


Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestone VI Report

Ampliación de Matemáticas I

30 de noviembre de 2022

Autora:

- Rosa Martínez Rubiella

1. Introducción

En primer lugar, el Hito VI añade el esquema temporal de Runge-Kutta Embebido (eRK) a la colección de esquemas numéricos del repositorio. A continuación, se analiza el problema restringido de los tres cuerpos y se implementa, obteniendo de esta forma los puntos de Lagrange de un sistema orbital. Finalmente, se analiza la estabilidad de dichos puntos mediante la jacobiana del sistema y sus autovalores, y se grafican órbitas con perturbaciones alrededor de dichos puntos. Al final del informe se hace un review del código implementado, así como de las dependencias funcionales entre los módulos, que han sido reestructurados.

2. Runge-Kutta Embebido

El esquema de Runge-Kutta Embebido permite controlar de forma automática el paso de tiempo que utiliza el integrador. Se realiza una aproximación local del error de truncación para, de esta forma, dedicar más esfuerzo computacional (un paso de tiempo más pequeño) a las regiones de la solución que así lo requieran. Para calcular dicho error se utilizan dos esquemas temporales tipo Runge-Kutta, de orden **q** y orden **q-1**. La solución en el instante (n+1) se aproxima por:

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

con

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

donde **a** es la matriz de Butcher. Esta matriz es triangular inferior y con diagonal nula, para que el Runge-Kutta pueda ser resuelto explícitamente. Restando las soluciones de los dos esquemas temporales, se obtiene, a través de la definición del error de truncación, el paso de tiempo mínimo a utilizar:

$$\Delta t_{min} = \Delta t \{ \frac{\epsilon}{||U_*^{n+1} - U^{n+1}||} \}^{\frac{1}{q}}$$

En la sección 6 se explica cómo se ha implementado este esquema en Python más detalladamente.

3. Problema Restringido de los tres cuerpos

El problema restringido circular de los tres cuerpos es un caso particular del problema de los N-cuerpos, implementado en el Milestone V. En este caso, se realiza la hipótesis de que existe un cuerpo de masa nula comparada con los dos cuerpos principales. Este cuerpo se mueve bajo la influencia gravitacional de los otros dos cuerpos, cuya masa se relaciona por el parámetro:

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

De esta forma, los cuerpos másicos quedan situados en las posiciones $(\mu, 0)$ y $((1-\mu), 0)$. Se asumen un movimiento contenido en el plano formado por las órbitas circulares, que dan como resultado un sistema de dos ecuaciones que definen el movimiento del tercer cuerpo:

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

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

con

$$p_1 = \sqrt{(x+\mu)^2 + y^2} \quad p_2 = \sqrt{(x-1+\mu)^2 + y^2}$$

En la sección 6 se muestra la implementación de estas ecuaciones mediante su vector de estado, así como la estructura de resolución mediante el ya existente módulo del Problema de Cauchy.

4. Puntos de Lagrange y estabilidad

Los puntos de Lagrange se corresponden a los puntos del espacio donde la velocidad y aceleración del tercer cuerpo son nulas, ya que se compensan los efectos gravitacionales de los cuerpos másicos. Es decir, los puntos donde la derivada del vector de estado, la función F, es nula. La solución a este problema son unos puntos de equilibrio estable o inestable, que son respectivamente los mínimos o máximos de la función potencial gravitatoria (ya que se trata de un sistema conservativo).

Para analizar la estabilidad de estos puntos de equilibrio, se calculan los autovalores de la matriz jacobiana del sistema en cada punto del espacio. Un punto de Lagrange se considera estable si la parte real de dichos autovalores es nula, es decir, si se trata de un autovalor puramente imaginario. En el caso que se va a tratar, el sistema Tierra-Luna, se tienen dos puntos de Lagrange estables (L4 y L5), con órbitas periódicas, y tres puntos inestables (L1, L2 y L3).

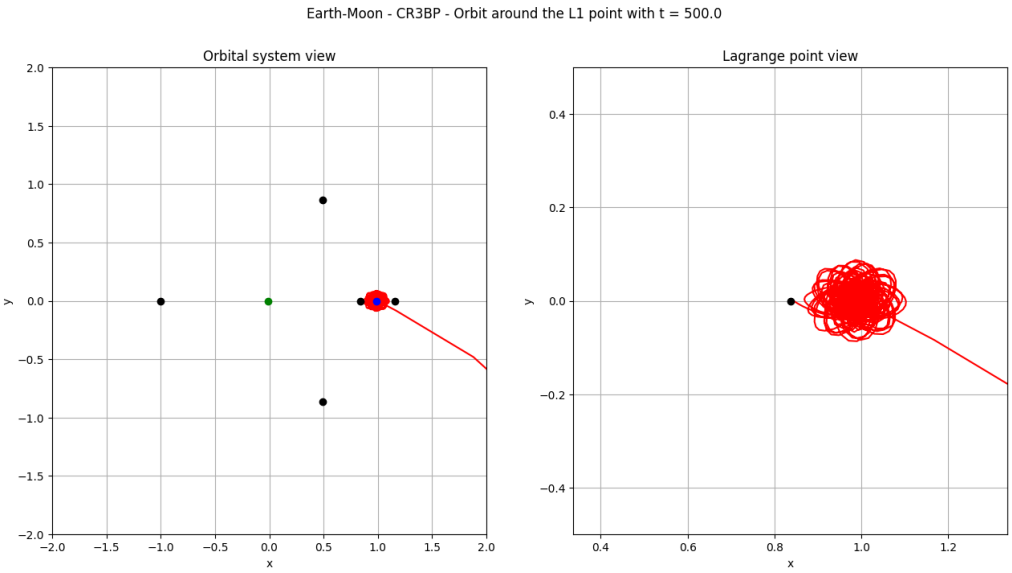
Sistema	μ	**L1**	**L2**	**L3**	**L4**	**L5**
Tierra-Luna	1.2151e-2	(0.83691309, 0.)	(1.15568376, 0.)	(-1.00506282, 0.)	(0.487849, 0.8660254)	(0.487849, -0.8660254)

Tabla 1: Puntos de Lagrange del sistema Tierra-Luna

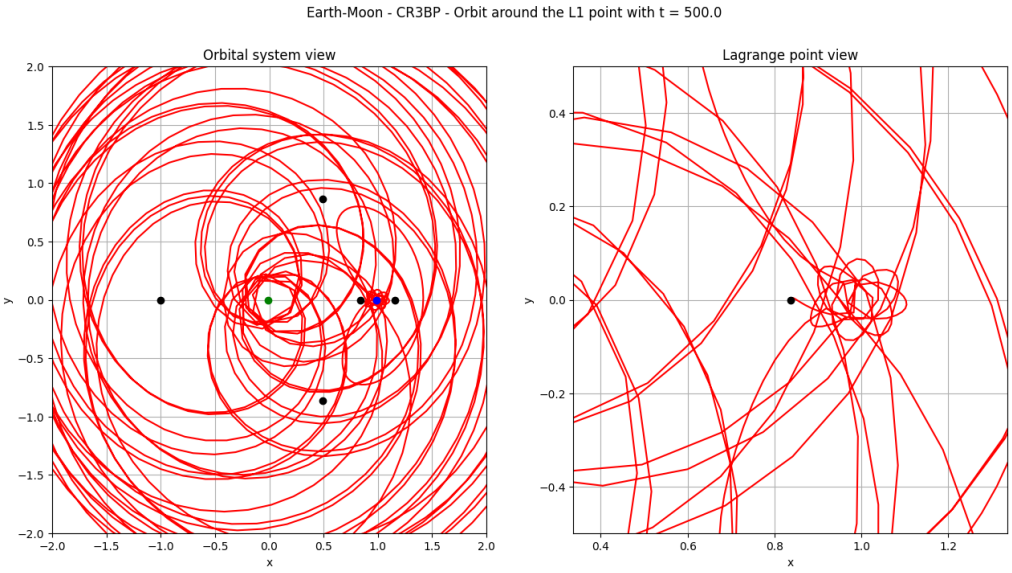
5. Simulación del sistema Tierra-Luna

Se ha simulado el sistema orbital Tierra-Luna. Se perturba el cuerpo levemente respecto a un punto de Lagrange para observar su comportamiento. De esta forma es posible comprobar que los puntos de Lagrange L4 y L5 son estables, el cuerpo realiza órbitas cuasi-periódicas a su alrededor. Primero, se ha utiliza un esquema temporal Runge-Kutta embebido tipo DOPRI-4-5, con una tolerancia de 1e-10. Se grafican en negro los puntos de Lagrange, en verde la Tierra y en azul la Luna. Las perturbaciones en cada simulación son aleatorias, se generan por un número random que demuestran el carácter caótico del sistema. La figura 2 representa dos casos diferentes de perturbar el punto L1 (inestable). En el primer caso el cuerpo se ve atraido por la Luna y orbita a su alrededor durante un

mayor tiempo, mientras que en el segundo caso es influenciado por el campo terrestre y la órbita se aleja de la Luna. La figura 1 muestra un caso con 1000 s de simulación para el mismo punto de Lagrange. Se puede observar como este punto de equilibrio inestable sirve de paso entre dos pozos gravitacionales diferentes.



(a) Perturbación-1, punto L1



(b) Perturbación-2, punto L1

Figura 1: Órbitas perturbadas entorno a L1, t = 1000 s, eRK

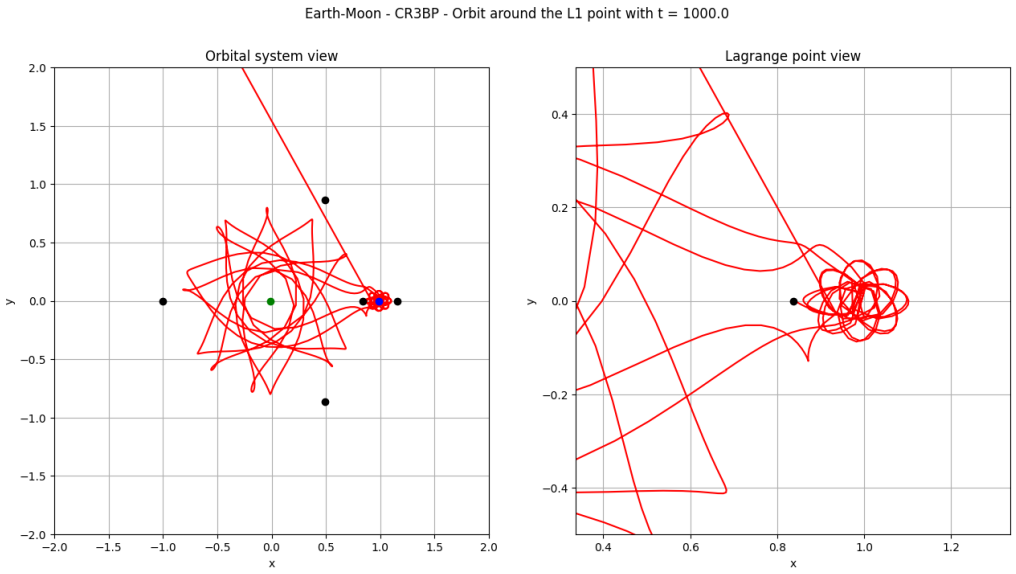
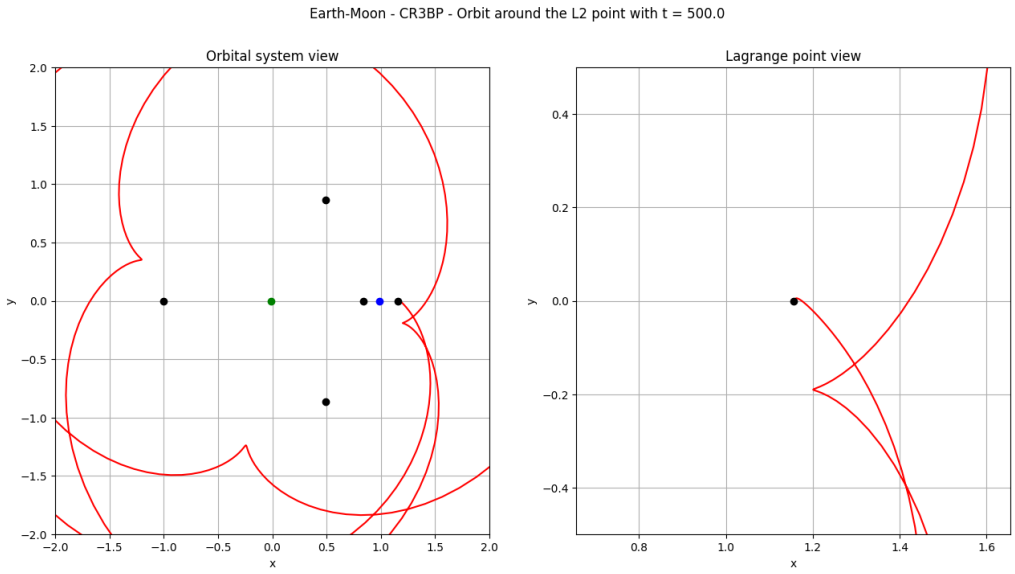


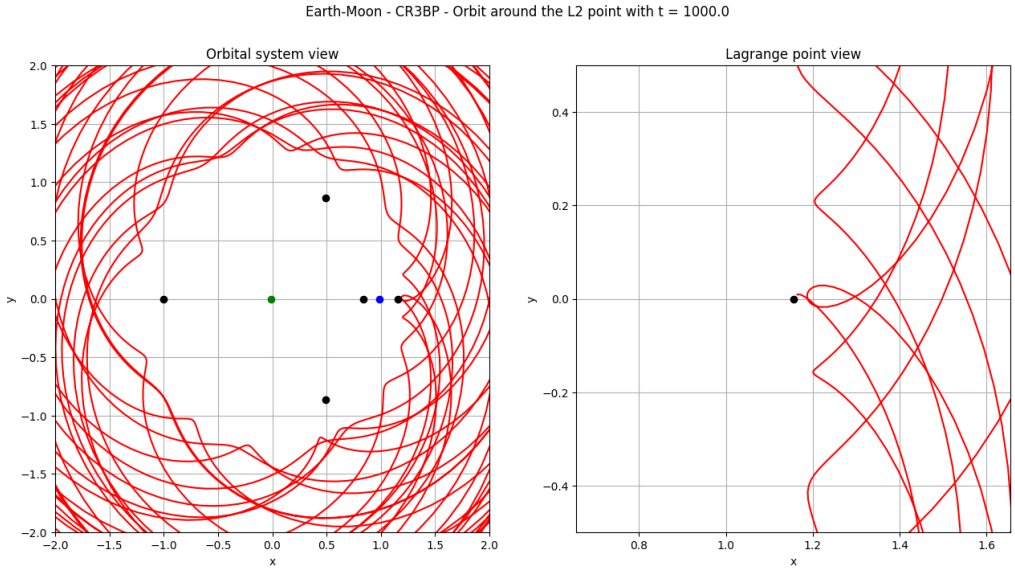
Figura 2: Órbitas perturbadas entorno a L1, t = 500 s, eRK

En la figura 3 se muestran dos simulaciones entorno al punto de Lagrange L2, también con carácter inestable. Y, de la misma forma, la figura 4 representa el caso del punto L3. Estos dos últimos casos muestran un carácter progresivo en el tiempo, es decir, aunque la perturbación para los dos tiempos de simulación es diferente, el progreso de la órbita es similar. El campo potencial de L2 es exterior al sistema orbital, mientras que el de L3 y L1 comparten zonas de propagación.

Sin embargo, como se puede observar en las figuras 5 y 6, los puntos L4 y L5 tienen carácter estable. El cuerpo mantiene una órbita cuasi periódica alrededor de dichos puntos, que además son prácticamente simétricos uno respecto del otro. En el caso del punto L4, en la simulación de 1000 s de duración, la órbita se perturba de forma que se semi-desestabiliza.

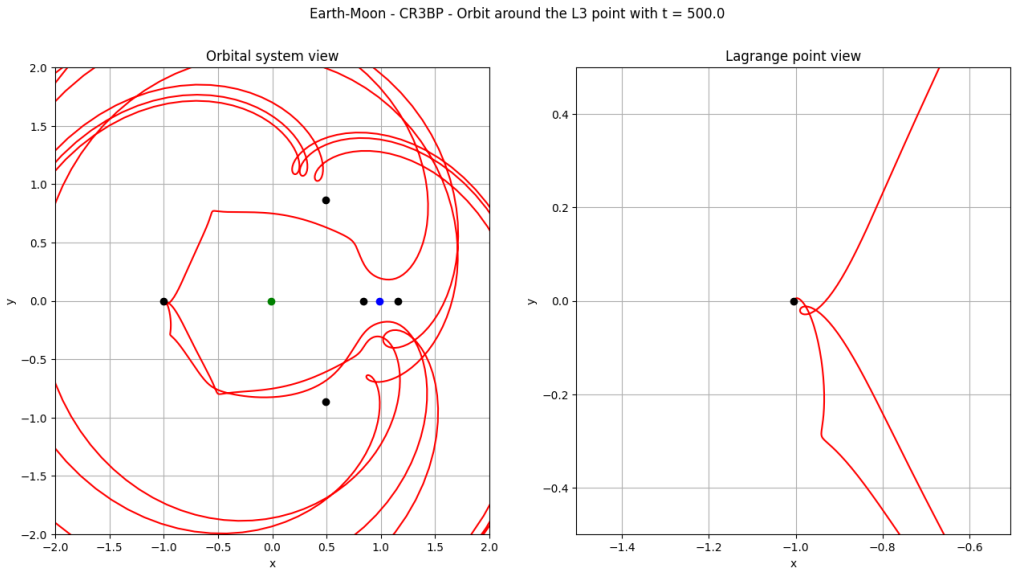


(a) Perturbación-1, punto L2

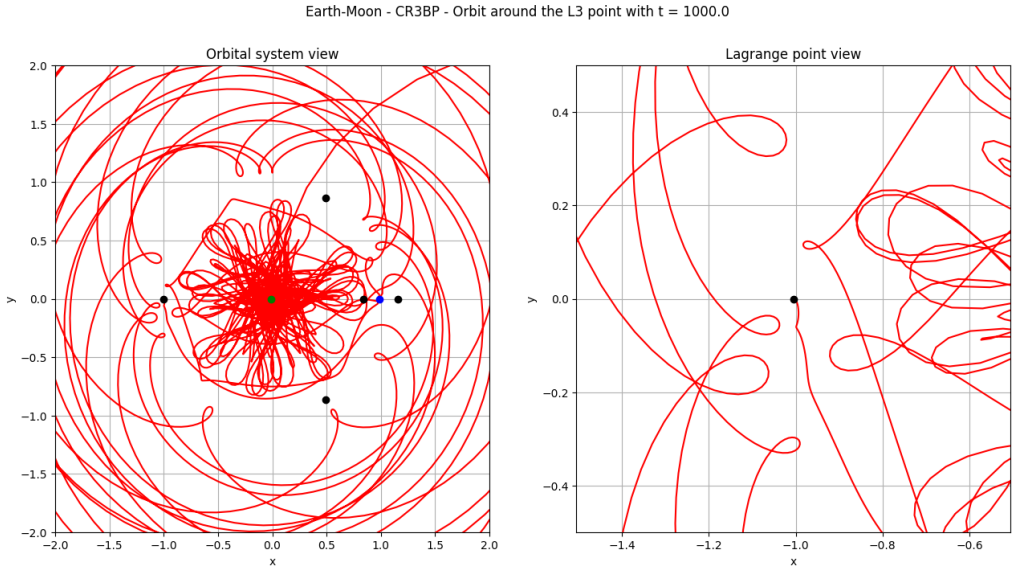


(b) Perturbación-2, punto L2

Figura 3: Órbitas perturbadas entorno a L2, t = 500 s y t = 1000 s, eRK

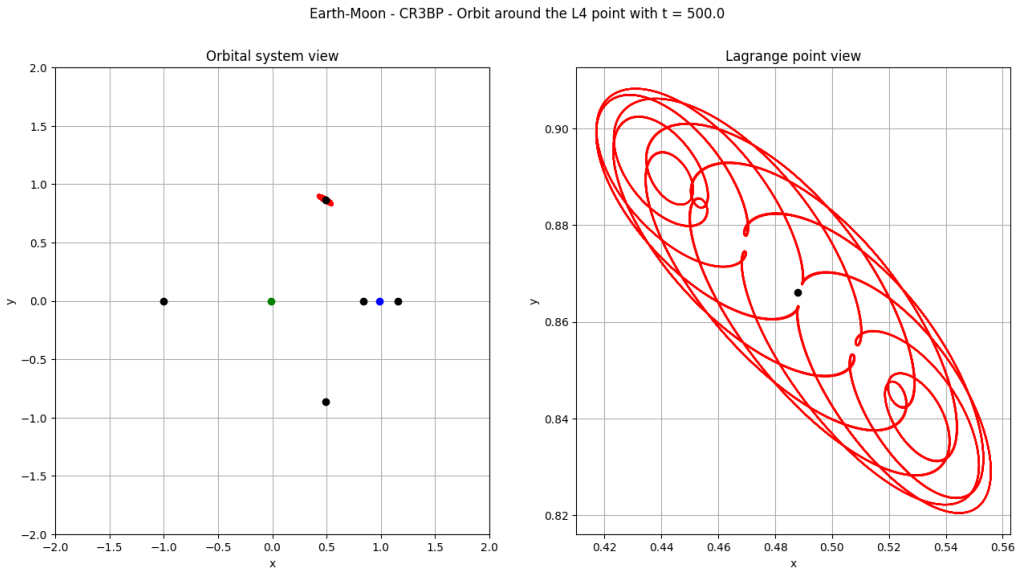


(a) Perturbación-1, punto L3

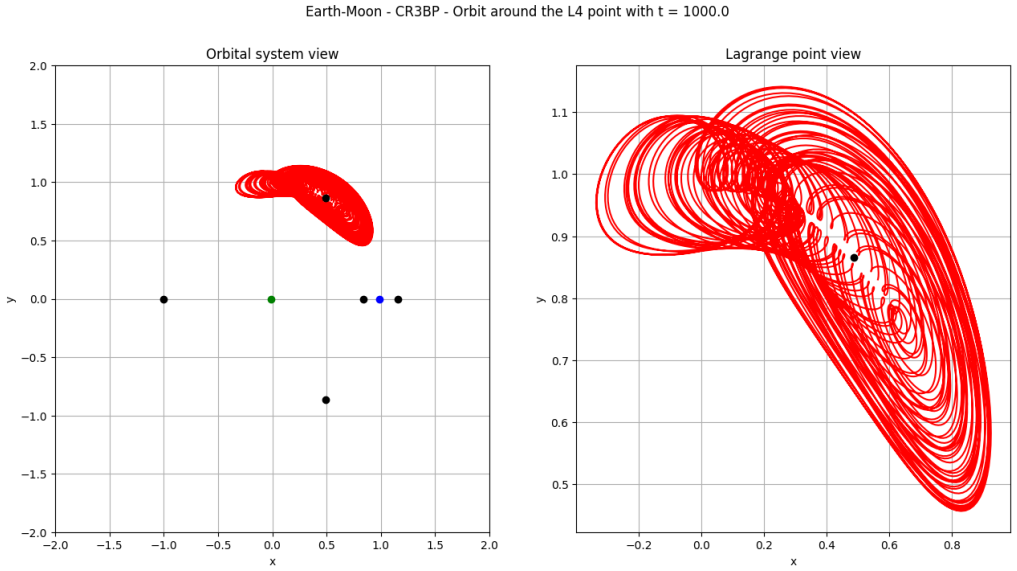


(b) Perturbación-2, punto L3

Figura 4: Órbitas perturbadas entorno a L3, t = 500 s y t = 1000 s, eRK

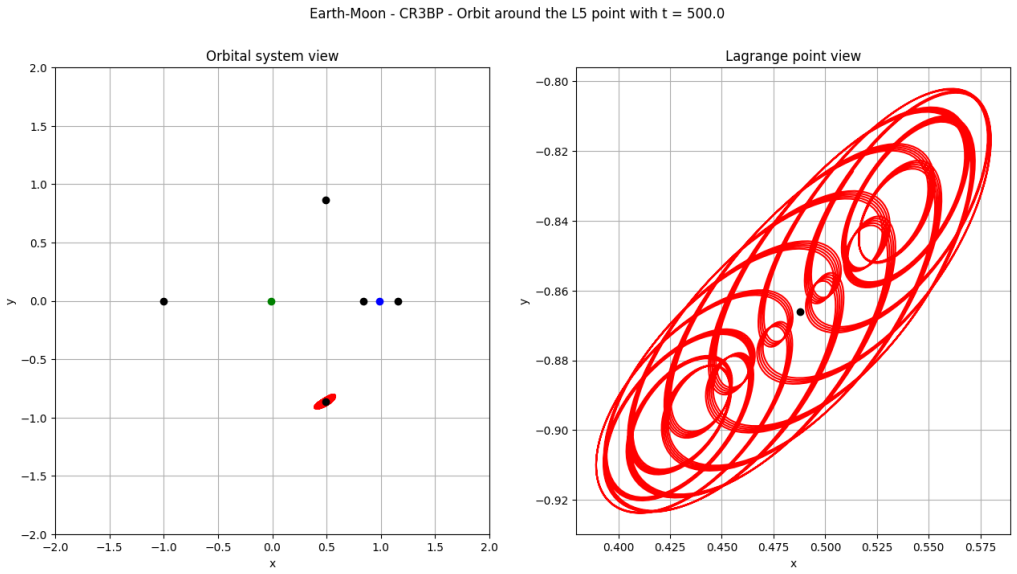


(a) Perturbación-1, punto L4

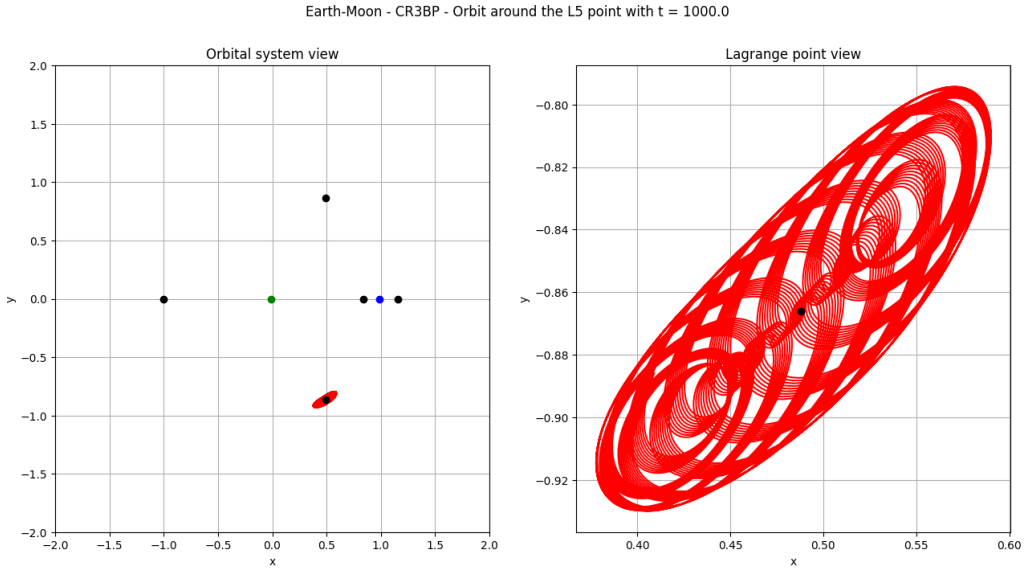


(b) Perturbación-2, punto L4

Figura 5: Órbitas perturbadas entorno a L4, t = 500 s y t = 1000 s, eRK



(a) Perturbación-1, punto L5



(b) Perturbación-2, punto L5

Figura 6: Órbitas perturbadas entorno a L5, t = 500 s y t = 1000 s, eRK

Además, se ha realizado una comparación entre diferentes esquemas temporales para una perturbación alrededor del punto de Lagrange L4, del que se conoce mejor el resultado a esperar. En las figuras 7 y 8 se puede observar el carácter disipativo del Euler Inverso, así como el carácter divergente del Euler Explícito. Los esquemas de Crank-Nicolson y Runge-Kutta-4 (figuras 9 y 10) muestran buenas aproximaciones del problema, mientras que el esquema de Leap-Frog no converge a la solución (figura 11).

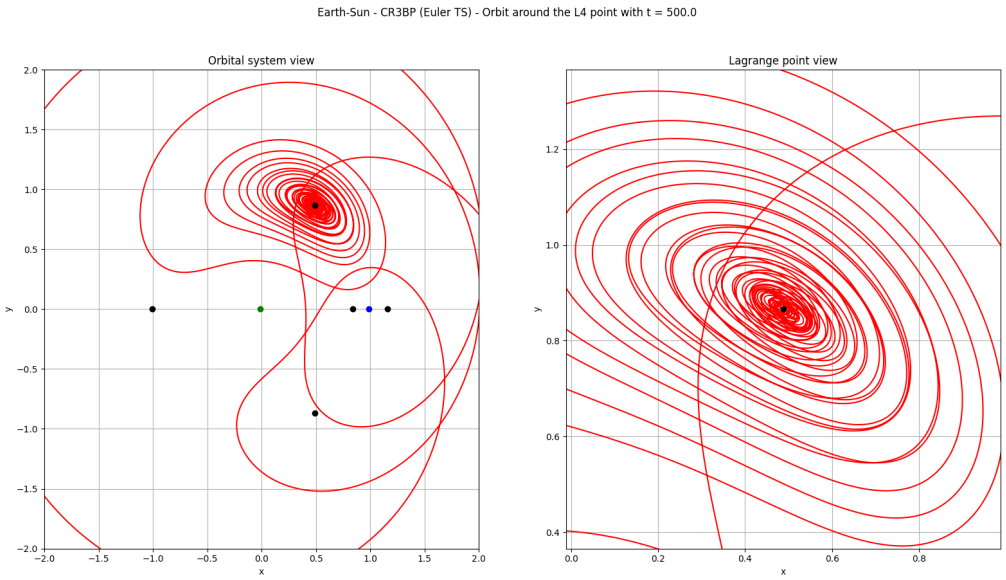


Figura 7: Órbita perturbada entorno a L4, t = 500 s, Euler Explícito

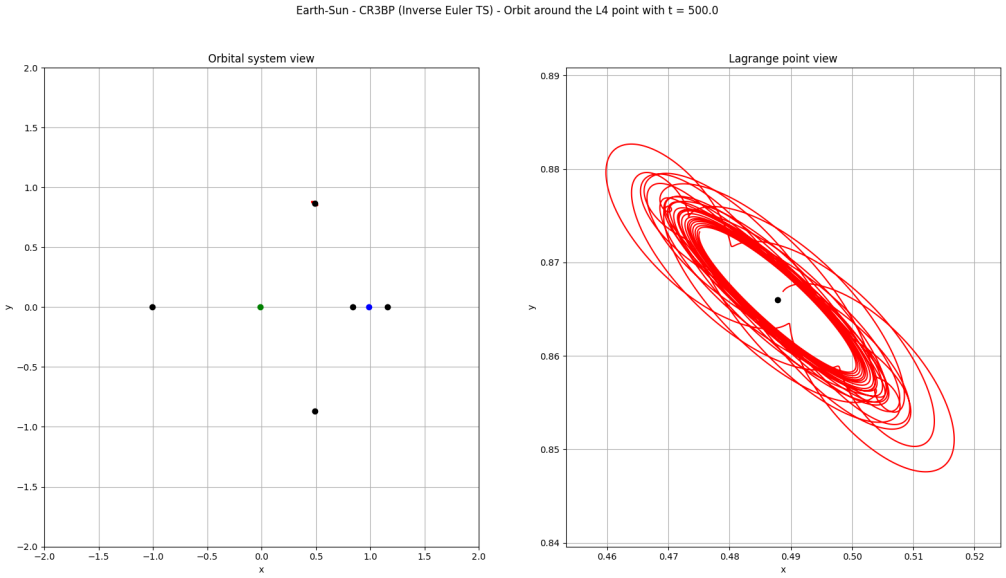


Figura 8: Órbita perturbada entorno a L4, t = 500 s, Euler Inverso

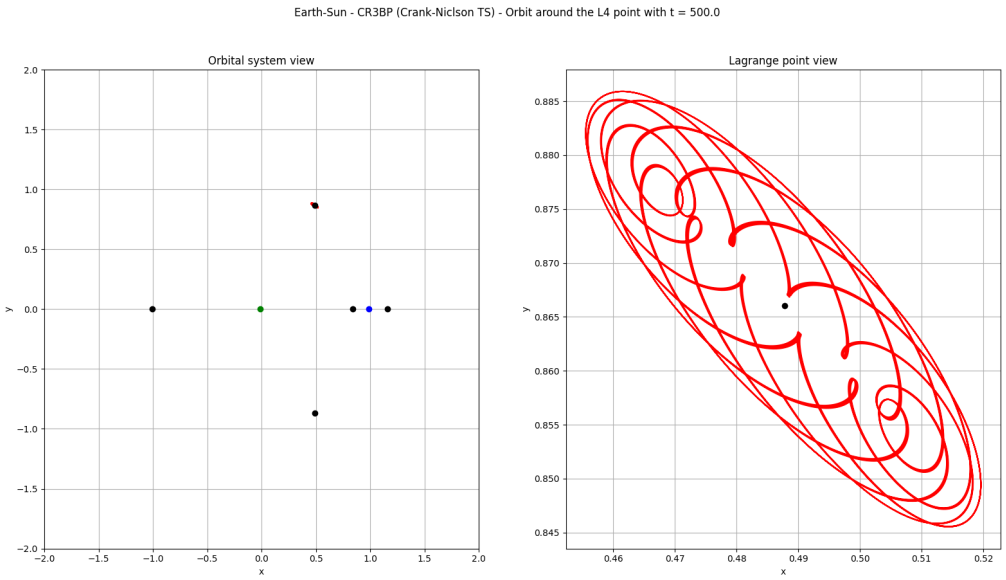


Figura 9: Órbita perturbada entorno a L4, t = 500 s, Crank-Nicolson

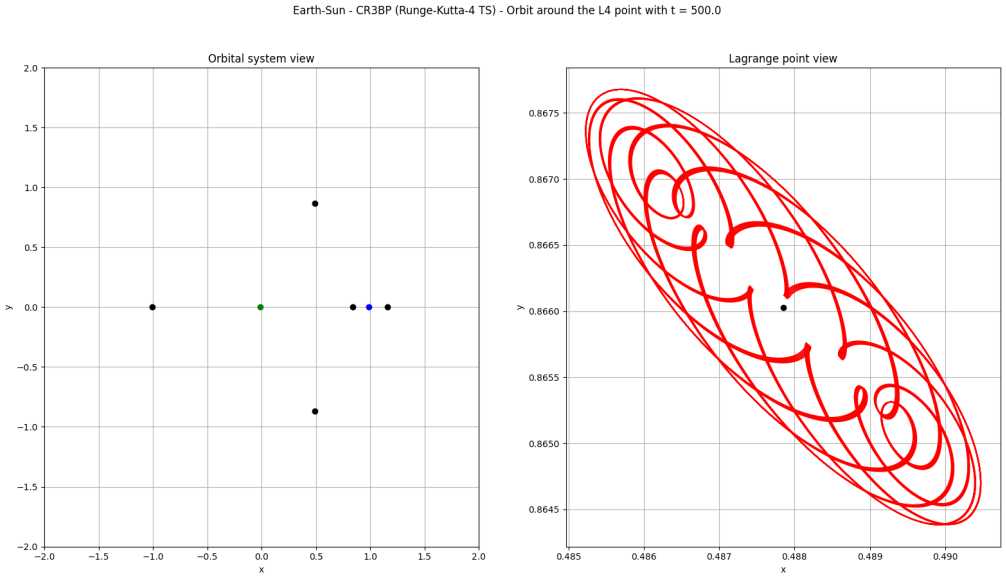


Figura 10: Órbita perturbada entorno a L4, t = 500 s, Runge-Kutta-4

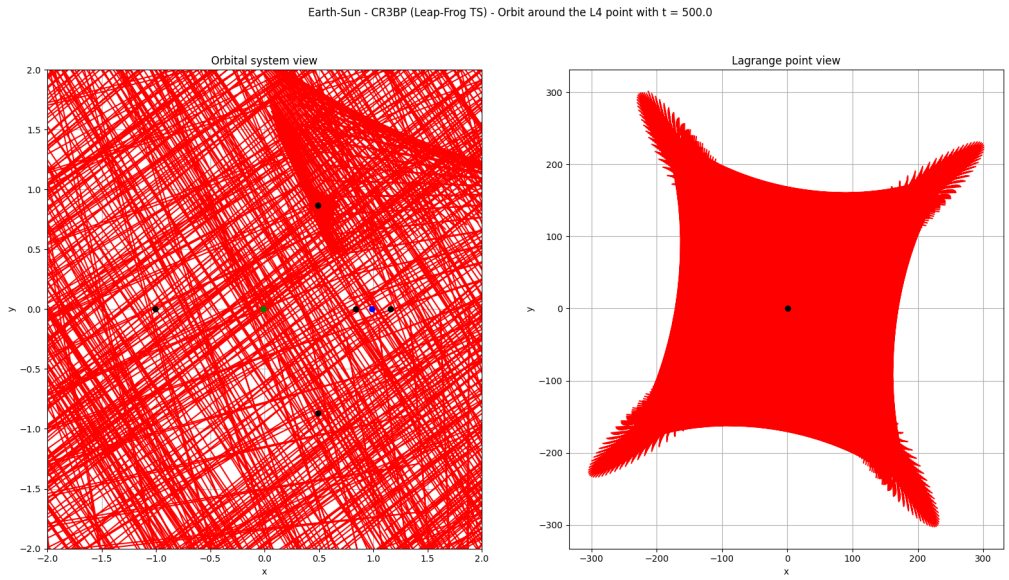


Figura 11: Órbita perturbada entorno a L4, t = 500 s, Leap-Frog

6. Code Review

Primero, se ha creado un módulo dentro de la carpeta de *Temporal Schemes* llamado *Adaptative Stepsize* que contiene las funciones necesarias para implementar el esquema temporal de Runge-Kutta Embebido. Además de las aquí mostradas, se tiene otra función adicional (no incluida por su extensión) que contiene las matrices de Butcher, y los datos necesarios para cada esquema embebido.

```
1
2  def Embedded_RK(U, dt, F, t):
3
4      RK_Method = "DOPRI54"  # Se selecciona un metodo embebido de
       Runge-Kutta
5      tol = 1e-10 # Se establece una tolerancia o error maximo
6
7      V1 = RK(RK_Method, "First", U, t, dt, F) # Se integra con el
       Runge-Kutta de orden inferior
8      V2 = RK(RK_Method, "Second", U, t, dt, F) # Se integra con el
       Runge-Kutta de orden superior
9
10     a, b, bs, c, q, Ne = Butcher_array(RK_Method) # Se obtienen los
       parametros del metodo seleccionado
11
12     h = min(dt, Step_Size(V1-V2, tol, dt,  min(q))) # El paso de tiempo
       minimo es calculado, se elige el minimo entre el paso que se ha
       predeterminado y el calculado por el propio codigo en la funcion
       step_size
13
14     N_n = int(dt/h)+1 # Nuevo numero de puntos de integracion
15     n_dt = dt/N_n # Nuevo mallado en el intervalo de tiempo predefinido
16
17     V1 = U
18     V2 = U
```

```
19
20      for i in range(N_n): # Se integra la solucion con el nuevo mallado,
        en el instante de tiempo correspondiente
21          time = t + i*dt/int(N_n)
22          V1 = V2
23          V2 = RK(RK_Method, "First", V1, time, n_dt, F)
24
25      U2 = V2
26
27      return U2
28
29  def RK(name, tag, U1, t, dt, F):
30      a, b, bs, c, q, Ne = Butcher_array(name) # Datos del metodo
        seleccionado
31      k = zeros([Ne, len(U1)])
32
33      k[0,:] = F(U1, t + c[0]*dt) # Se inicializa la matriz k
34
35      if tag=="First": # Runge-Kutta de orden inferior
36
37          for i in range(1,Ne):
38              Up = U1
39              for j in range(i):
40                  Up = Up + dt*a[i,j]*k[j,:]
41
42              k[i,:] = F(Up, t + c[i]*dt)
43
44          U2 = U1 + dt*matmul(b,k)
45
46      elif tag == "Second": # Runge-Kutta de orden superior
47
48          for i in range(1,Ne):
49              Up = U1
50              for j in range(i):
51                  Up = Up + dt*a[i,j]*k[j,:]
52
53              k[i,:] = F(Up, t + c[i]*dt)
54
55          U2 = U1 + dt*matmul(bs,k)
56
57      return U2
58
59  def Step_Size(dU, tol, dt, q): #dU es el error estimado, tol el error
        maximo permitido y q el orden de dicho error
60
61      normT = norm(dU) # Norma del error estimado
62
63      if normT > tol: # Se compara al error permitido
64          step_size = dt*(tol/normT)**(1/(q+1))
65      else:
66          step_size = dt # Si es menor, el paso de tiempo se queda como el
        preestablecido
67
68      return step_size
```

Listing 1: Embedded Runge-Kutta

Dentro de la carpeta de problemas de ODEs, se ha añadido un módulo llamado *Body3 Restricted*, que contiene la función F del problema restringido de los tres cuerpos, así como la

función que resuelve los puntos de Lagrange del sistema (mediante un método de Newton previamente implementado), y los autovalores de la matriz jacobiana del sistema (para el estudio de su estabilidad). Devido a que la función CR3BP tiene un argumento extra respecto al resto de funciones (el parámetro másico μ), se realiza un wrapper dentro del *main*, para poder meterla en el Problema de Cauchy con tal solo dos argumentos de entrada.

```
1
2  def CR3BP(U, t, mu): # Derivada del vector de estado del problema
       restringido de los tres cuerpos, se introduce el parametro masico
       como input
3      r = U[0:2] #Posicion
4      drdt = U[2:4] #Velocidad
5
6      p1 = sqrt((r[0] + mu)**2 + r[1]**2)
7      p2 = sqrt((r[0] - 1 + mu)**2 + r[1]**2)
8
9      dvdt_1 = -(1-mu)*(r[0]+mu)/(p1**3) - mu*(r[0] - 1 +mu)/(p2**3) #
       Derivada de la velocidad, aceleracion en x
10     dvdt_2 = -(1-mu)*r[1]/(p1**3) - mu*r[1]/(p2**3) # Aceleracion en y
11
12     return array([ drdt[0], drdt[1], 2*drdt[1] + r[0] + dvdt_1,
       -2*drdt[0] + r[1] + dvdt_2])
13
14 def Lagrange_points(U_0, NL, mu): #Encuentra los puntos de Lagrange, U_0
       son las condiciones iniciales, NL el numero de puntos de Lagrange y
       mu el parametro masico
15
16     LP = zeros([5,2])
17
18     def F(Y): # Es una funcion que wrappea a la funcion del problema de
       los tres cuerpos e impone unas condiciones iniciales nulas en
       velocidad
19         X = zeros(4)
20         X[0:2] = Y
21         X[2:4] = 0
22         FX = CR3BP(X, 0 , mu)
23         return FX[2:4]
24
25     for i in range(NL):
26         LP[i,:] = newton(F, U_0[i,0:2]) # Se llama al metodo de Newton
       para cada punto de Lagrange
27
28     return LP
29
30
31 def Stability_LP(U_0, mu): # Estabilidad de un punto de Lagrange
32
33     def F(Y): # Wrappea la funcion CR3BP con un solo argumento de
       entrada, en forma de funcion
34         return CR3BP(Y, 0 , mu)
35
36     A = Jacobian(F, U_0) # A es la matriz jacobiana del sistema
37     values, vectors = eig(A) # Se hallan los autovalores de la matriz A
38
39     return values
```

Listing 2: Body3 Restricted module

Finalmente, se refleja el *main document*, desde el cual se ejecuta el código implementado para este trabajo.

```

# CR3BP SOLUTION

N = 20000 # Numero de puntos de integracion

t = linspace(0, 100, N) # Dominio temporal de integracion

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

def F(U,t): # Wrapper para llamar a la funcion con tal solo dos
    argumentos de entrada

    return br.CR3BP(U, t, mu)

# LAGRANGE POINTS

NL = 5 # Numero de puntos de Lagrange
U_0L = zeros([NL,4])

U_0L[0,:] = array([0.8, 0.6, 0, 0]) # Condiciones iniciales de los
    puntos de Lagrange
U_0L[1,:] = array([0.8, -0.6, 0, 0])
U_0L[2,:] = array([-0.1, 0, 0, 0])
U_0L[3,:] = array([0.1, 0, 0, 0])
U_0L[4,:] = array([1.01, 0, 0, 0])

x_p = br.Lagrange_points(U_0L, NL, mu) # Posicion de los puntos de
    Lagrange

# ORBITS AROUND LAGRANGE POINTS

U_0LP = zeros(4)
U_0SLP = zeros(4)

eps = 1e-3*random() # Perturbacion
sel_LG = 1 # Punto de Lagrange seleccionado para estudiar su estabilidad
    y representar su orbita

U_0LP[0:2] = x_p[sel_LG-1,:] + eps
U_0LP[2:4] = eps

U_0SLP[0:2] = x_p[sel_LG-1,:]
U_0SLP[2:4] = 0

# STABILITY OF LAGRANGE POINTS

U_LP = Cauchy_Problem(F, t, U_0LP, eRK.Embedded_RK) # Orbita alrededor
    del punto de Lagrange seleccionado

eingvalues = br.Stability_LP(U_0SLP, mu) # Autovalores del jacobiano del
    sistema
print(around(eingvalues.real,8))

# PAINT THE ORBIT

```

```
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(x_p[i,0], x_p[i,1] , 'o', color = "k")

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

ax1.set_xlim(-2,2)
ax1.set_ylim(-2,2)
ax1.set_title("Orbital system view")
ax2.set_title("Lagrange point view")
ax2.set_xlim(x_p[sel_LG-1,0]-0.5,x_p[sel_LG-1,0]+0.5)
ax2.set_ylim(x_p[sel_LG-1,1]-0.5,x_p[sel_LG-1,1]+0.5)
fig.suptitle("Earth-Sun - CR3BP (Inverse Euler TS) - Orbit around the L2
    point with t = " + str(t[N-1]))
for ax in fig.get_axes():
    ax.set(xlabel='x', ylabel='y')
    ax.grid()

plt.show()
```

Listing 3: Hito VI main

Cabe destacar que se ha reorganizado el código en subcarpetas, donde se ha incluido el archivo __init__.py para poder usarlas como paquetes independientes al hacer los imports. De esta forma, el grafo de dependencias funcionales para este hito sería el siguiente (figura 12):

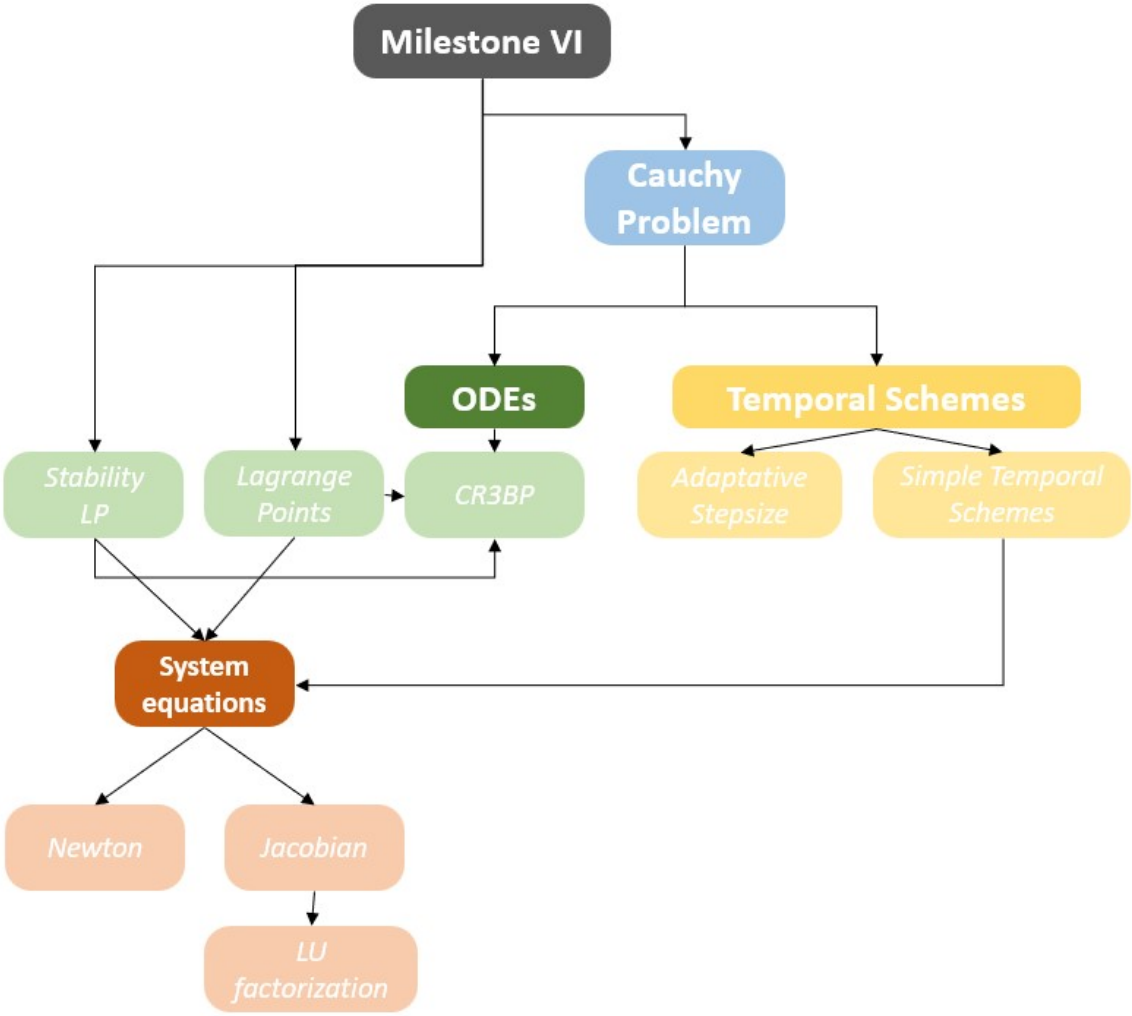


Figura 12: Esquema de las dependencias funcionales del Hito VI

7. Conclusiones y sugerencias

Para resolver el sistema Tierra-Sol se necesita un esfuerzo computacional mucho mayor, ya que el parámetro μ es del orden 1e-7. Para ello se podría incluir una paralelización del código mediante numba, que acelerase el proceso. En mi caso, al tener solamente dos cores y 8 GB de RAM, el proceso se hace muy lento. Además, para resolver el sistema de manera más precisa, se podría utilizar un método de Runge-Kutta embebido de órdenes 8-9.