


Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestone IV, V y VI

Ampliación de Matemáticas

Autor:

Miguel del Fresno Zarza

Índice

1. Introducción

En este informe, se presenta el trabajo realizado para la resolución de los hitos IV, V y VI. De esta manera, se comenzará explicando el enunciado propuesto para cada uno de los hitos junto con su fundamento teórico, después se expondrá el código correspondiente de cada uno de los hitos, seguidamente se expondrán los resultados y finalmente se realizarán unas conclusiones acerca de estos hitos.

1.1. Hito IV

En el Hito IV, se tiene como objetivo obtener un oscilador lineal mediante los métodos de integración nunérica de Euler, Euler inverso, Crank-Nicholson, Runge Kutta de cuarto orden y Leap−Frog.

Por otra parte, se deberán de obtener las regiones de estabilidad de cada uno de los métodos ya comentados y analizar los resultados.

1.1.1. Fundamento teórico

El fundamento teórico de cada uno de los métodos ya fue desarrollado mediante los anteriores hitos exceptuando el método de Leap-Frog, que se comentará a continuación junto con el oscilador y las regiones de estabilidad.

Oscilador

El esquema del oscilador se rige por la Ecuación 1 mostrada a continuación.

$$\begin{cases} \dot{x}(t) + x(t) = 0 \\ x(t=0) = x_0 = 1 \\ \dot{x}(t=0) = \dot{x}_0 = 0 \end{cases} \quad , \quad x, t \in \mathbb{R} \tag{1}$$

Leap−Frog

El método de Leap−Frog, se caracteriza fundamentalmente por el hecho de que calcula los datos del instante t_{n+1} conociendo los datos de t_{n-1}. De esta manera, este método únicamente presenta problemas para el instante inicial debido a que no conoce los datos del instante previo, por lo que para el inicio se calculará el primer paso mediante Euler. Este método queda descrito en la Ecuación 2.

$$U^{n+1} = U^n + 2\Delta t F^n \tag{2}$$

Regiones de Estabilidad

height**Esquema Temporal**	$\Pi(r,\omega)$
Euler	$r-1-w$
Euler Inverso	$r-\frac{1}{1-w}$
Cranck-Nicholson	$r-\frac{1+\frac{w}{2}}{1-\frac{w}{2}}$
Runge-Kutta de Cuarto Orden	$r-1-w-\frac{w^2}{2}-\frac{w^3}{6}-\frac{w^4}{24}$
Leap-Frog	r^2-1

1.2. Hito V

En este hito, se tiene como objetivo la resolución del problema de los *N-Cuerpos*, el cual queda explicado en la siguiente ecuación.

$$\begin{cases} \ddot{r}_i(t) = -\sum_{\substack{j=1 \\ j\neq 1}}^{j=N} \frac{r_i(t)-r_j(t)}{\|r_i(t)-r_j(t)\|^3} \\ r_i(t=0) = r_0 \\ \dot{r}_i(t=0) = \dot{r}_0 \end{cases} , i = 1,2,\ldots,N, \quad r_i \in \mathbb{R}^2, t \in \mathbb{R} \tag{3}$$

Las condiciones de contorno quedan recogidas en la Figura 1.

```
# Body 1
r01 = array([2, 2, 1])
v01 = array([-0.5, 0, 0])

# Body 2
r02 = array([-2,2,-1])
v02 = array([0,-0.5,0])

# Body 3
r03 = array([-2, -2, 1])
v03 = array([0.5, 0, 0])

# Body 4
r04 = array([2, -2, -1])
v04 = array([0, 0.5, 0])
```

Figura 1: Condiciones de contorno *N-Cuerpos*.

1.3. Hito VI

En este hito, se tiene como objetivo evaluar los puntos de Lagrange con su estabilidad y elaborar un esquema de *Runge-Kutta* embebido.

De esta manera, iniciamlente se debe de elaborar un esquema de *Runge-Kutta* embebido. Este esquema tiene un control del paso temporal autónomo que se ajustan según el error obtenido en cada instante. Las ecuaciones de este esquema se muestran a continuación.

$$\begin{cases} U_{n+1} = U_n + \Delta t_n \sum_{i=1}^{s} b_i k_i \\ k_i = F\left(U_n + \Delta t_n \sum_{j=1}^{s} a_{ij} k_j t_n + c_i \Delta t_n \right. \end{cases} \tag{4}$$

Por lo tanto, se procede a calcular los problemas de los puntos de Lagrange.

2. Exposición de Código

2.1. Hito IV

En cuanto al código de este hito, se encuentra como diferencia a los anteriores hitos la creación del código correspondiente a la función de Leap−Frog, la función de oscilación y la función de estabilidad, funciones que posteriormente se utilizarán en el archivo *"Main"*. Dichas funciones quedan mostradas en las siguientes figuras.

```
def Leap_Frog(U1, U2, dt, t, F):

    return U1 + 2*dt*F(U2,t)
```

Figura 2: Función de Leap−Frog.

```
def oscilation(U,t):
    return array([U[1], -U[0]])
```

Figura 3: Función del Oscilador.

```
def stability_regions(x, y, method):
    N = len(x)
    rho =  zeros((N, N),  dtype=float64)

    for i in range(N):
        for j in range(N):
            w = complex(x[i], y[j])
            if method == Leap_Frog:
                r = method(1, 1, 1, 0, lambda u, t: w*u)
            else:
                r = method(1, 0, 1, lambda u, t: w*u)

            rho[i, j] = abs(r)

    return rho
```

Figura 4: Función de estabilidad.

Posteriormente, se añade estas funciones en el código *"Main"*, el cual se muestra en las siguientes Figuras, las cuales son las correspondientes al cálculo final de las funciones de Oscilación y de estabilidad.

```
@numba.njit
def mil_IV_osc(U0, T, dt, schemes, tit_scheme):
    for j in range(size(schemes)):
        scheme = schemes[j]
        for i in range(size(dt)):
            n = int(T/dt[i])
            t = linspace(0, T, n)
            U = Cauchy(oscilation, t, U0, scheme)

            plt.plot(t, U[:,0])
        plt.title(tit_scheme[j])
        plt.xlabel("X")
        plt.ylabel("Y",rotation = 0)
        plt.legend(['dt = 0.1', 'dt = 0.01', 'dt = 0.001'])
        plt.savefig('MILESTONE IV media/' + 'Oscilator with '+ schemes[j].__name__+'.png')
        plt.show()
        plt.close()
```

Figura 5: Función Main de estabilidad.

```
@numba.njit
def milIV_stab(schemes, x, y, ST):
    for z in range(len(schemes)):
        ST[z] = stability_regions(x, y, schemes[z])
        plt.figure()
        plt.title(tit_scheme[z])
        plt.xlabel("Re")
        plt.ylabel("Im",rotation = 0)
        plt.contour(x, y, transpose(ST[z]), linspace(0, 1, 11))
        plt.draw()
        plt.savefig('MILESTONE IV media/' + 'Region '+ schemes[z].__name__+'.png')
        plt.show()
        plt.close()
```

Figura 6: Función Main de estabilidad.

Este hito no supone cierta dificultad a la hora de calcular las regiones de estabilidad ya que se debe de encontrar la manera de aplicar la regla nemotécnica para formar la ecuación de la región de estabilidad. El desarrollo de código para poder obtener estas regiones queda mostrado en la Figura 3.

2.2. Hito V

El código se encuentra estructurado en un archivo principal en el que se encuentra la función *Main*, donde se llaman a todas las funciones utilizadas en este hito. De esta manera, el código se estructura tal y como queda representado en las Figuras 7, 8 y 9.

```
@numba.njit
def main():

    T = 40                  # Time
    dt = 0.01               # Time increase
    n = int(T/dt) + 1
    t = linspace(0,T,n)

    schemes = [Euler, RK4, Euler_Inv, Crank_Nicolson, Leap_Frog]
    tit_scheme = ['Euler', 'Runge Kutta 4', 'Euler Inverso', 'Crank Nicolson', 'Leap Frog']

    U0 = init_state_nbody(Nb,Nc)        # Initial conditions
    U = zeros([len(U0), n])             # Positions vector
```

Figura 7: Código *Main*

```
for i in range(size(schemes)):
    U = Cauchy(Nbody,t,U0,schemes[i])
    fig, twoD = plt.subplots(figsize=(5,5))


    # 2D Representation

    twoD.plot(U[:,0], U[:,2], "b")
    twoD.plot(U[:,6], U[:,8], "r")
    if Nb == 3:
        twoD.plot(U[:,12],U[:,14],"k")
    elif Nb == 4:
        twoD.plot(U[:,12],U[:,14],"k")
        twoD.plot(U[:,18],U[:,20],"purple")
    twoD.set_xlabel("X")
    twoD.set_ylabel("Y",rotation = 0)
    twoD.set_title("Proyección en el plano XY de " + str(Nb) + " cuerpos")
    plt.savefig('MILESTONE V media/' + 'Nbodies 2D with '+ schemes[i].__name__+'.png')
    twoD.grid()
```

Figura 8: Código *Main*

```
    # 3D representation

    fig = plt.figure()
    ax1 = fig.add_subplot(111,projection='3d')
    ax1.plot_wireframe(U[:,0].reshape((-1, 1)), U[:,2].reshape((-1, 1)), U[:,4].reshape((-1, 1)), color= "red", label =
    ax1.plot_wireframe(U[:,6].reshape((-1, 1)), U[:,8].reshape((-1, 1)), U[:,10].reshape((-1, 1)), color= "blue", label
    if Nb == 3:
        ax1.plot_wireframe(U[:,12].reshape((-1, 1)), U[:,14].reshape((-1, 1)), U[:,16].reshape((-1, 1)), color= "black'
    elif Nb == 4:
        ax1.plot_wireframe(U[:,12].reshape((-1, 1)), U[:,14].reshape((-1, 1)), U[:,16].reshape((-1, 1)), color= "black'
        ax1.plot_wireframe(U[:,18].reshape((-1, 1)), U[:,20].reshape((-1, 1)), U[:,22].reshape((-1, 1)), color= "purpl

    plt.title(str(Nb) + ' cuerpos con metodo ' + tit_scheme[i] + ' y dt = ' + str(dt))
    plt.xlabel("X")
    plt.ylabel("Y",rotation = 0)
    plt.grid()
    plt.legend(loc = 'best')
    plt.savefig('MILESTONE V media/' + 'Nbodies 3D with '+ schemes[i].__name__+'.png')
    plt.show()
```

Figura 9: Código *Main*

En este hito, se ha tratado de llevar un orden claro de la estructura de la función principal, donde se pueda entender de manera sencilla a qué corresponde cada tramo del código. Por otra parte, se ha tratado de mejorar en cuanto al tratamiento de información obtenida, mejorando la representación de las gráficas y automatizando el guardado de las figuras en una carpeta, salvando de esta manera una gran cantidad de tiempo a la hora de obtener las imágenes deseadas, ya que no sólo se guardan solas, sino que no llega a suponer un problema cambiar parámetros para cambiar alguna cosa de las figuras, ya que esto se realiza de forma automática y en un corto periodo de tiempo.

Por otra parte, existen dos nuevas funciones para este hito. En estas funciones lo que se realizan son los cálculos de las condiciones iniciales del problema y parte del cálculo del problema a resolver de los *N-Cuerpos*.

```
def Nbody(U, t):

    (Nb,Nc) = (4,3)

    Us = reshape(U,(Nb,Nc,2))
    F  = zeros(len(U))
    Fs = reshape(F,(Nb,Nc,2))

    r  = reshape(Us[:,:,0],(Nb,Nc))     # N-Body locations
    v  = reshape(Us[:,:,1],(Nb,Nc))     # N-Body velocities

    drdt = reshape(Fs[:,:,0], (Nb,Nc))
    dvdt = reshape(Fs[:,:,1], (Nb,Nc))

    dvdt[:,:] = 0

    for i in range(Nb):

        drdt[i,:] = v[i,:]

        for j in range(Nb):

            if j != i:                  # Different bodies atract eachother

                d = r[j,:] - r[i,:]
                dvdt[i,:] += d[:]/(linalg.norm(d)**3)

    return F
```

Figura 10: Cálculo de la función de los *N-Cuerpos*

2.3. Hito VI

El código del Hito VI, al igual que el resto de hitos, presenta un archivo principal con una función *Main*. Por otra parte, este es el hito que presenta una mayor dificultad y desarrollo que los demás debido a todo el desarrollo implicado para los cálculos de los puntos de Lagrange y el proceso de la obtención del esquema de *Runge-Kutta* embebido.

A continuación se muestra el desarrollo del código *Main* del hito.

```
@numba.jit
def main():

    T = 500
    dt = 0.01
    n = int(T/dt) + 1
    t = linspace(0,T,n)
    mu = 3.0039e-7

    U0LP = array([[0.8, 0.6, 0, 0],[0.8, -0.6, 0, 0],[-0.1, 0, 0, 0],[0.1, 0, 0, 0],[1.01, 0, 0, 0]])
    Np = 5      # Lagrange Points
    LPAUX = Lpoints(U0LP, Np)
    LP = zeros([5,2])
    LP[0,:] = LPAUX[3,:] #Reordeno
    LP[1,:] = LPAUX[4,:]
    LP[2,:] = LPAUX[2,:]
    LP[3,:] = LPAUX[0,:]
    LP[4,:] = LPAUX[1,:]

    labelPTot = ['L1','L2','L3','L4','L5'] #ordenados
    ShapeLP = ["<",">","d","^","v"]
    ColorLP = ["yellow","cyan","violet","sienna","lightcoral"]
```

Figura 11: Código *Main*

```
    for i in range(5):
        plt.plot(LP[i,0],LP[i,1],ShapeLP[i],color = ColorLP[i],label=labelPTot[i])
    plt.plot(-mu, 0, 'o', color = "g", label = 'Tierra')
    plt.plot(1-mu, 0, 'o', color = "b", label = 'Luna')
    plt.grid()
    plt.title("Puntos de Lagrange del CR3BP Tierra-Luna")
    plt.legend(loc = 'upper left',bbox_to_anchor=(1., 0.95))
    plt.savefig('MILESTONE VI media/' + 'G1 '+ str(i) +'.png')
    plt.show()



    U0_LP_Sel = zeros(4)
    U0_LP_SelStab = zeros(4)
    eps = 1e-4*random()
```

Figura 12: Código *Main*

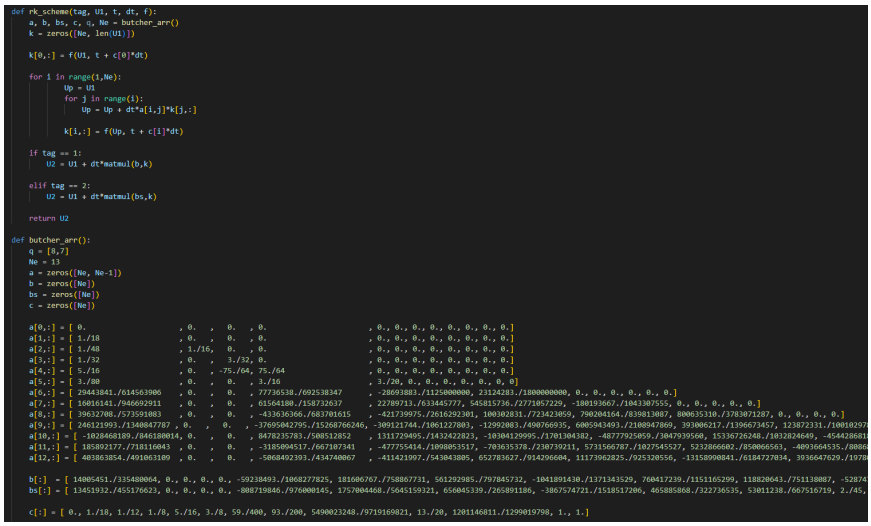
```
for k in range(5):

    sel = k + 1

    if sel == 1:
        labelP = 'L1'
    elif sel == 2:
        labelP = 'L2'
    elif sel == 3:
        labelP = 'L3'
    elif sel == 4:
        labelP = 'L4'
    elif sel == 5:
        labelP = 'L5'

    U0_LP_Sel[0:2] = LP[sel-1,:] + eps
    U0_LP_Sel[2:4] = eps

    U0_LP_SelStab[0:2] = LP[sel-1,:]
    U0_LP_SelStab[2:4] = 0

    Autoval_LP = LP_Stab(U0_LP_SelStab, mu) #estabilidad
    print(around(Autoval_LP.real,8))
```

Figura 13: Código *Main*

```
for j in range (size(1)):

    U_LP = Cauchy(FL, t, U0_LP_Sel, RK_emb) #ordenados tb

    #fig, (ax1, ax2) = plt.subplots(1, 2)
    plt.plot(U_LP[:,0], U_LP[:,1],'-',color = "k", label = 'Orbit')
    plt.plot(-mu, 0, 'o', color = "g", label = 'Tierra')
    plt.plot(1-mu, 0, 'o', color = "b", label = 'Luna')
    for i in range(5):
        plt.plot(LP[i,0],LP[i,1],ShapeLP[i],color = ColorLP[i],label=labelPTot[i])
    plt.xlim(-2,2)
    plt.ylim(-2,2)
    plt.title(f"Simulación CR3BP Tierra-Luna con esquema {RK_emb.__name__}. Órbita en {labelP}. t = {T}s, dt = {dt}. Vista completa" )
    plt.legend(loc = 'upper left',bbox_to_anchor=(1., 0.95))
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.savefig('MILESTONE VI media/' + 'G2 '+ str(k) + str(j) +'.png')
    plt.grid()
    plt.show()

    plt.plot(U_LP[:,0], U_LP[:,1],'-',color = "k", label = "Orbit" )
    plt.plot(LP[sel - 1,0], LP[sel - 1,1] , ShapeLP[sel-1],color = ColorLP[sel-1], label = labelPTot[sel-1])
    plt.title(f"Simulación CR3BP Tierra-Luna con esquema {RK_emb.__name__}. Detalle de órbita en {labelP}. t = {T}s, dt = {dt}" )
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.legend(loc = 'upper right',bbox_to_anchor=(1, 0.5))
    plt.savefig('MILESTONE VI media/' + 'G3 '+ str(k) + str(j) +'.png')
    plt.grid()
    plt.xlim(LP[sel - 1,0]-0.2,LP[sel - 1,0]+0.2)
    plt.ylim(LP[sel - 1,1]-0.2,LP[sel - 1,1]+0.2)
    plt.legend(loc = 'upper left',bbox_to_anchor=(1., 0.95))
    plt.savefig('MILESTONE VI media/' + 'G4 '+ str(k) + str(j) +'.png')
    plt.show()
```

Figura 14: Código *Main*

Por otra parte, se encuentra el código correspondiente al *Runge-Kutta* embebido que se mostrará a continuación.



```
def rk_scheme(tag, U1, t, dt, f):
    a, b, bs, c, q, Ne = butcher_arr()
    k = zeros([Ne, len(U1)])

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

    for i in range(1,Ne):
        Up = U1
        for j in range(i):
            Up = Up + dt*a[i,j]*k[j,:]

        k[i,:] = f(Up, t + c[i]*dt)

    if tag == 1:
        U2 = U1 + dt*matmul(b,k)

    elif tag == 2:
        U2 = U1 + dt*matmul(bs,k)

    return U2

def butcher_arr():
    q = [0,7]
    Ne = 13
    a = zeros([Ne, Ne-1])
    b = zeros([Ne])
    bs = zeros([Ne])
    c = zeros([Ne])

    a[0,:] = [ 0.                  , 0.   ,   0.   , 0.                   , 0., 0., 0., 0., 0., 0., 0.]
    a[1,:] = [ 1./18               , 0.   ,   0.   , 0.                   , 0., 0., 0., 0., 0., 0., 0.]
    a[2,:] = [ 1./48               , 1./16,   0.   , 0.                   , 0., 0., 0., 0., 0., 0., 0.]
    a[3,:] = [ 1./32               , 0.   ,   3./32, 0.                   , 0., 0., 0., 0., 0., 0., 0.]
    a[4,:] = [ 5./16               , 0.   , -75./64, 75./64               , 0., 0., 0., 0., 0., 0., 0.]
    a[5,:] = [ 3./80               , 0.   ,   0.   , 3./16                , 3./20, 0., 0., 0., 0., 0., 0]
    a[6,:] = [ 29443841./614563906 , 0.   ,   0.   , 77736538./692538347  , -28693883./1125000000, 23124283./1800000000, 0., 0., 0., 0., 0.]
    a[7,:] = [ 16016141./946692911 , 0.   ,   0.   , 61564180./158732637  , 22789713./63445777, 545815736./2771057229, -180193667./1043307555, 0., 0., 0., 0.]
    a[8,:] = [ 39632708./573591083 , 0.   ,   0.   , -433636366./683701615 , -421739975./2616292301, 100302831./723423059, 790204164./839813087, 800635310./3783071287, 0., 0., 0., 0.]
    a[9,:] = [ 246121993./1340847787 , 0.   ,   0.   , -37695042795./15268766246, -309121744./1061227803, -12992083./490766935, 6005943493./2108947869, 393006217./1396673457, 123872331./1001029789
    a[10,:] = [ -1028468189./846180014, 0.   ,   0.   , 8478235783./508512852  , 1311729495./1432422823, -10304129995./1701304382, -48777925059./3047939560, 15336726248./1032824649, -45442868181
    a[11,:] = [ 185892177./718116043 , 0.   ,   0.   , -3185094517./667107341  , -477755414./1098053517, -703635378./230739211, 5731566787./1027545527, 5232866602./850066563, -4093664535./808688
    a[12,:] = [ 403863854./491063109 , 0.   ,   0.   , -5068492393./434740067  , -411421997./543043805, 652783627./914206604, 11173962825./925320556, -13158990841./6184727034, 3936647629./1978

    b[:] = [ 14005451./335480064, 0., 0., 0., -59238493./1068277825, 181606767./758867731, 561292985./797845732, -1041891430./1371343529, 760417239./1151165299, 118820643./751138087, -528747
    bs[:] = [ 13451932./455176623, 0., 0., 0., -808719846./976000145, 1757004468./5645159321, 656045339./265891186, -3867574721./1518517206, 465885868./322736535, 53011238./667516719, 2./45,

    c[:] = [ 0., 1./18, 1./12, 1./8, 5./16, 3./8, 59./400, 93./200, 5490023248./9719169821, 13./20, 1201146811./1299019798, 1., 1.]
```

Figura 15: Código *Runge-Kutta*

```
@numba.njit
def step_size(dU, tol, q, h):
    normT = linalg.norm(dU)

    if normT > tol:
        ss = h*(tol/normT)**(1/(q+1))
    else:
        ss = h

    return ss
```

Figura 16: Código *Runge-Kutta*

```
@numba.njit
def Jac(F, U):
    N = size(U)
    J= zeros([N,N])
    t = 1e-10

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

@numba.njit
def LP_Stab( U0, mu ):

    A = Jac(CR3BP, U0)
    values, vectors = linalg.eig(A)

    return values
```

Figura 17: Código *Runge-Kutta*

Como se puede comprobar, es un código de mayor complejidad que es más costoso conseguir realizar correctamente.

Finalmente, se encuentra el código correspondiente al cálculo de los puntos de Lagrange.

```
@numba.njit
def CR3BP(U):
    mu = 3.0039e-7

    x = U[0]; y = U[1]
    vx = U[2]; vy = U[3]

    r1 = sqrt( (x+mu)**2 + y**2 )
    r2 = sqrt( (x-1+mu)**2 + y**2 )

    dxdt, dydt = vx, vy

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

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

@numba.njit
def FL(U,t):                # Wrapped de la función CR3BP
    return CR3BP(U)

@numba.njit
def Lpoints(U0, Np):
    LP = zeros([Np,2])

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

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

    return LP

@numba.njit
def Jac(F, U):
    N = size(U)
    J= zeros([N,N])
    t = 1e-10
```

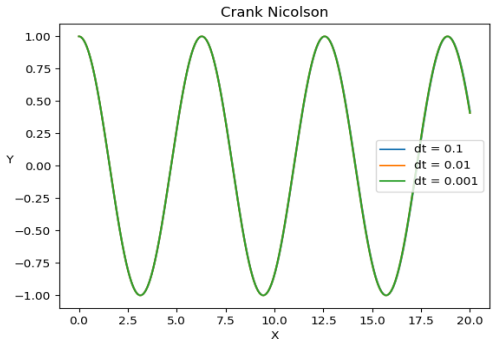
Figura 18: Código *Lagrange*

En la misma línea, este tramo de código también resulta complejo y puede dar problemas a la hora de escribirlo. Por otra parte, cabe mencionar, que tal y como se puede observar en la Figura 18, se ha escrito un decorator de la función *C3RBP*.

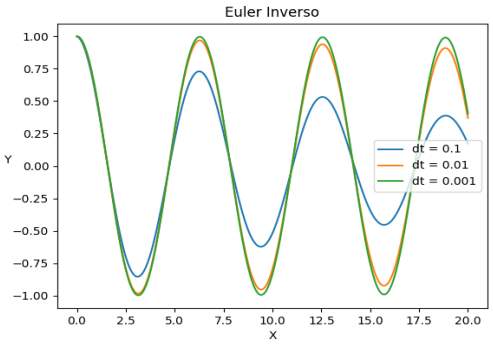
3. Resultados

3.1. Hito IV

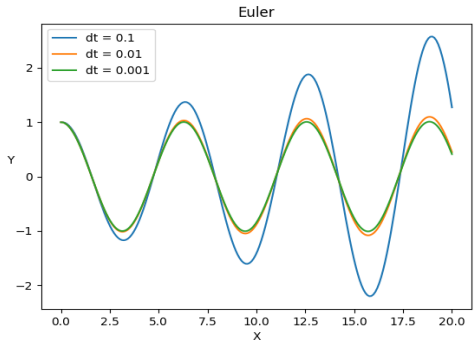
Los resultados de la función del oscilador calculada con los diferentes esquemas propuestos quedan mostrados a continuación.



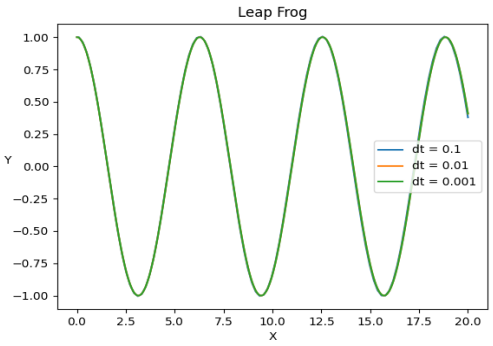
(Crank Nicolson)



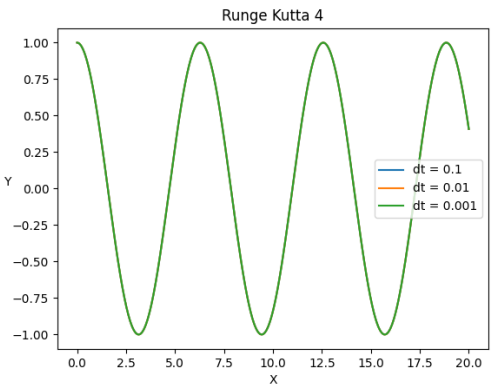
(Euler Inverso)



(Euler)



(Leap Frog)

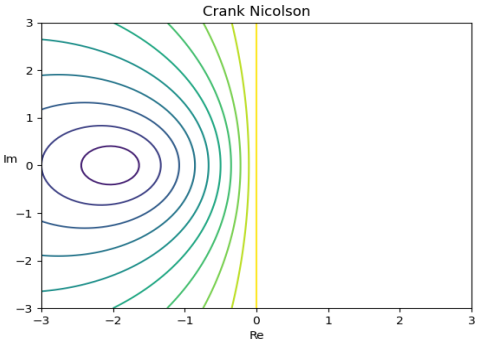


(Runge−Kutta de 4^o orden)

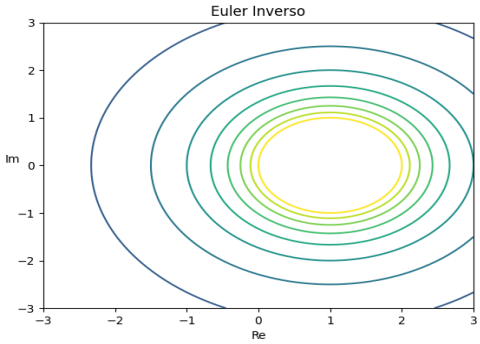
Figura 19: Oscilador con los diferentes métodos.

Como se puede apreciar, los esquemas de Euler presentan un error más elevado que el resto, el cual se ve notablemente incrementado conforme se avanza en la coordenada x.

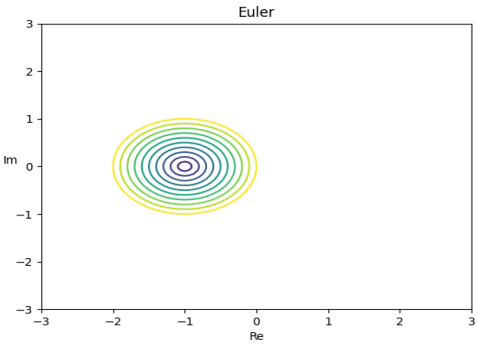
Por otra parte, se encuentran los resultados de las regiones de estabilidad, las cuales se muestran en las siguientes figuras.



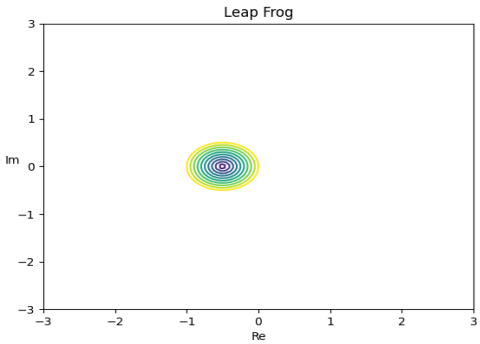
(Crank Nicolson)



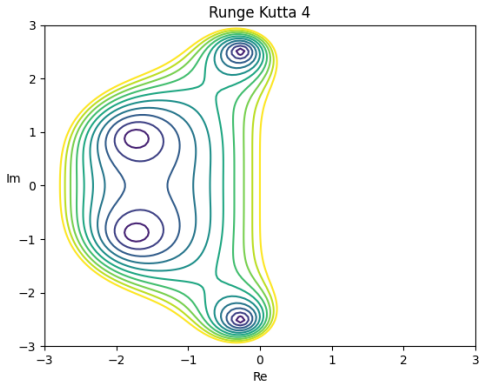
(Euler Inverso)



(Euler)



(Leap Frog)



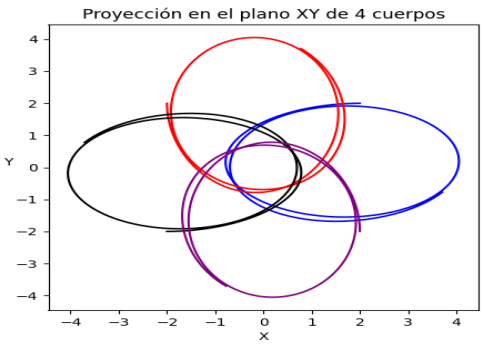
(Runge−Kutta de 4^o orden)

Figura 20: Regiones de estabilidad.

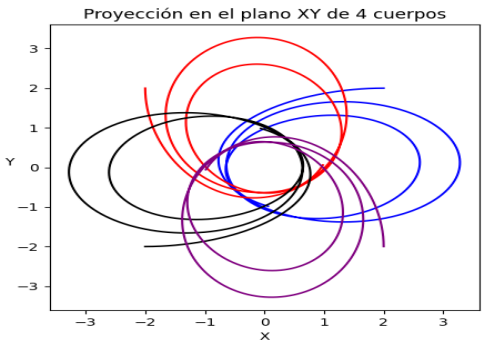
Se puede observar que finalmente se han obtenido de forma correcta las regiones de estabilidad.

3.2. Hito V

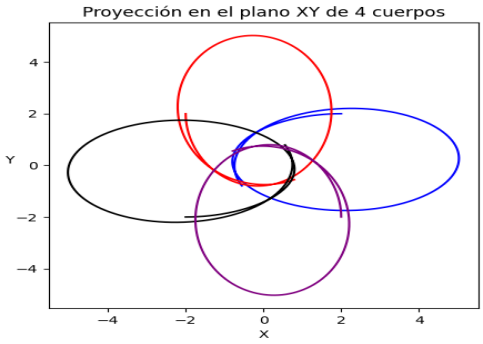
A continuación se mostrarán los resultados obtenidos para el problema de los *N-Cuerpos*.



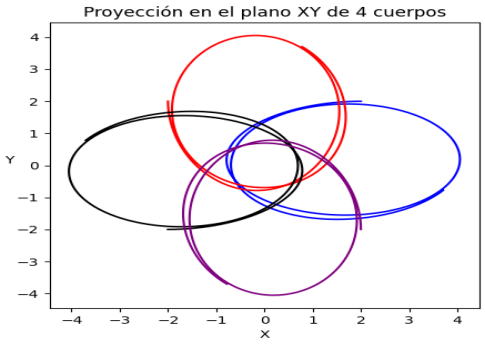
(Crank Nicolson)



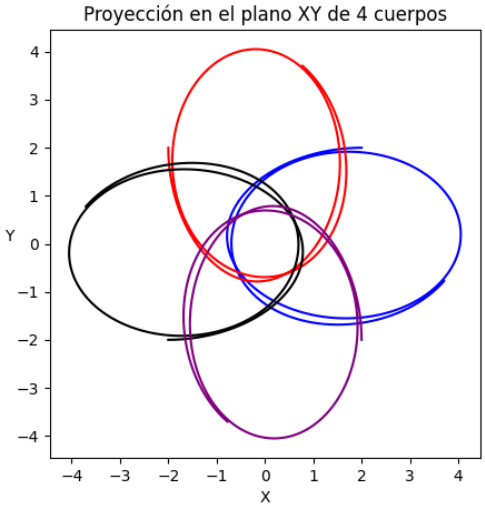
(Euler Inverso)



(Euler)

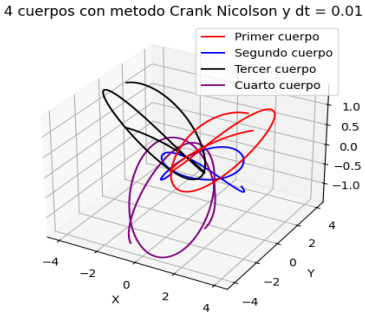


(Leap Frog)

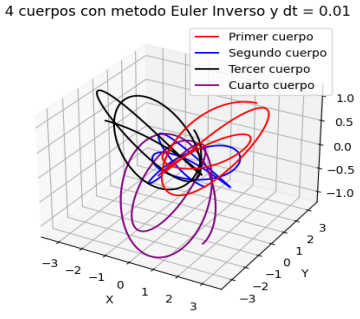


(Runge−Kutta de 4^o orden)

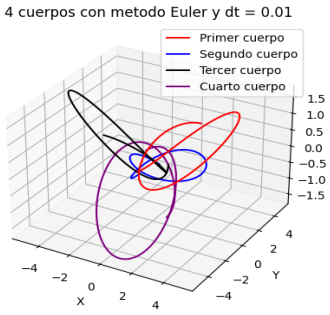
Figura 21: Representación 2D de los *N-Cuerpos*.



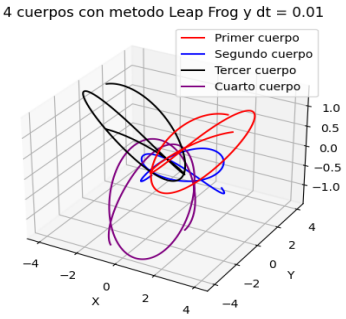
(Crank Nicolson)



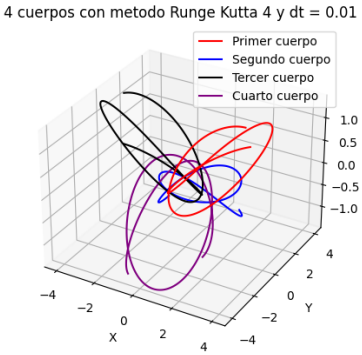
(Euler Inverso)



(Euler)



(Leap Frog)



(Runge−Kutta de 4^o orden)

Figura 22: Representación 3D de los *N-Cuerpos*.

Como se puede observar, se comprueba que el se ha logrado calcular correctamente la solución de los *N-Cuerpos*. En cuanto a los esquemas, se puede observar que el esquema de Euler Inverso es el que más error genera de todos los mostrados.

3.3. Hito VI

A continuación se mostrarán los resultados obtenidos en este hito, mostrando inicialmente los puntos de Lagrange Tierra-Luna en la Figura 23.

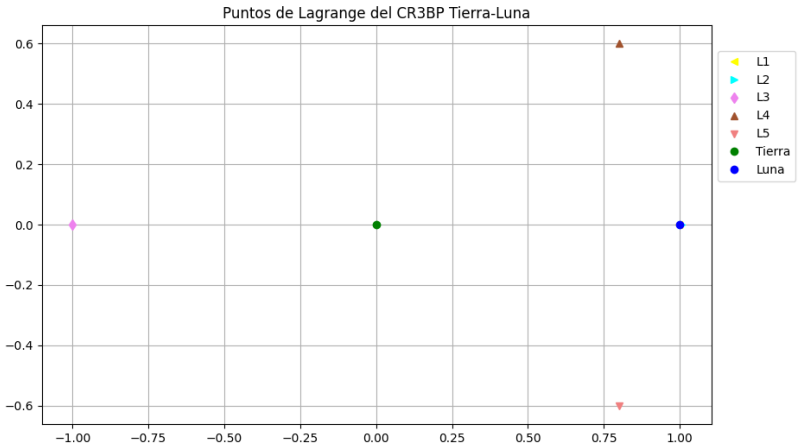
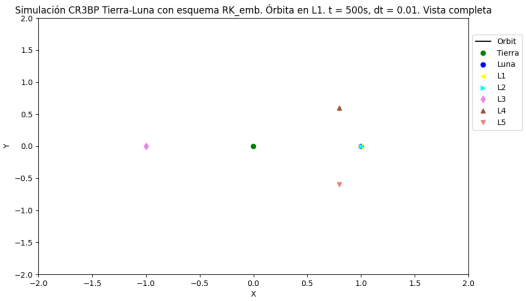
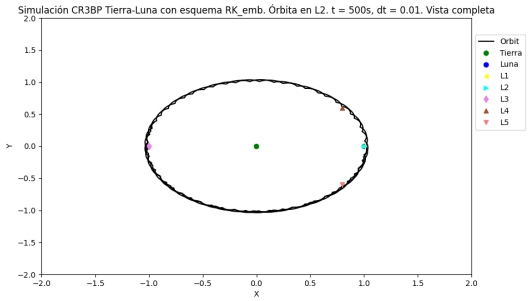


Figura 23: Puntos de Lagrange

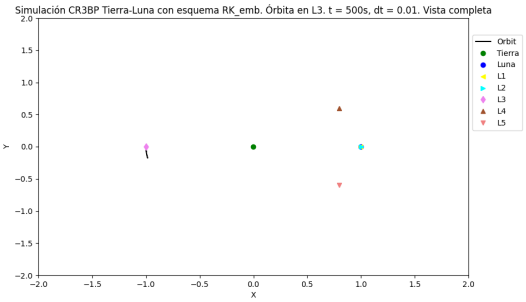
Seguidamente, se pasa a mostrar los resultados desde una vista alejada de los resultados en cada uno de los puntos de Lagrange y posteriormente se mostrarán desde una vista más cercana para observarlos en mayor detalle.



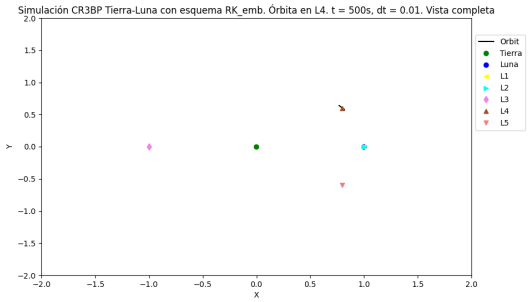
(L1)



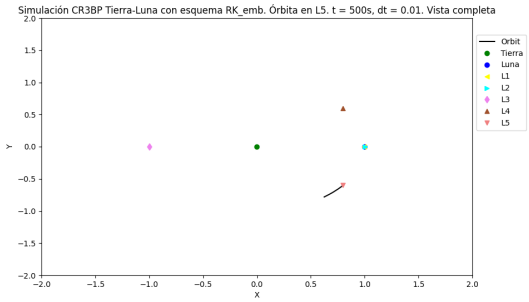
(L2)



(L3)

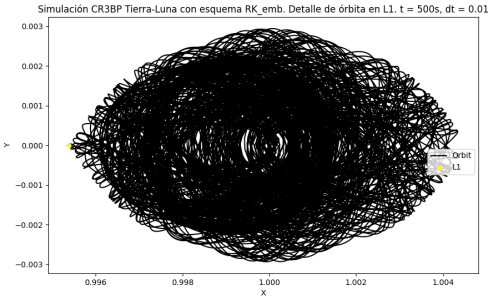


(L4)

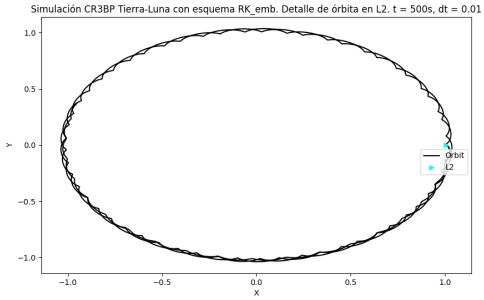


(L5)

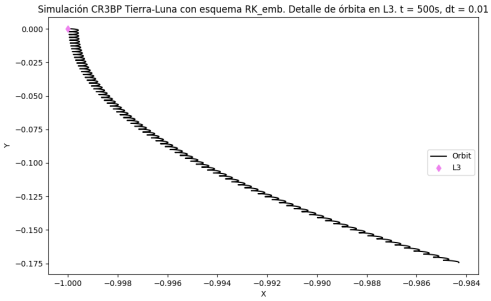
Figura 24: Vista alejada de los resultados obtenidos en los puntos de Lagrange.



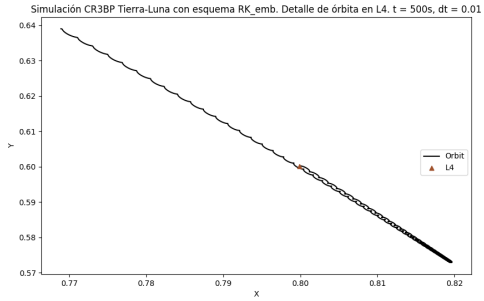
(L1)



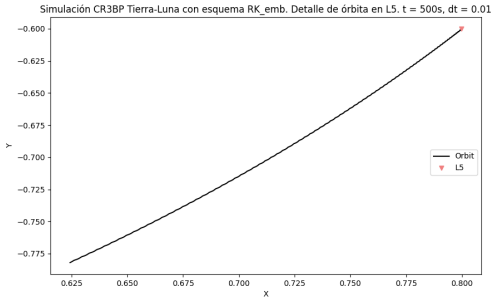
(L2)



(L3)

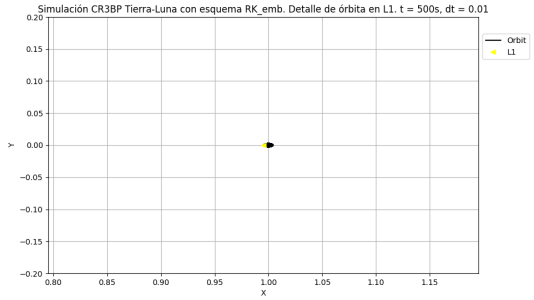


(L4)

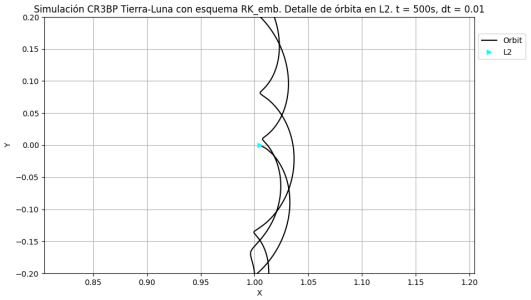


(L5)

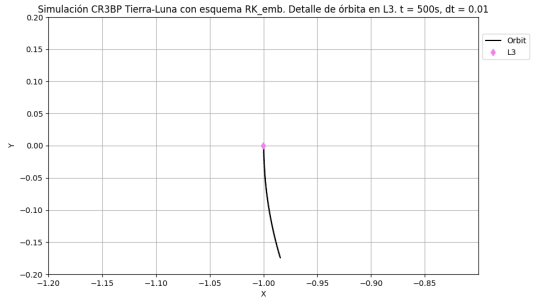
Figura 25: Vista cercana de los resultados obtenidos en los puntos de Lagrange.



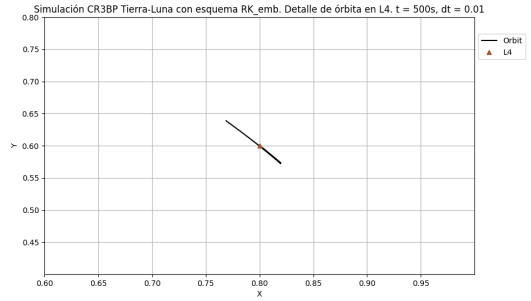
(L1)



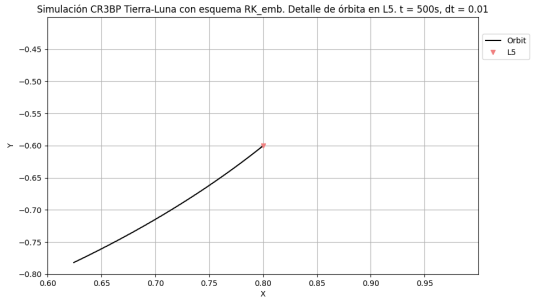
(L2)



(L3)



(L4)



(L5)

Figura 26: Vista cercana de los resultados obtenidos en los puntos de Lagrange.

4. Conclusiones

En cuanto al Hito IV, comentar que podría existir alguna manera más sencilla para obtener las ecuaciones de las regiones de estabilidad sin necesitar realizar reglas nemotécnicas para su obtención. Por otra parte, se ha podido abarcar un mayor conocimiento acerca de nuevos esquemas temporales y la programación implicada.

En cuanto a los Hitos V y VI, han resultado más complejos y costosos de hacer funcionar. Podría ser positivo tener alguna biblioteca que te proporcione las matrices con las condiciones iniciales ya dadas y matrices de *Runge-Kutta* embebido ya dadas también entre otras posibles ayudas.

Cabe mencionar, que con el desarrollo de estos hitos, se ha aprendido acerca de la programación, utilizando la metodología Top-Dow, haciendo un desarrollo ordenado de los archivos de código y tratando de hacerlo lo más intuitivo y claro posible.