UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
<75.12> ANÁLISIS NUMÉRICO

DATOS DEL TRABAJO PRÁCTICO

1	2016	
	AÑO	
	1er	
TP NRO	CUAT	TEMA

INTEGRANTES DEL GRUPO

12	CAVAIUOLO ADRIAN	91186
	APELLIDO Y NOMBRE	PADRÓN
	TULIPANI GASTON	96570
GRUPO	APELLIDO Y NOMBRE	PADRÓN

DATOS DE LA ENTREGA

.TXT	-	04/05/16	04/05/16
ARCHIVO	NRO CONTROL	FECHA VENC	FECHA ENTR

CORRECCIONES

FECHA	NOTA	OBSERVACIONES
DOCENTE	FIRMA	

Aclaraciones Iniciales

En primer lugar, se consideró correcto comenzar el Informe del TP2 describiendo el lenguaje de programación utilizado y las características técnicas de la PC utilizada para las corridas del programa.

- Lenguaje: C#
- Procesador: Intel® Core™ i7-4510 CPU
- Velocidad del Reloj: 2.00GHz
- Cantidad de Núcleos: 2 principales 4 lógicos
- Sistema Operativo: Windows 10 Home

Además, el código y los archivos con los resultados completos se encuentran subidos al repositorio: http://github.com/gtulipani/TP2Numerico

A) Análisis Clásico de órbitas: Leyes de Kepler

A.1) Primera Ley de Kepler (Euler)

La primera ley de Kepler se encuentra constituida por el siguiente enunciado: "*Todos los planetas se desplazan alrededor del Sol describiendo órbitas elípticas. El Sol se encuentra en uno de los focos de la elipse*". Para llevar a cabo la comprobación de dicha ley, se requirió calcular los semiejes mayor y menor de la órbita de **Mustafar** para N crecientes. El semieje menor se calculó tomando la distancia r entre el Sol y el planeta cuando se encuentra a un cuarto de órbita, y se aplicó Pitágoras junto con la distancia del centro al foco (a * e), mientras que para el semieje mayor se tomó la distancia r a la mitad de la órbita y se realizó una simple diferencia con la distancia del centro al foco. Se volcaron los valores en la siguiente tabla:

N	A_N [m]	ΔA_N [m]	B_N [m]	ΔB_N [m]
8	1,03E+11	1,03E+11	5,85E+10	5,85E+10
16	8,40E+10	1,89E+10	6,17E+10	3,27E+09
32	7,39E+10	1,01E+10	6,27E+10	9,28E+08
64	6,99E+10	3,99E+09	6,29E+10	2,19E+08
128	6,82E+10	1,71E+09	6,29E+10	5,15E+07
256	6,74E+10	7,90E+08	6,29E+10	1,24E+07
512	6,70E+10	3,79E+08	6,29E+10	3051520
1024	6,69E+10	1,85E+08	6,29E+10	753664
2048	6,68E+10	9,16E+07	6,29E+10	135168
4096	6,67E+10	4,57E+07	6,29E+10	90112
8192	6,67E+10	2,28E+07	6,29E+10	77824
16384	6,67E+10	1,13E+07	6,29E+10	16384
32768	6,67E+10	5734400	6,29E+10	225280
65536	6,67E+10	2875392	6,29E+10	393216
131072	6,67E+10	1064960	6,29E+10	4714496
262144	6,67E+10	1007616	6,29E+10	1781760
524288	6,67E+10	2457600	6,30E+10	1,04E+07

1048576	6,67E+10	2199552	6,29E+10	5,17E+07
2097152	6,67E+10	4993024	6,29E+10	4,32E+07

Conclusiones

Los valores encontrados para el semieje mayor son los esperados, ya que el "a" utilizado para las operaciones, se encuentra definido, de acuerdo al TP1 cómo:

$$\lambda = \frac{NP}{9.10^5} = \frac{96570}{90000} = 1.073$$

$$a = \lambda^2 5.791\ 10^{10}\ m = 1.073^2 5.791\ 10^{10}\ m = 6.667\ 10^{10}\ m$$

A.2) Segunda Ley de Kepler (Euler)

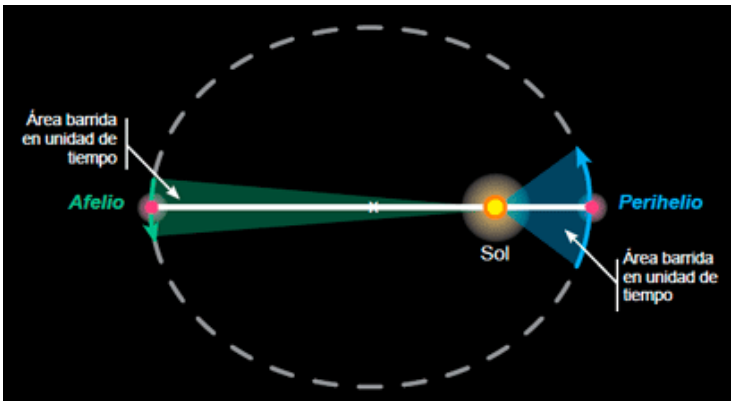
La segunda ley de Kepler se encuentra constituida por el siguiente enunciado: "*El radio vector que une un planeta y el Sol barre áreas iguales en tiempos iguales*". Este enunciado también es equivalente a afirmar que el momento angular específico de la órbita se mantiene constante, con lo cuál queda expresado mediante la siguiente ecuación diferencial:

$$\frac{dA}{dt} = \frac{1}{2}h$$

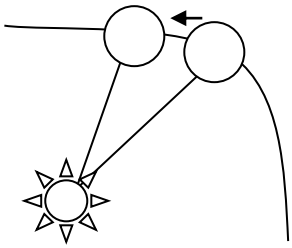
La forma de verificar esta ley, es hacer una tabla con el período T de la órbita **Mustafar**, para N crecientes. Si operamos la ecuación diferencial anterior nos queda lo siguiente:

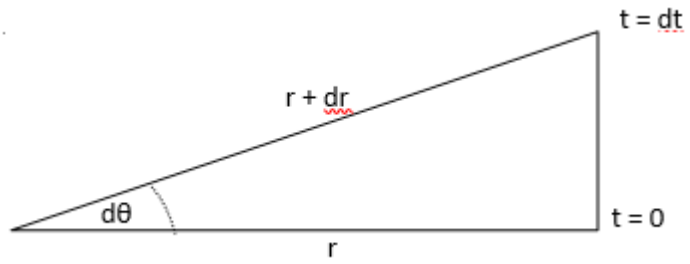
$$dA = dt\frac{1}{2}h \rightarrow A = \int_0^A dA = \frac{1}{2}h\int_0^T dt \rightarrow A = \frac{1}{2}h\,T \rightarrow T = \frac{2A}{h}$$

Es decir, para calcular el período solamente necesitamos calcular el área. Si tomamos la órbita de la siguiente forma:



Y si observamos con mayor detalle el movimiento de la masa planetario respecto a la masa en el foco:





Se dibujó un triángulo rectángulo, bajo la hipótesis que el movimiento genera una nueva distancia r + dr, tal que la diferencia entre ellos es mínima, con lo cuál se puede tomar un ángulo recto. Bajo lo mismo, tomamos que dr es prácticamente 0, y dθ al ser tan pequeño, sen(dθ) ≈ dθ.

$$dA = \frac{1}{2}r(r+dr)\sin d\theta = \frac{1}{2}r^2 d\theta \rightarrow A = \int_0^A dA = \frac{1}{2}\int_0^{2\pi} r^2 d\theta$$

Como vemos, para calcular el área hay que aplicar algún método de Integración Numérica visto en la clase. Para el presente TP, se decidió utilizar el método del *Trapecio* (orden 2), y sobre los resultados aplicar la *Extrapolación de Richardson* (teniendo en cuenta que el N se va duplicando), presentando así un resultado con orden 4 (la extrapolación de Richardson aumenta de a 2 el orden en el caso de la integración). Se presentan los resultados en la presente tabla:

N	T_N [s]	ΔT_N [s]
8	2,10E+07	2,10E+07
16	7404333	1,36E+07
32	1,03E+07	2898268
64	8918903	1383698
128	9231497	312594
256	9064470	167027
512	9091241	26771
1024	9068426	22815
2048	9069259	833
4096	9065609	3650
8192	9065147	462
16384	9064470	677
32768	9064240	230
65536	9063989	309
131072	9065048	1059
262144	9063795	1253
524288	9064473	678
1048576	9077518	13045
2097152	9036165	41353

Conclusiones

Cómo podemos observar, los cálculos efectuados tienen sentido, ya que el período converge y el área podemos verificarla utilizando la fórmula para una elipse cualquiera:

$$A_{elipse} = \pi ab = \pi\, 6.67\ 10^{10} 6.29\ 10^{10} = 1.318033207\ 10^{22}$$

$$h = \sqrt{h^2} = \sqrt{aGM(1-e^2)} = \sqrt{6.67\ 10^{10} 6.673\ 10^{-11} 2.134304\ 10^{30}(1-0.191612^2)} = 3.024423\ 10^{15}$$

$$T = \frac{2A}{h} = \frac{2\ 1.318033\ 10^{22}}{3.024423\ 10^{15}} = 9000000$$

A.3) Tercera Ley de Kepler (Euler)

La tercera ley de Kepler se encuentra constituida por el siguiente enunciado: "*Para cualquier planeta, el cuadrado de su período orbital es directamente proporcional al cubo de la longitud del semieje mayor de su órbita elíptica*", quedando determinada la siguiente ecuación:

$$\frac{T^2}{r^3} = c$$

La constante "c" es también llamada constante de Kepler, T es el período de la órbita y r es el semieje mayor calculado de la órbita. Para la verificación de la misma se utilizará la siguiente ecuación equivalente:

$$\frac{{T^2}_N}{\langle R_N \rangle^3}$$

Se volcaron los resultados en la siguiente tabla:

N	T_N^2 [s^2]	R_N^3 [m^3]	T_N^2 / R_N^3 [s^2 / m^3]	$\Delta T_N^2 / R_N^3$ [s^2 / m^3]
8	4,40E+14	1,09E+33	4,05E-19	4,05E-19
16	5,48E+13	5,93E+32	9,25E-20	3,12E-19
32	1,06E+14	4,04E+32	2,63E-19	1,70E-19
64	7,95E+13	3,42E+32	2,33E-19	3,02E-20
128	8,52E+13	3,17E+32	2,69E-19	3,59E-20
256	8,22E+13	3,06E+32	2,68E-19	4,24E-22
512	8,27E+13	3,01E+32	2,74E-19	6,18E-21
1024	8,22E+13	2,99E+32	2,75E-19	9,02E-22
2048	8,23E+13	2,98E+32	2,76E-19	1,18E-21
4096	8,22E+13	2,97E+32	2,77E-19	3,46E-22
8192	8,22E+13	2,97E+32	2,77E-19	2,55E-22
16384	8,22E+13	2,97E+32	2,77E-19	9,91E-23
32768	8,22E+13	2,96E+32	2,77E-19	5,74E-23
65536	8,22E+13	2,96E+32	2,77E-19	1,70E-23
131072	8,22E+13	2,96E+32	2,77E-19	7,80E-23

262144	8,22E+13	2,96E+32	2,77E-19	6,41E-23
524288	8,22E+13	2,96E+32	2,77E-19	1,08E-23
1048576	8,24E+13	2,96E+32	2,78E-19	7,71E-22
2097152	8,17E+13	2,96E+32	2,75E-19	2,46E-21

Conclusiones

De acuerdo a la ecuación diferencial que propone Newton, se puede calcular analíticamente la constante de Kepler a través del siguiente razonamiento.

La fuerza gravitacional crea la aceleración centrípeta necesaria para el movimiento circular:

$$\frac{GMm}{r^2} = m\frac{v^2}{r}$$

El cuadrado del tiempo de una órbita completa o periodo es:

$$T^2 = \frac{4\pi^2}{GM}r^3$$

Entonces:

$$\frac{T^2}{r^3} = \frac{4\pi^2}{GM} = C$$

Podemos ver en el ejemplo actual que los valores encontrados responden a la fórmula:

$$\frac{T^2}{r^3} = \frac{4\pi^2}{GM} = \frac{4\pi^2}{6.673\ 10^{-11}\ 2.13430459\ 10^{30}} = 2.7719297\ 10^{-19}$$

A.4) Conservación de la Energía (Euler)

El cuarto punto del TP consiste en calcular la energía en cada punto de la órbita, para N crecientes, y analizar si la Energía efectivamente se conserva (sino estaríamos "inventando" energía). La fórmula provista por la cátedra para el cálculo de la misma es la siguiente:

$$E_n = \frac{1}{2}{v_n}^2 - \mu u_n$$

Dónde la velocidad al cuadrado se calcula a través de:

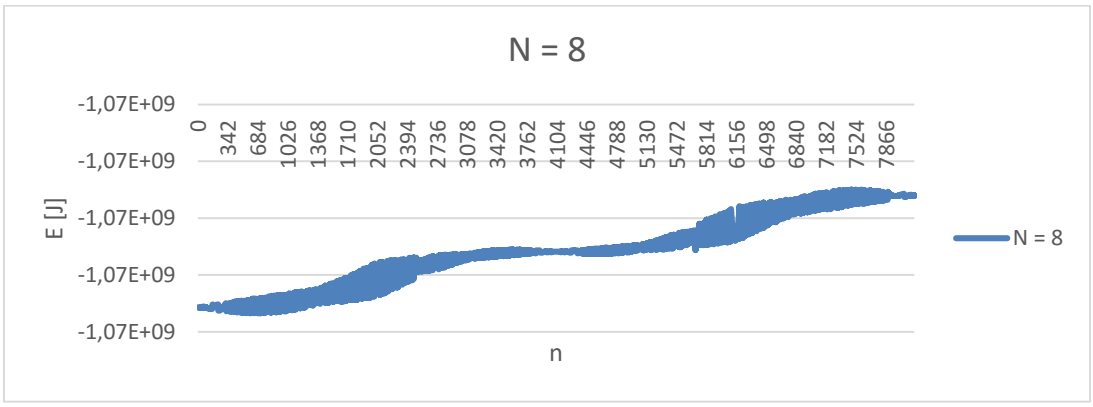
$${v_n}^2 = h^2\left[{u_n}^2 + \left(\frac{du}{d\theta}\right)^2\right]$$

Es decir que, para calcular la energía en cada punto de la órbita, hay que calcular la derivada en cada uno de ellos. Para ello, en el punto inicial y final de la órbita (el perihelio al principio y luego del cálculo), se utilizó la *derivada en adelanto* y *derivada en atraso*, respectivamente, mientras que para los puntos intermedios se utilizó la *derivada centrada*.

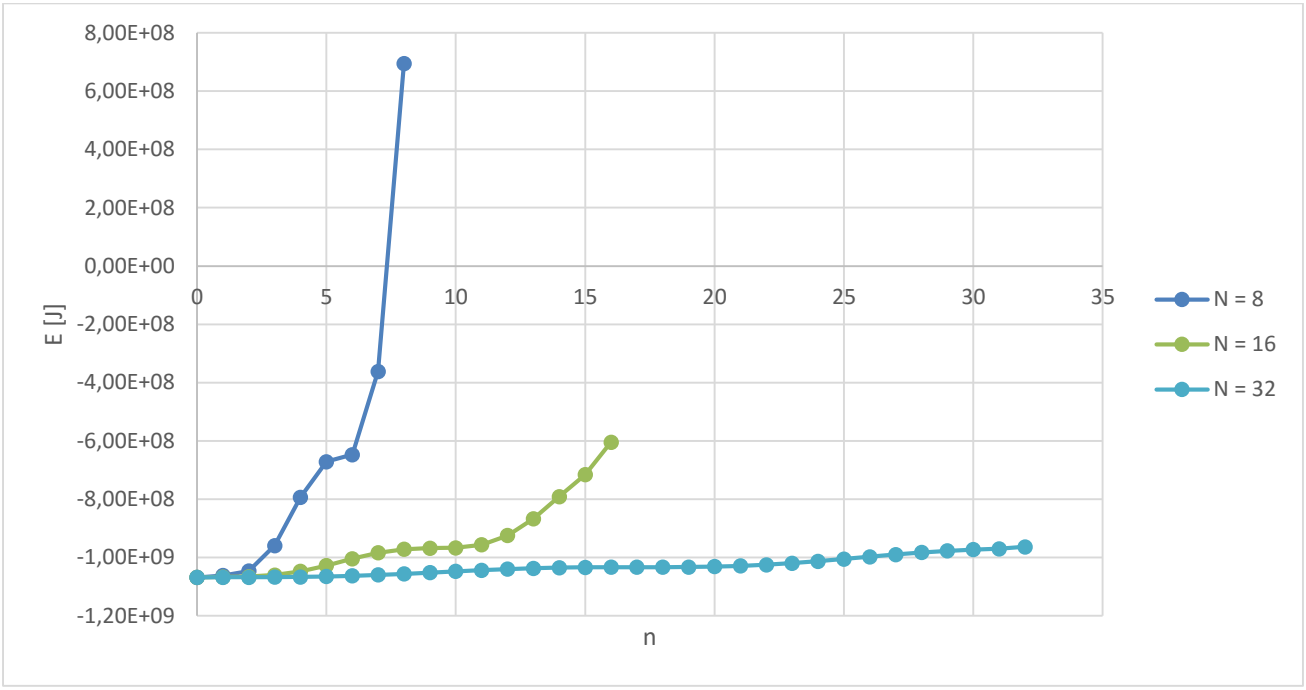
A continuación, se vuelcan algunos datos para un N considerablemente grande probado (8192):

n	u_n [1 / m]	E_n [J]
0	1,86E-11	-1068057000
1	1,86E-11	-1068057000
2	1,86E-11	-1068057000
3	1,86E-11	-1068057000
4	1,86E-11	-1068057000
5	1,86E-11	-1068057000
6	1,86E-11	-1068057000
7	1,86E-11	-1068057000
8	1,86E-11	-1068057000
9	1,86E-11	-1068057000
10	1,86E-11	-1068057000
...	...	...
8183	1,86E-11	-1067861000
8184	1,86E-11	-1067861000
8185	1,86E-11	-1067861000
8186	1,86E-11	-1067860000
8187	1,86E-11	-1067861000
8188	1,86E-11	-1067861000
8189	1,86E-11	-1067861000
8190	1,86E-11	-1067860000
8191	1,86E-11	-1067861000
8192	1,86E-11	-1067860000

El gráfico correspondiente al N utilizado arriba es el siguiente:



Además, se superponen los gráficos para los 3 primeros N (a partir de 8 y con una tasa de 2):



Conclusiones

Cómo podemos observar, si bien la diferencia de energía al principio de la órbita y luego de dar la vuelta es muy pequeña (notar los valores del eje de las ordenadas), **es una diferencia**. Esto no significa que el algoritmo está mal implementado, o que la diferencia se debe al redondeo de la máquina, sino que el método de Euler es un método **no conservativo** (se debería usar en su lugar el *Método de Taylor*, *Método de Newmark* o el *Método de la Rayuela*).

Por otro lado, podemos notar que con el primer N utilizado (8), la energía en el perihelio da positiva (), y esta tendencia se va diluyendo con el aumento de N (ya con N = 32, es casi constante).

Además, podemos verificar la energía calculada a través de la fórmula para las órbitas elípticas:

$$\frac{v^2}{2} - \frac{\mu}{r} = -\frac{\mu}{2a} = \epsilon < 0$$

Por lo pronto, podemos verificar que es menor a 0, faltaría verificar si efectivamente se cumple con el valor:

$$-\frac{\mu}{2a} = -\frac{GM}{2a} = -\frac{6.673\ 10^{-11}\ 2.13430459\ 10^{30}}{2\ 6.667\ 10^{10}} = -1.068\ 10^{9}$$

A.5) Primera Ley de Kepler (RK4)

Teniendo en cuenta las mismas consideraciones que en el punto *A.1)*, se vuelcan los resultados utilizando el método **Runge-Kutta de Orden 4**:

N	A_N [m]	ΔA_N [m]	B_N [m]	ΔB_N [m]
8	6,66E+10	6,66E+10	6,29E+10	6,29E+10
16	6,67E+10	1,10E+08	6,29E+10	4,55E+07
32	6,67E+10	3657728	6,29E+10	3452928
64	6,67E+10	114688	6,29E+10	229376
128	6,67E+10	4096	6,29E+10	16384
256	6,67E+10	45056	6,29E+10	16384
512	6,67E+10	61440	6,29E+10	24576
1024	6,67E+10	36864	6,29E+10	8192
2048	6,67E+10	53248	6,29E+10	24576
4096	6,67E+10	45056	6,29E+10	73728
8192	6,67E+10	143360	6,29E+10	20480
16384	6,67E+10	212992	6,29E+10	36864
32768	6,67E+10	12288	6,29E+10	348160
65536	6,67E+10	147456	6,29E+10	434176
131072	6,67E+10	360448	6,29E+10	4710400
262144	6,67E+10	339968	6,29E+10	1802240
524288	6,67E+10	2842624	6,30E+10	1,03E+07
1048576	6,67E+10	2387968	6,29E+10	5,16E+07
2097152	6,67E+10	4894720	6,29E+10	4,32E+07

Conclusiones

Es notorio que se llegó a los mismos resultados con un N menor que con el método de Euler (lo cual implica menos cantidad de pasos y más rapidez en la ejecución).

A.6) Segunda Ley de Kepler (RK4)

Teniendo en cuenta las mismas consideraciones que en el punto *A.2)*, se vuelcan los resultados utilizando el método **Runge-Kutta de Orden 4**:

N	T_N [s]	ΔT_N [s]
8	9065073	9065073
16	9064135	938
32	9064013	122
64	9064015	2
128	9064015	0
256	9064018	3
512	9064014	4
1024	9064018	4
2048	9064015	3
4096	9064015	0
8192	9064006	9

16384	9064021	15
32768	9063969	52
65536	9063945	24
131072	9064949	1004
262144	9063772	1177
524288	9064444	672
1048576	9077501	13057
2097152	9036159	41342

Conclusiones

Es notorio que se llegó a los mismos resultados con un N menor que con el método de Euler (lo cual implica menos cantidad de pasos y más rapidez en la ejecución).

A.7) Tercera Ley de Kepler (RK4)

Teniendo en cuenta las mismas consideraciones que en el punto *A.3)*, se vuelcan los resultados utilizando el método **Runge-Kutta de Orden 4**:

N	T_N^2 [s^2]	R_N^3 [m^3]	T_N^2 / R_N^3 [s^2 / m^3]	Δ T_N^2 / R_N^3 [s^2 / m^3]
8	8,22E+13	2,95E+32	2,79E-19	2,79E-19
16	8,22E+13	2,96E+32	2,77E-19	1,43E-21
32	8,22E+13	2,96E+32	2,77E-19	5,31E-23
64	8,22E+13	2,96E+32	2,77E-19	1,34E-24
128	8,22E+13	2,96E+32	2,77E-19	2,58E-26
256	8,22E+13	2,96E+32	2,77E-19	3,62E-25
512	8,22E+13	2,96E+32	2,77E-19	4,91E-25
1024	8,22E+13	2,96E+32	2,77E-19	1,81E-25
2048	8,22E+13	2,96E+32	2,77E-19	4,39E-25
4096	8,22E+13	2,96E+32	2,77E-19	5,43E-25
8192	8,22E+13	2,96E+32	2,77E-19	1,24E-24
16384	8,22E+13	2,96E+32	2,77E-19	1,76E-24
32768	8,22E+13	2,96E+32	2,77E-19	3,31E-24
65536	8,22E+13	2,96E+32	2,77E-19	3,62E-25
131072	8,22E+13	2,96E+32	2,77E-19	5,69E-23
262144	8,22E+13	2,96E+32	2,77E-19	6,78E-23
524288	8,22E+13	2,96E+32	2,77E-19	5,64E-24
1048576	8,24E+13	2,96E+32	2,78E-19	7,69E-22
2097152	8,17E+13	2,96E+32	2,75E-19	2,47E-21

Conclusiones

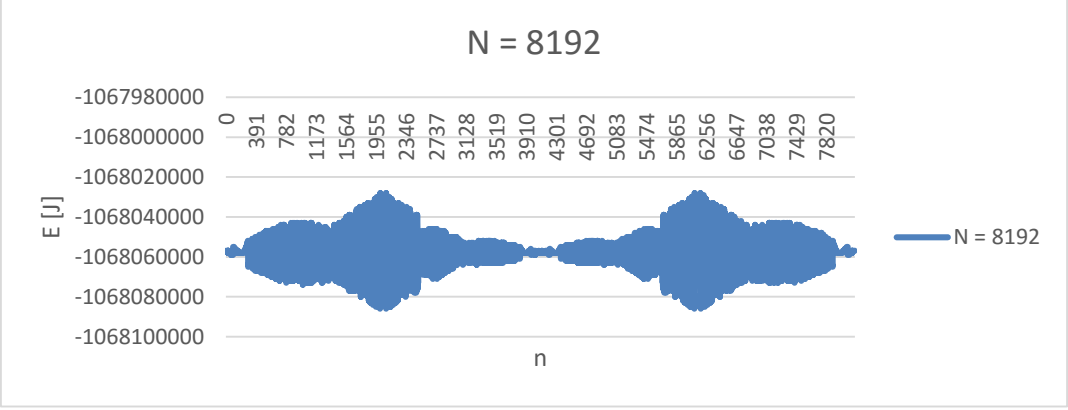
Es notorio que se llegó a los mismos resultados con un N menor que con el método de Euler (lo cual implica menos cantidad de pasos y más rapidez en la ejecución).

A.8) Conservación de la Energía (RK4)

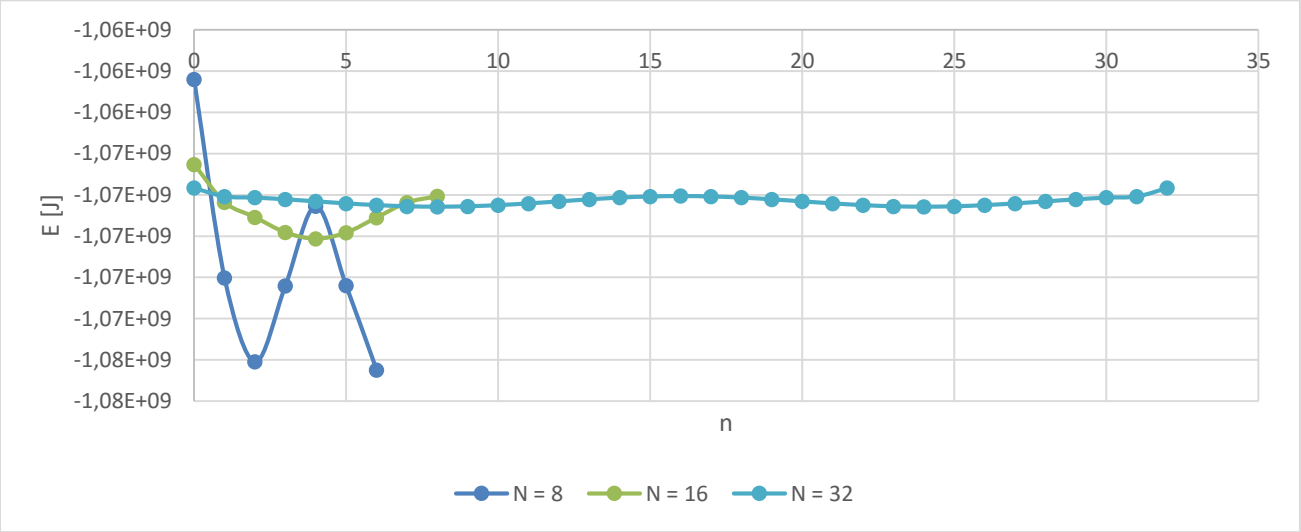
Teniendo en cuenta las mismas consideraciones que en el punto *A.4)*, se vuelcan algunos de los resultados utilizando el método **Runge-Kutta de Orden 4** para el máximo N probado:

n	u_n [1 / m]	E_n [J]
0	1,86E-11	-1068057000
1	1,86E-11	-1068057000
2	1,86E-11	-1068057000
3	1,86E-11	-1068057000
4	1,86E-11	-1068057000
5	1,86E-11	-1068057000
6	1,86E-11	-1068057000
7	1,86E-11	-1068057000
8	1,86E-11	-1068057000
9	1,86E-11	-1068057000
10	1,86E-11	-1068057000
...	...	...
8183	1,86E-11	-1068057000
8184	1,86E-11	-1068057000
8185	1,86E-11	-1068057000
8186	1,86E-11	-1068057000
8187	1,86E-11	-1068057000
8188	1,86E-11	-1068057000
8189	1,86E-11	-1068057000
8190	1,86E-11	-1068057000
8191	1,86E-11	-1068057000
8192	1,86E-11	-1068057000

El gráfico correspondiente al N utilizado arriba es el siguiente:



Además, se superponen los gráficos para los 3 primeros N (a partir de 8 y con una tasa de 2):



Conclusiones

Con el método utilizado (**Runge-Kutta Orden 4**) la diferencia entre el punto al principio de la órbita, y el punto al final es prácticamente nula (verificar Tabla). Sin embargo, al realizar el gráfico, es notorio que este valor **no es constante** a lo largo de toda la órbita, aunque oscila de forma distinta al Método de Euler (a la mitad de la órbita, el afelio y en el perihelio los valores son los mismos, mientras que los extremos se encuentran en el cuarto y tres cuartos de órbita). Nuevamente, esto se debe a que el método de Runge-Kutta Orden 4 es un método **no conservativo** (se debería usar en su lugar el *Método de Taylor*, *Método de Newmark* o el *Método de la Rayuela*).

Por otro lado, podemos notar que con el primer N utilizado (8), la energía en el perihelio no da positiva cómo en el método de Euler, y la oscilación se va diluyendo con el aumento de N (ya con N = 32, prácticamente no hay constante).

En definitiva, para el caso Newtoniano, el método Runge-Kutta Orden 4 parece mucho más indicado para la resolución.

B) Análisis Relativista de órbitas: Leyes de Einstein

B.1) Generalización del Método de Euler a la Relatividad y Precesión de Mercurio

La ecuación de movimiento del TP1 se generaliza en relatividad general de la siguiente manera:

$$\frac{d^2u}{d\theta^2} + u = \frac{\mu}{h^2} + 3\frac{GM}{c^2}u^2$$

$$\text{con } \theta = 0\,, \frac{du}{d\theta} = 0$$

Con lo cual, para generalizar el método de Euler, se hace un cambio de variable y se forma un sistema de dos ecuaciones con dos incógnitas.

$$\begin{cases} \dfrac{du}{d\theta} = v \\ \dfrac{dv}{d\theta} = -u + \dfrac{\mu}{h^2} + 3\dfrac{GM}{c^2}u^2 \end{cases}$$

El método de Euler explícito estima las derivadas de la siguiente forma:

$$\begin{cases} u_{n+1} = u_n + kv_n \\ v_{n+1} = v_n - ku_n + k\dfrac{\mu}{h^2} + 3k\dfrac{GM}{c^2}{u_n}^2 \end{cases}$$

Queda claro que el cambio que hay que hacer para adaptar el método de Euler es mínimo, pues únicamente se agrega un término en el cálculo de v_{n+1}.

Por lo tanto, el **Algoritmo 1-GR** (en el lenguaje C#) es el siguiente:

```
uN = u0 + k * v0;
vN = v0 - k * u0 + k * (GM / h2) + (k * 3f * GM * Pow(u0, 2) / c2);
```

Con los valores iniciales:

```
float u0 = 1 / (a * (1 - e));
float v0 = 0;
```

Al reducir el sistema a λ = 1, el sistema se reduce a Sol-Mercurio. Debe calcularse la precesión del sistema, que equivale a la diferencia angular de dos perihelios consecutivos. Lamentablemente, el resultado calculado es un ángulo mucho menor al esperado (44'' / siglo terrestre). Sin embargo, se mostrará a continuación el procedimiento realizado para el cálculo del mismo.

En la órbita relativista, al producirse una vuelta entera, el planeta no vuelve al mismo punto (cómo si pasaría en una órbita elíptica cerrada). El procedimiento consiste entonces en interpolar 3 puntos de la órbita y encontrar el mínimo (que corresponde con la mínima distancia al Sol – el nuevo *perihelio*).

Con **N = 1024**, se obtienen los siguientes valores:

- rAnteultimo = 4.58552975E+10
- rFinal = 4.585501E+10.
- rExtra = 4.58550067E+10.

rAnteultimo se calcula haciendo 1 / u_{8191}, rFinal se calcula haciendo 1 / u_{8192} y rExtra haciendo 1 / u_{8193}.
Si observamos los valores, la mínima distancia parece encontrarse entre rFinal y rExtra (lo cuál tiene sentido, ya que el perihelio debería correrse).

Para encontrar este mínimo, debemos usar algún método de Interpolación visto en clase, para ello usamos el método de *Interpolación de Newton* a través de *diferencias divididas*. Posicionamos los puntos en el eje cartesiano de la siguiente forma:

- x_0 se encuentra en el origen.
- x_1 se encuentra a distancia k (*2π/1024*).
- x_2 se encuentra a distancia 2k (*4π/1024*).

Los valores de la función serán los siguientes:

$$f(x_0) = f(0) = rAnteultimo = 4.5855297\ 10^{10}$$

$$f(x_1) = f(k) = rFinal = 4.585501\ 10^{10}$$

$$f(x_2) = f(2k) = rExtra = 4.58550067\ 10^{10}$$

La función interpoladora tendrá la siguiente forma:

$$f(x) = C_0 + C_1(x - x_0) + C_2(x - x_0)(x - x_1)$$

Calculamos los coeficientes a través de diferencias divididas:

$$C_0 = f[x_0] = rAnteultimo = \ 4.5855297\ 10^{10}$$

$$f[x_1] = rFinal = \ 4.585501\ 10^{10}$$

$$C_1 = f[x_0, x_1] = \frac{f[x_0] - f[x_1]}{x_0 - x_1} = \frac{rAnteultimo - rFinal}{-k} = \frac{4.5855297\ 10^{10} - 4.585501\ 10^{10}}{-\frac{2\pi}{1024}} = -4.67\ 10^{7}$$

$$f[x_2] = rExtra = 4.58550067\ 10^{10}$$

$$f[x_1, x_2] = \frac{f[x_1] - f[x_2]}{x_1 - x_2} = \frac{rFinal - rExtra}{-k} = \frac{4.585501\ 10^{10} - 4.58550067\ 10^{10}}{-\frac{2\pi}{1024}} = 5.38\ 10^{5}$$

$$C_2 = f[x_0, x_1, x_2] = \frac{f[x_0, x_1] - f[x_1, x_2]}{x_0 - x_2} = \frac{-4.67\ 10^{7} - 5.38\ 10^{5}}{-2k} = \frac{-4.72\ 10^{7}}{-\frac{4\pi}{1024}} = 3.849\ 10^{9}$$

Calculamos el mínimo derivando la función e igualando a 0:

$$\frac{df}{dx}(x) = C_1 + C_2(2x - x_1 - x_0)$$

Teniendo en cuenta que $x_0 = 0$ y $x_1 = k$:

$$\frac{df}{dx}(x) = C_1 + 2C_2x - C_2k = 0 \leftrightarrow x = \frac{C_2k - C_1}{2C_2} = \frac{3.849\ 10^{9}\ \frac{2\pi}{1024} - -4.67\ 10^{7}}{2\ 3.849\ 10^{9}} = 9.13\ 10^{-3}$$

Como suponíamos el valor del mínimo se encuentra entre k y 2k (entre rFinal y rExtra). Para calcular la diferencia angular, lo hacemos de la siguiente forma:

$$\tan\theta = \frac{minimo - k}{f(minimo)} \rightarrow \theta = \tan^{-1}\left(\frac{minimo - k}{f(minimo)}\right)$$

Puedo calcular la función en el mínimo (la mínima distancia) evaluando:

$$f(minimo) = C_0 + C_1(minimo) + C_2(minimo)(minimo - k)$$

$$f(9.13\ 10^{-3}) = 4.5855297\ 10^{10} - 4.67\ 10^{7}(9.13\ 10^{-3}) + 3.849\ 10^{9}(9.13\ 10^{-3})\left(9.13\ 10^{-3} - \frac{2\pi}{1024}\right)$$

$$f(9.13\ 10^{-3}) = 4.585497584\ 10^{10}$$

El valor tiene sentido, ya que es una distancia menor entre los dos r calculados. Ahora sí calculamos el ángulo en radianes:

$$\theta = \tan^{-1}\left(\frac{minimo - k}{f(minimo)}\right) = \tan^{-1}\left(\frac{9.13\ 10^{-3} - \frac{2\pi}{1024}}{4.585497584\ 10^{10}}\right) = 6.52945\ 10^{-14}$$

Ahora pasamos el valor anterior a grados:

$$\theta = 6.52945\ 10^{-14}\ \frac{180}{\pi} = 3.7411\ 10^{-12}$$

Ahora pasamos el valor anterior a segundos de arco:

$$\theta = \ 3.7411\ 10^{-12}\ 3600 = 1.3468\ 10^{-8}$$

Finalmente, pasamos el valor a segundos de arco por siglo terrestre, teniendo en cuenta que un año mercuriano son 87,97 días terrestres, y un año terrestre son 365,25 días:

$$\theta = 1.3468\ 10^{-8}\frac{365.25}{87.97}100 = 5.592\ 10^{-6}$$

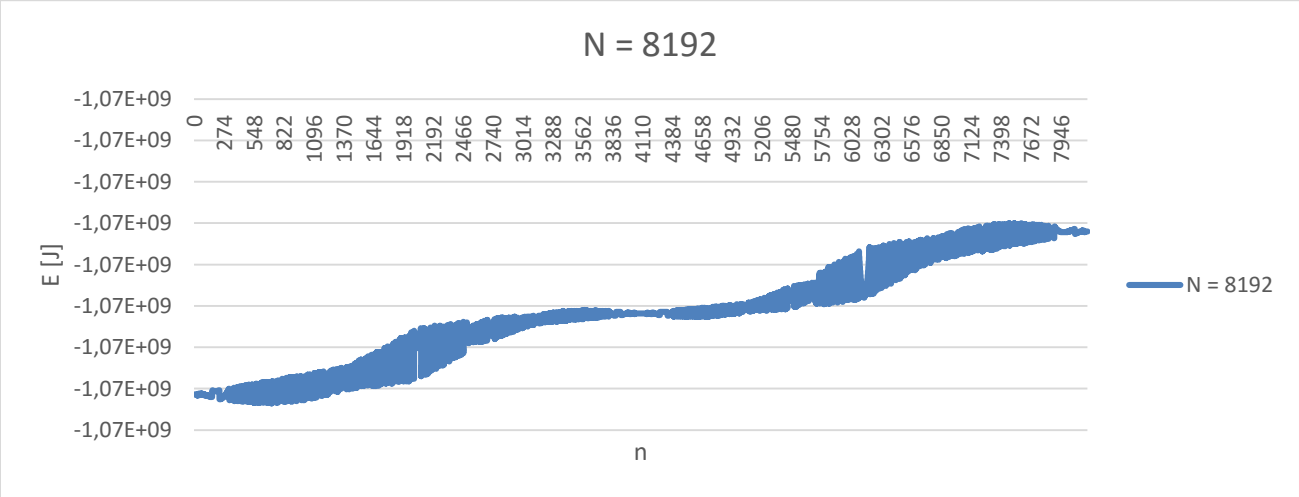
Como vemos, este valor es mucho menor que el esperado (44''). No he podido encontrar el error, y he revisado los cálculos tanto manualmente cómo computacionalmente (en simple y doble precisión). Además, he tratado de contactarme con alguno de los profesores por este inconveniente, pero no recibí respuesta. Por eso he determinado que no tiene sentido calcular la precesión para el sistema Mustafar (punto B.2 y B.4), ya que la precesión **conocida** para Mercurio no pudo ser verificada.

B.3) Conservación de la Energía

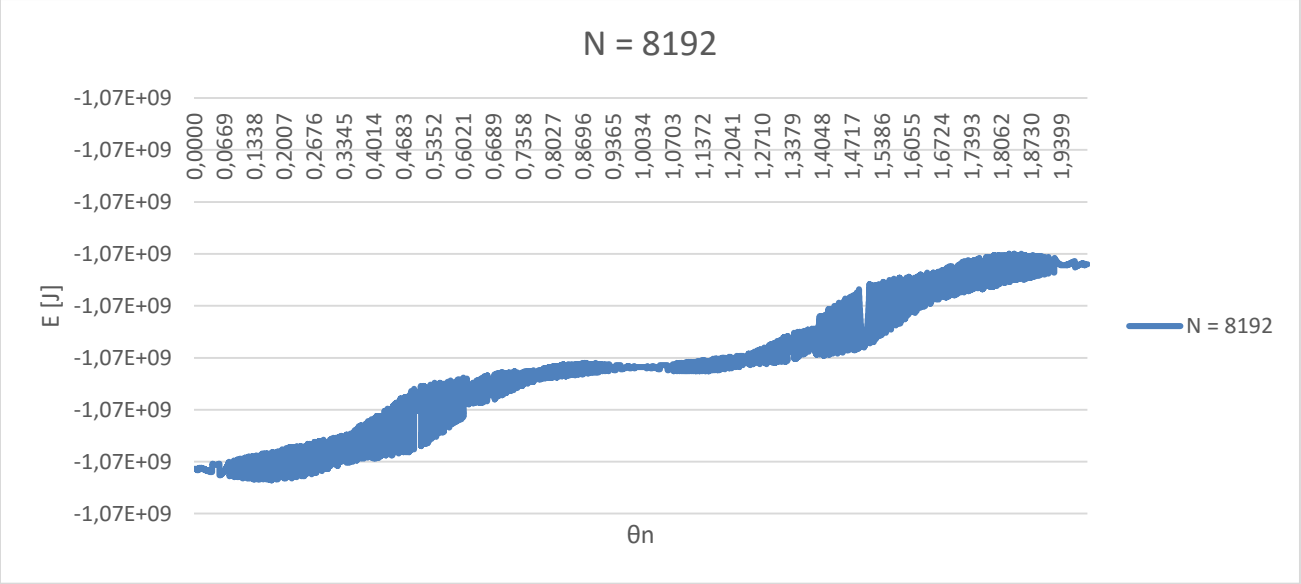
Este punto del TP consiste en calcular la energía en cada punto de la órbita (al igual que en el caso clásico), pero usando el método de Euler generalizado a la Relatividad, para un N determinado. Se eligió para el caso un N = 8192, y se vuelcan algunos resultados en la siguiente tabla:

n	u_n [1 / m]	E_n [J]
0	1,86E-11	-1,07E+09
1	1,86E-11	-1,07E+09
2	1,86E-11	-1,07E+09
3	1,86E-11	-1,07E+09
4	1,86E-11	-1,07E+09
5	1,86E-11	-1,07E+09
6	1,86E-11	-1,07E+09
7	1,86E-11	-1,07E+09
8	1,86E-11	-1,07E+09
9	1,86E-11	-1,07E+09
10	1,86E-11	-1,07E+09
...	...	...
8183	1,86E-11	-1,07E+09
8184	1,86E-11	-1,07E+09
8185	1,86E-11	-1,07E+09
8186	1,86E-11	-1,07E+09
8187	1,86E-11	-1,07E+09
8188	1,86E-11	-1,07E+09
8189	1,86E-11	-1,07E+09
8190	1,86E-11	-1,07E+09
8191	1,86E-11	-1,07E+09
8192	1,86E-11	-1,07E+09

El gráfico correspondiente al N utilizado arriba es el siguiente:



Además, se creó el siguiente gráfico, dónde se compara E_n vs θ_n, cómo fue requerido en el enunciado.



Conclusiones

Cómo suponíamos, a pesar de haber cambiado el modelo (*Newtoniano a Relativista*), sigue existiendo una ligera diferencia en la Energía a lo largo de la órbita. Nuevamente esto se debe a que el método de Euler es un método **no conservativo** (se debería usar en su lugar el *Método de Taylor*, *Método de Newmark* o el *Método de la Rayuela*).

B.5) Generalización del Método de Runge-Kutta Orden 4 a la Relatividad y Precesión de Mercurio

Al igual que con el método de Euler, agregamos el término correspondiente a la relatividad y el **Algoritmo 2-GR** (en el lenguaje C#) toma la siguiente forma:

```
float w1 = u0 + k * v0 / 2f;
float z1 = v0 + k * ((GM / h2) - u0 - (3f * GM * Pow(u0, 2) / c2)) / 2f;
float w2 = u0 + k * z1 / 2;
float z2 = v0 + k * ((GM / h2) - w1 - (3f * GM * Pow(w1, 2) / c2)) / 2f;
float w3 = u0 + k * z2;
float z3 = v0 + k * ((GM / h2) - w2 - (3f * GM * Pow(w2, 2) / c2));

uN = u0 + k * (v0 + 2 * z1 + 2 * z2 + z3) / 6;
vN = v0 + k * (6 * (GM / h2) - u0 - 2 * w1 - 2 * w2 - w3) / 6;
```

Con los valores iniciales:

```
float u0 = 1 / (a * (1 - e));
float v0 = 0;
```

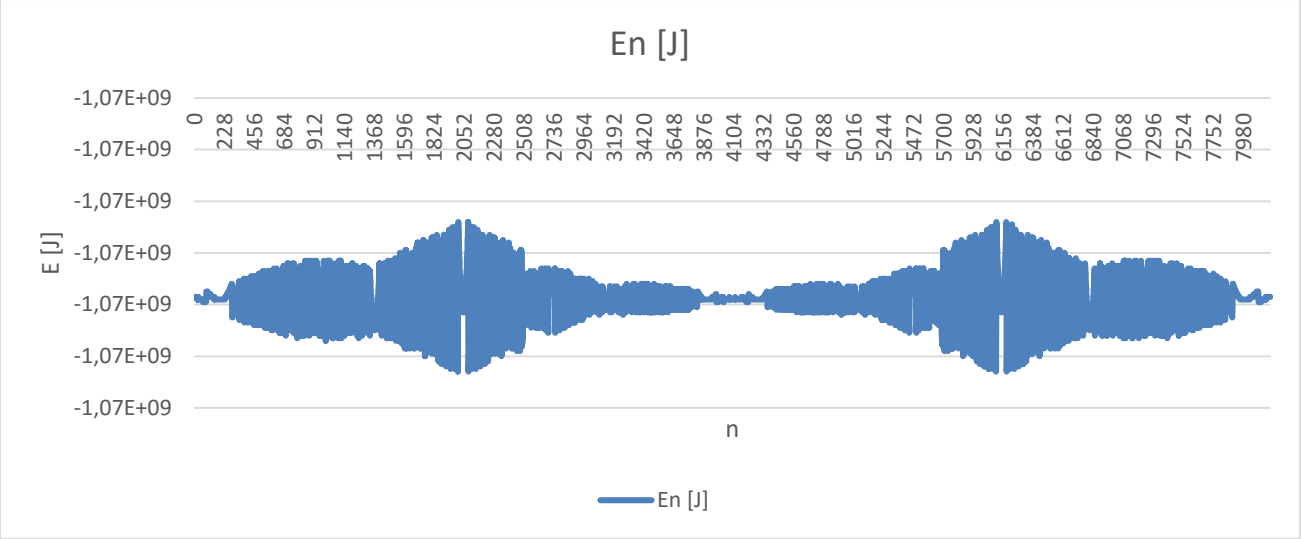
Al igual que fue explicado anteriormente, la precesión no pudo ser calculada correctamente, por lo que se omitió la resolución de los puntos B.6) y B.8)

B.7) Conservación de la Energía

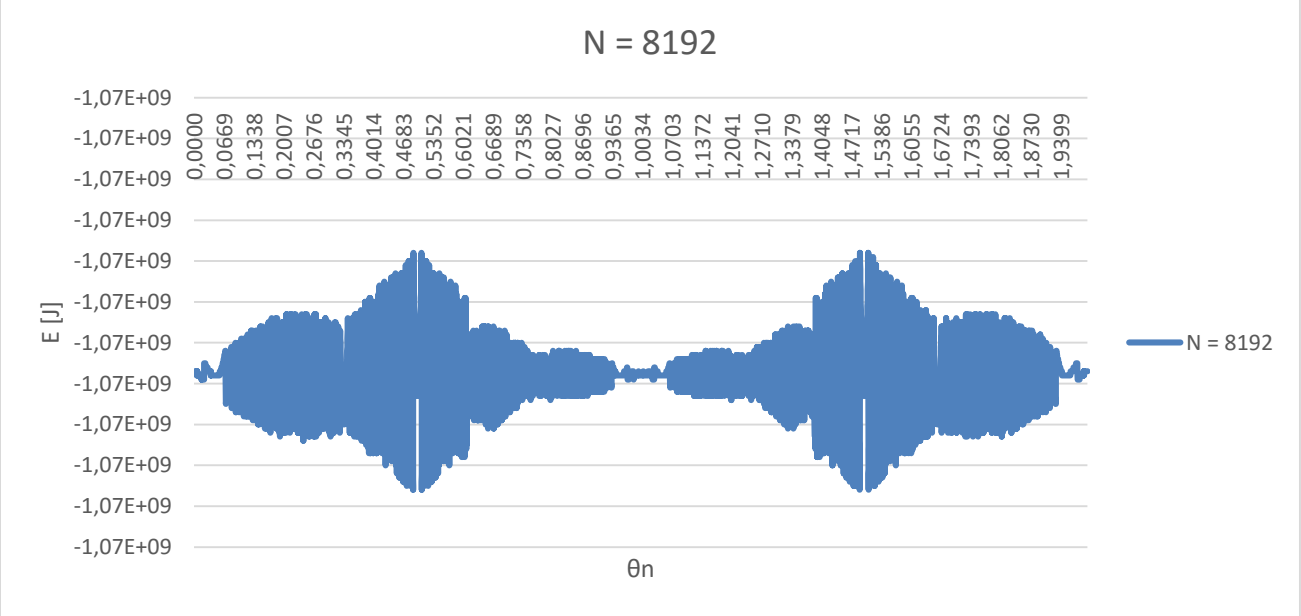
Nuevamente, en este punto del TP se debe calcular la energía en cada punto de la órbita (al igual que en el caso clásico), pero usando el método de **Runge-Kutta Orden 4** generalizado a la Relatividad, para un N determinado. Se eligió para el caso un N = 8192, y se vuelcan algunos resultados en la siguiente tabla:

n	u_n [1 / m]	E_n [J]
0	1,86E-11	-1,07E+09
1	1,86E-11	-1,07E+09
2	1,86E-11	-1,07E+09
3	1,86E-11	-1,07E+09
4	1,86E-11	-1,07E+09
5	1,86E-11	-1,07E+09
6	1,86E-11	-1,07E+09
7	1,86E-11	-1,07E+09
8	1,86E-11	-1,07E+09
9	1,86E-11	-1,07E+09
10	1,86E-11	-1,07E+09
…	…	…
8183	1,86E-11	-1,07E+09
8184	1,86E-11	-1,07E+09
8185	1,86E-11	-1,07E+09
8186	1,86E-11	-1,07E+09
8187	1,86E-11	-1,07E+09
8188	1,86E-11	-1,07E+09
8189	1,86E-11	-1,07E+09
8190	1,86E-11	-1,07E+09
8191	1,86E-11	-1,07E+09
8192	1,86E-11	-1,07E+09

El gráfico correspondiente al N utilizado arriba es el siguiente:



Además, se creó el siguiente gráfico, dónde se compara E_n vs θ_n, cómo fue requerido en el enunciado.



Conclusiones

Cómo suponíamos, a pesar de haber cambiado el modelo (*Newtoniano a Relativista*), sigue existiendo una ligera diferencia en la Energía a lo largo de la órbita. Con el método utilizado (**Runge-Kutta Orden 4**) la diferencia entre el punto al principio de la órbita, y el punto al final es prácticamente nula (verificar Tabla). Sin embargo, al realizar el gráfico, es notorio que este valor **no es constante** a lo largo de toda la órbita, aunque vuelve a oscilar de forma distinta al Método de Euler (a la mitad de la órbita, el afelio y en el perihelio los valores son los mismos, mientras que los extremos se encuentran en el cuarto y tres cuartos de órbita). Nuevamente, esto se debe a que el método de Runge-Kutta Orden 4 es un método **no conservativo** (se debería usar en su lugar el *Método de Taylor*, *Método de Newmark* o el *Método de la Rayuela*).

Conclusiones

En el presente Trabajo Práctico, pudimos aplicar de forma práctica, los mecanismos vistos del Análisis Numérico para resolver un problema matemático. La ecuación diferencial de segundo orden, a causa del propio significado de derivada (el límite del cociente incremental), es incalculable en máquinas de precisión finita. Ahora que se ha estudiado en el curso cómo estimar las soluciones de las ecuaciones diferenciales, se puede afirmar que el **Algoritmo 1** del TP1 se trataba del **Método de Euler explícito**, mientras que el **Algoritmo 2** del TP1 se trataba del **Método Runge-Kutta de Orden 4.** Ahora se comprende porque el segundo método convergía más rápido al resultado.

En el presente TP se presentaron errores, que no habían sido tan explícitos en el TP anterior: los **errores de discretización** (presentes tanto al integrar el área para el cálculo del período, cómo al calcular la propia derivada para comprobar la conservación de la Energía). Además, vimos que la Energía **no se conserva**, de acuerdo a los cálculos hechos, con lo que notamos la importancia del método que se debe elegir, de acuerdo al problema a tratar (en este caso, para un problema de Fuerzas Centrales y Conservativas, es menester utilizar un método conservativo).

Al igual que en el TP1, la diferencia más significativa entre el **Algoritmo 1-GR** y **Algoritmo 2-GR** consiste en la rapidez en que converge el método Runge-Kutta de Orden 4 (generalmente abreviado cómo RK4 o simplemente “Runge-Kutta”), lo cual explica porque es uno de los métodos más difundidos y utilizados.

Finalmente, otro detalle a resaltar del presente TP es la presencia de un tipo de error que no existía en el TP1: se trata de los **errores del modelo**. Queda claro en el presente TP que, al resolver la ecuación diferencial planteada por Newton, y por lo tanto despreciando el término relativista, la ecuación responde a una órbita elíptica cerrada, con lo cual la precesión de Mercurio no cuadra dentro de dicho modelo.

Código

```
using System;
using System.IO;

namespace TP2
{
   class Program
   {
      // Constantes del problema.
      static long maximoN = 2097152;
      static float np = 96570;
      static float lambdaOriginal = (np / 90000);
      static float G = 6.673f * Pow(10, -11);
      static float c = (3 * Pow(10, 8));
      static float c2 = Pow(c, 2);

      // Constantes dependientes de lambda
      static float lambda;
      static float m1;
      static float m2;
      static float GM;
      static float e;
      static float a;
      static float h2;
      static float h;
      static float q;

      public struct Ejes {
         private float semiejeMayor;
         private float semiejeMenor;

         public Ejes(int semiejeMayor, int semiejeMenor)
         {
            this.semiejeMayor = semiejeMayor;
            this.semiejeMenor = semiejeMenor;
         }

         public void setSemiejeMayor(float semiejeMayor)
         {
            this.semiejeMayor = semiejeMayor;
         }

         public void setSemiejeMenor(float semiejeMenor)
         {
            this.semiejeMenor = semiejeMenor;
         }

         public float getSemiejeMayor()
         {
            return this.semiejeMayor;
         }

         public float getSemiejeMenor()
         {
            return this.semiejeMenor;
         }

         public void copyFrom(Ejes semiejes)
         {
            this.setSemiejeMayor(semiejes.getSemiejeMayor());
            this.setSemiejeMenor(semiejes.getSemiejeMenor());
         }
      };

      static void Main(string[] args)
      {
         // Inicializo la aplicación y obtengo el nombre del archivo donde va
         // a quedar todo guardado.
         actualizarConstantesLambda(lambdaOriginal); // Le doy valor a las constantes dependientes de lambda
```

```
    var runName = "Files/";
    var folderName = runName + "/Algoritmo1 Clasico/";
    actualizarConstantesLambda(lambdaOriginal); // Le doy valor a las constantes dependientes de lambda
    var fileNameArea = InitializeAreaFile(folderName);
    var fileNamePrimeraLey = InitializePrimeraLeyFile(folderName);
    var fileNameSegundaLey = InitializeSegundaLeyFile(folderName);
    var fileNameTerceraLey = InitializeTerceraLeyFile(folderName);
    var fileNameEnergia = InitializeEnergyFile(folderName);
    Console.Write("Comenzando con Algoritmo1 Clasico...\n");
    var start = DateTime.Now;
    CorrerAlgoritmoNVeces(fileNameArea, fileNamePrimeraLey, fileNameSegundaLey, fileNameTerceraLey, fileNameEnergia);
    var end = DateTime.Now;
    var span = end - start;
    Console.Write("Finalizado Algoritmo 1 Clasico en " + ((int)span.TotalMilliseconds).ToString() + " milisegundos\n");
    // Cambio la carpeta para la segunda corrida.

    var folderName3 = runName + "/Algoritmo1 Relativista/";
    actualizarConstantesLambda(1); // Modifico las constantes dependientes de lambda para convertir el sistema a Mercurio y el Sol
    var fileNameEnergiaRelativista = InitializeEnergyFile(folderName3);
    Console.Write("Comenzando con Algoritmo1 Relativista...\n");
    var start3 = DateTime.Now;
    CorrerAlgoritmoNVeces_Relativista(fileNameEnergiaRelativista);
    var end3 = DateTime.Now;
    var span3 = end3 - start3;
    Console.Write("Finalizado Algoritmo 1 Relativista en " + ((int)span3.TotalMilliseconds).ToString() + " milisegundos\n");
    Console.Read();
}

private static void actualizarConstantesLambda(float valorLambda)
{
    lambda = valorLambda;
    m1 = lambda * 1.9891f * Pow(10, 30);
    m2 = lambda * 3.301f * Pow(10, 23);
    GM = G * (m1 + m2);
    e = 0.2056f / lambda;
    a = (Pow(lambda, 2)) * 5.791f * Pow(10, 10);
    h2 = a * GM * (1f - Pow(e, 2));
    h = Sqrt(h2);
    q = a * (1 - e);
}

#region Inicializadores

private static string InitializeAreaFile(string folderName)
{
    // Me aseguro de que exista una carpeta dónde dejar los archivos.
    var folder = folderName + "/Area/";
    var directoryInfo = new DirectoryInfo(folder);
    if (!directoryInfo.Exists)
    {
        directoryInfo.Create();
    }

    // Blanqueo el archivo de resultados, y le pongo las cabeceras a las columnas.
    var fileName = folder + "Area.csv";
    File.AppendAllText(fileName, string.Format("{0};{1};{2}{3}",
            "h",
            "A(O2) [m2]",
            "A(O4) [m2]",
            Environment.NewLine));
    return fileName;
}

private static string InitializePrimeraLeyFile(string folderName)
{
    // Me aseguro de que exista una carpeta dónde dejar los archivos.
    var folder = folderName + "/Primera Ley/";
    var directoryInfo = new DirectoryInfo(folder);
    if (!directoryInfo.Exists)
    {
        directoryInfo.Create();
```

```
    }

    // Blanqueo el archivo de resultados, y le pongo las cabeceras a las columnas.
    var fileName = folder + "Primera Ley.csv";
    File.AppendAllText(fileName, string.Format("{0};{1};{2};{3};{4}{5}",
          "N",
          "An [m]",
          "E(An) [m]",
          "Bn [m]",
          "E(Bn) [m]",
          Environment.NewLine));
    return fileName;
}

private static string InitializeSegundaLeyFile(string folderName)
{
    // Me aseguro de que exista una carpeta dónde dejar los archivos.
    var folder = folderName + "/Segunda Ley/";
    var directoryInfo = new DirectoryInfo(folder);
    if (!directoryInfo.Exists)
    {
        directoryInfo.Create();
    }

    // Blanqueo el archivo de resultados, y le pongo las cabeceras a las columnas.
    var fileName = folder + "Segunda Ley.csv";
    File.AppendAllText(fileName, string.Format("{0};{1};{2}{3}",
          "N",
          "T [s]",
          "E(T) [s]",
          Environment.NewLine));
    return fileName;
}

private static string InitializeTerceraLeyFile(string folderName)
{
    // Me aseguro de que exista una carpeta dónde dejar los archivos.
    var folder = folderName + "/Tercera Ley/";
    var directoryInfo = new DirectoryInfo(folder);
    if (!directoryInfo.Exists)
    {
        directoryInfo.Create();
    }

    // Blanqueo el archivo de resultados, y le pongo las cabeceras a las columnas.
    var fileName = folder + "Tercera Ley.csv";
    File.AppendAllText(fileName, string.Format("{0};{1};{2};{3};{4}{5}",
          "N",
          "Tn2 [s2]",
          "Rn3 [m3]",
          "Tn2 / Rn3 [s2/m3]",
          "E(Tn2 / Rn3) [s2/m3]",
          Environment.NewLine));
    return fileName;
}

private static string InitializeEnergyFile(string folderName)
{
    // Me aseguro de que exista una carpeta dónde dejar los archivos.
    var folder = folderName + "/Conservacion de la Energia/";
    var directoryInfo = new DirectoryInfo(folder);
    if (!directoryInfo.Exists)
    {
        directoryInfo.Create();
    }

    // Blanqueo el archivo de resultados, y le pongo las cabeceras a las columnas.
    var fileName = folder + "Conservacion de la Energia.csv";
    File.AppendAllText(fileName, string.Format("{0};{1};{2}{3}",
          "n",
          "un [1/m]",
```

```
            "En [J]",
            Environment.NewLine));
        return fileName;
    }

    #endregion

    #region EjecucionAlgoritmos

    private static void CorrerAlgoritmoNVeces(string fileNameArea, string fileNamePrimeraLey, string fileNameSegundaLey, string fileNameTercera-
Ley, string fileNameEnergia)
    {
        #region VariablesDelCiclo
        float areaAnterior = 0;
        float periodoAnterior = 0;
        float resultadoTerceraLeyAnterior = 0;
        Ejes semiejesAnterior = new Ejes(0, 0);
        float kAnterior = 0;
        #endregion
        bool calcularEnergia = false;

        for (long N = 8; N <= maximoN && N > 0; N *= 2)
        {
            Console.Write("Procesando para N = " + N.ToString() + "\n");
            float area;
            float periodo;
            float resultadoTerceraLey;
            Ejes semiejes = new Ejes(0, 0);
            float k = (2f * (float)Math.PI) / N; // Calculo el intervalo del ángulo (el paso de discretización)
            Algoritmo_Clasico(k, N, out area, ref semiejes, calcularEnergia, fileNameEnergia); // Corro el algoritmo y obtengo un resultado con (O2)
            realizarOperacionesPrimeraLey(fileNamePrimeraLey, N, semiejesAnterior, semiejes); // A.1
            area = realizarOperacionesArea(fileNameArea, N, areaAnterior, area, kAnterior, k); // A.2
            periodo = realizarOperacionesSegundaLey(fileNameSegundaLey, N, periodoAnterior, area); // A.2
            resultadoTerceraLey = realizarOperacionesTerceraLey(fileNameTerceraLey, N, periodo, semiejes, resultadoTerceraLeyAnterior); // A.3

            areaAnterior = area;
            periodoAnterior = periodo;
            resultadoTerceraLeyAnterior = resultadoTerceraLey;
            semiejesAnterior.copyFrom(semiejes);
            kAnterior = k;

            Console.Write("PROCESADO\n");
        }
    }

    private static void CorrerAlgoritmoNVeces_Relativista(string fileNameEnergia)
    {
        long N = 1048;
        Console.Write("Analizando la precesión de la órbita para N = " + N.ToString() + "\n");
        float k = (2f * (float)Math.PI) / N; // Calculo el intervalo del ángulo (el paso de discretización)
        Algoritmo_Relativista(k, N, fileNameEnergia);
        Console.Write("PROCESADO\n");
    }

    static void Algoritmo_Clasico(float k, long N, out float area, ref Ejes ejes, bool calculoEnergia, string fileNameEnergia)
    {
        #region VariablesMetodoDiferencial
        float uActual = 1 / (a * (1 - e));
        float uProximo = 0;
        float v0 = 0;
        float vN = 0;
        #endregion

        float uAnterior = uActual;
        float areaParcial = calcularArea((1 / uActual));
        for (long n = 0; n <= N - 1; n++)
        {
            metodoEuler(uActual, v0, ref uProximo, ref vN, k);
            #region CalculosSemiejes
            if (n == (N / 4 - 1)) // Me encuentro aproximadamente a un cuarto de la orbita, puedo calcular Bn
            {
```

```
            float hipotenusa = (1 / uProximo);
            float catetoMayor = (a * e); // Utilizo la distancia del centro al cuerpo estatico
            ejes.setSemiejeMenor(catetoPitagoras(hipotenusa, catetoMayor));
          }
          if (n == (N / 2 - 1)) // Me encuentro a la mitad de la orbita, puedo calcular An
          {
            ejes.setSemiejeMayor(Abs((1 / uProximo) - (a * e)));
          }
          #endregion
          #region CalculosArea
          float fN = calcularArea((1f / uProximo)); //Calculo el area para cada rN
          int factorTrapecio = 2;
          if (n == (N - 1)) //Para n inicial y final se suma una sola vez
          {
            factorTrapecio = 1;
          }
          areaParcial += (fN * factorTrapecio);
          #endregion
          #region CalculosEnergia
          if (calculoEnergia)
          {
            float energiaParcial = realizarOperacionesEnergia(N, n, uAnterior, uActual, uProximo, k, fileNameEnergia); // A.4
            uAnterior = uActual; // Me guardo el valor anterior para usarlo para el calculo de la derivada
          }
          #endregion
          uActual = uProximo;
          v0 = vN;
        }
        areaParcial *= (0.5f * k);
        area = areaParcial;
      }

      static void Algoritmo_Relativista(float k, long N, string fileNameEnergia)
      {
        #region VariablesMetodoDiferencial
        float uActual = 1 / (a * (1 - e));
        float uProximo = 0;
        float v0 = 0;
        float vN = 0;
        #endregion

        float uAnterior = uActual;
        float uAnteultimo = 0, uFinal, uExtra = 0; // Variables que se utilizaran para la Interpolacion
        for (long n = 0; n <= N - 1; n++)
        {
          metodoEulerRelativista(uActual, v0, ref uProximo, ref vN, k);
          #region CalculoEnergia
          float energiaParcial = realizarOperacionesEnergia(N, n, uAnterior, uActual, uProximo, k, fileNameEnergia);
          uAnterior = uActual; // Me guardo el valor anterior para usarlo para el calculo de la derivada
          #endregion
          if (n == N - 1)
          {
            uAnteultimo = uActual;
          }
          uActual = uProximo;
          v0 = vN;
        }
        uFinal = uActual;
        #region CalculoPrecesion
        metodoEulerRelativista(uActual, v0, ref uExtra, ref vN, k); // Obtengo un valor más de la órbita pasando el perihelio original

        float rAnteultimo = 1 / uAnteultimo;
        float rFinal = 1 / uActual;
        float rExtra = 1 / uExtra;
        Console.WriteLine("Los puntos utilizados para la interpolacion son: rAnteultimo = " + rAnteultimo.ToString() + ", rFinal = " + rFinal.ToString() + "
y rExtra = " + rExtra.ToString());
        float diferenciaPuntos = k;
        // El angulo alpha entre la masa en el foco y rAnteultimo o rExtra es k = 2pi/N
        Console.WriteLine("Para realizar la interpolacion se ubica rAnteulimo en el origen, y sucesivamente rFinal y rExtra con una diferencia de " +
diferenciaPuntos.ToString());
        float C0, C1, C2;
```

```
        interpolacionNewton(0, rAnteultimo, diferenciaPuntos, rFinal, 2 * diferenciaPuntos, rExtra, out C0, out C1, out C2); // Interpolo y obtengo los coeficientes del polinomio
        Console.WriteLine("C0 = " + C0.ToString());
        Console.WriteLine("C1 = " + C1.ToString());
        Console.WriteLine("C2 = " + C2.ToString());
        // El polinomio interpolador es de la forma f(x) = C0 + C1(x - x0) + C2(x - x0)(x - x1)
        // Si ubicamos rAnteultimo en el origen y sucesivamente rFinal y rExtra, x0 es 0, x1 = k y x2 = 2*k
        // f(x) = C0 + C1 (x) + C2(x)(x - k)
        // f(x) = x^2 (C2) + x (C1 - C2*k) + C0
        // f'(x) = 2*x*C2 + C1 - C2*k
        // f'(x) = 0 <==> 2*x*C2 + C1 - C2*k = 0 <==> x = (C2*k - C1) / (2*C2)
        float minimo = (((C2 * diferenciaPuntos) - C1) / (2f * C2));
        Console.WriteLine("El minimo calculado es " + minimo.ToString());
        float valorMinimo = (C0 + (C1 * minimo) + (C2 * minimo * (minimo - diferenciaPuntos)));
        Console.WriteLine("El valor de la funcion en el minimo es " + valorMinimo.ToString());
        // El ángulo theta equivalente a la precesión es la diferencia entre la posición del perihelio original (rFinal) y el máximo encontrado
        // Tan(theta) = (minimo - k) / valorMinimo ==> theta = Atan((minimo - k) / valorMinimo)
        float precesion = Atan((minimo - diferenciaPuntos) / valorMinimo);
        Console.WriteLine("El valor calculado de la precesion es " + precesion.ToString());
        float precesionSegundosDeArco = precesion / (float)Math.PI * 3600f * 180f * 365.25f * 100f / 87.97f;
        Console.WriteLine("El valor calculado de la precesion en segundos de arco por siglo terrestre es " + precesionSegundosDeArco.ToString());
        #endregion
    }

    #endregion

    #region LeyesDeKepler

    private static float calcularArea(float r)
    {
        return (0.5f * (Pow(r, 2)));
    }

    private static float calcularPeriodo(float area)
    {
        return ((2f * area) / h);
    }

    private static float realizarOperacionesArea(string fileName, long N, float areaAnterior, float areaCalculada, float k_anterior, float k)
    {
        float areaExtrapolada;
        if (areaAnterior == 0)
        {
            areaExtrapolada = areaCalculada; // Si es el primer resultado, no lo extrapolo
        }
        else
        {
            // Aplico la Extrapolacion de Richardson y obtengo un resultado con (O4)
            areaExtrapolada = extrapolacionRichardson(areaAnterior, areaCalculada, k_anterior, k, 2f);
        }

        // Armo el mensaje para mostrar para guardar en la tabla.
        var messageArea = string.Format("{0};{1};{2}{3}",
            "2pi/" + N.ToString(),
            areaCalculada.ToString(),
            areaExtrapolada.ToString(),
            Environment.NewLine);
        File.AppendAllText(fileName, messageArea); // Lo agrego al archivo.
        return areaExtrapolada;
    }

    private static void realizarOperacionesPrimeraLey(string fileNamePrimeraLey, long N, Ejes semiejesAnterior, Ejes semiejes)
    {
        float semiejeMayorCalculado = semiejes.getSemiejeMayor();
        float semiejeMenorCalculado = semiejes.getSemiejeMenor();
        float deltaSemiejeMayor = (Abs(semiejeMayorCalculado - semiejesAnterior.getSemiejeMayor()));
        float deltaSemiejeMenor = (Abs(semiejeMenorCalculado - semiejesAnterior.getSemiejeMenor()));

        var messagePrimeraLey = string.Format("{0};{1};{2};{3};{4}{5}", N,
            semiejeMayorCalculado.ToString(),
            deltaSemiejeMayor.ToString(),
```

```
            semiejeMenorCalculado.ToString(),
            deltaSemiejeMenor.ToString(),
            Environment.NewLine);
        File.AppendAllText(fileNamePrimeraLey, messagePrimeraLey); //Lo agrego al archivo.
    }

    private static float realizarOperacionesSegundaLey(string fileNameSegundaLey, long N, float periodoAnterior, float areaCalculada)
    {
        float periodoCalculado = calcularPeriodo(areaCalculada);

        float deltaPeriodo = Abs(periodoCalculado - periodoAnterior);

        var messageSegundaLey = string.Format("{0};{1};{2}{3}", N,
            periodoCalculado.ToString(),
            deltaPeriodo.ToString(),
            Environment.NewLine);
        File.AppendAllText(fileNameSegundaLey, messageSegundaLey); // Lo agrego al archivo.
        return periodoCalculado;
    }

    private static float realizarOperacionesTerceraLey(string fileNameTerceraLey, long N, float periodo, Ejes semiejes, float resultadoTerceraLeyAnterior)
    {
        // Resultados
        float periodoCuadrado = Pow(periodo, 2);
        float semiejeMayorCubico = Pow(semiejes.getSemiejeMayor(), 3);
        float resultadoTerceraLey = (periodoCuadrado / semiejeMayorCubico);

        // Error de la constante
        float deltaResultadoTerceraLey = Abs(resultadoTerceraLey - resultadoTerceraLeyAnterior);

        var messageTerceraLey = string.Format("{0};{1};{2};{3};{4}{5}", N,
            periodoCuadrado.ToString(),
            semiejeMayorCubico.ToString(),
            resultadoTerceraLey.ToString(),
            deltaResultadoTerceraLey.ToString(),
            Environment.NewLine);
        File.AppendAllText(fileNameTerceraLey, messageTerceraLey); // Lo agrego al archivo
        return resultadoTerceraLey;
    }

    #endregion

    #region Energia

    private static float calcularEnergia(float uN, float derivada)
    {
        float velocidadCuadrada = (h2 * (Pow(uN, 2) + Pow(derivada, 2)));
        return ((0.5f * velocidadCuadrada) - (GM * uN));
    }

    private static float realizarOperacionesEnergia(long N, long n, float uAnterior, float uActual, float uProximo, float k, string fileNameEnergia)
    {
        float energiaParcial;
        float energiaTotal = 0;
        if (n == 0)
        {
            energiaParcial = calcularEnergia(uActual, calcularDerivadaEnAdelanto(uActual, uProximo, k));
        }
        else energiaParcial = calcularEnergia(uActual, calcularDerivadaCentrada(uAnterior, uProximo, k));
        escribirResultadoEnergiaEnArchivo(n, uActual, energiaParcial, fileNameEnergia);
        energiaTotal += energiaParcial;
        if (n == N - 1)
        {
            energiaParcial = calcularEnergia(uProximo, calcularDerivadaEnAtraso(uActual, uProximo, k));
            escribirResultadoEnergiaEnArchivo(n + 1, uProximo, energiaParcial, fileNameEnergia);
            energiaTotal += energiaParcial;
        }
        return energiaTotal;
    }
```

```
private static void escribirResultadoEnergiaEnArchivo(long n, float uActual, float energiaParcial, string fileNameEnergia)
{
   var messageEnergia = string.Format("{0};{1};{2}{3}", n,
     uActual.ToString(),
     energiaParcial.ToString(),
     Environment.NewLine);
   File.AppendAllText(fileNameEnergia, messageEnergia); // Lo agrego al archivo.
}

#endregion

#region Operaciones Matematicas

static float Sin(double p)
{
   return (float)Math.Sin(p);
}

static float Cos(double p)
{
   return (float)Math.Cos(p);
}

static float Tan(double p)
{
   return (float)Math.Tan(p);
}

static float Atan(double p)
{
   return (float)Math.Atan(p);
}

static float Pow(float number, double pow)
{
   return (float)Math.Pow(number, pow);
}

static float Sqrt(double n)
{
   return (float)Math.Sqrt(n);
}

static float Abs(double n)
{
   return (float)Math.Abs(n);
}

static float catetoPitagoras(float hipotenusa, float cateto)
{
   return Sqrt(Pow(hipotenusa, 2) - Pow(cateto, 2));
}

private static float extrapolacionRichardson(float T1, float T2, float h1, float h2, float p)
{
   float q = (h1 / h2);
   return (T2 + ((T2 - T1) / ((Pow(q, p) - 1))));
}

private static void interpolacionNewton(float x0, float f0, float x1, float f1, float x2, float f2, out float C0, out float C1, out float C2)
{
   C0 = f0; // f[x0]
   C1 = diferenciasDivididas(f0, f1, (x0 - x1)); // f[x0,x1]
   float f1_f2 = diferenciasDivididas(f1, f2, (x1 - x2)); // f[x1,x2]
   C2 = diferenciasDivididas(C1, f1_f2, (x0 - x2)); // f[x0,x1,x2]
}

private static float diferenciasDivididas(float fi, float fn, float delta)
{
   return ((fi / delta) - (fn / delta));
}
```

```
    #endregion

    #region Diferenciacion

    private static float calcularDerivadaEnAtraso(float funcionEnAtraso, float funcion, float h)
    {
      return ((funcion - funcionEnAtraso) / h);
    }

    private static float calcularDerivadaEnAdelanto(float funcion, float funcionEnAdelanto, float h)
    {
      return ((funcionEnAdelanto - funcion) / h);
    }

    private static float calcularDerivadaCentrada(float funcionEnAtraso, float funcionEnAdelanto, float h)
    {
      return ((funcionEnAdelanto - funcionEnAtraso) / (2f * h));
    }

    private static void metodoEuler(float u0, float v0, ref float uN, ref float vN, float k)
    {
      uN = u0 + (k * v0);
      vN = v0 - (k * u0) + (k * (GM / h2));
    }

    private static void metodoEulerRelativista(float u0, float v0, ref float uN, ref float vN, float k)
    {
      uN = u0 + (k * v0);
      vN = v0 - (k * u0) + (k * (GM / h2)) + (k * 3f * GM * Pow(u0, 2) / c2);
    }

    private static void rungeKuttaOrden4(float u0, float v0, ref float uN, ref float vN, float k)
    {
      float w1 = u0 + k * v0 / 2f;
      float z1 = v0 + k * ((GM / h2) - u0) / 2f;
      float w2 = u0 + k * z1 / 2;
      float z2 = v0 + k * ((GM / h2) - w1) / 2f;
      float w3 = u0 + k * z2;
      float z3 = v0 + k * ((GM / h2) - w2);

      uN = u0 + k * (v0 + 2 * z1 + 2 * z2 + z3) / 6;
      vN = v0 + k * (6 * (GM / h2) - u0 - 2 * w1 - 2 * w2 - w3) / 6;
    }

    private static void rungeKuttaOrden4Relativista(float u0, float v0, ref float uN, ref float vN, float k)
    {
      float w1 = u0 + k * v0 / 2f;
      float z1 = v0 + k * ((GM / h2) - u0 - (3f * GM * Pow(u0, 2) / c2)) / 2f;
      float w2 = u0 + k * z1 / 2;
      float z2 = v0 + k * ((GM / h2) - w1 - (3f * GM * Pow(w1, 2) / c2)) / 2f;
      float w3 = u0 + k * z2;
      float z3 = v0 + k * ((GM / h2) - w2 - (3f * GM * Pow(w2, 2) / c2));

      uN = u0 + k * (v0 + 2 * z1 + 2 * z2 + z3) / 6;
      vN = v0 + k * (6 * (GM / h2) - u0 - 2 * w1 - 2 * w2 - w3) / 6;
    }

    #endregion

  }
}
```