UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

SIMULATION OF DIFFERENTIAL EQUATIONS USING NVIDIA CUDA

DANIEL MORENO OYA

Director/a: FERRAN MAZZANTI CASTRILLEJO (Departamento de Física)

Codirector/a: GRIGORI ASTRAKHARCHIK (Departamento de Física)

Titulación: Grado en Ingeniería Informática (Computación)

Memoria del trabajo de fin de grado

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

13/05/2024

Resumen

El proyecto trata sobre la segunda parte del decimosexto problema de Hilbert, es decir, la búsqueda de ciclos límites en sistemas de ecuaciones diferenciales cuadráticas. Se ha desarrollado un programa en CUDA C para tener soluciones numéricas que se ejecuta en GPU para utilizar la potencia computacional que tienen las GPUs.

El programa paralelo utiliza el método de Runge-Kutta de cuarto orden para solucionar el sistema de ecuaciones diferenciales con interpolación inversa cuadrática, para mejorar tiempo de ejecución sin perder precisión. Y a continuación, se aplica un mapeo de Poincaré para hacer la búsqueda de ciclos límite.

Índice

1. Contexto

El proyecto que definiré a continuación corresponde al Trabajo Final de Grado de la Facultad de Informática de Barcelona. Este proyecto tiene como director al profesor Ferran Mazzanti Castrillejo y como subdirector a Grigori Astrakharchik. Ambos profesores pertenecen al Departamento de Física de la FIB.

El proyecto tratará sobre la segunda parte del decimosexto problema definido por Hilbert. Se abordará este problema de forma numérica y desarrollando un algoritmo paralelo escrito en el lenguaje de programación CUDA. Este programa se ejecutará en nodos con tarjetas gráficas potentes para hacer cálculos masivos para hacer un análisis numérico del problema.

1.1. Términos y conceptos

Hilbert, en el Segundo Congreso Internacional de Matemáticas en 1900, propuso una lista de 23 problemas matemáticos [1]. El decimosexto trata sobre la cuestión del número máximo y posición de los ciclos frontera de Poincaré (ciclos límites) para un sistema de dos ecuaciones diferenciales bidimensionales:

$$\begin{aligned} \frac{dx}{dt} &= a_1x^2 + b_1xy + c_1y^2 + \alpha_1x + \beta_1y \\ \frac{dy}{dt} &= a_2x^2 + b_2xy + c_2y^2 + \alpha_2x + \beta_2y \end{aligned} \tag{1}$$

A continuación, definiré los principales conceptos que se necesitan para solucionar este problema matemático.

1.1.1. Ecuaciones diferenciales

Una ecuación diferencial es una ecuación que involucra a las derivadas de una función con la propia función y/o las variables de las que dependen[2]. Si la función depende solo de una variable, la ecuación se llama una ecuación diferencial ordinaria, en cambio, si la función depende de más de una variable, la ecuación se denomina ecuación diferencial parcial. Un ejemplo de ecuación diferencial ordinaria puede ser:

$$\frac{dy}{dx} = 2x, \tag{2}$$

donde la variable independiente es x, y la variable dependiente es y. En cambio, una ecuación diferencial parcial puede ser:

$$\frac{d^2V}{dx^2} + 2\frac{d^2V}{dy^2} = V, \tag{3}$$

donde las variables independientes son x e y, y la variable dependiente es V.

El orden de una ecuación diferencial se determina por el orden mayor de su derivada, en cambio, el grado de una ecuación diferencial viene dado por el exponente del mayor orden de su derivada.

Este tipo de ecuaciones son útiles en contextos donde se lleve a cabo un proceso que cambie continuamente en relación con el tiempo (rapidez de variación de una variable con respecto a otra).

1.1.2. Funciones racionales

Una función racional es aquella que viene dada por un cociente de polinomios,[3] como, por ejemplo:

$$f(x) = \frac{p(x)}{q(x)}, \tag{4}$$

donde p(x) y q(x) son polinomios sin factores comunes entre sí.

1.1.3. Ciclos frontera de Poincaré (ciclos límite)

Antes de definir el concepto de ciclos límites debemos conocer unos conceptos:

- Trayectoria: Es el camino que sigue un objeto en el espacio a lo largo del tiempo. Se describe con x(t) e y(t).
- Sistema dinámico: Es un sistema en el que una función describe la dependencia temporal de un punto en un espacio geométrico.[4]
- Espacio de fase: Es un espacio en el que se representan todos los estados posibles de un sistema, con cada estado posible correspondiente a un punto único en el espacio de fase.[5]

En el estudio de sistemas dinámicos con espacio de fase bidimensional, si se considera el siguiente sistema:

$$\begin{aligned} \frac{dx}{dt} &= f(x,y) \\ \frac{dy}{dt} &= g(x,y) \end{aligned} \tag{5}$$

$$\begin{aligned} x(t_0) &= x_0 \\ y(t_0) &= y_0 \end{aligned} \tag{6}$$

Si una solución del sistema 5 tiende a un punto cuando $t \to \infty$, entonces es un punto crítico (punto donde la derivada de la función es igual a cero o donde la función no es diferenciable). En los problemas no lineales, como el nuestro, aparecen algunas soluciones que repelen o atraen a las curvas integrales próximas. Estas soluciones atractoras (que no son puntos críticos) van a ser necesariamente periódicas y se llaman ciclos límite. [6]

Un método para detectar si hay ciclos límites es detectar si existen soluciones periódicas para el sistema. Es decir, si existe alguna solución x con $T > 0$ tal que $x(T + t) = x(t)$ para todo $t \geq 0$.

1.1.4. Método de Runge-Kutta de cuarto orden (RK4)

El método de Runge-Kutta de orden 4 es un algoritmo numérico utilizado para resolver ecuaciones diferenciales ordinarias [7]. En nuestro caso, para un sistema de dos ecuaciones diferenciales se define como un problema de valor inicial como:

$$\begin{aligned} \frac{dx}{dt} &= a_1x^2 + b_1xy + c_1y^2 + \alpha_1x + \beta_1y = f(x,y,t) \\ \frac{dy}{dt} &= a_2x^2 + b_2xy + c_2y^2 + \alpha_2x + \beta_2y = g(x,y,t) \end{aligned} \tag{7}$$

$$\begin{aligned} x(t_0) &= x_0 \\ y(t_0) &= y_0 \end{aligned} \tag{8}$$

Por lo tanto, el método RK4 para este problema está dado por el siguiente sistema de ecuaciones:

$$\begin{aligned} f(x,y,t+h) &= f(x,y,t) + \tfrac{1}{6} \cdot (h \cdot (kx1 + 2kx2 + 2kx3 + kx4)) \\ g(x,y,t+h) &= g(x,y,t) + \tfrac{1}{6} \cdot (h \cdot (ky1 + 2ky2 + 2ky3 + ky4)) \end{aligned} \tag{9}$$

Como observamos, en la ecuación 9, para calcular $f(x,y,t+h)$ necesitamos el valor de $f(x,t)$ y para $g(x,y,t+h)$ necesitamos el valor de $g(x,t)$. Los valores de $kx1$, $kx2$, $kx3$, $kx4$, $ky1$, $ky2$, $ky3$ y $ky4$ se calculan de la siguiente forma:

$$
\begin{aligned}
kx1 &= f(x_0, y_0, t_0) & ky1 &= g(x_0, y_0, t_0) \\
kx2 &= f(x_0 + \frac{1}{2}\cdot h \cdot kx1, y_0 + \frac{1}{2}\cdot h \cdot ky1, t + \frac{h}{2}) & ky2 &= g(x_0 + \frac{1}{2}\cdot h \cdot kx1, y_0 + \frac{1}{2}\cdot h \cdot ky1, t + \frac{h}{2}) \\
kx3 &= f(x_0 + \frac{1}{2}\cdot h \cdot kx2, y_0 + \frac{1}{2}\cdot h \cdot ky2, t + \frac{h}{2}) & ky3 &= g(x_0 + \frac{1}{2}\cdot h \cdot kx2, y_0 + \frac{1}{2}\cdot h \cdot ky2, t + \frac{h}{2}) \\
kx4 &= f(x_0 + h \cdot kx3, y_0 + h \cdot kyr, t + h) & ky4 &= g(x_0 + h \cdot kx3, y_0 + h \cdot ky3, t + h)
\end{aligned}
$$

Así, los siguientes valores $f(x, y, t+h)$ y $g(x, y, t+h)$ son determinados por los valores presentes $f(x, y, t)$ y $g(x, y, t)$ más el producto del tamaño del intervalo (h) por una pendiente estimada.

La pendiente es un promedio ponderado de pendientes, donde $kx1$ y $ky1$ son las pendientes al principio del intervalo, $kx2$ y $ky2$ son las pendientes en el punto medio del intervalo, usando para determinar el valor de x o y respectivamente en el punto $x_0 + h/2$ o $y_0 + h/2$ usando el método de Euluer. $kx3$ y $ky3$ son otra vez las pendientes del punto medio, pero ahora usando $kx2$ y $ky2$ para determinar el valor de x o y respectivamente; $kx4$ y $ky4$ son las pendientes al final del intervalo, con el valor de x o y determinado por $kx3$ o $ky3$. Promediando las cuatro pendientes, se le asigna mayor peso a las pendientes en el punto medio.

1.1.5. Método de la interpolación cuadrática inversa

El método de la interpolación cuadrática inversa es un método que se utiliza para encontrar las raíces de funciones, es decir, $f(x) = 0$.[8] Este método utiliza tres puntos para interpolar la curva mediante un polinomio. Esta interpolación es una aproximación de la curva original, pero en la interpolación es más fácil encontrar su raíz. En nuestro caso, los tres puntos que utilizamos para calcular esta interpolación son el penúltimo punto calculado con RK4, el último punto calculado con RK4 y el actual, también calculado con RK4 :

$$
\begin{aligned}
x\_ant\_ant, y\_ant\_ant &= RK4(x0, y0) \\
x\_ant, y\_ant &= RK4(x\_ant\_ant, y\_ant\_ant) \\
x, y &= RK4(x\_ant, y\_ant)
\end{aligned}
\tag{10}
$$

A continuación se aplica la siguiente fórmula para obtener la x interpolada:

$$
\begin{aligned}
a &= \frac{x\_ant\_ant \cdot y\_ant \cdot y}{(y\_ant\_ant - y\_ant) \cdot (y\_ant\_ant - y)} \\
b &= \frac{x\_ant \cdot y\_ant\_ant \cdot y}{(y\_ant - y\_ant\_ant) \cdot (y\_ant - y)} \\
c &= \frac{x \cdot y\_ant\_ant \cdot y\_ant}{(y - y\_ant\_ant) \cdot (y - y\_ant)} \\
x\_interpolada &= a + b + c
\end{aligned}
\tag{11}
$$

Se aplica este método con RK4 para poder tener un dt mayor en RK4 con el objetivo de disminuir el tiempo de ejecución sin perder precisión.

1.2. Actores implicados

Los actores implicados en el proyecto son Ferran Mazzanti Castrillejo, como director del proyecto, Grigori Astrakharchik, como codirector del proyecto, y yo como programador. El director y el codirector tienen como tarea dar soporte técnico y matemático y asegurarse que el proyecto se realiza de manera correcta.

Los principales beneficiarios del proyecto es la comunidad científica y matemática, ya que tendrán nuevos resultados del problema, o incluso el programa que se desarrolla para obtener más resultados. Los matemáticos que trabajan en encontrar solución al problema podrían utilizar estos resultados o el programa para inducir métodos, ya que la solución a este problema solo puede ser analítica.

Además, cualquiera que intente solucionar este problema podría iniciar la resolución utilizando este proyecto para perfeccionarlo, para intentar solucionar los obstáculos que puede tener este proyecto...

2. Justificación

2.1. Estudios previos

La segunda parte del problema 16 de Hilbert lleva más de 100 años sin resolverse, pero ha habido muchos matemáticos que han intentado resolver el problema, pero sin éxito. Solamente se conocen algunos campos vectoriales cuadráticos con cuatro ciclos límites. Pero todos estudios sobre el problema son estudios analíticos, no numéricos, por lo tanto, no podemos tener una solución concluyente.

2.2. Herramienas

Los estudios previos no nos sirven de mucha utilidad para iniciar nuestro proyecto, ya que ninguno ha obtenido un buen resultado al problema. Sin embargo, se creará un programa para ejecutar en GPU donde podemos hacer cálculos masivos del problema con diferentes parámetros que definen el sistema y después haremos una búsqueda de este sistema para hallar ciclos límites. Para realizar este programa hemos decidido utilizar el lenguaje CUDA, que se utiliza para hacer programación en GPU. El motivo de hacer programación en GPU es porque podemos realizar paralelización del programa y así poder realizar más cálculos del problema en menos tiempo de ejecución.

3. Alcance del proyecto

3.1. Objetivos del proyecto

Uno de los principales objetivos es desarrollar un programa que representa el problema anteriormente explicado, el cual estén integrados todos los conceptos matemáticos que aparecen en el problema. Este programa debería poder representar un sistema dinámico no lineal bidimensional y donde aparezcan trayectorias. También deberíamos poder hacer búsquedas de ciclos límites en el sistema que queremos representar y poder visualizarlos para analizar mejor los resultados.

Seguidamente, se divide este objetivo principal del proyecto en los siguientes subobjetivos:

- El programa debe tener el mejor algoritmo para representar sistemas dinámicos no lineales bidimensionales con trayectorias y también el mejor algoritmo para buscar los ciclos límites del sistema dinámico. Deberíamos tener dos versiones de este algoritmo para poder comparar resultados y tiempos de ejecución: secuencial y paralela.
- Paralelizar las versiones secuenciales de los algoritmos. Los programas paralelizados deben estar escritos en el lenguaje de programación CUDA para poder ser ejecutados en GPU en lugar de CPU para tener mejores tiempos de ejecución. Para obtener los mejores tiempos de ejecución debe tener una configuración óptima en cuanto las especificaciones que tiene la GPU que utilizaremos para ejecutar el programa. También debe estar el programa bien paralelizado para obtener los mejores resultados posibles.
- Programar una generación *random* y no *random* de parámetros que requiere el programa para poder hacer un cálculo masivo del problema anteriormente descrito.

3.2. Requisitos del proyecto

Los principales requisitos del programa que se desarrolla es que sea un programa que sea correcto, es decir, que haga correctamente los cálculos matemáticos y que la solución sea correcta. Para validar este requisito se ha hecho un script para medir la precisión del algoritmo. Otro requisito es que el código sea legible por personas externas al proyecto, ya que como hemos explicado en el apartado 1.2, el programa pueda ser utilizado en un futuro. Otro requisito del programa es que los resultados se puedan ver de manera visual para hacer la solución de forma más visual e intuitiva.

3.3. Riesgos del proyecto

Los principales posibles riesgos y obstáculos del proyecto que pueden aparecer son:

- Falta de conocimientos matemáticos en los términos anteriormente descritos. Este posible riesgo puede provocar un retraso en los tiempos y hacer disminuir el tiempo empleado en realizar el programa en sí. También puede provocar una representación errónea de los sistemas matemáticos en el programa y un mal análisis de los resultados obtenidos.
- Poco conocimiento en CUDA y en programación para GPU. Este tipo de programación tengo poca experiencia ya, que no es un tipo de programación muy común y no la he utilizado mucho. También puede provocar que las tareas indicadas para aprender este lenguaje se extiendan y disminuir el tiempo de programación del programa.

3.4. Metodología del proyecto

La metodología escogida entre el director, el codirector y yo es una especie de metodología Agile. La cual consiste en hacer una reunión de aproximadamente 1 hora cada semana con el objetivo de ir contando los avances del proyecto, de solucionar dudas y de anunciar los siguientes pasos con tal de lograr el objetivo marcado del proyecto.

Hemos escogido esta metodología para darnos feedback sobre el proyecto e ir marcando objetivos a corto plazo para así ir avanzando lentamente, pero sin errores. Para la parte del software, crearemos un repositorio en Github para que los directores puedan ir analizando el código del programa o pueden corregir errores que puedan surgir en un futuro.

La herramienta escogida para hacer la planificación de las tareas y el seguimiento es Jira. Jira es un software que se utiliza para la gestión de proyectos y el seguimiento de este. Es la principal herramienta que utilizan los proyectos agiles.

Debido a la metodología escogida, el método de validación para comprobar que se van cumpliendo los objetivos del proyecto descritos en el apartado 3.1, son las reuniones semanales que hacemos. En estas reuniones comprobaremos que se siguen los pasos adecuados para la realización del proyecto y se cumplen los objetivos marcados.

4. Planificación temporal

El periodo en el cual haremos el proyecto está definido entre el 16 de enero hasta el 13 de mayo de 2024 (fechas de convocatoria del tribunal). Está previsto que la realización de este proyecto suponga 450 horas de trabajo (25 horas por ECTS).

4.1. Personal y material

El proyecto requiere el siguiente personal: director de proyecto/codirector de proyecto y programador. A continuación, definiré sus tareas:

- El director/codirector de proyecto tiene como tarea definir las características y el alcance del proyecto. También tienen como tarea definir las tareas, su duración y en qué consiste cada tarea.
- El programador tiene como principal tarea realizar el código del programa para cumplir con los objetivos del proyecto. Primero también debe comprender el problema e integrarlo en el programa.

En cuanto al material, haremos una diferenciación entre el software y el hardware. En términos de hardware, se necesita solamente mi ordenador personal.

En cambio, en términos de software necesitaremos:

- Lenguajes de programación C y Python: lenguajes de programación que utilizaremos para realizar el programa.
- CUDA: la interfaz con la GPU para hacer la paralelización y la programación en GPU.
- Git: software de control de versiones.
- Overleaf: editor LaTeX colaborativo basado en la nube que se utiliza para escribir, editar y publicar documentos.
- Microsoft PowerPoint: editor de presentaciones para realizar la presentación de la defensa oral.
- Skype: aplicación que utilizaremos para las reuniones semanales de seguimiento.
- Gmail: correo electrónico que utilizaremos para la comunicación con el director/codirector

del proyecto.

4.2. Fases del proyecto

Este proyecto está compuesto por 2 partes: la realización del programa y la elaboración de la memoria. La realización del programa está compuesta por diferentes fases:

- Estudio inicial (EI)
- Diseño e implementación de algoritmos (DIA)
- Resultados (R)

En cambio, la parte de la elaboración de la memoria y seguimiento del proyecto no contienen fases ya que tienen como duración todo el proyecto.

5. Descripción de las tareas

En este apartado haré una pequeña descripción de las tareas que contiene este proyecto y también detallaré una estimación de la duración de la tarea y los recursos necesarios para realizar dicha tarea. Los recursos humanos están detallados en la tabla 3, con la dedicación en horas de cada uno en cada tarea.

5.1. Estudio inicial

EI1 ALCANCE Y REQUISITOS

Esta tarea consiste en documentar el alcance y los requisitos del proyecto. Esta tarea tiene estimada una duración de 30 horas. Los recursos utilizados son el Gmail y Skype para contactar con el director/codirector y Overleaf para la documentación.

EI2 PLANIFICACIÓN TEMPORAL

Esta tarea consiste en documentar la planificación temporal del proyecto. Esta tarea tiene estimada una duración de 15 horas. Los recursos utilizados son el Gmail y Skype para contactar con el director/codirector y Overleaf para la documentación.

EI3 PRESUPUESTO Y ANÁLISIS DE SOSTENIBILIDAD

Esta tarea consiste en la documentación del presupuesto estimado del proyecto y el análisis de sostenibilidad. Esta tarea tiene una duración estimada de 15 horas. Los recursos utilizados son el Gmail y Skype para contactar con el director/codirector y Overleaf para la documentación.

EI4 DOCUMENTACIÓN FINAL CON LA SÍNTESIS DEL PROYECTO

Esta tarea consiste en la documentación de la síntesis del proyecto teniendo en cuenta el *feedback* de los profesores. Esta tarea tiene una duración estimada de 15 horas. Los recursos utilizados son el Gmail y Skype para contactar con el director/codirector y Overleaf para la documentación.

EI5 ESTUDIO MATEMÁTICO

Esta tarea consiste en comprender la segunda parte del problema 16 de Hilbert y entender todos los conceptos matemáticos utilizados en el proyecto. Esta tarea tiene una duración estimada de 40 horas. Dicha tarea tiene como recursos Skype y Gmail para realizar reuniones con el director/codirector y la búsqueda de información sobre el problema.

5.2. Diseño e implementación de algoritmos

DIA1 INTEGRACIÓN DE TRAYECTORIAS

Realizar un programa secuencial que, dado un sistema con unos parámetros y un punto inicial, integre la trayectoria de la función. Esta tarea también consiste en poder ver estas trayectorias en el plano y observar los puntos que pertenecen al ciclo límite. A continuación, debe detectar si es un ciclo límite. La duración estimada de esta tarra es de 60 horas. Los recursos necesarios para esta tarea son el lenguaje de programación C y mi ordenador.

DIA2 ESTUDIO Y PREPARACIÓN DE PROGRAMACIÓN EN CUDA

Esta tarea consiste en el estudio de la programación en GPU con CUDA. Esta tarea consiste en ver tutoriales y documentación de CUDA para adquirir el conocimiento necesario para poder realizar el proyecto. A continuación, debemos preparar el entorno necesario para poder programar en CUDA en mi ordenador. Esta tarea tiene una duración estimada de 40 horas. Los recursos necesarios para poder realizar esta tarea son mi ordenador y las herramientas para buscar información.

DIA3 PARALELIZACIÓN DE TRAYECTORIAS

Esta tarea consiste en realizar un programa en CUDA que integre trayectorias en GPU dando varios puntos iniciales. Después debemos recuperar los resultados y hacer una búsqueda de ciclos límites. Este programa es una paralelización del programa realizado en la tarea DIA1. La estimación temporal de esta tarea es de 40 horas. Los recursos necesarios para realizar la tarea son mi ordenador y la GPU.

DIA4 GENERADOR RANDOM DE PARÁMETROS

Esta tarea consiste en realizar un programa que haga una generación *random* y no *random* de parámetros que requiere el programa realizado en la tarea DIA3. Esta tarea está estima en 1 hora. Los recursos necesarios para dicha tarea es mi ordenador.

DIA5 PROGRAMA FINAL

Esta tarea consiste en integrar los programas hechos en las tareas DIA3 y DIA4 con un script para ejecutar en GPU, donde este script haga la generación *random* y no *random* de parámetros y, a continuación, la exploración de ciclos límites. Finalmente, debemos definir como obtener

estos resultados para poder analizarlos y exportarlos. Esta tarea tiene estimada una duración de 35 horas. Dicha tarea tiene como recursos la GPU y mi ordenador.

5.3. Resultados

R1 RESULTADOS Y CONCLUSIÓN

Esta tarea consiste en obtener los resultados de todas las ejecuciones en GPU y hacer un análisis de los resultados. Esta tarea tiene una duración estimada de 35 horas. Dicha tarea tiene como recursos la GPU y mi ordenador.

5.4. Documentación y seguimiento del proyecto

DS1 DOCUMENTACIÓN DEL PROYECTO

Esta tarea consiste en hacer la documentación continua del proyecto. Esta tarea tiene una duración estimada de 5 horas por semana, es decir, 90 horas en total. Los recursos necesarios para realizar dicha tarea son mi ordenador, Overleaf y las herramientas para buscar información.

DS2 SEGUIMIENTO DEL PROYECTO

Esta tarea consiste en las reuniones semanales para realizar el seguimiento del proyecto. La duración estimada es de una hora por semana, es decir, 18 horas en total. Los recursos necesarios son Skype y Gmail.

DS3 DEFENSA ORAL

Esta tarea consiste en la elaboración y preparación de la presentación de la defensa oral de este proyecto. Esta tarea tiene como duración estimada 25 horas. Los recursos necesarios para realizar dicha tarea son mi ordenador, Overleaf, Microsoft PowerPoint y las herramientas para buscar información.

Código	Nombre	Dependencias	Tiempo estimado
EI1	Alcance y requisitos		30 horas
EI2	Planificación temporal	EI1	15 horas
EI3	Presupuesto y análisis de sostenibilidad	EI1-2	15 horas
EI4	Documentación final con la síntesis del proyecto	EI1-3	15 horas
EI5	Estudio matemático		40 horas
DIA1	Integración de trayectorias	EI5	60 horas
DIA2	Estudio y preparación de programación en CUDA		40 horas
DIA3	Paralelización de trayectorias	EI5, DIA1-2	40 horas
DIA4	Generador *random* de parámetros	DIA1	1 hora
DIA5	Programa final	DIA1-5	35 horas
R1	Resultados y conclusión	DIA5	35 horas
DS1	Documentación del proyecto	EI1-5, DIA1-5, R1	90 horas
DS2	Seguimiento del proyecto		18 horas
DS3	Defensa oral	DS1	25 horas
		TOTAL	459 horas

Tabla 1: Resumen de las tareas. Elaboración propia.

5.5. Diagrama de Gantt

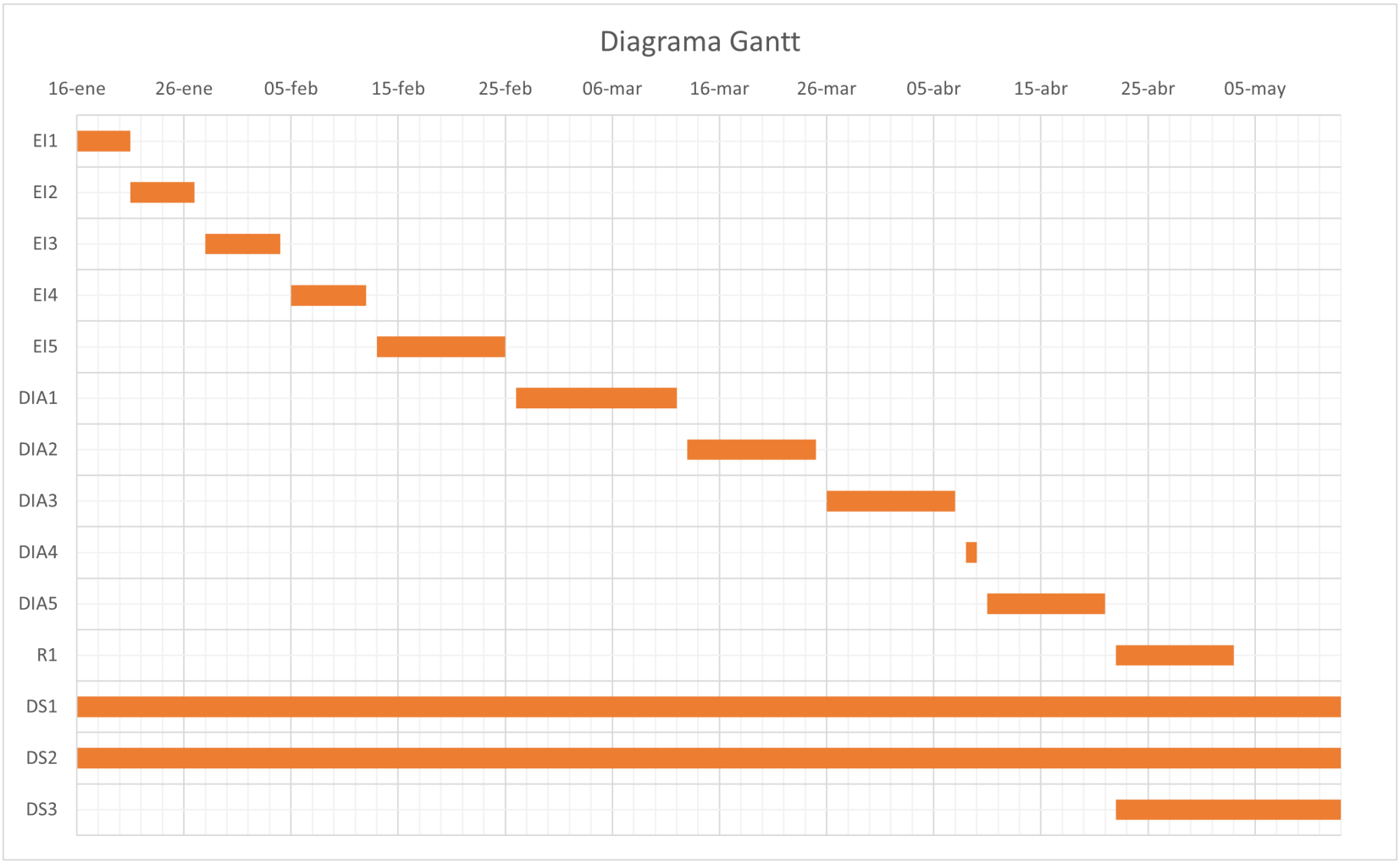

Figura 1: Diagrama de Gantt. Elaboración propia

6. Gestión del riesgo

En este apartado se describe los posibles obstáculos que puede haber desarrollando el proyecto y la manera de poder solucionarlos, o en el caso de que no haya solución, planes alternativos.

La mayoría de posibles obstáculos se pueden solucionar o minimizar, ya que hay tareas explícitas para mitigar esos obstáculos. Es el caso del obstáculo de falta de conocimientos matemáticos necesarios para comprender el problema. Para este posible obstáculo tenemos la tarea EI5 con 40 horas de duración estimada. Es una duración apropiada para intentar a comprender el problema, entender todos los conceptos matemáticos del el problema con el objetivo minimizar el obstáculo. Otra manera de minimizar este obstáculo son las reuniones semanales con mi director/codirector, dónde me pueden explicar el problema en detalle.

Otro posible riesgo es mi poco conocimiento de CUDA, lenguaje que se utiliza para paralelizar el código. Para este riesgo está la tarea DIA2 con 40 horas de duración estimada. Es un tiempo razonable para comprender el lenguaje y tener el conocimiento necesario para realizar los códigos del programa.

En el caso de que no se puedan superar estos obstáculos, existe un plan alternativo. Uno de ellos es dedicar más tiempo a las tareas EI5 y DIA2 reduciendo tiempo a otras tareas dependientas de estas, con el inconveniente de reducir calidad de los algoritmos. Otro plan alternativo es revisar el alcance del proyecto y hacerlo menos ambicioso para poder terminar el proyecto en los tiempos correspondientes.

7. Presupuesto

En este apartado, detallaremos el presupuesto de diferentes maneras: costes del personal por tarea, costes generales, costes de contingencia e imprevistos.

7.1. Costes del personal

A continuación, detallaremos un presupuesto estimado del coste del personal que trabajará en el proyecto. Para estimar el presupuesto primero debemos saber que sueldo tiene cada tipo de personal y hacer una estimación de cuántas horas dedica a cada tarea para calcular el coste.

Tipo personal	Salario Anual(€)	Coste anual (€)	Coste/Hora (€)
Director del proyecto	51144,00	66487,2	31,97
Codirector del proyecto	51144,00	66487,2	31,97
Programador junior	23184,00	30139,2	14,49

Tabla 2: Estimación del sueldo del personal. Elaboración propia.

El salario anual de cada tipo de personal son salarios brutos anuales.[9] [10] Para calcular los costes debemos añadir el coste de la Seguridad Social. Para ello multiplicamos el sueldo bruto por 1,3. Y finalmente, para calcular el coste por hora es dividir el coste anual entre 2080 (52 semanas por 40 horas semanales).

Para calcular el coste de cada tipo de personal debemos saber cuántas horas tiene asignadas en cada tarea:

Código	DP (horas)	CDP (horas)	PJ(horas)	Tiempo total (horas)	Coste (€)
EI1	5	5	25	30	681,95
EI2	2	2	13	15	316,25
EI3	2	2	13	15	316,25
EI4	2	2	13	15	316,25
EI5	5	5	35	40	826,85
DIA1	10	10	50	60	1363,9
DIA2	5	5	35	40	826,85
DIA3	5	5	35	40	826,85
DIA4	0	0	1	1	14,49
DIA5	5	5	30	35	754,4
R1	15	15	20	35	1248,9
DS1	10	10	80	90	1798,6
DS2	18	18	18	18	1411,74
DS3	2	2	23	25	461,15
				Total CPA	11164,43€

Tabla 3: Presupuesto del coste del personal por tarea. Elaboración propia.

7.2. Costes generales

7.2.1. Costes material hardware

El material de hardware necesario para realizar el proyecto viene descrito en el apartado 4.1.

Material hardware	Precio (€)	Unidades	Precio total (€)
Ordenador	1457,38	1	1457,38
		Total	1457,38€

Tabla 4: Precio del material de hardware. Elaboración propia.

La tabla 4 muestra los precios del material hardware. Para calcular las amortizaciones del hardware utilizaremos la fórmula: [11]

$$\text{Amortización} = \frac{\text{(Valor adquisición - Valor residual)}}{\text{nº años de vida útil}} \tag{12}$$

Los productos hardware tienen la siguiente amortización por año:

- Ordenador: La vida útil de material hardware es de 4 años y también hemos calculado que el valor residual es de 150€. Por lo tanto,

$$\text{Amortización} = \frac{(1457{,}38 - 300)}{4} = 289{,}35\,€/\text{año} \tag{13}$$

Finalmente debemos calcular la amortización del tiempo que utilizamos estos componentes en nuestro proyecto, hemos estimado que cada año tiene unos 250 días laborables:

- Ordenador: El tiempo estimado de utilización es de 438 horas, por lo tanto:

$$289{,}35\frac{€}{\text{año}} \cdot \frac{\text{año}}{250 \text{ días laborables}} \cdot \frac{1 \text{ día}}{8 \text{ horas laborables}} \cdot 459 \text{ horas} = 66{,}40\ € \tag{14}$$

7.2.2. Costes material software

El material de software necesario para realizar el proyecto viene descrito en apartado 4.1. El precio del material de hardware es el siguiente:

Material software	Precio (€)
Lenguaje de programación C y Python	0
Cuda	0
Git	0
Overleaf	0
Microsoft PowerPoint	7 (al mes)
Skype	0
Gmail	0
TOTAL	28€

Tabla 5: Precio del material de software. Elaboración propia.

Todos los costes de software que son gratuitos porque son software libre. En cambio, Microsoft Word y el Microsoft PowerPoint tienen un coste de 7 €/mes, el cual efectuaremos 4 pagos.[12]

El paquete de Microsoft Office solamente lo usaremos en las siguientes tareas: EI1, EI2, EI3, EI4, DS1 y DS3. Por lo tanto, lo utilizaremos 190 horas. Por lo tanto, la amortización será:

$$\text{Amortización} = \frac{28€}{120 \text{ días}} \cdot \frac{1 \text{ día}}{24 \text{ horas}} \cdot 190 \text{ horas} = 1{,}85 € \tag{15}$$

7.2.3. Costes energéticos

Los costes energéticos vienen generados por el consumo eléctrico del portátil y de las tarjetas gráficas.

Producto	Potencia (W)	Tiempo (horas)	Consumo (Wh)	Coste KWh	Coste (€)
Ordenador	650	459	298350	0,0697€/KWh	20,79
				TOTAL	20,79 €

Tabla 6: Estimación del coste energético. Elaboración propia.

La tabla 6 muestra el consumo eléctrico y el coste estimado de la utilización del ordenador.[13]. El coste del KWh es la media del día 27 de abril de 2024 en España.[14]

En cuanto el internet, actualmente estamos pagando 29 €/mes y vivimos 4 personas, por lo tanto, pagamos 7,25 €/mes por persona. El proyecto tiene estimado aproximadamente 120 días de proyecto y 3 horas al día dedicadas al proyecto. Por lo tanto, el coste será:

$$\text{Coste internet} = 7{,}25\frac{€}{\text{mes}} \cdot \frac{120 \text{ días proyecto}}{30 \text{ días por mes}} \cdot \frac{3 \text{ horas de proyecto}}{24 \text{ horas al día}} = 3{,}63€ \tag{16}$$

7.2.4. Costes espacio físico

El proyecto estará hecho en remoto y, por lo tanto, habrá que computar el gasto de alquiler de mi casa. Hemos estimado que el alquiler costaría 1400 €/mes, dónde vivimos 4 personas, por lo tanto, sería 350 €/mes por persona. El proyecto tiene estimado unos 120 días y 3 horas al día dedicadas al proyecto, por lo tanto:

$$\text{Coste espacio físico} = 350\frac{€}{\text{mes}} \cdot \frac{120 \text{ días proyecto}}{30 \text{ días por mes}} \cdot \frac{3 \text{ horas de proyecto}}{24 \text{ horas al día}} = 175€ \tag{17}$$

7.3. Costes de contingencia e imprevistos

Se ha incluido un coste de contingencia del 5 % respecto al total porque no hemos valorado costes de importe pequeño y también si ocurre cualquier imprevisto.

En cuanto a los imprevistos, están detallados en el apartado 3.3 y la manera de minimizarlos o solucionarlos está descrita en el apartado 6. En este apartado haremos un cálculo aproximado de los planes alternativos. El porcentaje estimado para que tengamos que activar estos planes alternativos para superar estos riesgos es del 25 %. Si debemos ejecutar estos planes alternativos, que en definitiva es ampliar el tiempo estimado de las tareas DIA5 y EI2 y con más presencia del director y codirector del proyecto, el coste de los imprevistos sería:

$$\begin{aligned}\text{Coste imprevistos} =& (15 \text{ horas extras director } \cdot 31,97 \frac{€}{\text{hora}}) \\ &+ (15 \text{ horas extras codirector } \cdot 31,97 \frac{€}{\text{hora}}) \\ &+ (15 \text{ horas extras programador} \cdot 14,49 \frac{€}{\text{hora}}) = 1176,59€\end{aligned} \tag{18}$$

Hemos estimado que cada personal hará 15 horas extras con tal de mitigar los riesgos. En definitiva, el coste final de los imprevistos sería:

$$\text{Coste imprevistos} = 1176,59€ \cdot 0,25 \text{ probabilidad} = 294,11€ \tag{19}$$

7.4. Presupuesto final

El presupuesto final del proyecto está representado en la tabla 7.

Tipo de coste	Presupuesto (€)
Coste personal por actividad (CPA)	11164,43
Amortización material hardware	66,40
Amortización material software	1,85
Coste energético	24,42
Coste espacio físico	175
TOTAL	11432,1€
Coste contigencia	571,61
TOTAL	12003,71€
Coste imprevistos	294,11
TOTAL	**12297,82€**

Tabla 7: Presupuesto final del proyecto. Elaboración propia.

7.5. Control de gestión

Con la estimación del presupuesto final del proyecto hemos definido mecanismos de control para no desviarse del presupuesto y no incrementar mucho el gasto. Para realizar este control sobre el presupuesto hemos definido unas métricas. Al finalizar cada tarea calcularemos estas métricas para saber si nos hemos desviado del presupuesto. También nos debemos fijar si la desviación es superior al coste de contingencia e imprevistos, que están incluidos en el presupuesto para casos como este. En el caso de que esta desviación sea superior al coste de contingencia e imprevistos, deberíamos hacer otro cálculo del presupuesto y actualizarlo con las nuevas desviaciones e inconvenientes que han aparecido.

Las métricas que utilizaremos son las siguientes:

- Desviación del coste de personal por tarea:

$$\text{(coste estimado tarea - coste real tarea)} \cdot \text{horas reales} \tag{20}$$

- Desviación del coste de material:

$$\text{(coste material real – coste material estimado)} \tag{21}$$

- Desviación total del coste de personal:

$$\text{(coste real personal – coste estimado personal)} \tag{22}$$

- Desviación total de horas:

$$\text{(horas reales – horas estimadas)} \tag{23}$$

- Desviación total del presupuesto:

$$\text{(coste total real – coste total estimado)} \tag{24}$$

8. Diseño de algoritmos secuenciales

8.1. Diseño de integración de trayectorias

Hasta la fecha, existen varios estudios donde el máximo número de ciclos límites en un sistema cuadrático polinomial bidimensional son 4. Para realizar el diseño de un algoritmo que integre las trayectorias de la ecuación 7, primero me he basado en uno de estos estudios [15]. Este estudio nos muestra los valores de las variables libres con el que obtenemos los 4 ciclos límites y un script de MatLab para poder visualizarlos. Se ha decidido acotar la búsqueda en los 3 ciclos límites del eje x positivo para no tener un espacio de búsqueda demasiado grande.

8.1.1. Algoritmo para visualizar tres ciclos límite

Para construir los 3 ciclos límites presentes en el eje x positivo, las trayectorias se deben construir con los siguientes valores de las variables libres de la ecuación 7 :

$$\begin{aligned} a_1 &= 1 \quad & a_2 &= -10{,}0 \\ b_1 &= 1 \quad & b_2 &= 2{,}2 \\ c_1 &= 0 \quad & c_2 &= 0{,}7 \\ \alpha_1 &= 0 \quad & \alpha_2 &= -72{,}7778 \\ \beta_1 &= 1 \quad & \beta_2 &= 0{,}0015 \end{aligned}$$

Tabla 8: Valor de las variables libres del sistema de ecuaciones 7. Valores extraidos de Visualization of four normal size limit cycles in two-dimensional polynomial quadratic system [15].

En nuestro código, que está en el archivo tfg_sec_inter.c, para simular estos 3 ciclos límite, primero debemos crear un vector de posiciones iniciales, en los cuales aplicaremos RK4 y buscaremos si en la trayectoria obtenida de RK4 que empieza en esta posición inicial contiene un ciclo límite. Como podemos ver en el código 1, se establece un rango de 0 a 16 en el eje x, ambos incluidos, con diferencia de 0.01 para calcular los puntos iniciales y almacenarlos en el vector init_pos. En cambio, todos los puntos iniciales tendrán $y = 0$. Por lo tanto, se calcularán 1600 trayectorias.

```
double start = 0; // Valor inicial del rango
double stop = 16; // Valor final del rango
double step = 0.01;  // Tamaño del paso entre números
// Calcula el tamaño del arreglo
int size = (stop - start) / step + 1;

// Crea un arreglo dinámico para almacenar los números
double* init_pos = (double*)malloc(size * sizeof(double));

// Llena el vector con los números en el rango
for (int i = 0; i < size; i++) {
    init_pos[i] = start + i * step;
}
```

Listing 1: Creación de los vectores de puntos iniciales, en cada posición inicial se aplicará RK4 con interpolación cuádratica inversa con mapeo en y=0. Elaboración propia.

El criterio que se ha definido para determinar si la trayectoria contiene un posible ciclo límite es si la trayectoria cruza dos veces el eje $y = 0$, ya que todas las trayectorias empiezan con $y = 0$. Este criterio lo aplicamos a cada trayectoria calculada con RK4 en cada punto inicial del vector init_pos con la ecuación 7 y los valores de las variables libres de la tabla 8.

Para una trayectoria $x(t)$, $y(t)$ con dos puntos de cruce en el eje $y = 0$ tenemos que $y(0) = y(T) = 0$, donde T es el periodo, es decir, el tiempo que pasa hasta el segundo cruce en el eje $y = 0$. Por lo tanto, la trayectoria contiene un ciclo límite si $x(T) - x(0) = 0$ porque significa que el segundo punto de cruce está en la misma posición del eje x que el punto inicial x ,y se forma un ciclo límite. Finalmente, se genera un gráfico del mapeo de $x(T) - x(0)$ como función de $x(0)$ para cada trayectoria calculada que tenga dos puntos de cruce. Este criterio de aceptación de existencia de ciclo límite se aplicará al mismo tiempo que RK4 para ahorrar memoria y tiempo de ejecución.

```
1  double* x_0 = (double*)malloc(capacidad * sizeof(double));
2  double* x_t = (double*)malloc(capacidad * sizeof(double));
3
4  struct Ciclo c;
5
6  for(int i=0;i<size;++i) {
7      if (longitud == capacidad) {
8          // Si el arreglo está lleno, aumenta la capacidad
9          capacidad *= 2;
10         x_0 = (double*)realloc(x_0, capacidad * sizeof(double));
11         x_t = (double*)realloc(x_t, capacidad * sizeof(double));
12         if (x_0 == NULL || x_t == NULL) {
13             fprintf(stderr, "Error al asignar memoria.\n");
14             return 1;
15         }
16     }
17     c = runge4_ciclos(init_pos[i]);
18     if(c.x != -1000.0 && c.y != 1000.0) {
19         x_0[longitud] = c.x;
20         x_t[longitud] = c.y;
21         ++longitud;
22     }
23 }
```

Listing 2: Creación de los vectores $x(0)$ y $x(T) - x(0)$ y aplicación de RK4 con interpolación cuádratica inversa con mapeo en $y = 0$ en cada posición del vector init_pos. Elaboración propia.

En el código 2 se puede ver el código donde se crean los vectores $x(0)$ y $x(T) - x(0)$ y después se aplica RK4 con mapeo en $y = 0$. En las líneas de código 1 y 2 se crean los vectores x_0 ($x(0)$) y x_t ($x(T) - x(0)$) con tamaño capacidad, en este caso es de 10 elementos. Esto se hace para ahorrar memoria, ya que no sabemos qué tamaño tendrán estos vectores porque dependen del número de trayectorias que tengan dos cruces en el eje $y = 0$.

A continuación, en el bucle for de la línea 6 a 23, primero se mira si los vectores están llenos y, en el caso de que sea así, se amplía el tamaño asignando un tamaño de $capacidad^2$ a los dos vectores. Después se aplica la función runge4_ciclos con el parámetro de entrada de un punto inicial, anteriormente calculado en el código 1. Esta función, que calcula RK4 con interpolación inversa cuadrática y con mapeo en $y = 0$, devuelve un struct Ciclo, el cual tiene como atributo x e y. Este struct representa un posible ciclo límite, por lo tanto, el atributo x es $x(0)$ y el atributo

y es $x(T) - x(0)$. Estos valores los guardamos en los dos vectores en el caso de que sean diferente a -1000 porque la función runge4_ciclos devuelve un struct Ciclo con valores -1000 en el caso que no haya dos cruces en el eje $y = 0$.

```
struct Ciclo runge4_ciclos(double x0) {
    double y0 = 0;
    double x,y;
    double t = 0.0;
    double    x_ant_ant = x0;
    double    y_ant_ant = y0;
    double    x_ant = x0;
    double    y_ant = y0;
    struct Punto p;
    p = calcular_posicion(x0,y0);
    x = p.x;
    y = p.y;
    int count = 0;
    while(t < 10.0 && count<2) {
        x_ant_ant = x_ant;
        y_ant_ant = y_ant;
        x_ant = x;
        y_ant = y;
        p = calcular_posicion(x_ant, y_ant);
        x = p.x;
        y = p.y;
        t = t + dt;
        double x_per;
        if(y*y_ant < 0) {
            count = count + 1;
            if(count == 1) {
                x_per = x0;
            }
            if(count == 2) {
                double x_inter = (x_ant_ant * (y_ant * y) / ((y_ant_ant - y_ant) * (y_ant_ant - y)) +
                                  + x_ant * (y_ant_ant * y) / ((y_ant - y_ant_ant) * (y_ant - y)) +
                                  + x * (y_ant_ant * y_ant) / ((y - y_ant_ant) * (y - y_ant)));
                double xdiff = x0 - x_inter;
                struct Ciclo c;
                c.x = x_per;
                c.y = xdiff;
                return c;
            }
        }
    }
    struct Ciclo c;
    c.x = -1000.0;
    c.y = -1000.0;
    return c;
}
```

Listing 3: Método de RK4 con interpolación cuadrática inversa y mapeo en $y = 0$. Elaboración propia.

En el código 3 se puede ver el código de la función runge4_ciclos donde se aplica RK4 con interpolación cuadrática inversa y mapeo en $y = 0$. La función runge4_ciclos tiene como parámetro de entrada un punto inicial x y devuelve un posible ciclo límite en forma de struct. En primer lugar, se pone como punto inicial $y0 = 0$ y la variable t a 0. Después se declaran las variables x_ant_ant, x_ant, y_ant_ant e y_ant, que son necesarias para la interpolación cuadrática inversa, y se les dan valor de x0 e y0, respectivamente. A continuación, se llama a la función calcular_posicion con los parámetros $x0$ e $y0$ y devuelve un struct Punto, que representa un punto en el plano. Este struct contiene dos atributos, x e y. La función calcular_posicion calcula Runge-Kutta y devuelve el punto obtenido. Se inicializa la variable count a 0, que representa el contador de cruces en el eje $y = 0$.

A continuación, se inicia el bucle while de la línea 14 con la condición que finalice en el caso que $t < 10{,}0$ y $count < 2$. Dentro del while, primero se actualizan las variables x_ant_ant, x_ant, y_ant_ant e y_ant con los valores correspondientes y después se llama a la función calcular_posicion con las variables x_ant e y_ant previamente calculadas. Se asigna este punto obtenido a las variables x e y y se suma dt a t, en este caso $dt = 1 \times 10^{-3}$. A continuación, si $y * y\_ant < 0$, es decir, se produce un cruce en el eje $y = 0$:

- Se suma 1 a la varible count, que es el contador de cruces del eje $y = 0$.
- A continuación, si es el primer cruce, se asigna la $x0$ a la variable x_per porque cuando $y0 = 0$ cuando $t = 0$. Y si es el segundo cruce, se asigna $x\_ant - x\_interpolada$, calculada con la fórmula 11, a la variable xdiff y se crea un Struct Ciclo con la atributo x con el valor x_per y el atributo y con el valor xdiff. Y finalmente se retorna este struct.

Después, ya fuera del while, se crea un Struct Ciclo con los atributos x e y con valor $-1000,0$ y se retorna este struct. Esto se hace en el caso de que no se encuentren dos cruces en el eje $y = 0$ y poder omitir estas trayectorias.

```
double a1,b1,c1,alpha1,beta1;
double a2,b2,c2,alpha2,beta2;

double dx_dt(double x, double y) {
    a1= 1;
    b1= 1;
    c1= 0;
    alpha1= 0;
    beta1= 1;
    return (a1*x*x) + (b1*x*y) + (c1*y*y) + (alpha1*x) + (beta1*y);
}

double dy_dt(double x, double y) {
    a2= -10.0;
    b2= 2.2;
    c2= 0.7;
    alpha2= -72.7778;
    beta2= 0.0015;
    return (a2*x*x) + (b2*x*y) + (c2*y*y) + (alpha2*x) + (beta2*y);
}

struct Punto calcular_posicion(double x_ant, double y_ant) {
    double k1_x = dx_dt(x_ant, y_ant);
    double k1_y = dy_dt(x_ant, y_ant);

    double k2_x = dx_dt(x_ant + 0.5*dt*k1_x, y_ant + 0.5*dt*k1_y);
    double k2_y = dy_dt(x_ant + 0.5*dt*k1_x, y_ant + 0.5*dt*k1_y);

    double k3_x = dx_dt(x_ant + 0.5*dt*k2_x, y_ant + 0.5*dt*k2_y);
    double k3_y = dy_dt(x_ant + 0.5*dt*k2_x, y_ant + 0.5*dt*k2_y);

    double k4_x = dx_dt(x_ant + dt*k3_x, y_ant + dt*k3_y);
    double k4_y = dy_dt(x_ant + dt*k3_x, y_ant + dt*k3_y);

    struct Punto punto;
    punto.x = x_ant + (dt/6.0)*(k1_x + 2*k2_x + 2*k3_x + k4_x);
    punto.y = y_ant + (dt/6.0)*(k1_y + 2*k2_y + 2*k3_y + k4_y);
    return punto;
}
```

Listing 4: Código del método RK4 y de las funciones del sistema de ecuaciones diferenciales 7 con los valores de las variables de la tabla 8. Elaboración propia.

En el código 4, se puedo observar la función calcular_punto con los parámetros de entrada

x_ant e y_ant, que representan las x e y obtenidas de RK4 de la iteración anterior, y devuelve el struct Punto obtenido de aplicar RK4. En las líneas 23-33 está el cálculo de RK4 explicado en el apartado 1.1.4 llamando a las funciones dx_dt y dy_dt, que representan las ecuaciones 7 con los valores de las variables de la tabla 8.

```
1  if(longitud > 0) {
2      FILE * temp = fopen("data.txt", "w");
3      for (int i=0; i < longitud; i++) {
4          fprintf(temp, "%f %f \n", x_0[i], x_t[i]); //Write the data to a temporary file
5      }
6      const char* gnuplotPath = "C:\\Program Files\\gnuplot\\bin";
7      char pathEnv[4096];
8      snprintf(pathEnv, sizeof(pathEnv), "PATH=%s;%s", gnuplotPath, getenv("PATH"));
9      putenv(pathEnv);
10
11     // Ahora puedes usar system("gnuplot") con "gnuplot" en el PATH.
12     system("gnuplot script1.gp");
13
14     fclose(temp);
```

Listing 5: Código para imprimir un gráfico a través de Gnuplot del mapeo de Poincaré en y=0 de los vectores x_0 y x_t. Elaboración propia.

A continuación, si los vectores x_0 y x_t tienen algún elemento, entonces se abre un archivo temporal data.txt y escribimos todos los elementos de estos vectores. Es necesario crear este archivo temporal porque se utiliza Gnuplot para la creación de gráficos. Gnuplot es un programa de código abierto que se utiliza comúnmente para crear gráficos y representaciones visuales de datos científicos.

Para utilizar Gnuplot debemos poner en la variable de entorno del sistema PATH la ruta de la carpeta bin del programa Gnuplot. Esto se hace en las líneas 6-9. A continuación, se ejecuta por sistema el comando "gnuplot script1.gp", que indica que Gnuplot ejecute el script indicado.

```
set terminal pngcairo enhanced font 'Verdana,12'
set output 'grafico.png'
set title 'Mapeo de Poincare en y=0'
set xlabel 'x(0)'
set ylabel 'x(T)-x(0)'
set yrange [-0.002:0.002]
plot 'data.txt' with lines
```

Listing 6: Código del script1.gp de Gnuplot para imprimir un gráfico a partir del archivo temporal data.txt generado en el código 5. El arcivo data.txt contiene los valores de los vectores x_0 y x_t. Elaboración propia.

En el código 6, se puede ver el script1.gp. En el cual, primero se elige la fuente del texto, después se elige el nombre y el formato del gráfico que se genera, se ponen títulos al gráfico y a los ejes, se pone un rango en el eje y, y finalmente se grafica el fichero temporal data.txt creado en el código 5 con líneas.

```
int count = 0; // Contador para los cruces por cero
double x_ciclos[longitud-2]; // Arreglo para almacenar valores de x en los cruces por cero

// Verificar los cruces por cero
for (int i = 0; i < longitud-1; i++) {
    if (x_t[i] * x_t[i + 1] < 0 && fabs(fabs(x_t[i]) - fabs(x_t[i + 1]))<1e-3) {
        x_ciclos[count] = x_0[i];
        printf("%f\n", x_ciclos[count]);
        count++;
    }
}

printf("Número de ciclos limite: %d\n", count);

free(x_0); free(x_t);
```

Listing 7: Código del recuento de ciclos límite calculando cuántos cruces existen en el eje y=0 con el vector x_t. Elaboración propia.

A continuación, como podemos ver en el código 7, se inicializa un contador a 0. Este contador representa el número total de ciclos límite. Después se crea un vector x_ciclos para almacenar

las x donde existe un ciclo límite, la y no hace falta, ya que siempre será 0. Posteriormente, como podemos ver entre las líneas 5-11 ,se realiza un bucle for en el vector x_t. En el caso de que haya un cruce en el eje $y = 0$ y la diferencia absoluta entre los valores absolutos de x_t[i] y x_t[i + 1] sea menor que 1×10^{-3}, esto se hace para asegurarse de que la magnitud de la diferencia entre los dos elementos adyacentes sea pequeña , entonces:

- Se almacena la x donde se produce el cruce en el eje $y = 0$, es decir, en el punto (x , 0.0) existe un ciclo límite.
- Se imprime por consola este punto.
- Se suma 1 a la varible count, que es el contador de ciclos límite.

Se ha escogido este criterio para contabilizar un ciclo límite porque la función no es exacta. Por lo tanto, si la función cruza el eje $y = 0$ quiere decir que hay un punto que $y = 0$, es decir, que $x(T) - x(0) = 0$. Entonces, podemos concluir que es un ciclo límite.

```
dt = 1e-6;
for(int i=0; i<count; ++i) {
    longitud = 0;
    capacidad = 10;
    double* X = (double*)malloc(capacidad * sizeof(double));
    double* Y = (double*)malloc(capacidad * sizeof(double));
    double t=0;
    int count = 0;
    struct Punto p;
    p = calcular_posicion(x_ciclos[i],0.0);
    X[longitud] = p.x;
    Y[longitud] = p.y;
    t = t + dt;
    ++longitud;
    while(t < 10.0 && count<2) {
        if (longitud == capacidad) {
            // Si el arreglo está lleno, aumenta la capacidad
            capacidad *= 2;
            X = (double*)realloc(X, capacidad * sizeof(double));
            Y = (double*)realloc(Y, capacidad * sizeof(double));
            if (X == NULL || Y == NULL) {
                fprintf(stderr, "Error al asignar memoria.\n");
                return 1;
            }
        }
        p = calcular_posicion(X[longitud-1], Y[longitud-1]);
        X[longitud] = p.x;
        Y[longitud] = p.y;
        t = t + dt;
        if(Y[longitud]*Y[longitud-1] < 0 && fabs(fabs(Y[longitud]) - fabs(Y[longitud + 1]))<1e-3) {
            count = count + 1;
        }
        ++longitud;
    }
    FILE * temp_ciclo = fopen("ciclos.txt", "w");
    for (int j=0; j < longitud; j++)
    {
        fprintf(temp_ciclo, "%f %f \n", X[j], Y[j]); //Write the data to a temporary file
    }
    char command[100];
    snprintf(command, sizeof(command),"set i=%d & gnuplot script3.gp", i+1);
    system(command);
    free(X); free(Y);
    fclose(temp_ciclo);
}
```

Listing 8: Código para representar los ciclos límite, con RK4 sin interpolación cuádratica inversa y con mapeo en y=0 ,con los puntos iniciales del vector x_ciclos creado en el código 7 a través de un script de Gnuplot. Elaboración propia.

Finalmente, una vez calculado los ciclos límites y sus puntos, tenemos que generar los gráficos para poder ver estos ciclos límites. Como se puede ver en el código 8, se pone $dt = 1 \times 10^{-6}$ y se hace un bucle for para iterar sobre cada ciclo límite encontrado. Por cada ciclo límite:

- Se crean los vectores X e Y para almacenar los puntos de los ciclos límites con tamaño capacidad, en este caso es de 10 elementos. Esto se hace para ahorrar memoria, ya que no sabemos que tamaño tendrán estos vectores porque dependen del número de puntos que contiene el ciclo límite.
- Se inicializan las variables t y count a 0. Representan el tiempo y el contador de cruces en el eje $y = 0$, respectivamente.
- Se llama a la función calcular_posicion con la x del ciclo límite y 0.0, ya que el primer punto del ciclo límite estará sobre el eje $y = 0$. Esta función devuelve el punto obtenido de aplicar RK4 y se almecenan en los vectores X e Y.
- Se inicia el bucle while con la condición que finalice este bucle en el caso que $t < 10,0$ y $count < 2$. Dentro del while, primero se mira si los vectores están llenos y se amplía el tamaño asignando un tamaño de $capacidad^2$ a los dos vectores. Después se llama a la función calcular_posicion con las puntos anteriores de los vectores X e Y. Se asigna este punto obtenido a los vectores X e Y y se suma dt a t, en este caso $dt = 1 \times 10^{-6}$. Posteriormente, si Y[longitud]*Y[longitud-1]<0 y la diferencia absoluta entre los valores absolutos de Y[i] y Y[i - 1] sea menor que 1×10^{-3}, es decir, se produce un cruce en el eje $y = 0$, se incrementa en 1 la variable count.
- A continuación, se crea el fichero temporal ciclos.txt y escribimos todos los elementos de los vectores X e Y en este archivo. Y por último, se ejecuta por sistema el comando "gnuplot script3.gp", que indica que Gnuplot ejecute el script indicado. También se asigna a la variable de entorno i el número de ciclo límite. Finalmente, se libera el espacio de los vectores X e Y y se cierra el fichero temporal.

```
set terminal pngcairo enhanced font 'Verdana,12'

i = int(system("echo %i%"))

set output sprintf('grafico%d.png', i)
set title sprintf('Grafico ciclo %d', i)
set xlabel 'Eje X'
set ylabel 'Eje Y'
plot 'ciclos.txt' with lines
```

Listing 9: Código del script3.gp de Gnuplot para imprimir un gráfico a partir de los archivos temporales ciclos.txt generados en el código 8

En el código 9, se puede ver el archivo script3.gp. En el cual, primero se elige la fuente del texto, después se asigna a la variable i el valor de la variable de entorno i ,se elige el nombre y el formato del gráfico obtenido, se ponen títulos al gráfico y a los ejes y finalmente se grafica el fichero temporal ciclos.txt creado en la imagen 8 con líneas.

Con el script1.gp se genera el siguiente gráfico:

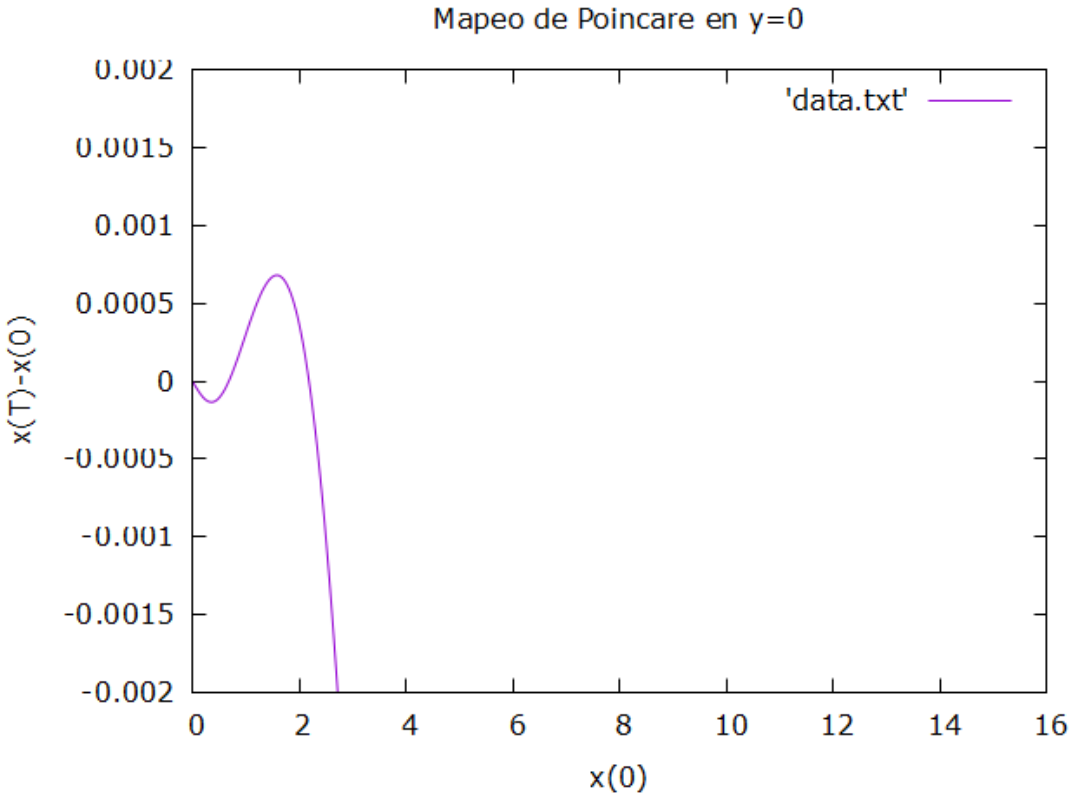

Figura 2: Mapeo de Poincaré en y=0 para el sistema 7 con los valores de las variables de la tabla 8 y con los puntos iniciales [0,16] con paso 0.01. Gráfico generado del script1.gp.

En la imagen 2 se puede ver el mapeo de Poincare en el eje $y = 0$. El eje x representa $x(0)$, es decir, donde se produce el primer cruce en el eje $y = 0$ de la trayectoria calculada en RK4, y el eje y representa $x(T) - x(0)$, es decir, la diferencia de las x entre el primer y el tercer cruce en el eje $y = 0$ de la trayectoria calculada en RK4. Como se observa hay varios cruces en el eje $y = 0$ del gráfico 2, por lo tanto hay varios ciclos límites:

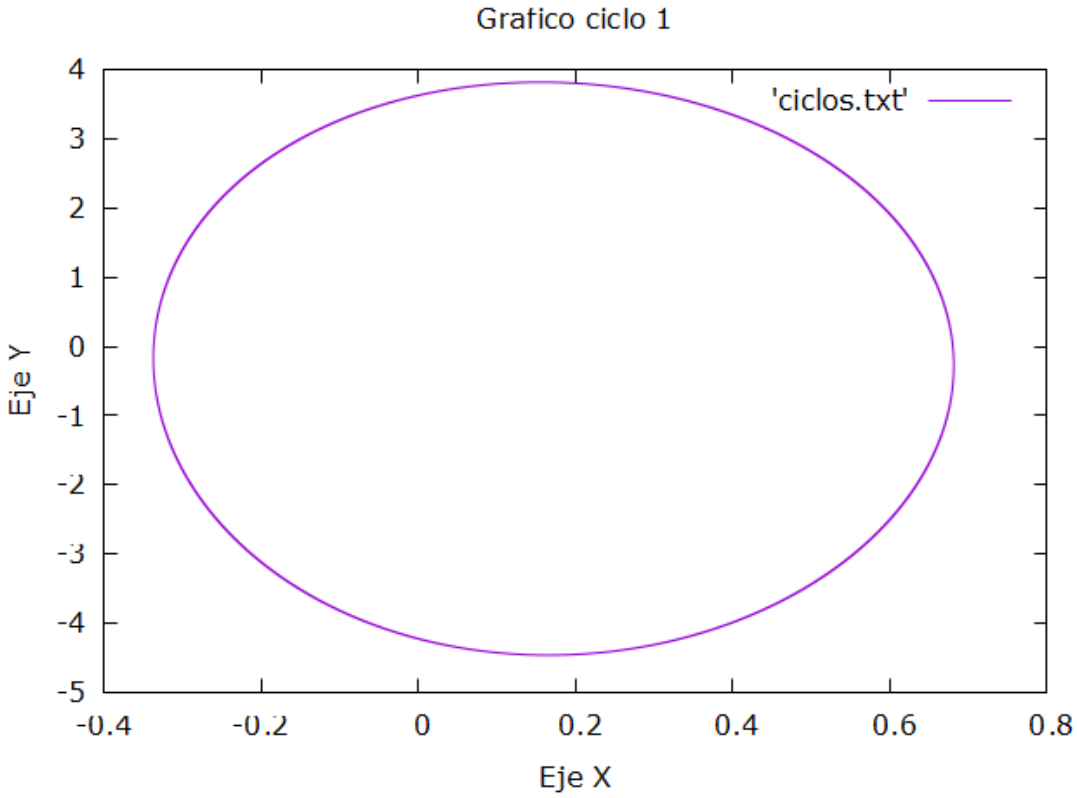

Figura 3: Ciclo 1 para el sistema 7 con los valores de las variables de la tabla 8 y con los puntos iniciales [0,16] con paso 0.01. Gráfico generado del script3.gp.

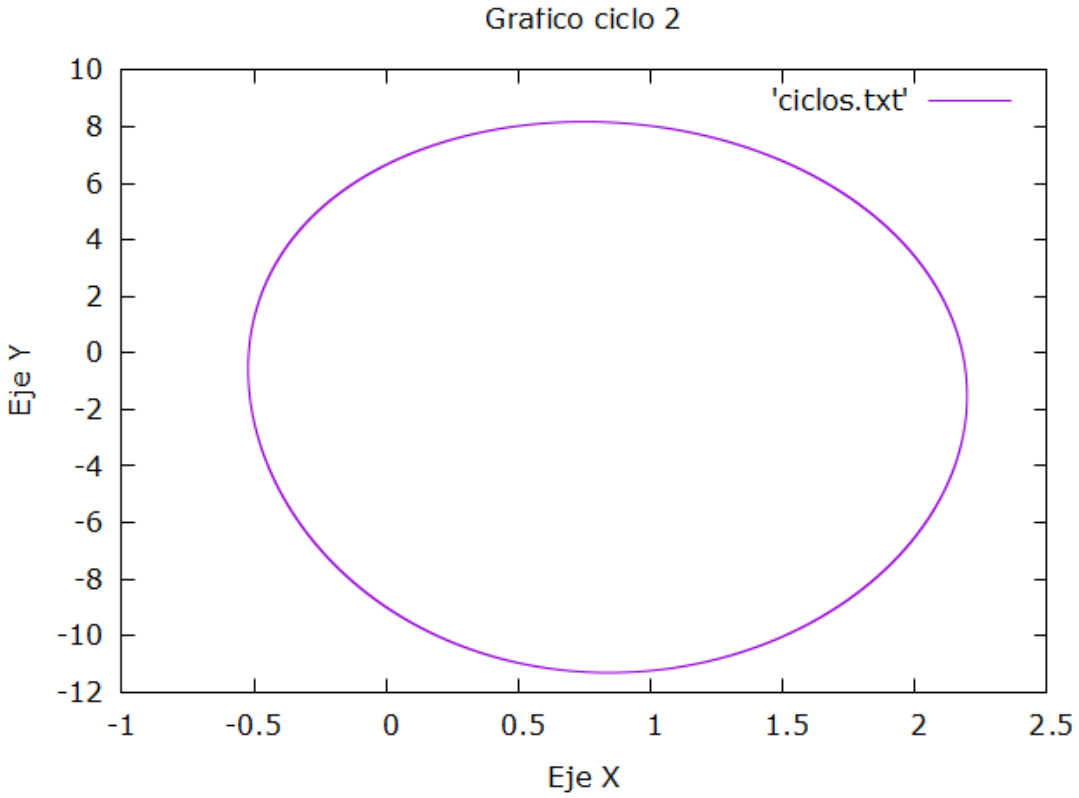

Figura 4: Ciclo 3 para el sistema 7 con los valores de las variables de la tabla 8 y con los puntos iniciales [0,16] con paso 0.01. Gráfico generado del script3.gp.

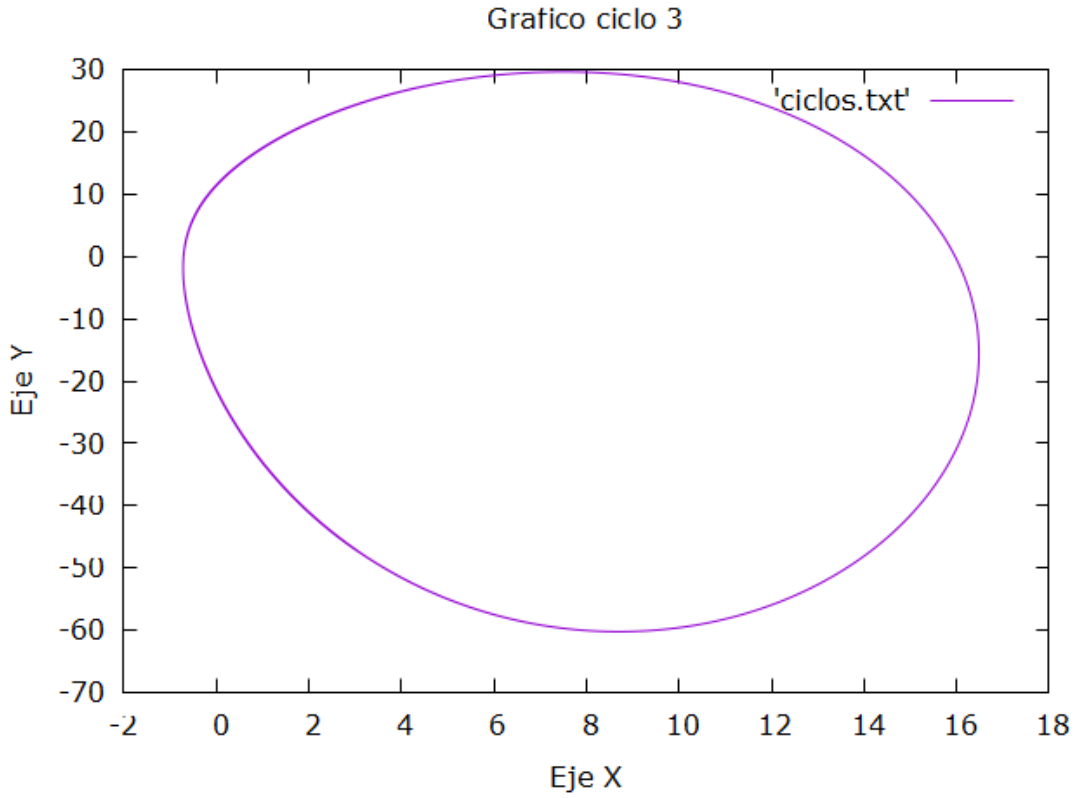

Figura 5: Ciclo 3 para el sistema 7 con los valores de las variables de la tabla 8 y con los puntos iniciales [0,16] con paso 0.01. Gráfico generado del script3.gp.

8.1.2. Algoritmo con varios juegos de parámetros

A continuación, se explica el algoritmo secuencial realizado para calcular el número de ciclos límite para varios juegos de parámetro de entrada. Este algoritmo corresponde al archvivo tfg_params.c. Estos paráametro de entrada son los de la tabla 8. Se han fijado los valores de la tabla 8 para las variables b_1, c_1, α_1, β_1, a_2, b_2, c_2, α_2 y β_2, excepto la variable a_1. El valor de la variable a_1, es el siguiente:

$$a_1 = -10,0 + j, \quad 0 \leq j < \text{NÚMERO DE JUEGOS DE PARÁMETROS} \tag{25}$$

Por ejemplo, para replicar el resultado que se obtiene en el algoritmo del apartado 8.1.1, el número de juegos de parámetros es 1 y, por lo tanto, el valor de a_1 será -10,0 y obtendremos 3 ciclos límite.

Primero se ha creado una variable global JUEGO_PARAMETROS donde podemos indicar el número de juegos de parámetros que queremos para conocer cuantos ciclos límites en cada trayectoria calculada. En total se calcularán $1600 *$ JUEGO_PARAMETROS trayectorias.

```
struct Ciclo c;
b2= 2.2;
c2= 0.7;
alpha2= -72.7778;
beta2= 0.0015;
int contador[1000] = {0};

for(int j=0; j<JUEGO_PARAMETROS; ++j) {
    double* x_0 = (double*)malloc(capacidad * sizeof(double));
    double* x_t = (double*)malloc(capacidad * sizeof(double));
    longitud = 0;
    for(int i=0;i<size;++i) {
        if (longitud == capacidad) {
            // Si el arreglo está lleno, aumenta la capacidad
            capacidad *= 2;
            x_0 = (double*)realloc(x_0, capacidad * sizeof(double));
            x_t = (double*)realloc(x_t, capacidad * sizeof(double));
            if (x_0 == NULL || x_t == NULL) {
                fprintf(stderr, "Error al asignar memoria.\n");
                return 1;
            }
        }
        c = runge4_ciclos(init_pos[i]);
        if(c.x != -1000.0 && c.y != 1000.0) {
            x_0[longitud] = c.x;
            x_t[longitud] = c.y;
            ++longitud;
        }
    }
    if(longitud > 0) {
        // Verificar los cruces por cero
        int count = 0;
        for (int i = 0; i < longitud - 1; i++) {
            if (x_t[i] * x_t[i + 1] < 0 && fabs(fabs(x_t[i]) - fabs(x_t[i + 1]))<1e-3) {
                count++;
            }
        }
        contador[count] = contador[count] + 1;
    }
    else {
        contador[0] = contador[0] + 1;
    }
    free(x_0); free(x_t);
    capacidad = 10;
}
```

Listing 10: Aplicación de RK4 con interpolación cuádratica inversa con mapeo en y = 0 en cada posición del vector init_pos para cada juego de parámetros diferente y el recuento de ciclos límite calculando cuántos cruces existen en el eje y=0 con el vector x_t de cada juego de parámetros. Elaboración propia.

Primero se fijan los valores de las variables b_2, c_2, α_2 y β_2 con los valores de la tabla 8, como se puede ver en las líneas 2-5 del código 10. A continuación, se crea el vector contador de 1000 elementos a 0. Cada posición del vector contador representa:

$$\text{contador[i]} = \text{Número de juegos de parámetros con i ciclos límite}, \quad 0 \leq i < 1000 \tag{26}$$

Después, en el for de la línea 8-45, para cada juego de parámetros:

- Se da valor a la variable a_1 con la ecuación 25.
- Se crean los vectores x_0 ($x(0)$) y x_t ($x(T) - x(0)$) con tamaño capacidad, en este caso es de 10 elementos. Esto se hace para ahorrar memoria, ya que no sabemos qué tamaño tendrán estos vectores porque dependen del número de trayectorias que tengan dos cruces en el eje $y = 0$. También, se le da valor 0 a la variable longitud, que sirve para recorrer los dos vectores.
- Se comprueba si los vectores están llenos y, en el caso que sea así, se amplía el tamaño asignando un tamaño de $capacidad^2$ a los dos vectores. Después, se aplica la función runge4_ciclos con el párametro de entrada de un punto inicial, anteriormente calculado en el código 1. Esta función, que calcula RK4 con interpolación inversa cuadrática y con mapeo en $y = 0$, devuelve un struct Ciclo, el cual tiene como atributo x e y. Este struct representa un posible ciclo límite, por lo tanto el atributo x es $x(0)$ y el atributo y es $x(T) - x(0)$. Estos valores los guardamos en los dos vectores en el caso que sean diferente a -1000 porque la función runge4_ciclos devuelve un struct Ciclo con valores -1000 en el caso que no haya dos cruces en el eje $y = 0$.
- Después, se mira el valor de longitud:
  - Si $longitud > 0$, significa que hay posibles ciclos límite con esos valores de las variables. Por lo tanto, se inicializa un contador a 0. Este contador representa el número total de ciclos límite. Posteriormente, como podemos ver entre las líneas 33-37 ,se realiza un bucle for en el vector x_t. En el caso que haya un cruce en el eje $y = 0$ y la diferencia entre los dos puntos donde se produce el cruce sea menor que 1×10^{-3}, significa que hay un ciclo límite y se suma 1 a la variable count. Después de recorrer todo el vector x_t, se suma 1 al contador[count]. Esto quiere decir que se incrementa en 1 el número

de juegos de parámetros con count ciclos límite.

- Si $longitud = 0$, significa que no hay ciclos límite cons esos valores de las variables. Por lo tanto, se suma 1 al número de trayectorias con 0 ciclos límite.

- Finalmente, se vacían los vectores x_0 y x_t y le damos el valor 10 a la variable capacidad para el siguiente juego de valores de los parámetros.

```
1  long long tiempoFin = obtenerTiempoEnMilisegundos();
2  long long duracion = tiempoFin - tiempoInicio;
3  printf("El programa tardo %lld milisegundos en ejecutarse.\n", duracion);
4  int count=0;
5  for(int i=0; i<1000; ++i) {
6      if(contador[i]!= 0){
7          count = count + contador[i];
8          printf("Hay %d trayectorias con %d ciclos.\n", contador[i],i);
9      }
10 }
11 printf("En total hay: %d.\n", count);
12 return 0;
```

Listing 11: Impresión por pantalla del número de ciclos límite. Elaboración propia.

En el código 11, primero se muestra el tiempo de ejecución del programa. A continuación, en el bucle for de la línea 5-10, para cada posición i del vector contador, que representa el número de juegos de parámetros que tienen i ciclos límite:

- Si $contador[i] \neq 0$, es decir, hay algun juego de parámetros con i ciclos límite, primero se suma $contador[i]$ a la variable count. Esta variable count, al final del bucle, debe tener el mismo valor que la variable global JUEGO_PARAMETROS. Y a continuación, se imprime por pantalla el número de juegos de parámetros que tienen i ciclos límite.

Finalmente, se imprime por pantalla el valor de count, para comprobar que coincida con el valor de la variable global JUEGO_PARAMETROS.

9. Análisis de rendimiento y precisión de los algoritmos secuenciales

En este apartado se compara los tiempos de ejecución entre las diferentes versiones secuenciales que se han programado, también se compara la precisión de las diferentes versiones de los algoritmos con otro problema donde la solución es conocida.

9.1. Análisis de rendimiento (tiempo de ejecución)

Todas las versiones de los algoritmos secuenciales se ejecutarán en una CPU AMD Ryzen 5 5600X 4.6Ghz 6-Core y una RAM 6Gb (2x8Gb) Corsair Vengeance LPX 3200Mhz CL16.

9.1.1. Análisis de rendimiento para un juego de parámetros

En este subapartado, se comparan los tiempos de ejecución del algoritmo del apartado 8.1.1, pero sin la visualizacin de los gráficos y de los ciclos límites obtenidos. En esta comparación tenemos 3 versiones:

- Versión 1: Código en Python del algoritmo sin interpolación cuadrática inversa con $dt = 1 \times 10^{-6}$. Corresponde al archivo tfg_vers1.py.
- Versión 2: Código en C del algoritmo sin interpolación cuadrática inversa con $dt = 1 \times 10^{-6}$. Corresponde al archivo tfg_sec_no_inter.c.
- Versión 3: Código en C del algoritmo con interpolación cuadrática inversa con $dt = 1 \times 10^{-3}$. Corresponde al archivo tfg_sec_inter.c.

En las tres versiones se calculan un total de 1600 trayectorias. Los valores de los parámetros son los de la tabla 8 para las 3 versiones.

Las diferentes versiones tienen dt diferentes para obtener los mismos resultados (3 ciclos limites) y no perder precisión. Los tiempos de ejecución obtenidos son:

Versión	Tiempo (ms)
Versión 1	8436956
Versión 2	61229
Versión 3	62

Tabla 9: Tiempos de ejecución de las 3 versiones para el algoritmo secuencial del apartado 8.1.1. Elaboración propia.

En este apartado, el speed-up de la versión 2 sobre la versión 1 se define:

$$\text{Speed-up} = \frac{\text{tiempo ejecución Versión 1}}{\text{tiempo ejecución Versión 2}} = \frac{8436956}{61229} \approx 137 \tag{27}$$

El speed-up nos indica que el algoritmo de la versión 2 es 137,7934639 veces más rápido que el de la versión 1. Esto es debido a que C es un lenguaje compilado y altamente optimizado, mientras que Python es un lenguaje interpretado que tiene una sobrecarga adicional debido a la interpretación en tiempo de ejecución y la gestión dinámica de tipos. El speed-up de la versión 3 sobre la versión 2 se define:

$$\text{Speed-up} = \frac{\text{tiempo ejecución Versión 2}}{\text{tiempo ejecución Versión 3}} = \frac{61229}{62} \approx 987 \tag{28}$$

El speed-up nos indica que el algoritmo de la versión 3 es 987,5645161 veces más rápido que el de la versión 2. Esto es debido a que el dt de la versión 2 es 1000 veces mayor que el dt de la versión 3 pero como en la versión 3 se aplica interpolación inversa cuadrática, entonces no se pierde precisión y se obtiene el mismo resultado (3 ciclos límite). Por lo tanto, el algoritmo escogido es el de la versión 3 por el tiempo de ejecución sin perder precisión en los resultados.

9.1.2. Análisis de rendimiento para varios juegos de parámetros

En este subapartado, se comparan los tiempos de ejecución del algoritmo del apartado 8.1.2, pero sin la visualización de los gráficos y de los ciclos límites obtenidos. En esta comparación tenemos 2 versiones:

- Versión 1: Código en C del algoritmo sin interpolación cuadrática inversa.

- Versión 2: Código en C del algoritmo con interpolación cuadrática inversa.

Para obtener los mismos resultados en las 2 versiones y no perder precisión, la versión 1 tiene un $dt = 1 \times 10^{-6}$ y la versión 2 un $dt = 1 \times 10^{-3}$. Estas dos versiones, en total calculan $1600 * \text{JUEGO\_PARAMETROS}$ trayectorias. Los tiempos obtenidos son:

Número juegos de parámetros	Versión 1 Tiempo (ms)	Versión 2 Tiempo (ms)	Speed-up
1	61229	62	987
10	878800	901	975
100	77448824	76285	1015

Tabla 10: Tiempos de ejecución de las 2 versiones para el algoritmo secuencial del apartado 8.1.2. Elaboración propia.

En este apartado, el speed-up de de la versión 2 sobre la versión 1 se define:

$$\text{Speed-up} = \frac{\text{tiempo ejecución Versión 1}}{\text{tiempo ejecución Versión 2}} \tag{29}$$

El speed-up nos indica que el algoritmo de la versión 2 es alrededor de 1000 veces más rápido que la versión 1. Esto es debido a que el dt de la versión 2 es 1000 veces mayor que el dt de la versión 1, pero como en la versión 2 se aplica interpolación inversa cuadrática, entonces no se pierde precisión y se obtiene el mismo resultado. Por lo tanto, el algoritmo escogido es el de la versión 2 por el tiempo de ejecución sin perder precisión en los resultados.

9.2. Análisis de precisión

En este subapartado se analiza la precisión del algoritmo de RK4 con interpolación inversa cuadrática y mapeo en $y = 0$. Para hacer la comparación de la precisión se ha escogido un problema donde el resultado es conocido. El problema es el siguiente:

$$\begin{aligned} \frac{dx}{dt} &= -y \\ \frac{dy}{dt} &= x \end{aligned} \tag{30}$$

$$\begin{aligned} x(0) &= 5,0 \\ y(0) &= 0,0 \end{aligned} \tag{31}$$

Este problema es conocido como un sistema de Oscilador armónico. El ciclo límite obtenido es el siguiente:

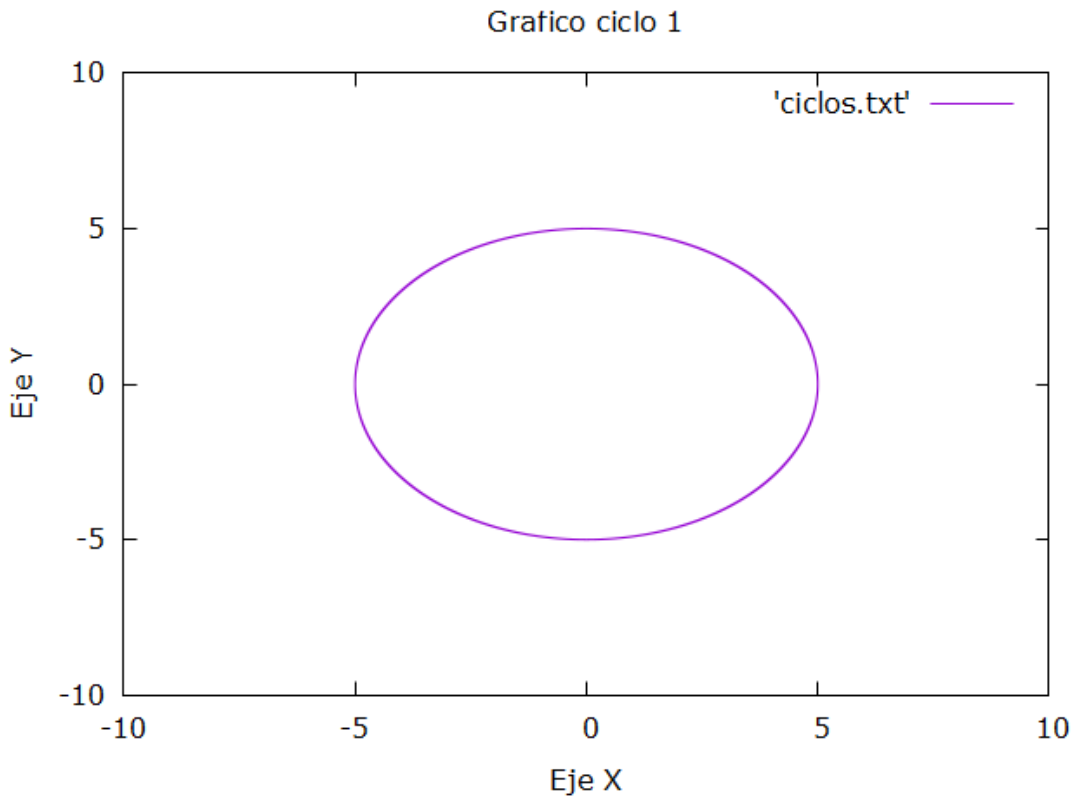

Figura 6: Ciclo límite obtenido en el sistema de oscilador armónico del problema 30. Generado por Gnuplot en el archivo error_double.c.

Como se observa en la imagen, existe un ciclo límite con origen en el punto (5.0, 0.0). Para hacer las comparaciones, se calcula el ciclo límite obtenido con las dos versiones de RK4 con diferentes dt y las comparamos con el ciclo límite teórico que tiene como punto de origen el punto (5.0, 0.0). Las dos versiones de RK4 son:

- Versión 1: Código en C del algoritmo sin interpolación cuadrática inversa. Corresponde al archivo error_no_inter.c.
- Versión 2: Código en C del algoritmo con interpolación cuadrática inversa. Corresponde al archivo error_inter.c

El error absoluto dx se define de la siguiente manera:

$$\text{dx} = |x\_teorica - x\_practica| \tag{32}$$

En la ecuación del error absoluto 32, la x_teorica es 5.0 y la x_practica es la obtenida al aplicar las dos versiones del algoritmo. Los resultados son los siguientes:

dt	dx Versión 1	dx Versión 2
1×10^{0}	1,268	0,022
1×10^{-1}	0,001	$3,812 \times 10^{-5}$
1×10^{-2}	0,000	$1,688 \times 10^{-9}$
1×10^{-3}	$1,659 \times 10^{-6}$	$6,195 \times 10^{-14}$
1×10^{-4}	$5,397 \times 10^{-10}$	$1,821 \times 10^{-17}$
1×10^{-5}	$5,506 \times 10^{-11}$	$9,064 \times 10^{-17}$
1×10^{-6}	$1,201 \times 10^{-12}$	$5,026 \times 10^{-16}$

Tabla 11: Errores absolutos para diferentes dt de las versiones de los algoritmos secuenciales de RK4 con interpolación y sin interpolación con mapeo en $y = 0$ del sistema de Oscilador armónico 30. Elaboración propia.

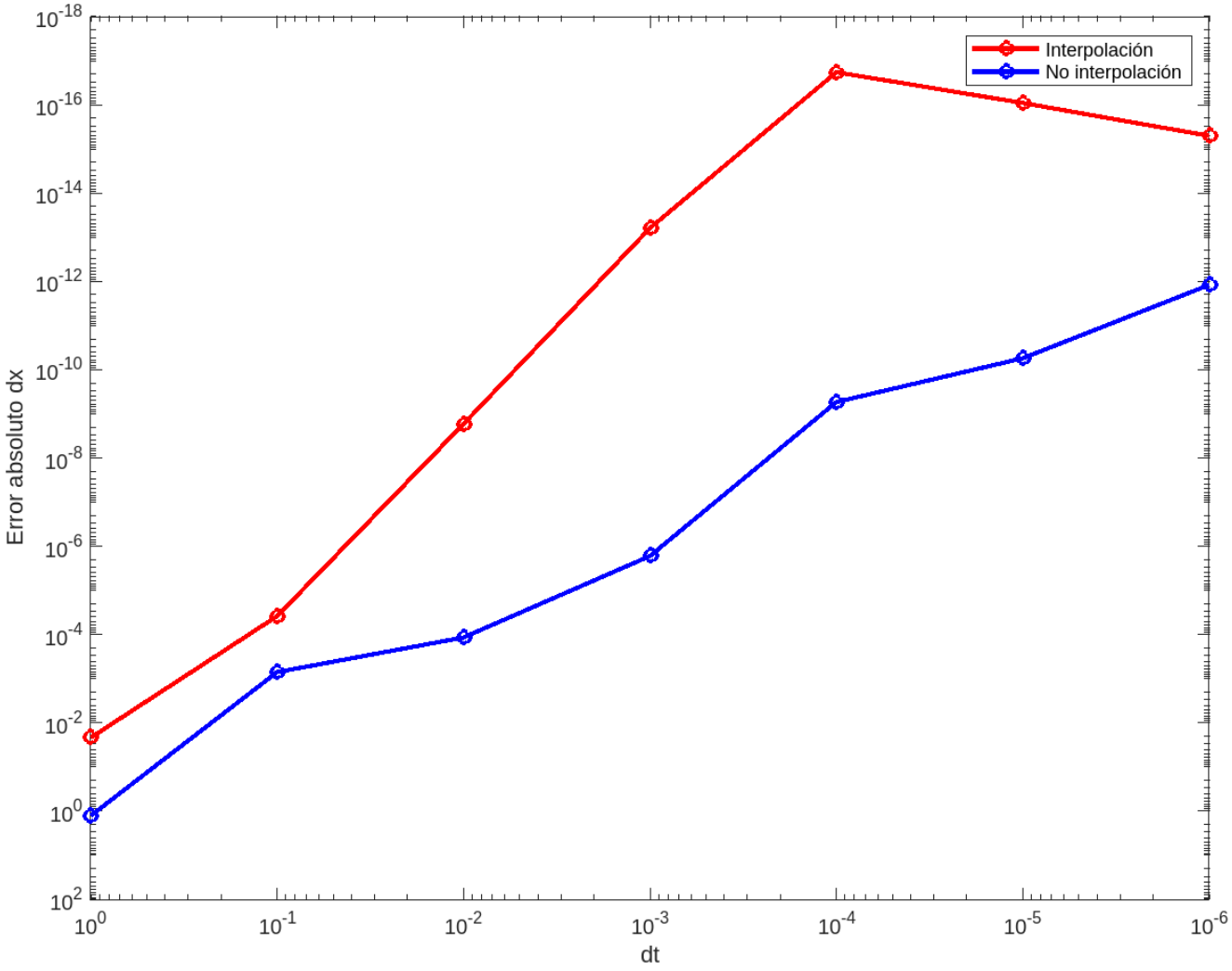

Figura 7: Gráfico de los errores absolutos para diferentes dt de RK4 con interpolación y sin interpolación con mapeo en $y = 0$ del sistema de Oscilador armónico 30. Gráfico obtenido de un script de MATLAB.

Como se puede observar en la tabla 11 y en la imagen 7, en los errores de la versión sin interpolación se debería observar un error de orden de dt, es decir, una dependencia lineal, pero se observa una dependencia cuadrática, ya que el error $\approx dt^2$. En cambio, en los errores de la versión con interpolación se debería observar un error del orden del método de la interpolación, pero como en nuestro caso se observa una dependencia cuadrática en la versión sin interpolación, pues se observa con una dependencia cuártica, ya que el error $\approx dt^4$.

En la versión 2 se obtienen errores absolutos más pequeños que la versión 1, incluso con dt más

grandes. Por ejemplo, el error de la versión 2 con $dt = 1 \times 10^{-3}$ es menor que el error de la versión 1 con $dt = 1 \times 10^{-6}$, por lo tanto, nos quedamos con la versión 2 con $dt = 1 \times 10^{-3}$, ya que se obtiene mejores tiempos de ejecución, como se demuestra en el apartado 9.1, incluso con mayor precisión que la versión sin interpolación con dt menor.

10. Algoritmo paralelo en CUDA

En este apartado se describira el algoritmo paralelo en CUDA. Este algoritmo está basado en el algoritmo secuencial explicado en el apartado 8.1.2 y los principales conceptos de CUDA que se utilizan. Corresponde al archivo tfg_cuda.cu.

10.1. Código del host (CPU)

```
1  #ifndef size
2  #define size 1601
3  #endif
4
5  struct Punto {
6      double x;
7      double y;
8  };
9
10 struct Ciclo {
11     double x;
12     double y;
13 };
14
15  // Tamaño del paso del tiempo
16 __device__ double dt = 1e-3;
17 __device__ double b2 = 2.2;
18 __device__ double a1,b1,c1,alpha1,beta1;
19 __device__ double a2,c2,alpha2,beta2;
20 __device__ int JUEGO_PARAMETROS = 100000;
21 double a2_h,b2_h,c2_h,alpha2_h,beta2_h;
```

Listing 12: Inicialización de las variables globales necesarias para el algoritmo. Elaboración propia.

Primero, como se puede ver en el código 12, se inicializa el valor 1601 a la variable size porque los vectores x_0 ($x(0)$) y x_t ($x(T) - x(0)$) se crean en GPU y no se pueden crear de manera dinámica como en el algoritmo secuencial. Por lo tanto, se crean con el tamaño máximo que podrían tener estos vectores. A partir de la línea 16, se crean las variables globales que son necesarias para el algoritmo. Las variables que tienen la directiva __device__ son variables globales que residen en la memoria global de la GPU y que pueden ser accedidas por los hilos (threads) ejecutados en el dispositivo.

```
clock_t inicio, fin;
double tiempo;
inicio = clock();

double start = 0; // Valor inicial del rango
double stop = 16; // Valor final del rango
double step = 0.01;  // Tamaño del paso entre números

// Crea un arreglo dinámico para almacenar los números
double* h_init_pos = (double*)malloc(size * sizeof(double));
int* h_ciclos = (int*)malloc(1000 * sizeof(int));

// Llena el arreglo con los números en el rango
for (int i = 0; i < size; i++) {
    h_init_pos[i] = start + i * step;
}
```

Listing 13: Creación de los vectores de puntos iniciales, en cada posición inicial se aplicará RK4 con interpolación cuádratica inversa con mapeo en y=0. Elaboración propia.

A continuación, como se observa en el código 13, se crean los vectores h_init_pos y el vector h_ciclos. Estos vectores son los mismos que init_pos y contador, respectivamente, del algoritmo del apartado 8.1.2.

```
double *d_init_pos;
int *d_ciclos;

cudaMalloc((void**)&d_ciclos, 1000 * sizeof(int));
cudaMalloc((void**)&d_init_pos, size * sizeof(double));
cudaMemcpy(d_init_pos, h_init_pos, size * sizeof(double),cudaMemcpyHostToDevice);

int threadsPerBlock = 256;
int blocksPerGrid = 640;

gpu_runge4<<<blocksPerGrid, threadsPerBlock>>>(d_init_pos ,d_ciclos);

cudaMemcpy(h_ciclos, d_ciclos, 1000 * sizeof(int), cudaMemcpyDeviceToHost);
```

Listing 14: Inicialización de las variables que se utilizan en la GPU y llamada a la función kernel. Elaboración propia.

En el código 14, primero se asigna memoria en el dispositivo (GPU). Esto se hace en las líneas 4 y 5. CudaMalloc es una función que se utiliza para asignar memoria en el dispositivo (GPU). Esta función tiene 2 parámetros:

- Un puntero a un puntero que apunta a la ubicación de memoria en el dispositivo (GPU) donde se almacenará el espacio de memoria asignado. En este caso, (void**)&d_ciclos y (void**)&d_init_pos.
- El tamaño en bytes de la memoria que se asignará en el dispositivo. En este caso, $1000 * sizeof(int)$ y $size * sizeof(double)$.

Estas dos líneas de código se deben ejecutar porque se debe reservar memoria en GPU para poder utilizar estos dos vectores en GPU. A continuación, en la línea 6, se utiliza la función cudaMemcpy. Esta función se utiliza en CUDA para copiar datos entre el host (CPU) y el dispositivo (GPU), o entre diferentes ubicaciones de memoria en el dispositivo. En este caso, se están copiando size elementos de tipo double desde el buffer h_init_pos en el host(CPU) al buffer d_init_pos en el dispositivo(GPU). Esto se debe hacer para pasar los datos del vector h_init_pos, creado e inicializado en el código 13, al vector que se utilizará en GPU.

```
1  int count=0;
2  for(int i=0; i<1000; ++i) {
3      if(h_ciclos[i]!= 0){
4          count = count + h_ciclos[i];
5          printf("Hay %d trayectorias con %d ciclos.\n", h_ciclos[i],i);
6      }
7  }
8  printf("En total hay: %d.\n", count);
9
10 fin = clock();
11 tiempo = (double)(fin - inicio) / CLOCKS_PER_SEC;
12 printf("Tiempo transcurrido: %f milisegundos\n", tiempo*1000);
```

Listing 15: Impresión por pantalla del número de ciclos límite. Elaboración propia.

Finalmente, como se puede observar en el código 15, se imprime por pantalla cuantos juegos de parámetros con i ciclos límite hay. También se imprime cuantos juegos de parámetros se han calculado y el tiempo de ejecución. Esta parte es exactamente igual que el algoritmo secuencial del apartado 8.1.2.

10.2. Modelo de programación en CUDA

En este aparatado se presentan los conceptos principales detrás del modelo de programación CUDA. [16]

10.2.1. Kernels

CUDA C extiende C al permitir al programador definir funciones de C, llamadas kernels, que, cuando se llaman, se ejecutan N veces en paralelo por N subprocesos CUDA diferentes, en lugar de solo una vez como las funciones normales de C. Un kernel se define usando la directiva de declaración __global__ y el número de subprocesos CUDA que ejecutan ese kernel para una llamada de kernel. En esta llamada se especifica usando una nueva sintaxis de configuración de ejecución «<...»>. Cada subproceso que ejecuta el kernel recibe un ID de subproceso única a la que se puede acceder dentro del kernel a través de variables integradas.

10.2.2. Jerarquía de threads

En este apartado se definen los siguientes conceptos:

- **Grid**: es una colección de threads. Los threads en un grid ejecutan una función del kernel y se dividen thread blocks.
- **Thread block**: es un grupo de threads que se ejecutan en el mismo multiprocesador. Los threads dentro de un thread block tienen acceso a la memoria compartida y se pueden sincronizar explícitamente.
- **Función del kernel**: es una subrutina implícitamente paralela que se ejecuta bajo el modelo de memoria y ejecución CUDA para cada thread en un grid.

ThreadIdx es un vector de 3 componentes, de modo que los threads pueden ser identificados utilizando un índice (unidimensional, bidimensional o tridimensional) de thread formando un thread block (unidimensional, bidimensional o tridimensional).

El índice de un thread y su thread ID se relacionan entre sí de forma sencilla:

- **Bloque unidimensional**: son el mismo.
- **Bloque bidimensional**: Para un bloque bidimensional de tamaño (Dx,Dy), el thread ID de un thread con índice (x,y) es $(x + y * Dx)$.

- **Bloque tridimensional**: Para un bloque tridimensional de tamaño ((Dx, Dy, Dz), el thread ID de un thread con índice (x,y,z) es $(x + y * Dx + z * Dx * Dy)$.

Existe un límite en la cantidad de threads por bloque, ya que se espera que todos los threads de un bloque residan en el mismo núcleo multiprocesador y deben compartir los recursos de memoria limitados de ese núcleo. En las GPU actuales, un bloque de subprocesos puede contener hasta 1024 subprocesos.

Los bloques se organizan en un grid unidimensional, bidimensional o tridimensional de threads blocks. El número de threads blocks en un grid generalmente viene dictado por el tamaño de los datos que se procesan, que generalmente excede el número de procesadores en el sistema.

El número de threads block y el número de blocks por grid especificados en la sintaxis «<...»>pueden ser de tipo int o dim3. Cada block dentro del grid se puede identificar mediante un índice único unidimensional, bidimensional o tridimensional accesible dentro del kernel a través de la variable blockIdx incorporada. Se puede acceder a la dimensión del bloque de threads dentro del kernel a través de la variable incorporada blockDim.

Los threads blocks deben ejecutarse de forma independiente: debe ser posible ejecutarlos en cualquier orden, en paralelo o en serie. Este requisito de independencia permite programar threads blocks en cualquier orden en cualquier número de núcleos, lo que permite a los programadores escribir código que se escala con el número de núcleos. En nuestro algoritmo, cada ejecución con un juego de parámetros diferente es totalmente independiente de otra ejecución, por lo tanto, se puede ejecutar en cualquier orden.

En nuestro caso, tanto los bloques por grid como los threads blocks son variables unidimensionales, como se puede ver en las líneas 8-9 del código 14. A continuación, en la línea 11, se hace la llamada a la función kernel gpu_runge4 especificando el número de bloques por grid y el número de threads por block. También se especifican los parámetros que se le pasa a la función kernel, previamente inicializados y con memoria asignada en la GPU. En este caso, se ha puesto threadsPerBlock = 256 y blocksPerGrid = 640 para tener 163840 threads y poder ejecutar el programa con 100000 juegos de parámetros.

A continuación, se copian los datos del vector d_ciclos, donde está los resultados obtenidos en GPU, al vector h_ciclos que está en el host (CPU). Esta acción se hace con la llamada a la función cudaMemcpy. La llamda a la función cudaMemcpy(void* **dst**, const void* **src**, size_t

count, cudaMemcpyKind **kind**) tiene los siguientes parámetros:

- **dst**: Es un puntero al destino de la copia. Puede ser un puntero al espacio de memoria en el dispositivo (para una copia de host a dispositivo) o en el host (para una copia de dispositivo a host). En este caso es una copia de GPU a CPU.
- **src**: Es un puntero al origen de la copia. Puede ser un puntero al espacio de memoria en el dispositivo o en el host, en este caso es en la GPU.
- **count**: Es el número de bytes que se copiarán.
- **kind**: Especifica la dirección de la copia y los principales valores que puede tomar son:
  - **cudaMemcpyHostToDevice**: La copia se realiza desde el host al dispositivo.
  - **cudaMemcpyDeviceToHost**: La copia se realiza desde el dispositivo al host.
  - **cudaMemcpyDeviceToDevice**: La copia se realiza de un espacio de memoria en el dispositivo a otro espacio de memoria en el dispositivo.

  Por lo tanto, en este caso será cudaMemcpyDeviceToHost.

10.3. Código del device (GPU)

Por lo tanto, tal y como está definido el algoritmo, cada thread ejecutará un juego de parámetros, es decir, 1600 trayectorias.

```
1  __global__ void gpu_runge4(double *init_pos, int *ciclos) {
2      int index = blockIdx.x * blockDim.x + threadIdx.x;
3      int longitud = 0;
4      if(index < JUEGO_PARAMETROS) {
5          struct Ciclo c;
6          double a2_var;
7          a2_var = -10.0 + index;
8          double x_0[size];
9          double x_t[size];
10         unsigned int i = 0;
11         while (i<size) {
12             c = runge4(init_pos[i], a2_var);
13             if(c.x != -1000.0 && c.y != 1000.0) {
14                 x_0[longitud] = c.x;
15                 x_t[longitud] = c.y;
16                 ++longitud;
17             }
18             ++i;
19         }
20         if(longitud > 0) {
21             int count = 0;
22             for (int i = 0; i < longitud - 1; i++) {
23                 if (x_t[i] * x_t[i + 1] < 0 && fabs(fabs(x_t[i]) - fabs(x_t[i + 1])) < 1e-3) {
24                     count++;
25                 }
26             }
27             atomicAdd(&ciclos[count], 1);
28         }
29         else {
30             atomicAdd(&ciclos[0], 1);
31         }
32     }
33 }
```

Listing 16: Función del kernel. En la función se crea el juego de parámetros actual y se aplica RK4 con interpolación inversa cuadrática con mapeo en $y = 0$ en cada posición del vector init_pos y se calcula el número de ciclos límite en cada juego de parámetros.Elaboración propia.

En el código 16, se puede observar la función del kernel que se ejecuta en paralelo. Para indicar que esta es la función del kernel se debe poner la directiva __global__ en la declaración de la función en la línea 1. En primer lugar, se inicializa el índice para indicar que juego de parámetros se utiliza en esa ejecución. El índice se calcula de la siguiente manera:

$$\text{índice} = \text{blockIdx.x} * \text{blockDim.x} + \text{threadIdx.x} \tag{33}$$

Esta ecuación se utiliza para calcular un índice único para cada hilo en la ejecución de un kernel en paralelo en la GPU. Cada elemento es:

- **blockIdx.x**: Este es un valor que representa el índice del bloque en la dimensión x. Los bloques en CUDA están organizados en un grid tridimensional, pero si solo se utiliza una dimensión (como en este caso), blockIdx.x representa el índice en esa dimensión.
- **blockDim.x**: Este es un valor que representa el tamaño del bloque en la dimensión x. Indica cuántos threads hay en cada bloque en esa dimensión.
- **threadIdx.x**: Este es un valor que representa el índice del thread dentro de su bloque en la dimensión x.

La línea de código blockIdx.x * blockDim.x + threadIdx.x calcula el índice único para cada hilo en la ejecución del kernel:

- **blockIdx.x * blockDim.x**: Calcula el índice base para el bloque actual multiplicando el índice del bloque por el tamaño del bloque. Esto posiciona cada bloque en una ubicación diferente dentro del grid tridimensional.
- **+ threadIdx.x**: Agrega el índice del thread dentro de su bloque al índice base del bloque, lo que da como resultado el índice único del thread en toda el grid.

A continuación, se inializa a 0 la variable longitud para saber que tamaño tendrán los vectores x_0 y x_t. Después, si el índice calculado es menor al número de juegos de parámetros que queremos calcular:

- Primero se inicializan todas las variables necesarias para la función. Se crea el Ciclo c, se crea la variable a_2 con valor -10 + índice (esta variable es la que es diferente en cada juego de parámetros), se crean los vectores x_0 y x_t con tamaño size y finalmente, se inicializa la variable i a 0, que es la variable con la que se recorrerá el vector de entrada init_pos.
- Después, dentro del bucle while $i < size$, se llama a la función runge4 con la posicón inicial y la variable a_2 de ese juego de parámetros. Esta función calcula RK4 con interpolación

cuadrática inversa y retorna un Struct Ciclo si esa trayectoria tiene 2 puntos de cruce en el eje $y = 0$. Después se mira si el ciclo que retorna runge4 es diferente a -1000 para añadirlo en los vectores x_0 y x_t.

- Finalmente, si la *longitud* < 0, es decir, si existen posibles ciclos límite en este juego de parámetros, primero se cuentan cuantos ciclos límite existen, es decir, cuantos cruces en el eje $y = 0$ hay en el vector x_t donde la diferencia entre las y de los puntos que producen el corte sea menor que 1×10^{-3} . Esto se hace con el for de las líneas 22-26. Finalmente, se añade 1 a la posición ciclos[count], donde count es el número de ciclos límite que existen para ese juego de parámetros.

  Esta adición en el vector ciclos se hace con la función atomicAdd(). Esta función se utiliza para realizar una suma atómica en una variable almacenada en la memoria global de la GPU. Esto significa que la operación de suma se ejecuta de forma atómica, lo que garantiza que no se produzcan condiciones de carrera cuando varios threads intentan actualizar el mismo valor al mismo tiempo.

Finalmente, tenemos la función runge4, donde se llama a la función calcular_posición, y a la vez, esta llama a las funciones dx_ y d_t. Estas funciones tienen el mismo código que el del código 4 pero con la diferencia que se ejecutan en el device (GPU).

```
1  __device__ struct Punto calcular_posicion(double x_ant, double y_ant, double a2_var) { }
2
3  __device__ double dx_dt(double x, double y) { }
4
5  __device__ double dy_dt(double x, double y, double a2_var) { }
```

Listing 17: Declaración de funciones del device (GPU). Elaboración propia.

Para indicar que se ejecutan en GPU, se pone la directiva __device__, como se indica en el código 17. Cuando una función se marca con __device__, significa que la función será compilada y ejecutada en la GPU, y puede ser llamada desde un kernel CUDA o desde otra función __device__.

11. Resultados

Todas las versiones de los algoritmos paralelos se ejecutarán en una GPU Gigabyte GeForce RTX 3060 Ti OC Gaming 8Gb GDDR6.

11.1. Número de ciclos límite

En este apartado se analizará el número de ciclos límite para 100000 juegos de parámetros con el algoritmo paralelo. Este algoritmo paralelo utiliza RK4 con $dt = 1 \times 10^{-3}$ y con interpolación cuadrática inversa. En cada ejecución se calculan $1600 *$ JUEGO_PARAMETROS trayectorias, ya que en cada posición del vector init_pos, que tiene un rango de 0 a 16, ambos incluidos, con diferencia de 0.01, se aplica RK4. El valor de las variables libres de la ecuación 7 son las de la tabla 8, pero variando el valor de una variable. Se ha utilizado los siguientes juegos de parámetros y se han obtenido los siguientes resultados:

- Solamente se ha cambiado el valor de la variable a_2 de la siguiente manera:

$$a_2 = -10,0 + j, \quad 0 \leq j < \text{NÚMERO DE JUEGOS DE PARÁMETROS} = 100000 \quad (34)$$

Y se han obtenido los siguientes resultados:

0 ciclos	3 ciclos
99999	1

Tabla 12: Número de juegos de parámetros con ciclos límites variando la variable a_2 en función de la ecuación 34. Elaboración propia.

Se ha obtenido un juego de parámetros con 3 ciclos límite, con $a\_2 = -10{,}0$, todos los demás juegos de parámetros tienen 0 ciclos límite.

- Solamente se ha cambiado el valor de la variable b_2 de la siguiente manera:

$$b_2 = 2,2 + j, \quad 0 \leq j < \text{NÚMERO DE JUEGOS DE PARÁMETROS} = 100000 \quad (35)$$

Y se han obtenido los siguientes resultados:

0 ciclos	1 ciclo	2 ciclos	3 ciclos
99481	513	5	1

Tabla 13: Número de juegos de parámetros con ciclos límite variando la variable b_2 en función de la ecuación 35. Elaboración propia.

- Solamente se ha cambiado el valor de la variable beta_2 de la siguiente manera:

$$beta_2 = 0,0015 - j, \quad 0 \leq j < \text{NÚMERO DE JUEGOS DE PARÁMETROS} = 100000 \tag{36}$$

  Y se han obtenido los siguientes resultados:

0 ciclos	1 ciclo	3 ciclos
99998	1	1

Tabla 14: Número de juegos de parámetros con ciclos límite variando la variable beta_2 en función de la ecuación 36. Elaboración propia.

- Solamente se ha cambiado el valor de la variable c_2 de la siguiente manera:

$$c_2 = -j * 0,1, \quad 0 \leq j < \text{NÚMERO DE JUEGOS DE PARÁMETROS} = 100000 \tag{37}$$

  Y se han obtenido los siguientes resultados:

0 ciclos	1 ciclo
99799	201

Tabla 15: Número de juegos de parámetros con ciclos límite variando la variable c_2 en función de la ecuación 37. Elaboración propia.

- Las variables tendrán valores random entre -50.0 y 50.0 en cada juego de parámetros, calculando un total de 100000 juegos de parámetros. Corresponde al archivo tfg_cuda_random.cu. Se han obtenido los siguientes resultados:

0 ciclos	1 ciclo	2 ciclos
99922	74	4

Tabla 16: Número de juegos de parámetros con ciclos límite con las variables con valores random entre -50.0 y 50.0. Elaboración propia.

Como se puede ver en los diferentes estudios realizados, en total 8×10^8 trayectorias (5 estudios* 100000 juegos de parámetros por estudio * 1600 trayectorias por juego de parámetros), en ninguna de las búsquedas se encontró 4 o más ciclos límites. Esto es debido a que, menos el último estudio, los valores de las variables eran muy parecidos a los de la tabla 8 y, por lo tanto, no iba haber mucha diferencia en el número de ciclos límite. También podemos observar que los juegos de parámetros que se obtiene 3 ciclos límite son los juegos de parámetros con los valores de las variables de la tabla 8. Por lo tanto, podemos concluir que la mayoría de casos se obtiene trayectorias con 0, 1 y 2 ciclos límite.

11.2. Análisis de rendimiento

En este subapartado se comparan los tiempos de ejecución del algoritmo secuencial del apartado 8.1.2, pero sin la visualización de los gráficos y de los ciclos límites obtenidos, con el algoritmo paralelo del apartado 10. Estos dos algoritmos utilizan RK4 con $dt = 1 \times 10^{-3}$ y con interpolación cuadrática inversa porque, como se ha demostrado en el apartado 9, es la versión más rápida y con una precisión alta. Estos dos algoritmos calculan $1600 *$ JUEGO_PARAMETROS trayectorias, ya que en cada posición del vector init_pos, que tiene un rango de 0 a 16, ambos incluidos, con diferencia de 0.01.

En este apartado se define el speed-up del algoritmo paralelo:

$$\text{Speed-up} = \frac{\text{tiempo ejecución secuencial}}{\text{tiempo ejecución paralelo}} \tag{38}$$

El speed-up nos marca las veces que es más rapido el algoritmo paralelo que el algoritmo secuencial. Los tiempos obtenidos son los siguientes:

Número juegos de parámetros	Tiempo Secuencial (ms)	Tiempo Paralelo (ms)	Speed-up
1	62	134	0,4626865672
10	901	6552	0,1375152625
100	76285	44489	1,71469352
1000	901463	181056	4,978918125
10000	9207698	188735	48,78638302
100000	91203180	997241	91,45550574

Tabla 17: Tiempos de ejecución del algoritmo secuencial del apartado 8.1.2 y del algoritmo paralelo del apartado 10. Elaboración propia.

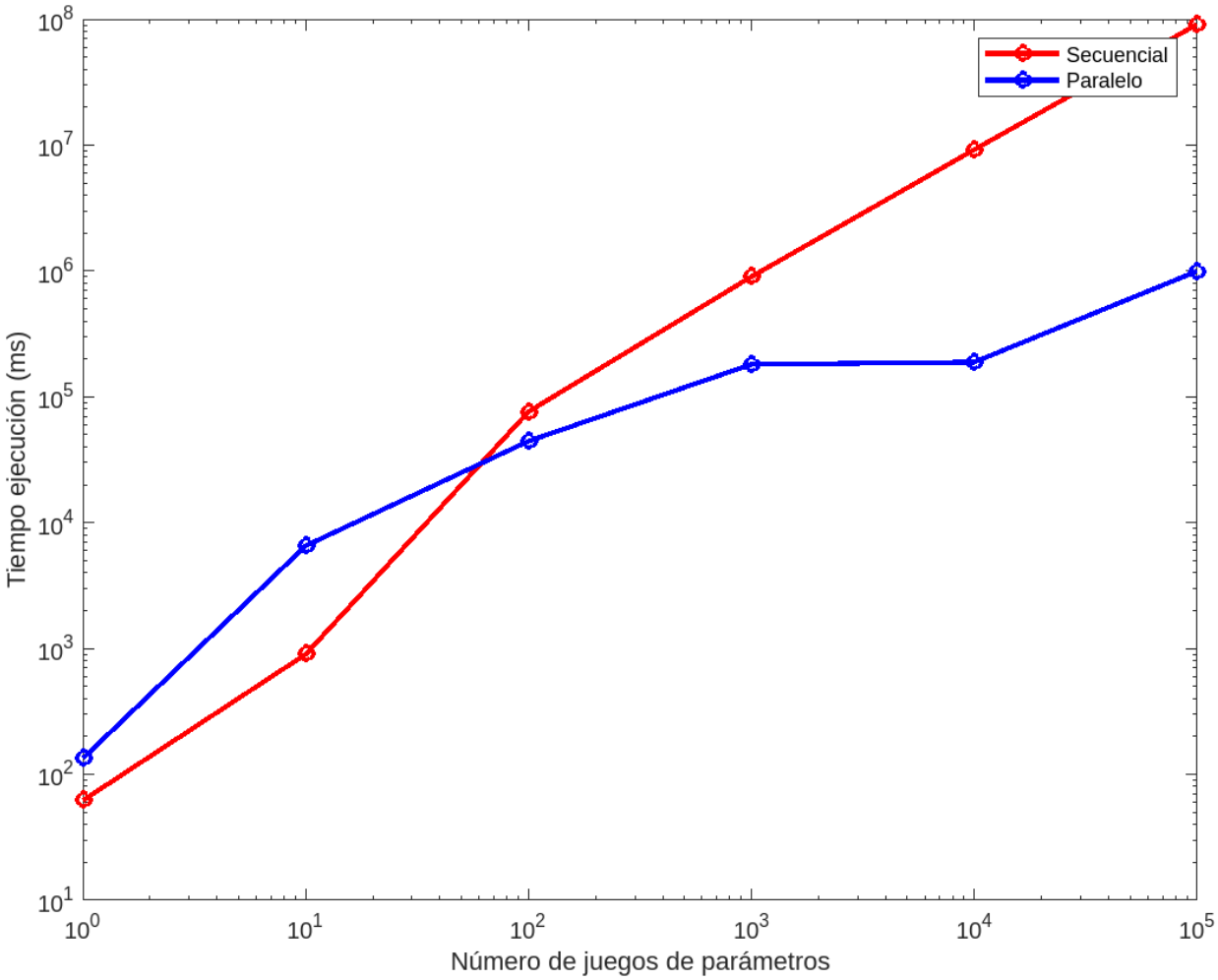

Figura 8: Gráfico de los tiempos de ejecución del algoritmo secuencial del apartado 8.1.2 y del algoritmo paralelo del apartado 10. Elaboración propia con un scrpit de MATLAB.

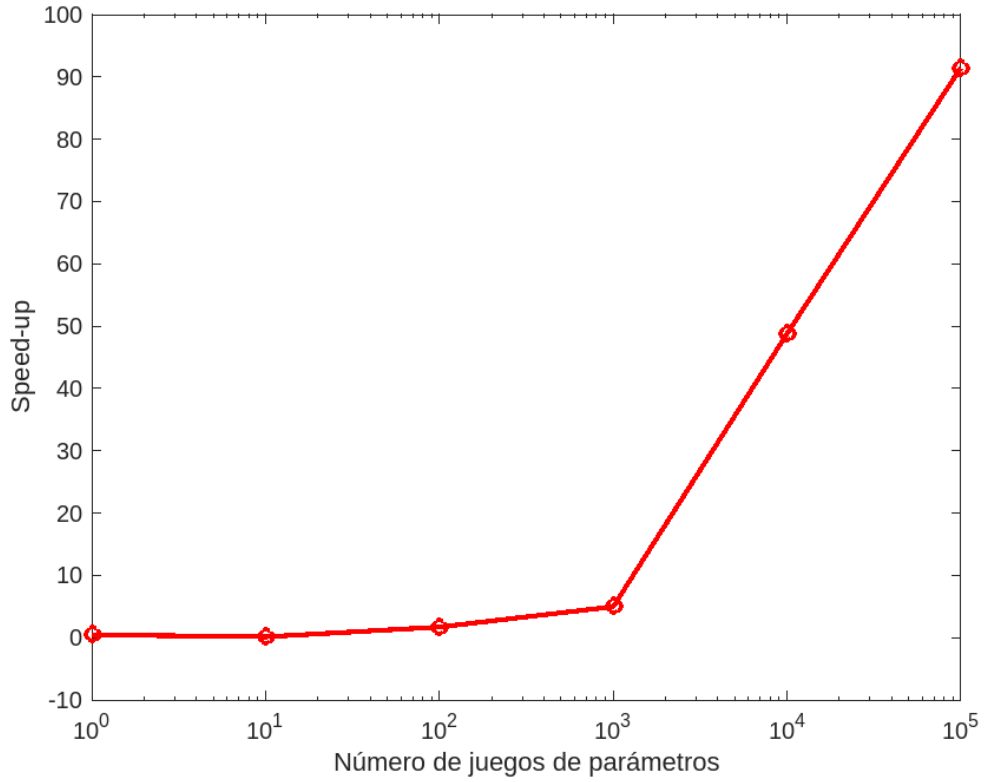

Figura 9: Gráfico del speed-up del algoritmo paralelo del apartado 10 sobre el secuencial del apartado 8.1.2. Elaboración propia con un scrpit de MATLAB.

Lo primero que se puede observar en la tabla 17 y en la imagen 8 es que los tiempos de ejecución del algoritmo secuencias es aproximadamente lineal al número de juegos de parámetros. También se puede observar que los tiempos del algoritmo paralelo son mucho más bajos que los del algoritmo secuencial para número de juegos de parámetros grandes. Para número de juegos de parámetros pequeños, el algoritmo paralelo no es mucho más rápido, o incluso más lento, ya que existe overhead producido en la paralelización. Se define el overhead como el tiempo que lleva realizar alguna operación que idealmente desearía que no tomara tiempo, y esto termina limitando la velocidad a la que puede realizar esa operación. Es tiempo invertido (latencia) sin realizar ningún trabajo útil en el kernel. Este overhead es producido por diferentes motivos:

- **CPU wrapper overhead**: Este es el overhead de los wrappers alrededor de un kernel CUDA en el lado de la CPU.
- **Overhead de la memoria**: Esta es el overhead de mover datos de la CPU a la GPU, o de la GPU a la CPU.

- **Overhead de lanzamiento de GPU**: Este es el tiempo que le toma a la GPU recuperar el comando y comenzar a ejecutarlo.

Por estos motivos, y como se observa en el gráfico 9, el algoritmo secuencial es más rápido que el paralelo para un número pequeño de juegos de parámetros. En cambio, para un número grande de juegos de parámetros, el overhead es insignificante en comparación de la ganancia de ejecutar el algoritmo en paralelo.

12. Conclusiones

En este proyecto se ha tratado la segunda parte del decimosexto problema de Hilbert. Esto me ha servido para estudiar en profundidad los conceptos de sistema dinámico, sistema de ecuaciones diferenciales y ciclos límite. También ha servido para hacer una búsqueda de métodos para resolver sistemas de ecuaciones y diferenciales e integrarlos en código de programación.

Este proyecto también ha servido para hacer comparaciones entre diferentes lenguajes de programación e intentar encontrar mejores tiempos de ejecución de estos algoritmos sin perder precisión.

También se ha mejorado el tiempo de ejecución a través de la paralelización con CUDA. Este era uno de los objetivos del proyecto y se ha conseguido porque la versión paralela es mucho más rápida que la versión secuencial.

Por último, no se ha encontrado una conclusión final con el número de ciclos límite, ya que en el espacio donde se ha buscado ciclos límite no se han encontrado muchos.

Finalmente, este algoritmo en CUDA es un buen inicio para hacer un estudio más exhaustivo del problema, ya que se puede mejorar y encontrar resultados concluyentes del problema.

13. Sostenibilidad

13.1. Autoevaluación

Durante todo el Grado de Ingeniería Informática nos han intentado inculcar la importancia de la sostenibilidad en el ámbito de la informática. Aun así, después de realizar la encuesta me he dado cuenta de que mis conocimientos en la materia son escasos.

En la encuesta sobre todo me he dado cuenta mi falta de conocimiento en calcular las métricas de las dimensiones económica, social y ambiental para después aplicar estrategias y métodos para reducir el impacto ambiental, económico y social. Aunque no he tenido muy en cuenta a la hora de diseñar el proyecto la sostenibilidad de éste, considero que es un proyecto que no impacta mucho sobre la sostenibilidad.

13.2. Dimensión económica

PPP: ¿Has estimado el coste de la realización del proyecto?

- La estimación del presupuesto del proyecto está en el apartado 1. Este presupuesto tiene en cuenta el coste por personal, por material hardware y software, coste energético, por espacio físico y costes por contingencia y posibles inconvenientes. Es un presupuesto aproximado de cuanto valdría el proyecto si fuera un proyecto empresarial en lugar de una tesis de grado universitario.

PPP: ¿Has cuantificado el coste (recursos humanos y materiales) de la realización del proyecto? ¿Qué decisiones has tomado para reducir el coste? ¿Has cuantificado el ahorro?

- La estimación del presupuesto del proyecto está en el apartado 1. Este presupuesto tiene en cuenta el coste por personal, por material hardware y software, coste energético, por espacio físico y costes por contingencia y posibles inconvenientes. Se ha estimado las mínimas horas posibles del personal para reducir gastos. También se ha estimado el mínimo material posible para hacer la realización del proyecto. No se ha cuantificado el ahorro.

PPP: ¿Se ha ajustado el coste previsto al coste final? ¿Has justificado las diferencias?

- Al final el coste ha sido superior porque se ha necesitado más tiempo para finalizarlas de lo estimado y, por lo tanto, el coste del personal ha aumentado.

Vida Útil: ¿Cómo se resuelve actualmente el problema que quieres abordar? ¿En qué mejorará económicamente tu solución a las existentes?

- Está pregunta no se puede responder, ya que nuestro problema está sin resolver. Actualmente, la comunidad matemática está intentando resolver el problema de manera analítica y nuestro proyecto daría más datos para que puedan intentar resolver el problema.

Vida Útil: ¿Qué coste estimas que tendrá el proyecto durante su vida útil? ¿Se podría reducir este coste para hacerlo más viable?

- La estimación del presupuesto del proyecto está en el apartado 1. Este presupuesto tiene en cuenta el coste por personal, por material hardware y software, coste energético, por espacio físico y costes por contingencia y posibles inconvenientes. Se ha utilizado lo estrictamente necesario para la realización del proyecto.

Vida Útil: ¿Se ha tenido en cuenta el coste de los ajustes/actualizaciones/reparaciones durante la vida útil del proyecto?

- Sí, por ejemplo, ha habido 2 tareas que se han extendido más en el tiempo que lo estimado. Por lo tanto, se ha sumado el coste del personal y del consumo de los materiales de las horas de más estimadas.

Riesgos: ¿Podrían producirse escenarios que perjudicasen la viabilidad del proyecto?

- Sí, por ejemplo, que deje de funcionar el ordenador y necesite una reparación y, por lo tanto, aumente el coste.

13.3. Dimensión ambiental

PPP: ¿Has estimado el impacto ambiental que tendrá la realización del proyecto? ¿Te has planteado minimizar el impacto, por ejemplo, reutilizando recursos?

- No he tenido en cuenta el impacto ambiental de este proyecto a la hora de decidir como abordarlo. Aun así, el proyecto no es muy perjudicial respecto al medio ambiente. Primero solamente hay un ordenador portátil, por lo tanto, reducimos el consumo eléctrico. Y también, al realizarse el proyecto en remoto, eliminamos emisiones de CO_2, ya que no hay desplazamientos. Todos los recursos personales que no son gratuitos ya estaban en mi posesión antes del proyecto, y los seguiré utilizando después del proyecto, como es el caso

del portátil.

PPP: ¿Has cuantificado el impacto ambiental de la realización del proyecto? ¿Qué medidas has tomado para reducir el impacto? ¿Has cuantificado esta reducción?

- No se ha cuantificado explícitamente, pero se ha hecho un cálculo de la estimación del consumo. Las medidas que se han tomado para reducir el impacto ha sido sobretodo sobre la utilización del ordenador. Se ha intentado apagarlo cada vez que no se usaba. Esta reducción se podrá cuantificar cuando se tengan todas las facturas y se pueda comparar con la estimación.

PPP: Si hicieras de nuevo el proyecto, ¿podrías realizarlo con menos recursos?

- La realidad es que no. Se han utilizado los recursos necesarios para la realización del proyecto.

Vida Útil: ¿Cómo se resuelve actualmente el problema que quieres abordar? ¿En qué mejoraría ambientalmente tu solución a las existentes?

- Como en la dimensión económica, el problema no tiene solución, por lo tanto, no hay nada que mejorar en la dimensión ambiental. Aun así, para proyectos futuros que abordaran el problema tendrían más datos para intentar resolver el problema y así reducir el impacto ambiental.

Vida Útil: ¿Qué recursos estimas que se usarán durante la vida útil del proyecto? ¿Cuál será el impacto ambiental de estos recursos?

- Los recursos que se utilizarán están detallados en el apartado 4.1. El único impacto ambiental generado es el producido por el consumo eléctrico que necesitan estos recursos.

Vida Útil: ¿El proyecto permitirá reducir el uso de otros recursos? ¿Globalmente, el uso del proyecto mejorará la huella ecológica?

- Uno de los objetivos del proyecto era mejorar el tiempo de ejecución del algoritmo secuencial, así que se reduce el tiempo de consumo del ordenador y, por lo tanto, se reduce la huella ecológica.

Riesgos: ¿Podrían producirse escenarios que hiciesen aumentar la huella ecológica del proyecto?

- No, lo único que si este proyecto se ejecuta con una CPU y GPU más lenta, pues el tiempo de ejecución sería más alto y, por lo tanto, aumentaría el consumo.

13.4. Dimensión social

PPP: ¿Qué crees que te va a aportar a nivel personal la realización de este proyecto?

- Este proyecto me puede aportar mucho personalmente. Por ejemplo, me otorgará mucho más rigor y ser más metódico a la hora de realizar proyectos en el futuro. También en tener en cuenta aspectos que hasta ahora no había tenido mucho en cuenta como puede ser el aspecto económico, ambiental y social.

PPP: ¿La realización de este proyecto ha implicado reflexiones significativas a nivel personal, profesional o ético de las personas que han intervenido?

- Sí, una reflexión que he tenido ha sido lo interesante que es la algoritmia y la paralelización con tal de mejorar programas y el tiempo de ejecución.

Vida Útil: ¿Cómo se resuelve actualmente el problema que quieres abordar? ¿En qué mejoraría socialmente tu solución a las existentes?

- No puede responder esta pregunta, cómo he indicado antes este problema no tiene solución existente.

Vida Útil: ¿Existe una necesidad real del proyecto?

- Nosotros que existe una necesidad del proyecto. Este problema existe desde el año 1900 y todavía no ha sido resuelto. Nuestro proyecto aportará nuevos datos con los cuales la comunidad matemática se puede beneficiar para seguir estudiando el problema.

Vida Útil: ¿Quién se beneficiará del uso del proyecto? ¿Hay algún colectivo que puede verse perjudicado por el proyecto? ¿En qué medida?

- Se puede beneficiar futuros matemáticos o informáticos que quieran estudiar este problema. Podrían utilizar mi algoritmo en paralelo como inicio para mejorarlo y hacer un estudio más exhaustivo.

Vida Útil: ¿En qué medida soluciona el proyecto el problema planteado inicialmente?

- El problema sigue sin solución, pero se ha creado un algoritmo paralelo rápido para hacer un estudio del problema.

Riesgos: ¿Podrían producirse escenarios que hiciesen que el proyecto fuese perjudicial para algún segmento particular de la población?

- No, al fin y al cabo el proyecto es un algoritmo paralelo.

Riesgos: ¿Podría crear el proyecto algún tipo de dependencia que dejase a los usuarios en posición de debilidad?

- No, al fin y al cabo el proyecto es un algoritmo paralelo.

14. Bibliografia

[1] Jaume Llibre. *Sobre el problema 16 de Hilbert*. 2015. URL: https://gaceta.rsme.es/abrir.php?id=1289.

[2] Mgs. Mario Suárez. *Introducción a las ecuaciones diferenciales. Teoría y ejemplos resueltos*. 2013. URL: https://www.monografias.com/trabajos97/introduccion-ecuaciones-diferenciales-teoria-y-ejemplos-resueltos/introduccion-ecuaciones-diferenciales-teoria-y-ejemplos-resueltos.

[3] Marta. *Funciones racionales*. 2020. URL: https://www.superprof.es/apuntes/escolar/matematicas/calculo/funciones/funciones-racionales.html.

[4] *Sistema dinámico*. URL: https://hmong.es/wiki/Dynamical_systems.

[5] *Espacio de fase*. URL: https://academia-lab.com/enciclopedia/espacio-de-fase/.

[6] *Ciclo límite*. URL: https://campusvirtual.ull.es/ocw/pluginfile.php/19251/mod_resource/content/15/mod_tema6_ocw.pdf.

[7] *Runge-Kutta de cuarto orden*. URL: https://esimecuanalisisnumerico.wordpress.com/2014/05/06/metodo-numerico-de-runge-kutta/.

[8] *Interpolación cuadrática inversa*. URL: https://es.frwiki.wiki/wiki/Interpolation_quadratique_inverse.

[9] *Sueldo programador junior*. URL: https://es.indeed.com/career/programador-junior/salaries/Barcelona--Barcelona-provincia.

[10] *Sueldo director de proyecto*. URL: https://es.indeed.com/career/director-de-proyecto/salaries/Barcelona-provincia.

[11] *Calcular amortización de un producto*. 2018. URL: https://tutobasico.com/calcular-amortizacion/.

[12] *Precio Microsoft 365 Personal*. URL: https://www.microsoft.com/es-ES/microsoft-365/buy/compare-all-microsoft-365-products.

[13] *Potencia ordenador*. URL: https://www.pccomponentes.com/cougar-gex650-650w-80-plus-gold-modular.

[14] *Precio medio de la electricidad*. URL: https://tarifaluzhora.es/.

[15] Leonov G.A Kuznetsov N.V. Kuznetsova O.A. *Visualization of four normal size limit cycles in two-dimensional polynomial quadratic system*. 2012. URL: https://www.math.spbu.ru/

user/nk/PDF/2012-DEDS-16-Hilbert-problem-four-limit-cycles-quadratic-system.pdf.

[16] *Modelo de programación CUDA*. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#programming-model.