Trabajo de fin de grado: Movimiento de asteroides en las Zanjas de Kirkwood

Grado en Física, Universidad
Autónoma de Madrid

Mayo 2019

Roberto Mazo Fernández

Tutor: David Martín y Marero

Resumen

El objetivo principal de este trabajo es estudiar el comportamiento de los asteroides cerca de las Zanjas de Kirkwood y tratar de diseñar un método para determinar la anchura de dichas zanjas. Para ello hemos desarrollado un código en lenguaje Python con el que realizaremos experimentos de simulación del movimiento de los asteroides. Además, hemos comparado distintos métodos de integración numérica y finalmente hemos empleado el método de Velocidad Verlet. Asimismo, hemos estudiado el efecto que tiene Marte y Saturno en la creación de las zanjas.

1. Introducción

En torno a 1850 el astrónomo Daniel Kirkwood graficó el número de asteroides en función de su distancia al Sol y el resultado fue el siguiente:

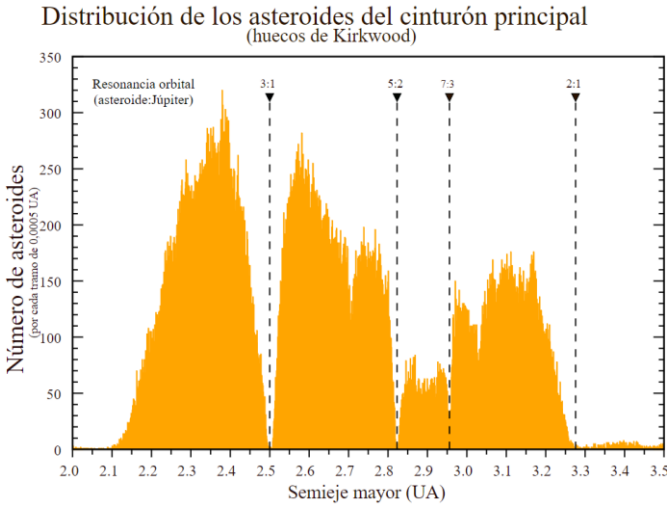

Figura 1: Numero de asteroides en función de la distancia al Sol. Donde no hay asteroides, es lo que se conoce como Zanjas de Kirkwood. Adaptado de [5]

Como vemos en la *Figura 1,* existen unas regiones en las cuales la densidad de asteroides se ve notablemente reducida respecto a la media del cinturón, estas regiones son conocidas como Zanjas de Kirkwood. En estas regiones los asteroides se encuentran en resonancia con Júpiter en una proporción determinada, lo que podría da lugar a un vaciamiento de asteroides. Cuando decimos que un asteroide se encuentra en resonancia 5/2 significa que por cada dos vueltas que da Júpiter el asteroide da 5. Ese caso coincide con una región de vaciamiento cuyo primer origen se debe a la atracción resonante que Júpiter tiene sobre el asteroide en el punto de máxima aproximación. Se cree que la acumulación de estos impulsos

en resonancia a lo largo del tiempo hace que el asteroide entre en un régimen caótico tal que cualquier perturbación futura lo expulse de su órbita [6].

Para estimar la anchura de las zanjas calcularemos las posiciones de los asteroides a lo largo del tiempo y haremos una gráfica comparando su posición final con la inicial. Si se encuentra en una zanja o en sus proximidades, veremos como efectivamente su órbita se desvía considerablemente.

Los métodos de integración numérica que compararemos en este proyecto son los métodos de: Euler, Euler-Cromer, Velocidad-Verlet y Runge-Kutta 4.

Finalmente estudiaremos la influencia de Marte y Saturno en la creación de dichas zanjas. Para ello veremos el efecto que tienen sobre las zanjas mas próximas a ellos, es decir, la que se encuentra en resonancia 3/1 para el caso de Marte y la que se encuentra en resonancia 2/1 para el caso de Saturno.

2. Teoría general

2.1 Modelo de Titius-Bode

La fórmula de Titius-Bode es una explicación fenomenológica que nos permite estimar la distancia de un planeta al Sol mediante una sucesión:

$$a = \frac{n+4}{10} \tag{1}$$

Donde a representa el semieje mayor de la órbita y n_i es dos veces el numero anterior, $n_i = 2n_{(i-1)}$, con n=0 para i=0 y n=3 para i=1.

Surge puramente de la observación y predice la existencia de un planeta entre Marte y Júpiter, a una distancia de unas 2,80 unidades astronómicas.

Planet	*a* (actual) (AU)	Titus-Bode (AU)
Mercury	0.39	0.40
Venus	0.72	0.70
Earth	1.00	1.00
Mars	1.52	1.60
???	–	2.80
Jupiter	5.20	5.20
Saturn	9.54	10.00
Uranus	19.19	19.60
Neptune	30.06	38.80
Pluto	39.53	77.20

Tabla 1: Comparación del semieje mayor actual de las órbitas de los planetas, a, con las predicciones de la fórmula de Titius-Bode.[1]

Sin embargo, en torno a 1800 un "planeta" fue descubierto a esta distancia, y desafortunadamente para el modelo, muchos objetos más fueron descubiertos aproximadamente en torno a este radio. Es lo que hoy denominamos cinturón de asteroides. Como vemos esta fórmula no es buena para estimar la distancia de Neptuno y Plutón [1].

2.2 Problema de dos cuerpos

Empezaremos tratando el movimiento de la Tierra alrededor del Sol. De acuerdo con la segunda ley de Newton, la fuerza gravitatoria entre dos cuerpos es:

$$\mathrm{F_g} = -\frac{\mathrm{GM_sM_T}}{\mathrm{r^2}} \tag{2}$$

Donde "G" es la constante de la gravitación universal, "M_S" es la masa del Sol, "M_T" es la masa de la Tierra y "r" la distancia entre ambos cuerpos. Debido a que la masa del Sol es mucho más elevada que el del resto de planetas del Sistema Solar, asumimos que permanece fijo y en el origen de coordenadas.

Para trabajar en coordenadas cartesianas podemos que descomponer la fuerza, ya que va en la dirección radial.

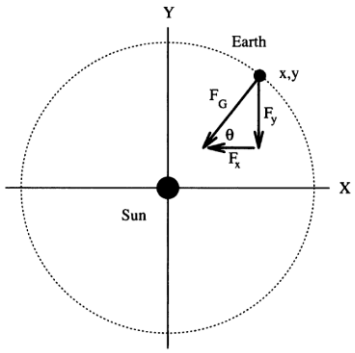

Figura 2: Órbita de la Tierra alrededor del Sol. Este permanece fijo en el (0,0). Adaptado de [1]

Teniendo en cuenta que lo que nos interesa es la expresión de la fuerza en función de la posición y que $\mathrm{x} = \mathrm{r} \cdot \cos(\theta)$ e $\mathrm{y} = \mathrm{r} \cdot \sin(\theta)$ obtenemos:

$$F_{Gx} = -\frac{G \cdot M_S \cdot M_T}{r_{ST}^2} \cdot \cos(\theta) = -\frac{G \cdot M_S \cdot M_T}{r_{ST}^3} \cdot \mathrm{x} \tag{3}$$

$$F_{Gy} = -\frac{G \cdot M_S \cdot M_T}{r_{ST}^2} \cdot \sin(\theta) = -\frac{G \cdot M_S \cdot M_T}{r_{ST}^3} \cdot \mathrm{y} \tag{4}$$

El sistema de unidades que emplearemos será: unidades astronómicas (UA) para la distancias y años para el tiempo, ya que son las unidades más adecuadas para un tratamiento computacional. Para completar el sistema de unidades es necesario definir las unidades de masa. Para ello definiremos la siguiente constante suponiendo que la órbita entre el Sol y la Tierra es circular y es válida la aproximación $F_{centrípeta}=F_{gravitatoria}$

$$G \cdot M_s = v^2 \cdot r = 4\,\pi^2\,AU^3/año^2$$

Donde hemos tenido en cuenta que la velocidad de la Tierra es $v=2\pi \cdot r\ /(1año) = 2\pi$ UA/año.

En nuestro caso vamos a trabajar con órbitas elípticas:

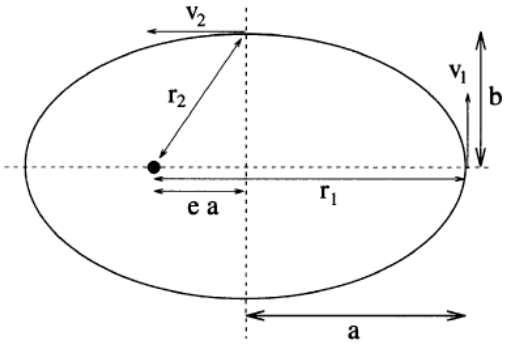

Figura 3: Órbita elíptica de un planeta alrededor del Sol.
Adaptado de [1]

Donde "a" es el semieje mayor, "b" es el semieje menor, "r_1" y "r_2" distancias, "v_1" y "v_2" las velocidades en esos puntos y "e" es la excentricidad.

Como sabemos, bajo la acción de una fuerza conservativa la energía mecánica y el momento angular se conservan.

$$\frac{1}{2}\,M_p\,v_1^2 - \frac{GM_sM_p}{r_1} = -\frac{GM_sM_p}{r_2} + \frac{1}{2}\,M_p\,v_2^2 \qquad (5)$$

$$M_p v_1 r_1 = M_p v_2 r_2 \qquad (6)$$

Esto hace que en el afelio la velocidad sea mínima y en el perihelio máxima:

$$\boldsymbol{V_{max} = \sqrt{\frac{GM(1+e)\cdot(1+\frac{M_{jup}}{Msun})}{a(1-e)}}} \qquad (7) \qquad\qquad \boldsymbol{V_{min} = \sqrt{\frac{GM(1-e)\cdot(1+\frac{M_{jup}}{Msun})}{a(1+e)}}} \qquad (8)$$

2.3 Problema de tres cuerpos

Mientras que el problema de los dos cuerpos tiene solución mediante métodos de integración numérica, el problema de tres cuerpos no tiene solución general por dicho método y en algunos casos su solución puede ser caótica. Esto significa que pequeñas variaciones en las condiciones iniciales pueden llevar a destinos totalmente diferentes.

En esta situación de caos podemos resolverlo como un problema de dos cuerpos y considerar que el tercero perturba la posición de los otros dos [8].

Para explicar el problema de los tres cuerpos haremos las siguientes aproximaciones:

1. El Sol debido a su masa permanecerá quieto
2. La fuerza que ejerce el asteroide sobre Júpiter es despreciable

Si añadimos la fuerza que ejerce Júpiter sobre el asteroide a las ecuaciones (1) y (2) obtenemos:

$$F_{Gx} = -\frac{G \cdot M_S \cdot M_a}{r_{aS}^2} \cdot \mathrm{x} - \frac{G \cdot M_J \cdot M_a}{{r_{aJ}}^3} \cdot (x_a - x_J) \tag{9}$$

$$F_{Gy} = -\frac{G \cdot M_S \cdot M_a}{r_{aS}^2} \cdot \mathrm{y} - \frac{G \cdot M_J \cdot M_a}{{r_{aJ}}^3} \cdot (y_a - y_J) \tag{10}$$

Donde los subíndices "a" corresponden al asteroide, "S" al Sol y "J" al Júpiter.

Para seguir empleando las unidades explicadas anteriormente escribimos:

$$GM_J = GM_S \frac{M_J}{M_S} = 4\ \pi^2\ \frac{M_J}{M_S}\ \mathrm{UA}^3/\mathrm{año}^2$$

2.4 Interacción con Marte

Uno podría pensar cual es la influencia que tiene Marte sobre los asteroides que se encuentran ubicados entre él y Júpiter. Es por ello por lo que añadiremos a nuestra simulación este planeta y estudiaremos el efecto que tiene sobre los asteroides.

Dado que la masa de Marte es bastante inferior a la masa del Sol y a la de Júpiter, asumiremos que el efecto que tiene Marte sobre Júpiter es despreciable. Sin embargo, los asteroides estarán sometidos a la fuerza gravitatoria del Sol, Marte y Júpiter. Las ecuaciones serán por tanto:

$$F_{Gx} = -\frac{G \cdot M_S \cdot M_a}{r_{aS}^2} \cdot \mathrm{x} - \frac{G \cdot M_J \cdot M_a}{{r_{aJ}}^3} \cdot (x_a - x_J) - \frac{G \cdot M_M \cdot M_a}{{r_{aM}}^3} \cdot (x_a - x_M) \quad (11)$$

$$F_{Gy} = -\frac{G \cdot M_S \cdot M_a}{r_{aS}^2} \cdot \mathrm{y} - \frac{G \cdot M_J \cdot M_a}{{r_{aJ}}^3} \cdot (y_a - y_J) - \frac{G \cdot M_M \cdot M_a}{{r_{aM}}^3} \cdot (y_a - y_M) \quad (12)$$

Donde las unidades empleadas son las explicadas en el 2.2.

2.5 Interacción con Saturno

Aunque Saturno se encuentra bastante lejos, al ser el segundo planeta mas masivo del sistema solar, procederemos a la simulación y estudio de su efecto. Análogamente al caso de Marte, hemos calculado la trayectoria de los asteroides en su presencia.

2.6 Cálculo del período

Dada una órbita elíptica alrededor de un objeto masivo central el periodo orbital T, lo calculamos con la tercera ley de Kepler.

$$T = 2\pi \cdot \sqrt{\frac{a^3}{GM}} \quad (13)$$

Donde "a" es la distancia del semieje mayor, "G" es la constante de gravitación universal y "M" la masa del objeto más masivo.

2.7 Métodos de integración.

De acuerdo con la segunda ley de Newton tenemos que resolver las siguientes ecuaciones diferenciales ordinarias.

$$m\frac{d^2x}{dt^2} = -\frac{GMm}{r^3}x \quad (14) \qquad m\frac{d^2y}{dt^2} = -\frac{GMm}{r^3}y \quad (15)$$

Donde "m" y "M" son las masas de los cuerpos, "r" es distancia entre ellos, "G" la constante de gravitación universal, "x" e "y" las posiciones del cuerpo que gira (*ver Figura 2*) y la derivada segunda es la aceleración [1].

Podemos transformar estas dos ecuaciones diferenciales de segundo orden en cuatro de primer orden. Para ello emplearemos que la derivada de la posición es la velocidad y la derivada de la velocidad es la aceleración:

$$\frac{dx}{dt} = f(t, v) \quad (16) \qquad \frac{dv}{dt} = g(t, x) \quad (17)$$

Análogo para "y". Existen varios métodos para resolver numéricamente este sistema de ecuaciones:

2.7.1 Método de Euler

Es uno de los métodos de integración numérica más sencillos que nos permite resolver ecuaciones diferenciales ordinarias a partir de unas condiciones iniciales. El método toma pequeños pasos de tiempo durante los cuales se aproxima a la pendiente de la función para que sea constante. Esta aproximación se basa en el desarrollo de Taylor de la posición hasta el orden 1:

$$x(t + \Delta t) = x(t) + \frac{\partial x}{\partial t}\Delta t + \ldots \tag{18}$$

Este método no es útil para resolver movimientos cíclicos ya que, al ser una aproximación por Taylor hasta el orden uno, desprecia términos que se manifestarán como un aumento de la energía al cabo de varios ciclos [2].

Las ecuaciones tomarían la forma:

$$x_{n+1} = x_n + f(t_n, v_n)\Delta t \tag{19}$$

$$v_{n+1} = v_n + g(t_n, x_n)\Delta t \tag{20}$$

Donde $\Delta t = t_{n+1} - t_n$, es lo que denominaremos "tamaños del paso" cuanto menor sea, más precisión obtendremos. Todos los métodos de integración discutidos emplearán el mismo paso de tiempo.

2.7.2 Método de Euler-Cromer

Este método es una variación del método de Euler que pretende corregir la energía que introduce al sistema en los movientes periódicos. Para las mismas ecuaciones dadas en Ec.(1)y Ec.(2)

$$v_{n+1} = v_n + g(t_n, x_n)\Delta t \tag{21}$$

$$x_{n+1} = x_n + f(t_n, v_{n+1})\Delta t \tag{22}$$

A diferencia de Euler, este método usa la v_{n+1} en el cálculo de la posición x_{n+1} en vez de v_n, es decir, asumimos que la velocidad con la que se mueve en el intervalo es con la del paso siguiente.

2.7.3 Método de Verlet

Este método nos permite calcular la posición sin necesidad de calcular la velocidad. Para ello empleamos el desarrollo en serie de Taylor hasta el orden 3.

$$x(t+\Delta t) = x(t) + \frac{\partial x}{\partial t}\Delta t + \frac{1}{2}\frac{\partial^2 x}{\partial t^2}\Delta t^2 + \frac{1}{6}\frac{\partial^3 x}{\partial t^3}\Delta t^3 \tag{23}$$

$$x(t-\Delta t) = x(t) - \frac{\partial x}{\partial t}\Delta t + \frac{1}{2}\frac{\partial^2 x}{\partial t^2}\Delta t^2 - \frac{1}{6}\frac{\partial^3 x}{\partial t^3}\Delta t^3 \tag{24}$$

Combinando las Ecs. (7) y (8) obtenemos:

$$x(t+\Delta t) = 2x(t) - x(t-\Delta t) + \frac{\partial^2 x}{\partial t^2}\Delta t^2 \tag{25}$$

Sin embargo, la velocidad es útil para calcular otras propiedades como por ejemplo la energía del sistema. Usando el método de las diferencias contadas:

$$v(t) = \frac{x(t+\Delta t) - x(t-\Delta t)}{2\Delta t} \tag{26}$$

Para tener una buena aproximación de la posición Δt tiene que ser muy pequeño. Como vemos para hallar la velocidad habría que dividir entre Δt por lo que nuestro calculo seria menos exacto.

2.7.4 Método de Velocidad Verlet

Es una variación del método de Verlet que soluciona la perdida de precisión al calcular la velocidad. Para ello incorporamos la velocidad a la integración. Las ecuaciones toman la forma:

$$x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 \tag{27}$$

$$v(t+\Delta t) = v(t) + \frac{a(t)+a(t+\Delta t)}{2}\Delta t \tag{28}$$

Es el método más usado en astronomía por su precisión y debido a que no añade energía al sistema [2].

2.7.5 Método de Runge-Kutta 4

El método de Runge-Kutta no es sólo un único método, sino una importante familia de métodos iterativos, para aproximar las soluciones de ecuaciones diferenciales ordinarias. Uno de los más usados el de 4 orden, que es el que usaremos. Es un método mas complejo

que los anteriores. Para hallar la posición y la velocidad hay que calcular los siguientes coeficientes primero

$$x_{n+1} = x_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Donde:

$$k_1 = f(t_n, v_n) \tag{30}$$

$$k_2 = f(t_n + \frac{\Delta t}{2}, v_n + \frac{\Delta t}{2}k_1) \tag{31}$$

$$k_3 = f(t_n + \frac{\Delta t}{2}, v_n + \frac{\Delta t}{2}k_2) \tag{32}$$

$$k_4 = f(t_n + \Delta t, v_n + \Delta t\, k_3) \tag{33}$$

La solución de RK4 para x_{n+1} está determinada por el valor actual x_n y el producto del tamaño de paso Δt por promedio ponderado de la pendiente en el paso de tiempo. Para calcular la velocidad se haría de manera similar [3]. De hecho, el método de Euler es un método de Runge-Kutta de orden 1.

3. Metodología

El lenguaje de programación escogido para la realización del trabajo ha sido Python. Hemos empleado las conocidas librerías: *"numpy", "math", "pylab", "timeit" y "matplotlib.pylot".* Asimismo hemos creado varias funciones donde definimos la gravedad y energía mecánica dependiendo del número de cuerpos. Además, hemos creado funciones encargadas de ejecutar los métodos de integración numérica explicados anteriormente. Finalmente para caracterizar la anchura de las zanjas hemos creado un bucle que vaya desde las 2.2 UA hasta las 3.45 UA y estudie la posición inicial del asteroide y su posición final para obtener su desviación.

3.1 Elección del método de integración

La elección del método es muy importante y por eso merece especial atención. Para ello hemos creado una seria de funciones que nos calculen las órbitas de los cuerpos celestes empleando los métodos de integración ya explicados. Los parámetros de entrada y salida de estas funciones los podemos ver en la *Figura 4.*

```
Funcion que nos vale para calcular la posicion y la velocidad de un objeto acelerado en funcion del tiempo. Metodo de Verlet
    Input:
         a=     aceleracion que sufre el cuerpo
         r0=    vector que contenga posicion inicial
         v0=    vector que contenga la velocidad inicial
         dt=    paso de tiempo
         tend=  tiempo final

    Output:
        x_vector, y_vector, z_vector =      Vectores que contienen las posiciones del cuerpo en funcion del tiempo
        vx_vector, vy_vector, vz_vector =   Vectores que contienen las velocidades del cuerpo en funcion del tiempo
        t_vector =                          Vector que contiene los tiempos

Ejemplo de funcion aceleracion
 def a(x,y,z):
    ax= -G*M*x/(x**2+y**2+z**2)**(3/2)
    ay= -G*M*y/(x**2+y**2+z**2)**(3/2)
    az= -G*M*z/(x**2+y**2+z**2)**(3/2)
    return [ax,ay,az]
```

Figura 4: Parámetros de entrada y salida de la función "Verlet", la cual aplica el método de Velocidad Verlet explicado anteriormente.

Los parámetros observados en la *Figura 4* son los mismos para el resto de los métodos.

Empezaré explicando los bucles principales de dichas funciones:

```
for i in range (0,n):

    [ax, ay, az] = a(x, y,z)

    x_next = x + vx * dt
    y_next = y + vy * dt
    z_next = z + vz * dt

    vx_next = vx + ax * dt
    vy_next = vy + ay * dt
    vz_next = vz + az * dt
```

Figura 5: bucle principal de la función encargada de aplicar el método de Euler

```
for i in range (0,n):

    [ax, ay, az] = a(x, y, z)

    vx_next = vx + ax * dt
    vy_next = vy + ay * dt
    vz_next = vz + az * dt

    x_next = x + vx_next * dt
    y_next = y + vy_next * dt
    z_next = z + vz_next * dt
```

Figura 6: bucle principal de la función encargada de aplicar el método de Euler Cromer. Obsérvese que a diferencia de la Figura 5 en este caso calculamos primero la velocidad y luego la posición

Si nos fijamos en las *Figuras 5 y 6*, la principal diferencia reside en que en el método de Euler primero calculamos las posiciones y luego las velocidades y en el de Euler Cromer calculamos primero las velocidades para poder calcular las posiciones, ya que como se explicó en la sección 2.5.2, asumimos que el cuerpo se mueve con la velocidad del paso siguiente.

```
for i in range (0,n):

    [ax, ay, az] = a(x, y, z)
    x_next= x + vx * dt + 0.5 * ax * dt ** 2
    y_next= y + vy * dt + 0.5 * ay * dt ** 2
    z_next= z + vz * dt + 0.5 * az * dt ** 2

    [ax_next, ay_next, az_next] = a(x_next, y_next, z_next)
    vx_next = vx + 0.5 * ( ax + ax_next ) * dt
    vy_next = vy + 0.5 * ( ay + ay_next ) * dt
    vz_next = vz + 0.5 * ( az + az_next ) * dt
```

Figura 7: bucle principal de la función encargada de aplicar el método de Velocidad Verlet

En la *Figura* 7 el bucle calcula primero las aceleraciones del tiempo en el que estamos y posteriormente las aceleraciones del tiempo siguiente, ya que como vimos en la *ecuación 12*, para hallar la velocidad es necesario hacer una media de las aceleraciones.

El método de Runge-Kutta 4 al ser bastante mas largo puede encontrarse en *la sección 1 del anexo* junto con los métodos mencionados anteriormente.

Para compararlos hemos calculando el tiempo que tarda cada método en calcular las órbitas empleando el comando *timer()* de la librería *timeit*.

```
start1=timer()
t_jup_E, x_jup_E, y_jup_E, z_jup_E, vx_jup_E, vy_jup_E, vz_jup_E = Euler (grav_sun , r0_jup, v0_jup, deltat,tend)
end1=timer()
start2=timer()
t_jup_EC, x_jup_EC, y_jup_EC, z_jup_EC, vx_jup_EC, vy_jup_EC, vz_jup_EC = Euler_Cromer (grav_sun , r0_jup, v0_jup, deltat,tend)
end2=timer()
start3=timer()
t_jup_V, x_jup_V, y_jup_V, z_jup_V, vx_jup_V, vy_jup_V, vz_jup_V = Verlet (grav_sun , r0_jup, v0_jup, deltat,tend)
end3=timer()
start4=timer()
t_jup_RK4, x_jup_RK4, y_jup_RK4, z_jup_RK4, vx_jup_RK4, vy_jup_RK4, vz_jup_RK4 =  RungeK_4 (grav_sun , r0_jup, v0_jup, deltat,tend)
end4=timer()
```

Figura 8: comparación del tiempo en calcular las órbitas de Júpiter por los distintos métodos de integración

En la *Figura 8* llamamos a las funciones y comparamos el tiempo que tardan en realizar los cálculos *(Sección 3 del anexo).* Para ver la precisión de dichos métodos hemos visto cuando mide el afelio en cada órbita para ello hemos calculado los afelios con cada método y luego hemos hecho la media, así como su desviación cuadrática media para obtener la precisión.

```
if fabs(x_jup_V[i]-x_jup_V[0])<tol:
    T1_jup=T2_jup
    T2_jup = deltat * i
    T_jup1 =T2_jup-T1_jup
    T_jup.append(T_jup1)
    x_afel.append(fabs(x_jup_V[i]-x_jup_V[0]))
```

Figura 9: Ejemplo del cálculo del afelio de Júpiter y el periodo con el método de Verlet

Para calcular el afelio con los distintos métodos hemos hecho un bucle en el cual si la diferencia de la posición en el eje "x" y la posición inicial (afelio) es menor que una tolerancia, significa que el Júpiter está en su afelio *(Figura 9).* Para más detalle *ver sección 3.2 anexo.*

Para ver la variación en energía hemos creado una función la cual nos calcule la energía mecánica *(Figura 10)*

```
for i in range(n): #No importan las unidades, queremos ver si varia
    v[i] =sqrt( vx[i] ** 2 + vy[i] ** 2 + vz[i] ** 2)
    Energia_cinet[i]= 0.5*v[i] ** 2 * Mplaneta1
    r[i] =sqrt( x[i] ** 2 + y[i] ** 2 + z[i] ** 2)
    Energia_grav[i] = - 4*pi**2*Mplaneta1/ r[i]
    Energia_mec[i]= Energia_grav[i] + Energia_cinet[i]
```

Figura 10: bucle principal de la función encargada de calcular la energía mecánica para el cuerpo celeste.

Las unidades empleadas para el cálculo de la energía no están en el Sistema Internacional, pero no son relevantes ya que lo que queremos ver es una variación. Por lo tanto nos da lo mismo el tener que multiplicar por un factor para ponerlas en el SI (S*ección 2.2 del anexo).*

Posteriormente, hemos representado la evolución de la órbita de Júpiter a lo largo de 200 años con los distintos métodos de integración, así como la energía mecánica.

Asimismo, hemos calculado el periodo de Júpiter y lo hemos comparado con el bibliográfico. Para ello hemos hecho un promedio con los periodos obtenidos y su desviación cuadrática media. El período de Júpiter que hemos obtenido es de : $T_{experimental}$=11.8878 ± 0.0005 años
Como vemos se asemeja bastante al bibliográfico: $T_{bilbiografico}$=11.872 ± 0.003 años

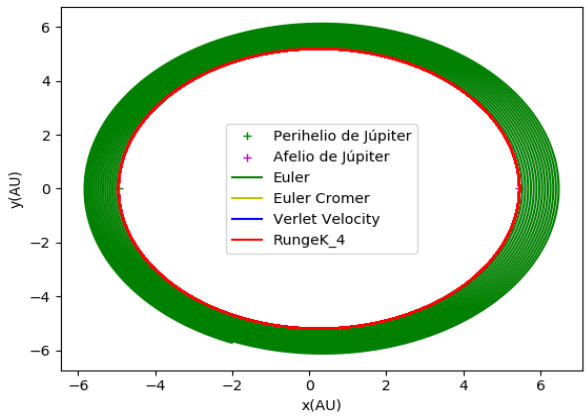

Figura 11: Comparación de la órbita de Júpiter al cabo de 200 años para los distintos métodos de integración.

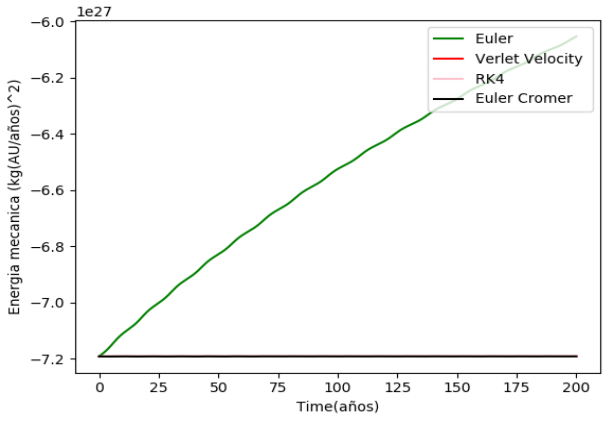

Figura 12: Comparación de la energía mecánica de Júpiter en su órbita con los distintos métodos de integración.

En la *Figura 11* para que quede claro que la órbita de Jupiter es eliptica y no circular hemos representado el afelio y perihelio de Júpiter. Los valores exactos son: 5.45 UA y 4.95 UA respectivamente [3].

Como era de esperar con el método de Euler aumenta la energía del sistema *(Figura 12).* Este aumento de energía tiene como consecuencia que su órbita describa una espiral de radio

cada vez mayor, *Figura 11*. Sin embargo, nos queda la duda de que método emplear de los otros tres. En la *Tabla 2* hemos calculado el tiempo que tardan en realizar el calculo de la trayectoria los distintos métodos y su desviacion en el afelio .

Método	Tiempo computacional (s)	Desviación del afelio (UA)
Euler	0.693	No procede
Euler Cromer	0.825	4e-07
Velocidad Verlet	1.129	2e-07
Rungekutta4	1.521	2e-07

Tabla 2: Comparación entre los métodos de integración al realizar el cálculo de la trayectoria y su desviación del afelio al cabo de T=200años

Como vemos en la *Tabla 2*, el método que más tarda en calcularlo es el de Rungekutta4, que tarda el doble que Euler-Cromer prácticamente. Además, como se discutió en el apartado 2.1.4, al fin y al cabo, es una aproximación por Taylor al cuarto orden, por lo que al cabo de mucho tiempo este error se manifestaría. Las consecuencias serían similares a las de Euler, el cual es una aproximación por Taylor a primer orden.

Aun así hemos representado la evolución de la órbita con estos métodos y hemos visto cual se desvía más del valor observado.

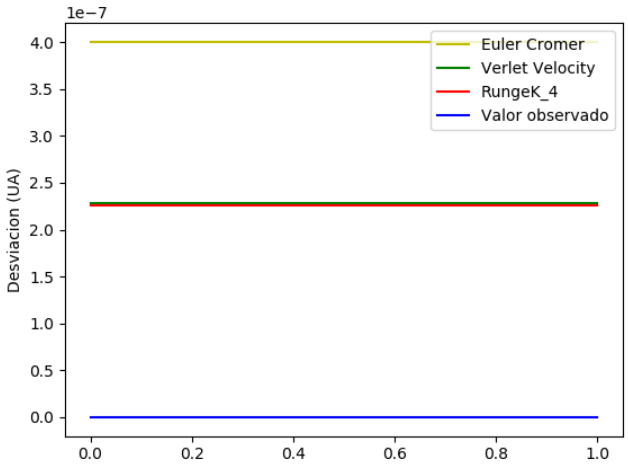

Figura 13: Comparación de los métodos al pasar cerca del afelio de Júpiter.

Si nos fijamos los que más se acercan al afelio son RK4 y Verlet, se desvían 31km y 32km respectivamente, mientas que con Euler Cromer tenemos una desviación de unos 60km.

Como el radio de Júpiter es de 69.911 km estaría dentro de nuestro error y no sería del todo correcto emplear este resultado.

Para solucionar este problema y que la desviación sea superior al radio del cuerpo celeste, hemos escogido un asteroide cuyos parámetros orbitales se conocen con bastante exactitud, Juno [10]. Los códigos se asemejan a los del caso de Júpiter *(sección 4 del anexo),* con ligeros cambios como el cálculo del afelio de Juno *(Figura 14)*, dado que su órbita precesa.

```
if r_afel[i]>r_afel[i-1] and r_afel[i]>r_afel[i+1]:
    rmaxV.append(r_afel[i])
```

Figura 14: Calculo del afelio de Juno

Como el afelio es el punto más alejado del Sol donde se encuentra un cuerpo. Para identificarlo basta con decir que si la distancia calculada es mayor que la anterior y menor que la siguiente nos encontramos en el mínimo *(Figura 14).*

Los resultados para la órbita han sido los siguientes:

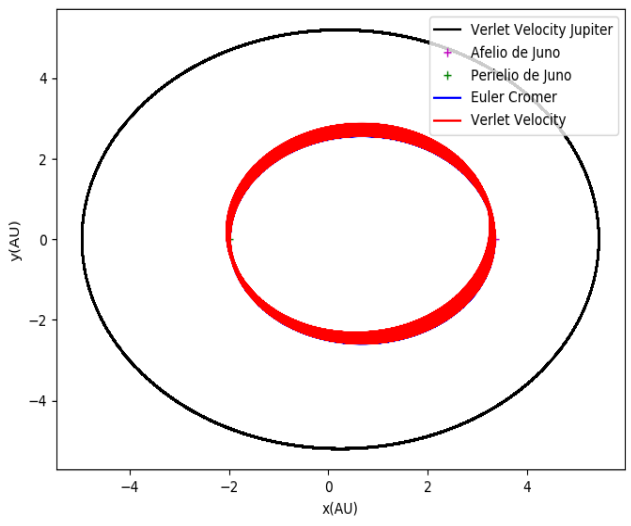

Figura 15: Trayectoria de Júpiter y del asteroide Juno con Verlet y Euler Cromer con T=2000 años terrestres

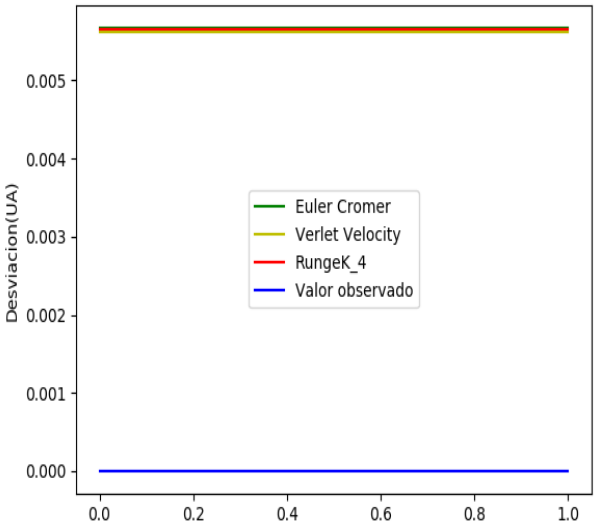

Figura 16: Desviación de los distintos métodos al pasar cerca del afelio de Juno con T=2000 años terrestres

Método	Tiempo computacional(s)	Desviación del afelio (UA)
Euler Cromer	32.421	0.006
Velocidad Verlet	37.508	0.006
Rungekutta4	65.548	0.006

Tabla 3: Comparación de los métodos de integración al realizar el cálculo de la trayectoria de Júpiter y Juno, así como la desviación del afelio de Juno al cabo de 2000 años terrestres

Observando la *Figura 16,* vemos que las órbitas se acercan más al afelio real de Juno [10] son las realizadas con Velocidad Verlet y RK4. Como vemos en la *Tabla 3*, se ha realizado la desviación cuadrática media para los tres métodos y el resultado ha sido el mismo. Como el radio de Juno es de 116.95km, ahora nuestra desviación seria mayor que el radio del asteroide, por lo que la comparación es apropiada.

Concluimos que, aunque el método de Euler Comer tarda menos, preferimos usar el de Velocidad Verlet porque es más exacto (como habíamos visto también en el caso de Júpiter) y la diferencia de tiempo no es tan grande como en el caso de RK4.

Además, al tratarse ahora de un problema de 3 cuerpos hemos vuelto a calcular la energía para ver si se conserva.

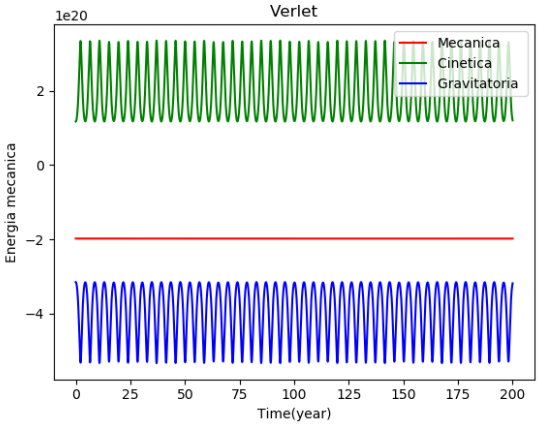

Figura 17: Representación de las energías de Juno con el método de Verlet..

Como vemos en la *Figura 17,* la energía cinética se compensa con la energía potencial gravitatoria, de tal manera que cuando una es máxima la otra es mínima, dando así un valor para la energía mecánica constante a lo largo de una órbita elíptica (Sección 2.2 anexo).

Concluimos que el método de Velocidad Verlet conserva la energía y concuerda con la teoría ya que en una órbita elíptica, descrita sobre un plano, se conserva el momento angular, lo que da lugar a que la energía mecánica sea constante.

3.2 Estudio Zanjas de Kirkwood

```
for radius in np.arange(2.2,3.45,0.001):

    V_i = sqrt(4 * pi ** 2 / (radius))

    Init_pos = [radius, 0, 0]
    V_init = [0, V_i, 0]

    t_ast, x_ast, y_ast, z_ast, vx_ast, vy_ast, vz_ast =\
    Verlet2 (grav_sun_Jupiter , Init_pos , V_init, deltat,tend,x_jup , y_jup , z_jup )

    x_ast = np.array(x_ast,float)
    y_ast = np.array(y_ast,float)
    r_ast = ( x_ast**2 + y_ast**2 ) ** (1/2)
    desviacion=abs(max(r_ast)-min(r_ast))
```

Figura 18: Bucle que nos permite simular las órbitas de los asteroides desde las 2.2 UA hasta las 3.45 UA y calcular su desviación. Para ver código completo ver Sección 6 del anexo

Para localizar las Zanjas de Kirkwood lo que hemos hecho un bucle que nos calcule las órbitas de los asteroides desde las 2.2 UA hasta las 3.45 UA, ya que es la zona a estudiar.

Como vemos en la *Figura 18* el bucle nos genera un asteroide a una distancia determinada, llamada "*radius*", esta distancia irá variando en intervalos de 0.001 simulando cada vez un nuevo asteroide. Una vez tenemos los parámetros iniciales del mismo, calculamos su órbita empleando el método de Velocidad Verlet. Lo que nos interesa es ver lo que se desvía un asteroide respecto de su órbita inicial y por eso hemos creado un vector con la distancia a la que se encuentra el asteroide para cada tiempo. Finalmente la desviación se calcula haciendo la diferencia entre el máximo del vector de distancias y el mínimo. Posteriormente representaremos los resultados con *Origin [11]* y realizaremos una gráfica de la desviación de un asteroide en función de su posición inicial. Calculando la anchura de la lorentziana estimaremos el ancho de las Zanjas de Kirkwood

4. Resultados y discusiones

4.1 Zanjas de Kirkwood

Una vez que hemos elegido el método que vamos a usar hemos realizado una comparación

asumiendo primero la órbita de Júpiter circular y luego añadiendo elementos mas realistas a la simulación tales como que la órbita sea elíptica, exista interacción de los asteroides con Marte y con Saturno.

4.1.1 Órbita Júpiter circular vs órbita elíptica

Para poder estudiar el comportamiento en las Zanjas de Kirkwood y sus proximidades hemos realizado un estudio de la desviación que sufren los asteroides respecto de su posición inicial en función del tiempo.

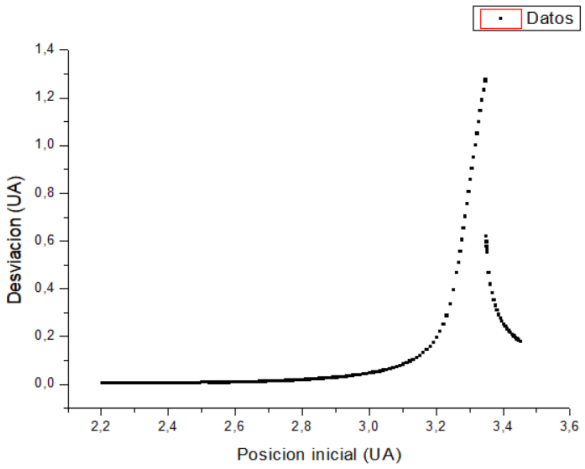

Figura 19: Desviación de un asteroide en función de su posición inicial respecto al Sol al cabo de 1000 años asumiendo órbita de Júpiter circular

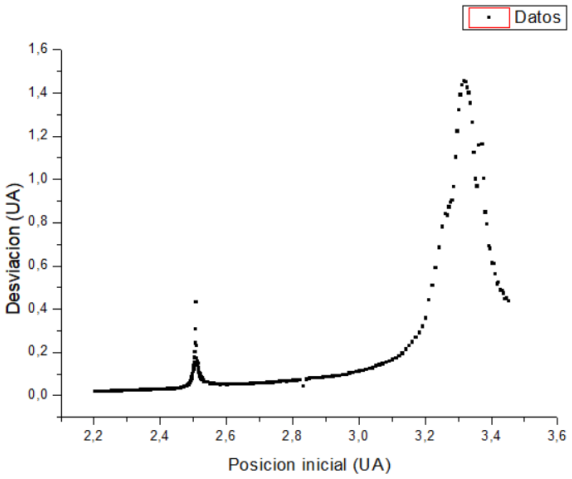

Figura 20: Desviación de un asteroide en función de su posición inicial respecto al Sol al cabo de 1000 años asumiendo órbita de Júpiter elíptica

En la *Figura 19* la órbita que describe Júpiter es circular, esto hace que solo observamos una Zanja de Kirkwood, esta es la que se encuentra en resonancia con Júpiter 2:1. Observando con mas detalle dicha grafica se puede observar que los asteroides sufren una pequeña desviación en torno a las 2.50 UA, zona que corresponde a la Zanja que ese encuentra en resonancia 3:1.

Sin embargo, en la *Figura 20* hemos asumido que la órbita de Júpiter es elíptica, como sucede en la realidad. Esto hace que se vean con más claridad las Zanjas mencionadas anteriormente. Además, empezamos a ver un comportamiento extraño en torno a las 2.82 UA. Esto no es casualidad dado que en esa zona los asteroides se encuentran en resonancia con Júpiter 5:2 como vimos en la *Figura 1*. El origen de este comportamiento escapa de los objetivos de la investigación de este trabajo, debido a la duración requerida.

Para comprobar que los asteroides próximos a esas zonas ven modificadas sus órbitas hemos representado varios casos.

4.1.1.1 Asteroides en resonancia 2:1 con Júpiter.

Hemos escogido tres asteroides que se encuentran próximos a la zanja de Kirkwood en resonancia 2:1 con Júpiter. Los parámetros iniciales de dichos asteroides han sido:

Cuerpo celeste	Distancia (AU)	Velocidad inicial (AU/año)
Asteroide 1	3.000	3.628
Asteroide 2	3.276	3.471
Asteroide 3	3.600	3.312

Tabla 4: Posiciones iniciales y velocidades usadas para tres asteroides hipotéticos en la vecindad de la zanja de Kirkwood con resonancia 2/1

Para comenzar nuestra simulación hemos asumido que la órbita de Júpiter es circular y posteriormente la hemos hecho elíptica para comparar el comportamiento de los asteroides.

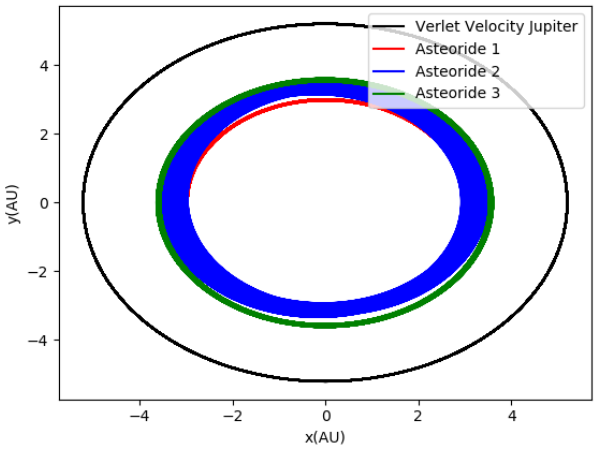

Figura 21: representación de los 3 asteroides en resonancia 2:1 con Júpiter de la Tabla 4. T=1000 años terrestres. Órbita de Júpiter circular

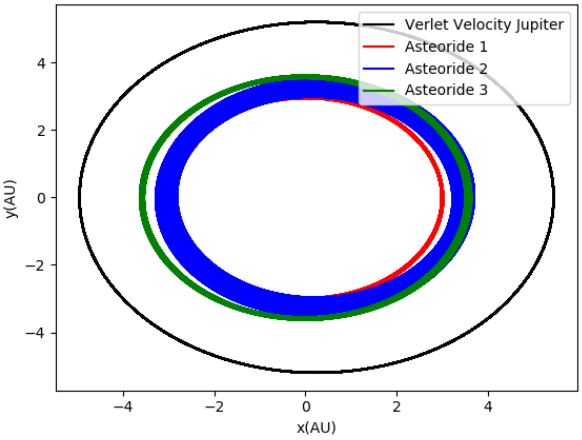

Figura 22: representación de los 3 asteroides en resonancia 2:1 con Júpiter de la Tabla 4. T=1000 años terrestres. Órbita Júpiter elíptica

En las *Figuras 21 y 22* nos llama la atención el asteroide 2 (con una posición inicial en 3.276UA) ya que exhibe un comportamiento distinto al de los otros asteroides.

Este resultado es coherente con los resultados obtenido anteriormente en las *Figura 19 y 20,* ya que dicho asteroide está en una Zanja de Kirkwood para el caso en el que la órbita de Júpiter es circular y para el caso elíptico. Este se encuentra en resonancia 2:1 con Júpiter.

Además se ha representado la distancia de los asteroides al Sol en función del tiempo

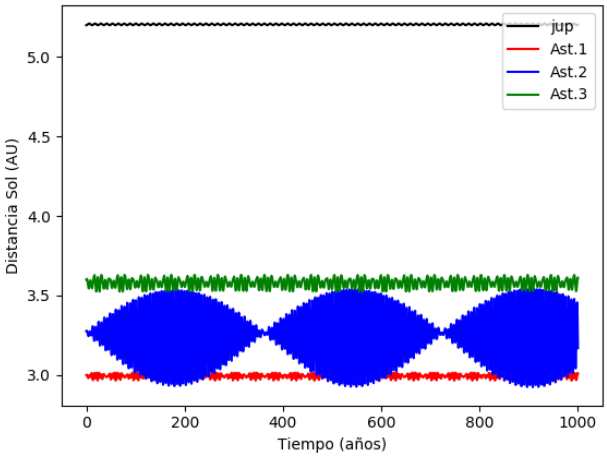

Figura 23: distancia al Sol de Júpiter y los 3 asteroides Órbita de Júpiter circular. T=1000 años

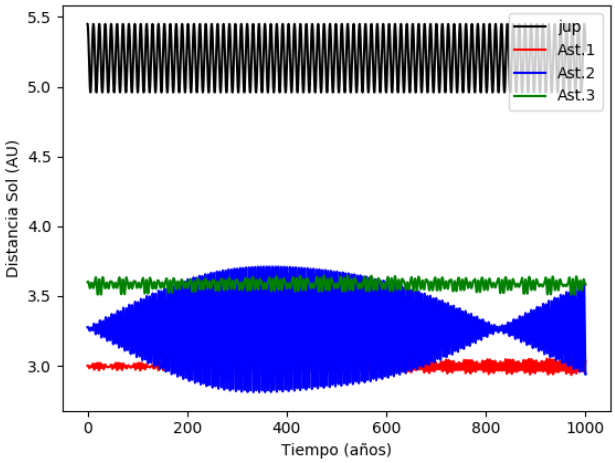

Figura 24: distancia al Sol de Júpiter y los 3 asteroides. Órbita de Júpiter elíptica. T=1000 años

En la *Figura 24*, la distancia de Júpiter al Sol oscila ya que unas veces esta en el afelio y otras en el perihelio, a diferencia de la *Figura 23* donde vemos que la distancia de Júpiter al Sol es constante. Además vemos que el asteroide 2 describe un movimiento oscilatorio con una amplitud y un periodo considerable, siendo mayor en el caso elíptico. Esto se debe a que al describir Júpiter una trayectoria elíptica, el asteroide tarda más tiempo en volver a la posición inicial, es decir a repetir posiciones, que si fuera una trayectoria circular.

4.1.1.2 Asteroides en resonancia 3:1 con Júpiter

La otra zona que observamos fue en torno a las 2.50 UA, por eso hemos realizado el proceso análogo para 3 asteroides en la vecindad a esta zona *(Seccion 5 anexo).*

Cuerpo celeste	Distancia (AU)	Velocidad inicial (AU/año)
Asteroide 1	2.4000	4.056
Asteroide 2	2.3000	4.143
Asteroide 3	2.5000	3.974

Tabla 5: Posiciones iniciales y velocidades usadas para tres asteroides hipotéticos en la vecindad de la Zanja de Kirkwood con resonancia 3:1

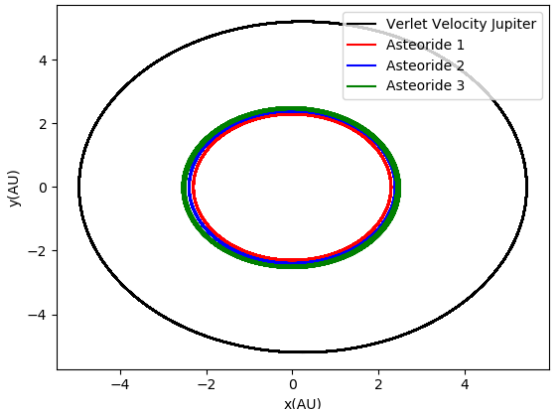

Figura 25: representación de los 3 asteroides en resonancia 3:1 con Júpiter de la Tabla 3. T=1000 años terrestres. Órbita Júpiter circular

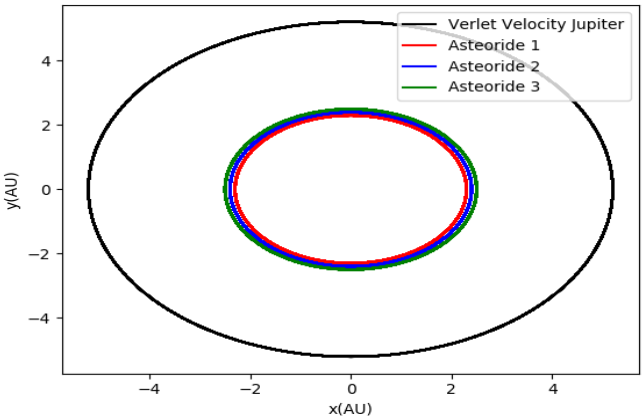

Figura 26: representación de los 3 asteroides en resonancia 3:1 con Júpiter de la Tabla 3. T=1000 años terrestres. Órbita Júpiter elíptica

Para observar con más detalle la trayectoria de los asteroides las hemos ampliado.

En la *Figura 27* observamos que para el caso circular la órbita de los asteroides permanece prácticamente invariante, resultado que concuerda con el obtenido en la *Figura 19* ya que solo pudimos observar una Zanja de Kirkwood en torno a las 3.30 UA.

Si embargo en el caso elíptico,*Figura 28* , el asteroide 3 describe un comportamiento distinto al de los otros dos, esto se debe a que como vimos en la *Figura 20* en torno al 2.50 existe una Zanja de Kirkwood ya que en esa posición los asteroides están en resonancia 3:1 con Júpiter.

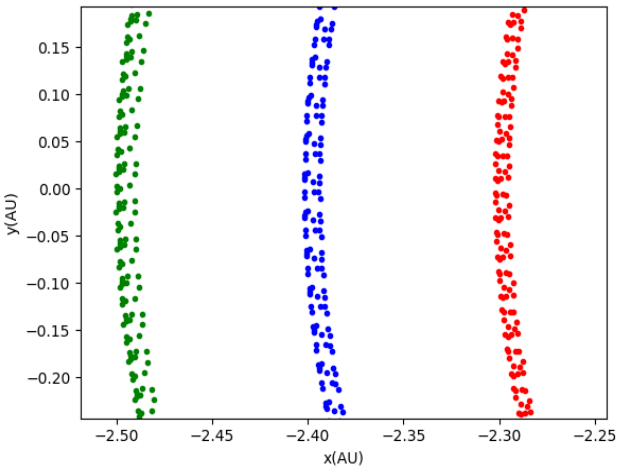

Figura 27: Ampliación de la figura 25. Hemos representado algunas órbitas con puntos una mayor claridad. Caso circular

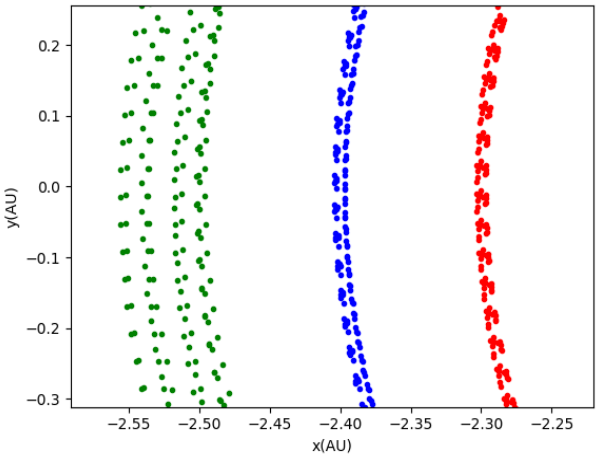

Figura 28: Ampliación de la Figura 26. Hemos representado algunas órbitas con puntos para una mayor claridad. Caso elíptico

Analogamente al caso anterior, hemos representado las distancia de dichos asteroides al Sol.

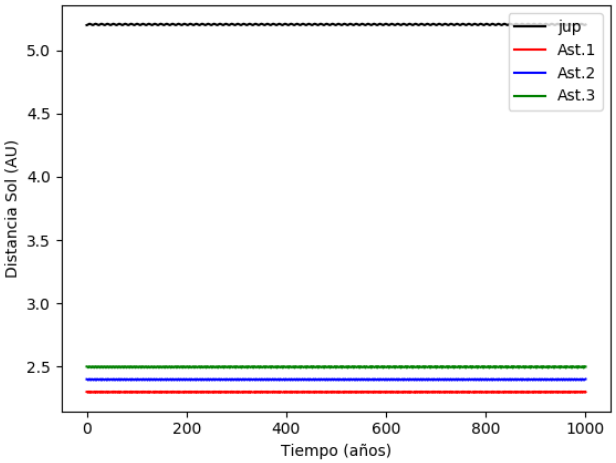

Figura 29: distancia al Sol de Júpiter y los 3 asteroides
Órbita de Júpiter circular

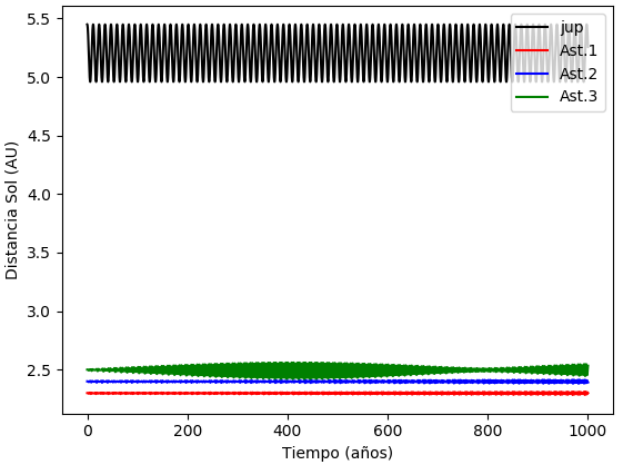

Figura 30: distancia al Sol de Júpiter y los 3 asteroides.
Órbita de Júpiter elíptica

Nuevamente verificamos que los resultados obtenidos anteriormente son coherentes, dado que en el caso circular las órbitas de todos los asteroides permanecen prácticamente invariantes *(Figura 29).* Como cabe esperar en el caso elíptico (*Figura 30*) la distancia al Sol del asteroide 3 (con una posición inicial en 2.50UA) describe un comportamiento oscilatorio, a diferencia de los otros asteroides. Esto se debe a que en esa zona el asteroide se encuentra en resonancia con Júpiter 2:1.

Hemos comprobado que es cierto que existen unas determinadas zonas en las cuales los asteroides ven modificadas sus órbitas a lo largo del tiempo, estas zonas son las Zanjas de Kirkwood, en ellas los asteroides están en resonancia con Júpiter.

Además hemos visto que a medida que nuestro sistema se vuelve más realista el efecto se nota más.

4.1.2 Caracterización de la achura de la zanja

4.1.2.1 Zona próxima a 2.50 UA

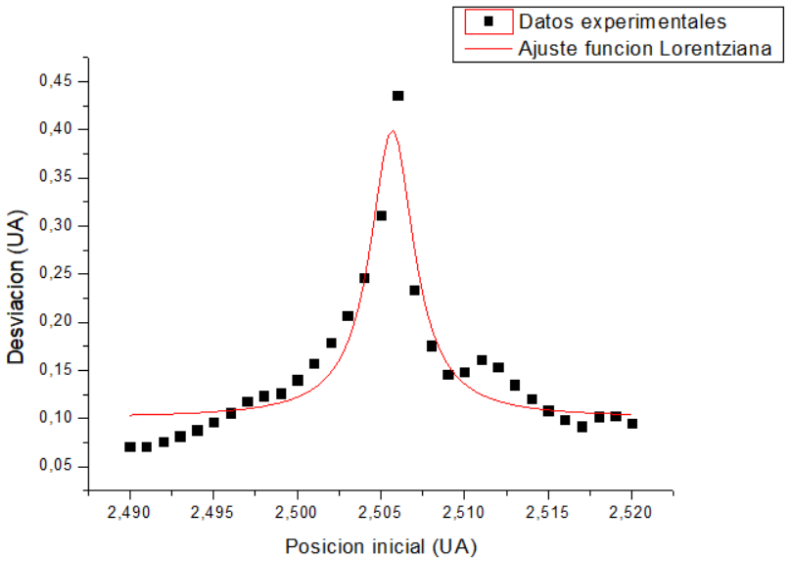

Figura 31: Desviación de un asteroide (en resonancia 3:1 con Júpiter) en función de su posición inicial respecto al Sol, T=1000años. Los datos han sido ajustados a una Lorentziana

En este caso hemos observado que la desviación en función de la posición inicial del asteroide parece seguir una distribución lorentziana.

Una vez hemos ajustado los datos:

$$\text{Centro de la lorentziana} = 2.5056 \pm 0.0001 \text{ UA} \qquad \text{Anchura} = 0.0032 \pm 0.0004 \text{ UA}$$

Gracias a esta representación concluimos que existe una Zanja de Kirkwood entorno a las 2.5056 UA y decimos que un asteroide se encuentra en sus proximidades si está situado entre 2.5040 y 2.5072 UA.

4.2.1.2 Zona próxima a 3.30 UA

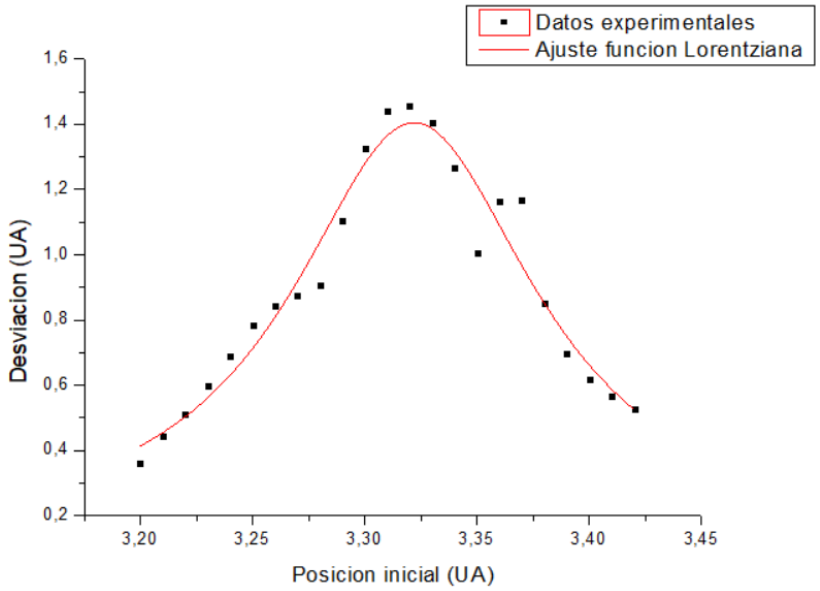

Figura 32: Desviación de un asteroide (en resonancia 2:1 con Júpiter) en función de su posición inicial respecto al Sol, T=1000 años. Los datos han sido ajustados a una Lorentziana

Nuevamente observamos que la desviación en función de la posición inicial del asteroide sigue una distribución lorentziana. Una vez ajustados hemos obtenido.

$$\text{Centro de la lorentziana} = 3.322 \pm 0.002 \text{ UA} \qquad \text{Anchura} = 0.13 \pm 0.01 \text{ UA}$$

Por lo tanto podemos afirmar que existe una Zanja de Kirkwood en 3.32 UA y decimos que un asteroide se encuentra en sus proximidades si está situado entre 3.39 y 3.25 UA.

4.2.2 Influencia de Marte en los asteroides

Para ver la influencia de Marte en los asteroides hemos realizado la simulación de los asteroides en el caso más favorable posible, es decir, en la zanja que se encuentra en torno al 2.50 por ser la que más cerca se encuentra de Marte. En este caso hemos simulado 100 000 años para dar tiempo a que se manifieste la interacción con dicho planeta. El código puede verse en la *Sección 7.1 del anexo.*

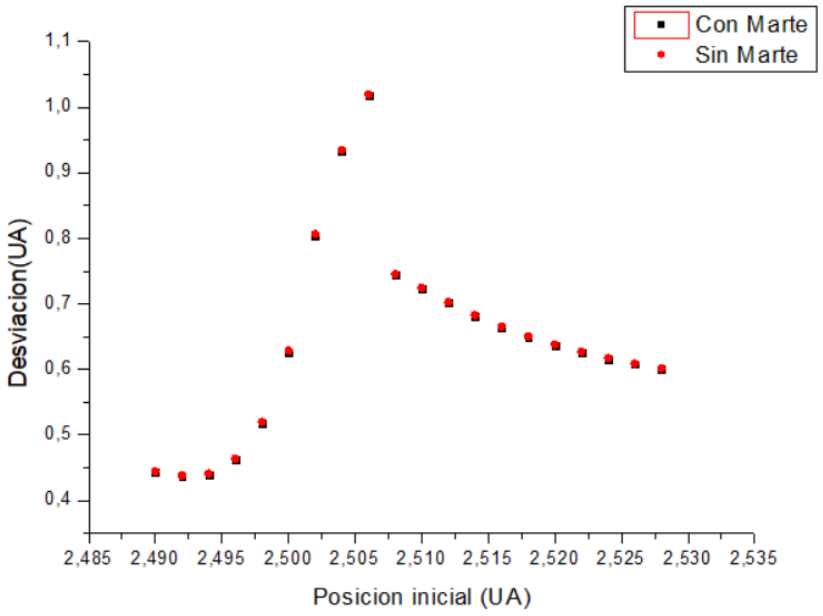

Figura 33: Comparación de la desviación de un asteroide en función de su posición inicial respecto al Sol al cabo de 100 000 años con la presencia de Marte y sin la presencia de Marte.

Los resultados parecen coincidir con lo esperado ya que como vemos en la *Figura 33*, el efecto que tiene Marte sobre los asteroides es despreciable en el rango de tiempo estudiado. Aunque Marte está más cerca de la mayoría de los asteroides, la diferencia en la distancia no se compensa con la diferencia en masa, ya que la masa de Marte es aproximadamente 10^7 veces menor a la del Sol.

La ecuación de la fuerza gravitatoria que sufren los asteroides debido a Marte sería:

$$F_G = - \frac{G \cdot M_M \cdot M_a}{{r_{aM}}^3} \cdot (r_a - r_M) \tag{35}$$

Con $GM_M = 4\,\pi^2 \frac{M_M}{M_S}$ $AU^3/año^2$, el cociente entre M_M y M_S es aproximadamente 10^{-7}.

Como en esta zona, que es donde los asteroides están más cerca de dicho planeta, no observamos ningún efecto. Concluimos que el efecto que tiene Marte sobre los asteroides es despreciable.

4.2.3 Influencia de Saturno en los asteroides

Análogamente al caso de Marte hemos hecho lo mismo con Saturno pero en la zanja más próxima a él. El código aparece en la *Sección 7.2 del anexo.*

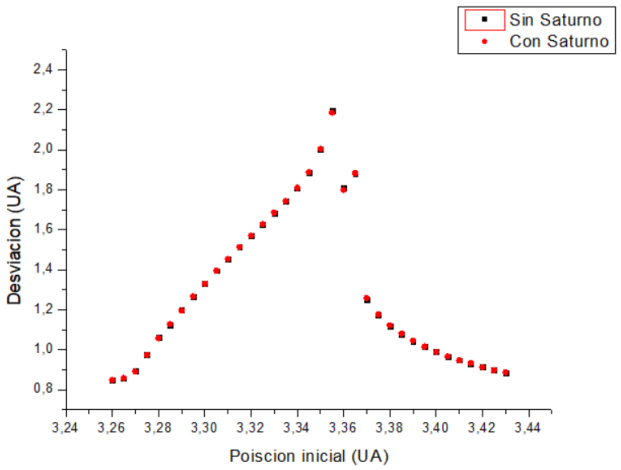

Figura 34: Comparación de la desviación de un asteroide en función de su posición inicial respecto del Sol al cabo de 100 000 años con la presencia de Saturno y sin la presencia de Saturno

En la *Figura 34* podemos ver que al igual que sucede con Marte la interacción de los asteroides con Saturno es prácticamente despreciable en el rango de tiempos estudiado. Como vimos en la ecuación (2) la fuerza es inversamente proporcional al cuadrado de la distancia y directamente proporcional a la masa del planeta. Por lo tanto, aunque la masa de Saturno ($5.98 \cdot 10^{26}$ kg) sea unas 10 menor que la de Júpiter como su semieje mayor es aproximadamente 9.58 UA se compensa.

5. Conclusiones

A la vista de los resultados concluimos por tanto:

- El método de Velocidad Verlet es el mas eficiente para el cálculo de órbitas,
- Existen unas regiones entre Marte y Júpiter en las cuales los asteroides exhiben un comportamiento inusual.
- Cuando el sistema se vuelve más realista, órbita de Júpiter elíptica, el efecto sobre los asteroides se acentúa, quedando mejor definidas las Zanjas de Kirkwood.
- Con el código desarrollado es posible estimar la anchura de dichas zanjas.
- La influencia de Marte y Saturno en los asteroides que se encuentran en las Zanjas de Kirkwood es despreciable.

Bibliografía:

[1] GIORDANO, Nicholas. *Computational Physics.* Department of Physics, Purdue University,2006.

[2] ANZALONE, Evan; CHAI, Patrick. Numerical Integration Techniques in Orbital Mechanics Applications.

[3] VOESENEK, C. J. Implementing a Fourth Order Runge-Kutta Method for Orbit Simulation. 2008.

[4] MURRAY, Carl D. Real and imaginary Kirkwood gaps. *Monthly Notices of the Royal Astronomical Society*, 1996, vol. 279, no 3, p. 978-986.

[5] Wisdom, J. The origin of the Kirkwood gaps-A mapping for asteroidal motion near the 3/1 commensurability. *The Astronomical Journal*, 1982.

[6] Moons, M. Review of the dynamics in the Kirkwood gaps. *Celestial Mechanics and Dynamical Astronomy*, 1996.

[7] Nieto, M. M. *The Titius-Bode law of planetary distances: its history and theory* (Vol. 47). Elsevier 2014.

[8] Calogero, F. Solution of a three-body problem in one dimension. *Journal of Mathematical Physics*, 1969.

[9] ROGERS, John H. *The giant planet Jupiter*. Cambridge University Press, 1995.

[10] GROENEVELD, Ingrid; KUIPER, Gerard P. Photometric studies of asteroids. II. *The Astrophysical Journal*, 1954, vol. 120, p. 529.

[11] Origin (Pro), *Version 8.1*. OriginLab Corporation, Northamptom MA, USA

Anexo

1. Funciones para los métodos de integración numérica:

```
def Euler(a,r0,v0,dt,tend):

#Creamos vectores de las posiciones, tiempo y velocidades

    x_vector = []
    y_vector = []
    z_vector = []

    t_vector = []

    vx_vector = []
    vy_vector = []
    vz_vector = []

#Condiciones iniciales

    x = r0[0]
    y = r0[1]
    z = r0[2]
    t = 0                    #Tiempo a partir del cual empezamos a contar
    vx = v0[0]
    vy = v0[1]
    vz = v0[2]

#Decimos que el primer elemento de nuestros vectores sea:

    x_vector.append(x)
    y_vector.append(y)
    z_vector.append(z)

    vx_vector.append(vx)
    vy_vector.append(vy)
    vz_vector.append(vz)

    t_vector.append(t)
    n=int(tend/dt)       #Numero de elementos que contendran nuestros vectores

    for i in range (0,n):

        [ax, ay, az] = a(x, y,z)

        x_next = x + vx * dt
        y_next = y + vy * dt
        z_next = z + vz * dt

        vx_next = vx + ax * dt
        vy_next = vy + ay * dt
        vz_next = vz + az * dt

        x_vector.append(x_next)
        y_vector.append(y_next)
        z_vector.append(z_next)

        vx_vector.append(vx_next)
        vy_vector.append(vy_next)
        vz_vector.append(vz_next)
#
        t_next = t + dt
#
        t_vector.append(t_next)

        x = x_next
        y = y_next
        z = z_next

        t=t_next

        vx = vx_next
        vy = vy_next
        vz = vz_next

    return t_vector, x_vector, y_vector, z_vector, vx_vector, vy_vector, vz_vector
```


Figura 35: Código del método de Euler teniendo en cuenta interacción solo con el Sol

```
109 def Euler_Cromer2(a,r0,v0,dt,tend,x_jup_EC, y_jup_EC ,z_jup_EC):
110
111 #Creamos vectores de las posiciones, tiempo y velocidades
112
113     x_vector = []
114     y_vector = []
115     z_vector = []
116
117     t_vector = []
118
119     vx_vector = []
120     vy_vector = []
121     vz_vector = []
122
123 #Condiciones iniciales
124
125     x = r0[0]
126     y = r0[1]
127     z = r0[2]
128     t=0          #Tiempo a partir del cual empezamos a contar
129     vx = v0[0]
130     vy = v0[1]
131     vz = v0[2]
132
133
134 #Decimos que el primer elemento de nuestros vectores sea:
135
136     x_vector.append(x)
137     y_vector.append(y)
138     z_vector.append(z)
139
140     vx_vector.append(vx)
141     vy_vector.append(vy)
142     vz_vector.append(vz)
143
144     t_vector.append(t)
145
146     n=int(tend/dt)       #Numero de elementos que contendran nuestros vectores
148    for i in range (0,n):
149
150 # _next significa del paso siguiente y si no lleva nada significa que es el actual
151
152         [ax, ay, az] = a(x, y, z, x_jup_EC[i], y_jup_EC[i] ,z_jup_EC[i])
153
154         vx_next = vx + ax * dt         # La velocidad del paso siguiente = a la v del paso actual
155         vy_next = vy + ay * dt         # mas la aceleracion del paso siguiente por intervalo de tiempo
156         vz_next = vz + az * dt
157
158         x_next = x + vx_next * dt
159         y_next = y + vy_next * dt
160         z_next = z + vz_next * dt
161
162         x_vector.append(x_next)             # Guardamos los valores en nuestros vectores
163         y_vector.append(y_next)
164         z_vector.append(z_next)
165
166         vx_vector.append(vx_next)
167         vy_vector.append(vy_next)
168         vz_vector.append(vz_next)
169
170         t_next = t + dt
171
172         t_vector.append(t_next)
173
174         x = x_next                      # Actualizamos los valores  para realizar de nuevo el bucle
175         y = y_next
176         z = z_next
177
178         t = t_next
179
180         vx = vx_next
181         vy = vy_next
182         vz = vz_next
183
184     return t_vector,x_vector, y_vector, z_vector, vx_vector, vy_vector, vz_vector
```

Figura 36: Código del método de Euler-Cromer, teniendo en cuenta interacción con Sol y Júpiter

```python
def Verlet(a,r0,v0,dt,tend):
#Creamos vectores de las posiciones, tiempo y velocidades

    x_vector = []
    y_vector = []
    z_vector = []

    t_vector = []

    vx_vector = []
    vy_vector = []
    vz_vector = []

#Condiciones iniciales

    x = r0[0]
    y = r0[1]
    z = r0[2]
    t=0          #Tiempo a partir del cual empezamos a contar
    vx = v0[0]
    vy = v0[1]
    vz = v0[2]

#Decimos que el primer elemento de nuestros vectores sea:

    x_vector.append(x)
    y_vector.append(y)
    z_vector.append(z)

    vx_vector.append(vx)
    vy_vector.append(vy)
    vz_vector.append(vz)

    t_vector.append(t)

    n=int(tend/dt)       #Numero de elementos que contendran nuestros vectores
```

```python
   for i in range (0,n):

       [ax, ay, az] = a(x, y, z)     #Aceleracion

       x_next= x + vx * dt + 0.5 * ax * dt ** 2
       y_next= y + vy * dt + 0.5 * ay * dt ** 2
       z_next= z + vz * dt + 0.5 * az * dt ** 2

       [ax_next, ay_next, az_next] = a(x_next, y_next, z_next) # Aceleraciones del paso siguiente

       vx_next = vx + 0.5 * ( ax + ax_next ) * dt  # La velocidad del paso siguiente = a la v del paso actual
       vy_next = vy + 0.5 * ( ay + ay_next ) * dt  # + 0.5 de la suma de la aceleracion actual mas la aceleracion siguiente
       vz_next = vz + 0.5 * ( az + az_next ) * dt

       x_vector.append(x_next)      # Guardamos los valores en nuestros vectores
       y_vector.append(y_next)
       z_vector.append(z_next)

       vx_vector.append(vx_next)
       vy_vector.append(vy_next)
       vz_vector.append(vz_next)

       t_next = t + dt

       t_vector.append(t_next)

       x = x_next      # Actualizamos los valores  para realizar de nuevo el bucle
       y = y_next
       z = z_next

       t = t_next

       vx = vx_next
       vy = vy_next
       vz = vz_next

   return t_vector,x_vector, y_vector, z_vector, vx_vector, vy_vector, vz_vector
```

Figura 37: Código del método Velocidad Verlet, teniendo en cuenta interacción con el Sol- Para tener en cuenta la interacción con Júpiter habría que añadirlo en los parámetros de entrada de la función

```python
def Verlet3(a,r0,v0,dt,tend,x_jup , y_jup ,z_jup, x_mar, y_mar, z_mar ):
#Creamos vectores de las posiciones, tiempo y velocidades

    x_vector = []
    y_vector = []
    z_vector = []

    t_vector = []

    vx_vector = []
    vy_vector = []
    vz_vector = []

#Condiciones iniciales

    x = r0[0]
    y = r0[1]
    z = r0[2]
    t=0          #Tiempo a partir del cual empezamos a contar
    vx = v0[0]
    vy = v0[1]
    vz = v0[2]

#Decimos que el primer elemento de nuestros vectores sea:

    x_vector.append(x)
    y_vector.append(y)
    z_vector.append(z)

    vx_vector.append(vx)
    vy_vector.append(vy)
    vz_vector.append(vz)

    t_vector.append(t)

    n=int(tend/dt)       #Numero de elementos que contendran nuestros vectores

  for i in range (0,n):

      [ax, ay, az] = a(x, y, z,x_jup[i], y_jup[i] ,z_jup[i],x_mar[i], y_mar[i] ,z_mar[i])     #Aceleraciones

      x_next= x + vx * dt + 0.5 * ax * dt ** 2
      y_next= y + vy * dt + 0.5 * ay * dt ** 2
      z_next= z + vz * dt + 0.5 * az * dt ** 2

      [ax_next, ay_next, az_next] = a(x_next, y_next, z_next, x_jup[i], y_jup[i] ,z_jup[i],x_mar[i], y_mar[i] ,z_mar[i]) # Aceleraciones del paso siguiente

      vx_next = vx + 0.5 * ( ax + ax_next ) * dt  # La velocidad del paso siguiente = a la v del paso actual
      vy_next = vy + 0.5 * ( ay + ay_next ) * dt  # + 0.5 de la suma de la aceleracion actual mas la aceleracion siguiente
      vz_next = vz + 0.5 * ( az + az_next ) * dt

      x_vector.append(x_next)      # Guardamos los valores en nuestros vectores
      y_vector.append(y_next)
      z_vector.append(z_next)

      vx_vector.append(vx_next)
      vy_vector.append(vy_next)
      vz_vector.append(vz_next)
#
      t_next = t + dt
#
      t_vector.append(t_next)

      x = x_next      # Actualizamos los valores  para realizar de nuevo el bucle
      y = y_next
      z = z_next

      t = t_next

      vx = vx_next
      vy = vy_next
      vz = vz_next

  return t_vector,x_vector, y_vector, z_vector, vx_vector, vy_vector, vz_vector
```

Figura 38: Código del método de Velocidad Verlet, teniendo en cuenta interacción con el Sol, Júpiter y otro planeta como Saturno o Marte. Como vemos solo hay que modificar los parámetros de entrada en la función y en ele cálculo de las aceleraciones.

```
134 def RungeK_42(a,r0,v0,dt,tend,x_jup_RK4, y_jup_RK4 ,z_jup_RK4):
135
136 #Creamos vectores de las posiciones, tiempo y velocidades
137
138     x_vector = []
139     y_vector = []
140     z_vector = []
141
142     t_vector = []
143
144     vx_vector = []
145     vy_vector = []
146     vz_vector = []
147
148 #Condiciones iniciales
149
150     x = r0[0]
151     y = r0[1]
152     z = r0[2]
153     t=0          #Tiempo a partir del cual empezamos a contar
154     vx = v0[0]
155     vy = v0[1]
156     vz = v0[2]
157
158 #Decimos que el primer elemento de nuestros vectores sea:
159
160     x_vector.append(x)
161     y_vector.append(y)
162     z_vector.append(z)
163
164     vx_vector.append(vx)
165     vy_vector.append(vy)
166     vz_vector.append(vz)
167
168     t_vector.append(t)
169
170     n=int(tend/dt)      #Numero de elementos que contendran nuestros vectores
```

```
    for i in range (0,n):

#Las  k de las velocidades que las denotaremos como k1vx,k1vy,k1vz
#Las k para las posiciones las denotaremos como k1x, k1y,k1z

#Ecuaciones Rungekutta4

        [k1vx, k1vy, k1vz] = a(x, y, z,x_jup_RK4[i], y_jup_RK4[i] ,z_jup_RK4[i])

        k1x = vx
        k1y = vy
        k1z = vz

        [k2vx, k2vy, k2vz] = a(x + k1x * (dt / 2), y + k1y * (dt / 2), z + k1z * (dt / 2),x_jup_RK4[i], y_jup_RK4[i] ,z_jup_RK4[i])

        k2x = vx + k1vx * (dt / 2)
        k2y = vy + k1vy * (dt / 2)
        k2z = vz + k1vz * (dt / 2)

        [k3vx, k3vy, k3vz] = a(x + k2x * (dt / 2), y + k2y * (dt / 2), z + k2z * (dt / 2),x_jup_RK4[i], y_jup_RK4[i] ,z_jup_RK4[i])

        k3x = vx + k2vx * (dt / 2)
        k3y = vy + k2vy * (dt / 2)
        k3z = vz + k2vz * (dt / 2)

        [k4vx, k4vy, k4vz] = a(x + k3x * dt, y + k3y * dt, z + k3z * dt,x_jup_RK4[i], y_jup_RK4[i] ,z_jup_RK4[i])

        k4x = vx + k3vx * dt
        k4y = vy + k3vy * dt
        k4z = vz + k3vz * dt

#Finalmente obtenemos las velocidades y posiciones siguientes

        vx_next = vx + (dt / 6) * (k1vx + 2 * k2vx + 2 * k3vx + k4vx)
        vy_next = vy + (dt / 6) * (k1vy + 2 * k2vy + 2 * k3vy + k4vy)
        vz_next = vz + (dt / 6) * (k1vz + 2 * k2vz + 2 * k3vz + k4vz)

        x_next = x + (dt / 6) * (k1x + 2 * k2x + 2 * k3x + k4x)
        y_next = y + (dt / 6) * (k1y + 2 * k2y + 2 * k3y + k4y)
        z_next = z + (dt / 6) * (k1z + 2 * k2z + 2 * k3z + k4z)
```

```
        x_vector.append(x_next)             # Guardamos los valores en nuestros vectores
        y_vector.append(y_next)
        z_vector.append(z_next)

        vx_vector.append(vx_next)
        vy_vector.append(vy_next)
        vz_vector.append(vz_next)

        t_next = t + dt

        t_vector.append(t_next)

        x = x_next                      # Actualizamos los valores  para realizar de nuevo el bucle
        y = y_next
        z = z_next

        t = t_next

        vx = vx_next
        vy = vy_next
        vz = vz_next

    return t_vector,x_vector, y_vector, z_vector, vx_vector, vy_vector, vz_vector
```

Figura 38: Código del método Runge-Kutta 4, teniendo en cuenta interacción con el Sol y Júpiter

2. Funciones para la gravedad y la energía mecánica

2.1 Funciones que definen la gravedad

```python
'''FUNCIONES DE LA GRAVEDAD'''
#Gravedad que sufre el cuerpo celeste debido al Sol

def grav_sun(x,y,z):
    ax= -4 * pi ** 2 * x / ( x ** 2 + y ** 2 + z ** 2 ) ** (3/2)
    ay= -4 * pi ** 2 * y / ( x ** 2 + y ** 2 + z ** 2 ) ** (3/2)
    az= -4 * pi ** 2 * z / ( x ** 2 + y ** 2 + z ** 2 ) ** (3/2)
    return [ax,ay,az]

#Gravedad que sufre el cuerpo celeste debido al Sol y Jupiter

def grav_sun_Jupiter(xa,ya,za, xj,yj,zj): #Podriamos generalizarla si quisieramos, poniendo Mjup y Msun como imput
    Mjup=1.898E27
    Msun=1.989E30

    raj=sqrt((xa-xj)**(2)+(ya-yj)**(2)+(za-zj)**(2))    # Distancia a Jupiter
    ra=( xa ** 2 + ya ** 2 + za ** 2 ) ** (1/2)         # Distancia al Sol

    ax= -4 * pi ** 2 * xa / ra ** 3 - 4 * pi ** 2 * (Mjup/Msun) * (xa-xj)/raj**3 #Aceleracion debida al sol y a jupiter
    ay= -4 * pi ** 2 * ya / ra ** 3 - 4 * pi ** 2 * (Mjup/Msun) * (ya-yj)/raj**3
    az= -4 * pi ** 2 * za / ra ** 3 - 4 * pi ** 2 * (Mjup/Msun) * (za-zj)/raj**3
    return [ax,ay,az]

#Gravedad que sufre el cuerpo celeste debido al Sol, Jupiter y otro planeta

def grav_sun_Jupiter_Marte(xa,ya,za, xj,yj,zj,xm,ym,zm): #Podriamos generalizarla si quisieramos, poniendo Mjup y Msun como imput
    Mjup=1.898E27
    Msun=1.989E30
    Mmar=6.415E23 #En este caso es la masa de Marte, pero podria ser la de otro planeta

    ram=sqrt((xa-xm)**(2)+(ya-ym)**(2)+(za-zm)**(2))    # Distancia a Marte
    raj=sqrt((xa-xj)**(2)+(ya-yj)**(2)+(za-zj)**(2))    # Distancia a Jupiter
    ra=( xa ** 2 + ya ** 2 + za ** 2 ) ** (1/2)         # Distancia al Sol

    #Aceleracion debida al Sol, Jupiter y Marte
    ax= -4 * pi ** 2 * xa / ra ** 3 - 4 * pi ** 2 * (Mjup/Msun) * (xa-xj)/raj**3 - 4 * pi ** 2 * (Mmar/Msun) * (xa-xm)/ram**3
    ay= -4 * pi ** 2 * ya / ra ** 3 - 4 * pi ** 2 * (Mjup/Msun) * (ya-yj)/raj**3 - 4 * pi ** 2 * (Mmar/Msun) * (ya-ym)/ram**3
    az= -4 * pi ** 2 * za / ra ** 3 - 4 * pi ** 2 * (Mjup/Msun) * (za-zj)/raj**3 - 4 * pi ** 2 * (Mmar/Msun) * (za-zm)/ram**3
    return [ax,ay,az]
```

Figura 39: Funciones para definir la gravedad en función del número de cuerpos celestes implicados.

2.2 Funciones que definen la energía mecánica

```
'''FUNCIONES DE ENERGIA'''
#Energia mecanica teniendo en cuenta la interaccion con el Sol
def energy_mecanica_S(x,y,z,vx,vy,vz,Mplaneta1,Msun):
    n=np.size(vx)
    Energia_cinet=np.zeros(n)
    Energia_grav=np.zeros(n)
    Energia_mec=np.zeros(n)
    v=np.zeros(n)
    r=np.zeros(n)
    for i in range(n):
        v[i] =sqrt( vx[i] ** 2 + vy[i] ** 2 + vz[i] ** 2)
        Energia_cinet[i]= 0.5*v[i] ** 2 * Mplaneta1
        r[i] =sqrt( x[i] ** 2 + y[i] ** 2 + z[i] ** 2)
        Energia_grav[i] = - 4*pi**2*Mplaneta1/ r[i]
        Energia_mec[i]= Energia_grav[i] + Energia_cinet[i]
    return np.array(Energia_cinet), np.array(Energia_grav),np.array(Energia_mec)

#Energia mecanica teniendo en cuenta la interaccion con el Sol y Jupiter
def energy_mecanica(x,y,z,vx,vy,vz,xj,yj,zj,Masteroide):
    n=np.size(vx)
    Mjup=1.898E27
    Msun=1.989E30
    Energia_cinet=np.zeros(n)
    Energia_grav=np.zeros(n)
    Energia_mec=np.zeros(n)
    v=np.zeros(n)
    rast=np.zeros(n)
    rajup=np.zeros(n)
    for i in range(n):
        v[i] =sqrt( vx[i] ** 2 + vy[i] ** 2 + vz[i] ** 2)
        Energia_cinet[i]= 0.5 *v[i] ** 2 * Masteroide
        rast[i] =sqrt( x[i] ** 2 + y[i] ** 2 + z[i] ** 2)
        rajup[i] =sqrt( (x[i]-xj[i]) ** 2 + (y[i]-yj[i]) ** 2 + (z[i]-zj[i]) ** 2)

        Energia_grav[i] = - 4*pi**2*Masteroide/ rast[i]  - 4 * pi**2 *(Mjup/Msun)*Masteroide /rajup[i]
        Energia_mec[i]= Energia_grav[i] + Energia_cinet[i]
    return np.array(Energia_cinet), np.array(Energia_grav),np.array(Energia_mec)
```

Figura 39: Funciones para calcular la energía mecánica función del número de cuerpos celestes implicados.

3. Comparación de los distintos métodos órbita de Júpiter

```python
import numpy as np
from math import sin,cos,pi,sqrt,exp,fabs
import matplotlib.pyplot as plt
from pylab import plot,title,xlabel,ylabel,legend,figure,show,subplot,size
from timeit import default_timer as timer
import statistics as stats

from Funciones import grav_sun, energy_mecanica_S
from Metodo_Euler import Euler
from Metodo_Euler_Cromer import Euler_Cromer
from Metodo_Verlet import Verlet
from Metodo_Rungekutta_4 import RungeK_4

#Definimos constantes
tend=200
deltat=0.002          #year
Msun=1.989e30

#Parametros iniciales Jupiter Con el origen en el Sol
y0_jup=0
z0_jup=0
exce_jupiter=0.048
x0_jup= (1+exce_jupiter) * 5.20     #UA
M_jup= 1.898e27                      #kg

vx0_jup=0
vy0_jup= sqrt(4 * pi ** 2 * (1-exce_jupiter) / x0_jup * (1+M_jup/Msun))  #AU/year
vz0_jup=0

#Posicion de Jupiter
r0_jup=[x0_jup , 0, 0]
v0_jup=[vx0_jup, vy0_jup, vz0_jup]
```

```python
#LLamamos a las funciones que contienen los metodos de integracion numerica
start1=timer()
t_jup_E, x_jup_E, y_jup_E, z_jup_E, vx_jup_E, vy_jup_E, vz_jup_E = Euler (grav_sun , r0_jup, v0_jup, deltat,tend)
end1=timer()
start2=timer()
t_jup_EC, x_jup_EC, y_jup_EC, z_jup_EC, vx_jup_EC, vy_jup_EC, vz_jup_EC = Euler_Cromer (grav_sun , r0_jup, v0_jup, deltat,tend)
end2=timer()
start3=timer()
t_jup_V, x_jup_V, y_jup_V, z_jup_V, vx_jup_V, vy_jup_V, vz_jup_V = Verlet (grav_sun , r0_jup, v0_jup, deltat,tend)
end3=timer()
start4=timer()
t_jup_RK4, x_jup_RK4, y_jup_RK4, z_jup_RK4, vx_jup_RK4, vy_jup_RK4, vz_jup_RK4 =  RungeK_4 (grav_sun , r0_jup, v0_jup, deltat,tend)
end4=timer()

#Comparamos los metodos
print('El metodo de Euler tarda en calcular la trayectoria',end1-start1, 's')
print('El metodo de Euler Cromer tarda en calcular la trayectoria',end2-start2, 's')
print('El metodo de Verlet tarda en calcular la trayectoria',end3-start3, 's')
print('El metodo de Rungekutta4 tarda en calcular la trayectoria',end4-start4, 's')
```

```
#%%Comparamos los 4 metodos

figure(1)
#Pintamos el afelio y el perihelio
plt.plot(-(1-exce_jupiter) * 5.20 ,0,'+g',linewidth=50,label='Perihelio de Júpiter')
plt.plot( (1+exce_jupiter) * 5.20 ,0,'+m',linewidth=50,label='Afelio de Júpiter')
#Pintamos las orbitas
plt.plot(x_jup_E, y_jup_E,'g', label='Euler')
plt.plot(x_jup_EC, y_jup_EC,'y', label='Euler Cromer ')
plt.plot(x_jup_V, y_jup_V,'b', label='Verlet Velocity')
plt.plot(x_jup_RK4, y_jup_RK4,'r', label='RungeK_4')

plt.xlabel('x(AU)')
plt.ylabel('y(AU)')


plt.legend()
plt.legend( loc = 'center')

show()
```

Figura 40: Codigo principal para comparar los distintos métodos al calcular la orbita de Júpiter en un intervalo de tiempo de 200 años. Importamos las funciones que vamos a emplear de librerías conocidas y de las que hemos creado nosotros. A continuación establecemos los parámetros iniciales de Júpiter y llamamos a las funciones que contienen los métodos de integración. Asimismo, con la función timer () calculamos el tiempo que tarda en realizar el calculo cada método.

3.1 Comparación de las energías mecánicas con los 4 métodos

```
#%% Energias con los distintos metodos

Ecin_jup_E,Egrav_jup_E,Emec_jup_E= energy_mecanica_S(x_jup_E, y_jup_E, z_jup_E, vx_jup_E, vy_jup_E, vz_jup_E, M_jup,Msun)
Ecin_jup_V,Egrav_jup_V,Emec_jup_V= energy_mecanica_S(x_jup_V, y_jup_V, z_jup_V, vx_jup_V, vy_jup_V, vz_jup_V, M_jup,Msun)
Ecin_jup_EC,Egrav_jup_Ec,Emec_jup_EC= energy_mecanica_S(x_jup_EC, y_jup_EC, z_jup_EC, vx_jup_EC, vy_jup_EC, vz_jup_EC , M_jup,Msun)
Ecin_jup_RK4,Egrav_jup_RK4,Emec_jup_RK4= energy_mecanica_S(x_jup_RK4, y_jup_RK4, z_jup_RK4, vx_jup_RK4, vy_jup_RK4, vz_jup_RK4, M_jup,Msun)

figure(3)
plt.plot(t_jup_E,Emec_jup_E,'g',label='Euler ')
plt.plot(t_jup_V,Emec_jup_V,'red',label='Verlet Velocity ')
plt.plot(t_jup_E,Emec_jup_RK4,'pink',label='RK4 ')
plt.plot(t_jup_V,Emec_jup_EC,'black',label='Euler Cromer')
plt.xlabel('Time(años)')
plt.ylabel('Energia mecanica (kg(AU/años)^2)')

plt.legend(loc='upper right')

show()
```

Figura 41: Parte del código principal donde calculamos las energías mecánicas con los distintos métodos para las órbitas de Júpiter en un intervalo de tiempo de 200 años . Para ello llamamos a las funciones creadas anteriormente (Figura 39)

3.2 Cálculo del período de Júpiter

```
#%% Calculamos el periodo  y lo comparamos con el valor Real

n=int(tend/deltat)
T1_jup=0
T2_jup=0
T_jup=[]
x_afel=[]
x_afel1=[]
x_afel2=[]
tol=0.000001
for i in range(0,n):
    if fabs(x_jup_V[i]-x_jup_V[0])<tol:                              #abs
        T1_jup=T2_jup
        T2_jup = deltat * i
        T_jup1 =T2_jup-T1_jup
        T_jup.append(T_jup1)
        x_afel.append(fabs(x_jup_V[i]-x_jup_V[0]))

    if fabs(x_jup_EC[i]-x_jup_EC[0])<tol:                             #abs
        x_afel1.append(fabs(x_jup_EC[i]-x_jup_EC[0]))

    if fabs(x_jup_RK4[i]-x_jup_RK4[0])<tol:                           #abs
        x_afel2.append(fabs(x_jup_RK4[i]-x_jup_RK4[0]))

Tjup=5.204267**(3/2)*2*pi/sqrt(4*pi**2)
x_afelm=sum(x_afel)/(np.size(x_afel)-1)
x_afel1m=sum(x_afel1)/(np.size(x_afel1)-1)
x_afel2m=sum(x_afel2)/(np.size(x_afel2)-1) #Quitamos el primer elemento porque es un 0

#Perido lo convertimos en lista
T_jupm=sum(T_jup)/(np.size(T_jup)-1)
lista=np.array(T_jup[1:],float)
media=lista.mean()
desviacion=lista.std()
print('La media es=',media, 'La desviacion es',desviacion,'La varianza es', lista.var())
print('La desviacion respecto al valor real',stats.stdev(lista))
```

```
print('El periodo real de Jupiter es= ' ,Tjup ,'años')
print('La desviacion en el afelio con Verlet es=',x_afelm)
print('La desviacion en el afelio con Euler Cromer es=',x_afel1m)
print('La desviacion en el afelio con RK4 es=',x_afel2m)
print('El periodo experimental de jupiter es=',T_jupm)

#Pintamos los valores en una recta
a=[x_afel1m,x_afel1m]
b=[x_afelm,x_afelm]
c=[x_afel2m,x_afel2m]
d=[0,1]
figure(2)
plt.plot( d,a,'y', label='Euler Cromer ')
plt.plot(d,b,'g', label='Verlet Velocity')
plt.plot( d,c,'r', label='RungeK_4')
plt.ylabel('Desviacion (UA)')
plt.plot(d,[0,0],'b',label='Valor observado')

plt.legend(loc='upper right')
show()
```

Figura 42: Parte del código principal para comparar los distintos métodos al calcular la órbita de Júpiter en un intervalo de tiempo de 200 años. Con el bucle "for" hemos calculado el periodo de Júpiter y posteriormente hemos realizado una grafica con la desviación del afelio de los distintos métodos.

4. Comparación de los distintos métodos órbita de Juno

```
import numpy as np
from math import sin,cos,pi,sqrt,exp,fabs
import matplotlib.pyplot as plt
from pylab import plot,title,xlabel,ylabel,legend,figure,show,subplot
from timeit import default_timer as timer
import statistics as stats

from Funciones_gravedad import grav_sun,grav_sun_Jupiter,energy_mecanica
from Metodo_Verlet import Verlet, Verlet2
from Metodo_Euler_Cromer import Euler_Cromer,Euler_Cromer2
from Metodo_Rungekutta_4 import RungeK_42,RungeK_4

#Definimos constantes
tend=2000
deltat=0.002          #year
M_jup= 1.898e27                      #kg
Msun=1.989e30
#Parametros iniciales Jupiter
y0_jup=0
z0_jup=0
exce_jupiter=0.048
x0_jup= (1+exce_jupiter) * 5.20      #Distancia al fooco
r0_jup=[x0_jup , 0, 0]
#Velocidades
vx0_jup=0
vy0_jup= sqrt(4 * pi ** 2 * (1-exce_jupiter) / x0_jup * (1+M_jup/Msun))
vz0_jup=0
v0_jup=[vx0_jup, vy0_jup, vz0_jup]

#Asteoride Juno
#Posicion
exce_ast=0.2554
x0_ast1=3.353
y0_ast1=0
z0_ast1=0
r0_ast1=[x0_ast1 , 0, 0]

#Velocidades

vx0_ast1=0
vy0_ast1=sqrt(4 * pi ** 2 * (1-exce_ast) / x0_ast1 )  #AU/year
vz0_ast1=0
v0_ast1=[vx0_ast1, vy0_ast1, vz0_ast1]
```

```
#Calculo primero las posiciones de Jupiter

start2=timer()
t_jup_EC, x_jup_EC, y_jup_EC, z_jup_EC, vx_jup_EC, vy_jup_EC, vz_jup_EC =\
    Euler_Cromer (grav_sun , r0_jup, v0_jup, deltat,tend)
start3=timer()
t_jup_V, x_jup_V, y_jup_V, z_jup_V, vx_jup_V, vy_jup_V, vz_jup_V =\
    Verlet (grav_sun , r0_jup, v0_jup, deltat,tend)
start4=timer()
t_jup_RK4, x_jup_RK4, y_jup_RK4, z_jup_RK4, vx_jup_RK4, vy_jup_RK4, vz_jup_RK4 =\
    RungeK_4 (grav_sun , r0_jup, v0_jup, deltat,tend)

#Calculo las del asteoride
t_ast1_EC, x_ast1_EC, y_ast1_EC, z_ast1_EC, vx_ast1_EC, vy_ast1_EC, vz_ast1_EC =\
    Euler_Cromer2 (grav_sun_Jupiter , r0_ast1, v0_ast1, deltat,tend,x_jup_EC, y_jup_EC, z_jup_EC)
end2=timer()

t_ast_V, x_ast1_V, y_ast1_V, z_ast1_V, vx_ast1_V, vy_ast1_V, vz_ast1_V =\
    Verlet2 (grav_sun_Jupiter , r0_ast1, v0_ast1, deltat,tend,x_jup_V, y_jup_V, z_jup_V)
end3=timer()

t_ast1_RK4, x_ast1_RK4, y_ast1_RK4, z_ast1_RK4, vx_ast1_RK4, vy_ast1_RK4, vz_ast1_RK4 =\
    RungeK_42 (grav_sun_Jupiter , r0_ast1, v0_ast1, deltat,tend,x_jup_RK4, y_jup_RK4, z_jup_RK4)
end4=timer()

#Calculamos los tiempos de ejecucion y hacemos las gráficas
print('El metodo de Euler Cromer tarda en calcular la trayectoria',end2-start2, 's')
print('El metodo de Verlet tarda en calcular la trayectoria',end3-start3, 's')
print('El metodo de Rungekutta4 tarda en calcular la trayectoria',end4-start4, 's')

figure (1)
plt.plot(x_jup_V, y_jup_V,'black',label='Verlet Velocity Jupiter')

plt.plot(3.353,0,'+m',linewidth=50,label='Afelio de Juno')
plt.plot(-1.988,0,'+g',linewidth=50,label='Perielio de Juno')
plt.plot(x_ast1_EC,y_ast1_EC,'b', label='Euler Cromer ')
plt.plot(x_ast1_V, y_ast1_V,'r',label='Verlet Velocity')
plt.xlabel('x(AU)')
plt.ylabel('y(AU)')

plt.legend( loc = 'upper right')
show()
```

Figura 43: Código para comparar los distintos métodos al calcular la órbita de Juno en un intervalo de tiempo de 2000 años. Importamos las funciones que vamos a emplear de librerías conocidas y de las que hemos creado nosotros. A continuación, establecemos los parámetros iniciales de Júpiter y Juno y llamamos a las funciones que contienen los métodos de integración. Asimismo, con la función timer () calculamos el tiempo que tarda en realizar el cálculo cada método.

```
#%% Comparacion de los metodos en el afelio

n=int(tend/deltat)

#Convertimos en vectores las listas de Verlet
x1=np.array(x_ast1_V,float)
y1=np.array(y_ast1_V,float)
r_afel=(x1**2+y1**2)**(1/2)

#Convertimos en vectores las listas de Euler Cromer
x2=np.array(x_ast1_EC,float)
y2=np.array(y_ast1_EC,float)
r_afel2=(x2**2+y2**2)**(1/2)
#Convertimos en vectores las listas de RK4
x3=np.array(x_ast1_RK4,float)
y3=np.array(y_ast1_RK4,float)
r_afel3=(x3**2+y3**2)**(1/2)

#Hacemos el bucle para hallar el afelio
rmaxV=[]
rmaxEC=[]
rmaxRK4=[]

for i in range(0,n):
    if r_afel[i]>r_afel[i-1] and r_afel[i]>r_afel[i+1]:
       rmaxV.append(r_afel[i])
    if r_afel2[i]>r_afel2[i-1] and r_afel2[i]>r_afel2[i+1]:
       rmaxEC.append(r_afel2[i])
    if r_afel3[i]>r_afel3[i-1] and r_afel3[i]>r_afel3[i+1]:
       rmaxRK4.append(r_afel3[i])

#Los convertimos en vectores  para poder aplicar las funciones mean y stdev
rV=np.array(rmaxV,float)
rEC=np.array(rmaxEC,float)
rRK4=np.array(rmaxRK4,float)

#Calculamos las desviaciones
print('La media con Verlet es=',rV.mean())
print('La desviacion respecto al valor real',stats.stdev(rV))
print('La media con Euler Cromer es=',rEC.mean())
print('La desviacion respecto al valor real',stats.stdev(rEC))

print('La media con RK4 es=',rRK4.mean())
print('La desviacion respecto al valor real',stats.stdev(rRK4))
```

```
#Pintamos los valores en una recta

a=[stats.stdev(rV),stats.stdev(rV)]
b=[stats.stdev(rEC),stats.stdev(rEC)]
c=[stats.stdev(rRK4),stats.stdev(rRK4)]

d=[0,1]
figure(2)
plt.plot( d,b,'g', label='Euler Cromer ')
plt.plot(d,a,'y', label='Verlet Velocity')
plt.plot( d,c,'r', label='RungeK_4')
plt.ylabel('Desviacion(UA)')
plt.plot(d,[0,0],'b',label='Valor observado')

plt.legend(loc='center')
show()
```

Figura 44: Parte del código para comparar los distintos métodos al calcular la órbita de Juno en un intervalo de tiempo de 2000 años. Las listas con las posiciones obtenidas con los distintos métodos las hemos convertido en vectores y hemos calculado la desviación cuadrática media de los valores

```
158 #%% Energias
159 M_ast=2.67*10**19
160
161 Ecin_ast_V,Egrav_ast_V,Emec_ast_V= energy_mecanica(x_ast1_V, y_ast1_V, z_ast1_V,\
162                 vx_ast1_V, vy_ast1_V, vz_ast1_V,x_jup_V, y_jup_V, z_jup_V, M_ast)
163 figure(3)
164
165 plt.plot(t_jup_V,Emec_ast_V,'red',label='Mecanica')
166 plt.plot(t_jup_V,Ecin_ast_V,'g',label='Cinetica ')
167 plt.plot(t_jup_V,Egrav_ast_V,'b',label='Gravitatoria ')
168 plt.xlabel('Time(year)')
169 plt.ylabel('Energia mecanica')
170 title('Verlet')
171 plt.legend(loc='upper right')
172
173 show()
```

Figura 45: Parte del código para comparar los distintos métodos al calcular la órbita de Juno en un intervalo de tiempo de 2000 años. Hemos calculado la energía del asteroide a lo largo de la trayectoria, para comprobar que la energía mecánica se conserva

5. Órbitas de los asteroides en las proximidades de las Zanjas de Kirkwood

5.1 Zanja en resonancia 3:1

```
import numpy as np
from math import sin,cos,pi,sqrt,exp,fabs
import matplotlib.pyplot as plt
from pylab import plot,title,xlabel,ylabel,legend,figure,show,subplot
from timeit import default_timer as timer
import statistics as stats

from Funciones_gravedad import grav_sun,grav_sun_Jupiter
from Metodo_Verlet import Verlet, Verlet2

#Definimos constantes
tend=1000
deltat=0.01         #year
M_jup= 1.898e27                  #kg
Msun=1.989e30
#Parametros iniciales Jupiter
y0_jup=0
z0_jup=0
exce_jupiter=0.048            # poner 0 para el caso circular
x0_jup= (1+exce_jupiter) * 5.20     #Distancia al fooco
r0_jup=[x0_jup , 0, 0]
#Velocidades
vx0_jup=0
vy0_jup= sqrt(4 * pi ** 2 * (1-exce_jupiter) / x0_jup * (1+M_jup/Msun))
vz0_jup=0
v0_jup=[vx0_jup, vy0_jup, vz0_jup]

#Asteoride 1
#Posicion
x0_ast1=2.30
y0_ast1=0
z0_ast1=0
r0_ast1=[x0_ast1 , 0, 0]

#Velocidades
vx0_ast1=0
vy0_ast1=sqrt(4 * pi ** 2 / abs(x0_ast1 ))  #AU/year
vz0_ast1=0
v0 ast1=[vx0 ast1, vy0 ast1, vz0 ast1]
```

```
#Asteoride 2
#Posicion
x0_ast2=2.4
y0_ast2=0
z0_ast2=0
r0_ast2=[x0_ast2 , 0, 0]

#Velocidades
vx0_ast2=0
vy0_ast2=sqrt(4 * pi ** 2 / abs(x0_ast2 ))   #AU/year
vz0_ast2=0
v0_ast2=[vx0_ast2, vy0_ast2, vz0_ast2]

#Asteoride 3
#Posicion
x0_ast3=2.5
y0_ast3=0
z0_ast3=0
r0_ast3=[x0_ast3 , 0, 0]

#Velocidades
vx0_ast3=0
vy0_ast3=sqrt(4 * pi ** 2 / abs(x0_ast3) )  #AU/year
vz0_ast3=0
v0_ast3=[vx0_ast3, vy0_ast3, vz0_ast3]

t_jup_V, x_jup_V, y_jup_V, z_jup_V, vx_jup_V, vy_jup_V, vz_jup_V = Verlet (grav_sun , r0_jup, v0_jup, deltat,tend)


#Calculo las trayectorias de los 3 asteorides
t_ast1_V, x_ast1_V, y_ast1_V, z_ast1_V, vx_ast1_V, vy_ast1_V, vz_ast1_V =\
 Verlet2 (grav_sun_Jupiter , r0_ast1, v0_ast1, deltat,tend,x_jup_V, y_jup_V, z_jup_V)
t_ast2_V, x_ast2_V, y_ast2_V, z_ast2_V, vx_ast2_V, vy_ast2_V, vz_ast2_V = \
 Verlet2 (grav_sun_Jupiter , r0_ast2, v0_ast2, deltat,tend,x_jup_V, y_jup_V, z_jup_V)
t_ast3_V, x_ast3_V, y_ast3_V, z_ast3_V, vx_ast3_V, vy_ast3_V, vz_ast3_V =\
 Verlet2 (grav_sun_Jupiter , r0_ast3, v0_ast3, deltat,tend,x_jup_V, y_jup_V, z_jup_V)
```

```
#%%Pintamos 12 orbitas de los asteorides,con puntos para que se vea bien la diferencia
y1a=[]
x1a=[]
x2a=[]
y2a=[]
x3a=[]
y3a=[]
for i in range (12):
    x1a.append(x_ast1_V[8100*i:8100*i+350])
    y1a.append(y_ast1_V[8100*i:8100*i+350])
    x2a.append(x_ast2_V[8000*i:8000*i+370])
    y2a.append(y_ast2_V[8000*i:8000*i+370])
    x3a.append(x_ast3_V[7920*i:7920*i+396])
    y3a.append(y_ast3_V[7920*i:7920*i+396])

figure(1)
plt.plot(x_jup_V, y_jup_V,'black',label='Verlet Velocity Jupiter')
plt.plot(x1a,y1a,'r.')
plt.plot(x2a,y2a,'b.')
plt.plot(x3a,y3a,'g.')
plt.xlabel('x(AU)')
plt.ylabel('y(AU)')
```

Figura 44: Código con el que calculamos las órbitas de 3 asteroides cerca de la Zanja de Kirkwood que se encuentra en resonancia 3:1 con Júpiter en un intervalo de tiempo de 1000 años. Además, hemos representado unas 12 orbitas de cada asteroide y las distancias al Sol. Para la zanja 2:1 simplemente hay que cambiar los parámetros iniciales

```
#%% Distancia al Sol con el paso del Tiempo
x1=np.array(x_ast1_V,float)
y1=np.array(y_ast1_V,float)
r_sun1=(x1**2+y1**2)**(1/2)

x2=np.array(x_ast2_V,float)
y2=np.array(y_ast2_V,float)
r_sun2=(x2**2+y2**2)**(1/2)
x3=np.array(x_ast3_V,float)
y3=np.array(y_ast3_V,float)
r_sun3=(x3**2+y3**2)**(1/2)

xj=np.array(x_jup_V,float)
yj=np.array(y_jup_V,float)
r_sunj=(xj**2+yj**2)**(1/2)

#Distancia al Sol de los tres asteorides y Jupiter
figure(2)
plt.plot(t_jup_V,r_sunj,'black',label='jup')
plt.plot(t_jup_V,r_sun1,'r',label='Ast.1')
plt.plot(t_ast2_V,r_sun2,'b',label='Ast.2')
plt.plot(t_ast3_V,r_sun3,'g',label='Ast.3')
plt.ylabel('Distancia Sol (AU)')
plt.xlabel('Tiempo (años)')
plt.legend( loc = 'upper right')

#Orbita de los 3 asteroides
figure(3)
plt.plot(x_jup_V, y_jup_V,'black',label='Verlet Velocity Jupiter')
plt.plot(x_ast1_V, y_ast1_V,'r',label='Asteoride 1')
plt.plot(x_ast2_V, y_ast2_V,'b',label='Asteoride 2')
plt.plot(x_ast3_V, y_ast3_V,'g',label='Asteoride 3')
plt.xlabel('x(AU)')
plt.ylabel('y(AU)')

plt.legend( loc = 'upper right')
show()
```

Figura 45: Código para calcular el ancho de las Zanjas de Kirkwood. Hemos estudiado la posición de los asteroides a lo largo de 1000 años, comparando su valor inicial con su valor final. Variando el "arange" del bucle "for" ponemos la zona que queremos estudiar. Posteriormente hemos guardado los datos para representarlos con el programa: Origin

6. Ancho Zanjas de Kirkwood

```
import random
import numpy as np
from math import sin,cos,pi,sqrt,exp,fabs
import matplotlib.pyplot as plt
from pylab import figure
from Funciones import grav_sun,grav_sun_Jupiter
from Metodo_Verlet import Verlet, Verlet2
from scipy import asarray as ar,exp

#Definimos constantes
tend    =  1000
deltat  =  0.01                         #year
M_jup   =  1.898e27                      #kg
Msun    =  1.989e30

#Parametros iniciales Jupiter

exce_jupiter=0.048   #Caso eliptico, para caso circular poner excentricidad= 0
x0_jup= (1+exce_jupiter) * 5.20     #Distancia al fooco
r0_jup=[x0_jup , 0, 0]

#Velocidades
vy0_jup= sqrt(4 * pi ** 2 * (1-exce_jupiter) / x0_jup * (1+M_jup/Msun))
v0_jup=[0, vy0_jup, 0]

#Calculo la trayectoria de Jupiter
t_jup , x_jup , y_jup , z_jup , vx_jup , vy_jup , vz_jup  =\
 Verlet (grav_sun , r0_jup, v0_jup, deltat,tend)

distancias=[]
semieje=[]
Asteroides=open('Asteorides 2.2 3.45 1000 años 0.001 Jupiter eliptico.txt','w')
```

```
for radius in np.arange(2.2,3.45,0.001):

    V_i = sqrt(4 * pi ** 2 / (radius))

    Init_pos = [radius, 0, 0]
    V_init = [0, V_i, 0]

    t_ast, x_ast, y_ast, z_ast, vx_ast, vy_ast, vz_ast =\
    Verlet2 (grav_sun_Jupiter , Init_pos , V_init, deltat,\
             tend,x_jup , y_jup , z_jup )

    x_ast = np.array(x_ast,float)
    y_ast = np.array(y_ast,float)
    r_ast = ( x_ast**2 + y_ast**2 ) ** (1/2)
    desviacion=abs(max(r_ast)-min(r_ast))

    distancias.append(desviacion)
    semieje.append(radius)
    print(semieje)
    Asteroides.write(str(desviacion))
    Asteroides.write(' ')
    Asteroides.write(str(radius))
    Asteroides.write('\n')


Asteroides.close()
```

Figura 46: Código para calcular el ancho de las Zanjas de Kirkwood. Hemos estudiado la posición de los asteroides a lo largo de 1000 años, comparando su valor inicial con su valor final. Variando el "arange" del bucle "for" ponemos la zona que queremos estudiar. Posteriormente hemos guardado los datos para representarlos con el programa: Origin

7. Influencia de otros planetas en los asteroides.

7.1 Influencia de Marte

```
import random
import numpy as np
from math import sin,cos,pi,sqrt,exp,fabs
import matplotlib.pyplot as plt
from pylab import figure

from Funciones_gravedad import grav_sun,grav_sun_Jupiter,grav_sun_Jupiter_Marte
from Metodo_Verlet import Verlet, Verlet2,Verlet3

#Definimos constantes
tend    =   100000
deltat  =   0.01                        #year
M_jup   =   1.898e27                    #kg
Msun    =   1.989e30
M_mar   =   6.415E23
#Parametros iniciales Jupiter

exce_jupiter = 0.048
x0_jup = (1+exce_jupiter) * 5.20     #Distancia al foco
r0_jup = [x0_jup , 0, 0]
#Velocidades
vy0_jup= sqrt(4 * pi ** 2 * (1-exce_jupiter) / x0_jup * (1+M_jup/Msun))
v0_jup=[0, vy0_jup, 0]

#Parametros iniciales Marte
exce_mar=0.093
x0_mar= (1+exce_mar) * 1.52     #Distancia al perihelio
r0_mar=[x0_mar , 0, 0]
#Velocidades
vy0_mar= sqrt(4 * pi ** 2 * (1-exce_mar) / x0_mar * (1+M_mar/Msun))
v0_mar=[0, vy0_mar, 0]

#Calculo primero las posiciones de Jupiter
t_jup, x_jup, y_jup, z_jup, vx_jup, vy_jup, vz_jup =\
Verlet (grav_sun , r0_jup, v0_jup, deltat,tend)

#Calculo las posiciones de Marte
t_mar, x_mar, y_mar, z_mar, vx_mar, vy_mar, vz_mar = \
Verlet2 (grav_sun_Jupiter ,r0_mar, v0_mar, deltat,tend,x_jup, y_jup, z_jup)
```

```
distancias=[]
semieje=[]
Asteroides=open('Asteorides Marte 100000 0.002.txt','w')

for a in np.arange(2.49,2.53,0.002):

    radius = a
    V_i = sqrt(4 * pi ** 2 / (radius))

    Init_pos = [radius, 0, 0]
    V_init = [0, V_i, 0]

    #Calculo las posiciones del asteroide
    t_ast, x_ast, y_ast, z_ast, vx_ast, vy_ast, vz_ast =\
    Verlet3 (grav_sun_Jupiter_Marte , Init_pos, V_init, deltat,tend,x_jup,\
             y_jup, z_jup,x_mar,y_mar,z_mar)

    x_ast = np.array(x_ast,float)
    y_ast = np.array(y_ast,float)
    r_ast = ( x_ast**2 + y_ast**2 ) ** (1/2)

    desviacion=abs(max(r_ast)-min(r_ast))
    distancias.append(desviacion)
    semieje.append(radius)
    Asteroides.write(str(desviacion))
    Asteroides.write(' ')
    Asteroides.write(str(radius))
    Asteroides.write('\n')


Asteroides.close()
```

Figura 47: Código para calcular la influencia de Marte en los asteroides de la zanja que se encuentra en resonancia 3:1, en un intervalo de tiempo de 100000 años. Establecemos los parámetros iniciales de Júpiter y Marte y calculamos sus órbitas. Teniendo en cuenta el Sol y estos planetas comparamos la posición inicial del asteroide y su posición final. Posteriormente hemos guardado los datos para representarlos con el programa: Origin

7.2 Influencia de Saturno

```
import random
import numpy as np
from math import sin,cos,pi,sqrt,exp,fabs
import matplotlib.pyplot as plt
from pylab import figure

from Funciones_gravedad import grav_sun,grav_sun_Jupiter,grav_sun_Jupiter_Marte
from Metodo_Verlet import Verlet, Verlet2,Verlet3

#Definimos constantes
tend = 100000
deltat = 0.01                          #year
M_jup = 1.898e27                       #kg
Msun = 1.989e30
M_sat= 5.683E26

#Parametros iniciales Jupiter

exce_jupiter = 0.048
x0_jup = (1+exce_jupiter) * 5.20      #Distancia al afelio
r0_jup = [x0_jup , 0, 0]
#Velocidades
vy0_jup = sqrt(4 * pi ** 2 * (1-exce_jupiter) / x0_jup * (1+M_jup/Msun))
v0_jup = [0, vy0_jup, 0]

#Parametros iniciales Saturno

exce_sat=0.056
x0_sat= (1+exce_sat) * 9.582     #Distancia afelio
r0_sat=[x0_sat , 0, 0]
#Velocidades
vx0_sat=0
vy0_sat= sqrt(4 * pi ** 2 * (1-exce_sat) / x0_sat * (1+M_sat/Msun))
vz0_sat=0
v0_sat=[vx0_sat, vy0_sat, 0]
```

```
#Calculo las posiciones de Saturno
t_sat, x_sat, y_sat, z_sat, vx_sat, vy_sat, vz_sat = \
Verlet2 (grav_sun_Jupiter ,r0_sat, v0_sat, deltat,tend,x_jup, y_jup, z_jup)

distancias=[]
semieje=[]
Asteroides=open('Asteorides Saturno 3.26 3.45 100000 0.005 .txt','w')

for a in np.arange(3.26,3.43,0.005):

    radius = a
    V_i = sqrt(4 * pi ** 2 / (radius))

    Init_pos = [radius, 0, 0]
    V_init = [0, V_i, 0]

    #Calculo las posiciones del asteroide
    t_ast, x_ast, y_ast, z_ast, vx_ast, vy_ast, vz_ast =\
    Verlet3 (grav_sun_Jupiter_Marte , Init_pos, V_init, deltat,tend,\
             x_jup, y_jup,z_jup,x_sat,y_sat,z_sat)

    x_ast = np.array(x_ast,float)
    y_ast = np.array(y_ast,float)
    r_ast = ( x_ast**2 + y_ast**2 ) ** (1/2)

    desviacion=abs(max(r_ast)-min(r_ast))
    distancias.append(desviacion)
    semieje.append(radius)
    print(semieje)
    Asteroides.write(str(desviacion))
    Asteroides.write(' ')
    Asteroides.write(str(radius))
    Asteroides.write('\n')


Asteroides.close()
```

Figura 48: Código para calcular la influencia de Saturno en los asteroides que se encuentran en la zanja en resonancia 3:1, en un intervalo de tiempo de 100000 años. Establecemos los parámetros iniciales de Júpiter y Saturno y calculamos sus órbitas. Teniendo en cuenta el Sol y estos planetas comparamos la posición inicial del asteroide y su posición final. Posteriormente hemos guardado los datos para representarlos con el programa: Origin