



FACULTAD DE INFORMÁTICA DE BARCELONA

Simulaciones clásicas de dinámica en
sistemas unidimensionales con diferentes
tipos de interacción

Proyecto Fin de Carrera

Autor: Bryan Leonardo salto salao

director: Grigori Astrakharchik

codirector: Pietro Massignan

Tutor GEP: Joan Subirats Soler

Computación

Miércoles, 22 de junio de 2022

Universidad Politècnica de Catalunya

Resumen

La relajación violenta es un proceso que ocurre en sistemas con interacciones de núcleo blando u duro (sistemas clásicos). Tiene una característica peculiar de amplificar dramáticamente pequeñas perturbaciones, lo que lleva a un colapso muy rápido en configuraciones de evolución lenta conocidas como estados cuasiestacionarios.

El objetivo del proyecto es realizar simulaciones clásicas de dinámica en sistemas unidimensionales clásicos con diferentes tipos de interacción. Queremos replicar plots del artículo *Violent relaxation in quantum fluids with long-range interactions*, para verificar su veracidad. Además, queremos llegar un poco más lejos y ver que sucede con otros tipos de interacción. Para ello hemos estudiado otros potenciales de interacción como por ejemplo Power-law Potential $V(x) = \frac{A}{|1+x|^P}$, la potencia directa $V(x) = |x|^P$, la Power-law Potential en función de la distancia chord (distancia recta entre dos puntos de una circunferencia) $V(x) = \frac{A}{|\text{chord}(x)|^P}$, entre otras. Para ello hemos desarrollado un programa que dado unos datos de entrada te puede plotear una gráfica tiempo/θ . Aparte hemos creado funciones auxiliares para poder comprobar el correcto funcionamiento de nuestro programa.

El programa es simple y fácil de modificar, para, si se desea, uno pueda añadir un nuevo tipo de interacción y comprobar sus resultados. También se pueden guardar los datos, o leerlos. Comprobar si el valor medio del Hamiltoniano se conserva y más cosas.

Como se puede observar, hay conceptos de física y matemáticas que se tienen que aprender o al menos tener en cuenta para poder realizar el programa. En el presente proyecto se encuentran todos los cálculos y conceptos que consideramos importantes para que se puedan entender los resultados y analizar su correcto funcionamiento. También se encuentran los problemas y los límites a los que hemos llegado. Aun así, el presente programa es una gran base para ampliar el estudio a nuevas interacciones y cálculos.

Resum

La relaxació violenta és un procés que es produeix en sistemes amb interaccions de nucli tou o dur (sistemes clàssics). Té una característica peculiar d'amplificar de manera espectacular les petites perturbacions, donant lloc a un col·lapse molt ràpid en configuracions d'evolució lenta conegudes com a estats quasi-estacionaris.

L'objectiu del projecte és fer simulacions clàssiques de dinàmica en sistemes unidimensionals clàssics amb diferents tipus d'interacció. Volem replicar plots de l'article *Violent relaxation in quantum fluids with long-range interactions*, per verificar-ne la veracitat. A més, volem arribar una mica més lluny i veure què passa amb altres tipus d'interacció. Per això hem estudiat altres potencials d'interacció com ara Power-law Potential $V(x) = \frac{A}{|1+x|^P}$, la potència directa $V(x) = |x|^P$, la Power-law Potential en funció de la distància chord (distància recta entre dos punts d'una circumferència) $V(x) = \frac{A}{|\text{chord}(x)|^P}$, entre d'altres. Per això hem desenvolupat un programa que donat unes dades d'entrada et pot plotejar una gràfica temps/θ . A banda hem creat funcions auxiliars per poder

comprovar el correcte funcionament del nostre programa.

El programa és simple i fàcil de modificar, per, si es vol, un pugui afegir un nou tipus d'interacció i comprovar-ne els resultats. També es poden guardar les dades, o llegir-les. Comprovar si el valor mitjà de l'Hamiltonià es conserva i més coses.

Com es pot observar, hi ha conceptes de física i matemàtiques que cal aprendre o almenys tenir en compte per poder realitzar el programa. En aquest projecte es troben tots els càlculs i conceptes que considerem importants perquè es puguin entendre els resultats i analitzar-ne el funcionament. També hi ha els problemes i els límits als què hem arribat. Tot i així, aquest programa és una gran base per ampliar l'estudi a noves interaccions i càlculs.

Abstract

Violent relaxation is a process that occurs in systems with soft-core an hard-core interactions. It has a peculiar feature of dramatically amplifying small perturbations, leading to a very rapid collapse into slowly evolving configurations known as quasistationary states.

The aim of the project is to perform classical simulations of dynamics in classical one-dimensional systems with different types of interaction. We want to replicate plots from the article *Violent relaxation in quantum fluids with long-range interactions*, to verify their veracity. Also, we want to go a little further and see what happens with other types of interaction. For this we have studied other interaction potentials such as Power-law Potential $V(x) = \frac{A}{|1+x|^P}$, the direct power $V(x) = |x|^P$, the Power-law Potential as a function of the chord distance (straight distance between two points on a circumference) $V(x) = \frac{A}{|chord(x)|^P}$, among others. For this we have developed a program that, given some input data, can plot a *time*/ θ graph. Besides, we have created auxiliary functions to be able to check the correct functioning of our program.

The program is simple and easy to modify, so that, if desired, one can add a new type of interaction and check its results. The data can also be saved, or read. Check if the mean value of the Hamiltonian is conserved and more.

As can be seen, there are concepts of physics and mathematics that must be learned or at least taken into account in order to carry out the program. In this project we find all the calculations and concepts that we consider important so that the results can be understood and their correct functioning analyzed. There are also the problems and the limits we have reached. Even so, the present program is a great base to extend the study to new interactions and calculations.

Índice

1. Contexto	6
1.1. Introducción	6
1.2. Problema	6
1.3. Actores implicados	8
2. Planificación Inicial	9
2.1. Justificación	9
2.1.1. Situación actual	9
2.1.2. Selección del software	9
2.2. Alcance y obstáculos	10
2.2.1. Objetivo	10
2.2.2. Subobjetivos e indicadores	10
2.2.3. Requisitos	11
2.2.4. Obstáculos	11
2.2.5. Riesgos	12
2.3. Metodología y seguimiento	12
2.3.1. Metodología del trabajo	12
2.3.2. Seguimiento	13
2.3.3. Planificación temporal	13
2.3.4. Fases del proyecto	13
2.3.5. Descripción de las tareas	14
2.3.6. diseño (D)	14
2.3.7. Implementación (I)	14
2.3.8. Pruebas (PR)	15
2.3.9. Documentación (DO)	15
2.3.10. Reuniones de seguimiento (R)	15
2.4. Tabla de Tareas	16
2.5. Dependencias de las tareas	17
2.6. Tiempo de las tareas	17
2.7. Diagrama de Gantt	18
2.8. Personal y material	21
2.9. Gestión de riesgos	21
2.10. Presupuesto	23
2.10.1. Identificación de costes	23
2.11. Coste de personal	23
2.11.1. Costes de desarrollo	25
2.11.2. Coste de Energía	26
2.12. Contingencias	26
2.13. Imprevistos	26
2.14. Coste total del proyecto	27
2.15. Control de gestión	27
2.16. Sostenibilidad	29
2.16.1. Reflexión	29
2.16.2. Dimensión económica	29

2.16.3. Dimensión social	30
3. Análisis	31
3.1. Modelo utilizado	31
3.2. Método numérico	35
4. Desarrollo	36
4.1. Implementación	36
4.1.1. Lectura	36
4.1.2. Simulador	37
4.2. Representación de los resultados	42
5. Pruebas y Resultados	44
5.1. Análisis de los Resultados	44
5.1.1. Figura a	44
5.1.2. Figura b	45
5.2. Conservación de la energía	47
5.3. Análisis temporal	50
5.4. Nuevas interacciones	52
5.4.1. Potencia de x	52
5.4.2. Power-law Potential	55
6. Obstáculos y cambios en el proyecto	57
7. Aclaraciones programa final	58
8. Conclusiones	59
9. Referencias	61

Índice de figuras

1.	trayectoria temporal de 52 partículas con fuerzas atractivas y repulsivas.[9]	7
2.	Resumen de las tareas	16
3.	Dependencias entre las tareas del proyecto	17
4.	Diagrama de Gantt	19
5.	Diagrama de Gantt simplificado	20
6.	Costes salariales del proyecto	23
7.	Costes salariales del proyecto por tareas	24
8.	Estimación de los costes genéricos. Elaboración propia.	25
9.	Estimación de los imprevistos. Elaboración propia.	27
10.	Estimación del coste total del proyecto. Elaboración propia.	27
11.	Ejemplo de anillo. Elaboración propia.	32
12.	Parte principal del simulador. Elaboración propia.	38
13.	Funciones auxiliares del simulador. Elaboración propia.	38
14.	Matrices alternativa 2. Elaboración propia.	39
15.	Parte principal del simulador alternativa 2. Elaboración propia.	40
16.	Fragmento del código encargado de generar los plots. Elaboración propia.	42
17.	Imagen comparativa <i>figura a</i> del artículo <i>vs</i> obtenida con el programa creado. Elaboración propia.	44
18.	Imagen comparativa <i>figura b</i> del artículo <i>vs</i> obtenida con el programa creado. Elaboración propia.	45
19.	Funciones <i>calcPotencia()</i> y <i>calcCinetic()</i> . Elaboración propia.	47
20.	Estudio de la convergencia. Ejemplo de artefactos en la energía del sistema debidos al paso del tiempo demasiado grande. Elaboración propia.	48
21.	Energía cinética, potencial y total en función del tiempo. Elaboración propia.	49
22.	Evolución del tiempo según el número de partículas N . Elaboración propia.	50
23.	Energía cinética, potencial y total en función del tiempo. Elaboración propia.	51
24.	Trayectoria temporal con $\epsilon = -1$, $N = 51$, $\Delta t = 0,0003$. Elaboración propia.	53
25.	Trayectoria temporal con $\epsilon = 1$, $N = 51$, $\Delta t = 0,0003$. Elaboración propia.	53
26.	Trayectoria temporal con $\epsilon = -1$, $N = 51$, $\Delta t = 0,0002$. Elaboración propia.	54
27.	Interacción con $\epsilon = 1$, $N = 51$, $\Delta t = 0,0003$. Elaboración propia.	55
28.	Trayectoria temporal con $P = 2$. Elaboración propia.	56

1. Contexto

1.1. Introducción

”La relajación violenta (también llamada ”Violent relaxation”) es un proceso que ocurre en sistemas con interacciones de largo alcance(conocidas como ”long-range interactions”). Dicho proceso tiene la característica peculiar de amplificar dramáticamente pequeñas perturbaciones y, en lugar de llevar el sistema al equilibrio, conduce a configuraciones de evolución lenta conocidas como estados cuasiestacionarios que caen fuera del paradigma estándar de la mecánica estadística. La relajación violenta se identificó originalmente en la dinámica estelar impulsada por la gravedad; aquí, ampliamos la teoría al régimen cuántico mediante el desarrollo de una versión cuántica del modelo de campo medio hamiltoniano (HMF) que ejemplifica muchas de las propiedades genéricas de los sistemas con interacciones de largo alcance(”long-range interacting systems”). El modelo HMF puede verse como una descripción de partículas que interactúan a través de un potencial coseno, o de manera equivalente como el modelo cinético XY con interacciones de rango infinito, y su dinámica cuántica de fluidos puede obtenerse de una ecuación generalizada de Gross-Pitaevskii.”(Ryan Plestid, Perry Mahon, and DHJ O’Dell. Violent relaxation in quantum fluids with long-range interactions. *Physical Review E*, 2018.) 28) [9]

En el artículo *Violent relaxation in quantum fluids with long-range interactions*[9] se demuestra que las cúspides singulares que se forman durante la relajación violenta están reguladas por efectos de interferencia de una manera universal descrita por la teoría de catástrofes de Thom[11] aplicada a las ondas y esto conduce a escalas de longitud y escalas de tiempo emergentes que no están presentes en el problema clásico. En el régimen cuántico profundo encontramos que la relajación violenta es suprimida por completo por el movimiento cuántico de punto cero. Sus resultados son relevantes para los estudios de laboratorio de autoorganización en gases atómicos fríos con interacciones de largo alcance.

Como se puede observar este problema planteado se aleja un poco del marco de la FIB, de hecho es un problema/investigación dentro del marco de DFIS [2], el departamento de física. Aunque resolver los cálculos de los modelos planteados, sí que entra dentro del marco de la FIB. Necesitan la ayuda de un informático con conocimientos de computación. Uno que sea capaz de resolver los cálculos con un programa, y además, pueda participar activamente en el proyecto. El principal problema será la resolución de ecuaciones diferenciales[7] y la representación de los resultados.

1.2. Problema

Dentro del Departamento de Física de la Universidad Politécnica de Barcelona (DFIS)[2] los profesores/investigadores *Grigori Astrakharchik* y *Pietro Massignan* están tratando de replicar y avanzar respecto el problema planteado anteriormente. Para ello necesitan un programa capaz de simular un sistema unidimensional clásico con diferentes tipos de interacciones, el modelo explicado en el apartado anterior. Con dicho programa se necesita comprobar/replicar los resultados del artículo *Violent relaxation in quantum fluids with long-range interactions*[9] y además llevar varias

simulaciones extras. Para ello es necesario un programador capaz de realizar un programa eficiente y fácil de modificar. Además de ser capaz de realizar cálculos de diversa complejidad (Ecuaciones diferenciales) todo esto por supuesto dentro del campo de la computación.

El problema que se nos plantea es realizar el programa, testearlo y finalmente recopilar datos; no analizarlos y sacar conclusiones de como encaja todo dentro de la mecánica clásica, de eso se encargarían *Grigori Astrakharchik* y *Pietro Massignan*, los directores de este proyecto. Aun así, para realizar un buen programa es necesario entender el contexto y saber interpretar los resultados que se obtengan. Parte del proyecto será analizar bien el problema y mirar cuál será la mejor estrategia para resolverlo.

Dicho programa tendrá que ser capaz de simular una versión del modelo de campo medio Hamiltoniano (HMF)[11]. Que sigue las fórmulas 1 y 2:

$$H = \sum_i \frac{L_i^2}{2mR^2} + \frac{\epsilon}{N} \sum_{i<j} \cos(\Theta_i - \Theta_j) \quad (1)$$

$$\frac{df}{dt} = \partial_t f + \dot{\theta} \partial_\theta f + \dot{L} \partial_L f = 0 \quad (2)$$

Una vez finalizado el programa tendremos que verificar los plots de las figura 1. Para ello es importante elegir un buen lenguaje de programación, uno enfocado ya al cálculo como por ejemplo R o uno muy utilizado en la comunidad científica por su sencillez como Python.

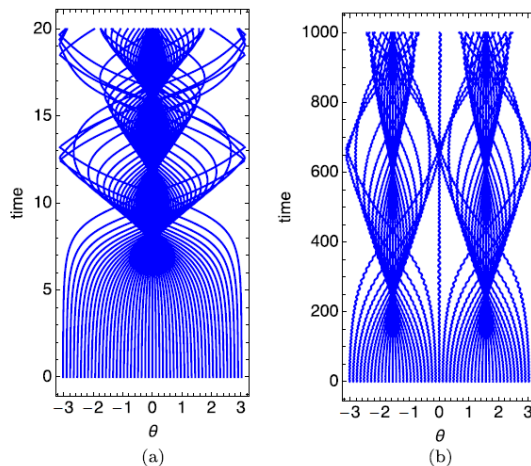


Figura 1: trayectoria temporal de 52 partículas con fuerzas atractivas y repulsivas.[9]

Las interacciones de largo alcance tienden a amplificar las perturbaciones iniciales. Ilustramos esta característica aquí con las trayectorias newtonianas de 52 partículas que obedecen al modelo HMF clásico con interacciones (a) atractivas ($\epsilon < 0$) y (b) repulsivas ($\epsilon > 0$). En $t = 0$, las partículas están espaciadas uniformemente alrededor del anillo con velocidades iniciales $v_i(\theta, 0) = 0,005\cos(\theta)$ que varían muy poco. En ambos casos, el LRI(long-range interactions) hace que las partículas se agrupen y este comportamiento se repite de tal manera que las envolturas de las trayectorias forman una serie cuasiperiódica de cúspides en forma de chevrones (o “chevrone” [75,76]). Sin embargo, existen diferencias clave: primero, las interacciones atractivas dan lugar a un único punto de agrupación (monocluster) alrededor del anillo, mientras que las interacciones repulsivas producen dos puntos de agrupación (bicluster), y segundo, hay escalas de tiempo muy diferentes asociadas con los dos casos con el caso repulsivo es mucho más lento (nótese las diferentes escalas en los ejes de tiempo). Fragmento del artículo *Violent relaxation in quantum fluids with long-range interactions*[9].

1.3. Actores implicados

En este proyecto hay diversos actores implicados:

- **El personal del proyecto.** Nosotros desarrollaremos el programa para representar el modelo (HMF). Y una vez hecho lo analizaremos para poder compartirlo con la comunidad científica.
- **La comunidad científica.** La realización de este proyecto viene precedida por la realización de otros estudios. Así mismo, esperamos que con los resultados que obtengamos podamos servir de ayuda a otros científicos y así poder seguir avanzando en conjunto.
- **Ingenieros.** Las futuras aplicaciones de este tipo de estudios está en auge. Los ingenieros serán los encargados de sacarles provecho.

2. Planificación Inicial

2.1. Justificación

2.1.1. Situación actual

Como se ha mencionado anteriormente. DFIS[2] no dispone del software utilizado en estudios y artículos de otras universidades. Por ello necesitan de un programador con los conocimientos necesarios para realizar las simulaciones con un número definido de partículas. La idea es empezar por una e ir sumando para cerciorarnos del buen funcionamiento del programa y al mismo tiempo observar como se comportan.

Para ello se necesitan conocimientos básicos de Física, concretamente del campo a estudiar *Física de partículas*. Por lo tanto, el estudio del problema a resolver también ocupará una parte importante del tiempo.

Para poder realizar el proyecto y poder obtener todos los datos para resolver el problema. El programador tendrá la ayuda de sus dos directores, *Grigori Astrakharchik* y *Pietro Massignan*, quienes le supervisarán durante todo el proceso y le ayudarán en cualquier cosa que pueda llegar a necesitar.

2.1.2. Selección del software

Hay diversas opciones para resolver el problema presentado. Es importante escoger un buen lenguaje de programación para poder realizar la simulación. Al mismo tiempo, para realizar los plots podemos optar por utilizar el mismo lenguaje o buscar otra que nos ofrezca más posibilidades. A continuación pasaremos a analizar algunos de los lenguajes que serían una buena alternativa.

- **Python.**[6] Es el Lenguaje más usado hoy en día. En diferentes campos, tanto en el ámbito empresarial como en el ámbito científico. Aunque no es el que más hemos usado en la carrera, no requiere de mucha complejidad y hay un gran número de librerías que nos pueden ayudar a implementar las simulaciones y representar sus resultados.
- **C++** Es el lenguaje que más he tocado en la universidad, junto con C. Requiere cierta complejidad, pero es muy eficiente.
- **Matlab.**[14] Es un sistema de cómputo numérico que ofrece un entorno de desarrollo integrado. Es una opción de pago, pero gracias a la UPC se puede tener acceso a él de manera gratuita como estudiante. Al estar enfocado al cómputo numérico nos ofrece muchas facilidades, a la hora de calcular ecuaciones diferenciales, en comparación con C++ y también nos es más fácil representar los datos. Por otra parte, es más complejo que Python y necesitaríamos una curva de aprendizaje mayor en comparación a los otros dos lenguajes.

Mirando los requisitos y tras una pequeña búsqueda en la red, nos acabaríamos decantando por Python. Es el lenguaje más usado a nivel global. Es importante recordar que los usuarios finales serán físicos y estarán más familiarizados con Python que con C++. Aparte podemos encontrar más información y artículos en la red de algoritmos escritos en Python que en C++. Por otra parte, descartaríamos Matlab por la curva de aprendizaje, aunque no sería mala idea utilizarlo para algunos casos puntales.

2.2. Alcance y obstáculos

2.2.1. Objetivo

El objetivo final de este proyecto es implementar un simulador que represente el modelo clásico HMF con diferentes entradas (Número de partículas, por ejemplo). Para ello tendremos que implementar un programa lo más eficiente posible, fácil de entender y modificar. Para acabar, se tendrán que representar los datos y comprobar que coinciden con la gráfica 1.

Es importante que la entrada sea fácil de modificar y que la salida sea fácil de procesar y representar. Al mismo tiempo es buena idea que el código sea eficiente y simple de entender, para futuras aplicaciones.

2.2.2. Subobjetivos e indicadores

Los subobjetivos e indicadores son los siguientes.

1. Control y gestión de la entrada.

- a)* El programa es capaz de dar un valor por defecto a los parámetros no pasados.
- b)* El programa indica si hay algún error al procesar la entrada, en caso de haberlo indica cuál es.

2. Control y gestión de la salida del programa.

- a)* La Salida tiene el formato adecuado para posteriormente poder representar los datos gráficamente.
- b)* El mismo programa genera los gráficos directamente y se comunica con el generador de plots.

3. Buen funcionamiento del programa.

- a)* El programa es eficiente en términos generales.
- b)* El programa indica que parte de la simulación lleva hecha y cuanta queda por hacer.
- c)* Los resultados del programa son correctos y se pueden verificar.

4. Simulación computacional.

- a) El programa utiliza e implementa los algoritmos y estructura necesaria para realizar una simulación computacional.

2.2.3. Requisitos

Los requisitos que tienen que seguir el programa principal y el generador de gráficas son los siguientes:

1. Los parámetros tienen que ser fáciles de modificar. Por lo tanto, es necesario una entrada limpia y fácil de modificar, sería interesante que el programa utilizará una plantilla, y que solo tengamos que modificar los campos de ella.
2. El programa tiene que poder aprovechar al máximo el paralelismo. I acortar los tiempos de la simulación lo máximo posible.
3. La salida es fácil de interpretar y se puede aprovechar para crear las gráficas de manera directa. O generar dos ficheros diferentes, una para el usuario y otra para usarla posteriormente.
4. El programa tiene que estar documentado y facilitar cualquier posible mejora para próximos experimentos.

2.2.4. Obstáculos

Los obstáculos que tendremos al momento de realizar el proyecto planteado son los siguientes:

- *Inexperiencia.* En el momento de iniciar el proyecto y planificar su alcance, el autor no tiene conocimientos sólidos de como crear un simulador o la mejor alternativa para calcular ecuaciones diferenciales.
- *Desconocimiento de física.* El autor aunque tiene conocimientos básicos de física y de cálculo de derivadas. Tiene que reservar gran parte del proyecto a estudiar e investigar como puede resolver el problema planteado, entender lo que está programando e interpretar los resultados obtenidos.
- *Idioma* Aunque el autor tenga el nivel necesario para llegar a entender los artículos citados y sus tutores hablen perfectamente el español. Analizar y recopilar información le llevará más tiempo de lo normal, pero al mismo tiempo, le ayudará a mejorar su nivel de inglés necesario para obtener su título.

2.2.5. Riesgos

- *La gestión del tiempo.* En el semestre en el que este autor realiza su TFG, también está cursando otra materia, realizando un convenio de 45 h semanales, siendo mentor de tres estudiantes SALSAM (*Erasmus*)[3] y participa en la delegación de estudiantes DEFIB(delegación de estudiantes de la FIB). Por este motivo, picos de actividad inesperada de las otras actividades pueden forzar a cambios de la planificación del trabajo.
- *Accidentes* Aunque sea habitual tener accidentes, el riesgo de sufrir uno frecuentemente es ignorado. Vale la pena tenerlo en cuenta.
- *No correspondencia de los resultados.* Como hemos mencionado anteriormente, no disponemos el software original con el que se han efectuado los plots originales. Por lo que tampoco tendremos tampoco los outputs. Será un ejercicio de prueba y error.
- *Existencia de elementos desconocidos.* Como hemos mencionado en los obstáculos. No tenemos conocimientos amplios de física, ni de como crear un simulador. Vale la pena tener en cuenta que esto puede llegar a retrasarnos.

2.3. Metodología y seguimiento

2.3.1. Metodología del trabajo

Este proyecto es complejo por varios motivos:

- Es un proyecto que no está vinculado únicamente con la computación, gran parte del proyecto está relacionada con la física de partículas. Un Tema que no he tratado al largo de la carrera.
- Gran parte de las cosas a implementar no las he tocado anteriormente o el conocimiento que tengo es muy básico.
- El programa principal y la obtención de los plots lo realizará una sola persona (con ayuda de sus directores), teóricamente tendrá que dedicar 450 h al largo de tres meses y medio.

Este es el primer TFG que hace el autor, nunca a trabajado con un problema parecido. Aparte, para el autor, es muy importante la comunicación con sus directores, para poder ir avanzando en el proyecto. De manera que se necesita una metodología de trabajo que no requiera conocimiento previo de todo lo que se hará y lo permita definir, de manera dinámica y en función del progreso que se haya conseguido anteriormente, las tareas a ejecutar.

Scrum[12] es una metodología de trabajo muy adecuada para proyectos complejos y tiene en cuenta que el equipo de desarrollo no lo sabe todo al comienzo del proyecto, y basándonos en lo expuesto, hemos considerado que es el más adecuado para este proyecto.

2.3.2. Seguimiento

Para verificar que el desenvolvimiento se adapta a los requisitos del proyecto y poder corregir las desviaciones que puedan surgir, realizaremos reuniones semanales, en las que, analizaremos el progreso del programa y de la memoria, las dificultades que tengamos y las posibles mejoras que podamos hacer. También comprobaremos si cumplimos los requisitos establecidos con ayuda de los indicadores.

2.3.3. Planificación temporal

Hemos previsto que el proyecto supondrá 450 horas ($25 \frac{\text{horas}}{\text{ECTS}} * 18 \text{ECTS}$). El proyecto se comenzó a preparar el día 21 de febrero, cuando tuvimos la primera sesión informativa de GEP. Nuestra previsión es acabar el proyecto el domingo 29 de mayo del 2022.

El motivo de acabar el proyecto con unas semanas de antelación es por tener en cuenta los riesgos y obstáculos que podamos tener. Así tendríamos unas semanas de margen para evitar cualquier contratiempo. El turno de lecturas de los TFG de junio de 2022 empiezan el lunes 27 de junio, tendríamos que tener el trabajo acabado y la memoria hecha como muy tarde el lunes 20 de junio[4].

2.3.4. Fases del proyecto

Este proyecto tiene tres partes principales: El desarrollo del programa que realiza la simulación(S), el programa que genera los plots (P) y la elaboración de la documentación. El trabajo de las tres partes se puede ejecutar de manera más o menos paralela una vez el proyecto este planificado. A demás, habrá reuniones semanales de seguimiento(R) al largo del proyecto, que forman parte de la metodología Scrum.

Para las partes que implican escribir código (*S y P*), podemos identificar cuatro fases:

- Estudio previo (EP)
- Diseño (D)
- Implementación (I)
- Pruebas (CF)

La elaboración de la documentación consta de una sola fase que se alargará durante todo el proyecto y únicamente consta de dos tareas.

Las reuniones semanales tampoco se dividirán en más fases. Ya que todas las reuniones serán muy parecidas. Consistirán en analizar lo que se ha hecho durante la semana y planificar la próxima. Aunque habrá algunas reuniones con un enfoque más global, como se explicará más adelante.

2.3.5. Descripción de las tareas

Estudio previo (EP)

- **EP1: Alcance y requerimientos.** Para poder inscribir un TFG, es necesario especificar un título, una descripción y las competencias a desarrollar. Aun así, como no deja de ser un problema complejo, es imprescindible tener unos fundamentos más sólidos del proyecto y enfocar de manera clara que queremos construir que características específicas tiene que tener para qué se adapte a nuestras necesidades.
- **EP2: Planificación temporal.** Una vez ya tenemos nuestro proyecto planteado, es necesario decidir como organizaremos el proyecto en tareas y cuanto tiempo le dedicaremos a cada una.
- **EP3: Presupuesto y análisis de sostenibilidad.** Con la planificación temporal hecha, decidiremos que perfil de trabajadores se encargaría de llevar a cabo el proyecto si se tratase de una empresa, por ejemplo. También se analizarían los costes y la sostenibilidad.
- **EP4: Medura del consumo eléctrico y tiempo de ejecución.** Realizar ecuaciones diferenciales en una simulación puede conllevar un gran costo computacional. Por ello es buena praxis, calcular cuál será el coste aproximado de realizar la simulación, comparándolo con otros estudios parecidos, para poder analizar mejor nuestra implementación una vez realizada.
- **EP5: Estudio del contexto del proyecto.** Necesitamos adquirir los conocimientos necesarios para entender lo que queremos simular y representar.

2.3.6. diseño (D)

- **D1: Diseño del algoritmo del simulador.** Tenemos que establecer como funcionara nuestro simulador. Escoger los algoritmos adecuados para cada tipo de cálculo. Especificar como será la entrada del programa y como será la salida.
- **D2: Diseño del generador de plots.** Determinar que librerías serán las más adecuadas para poder procesar los datos obtenidos con el simulador.
- **D3: Diseño de la interfaz.** Se requiere de una pequeña interfaz o input para poder ejecutar el programa. No es necesario nada muy complejo, pero, es importante que el usuario pueda modificar los parámetros y obtener los resultados que quiera sin tener conocimientos de programación. Para ello es necesario establecer una pequeña entrada y salida, ya sea des de consola o con una interfaz.

2.3.7. Implementación (I)

- **I1: Implementación del código del simulador.** Desarrollar el código que realiza las ecuaciones diferenciales, los sumatorios y todos los cálculos matemáticos.
- **I2: Implementación del generador de plots.** Desarrollar el código encargado de analizar los datos del simulador y generar/guardar plots.
- **I3: Implementación de la interfaz.** Desarrollar el código encargado de leer las peticiones del usuario e informar de cualquier error.

2.3.8. Pruebas (PR)

- **PR1: Pruebas parciales del simulador.** Mientras vamos realizando el simulador, iremos realizando pruebas para comprobar que funciona correctamente y poder corregir posibles errores. También se irá aumentando el nivel de complejidad de las pruebas, por ejemplo, ir aumentando en número de partículas a simular.
- **PR2: Pruebas Finales del simulador.** Una vez consideremos que el programa funciona correctamente, realizaremos las pruebas finales y compararemos los resultados obtenidos con los resultados del artículo *Violent relaxation in quantum fluids with long-range interactions*.
- **PR3: Pruebas del generador de plots.** Mientras vamos haciendo el generador de plots, iremos haciendo pruebas para revisar que funciona correctamente y poder corregir posibles errores.

2.3.9. Documentación (DO)

- **DO1: Documentos de gestión del proyecto.** Crearemos un documento para explicar lo que hemos concluido sobre GEP en lo que se refiere a nuestro trabajo, incluyendo todo lo que se haya observado, con tal de tener una buena base para la memoria.
- **DO2: Memoria.** Esta tarea tiene una dedicación bastante elevada de tiempo. Aun así, la tarea a ejecutar es la misma siempre, por lo que creemos que no tiene sentido dividirla. Es probable que el texto se tenga que modificar o reorganizar según vaya avanzando el proyecto.

2.3.10. Reuniones de seguimiento (R)

- **R1: Reunión inicial.** En esta reunión decidiremos con el director del proyecto los objetivos y requerimientos. También comprobaremos que los obstáculos o los riesgos no sean en realidad impedimentos.
- **R2: Reuniones de seguimiento normales.** Semanalmente nos reuniremos con el director para revisar el progreso del proyecto durante la semana anterior y planificar lo que se tendrá que hacer la semana próxima. También miraremos de establecer las medidas necesarias si se llega a detectar alguna desviación.
- **R3: Reunión intermedia.** En Nuestra facultad es necesario realizar una reunión entre el director y el alumno para evaluar si se han logrado los objetivos del proyecto, que tendrían que estar ya realizados, a mitad del trabajo.
- **R4: Reunión final.** Un par de semanas antes de la entrega de la memoria, nos reuniremos con el director para: validar que los objetivos y requerimientos se han cumplido, resolver los posibles errores y prepara la presentación oral ante el tribunal.

2.4. Tabla de Tareas

Mostramos la tabla resumen de tareas en la figura 2. Como se verá más adelante, los nombres de los perfiles profesionales provienen del convenio sectorial aplicable hasta el 2018 (*Resolución de 18 de marzo de 2009, de la Dirección General de Trabajo, por la que se registra y publica el XVI Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública 2009, March 18*).

Id.	Tarea	Horas	Perfil
EP1	Alcance y requerimientos.	17	Director del proyecto
EP2	Planificación temporal.	17	Director del proyecto
EP3	Presupuesto y análisis de sostenibilidad.	17	Director del proyecto
EP4	Mesura del consumo eléctrico y tiempo de ejecución.	10	Programador
EP5	Estudio del contexto del proyecto	10	Analista
D1	Diseño del algoritmo del simulador.	15	Analista
D2	Diseño del generador de plots.	11	Analista
D3	Diseño de la interfaz	10	Analista
I1	Implementación del código del simulador.	50	Programador
I2	Implementación del generador de plots.	20	Programador
I3	Implementación de la interfaz.	15	Programador
PR1	Pruebas parciales del simulador.	50	Analista
PR2	Pruebas Finales del simulador	60	Analista
PR3	Pruebas del generador de plots	10	Analista
DO1	Documentos de gestión del proyecto	30	Analista
DO2	Memoria	80	Analista
R1	Reunión inicial.	2	Director del proyecto
R2	Reuniones de seguimiento normales.	22	Director del proyecto
R3	Reunión intermedia.	2	Director del proyecto
R4	Reunión final.	2	Director del proyecto

Figura 2: Resumen de las tareas

- Hemos valorado que, debido a que la memoria es un testimonio muy relevante del trabajo y requiere un trabajo especial, el analista le dedicara 6 horas semanales.
- Por lo que respecta a las reuniones, consideramos que basta con una reunión semanal de máximo dos horas, aplicable a los tres tipos de reuniones mencionados anteriormente.

2.7. Diagrama de Gantt

El diagrama de Gantt del proyecto se puede ver en las figuras 4 y 5. Las relaciones de dependencia final-Inicio se muestran marcadas mediante líneas continuas negras y azules (las azules son debido a que la tarea no se realiza justo después de acabar la anterior). Las tareas inicio-inicio no aparecen en el diagrama debido a que el software utilizado no los soporta.

En el diagrama no aparecen ni los objetivos principales ni los indicadores por dos motivos. Primero, hay muchos objetivos y subobjetivos y resultaría confuso. Aparte, no sabemos que objetivos/subobjetivos resolveremos antes, se irá trabajando al largo de la implementación y las reuniones con el director.

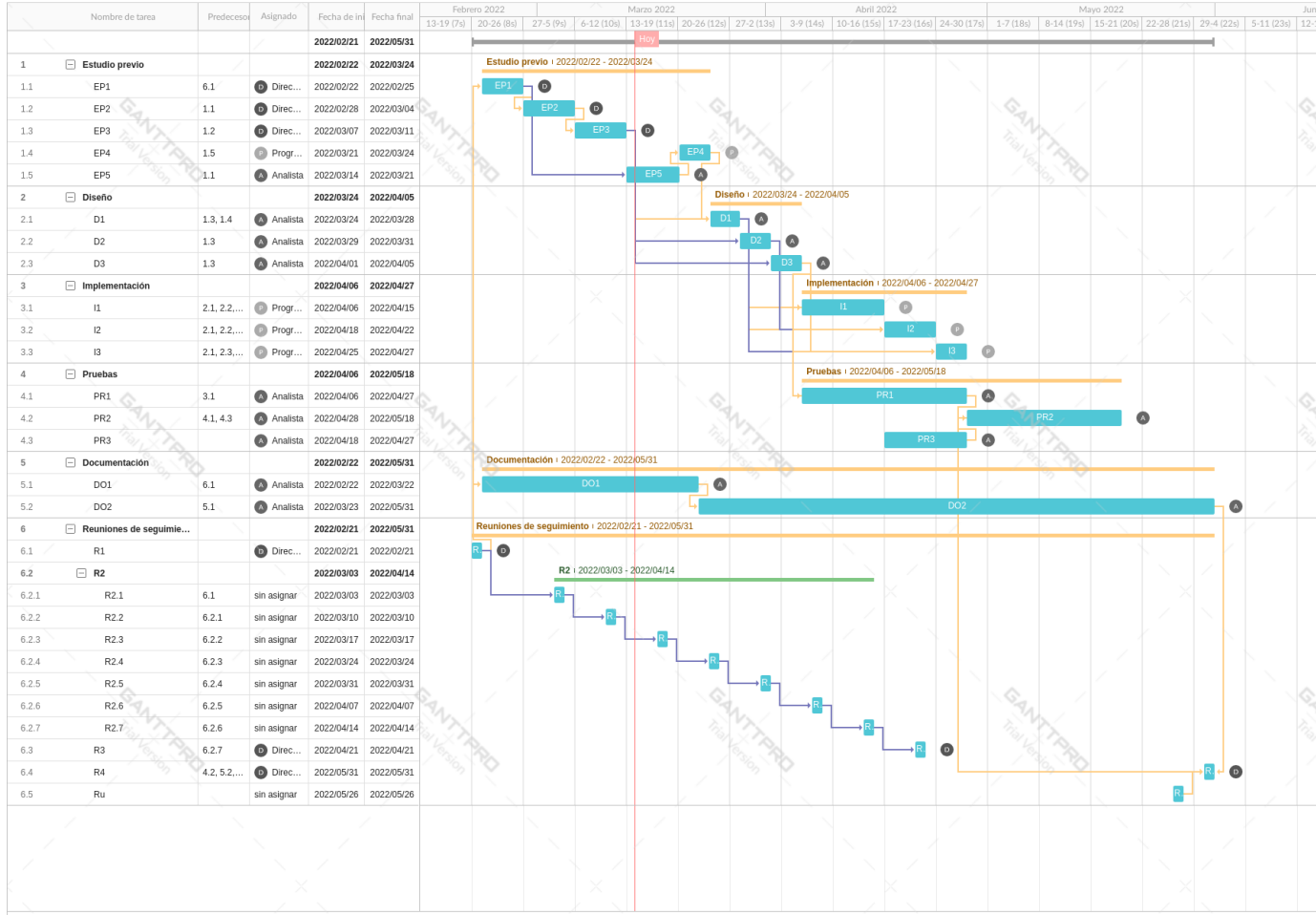


Figura 4: Diagrama de Gantt

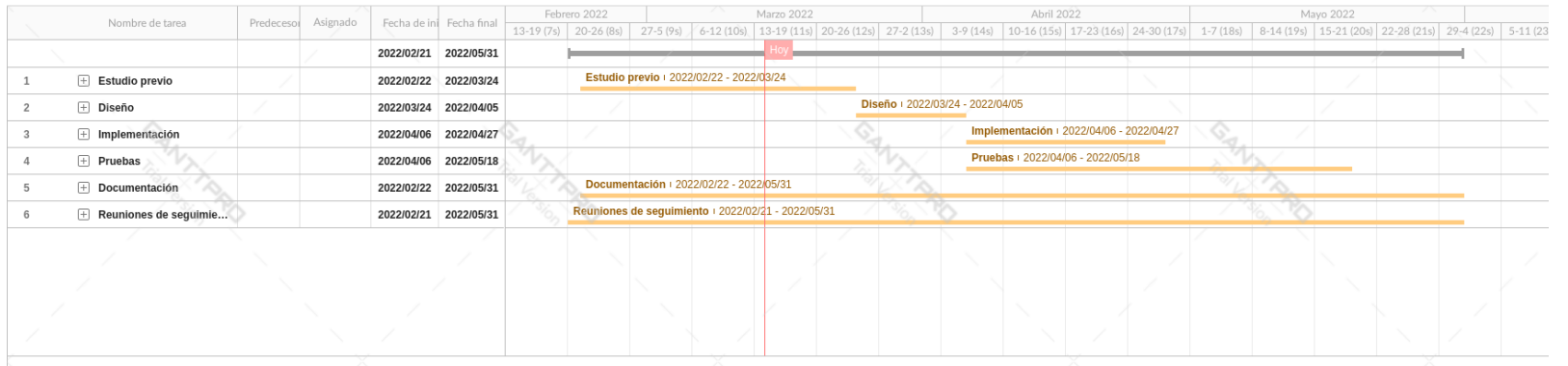


Figura 5: Diagrama de Gantt simplificado

2.8. Personal y material

Por lo que respecta al personal, el proyecto necesita dos directores, una analista y un programador. En concreto:

- Los *directores del proyecto* se encargan de establecer el alcance y las características del proyecto, como las diferentes tareas a realizar (*simular o representar*). También coordinarán los miembros del equipo y se reunirán con el programador y el analista para poder avanzar con el proyecto.
- El *analista* se ocupará de analizar el programa y los resultados obtenidos. En realidad será una tarea del director y el programador unidos.
- El *programador* tiene como función principal escribir el código necesario para poder realizar las simulaciones y obtener los plots.

El material que necesita el proyecto será:

- Tres ordenadores, para los dos directores y el programador.
- Un espacio de trabajo para los trabajadores con internet.

2.9. Gestión de riesgos

Como hemos explicado en el punto 2.2.5, prevemos cuatro tipos de riesgos en nuestro proyecto: picos de trabajo relacionados con otras actividades ajenas al TFG, la existencia de elementos desconocidos; dado que es un campo de la computación que no hemos tratado demasiado durante la carrera, accidentes y que los resultados no correspondan con otros resultados de otros experimentos.

Por lo que respecta a los picos de trabajo y los accidentes, que pueden aparecer al largo del desarrollo del proyecto, hemos previsto dos estrategias:

- *Dedicación adicional*. Si una semana no podemos dedicar las suficientes horas al proyecto, durante las siguientes semanas (como antes, mejor!!!) le dedicaremos tiempo extra, para lograr ponernos al día.
- *Fines de semana*. En caso de llevar retraso, se tendrá que dar prioridad al TFG los fines de semana e intentar hacer el trabajo que no se pudo realizar en la semana. También darle prioridad en festivos o en momentos en que la carga de trabajo sea menor.

Para evitar estancarnos frente a nuevos problemas, es importante tener el apoyo del director para que nos oriente. También preguntar a otros profesores si en algún momento no sabemos que hacer y el director no nos puede ayudar.

Por lo que respecta al último riesgo, la posibilidad que nuestros resultados no se correspondan con los de otros científicos, después de la reunión final todavía hay un margen de tiempo para

resolver cualquier problema. Aun así, con las pruebas hechas durante el proyecto, creemos que es muy difícil que esto suceda.

2.10. Presupuesto

2.10.1. Identificación de costes

Consideramos que los costes del proyecto se pueden clasificar en las categorías siguientes:

- *Personal*: Director del proyecto, analista y programador.
- *Desarrollo*: Espacio y ordenadores personales para los trabajadores.
- *Energía*: Consumo eléctrico de los ordenadores.

2.11. Coste de personal

Como hemos mencionado en el punto 2.8, el proyecto requiere de un director, una analista y un programador. Al sector de la informática le es aplicable el *Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de opinión pública*”.

Hemos utilizado el XVI convenio para nombrar las categorías profesionales (*Resolución de 18 de marzo de 2009, de la Dirección General de Trabajo, por la que se registra y publica el XVI Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública 2009, March 18, art. 15*), con tal de encontrar los puestos de trabajo del analista y el programador.

Igualmente, al momento de calcular los salarios hemos aplicado los criterios generales del XVII convenio (*Resolución de 22 de febrero de 2018, de la Dirección General de Empleo, por la que se registra y publica el XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública 2018, February, 22, art. 15*).

También, hemos tenido en cuenta la jornada máxima anual de 1800 horas (*Resolución de 22 de febrero de 2018, de la Dirección General de Empleo, por la que se registra y publica el XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública 2018, February 22, art. 20.1*).

Perfil	Grupo	Nivel	Salario	Cotización	Coste/año	Coste/hora	Horas	Coste
Dir. del proyecto	A	-	26790,31	8010,30	34800,61	19,33	79	1527,36
Analista	B	1	25986,59	7769,99	33756,58	18,75	276	5176,01
Programador	C	1	24640,37	7367,47	32007,84	17,78	95	1689,30
Total							450	8392,67

Figura 6: Costes salariales del proyecto

Tarea	Salario para ...	Horas	Precio/hora	Precio
EP1	Director del proyecto	17	19,33	328,67
EP2	Director del proyecto	17	19,333	328,67
EP3	Director del proyecto	17	19,33	328,67
EP4	Programador	10	17,78	177,82
EP5	Analista	10	18,75	187,54
D1	Analista	15	18,75	281,30
D2	Analista	11	18,75	206,29
D3	Analista	10	18,75	187,54
I1	Programador	50	17,78	889,11
I2	Programador	20	17,78	355,64
I3	Programador	15	17,78	266,73
PR1	Analista	50	18,75	937,68
PR2	Analista	60	18,75	1125,22
PR3	Analista	10	18,75	187,54
DO1	Analista	30	18,75	562,61
DO2	Analista	80	18,75	1500,29
R1	Director del proyecto	2	19,33	38,67
R2	Director del proyecto	22	19,33	425,34
R3	Director del proyecto	2	19,33	38,67
R4	Director del proyecto	2	19,33	38,67
	Total			8392,67

Figura 7: Costes salariales del proyecto por tareas

Como los salarios están entre las bases de cotización mínima y máxima del grupo 2 (ingenieros técnicos) (*Seguridad Social: Bases y tipos de cotización 2019 n.d.*), los trabajadores cotizan en base a su salario. Hemos aplicado los límites y tipos de cotización del año 2019, que suman un total de un 29'9%. Para simplificar los cálculos hemos supuesto que los trabajadores trabajan a jornada completa, y el resto del tiempo lo dedican a otros proyectos.

Con todo lo expuesto, hemos calculado los costes salariales de los tres trabajadores en la figura 6, mientras que, en la figura 7 se puede ver a nivel de tarea.

2.11.1. Costes de desarrollo

Los siguientes costes son calculados de manera genérica, ya que no dependen de las tareas del proyecto (pero intervienen en todas). Consideramos como costes de desarrollo las amortizaciones, el espacio de trabajo y el internet.

Amortizaciones

- **Hardware.** Durante el proyecto se necesitarán dos portátiles de gama media-baja de un coste de 500€ (uno para el programador y otro para el director-analista). Suponiendo que de media el tiempo de vida útil es de 4 años (48 meses) tendremos una amortización de $1000 * \frac{450\text{hours}}{4\text{years} * 220 \frac{\text{days}}{\text{years}} * 4 \frac{\text{hours}}{\text{days}}} = 127,84$.
- **Software.** Todos los programas que utilizaremos serán gratuitos. Por lo que la amortización será de 0€.

Espacio de trabajo El proyecto se desarrollará de manera online, aun así, si es posible, se harán reuniones presenciales y se dispondrá de un despacho común. Los despachos, aunque son gratuitos, se podría valorar en unos 300€ mensuales. En el grupo de investigación seríamos tres personas compartiendo el despacho, por lo que el coste se podría concretar en $300 * 3,5\text{meses} / 3\text{personas} = 350$.

Internet Las facturas de internet son unos 50€ mensuales. Con una jornada laboral media de 4 horas diarias durante 3 meses y medio, la parte proporcional de la factura de internet sube a $5 * 50 * 4 / 24 = 29,17$.

Concepto	Coste
Hardware	127,84€
Software	0€
Espacio de trabajo	350€
Internet	29,17€
Total CG	507,01

Figura 8: Estimación de los costes genéricos. Elaboración propia.

2.11.2. Coste de Energía

Después de comprobar el precio actual de la electricidad, que es de 0.358€/kwh[10], podemos estimar el coste de mantener en funcionamiento los dispositivos mencionados previamente. El coste sería $50w * 450horas * 0,358/1000 = 8,06$.

2.12. Contingencias

Es necesario contemplar la posibilidad de que sucedan contratiempos y complicaciones durante el proyecto, los cuales pueden hacer que nos quedemos cortos con nuestro presupuesto. Por este motivo, preparamos una medida de contingencia con tal de poder afrontar cualquier tipo de situación. Los valores típicos a nivel de contingencia son de entre el 10 y el 20 por ciento. Debido, como ya hemos comentado antes, a que utilizamos el método SCRUM y que iremos adaptándonos y resolviendo los problemas a través de reuniones, pensamos que un valor del 15 por ciento es adecuado.

Calcularemos el coste de la siguiente manera: $(Totalcostepersonal + Totalcostedesarrollo + Costeenergia) * 0,15$. Por tanto, el coste será de $(8392,67 + 507,01 + 8,06) * 0,15 = 1336,17€$

2.13. Imprevistos

Para acabar, es necesario valorar el coste de implementar los planes alternativos en caso de enfrentar un imprevisto/obstáculo. El coste de cada uno se calcula con el coste que supondría multiplicado por la probabilidad de que suceda.

En el punto 2.2.5 hemos predicho cuatro riesgos:

- **Picos altos de Trabajo.** Este riesgo no tendría ningún coste económico debido a que las horas de trabajo no se verían aumentadas solo aplazadas. Su probabilidad es alta, pero su coste es 0 (Económico). Esto no significa que no sea un riesgo importante y que no se tenga que tener en cuenta.
- **Existencia de elementos desconocidos.** Este caso, por otra parte, conlleva un aumento en las horas de trabajo. Como hemos mencionado anteriormente, el número de horas que le dedicaremos al TFG será de 450. Si esto llega a suceder se tendría que dedicar una 15h extras de trabajo(aprox.).
- **No correspondencia de los resultados.** Misma situación que en el caso anterior, se tendría que dedicar tiempo extra a resolver los errores. 10 horas (aprox.).
- **Accidentes** Misma situación que en el primer caso. Tendríamos que recuperar las horas perdidas.

Imprevisto	Perdida	Probabilidad	Coste
Elementos desconocidos (20h)	375,07	20%	75,01
No correspondencia de resultados (10h)	187,54	10%	18,75
Total imprevistos			75,01

Figura 9: Estimación de los imprevistos. Elaboración propia.

2.14. Coste total del proyecto

En la figura 10 se muestra el coste total del proyecto, sumando los costes calculados en las secciones anteriores.

Concepto	Coste
Coste personal	8392,67
Coste de desarrollo	507,01
Coste energia	8,06
Contingencias	1336,16
Imprevistos	75,01
Total	10318,91

Figura 10: Estimación del coste total del proyecto. Elaboración propia.

2.15. Control de gestión

Durante el transcurso del proyecto se irán haciendo revisiones de los costes para asegurar que no nos desviemos de los costes estimados. Para ello se utilizarán las siguientes métricas:

$$\text{Desviación de coste} = (\text{CE} - \text{CR}) \times \text{CHR}$$

$$\text{Desviación de consumo} = (\text{CHE} - \text{CHR}) \times \text{CE}$$

con:

- **CE:** Coste estimado
- **CR:** Coste real
- **CHR:** Consumo de horas real **CHE:** Consumo de horas estimado

Gracias a estos controles podremos detectar cualquier desviación y saber en qué tarea se ha producido y de qué desviación se trata. Si las desviaciones son debidas a imprevistos mencionados, la partida de imprevistos cubrirá los costes. En caso contrario, se encargará la partida de contingencia. Si los costes son muy elevados, se podrá usar la partida de contingencia o imprevistos, aunque no corresponda usar esa de primeras. Si aun así, las desviaciones siguen siendo demasiado grandes, se tendrá que reorganizar el alcance del proyecto y ceñirse al presupuesto.

También se intentará evitar utilizar recursos físicos, como por ejemplo papel, a favor de los recursos digitales. Intentaremos que sea un proyecto respetuoso con el medio ambiente.

2.16. Sostenibilidad

2.16.1. Reflexión

Después de haber contestado la encuesta dirigida a universitarios por parte de EDINSOST[1]. Puedo llegar a hacerme a la idea del nivel de conocimiento que tengo de las TIC a nivel de la sostenibilidad. Creo que parto de una buena base. Desde muy pequeño, en la primaria; bachiller, ya nos han enseñado a tener consciencia de ello. Durante toda la carrera, también nos han mostrado los pros y los contras de la informática. Todo lo que puede aportar y todos los costes o problemas que pueden llegar a provocar. Por otra parte, a nivel práctico no pienso que tenga tanta experiencia. Este es el primer proyecto donde tomo todos estos valores más en cuenta y tengo que hacer un análisis de él.

Por otra parte, debido a mi especialidad (Computación), más enfocada a la investigación; y el tema del proyecto, donde no intervienen materias primeras o el objetivo no es un producto en sí, con costes de mantenimiento; mi capacidad para realizar acciones que marquen un gran impacto en la sostenibilidad ya están muy limitadas comparada con otros casos.

2.16.2. Dimensión económica

En relación a la dimensión económica del proyecto, podemos sacar las conclusiones que comentaremos a continuación:

- En apartados anteriores se han tenido en cuenta los *diferentes cosas* del desarrollo: ordenadores, sala de trabajo, personal, ...
- No hay gastos de mantenimiento. Como hemos comentado en la reflexión. Nuestro proyecto es de investigación, y una vez tengamos el programa, los costes posibles solo serán de ejecución, modificación o mejora. Una vez nuestro equipo tenga realizados los cálculos, no habrá ningún coste más. Aunque puede servir de base para otro proyecto del mismo ámbito.
- Los costes del proyecto ya son muy reducidos. Utilizamos un ordenador portátil, en lugar de uno de sobremesa, porque cuesta menos y consume menos. El sueldo de los trabajadores es el adecuado según convenio y los precios de la luz o alquiler, no dependen de nosotros.
- El tiempo que le dedicaremos a cada tarea se basa en lo que puede aportar al proyecto. Tampoco podemos usar más herramientas externas aparte de alguna librería .“open-source” porque el código que desarrollaremos es muy específico.
- Este proyecto es una contribución a DFIS[2].

Dimensión ambiental En relación con la *dimensión ambiental* del proyecto, concluimos lo siguiente:

- Los *recursos* que necesitamos en el proyecto están indicados más arriba. No extraemos recursos naturales, ni emitimos substancias al exterior.
- La *actividad* que vamos a realizar es desarrollar un programa, no es particularmente muy intensivo y tiene un impacto muy reducido en el medio ambiente. Nuestro programa es bastante específico, aunque se intentará aprovechar recursos y librerías de terceros.
- Nuestro proyecto se puede aprovechar en futuros proyectos. Como intentaremos hacer un código fácil de interpretar y utilizar, el coste de hacer modificaciones o mejoras también será muy pequeño. Como lo compartiremos, sí alguien necesita hacer estos cálculos, podrá aprovechar nuestro código y gastar menos tiempo y energía.

2.16.3. Dimensión social

A nivel personal creo que el proyecto me puede aportar muchas cosas. En primer lugar, la temática del proyecto es un poco ajena a lo que he ido estudiando en la carrera. Hay la parte de física, en la que podré adquirir ciertos conocimientos, que de primera vista, ya me parecen muy interesantes. También la parte informática, con muchas cosas que no conozco a priori, pero que me parecen muy interesantes y útiles si acabo realizando más experimentos de este tipo.

Actualmente, en el problema que queremos abordar ya existe un programa que resuelvo los cálculos. Pero no tenemos acceso a él. Una vez tengamos nuestro programa, lo compartiremos con la comunidad científica y podremos avanzar en conjunto. A nivel social, no existe una necesidad real para resolver el proyecto. Pero de manera indirecta estaremos ayudando a diferentes investigadores alrededor del mundo y en algún momento se podría llegar a resolver una necesidad real.

3. Análisis

3.1. Modelo utilizado

El objetivo de este trabajo de fin de grado es realizar una simulación clásica de las dinámicas en un sistema unidimensional clásico con diferentes tipos de interacciones. Como hemos mencionado, la *Violent relaxation* es un proceso que ocurre en un sistema clásico tanto de interacción blanda como fuerte. Las más analizadas en el artículo son la soft-core (núcleo blando), pero también trataremos de analizar algunas hard-core. Este proceso tiene la característica particular de amplificar dramáticamente pequeñas perturbaciones, llevando a un rápido colapso en configuraciones de evolución lenta conocidas como cuasiestacionarias. Este comportamiento tan particular se puede observar en la figura 1. Nuestro objetivo es simular este proceso y verificar si los resultados del artículo[9] son correctos.

Antes de resolver las derivadas parciales haremos un pequeño repaso de mecánica Hamiltoniana.

La mecánica hamiltoniana es una reformulación de la mecánica clásica que se introdujo en 1833 por el matemático irlandés William Rowan Hamilton. Surgió a partir de la mecánica lagrangiana, una reformulación de la anterior mecánica clásica introducida por Joseph Louis Lagrange en 1788, pero puede ser formulada sin recurrir a la mecánica lagrangiana con espacios simpléctica.

Al igual que con la mecánica de Lagrange, Hamilton proporciona un equivalente y nueva forma de ver la mecánica clásica. En general, estas ecuaciones no proporcionan una manera más conveniente de resolver un problema particular.

Por el contrario, proporcionan una visión más profunda tanto de la estructura general de la mecánica clásica y su relación con la mecánica cuántica, tal como se entiende a través de la mecánica hamiltoniana, así como su conexión con otras áreas de la ciencia.

*El valor del Hamiltoniano es la .energía"total del sistema que se describe. Para un sistema cerrado, es la suma de la cinética y energía potencial en el sistema. Hay un conjunto de ecuaciones diferenciales conocido como las ecuaciones de Hamilton que dan la evolución temporal del sistema. Hamiltonianos se puede utilizar para describir tales sistemas simples como una pelota que rebota, un péndulo o un resorte oscilante en el que los cambios de energía cinética a potencial y de regreso otra vez. Hamiltonianos también puede ser empleado para modelar la energía de otros sistemas dinámicos más complejos, tales como las órbitas planetarias en mecánica celeste, y también en la **mecánica cuántica**. [5]*

Necesitamos un modelo y un comportamiento (fórmulas) con las que podamos representar el comportamiento de las partículas. En este caso el modelo es **HMF**, (campo medio hamiltoniano), que sigue las fórmulas de las ecuaciones 1. A pesar de su nombre, proporciona una descripción exacta de un sistema de muchos cuerpos en una dimensión. Definido en un anillo de radio R , describe N partículas que interactúan a través de un potencial por pares que varía como $\cos(\theta_i - \theta_j)$

Hemos decidido calcular las derivadas a mano y luego realizar un programa que calcule la posición de cada partícula en cada momento con el fin de poderlo representar en una figura igual a la del artículo *Violent relaxation in quantum fluids with long-range interactions* [9].

Antes de calcular la derivada, indicaremos a que corresponde cada elemento de la fórmula para hacernos una idea de que estamos calculando.

$$H = \sum_i \frac{L_i^2}{2mR^2} + \frac{\epsilon}{N} \sum_{i < j} \cos(\Theta_i - \Theta_j) \quad (3)$$

- L es el momento angular. L_i es el momento angular de la partícula i .
- m es la masa.
- R es el radio del anillo.
- ϵ indica si la fuerza es atractiva (-1) o repulsiva (1).
- N es el número de partículas.
- θ es la posición de la partícula. La posición estará acotada a 2π , $[0; 2\pi]$ o bien $[-\pi; \pi]$

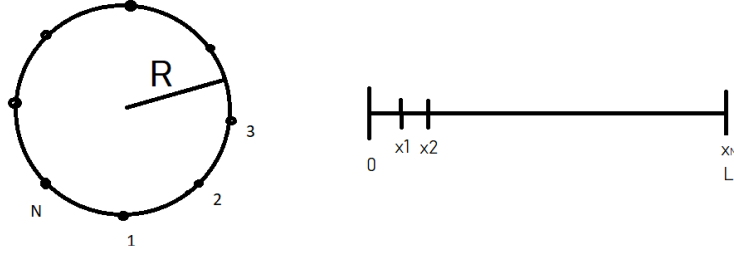


Figura 11: Ejemplo de anillo. Elaboración propia.

La figura representa un anillo de radio R con N partículas repartidas de manera equidistante. El perímetro del anillo tiene una longitud L o $(2\pi R)$. Si escogamos un punto del anillo como punto 0 y le sumamos L , llegaremos al mismo punto. Tomando esto como base podemos representar la posición de una partícula i como x_i . Así mismo podemos representar la posición de una partícula a través de su ángulo(θ) utilizando una simple conversión. La figura de la izquierda representa las partículas repartidas en el anillo. La figura de la derecha representa las partículas en una línea recta de tamaño L o $(2\pi R)$.

Seguidamente, procederemos al cálculo de las derivadas parciales necesarias para el programa:

Como sabemos, el momento angular sigue la siguiente fórmula: $L_i = mv_i R$, por tanto:

$$\sum_i \frac{L_i^2}{2mR^2} = m \frac{R^2}{R^2} \sum_i \frac{v_i^2}{2} = \frac{m}{2} \sum_i v_i^2 \quad (4)$$

La siguiente fórmula nos permite pasar de una posición x_i a un ángulo θ_i o a la inversa, para entenderlo mejor se puede ver la representación de la figura 11:

$$\begin{aligned} x_i &= \frac{\theta_i}{2\pi} L \\ \theta_i &= \frac{2\pi}{L} x_i \end{aligned} \quad (5)$$

Con los cálculos previos podemos reformular el hamiltoniano:

$$H = \frac{m}{2} \sum_i v_i^2 + \frac{\varepsilon}{N} \sum_i \cos \frac{2\pi}{L} (x_i - x_j) \quad (6)$$

Recordemos que el hamiltoniano representa las energías en un sistema cerrado, en este caso energía cinética $V = \frac{m}{2} \sum_i v_i^2$ y la energía potencial $P = \frac{\varepsilon}{N} \sum_i \cos \frac{2\pi}{L} (x_i - x_j)$. Sabiendo que la derivada de la velocidad es la aceleración, conociendo la fórmula de la fuerza y poniendo la velocidad en función del momento $v_i = \frac{L_i}{mR}$, tenemos que:

$$\begin{aligned} F_i &= m \cdot a_i = m \frac{L_i}{mR} = - \frac{\partial}{\partial x_i} E_{pot} \\ &= + \frac{\varepsilon}{N} \sum_{j \neq i} \frac{2\pi}{L} \sin \frac{2\pi}{L} (x_i - x_j) \\ &\implies \\ m \frac{1}{mR} \frac{dL_i}{dt} &= \frac{2\pi\varepsilon}{NL} \sum_{i \neq j} \sin(v_i - v_j) \\ \frac{dL_i}{dt} &= \frac{2\pi R}{L} \frac{\varepsilon}{N} \sum_{i \neq j} \sin(v_i - v_j) \end{aligned} \quad (7)$$

Substituyendo las posiciones x_i por θ_i y sabiendo que la derivada de la posición es la velocidad, logramos llegar a:

$$\begin{aligned} x_i &= \frac{L}{2\pi} \theta_i \\ \frac{dx_i}{dt} &= v_i \\ \frac{L}{2\pi} \frac{d\theta_i}{dt} &= v_i = \frac{L_i}{mR} \\ \frac{d\theta_i}{dt} &= 2\pi \frac{1}{mRL} L_i = \frac{2\pi}{mR} \frac{L_i}{2\pi R} = \frac{L_i}{mR^2} \\ &\frac{d\theta_i}{dt} = \frac{L_i}{mR^2} \end{aligned} \quad (8)$$

Finalmente, ya tenemos las dos ecuaciones diferenciales que necesitamos para representar las partículas que seguirán el **HMF**:

$$\begin{aligned} \frac{dL_i}{dt} &= \frac{\varepsilon}{N} \sum_{i \neq j} \sin(\theta_i - \theta_j) \\ \frac{dv_i}{dt} &= \frac{L_i}{mR^2} \end{aligned} \tag{9}$$

Para realizar los cálculos, asignamos los valores $m = 1$ y $R = 1$.

3.2. Método numérico

Para realizar los cálculos hemos optado por realizarlos nosotros mismos y no utilizar librerías de terceros, gracias a que las derivadas parciales ya las resolvimos a mano (apartado 3.1). Las únicas librerías que hemos llegado a usar son las necesarias para representar los datos: Matplotlib y NumPy y las necesarias para efectuar cálculos y controlar los tiempos: math y time.

El método que hemos seguido es partir de un caso inicial e ir calculando el ángulo y el momento de cada partícula al reaccionar por la fuerza de las otras partículas.

Los dos posibles casos son los siguientes:

Caso inicial:

$$\begin{aligned}\theta_i^{n=0} &= \frac{2\pi i}{N}, i = 1, \dots, N \\ L_i^{n=0} &= 0,005 * \cos(\theta_i^{n=0})\end{aligned}\tag{10}$$

Resto de casos:

$$\begin{aligned}\theta_i^{n+1} &= \theta_i^n + L_i^n * \Delta t, \Delta = 0,01 \\ L_i^{n+1} &= L_i^n + \Delta t * \frac{\epsilon}{N} \sum_{i \neq j} \sin(\theta_i - \theta_j)\end{aligned}\tag{11}$$

Significado de cada campo:

- **n** corresponde al número de la iteración. El caso inicial es la iteración cero, para calcular cualquier iteración necesitamos los datos de la iteración anterior.
- **i** corresponde a una partícula en concreto.
- **N** es el número total de partículas.
- ϵ representa la fuerza que se produce, $\epsilon = -1$ atractiva y $\epsilon = 1$ repulsiva.

Para poder realizar la simulación hay ciertos parámetros a los que les asignaremos un valor que no será cambiado, independientemente del caso. Buscando los campos restantes, se puede ver que son la masa(m) y el radio(R) que pasarán a tener valor 1, como ya hemos mencionado.

4. Desarrollo

4.1. Implementación

Como hemos mencionado en apartados anteriores, el programa se puede dividir en tres partes bien diferenciadas: La lectura de la petición, el código que ejecuta la simulación principal y la representación de los resultados. Cada una de estas partes se han realizado en paralelo, ya que son necesarias las unas con las otras. Por lo que cada parte ha ido sufriendo cambios y ha acabado al mismo tiempo.

4.1.1. Lectura

Para la lectura, al principio, creamos variables globales que nos permitían modificar los valores de manera fácil y segura. La idea era después generar un documento donde el usuario pudiese escribir los valores que quería meter al programa, pero, durante la implementación de dicha función de lectura, nos dimos cuenta de que para el usuario es más fácil editar las variables globales directamente. Sobre todo, porque la información que modificará será exactamente la misma. Es más, generar una función de lectura sería ineficiente y redundante. Es más fácil dejar al usuario una pequeña guía con la funcionalidad de cada atributo y los posibles valores a escoger. También marcarle la zona donde se encuentran los datos que se pueden modificar y poco más. Como los usuarios son científicos, estarán acostumbrados a Python y a entornos como anaconda o Spyder, muy fáciles de utilizar. Donde solo tendrían que modificar los valores y darle al Play. En caso de no tener esta opción, únicamente tendrían que ejecutar el programa desde una terminal con el comando "python3 tfg1.py" en el directorio donde se encontrase el programa. Es importante tener todas las librerías necesarias instaladas. Para instalarlas solamente se tiene que escribir en una terminal "pip nombreLibreria".

La parte de implementación tendrá que tratar estas variables y no modificarlas. Las variables que hemos generado son:

- **E** representa ϵ .
- **N** es el número de partículas.
- **vart** representa Δt , la variación de tiempo entre cada iteración.
- **pointN** es el número de iteraciones total. Para cada partícula se calcularán pointN puntos.
- **sizeX** representa la anchura de la gráfica resultante.
- **sizeY** representa la altura de la gráfica resultante.
- **sizeP** el tamaño de cada partícula en el gráfico en cada iteración.
- **title** modifica el título del plot.
- **titleX** modifica el título del eje x.

- **titleY** modifica el título del eje y.
- **sizeText** modifica el tamaño de la letra en el plot.
- **P** modifica el tamaño de las partículas en el plot.
- **tipo** Indica que tipo de interacción se quiere simular: sinus, potencial, utilizando chord,...
- **typePlot** Indica el tipo de gráfica que se quiere plotear. Una gráfica tiempo/ángulo, una gráfica que muestre la energía, ...
- **Ns** En el caso que typePlot = "timeN" indica los diferentes números de partículas que se quieren analizar. Ordenados
- **pointNs** En el caso que typePlot = "timepointN" indica los Diferentes números de iteraciones que se quieren analizar. Ordenados
- **intervals** Indica el intervalo que se quiere plotear, si se encuentra vacío se plotea de principio a fin.
- **load** Indica si se desea leer posiciones.txt y momentos.txt para continuar desde un punto de esa simulación.
- **save** Indica si se quieren guardar los datos en posiciones.txt y momentos.txt.
- **startIteration** Si se leen datos, aquí se indica el número de la iteración en la que se quiere empezar.

4.1.2. Simulador

Es la parte principal del programa y donde radica toda la complejidad. En sí, la intención es que el código sea corto y fácil de comprender, pero, que aun así, sea lo suficientemente rápido.

Al largo del tiempo hemos ido modificando el código y reparando errores y dificultades que nos hemos encontrado por el camino. Uno de los principales problemas a tratar es la representación de los datos. Para ello hemos optado por matrices donde cada fila corresponde a una partícula y el número de columnas es el número de iteraciones que realiza el programa.

Al final acabamos con dos algoritmos operativos, uno lento(el primero) y otro rápido. Analizaremos las dos, pero la que avanzaremos y juntaremos con la parte de lectura y representación final será la segunda alternativa.

Primera alternativa

```
def calprimeros():
    for i in range(52):
        primerospos(i)
        primerosmoment(i)

def simulacion():
    for i in range(1,1000):
        for e in range(52):
            vi = math.fmod(pos[e][i-1] + moment[e][i-1] * vart, 2*pi)
            if vi < 0:
                vi = 2*pi + vi
            pos[e].append(vi)
            li = moment[e][i-1] + vart * (E/N) * sumpsin(e)
            moment[e].append(li)

start = time.perf_counter()

calprimeros()
simulacion()

end = time.perf_counter()
print(end-start)
```

Figura 12: Parte principal del simulador. Elaboración propia.

```
pos = []
moment = []

def v(i):
    return ((2*pi*i)/N)

def primerospos(i):
    pos.append([v(i+1)])

def primerosmoment(i):
    moment.append([(0.005 * sin(pos[i][0]))])

def sumpsin(vi):
    aux = 0
    size = len(moment[51])
    aux2 = pos[vi][size-1]
    for i in range(52):
        if (vi != i):
            aux += sin(aux2 - pos[i][size-1])
    time2 = time.perf_counter()
    return aux
```

Figura 13: Funciones auxiliares del simulador. Elaboración propia.

Decidí crear las matrices a partir de listas vacías, ya que lo compare con alternativas de otras librerías, como por ejemplo NumPy, y las listas resultan más rápidas, aunque no te dan tantas opciones. Las Matrices *pos* y *moment* corresponden a las matrices que guardan los datos de θ_i^n y L_i^n , en ese orden.

El algoritmo es bastante simple. Primero calculamos el caso inicial explicado en el apartado anterior. Para ello nos ayudamos de tres funciones auxiliares:

- `v(i)`. Calcula $\theta_i^{n=0}$.
- `primerosmoment(i)`. Calcula $L_i^{n=0}$
- `primerospos(i)`. Función inmersiva.

La resta de cálculos se llevan a cabo en la función `simulación()`. Esta función consiste en ejecutar dos bucles. El primero es el número de iteraciones que ejecutaremos menos una y el segundo es el cálculo de los datos para cada partícula. Allí se calcula el **resto de casos** explicado en la sección 3.2. Para efectuar los cálculos nos ayudamos de la librería `math`, sobre todo para calcular (recordemos que las posiciones están entre 0 y 2π) los módulos. Los resultados los guardamos en las dos matrices `pos` i `moment`. Para simplificar el código creamos otra función llamada `sumsin(vi)`.

`sumsin(vi)` realiza el cálculo $\sum_{i \neq j} \sin(\theta_i - \theta_j)$ para θ_i . Realicé una mejora aplicando programación dinámica, (guardando los senos ya calculados para evitar repetir operaciones). El tiempo se redujo a casi la mitad, pero aun así, eran tiempos muy altos.

El coste de esta alternativa depende principalmente de dos valores: `pointN` y `N` dentro de la función `simulación` con su bucle triple. Por lo que el coste resultante es $\Theta = N^2 * pointN$. Debemos ser precavidos con `pointN`, ya que puede llegar a tener valores muy altos.

Con esta alternativa, si utilizamos los valores `N = 52` y `pointN = 1000`, tenemos tiempos medios de 28 minutos. Aplicando programación dinámica logramos reducir su tiempo a 18 minutos, un 36%. Aun así, es mucho tiempo, por lo que optamos por enfocar el problema por otro camino, vectorizar.

Segunda alternativa

```
pos = np.empty((N,pointN), dtype = float)
moment = np.empty((N,pointN), dtype = float)
sinus = np.zeros((N,N), dtype = float)
```

Figura 14: Matrices alternativa 2. Elaboración propia.

```

def calcular_sinus(p):
    for i in range(N):
        vi = math.fmod(pos[i][p-1] + moment[i][p-1] * vart, 2*pi)
        if vi < 0:
            vi = 2*pi + vi
        pos[i][p] = vi

        aux = pos[i][p-1]
        sinus[i][:] = np.sin(-pos.T[p-1] + aux)
        moment[i][p] = moment[i][p-1] + vartEN * sum(sinus[i][:])

def simulacion():
    for i in range(1,pointN):
        calcular_sinus(i)

```

Figura 15: Parte principal del simulador alternativa 2. Elaboración propia.

En este caso he substituido la función `sumsin(vi)` por la función `calcularsinus(p)` que engloba todos los cálculos y bucles internos. Aquí en lugar de realizar un bucle triple el bucle es doble y se utiliza la vectorización. Para ello, en lugar de usar las listas definidas por Python, usamos la librería NumPy para poder crear las matrices reservando espacio (así evitando problemas de espacio también), aparte, las columnas indican el número de la iteración y las filas las partículas que estamos tratando. Dado que queremos vectorizar los cálculos, operar por columnas nos era imposible en el apartado anterior, pero gracias a NumPy solo tenemos que transponer las filas.

Con estos pequeños cambios logramos simplificar el código aún más y obtener mejores resultados respecto al tiempo. Conseguimos hacer el sumatorio de la fórmula en una línea de código y sumarlo al resto de la fórmula en una línea más. Es decir, ejecutamos las N iteraciones del bucle más interno de una pasada.

Los resultados con esta alternativa han sido mucho mejores. Hemos pasado de tiempos medios de 28 minutos ($N = 52$ y $\text{pointN} = 1000$) a tiempos medios de 1,1 segundos, una mejora de 99,9%. Estos tiempos tienen mucho más sentido, teniendo en cuenta los parámetros y las características de nuestro portátil, que aunque, no sea el más potente, no tiene más de 5 años.

Conclusión

Hemos dedicado bastante tiempo a la mejora que esta fracción de código. Como hemos mencionado en la planificación, la idea era partir con un número pequeño de partículas y de allí ir subiendo. Nuestro primer objetivo era que siguiese las fórmulas y que los resultados fueran correctos. Con la primera alternativa lo conseguimos con creces. Pero los tiempos eran muy malos. Analizando y debugando, llegamos a dos posibles problemas.

El primero, la matriz se está alcatando más de una vez, por qué al momento de crearse el tamaño es muy pequeño y esos tiempos son debidos a la creación de una copia con mayor tamaño. Esta opción quedó descartada al ver que utilizando la librería NumPy y reservando él espacio necesario, los tiempos eran aún más grandes. Además, hablando con los directores del proyecto, me explicaron que estos tamaños de matrices eran bastante pequeños.

La segunda opción fue, es debido al número de operaciones y al coste del algoritmo, pero vemos que N y $\text{point}N$ no tienen valores tan elevados. Nuestro portátil funciona a 2.4 Ghz, por lo que en pocos segundos tendría que ser capaz de resolver todos los cálculos. Aun así, miré cuanto tardaba en realizar cada iteración y donde le dedicaba más tiempo. Allí me di cuenta de que la mayor parte del tiempo se la dedicaba a calcular los senos.

Para mejora los tiempos implementamos la mejora de programación dinámica en la alternativa 1, y aunque, tuvimos mejoras notables, los tiempos seguían siendo muy lentos. Finalmente, gracias a un consejo de mi codirector *Pietro Massignan*, decidí vectorizar y averiguar en que consistía esta técnica. Haciendo esto obtuvimos los mejores resultados. También cambié la librería que calculaba los senos, pasando de *SimPy* a **NumPy**, ya que puede calcular todos los senos de una pasada.

El número de operaciones es el mismo en las dos alternativas, pero al realizarlas en conjunto el tiempo es mucho menor. Podríamos decir que el coste de la segunda alternativa sería es de $\Theta = N * \text{point}N$. Aprovechando el procesador y el tipo de operaciones, para ejecutar cálculos a la vez.

4.2. Representación de los resultados

```
if typePlot == "energyTheta":
    plt.scatter(primeras, segundas,s=sizeP)
elif typePlot == "timeTheta":
    #if tipo == "potenciadex" and E == 1:
    #    for i in range(N):
    #        for j in range(pointN):
    #            if pos[i][j] < pi:
    #                pos[i][j] += pi
    #            else:
    #                pos[i][j] -= pi
    #if intervals == []:
    #    for i in range(N):
    #        plt.scatter(pos[i], tiempos,s=sizeP, color='blue')
    #else:
    #    for i in range(N):
    #        plt.scatter(pos[i][intervals[0]:intervals[1]], tiempos[intervals[0]:intervals[1]], s=sizeP,dpi=100)
elif typePlot == "variance":
    titleX = 'r'\epsilon$'
    titleY = "Time"
    plt.xlabel(titleX, fontsize=sizeText)
    plt.ylabel(titleY, fontsize=sizeText)
    calcVarianca()
    for i in range(1, pointN-1):
        if varianca[i-1] < varianca[i] and varianca[i+1] < varianca[i]:
            tiempoMax.append(tiempos[i])
            maximos.append(varianca[i])
            break
#global vartEN
for e in range(2,6):
    vartEN = vart * (-e/N)
    calprimeros()
    simulacion()
```

Figura 16: Fragmento del código encargado de generar los plots. Elaboración propia.

Finalmente, queda la representación de los resultados. Para ello utilizaremos la librería Matplotlib [8]. Como ya tenemos todos los datos tratados, solo tendremos que plotear cada partícula en forma de puntos(usando la opción scatter de Matplotlib) para el caso de la gráfica $time/\theta$. Además, como función añadida, podemos escoger que porción de los datos queremos plotear rellenando el campo *intervals*.

Como se pueden crear diferentes tipos de gráficas aparte de la $time/\theta$ usaremos una secuencia de ifs para escoger que datos plotear en función del campo typePlot. Hay diferentes plots que podemos crear, entre los cuales se encuentran:

- **timeTheta** Representa la trayectoria temporal de las partículas para los diferentes tipos de interacción.
- **variance**. Genera una gráfica que indica cuanto tiempo tarda un sistema en llegar a su máximo de energía potencial en función de la amplitud.
- **energys** produce una gráfica que representa la energía potencial y cinética en función del tiempo de ejecución.
- **timeN** Crea una gráfica para medir el tiempo de ejecución en función al número de partículas que se están simulando. Para ejecutarlo se tendrá que rellenar el vector *Ns* con los diferentes casos que se quieran analizar.

- **timepointN** Crea una gráfica para medir el tiempo de ejecución en función al número de iteraciones que se quieran simular. Para ejecutarlo se tendrá que rellenar el vector *pointNs* con los diferentes casos que se quieran analizar.

Aparte los campos *title*, *tittleX*, *tittleY* *sizeY*, *sizeX*, *sizeP*, *sizeText* modificarán el título, título del eje x, título del eje y, anchura, altura, tamaño de los puntos y tamaño del texto respectivamente.

5. Pruebas y Resultados

5.1. Análisis de los Resultados

5.1.1. Figura a

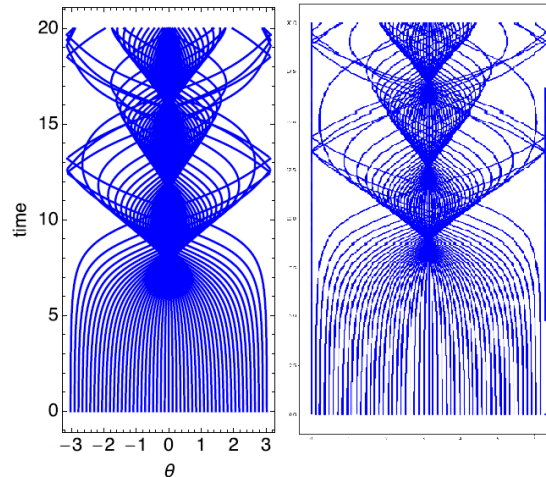


Figura 17: Imagen comparativa *figura a* del artículo *vs* obtenida con el programa creado. Elaboración propia.

Los valores que hemos utilizado para crear nuestro plot han sido:

- $E = -1$
- $N = 51$
- $\text{pointN} = 1000$
- $\text{eixX} = 10$
- $\text{eixY} = 20$
- $\text{sizeP} = 2$
- $L_i^{n=0} = 0,005 * \sin(\theta_i^{n=0})$

La figura 17 muestra la trayectoria temporal del artículo *vs* la creada con nuestro programa. Lo primero que cabe destacar de los dos plots es que no son idénticos. Lo único que disponíamos para recrear los plots eran las fórmulas y la imagen de la izquierda. He intentado darle las dimensiones más cercanas a la imagen original. Aun así, basta para comprobar que los datos del artículo son correctos y siguen el mismo comportamiento.

A simple vista se puede ver que el comportamiento de cada partícula es el mismo en las dos figuras, no hay ninguna diferencia aparte del grosor de las líneas que se puede modificar al gusto de cada uno. Aquí vemos el mismo comportamiento que se menciona en *Violent relaxation in quantum fluids with long-range interactions* [9], las partículas tienden a agruparse en grupos, en este caso en un monocluster. En las dos figura se ve un comportamiento que a primera vista parece ser cíclico, a los 8.5 segundos se juntan las partículas en un monocluster y despues van oscilando de manera uniforme.

Comentaremos algunos puntos a destacar:

- En el artículo pone que son 52 partículas, pero las contamos, y son 51.
- Si aplicamos el punto de partida del artículo ($L_i^{n=0} = 0,005 * \sin(\theta_i^{n=0})$) los resultados no nos salen centrados como en el plot original, por lo que, decidimos cambiar el *cos* por un *sin*. El resultado es el mismo, únicamente lo centramos en la gráfica.

Los tiempos de ejecución de este caso han sido de 0.98 segundos, en un portátil dual-core de 2,5Ghz.

5.1.2. Figura b

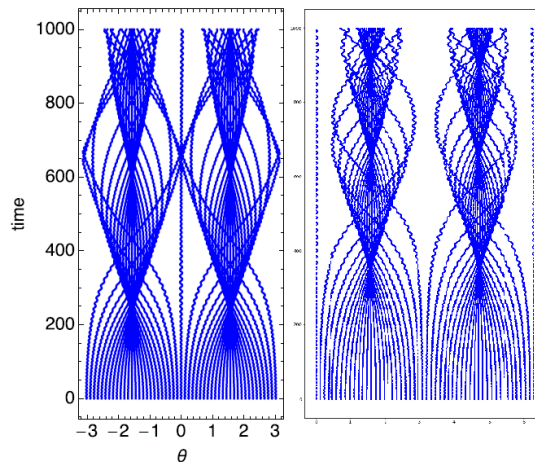


Figura 18: Imagen comparativa figura b del alticulo *vs* obtenida con el programa creado. Elaboración propia.

Los valores que hemos utilizado para crear nuestro plot han sido:

- $E = 1$

- $N = 51$
- $\text{pointN} = 100000$
- $\text{eixX} = 10$
- $\text{eixY} = 20$
- $\text{sizeP} = 2$
- $L_i^{n=0} = 0,005 * \cos(\theta_i^{n=0})$

La figura 18 muestra la trayectoria temporal del artículo *vs* la creada con nuestro programa. Al igual que en el caso de la figura a, los plots creados por nuestro programa no son idénticos a los originales debido a los mismos motivos. En este caso, en que las fuerzas son repulsivas, se nota incluso más. Esto se puede observar en el tiempo que transcurre en los plots. El tiempo es mayor y el número de puntos calculados también, si no, se puede perder calidad y obtener resultados para nada cercanos a la realidad.

A simple vista se puede ver que el comportamiento de cada partícula es el mismo en las dos figuras, no hay ninguna diferencia aparte del grosor de las líneas que se puede modificar al gusto de cada uno. Aquí vemos el mismo comportamiento que se menciona en *Violent relaxation in quantum fluids with long-range interactions* [9], las partículas tienden a agruparse en grupos, en este caso en un bi-cluster. Aun así, en este caso, hay sutiles diferencias. La primera es que la partícula del medio del plot original está a los extremos en el plot creado con nuestro programa, esto no es nada grave, solo se encuentra desplazado, pero el comportamiento es el mismo. El otro es el comportamiento de las partículas en los extremos, en el caso original parece que de un extremo va al otro (recordar que hablamos de módulos .es un anillo, pasa de 2π a 0 o viceversa”), mientras que en nuestra implementación los clústeres se ven más marcados. Esto puede ser debido a la resolución y al número de puntos calculados. Más adelante veremos como en estos casos la resolución y el número de partículas causan bastantes cambios en los resultados. Creemos que nuestros resultados son un poco más precisos que los del artículo, al menos en este caso. Esto lo notamos en que si reducimos Δt y aumentamos el número de pasos, el biclúster es un poco más notable.

También hemos observado que en el artículo pone que son 52 partículas, pero las contamos, y son 51.

El tiempo de ejecución de este caso ha sido de 1.66 minutos, en un ordenador con un dual-core de 2,5Ghz. Teniendo en cuenta de para cada partícula, de las 51 que hay, calculamos 100000 puntos, consideramos que no está nada mal.

5.2. Conservación de la energía

Para asegurarnos que el programa que hemos creado funciona correctamente, podemos comprobar que el Hamiltoniano se conserve durante todas las iteraciones y se mantenga constante el valor de H . En este caso lo interpretaremos como Energía. Pero es importante destacar que el hamiltoniano no corresponde a la energía [13]. Bajo ciertas condiciones relacionadas con las características del sistema (sistema conservativo) y las coordenadas empleadas, el hamiltoniano puede identificarse con la energía mecánica del sistema, aunque esto no sucede para todos los sistemas. En este caso, y como se comenta en el artículo [9], sí que corresponde a la energía mecánica. Concretamente a la energía cinética y energía potencial que aparecen en la fórmula 1.

Para revisar los resultados solo tenemos que calcular la energía para cada partícula, tanto E_{pot} como E_{kin} , y realizar su sumatorio en cada momento. Sí, vemos que la Energía total (la suma) se mantiene constante, significa que la fórmula 1 se cumple. Para ello hemos creado dos funciones que realizarán el cálculo de las energías: `calcPotencia()` y `calcCinetic()`.

```
def calcPotencia():
    for i in range(pointN):
        aux1=0.0
        for e in range(N):
            aux = pos[e][i]
            for j in range(e+1,N):
                aux1 += np.cos(-pos[j][i] + aux)
        varianca[i] = aux1*E/N

def calcCinetic():
    for i in range(pointN):
        vel[i] = sum(np.power(moment.T[i],2))/2
```

Figura 19: Funciones `calcPotencia()` y `calcCinetic()`. Elaboración propia.

- **CalcPotencia()** realiza $\frac{E}{N} \sum_{i < j} \cos(\theta_i - \theta_j)$. Al haber la restricción $i < j$ no podemos vectorizar y nos vemos obligados a utilizar un bucle extra y una variable auxiliar para guardar los resultados. La variable **pos** contiene las posiciones de cada partícula para cada momento de tiempo.
- **CalcCinetic()** realiza $\frac{m}{2} \sum_i v_i^2$. **moment** contiene el momento L_i de cada partícula i en un tiempo determinado, y como hemos visto anteriormente que $L_i = v_i$, lo podemos substituir directamente.

Ahora procederemos a ver si la energía total se mantiene constante. Para ello, utilizaremos los mismos valores que usamos en el análisis de la *figura a* aumentando el tiempo. Pasaremos de los 20 segundos a 80 segundos, es decir, `pointN` será igual a 8000. Los resultados son los siguientes:

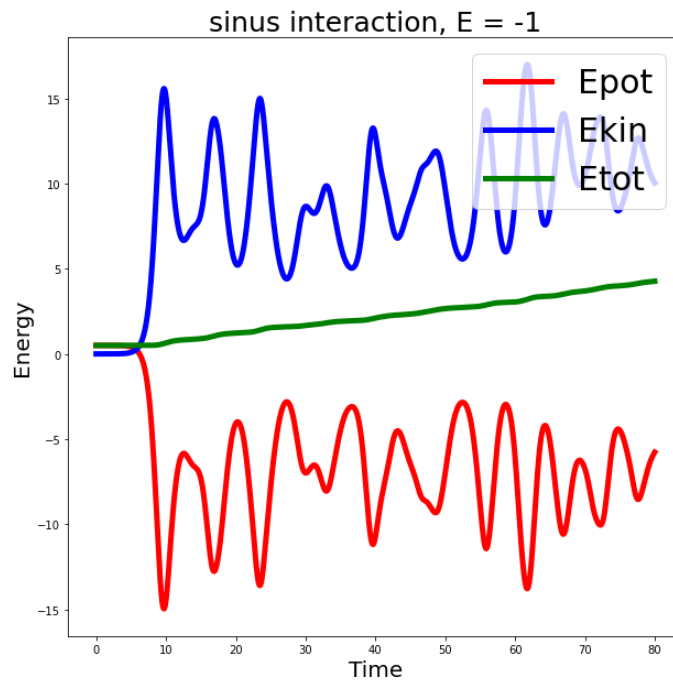


Figura 20: Estudio de la convergencia. Ejemplo de artefactos en la energía del sistema debidos al paso del tiempo demasiado grande. Elaboración propia.

La figura representa la evolución de la energía en el sistema en función del tiempo. El eje x representa el paso del tiempo, mientras, que el eje y representa la energía. Hay tres energías que se calculan: la energía potencial (E_{pot}), la energía cinética (E_{kin}) y la total (E_{tot}). Para realizar dicho calculo hemos utilizado $\Delta t = 0,01$, $\epsilon = -1$ y $N = 51$.

Se observa Como el Hamiltoniano no se mantiene. Miramos de aumentar la resolución cambiando $\Delta t = 0,01$ a $\Delta t = 0,001$. Para comprobar si ahora sí se mantiene el Hamiltoniano.

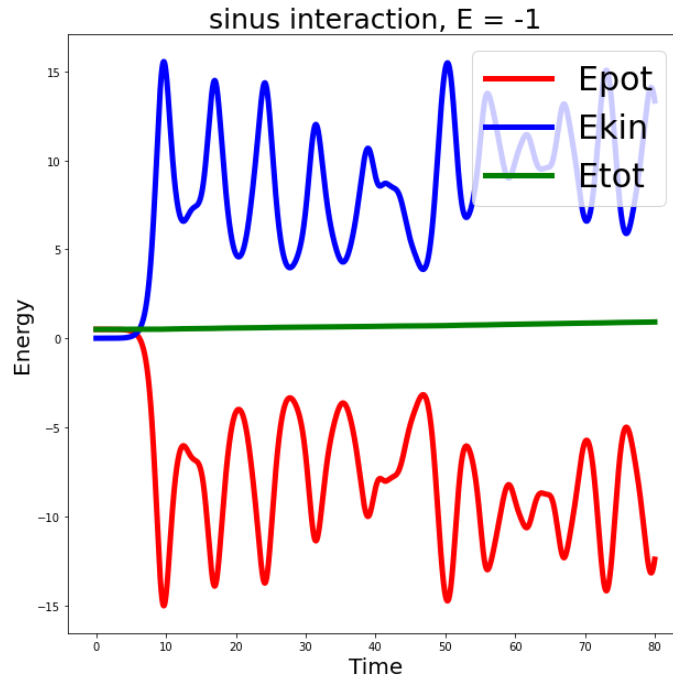


Figura 21: Energía cinética, potencial y total en función del tiempo. Elaboración propia.

La figura representa la evolución de la energía en el sistema en función del tiempo. El eje x representa el paso del tiempo, mientras, que el eje y representa la energía. Hay tres energías que se calculan: la energía potencial (E_{pot}), la energía cinética (E_{kin}) y la total (E_{tot}). Para realizar dicho calculo hemos utilizado $\Delta t = 0,001$, $\epsilon = -1$ y $N = 51$.

Ahora sí, la fórmula del Hamiltoniano se mantiene constante. Es necesario tener una Δt pequeño para evitar errores de resolución o imperfecciones, aunque esto puede llevar a grandes aumentos de tiempo. Para verificar que el programa funcionaba bien nos ha bastado con una resolución no demasiada pequeña, pero para otros casos, como la figura b, necesitamos aumentar la resolución (el número de puntos) y reducir Δt tal y como lo hicimos desde un principio. También podemos observar claras diferencias entre las figuras 20 y 21. Mientras que en la figura 20 la energía no se mantiene constante, en la figura 21 sí que lo hace. Por otra parte, el comportamiento de la energía es exactamente el mismo. Las dos figuras alcanzan sus mínimos y máximos en los mismos momentos y fluctúan de la misma forma.

5.3. Análisis temporal

Procederemos a hacer el análisis temporal de nuestro código, para diferentes valores de pointN y N , qué, como hemos comentado en puntos anteriores, son los que influyen en su coste computacional. Los valores que hemos utilizado como invariables son:

- $E = -1$
- $\Delta t = 0,01$
- $\text{pointN} = 1000$

Iremos modificando el valor de N y calcularemos su tiempo de ejecución (solo la simulación, la parte principal del código), para comprobar si el coste es lineal o exponencial.

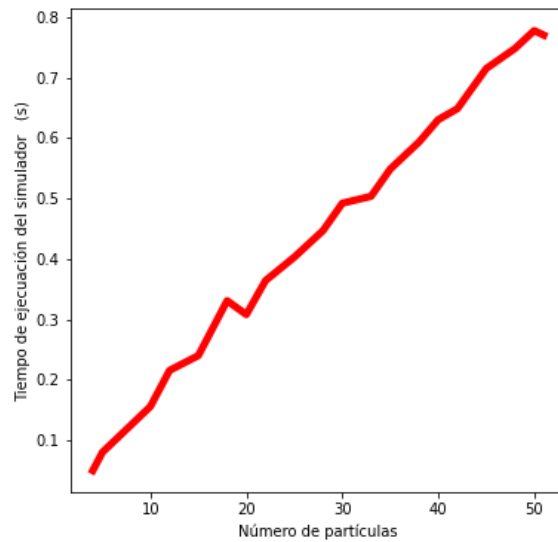


Figura 22: Evolución del tiempo según el número de partículas N . Elaboración propia.

La figura 22 representa la evolución del tiempo en ejecutar la simulación en función al número de partículas, concretamente el tiempo de cálculo, encargado de calcular los momentos y las posiciones que se representan en las gráficas de trayectoria temporal (tiempo/θ). En otras palabras, indica cuanto tarda en ejecutar la parte del simulador, explicada en el punto 4.1.2, en función al número de partículas.

Como se ve, el coste es claramente lineal. Ahora comprobaremos si se comporta de la misma forma en función de pointN .

Campos invariables:

- $E = -1$
- $\Delta t = 0,01$
- $N = 51$

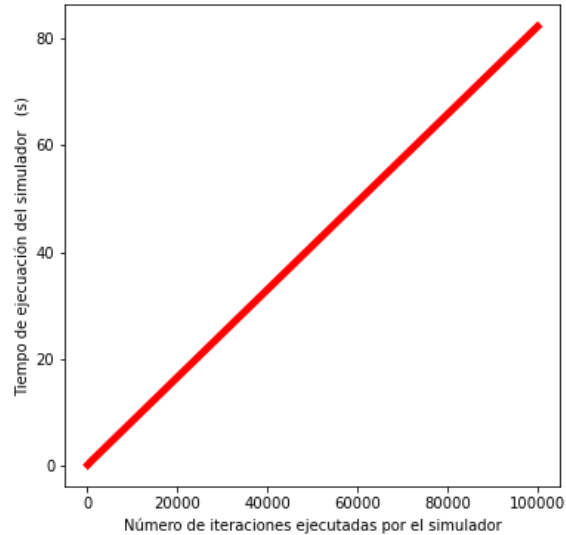


Figura 23: Energía cinética, potencial y total en función del tiempo. Elaboración propia.

La figura 23 representa la evolución del tiempo en ejecutar la simulación en función al número de iteraciones, concretamente el tiempo de cálculo, encargado de calcular los momentos y las posiciones que se representan en las gráficas de trayectoria temporal(*tiempo/ θ*). En otras palabras, indica cuanto tarda en ejecutar la parte del simulador, explicada en el punto 4.1.2, en función al número de iteraciones.

En los dos casos el coste es lineal, gracias a la vectorización conseguimos tiempos muy buenos y no obtenemos tiempos exponenciales como esperabamos cuando empezamos a escribir el código. Las gráficas analizadas son suficiente para justificar el coste de nuestro código y serciorarnos que es lineal y no exponencial.

5.4. Nuevas interacciones

Una vez hemos verificado los resultados del artículo *Violent relaxation in quantum fluids with long-range interactions*[9] procederemos a realizar simulaciones con diferentes tipos de interacciones.

Recordemos que:

- La fuerza es igual a masa por aceleración.
- La aceleración es la derivada de la velocidad y la velocidad es la derivada de la posición respecto el tiempo.
- $sign(x)$ retorna el signo de x , si es negativo retorna -1, si es positivo +1.

$$m \cdot \ddot{x}_i(t) = \sum_{i \neq j} F(x_{ij}) \quad (12)$$
$$\dot{x}_i = v_i$$

- x_i - Posición de la partícula i , $i = 1, \dots, N$.
- v_i - Velocidad de la partícula i , $i = 1, \dots, N$.

$$\dot{v}_i(t) = \sum_{i \neq j} \frac{1}{m} F(x_{ij}) sign(x_i - x_j) \quad (13)$$
$$x_{ij} = x_i - x_j$$
$$F(x_{ij}) = -\frac{\partial}{\partial x_{ij}} Vpot(x_{ij})$$

5.4.1. Potencia de x

Empezaremos con una alternativa sencilla:

$$Vpot(x) = A \cdot |x|^P \quad (14)$$
$$F(x) = +A \cdot P \cdot |x|^{P-1}$$

Ahora solo tenemos que añadir una parte del código que realice el cálculo de las fuerzas según la nueva fórmula.

Los casos que simularemos serán: $P = 2$ y $P = 3$. $P = 1$ no vale la pena analizarlo porque es una constante.

$P = 2$ Para el caso atractivo hemos obtenido el siguiente plot:

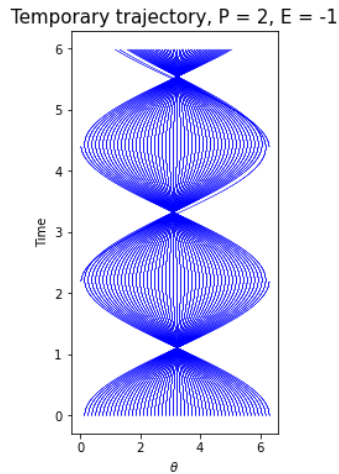


Figura 24: Trayectoria temporal con $\epsilon = -1$, $N = 51$, $\Delta t = 0,0003$. Elaboración propia.

Representa la trayectoria temporal de 51 partículas con una fuerza atractiva y un Δt de 0.0003s. Como se puede ver, también tiende a un rápido colapso en el segundo 1 aproximadamente. Además, lleva un comportamiento cíclico. Es un caso muy parecido al de la figura a aunque en este caso hay menos variación. Igualmente se comprueba que se forma un monoclúster.

Para el caso repulsivo el siguiente:

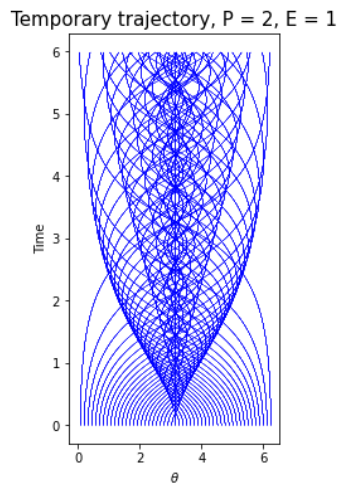


Figura 25: Trayectoria temporal con $\epsilon = 1$, $N = 51$, $\Delta t = 0,0003$. Elaboración propia.

Representa la trayectoria temporal de 51 partículas con fuerzas repulsivas y un Δt de 0.0003. En

este caso también se puede observar un comportamiento cíclico y repetitivo. Parece que se forman diferentes niveles, donde las partículas se unen en pequeños clústeres. Se puede llegar a ver tres capas bien diferenciadas, cada una sobre otra.

P = 3 Para el caso atractivo hemos obtenido el siguiente plot:

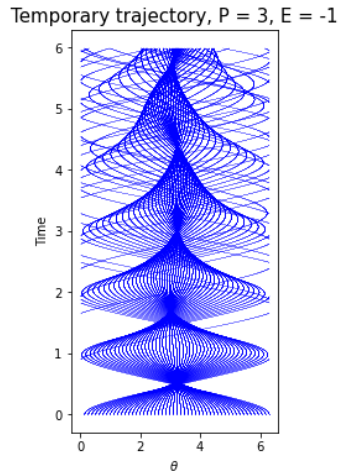


Figura 26: Trayectoria temporal con $\epsilon = -1$, $N = 51$, $\Delta t = 0,0002$. Elaboración propia.

Representa la trayectoria temporal de 51 partículas con fuerzas atractivas y un Δt de 0.0002. Los resultados son muy parecidos al caso de $P = 2$, aunque aquí podemos observar ciertas características interesantes a comentar. La primera es que ahora las partículas colapsan en la mitad de tiempo aproximadamente(0.5 segundos), esto tiene mucho sentido debido a la potencia, que es mayor. Además, tienden a seguir una forma de campana donde la punta va desplazándose de un lado al otro.

Para el caso repulsivo el siguiente:

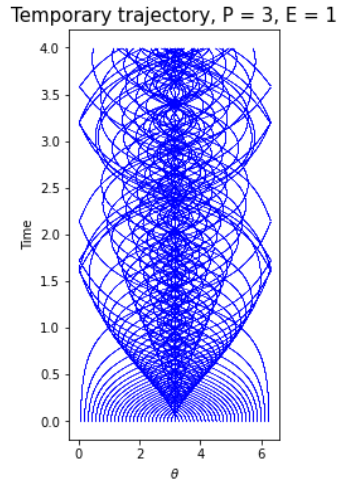


Figura 27: Interacción con $\epsilon = 1$, $N = 51$, $\Delta t = 0,0003$. Elaboración propia.

Representa la trayectoria temporal de 51 partículas con fuerzas repulsivas y un Δt de 0.0002. La figura es muy parecida al caso $P = 2$, aunque como comentamos antes, las partículas colapsan antes.

5.4.2. Power-law Potential

Primero de todo, haremos que sigan la **power-law potential** con diferentes tipos de potenciales. Para ello añadiremos nuevas funciones para poder seguir este nuevo tipo de interacciones.

$$\begin{aligned}
 V_{pot}(x) &= \frac{A}{|1+x|^P} \\
 F(x) &= + \frac{A \cdot P}{|1+x|^{P+1}}
 \end{aligned}
 \tag{15}$$

- P - exponente del potencial
- A - amplitud

Los posibles casos que vamos a analizar son $P = 1, 2, 3$ con fuerza repulsiva $\epsilon = 1$ o atractiva $\epsilon = -1 = A$:

P = 1

$$\begin{aligned}
 V_{pot}(x) &= \frac{A}{|x|} \\
 F(x) &= + \frac{A}{|1+x|^2}
 \end{aligned}
 \tag{16}$$

P = 2

$$V_{pot}(x) = \frac{A}{|x|^2}$$

$$F(x) = + \frac{A \cdot 2}{|1+x|^3} \quad (17)$$

P = 3

$$V_{pot}(x) = \frac{A}{|x|^3}$$

$$F(x) = + \frac{A \cdot 3}{|1+x|^4} \quad (18)$$

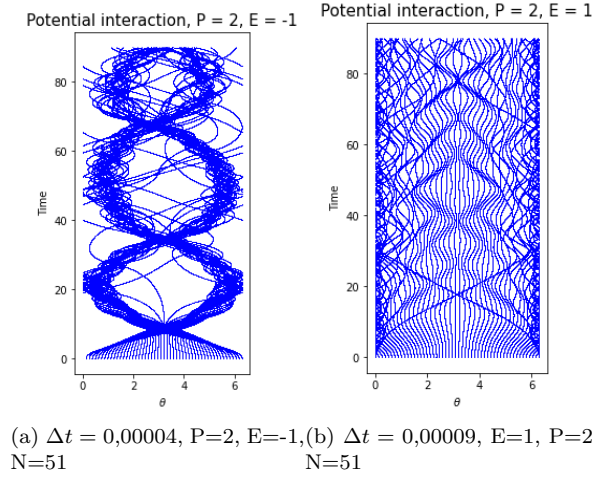


Figura 28: Trayectoria temporal con $P = 2$. Elaboración propia.

La figura 28 contiene dos plots que representan la trayectoria temporal cuando $P = 2$. La figura *a* representa las interacciones con fuerzas atractivas, mientras que la figura *b* representa fuerzas repulsivas. En los dos casos vemos cosas curiosas de analizar.

En la figura *a* vemos como las partículas colapsan rápidamente por el segundo 10 aproximadamente. Además, vemos como las partículas forma un monoclúster, que aun así, se dividen en dos subclústeres que van colapsando pasado un tiempo.

En la figura *b* se puede ver como las partículas inicialmente tienden a separarse para después juntarse y luego volver a separarse. Recuerda bastante a la figura *b* del artículo *Violent relaxation in quantum fluids with long-range interactions*[9] que fluctuaba hasta que, al cabo de bastantes segundos, colapsaba. En ambos casos se puede ver un ciclo que se va repitiendo.

Los casos con $P = 1$ y $P = 2$ tiene los mismos comportamientos pero en diferentes escalas de tiempo. Cuando $P = 1$, se tarda más tiempo en colapsar. Mientras que cuando $P = 3$ se tarda menos tiempo (en comparación con $P = 2$). Por esto, no hemos incluido más figuras, ya que serían repetitivas.

6. Obstáculos y cambios en el proyecto

Al largo del proyecto nos hemos ido encontrando algunos obstáculos, comentamos en el punto 2.2.4:

- *Inexperiencia.* Al momento de empezar el proyecto tenía pequeñas nociones de física. Pero al empezar, muchos campos abandonados (por mí) de la física y las matemáticas aparecieron de golpe. Tuve que dedicarle un tiempo extra a refrescar conceptos para entender como implementar el simulador y que estaba calculando.
- *Desconocimiento de física.* Gran parte del tiempo, sobre todo en la fase de Estudio previo y diseño. Tuve que aprender o repasar conceptos de física que se mencionan en el artículo o necesarios para calcular las derivadas.

Aparte, también hemos tenido algunos problemas comentados en el apartado de Riesgos, punto 2.2.5:

- La **gestión del tiempo** ha sido bastante complicada. Lo más difícil ha sido compaginar el TFG, el trabajo y las clases, que eran igual de importantes y, en ocasiones, restaban un poco el tiempo que podía dedicarle al TFG. En caso de las otras actividades comentadas, se podía marcar prioridades y buscar un hueco para el TFG.
- *Accidentes* A habido días puntuales en los que tuve un resfriado bastante fuerte y pause el TFG para descansar.
- *No correspondencia de los resultados.* En la parte final del proyecto, al añadir nuevas interacciones no hemos conseguido plotear simulaciones con sentido cuando $\epsilon = -1$. Le hemos dedicado más tiempo, pero aun así, no he sido capaz de resolverlo.

Hemos ido revisando los costes a lo largo del proyecto y los costes se mantienen. Aparte de la desviación más grande debida a la no correspondencia de los resultados, no hemos tenido otras desviaciones. Es importante remarcar, que se trata de interacciones añadidas al programa. El programa final funciona correctamente, y es apto para futuras mejoras y reparaciones, como queríamos desde un principio.

7. Aclaraciones programa final

Aparte del programa principal encargado de plotear una gráfica $time/\theta$ hemos añadido nuevas funcionalidades que hemos ido comentando a lo largo del proyecto. No obstante, es relevante volver a comentarlas y hacer ciertas aclaraciones.

Dependiendo de la opción que escojamos habrá campos que se tendrán que rellenar de manera obligatoria y otros que serán ignorados por no ser útiles.

Los gráficos que se pueden crear son:

- **timeTheta**. Sirve para verificar y observar el comportamiento de las partículas. Cuando escogemos este campo podemos jugar con diferentes atributos: *intervals*, *load*, *save* y *startIteration*. *Intervals* nos permite plotear solo un fragmento de los datos, si queremos ver todos los datos es fundamental dejar el vector (es un vector) vacío, en caso contrario se tiene que añadir el mínimo y el máximo que acotará la parte de datos que visualizaremos. *Save* guarda los datos en *posiciones.txt* y *momentos.txt*. *Load* y *startIteration* van de la mano, *load* lee los ficheros *posiciones.txt* y *momentos.txt* y *startIteration* indica que posición se quiere que sea el inicio de la nueva simulación (Es como empezar una simulación en mitad de otra con los datos previamente ya calculados). *Load* y *startIteration* van de la mano, es obligatoria utilizar las dos, si vas a leer datos los tendrás que utilizar.
- **energys**. Plotea La energía potencial y cinética que hay en cada momento de la simulación. Sirve para comprobar si la energía se conserva y los datos son de fiar.
- **variance**. Crea un plot en el que indica el tiempo que se tarda hasta llegar a un máximo de energía potencial en función de la amplitud o ϵ .
- **timeN**. Genera un plot en función del tiempo y el número de partículas. Sirve para observar como fluctúa el tiempo de ejecución en función del número de partículas. En este caso tenemos que rellenar el campo *Ns*, con todos los casos que queramos analizar en orden creciente. También es importante cambiar *N* por el valor máximo de los valores añadidos al vector *Ns*, para reservar el espacio necesario.
- **timepointN**. Produce un plot en función del tiempo y el número de iteraciones (pasos). Sirve para observar como fluctúa el tiempo de ejecución en función al número de iteraciones a realizar. En este caso tenemos que rellenar el campo *pointNs*, con todos los casos que queramos analizar en orden creciente. También es fundamental cambiar *pointN* por el valor máximo de los valores añadidos al vector, para reservar el espacio necesario.

Si los datos a primera vista no parecen tener mucho sentido es posible que estemos tomando la simulación con un input incorrecto, para resolverlo es importante ir probando y usar el sentido común. Por ejemplo, si lo que vemos son líneas rectas tenemos de aumentar en número de pasos (*pointN*) o la variación del tiempo Δt . Si se ve muy aplanado, reducir el tamaño de *sizeX*, ...

El programa puede dar errores de espacio en ciertas condiciones. Como por ejemplo, estamos ejecutando la opción *variance* y tenemos un *Npoint* muy pequeño. Nuestro programa al inicio de todo reserva espacio para realizar los cálculos, si sabemos que en algún momento vamos a necesitar más, es necesario poner el valor máximo al que puede llegar. Esto también puede pasar cuando utilizamos la opción *timeN* y *timepointN*.

8. Conclusiones

El objetivo final de este proyecto era crear un programa capaz de simular diferentes tipos de interacciones en un sistema unidimensional. El programa cumple con lo establecido, pero al momento de añadir nuevos tipos de interacciones hemos tenido ciertos problemas y no podemos dar por válido lo últimos plots generados. Se tiene que revisar y buscar donde está el error. Aun así, el programa se puede tomar como punto de partida o ayuda a futuros proyectos. Por esta parte nos hemos quedado con un sabor de boca un tanto amargo, sobre todo por el gran tiempo invertido intentando arreglar el error sin éxito. Aun así, hemos añadido funcionalidades que no habías contemplado desde un principio y yo personalmente he aprendido y refrescado conceptos muy interesantes.

En el apartado 2.2.2 detallamos los objetivos que queríamos alcázar. Ahora pasaremos a comprobar si lo hemos conseguido o si no. En caso negativo, miraremos el porqué. Los subobjetivos e indicadores son los siguientes.

1. Control y gestión de la entrada.

- a) El programa es capaz de dar un valor por defecto a los parámetros no pasados. → Al decantarnos por modificar las variables directamente del código, no hay valores por defecto, simplemente no tienes que tocar los campos o establecer una de las opciones disponibles.
- b) El programa indica si hay algún error al procesar la entrada, en caso de haberlo indica cuál es. → Sí. El programa trata algunos errores, pero falta ampliar el análisis.

2. Control y gestión de la salida del programa.

- a) La Salida tiene el formato adecuado para posteriormente poder representar los datos gráficamente. → Sí.
- b) El mismo programa genera los gráficos directamente y se comunica con el generador de plots. → Sí.

3. Buen funcionamiento del programa.

- a) El programa es eficiente en términos generales. → Sí.
- b) El programa indica que parte de la simulación lleva hecha y cuanta queda por hacer. → Sí.
- c) Los resultados del programa son correctos y se pueden verificar. → Los resultados del artículo han sido comprobados, pero en la ampliación de interacciones no hemos podido obtener resultados correctos para algunas de las interacciones.

4. Simulación computacional.

- a) El programa utiliza e implementa los algoritmos y estructura necesaria para realizar una simulación computacional. → Sí

Conclusiones personales

Estoy bastante satisfecho con el proyecto que he realizado, todo y los pequeños problemas que tiene. En términos generales, funciona correctamente y es apto para añadir nuevos tipos de interacciones.

El proyecto me ha aportado una experiencia muy valiosa con Python, ya que es un lenguaje que aunque sea muy práctico y utilizado, yo, personalmente, lo tenía muy abandonado y siempre me ha llevado a dificultades. A sí mismo, me ha servido para repasar conceptos de física y devolverme el interés por este campo de la ciencia que no tocaba desde que iba al bachillerato.

Ciertamente, el TFG ha sido un trabajo muy interesante de hacer y un poco difícil a la vez. Repasar conceptos, buscar tiempo libre para poder avanzar. Me ha recordado mucho a la experiencia que llegue a tener con mi Trabajo de Recerca. En los dos casos de una temática ajena a la que estaba estudiando en aquellos momentos. Pero en los dos casos, con buenos resultados y con conocimientos adquiridos que, seguro, me serán muy útiles para futuros proyectos.

¡Muchas gracias por todo!!!

9. Referencias

Referencias

- [1] Silvia Albareda-Tiana, Jorge Ruíz-Morales, Pilar Azcárate, Rocío Valderrama-Hernández, and José Manuel Muñoz. The edinsost project: Implementing the sustainable development goals at university level. In *Universities as Living Labs for Sustainable Development*, pages 193–210. Springer, 2020.
- [2] UPC. Universitat Politècnica de Catalunya. Español, Nov 2021.
- [3] UPC. Universitat Politècnica de Catalunya. Salsa'm. programa de mentoria de la upc, Mar 2022.
- [4] Facultat d'Informàtica de Barcelona. Normativa del treball final de grau del grau en enginyeria ...
- [5] Monica González. Mecánica hamiltoniana, Oct 2010.
- [6] Jesús Daniel Arias Hernández, Andrés Fernando Jiménez López, and Hernán Oswaldo Porras Castro. Desarrollo de aplicaciones en python para el aprendizaje de física computacional. *Ingeniería Investigación y Desarrollo: I2+ D*, 16(1):72–82, 2016.
- [7] Kavita Jain, Freddy Bouchet, and David Mukamel. Relaxation times of unstable states in systems with long range interactions. *Journal of Statistical Mechanics: Theory and Experiment*, 2007(11):P11008, 2007.
- [8] matplotlib. Tutorials.
- [9] Ryan Plestid, Perry Mahon, and DHJ O'Dell. Violent relaxation in quantum fluids with long-range interactions. *Physical Review E*, 98(1):012112, 2018.
- [10] Tarifasgaluz. Consulta el precio de la luz (€/kwh): Tarifas y comparativa, Mar 2022.
- [11] René Thom. Structural stability and morphogenesis wa benjamin. *Reading, MA*, 1975.
- [12] Manuel Trigás Gallego. Metodología scrum. *Metodología scrum*, 2012.
- [13] wikipedia. Hamiltoniano (mecánica clásica), Oct 2021.
- [14] Wikipedia. Matlab, Aug 2021.

Resolución de 18 de marzo de 2009, de la Dirección General de Trabajo, por la que se registra y publica el XVI Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública 2009, March 18.

Resolución de 22 de febrero de 2018, de la Dirección General de Empleo, por la que se registra y publica el XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública 2018, February, 22, art. 15

Seguridad Social: Bases y tipos de cotización 2019 n.d.