



Licenciatura en Ingeniería Física

DINÁMICA DE SISTEMAS ESTELARES DESPUÉS DE LA FRAGMENTACIÓN A GRAN ESCALA Y DE DISCOS PROTOPLANETARIOS

Proyecto de investigación

Trimestre 22-O

Tonatiuh Sánchez Madrid

Matricula: 2123033869

Al2123033869@azc.uam.mx

Firma

Asesor:

Dr. Fidel Cruz Peregrino

Departamento de Ciencias Básicas

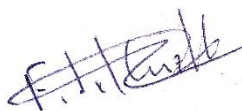
N° Económico: 38470

fcruz@azc.uam.mx

Firma

Declaratoria

Yo, Fidel Cruz Peregrino, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Dr. Fidel Cruz Peregrino

Yo, Tonatiuh Sánchez Madrid, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Tonatiuh Sánchez Madrid

Resumen

En el presente proyecto se desarrollan dos programas. Un primero capaz de extraer datos de archivos binarios resultantes al implementar el código simulaciones cosmológicas GADGET, el cual dentro de sus funciones está el generar una respuesta gráfica, detención automática del número de cuerpos que aparecen basados en los picos de densidad, determinar criterios de elección de límites para asignar masas y fijar posiciones y velocidades a los centros de masa de dichos cuerpos. El segundo programa tomó los datos resultantes de la manipulación de datos del primer programa e hizo una simulación de N-cuerpos, resolviendo de manera numérica la ecuación de movimiento resultante de aplicar las leyes de Newton y la de la gravitación universal, viendo como las soluciones a dicha ecuación fueron trayectorias cónicas. Se hizo también un análisis, desde el punto de vista de la mecánica clásica no relativista, al comportamiento de los fragmentos protoestelares ya que no se trabajó con escalas muy grandes fueron apenas longitudes menores a un año luz, una masa solar, velocidades nada cercanas a la luz y sin campos gravitacionales grandes. Se logró ver más allá del tiempo en el que se pausó la simulación en GADGET notando como nuestra simulación en particular se convirtió en un par de sistemas binarios que se alejaron de su interacción gravitatoria.

Contenido

Declaratoria.....	2
Resumen	3
Contenido.....	4
Introducción.....	9
El Problema de los N-Cuerpos.....	9
Simulaciones de Sistemas Estelares.....	9
Antecedentes	10
Justificación.....	12
Objetivos	12
Objetivo General.....	12
Objetivos Específicos	12
Simulaciones por computadora y modelos matemáticos	13
Simulaciones astronómicas.....	15
Extracción de datos a partir de simulaciones	17
Problema de N-cuerpos	19
Determinación de un método de solución	21
Método de integración directa	22
Metodología.....	23
Extracción y manipulación de datos a partir de simulaciones hechas en GADGET	24
Obtención de las posición, velocidad y densidad de cada partícula.....	24
Limpieza y agrupación de datos.....	26
Representación gráfica	30
Delimitación de radios de los nuevos objetos	32
Encontrar objetos dentro de la simulación.....	33
Delimitando rangos donde se encuentra el centro de masa.....	37
Datos relevantes de los centros de masas.....	38
Simulación de la solución al problema de los N-cuerpos	40
Desarrollo de algoritmo de N-cuerpos usando el método de Euler	40
Valores iniciales, número de cuerpos y vectores a usar	42
Resultados.....	44
Evolución gráfica de cada <i>snap</i>	44
Datos relevantes a los fragmentos protoestelares.....	54

Datos para la simulación de N-cuerpos	59
Gráficas de la simulación de N cuerpos	61
Análisis y discusión de resultados.....	64
Sobre las respuestas gráficas de la simulación en GADGET	64
Sobre la evolución de los fragmentos protoestelares	65
Velocidad.....	66
Densidad.	67
Masa.....	69
Sobre la simulación de N-cuerpos	71
Conclusiones	72
Referencias.....	76
Entregables	77

Tabla de ilustraciones

Figura 1. Ejemplo de simulaciones de formación de proto estrellas binarias. Adaptada de Equatorial density evolution in a 266 au region around the cloud center for model U2 after large-scale fragmentation into a binary protostar de Sigalotti et al. 1992.	10
Figura 2. Ejemplo del resultado obtenido por simulaciones en GADGET-2. Tomado de Colliding Disk Galaxies, https://wwwmpa.mpa-garching.mpg.de/gadget/collision/index.html , Springel 2005.	16
Figura 3. Sistema de N-cuerpos. Cada cuerpo de masa m_i interactúa con cada uno de los cuerpos adyacentes.	20
Figura 4. Diagrama de flujo del trabajo experimental.	23
Figura 5. DataFrame resultante, resumido por la cantidad tan grande de datos.	27
Figura 6. Representación del método de agrupación y reducción.	28
Figura 7. Nueva hoja de datos con los valores agrupados y reducidos.	30
Figura 8. Gráficas de densidades con respecto al eje x y al eje y . Snap 0090.	32
Figura 9. Representación gráfica del Snap 0090.	32
Figura 10. Asignación de objetos de manera visual.	33
Figura 11. Delimitación de objetos a partir de las gráficas de densidad.	33
Figura 12. Radio entre picos y delimitación de altura para reducir datos.	34
Figura 13. Ayuda gráfica para definir el criterio del porcentaje mínimo para tomar un dato válido.	37
Figura 14. Cambio de sistema de referencia a un nuevo sistema primado con centro en el centro de masa de los objetos 1 y 2.	43
Figura 15. Snap 0070.	45
Figura 16. Snap 0080.	45
Figura 17. Snap 0100.	46
Figura 18. Snap 0110.	46
Figura 19. Snap 0120.	47
Figura 20. Snap 0130.	47
Figura 21. Snap 0140.	48
Figura 22. Snap 0150.	48
Figura 23. Snap 0160.	49
Figura 24. Snap 0170.	49
Figura 25. Snap 0180.	50
Figura 26. Snap 0190.	50
Figura 27. Snap 0200.	51
Figura 28. Snap 0210.	51
Figura 29. Snap 0220.	52
Figura 30. Snap 0230.	52
Figura 31. Snap 0240.	53
Figura 32. Snap 0250.	53
Figura 33. Elección de sistemas a analizar.	59
Figura 34. Respuesta gráfica para el sistema principal de 4 cuerpos.	61
Figura 35. Respuesta gráfica para el subsistema de dos cuerpos. El cuerpo 1 hace referencia al sistema 1-3, mientras que el 2 al sistema 2-4.	61
Figura 36. Respuesta gráfica para el sistema 1-3.	62

Figura 37. Cambio de la posición en x con respecto al tiempo. Sistema 1-3.	62
Figura 38. Cambio de la posición en y con respecto al tiempo. Sistema 1-3.	62
Figura 39. Respuesta gráfica para el sistema 2-4.....	63
Figura 40. Cambio de la posición en x con respecto al tiempo. Sistema 2-4.	63
Figura 41. Cambio de la posición en y con respecto al tiempo. Sistema 2-4.	63
Figura 42. Comparación de resolución en la representación gráfica.	64
Figura 43. Trayectorias de los cuatro fragmentos protoestelares.....	65
Figura 44. Trayectoria de los sistemas 1-3 y 2-4.....	65
Figura 45. Velocidad del objeto 1.	66
Figura 46. Velocidad del objeto 2.	66
Figura 47. Velocidad del objeto 3.	66
Figura 48. Velocidad del objeto 4.	67
Figura 49. Densidad del objeto 1.	67
Figura 50. Densidad del objeto 3.	68
Figura 51. Densidad del objeto 2.	68
Figura 52. Densidad del objeto 4.	69
Figura 53. Masa del objeto 1.	69
Figura 54. Masa del objeto 3.	70
Figura 55. Masa del objeto 2.	70
Figura 56. Masa del objeto 4.	71
Figura 57. Simulación de N-cuerpos con datos iniciales del primer Snap (0070).	72

Índice de ecuaciones

(1).....	9
(2).....	9
(3).....	18
(4).....	19
(5).....	20
(6).....	20
(7).....	21
(8).....	34
(9).....	38
(10).....	40
(11).....	41
(12).....	41
(13).....	44
(14).....	44
(15).....	54
(16).....	54
(17).....	54

Índice de códigos

Código 1. Ejemplo del funcionamiento básico de las funciones de pygadgetreader.	25
Código 2. reducir_rango(), función para leer los datos del snap, almacenarlos y reducir los datos útiles dentro del rango de acción.	26
Código 3. Inicialización de dataframe y eliminación de datos duplicados.	27
Código 4. Reducción de decimales, eliminación de valores innecesarios y agrupación de datos.	29
Código 5. Paso intermedio para que los datos sean aptos para graficarse.	30
Código 6. Método para generar gráfica de x vs Densidad	31
Código 7. Método para generar gráfica de y vs Densidad.	31
Código 8. Método para generar la representación gráfica de la simulación.	31
Código 9. Primera parte del algoritmo detector de objetos. Limpieza de datos para encontrar los cuerpos de la simulación.	35
Código 10. Segunda parte del algoritmo detector de objetos. Limpieza de datos para encontrar los cuerpos de la simulación.	36
Código 11. Salidas al código de búsqueda de objetos.	37
Código 12. rangos(). Función para delimitar los rangos para cada uno de los valores máximo n	38
Código 13. centro_masa(). Función que retorna los datos del n cuerpo. *Por fines prácticos se le llamó masa a la densidad.	39
Código 14. Salida con todos los datos del centro de masa tal que: $[x, y, z, vx, vy, vz, \rho, [xi, xf, yi, yf]]$	40
Código 15. Inicialización del código de N-cuerpos.	41
Código 16. Cálculo de trayectorias usando el método de Euler en código Python.	42
Código 17. Graficar las posiciones de los cuerpos con respecto al tiempo.	42

Índice de tablas

Tabla 1. Parámetros de las funciones de pygadgetreader.	25
Tabla 2. Posiciones y velocidades del objeto 1.	54
Tabla 3. Posiciones y velocidades del objeto 2.	55
Tabla 4. Posiciones y velocidades del objeto 3.	55
Tabla 5. Posiciones y velocidades del objeto 4.	56
Tabla 6. Datos escalares del objeto 1 y delimitaciones.	56
Tabla 7. Datos escalares del objeto 2 y delimitaciones.	57
Tabla 8. Datos escalares del objeto 3 y delimitaciones.	57
Tabla 9. Datos escalares del objeto 4 y delimitaciones.	58
Tabla 10. Datos iniciales para el sistema principal de 4 cuerpos.	59
Tabla 11. Datos de la masa total y los centros de masa de cada sistema.	60
Tabla 12. Datos iniciales para el sistema 1-3.	60
Tabla 13. Datos iniciales para el sistema 2-4.	60
Tabla 14. Datos para el subsistema de 2 cuerpos.	60

Introducción

El Problema de los N-Cuerpos

El problema físico, se puede enunciar de manera informal como sigue: “Dadas en un instante las posiciones y las velocidades de dos o más partículas que se mueven bajo la acción de sus atracciones gravitatorias mutuas, siendo conocidas las masas de las partículas, para calcular sus posiciones y velocidades para otro instante” (Artigue & Pollio, 2013, pág. 3955).

En un lenguaje matemático, se tiene la siguiente expresión, para la fuerza de interacción gravitatoria entre dos cuerpos:

$$\mathbf{F} = -G \frac{m_1 m_2}{r^2} \mathbf{u}_r, \quad (1)$$

donde G es la constante de gravitación universal, m las masas y r es la distancia entre dichas masas. Esta expresión se resuelve con facilidad, la complejidad viene cuando se trata con más de 2 cuerpos, debido a la dependencia no línea. Además de que, al decaer la magnitud de manera cuadrática, no es suficiente rápido para despreciar a objetos lejanos. De lo anterior, viene el nombre de problema de N-cuerpos y las ecuaciones que describen el movimiento de N partículas, que tienen por interacción a la fuerza de gravedad mutua, dan como resultado un sistema de ecuaciones diferenciales acopladas de segundo orden:

$$m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i = - \sum_{j \neq i} \frac{G m_i m_j}{r_{ij}^3} \mathbf{r}_{ij}, \quad i = 1, 2, \dots, N, \quad (2)$$

donde m_i es la masa y \mathbf{r}_i el vector de posición de la i -ésima partícula; \mathbf{F}_i la fuerza total sobre esta y $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$. La ecuación (2) requiere de N^2 operaciones para evaluar todo el sistema ya que involucra la sumatoria de $N - 1$ partículas restantes (Aguilar, 1992).

Simulaciones de Sistemas Estelares

Sigalotti et al. (2018), en su publicación mencionan que los sistemas estelares constan de dos o más estrellas que fueron creadas por dos diferentes mecanismos; el primero, fragmentación a gran escala

de núcleos de nubes de gas y polvo durante su temprana fase de colapso isotérmico y la fragmentación a pequeña escala de sus discos circunstelares a principios del colapso debido a inestabilidades gravitatorias. En su trabajo realizan una simulación del colapso de un disco proto estelar usando un código consistente SPH (Smoothed Particle Hydrodynamics), dando como resultado la aparición de estrellas binarias.

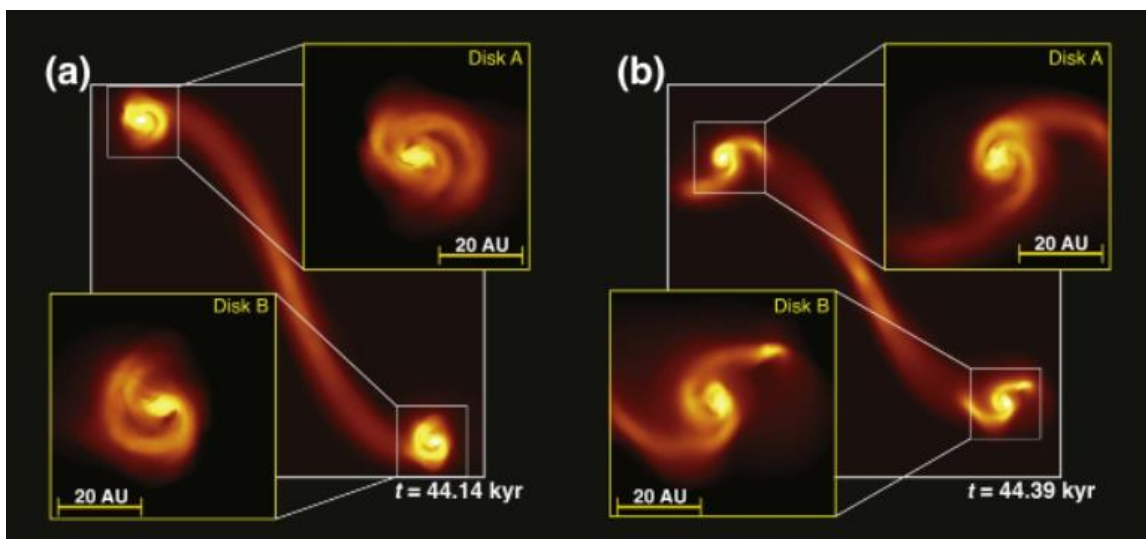


Figura 1. Ejemplo de simulaciones de formación de proto estrellas binarias. Adaptada de *Equatorial density evolution in a 266 au region around the cloud center for model U2 after large-scale fragmentation into a binary protostar* de Sigalotti et al. 2018.

Simulaciones de este tipo nos arrojan mucha información con la que se puede trabajar. Particularmente es de interés las posiciones y velocidades de los fragmentos proto-estelares, así como la masa, la cual, se determina por métodos relacionados con la densidad de partículas que los forman agrupadas por interacción gravitatoria.

Antecedentes

El estudio del movimiento de los astros comienza desde los primeros observadores de los planetas. A medida que se ha avanzado en las técnicas de observación, como la invención del telescopio, crece más la necesidad de encontrar modelos matemáticos capaces de predecir dichos movimientos.

Hemos pasado por modelos antiguos desde Platón y Aristarco de Samos donde proponen un a la tierra como centro del universo y el primer modelo heliocéntrico, respectivamente; hasta modelos más

cercanos a lo que conocemos hoy día, con Copérnico en 1536 y Galileo Galilei en 1609. Posteriormente, llegaría Johannes Kepler que, basado en los trabajos de Tycho Brahe, formularía una serie de reglas sobre el movimiento planetario que funcionaron para predecir el comportamiento de estos cuerpos. Tycho Brahe contaba con una gran cantidad de datos astronómicos basados en observaciones de su, centro de observación astronómica. Pero fue hasta la formulación de las leyes de movimiento de Newton y particularmente su Ley de la Gravitación Universal, que se tuvo una mejor aproximación al comportamiento de los astros (Artigue & Pollio, 2013).

Intentar resolver un problema de 3 cuerpos ya parece complicado, pero para ello es que se han desarrollado métodos numéricos para su fácil resolución y mejor aún, éstos se han podido resolver rápidamente con la llegada de las computadoras y se hacen más viables conforme la velocidad de cómputo aumenta.

Muchos algoritmos de N-cuerpos han sido desarrollados a partir de distintos métodos numéricos, teniendo como pionero a los trabajos de von Hoerner en la década de los sesenta con el método de integración directa que, posteriormente, se convirtió en el método de desarrollo en funciones armónicas y de igual manera al método de Fourier (Aarseth, 2003).

En un contexto cosmológico, la simulación de eventos astronómicos adquiere un importante papel para el estudio de nuestro universo, cómo será el futuro y cómo entender lo que pasó. Este tipo de simulaciones requieren de un poder de procesamiento bastante alto y no pueden ser corridas por tiempo indefinido. De hecho, apenas si se pueden conseguir algunos segundos de video haciendo difícil ver la evolución de dichos sistemas.

Aquí entra la simulación de N-Cuerpos, ya se pasa de un sistema de millones de partículas (gas y polvo estelar) a un sistema de dos o pocos más cuerpos estelares que están regidos casi por completo por sus interacciones gravitatorias. Estos algoritmos han sido usados mayormente en la astronomía por ser

un poco más sencillos y que no requiere de mucha capacidad de procesamiento que al seguir simulando las nubes de partículas.

Justificación

Las simulaciones con objetos astronómicos son de vital importancia para la ciencia actual, nos ayudan a tener un mayor conocimiento de cómo funciona nuestro universo. Por ello, la generación de nuevos métodos numéricos y algoritmos más precisos es crucial. Los eventos que le competen a la astronomía tienen la gran limitación de suceder a escalas de tiempo muy grandes y es casi imposible ver la evolución de los cuerpos celestes. Estas herramientas nos ayudan a poder trabajar en lapsos razonables para lograr un mejor entendimiento del cosmos.

Actualmente, el cosmos se entiende mejor con la relatividad general de Einstein, más los fenómenos estudiados en este trabajo no ocurren a velocidades cercanas a la luz ni en campo gravitacionales intensos, aún es viable usar conceptos de mecánica clásica no relativista, por lo que, algoritmos del tipo N-Cuerpos, seguirán dando frutos por dar aproximaciones muy buenas y más fáciles de manejar que los conceptos avanzados de la física moderna. Por lo anterior, estos conceptos siguen teniendo mucha relevancia en estudios en escalas y condiciones cotidianas al ser humano.

Objetivos

Objetivo General

Analizar con dinámica clásica, en el dominio no relativista, la evolución de sistemas estelares por medio de un análisis de N-cuerpos.

Objetivos Específicos

- Aprender a leer y extraer la información física, con un programa en Python, a partir de simulaciones numéricas ya hechas.

- Expresar vectores de posiciones y velocidades de los centros de masa de los fragmentos (protoestrellas).
- Reconocer los criterios para determinar las masas asignadas a dichos fragmentos.
- Simular la evolución de las protoestrellas utilizando un algoritmo de N-Cuerpos en el lenguaje Python.
- Analizar los resultados en el contexto de la dinámica estelar (mecánica clásica).

Simulaciones por computadora y modelos matemáticos

El fin primario del trabajo de un científico, es dar una explicación a cualquier fenómeno que ocurre a nuestro alrededor. En física, estos se explican por medio de modelos matemáticos los cuales Bellomo y Preziosi (1995) describen como un conjunto de ecuaciones que pueden utilizarse para calcular la evolución espacio-temporal de un sistema físico. A esta definición podemos agregar la necesidad de conocer valores iniciales o a la frontera de nuestro modelo, incluso datos intermedios.

Una simulación es cuando aplicamos un modelo matemático con el fin de llegar a estrategias que nos ayuden a resolver un problema o responder a una pregunta relacionada con un fenómeno determinado (Velten, 2009), ahora bien, en la actualidad es posible llevar ese modelo matemático a simulaciones hechas por computadora.

El ser humano, en sus capacidades normales es capaz de simular problemas sencillos de manera manual como por ejemplo, saber a qué hora llegará un tren a su destino sabiendo su velocidad y a dónde va, incluso más complejas como calcular cantidades de energía necesarias para que opere una planta de trabajo. El problema comienza cuando dejan de ser problemas que involucran contadas variables y se empieza a trabajar con cientos, miles o incluso millones. Esta complejidad implica un gran reto, el principal es el tiempo, aunque un segundo es que a veces nuestros conocimientos no nos permiten resolver algo de manera directa y tenemos que recurrir a métodos matemáticos, que llegan a ser muy

laboriosos y el camino lógico es ayudarnos en la tecnología, computadoras y software especializado (Santana Ortega, 2018) .

Entre los distintos tipos de simulaciones, nos enfocamos en conocer las que se hacen a partir de modelos mecánico, los cuales, usan información de los mecanismos internos de propio sistema, aquí conocemos bien que es lo que pasa dentro de la resolución y evitamos las llamadas “cajas negras” (Velten, 2009).

Por “conocer el funcionamiento interno” nos referimos a que nuestro sistema en estudio estará descrito por ecuaciones bien definidas ya sea por una ley física o un principio de ingeniería previamente resuelto en papel y del cual esperamos un resultado conocido. Aplicar las leyes de Newton o de Coulomb sobre partículas con masa y cargadas, respectivamente, son buen ejemplo sobre crear un modelo mecánico y simularlo.

Como ya mencionamos, el reto es el tiempo. Partamos de un cálculo sencillo, digamos que una persona puede dedicarle hasta 10 horas de trabajo al día a resolver operaciones matemáticas sencillas (sumas, multiplicaciones, etc.) y si tomamos un estimado de que realice unas 100 de estas por hora, al día logrará tener 1000 problemas resueltos, inclusive aunque pudiera hacer 1000 por hora, apenas haría 10000 al día. Si bien parece un número muy grande, es pequeño con la cantidad de operaciones que necesita una simulación de muchos cuerpos o partículas interactuando entre sí, que además los cálculos ya no serán operaciones matemáticas sencillas, sino que su grado de complejidad puede llegar a ser muy alto, llegando a extremos de resolver una ecuación en un día o dos.

Hagamos un segundo cálculo, nuestro trabajo va de la interacción gravitatoria entre un número grande de partículas, pero tomemos solo 100, sin que nos importe si colisionan u otras cosas. La premisa es que cada partícula va a interactuar con sus otras 99 vecinas, por lo que, para tener una respuesta de cual es el estado del modelo tendríamos que revisar 9900 interacciones, que si fuéramos a la

una capacidad humana normal, nos tomaría entre 1 a 9 días terminar y claro, la complejidad aumenta si queremos más parámetros y las simulaciones que queremos son de millones de partículas.

Toca analizar un poco a las computadoras, con ayuda de los FLOPS que es la unidad que determina el número de operaciones de punto flotante por segundo que se pueden realizar. Una laptop convencional de uso personal, trabaja a una capacidad de unos 17 gigaflops (Garrido, 2018) o bien hace 17 mil millones de operaciones por segundo, un número exorbitantemente mayor que lo que puede hacer un ser humano. No obstante con eso, existen también las super computadoras, que son mega estructuras pensadas para hacer cálculos muy complejos, principalmente destinadas a la investigación científica. En México tenemos a ABACUS, Laboratorio de Matemática Aplicada y Cómputo de Alto Rendimiento del Departamento de Matemáticas, una super computadora con una capacidad aproximada de 430 teraflops, lo que equivale a unas 25 mil laptops (Bonilla, 2018).

Todo lo anterior nos da una buena premisa de la importancia que tiene incurrir en las simulaciones por computadora, rompemos el factor más importante, el tiempo. No solo por la cantidad de operaciones, pero algo maravilloso es que podemos simular eventos que tardan miles de años en que siquiera puedan observarse cambios pequeños. Justamente en la astronomía, los eventos cosmológicos como la formación de planetas, tardan miles o incluso millones de años, pero con las computadoras y los modelos matemáticos que los describen, podemos acortar el tiempo a unas cuantas horas, días e incluso semanas, pero que en comparativa al evento real, es muy poco.

Simulaciones astronómicas

Dentro de las ciencias naturales, la astronomía resulta ser de las más difíciles en cuanto a su estudio y no necesariamente por su grado de complejidad matemática, si no, debido a que es la única rama de la física que tenemos que estudiar basada únicamente en observación, ya que resulta muy difícil, por no decir imposible, recrear en un laboratorio una supernova, una colisión de planetas o un agujero negro (Solbes, 2011). Por esa razón la forma más práctica de corroborar la teoría es la simulación.

No están dentro de los objetivos del presente trabajo describir los modelos matemáticos que nos llevan a simulaciones muy complejas, nos enfocaremos particularmente en un tipo específico, la evolución de una nube de gas, compuesta por algunos millones de partículas que, al sufrir un colapso gravitatorio, estas se agrupan para formar cúmulos proto-estelares o proto-planetarios. Lo anterior funciona bien como un resumen muy práctico y fácil de entender ya que detrás de todo esto hay una complejidad matemática que amerita su propio desarrollo teórico.

Actualmente tenemos muchas herramientas que se han desarrollado por investigadores de alto nivel en las simulaciones cosmológicas. La que se usa para realizar las simulaciones que analizaremos es una llamada GADGET-2, un código escrito con el fin de hacer más sencillo el estudio del espacio, “La estructura principal de GADGET-2 es la de un código TreeSPH, donde las interacciones gravitatorias se calculan con una expansión multipolar jerárquica, y la dinámica del gas se sigue con hidrodinámica de partículas suavizada (SPH). El gas y la materia oscura sin colisiones¹ están representados por partículas en este esquema. este esquema” (Springel, 2005). Además de estos algoritmos, se incluye un método básico de N-cuerpos para representar un fluido sin colisiones, usado en la mayoría de los códigos de simulación cosmológica.



Figura 2. Ejemplo del resultado obtenido por simulaciones en GADGET-2. Tomado de Colliding Disk Galaxies, <https://wwwmpa.mpa-garching.mpg.de/gadget/collision/index.html>, Springel 2005.

Las simulaciones de este tipo suelen tener una duración característica, sucede que conforme pasa el tiempo, se vuelve tedioso incluso para una supercomputadora, seguir con los cálculos. Resulta más práctico tomar a los nuevos cúmulos formados como objetos puntuales que seguir con los millones de partículas. Por ejemplo, en la Figura 2 notamos que en cierto momento ya se observan objetos bien definidos que podemos tomar como objetos independientes con sus propios centros de masa y velocidades, lo que reduce notablemente el trabajo de cómputo y podemos llevar la simulación a tiempos mucho mayores.

Extracción de datos a partir de simulaciones

Hemos planteado ya un par de argumentos que se resumen en pasar de algo complejo a algo más manejable, o bien, de trabajar con millones de partículas a trabajar con contados cuerpos como objetos puntuales.

Las simulaciones echas por GADGET-2 arrojan distintos valores útiles, pero en su mayoría son datos por cada partícula en la simulación como la densidad, la velocidad, la posición (Thompson, Davé, & Nagamine, The rise and fall of a challenger: the Bullet Cluster in Λ cold dark matter simulations, 2015) para cada tiempo específico en forma de un archivo de nombre `snapXXXX.X`. Este archivo se trata de un binario que debe de ser leído por un programa externo.

Para extraer los datos de cada *snap* salida de GADGET-2, utilizaremos un código escrito en el lenguaje Python desarrollado por Thompson (2014) llamado `pygadreader`, el cuál, entre sus muchas funciones, nos interesan las que nos dan los valores posición, velocidad y densidad de cada una de las partículas.

Los datos extraídos no representan apenas nada, es si no hasta que se les da un tratamiento y les toma como un conjunto que adquieren un peso mayor. Como es de esperarse, la posición (\mathbf{r}) y la velocidad (\mathbf{v}) las tenemos en formas de componentes vectoriales (componentes cartesianos en x , y y z), cada

partícula tiene su propio vector que representa dichas cantidades y la densidad, al ser una cantidad escalar, solo tenemos un dato.

Como podemos observar en la Figura 2, las partículas en la simulación pueden unirse por interacción gravitatoria y formar cuerpos bien definidos. Con los datos de posición, velocidad y densidad individuales, encontraremos cantidades propias de dichos cuerpos, la posición y la velocidad del centro de masa (CM) y la masa total de este nuevo cuerpo. Estas tres cantidades dotan al conjunto de partículas el carácter de un objeto individual.

El centro de masa es un punto en un sistema físico compuesto por varios objetos, en el cual se concentra la masa total del sistema. Es un punto geométrico que se puede calcular promediando las posiciones de todas las partículas que componen el sistema, con relación a su masa. Es un punto de referencia importante en la dinámica de sistemas compuestos ya que las leyes de la física, como la conservación del momento lineal, se refieren al movimiento de los cuerpos con respecto al centro de masa del sistema (Goldstein, 1987). El centro de masa también se conoce como centro de gravedad, es el punto en el cual se concentra la masa de un sistema y es el punto alrededor del cual gira el sistema en equilibrio, es decir, si el sistema estuviera en reposo.

El principio de conservación lineal del momento nos dice como en un sistema lineal el momento $\mathbf{p} = m\mathbf{v}$ se conserva, dicho de otra forma, la suma de las contribuciones que tiene cada partícula al sistema es siempre la misma. Podemos replantear este concepto en una forma útil con el centro de masa. Sabemos que tenemos muchas partículas con masas m_1, m_2, \dots con coordenadas r_1, r_2, \dots respectivamente (Young & Freedman, 2013), definimos al centro de masa del sistema de la siguiente manera:

$$\mathbf{r}_{CM} = \frac{m_1\mathbf{r}_1 + m_2\mathbf{r}_2 + m_3\mathbf{r}_3 + \dots}{m_1 + m_2 + m_3 + \dots} = \frac{\sum_i m_i\mathbf{r}_i}{M}, \quad (3)$$

donde M es la suma de todas las masas individuales, la cual podemos ver como la masa total del sistema.

Pero para comprender la importancia del centro de masa de un conjunto de partículas, tenemos que

preguntarnos qué ocurre cuando este se mueve, es fácil encontrar la velocidad del centro sabiendo que $\frac{dr}{dt} = \mathbf{v}$ (Young & Freedman, 2013), por lo que la ecuación vectorial para la velocidad del centro de masa es:

$$\mathbf{v}_{CM} = \frac{m_1\mathbf{v} + m_2\mathbf{v}_2 + m_3\mathbf{v}_3 + \dots}{m_1 + m_2 + m_3 + \dots} = \frac{\sum_i m_i\mathbf{v}_i}{M}, \quad (4)$$

que son muy similares entre sí y en ambas se involucra la masa total del sistema.

Una vez que el sistema ha adquirido la calidad de un cuerpo hemos logrado el objetivo de reducir el trabajo de trabajar con millones de partículas a apenas una cuantas y podemos buscar nuevos métodos de simulación que no necesiten una gran cantidad de recursos y puedan ser llevados a tiempos mucho más lejanos.

Problema de N-cuerpos

En mecánica clásica existe un problema muy común que nace como una consecuencia directa de aplicar las leyes de Newton, en particular la segunda ley (2) y la Ley de la Gravitación Universal (1), se le conoce como el problema de los N-cuerpos.

En el caso de N cuerpos, cada cuerpo experimenta una serie de fuerzas gravitatorias debidas a todos los demás cuerpos. Estas fuerzas se pueden calcular utilizando la ley de gravitación universal entre cada par de cuerpos. Una vez calculadas estas fuerzas, se pueden calcular las aceleraciones de cada cuerpo utilizando la segunda ley de Newton, que establece que la aceleración de un cuerpo es igual a la fuerza neta que actúa sobre él dividida por su masa (Newton, 1687).

Supongamos un sistema con N partículas puntuales, como lo vemos en la Figura 3, interactuando entre ellas bajo interacción gravitacional, cada una posee una masa m_i y un vector de posición \mathbf{r}_i . La ecuación (1) nos ayuda a entender que para dos cuerpos existe una fuerza proporcional a sus masas y al inverso cuadrado de la distancia entre ellas, pero al agregar una partícula más, esta interactuaría con las dos que ya estaban previamente.

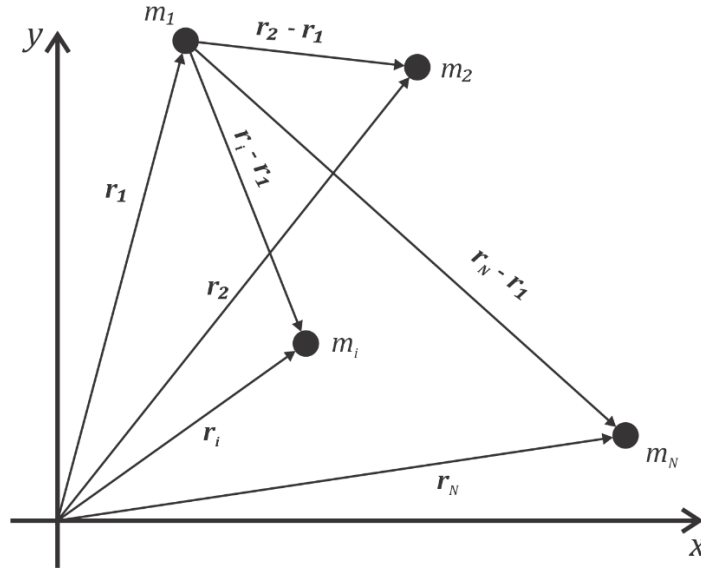


Figura 3. Sistema de N-cuerpos. Cada cuerpo de masa m_i interactúa con cada uno de los cuerpos adyacentes.

Sabemos que se trata de un sistema lineal y conservativo, por lo que cumple con las leyes de Newton, las cuales nos dicen que la aceleración que tiene cada partícula es la suma individual de todas las contribuciones, por el principio de superposición, la fuerza de atracción que siente una partícula i con respecto a una partícula j es igual a:

$$\mathbf{F}_{ij} = -G \frac{m_i m_j}{|\mathbf{r}_i - \mathbf{r}_j|^3} (\mathbf{r}_i - \mathbf{r}_j), \quad (5)$$

la fuerza tiene la misma dirección del vector que separa a ambas partículas y sabemos, por la tercera ley de Newton, que la partícula j atrae a la partícula i con una fuerza \mathbf{F}_{ji} , de igual magnitud, pero con signo contrario (Newton, 1687).

Ahora bien, para encontrar la aceleración total que siente la partícula i y posteriormente aplicar la segunda ley de Newton $\mathbf{F} = m\mathbf{a} = m\ddot{\mathbf{r}}$, para ello tenemos que saber la fuerza total sobre esta, que es la suma de todas las contribuciones, tal que:

$$\ddot{\mathbf{r}}_i = \frac{\mathbf{F}_i}{m_i} = -G \sum_{j \neq i}^N \frac{m_j}{|\mathbf{r}_i - \mathbf{r}_j|^3} (\mathbf{r}_i - \mathbf{r}_j). \quad (6)$$

Esta expresión se trata de una ecuación diferencial de segundo orden con respecto al tiempo que al resolverla nos da la posición de la partícula i , por lo tanto, su velocidad y su aceleración ya que:

$$\mathbf{v} = \frac{d\mathbf{r}}{dt} \quad \& \quad \mathbf{a} = \frac{d\mathbf{v}}{dt} = \frac{d^2\mathbf{r}}{dt^2}. \quad (7)$$

Con todo lo anterior podemos conocer la evolución del sistema resolviendo las N ecuaciones correspondientes a cada una de las N partículas. La parte complicada en resolver la ecuación (6), teniendo que usar cualquier método matemático o bien algún método numérico con ayuda de las simulaciones por computadora.

Determinación de un método de solución

El problema de los N cuerpos es considerado uno de los problemas abiertos más importantes en la dinámica celeste, ya que es altamente complejo y a menudo no tiene soluciones analíticas. Henri Poincaré fue un matemático francés y uno de los principales contribuyentes al estudio de este problema en la dinámica celeste.

Poincaré se interesó especialmente en el problema de los N cuerpos e hizo contribuciones fundamentales en su estudio. En particular, se concentró en el estudio de las soluciones periódicas y las soluciones caóticas. En su obra "Les Méthodes Nouvelles de la Mécanique Céleste" (1892-1899) Poincaré desarrolló una teoría matemática para describir el movimiento de los cuerpos en el sistema de N cuerpos y mostró la existencia de soluciones periódicas y soluciones caóticas (Bor & Montgomery, 2014).

Nos relata Poincaré (1892-1899) como puede ser la solución de los sistemas antes descritos, básicamente juega un papel importante el número de cuerpos en interacción, las soluciones periódicas aparecen cuando N es un número pequeño, el ejemplo es nuestro sistema solar, donde los planetas describen órbitas circulares. Pero, desde un punto de vista más general, las soluciones a las ecuaciones de movimiento que son periódicas pueden describir trayectorias cónicas (elipse, parábola e hipérbola).

En cuanto a las soluciones caóticas, provienen de sistemas con N muy grande, altas velocidades y distancias relativamente cortas. Como ejemplo son las simulaciones de la Figura 1 o la Figura 2 que requieren métodos de solución mucho más avanzados.

Aguilar (1992) nos habla sobre los sistemas colisionales y no colisionales los cuales están realcionados proporcionalmente con el número de partículas, para pocas partículas y muchas, respectivamente. Tambien nos dice qué método de solución es más conveniente aplicar en una simulación para cada caso. Debido a que nuestro sistema es colisional, basta con aplicar una solución numérica a la ecuación (6) para obtener soluciones explícitas.

Método de integración directa

El método de integración directa es una técnica utilizada para resolver ecuaciones diferenciales numéricamente, consiste en aproximar la solución de la ecuación diferencial en un intervalo de tiempo determinado mediante el uso de una función aproximada. Es un método simple y directo, pero no siempre es preciso. El método es aplicable a una gran variedad de problemas en ciencia e ingeniería, incluyendo la dinámica de sistemas mecánicos, el transporte de calor y la propagación de ondas (Chapra & Raymond, 1988). Hay varios métodos de integración directa, algunos de los más populares son:

- El método de Euler: es el método más simple y directo, se basa en una aproximación lineal para calcular los cambios en las variables a lo largo del tiempo.
- El método de Runge-Kutta: es un método más preciso que el método de Euler, utiliza cuatro pasos intermedios para calcular la posición y velocidad en un instante dado y luego los promedia para obtener una solución más precisa.
- El método de Taylor: es un método que se basa en la expansión en serie de Taylor para calcular la solución de una ecuación diferencial en un intervalo de tiempo determinado.

Nuestro problema por resolver se trata de uno con valores iniciales, el cual es fácilmente resuelto aplicando cualquiera de estos métodos, pero nos enfocaremos en el método de Euler que consiste en dividir el intervalo temporal en pequeños pasos (dt) y utilizar las aceleraciones calculadas en el paso anterior para actualizar las velocidades y posiciones de los cuerpos en el paso siguiente. Este proceso se repite para cada paso temporal hasta que se alcanza el tiempo final de la simulación.

Metodología

La fase del desarrollo experimental fue llevada a cabo en dos fases principales, en ambas se llegó al desarrollo de programas capaces de hacer lo planteado en los objetivos. Aunque ambas fases suceden por separado, una ocurre justo hasta que la primera se concluye.

En el siguiente diagrama de la Figura 4 podemos observar de manera resumida las fases del trabajo experimental y sus principales sub-fases.

Todo el trabajo experimental fue desarrollado en computadora y principalmente usando el lenguaje de programación Python, muy bueno para trabajar con cantidades de datos grandes.

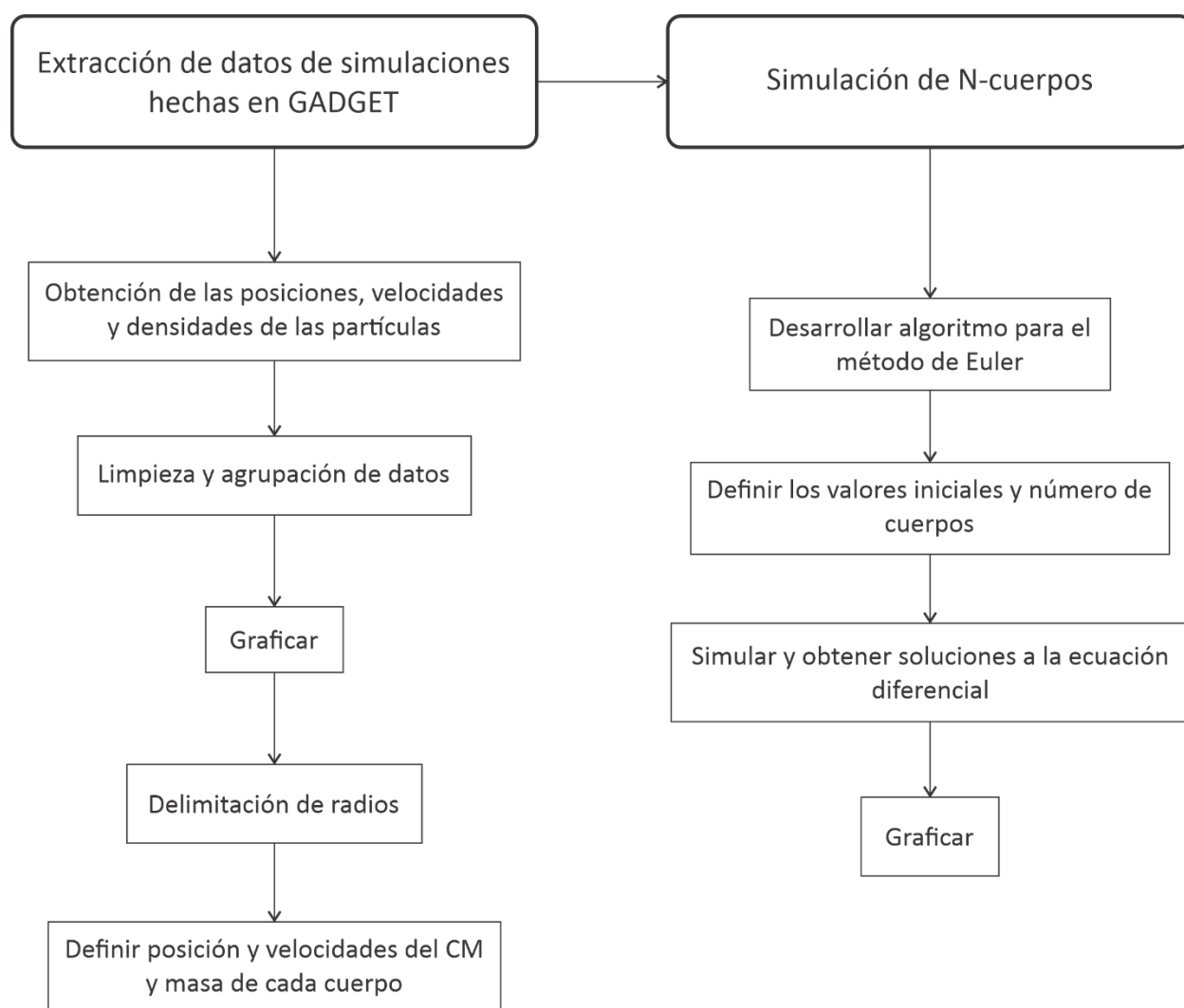


Figura 4. Diagrama de flujo del trabajo experimental.

Extracción y manipulación de datos a partir de simulaciones hechas en GADGET

Las simulaciones hechas en GADGET entregan como salida archivos binarios con el nombre de “snapshot_xxxx.x” con un número que simboliza el avance de la simulación en el tiempo. Cada archivo cuenta con un encabezado con datos generales como el número total de partículas y más datos que nos dicen los parámetros de cómo fue hecha la simulación. El contenido general está dado por los datos de cada partícula como la posición, la velocidad, la densidad, etc. con un identificador específico para cada partícula. Cada *snap* está conformada de ocho partes que en su conjunto contienen todos los datos.

Se me entregaron 19 snaps de la 0070, saltando de diez en diez hasta la 0250, correspondientes a una simulación de un disco protoplanetario en fragmentación compuesto de partículas de gas. Con ayuda del lenguaje Python fui realizando cada una de las tareas hasta tener un programa funcional para la extracción y manipulación de los datos de los archivos binarios procedentes de simulaciones en GADGET. Para describir el desarrollo se utiliza como ejemplo el *snap* 0090.

Nota: El código entero, así como las bibliotecas usadas, se encuentran en la parte de entregables y se infiere que el lector conoce tipos de datos y funciones en Python.

Obtención de las posición, velocidad y densidad de cada partícula

La primera labor fue obtener todos los datos relevantes de las partículas en la simulación. Usé la biblioteca *pygadgetreader* como herramienta de apoyo, ya que, extrae los datos de los archivos binarios de GADGET y los entrega en forma de arreglos que se manipulan fácilmente con Python. El complemento tiene dos funciones `readheader()` y `readsnap()` a las cuales se les deben incluir parámetros para su funcionamiento. La primera lee el encabezado y arroja datos generales, la segunda nos arroja datos específicos según le indiquemos.

En la Tabla 1 podemos ver los parámetros necesarios para el funcionamiento de las funciones antes mencionadas. Los parámetros listados son solo los que usamos en el presente trabajo, la lista

completa de parámetros se encuentra en la página del creador (Thompson, pyGadgetReader: GADGET snapshot reader for python, 2014).

Tabla 1. Parámetros de las funciones de pygadgetreader.

readsnap(a,b,c)			readheader(a,b)	
a	b		a	b
Nombre del archivo como cadena de caracteres	Dato que se quiere extraer		Nombre del archivo como cadena de caracteres	Dato que se quiere extraer
	"pos"	Posición		
	"vel"	Velocidad		
	"pid"	Identificador		
	"rho"	Densidad		"header"
		Tipo de partícula		
		"gas"		

Un ejemplo del resultado de usar estas funciones es el siguiente:

```
readheader("/mnt/d/pt/snaps/snapshot_0090", "header")
{'npartThisFile': array([323668, 0, 0, 0, 0, 0], dtype=uint32),
 'npartTotal': array([2400002, 0, 0, 0, 0, 0], dtype=uint32),
 'npartTotalHW': array([0, 0, 0, 0, 0, 0], dtype=uint32),
 'ngas': 2400002, 'ndm': 0, 'ndisk': 0, 'nbulge': 0, 'nstar': 0, 'nbdry': 0,
 'massTable': array([4.16666319e-07, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00]), 'time': 10.899999963119628}
```

```
readsnap("/mnt/d/pt/snaps/snapshot_0090.0", "vel", "gas")
Returning GAS Velocities
array([[ -0.16684662, -0.40250614, -0.03402008],
       [ -0.17216721, -0.40550575, -0.02126304],
       [ -0.16765852, -0.39772075, -0.04697778],
       ...,
       [ -0.16615996, -0.40389287,  0.0036071 ],
       [ -0.1664789 , -0.40987742,  0.01706672],
       [ -0.17326403, -0.4044617 ,  0.02782467]], dtype=float32)
```

Código 1. Ejemplo del funcionamiento básico de las funciones de pygadgetreader.

Cabe mencionar que para la primera función se menciona el nombre de la snap, mientras que para la segunda se tiene que mencionar qué parte se desea leer. En el ejemplo podemos ver que los datos, en este caso la velocidad, son guardados en arreglos, los cuales no tienen un orden específico.

Las unidades de todos los datos extraídos no vienen en unidades del sistema métrico, pero para usos prácticos se trabajó así hasta su análisis.

Por cada *snap* juntaron sus ocho partes en un mismo arreglo. El rango total de la simulación es de 5 unidades, pero, como veremos más tarde, la mayor concentración de partículas se encuentra un radio de 0.5 unidades, por lo que, valió la pena almacenar las partículas que se encuentran dentro de este radio de acción y así evitar tener listas demasiado grandes.

```
def reducir_rango():
    for e in range(8):
        pid_pr = readsnap(open_snap[e].name, "pid", "gas")
        pos_pr = readsnap(open_snap[e].name, "pos", "gas")
        rho_pr = readsnap(open_snap[e].name, "rho", "gas")
        vel_pr = readsnap(open_snap[e].name, "vel", "gas")
        for i in range(len(pid_pr)):
            if -0.5 <= pos_pr[i][0] <= 0.5 and -0.5 <= pos_pr[i][1] <= 0.5 and -0.1 <= pos_pr[i][2] <= 0.1:
                pid.append(pid_pr[i])
                pos_x.append(pos_pr[i][0])
                pos_y.append(pos_pr[i][1])
                pos_z.append(pos_pr[i][2])
                vel_x.append(vel_pr[i][0])
                vel_y.append(vel_pr[i][1])
                vel_z.append(vel_pr[i][2])
                rho.append(rho_pr[i])
```

Código 2. reducir_rango(), función para leer los datos del snap, almacenarlos y reducir los datos útiles dentro del rango de acción.

Los datos de posición y velocidad son entregados como vectores, pero en forma de sus componentes, en este caso cartesianas. Cada dato quedó en un arreglo diferente, los ya mencionados más la ID y la densidad.

Limpieza y agrupación de datos

Python trabajo con una biblioteca llamada Pandas que en resumen es de gran ayuda para trabajar con miles o millones de datos, debido a que nuestras listas o arreglos son demasiado grandes, es más conveniente el uso de esta herramienta.

Dentro de Pandas, existe una característica llamada *DataFrame*, que almacena y relaciona listas de manera indexada en forma de tablas de datos. Se eligió trabajar con estas hojas de datos ya que cuenta con funciones como reacomodar filas y columnas, conocer máximos, reducir valores, buscar datos nulos o repetidos, etc.

En el Código 3 podemos ver la forma de inicializar el dataframe agregando los arreglos resultantes del Código 2 como columnas. De igual manera, se detectó que las listas estaban llenas de datos repetidos que venían de manera natural del archivo binario, por fortuna el ip de cada partícula es único e irrepetible así que se pudo limpiar fácilmente la tabla al quitar los duplicados.

```
df = pd.DataFrame()
df["Id"] = pid
df["Posicion_x"] = pos_x
df["Posicion_y"] = pos_y
df["Posicion_z"] = pos_z
df["Velocidad_x"] = vel_x
df["Velocidad_y"] = vel_y
df["Velocidad_z"] = vel_z
df["Densidad"] = rho
df = df.drop_duplicates(df.columns[~df.columns.isin(["Id"])],keep="first")
```

Código 3. Inicialización de dataframe y eliminación de datos duplicados.

	Id	Posicion_x	Posicion_y	Posicion_z	Velocidad_x	Velocidad_y	Velocidad_z	Densidad
0	2348938	-0.019539	-0.003666	-0.089040	-0.263020	-1.088658	1.033926	0.374727
1	2348949	-0.020241	-0.020202	-0.090399	-0.248155	-1.037061	1.031488	0.381036
2	265878	-0.042690	-0.051366	-0.090035	0.234082	0.242352	-0.607920	0.449911
3	266403	-0.033614	-0.042980	-0.089398	0.096367	0.235966	-0.286497	0.434445
4	2329938	-0.041879	-0.033039	-0.094871	-0.125000	-1.116317	0.991198	0.358683
...
1107416	2190429	-0.039667	-0.040203	0.098622	0.229091	0.307075	0.673285	0.390861
1107417	2200658	-0.042152	-0.030328	0.099753	0.072897	0.280757	0.396550	0.361582
1107418	2200303	-0.049042	-0.027547	0.099404	0.119411	0.403079	0.493284	0.360929
1107419	501858	-0.091654	-0.031396	0.099418	0.162244	-1.283953	-0.705328	0.349025
1107420	501852	-0.086090	-0.067484	0.099966	0.121768	-1.229459	-0.889341	0.432845
1107421 rows × 8 columns								

Figura 5. DataFrame resultante, resumido por la cantidad tan grande de datos.

La tabla resultante no tiene un orden en particular, pero fue indexada por orden en que cada dato fue agregado. La Figura 5 nos da el dato de que se está trabajando con 1,107,421 partículas, que son casi la mitad del total en un radio que es apenas una pequeña fracción del radio total, cosa que nos ayuda a comprobar que la mayoría de las partículas se encuentra aquí y las demás dispersas en todo el volumen restante.

El fin primero de esta extracción de datos es generar una imagen gráfica de la simulación, podríamos simplemente graficar cada partícula con sus coordenadas y distinguir a cada una por el valor de su densidad. Las limitantes a esto son técnicas, la primera es el poder de procesamiento, las computadoras convencionales tardan cuando se trata de representaciones gráficas y la segunda limitante es la resolución, normalmente buscamos obtener imágenes de buena calidad, pero manipulables, por lo que, graficar los datos así haría que se sobrepongan datos en un mismo píxel.

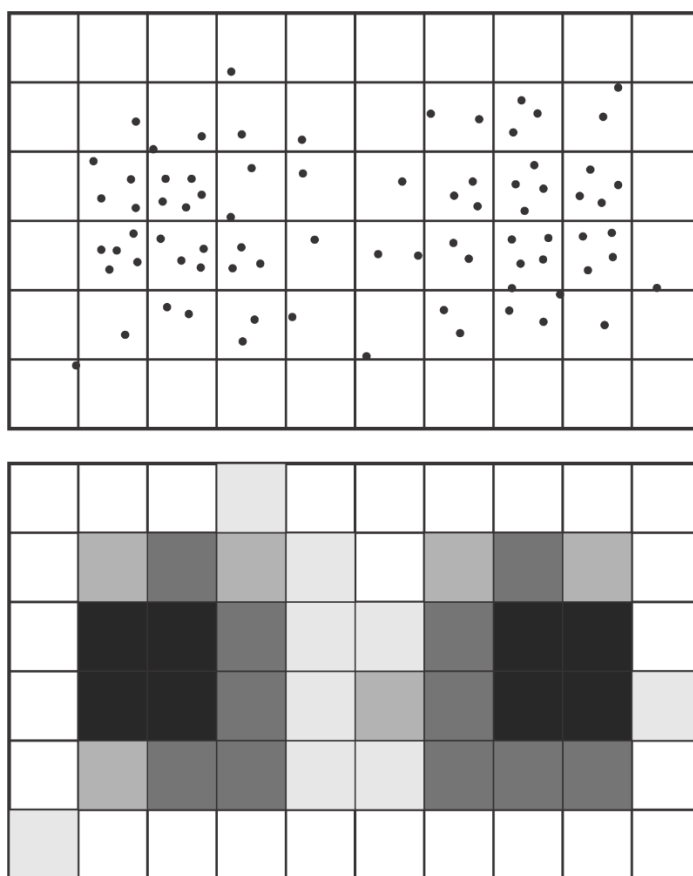


Figura 6. Representación del método de agrupación y reducción.

Se optó por un método de agrupación en pequeñas áreas, tomar el área de trabajo y dividirla en cuadros pequeños, la idea es tomar el promedio de la densidad de todas las partículas que se encuentran dentro y asignarle nuevas coordenadas a cada cuadro, así reducir el número de datos a graficar, de manera esquemática lo podemos ver en la Figura 6.

Desde el *dataframe* se pudo hacer esta agrupación, ya que estas hojas de datos permiten agrupar valores iguales y promediar los valores dependientes. El primer paso fue hacer un redondeo de decimales de los valores de posición en x y y , como vemos en la Figura 5, se trabaja con hasta 6 decimales, por lo que, podemos reducirlos a 2 para que al agruparlos las distancias entre cuadrados sean de 0.01 unidades. En el renglón 3 del Código 4 guardamos una copia del *dataframe* original solo quitando el ID, ya que no nos servirá más. Ahora quitamos también los datos de posición en z y las velocidades para por último se agrupó por los valores de posición x y y los datos de densidad.

```
df["Posicion_x"] = np.round(df["Posicion_x"], decimals=2)
df["Posicion_y"] = np.round(df["Posicion_y"], decimals=2)
df_complp = df.drop(["Id"], axis=1)
df = df.drop(["Id"], axis=1)
df = df.drop(["Posicion_z"], axis=1)
df = df.drop(["Velocidad_x"], axis=1)
df = df.drop(["Velocidad_y"], axis=1)
df = df.drop(["Velocidad_z"], axis=1)
df = df.groupby(by=["Posicion_x", "Posicion_y"]).mean()
```

Código 4. Reducción de decimales, eliminación de valores innecesarios y agrupación de datos.

La respuesta al método anterior fue una nueva tabla de datos (Figura 7) que ha reducido su tamaño de poco más de un millón de datos a apenas unos diez mil, que es poco menos del uno por ciento. Este nuevo dato de densidad como si fuera una nueva partícula con una posición nueva, pero que sigue representando lo mismo, pero a una resolución más manejable de 100 x 100.

Debemos de mencionar que existen más maneras de llegar al mismo resultado, pero para optimizar es que se tomó la decisión de trabajar con el método que ya se estaba usando y ahorrar mucho código, qué es de por sí una máxima en el mundo de la programación. Además de que fácilmente se puede cambiar de resolución a potencias de 10, 1000 x 1000, por ejemplo.

		Densidad
Posicion_x	Posicion_y	
-0.5	-0.50	0.449358
	-0.49	0.500830
	-0.48	0.628414
	-0.47	0.837028
	-0.46	1.043918
...		...
0.5	0.46	1.348632
	0.47	0.998623
	0.48	0.713980
	0.49	0.489915
	0.50	0.429491

10183 rows × 3 columns

Figura 7. Nueva hoja de datos con los valores agrupados y reducidos.

Representación gráfica

Quizás el paso más fácil debido a que Python grafica de manera sencilla, pues basta con declarar quien será x y quien y , además de que puede hacerlo directamente desde la hoja de datos. Pero hay un paso intermedio que se hizo ya que la tabla de la Figura 7 no es apta para graficar, se puede ver el Código 5 que se crea una nueva *dataframe* que es la que se estará usando.

```
x_graf = []
y_graf = []
for i in range(len(df)):
    x_graf.append(df.index[i][0])
    y_graf.append(df.index[i][1])
dfn = pd.DataFrame()
dfn["Posicion_x"] = x_graf
dfn["Posicion_y"] = y_graf
dfn["Densidad"] = df["Densidad"].values
```

Código 5. Paso intermedio para que los datos sean aptos para graficarse.

Para fines analíticos, además de la representación de las partículas, se tomó la decisión de hacer también gráficas de la densidad con respecto a las posiciones por separado. Los códigos los podemos ver a continuación.

```
ax1 = plt.axes()
plt.plot(dfn["Posicion_x"], dfn["Densidad"])
plt.title(snap)
plt.xlabel("x")
plt.ylabel(r"$\rho$")
ax1.yaxis.grid(True, which='major')
```

Código 6. Método para generar gráfica de x vs Densidad

```
ax1 = plt.axes()
dfaux = pd.DataFrame()
dfaux["Posicion_x"] = list(dfn.sort_values("Posicion_y")["Posicion_x"])
dfaux["Posicion_y"] = list(dfn.sort_values("Posicion_y")["Posicion_y"])
dfaux["Densidad"] = list(dfn.sort_values("Posicion_y")["Densidad"])
plt.plot(dfaux["Posicion_y"], dfaux["Densidad"])
plt.title(snap)
plt.xlabel("y")
plt.ylabel(r"$\rho$")
ax1.yaxis.grid(True, which='major')
```

Código 7. Método para generar gráfica de y vs Densidad.

```
ax = plt.axes()
s = plt.scatter(dfn["Posicion_x"], dfn["Posicion_y"], c=dfn["Densidad"],
               cmap="turbo", norm=colors.PowerNorm(0.3))
c = plt.colorbar()
plt.xlabel("x")
plt.ylabel("y")
plt.title(snap)
ax.yaxis.grid(True, which='major')
ax.xaxis.grid(True, which='major')
plt.clim(vmin=df["Densidad"].min(), vmax=df["Densidad"].max())
```

Código 8. Método para generar la representación gráfica de la simulación.

Tenemos que hacer mención de que en el segundo código (Código 7) se creó una *dataframe* auxiliar con el fin de reagrupar los datos para que al hacer la gráfica se viera de mejor manera. De igual manera, se le aplicó una normalización logarítmica propia del método para graficar a la escala de colores (Código 8) para que sean perceptibles, ya que la densidad varía desde fracciones de unidad hasta decenas de miles.

Para fines demostrativos del desarrollo experimental, colocamos a continuación las gráficas y la representación de la simulación. Las correspondientes a las otras 18 snaps se mostrarán en la sección de resultados.

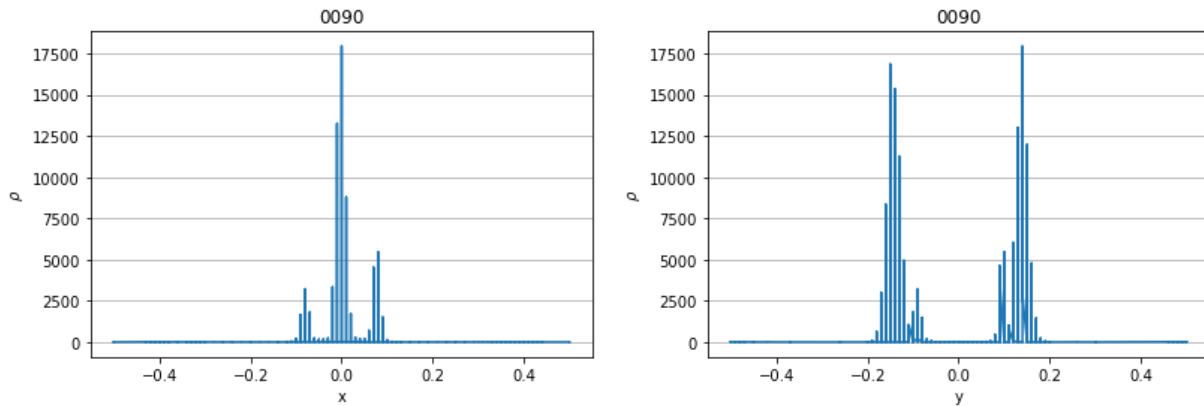


Figura 8. Gráficas de densidades con respecto al eje x y al eje y . Snap 0090.

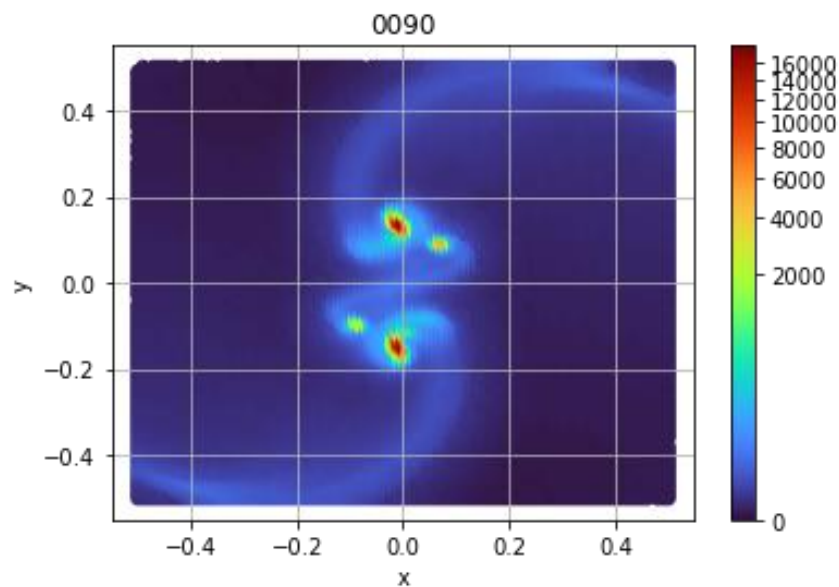


Figura 9. Representación gráfica del Snap 0090.

Delimitación de radios de los nuevos objetos

Podemos decir un par de cosas a partir de la representación gráfica con respecto a los objetos que aparecen. Estos pequeños círculos mostrados en rojo y en verde son los objetos que se van formando cuando sucede el fraccionamiento del disco protoplanetario.

Por el momento nos interesa solo conocer un poco más de estos nuevos objetos como un conjunto de partículas ya que para continuar nuestro trabajo necesitamos los centros y las velocidades de los centros de masa, además de la masa de cada punto, ya que hasta este momento seguimos conservando los datos de las partículas.

Ya que todo el proceso queda automatizado, los pasos a seguir son los siguientes: primero tenemos saber cuántos objetos tiene nuestra simulación, por las gráficas es obvio, pero debe de ser analíticamente; después hay que delimitar un radio alrededor de cada nuevo objeto para poder sumar las contribuciones de cada partícula dentro de este y encontrar los centros.

Encontrar objetos dentro de la simulación.

Como vimos en las gráficas de densidades, podemos conocer los objetos que aparecen en la simulación fijándonos en los picos de densidad ya que corresponden a los objetos en la representación gráfica como vemos en la Figura 10 y la Figura 11 y con un sencillo escrutinio visual notamos como cada pico corresponde a un objeto.

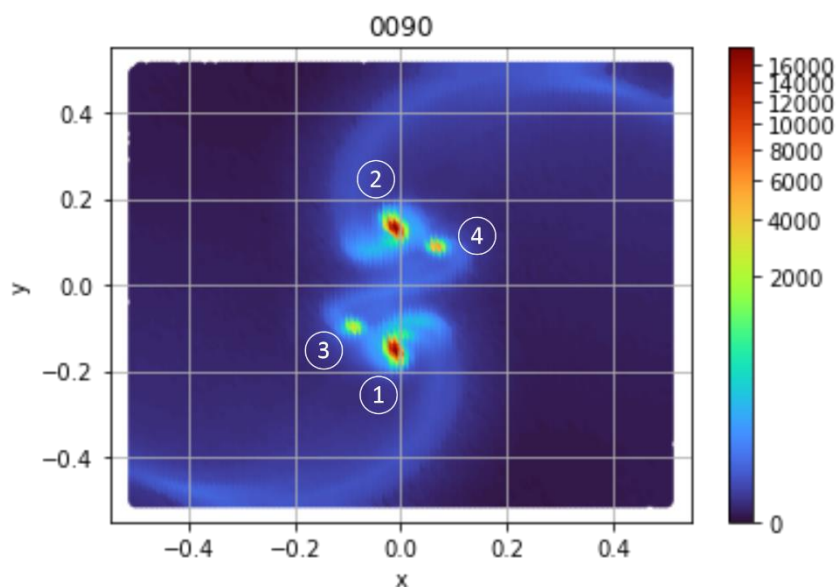


Figura 10. Asignación de objetos de manera visual.

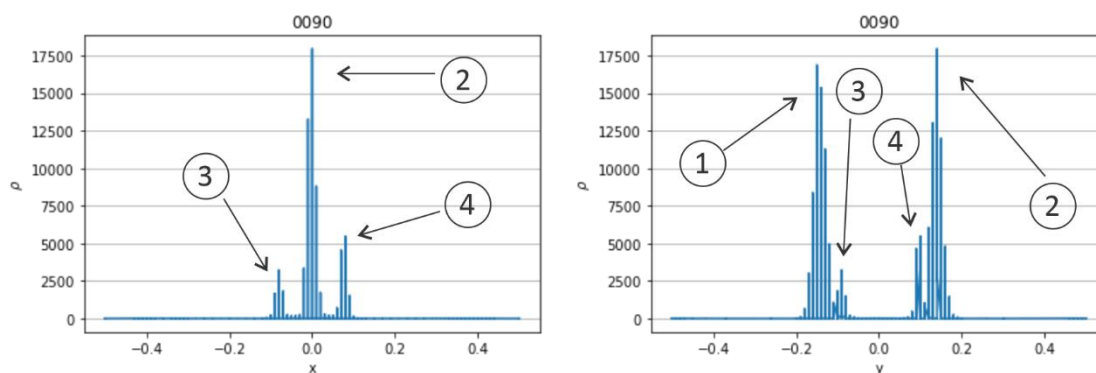


Figura 11. Delimitación de objetos a partir de las gráficas de densidad.

Analíticamente tenemos que cumplir con un par de condiciones para determinar qué es un objeto y qué no, la primera es que se trata evidentemente de un pico o máximo dentro de los datos y la segunda es que este máximo tiene que ser un máximo local, en otras palabras, encontrarse alejado a un radio determinado. Esta segunda condición nace a partir de la naturaleza de los datos, en las gráficas no se aprecia, pero existen picos que el software promedió pero que están en las tablas de datos.

Para el desarrollo del algoritmo se tomó también una altura mínima h , solo con el fin de evitar picos pequeños que seguro existen y que la limpieza de datos sea más práctica desde el principio, esta altura se determinó a un porcentaje del dato máximo de la densidad (6%), por lo que, cada *snap* tendrá una altura propia. En cuanto al radio de separación fue delimitado arbitrariamente a 0.03 unidades.

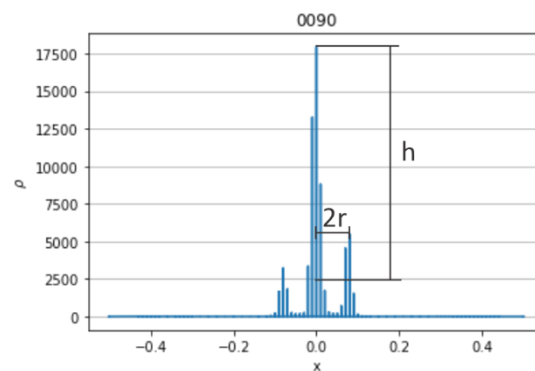


Figura 12. Radio entre picos y delimitación de altura para reducir datos.

El algoritmo desarrollado da el número de objetos de los que consta la *snap* y la posición del pico de densidad. Este consta de dos partes, en la primera se hace una limpieza de los datos, solo se toman en cuenta aquellos que están arriba de la altura h y a su vez se listan todos los picos que sean encontrados con ayuda de la condición:

$$dato[i - 1] < dato[i] > dato[i + 1], \quad (8)$$

en otras palabras, se queda el dato que sea siempre mayor que su dato anterior y posterior, tal como lo vemos en el Código 9.

```
r = 0.03
mini = dfn["Densidad"].max()*0.06
for i in range(len(dfn)):
    if dfn["Densidad"][i] > mini:
```

```

    if dfn["Densidad"][i-1] < dfn["Densidad"][i] and dfn["Densidad"][i] >
dfn["Densidad"][i+1]:
        picos1.append(dfn["Densidad"][i])
        px1.append(dfn["Posicion_x"][i])
        py1.append(dfn["Posicion_y"][i])
for i in range(len(dfaux)):
    if dfaux["Densidad"][i] > mini:
        if dfaux["Densidad"][i-1] < dfaux["Densidad"][i] and dfaux["Densidad"][i] >
dfaux["Densidad"][i+1]:
            picos2.append(dfaux["Densidad"][i])
            px2.append(dfaux["Posicion_x"][i])
            py2.append(dfaux["Posicion_y"][i])

```

Código 9. Primera parte del algoritmo detector de objetos. Limpieza de datos para encontrar los cuerpos de la simulación.

Los datos se almacenas en nuevas variables. El algoritmo se ejecuta dos veces, pero con la diferencia de que la segunda vez utilizamos el *dataframe* auxiliar que creamos para graficar en *y* para tener las dos perspectivas. Tal como lo vemos en la Figura 8 y la Figura 11 habrá ocasiones en donde una sola gráfica no muestre el total de los objetos, en la gráfica respecto a *x* no aparece el objeto 1 ya que está en la misma coordenada en *x* aunque no de *y*.

La segunda parte del algoritmo (Código 10) realiza la búsqueda de estos máximos locales, tomando las listas creadas se hizo una comparación de valores uno a uno, o bien, cada valor *i* se compara con cada valor *e* con $i \neq e$, primero se compara la distancia entre estos, si es mayor al radio *r*, ya no se cuanta, pero si está dentro, se aplica nuevamente la comparación de la expresión (8) salvo que a la primera vez que se encuentre que el valor es más pequeño que cualquier otro, la comparación acaba y en el caso de que sea el mayor dentro del radio, se agrega a la lista de máximos final.

Como en el primer parte del algoritmo, esta segunda parte también se repite una vez más para valores desde el punto de vista de *y*. Los valores encontrados aquí, también se agregan a la lista de máximos finales, así como también se crean listan con las posiciones de dichos máximos. Es evidente que va a haber máximos que se repitan ya que el valor de los picos de densidad es el mismo en ambas perspectivas. Así que simplemente comparamos cada valor con todos los demás y desechamos los que estén repetidos, quedando así la lista con valores únicos y sus respectivas posiciones.

```

for i in range(len(picos1)):
    for e in range(len(picos1)):
        if i != e and abs(px1[i]-px1[e]) < r:
            if picos1[i] < picos1[e]:
                reg = 1
                break
            elif picos1[i] > picos1[e]:
                reg = 2
    if reg == 2:
        maximos.append(picos1[i])
        max_x.append(px1[i])
        max_y.append(py1[i])
for i in range(len(picos2)):
    for e in range(len(picos2)):
        if i != e and abs(py2[i]-py2[e]) < r:
            if picos2[i] < picos2[e]:
                reg = 1
                break
            elif picos2[i] > picos2[e]:
                reg = 2
    if reg == 2:
        maximos.append(picos2[i])
        max_x.append(px2[i])
        max_y.append(py2[i])
ind = []
for i in range(len(maximos)-1):
    for e in range(i+1,len(maximos)):
        if maximos[i] == maximos[e]:
            ind.append(e)
ind.sort()
for i in reversed(ind):
    maximos.pop(i)
    max_x.pop(i)
    max_y.pop(i)
N = len(maximos)

```

Código 10. Segunda parte del algoritmo detector de objetos. Limpieza de datos para encontrar los cuerpos de la simulación.

Corriendo el código del algoritmo buscados de objetos en nuestro ejemplo del *Snap* 0090 tenemos

las siguientes salidas:

```

print(N)
print(maximos)
print(max_x)
print(max_y)

```

4

```
[3216.6191, 17931.557, 5458.54, 16845.564453125]
[-0.07999999821186066, -0.0, 0.07999999821186066, -0.0]
[-0.090000000357627869, 0.14000000059604645, 0.10000000149011612,
0.15000000059604648]
```

Código 11. Salidas al código de búsqueda de objetos.

Estos valores coinciden con las gráficas, pero aún no podemos concluir nada sobre los centros de masa. Para ello aún necesitamos delimitar el radio o los intervalos donde se encuentran conglomeradas la mayoría de las partículas que en su conjunto forman uno de estos nuevos N cuerpos.

Delimitando rangos donde se encuentra el centro de masa.

Se hizo un algoritmo de comparación es simple, si el valor de la densidad es mayor a un porcentaje del máximo local y está más cerca de este que de cualquier otro máximo se queda para delimitar el rango, y el rango se actualizará con el valor de la posición que esté más lejana a la posición del pico.

El criterio que se tomó para la elección del porcentaje fue uno basado en la evidencia gráfica y un poco por sentido común matemático. Como podemos ver en la Figura 12, hay una recta al 1% del máximo, observe como prácticamente todos los valores relevantes de densidad se encuentran por encima de esta línea, por otro lado, de manera analítica sabemos que, para nuestro ejemplo, el 1% de 17,931 (máximo analizado) es de 179.31, valor que parece grande, pero que en comparación no lo es y se aprecia en la gráfica como valores inferiores a este puede hacer que el radio se agrande fuera de la curva trazada que representa a los valores válidos. Por lo que, se decidió dejar como porcentaje al 1%, cosa que también hará que los picos pequeños no se vean afectados.

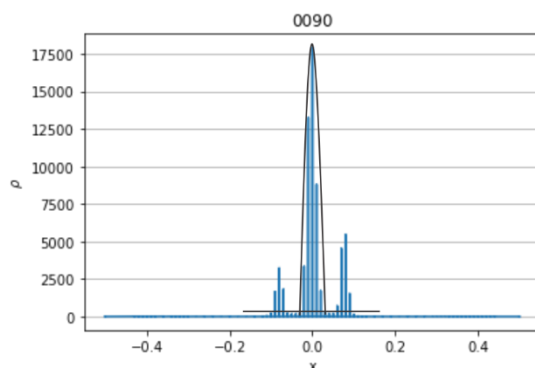


Figura 13. Ayuda gráfica para definir el criterio del porcentaje mínimo para tomar un dato válido

```

def rangos(n):
    rango_i_x = max_x[n]
    rango_f_x = max_x[n]
    rango_i_y = max_y[n]
    rango_f_y = max_y[n]
    for i in range(len(df_complp)):
        check = 0
        if df_complp["Densidad"][i] >= maximos[n]*0.01:
            for e in range(N):
                if e != n and (abs(df_complp["Posicion_x"][i]-max_x[n]) >
abs(df_complp["Posicion_x"][i]-max_x[e]) or abs(df_complp["Posicion_y"][i]-
max_y[n]) > abs(df_complp["Posicion_y"][i]-max_y[e])):
                    check = 1
            if check == 0:
                if df_complp["Posicion_x"][i] < rango_i_x:
                    rango_i_x = df_complp["Posicion_x"][i]
                elif df_complp["Posicion_x"][i] > rango_f_x:
                    rango_f_x = df_complp["Posicion_x"][i]
                if df_complp["Posicion_y"][i] < rango_i_y:
                    rango_i_y = df_complp["Posicion_y"][i]
                elif df_complp["Posicion_y"][i] > rango_f_y:
                    rango_f_y = df_complp["Posicion_y"][i]
            return [rango_i_x, rango_f_x, rango_i_y, rango_f_y]

```

Código 12. rangos(). Función para delimitar los rangos para cada uno de los valores máximo n.

El Código 12 fue escrito como una función que recibe un valor entero que representa a que objeto de la lista de máximos se quiere analizar, regresando los valores de los rangos donde se encuentra el centro de masa.

Datos relevantes de los centros de masas

Este último paso puede resumirse en aplicar las ecuaciones (3) y (4) salvo con el cambio de que no temeos la masa explícita, que partimos de los datos iniciales de la simulación, donde nos dicen que la masa inicial era de una masa solar, por lo que cada partícula tiene una masa igual a:

$$m_i = M_s/2,40 \quad (9)$$

Debido a que queremos datos más exactos, recurrimos un *dataframe* que dejamos con todos los datos y sin modificaciones en el Código 4, es útil debido a que necesitamos el número de partículas involucradas para conocer la masa de cada uno de los N-cuerpos.

El Código 13 utiliza los valores de los rangos ya obtenidos para cada máximo y suma todas las partículas que se encuentren dentro de estos límites para obtener los datos relevantes al centro de masa.

```
def centro_masa(n):
    sum_x, sum_y, sum_z, sumv_x, sumv_y, sumv_z, densi = 0, 0, 0, 0, 0, 0, 0
    cont = 0
    rango = rangos(n)
    for i in range(len(df_complp)):
        if rango[0] <= df_complp["Posicion_x"][i] <= rango[1] and rango[2] <=
df_complp["Posicion_y"][i] <= rango[3]:
            sum_x = sum_x+(df_complp["Posicion_x"][i]*(1.989*10**30)/2400002)
            sum_y = sum_y+(df_complp["Posicion_y"][i]*(1.989*10**30)/2400002)
            sum_z = sum_z+(df_complp["Posicion_z"][i]*(1.989*10**30)/2400002)
            sumv_x = sumv_x+(df_complp["Velocidad_x"][i]*(1.989*10**30)/2400002)
            sumv_y = sumv_y+(df_complp["Velocidad_y"][i]*(1.989*10**30)/2400002)
            sumv_z = sumv_z+(df_complp["Velocidad_z"][i]*(1.989*10**30)/2400002)
            cont = cont + 1
            densi = densi+df_complp["Densidad"][i]
    x_cm = sum_x/(cont*(1.989*10**30)/2400002)
    y_cm = sum_y/(cont*(1.989*10**30)/2400002)
    z_cm = sum_z/(cont*(1.989*10**30)/2400002)
    vel_x_cm = sumv_x/(cont*(1.989*10**30)/2400002)
    vel_y_cm = sumv_y/(cont*(1.989*10**30)/2400002)
    vel_z_cm = sumv_z/(cont*(1.989*10**30)/2400002)
    return [x_cm, y_cm, z_cm, vel_x_cm, vel_y_cm, vel_z_cm, densi, cont, rango]
```

*Código 13. centro_masa(). Función que retorna los datos del n cuerpo. *Por fines prácticos se le llamó masa a la densidad.*

Este último código retorna todos los valores de las tres componentes de la posición, las tres componentes de la velocidad, la densidad total, el número de partículas involucradas y el rango donde se encuentra el nuevo cuerpo, todos datos relacionados al centro de masa.

Al invocar la función en un bucle de tamaño N, estos son los resultados para nuestro *snap* de ejemplo:

```
cm = []
for i in range(N):
    cm.append(centro_masa(i))
[[-0.07952134225132927, -0.09017121126428447, 0.0025231109601524257,
  0.40842307174927783, -0.7051007226139971, 0.016978827485959237,
  77041470.61668447, 60175, [-0.12, -0.04, -0.12, -0.01]],
[-0.002145551072995929, 0.14100562480983267, 0.0033568428144008463,
-0.9612899802401865, -0.48332879735395085, 0.011330073234460442,
  2315922642.32714, 228280, [-0.03, 0.03, 0.13, 0.18]],
```

```
[0.07676615206527838, 0.09642246392327095, 0.0027294232300918873,
-0.4456969718313029, 0.7004036434084605, 0.016434573991907495,
218049936.87350255, 61500, [0.04, 0.11, 0.07, 0.12]],
[-0.0021285022555671655, -0.14402940767678693, 0.0033298117431496015,
1.0205289937962212, 0.35324373677749255, 0.01121518110937188,
2283476515.893003, 225944, [-0.03, 0.02, -0.19, -0.13]]]
```

Código 14. Salida con todos los datos del centro de masa tal que: $[x, y, z, v_x, v_y, v_z, \rho, [x_i, x_f, y_i, y_f]]$.

Simulación de la solución al problema de los N-cuerpos

En lo que respecta a los datos provenientes de las simulaciones en GADGET se ha llegado a un punto de reducirlos a tan solo 4 cuerpos. A la segunda parte de este trabajo le compete seguir la simulación en el tiempo, usando un algoritmo de N-cuerpos desde el punto de vista de la mecánica clásica con ayuda del método de Euler.

Desarrollo de algoritmo de N-cuerpos usando el método de Euler

Se eligió el método de Euler por su simpleza y porque nuestro modelo en específico es del tipo colisional y como ya se vio, basta con alguno de estos métodos de integración numérica para resolver la ecuación de movimiento.

Cabe mencionar que con esto no esperamos una solución escrita para la ecuación (6) como una fórmula de la aceleración, sino más bien una respuesta gráfica de la posición de los cuerpos.

La idea básica es la siguiente, se tiene que resolver la ecuación de la aceleración con un valor inicial de $\mathbf{r} = \mathbf{r}_0$, para un valor de tiempo inicial, que convenientemente puede ser cero. Una vez obtenido el valor de la aceleración en t_0 , se evalúan la velocidad y la posición con las fórmulas:

$$\mathbf{v} = \mathbf{v}_0 + \mathbf{a}\Delta t \quad \& \quad \mathbf{r} = \mathbf{r}_0 + \mathbf{v}\Delta t, \quad (10)$$

el paso más importante es multiplicar por pasos muy pequeños de tiempo, entre más pequeños el método tiende a ser más exacto. Estos nuevos valores se almacenan y el vector \mathbf{r} retroalimenta la siguiente solución de la aceleración haciendo un ciclo.

De manera general para los siguientes ciclos, las ecuaciones lucen de la siguiente forma:

$$\mathbf{a}_{ij}[t] = -G \frac{m_j}{|\mathbf{r}_i[t - \Delta t] - \mathbf{r}_j[t - \Delta t]|^3} (\mathbf{r}_i[t - \Delta t] - \mathbf{r}_j[t - \Delta t]), \quad (11)$$

y la velocidad y posición:

$$\mathbf{v}[t] = \mathbf{v}[t - \Delta t] + \mathbf{a}[t]\Delta t \quad \& \quad \mathbf{r}[t] = \mathbf{r}[t - \Delta t] + \mathbf{v}[t]\Delta t, \quad (12)$$

mientras que el tiempo va siendo la suma de estos pequeños pasos.

Con esto en mente ya es más fácil pasar al código. En Python, como en muchos otros lenguajes, se da muy bien trabajar con vectores en forma de arreglos o listas. La biblioteca *numpy* ofrece facilidades para no tener que ir evaluando componente por componente, si no que trabaja con el vector entero de una vez y mejor aún, puede trabajar con múltiples vectores, muy útil en nuestra situación donde tenemos 4 vectores, tanto de posición, de velocidad y de aceleración.

El código en sí no es muy largo ya que solo tenemos que realizar una función, en un ciclo y graficar los resultados, consta de tres unidades descritas a continuación.

La primera parte es breve, se necesita definir la constante de gravitación G , declarar los arreglos que contendrán los valores iniciales y definir la duración de la simulación, así como el tamaño de los intervalos de tiempo.

```
G = 6.67408*10**(-11) # N m^2 / kg^2
# datos iniciales
N = 2 # número de cuerpos
m = np.array([2.37534E+29, 9.6604E+28])
r = np.array([[3.17068E+12, -1.1892E+12], [-7.79619E+12, 2.92405E+12]]) # m
v = np.array([[59.13924308, 374.2431033], [-145.4138637, -920.2034515]]) # m/s
t0 = 0.0 # tiempo inicial, en s
tf = 259200000000.0 # tiempo final, en s
dt = tf/10000 # paso temporal, en s
# almacenar las trayectorias
r_list = [r*1] # posiciones de los cuerpos en cada paso temporal
t_list = [t0] # tiempo en cada paso temporal
```

Código 15. Inicialización del código de N-cuerpos.

LA segunda parte es más robusta ya que contiene al algoritmo en sí, la manera más correcta de implementarlo es con tres ciclos *for*, uno para recorrer el tiempo en pasos pequeño y un segundo para recorrer entre cada cuerpo y el último para medir la interacción con los otros $N - 1$ cuerpos.

En el Código 16 podemos ver la implementación de las ecuaciones (11) y (12), como se aprecia, no es necesario trabajar componente a componente o vector tras vector.

```
for t in np.arange(t0, tf, dt):
    # vectores de aceleración
    a = np.zeros_like(r) # inicializar aceleraciones en cero
    for i in range(N):
        for j in range(N):
            if i != j:
                r_rel = r[i] - r[j] # vector de posición relativa
                r_rel_mag = np.linalg.norm(r_rel) # magnitud del vector de
                posición relativa
                a[i] += -G * m[j] * r_rel / r_rel_mag**3 # sumar aceleración
                debida a cada cuerpo
    # actualizar velocidades y posiciones
    v += a * dt
    r += v * dt
    # guardar trayectorias
    r_list.append(r*1)
    t_list.append(t)
```

Código 16. Cálculo de trayectorias usando el método de Euler en código Python.

La última parte solo grafica la posición a lo largo del tiempo, mostrando la trayectoria de los cuerpos que se hayan simulado. El Código 17 muestra la manera para todos los cuerpos en ambas coordenadas, pero para el análisis es igual conveniente hacerlo en un solo eje y con un solo objeto, pero para ello basta con modificar un parámetro al llamar los arreglos.

```
# convertir listas a arreglos
r_list = np.array(r_list)
t_list = np.array(t_list)

# graficar trayectorias
for i in range(N):
    plt.plot(r_list[:, i, 0], r_list[:, i, 1], label='cuerpo {}'.format(i+1))
plt.legend()
plt.show()
```

Código 17. Graficar las posiciones de los cuerpos con respecto al tiempo.

Valores iniciales, número de cuerpos y vectores a usar

Como ya fue mencionado, esta parte del proyecto se realizó justo después de concluir la primera parte, ya que de ahí se extrajeron los datos que aquí abordaremos.

Adelantándonos un poco a los resultados, sabemos que tenemos 4 cuerpos en la última *snap* cada uno con sus respectivos vectores de posición y velocidad, así como sus masas. Podríamos simplemente dar estos valores como iniciales y correr la simulación, pero se tomó la decisión de cambiar el plano de referencia a el plano que tiene como centro al centro de masa, ya que no seguro que la simulación de GADET entregue estos vectores con respecto a su centro de masa.

En la física es muy común cambiar de plano de referencia para analizar el movimiento de un cuerpo que se mueve respecto a otro a cierta velocidad, siempre y cuando estos sean marcos de referencia inercial, en otras palabras, que su aceleración sea igual a cero.

El cambio de marco referencia se puede explicar con una suma vectorial, queremos pasar de expresar los vectores desde un punto cualquiera en el espacio al objeto a vectores que vayan desde el centro de masa.

Hagamos un ejemplo rápido, tenemos dos cuerpos con masas $m_1 > m_2$ con vectores de posición iguales a \mathbf{r}_1 y \mathbf{r}_2 , con ayuda de la ecuación (3) podemos calcular el vector de posición del centro de masa \mathbf{r}_{CM} . Podemos verlo de manera esquemática en la Figura 14.

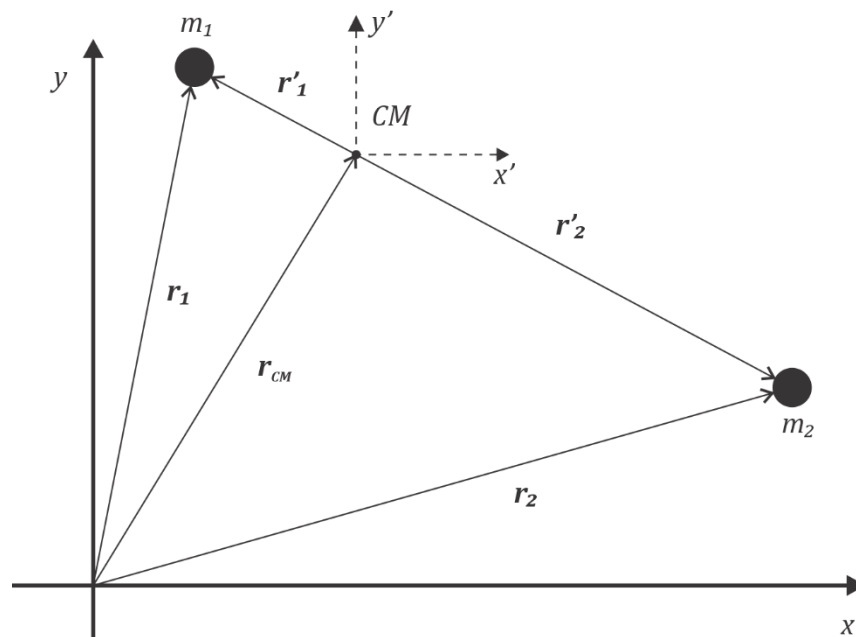


Figura 14. Cambio de sistema de referencia a un nuevo sistema primado con centro en el centro de masa de los objetos 1 y 2.

Queremos hacer el cambio de marco de referencia xy a un plano primado $x'y'$ que se encuentre justo en el centro de masa, para lo cual basta con hacer una suma vectorial para hallar los vectores primados de posición de nuestros objetos, que basados en el esquema anterior estos quedan como:

$$\mathbf{r}'_i = \mathbf{r}_i - \mathbf{r}_{CM}, \quad (13)$$

pasa lo mismo con los vectores de velocidad:

$$\mathbf{v}'_i = \mathbf{v}_i - \mathbf{v}_{CM}, \quad (14)$$

esta última ecuación nos dice que el nuevo plano de referencia puede estar en movimiento, pero con la resta vectorial le “quita” el movimiento.

Estos nuevos vectores ya podemos considerarlos como nuestros valores iniciales, junto con la masa (que no cambia a pasar del cambio de marco referencial). Además de que con estos cambios podemos analizar diferentes puntos de vista, como aislando un sistema de un objeto i interaccionando con un objeto j .

En lo que respecta a la metodología de esta segunda parte sería todo, queda correr el código y ver gráficamente los resultados de esta simulación.

Resultados

En esta sección se encuentran los resultados que recabamos tanto de la interpretación de la simulación en GADGET, como de la simulación de N-cuerpos.

Evolución gráfica de cada *snap*

Parte de los objetivos fue saber extraer e interpretar la información de simulaciones ya hechas. Aquí se presentan las interpretaciones gráficas de esos datos para ver el seguimiento de la simulación de GADGET con la que trabajamos.

Hacemos hincapié en que ya se mostraron imágenes de la *Snap* 0090 en la sección anterior, por lo que aquí se omitirá.

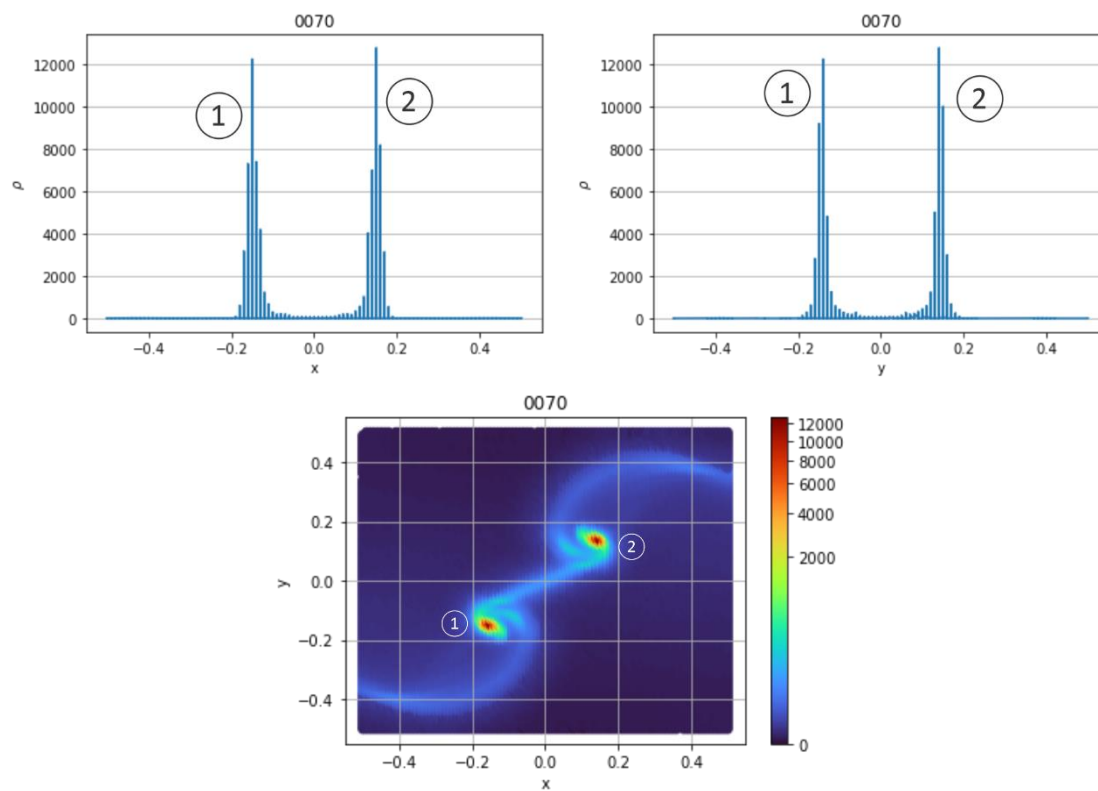


Figura 15. Snap 0070.

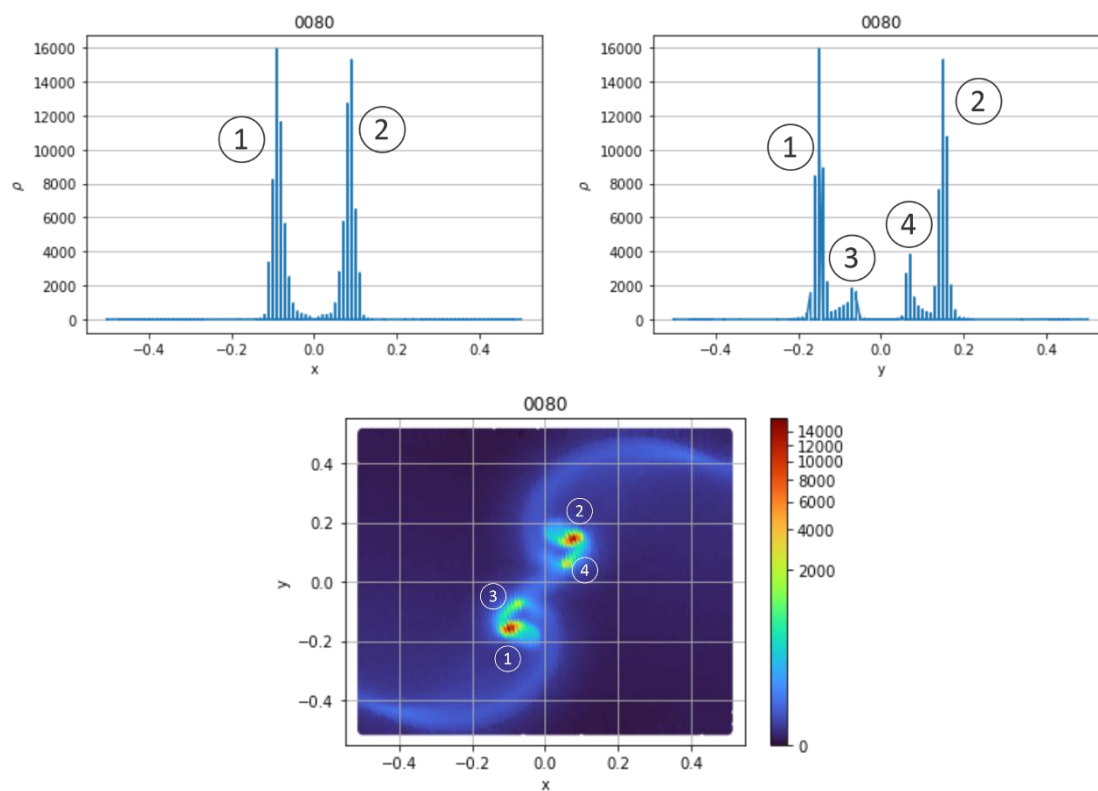


Figura 16. Snap 0080.

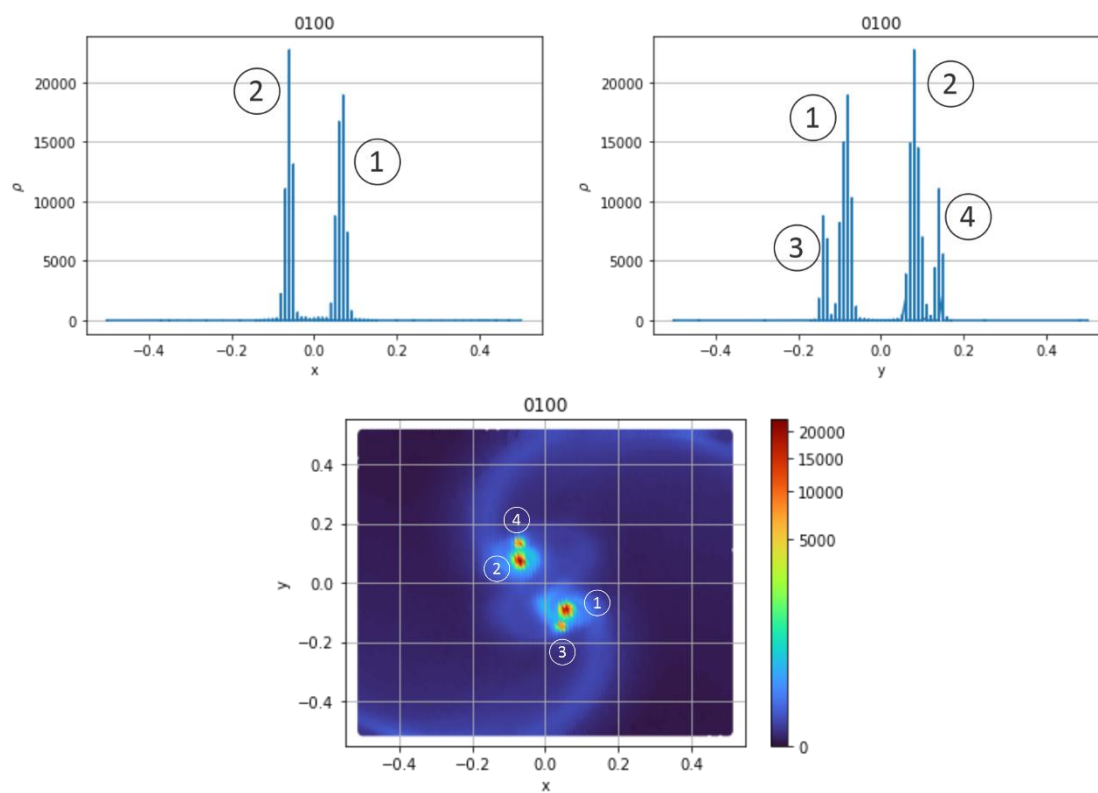


Figura 17. Snap 0100.

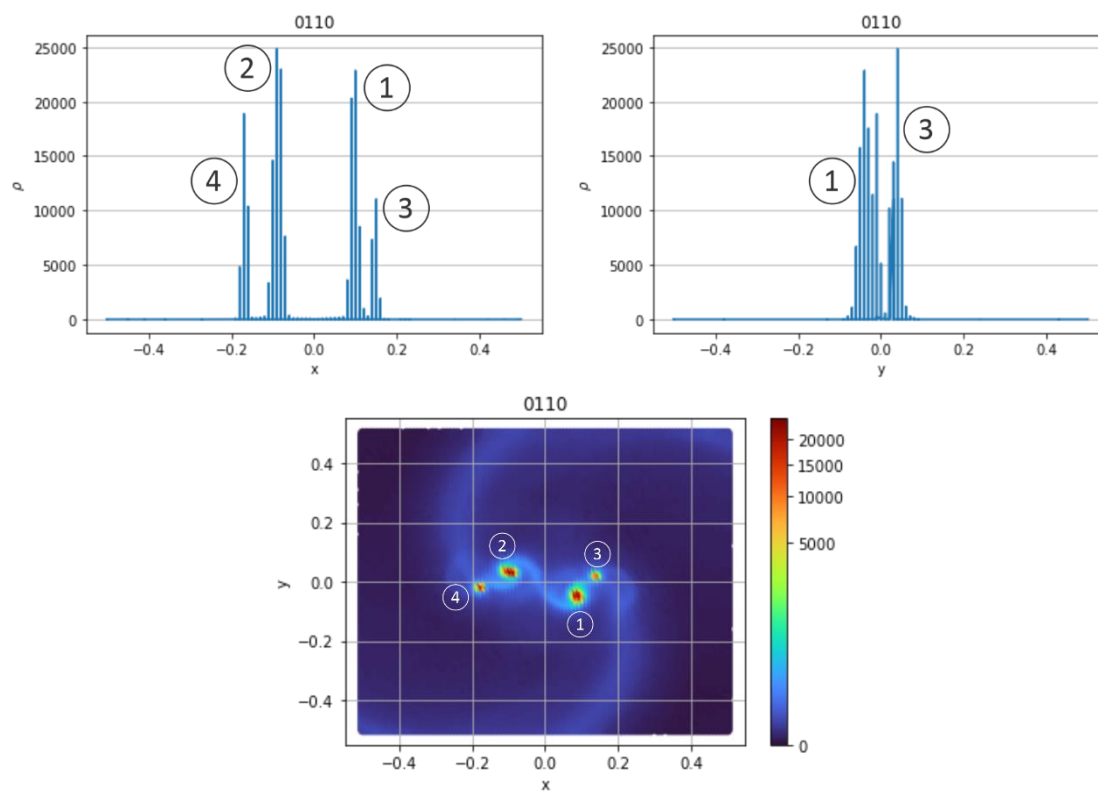


Figura 18. Snap 0110

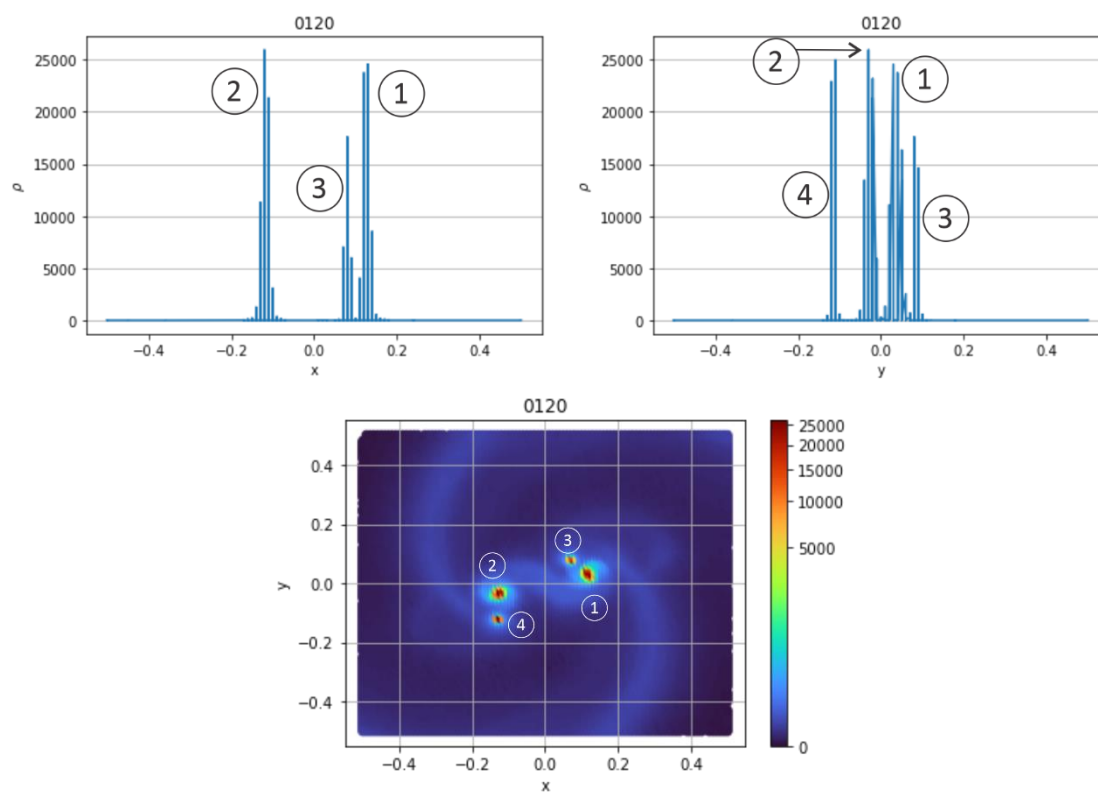


Figura 19. Snap 0120

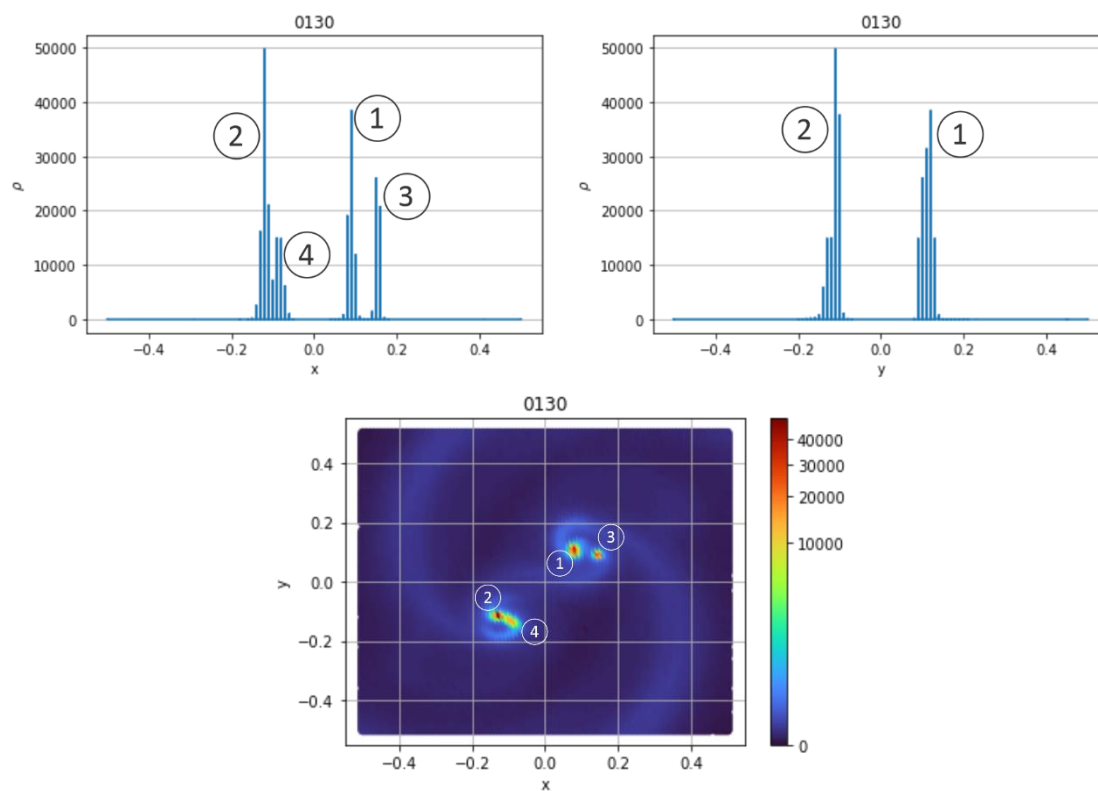


Figura 20. Snap 0130

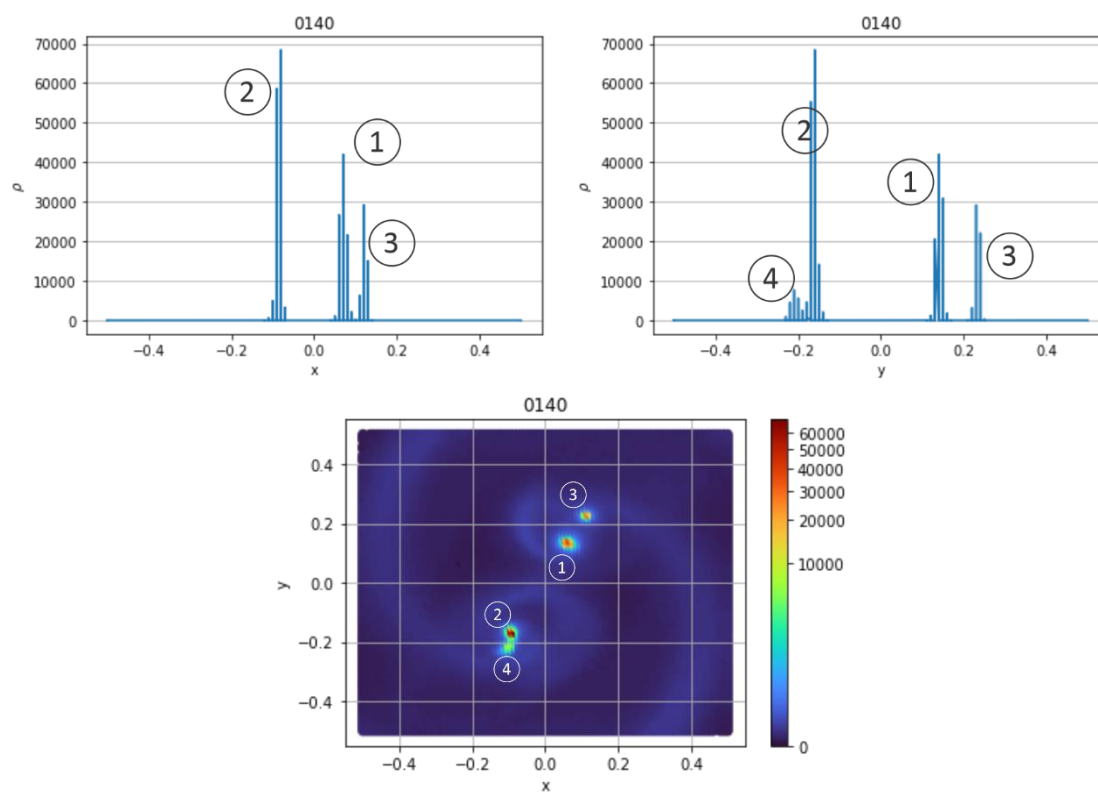


Figura 21. Snap 0140

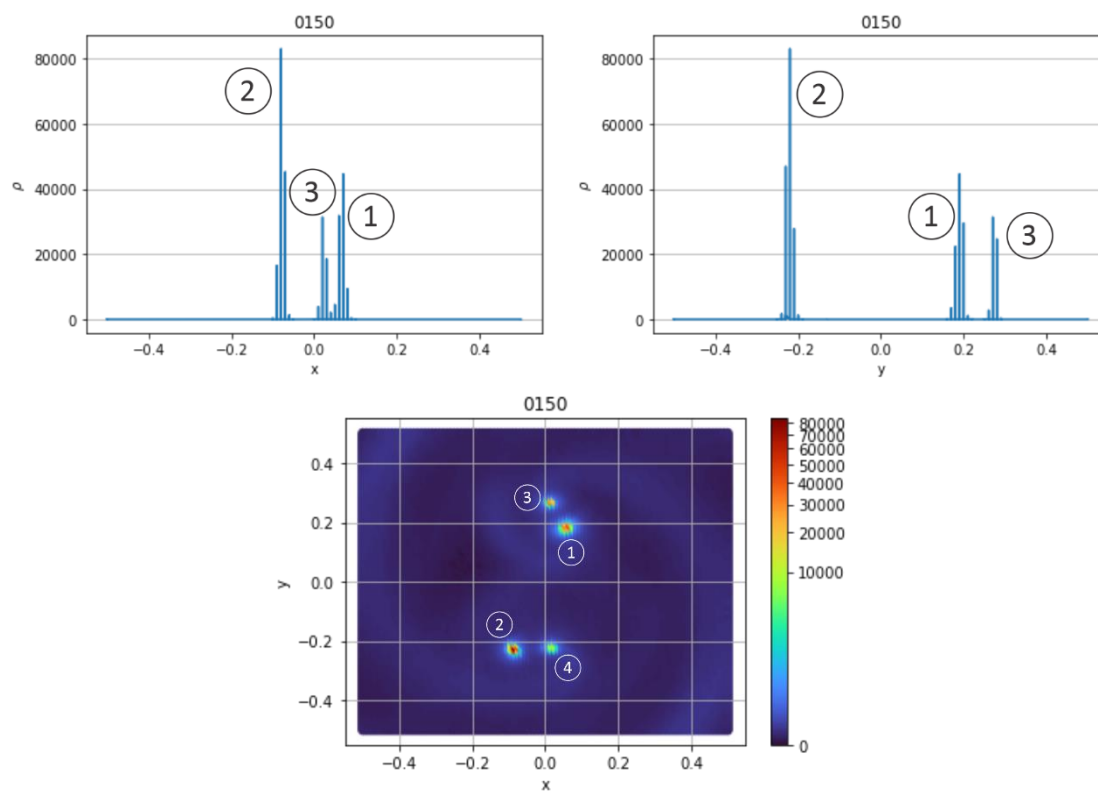


Figura 22. Snap 0150.

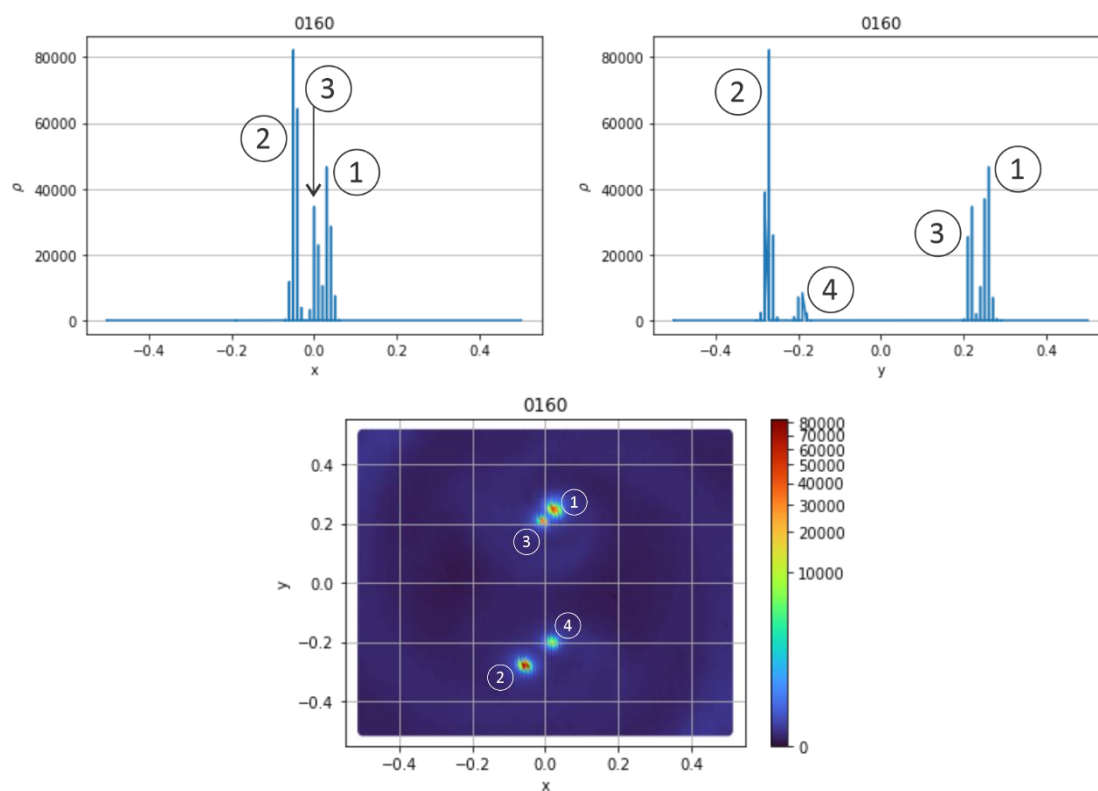


Figura 23. Snap 0160.

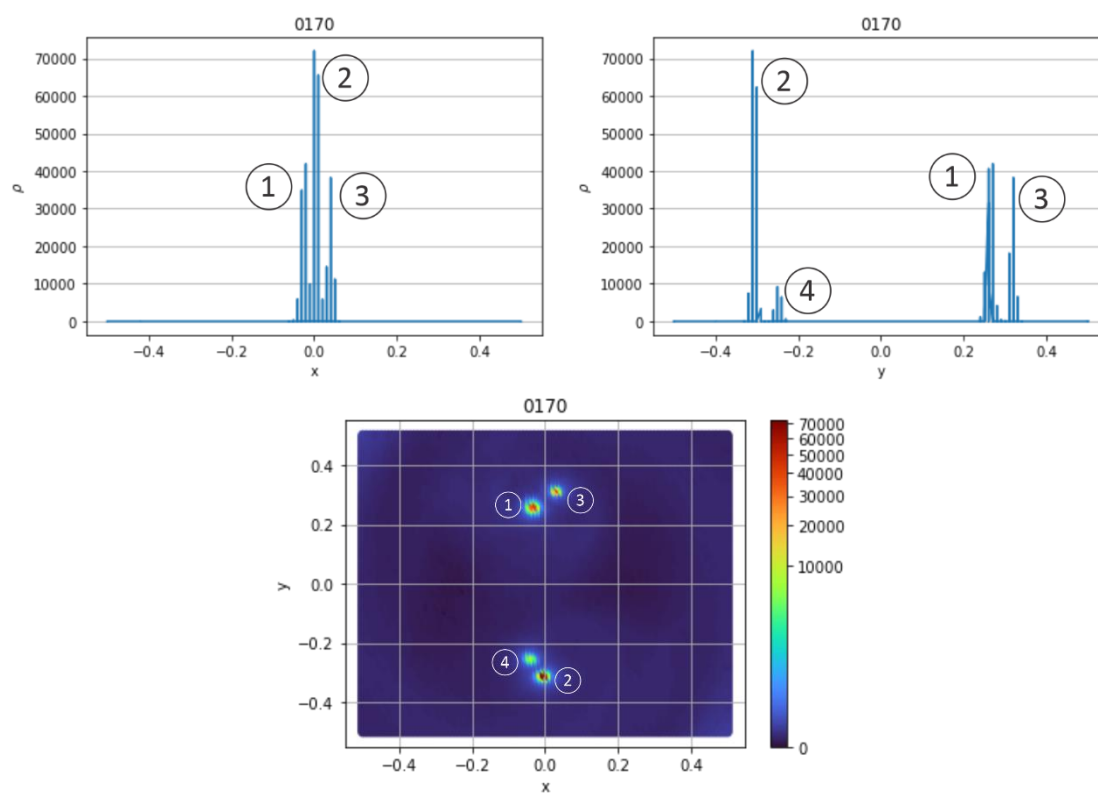


Figura 24. Snap 0170.

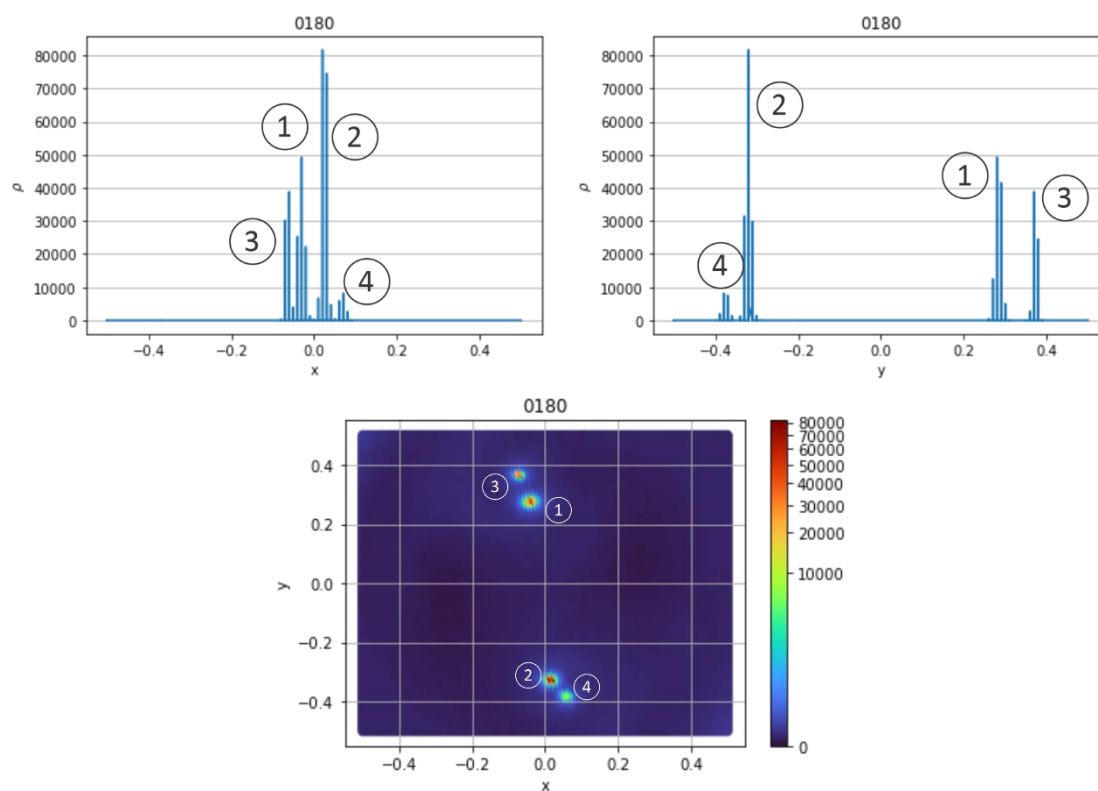


Figura 25. Snap 0180.

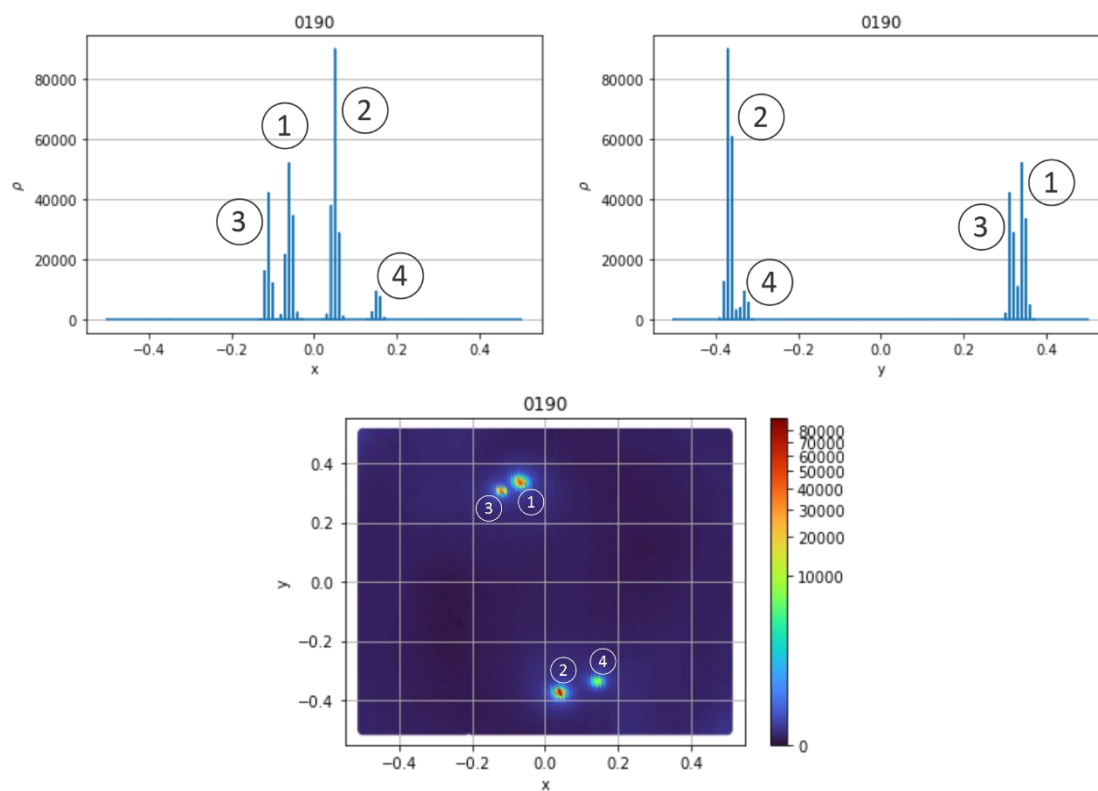


Figura 26. Snap 0190.

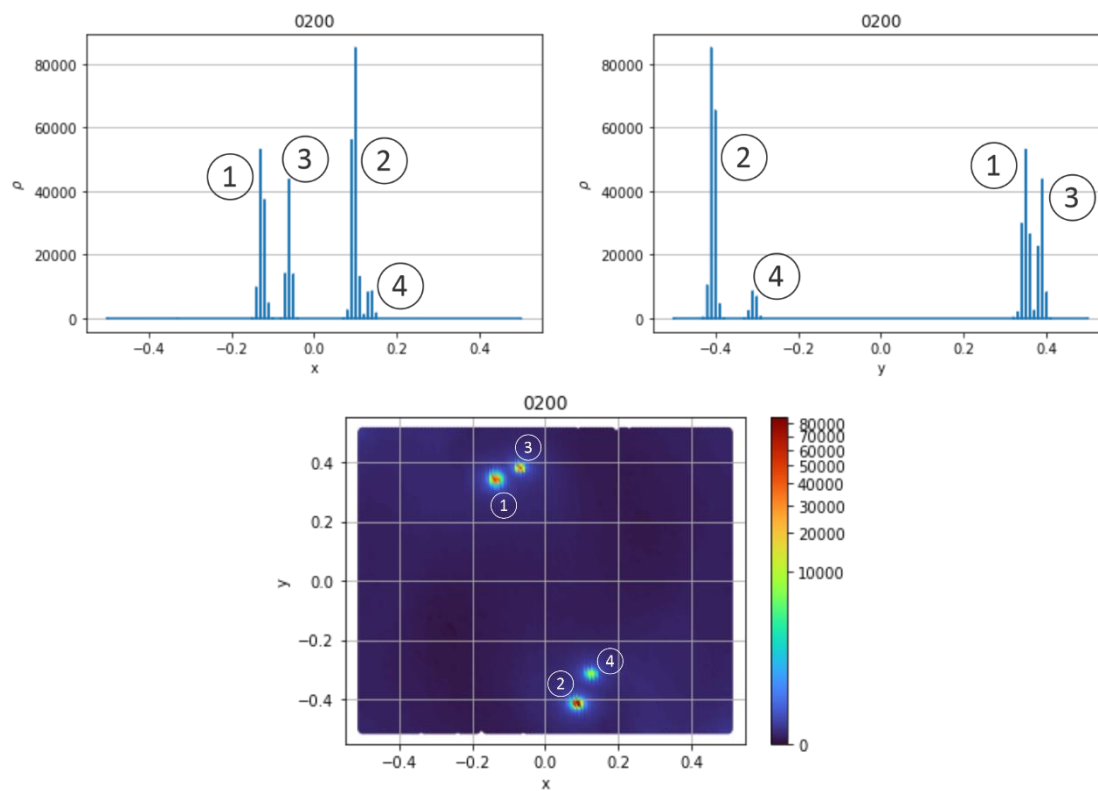


Figura 27. Snap 0200.

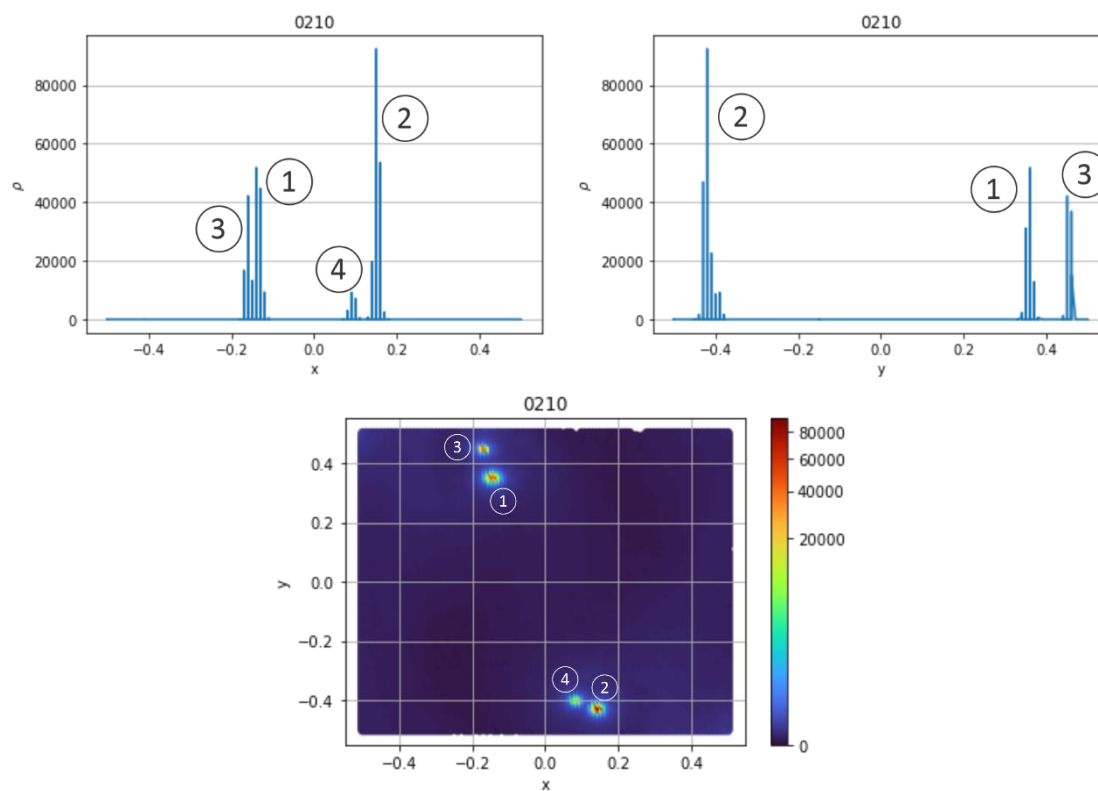


Figura 28. Snap 0210.

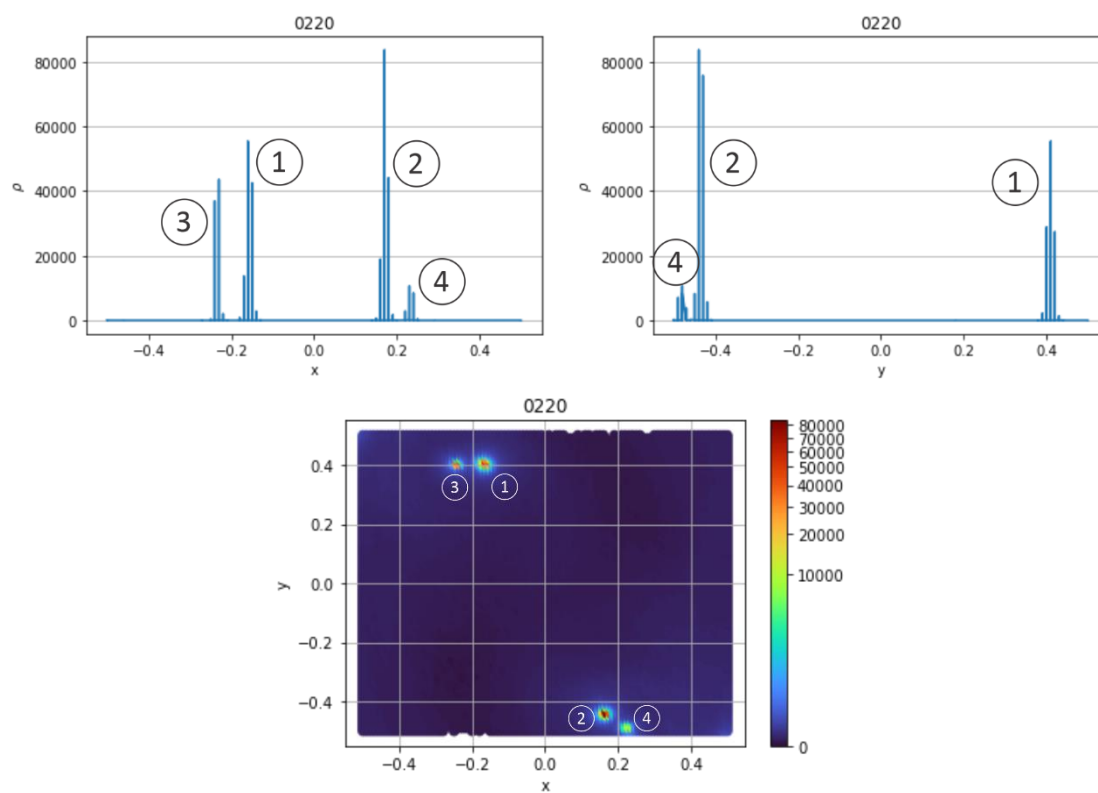


Figura 29. Snap 0220.

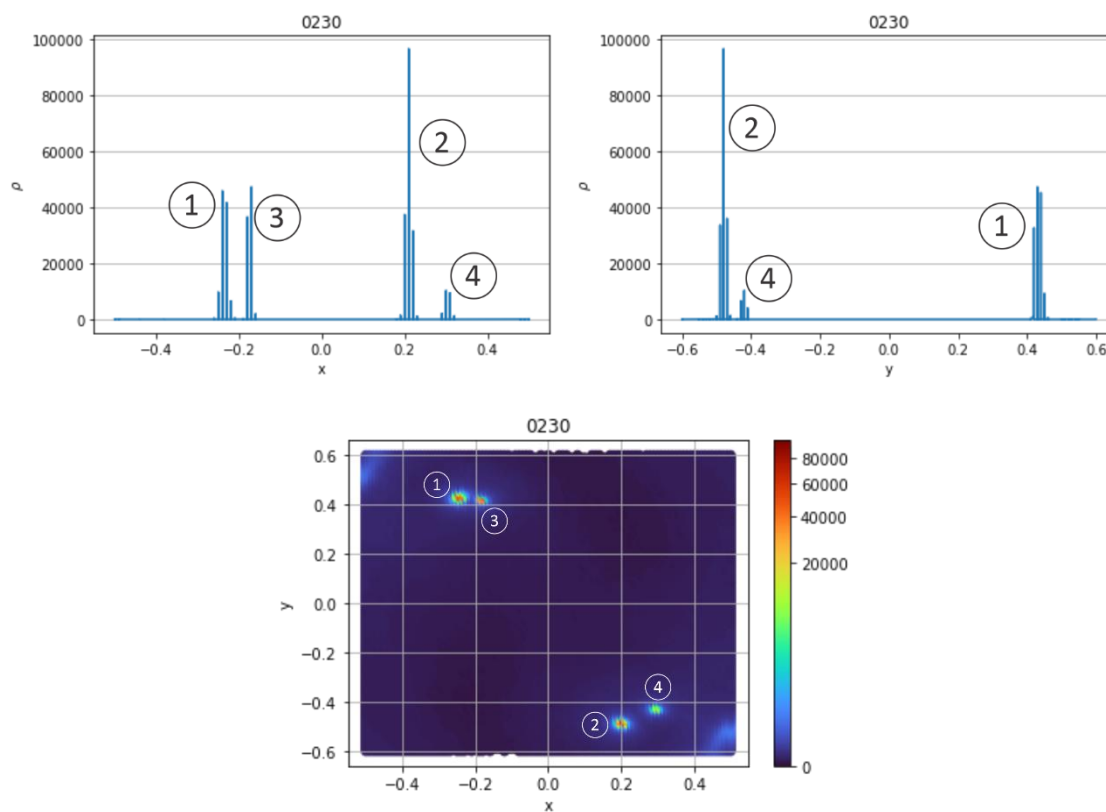


Figura 30. Snap 0230.

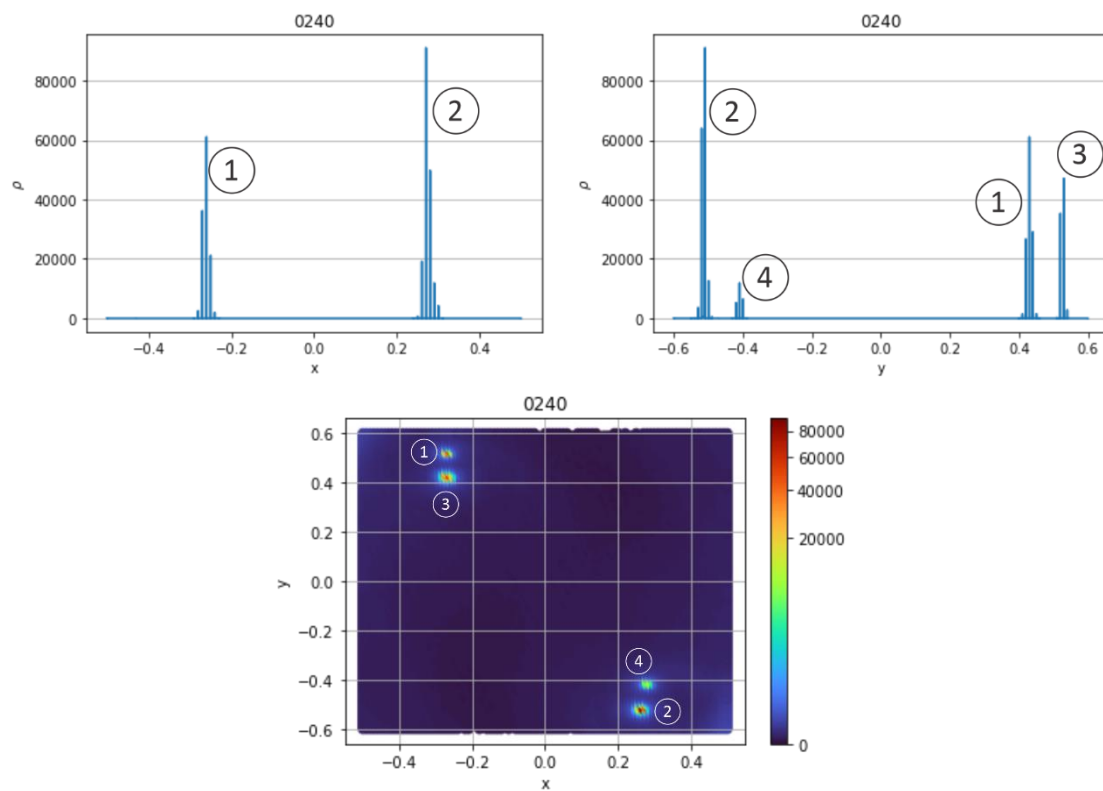


Figura 31. Snap 0240.

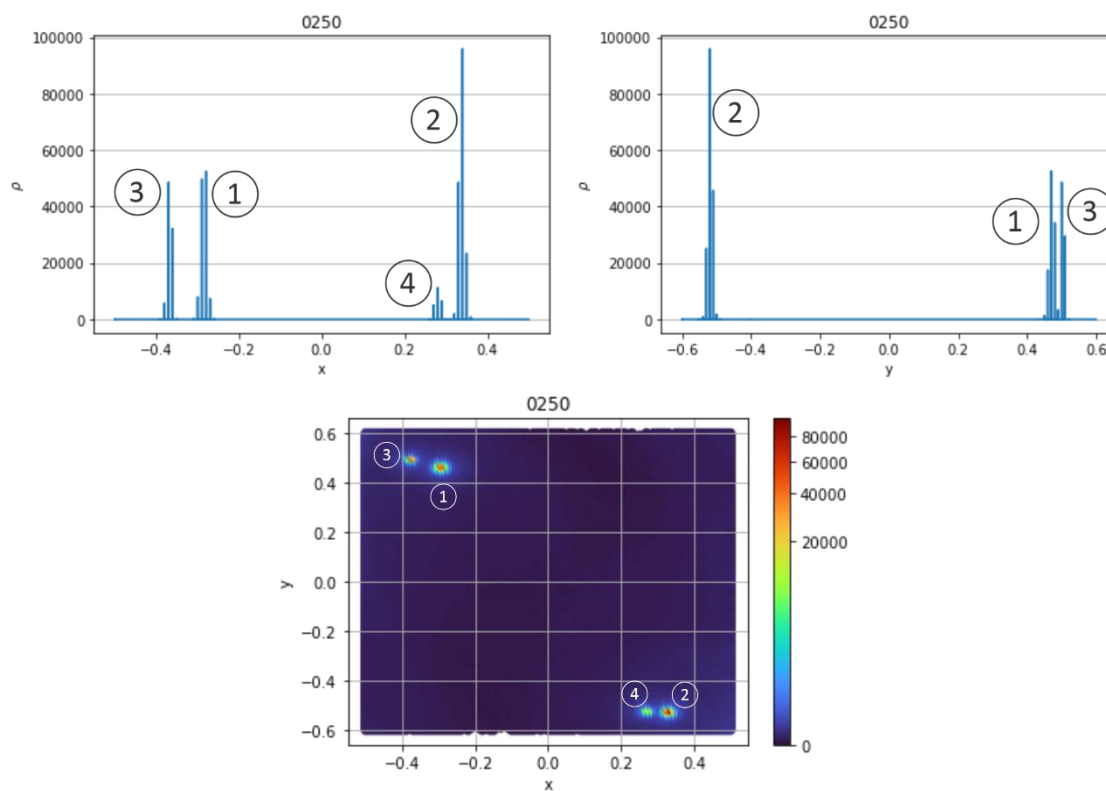


Figura 32. Snap 0250.

Datos relevantes a los fragmentos protoestelares

Como podemos apreciar en la última *snap* de la simulación, el disco de gas ya se ha fragmentado en cuatro bien definidos cuerpos protoestelares de los cuales se obtuvieron datos que veremos a continuación.

Los datos que recabamos no se encuentran en unidades con las podemos trabajar, pero tenemos los respectivos valores de conversión para pasarlas al sistema métrico, los cuales enlisto a continuación:

$$Masa \rightarrow 1 \text{ partícula} = \frac{M_s}{2,400,001} kg = 8.2875 \times 10^{23} kg, \quad (15)$$

$$Longitud \rightarrow 1u = 1.32746 \times 10^{14} m, \quad (16)$$

$$Velocidad \rightarrow 1u = 1000 \frac{m}{s}, \quad (17)$$

todas las partículas cuentan con la misma masa.

Las siguientes tablas muestran los valores relevantes recabados de la simulación de GADGET como cuerpos visibles que nacen como un conjunto de partículas.

Tabla 2. Posiciones y velocidades del objeto 1.

Snap	CM_x [m]	CM_y [m]	CM_z [m]	v _x [m/s]	v _y [m/s]	v _z [m/s]
0070	-1.846E+13	-1.746E+13	1.622E+11	528.63	-122.84	6.25
0080	-1.278E+13	-1.996E+13	2.713E+11	823.33	-950.22	14.78
0090	-2.782E+11	-1.951E+13	4.419E+11	1166.10	311.53	12.06
0100	9.674E+12	-1.112E+13	5.302E+11	214.94	1445.42	-1.54
0110	1.279E+13	-5.774E+12	5.451E+11	585.33	535.65	-8.03
0120	1.682E+13	4.751E+12	4.168E+11	175.06	977.55	-4.96
0130	1.184E+13	1.555E+13	4.443E+11	-764.77	302.61	8.46
0140	9.390E+12	1.868E+13	5.406E+11	-45.56	344.75	0.32
0150	8.790E+12	2.511E+13	4.858E+11	-111.35	638.08	0.29
0160	4.544E+12	3.384E+13	5.519E+11	-813.43	659.43	2.52
0170	-2.524E+12	3.500E+13	4.742E+11	-212.94	1004.47	-2.91
0180	-4.082E+12	3.760E+13	5.639E+11	-51.88	476.07	-7.21
0190	-7.838E+12	4.559E+13	5.246E+11	-688.71	629.14	13.04
0200	-1.675E+13	4.640E+13	5.277E+11	-229.21	-153.26	-8.31
0210	-1.784E+13	4.750E+13	5.849E+11	-70.79	498.63	3.16
0220	-2.090E+13	5.437E+13	5.280E+11	-498.51	646.22	5.16
0230	-3.127E+13	5.789E+13	5.630E+11	-589.30	-231.06	-7.29
0240	-3.462E+13	5.710E+13	6.005E+11	-125.06	194.11	0.97

0250	-3.783E+13	6.254E+13	5.701E+11	-414.26	586.68	7.65
-------------	------------	-----------	-----------	---------	--------	------

Tabla 3. Posiciones y velocidades del objeto 2.

Snap	CM_x [m]	CM_y [m]	CM_z [m]	v_x [m/s]	v_y [m/s]	v_z [m/s]
0070	1.832E+13	1.744E+13	1.667E+11	-528.53	110.39	6.09
0080	1.273E+13	2.019E+13	2.798E+11	-868.48	1007.98	13.86
0090	-2.736E+11	1.911E+13	4.463E+11	-1124.04	-415.92	11.76
0100	-7.834E+12	1.078E+13	5.370E+11	-245.03	-463.69	4.07
0110	-1.183E+13	5.838E+12	5.320E+11	-810.10	-549.59	-6.99
0120	-1.560E+13	-3.606E+12	4.248E+11	-201.70	-828.99	-3.33
0130	-1.393E+13	-1.531E+13	4.578E+11	188.56	-584.08	7.43
0140	-1.061E+13	-2.167E+13	5.320E+11	-132.61	784.53	1.67
0150	-1.039E+13	-2.935E+13	4.831E+11	241.24	-611.33	-2.51
0160	-6.171E+12	-3.599E+13	5.402E+11	448.01	-490.70	4.76
0170	5.774E+11	-4.060E+13	4.926E+11	508.44	-136.91	-0.08
0180	3.249E+12	-4.248E+13	5.538E+11	89.45	-397.85	-2.28
0190	2.035E+13	-4.362E+13	6.140E+11	223.12	505.90	-2.15
0200	1.291E+13	-5.393E+13	5.383E+11	550.11	-306.92	-5.30
0210	2.017E+13	-5.593E+13	5.659E+11	449.28	21.80	3.27
0220	2.277E+13	-5.782E+13	5.452E+11	203.55	-409.32	4.86
0230	2.783E+13	-6.370E+13	5.782E+11	527.52	-419.28	-4.89
0240	3.605E+13	-6.813E+13	5.680E+11	698.45	-249.40	6.59
0250	4.496E+13	-6.889E+13	5.844E+11	502.76	110.60	-6.47

Tabla 4. Posiciones y velocidades del objeto 3.

Snap	CM_x [m]	CM_y [m]	CM_z [m]	v_x [m/s]	v_y [m/s]	v_z [m/s]
0070	-	-	-	-	-	-
0080	-7.385E+12	-8.746E+12	2.072E+11	-381.51	355.17	5.63
0090	-1.026E+13	-1.088E+13	3.492E+11	199.76	-607.69	17.11
0100	6.730E+12	-1.779E+13	5.938E+11	2091.53	243.88	11.07
0110	1.956E+13	3.362E+12	5.303E+11	-338.95	1466.47	-4.40
0120	1.054E+13	1.119E+13	4.430E+11	-543.55	-375.43	-9.63
0130	2.038E+13	1.303E+13	4.208E+11	555.51	1555.04	10.12
0140	1.608E+13	3.091E+13	5.674E+11	-866.89	949.03	11.11
0150	2.905E+12	3.625E+13	6.505E+11	-924.93	-188.80	-0.60
0160	2.691E+11	2.881E+13	5.251E+11	1009.74	-460.56	-15.14
0170	5.272E+12	4.231E+13	5.031E+11	-715.40	1172.31	1.10
0180	-8.364E+12	4.944E+13	5.196E+11	-1062.64	-95.61	3.21
0190	-1.468E+13	4.149E+13	5.449E+11	570.93	-693.49	-3.76
0200	-7.948E+12	5.159E+13	5.848E+11	-576.63	1280.15	4.98

0210	-2.133E+13	6.026E+13	5.762E+11	-1118.29	29.61	-3.22
0220	-3.100E+13	5.415E+13	5.365E+11	-49.94	-780.04	-3.48
0230	-2.300E+13	5.669E+13	5.925E+11	43.51	1444.85	11.57
0240	-3.463E+13	6.994E+13	6.300E+11	-1274.13	342.11	-3.06
0250	-4.880E+13	6.665E+13	5.596E+11	-618.81	-707.77	-6.43

Tabla 5. Posiciones y velocidades del objeto 4.

Snap	CM_x [m]	CM_y [m]	CM_z [m]	v_x [m/s]	v_y [m/s]	v_z [m/s]
0070	-	-	-	-	-	-
0080	8.087E+12	9.262E+12	2.129E+11	386.44	-350.30	7.01
0090	1.009E+13	1.282E+13	3.611E+11	-402.19	720.75	16.92
0100	-8.078E+12	1.799E+13	5.981E+11	-2035.66	-468.79	9.24
0110	-2.237E+13	-1.470E+12	5.551E+11	-26.07	-1555.81	-7.57
0120	-1.594E+13	-1.523E+13	4.375E+11	786.94	-400.13	-9.55
0130	-	-	-	-	-	-
0140	-1.241E+13	-2.790E+13	5.491E+11	1751.45	-1375.45	-2.02
0150	-	-	-	-	-	-
0160	3.874E+12	-2.572E+13	4.552E+11	-363.90	45.11	-2.63
0170	-4.317E+12	-3.283E+13	4.767E+11	-532.47	-1480.58	5.17
0180	8.965E+12	-4.985E+13	5.811E+11	1609.90	75.72	8.24
0190	2.035E+13	-4.362E+13	6.140E+11	223.12	505.90	-2.15
0200	1.798E+13	-4.070E+13	5.580E+11	-496.68	-172.55	-5.02
0210	1.232E+13	-5.234E+13	5.009E+11	152.09	-1682.76	0.01
0220	3.095E+13	-6.397E+13	6.120E+11	1442.33	426.01	4.90
0230	4.042E+13	-5.597E+13	6.214E+11	145.63	491.12	-1.27
0240	3.828E+13	-5.434E+13	5.829E+11	-360.52	-348.40	-4.31
0250	3.729E+13	-6.847E+13	5.591E+11	864.26	-1751.24	4.24

Estas cuatro primeras tablas muestran las cantidades vectoriales (en sus componentes cartesianos), mientras que en las siguientes destacan los datos escalares.

Tabla 6. Datos escalares del objeto 1 y delimitaciones.

Snap	n partículas	Densidad	Masa [kg]	l_x [m]	l_y [m]
0070	329851	1.273E+09	2.734E+29	1.726E+13	1.858E+13
0080	126187	1.050E+09	1.046E+29	3.982E+12	7.965E+12
0090	225944	2.283E+09	1.873E+29	6.637E+12	7.965E+12
0100	145422	1.700E+09	1.205E+29	2.655E+12	7.965E+12
0110	266467	3.752E+09	2.208E+29	7.965E+12	6.637E+12
0120	268275	4.639E+09	2.223E+29	5.310E+12	6.637E+12
0130	223130	5.866E+09	1.849E+29	5.310E+12	3.982E+12

0140	278195	7.425E+09	2.306E+29	5.310E+12	5.310E+12
0150	281361	7.980E+09	2.332E+29	5.310E+12	5.310E+12
0160	282540	8.558E+09	2.342E+29	5.310E+12	5.310E+12
0170	161205	5.474E+09	1.336E+29	1.327E+12	6.637E+12
0180	280395	9.172E+09	2.324E+29	3.982E+12	5.310E+12
0190	286909	9.351E+09	2.378E+29	5.310E+12	3.982E+12
0200	283688	9.727E+09	2.351E+29	3.982E+12	3.982E+12
0210	266946	9.832E+09	2.212E+29	3.982E+12	5.310E+12
0220	290388	1.029E+10	2.407E+29	5.310E+12	5.310E+12
0230	291270	1.058E+10	2.414E+29	6.637E+12	5.310E+12
0240	291791	1.080E+10	2.418E+29	5.310E+12	5.310E+12
0250	286617	1.088E+10	2.375E+29	3.982E+12	3.982E+12

Tabla 7. Datos escalares del objeto 2 y delimitaciones.

Snap	n partículas	Densidad	Masa [kg]	l_x [m]	l_y [m]
0070	334168	1.337E+09	2.769E+29	1.858E+13	1.858E+13
0080	119694	9.633E+08	9.920E+28	3.982E+12	7.965E+12
0090	228280	2.316E+09	1.892E+29	7.965E+12	6.637E+12
0100	283425	3.046E+09	2.349E+29	1.062E+13	9.292E+12
0110	210185	3.348E+09	1.742E+29	7.965E+12	3.982E+12
0120	277418	4.531E+09	2.299E+29	9.292E+12	6.637E+12
0130	375652	8.221E+09	3.113E+29	1.062E+13	9.292E+12
0140	152711	8.299E+09	1.266E+29	1.327E+12	5.310E+12
0150	300596	1.641E+10	2.491E+29	3.982E+12	5.310E+12
0160	302905	1.730E+10	2.510E+29	3.982E+12	5.310E+12
0170	303678	1.773E+10	2.517E+29	3.982E+12	3.982E+12
0180	307390	1.814E+10	2.547E+29	3.982E+12	5.310E+12
0190	304911	1.856E+10	2.527E+29	5.310E+12	2.655E+12
0200	309681	1.880E+10	2.566E+29	3.982E+12	3.982E+12
0210	310849	1.898E+10	2.576E+29	5.310E+12	3.982E+12
0220	312396	1.918E+10	2.589E+29	5.310E+12	3.982E+12
0230	313891	1.932E+10	2.601E+29	5.310E+12	5.310E+12
0240	310766	1.945E+10	2.575E+29	3.982E+12	3.982E+12
0250	315349	1.955E+10	2.613E+29	5.310E+12	5.310E+12

Tabla 8. Datos escalares del objeto 3 y delimitaciones.

Snap	n partículas	Densidad	Masa [kg]	l_x [m]	l_y [m]
0070	-	-	-	-	-
0080	66540	56655854	5.514E+28	1.062E+13	1.460E+13
0090	60175	77041471	4.987E+28	1.062E+13	1.460E+13

0100	71881	412841915	5.957E+28	6.637E+12	6.637E+12
0110	68073	609676539	5.642E+28	5.310E+12	2.655E+12
0120	77675	1083420686	6.437E+28	5.310E+12	3.982E+12
0130	90598	1909234440	7.508E+28	3.982E+12	3.982E+12
0140	95657	2291657767	7.928E+28	3.982E+12	3.982E+12
0150	99097	2556710207	8.213E+28	3.982E+12	3.982E+12
0160	103853	2925632235	8.607E+28	2.655E+12	3.982E+12
0170	106740	3314896987	8.846E+28	2.655E+12	3.982E+12
0180	107883	3475222966	8.941E+28	3.982E+12	3.982E+12
0190	109639	3779704770	9.086E+28	2.655E+12	2.655E+12
0200	111528	4080316907	9.243E+28	2.655E+12	2.655E+12
0210	112749	4126273766	9.344E+28	2.655E+12	3.982E+12
0220	114235	4306177619	9.467E+28	3.982E+12	2.655E+12
0230	115385	4547554663	9.563E+28	3.982E+12	3.982E+12
0240	115904	4639282471	9.606E+28	2.655E+12	3.982E+12
0250	116566	4711189634	9.660E+28	3.982E+12	2.655E+12

Tabla 9. Datos escalares del objeto 4 y delimitaciones.

Snap	n partículas	Densidad	Masa [kg]	l_x [m]	l_y [m]
0070	-	-	-	-	-
0080	71370	116680145	5.915E+28	1.062E+13	1.062E+13
0090	61500	218049937	5.097E+28	9.292E+12	6.637E+12
0100	87656	530152012	7.264E+28	9.292E+12	6.637E+12
0110	86004	1194166895	7.128E+28	5.310E+12	3.982E+12
0120	95513	1965389275	7.916E+28	5.310E+12	3.982E+12
0130	-	-	-	-	-
0140	69386	361482173	5.750E+28	3.982E+12	6.637E+12
0150	-	-	-	-	-
0160	64504	394688013	5.346E+28	3.982E+12	6.637E+12
0170	57722	395139030	4.784E+28	2.655E+12	5.310E+12
0180	63144	406716270	5.233E+28	5.310E+12	6.637E+12
0190	63943	445207852	5.299E+28	5.310E+12	5.310E+12
0200	64489	467848300	5.345E+28	3.982E+12	5.310E+12
0210	66117	504147880	5.479E+28	5.310E+12	3.982E+12
0220	66244	548960226	5.490E+28	5.310E+12	5.310E+12
0230	66698	560279761	5.528E+28	3.982E+12	5.310E+12
0240	67573	586367157	5.600E+28	3.982E+12	5.310E+12
0250	68591	606396551	5.684E+28	6.637E+12	3.982E+12

Los valores de l_x y de l_y hacen referencia a al tamaño delimitado por los rangos x_i y x_f y y_i y y_f .

Datos para la simulación de N-cuerpos

Basados en el último de los *snap*s podemos definir cómo fue que aplicamos el algoritmo de N-cuerpos. Se definieron 4 sistemas, el primero y principal es el sistema compuesto por los 4 cuerpos, los tres restantes son los compuestos por los cuerpos 1 y 3 (sistema 1-3), por los cuerpos 2 y 4 (sistema 2-4) y un último también de dos cuerpos, pero ahora tomando a estos dos sistemas como nuevos objetos en su conjunto. Podemos apreciar estos sistemas en la Figura 33.

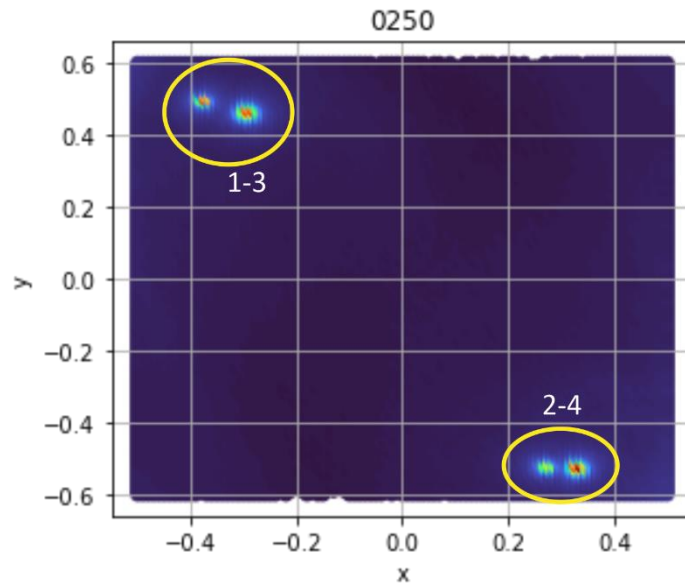


Figura 33. Elección de sistemas a analizar.

Usando las ecuaciones (13) y (14) y los datos del *snap* 0250 de todas las tablas anteriores podemos cambiar de marco referencial y definir los datos iniciales que usamos en la simulación. Dichos datos se muestran a continuación.

Tabla 10. Datos iniciales para el sistema principal de 4 cuerpos.

$M [kg]$				$6.523E+29$			
$x_1 [m]$	$y_1 [m]$	$x_2 [m]$	$y_2 [m]$	$x_3 [m]$	$y_3 [m]$	$x_4 [m]$	$y_4 [m]$
$-3.809E+13$	$6.346E+13$	$4.470E+13$	$-6.797E+13$	$-4.906E+13$	$6.758E+13$	$3.703E+13$	$-6.755E+13$
$v_{x1} [m/s]$	$v_{y1} [m/s]$	$v_{x2} [m/s]$	$v_{y2} [m/s]$	$v_{x3} [m/s]$	$v_{y3} [m/s]$	$v_{x4} [m/s]$	$v_{y4} [m/s]$
-448.51	586.16	468.51	110.08	-653.06	-708.29	830.01	-1751.76
$m_1 [kg]$		$m_2 [kg]$		$m_3 [kg]$		$m_4 [kg]$	
$2.375E+29$		$2.613E+29$		$9.660E+28$		$5.684E+28$	

Los datos de las componentes vectoriales de la Tabla 10 ya están con respecto al centro de masa del sistema, al igual que los de las tablas subsecuentes.

Tabla 11. Datos de la masa total y los centros de masa de cada sistema.

Sistema	$x_{CM} [m]$	$y_{CM} [m]$	$v_{xCM} [m/s]$	$v_{yCM} [m/s]$	$M [kg]$
1-3	-4.100E+13	6.373E+13	-473.40	212.43	3.341E+29
2-4	4.359E+13	-6.881E+13	567.34	-222.02	3.182E+29
N=2 & N=4	2.575E+11	-9.215E+11	34.25	0.52	6.523E+29

Es de esperar que los sistemas de $N = 4$ y $N = 2$ compartan los datos de sus centros de masa y la masa total debido a que están compuestos por la misma cantidad de cuerpos.

Tabla 12. Datos iniciales para el sistema 1-3.

M_{13}		$3.341E+29$	
$x_1 [m]$	$y_1 [m]$	$x_3 [m]$	$y_3 [m]$
3.171E+12	-1.189E+12	-7.796E+12	2.924E+12
$v_{x1} [m/s]$	$v_{y1} [m/s]$	$v_{x3} [m/s]$	$v_{y3} [m/s]$
59.14	374.24	-145.41	-920.20
m_1		m_3	
2.375E+29		9.660E+28	

Tabla 13. Datos iniciales para el sistema 2-4.

M_{24}		$3.182E+29$	
$x_2 [m]$	$y_2 [m]$	$x_4 [m]$	$y_4 [m]$
1.370E+12	-7.504E+10	-6.296E+12	3.450E+11
$v_{x2} [m/s]$	$v_{y2} [m/s]$	$v_{x4} [m/s]$	$v_{y4} [m/s]$
-64.58	332.62	296.92	-1529.22
m_2		m_4	
2.613E+29		5.684E+28	

Tabla 14. Datos para el subsistema de 2 cuerpos.

M		$6.523E+29$	
$x_{13} [m]$	$y_{13} [m]$	$x_{24} [m]$	$y_{24} [m]$
-4.126E+13	6.465E+13	4.333E+13	-6.789E+13
$v_{x13} [m/s]$	$v_{y13} [m/s]$	$v_{x24} [m/s]$	$v_{y24} [m/s]$
-507.65	211.92	533.09	-222.54
m_{13}		m_{24}	
3.34138E+29		3.1819E+29	

Gráficas de la simulación de N cuerpos

Habiendo fijado los valores iniciales de posición, velocidad y masa para que el algoritmo de N-cuerpos pueda trabajar, se procedió a correr las simulaciones y las respuestas gráficas obtenidas se presentan a continuación.

Las cuatro simulaciones se hicieron con un $tf = 5,000$ años *terrestres* lo que equivale a unos $1.5768 \times 10^{11} s$ con pasos iguales a $dt = tf/10,000$ lo que se traduce a 10,000 iteraciones, que es un número por demás aceptable para el método de Euler.

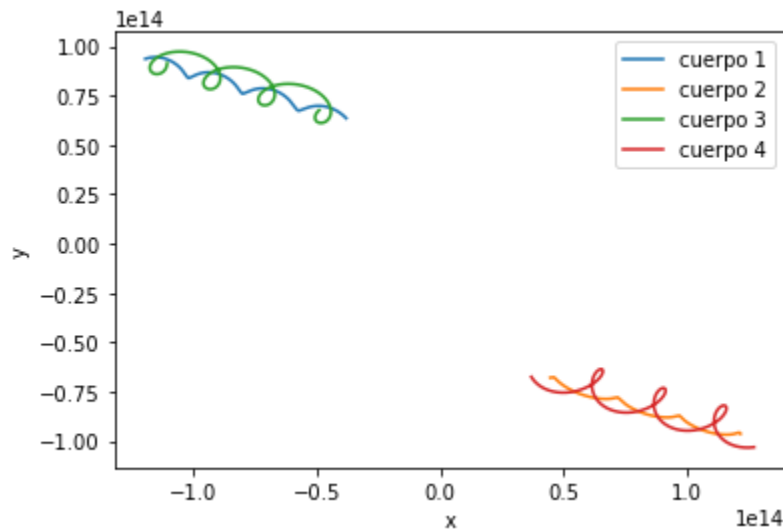


Figura 34. Respuesta gráfica para el sistema principal de 4 cuerpos.

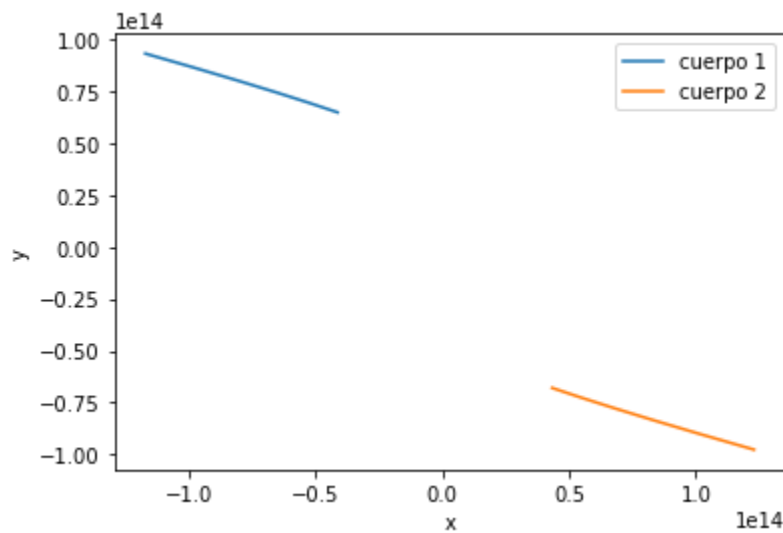


Figura 35. Respuesta gráfica para el subsistema de dos cuerpos. El cuerpo 1 hace referencia al sistema 1-3, mientras que el 2 al sistema 2-4.

Para los sistemas faltantes se realizaron también gráficas de posiciones unidimensionales (x y y por separado) con respecto al tiempo.

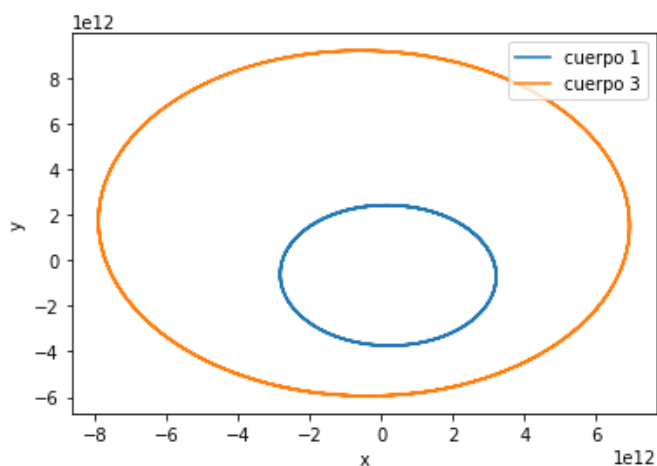


Figura 36. Respuesta gráfica para el sistema 1-3.

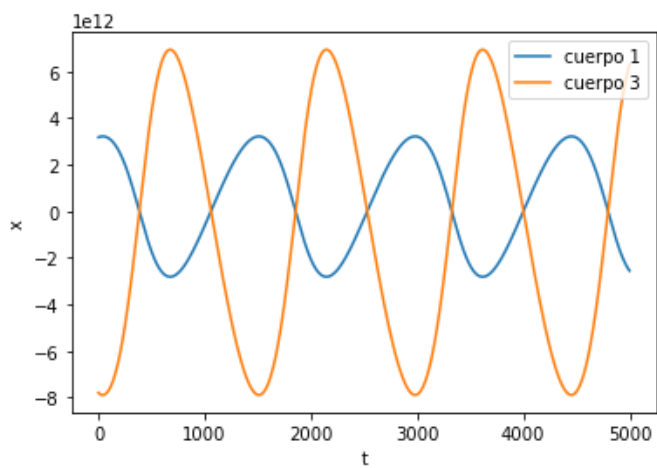


Figura 37. Cambio de la posición en x con respecto al tiempo. Sistema 1-3.

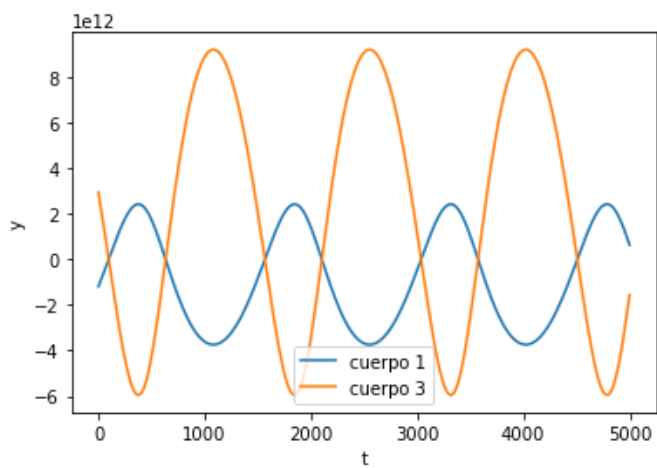


Figura 38. Cambio de la posición en y con respecto al tiempo. Sistema 1-3.

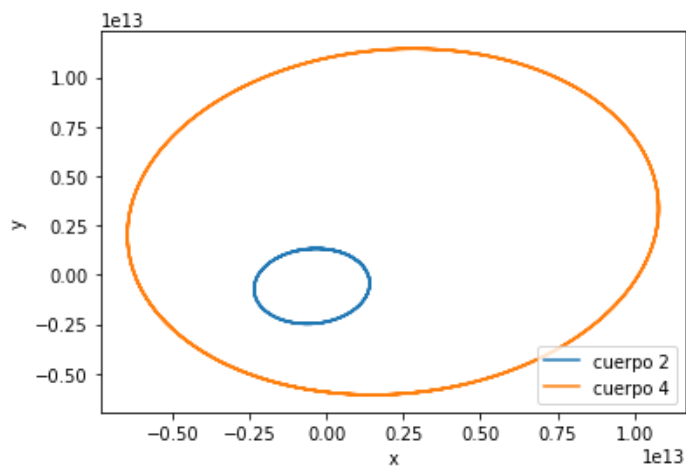


Figura 39. Respuesta gráfica para el sistema 2-4.

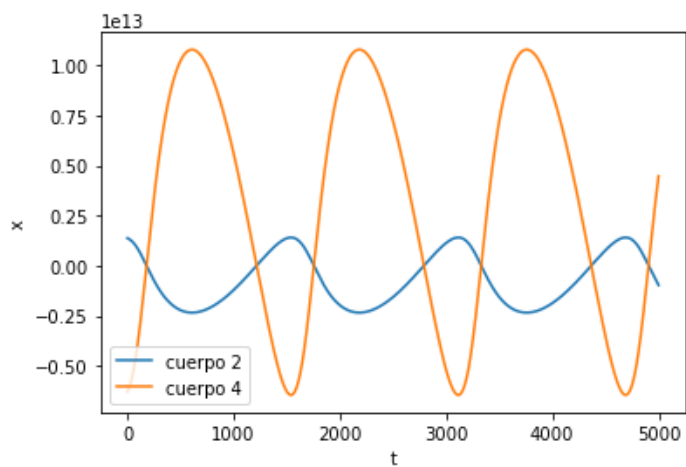


Figura 40. Cambio de la posición en x con respecto al tiempo. Sistema 2-4.

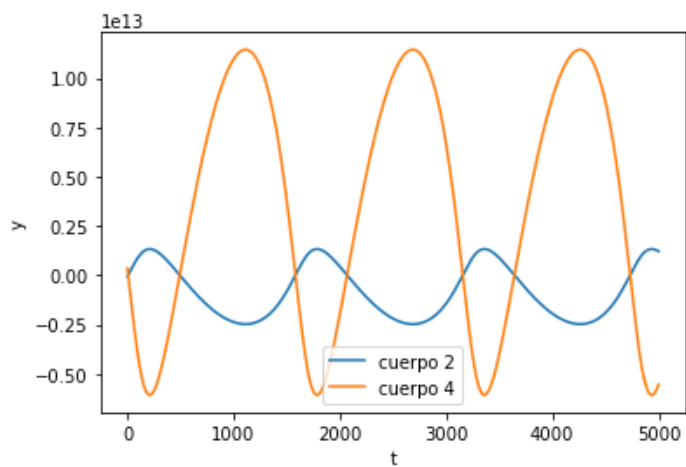


Figura 41. Cambio de la posición en y con respecto al tiempo. Sistema 2-4.

Con estas gráficas concluyen todo lo obtenido en el presente proyecto, en la siguiente sección se hará un análisis comparativo de lo aquí presentado.

Análisis y discusión de resultados

En esta sección se abordarán los puntos importantes a los que se llegó en el presente trabajo, haciendo un análisis de los resultados que se enlistaron en la sección anterior.

Sobre las respuestas gráficas de la simulación en GADGET

Lo primero a comentar es que todas las imágenes obtenidas reflejan fielmente los datos extraídos de los archivos binarios. La resolución máxima de la imagen fue de 100x100, pero sin duda, teniendo potencia de procesamiento, puede llegarse a tener una resolución donde un píxel represente a una partícula de la simulación, con solo ajustar unas líneas de código, sin que afecte al programa entero, debido a la naturaleza modular en que fue desarrollado.

Por tener una comparación, hice una gráfica con una división de 10,000 x 10,000 la cuadrícula usada en el Código 4 y la Figura 6. El resultado lo podemos observar en la Figura 42, notamos que solo es viable para fines estéticos, ya que con la imagen de baja resolución se pueden obtener los mismos resultados finales, pero con el punto negativo es que crece el número de datos a graficar de 10,183 a 764,447 y el tiempo de ejecución del módulo encargado de hacer la gráfica aumenta de 3.2 s a 11.1 s, que es casi tres veces más y se vuelve muy impráctico para fines de lecturas en masa.

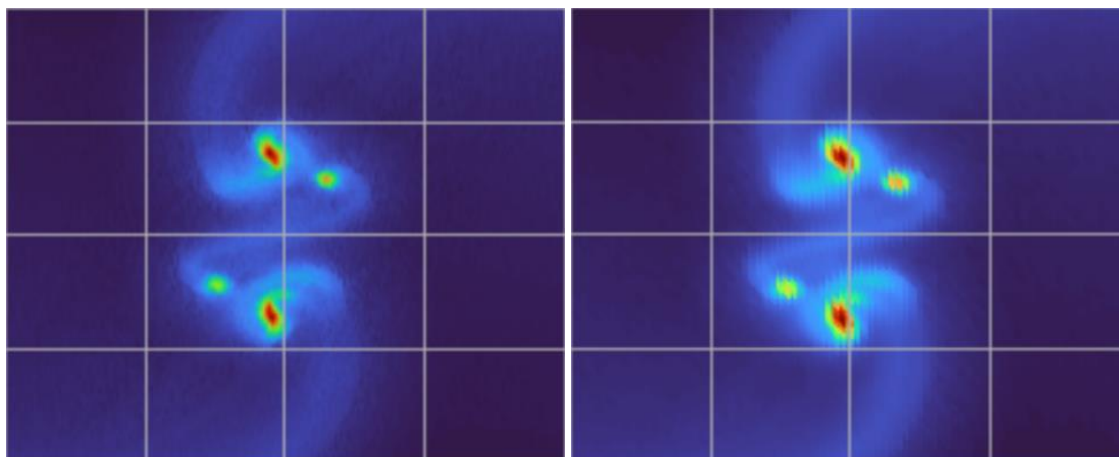


Figura 42. Comparación de resolución en la representación gráfica.

En términos generales el código implementado se comporta bien y entrega los resultados esperados. Salvo por dos situaciones en concreto. La primera es en el *Snap* 0130, no detecta al objeto 4

debido a la cercanía al objeto 2 (Figura 20), reducir criterio del radio haría que fuera detectado, pero haría que pudiera detectar pequeños ruidos en forma de ruido. La senda es el *Snap* 0150, aquí hay un caso muy particular, como se puede ver en las gráficas de la densidad de la Figura 22, el objeto 4 no aparece a pesar de que en la representación se ve muy claro que existe. El problema es el algoritmo implementado ya que dicho objeto tendría que aparecer como un pico en los datos, pero sucede que justo es interferido por los objetos 2 y 3 en las vistas x y y , respectivamente.

Sobre la evolución de los fragmentos protoestelares

Quisiera empezar con dos imágenes sobre como son las trayectorias de los cuerpos 1 a 4, la primera (Figura 43), un tanto caótica, pero refleja como fue el “baile” que tuvieron los cuerpos y se aprecia como la interacción gravitatoria conformó dos sistemas bien definidos. La segunda imagen (Figura 44), ya más noble, muestra justo las trayectorias de estos sistemas, ambas imágenes son similares ya que muestran lo mismo, pero desde dos formas diferentes de interpretarlos.

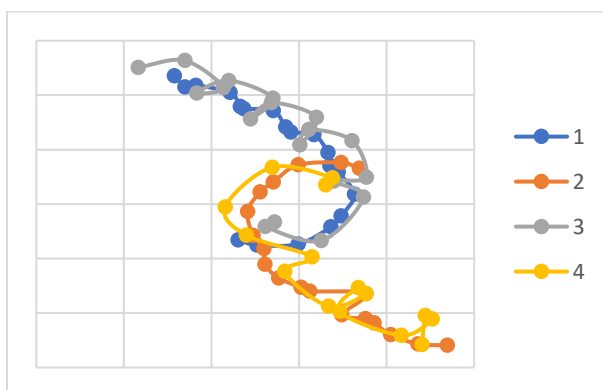


Figura 43. Trayectorias de los cuatro fragmentos protoestelares.

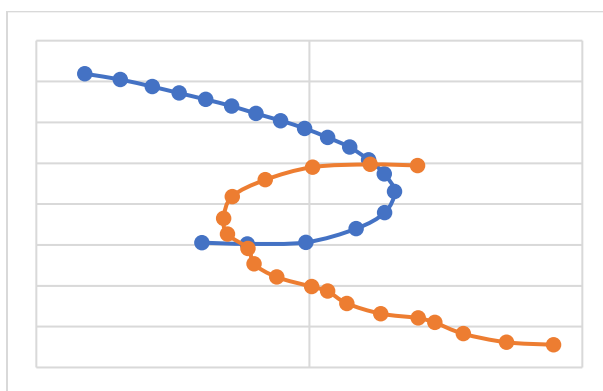


Figura 44. Trayectoria de los sistemas 1-3 y 2-4.

Velocidad.

Haremos una comparación entre las magnitudes de las velocidades de cada objeto para ver cómo es su comportamiento.

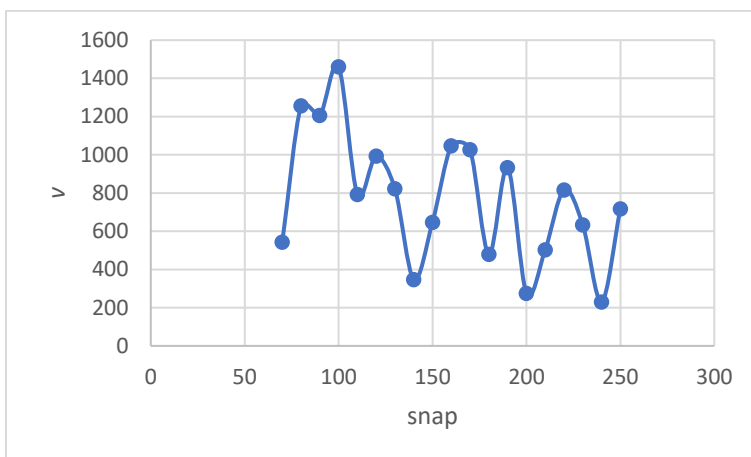


Figura 45. Velocidad del objeto 1.

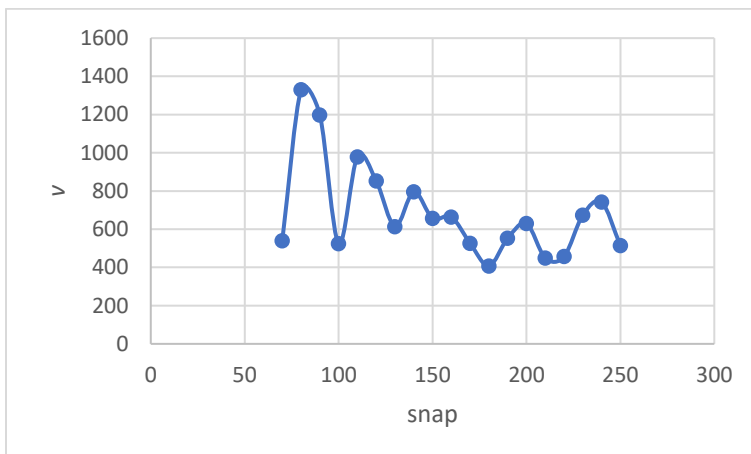


Figura 46. Velocidad del objeto 2.

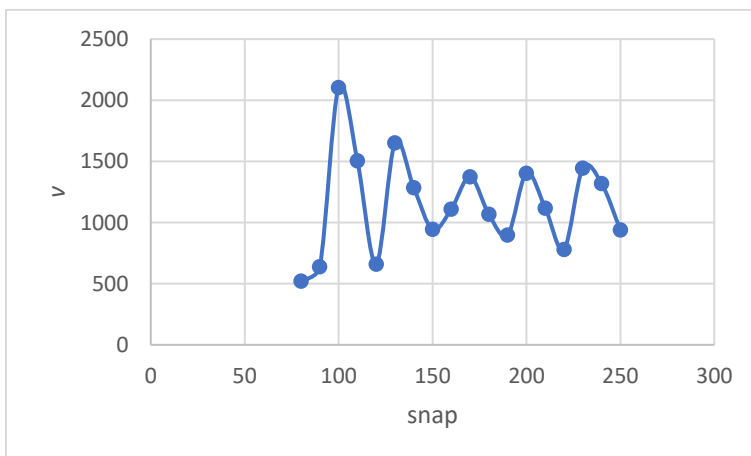


Figura 47. Velocidad del objeto 3.

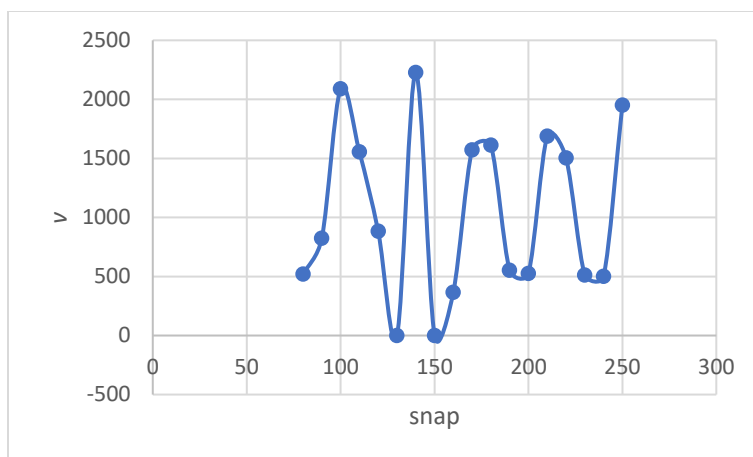


Figura 48. Velocidad del objeto 4.

Lo que quería hacer notar con esto es como la velocidad tiene esa tendencia oscilatoria, por desgracia la simulación no arroja más de un periodo en cuanto a su posición y no se lograría ver la oscilación. Pero aún así, se refleja bien en las velocidades.

Otra cosa para notar es como la velocidad tiene una tendencia a reducir su amplitud, en otras palabras, los objetos tienden a desacelerar con el paso del tiempo.

Densidad.

Si bien el dato de la densidad solo tuvo relevancia para encontrar la cantidad de objetos en la simulación también se analizó y ver qué pasa con la ella, pero esta vez comparando los cuerpos dentro de su sistema (1-3 y 2-4). Los resultados son los siguientes.

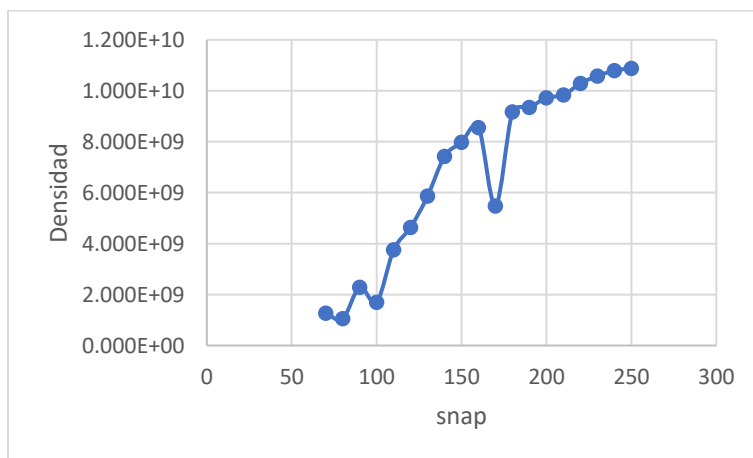


Figura 49. Densidad del objeto 1.

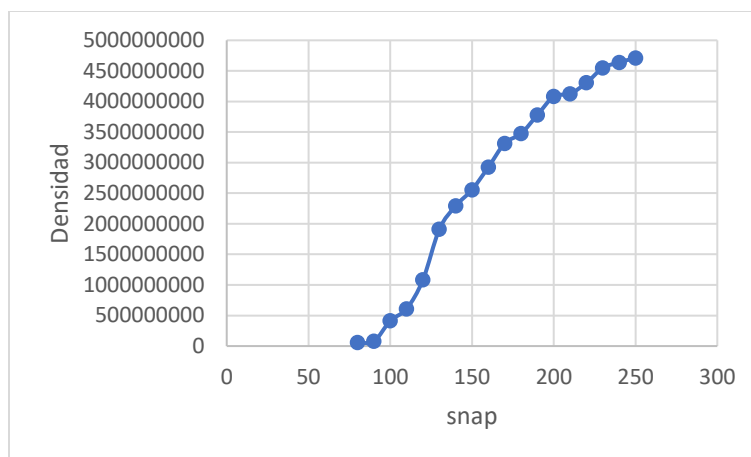


Figura 50. Densidad del objeto 3.

Lo primero a comentar es un error del programa de extracción de datos que se hace notar cuando observamos la Figura 49. En la lectura del *Sanp* 0170, la densidad cae de manera abrupta, saliendo de su tendencia, esto se debe a algo similar como lo que pasó con el Snap 0150. Si miramos la densidad desde el eje x notamos como casi se sobrepone el objeto 1 con el objeto dos, esto genera que el programa arroje un rango de lectura en x muy pequeño, en consecuencia, muchas partículas quedan fuera y no aportan al valor de la densidad.

Fuera de esta discrepancia en el programa de extracción, ambas gráficas se comportan de manera adecuada y esperada, ya que lo lógico es que la densidad aumente en el tiempo, debido a que, una mayor cantidad de partículas (por ende, masa) se aglomeran en un volumen determinado gracias al efecto de la gravedad.

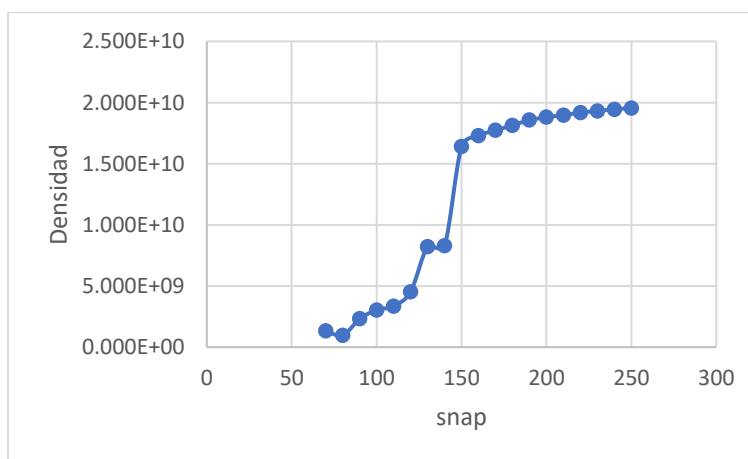


Figura 51. Densidad del objeto 2.

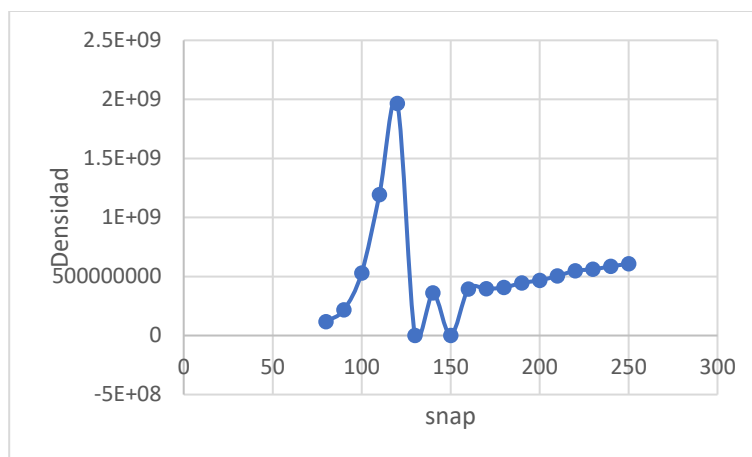


Figura 52. Densidad del objeto 4.

Analizar estas dos graficas como un sistema nos ofrece más información, es muy notorio como en la gráfica del objeto 4 tiene una caída abrupta entre el *Snap* 0120 y el 0130 mientras que en la gráfica del objeto 2 la densidad crece mucho. Como vimos en las representaciones gráficas el objeto 4 aparentemente pasa a formar parte de 2 pero casi de inmediato de vuelven a separar, esta unión es el que observamos en las gráficas. Después de esta eventualidad, las densidades crecen como de manera normal. Los picos a cero en el objeto 4 son debido a los errores del programa mencionados en la sección anterior.

Masa.

Igual que el análisis anterior, conviene analizar pares de gráficas dentro del sistema a que pertenecen.

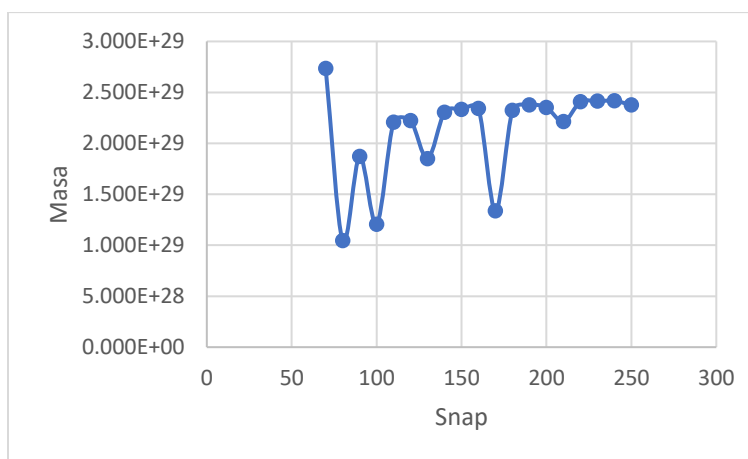


Figura 53. Masa del objeto 1.

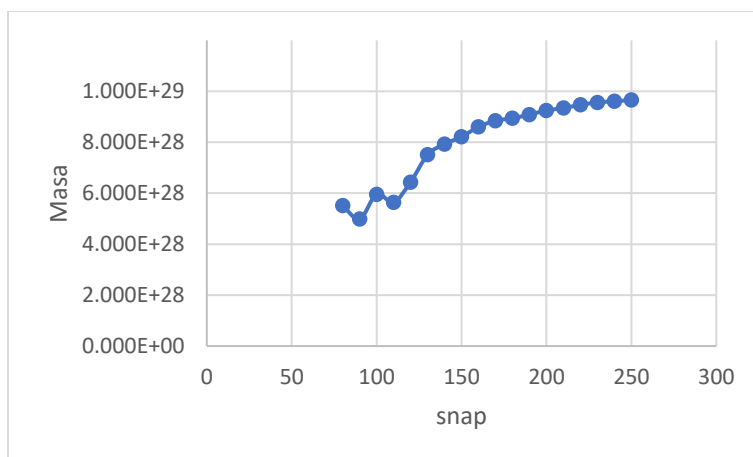


Figura 54. Masa del objeto 3.

La primera caída en la masa del objeto 1 es justo por la aparición del objeto 3, ya que este se comienza a ver en el *Snap* 0080. Las subsecuentes subidas y bajadas que se observan son debidas al intercambio de masa que hay entre estos cuerpos, ya que cuando baja en uno, sube el otro.

En el objeto 1 aparece de nuevo el error del *Snap* 0150 mencionado en la parte de las densidades y aparentemente sucede algo similar en el *Snap* 0210, el cual no se trata de un bajón por ceder masa, ya que se observaría una subida en la gráfica del objeto 3.

la tendencia de las masas es que van creciendo, pero no lo hace de manera abrupta como las densidades, el crecimiento es más suave y parece estabilizarse a un valor que ya conservaría en el tiempo, ya que en número de partículas ya no crece, puesto que no pueden aparecer partículas de la nada.

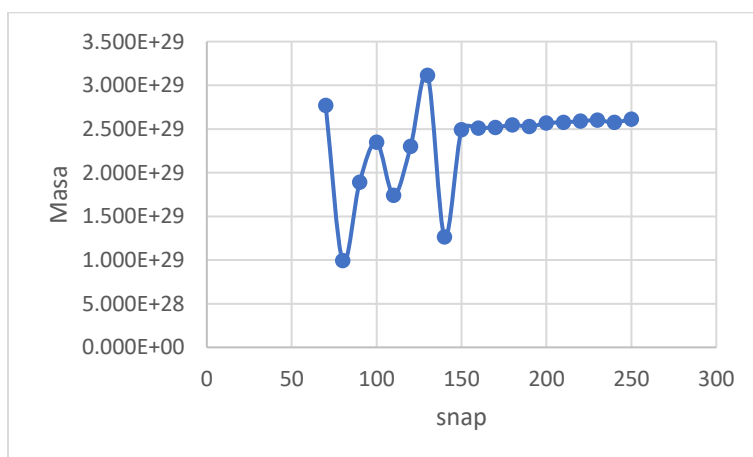


Figura 55. Masa del objeto 2.

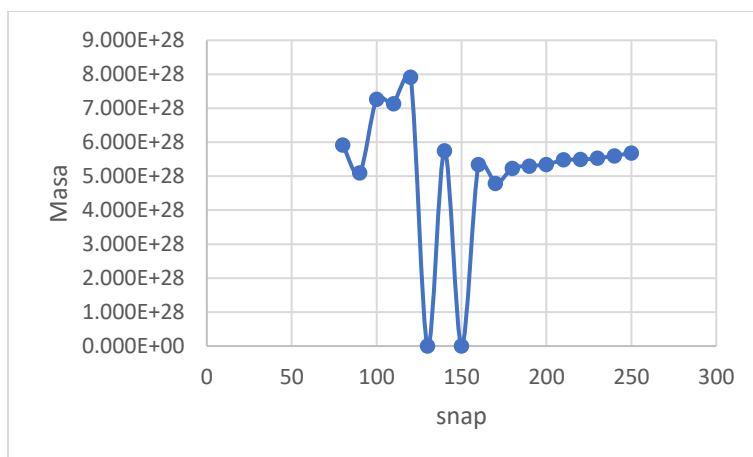


Figura 56. Masa del objeto 4.

En este sistema sucede casi lo mismo que en el anterior, comienza la partícula principal con una masa mayor, sucede la fluctuación por el intercambio de materia entre objetos y la tendencia a crecer hasta un punto estable se nota.

Las diferencias es que volvemos a ver los errores en la detección del objeto 4 en los *Snap* 0130 y 0150, pero también notamos la posible colisión entre que ocurrió en los cuerpos de este sistema, ya que la masa 4 va creciendo hasta el punto justo donde cae de manera abrupta y la masa 2 tiene un crecimiento grande.

Sobre la simulación de N-cuerpos

Hay poco que hablar sobre el código de la simulación de N-cuerpos, ya que funciona sin aparentes errores. Al menos se comportó bien en todo lo que le fue pedido que hiciera.

En cuanto a los resultados quiero comenzar argumentando la trayectoria que tienen los cuerpos en la respuesta al sistema principal de $N = 4$, vemos como el hablando desde el punto de vista de los sistemas 1-3 y 3-4, estos tendrán una tendencia de alejarse sin aparente retorno, lo que se traduce en el que ambos dejan de sentir la atracción gravitatoria que haría que volvieran a estar cerca.

Como ejercicio extra me tomé la libertad de hacer una simulación tomando como valores iniciales a los datos del *Snap* 0070, justo al comienzo, cuando aún no aparecen el cuerpo 3 y 4 y este es el resultado (Figura 57). La duración que le di fue de 40,000 años.

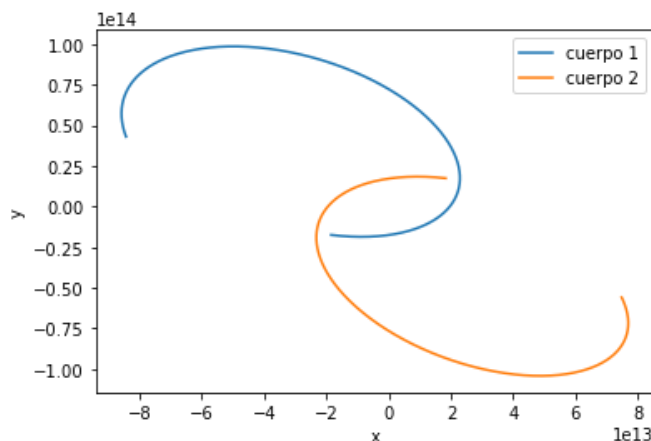


Figura 57. Simulación de N-cuerpos con datos iniciales del primer Snap (0070).

Como se puede observar, con esos valores iniciales, los cuerpos tenderían a volver y tener una aparente órbita elíptica. Aunque no sucede esto en lo que reflejan mis resultados, sí es la misma tendencia que tienen los cuerpos de las *snap*s.

Lo que sucede en los sistemas aislado 1-3 y 2-4 es que ambos fragmentos protoestelares que los componen forman trayectorias elípticas alrededor de su centro de masa, justo como un sistema planetario. Además, notamos como estas trayectorias reflejan sistemas oscilatorios, que se puede apreciar mejor cuando vemos la posición de cada eje por separado con respecto al tiempo (Figura 37, Figura 38, Figura 40 y Figura 41).

Conclusiones

Los objetivos planteados para el presente proyecto fueron alcanzados y cumplidos satisfactoriamente.

En definitiva, vemos los beneficios del uso de la tecnología en la física y en general en cualquier ciencia, ya que, podemos analizar fenómenos en el tiempo sabiendo cuáles son sus relaciones matemáticas y estas siendo simuladas a través de un lenguaje de programación. Esto abrirá muchas más oportunidades a conocer y predecir el comportamiento de todo lo que nos rodea sin esperar largos periodos.

Lo que se hizo en este proyecto refleja cómo puede analizarse un proceso complejo en una forma más fácil. Es claro que hacer la simulación en GADGET requiere de conocimiento mucho más profundo que escapa a aprendido en este nivel educativo, pero llegan a un punto donde podemos aplicar conocimientos relativamente más sencillos y que a este nivel se manejan por de más bien, los resultados no cambian demasiado, son buenas aproximaciones y el beneficio es la practicidad.

Entrando en materia, tiene muchos beneficios el conocer un poco de manipulación de grandes cantidades de datos para la resolución de los problemas aquí atacados. Se sabe que no existe un camino único para llegar a un fin, pero tener conocimientos sobre la herramienta que usas te garantiza encontrar la forma más practica y que entregue mejores resultados.

Se consiguió la meta de saber extraer información de simulaciones físicas y darles una interpretación gráfica. Hasta antes de este trabajo me era muy complicado concebir imágenes de objetos extraños en el universo que no se pueden ver, pero se tienen datos. Ahora sé cómo pasar de información a una imagen que puedo apreciar mucho mejor y es satisfactorio.

Logramos saber dar criterios para la asignación de masa a los objetos, tomando como rangos que delimitaron a nuestro objeto, a la posición de las partículas que tuvieran al menos un 1% de la densidad de los picos.

Vale la pena mencionar que siempre se puede mejorar un resultado o en este caso, el programa resultante para la lectura de datos de la simulación, ya que, en el afán de hacerlo automático, se comprometió el funcionamiento que se tradujeron en cuatro discrepancias que pudimos encontrar al analizar los datos. La parte de entregar una respuesta gráfica no implicó mayor problema, pero al momento de reconocer los objetos y asignar masas es donde no cumplió al cien con lo esperado, pero sí estuvo por encima de un 90% de efectividad.

Queda como constancia la asignación de centros de masa a los fragmentos protoestelares de que las computadoras nos ayudan mucho al momento de no trabajar en exceso, trabajamos con cientos de

miles de partículas que aportaban masa al sistema, calcular eso a mano pudo haber tomado días, quizás hasta meses, pero se redujo a tan solo unos cuantos segundos sabiendo implementar bien unas cuantas líneas de código.

Logré comprender la importancia del cambio de marco o plano de referencia inercial, como físicos normalmente estamos acostumbrados a analizar todo siempre desde un punto de vista para no complicar el proceso, pero tiene muchas ventajas para encontrar mejores comportamientos o cosas que no se pueden ver al no cambiar.

Como se mencionó en un principio, el problema de los N-cuerpos fue muy complejo en su tiempo cuando nació como consecuencia de las leyes de Newton, pero gracias a la computación el problema puede ser analizado con relativa facilidad y no solo ese problema, ya que la mayoría de los fenómenos puede explicarse con una ecuación diferencial, que conociendo, se puede implementar en código. El algoritmo no fue ni siquiera complejo, gracias a que solo analizamos 4 cuerpos como mucho y ya se han desarrollado muchos métodos matemáticos para la resolución de ecuaciones muy complejas.

Queda también como segunda constancia de lo mucho que nos ayudan las computadoras en la solución del problema de N-cuerpos, ya que el método matemático usado requiere de dividir el tiempo en intervalos muy pequeños además de que estas iteraciones tienen que ser multiplicadas por N^2 operaciones correspondientes a las interacciones gravitatorias entre un cuerpo y otro.

Como era de esperarse, según lo dicho por Poincaré, los resultados a las ecuaciones de movimiento gravitacional arrojan trayectorias cónicas y justo fue lo que se obtuvo. Las respuestas a los sistemas 1-3 y 2-4 fueron elipses y el sistema principal, puede que resultara en un movimiento hiperbólico ya que los cuerpos se alejaron sin posibilidad de retorno. Este fue el resultado final que tuvimos, los fragmentos protoestelares resultaron convertirse en dos sistemas binarios por separado y bien definidos.

También se comprobó como los sistemas planetarios son de hecho osciladores armónicos, con sus frecuencias y amplitudes bien definidas.

Como nota final, usé buenas prácticas de programadores para desarrollar ambos códigos, ambos fueron escritos de manera modular, para que puedan realizarse cambios a alguna función en específico sin que se afecte el resto del programa. También lo hace escalable, abierto a mejoras y fácil de implementar en cualquier otro lenguaje de programación que sea de la predilección de cada desarrollador, que cumpla claro con el manejo de grandes datos din problemas.

Referencias

- Aarseth, S. J. (2003). *Gravitational N-Body Simulation*. Cambridge, New York: Cambridge University Press.
- Aguilar, L. A. (1992). El problema de N cuerpos en Astronomía. *Revista Mexicana de Física*, 38(5), 701-738.
- Artigue, V., & Pollio, A. (2013). Aspectos Históricos del Problema de los N Cuerpos y su Influencia en el Desarrollo de la Física y La Matemática: Motivación para Diseñar Actividades de Enseñanza. *Actas del VII CIBEM*, 3954-3961.
- Bellomo, N., & Preziosi, L. (1995). *Modelling Mathematical Methods and Scientific Computation*. CRC Press, Inc.
- Bonilla, A. (23 de 08 de 2018). *Abacus, la supercomputadora más poderosa de México*. Obtenido de Cienciamx NOTICIAS: <http://www.cienciamx.com/index.php/tecnologia/tic/23424-abacus-supercomputadora-mexico>
- Bor, G., & Montgomery, R. (2014). Poincaré y el problema de n-cuerpos. *Miselánea matemática*, 83-102.
- Chapra, S., & Raymond, C. (1988). *Numerical Methods for Engineers*.
- Garrido, R. (6 de 07 de 2018). *En México también hay supercomputación, estas son las 8 supercomputadoras más potentes en el país*. Obtenido de Xataka México: <https://www.xataka.com.mx/otros-1/en-mexico-tambien-hay-supercomputacion-estas-son-las-7-supercomputadoras-mas-potentes-en-el-pais>
- Goldstein, H. (1987). *Classical Mechanics*. Addison-Wesley.
- Newton, I. (1687). *Philosophiæ Naturalis Principia Mathematica*.
- Poincaré, H. (1892-1899). *Les Méthodes Nouvelles de la Mécanique Céleste*.
- Santana Ortega, A. (2018). Experimentación, modelación y simulación matemática en la formación de profesoras de telesecundaria. *Memorias CONISEN 2018*(5), 42.
- Sigalotti, L. D., Cruz, F., Gabbasov, R., Klapp, J., & Ramirez-Velazquez, J. (2018). From Large-scale to Protostellar Disk Fragmentation into Close Binary Stars. *The Astrophysical Journal*, 857(40), 1-11.
- Solbes, J. (2011). ¿Por qué resulta tan difícil la comprensión de la astronomía a los estudiantes? *Didáctica de las ciencias experimentales y sociales*(25), 187-211.
- Springel, V. (2005). The cosmological simulation code GATGET-2. *The Author. Journal compilation*, 1105-1134.
- Thompson, R. (2014). pyGadgetReader: GADGET snapshot reader for python. *Astrophysics Source Code Library(ascl:1411.001)*. Obtenido de <https://github.com/jveitchmichaelis/pygadreader>
- Thompson, R., Davé, R., & Nagamine, K. (2015). The rise and fall of a challenger: the Bullet Cluster in Λ cold dark matter simulations. *Monthly Notices of the Royal Astronomical Society*, 452, 3030-3037.

Velten, K. (2009). *Mathematical Modeling and Simulation, Introduction for Scientists and Engineers*. WILEY-VCH Verlag GmbH & Co.

Young, H., & Freedman, R. (2013). *Física Universitaria* (Décimo tercera ed., Vol. 1). PEARSON.

Entregables

Los entregables para este proyecto son los códigos del programa de extracción de datos de simulaciones hechas en GADGET code y el programa de simulación de N-cuerpos. A ambos se puede tener acceso en la siguiente liga:

https://github.com/tdack21/n_body