Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO FIN DE GRADO

Grado en Matemáticas

Los métodos de Gauss en la simulación del sistema solar

Autor: Ismael García Palomino

Tutora: María Paz Calvo Cabrero

2025

Resumen: Este Trabajo de Fin de Grado estudia los métodos de Gauss-Legendre y demuestra algunas de sus propiedades geométricas que son relevantes para simular sistemas Hamiltonianos durante largos periodos de tiempo. Se muestran resultados numéricos obtenidos al integrar con dichos métodos los problemas de Kepler y de los planetas exteriores utilizando el software matemático MatLab para las distintas implementaciones realizadas.

Palabras clave: Métodos de Gauss-Legendre, Sistemas Hamiltonianos, Simplecticidad, Simetría, Simulación, Problema de Kepler, Problema de los Planetas Exteriores.

Abstract: This work studies the Gauss-Legendre methods and demonstrates some of their geometric properties, which are relevant for simulating Hamiltonian systems over long periods of time. Numerical results are presented from integrating the Kepler problem and the Outer Solar System problem using these methods, with different implementations carried out in the mathematical software MATLAB.

Keywords: Gauss-Legendre Methods, Hamiltonian Systems, Symplecticity, Symmetry, Simulation, Kepler Problem, Outer Solar System Problem.

Agradecimientos

Esta sección no será breve. Y es que considero que este Trabajo Fin de Grado será uno de mis mayores logros académicos, si no el que más. A lo largo de mi vida, he tenido la suerte de conocer a muchas personas que me han ayudado y acompañado, a quienes estoy profundamente agradecido. Es por ello que les dedico esta memoria.

Quiero empezar por todos los profesores que me han enseñado, tanto en mi colegio, el San Agustín, como en la Universidad de Valladolid. Mención especial, como no, a Mari Paz, la tutora de esta memoria. Su empeño, conocimiento, y guía, han sido fundamentales para la realización de este trabajo.

También quiero nombrar a mis amigos de Valladolid, algunos de los cuales conozco desde muy pequeño. En particular, a mi amigo Fabio, quien no solo me ha acompañado en muchas aventuras, sino que también me presentó a personas que, con el tiempo, se han convertido en amigos tan importantes para mí como los del colegio.

Un poco en la misma línea, no me puedo olvidar de mi pueblo, Olmos de Esgueva, y de su gente, que siempre me ha tenido una gran estima. En especial, a mis amigos del pueblo, a los que les tengo un cariño especial. Los veranos que he pasado allí han sido la pieza clave para recargar pilas de cara al siguiente año académico.

Y por supuesto, lo mejor sin duda de mi etapa universitaria y por lo que considero que cumplí un sueño de vida, es haber podido compartir estos últimos cinco años la afición de las matemáticas con un grupo de personas inigualable. No solo han sido buenos compañeros de clase, ayudándome y motivándome a partes iguales. También han conseguido en las duras épocas de exámenes, o simplemente en días que se hacían más pesados, hacerme

reir. Nunca olvidaré lo bien recibido que fui por mis compañeros del doble grado cuando acepté el reto de estudiar las dos carreras. Todos los momentos vividos en clase, en la cafetería, en las noches de los jueves, o en los viajes que hemos hecho, han sido mágicos. Sin lugar a duda, considero que he llegado hasta aquí por haber estado tan bien acompañado de "Escobilla".

Quiero destacar igualmente los últimos años de carrera. Durante el año de Erasmus pude vivir experiencias increibles con la gente que conocí en la Piazza de Perugia. Cuando recuerdo aquel año es inevitable que me salga una sonrisa, ya que tuve el privilegio y la fortuna de conocer a personas de toda España, y vivir con ellos un año inolvidable. Y para acabar, tuve la oportunidad en Segovia de experimentar lo que era vivir en una residencia universitaria, y poder hacer nuevos amigos, tanto en la propia residencia, como entre mis compañeros de clase de Ingeniería Informática. Todos ellos saben cuánto valoro este lugar, y es que pocas cosas pueden superar lo que significa "La Vida en Segovia".

Ahora bien, absolutamente nada de todo lo anterior tendría sentido si no hubiera tenido el marco de seguridad, confianza, y amor, que todos los días me ha dado mi familia, en especial, mi padre y mi madre. Gracias a ellos he podido concentrarme en lo que más me gustaba, sin preocuparme de nada más. Han sido la mayor fuente de inspiración y la razón principal de todo mi éxito académico a lo largo de mi vida, y por ello les debo y les quiero tanto. Gracias Papá. Gracias Mamá. Lo conseguimos.

Ismael García Palomino
Valladolid, 10 de Abril de 2025.

Índice general

Capítulo 1

Introducción

En el Trabajo Fin de Grado que presentamos se abordan los métodos de Gauss-Legendre, con el propósito final de realizar de manera eficiente simulaciones a tiempos largos de sistemas Hamiltonianos, y prestando atención a la propagación de los errores de redondeo.

Los Capítulos 2 y 3 están dedicados al estudio de los métodos Runge-Kutta y de los métodos de Gauss, fundamentales en la resolución numérica de ecuaciones diferenciales ordinarias. Se estudian algunas de sus propiedades geométricas más relevantes, como la simetría y la simplecticidad, que son la clave de los buenos resultados que obtendremos cuando las implementemos en MatLab. Además, se incluyen ejemplos ilustrativos y demostraciones que permiten comprender mejor su construcción y comportamiento.

Dentro del Capítulo 4, se presenta una primera implementación práctica de estos métodos en MatLab, aplicándolos a la integración del *Problema de Kepler*. Se utilizan tanto la iteración de Newton como la iteración de punto fijo para resolver las ecuaciones no lineales resultantes y se comparan cuatro implementaciones distintas. A partir de los resultados obtenidos, se sacan algunas conclusiones que nos servirán de guía para el capítulo siguiente.

El Capítulo 5 está dedicado al *Problema de los 5 planetas exteriores*, un problema más complejo que el anterior y cuya integración numérica es el objetivo final de este Trabajo Fin de Grado. De nuevo, la implementación se ha llevado en MatLab y se han incorporado algunas mejoras en los métodos para aumentar la precisión de los resultados.

La memoria incluye también la bibliografía, que proporciona al lector referencias adicionales sobre el tema, y los Apéndices que contienen el código implementado en MatLab que permite ejecutar los mismos programas empleados en el desarrollo de este trabajo.

Capítulo 2

Métodos Runge-Kutta y sus propiedades geométricas

Aunque en la asignatura obligatoria de *Ampliación de Análisis Numérico* del Grado en Matemáticas se dedica un tema a revisar brevemente las familias básicas de métodos numéricos para la resolución de ecuaciones diferenciales ordinarias, para aquellos que no han cursado la asignatura optativa *Solución Numérica de Ecuaciones Diferenciales*, en la que se desarrolla con más detalle la teoría básica de los métodos Runge-Kutta, entre otros, se han incluido en la primera sección de este primer capítulo los conceptos básicos relacionados con los métodos Runge-Kutta para que se puedan entender mejor los conceptos y resultados que se van a presentar en el resto del capítulo y en el Capítulo 3 relativo a los métodos de Gauss.

2.1. Métodos Runge-Kutta para sistemas diferenciales generales

2.1.1. Definición de un método Runge-Kutta

Consideremos un sistema de ecuaciones diferenciales

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(t, \mathbf{x}(t)), \tag{2.1}$$

junto con una condición inicial $\mathbf{x}(t_0) = \mathbf{x}_0$, con $\mathbf{x}_0 \in \mathbb{R}^D$, y donde $\mathbf{f} : \mathbb{R} \times \mathbb{R}^D \to \mathbb{R}^D$ es suficientemente regular para garantizar la existencia y unicidad de la solución de dicho problema. Las ecuaciones para avanzar un paso (de longitud h) en la integración numérica de (2.1) mediante un método Runge-Kutta de s etapas, partiendo de una aproximación $\mathbf{x}_n$ a $\mathbf{x}(t_n)$ son las siguientes

$$\mathbf{k_i} = \mathbf{f}\left(t_n + c_i h, \mathbf{x}_n + h\sum_{j=1}^{s} a_{ij}\mathbf{k}_j\right), \qquad 1 \le i \le s, \tag{2.2}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\sum_{i=1}^{s} b_i \mathbf{k}_i, \tag{2.3}$$

$$t_{n+1} = t_n + h. \tag{2.4}$$

Con nuestra notación, h es la *longitud de paso de integración*, mientras que los coeficientes b_i, $1 \le i \le s$, y a_{ij}, $1 \le i, j \le s$ definen el método. Además, supondremos que se cumple que $c_i = \sum_{j=1}^{s} a_{ij}$, $1 \le i \le s$.

Alternativamente, las ecuaciones (2.2) – (2.4) se pueden reescribir como

$$\mathbf{X}_i = \mathbf{x}_n + h\sum_{j=1}^{s} a_{ij}\mathbf{f}\left(t_n + c_j h, \mathbf{X}_j\right), \qquad 1 \le i \le s, \tag{2.5}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\sum_{i=1}^{s} b_i \mathbf{f}\left(t_n + c_i h, \mathbf{X}_i\right), \tag{2.6}$$

$$t_{n+1} = t_n + h.$$

Los vectores $\mathbf{X}_i$, $1 \le i \le s$, son las llamadas *etapas intermedias*, que son aproximaciones a la solución de (2.1) en los tiempos $t_n + c_i h$, $1 \le i \le s$.

Además, de manera más compacta, podemos representar un método Runge-Kutta con el llamado *tablero de Butcher*

$$\begin{array}{c|c} \mathbf{c} & A \\ \hline & \mathbf{b} \end{array} \quad = \quad \begin{array}{c|ccc} c_1 & a_{11} & \dots & a_{1s} \\ \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & \dots & a_{ss} \\ \hline & b_1 & \dots & b_s \end{array} . \tag{2.7}$$

Cabe destacar que si $a_{ij} = 0$ para $i \le j$ (la matriz A es estrictamente triangular inferior), la evaluación en cada etapa intermedia se hace de manera explícita en función de las evaluaciones en las etapas intermedias anteriores. Se dice que el método es *explícito*.

Definición 1. *Se dice en este caso que un sistema de ecuaciones diferenciales es autónomo si* $\boldsymbol{f}$ *no tiene dependencia explícita del tiempo, es decir si* (2.1) *tiene la forma*

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t)). \tag{2.8}$$

En este caso, la ecuación (2.2) se reduce a $\mathbf{k_i} = \mathbf{f}\left(\mathbf{x}_n + h\sum_{j=1}^{s} a_{ij}\mathbf{k}_j\right)$, con $1 \leq i \leq s$. De manera análoga, si (2.8) se integra numéricamente con un método Runge-Kutta, se puede definir el flujo numérico asociado.

A continuación, recordamos lo que es el *flujo* de un sistema diferencial, particularizando al caso de sistemas Hamiltonianos.

Definición 2. *Sean* t *y* t_0 *números reales de un intervalo de tiempo* I *en el que está definida la solución de* (2.1)*. El flujo del un sistema diferencial tras* $t - t_0$ *unidades de tiempo, denotado como* $\Phi_f(t - t_0)$*, es una transformación del espacio de fases* Ω *en sí mismo de manera que, para* $\boldsymbol{x}_0 \in \Omega$ *se tiene que*

$$\boldsymbol{x} = \Phi_H(t - t_0)\boldsymbol{x}_0 \tag{2.9}$$

es el valor de la solución de (2.1) *en tiempo* t *que en* t_0 *tiene como condición inicial* $\boldsymbol{x}_0$*.*

Nota 1: En el caso de sistemas autónomos (2.8) podemos suponer sin pérdida de generalidad que $t_0 = 0$, y si no es necesario hacer referencia explícita a la función $\mathbf{f}$ denotamos el flujo tras t unidades de tiempo por Φ_t.

Nota 2: Por las propiedades de grupo, se verifica que

$$\Phi_{t+s} = \Phi_t \circ \Phi_s.$$

Definición 3. *El flujo numérico de un sistema de ecuaciones diferenciales asociado a un método Runge-Kutta es la aplicación* Ψ_h *que avanza la solución numérica* h *unidades de tiempo. Dicho de otra manera, la imagen por* Ψ_h *de la aproximación numérica en* t_n *es la aproximación en* $t_{n+1} = t_n + h$*.*

$$\begin{aligned} \Psi_h : \mathbb{R}^D &\to \mathbb{R}^D \\ \mathbf{x}_n &\mapsto \Psi_h(\mathbf{x}_n) = \mathbf{x}_{n+1} \end{aligned}$$

2.1.2. Error local y orden de convergencia

Una vez visto un método Runge-Kutta genérico podemos empezar a definir conceptos clave a lo largo del texto, como son el *error local* y el *orden de convergencia*.

Definición 4. *El error local de un método Runge-Kutta es el error cometido tras un paso, es decir*

$$\rho_{n+1} = \tilde{\mathbf{x}}(t_{n+1}) - \mathbf{x}_{n+1}, \tag{2.10}$$

donde $\tilde{\mathbf{x}}(t_n)$ *es la solución exacta de* (2.1) *o* (2.8) *con condición inicial* $\tilde{\mathbf{x}}(t_n) = \mathbf{x}_n$.

Definición 5. *Se dice que un método Runge-Kutta es de orden p si los errores locales se comportan como* $O(h^{p+1})$ *cuando* $h \to 0$. *Alternativamente, podemos decir que un método Runge-Kutta es de orden p si los desarrollos de Taylor de* $\mathbf{x}(t_{n+1})$ *y* $\tilde{\mathbf{x}}_{n+1}$ *coinciden hasta los términos en* h^p *incluidos.*

2.1.3. Ejemplos

Incluimos a continuación dos ejemplos que nos pueden ayudar a familiarizarnos con el funcionamiento de los métodos Runge-Kutta. Son *el método de Runge* y la *regla implícita del punto medio*.

- **Método de Runge**

  Es un método de 4 etapas ($s = 4$) con orden de convergencia cuatro ($p = 4$), cuyo tablero de Butcher viene dado por

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} \end{array} .$$

  Las ecuaciones para avanzar un paso de longitud h con el método, en

la interpolación de (2.1) partiendo de t_n y $\mathbf{x}_n$ son

$$\begin{aligned}
\mathbf{k}_1 &= \mathbf{f}\left(t_n, \mathbf{x}_n\right), \\
\mathbf{k}_2 &= \mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{x}_n + h\frac{\mathbf{k}_1}{2}\right), \\
\mathbf{k}_3 &= \mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{x}_n + h\frac{\mathbf{k}_2}{2}\right), \\
\mathbf{k}_4 &= \mathbf{f}\left(t_n + h, \mathbf{x}_n + h\mathbf{k}_3\right), \\
\mathbf{x}_{n+1} &= \mathbf{x}_n + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4), \\
t_{n+1} &= t_n + h.
\end{aligned}$$

El método es explícito porque cada $\mathbf{k}_i$ se puede calcular a partir de los valores $\mathbf{k}_j$, con $1 \leq j < i$, previamente hallados.

- **Regla implícita del punto medio**

  Se trata del único método Runge-Kutta con una etapa ($s = 1$) y orden dos ($p = 2$). Su tablero de coeficientes es

$$\begin{array}{c|c} \frac{1}{2} & \frac{1}{2} \\ \hline & 1 \end{array},$$

  y las ecuaciones para avanzar un paso de longitud h en la integración de (2.1) son

$$\begin{aligned}
\mathbf{X}_1 &= \mathbf{x}_n + \frac{h}{2}\mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{X}_1\right), \\
\mathbf{x}_{n+1} &= \mathbf{x}_n + h\mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{X}_1\right), \\
t_{n+1} &= t_n + h.
\end{aligned}$$

  El método es implícito porque la etapa intermedia $\mathbf{X}_1$ aparece como argumento de $\mathbf{f}$ en la ecuación que define $\mathbf{X}_1$. Reemplazando en la primera ecuación la evaluación de $\mathbf{f}$ en la etapa intermedia, por el resultado de despejarla en la segunda ecuación, se llega a la formulación alternativa del método

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{f}\left(t_n + \frac{h}{2}, \frac{\mathbf{x}_n + \mathbf{x}_{n+1}}{2}\right),$$

  que justifica su nombre.

2.2. Métodos Runge-Kutta simplécticos

En esta sección revisaremos brevemente la definición de sistema Hamiltoniano autónomo, y algunos conceptos y resultados relacionados con su integración numérica mediante métodos Runge-Kutta simplécticos.

Definición 6. *Sea Ω un dominio (conjunto abierto, no vacío y conexo) de $\mathbb{R}^{2d}$ formado por puntos de la forma $(\mathbf{p}, \mathbf{q}) = (p_1, ..., p_d, q_1, ..., q_d)$, y sea $H = H(\mathbf{p}, \mathbf{q})$ una función real, suficientemente regular definida en Ω. Un sistema Hamiltoniano de ecuaciones diferenciales con función Hamiltoniana H está definido por las ecuaciones*

$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i}, \quad \frac{dq_i}{dt} = \frac{\partial H}{\partial p_i}, \qquad i = 1, ..., d. \tag{2.11}$$

El número natural d es conocido como el número de *grados de libertad* del sistema. La regularidad de H depende de cada caso, pero para lo que nos concierne podemos suponer que es de clase C^2, ya que con eso conseguimos que el lado derecho de (2.11) sea C^1.

Otra forma de reescribir las ecuaciones (2.11) es la siguiente. Definimos la matriz J como

$$J = \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix}, \tag{2.12}$$

donde I es la matriz identidad de dimensión d y 0 la matriz nula del mismo tamaño. De este modo, tendríamos una única expresión que describiría nuestro sistema diferencial (2.11).

$$\frac{d}{dt}\mathbf{x} = J^{-1}\nabla H(\mathbf{x}), \tag{2.13}$$

con $\mathbf{x} = (\mathbf{p}, \mathbf{q})^T$. Así, el sistema Hamiltoniano (2.13) se puede expresar en el formato (2.8) con $D = 2d$.

En muchos casos, la función Hamiltoniana será la energía de un sistema físico, que en el caso autónomo es una cantidad conservada. Podemos dar un primer ejemplo de este caso que más tarde estudiaremos: *el problema de Kepler en coordenadas cartesianas.*

El problema de Kepler describe el movimiento en un plano de un punto material que es atraído hacia el origen con una fuerza inversamente proporcional al cuadrado de la distancia. En forma adimensional, la función Hamiltoniana viene dada por

$$H = \frac{1}{2}(p_1^2 + p_2^2) - \frac{1}{\sqrt{q_1^2 + q_2^2}}. \tag{2.14}$$

y las ecuaciones del movimiento son las siguientes

$$\frac{dp_i}{dt} = -\frac{q_i}{(q_1^2 + q_2^2)^{3/2}}, \quad \frac{dq_i}{dt} = p_i \qquad i = 1, 2. \tag{2.15}$$

Se trata pues, de un sistema Hamiltoniano con dos grados de libertad.

Definición 7. *Se dice que una aplicación lineal $A : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ es simpléctica si*

$$A^T J A = J,$$

siendo J la matriz definida en (2.12).

En la figura (2.1) podemos observar un ejemplo sencillo de una aplicación lineal simpléctica.

Definición 8. *Se dice que una transformación $\psi : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ que lleva $(\mathbf{p}, \mathbf{q})$ en $\psi(\mathbf{p}, \mathbf{q}) = (\mathbf{p}^*, \mathbf{q}^*)$ es simpléctica si*

$$(\psi')^T J \, \psi' = J, \tag{2.16}$$

donde

$$\psi' = \begin{pmatrix} \partial \mathbf{p}^* / \partial \mathbf{p} & \partial \mathbf{p}^* / \partial \mathbf{q} \\ \partial \mathbf{q}^* / \partial \mathbf{p} & \partial \mathbf{q}^* / \partial \mathbf{q} \end{pmatrix}.$$

En el caso $d = 1$, la condición (2.16) que debe cumplir una transformación para ser simpléctica equivale a decir que dicha transformación conserva el área y su orientación. Para dimensiones mayores es intuitivo pensar en la conservación del volumen. Sin embargo, (2.16) es más fuerte que eso. Tenemos que considerar superficies orientadas bidimensionales para trabajar con sus proyecciones sobre los planos de variables (p_i, q_i), con $1 \leq i \leq d$. La cantidad conservada por una transformación simpléctica es la suma de las áreas orientadas de cada una de las d proyecciones sobre los planos coordenados (p_i, q_i). En algunos libros también se llama *transformación canónica*.

Podemos ya enunciar el teorema que nos interesa.

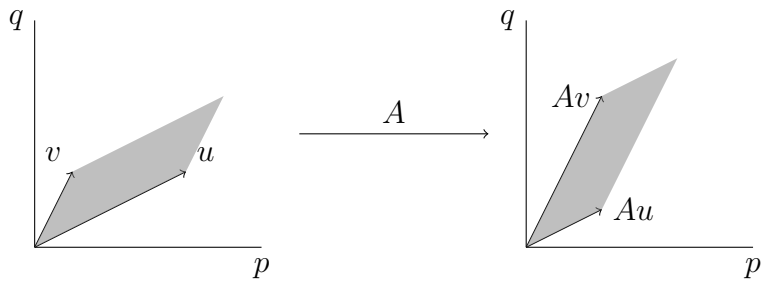


Figura 2.1: Aplicación simpléctica.

Teorema 1. *Para cada t en el intervalo de tiempos I en el que está definida la solución del sistema Hamiltoniano* (2.11) *el flujo Φ_H es una transformación simpléctica.*

Demostración. Por simplicidad, usaremos como se ha indicado antes Φ_t en vez de $\Phi_H(t)$ para denotar el flujo del sistema Hamiltoniano (2.13). Queremos demostrar que

$$(\Phi_t')^T J \Phi_t' = J. \tag{2.17}$$

Para $t = 0$ es claro que se cumple, por ser Φ_0 el operador identidad. Basta entonces ver que la derivada con respecto del tiempo del lado izquierdo de la igualdad (2.17) es nula. Como sabemos que Φ_t es solución de (2.13), se cumple que

$$\frac{d}{dt}(\Phi_t') = J^{-1}\nabla^2 H(\Phi_t)\Phi_t', \tag{2.18}$$

donde $\nabla^2 H(\mathbf{x})$ es la matriz Hessiana de $H(\mathbf{x})$, que es simétrica. Por otro lado, derivando en (2.17) y haciendo uso de (2.18) se tiene

$$\begin{aligned}
\frac{d}{dt}\left((\Phi_t')^T J\, \Phi_t'\right) &= \left(\frac{d}{dt}\Phi_t'\right)^T J\, \Phi_t' + (\Phi_t')^T J \left(\frac{d}{dt}\Phi_t'\right) \\
&= (\Phi_t')^T \nabla^2 H(\Phi_t)(J^{-1})^T J\, \Phi_t' + (\Phi_t')^T J J^{-1} \nabla^2 H(\Phi_t)\Phi_t' \\
&= -(\Phi_t')^T \nabla^2 H(\Phi_t)\Phi_t' + (\Phi_t')^T \nabla^2 H(\Phi_t)\Phi_t' \\
&= 0,
\end{aligned}$$

ya que, por (2.12), $J^{-1} = J^T$ y $J^2 = -I$.

□

Por otro lado, vamos a ver que condiciones tienen que satisfacer coeficientes de un método Runge-Kutta para que el flujo numérico asociado sea una transformación simpléctica si el sistema diferencial que se quiere integrar es Hamiltoniano.

Teorema 2. *Si los coeficientes de un método Runge-Kutta con tablero de Butcher* (2.7) *satisfacen*

$$b_i a_{ij} + b_j a_{ji} - b_i b_j = 0, \qquad 1 \leq i, j \leq s, \tag{2.19}$$

entonces cuando se aplica a un sistema Hamiltoniano (2.13) *la transformación que define en el espacio de fases es simpléctica.*

Demostración. Para probar que el flujo numérico asociado a un método Runge-Kutta definido por (2.5) y (2.6) es una transformación simpléctica tenemos que ver que la matriz jacobiana de la transformación que avanza un paso de longitud h con el método, denotada como $\left(\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n}\right)$, cumple la condición de simplecticidad (2.17), es decir, que

$$\left(\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n}\right)^T J \left(\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n}\right) = J. \tag{2.20}$$

De la formulación del método Runge-Kutta (2.5) y (2.6) en el caso de problemas autónomos tenemos que

$$\left(\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n}\right) = I + h \sum_{i=1}^{s} b_i D\mathbf{f}\left(\mathbf{X}_i\right) \frac{\partial \mathbf{X}_i}{\partial \mathbf{x}_n}, \tag{2.21}$$

$$\left(\frac{\partial \mathbf{X}_i}{\partial \mathbf{x}_n}\right) = I + h \sum_{j=1}^{s} a_{ij} D\mathbf{f}\left(\mathbf{X}_j\right) \frac{\partial \mathbf{X}_j}{\partial \mathbf{x}_n}, \ 1 \leq i \leq s, \tag{2.22}$$

donde $D\mathbf{f}$ denota la matriz jacobiana de $\mathbf{f}$.

Si para $1 \leq i \leq s$, denotamos $\mathbf{D}_i = D\mathbf{f}\left(\mathbf{X}_i\right)$, e $\mathbf{Y}_i = \frac{\partial \mathbf{X}_i}{\partial \mathbf{x}_n}$ entonces vamos a ver que se verifica

$$J\mathbf{D}_i + \left(\mathbf{D}_i\right)^T J = 0. \tag{2.23}$$

Para demostrar esta igualdad, es importante recordar que si el sistema diferencial es Hamiltoniano $\mathbf{f} = J^{-1}\nabla H$ y que la matriz J cumple $(J^{-1})^T = J$ y $J^2 = -I$. Sustituyendo estas expresiones en la ecuación (2.20), se puede comprobar el resultado.

Para ello, empezamos a operar utilizando (2.21)

$$\begin{aligned}
\left(\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n}\right)^T J \left(\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n}\right) &= \left(I + h\sum_{i=1}^{s} b_i \mathbf{D}_i \mathbf{Y}_i\right)^T \left(J + h\sum_{j=1}^{s} b_j J \mathbf{D}_j \mathbf{Y}_j\right) \\
&= \left(I + h\sum_{i=1}^{s} b_i (\mathbf{D}_i \mathbf{Y}_i)^T\right) \left(J + h\sum_{j=1}^{s} b_j J \mathbf{D}_j \mathbf{Y}_j\right) \\
&= J + h\sum_{i=1}^{s} b_i (\mathbf{D}_i \mathbf{Y}_i)^T J + h\sum_{j=1}^{s} b_j J \mathbf{D}_j \mathbf{Y}_j \\
&\quad + \left(h\sum_{i=1}^{s} b_i (\mathbf{D}_i \mathbf{Y}_i)^T\right) \left(h\sum_{j=1}^{s} b_j J \mathbf{D}_j \mathbf{Y}_j\right) \\
&= J + h\left[\sum_{i=1}^{s} b_i \left[(\mathbf{D}_i \mathbf{Y}_i)^T J + J \mathbf{D}_i \mathbf{Y}_i\right]\right] \\
&\quad + h^2 \left[\left(\sum_{i=1}^{s} b_i (\mathbf{D}_i \mathbf{Y}_i)^T\right) J \left(\sum_{j=1}^{s} b_j \mathbf{D}_j \mathbf{Y}_j\right)\right].
\end{aligned}$$

Además, de (2.22) y usando la notación que hemos adoptado se sigue que

$$\mathbf{Y}_i = I + h\sum_{j=1}^{s} a_{ij} \mathbf{D}_j \mathbf{Y}_j$$

y, en consecuencia,

$$\begin{aligned}
(\mathbf{D}_i \mathbf{Y}_i)^T J \mathbf{Y}_i &= (\mathbf{D}_i \mathbf{Y}_i)^T J + h\sum_{j=1}^{s} a_{ij} (\mathbf{D}_i \mathbf{Y}_i)^T J \mathbf{D}_j \mathbf{Y}_j, \\
\mathbf{Y}_i^T J \mathbf{D}_i \mathbf{Y}_i &= J \mathbf{D}_i \mathbf{Y}_i + h\sum_{j=1}^{s} a_{ij} (\mathbf{D}_j \mathbf{Y}_j)^T J \mathbf{D}_i \mathbf{Y}_i.
\end{aligned}$$

De aquí, despejando el primer término de cada lado derecho, se deduce que

$$\begin{aligned}
(\mathbf{D}_i \mathbf{Y}_i)^T J &= (\mathbf{D}_i \mathbf{Y}_i)^T J \mathbf{Y}_i - h\sum_{j=1}^{s} a_{ij} (\mathbf{D}_i \mathbf{Y}_i)^T J \mathbf{D}_j \mathbf{Y}_j, \\
J \mathbf{D}_i \mathbf{Y}_i &= \mathbf{Y}_i^T J \mathbf{D}_i \mathbf{Y}_i - h\sum_{j=1}^{s} a_{ij} (\mathbf{D}_j \mathbf{Y}_j)^T J \mathbf{D}_i \mathbf{Y}_i.
\end{aligned}$$

Por último, de los cálculos anteriores y agrupando deducimos que

$$\begin{aligned}\left(\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n}\right)^T J \left(\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n}\right) &= J + h^2 \left[\left(\sum_{i=1}^{s} b_i (\mathbf{D}_i \mathbf{Y}_i)^T\right) J \left(\sum_{j=1}^{s} b_j \mathbf{D}_j \mathbf{Y}_j\right)\right] \\ &\quad + h \sum_{s=1}^{s} b_i \mathbf{Y}_i^T (\mathbf{D}_i^T J + J \mathbf{D}_i) \mathbf{Y}_i \\ &\quad - h \sum_{i=1}^{s} b_i \left[h \sum_{j=1}^{s} a_{ij} (\mathbf{D}_i \mathbf{Y}_i)^T J \mathbf{D}_j \mathbf{Y}_j \right. \\ &\quad \left. + h \sum_{j=1}^{s} a_{ij} (\mathbf{D}_j \mathbf{Y}_j)^T J \mathbf{D}_i \mathbf{Y}_i \right] \\ &= J + h^2 \sum_{i=1}^{s} \sum_{j=1}^{s} (b_i b_j - b_i a_{ij} - b_j a_{ji}) (\mathbf{D}_i \mathbf{Y}_i)^T J \mathbf{D}_j \mathbf{Y}_j \\ &= J.\end{aligned}$$

En la penúltima igualdad hemos usado que el tercer sumando es nulo por (2.23) y hemos cambiado los índices de algunos sumatorios i por j. La última igualdad es consecuencia de nuestra hipótesis (2.19) sobre los coeficientes del método Runge-Kutta.

Por tanto, hemos demostrado

$$\left(\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n}\right)^T J \left(\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n}\right) = J,$$

que es lo que queríamos ver. □

En [6] se prueba que las condiciones (2.19) son también necesarias para que un método Runge-Kutta sin etapas redundantes sea simpléctico, lo que en esta memoria cuando hablemos de métodos Runge-Kutta simplécticos nos referiremos a aquellos cuyos coeficientes satisfacen (2.19).

Corolario 1. *No existen métodos Runge-Kutta simplécticos y explícitos.*

Demostración. Si un método Runge-Kutta es explícito, se tiene que $a_{ij} = 0$ para $i \leq j$. En particular, $a_{ii} = 0$ para $1 \leq i \leq s$, y si los coeficientes del método satisfacen (2.19) se deduce que $b_i^2 = 0$, es decir, $b_i = 0$ para $i = 1, ..., s$, y en consecuencia $\mathbf{x}_{n+1} = \mathbf{x}_n$. Llegamos entonces a un absurdo pues el método no sería consistente. □

2.3. Métodos Runge-Kutta simétricos

En esta subsección restringiremos nuestra atención a la familia de *sistemas diferenciales reversibles*, que serán definidas convenientemente, y a la propiedad de *simetría* de los métodos Runge-Kutta.

Definición 9. *Sea ρ una transformación lineal e invertible en $\mathbb{R}^d$. Un sistema de ecuaciones diferenciales autónomo $\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t))$es reversible (o ρ-reversible) si*

$$\rho\,\mathbf{f}(\mathbf{x}) = -\mathbf{f}(\rho\,\mathbf{x}),\ \textit{para todo}\ \mathbf{x} \in \mathbb{R}^d. \tag{2.24}$$

Un primer ejemplo de sistema reversible es el sistema Hamiltoniano con un grado de libertad y función Hamiltoniana $H = \frac{1}{2}p^2 + V(q)$, (siendo V una función potencial arbitraria), respecto de la transformación lineal $\rho(p, q) = (-p, q)$.

Definición 10. *Una aplicación ϕ es ρ-reversible si*

$$\rho \circ \phi = \phi^{-1} \circ \rho$$

.

Teorema 3. *El flujo Φ_t de un sistema diferencial autónomo reversible verifica que*

$$\rho \circ \Phi_t = \Phi_{-t} \circ \rho = \Phi_t^{-1} \circ \rho. \tag{2.25}$$

Demostración. Para demostrar la primera igualdad, derivamos las dos expresiones y usamos (2.24) para obtener

$$\begin{aligned} \frac{d}{dt}(\rho \circ \Phi_t)(\mathbf{x}) &= \rho\,\mathbf{f}(\Phi_t(\mathbf{x})) = -\mathbf{f}((\rho \circ \Phi_t)(\mathbf{x})), \\ \frac{d}{dt}(\Phi_{-t} \circ \rho)(\mathbf{x}) &= -\mathbf{f}((\Phi_{-t} \circ \rho)(\mathbf{x})) \end{aligned}$$

Como ambas expresiones, $\rho \circ \Phi_t$ y $\Phi_{-t} \circ \rho$ son solución de la misma ecuación diferencial, con misma condición inicial en $t = 0$, la igualdad es cierta para todo t.

Finalmente, la segunda igualdad está justificada por las propiedades de grupo del flujo. Puesto que $\Phi_t \circ \Phi_s = \Phi_{t+s}$, y en particular, $\Phi_t \circ \Phi_{-t}$ es el operador identidad, entonces se tiene $\Phi_{-t} = \Phi_t^{-1}$. □

En la Figura 2.2 se ilustra el resultado del Teorema 3 cuando ρ es la simetría respecto del eje horizontal.

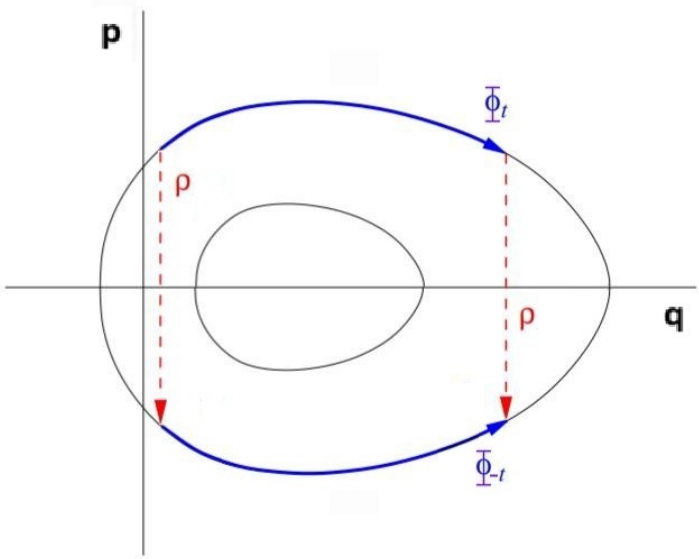


Figura 2.2: Ejemplo para ilustrar (2.25).

2.3.1. Adjunto de un método

Definición 11. *Dado un método de un paso* $\mathbf{x}_{n+1} = \Psi_h(\mathbf{x}_n)$*, su método adjunto,* Ψ_h^**, es aquel que satisface*

$$\mathbf{x}_n = \Psi_{-h}(\mathbf{x}_{n+1}) \iff \mathbf{x}_{n+1} = \Psi_h^*(\mathbf{x}_n). \tag{2.26}$$

La expresión del adjunto de un método de un paso se obtiene intercambiando n *por* $n+1$ *y* h *por* $-h$ *en las ecuaciones que describen el método original, y despejando de dichas ecuaciones el término* $\mathbf{x}_{n+1}$*.*

Ahora vamos a ver con un ejemplo sencillo cómo podemos calcular el adjunto de un método Runge-Kutta. Partimos del método de Euler explícito.

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{f}(t_n, \mathbf{x}_n).$$

Intercambiando como hemos dicho anteriormente, n por $n+1$ y h por $-h$, obtenemos

$$\mathbf{x}_n = \mathbf{x}_{n+1} - h\mathbf{f}(t_{n+1}, \mathbf{x}_{n+1}).$$

Finalmente, despejando $\mathbf{x}_{n+1}$ conseguimos

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{f}(t_{n+1}, \mathbf{x}_{n+1}),$$

que resulta ser el método de Euler implícito. Por lo tanto, acabamos de probar que el adjunto del método de Euler explícito es el método de Euler implícito, y viceversa, ya que el adjunto del adjunto de un método es el propio método, es decir, $(\Psi_h^*)^* = \Psi_h$.

Definición 12. *Un método de un paso* $\mathbf{x}_{n+1} = \Psi_h(\mathbf{x}_n)$ *es simétrico (o autoadjunto) si coincide con su adjunto, es decir, si*

$$\Psi_h = \Psi_h^*. \tag{2.27}$$

Nota 3: Equivalentemente, se puede decir que un método es simétrico si $\Psi_h = (\Psi_{-h})^{-1}$, o bien, si $\Psi_h \circ \Psi_{-h}$ es la identidad.

Una vez vistas las definiciones de reversibilidad y simetría, podemos ya enunciar la relación que hay entre ambas.

Teorema 4. *Si un método numérico de un paso aplicado a un sistema diferencial* ρ*-reversible satisface*

$$\rho \circ \Psi_h = \Psi_{-h} \circ \rho \tag{2.28}$$

entonces el flujo numérico Ψ_h *es una aplicación* ρ*-reversible si, y solo si, el método es simétrico.*

Demostración. Como consecuencia de (2.28) y de la Definición 10, el flujo numérico es una aplicación ρ-reversible si, y solo si, $\Psi_{-h} \circ \rho = \Psi_h^{-1} \circ \rho$. Como ρ es una transformación invertible por definición, aplicando ρ^{-1} por la derecha a la igualdad anterior, nos queda $\Psi_{-h} = \Psi_h^{-1}$ que, según se ha visto en la Nota 2.3.1, equivale a la simetría del método. □

Nota 4: Un buen número de métodos de un paso verifican (2.28), como se puede ver en [6].

Para finalizar esta sección, vamos a caracterizar los métodos Runge-Kutta simétricos en términos de sus coeficientes.

Teorema 5. *Si los coeficientes de un método Runge-Kutta con tablero de Butcher* (2.7) *satisfacen*

$$a_{ij} = b_j - a_{s+1-i,s+1-j}, \qquad 1 \leq i, j \leq s, \tag{2.29}$$

$$b_i = b_{s+1-i}, \qquad 1 \leq i \leq s, \tag{2.30}$$

entonces el método es simétrico. Además, podemos deducir que $1 - c_i = c_{s+1-i}$, $1 \leq i \leq s$.

Demostración. Tenemos que ver que $\Psi_h = \Psi_h^*$ para poder concluir que el método es simétrico. Basta entonces comprobar que, si denotamos por a_{ij}^* y b_i^* a los coeficientes del método adjunto, se cumplen las relaciones $a_{ij}^* = a_{ij}$ y $b_i^* = b_i$. Empezamos construyendo el adjunto de un método Runge-Kutta y viendo que es también un método Runge-Kutta. Para ello, intercambiamos $n+1$ por n y h por $-h$ en las ecuaciones que describen el método (2.5) y (2.6). Como resultado, obtenemos

$$\begin{aligned} \mathbf{X}_i &= \mathbf{x}_{n+1} - h\sum_{j=1}^{s} a_{ij}\mathbf{f}(t_{n+1} - c_j h, \mathbf{X}_j), \\ \mathbf{x}_n &= \mathbf{x}_{n+1} - h\sum_{i=1}^{s} b_i\mathbf{f}(t_{n+1} - c_i h, \mathbf{X}_i). \end{aligned}$$

Puesto que $t_n = t_{n+1} - h$, si restamos la segunda ecuación a la primera, y en la segunda despejamos $\mathbf{x}_{n+1}$, nos queda

$$\begin{aligned} \mathbf{X}_i &= \mathbf{x}_n + h\sum_{j=1}^{s}(b_j - a_{ij})\mathbf{f}(t_n + (1-c_j)h, \mathbf{X}_j), \\ \mathbf{x}_{n+1} &= \mathbf{x}_n + h\sum_{i=1}^{s} b_i\mathbf{f}(t_n + (1-c_i)h, \mathbf{X}_i). \end{aligned}$$

Reemplazando los índices i, j por $s+1-i$, y por $s+1-j$, respectivamente, tendríamos finalmente

$$a_{ij}^* = b_{s+1-j} - a_{s+1-i,s+1-j}, \tag{2.31}$$
$$b_i^* = b_{s+1-i}. \tag{2.32}$$

Aplicando nuestras hipótesis conseguimos ya que $b_i = b_i^*$. En particular, como $b_j = b_{s+1-j}$, de (2.31) deducimos que $a_{ij} = a_{ij}^*$.

Para la última parte, basta recordar que

$$\begin{aligned} 1 - c_i &= \sum_{j=1}^{s}(b_j - a_{ij}) = \sum_{j=1}^{s}(b_{s+1-j} - a_{i,s+1-j}) \\ &= \sum_{j=1}^{s}(b_j - a_{i,s+1-j}) = \sum_{j=1}^{s} a_{s+1-i,j} = c_{s+1-i}, \end{aligned}$$

ya que en la segunda igualdad hemos reordenado los términos del sumatorio para después usar las hipótesis (2.30) y (2.29) en las siguientes igualdades respectivamente. □

Respecto de la propiedad de simetría de los métodos Runge-Kutta, se puede establecer un resultado análogo al Corolario 1. Tampoco existen métodos Runge-Kutta explícitos y simétricos [6].

Capítulo 3

Los métodos de Gauss

Vamos a estudiar los métodos de Gauss-Legendre, con algunas de sus propiedades y características, como la simetría y la simplecticidad, ya que estos métodos son grandes candidatos para realizar simulaciones a tiempos largos de sistemas Hamiltonianos.

3.1. Métodos de colocación

Los métodos de colocación son usados para desarrollar métodos Runge-Kutta implícitos. A continuación vamos a explicar la idea que hay detrás hasta llegar a lo que se conoce como *los métodos de Gauss*.

Definición 13. *Sea s un número entero positivo y sean $c_1, ..., c_s$ números reales distintos. El correspondiente polinomio de colocación $\mathbf{u}(t)$ es un polinomio de grado menor o igual que s que cumple*

- $\mathbf{u}(t_n) = \mathbf{x}_n$.
- $\mathbf{u}(t)$ *satisface el sistema de ecuaciones diferenciales* (2.1) *en los puntos* $t_n + c_i h$ *para* $i = 1, ..., s$. *Es decir,*

$$\mathbf{u}'(t_n + c_i h) = \mathbf{f}(t_n + c_i h, \mathbf{u}(t_n + c_i h)). \tag{3.1}$$

- *Se puede tomar como la solución numérica de* (2.1) *en* $t_{n+1} = t_n + h$ *la definida como* $\mathbf{x}_{n+1} = \mathbf{u}(t_{n+1})$.

A partir del *polinomio de colocación* se obtiene un procedimiento para aproximar la solución de (2.1). Vamos a ver que dicho procedimiento no es más que un método Runge-Kutta con coeficientes a_{ij} y b_i apropiados.

Teorema 6. *El método de colocación construido según se ha indicado en la Definición 13 es equivalente a un método de Runge-Kutta de s etapas dado por* (2.2), (2.3), *y* (2.4), *con coeficientes*

$$a_{ij} = \int_0^{c_i} l_j(y)dy, \qquad b_i = \int_0^1 l_i(y)dy, \qquad 1 \leq i, j \leq s, \tag{3.2}$$

donde $l_i(y)$ *es el polinomio de la base de Lagrange, asociado al nodo* c_i, *es decir,*

$$l_i(y) = \prod_{p \neq i} \frac{y - c_p}{c_i - c_p}.$$

Demostración. Sea $\mathbf{u}(t)$ el polinomio de colocación. Particularizando en (3.1) en $n = 0$, se define, para $1 \leq i \leq s$,

$$\mathbf{k}_i = \mathbf{u}'(t_0 + c_i h). \tag{3.3}$$

Por la fórmula de interpolación de Lagrange, se puede escribir

$$\mathbf{u}'(t_0 + yh) = \sum_{j=1}^{s} \mathbf{k}_j \cdot l_j(y),$$

e integrando entre 0 y c_i obtenemos

$$\mathbf{u}(t_0 + c_i h) = \mathbf{x}_0 + \int_0^{c_i} \mathbf{u}'(t_0 + yh)dy = \mathbf{x}_0 + h \sum_{j=1}^{s} \mathbf{k}_j \int_0^{c_i} l_j(y)dy.$$

Relacionando las ecuaciones (3.3) y (3.1), además de tener en cuenta (2.2), llegamos a ver que el término de la última igualdad que acabamos de obtener es igual al segundo parámetro de la función $\mathbf{f}$ en (2.2), quedando demostrado el teorema para los coeficientes a_{ij}. Para los b_i basta repetir el proceso integrando entre 0 y 1. □

Dado que los métodos de colocación introducidos en la Definción 13 se han construido mediante un polinomio interpolador de grado menor o igual que s, el método Runge-Kutta de s etapas equivalente tiene orden mayor

o igual que s, con independencia de la elección de las abscisas distintas $c_1, ..., c_s$. Sin embargo, se pueden elegir dichas abscisas para que con el mismo número de etapas se consiga orden estrictamente mayor que s.

El método de Gauss de s etapas es el método Runge-Kutta que se obtiene a partir del método de colocación cuyas abscisas $c_1, ..., c_s$ son precisamente los ceros del polinomio de Legendre trasladado de grado s,

$$P_s(x) = \frac{d^s}{dx^s}\left[x^s(x-1)^s\right]. \tag{3.4}$$

Dichas abscisas no son otras que nodos de la fórmula de cuadratura Gaussiana.

Aunque no lo demostraremos en esta memoria, el método de Gauss de s etapas tiene orden $2s$, el máximo orden que puede tener un método Runge-Kutta de s etapas.

3.2. Coeficientes de los primeros métodos de Gauss

Ahora procederemos a mostrar los tableros de coeficientes de los métodos de Gauss de 2 y 4 etapas (con órdenes 4 y 8, respectivamente). El método de Gauss de 1 etapa es la regla implícita del punto medio, vista en la Sección 2.1.3, que como dijimos, tiene orden 2.

- **Tablero de coeficientes del método de Gauss de orden 4**

$$\begin{array}{c|cc} \frac{3-\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4}-\frac{\sqrt{3}}{6} \\ \frac{3+\sqrt{3}}{6} & \frac{1}{4}+\frac{\sqrt{3}}{6} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

- **Tablero de coeficientes del método de Gauss de orden 8**

$\frac{1}{2}-w_2$	w_1	$w_1^*-w_3+w_4^*$	$w_1^*-w_3-w_4^*$	w_1-w_5
$\frac{1}{2}-w_2^*$	$w_1-w_3^*+w_4$	w_1^*	$w_1^*-w_5^*$	$w_1-w_3^*-w_4$
$\frac{1}{2}+w_2^*$	$w_1+w_3^*+w_4$	$w_1^*+w_5^*$	w_1^*	$w_1+w_3^*-w_4$
$\frac{1}{2}+w_2$	w_1+w_5	$w_1^*+w_3+w_4^*$	$w_1^*+w_3-w_4^*$	w_1
	$2w_1$	$2w_1^*$	$2w_1^*$	$2w_1$

donde

$$
\begin{aligned}
w_1 &= \frac{1}{8}-\frac{\sqrt{30}}{144}, & w_1^* &= \frac{1}{8}+\frac{\sqrt{30}}{144},\\
w_2 &= \frac{1}{2}\left(\frac{15+2\sqrt{30}}{35}\right)^{1/2}, & w_2^* &= \frac{1}{2}\left(\frac{15-2\sqrt{30}}{35}\right)^{1/2},\\
w_3 &= w_2\left(\frac{1}{6}+\frac{\sqrt{30}}{24}\right), & w_3^* &= w_2^*\left(\frac{1}{6}-\frac{\sqrt{30}}{24}\right),\\
w_4 &= w_2\left(\frac{1}{21}+\frac{5\sqrt{30}}{168}\right), & w_4^* &= w_2^*\left(\frac{1}{21}-\frac{5\sqrt{30}}{168}\right),\\
w_5 &= w_2-2w_3, & w_5 &= w_2^*-2w_3^*.
\end{aligned}
$$

3.3. Los métodos de Gauss y la simplecticidad

Una de las propiedades más destacadas de los métodos de Gauss es su carácter simpléctico, lo que les convierte, como veremos, en claros candidatos para las simulaciones a largo plazo de sistemas Hamiltonianos.

En el estudio de las condiciones de orden de los métodos Runge-Kutta

juegan un papel destacado las siguientes *condiciones simplificadoras*:

$$B(p) \;:\; \sum_{i=1}^{s} b_i c_i^{k-1} = \frac{1}{k}, \qquad k = 1, ..., p, \tag{3.5}$$

$$C(\eta) \;:\; \sum_{j=1}^{s} a_{ij} c_j^{k-1} = \frac{c_i^k}{k}, \qquad k = 1, ..., \eta, \tag{3.6}$$

$$D(\zeta) \;:\; \sum_{i=1}^{s} b_i a_{ij} c_i^{k-1} = \frac{b_j(1 - c_j^k)}{k}, \qquad j = 1, ..., s \quad k = 1, ..., \zeta. \tag{3.7}$$

donde los coeficientes a_{ij}, b_i, y c_i son los definidos en (2.7) que, en el caso de los métodos de Gauss vienen dados por (3.2). Es fácil reconocer en (3.5) las condiciones que deben cumplir las abscisas y los pesos de una fórmula de cuadratura para tener grado de exactitud p. En el caso de las fórmulas de cuadratura Gaussiana de s nodos, cuyas abscisas son los ceros de (3.1), se cumple $B(s)$. Más precisamente, se tiene el siguiente resultado, que incluimos sin demostración, y que nos ayudará a establecer de manera bastante sencilla que los coeficientes de los métodos de Gauss satisfacen las condiciones (2.19). La demostración puede verse en [4].

Teorema 7. *Los métodos de Gauss satisfacen las condiciones simplificadoras $B(2s)$, $C(s)$, y $D(s)$, definidas en* (3.5), (3.6) *y* (3.7).

A continuación, podemos enunciar y demostrar el teorema fundamental de esta sección.

Teorema 8. *Para cada $s = 1, 2, ...$ el método de Gauss de s etapas es simpléctico.*

Demostración. Queremos ver que los coeficientes del método de Gauss de s etapas satisfacen

$$b_i a_{ij} + b_j a_{ji} - b_i b_j = 0, \qquad 1 \leq i, j \leq s.$$

Para ello, notamos que por cumplirse $D(s)$,

$$\sum_{i=1}^{s} b_i a_{ij} c_i^{k-1} = \frac{b_j(1 - c_j^k)}{k}, \qquad 1 \leq j, k \leq s. \tag{3.8}$$

Por otro lado, utilizando $C(s)$ se tiene que para $1 \leq k \leq s$,

$$\sum_{j=1}^{s} b_i a_{ij} c_j^{k-1} = \frac{b_i c_i^k}{k}, \qquad 1 \leq k \leq s,$$

intercambiando los papeles de i y j en la igualdad anterior obtenemos,

$$\sum_{i=1}^{s} b_j a_{ji} c_i^{k-1} = \frac{b_j c_j^{k-1}}{k}, \qquad 1 \leq k \leq s. \tag{3.9}$$

Por último, la condición $B(s)$ implica que

$$\sum_{i=1}^{s} b_j b_i c_i^{k-1} = \frac{b_j}{k}, \qquad 1 \leq k \leq s. \tag{3.10}$$

Restando (3.9) a (3.10) e igualando con (3.8) llegamos a

$$\sum_{i=1}^{s} b_i a_{ij} c_i^{k-1} = \sum_{i=1}^{s} b_j b_i c_i^{k-1} - \sum_{i=1}^{s} b_j a_{ji} c_i^{k-1}, \tag{3.11}$$

que se puede reescribir como

$$\sum_{i=1}^{s} (b_i a_{ij} + b_j a_{ji} - b_i b_j) c_i^{k-1} = 0, \qquad 1 \leq j, k \leq s. \tag{3.12}$$

Llamando M a la matriz de tamaño $s \times s$ cuyo elemento (i, j) es $m_{ij} = b_i a_{ij} + b_j a_{ji} - b_i b_j$, siendo $\mathbf{c}^{(k-1)} = [c_1^{k-1}, ..., c_s^{k-1}]^T$, y V a la traspuesta de Vandermonde asociada a los nodos $c_1, ..., c_s$, las relaciones (3.12) se pueden escribir como

$$V \cdot M = 0_{s \times s}$$

y al ser V invertible por ser $c_1, .., c_s$ distintos dos a dos, se supone que $M \equiv 0$, y, en consecuencia, el método es simpléctico en virtud del Teorema 2. □

3.4. Los métodos de Gauss y la simetría

Resulta que los métodos de Gauss también son métodos simétricos, pero veamos antes un teorema que nos ayudará para entenderlo.

Teorema 9. *El método adjunto de un método de colocación basado en las abscisas $c_1, ..., c_s$ es un método de colocación basado en las abscisas $c_1^*, ..., c_s^*$ dadas por*

$$c_i^* = 1 - c_{s+1-i}, \qquad 1 \leq i \leq s. \tag{3.13}$$

Además, si $c_i = 1 - c_{s+1-i}$ para $1 \leq i \leq s$, entonces el método es simétrico.

Demostración. Si en la Definición 13, al construir el polinomio de colocación, intercambiamos t_n por t_{n+1}, $\mathbf{x}_n$ por $\mathbf{x}_{n+1}$, y h por $-h$ obtenemos un nuevo método de colocación que viene dado por las condiciones

- $\mathbf{u}(t_{n+1}) = \mathbf{x}_{n+1}$
- $\mathbf{u}'(t_{n+1} - c_i h) = \mathbf{f}(t_{n+1} - c_i h, \mathbf{u}(t_{n+1} - c_i h)), \qquad 1 \leq i \leq s,$
- $\mathbf{x}_n = \mathbf{u}(t_n)$.

Teniendo en cuenta que $t_{n+1} - c_i h = t_n + h - c_i h = t_n + h(1 - c_i)$, las tres relaciones anteriores se pueden reescribir como

- $\mathbf{u}(t_n) = \mathbf{x}_n$
- $\mathbf{u}'(t_n + h(1 - c_i)) = \mathbf{f}(t_n + h(1 - c_i), \mathbf{u}(t_n + h(1 - c_i))), \qquad 1 \leq i \leq s,$
- $\mathbf{x}_{n+1} = \mathbf{u}(t_{n+1})$.

Reemplazando según (3.13) $1 - c_i$ por c_{s+1-i}^*, para $1 \leq i \leq s$, y teniendo en cuenta que los nodos de la interpolación c_i^* pueden reordenarse arbitrariamente, concluimos la demostración. □

Como consecuencia del teorema anterior y haciendo uso únicamente del hecho de que los métodos de Gauss son métodos de colocación, podemos concluir con el siguiente corolario.

Corolario 2. *Para cada $s = 1, 2, ...$ el método de Gauss de s etapas es simétrico.*

Demostración. Sabemos que los coeficientes c_i del método de Gauss de s etapas son los ceros del polinomio de Legendre de grado s. Basta entonces ver que cumplen la propiedad $c_i = 1 - c_{s+1-i}$, o equivalentemente, $1 - c_i = c_{s+1-i}$ para $1 \leq i \leq s$.

Sea c_i un cero del polinomio de Legendre trasladado de grado s, es decir que $P_s(c_i) = 0$. Veamos ahora que $P_s(x) = (-1)^s P_s(1-x)$. Para ello, recordemos que el polinomio de Legendre trasladado de grado s estaba definido en (3.4). Si le aplicamos el cambio de variable, $x = 1-y \leftrightarrow y = 1-x$, con $\frac{d}{dx} = -\frac{d}{dy}$, entonces

$$\begin{aligned} P_s(x) &= \frac{d^s}{dx^s}\left[x^s(x-1)^s\right] \\ &= (-1)^s \frac{d^s}{dy^s}\left[(1-y)^s(-y)^s\right] \\ &= (-1)^s \frac{d^s}{dy^s}\left[(-1)^s(y-1)^s(-1)^s y^s\right] \\ &= (-1)^s \frac{d^s}{dy^s}\left[(y-1)^s y^s\right] = (-1)^s P_s(y) = (-1)^s P_s(1-x). \end{aligned}$$

Dado que hemos supuesto que $P_s(c_i) = 0$ y hemos demostrado que $P_s(x) = (-1)^s P_s(1-x)$, deducimos que $P_s(1-c_i) = 0$. Por tanto, $1 - c_i$ es también un cero del polinomio de Legendre trasladado de grado s, y por simetría podemos concluir que $1 - c_i = c_{s+1-i}$, para $1 \leq i \leq s$. Aplicando el Teorema 9 aseguramos que el método de Gauss de s etapas es simétrico. □

Capítulo 4

Implementación básica

En este capítulo empezaremos a implementar los métodos numéricos, empezando con versiones básicas para más adelante ir aplicando algunas mejoras que se proponen en [1], en relación con la teoría que hemos visto en los capítulos anteriores.

El problema que estudiaremos será el problema de Kepler, que describe el movimiento de un cuerpo (en nuestro caso, planetas), bajo la influencia gravitatoria de otros cuerpos como el Sol y otros planetas. Lo resolveremos con métodos de Gauss de distintos órdenes, utilizando para resolver las ecuaciones no lineales el método de Newton y la iteración de punto fijo, para después comparar los resultados obtenidos.

4.1. Reformulación de las ecuaciones

Queremos reducir la influencia de los errores de redondeo, por ello, consideramos, en vez de las etapas intermedias $\mathbf{X}_i$, $1 \leq i \leq s$, los incrementos

$$\mathbf{Z}_i = \mathbf{X}_i - \mathbf{x}_n, \qquad 1 \leq i \leq s. \tag{4.1}$$

Así, las ecuaciones (2.5) de las etapas intermedias quedarían reescritas de la siguiente forma

$$\mathbf{Z}_i = h \sum_{j=1}^{s} a_{ij} \mathbf{f}(t_n + c_j h, \mathbf{x}_n + \mathbf{Z}_j), \qquad 1 \leq i \leq s. \tag{4.2}$$

Cuando una solución $\mathbf{Z}_i$ con $i = 1, ..., s$ es encontrada, entonces aplicar (2.6) para obtener $\mathbf{x}_{n+1}$ requiere s evaluaciones de $\mathbf{f}$. Podemos evitar esto si la matriz $A = (a_{ij})$ de los coeficientes del método Runge-Kutta es invertible. De hecho, (4.2) puede escribirse como

$$\begin{bmatrix} \mathbf{Z}_1 \\ \vdots \\ \mathbf{Z}_s \end{bmatrix} = h(A \otimes I) \begin{bmatrix} \mathbf{f}(t_n + c_1 h, \mathbf{x}_n + \mathbf{Z}_1) \\ \vdots \\ \mathbf{f}(t_n + c_s h, \mathbf{x}_n + \mathbf{Z}_s) \end{bmatrix}, \tag{4.3}$$

convirtiéndose (2.6) en

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \sum_{i=1}^{s} d_i \mathbf{Z}_i, \tag{4.4}$$

con $(d_1, ..., d_s) = (b_1, ..., b_s)A^{-1}$. Nótese que $\otimes$ denota el producto de Kronecker de matrices.

4.2. Implementación: iteración de Newton

La aplicación del método de Newton para resolver (4.3) conlleva la solución de un sistema lineal en cada iteración con matriz de la forma

$$\begin{bmatrix} I - ha_{11}J_1^{[k]} & \dots & -ha_{1s}J_s^{[k]} \\ \dots & \dots & \dots \\ -ha_{s1}J_1^{[k]} & \dots & I - ha_{ss}J_s^{[k]} \end{bmatrix}, \tag{4.5}$$

donde $J_i^{[k]}$ denota la matriz jacobiana de $\mathbf{f}$ respecto de $\mathbf{x}$ evaluada en $(t_n + c_i h, \mathbf{x}_n + \mathbf{Z}_i^{[k]})$. Seguiríamos este esquema

$$(I - hA \otimes J^{[k]})\Delta\mathcal{Z}^{[k]} = -\mathcal{Z}^{[k]} + h(A \otimes I)\mathcal{F}(\mathcal{Z}^{[k]}), \tag{4.6}$$

$$\mathcal{Z}^{[k+1]} = \mathcal{Z}^{[k]} + \Delta\mathcal{Z}^{[k]}. \tag{4.7}$$

Hemos denotado por $\mathcal{Z}^{[k]}$ al vector de $\mathbb{R}^{D\times s}$ que se tiene al agrupar los s vectores $\mathbf{Z}_i^{[k]}$ de dimensión D. De manera similar, $\mathcal{F}(\mathcal{Z}^{[k]})$ es el vector que agrupa los s vectores D-dimensionales $\mathbf{f}(t_n + c_i h, \mathbf{x}_n + \mathbf{Z}_i^{[k]})$.

Para poder utilizar una única factorización LU de la matriz (4.5) en todas las iteraciones realizadas en un paso, en vez de actualizar en cada

iteración el jacobiano $J_i^{[k]}$, lo que conllevaría un aumento del coste computacional, éste es sustituido por

$$J \approx \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(t_n, \mathbf{x}_n),$$

lo que da lugar al método conocido como iteración de Newton simplificada.

4.3. Implementación: iteración de punto fijo

Siguiendo el esquema de (4.2), y denotando con superíndice k la iteración en la que estamos, la iteración de punto fijo quedaría

$$\mathbf{Z}_i^{[k]} = h\sum_{j=1}^{s} a_{ij}\mathbf{f}(t_n + c_j h, \mathbf{x}_n + \mathbf{Z}_j^{[k-1]}), \qquad 1 \leq i \leq s. \tag{4.8}$$

Finalmente, para avanzar un paso se utiliza (4.4) con el resultado final que haya proporcionado al parar la iteración.

Sin embargo, vamos a implementar de base una mejora que encontramos en [2]. Con este nuevo procedimiento lo que se pretende es replicar de manera más precisa el caracter simpléctico de los métodos de Gauss cuando se trabaja con aritmética finita. Esto es debido a que los coeficientes a_{ij} y b_i almacenados en el ordenador no cumplen de manera exacta la condición de simplecticidad (2.19). Se introducen los nuevos coeficientes μ_{ij} dados por $\mu_{ij} = a_{ij}/b_j$, con $1 \leq i, j \leq s$, y se reescriben las ecuaciones para avanzar un paso en términos de las etapas intermedias de la siguiente forma

$$\mathbf{X}_i = \mathbf{x}_n + \sum_{j=1}^{s} \mu_{ij}, \mathbf{L}_j \qquad i = 1, ..., s, \tag{4.9}$$

$$\mathbf{L}_i = hb_i\mathbf{f}(t_n + c_i h, \mathbf{X}_i), \qquad i = 1, ..., s, \tag{4.10}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \sum_{i=1}^{s} \mathbf{L}_i. \tag{4.11}$$

La iteración de punto fijo aplicada a (4.9) y (4.10) proporciona aproximaciones $\mathbf{X}_i^{[k]}$ y $\mathbf{L}_i^{[k]}$ a $\mathbf{X}_i$ y $\mathbf{L}_i$ siguiendo el esquema

$$\mathbf{L}_i^{[k]} = hb_i\mathbf{f}(t_n + c_i h, \mathbf{X}_i^{[k-1]}), \qquad i = 1, ..., s \tag{4.12}$$

$$\mathbf{X}_i^{[k]} = \mathbf{x}_n + \sum_{j=1}^{s} \mu_{ij}\mathbf{L}_j^{[k]}, \qquad i = 1, ..., s. \tag{4.13}$$

4.4. Fin de las iteraciones

En el caso del método de Newton, para el primer iterante de cada paso se ha optado por el vector nulo de $\mathbb{R}^D$ ($\mathbf{Z}_i^{[0]} = \mathbf{0}$) para todo $i = 1, ..., s$, mientras que para la iteración de punto fijo se ha definido $\mathbf{X}_i^{[0]} = \mathbf{x}_n$, con $i = 1, ..., s$.

Para detener las iteraciones en el método de Newton simplificado o en la iteración de punto fijo (4.8), utilizando el hecho de que la convergencia de los iterantes es lineal, se puede pensar que

$$\left\|\Delta\mathcal{Z}^{[k+1]}\right\| \leq \Theta \left\|\Delta\mathcal{Z}^{[k]}\right\|,$$

con $0 < \Theta < 1$, siendo $\Delta\mathcal{Z}^{[k]} = \mathcal{Z}^{[k+1]} - \mathcal{Z}^{[k]}$. De hecho, si representamos la solución exacta del sistema (4.2) como $\mathcal{Z}$, entonces el error al aproximar $\mathcal{Z}$ por el iterante $k+1$ satisface

$$\begin{aligned}
\left\|\mathcal{Z}^{[k+1]} - \mathcal{Z}\right\| &\leq \left\|\left(\mathcal{Z}^{[k+1]} - \mathcal{Z}^{[k+2]}\right)\right\| + \left\|\left(\mathcal{Z}^{[k+2]} - \mathcal{Z}^{[k+3]}\right)\right\| + \cdots \\
&\leq \left\|\Delta\mathcal{Z}^{[k+1]}\right\| + \left\|\Delta\mathcal{Z}^{[k+2]}\right\| + \cdots \\
&\leq \left(\Theta + \Theta^2 + \cdots + \Theta^k + \cdots\right) \left\|\Delta\mathcal{Z}^{[k]}\right\| = \frac{\Theta}{1-\Theta} \left\|\Delta\mathcal{Z}^{[k]}\right\|.
\end{aligned}$$

El factor de convergencia Θ puede ser estimado en cada iteración mediante el cociente

$$\Theta_k = \frac{\left\|\Delta\mathcal{Z}^{[k]}\right\|}{\left\|\Delta\mathcal{Z}^{[k-1]}\right\|}, \qquad k = 1, 2, ...$$

Por tanto, las iteraciones pueden pararse cuando

$$\frac{\Theta_k}{1-\Theta_k} \left\|\Delta\mathcal{Z}^{[k]}\right\| \leq TOL,$$

donde TOL es una tolerancia dada. Nótese que este criterio de parada necesita como mínimo dos iteraciones y que la norma usada para determinar el fin de las iteraciones es la euclídea (aunque podría ser cualquier otra).

Sin embargo, en nuestros experimentos numéricos pararemos la iteración de Newton (4.6) – (4.7) cuando $\left\|\Delta\mathcal{Z}^{[k]}\right\| \leq TOL$. En la iteración de punto fijo (4.12) – (4.13), siguiendo [2], el proceso iterativo se detiene cuando $\|\Delta\mathcal{X}^{[k]}\| \leq TOL$, siendo

$$\Delta\mathcal{X}^{[k]} = \left(\mathbf{X}_1^{[k]} - \mathbf{X}_1^{[k-1]}, ..., \mathbf{X}_s^{[k]} - \mathbf{X}_s^{[k-1]}\right). \tag{4.14}$$

Nótese que una forma más eficiente de calcular $\Delta\mathcal{X}^{[k]}$ es utilizando que

$$\mathbf{X}_i^{[k]} - \mathbf{X}_i^{[k-1]} = \sum_{j=1}^{s} \mu_{ij} \left(\mathbf{L}_j^{[k]} - \mathbf{L}_j^{[k-1]} \right), \qquad i = 1, ..., s.$$

De este modo no tenemos que sumar y restar el término $\mathbf{x}_n$ que aparece en (4.13).

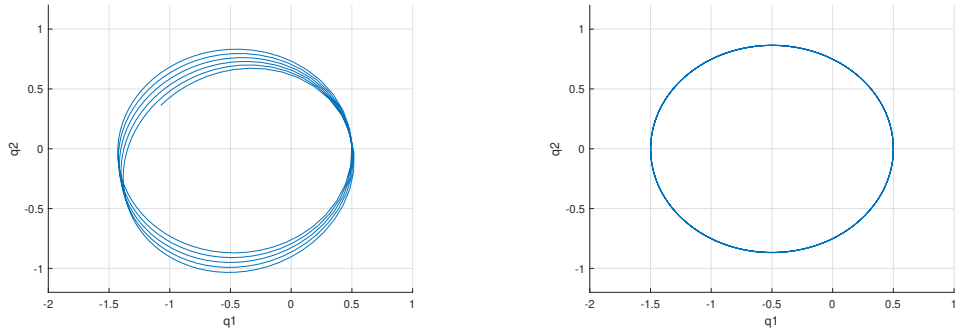
Finalmente, tanto en el método de Newton, como con la iteración de punto fijo, además de la tolerancia, se define un parámetros adicional que es el número máximo de iteraciones, *maxiter* para detener la iteración en el caso de que no converja.

4.5. Resultados numéricos

Se ha realizado la integración numérica del problema de Kepler (2.15) con condición inicial el vector $\mathbf{x}_0 = \left[0, \quad \sqrt{\frac{1+e}{1-e}}, \quad 1-e, \quad 0\right]^T$, donde e es la excentricidad de la órbita ($0 \leq e < 1$). Con esta condición inicial sabemos que la solución exacta es 2π-periódica y, por tanto, es fácil medir errores en tiempos que sean múltiplos enteros de 2π.

Se han usado cuatro algoritmos distintos: dos con la iteración de punto fijo y otros dos con la iteración de Newton. Los métodos Runge-Kutta utilizados han sido la regla implícita del punto medio (el método de Gauss de una etapa y orden dos) y el método de Gauss de dos etapas y orden cuatro.

En las Figuras 4.1a y 4.1b hemos querido ilustrar el comportamiento periódico de la solución de (2.15) con la condición inicial dada, integrando numéricamente durante cinco periodos una órbita que tiene excentricidad $e = 0.5$. Podemos apreciar diferencias en la representación de las órbitas debido a las distintas longitudes de paso usadas. En la Figura 4.1a, con una longitud de paso $h = \frac{2\pi}{64}$, la solución numérica calculada no es periódica, mientras que en la Figura 4.1b, con una longitud de paso $h = \frac{2\pi}{512}$, se observa claramente el carácter periódico de la solución numérica. Esto es debido a que, al avanzar con pasos más pequeños en cada periodo, las aproximaciones son más precisas y el error se reduce.



(a) Longitud de paso $h = \frac{2\pi}{64}$ (b) Longitud de paso $h = \frac{2\pi}{512}$

Figura 4.1: Órbitas calculadas con la regla implícita del punto medio con iteración de punto fijo.

Para el estudio del promedio de iteraciones requerido por cada algoritmo se ha fijado $e = 0.5$ para la excentricidad, y tolerancia $TOL = 10^{-8}$ para detener las iteraciones, y se ha realizado una simulación durante 3 periodos. Los resultados se muestran en las Tablas 4.1 y 4.2

h	Iteración de Newton	Iteración del Punto Fijo
$2\pi/64$	3.60	7.63
$2\pi/128$	3.18	6.02
$2\pi/256$	2.79	5.25
$2\pi/512$	2.35	4.39
$2\pi/1024$	2.15	4.11
$2\pi/2048$	2.00	3.43

Tabla 4.1: Método de Gauss de 1 etapa y orden 2

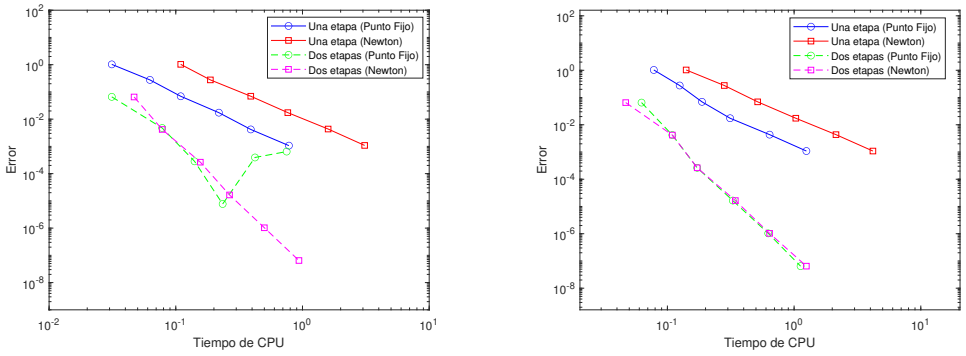
h	Iteración de Newton	Iteración del Punto Fijo
$2\pi/64$	3.38	6.67
$2\pi/128$	3.13	5.57
$2\pi/256$	2.97	5.00
$2\pi/512$	2.37	4.29
$2\pi/1024$	2.17	4.09
$2\pi/2048$	2.00	3.47

Tabla 4.2: Método de Gauss de 2 etapas y orden 4

Podemos observar que para los dos métodos de Gauss utilizados, el promedio de iteraciones cuando se utiliza el método de Newton es siempre menor que el promedio de iteraciones de punto fijo utilizadas. Lo que no se puede apreciar en las tablas es que esto es a costa de un mayor coste computacional, como se puede comprobar en las Figuras 4.2a y 4.2b. Se observa también que cuando h es más pequeño hacen falta menos iteraciones de cualquiera de los dos métodos, porque el iterante inicial está más cerca de la solución exacta de las ecuaciones no lineales.

En las gráficas que aparecen ambas figuras, hemos representado el error frente al tiempo de CPU necesario para integrar a lo largo de 50 periodos, con unos parámetros iniciales de $e = 0.5$ para la excentricidad y seis longitudes de paso distintas: $h = \frac{2\pi}{512}, ..., \frac{2\pi}{16384}$ para los algoritmos de una etapa y $h = \frac{2\pi}{64}, ..., \frac{2\pi}{2048}$ para los algoritmos de dos etapas.

En la Figura 4.2a se ha utilizado $TOL = 10^{-8}$, como en las tablas, mientras que en la Figura 4.2b se han considerado tolerancias que van disminuyendo a medida que lo hace la longitud del paso de integración ($TOL = \frac{1}{2} \cdot h \cdot 10^{-12}$). Para la Figura 4.4a se han usado ambas tolerancias.



(a) Tolerancia $TOL = 10^{-8}$. (b) Tolerancia $TOL = \frac{1}{2} \cdot h \cdot 10^{-12}$.

Figura 4.2: Error frente a tiempo de CPU

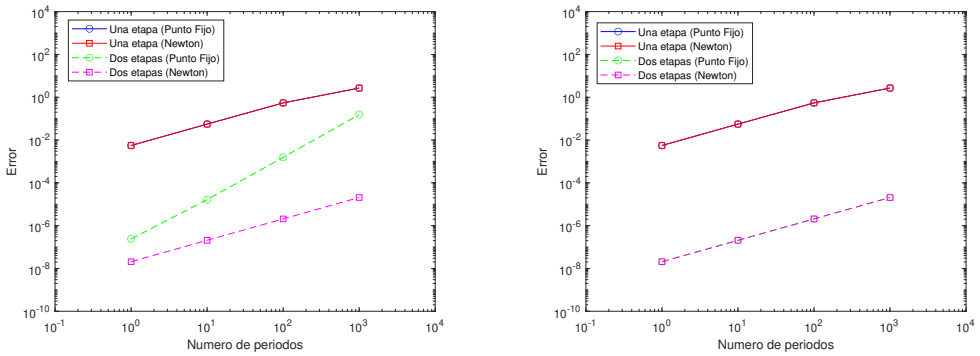
Podemos concluir varias cosas. En primer lugar, y como es de esperar, los algoritmos de dos etapas tienen una mayor precisión que los de una sola etapa (tienen un error más pequeño para la misma longitud de paso), aunque también necesitan mayor tiempo de CPU. Podemos ver también que, cuando TOL es suficientemente pequeña, los errores con la iteración

de punto fijo y con la iteración de Newton son los mismos, pero el tiempo de CPU es ligeramente inferior en el primer caso (dicha ventaja es más evidente en el caso $s = 1$).

Observamos igualmente que $TOL = 10^{-8}$ resulta insuficiente en el caso del método de orden 4 con las longitudes de paso h más pequeñas. Esto es debido a que el error de la iteración de punto fijo contamina el error de la integración, que además se propaga durante 50 periodos. Finalmente, examinando las cuatro implementaciones, concluimos que la más eficiente es $s = 2$ con punto fijo.

En la Figura 4.3 se muestra el error frente al número de periodos durante los que se ha integrado, que han sido 1, 10, 100 y 1000. De nuevo, la excentricidad ha sido $e = 0.5$ y la longitud de paso escogida ha sido $h = \frac{2\pi}{1024}$ para ambas gráficas.

Las Figuras 4.3a y 4.3b confirman lo que habíamos dicho anteriormente: la superposición de los resultados indica que los errores cometidos con los dos tipos de iteraciones son los mismos. Además, para el método de orden 4 en la iteración de punto fijo con $TOL = 10^{-8}$ se ve reflejada la influencia del error de la iteración en el error del integrador temporal.



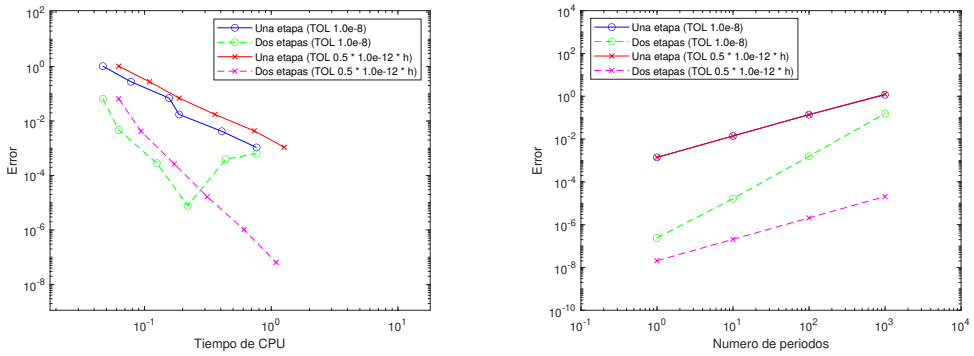
(a) Tolerancia $TOL = 10^{-8}$. (b) Tolerancia $TOL = \frac{1}{2} \cdot h \cdot 10^{-12}$.

Figura 4.3: Error frente a número de periodos

La Figura 4.4b utiliza una longitud de paso de $h = \frac{2\pi}{2048}$ para los algoritmos de una etapa y $h = \frac{2\pi}{1024}$ para los de dos etapas. Distinguimos en los resultados dos pendientes diferentes. Los algoritmos de una etapa y el de dos etapas con $TOL = \frac{1}{2} \cdot h \cdot 10^{-12}$ tienen pendiente 1, es decir, el error

crece de manera lineal comparado con el número de periodos. Sin embargo, para el algoritmo de dos etapas y $TOL = 10^{-8}$ tiene pendiente 2, lo que implica que el crecimiento es cuadrático, justificado por la no simplecticidad del método en la implementación. Cabe destacar que, para otros valores de h, llegamos a las mismas conclusiones.

En consecuencia, tras las pruebas realizadas, deducimos que la opción más eficiente es la iteración de punto fijo con una tolerancia suficientemente pequeña, por lo que es lo que utilizaremos en el siguiente capítulo.



(a) Error frente a tiempo de CPU (b) Error frente a número de periodos

Figura 4.4: Resultados con ambas tolerancias usando la iteración de punto fijo

Capítulo 5

El problema de los 5 planetas exteriores

En este último capítulo estudiaremos la simulación a largo plazo del problema de los 5 planetas exteriores, también conocido como el problema de los 6 cuerpos, que serán el Sol, Júpiter, Saturno, Urano, Neptuno y Plutón (no considerado planeta desde 2006). Comenzaremos con una implementación parecida a la del capítulo anterior para después ir incorporando una serie de mejoras propuestas en [1]. Estudiaremos los resultados que se obtengan con dichas mejoras para concluir la ejecución de código más eficiente en la simulación durante largos intervalos de tiempo.

5.1. Ecuaciones del problema

El problema de los 5 planetas exteriores está caracterizado por tener un Hamiltoniano

$$H(\mathbf{p},\mathbf{q}) = \frac{1}{2}\sum_{i=0}^{5}\frac{\|\mathbf{p}_i\|^2}{m_i} - G\sum_{i=0}^{5}\sum_{j=i+1}^{5}\frac{m_i m_j}{\|\mathbf{q}_i-\mathbf{q}_j\|}, \tag{5.1}$$

donde m_i representa la masa de los cuerpos, $\mathbf{p}_i$ los momentos, y $\mathbf{q}_i$ las posiciones. En la Tabla 5.1 se muestran los datos del 5 de septiembre de 1994 a las 00:00 horas, con los que se realizará la simulación, y que se han obtenido de [6].

i	**Cuerpo**	**Masa**	**Posición Inicial**	**Velocidad Inicial**
0	Sol	1.00000597682	0 0 0	0 0 0
1	Júpiter	0.000954786104043	−3.5023653 −3.8169847 −1.5507963	0.00565429 −0.00412490 −0.00190589
2	Saturno	0.000285583733151	9.0755314 −3.0458353 −1.6483708	0.00168318 0.00483525 0.00192462
3	Urano	0.0000437273164546	8.3101120 −16.2901086 −7.2521278	0.00354178 0.00137102 0.00055029
4	Neptuno	0.0000517759138449	11.4707666 −25.7294829 −10.8169456	0.00288930 0.00114527 0.00039677
5	Plutón	$1/(1.3 \cdot 10^8)$	−15.5387357 −25.2225594 −3.1902382	0.00276725 −0.00170702 −0.00136504

Tabla 5.1: Datos iniciales para el problema de los 6 cuerpos.

Nótese que las unidades escogidas son relativas a la masa del Sol, para que éste tenga masa 1. Se ha tomado $m_0 = 1.00000597682$ para tener en cuenta también la masa de los planetas interiores (Mercurio, Venus, Tierra y Marte). Las distancias están en unidades astronómicas (1 [UA] = 149 597 870 [km]), el tiempo en días terrestres, y la constante gravitacional es $G = 2.95912208286 \cdot 10^{-4}$.

Las ecuaciones del movimiento son las siguientes

$$\frac{d\mathbf{p}_i}{dt} = -\frac{\partial H}{\partial \mathbf{q}_i} = -G \sum_{\substack{j=0 \\ j \neq i}}^{5} \frac{m_i m_j (\mathbf{q}_i - \mathbf{q}_j)}{\|\mathbf{q}_i - \mathbf{q}_j\|^3}, \qquad 0 \leq i \leq 5, \tag{5.2}$$

$$\frac{d\mathbf{q}_i}{dt} = \frac{\partial H}{\partial \mathbf{p}_i} = \frac{\mathbf{p}_i}{m_i}, \qquad 0 \leq i \leq 5. \tag{5.3}$$

Si se quiere mantener fija la posición del Sol se tomará $\dfrac{d\mathbf{p}_0}{dt} = \dfrac{d\mathbf{q}_0}{dt} = \mathbf{0}$, lo que conduce a $\mathbf{p}_0(t) = \mathbf{q}_0(t) = \mathbf{0}$.

5.2. Solución de referencia

En el estudio del problema de los 6 cuerpos, a diferencia del problema de Kepler, nos enfrentamos a la dificultad de que no conocemos la solución exacta. Es por ello que necesitamos tener una solución de referencia para comparar nuestros resultados y ver el efecto de las posteriores mejoras que se adoptan.

MatLab tiene varias funciones para resolver sistemas de ecuaciones diferenciales ordinarias. Una de ellas es **ode89** (documentación en [9]), que implementa con paso variable un par encajado de métodos Runge-Kutta de órdenes 9 y 8 diseñado para adaptar las longitudes del paso de integración al problema que se quiere integrar, garantizando errores locales por debajo de una tolerancia fijada. El uso de **ode89** con una tolerancia muy exigente es la primera opción para calcular una solución de referencia, que nos permitirá medir los errores de nuestras simulaciones.

Otra opción que hemos usado es **VPA** (Variable Precision Arithmetic, [10]), del paquete de software *Symbolic Math Toolbox*. Se utiliza una aritmética de precisión variable para obtener resultados con mayor precisión. En concreto, se ha trabajado con 34 dígitos significativos, lo que equivale a la aritmética cuádruple. El principal inconveniente que tiene es que, al trabajar con más cifras significativas, el tiempo de ejecución de la simulación es considerablemente alto en comparación con el requerido por **ode89** y por los demás algoritmos. Por ello, una vez aplicado este método a la implementación de base (explicada a continuación) se han guardado los datos de las posiciones y momentos de los cuerpos en cada paso a lo largo del tiempo de simulación. Para las pruebas, se ha fijado $T_{end} = 10^5$ con longitudes de paso $h = \frac{250}{3}, \frac{500}{3}$ días. Para generar estos datos se ha tardado entre 3 y 4 horas para la primera longitud de paso, y entre 2 y 3 horas para la segunda.

5.3. Implementación básica

Siguiendo la línea del capítulo anterior, utilizaremos la implementación de la iteración de punto fijo ya estudiada en la Sección 4.3, pero esta vez para el método de Gauss de cuatro etapas y orden ocho, cuyos coeficientes están

en la Sección 3.2. Sin embargo, marcaremos algunas diferencias. En primer lugar, afinaremos un poco más la tolerancia para detener la iteración de punto fijo y la fijaremos en $TOL = \frac{1}{2} \cdot h \cdot 10^{-18}$, además de definir el numero máximo de iteraciones en $maxiter = 20$. Adicionalmente, tendremos que pasar como parámetro el tiempo de simulación, T_{end}, y la longitud de paso h. Por último, puesto que el objetivo es simular durante largos intervalos de tiempo, se define el parámetro $num\_guardar\_datos$ para determinar cada cuantos pasos se guardan los datos de la simulación.

Para mejorar el carácter simpléctico del método, siguiendo [2], se definen los coeficientes $hb_i = h \cdot b_i$, para $i = 2, ..., s-1$, y $hb_1 = hb_s = \left(h - \sum_{i=2}^{s-1} hb_i\right)/2$. Con ello, conseguimos que en la práctica, la suma de los coeficientes b_i sea igual a 1, además de ahorrar operaciones en la iteración que podrían dar lugar a errores de redondeo. Al mismo tiempo, la condición de simplecticidad (2.19), en términos de los coeficientes μ_{ij} introducidos en la Sección 4.3 quedaría

$$\mu_{ij} + \mu_{ji} - 1 = 0 \qquad 1 \leq i, j \leq s. \tag{5.4}$$

En consecuencia, fijados los coeficientes μ_{ij} con $1 \leq i < j \leq 4$, se definen los coefientes $\mu_{ji} = 1 - \mu_{ij}$, con $1 \leq i < j \leq 4$ para cumplir en aritmética finita la condición (5.4).

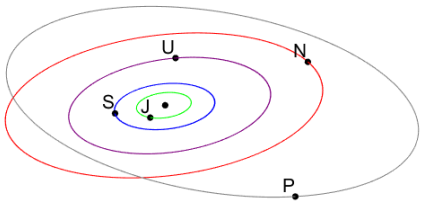


Figura 5.1: Órbitas de los 5 planetas exteriores con respecto al Sol en una simulación de $T_{end} = 2 \cdot 10^5$ y $h = 10$.

5.4. Mejoras sobre la implementación básica

En [1] se presentan una serie de mejoras que hemos implementado en nuestros códigos (ver Apéndice B) con el objetivo de evaluar su efectividad

en el problema de los seis cuerpos, y deducir así la mejor implementación posible. A continuación, presentamos cada una de estas mejoras.

5.4.1. Criterio de parada

Definir correctamente un criterio de parada en los métodos iterativos es esencial. Parar las iteraciones demasiado tarde puede afectar a la eficiencia del método, mientras que si se paran demasiado pronto puede no alcanzarse la precisión deseada, lo que, en el caso que nos ocupa, puede implicar que el método deje de ser simpléctico mostrando un comportamiento del error con el tiempo distinto del esperado.

En [1] se estudia un criterio de parada que mejore la simulación. Tras varias alternativas, logran definir un nuevo criterio de parada que no depende de una tolerancia, ya que se trabaja con $\Delta\mathcal{X}^{[k]} \in \mathbb{R}^{sD}$, ya definido en (4.14). En nuestro caso, el número de etapas es cuatro ($s = 4$), y la dimensión del problema es $D = 36$, ya que tenemos 3 componentes para cada momento, 3 componentes para cada posición, y todo ello para cada uno de los 6 cuerpos. En consecuencia, tenemos $s \cdot D = 4 \cdot 36 = 144$ componentes, que denotaremos con el subíndice j. Una vez vamos obteniendo $\Delta\mathcal{X}^{[k]}$, la idea es que se detenga la iteración de punto fijo cuando $\Delta\mathcal{X}^{[k]} = \mathbf{0}$ o bien cuando se cumpla la siguiente condición en dos iteraciones consecutivas

$$\min_{1\leq j\leq sD} \Big\{ \{|\Delta\mathcal{X}_j^{[1]}|, \ldots, |\Delta\mathcal{X}_j^{[k-1]}|\} \,/\, \{0\} \Big\} \leq |\Delta\mathcal{X}_j^{[k]}|. \tag{5.5}$$

En otras palabras, se comprueba para cada componente j si el valor de $|\Delta\mathcal{X}_j^{[k]}|$ en la iteración actual es mayor o igual que el valor mínimo de $\{|\Delta\mathcal{X}_j^{[1]}|, \ldots, |\Delta\mathcal{X}_j^{[k-1]}|\}$, sin contar el 0. En caso de que esto ocurra para todo j durante dos iteraciones consecutivas, se dará por finalizada la iteración, pues en las últimas dos iteraciones no se observa mejora en ninguna de las sD componentes.

La principal característica de este nuevo criterio de parada es que tenemos asegurado que, o bien $\Delta\mathcal{X}^{[k]} = \mathbf{0}$, lo cual indica que ya tenemos la solución exacta de las ecuaciones no lineales, o bien que no merece la pena seguir iterando porque no estamos consiguiendo convergencia en la iteración.

De todos modos, el número máximo de iteraciones también viene mar-

cado por el parámetro *maxiter*. Si en alguna iteración se alcanzara su valor, ésta se detiene y comenzaría a iniciarse el siguiente paso en la simulación.

En los resultados numéricos de la Sección 5.5 nos referiremos a la implementación del método de Gauss de 4 etapas con el criterio de parada que acabamos de describir con Gauss8v1.

5.4.2. Suma compensada de Kahan

El algoritmo de sumación compensada de Kahan es conocido por reducir el efecto de redondeo que se produce al sumar números muy pequeños y números muy grandes en una misma operación. Fue diseñado por el matemático canadiense William "Velvel" Morton Kahan en 1965 [7] para mitigar los problemas que tienen los ordenadores al realizar estas operaciones con aritmética de coma flotante, ya que no pueden representar con exactitud los números reales.

Algoritmo 1 Algoritmo de sumación compensada de Kahan

Entradas: $x_1, x_2, \ldots, x_n$ (números a sumar)
Salida: S (suma final compensada)

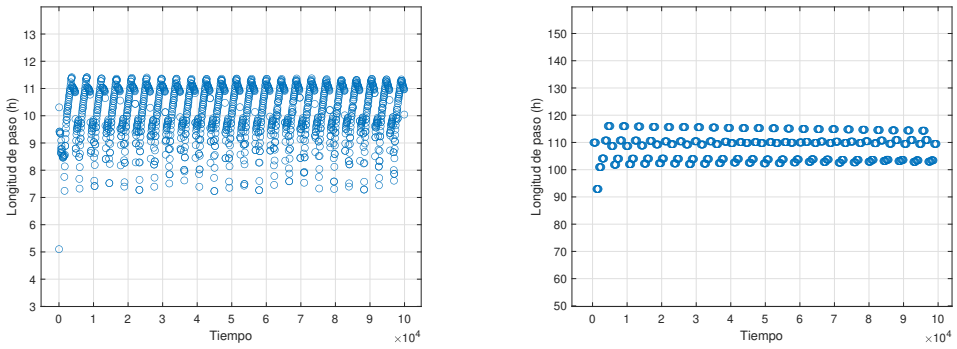
1: $S \leftarrow 0$ ▷ Inicializar la suma
2: $C \leftarrow 0$ ▷ Inicializar la compensación
3: **for** $i = 1$ to n **do**
4: $\quad y \leftarrow x_i - C$ ▷ Restar la compensación al sumando actual
5: $\quad t \leftarrow S + y$ ▷ Sumar el término corregido
6: $\quad C \leftarrow (t - S) - y$ ▷ Actualizar la compensación
7: $\quad S \leftarrow t$ ▷ Actualizar la suma
8: **end for**

La idea del algoritmo de Kahan es introducir un complemento de compensación para ajustar estos errores de redondeo durante el proceso de la suma. Este complemento actúa como una corrección dinámica durante la suma de cada término. El algoritmo guarda la diferencia entre la suma exacta y la suma calculada en cada paso. Cuando un número pequeño se suma a un número grande, esta diferencia se añade al resultado de la suma para corregir el error de redondeo que podría haberse introducido, evitando que el número pequeño sea ignorado en la operación.

El pseudocódigo se muestra en el Algoritmo 1. En nuestro caso, lo hemos adaptado de manera matricial (Apéndice B) para aplicarlo directamente en (4.9) y (4.11). Utilizaremos Gauss8v2 para referirnos al método de Gauss de 4 etapas en el que la implementación Básica se ha mejorado con el algoritmo de sumación compensada de Kahan. Adicionalmente, utilizaremos Gauss8v3 para la implementación que, además de la sumación compensada de Kahan, utiliza el criterio de parada explicado en la Sección 5.4.1.

5.5. Resultados numéricos

Presentamos en esta sección los resultados obtenidos tras las pruebas realizadas. En primer lugar, ya hemos visto que al no tener una solución de referencia, hemos acudido a dos métodos distintos (**ode89** y al propio método de Gauss de orden 8 implementado con **VPA**) para obtener una solución con la que comparar nuestros resultados. Una de las principales ventajas del **ode89** es que usa automáticamente un paso variable para calcular las soluciones de manera más precisa y eficiente. Lo ideal sería usar esta ventaja y aplicar la precisión cuádruple que aporta **VPA** a **ode89**. Sin embargo, MatLab no lo permite.



(a) Precisión exigente: error relativo menor que 10^{-13} y error absoluto menor que 10^{-16}.

(b) Precisión predeterminada: error relativo menor que 10^{-3} y error absoluto menor que 10^{-6}.

Figura 5.2: Longitudes de paso usadas por **ode89** con $T_{end} = 10^5$.

Además, en las simulaciones realizadas hemos observado que las longitudes de paso que utiliza **ode89** para este problema no cambian significa-

tivamente a lo largo del tiempo de integración, como se ve en la Figura 5.2 en donde ejecutamos dos veces **ode89** en una simulación de $T_{end} = 10^5$ días. En la Figura 5.2a se han fijado tolerancias 10^{-13} y 10^{-16} para el error relativo y el error absoluto, respectivamente. Observamos que **ode89** utiliza longitudes de paso entre $h = 7$ y $h = 12$. Por otro lado, en la Figura 5.2b dejamos las tolerancias por defecto que son 10^{-3} y 10^{-6}, y el algoritmo utiliza longitudes de paso entre $h = 90$ y $h = 120$. En ambos casos, **ode89** no considera que haya alguna situación en la simulación en la que se pueda dar pasos más largos o más cortos según las posiciones o momentos de los cuerpos, manteniéndose en toda ella una longitud de paso casi constante. Es por ello que usaremos como solución de referencia la calculado con el método de Gauss de orden 8 en cuádruple precisión utilizando **VPA**, ya que para este problema en concreto, usar una longitud de paso fija nos permite disponer de una solución más precisa que si usamos una longitud de paso variable con doble precisión.

Cabe destacar que para las aproximaciones obtenidas a lo largo de las simulaciones realizadas, se ha optado por usar una matriz de seis filas (las tres primeras con los momentos y las tres últimas con las posiciones) y seis columnas (los seis cuerpos). Para ver la diferencia entre las aproximaciones usamos la norma de Frobenius, dada por $\|A\|_F = \left(\sum_{i,j=1}^{6} |a_{ij}|^2\right)^{1/2}$. Esta norma no es otra que la norma euclídea aplicada al vector de 36 elementos con los 18 momentos y las 18 posiciones, como sería nuestro caso si el formato usado hubiera sido vectorial.

A lo largo de las simulaciones realizadas, en algunas ocasiones no se ha cumplido el criterio de parada de la iteración, bien sea porque la diferencia entre los iterantes sigue siendo mayor que la tolerancia $TOL = \frac{1}{2} \cdot h \cdot 10^{-18}$, o bien porque no se ha cumplido el criterio de parada explicado en la Sección 5.4.1. A pesar de que en algún paso se supera el número máximo de iteraciones fijado, esto sucede muy raramente y no parece afectar al comportamiento general de los algoritmos, tal y como muestran las Tablas 5.2 y 5.3, que reflejan el promedio de iteraciones dadas en la simulación, el número total de iteraciones que han terminado sin cumplir el criterio de parada, y el porcentaje que estas representan sobre el total para las cuatro implementaciones la Tabla 5.2 corresponde al tiempo de integración 10^5 y la Tabla 5.3 a 10^7. Además de revelar que los algoritmos con el nuevo criterio de parada incorporado (Gauss8v1 y Gauss8v3) presentan más iteraciones fallidas que los que no lo incorporan (Básico y Gauss8v2), también se aprecia que aun-

que aumente el tiempo de simulación, el porcentajo de iteraciones fallidas no solo no aumenta, sino que incluso disminuye.

Algoritmo	**Promedio de iteraciones**	**Iteraciones fallidas**	**Porcentaje de pasos fallidos**
Básico	14.71	1	0.167 %
Gauss8v1	15.94	13	2.167 %
Gauss8v2	14.68	1	0.167 %
Gauss8v3	15.83	9	1.5 %

Tabla 5.2: Promedio de iteraciones por paso e iteraciones fallidas en la simulación del problema de los 5 planetas exteriores con $T_{end} = 10^5$ y $h = \frac{500}{3}$

Algoritmo	**Promedio de iteraciones**	**Iteraciones fallidas**	**Porcentaje de pasos fallidos**
Básico	14.70	90	0.15 %
Gauss8v1	15.57	779	1.298 %
Gauss8v2	14.70	79	0.132 %
Gauss8v3	15.56	779	1.298 %

Tabla 5.3: Promedio de iteraciones por paso e iteraciones fallidas en la simulación del problema de los 5 planetas exteriores con $T_{end} = 10^7$ y $h = \frac{500}{3}$

A continuación pasamos a describir los distintos experimentos numéricos que hemos llevado a cabo.

En el primer experimento que hemos realizado, comparamos el error al final de la simulación de nuestras cuatro implementaciones (Básica, Gauss8v1, Gauss8v2, y Gauss8v3) frente a la longitud de paso h. Los parámetros para esta prueba han sido $T_{end} = 10^5$ como tiempo de simulación en días, y como longitudes de paso (también en días) $h = 50, 100, 200, 400, 800$. Como solución de referencia para calcular los errores, hemos usado la solución numérica calculada con el método de Gauss de orden 8 y **VPA**, usando cuádruple precisión con una longitud de paso de $h = 250/3$.

En la Figura 5.3 hemos representado en escala doblemente logarítmica, el error frente a la longitud del paso h utilizado. Observamos una línea con

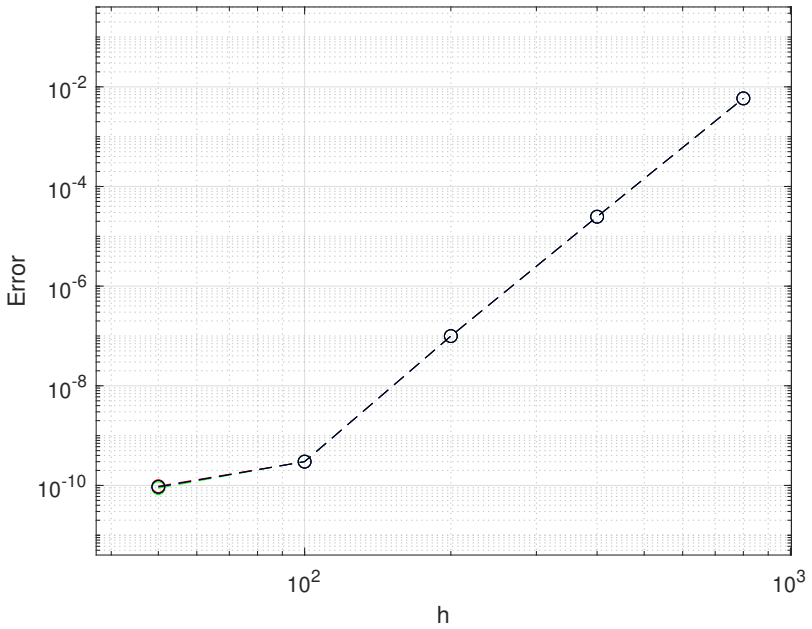


Figura 5.3: Error frente a longitud de paso h con un tiempo de simulación de $T_{end} = 10^5$.

pendiente que se mantiene próxima a 8 según va disminuyendo h. Esto es razonable, puesto que estamos dividiendo la longitud de paso a la mitad, y como el método es de orden 8, el error global se divide cada vez por $2^8 = 256$. Sin embargo, nótese que la última longitud de paso es $h = 50$, menor que la longitud de paso utilizada con **VPA** que es de $h = \frac{250}{3} = 83,33$. De ahí que la pendiente del tramo más a la izquierda de la gráfica se aplane, ya que la solución calculada con **VPA** no es suficientemente precisa para ser comparada con la solución de las otras implementaciones cuando $h = 50$. Por último, cabe mencionar que las cuatro implementaciones consideradas producen los mismos errores globales, por lo que solo distinguimos una línea en la gráfica.

En un segundo experimento, hemos querido analizar el comportamiento de los errores globales respecto del tiempo de integración. Para ello en la Figura 5.4 mostramos los errores que se producen en cada paso, cuando se utiliza Gauss8v3 con longitudes de paso $h = \frac{250}{3}$, $\frac{500}{3}$ y $\frac{1000}{3}$ frente al tiempo. Para la solución de referencia hemos utilizado la misma implementación con $h = \frac{125}{3}$, de tal manera que veremos como se comporta el error según se va

doblando la longitud de paso. Aprovechamos que el coste computacional no es tan elevado usando **VPA** con cuádruple precisión para tomar valores de $T_{end} = 10^7$ días.

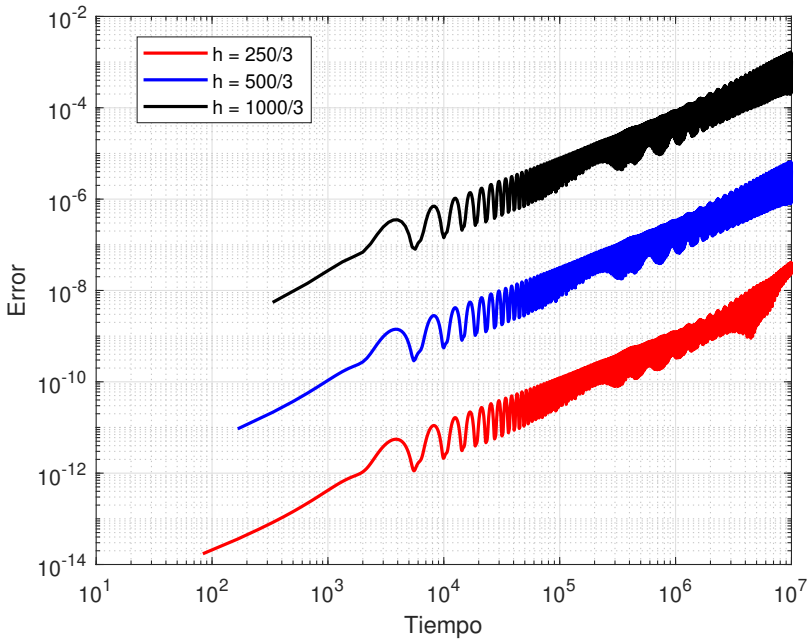


Figura 5.4: Error en las aproximaciones con $T_{end} = 10^7$. En rojo $h = \frac{250}{3}$, en azul $h = \frac{500}{3}$, y en negro $h = \frac{1000}{3}$.

En primer lugar, vemos de nuevo que si doblamos la longitud del paso de integración los errores se multiplican por 256 al ser nuestro método de orden ocho, como se aprecia en la separación vertical entre las tres curvas de error de la Figura 5.4. En cuanto a la pendiente de la envolvente de cada una de las tres curvas, observamos que es aproximadamente 1, lo cual indica una crecimiento lineal del error con el tiempo de simulación. Se observa, no obstante, que para el método con $h = \frac{250}{3}$ (en color rojo) al final de la simulación esta pendiente crece ligeramente. Esto podría deberse a que la solución de referencia con la que se está midiendo el error no es suficientemente precisa.

Para la siguiente prueba, volvemos a estudiar el error en cada paso de las aproximaciones generadas con las cuatro implementaciones del método de Gauss de orden 8, cuando $h = \frac{250}{3}$, la misma longitud de paso utilizada al calcular la solución de referencia en cuádruple precisión con **VPA**, y de

nuevo, fijando un tiempo de integración de $T_{end} = 10^5$ días. Cabe destacar que esta figura se ha generado calculando las órbitas de los cinco planetas con respecto al Sol.

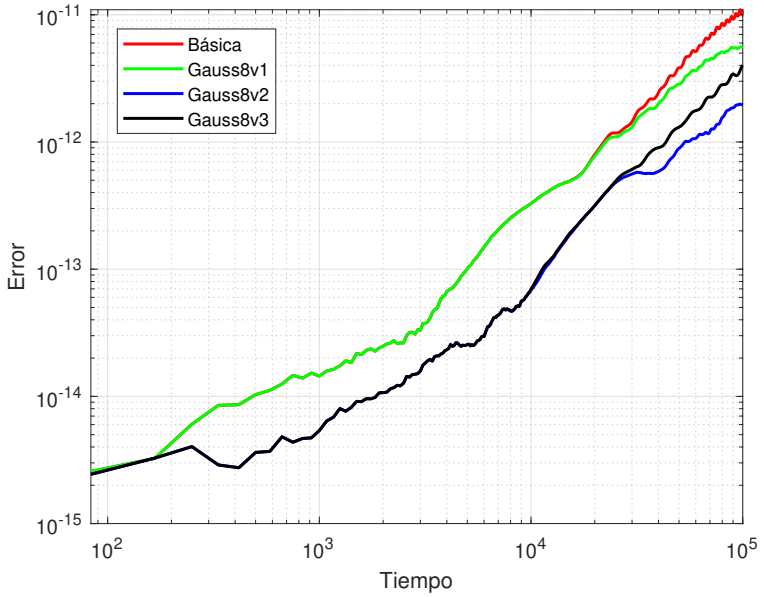
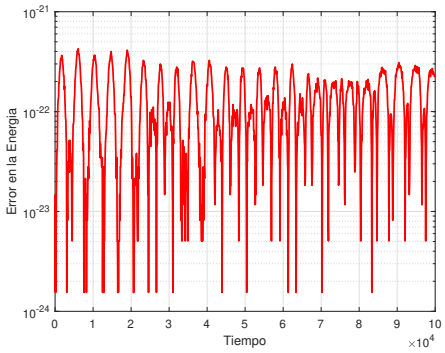


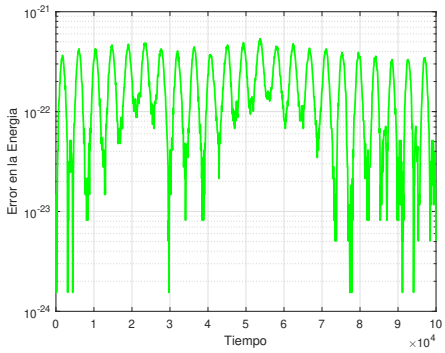
Figura 5.5: Error en las cuatro aproximaciones con $h = \frac{250}{3}$ respecto de la solución de referencia calculada con $h = \frac{250}{3}$ en cuádruple precisión.

En la Figura 5.5 se han representado dichos errores, usando color rojo para la implementación Básica, verde para Gauss8v1, azul para Gauss8v2, y negro para Gauss8v3. Podemos observar que las implementaciones que incorporan el algoritmo de sumación de Kahan son algo más precisas que las que no lo hacen, prácticamente desde el inicio. Durante toda la simulación, las pendientes de las cuatro implementaciones se mantienen entre 1/2 y 2, y en la parte final Gauss8v1 y Gauss8v2 muestran un crecimiento lineal del error con el tiempo de integración, mientras que Gauss8v3 presenta un crecimiento algo mayor, pero sin llegar al cuadrático. Si comparamos con la Figura 5.4, nos damos cuenta de que lo que muestra la Figura 5.5 es de hecho la propagación de los errores de redondeo. Éstos se han obtenido al comparar los resultados producidos con el mismo método y con la misma longitud de paso, pero calculados con doble y cuádruple precisión. Claramente, el algoritmo Básico produce los peores resultados, con una pendiente cercana a 2.

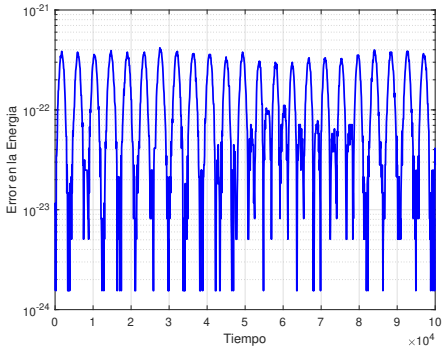
Una de las características que tiene el problema de los 6 cuerpos es que sabemos que por ser un sistema Hamiltoniano, la energía total del sistema se conserva a lo largo de la solución exacta [11]. Por tanto, una herramienta útil para comparar nuestros resultados es evaluar el Hamiltoniano (5.1) en cada paso de las aproximaciones numéricas generadas y compararlo con el valor H_0 que toma el Hamiltoniano en la condición inicial marcadas en la Tabla 5.1.



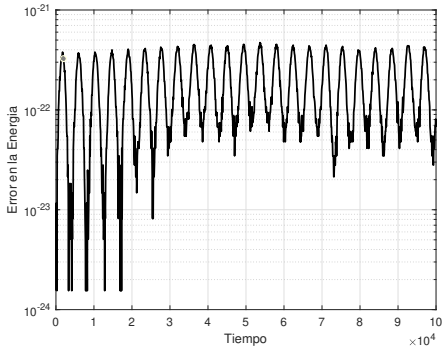
(a) Básica



(b) Nuevo criterio de parada



(c) Suma de Kahan



(d) Ambas

Figura 5.6: Error en la energía con $T_{end} = 10^5$ y $h = \frac{250}{3}$.

En la Figura 5.6 se muestra el error en la energía frente al tiempo de integración para las cuatro implementaciones del método de Gauss de 4 etapas. En esta ocasión solo se ha utilizado escala logarítmica en el eje vertical. Pese a que se distinguen diferencias entre las cuatro figuras, no son muy significativas porque se llega a un nivel de precisión en el que los errores de redondeo juegan un papel muy importante. Observamos que en todos los casos el error en la energía se mantiene entre 10^{-24} y 10^{-21}, a pesar de que

para el mismo valor de h, el error en las soluciones está siempre por encima de 10^{-15}. Además, en todos los casos, el error en la energía no parece crecer con el tiempo de integración.

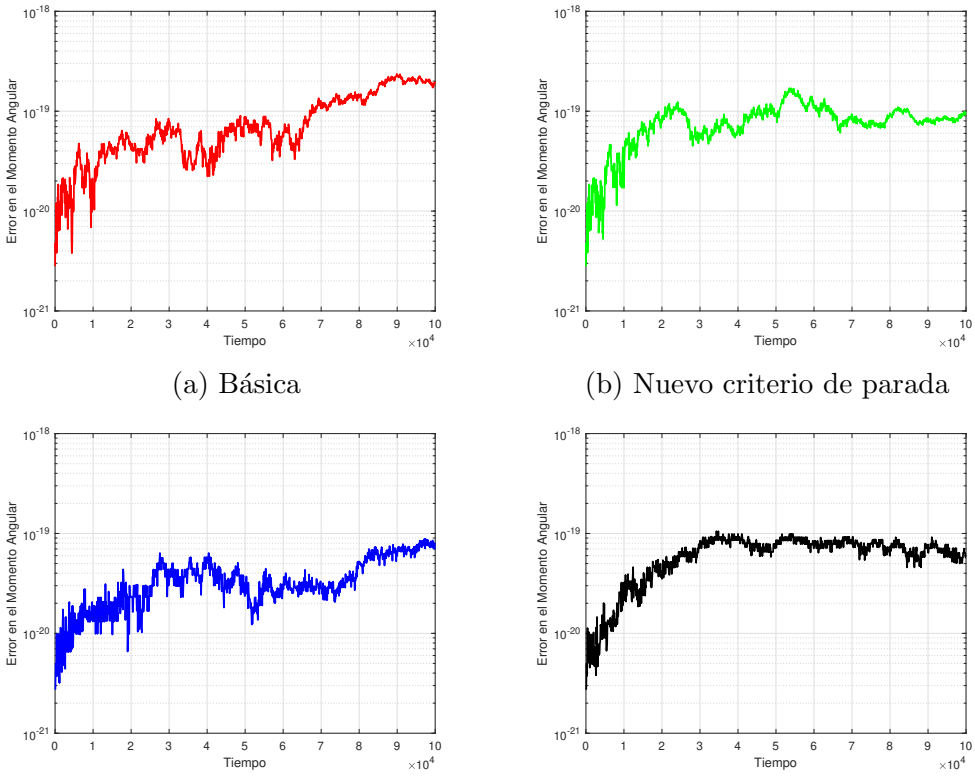
Algo similar ocurre con el momento angular total del sistema, que es otra magnitud física que es exactamente conservada por la solución exacta (5.2) – (5.3) [6]. En este caso, el momento angular total se expresa de la siguiente manera

$$I(\mathbf{p},\mathbf{q}) = \sum_{i=0}^{5} (\mathbf{q}_i \times \mathbf{p}_i) = \sum_{i=0}^{5} \begin{bmatrix} p_{i2}\ q_{i3} - p_{i3}\ q_{i2} \\ p_{i3}\ q_{i1} - p_{i1}\ q_{i3} \\ p_{i1}\ q_{i2} - p_{i2}\ q_{i1} \end{bmatrix}. \tag{5.6}$$

Al igual que hicimos con la energía, vamos a comparar el valor de I en la solución numérica en cada paso con I_0, que es el momento angular total de la condición inicial de la Tabla 5.1. A la vista de (5.6) es claro que el número de operaciones necesarias para evaluar el momento angular total en cada paso es muy inferior al requerido para evaluar la energía según (5.1).

En las Figuras 5.7 vemos que las cuatro implementaciones comienzan con errores del orden de 10^{-20} y que a lo largo de la simulación todos ellos están acotados por 10^{-18}. En concreto, las mejoras de la suma compensada de Kahan y el nuevo criterio de parada provocan que los errores en Gauss8v1, Gauss8v2 y Gauss8v3 sean menores que los producidos por la implementación Básica. En particular, se percibe que los dos algoritmos que incorporan la suma de Kahan, Gauss8v2 y Gauss8v3, producen errores menores que 10^{-19}, ligeramente más pequeños que los otros dos. El algoritmo Básico es claramente la peor opción.

Para finalizar el capítulo, hemos querido hacer una última prueba con un enfoque estadístico, siguiendo [2]. Hemos integrado con Gauss8v3 partiendo de $P = 1000$ condiciones iniciales aleatorias que son perturbaciones pequeñas de la condición inicial con la que hemos trabajado hasta ahora. Para conseguir estas condiciones iniciales, hemos tomado los datos de la Tabla 5.1 y hemos sumado a cada componente un número generado con una distribución normal de media 0 y desviación estándar 1, multiplicando por 10^{-12} para los momentos, y por 10^{-9} para las posiciones. De este modo las perturbaciones relativas son de tamaño 10^{-6}. Para poder realizar el experimento en un tiempo razonable, se ha tomado $T_{end} = 10^4$ y se ha reducido la longitud de paso a $h = 10$ días, de modo que tendríamos por



(a) Básica (b) Nuevo criterio de parada

(c) Suma de Kahan (d) Ambas

Figura 5.7: Error en el momento angular con $T_{end} = 10^5$ y $h = \frac{250}{3}$.

cada una de las P simulaciones $\frac{T_{end}}{h} = 1000$ aproximaciones (sin contar la aproximación inicial perturbada, que no usaremos). Hemos tomado como muestra los datos obtenidos cada $M = 20$ pasos, haciendo uso del parámetro *num_guardar_datos*, definido en la Sección 5.3, y hemos calculado el valor de los saltos relativos en la energía en los tiempos de la muestra

$$\frac{H(\mathbf{x}_{kM}) - H(\mathbf{x}_{kM-M})}{H_0}, \qquad k = 1, 2, ..., \tag{5.7}$$

donde H_0 es la energía inicial del sistema.

La Figura 5.8 muestra un histograma de frecuencias correspondiente a los saltos relativos en la energía (5.7) de la muestra generada. Idealmente, se esperaría que el vector de saltos relativos en la energía obtenido tuviera media $\mu = 0$. En nuestro caso, hemos obtenido una media $\mu = 2.929 \cdot 10^{-18}$ y una desviación estándar $\sigma = 6.146 \cdot 10^{-16}$, que indican que se trata de

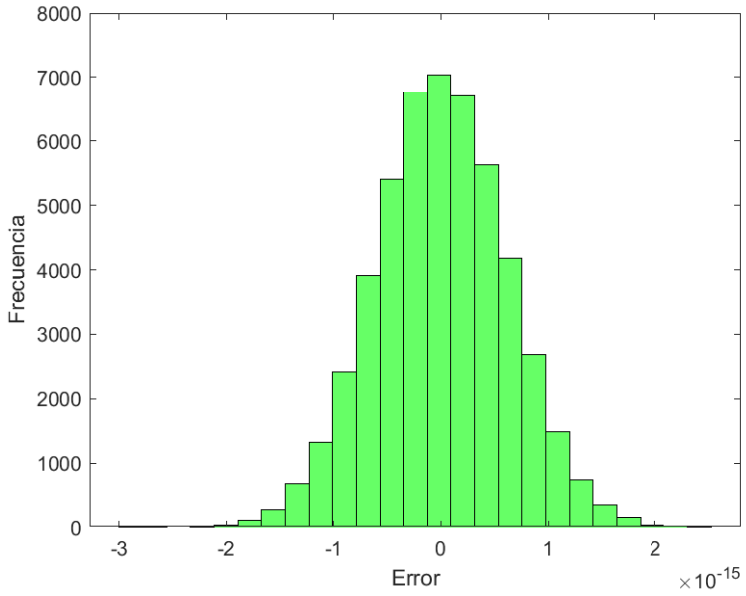


Figura 5.8: Histograma de frecuencias de los saltos relativos en la energía con $P = 1000$, $T_{end} = 10^4$, $M = 20$, $h = 10$. Media $\mu = 2.929 \cdot 10^{-18}$. Desviación estándar $\sigma = 6.146 \cdot 10^{-16}$.

errores de redondeo. Se puede apreciar en la Figura 5.8 que el histograma sigue bastante fielmente una distribución normal.

Capítulo 6

Conclusión

Los métodos Runge-Kutta simplécticos son una gran herramienta para integrar numéricamente sistemas Hamiltonianos. En concreto, hemos visto que los métodos de Gauss-Legendre, también conocidos como los métodos de Gauss, tienen la propiedad de que si el método tiene s etapas, el orden es $2s$, cosa que no ocurre con otros métodos.

La simplecticidad ayuda a que las aproximaciones generadas en las simulaciones durante largos intervalos de tiempo sean más precisas que las generadas con otros métodos que se no tienen en cuenta esta propiedad. Para que se mantenga el carácter simpléctico del método y las buenas propiedades que de ellos se derivan en las simulaciones a tiempos muy largos es importante que los coeficientes del mismo cumplan las condiciones de canonicidad en la aritmética de coma flotante utilizada. Sumado a ello, las implementaciones que incorporen técnicas o procedimientos que amortigüen la propagación de los errores de redondeo y de truncación de la máquina, como puede ser el algoritmo de sumación compensada de Kahan, proporcionarán unos resultados más precisos.

Otro aspecto importante a valorar es que para muchos de los problemas prácticos, no se dispone de una solución exacta del mismo, por lo que hay que construir una solución de referencia, lo que puede requerir el uso de aritmética de cuádruple precisión como en este Trabajo de Fin de Grado, si se quiere comparar el comportamiento de los errores de redondeo de diferentes algoritmos aplicados a un mismo problema.

Bibliografía

[1] M. Antoñana, *Implementación eficiente de métodos Runge-Kutta implícitos simplécticos y su aplicación en la simulación del sistema solar*, Tesis Doctoral, Universidad del País Vasco, 2017.

[2] M. Antoñana, J. Makazaga & A. Murua, *Reducing and monitoring round-off error propagation for symplectic implicit Runge-Kutta schemes*, Numerical Algorithms 76(4), 2017, 861 – 880.

[3] M. Antoñana, J. Makazaga & A. Murua, *New Integration Methods for Perturbed ODEs Based on Symplectic Implicit Runge–Kutta Schemes with Application to Solar System Simulations*, Journal of Scientific Computing 76(1), 2018, 630 – 650.

[4] J.C. Butcher. *Implicit Runge-Kutta processes*. Mathematics of Computation 18(85), 1964, 50 – 64.

[5] M.P. Calvo & E. Hairer, *Accurate Long-Term Integration of Dynamical Systems*, Applied Numerical Mathematics 18(1–3), 1995, 95-105.

[6] E. Hairer, Ch. Lubich & G. Wanner, *Geometric Numerical Integration*, Springer, 2006.

[7] W. Kahan. *Further remarks on reducing truncation errors*. Communications of the ACM 8(1), 1965, 40.

[8] F. Kang & Q. Mengzhao, *Symplectic Geometric Algorithms for Hamiltonian Systems*, Springer, 2010.

[9] MathWorks. Marzo de 2025. *ode89 Documentation*. Recuperado de `https://es.mathworks.com/help/matlab/ref/ode89.html`

[10] MathWorks. Marzo de 2025. *Symbolic Math Toolbox User's Guide*. Recuperado de `https://es.mathworks.com/help/pdf_doc/symbolic/symbolic_ug.pdf`

[11] J.M. Sanz-Serna & M.P. Calvo, *Numerical Hamiltonian Problems*, Chapman & Hall, 1994.

Apéndice A

Código del problema de Kepler

En esta sección se incluyen los algoritmos y códigos más relevantes diseñados para el problema de Kepler en el software matemático MatLab en su versión 2024b.

- **Programa Principal**

  Programa principal para el problema de Kepler.

```
% SCRIPT PRINCIPAL PARA EL PROBLEMA DE KEPLER

% Parametros iniciales
exc = 0.5; % Excentricidad iterante inicial
opcion_tol = 0; % Opcion tolerancia
maxiter = 50; % Numero maximo de iteraciones
num_periodos = 50; % Numero de periodos

% Ejecucion programas
[aproxUnaEtapa, errorUnaEtapa, tiemposUnaEtapa] =
   una_etapa_newton_kepler(exc, opcion_tol, ...
    maxiter, num_periodos);

[aproxDosEtapas, errorDosEtapas, tiemposDosEtapas] =
   dos_etapas_pfijo_kepler(exc, opcion_tol, ...
    maxiter, num_periodos);

% Grafica Error VS Tiempo CPU
```

```
figure;
loglog(tiemposUnaEtapa, errorUnaEtapa, '-s', 'Color', 'r', 'DisplayName', 'Una Etapa Newton');
hold on;
loglog(tiemposDosEtapas, errorDosEtapas, '--o', 'Color', 'g', 'DisplayName', 'Dos Etapas Punto Fijo');
axis([1.0e-2 1.0e1 1.0e-9 1.0e2]);
xlabel('Tiempo de CPU');
ylabel('Error');
legend;
hold off;
```

Código A.1: principalKepler.m

- **Kepler**

  Función **f** del problema de Kepler.

```
function [fx] = kepler(x)
% PARAMETROS DE ENTRADA
% x: vector con los momentos y posiciones
% PARAMETROS DE SALIDA
% fx: vector con los nuevos momentos y posiciones

aux = (x(3)^2 + x(4)^2)^(3/2);
fx = [-x(3)/ aux; -x(4)/aux; x(1); x(2)];

end
```

Código A.2: kepler.m

- **Jacobiano Kepler**

  Función para calcular el jacobiano J del problema de Kepler.

```
function [Jf] = jacobiano_kepler(x)
% PARAMETROS DE ENTRADA
% x: vector con los momentos y posiciones
% PARAMETROS DE SALIDA
% Jf: matriz jacobiana del problema de Kepler

aux = (x(3)^2 + x(4)^2)^(5/2);
```

```
8  Jf = [0, 0, (2 * x(3)^2 - x(4)^2) / aux, (3 * x(3) * x
       (4)) / aux;
9       0, 0, (3 * x(3) * x(4)) / aux, (2 * x(4)^2 - x(3)
           ^2) / aux;
10      1, 0, 0, 0; 0, 1, 0, 0];
11
12 end
```

Código A.3: jacobiano_kepler.m

- **Una etapa Newton**

  Función para el problema de Kepler usando la regla implícita del punto medio con la iteración de Newton.

```
1  function [xx, error, tiemposCPU] =
       una_etapa_newton_kepler(exc, opcion_tol, maxiter,
       num_periodos)
2  % PARAMETROS DE ENTRADA
3  % exc: excentricidad del iterante inicial(0 <= exc < 1)
4  % tol: eleccion de la tolerancia 0 para 1.0e-8, 1 para
       1/2*h*1.0e-12
5  % maxiter: numero maximo de iteraciones
6  % num_periodos: numero de periodos
7  % PARAMETROS DE SALIDA
8  % xx: matriz con las aproximaciones finales
9  % error: vector de errores tras la simulacion
10 % tiemposCPU: vector de tiempos de CPU
11
12 % INICIO DEL PROGRAMA
13 error = zeros(1, 6); % Inicio error
14 xx = zeros(4, 7); % Inicio aproximaciones
15 tiemposCPU = zeros(1, 6); % Inicio tiempos de CPU
16 xx(:, 1) = [0; sqrt((1+exc)/(1-exc)); 1-exc; 0]; %
       Aproximacion inicial
17
18 % Bucle for para estudiar cada longitud de paso
19 for i = 1 : 6
20   % Carga de datos
21   tic % Inicio del tiempo de CPU
22   x = xx(:, 1); % Iterante inicial
23   Z = [0; 0; 0; 0];
```

```
24  pasos = 256 * 2^i; % Numero de pasos en un periodo
25  h = 2 * pi / pasos; % Longitud de paso
26  iter_total = 0; % Numero de iteraciones totales
27
28  % Eleccion de la tolerancia
29  if(opcion_tol == 0)
30      tol = 1.0e-8;
31  else
32      tol = 1/2 * 1.0e-12 * h;
33  end
34
35  % Bucle para los periodos
36  for periodo = 1 : num_periodos
37      % Bucle para los pasos de un periodo
38      for j = 1 : pasos
39          % Valores iniciales para entrar en el bucle
               while
40          niter = 0;
41          dif = tol + 1;
42
43          Jfx = jacobiano_kepler(x); % Jacobiano
44
45          % Bucle para calcular la siguiente
               aproximacion
46          while (niter < maxiter) && (dif > tol)
47              % Para actualizar el jacobiano en cada
                   iteracion
48              % Jfx = jacobiano_kepler(x+Z);
49
50              % Calculo de DeltaZ
51              deltaZ = (eye(4) - h/2 * Jfx) \ (-Z + h/2
                    * kepler((x + Z)));
52
53              % Actualizacion de datos
54              Z = Z + deltaZ;
55              niter = niter + 1;
56              dif = norm(deltaZ); % Calculo del error
                   en la iteracion
57          end
58
59          % Mostrar por pantalla los resultados en caso
```

```
                de no converger
60          if(dif > tol)
61              fprintf(['No se ha alcanzado la
                    aproximacion con la ' ...
62                  'tolerancia deseada, con h = 2*pi / %
                        i, periodo = %i, ' ...
63                  'paso = %i'], 32 * 2^i, periodo, j );
64          end
65
66          % Aproximacion final y actualizacion
67          iter_total = iter_total + niter;
68          x = x + 2 * Z;
69      end
70  end
71
72  % Calculos tras la simulacion
73  xx(:, 1+i) = x;
74  error(i) = norm(xx(:, 1) - xx(:, 1+i)); % Error
75  tiemposCPU(i) = toc; % Fin del tiempo de CPU
76  end
77  end
```

Código A.4: una_etapa_newton_kepler.m

- **Dos etapas punto fijo**

  Función para el problema de Kepler usando el método de Gauss de dos etapas, con la iteración de punto fijo.

```
1   function [xx, error, tiemposCPU] =
        dos_etapas_pfijo_kepler(exc, opcion_tol, maxiter,
        num_periodos)
2   % PARAMETROS DE ENTRADA
3   % exc: excentricidad del iterante inicial(0 <= exc < 1)
4   % tol: eleccion de la tolerancia 0 para 1.0e-8, 1 para
        1/2*h*1.0e-12
5   % maxiter: numero maximo de iteraciones
6   % num_periodos: numero de periodos
7   % PARAMETROS DE SALIDA
8   % xx: matriz con las aproximaciones finales
9   % error: vector de errores tras la simulacion
10  % tiemposCPU: vector de tiempos de CPU
```

```
11
12 % INICIO DEL PROGRAMA
13 error = zeros(1, 6); % Inicio error
14 xx = zeros(4, 7); % Inicio aproximaciones
15 tiemposCPU = zeros(1, 6); % Inicio tiempos de CPU
16 xx(:, 1) = [0; sqrt((1+exc)/(1-exc)); 1-exc; 0]; %
       Aproximacion inicial
17
18 % Bucle for para estudiar cada longitud de paso
19 for i = 1 : 6
20
21   % Carga de datos
22   tic % Inicio del tiempo de CPU
23   x = xx(:, 1); % Iterante inicial
24   Y1 = x; % Primera componente
25   Y2 = x; % Segunda componente
26   X = zeros(8,1); % Cuentas en la iteracion
27   % Matriz de coeficientes modificados RK
28   mu = [1/2, (1/2 - sqrt(3)/3); (1/2 + sqrt(3)/3),
         1/2];
29   pasos = 32 * 2^i; % Numero de pasos en un periodo
30   h = 2 * pi / pasos; % Longitud de paso
31   iter_total = 0; % Numero de iteraciones totales
32
33   % Eleccion de la tolerancia
34   if(opcion_tol == 0)
35       tol = 1.0e-8;
36   else
37       tol = 1/2 * 1.0e-12 * h;
38   end
39
40   % Bucle para los periodos
41   for periodo = 1 : num_periodos
42     % Bucle para los pasos de un periodo
43       for j = 1 : pasos
44       % Valores iniciales bucle while
45       niter = 0;
46       dif = tol + 1;
47
48       % Bucle para calcular la siguiente aproximacion
49       while (niter < maxiter) && (dif > tol)
```

```
50          % Inicio iteracion punto fijo
51          L = h .* (1/2) .* [kepler(Y1); kepler(Y2)];
52          Y1 = mu(1, 1) .* L(1:4) + mu(1, 2) .* L(5:8);
53          Y2 = mu(2, 1) .* L(1:4) + mu(2, 2) .* L(5:8);
54          X1 = [Y1; Y2];
55
56          % Actualizar datos
57          dif = norm(X1 - X); % Error en la iteracion
58          X = X1;
59          Y1 = x + Y1;
60          Y2 = x + Y2;
61          niter = niter + 1;
62        end
63
64        % Mostrar por pantalla los resultados en caso de
             no converger
65        if(dif > tol)
66          fprintf(['No se ha alcanzado la aproximacion
               con la ' ...
67            'tolerancia deseada, con h = 2*pi / %i,
                 periodo = %i, ' ...
68            'paso = %i'], 32 * 2^i, periodo, j);
69        end
70
71        % Aproximacion final y actualizacion
72        iter_total = iter_total + niter;
73        x = x + L(1:4) + L(5:8);
74        Y1 = x + Y1;
75        Y2 = x + Y2;
76        end
77    end
78
79  % Calculos tras la simulacion
80  xx(:, 1+i) = x;
81  error(i) = norm(xx(:, 1) - xx(:, 1+i)); % Error
82  tiemposCPU(i) = toc; % Fin del tiempo de CPU
83  end
84  end
```

Código A.5: dos_etapas_newton_kepler.m

Apéndice B

Código del problema de los 5 planetas exteriores

En esta sección se incluyen los algoritmos y códigos más relevantes diseñados para el problema de los 5 planetas exteriores en el software matemático MatLab en su versión 2024b.

- **Programa principal**

  Programa principal para el problema de los 5 planetas exteriores. Se dibujan las gráficas del error en salida de datos y el error en la energía de Gauss8v3.

```
% SCRIPT PRINCIPAL PARA EL PROBLEMA DE LOS 6 CUERPOS

% Carga de datos iniciales (Cada componente i se
   refiere, respectivamente, a: Sol, Jupiter, Saturno,
   Urano, Neptuno y Pluton)

% Masas de los cuerpos
global m G;
m = [1.00000597682;
     0.000954786104043;
     0.000285583733151;
     0.0000437273164546;
     0.0000517759138449;
```

```
    (1/(1.3e8))];

% Velocidades iniciales
v(:,1) = [0; 0; 0];
v(:,2) = [0.00565429; -0.00412490; -0.00190589];
v(:,3) = [0.00168318; 0.00483525; 0.00192462];
v(:,4) = [0.00354178; 0.00137102; 0.00055029];
v(:,5) = [0.00288930; 0.00114527; 0.00039677];
v(:,6) = [0.00276725; -0.00170702; -0.00136504];

% Momentos iniciales
p = v .* m';

% Posiciones iniciales
q(:,1) = [0; 0; 0];
q(:,2) = [-3.5023653; -3.8169847; -1.5507963];
q(:,3) = [9.0755314 ; -3.0458353; -1.6483708];
q(:,4) = [8.3101120; -16.2901086; -7.2521278];
q(:,5) = [11.4707666; -25.7294829; -10.8169456];
q(:,6) = [-15.5387357; -25.2225594; -3.1902382];

% Constante de gravitacion universal
G = 2.95912208286e-4;

% Parametros iniciales
format short e
t_simulacion = 1.0e5; % Tiempo de simulacion
h = 250/3; % Longitud de paso
num_guardar_datos = 1; % Guardar aproximaciones
maxiter = 50; % Numero maximo de iteraciones
x0 = [p;q]; % Momentos y posiciones iniciales

% Ejecucion programas
aprox_v3 = Gauss8v3(maxiter, h, t_simulacion,
   num_guardar_datos, x0);

% Tiempos para las graficas
tiempos = 0 : h : t_simulacion;
N = length(tiempos);

% Carga datos vpa
```

```
load("datosVPA_tsim_e5_h_250.mat");
aprox_vpa = aproximaciones;

% GRAFICA ERROR SALIDA DATOS

% Calculo error en la solucion
error_v3 = zeros(1, N);
for t = 1:N
    error_v3(t) = norm(aprox_vpa(:,:,t) - aprox_v3(:,:,
       t), 'fro');
end

% Figura
figure;
loglog(tiempos, error_v3, '-k', 'LineWidth', 1.5);
title('Gauss8v3');
xlabel('Tiempo');
ylabel('Error');
grid on

% GRAFICA ERROR ENERGIA

% Calculo error de la energia
error_ham_v3 = zeros(1, N);
H0 = hamiltoniano_6cuerpos(x0); % Energia inicial
for i = 1 : N
    error_ham_v3(i) = abs(H0 - hamiltoniano_6cuerpos(
       aprox_v3(:,:,i)));
end

% Figura
figure
semilogy(tiempos, error_ham_v3, '-k', 'LineWidth', 1.5)
   ;
title('Gauss8v3');
xlabel('Tiempo');
ylabel('Error en la Energia');
grid on
```

Código B.1: principal6cuerpos.m

- **Función 6 cuerpos**

  Función **f** para el problema de los 6 cuerpos.

```
function [fx] = funcion_6cuerpos(x)
% PARAMETROS DE ENTRADA
% x: matriz 6x6 de (momentos-posiciones, cuerpos)
% PARAMETROS DE SALIDA
% fx: matriz con las derivadas de momentos y posiciones

% Definicion de variables
p = x(1:3, :);
q = x(4:6, :);
dp = zeros(3, 6);
dq = zeros(3, 6);
fx = zeros(6, 6);
% Calculo dp_i/dt
for i = 2 : 6
  for j = 1 : i-1
      % Cubo de la distancia entre i y j
      d3(i,j) = norm(q(:, i) - q(:, j))^3;
      d3(j,i) = d3(i,j);
  end
end
for i = 1 : 6
  for j = 1 : 6
      if j ~= i
        % Contribucion de j al cambio de momento de i
          dp(:, i) = dp(:, i) - (m(i) * m(j) / d3(i,j))
              .* (q(:, i) - q(:, j));
      end
  end
  dp(:, i) = G .* dp(:, i);
end
% Calcular dq_i/dt
dq = p ./ m';
% Calculo fx
fx(1:3, :) = dp;
fx(4:6, :) = dq;
end
```

Código B.2: funcion_6cuerpos.m

- **Hamiltoniano 6 cuerpos**

  Función para calcular el Hamiltoniano del problema de los 6 cuerpos.

```
function [H] = hamiltoniano_6cuerpos(x)
% PARAMETROS INICIALES
% x: matriz de momentos y posiciones del Sol y los 5
   planetas.
% PARAMETROS DE SALIDA
% H: valor del hamiltoniano

% UNIDADES: la distancia se mide en Unidades
   Astronomicas (UA) (1 [A.U.] =
% 149597870 [km]), y el tiempo en dias terrestres.
% Carga de variables globales
global m;
global G;
% Definicion de momentos y posiciones
p = x(1:3, :); % Momentos p_i
q = x(4:6, :); % Posiciones q_i

% Funcion Hamiltoniana H = H1 + H2
% Calculo del primer termino H1
H1 = 0;
for i = 1 : 6
    H1 = H1 + (p(:,i)' * p(:,i) / m(i));
end
H1 = H1 / 2;

% Calculo del segundo termino H2
H2 = 0;
for i = 1 : 5
    for j = i+1 : 6
        H2 = H2 + (m(i)*m(j)) / norm(q(:,i) - q(:,j));
    end
end
H2 = -G * H2;

% Funcion Hamiltoniana
H = H1 + H2;
end
```

Código B.3: hamiltoniano_6cuerpos.m

- **Coeficientes Runge-Kutta del método de 4 etapas y orden 8**

  Función auxiliar para cargar los coeficientes b_i y μ_{ij} del método Runge-Kutta de cuatro etapas y orden ocho.

```
function [mu, b] = coef_cuatro_etapas()
% PARAMETROS FINALES
% mu: matriz de los coeficientes mu(ij).
% b: vector de los coeficienetes b(i).

% Variables auxiliares
w1 = (1/8) - (sqrt(30)/144);
ww1 = (1/8) + (sqrt(30)/144);
w2 = 1/2 * (((15 + 2 * sqrt(30)) / 35)^(1/2));
ww2 = 1/2 * (((15 - 2 * sqrt(30)) / 35)^(1/2));
w3 = w2 * (1/6 + sqrt(30)/24);
ww3 = ww2 * (1/6 - sqrt(30)/24);
w4 = w2 * (1/21 + (5 * sqrt(30)) / 168);
ww4 = ww2 * (1/21 - (5 * sqrt(30)) / 168);
w5 = w2 - 2 * w3;
ww5 = ww2 - 2 * ww3;
% Vector b(i)
b = 2 * [w1; ww1; ww1; w1];
% Matriz A(i,j)
mu(1,:) = [w1; ww1-w3+ww4; ww1-w3-ww4; w1-w5];
mu(2,:) = [w1-ww3+w4; ww1; ww1-ww5; w1-ww3-w4];
mu(3,:) = [w1+ww3+w4; ww1+ww5; ww1; w1+ww3-w4];
mu(4,:) = [w1+w5; ww1+w3+ww4; ww1+w3-ww4; w1];
% Coeficientes mu
mu(:, 1) = mu(:, 1) / b(1);
mu(:, 2) = mu(:, 2) / b(2);
mu(:, 3) = mu(:, 3) / b(3);
mu(:, 4) = mu(:, 4) / b(4);
% Redefinicion de los coeficientes mu
for i = 1 : 4
  for j = i+1 : 4
      mu(j, i) = 1 - mu(i, j);
  end
end
end
```

Código B.4: coef_cuatro_etapas.m

- **Suma compensada de Kahan**

  Algoritmo de sumación compensada de Kahan, aplicado de forma matricial.

```
function suma = sumaKahan(x, L, coef)
% PARAMETROS DE ENTRADA
% x: Matriz 2D 6x6 a la que hay que aplicar el algoritmo de Kahan
% L: Matriz 3D 4x6x6 con los sumandos
% coef: coeficientes de L
% PARAMETROS DE SALIDA
% suma: Resultado de la suma

% Inicializacion de datos
suma = x; % Matriz de la suma
c = zeros(6,6); % Matriz de compensaciones
n = size(L, 1); % Numero de sumandos

% Aplicar el algoritmo de Kahan en cada elemento de la matriz
for j = 1 : n
  % Multiplicacion a Li los coeficientes mu
  L(j, :, :) = coef(j) .* squeeze(L(j, :, :));
  X = squeeze(L(j, :, :)) - c;
  t = suma + X;
  c = (t - suma) - X;
  suma = t;
end
end
```

Código B.5: sumaKahan.m

- **Gauss8v0 VPA**

  Método de Gauss de cuatro etapas y orden ocho, en su versión Básica y aplicando **VPA**.

```
function [aproximaciones] = Gauss8VPA(maxiter, h, t_simulacion, num_guardar_datos, x0)
% PARAMETROS DE ENTRADA
```

```
% maxiter: numero maximo de iteraciones
% h: longitud de paso
% t_simulacion: tiempo de simulacion
% num_guardar_datos: numero de pasos para guardar datos
% x0: matriz con los momentos-posiciones iniciales
% PARAMETROS DE SALIDA
% aproximaciones: Matriz con las aproximaciones de 3
   dimensiones: momentos-posiciones, cuerpos, tiempos

% Configurar precision cuadruple
digits(34);

% Variables de las dimensiones
D = 3; % Dimension del problema
num_cuerpos = 6; % Numero de cuerpos del problema
tol = vpa(1/2 * 1.0e-18 * h); % Tolerancia

% Variables para guardar datos
contador_guardado = 1;
pasos = vpa(round(t_simulacion / h));
aproximaciones = vpa(zeros(double(2*D), double(
   num_cuerpos), double(pasos/num_guardar_datos)));
x = vpa(zeros(2*D, num_cuerpos));

% Carga de coeficientes
[mu, b] = coef_cuatro_etapas();
mu = vpa(mu);
b = vpa(b);

% Guardar coeficientes h*b
hb = vpa(zeros(size(b)));
hb(2:end-1) = h .* b(2:end-1);
hb(1) = (h - sum(hb)) / 2;
hb(end) = hb(1);

% Matriz de posiciones iniciales
x = x0;
aproximaciones(:,:,1) = x;

% Bucle para los pasos en la simulacion
for num_paso = 1 : double(pasos)
```

```
42  Y1 = x;
43  Y2 = x;
44  Y3 = x;
45  Y4 = x;
46  L = vpa(zeros(4, 2*D, num_cuerpos));
47  Y = vpa(zeros(2*D*4, num_cuerpos));
48
49  % Valores iniciales
50  niter = 0;
51  dif = tol + 1;
52
53  while (niter < maxiter) && (dif > tol)
54      Y1 = funcion_6cuerpos(Y1);
55      Y2 = funcion_6cuerpos(Y2);
56      Y3 = funcion_6cuerpos(Y3);
57      Y4 = funcion_6cuerpos(Y4);
58
59      L(1,:,:) = hb(1) .* Y1;
60      L(2,:,:) = hb(2) .* Y2;
61      L(3,:,:) = hb(3) .* Y3;
62      L(4,:,:) = hb(4) .* Y4;
63
64      % Limpiar las variables Y_i's
65      Y1 = vpa(zeros(2*D, num_cuerpos));
66      Y2 = vpa(zeros(2*D, num_cuerpos));
67      Y3 = vpa(zeros(2*D, num_cuerpos));
68      Y4 = vpa(zeros(2*D, num_cuerpos));
69
70      for j = 1 : 4
71          Y1 = Y1 + (mu(1, j) .* squeeze(L(j, :, :)));
72          Y2 = Y2 + (mu(2, j) .* squeeze(L(j, :, :)));
73          Y3 = Y3 + (mu(3, j) .* squeeze(L(j, :, :)));
74            Y4 = Y4 + (mu(4, j) .* squeeze(L(j, :, :)))
                ;
75      end
76
77      % Error en la iteracion
78      dif = vpa(norm(Y - [Y1;Y2;Y3;Y4], 'fro'));
79
80      % Actualizar datos
81      Y = [Y1;Y2;Y3;Y4];
```

```
82          Y1 = x + Y1;
83          Y2 = x + Y2;
84          Y3 = x + Y3;
85          Y4 = x + Y4;
86          niter = niter + 1;
87      end
88
89      if(dif > tol)
90          fprintf('No se ha alcanzado la tolerancia deseada
                \n');
91      end
92
93      % Aproximacion final y actualizacion
94      for j = 1 : 4
95          x = x + squeeze(L(j, :, :));
96      end
97
98      % Guardar datos
99      if mod(num_paso, num_guardar_datos) == 0
100         aproximaciones(:,:,contador_guardado + 1) = x;
101         contador_guardado = contador_guardado + 1;
102     end
103 end
104 save('datosVPA_tsim_e5_h_250.mat', 'aproximaciones');
105 end
```

Código B.6: Gauss8VPA.m

- **Gauss8v3**

  Método de Gauss de cuatro etapas y orden ocho, incorporando la mejora del criterio de parada y el algoritmo de sumación compensada de Kahan.

```
1  function [aproximaciones] = Gauss8v3(maxiter, h,
       t_simulacion, num_guardar_datos, x0)
2  % PARAMETROS DE ENTRADA
3  % maxiter: numero maximo de iteraciones
4  % h: longitud de paso (en dias)
5  % t_simulacion: tiempo de simulacion (en dias)
6  % num_guardar_datos: numero de pasos para guardar datos
```

```
% x0: matriz con los momentos-posiciones iniciales
% PARAMETROS DE SALIDA
% aproximaciones: % Matriz con las aproximaciones de 3 dimensiones: momentos-posiciones, cuerpos, tiempos

% INICIO DEL PROGRAMA
% Variables de las dimensiones
D = 3; % Dimension del problema
num_cuerpos = 6; % Numero de cuerpos del problema

% Variables para guardar datos
contador_guardado = 1; % Contador para guardar datos
pasos = round(t_simulacion / h);  % Numero de pasos en la simulacion
aproximaciones = zeros(2*D, num_cuerpos, pasos/num_guardar_datos); % Matriz con las aproximaciones de 3 dimensiones: momentos-posiciones, cuerpos, tiempos

% Carga de coeficientes
[mu, b] = coef_cuatro_etapas();

% Guardar coeficientes h*b
hb = zeros(size(b));
hb(2:end-1) = h .* b(2:end-1);
hb(1) = (h - sum(hb)) / 2;
hb(end) = hb(1);

% Matriz de posiciones iniciales
x = x0; % Matriz de momentos y posiciones
aproximaciones(:,:,1) = x; % Aproximacion inicial

% Bucle para los pasos en la simulacion
for num_paso = 1 : pasos

  % Inicio datos etapas
  Y1 = x; % Primera etapa
  Y2 = x; % Segunda etapa
  Y3 = x; % Tercera etapa
  Y4 = x; % Cuarta etapa
  L = zeros(4, 2*D, num_cuerpos); % Matriz 3D de etapas
```

```
    , componentes y cuerpos
Y = zeros(2*D*4, num_cuerpos);

% Valores iniciales para entrar en el bucle while
niter = 0;
dif_min = ones(2*D*4, num_cuerpos);
criterio = false;
flag = 0;

% Bucle para calcular la siguiente aproximacion
while (niter < maxiter) && (~ criterio)

    % Inicio de la iteracion de punto fijo
    Y1 = funcion_6cuerpos(Y1);
    Y2 = funcion_6cuerpos(Y2);
    Y3 = funcion_6cuerpos(Y3);
    Y4 = funcion_6cuerpos(Y4);

    L(1,:,:) = hb(1) .* Y1;
    L(2,:,:) = hb(2) .* Y2;
    L(3,:,:) = hb(3) .* Y3;
    L(4,:,:) = hb(4) .* Y4;

    % Actualizar las Y_i's
    Y1 = sumaKahan(zeros(6,6), L, mu(1,:));
    Y2 = sumaKahan(zeros(6,6), L, mu(2,:));
    Y3 = sumaKahan(zeros(6,6), L, mu(3,:));
    Y4 = sumaKahan(zeros(6,6), L, mu(4,:));

    % Error en la iteracion
    dif = abs(Y - [Y1;Y2;Y3;Y4]);
    if (dif == 0)
        criterio = true; % Fin de las iteraciones
    else
        criterio = all(dif_min <= dif, 'all');
        if (criterio)
            if(flag == 0) % Primera vez
                flag = 1;
                criterio = false;
                % Actualizar dif_min
                mask = (dif > 0); % Evitar ceros
```

```
83                      dif_min(mask) = min(dif_min(mask),
                           dif(mask));
84                  end
85              else
86                  flag = 0;
87                  % Actualizar dif_min
88                  mask = (dif > 0); % Evitar ceros
89                  dif_min(mask) = min(dif_min(mask), dif(
                       mask));
90              end
91          end
92
93          % Actualizar datos
94          Y = [Y1;Y2;Y3;Y4];
95          Y1 = x + Y1;
96          Y2 = x + Y2;
97          Y3 = x + Y3;
98          Y4 = x + Y4;
99          niter = niter + 1;
100
101     end
102
103     % Mensaje en caso de no converger
104     if(niter == maxiter)
105         fprintf(['\nNo se ha alcanzado la aproximacion
                con la ' ...
106              'tolerancia deseada'])
107     end
108
109     % Aproximacion final y actualizacion
110     x = sumaKahan(x, L, ones(1,4));
111
112     % Guardar datos
113     if mod(num_paso, num_guardar_datos) == 0
114         aproximaciones(:,:,contador_guardado + 1) = x;
115         contador_guardado = contador_guardado + 1;
116     end
117 end
118 end
```

Código B.7: Gauss8v3.m