4
760            X4 = Y4
761            Y4 = X5
762
763            X = X1
764            Y = Y1
765            X1 = Y1
766            Y1 = X2
767            X2 = Y2
768            Y2 = X3
769            X3 = Y3
770            Y3 = X4
771            X4 = Y4
772            Y4 = X5
773
774            X = X1
775            Y = Y1
776            X1 = Y1
777            Y1 = X2
778            X2 = Y2
779            Y2 = X3
780            X3 = Y3
781            Y3 = X4
782            X4 = Y4
783            Y4 = X5
784
785            X = X1
786            Y = Y1
787            X1 = Y1
788            Y1 = X2
789            X2 = Y2
790            Y2 = X3
791            X3 = Y3
792            Y3 = X4
793            X4 = Y4
794            Y4 = X5
795
796            X = X1
797            Y = Y1
798            X1 = Y1
799            Y1 = X2
800            X2 = Y2
801            Y2 = X3
802            X3 = Y3
803            Y3 = X4
804            X4 = Y4
805            Y4 = X5
806
807            X = X1
808            Y = Y1
809            X1 = Y1
810            Y1 = X2
811            X2 = Y2
812            Y2 = X3
813            X3 = Y3
814            Y3 = X4
815            X4 = Y4
816            Y4 = X5
817
818            X = X1
819            Y = Y1
820            X1 = Y1
821            Y1 = X2
822            X2 = Y2
823            Y2 = X3
824            X3 = Y3
825            Y3 = X4
826            X4 = Y4
827            Y4 = X5
828
829            X = X1
830            Y = Y1
831            X1 = Y1
832            Y1 = X2
833            X2 = Y2
834            Y2 = X3
835            X3 = Y3
836            Y3 = X4
837            X4 = Y4
838            Y4 = X5
839
840            X = X1
841            Y = Y1
842            X1 = Y1
843            Y1 = X2
844            X2 = Y2
845            Y2 = X3
846            X3 = Y3
847            Y3 = X4
848            X4 = Y4
849            Y4 = X5
850
851            X = X1
852            Y = Y1
853            X1 = Y1
854            Y1 = X2
855            X2 = Y2
856            Y2 = X3
857            X3 = Y3
858            Y3 = X4
859            X4 = Y4
860            Y4 = X5
861
862            X = X1
863            Y = Y1
864            X1 = Y1
865            Y1 = X2
866            X2 = Y2
867            Y2 = X3
868            X3 = Y3
869            Y3 = X4
870            X4 = Y4
871            Y4 = X5
872
873            X = X1
874            Y = Y1
875            X1 = Y1
876            Y1 = X2
877            X2 = Y2
878            Y2 = X3
879            X3 = Y3
880            Y3 = X4
881            X4 = Y4
882            Y4 = X5
883
884            X = X1
885            Y = Y1
886            X1 = Y1
887            Y1 = X2
888            X2 = Y2
889            Y2 = X3
890            X3 = Y3
891            Y3 = X4
892            X4 = Y4
893            Y4 = X5
894
895            X = X1
896            Y = Y1
897            X1 = Y1
898            Y1 = X2
899            X2 = Y2
900            Y2 = X3
901            X3 = Y3
902            Y3 = X4
903            X4 = Y4
904            Y4 = X5
905
906            X = X1
907            Y = Y1
908            X1 = Y1
909            Y1 = X2
910            X2 = Y2
911            Y2 = X3
912            X3 = Y3
913            Y3 = X4
914            X4 = Y4
915            Y4 = X5
916
917            X = X1
918            Y = Y1
919            X1 = Y1
920            Y1 = X2
921            X2 = Y2
922            Y2 = X3
923            X3 = Y3
924            Y3 = X4
925            X4 = Y4
926            Y4 = X5
927
928            X = X1
929            Y = Y1
930            X1 = Y1
931            Y1 = X2
932            X2 = Y2
933            Y2 = X3
934            X3 = Y3
935            Y3 = X4
936            X4 = Y4
937            Y4 = X5
938
939            X = X1
940            Y = Y1
941            X1 = Y1
942            Y1 = X2
943            X2 = Y2
944            Y2 = X3
945            X3 = Y3
946            Y3 = X4
947            X4 = Y4
948            Y4 = X5
949
950            X = X1
951            Y = Y1
952            X1 = Y1
953            Y1 = X2
954            X2 = Y2
955            Y2 = X3
956            X3 = Y3
957            Y3 = X4
958            X4 = Y4
959            Y4 = X5
960
961            X = X1
962            Y = Y1
963            X1 = Y1
964            Y1 = X2
965            X2 = Y2
966            Y2 = X3
967            X3 = Y3
968            Y3 = X4
969            X4 = Y4
970            Y4 = X5
971
972            X = X1
973            Y = Y1
974            X1 = Y1
975            Y1 = X2
976            X2 = Y2
977            Y2 = X3
978            X3 = Y3
979            Y3 = X4
980            X4 = Y4
981            Y4 = X5
982
983            X = X1
984            Y = Y1
985            X1 = Y1
986            Y1 = X2
987            X2 = Y2
988            Y2 = X3
989            X3 = Y3
990            Y3 = X4
991            X4 = Y4
992            Y4 = X5
993
994            X = X1
995            Y = Y1
996            X1 = Y1
997            Y1 = X2
998            X2 = Y2
999            Y2 = X3
1000           X3 = Y3
1001          Y3 = X4
1002          X4 = Y4
1003          Y4 = X5
1004
1005          X = X1
1006          Y = Y1
1007          X1 = Y1
1008          Y1 = X2
1009          X2 = Y2
1010         Y2 = X3
1011         X3 = Y3
1012         Y3 = X4
1013         X4 = Y4
1014         Y4 = X5
1015
1016         X = X1
1017         Y = Y1
1018         X1 = Y1
1019         Y1 = X2
1020         X2 = Y2
1021         Y2 = X3
1022         X3 = Y3
1023         Y3 = X4
1024         X4 = Y4
1025         Y4 = X5
1026
1027         X = X1
1028         Y = Y1
1029         X1 = Y1
1030         Y1 = X2
1031         X2 = Y2
1032         Y2 = X3
1033         X3 = Y3
1034         Y3 = X4
1035         X4 = Y4
1036         Y4 = X5
1037
1038         X = X1
1039         Y = Y1
1040         X1 = Y1
1041         Y1 = X2
1042         X2 = Y2
1043         Y2 = X3
1044         X3 = Y3
1045         Y3 = X4
1046         X4 = Y4
1047         Y4 = X5
1048
1049         X = X1
1050         Y = Y1
1051         X1 = Y1
1052         Y1 = X2
1053         X2 = Y2
1054         Y2 = X3
1055         X3 = Y3
1056         Y3 = X4
1057         X4 = Y4
1058         Y4 = X5
1059
1060         X = X1
1061         Y = Y1
1062         X1 = Y1
1063         Y1 = X2
1064         X2 = Y2
1065         Y2 = X3
1066         X3 = Y3
1067         Y3 = X4
1068         X4 = Y4
1069         Y4 = X5
1070
1071         X = X1
1072         Y = Y1
1073         X1 = Y1
1074         Y1 = X2
1075         X2 = Y2
1076         Y2 = X3
1077         X3 = Y3
1078
```

```

28     FX = RBP(X, 0 , mu)
29     return FX[2:4]
30
31     for i in range(N_LP):
32         LP[i,:] = fsolve( F, U0[i,:2] )
33
34     return LP
35
36 def stb_Lp(U_0, mu):
37
38     def F(Y):
39         return RBP(Y, 0 , mu)
40
41     J = Jacobiano(F, U_0)
42
43     Autovalor, autovector = eig(J)
44
45     return Autovalor

```

Listing 3.3: Problema restringido

Por último, se añade otra función que grafica los puntos lagrangianos y las órbitas calculadas alrededor de los mismo. Como se puede ver en el Listing 3.4, se realizan dos figuras, la primera de ella para todas las órbitas y la segunda de ellas realiza figuras centradas en los puntos L4 y L5, los cuales al ser estables requieren una observación más cercana, como se podrán observar en la Sección 3.4.

```

1 def Plot_LPO(U, L_P, mu, T_S, j, Save):
2
3     LP = ["L4", "L5", "L3", "L1", "L2"]
4
5     fig, ax = plt.subplots( figsize = (10, 10) )
6
7     for i in range( len(L_P) ):
8         ax.plot( L_P[i, 0], L_P [i, 1], 'o', label = LP[i] )
9
10    ax.plot( U[0,:], U[1,:], color = 'b', label = "Orbit" )
11    ax.set_xlabel("x")
12    ax.set_ylabel("y")
13    ax.set_title("Lagrange Points of Earth-Moon System and Orbit arround"
14                 " + LP[j]")
15    plt.legend()
16    ax.grid()
17
18    name = T_S.__name__
19    file = name + "_Lagrange Point_" + str(j) + ".png"
20
21    Save_plot(Save, file)
22
23    if LP[j] == "L4" or LP[j] == "L5":
24        fig, ay = plt.subplots( figsize = (7,7) )
25        ay.plot( L_P[j, 0], L_P [j, 1], 'o', label = LP[j] )
26        ay.plot( U[0,:], U[1,:], color = 'b', label = "Orbit" )
27        ay.set_xlabel("x")
28        ay.set_ylabel("y")
29        ay.set_title("Orbit arround" + LP[j])

```

```

29     plt.legend()
30     ay.grid()
31
32     name = T_S.__name__
33     file = name + "_Lagrange_Point_Zoom_" + str(j) + ".png"
34     Save_plot(Save, file)

```

Listing 3.4: Plot del problema restringido

### 3.4. Resultados

En esta Sección se mostrarán los resultados obtenidos en simulaciones con 10000 iteraciones y un tiempo de 100 segundos.

#### Puntos de Lagrange sistema Tierra-Luna

En primer lugar, tras el cálculo de los puntos de Lagrange y el análisis del la matriz Jacobiana particularizada en cada punto de Lagrange, se obtienen los resultados mostrados en la Tabla 3.1.

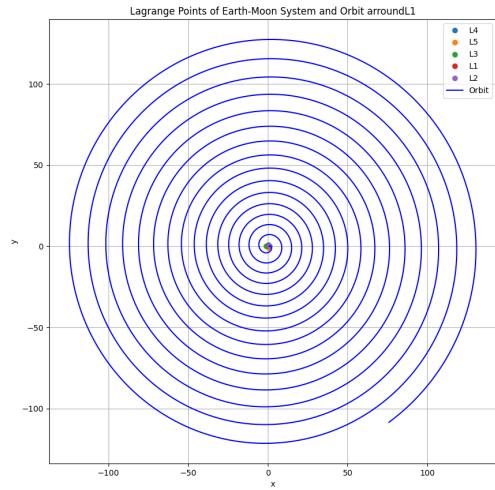
Tabla 3.1: Puntos de Lagrange Sistema Tierra-Luna

Puntos de Lagrange	Coordenada X	Coordenada Y	Estabilidad
L1	0.83691309	0	Inestable
L2	1.15568376	0	Inestable
L3	-1.00506282	0	Inestable
L4	0.487849	0.8660254	Estable
L5	0.487849	-0.8660254	Estable

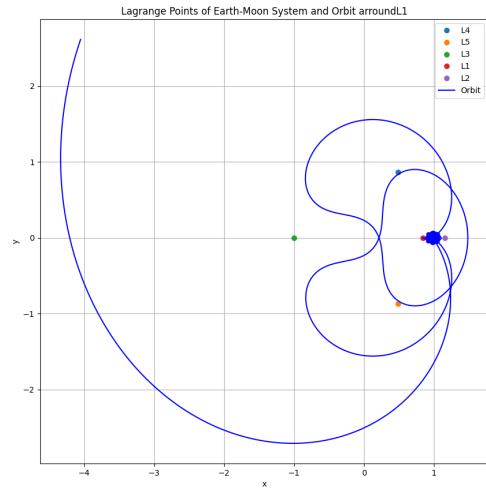
#### Órbitas entorno a los puntos de Lagrange

Una vez definidos los puntos Lagrangianos del sistema Tierra-Luna, se calculan una serie de órbitas a través de diferentes esquemas numéricos. Para una mejor comparación, la perturbación introducida en las órbitas siempre es la misma:  $\varepsilon = 3 \cdot 10^{-3}$ .

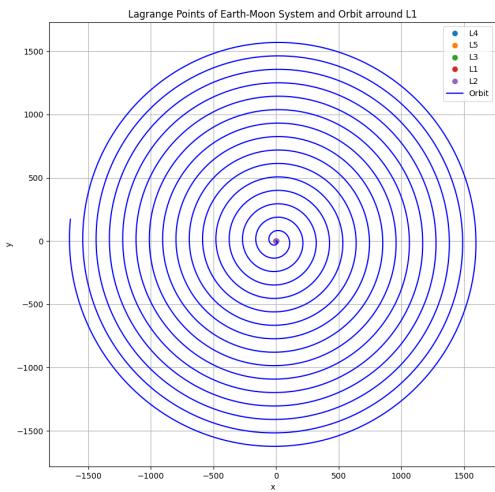
En primer lugar, en la Figura 3.2 se presentan los diferentes resultados obtenidos para órbitas en torno L1. Como se puede observar, a excepción de Runge-Kutta 4 o Runge-Kutta embobido, se obtienen malas convergencias de órbitas entorno a este punto. Comparando las Figuras 3.2b y 3.2f, se puede observar que para RK4 el resultado termina divergiendo, mientras que para el embobido no. Se puede ver que el punto L1 es un punto inestable, tal y como se refleja en la Tabla 3.1.



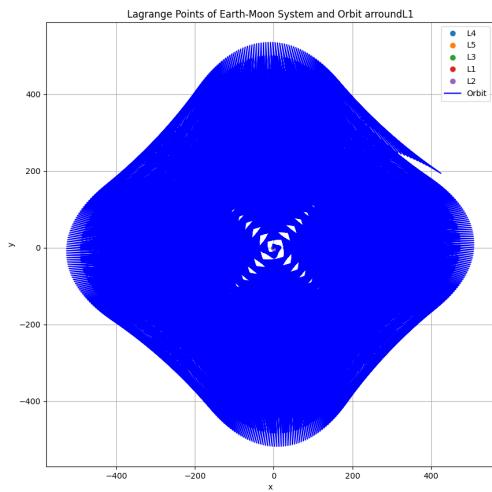
(a) Euler



(b) RK4

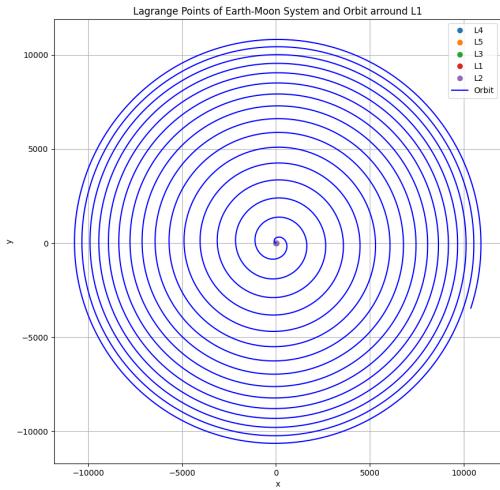


(c) Crank-Nicolson

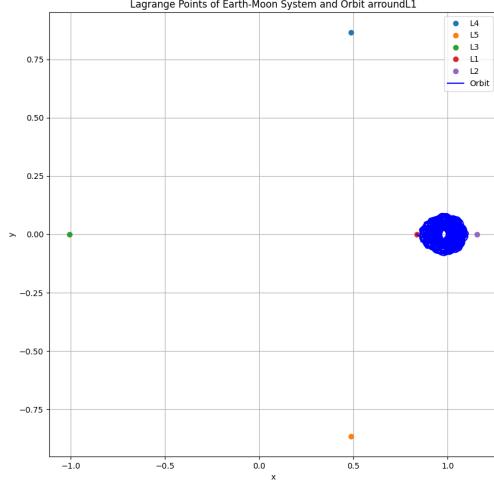


(d) LeapFrog

Figura 3.2: Órbitas entorno a L1



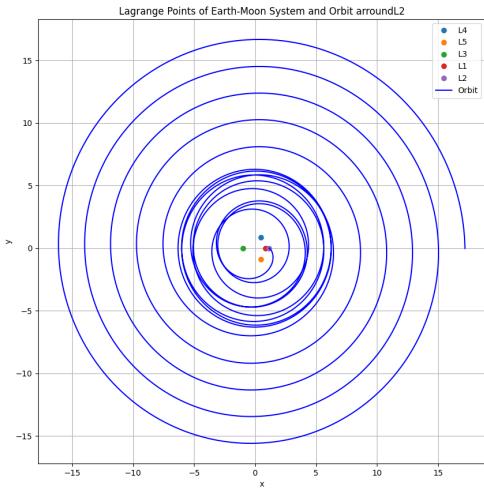
(e) Euler inverso



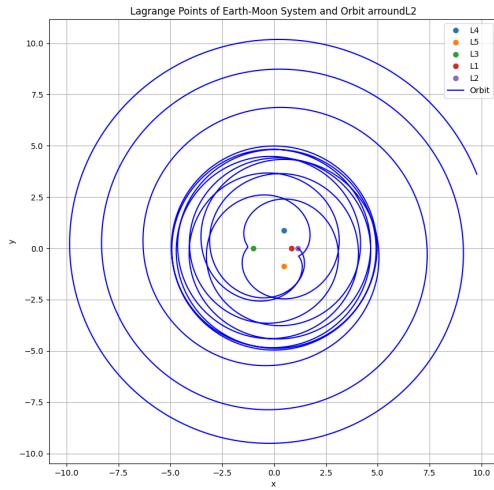
(f) ERK

Figura 3.2: Órbitas entorno a L1

Continuando con el punto L2, mostrado en el Figura 3.3, se observa que tanto RK4, como C-N o ERK obtienen resultados que terminan divergiendo. En el caso de los esquemas temporales de Euler o LeapFrog no se obtienen buenos resultados. Se puede ver que el punto L2 es un punto inestable, tal y como se refleja en la Tabla 3.1.

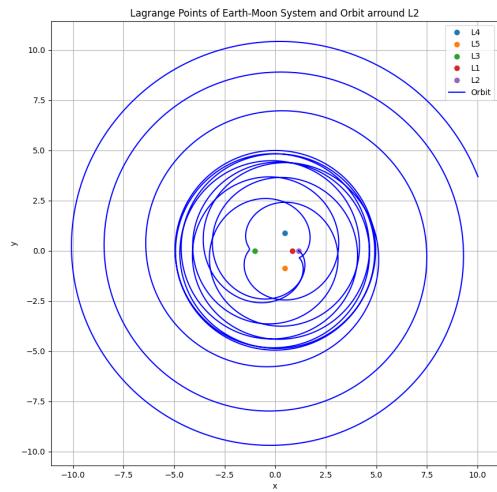


(a) Euler

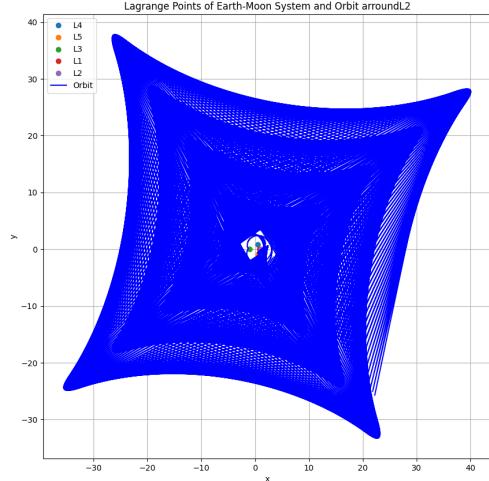


(b) RK4

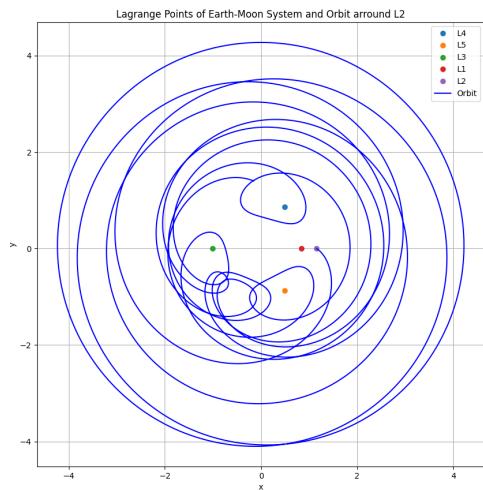
Figura 3.3: Órbitas entorno a L2



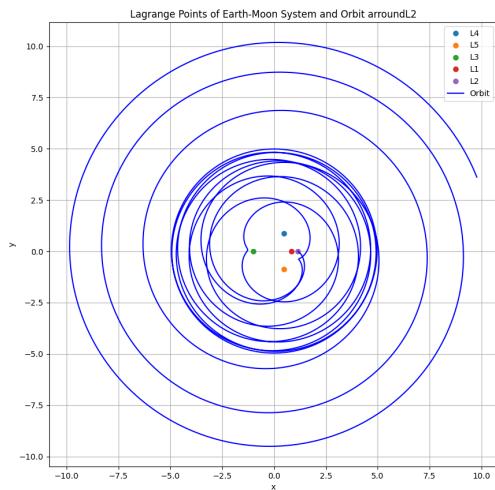
(c) Crank-Nicolson



(d) LeapFrog



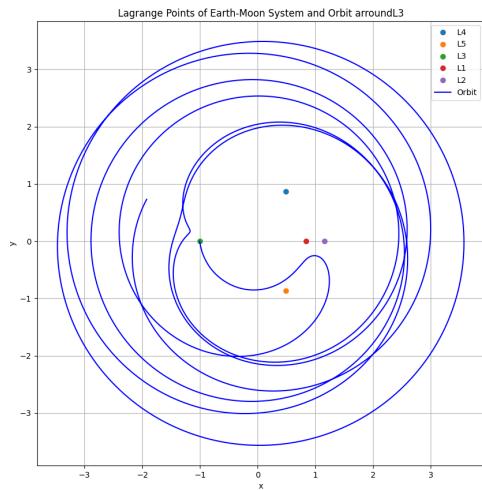
(e) Euler inverso



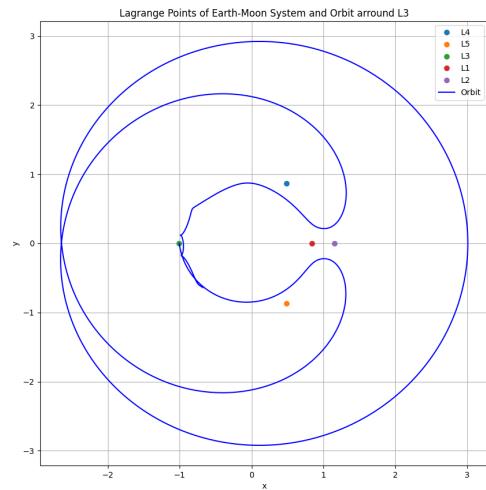
(f) ERK

Figura 3.3: Órbitas entorno a L2

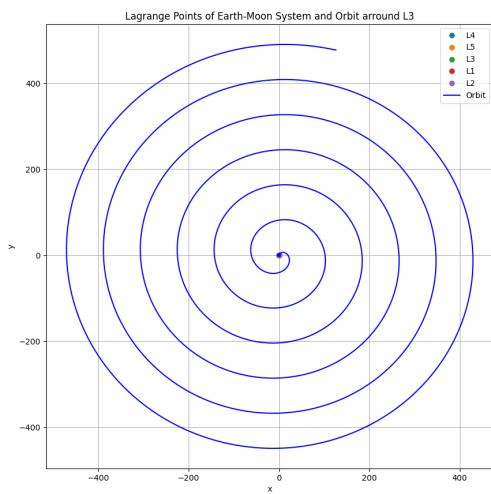
Continuando con los resultados en L3 (Figura 3.4, se puede observar que los resultados más buenos son los correspondientes a los métodos RK4 y ERK, Figuras 3.4b y 3.4f respectivamente. Estas órbitas son fácilmente comprobables, pues a través del punto L3 se puede realizar cambios de órbitas, pasando del interior al exterior y viceversa. Se puede ver que el punto L3 también es un punto inestable, tal y como se refleja en la Tabla 3.1.



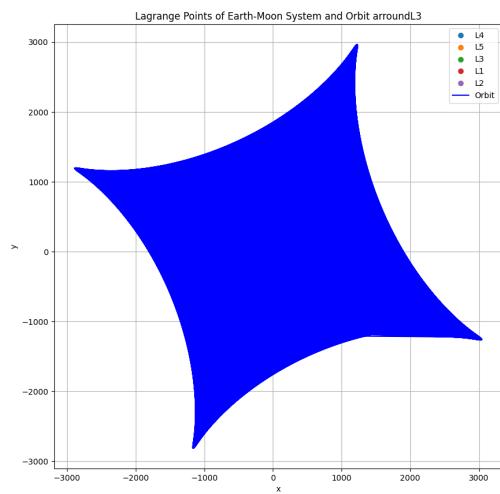
(a) Euler



(b) RK4

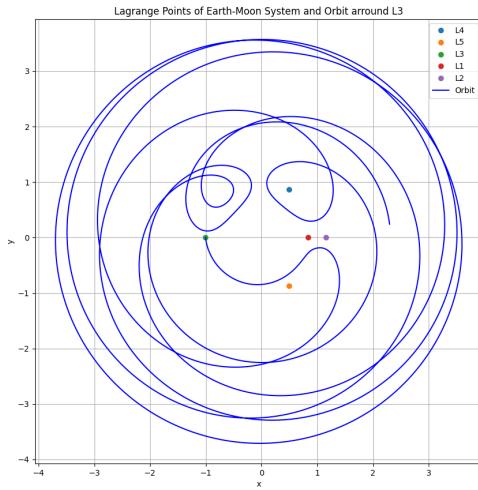


(c) Crank-Nicolson

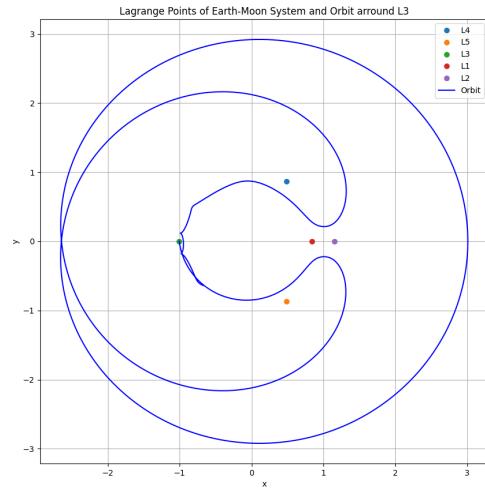


(d) LeapFrog

Figura 3.4: Órbitas entorno a L3



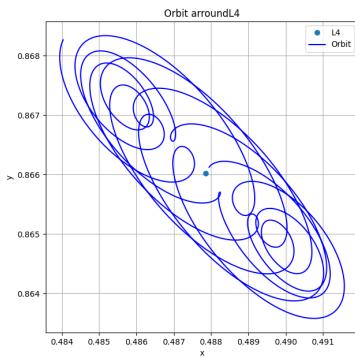
(e) Euler inverso



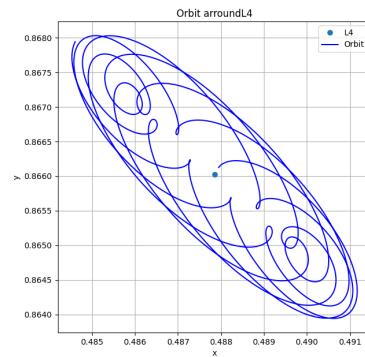
(f) ERK

Figura 3.4: Órbitas entorno a L3

Continuando con los Puntos L4 y L5 (Figuras 3.5 y 3.6), se puede observar que, todos los esquemas numéricos tienen una buena convergencia, aportando resultados similares. Como ya se mencionó en la Tabla 3.1, los puntos L4 y L5 son puntos estables. Estos resultados corroboran dicha afirmación.



(a) Euler



(b) RK4

Figura 3.5: Órbitas entorno a L4

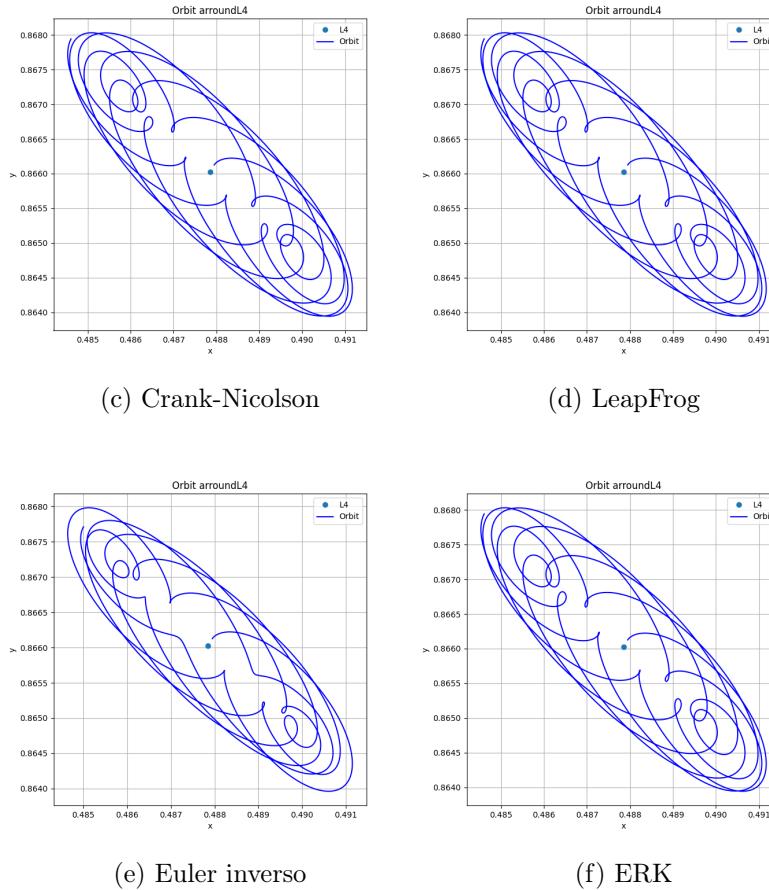


Figura 3.5: Órbitas entorno a L4

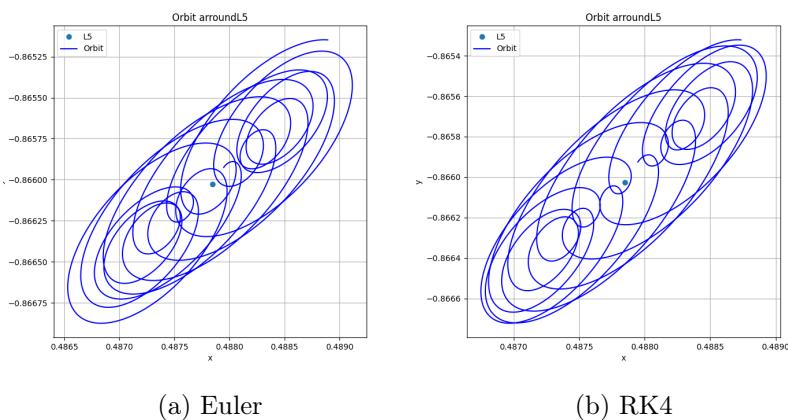


Figura 3.6: Órbitas entorno a L5

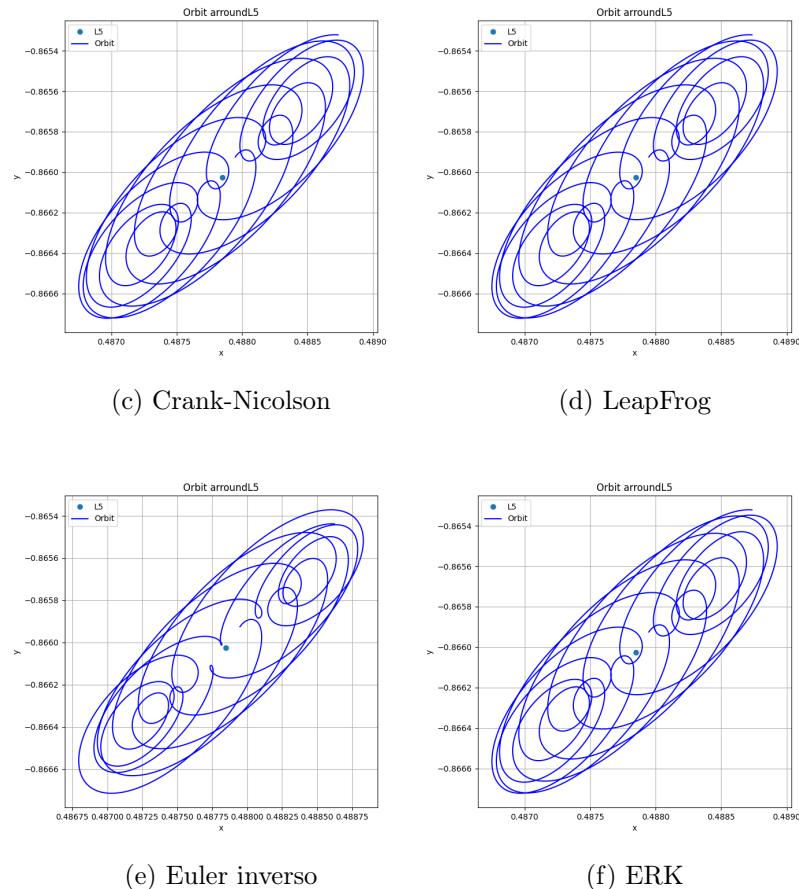


Figura 3.6: Órbitas entorno a L5

### 3.5. Comentarios y posibles mejoras

Cabe destacar la extremada sensibilidad de los resultados a la perturbación introducida, pues si es demasiado grande, los resultados no son buenos, y de forma análoga si son demasiado pequeños. Del mismo modo, "traducir"de Fortran a Python tiene una gran complejidad en los bucles, pues como Python comienza en 0 y Fortran en 1, hay que tener extremada precaución para no cometer errores en la iteraciones. Del mismo modo, también extremado cuidado en los bucles pues Python interpreta el intervalo seleccionado como cerrado y abierto, por ejemplo [0, 8).

El coste computacional es mayor que en los hitos 4 y 5, por lo que se podría implementar un procesamiento en paralelo reducir el tiempo de computación. Del mismo modo, se podrían introducir *decorators* que permitan graficar todas las figuras y también se podrían emplear *Kwarqs* para no depender del posicionamiento de los diferentes parámetros.

## Referencias

- [1] Miguel Ángel Rapado Tamarit Juan A. Hernández Ramos. *Advanced Programming for Numerical Calculations: Climbing Python & Fortran*.
- [2] Juan A. Hernández Ramos; Javier Escoto López. *How to learn Applied Mathematics through modern FORTRAN*.