UNIVERSIDAD
REY JUAN CARLOS

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Curso Académico 2012/2013

Proyecto de Fin de Carrera

IMPLEMENTACIÓN DE UN ALGORITMO PSO PARA BÚSQUEDA DE ÓRBITAS PERIÓDICAS EN SISTEMAS CAÓTICOS

Autor: Miguel Ángel Montes García

Tutores: Juan Carlos Vallejo Chavarino

Jesús Seoane Sepúlveda


UNIVERSIDAD REY JUAN CARLOS
MCMXCVI

Agradecimientos

La culminación de este proyecto no hubiese sido posible sin la infinita paciencia mostrada por Juan Carlos Vallejo Chavarino, tutor inicial del proyecto; así como tampoco sin la inestimable ayuda de Jesús Miguel Seoane Sepúlveda, profesor titular del departamento de física de la URJC con sus amplios conocimientos sobre el tema en el que he trabajado.

Agradecer también a todos mis compañeros de carrera y amigos que me han ayudado tanto en los buenos momentos como en los más difíciles para llegar a este punto.

También agradecer a mi familia el apoyo recibido y la comprensión mostrada ante las dificultades que me he ido encontrando a lo largo de este tiempo.

Pero por encima de todo, tengo que agradecer a mi padre la ilusión y la confianza que depositó en mí cuando elegí el camino de la informática y por ello con estas líneas quería que quedase constancia de ello.

Allá donde esté, seguro que se encuentra muy orgulloso por ver el trabajo concluido.

Gracias a todos.


UNIVERSIDAD REY JUAN CARLOS
MCMXCVI

Resumen

El proyecto descrito en las siguientes hojas ha tenido como objetivo principal la implementación de un algoritmo de búsqueda de órbitas periódicas mediante el uso de la técnica denominada como PSO (*Particle Swarm Optimization)* en un sistema dinámico de partículas.

El sistema sobre el que se ha trabajado es conservativo con la energía, de tipo Henon-Heiles y la idea del algoritmo se basa en explorar el comportamiento de una serie de puntos a lo largo del tiempo y comprobar su intersección con una sección de Poincaré.

La implementación del algoritmo y sus funciones auxiliares se ha realizado en lenguaje C, dado que la base del trabajo previamente realizada por el tutor Juan Carlos Vallejo se encuentra en este lenguaje de programación y se consideró que ofrecía la suficiente versatilidad para poder alcanzar los objetivos definidos.

La representación gráfica de las distintas órbitas se ha realizado mediante el uso del programa open source *TopCat* [9] *(Tool for OPerations on Catalogues And Tables),* el cual permite trabajar con la información de los puntos generada en una tabla durante la fase de pruebas.

Dadas las grandes posibilidades de investigación futura que ofrece este trabajo, se describen distintas propuestas a valorar y que están enfocadas principalmente a su optimización y eficiencia como puede ser la computación distribuida (*Grid*), o la mejora de determinadas funciones del algoritmo.


UNIVERSIDAD REY JUAN CARLOS
MCMXCVI

Índice

1. Introducción

1.1. Presentación del trabajo

Un sistema dinámico es un sistema físico que describe el comportamiento de objetos o partículas que tienen una evolución en su estado a lo largo del tiempo. Dentro de un sistema dinámico, una órbita periódica es una solución particular al sistema cuya característica principal es la de mostrar un comportamiento que se repite a lo largo del tiempo en forma de ciclos.

El estudio del comportamiento de los sistemas dinámicos no lineales (aquellos en los que no es posible representar su comportamiento mediante ecuaciones lineales) ha sufrido un avance muy significativo con la llegada de los computadores. Esto se debe a que muchos de los cálculos implican a menudo la resolución de ecuaciones diferenciales complejas y muy difíciles de resolver de manera analítica.

Sin lugar a dudas, la figura del matemático francés Henri Poincaré (figura 1), representa una de las principales fuentes de conocimiento que permiten la realización del presente trabajo. A finales del siglo XIX, usó por primera vez la denominada aplicación de Poincaré en el estudio del problema de los tres cuerpos (generalizado posteriormente como problema de los n-cuerpos) [4]; la aplicación consistía en restringir el estudio de las partículas de un sistema dinámico a un subespacio de dimensión inferior denominado sección de Poincaré. Dicha sección permitiría representar la evolución de las partículas del sistema dinámico representando su corte con la sección de Poincaré, definida transversalmente al flujo del sistema (figura 2).

Figura 1: Henri Poincaré (1854-1912)

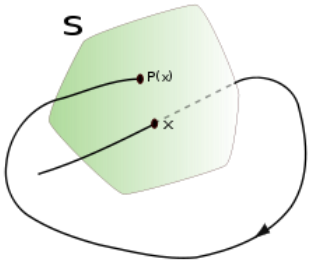


Figura 2: representación de la aplicación de Poincaré sobre la sección S.

No será hasta un siglo después cuando el otro componente principal de nuestro trabajo, el algoritmo PSO, sea descrito por primera a finales del siglo XX por James Kennedy y Russel C. Eberhart [7].

Las siglas PSO se refieren a *Particle Swarm Optimization*, (Optimización de Enjambres de Partículas en castellano) y se basa en una idea observada en la naturaleza por ejemplo en comunidades de insectos. A grandes rasgos, el algoritmo propone que un conjunto de partículas (el enjambre) explore un espacio predefinido en busca de un objetivo y se desplacen a nuevas zonas de exploración en caso de no tener éxito condicionadas por aquellas que se encuentren en las mejores posiciones de la anterior exploración. Este algoritmo tiene utilidades en distintos campos de la ciencia y la tecnología tales como algoritmos genéticos [6], computación distribuida o en la planificación de rutas de transporte óptimas para vehículos [1].

En el año 2005, aparece en un artículo [8] publicado en la revista mensual de la *Royal Astronomical Society* la idea de aplicar el algoritmo PSO en la búsqueda de órbitas periódicas atendiendo a la sección de Poincaré generada por cada partícula. El artículo lo firman Charalampos Skokos, Konstantinos E. Parsopoulos, Michael N. Vrahatis y Panos A. Patsis, investigadores de las universidad de Patras, Grecia y el Centro de Investigación de Astronomía en Atenas.

1.2. Estructura de la memoria

La presente memoria está estructurada en seis capítulos, definidos de la siguiente manera:

- El presente capítulo, a modo de introducción al PFC desarrollado, permite contextualizar y situar cronológicamente el origen de los estudios que se han tomado como referencia así como su idea general.

- El segundo capítulo presenta los objetivos principales y secundarios del proyecto así como de la metodología de trabajo empleada durante su desarrollo.

- El tercer capítulo incluye en primer término la especificación del algoritmo desde el punto de vista matemático para posteriormente describir la especificación, diseño e implementación del programa.

- El cuarto capítulo explica el proceso de ejecución del programa para posteriormente analizar los resultados obtenidos.

- El quinto capítulo recoge las conclusiones obtenidas tras la evaluación de los resultados obtenidos, para después proponer una serie de mejoras y trabajos futuros para ampliar el alcance del trabajo.

- El sexto y último capítulo presenta la bibliografía utilizada durante todo el proceso de desarrollo y posterior redacción de la memoria.

Junto con la memoria se encuentra un CD en el que se puede encontrar el código fuente de la aplicación así como las indicaciones necesarias para su configuración y ejecución.

2. Objetivos

El objetivo principal del presente proyecto es implementar un algoritmo de búsqueda de órbitas periódicas, usando para ello un algoritmo de tipo PSO que será adaptado a nuestro problema.

Para alcanzar el objetivo principal, se definen los siguientes objetivos parciales:

1. Estudio de modelos matemáticos y físicos implicados en el proyecto.
2. Desarrollo de las estructuras y funciones necesarias derivadas del análisis de cada apartado del artículo.
3. Pruebas y aceptación de la aplicación desarrollada.

La especificación e implementación del algoritmo se realizará atendiendo a la información contenida en la publicación *Particle Swarm Optimization: An efficient method for tracing periodic orbits in 3D galactic potentials* al que hicimos referencia en la introducción del trabajo.

Los resultados de la implementación de nuestro algoritmo se presentan mediante la sección de Poincaré del punto obtenido al evaluarlo durante un periodo de tiempo con la ayuda de un integrador numérico. De esta manera obtendremos una gráfica con todos los cortes que efectúa la órbita con el plano y podremos observar cómo se concentran en un número concreto de puntos dependiendo del orden de la órbita periódica obtenida.

Para evaluar el grado de éxito de los resultados que arroje la ejecución, se espera que las órbitas obtenidas sean similares a alguno de los tipos que se muestran en las siguientes figuras (figura 3 - figura 6), y que sirven como referencias para catalogar órbitas según su tipología [2].

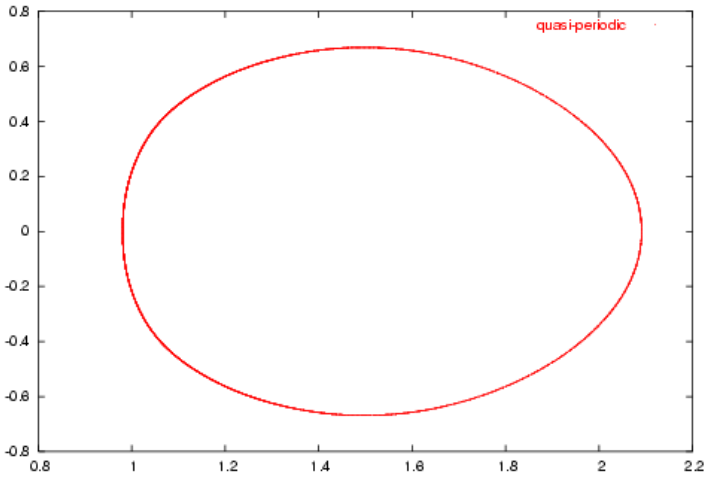


Figura 3: sección de órbita cuasi-periódica

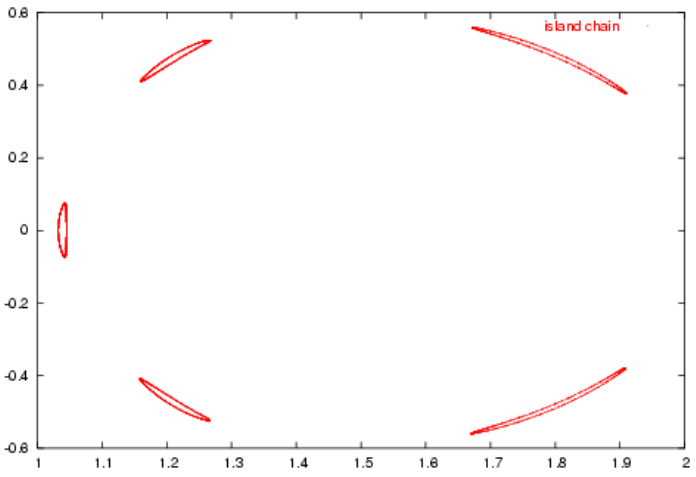


Figura 4: sección de órbita periódica cuyos cortes forman islas en el plano

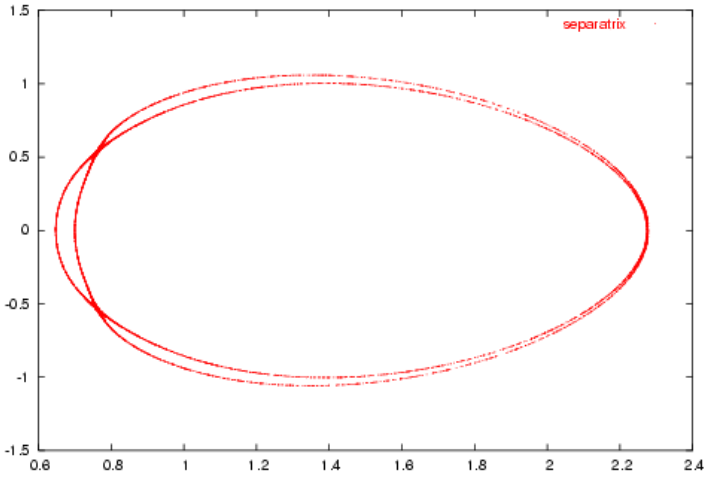


Figura 5: sección de órbita de tipo separatrix

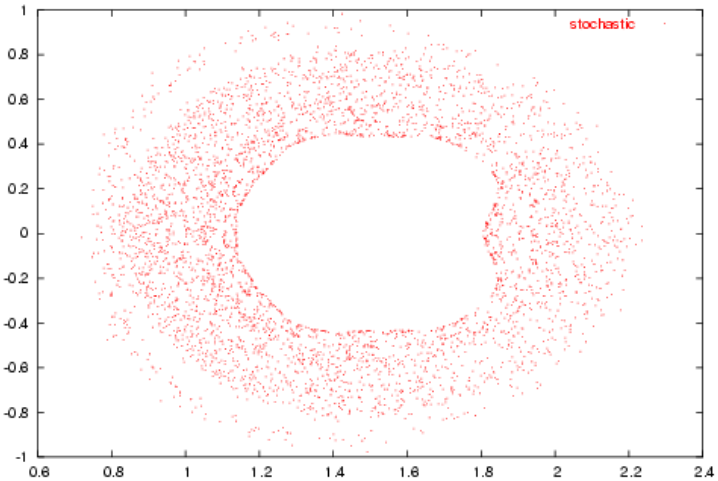


Figura 6: sección de órbita de tipo estocástica.

2.1. Metodología de trabajo

El hecho de compaginar la realización de este proyecto fin de carrera con el mundo laboral, permite poder aplicar determinadas metodologías de trabajo y sanas costumbres en la resolución del problema.

Debido a la fuerte componente matemática del proyecto, se ha seguido un modelo de desarrollo en espiral en el que para cada iteración del programa, se ha ido optimizando una parte del programa, incorporando nuevas funcionalidades hasta concluir en la última iteración con la solución completa.

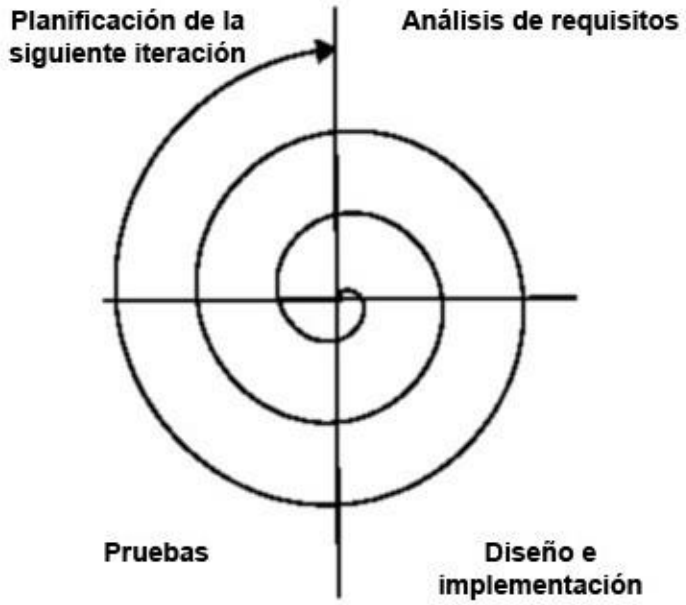


Figura 7: etapas implicadas en el modelo de desarrollo en espiral.

Para este proyecto, se han realizado cuatro ciclos de trabajo organizados de la siguiente manera:

- Comprensión y estudio de la parte matemática y física que permite el cálculo de la sección de Poincaré y su aplicación para nuestro trabajo.

- Desarrollo y pruebas de una solución primitiva que permite estudiar para más de una partícula la sección de Poincaré y evaluar el resultado mediante la resta simple de los puntos obtenidos.

- Desarrollo de la primera implementación del algoritmo PSO sobre un número pequeño de condiciones iniciales y uso de una función básica de movimiento de partículas.

- Desarrollo de la función de movimiento de partículas completa tal como se describe en el artículo de referencia e integración de las nuevas funciones con el resto de código.

2.1.1. Ventajas del modelo

Este modelo de desarrollo permite obtener soluciones incrementales minimizando riesgos en cada ciclo de trabajo, ya que en cada uno de ellos se están realizando análisis de nuevos requisitos, desarrollos y también de manera indirecta el mantenimiento de la aplicación (detección y corrección de incidencias durante las pruebas de los sucesivos ciclos).

2.1.2. Inconvenientes del modelo

El principal inconveniente de este modelo es la alta exigencia de recursos que se han de destinar a él para cumplir los objetivos en cada una de las etapas del mismo; esto provoca que en el ámbito laboral, donde normalmente prevalece el factor económico y el tiempo invertido en desarrollo, no se opte por este tipo de modelo.

3. Especificación y desarrollo del trabajo

3.1. Consideraciones previas

El proyecto parte de un trabajo original realizado por Juan Carlos Vallejo consistente en un conjunto de librerías en el que se encuentra implementada la parte matemática encargada de realizar las integraciones sobre las partículas hasta que se produce un corte con la sección de Poincaré, cuyo plano está definido en las librerías proporcionadas.

Como punto de partida, se mostró a modo de ejemplo un algoritmo básico de cálculo de órbitas periódicas en el cual se evaluaba una partícula que se sabía periódica y se aplicaba un algoritmo básico consistente en restar los vectores que representaban cada corte con la sección de Poincaré y comprobar que la distancia era cercana a cero.

Con este ejemplo, se aseguraba el correcto funcionamiento del integrador numérico resultaba y el programa producía resultados válidos.

3.2. Requisitos

Los requisitos establecidos al inicio de proyecto han permitido delimitar la línea de trabajo y así evitar desviaciones en cada una de las etapas que lo componen:

- La solución debe hacer uso de las librerías proporcionadas para el cálculo matemático.
- Se utilizará el lenguaje de programación C para poder integrar la implementación del algoritmo con el código proporcionado.
- El algoritmo de integración usado será el de Runge-Kutta-Fehlberg, probado con éxito en una versión de demostración.
- Los ficheros obtenidos tras la ejecución seguirán una serie de normas de estilo fijadas por el tutor Juan Carlos Vallejo.
- El resultado de la ejecución del algoritmo se verificará usando para ello un integrador implementado con la librería aplicado sobre las condiciones iniciales obtenidas (en caso de encontrarlas).
- El orden de magnitud para la función objetivo es, al menos de 10^{-5}

3.3. Entorno de desarrollo

El diseño, implementación y pruebas de la solución se han llevado a cabo utilizando la plataforma Eclipse [3] y más concretamente el plugin CDT que proporciona un IDE para aplicaciones escritas en lenguaje C o C++. La versión usada es la 8.1.2 para Eclipse Juno obtenida desde la página web del proyecto CDT.

El sistema operativo principal usado para la plataforma Eclipse ha sido una distribución de Linux basada en el proyecto Debian: Ubuntu. En las últimas etapas del proyecto se ha usado también la distribución Linux Mint, de características muy similares a Ubuntu.

Las razones de la elección de este IDE son principalmente su uso desde el 2º curso de carrera con el añadido de haberlo usado de manera prolongada en el ámbito laboral.

Una de las características más interesantes de este IDE es el denominado *Quick Preview* (figura 8), que permite previsualizar la implementación de una función con sólo pasar el puntero del ratón sobre cualquier zona de código en la que sea utilizada. Dado que parte del trabajo se apoya en librerías y funciones desarrolladas por otra persona, esta propiedad agiliza enormemente la implementación de las funciones específicas del algoritmo.

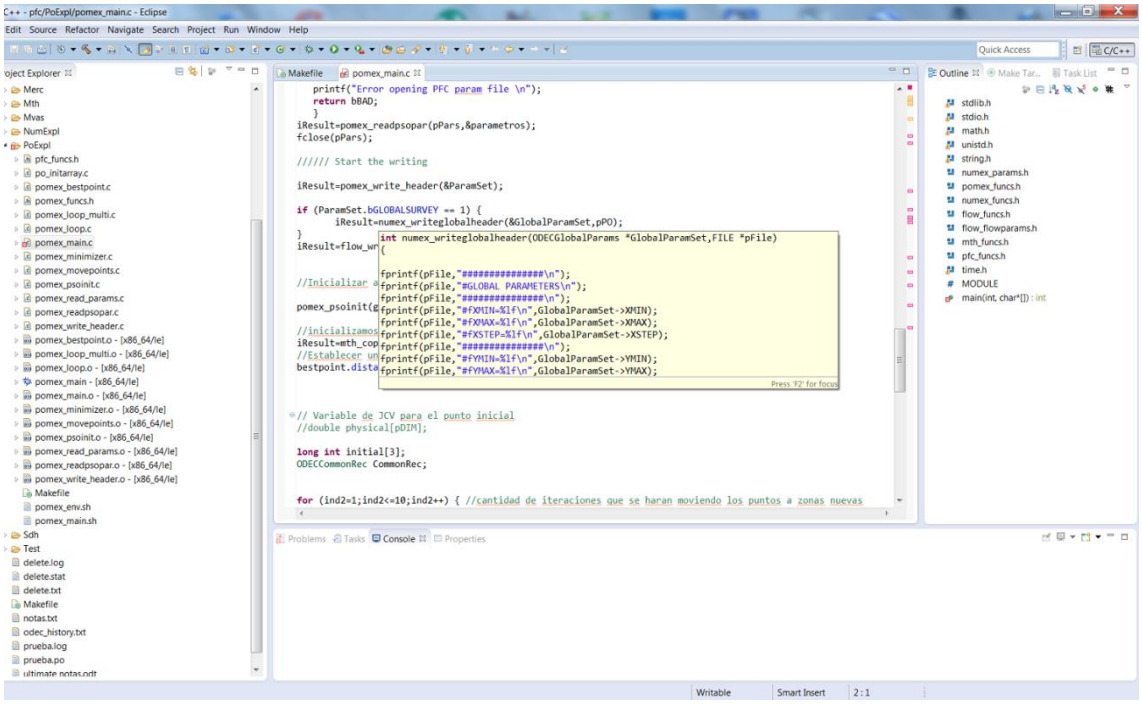


Figura 8: Función Quick Preview del entorno de desarrollo Eclipse.

La posibilidad de depurar la ejecución del programa desde el propio IDE ha permitido incluso proponer mejorasen el código de determinadas funciones proporcionadas para el proyecto.

En etapas iniciales del proyecto, el IDE usado fue *Anjuta DevStudio*, aunque tras completar la primera etapa, se dejó de utilizar por presentar algunos problemas con el depurador que provocaban una pérdida de tiempo innecesaria y totalmente evitable disponiendo de otros IDE´s al alcance de manera gratuita y más eficaces para nuestro trabajo.

3.4. Descripción del algoritmo

En su artículo [8], Skokos, Parsopoulos y Vrahatis proponen una alternativa eficiente en el cálculo de órbitas periódicas, que consiste en aplicar un algoritmo de tipo PSO para evaluar la posición de cada partícula que forma el conjunto de puntos a explorar (swarm) en el momento del corte con la sección de Poincaré establecida. Mediante la aplicación de una función matemática (el cuadrado de la distancia Euclídea entre dos puntos) se puede concluir si el nuevo punto calculado se encuentra lo suficientemente cercano al inicial como para validar la solución y considerarlo órbita periódica de un determinado orden.

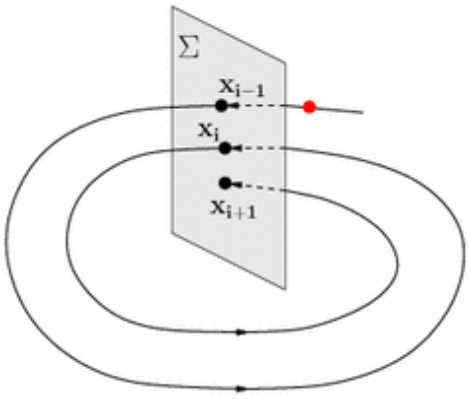


Figura 9: Intersecciones sucesivas de una órbita con la sección de Poincaré.

En caso de que ninguna partícula cumpla con la condición de "distancia cero", se desplazan hacia una zona nueva del espacio mediante dos funciones que permiten obtener la velocidad y posición de cada nuevo punto teniendo en cuenta la mejor posición obtenida hasta el momento..

Este proceso se repetirá hasta que se alcance el número máximo de iteraciones establecidas para la exploración, o bien, hasta que alguno de los puntos sea minimizador de la función de distancia, en cuyo caso detendremos la exploración.

A continuación se explica más detalladamente los conceptos fundamentales que componen el algoritmo sobre el que se va a trabajar:

Las partículas estudiadas pertenecen al espacio de fases 6D (x, y, z, ẋ, ẏ, ẑ) de un sistema Hamiltoniano en el que definimos una sección de Poincaré fijada por las condiciones y=0 e ẏ>0.

De esta manera, la partícula puede definirse empleando únicamente 4 de las 6 coordenadas ya que conociendo su Energía (representada por el hamiltoniano H), podemos obtener el resto de componentes de la siguiente expresión:

$$\boldsymbol{H = \frac{1}{2}\left(p_x^2 + p_y^2 + p_z^2\right) + V_D + V_S + V_B - \Omega_b(xp_y - yp_x)}$$

$$\boldsymbol{p_x = \dot{x} - \Omega_b y}$$
$$\boldsymbol{p_y = \dot{y} + \Omega_b}$$
$$\boldsymbol{p_z = \dot{z}}$$

Figura 10: Ecuaciones que permiten obtener el Hamiltoniano (energía).

Las cuatro variables que utilizaremos en nuestro algoritmo para describir la órbita son $\boldsymbol{X = (x, x, \dot{z}, \dot{z})}$.

Otro concepto importante del algoritmo es el cálculo del siguiente punto de corte de la órbita y la sección de Poincaré, el cual se expresa formalmente de la siguiente manera:

$$\Phi(\boldsymbol{X_0}) = [\Phi_{\boldsymbol{x}}(\boldsymbol{X_0}), \Phi_{\boldsymbol{z}}(\boldsymbol{X_0}), \Phi_{\dot{\boldsymbol{x}}}(\boldsymbol{X_0}), \Phi_{\dot{\boldsymbol{z}}}(\boldsymbol{X_0})]^{\boldsymbol{T}}; \quad \mathbb{R}^{\boldsymbol{4}} \rightarrow \mathbb{R}^{\boldsymbol{4}}]$$

Figura 11: Expresión del punto de corte de la órbita con la sección de Poincaré

Si el nuevo punto de corte está lo suficientemente cerca del inicial, cumplirá con la siguiente expresión que calcula la resta entre dicho punto y el inicial:

$$\Phi^{p}(\boldsymbol{X}) = \boldsymbol{X} \rightarrow \Phi^{p}(\boldsymbol{X}) - \boldsymbol{X} = \boldsymbol{\Theta_4} \rightarrow$$

$$\Phi(\boldsymbol{X_0}) \begin{pmatrix} \Phi_{x}^{p}(\boldsymbol{X}) \\ \Phi_{z}^{p}(\boldsymbol{X}) \\ \Phi_{\dot{x}}^{p}(\boldsymbol{X}) \\ \Phi_{\dot{z}}^{p}(\boldsymbol{X}) \end{pmatrix} - \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{z} \\ \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{z}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{pmatrix}$$

Figura 12: Condición de igualdad entre el punto de corte X respecto al inicial

El objetivo por tanto, será encontrar las condiciones iniciales de una órbita que sean minimizador global de la función descrita en la siguiente figura, en la cual se calcula el cuadrado de la distancia Euclídea entre el punto inicial y la intersección de orden *p*.

$$\boldsymbol{f(X)} = \left[\Phi_{x}^{p}(\boldsymbol{X}) - \boldsymbol{x}\right]^2 + \left[\Phi_{x}^{p}(\boldsymbol{X}) - \boldsymbol{z}\right]^2 + \left[\Phi_{x}^{p}(\boldsymbol{X}) - \dot{\boldsymbol{x}}\right]^2 + \left[\Phi_{x}^{p}(\boldsymbol{X}) - \dot{\boldsymbol{z}}\right]^2$$

Figura 13: Función Objetivo

El orden de la órbita periódica encontrada vendrá determinado por el número de intersecciones con la sección de Poincaré que han sido necesarias para encontrar dicho minimizador.

Conociendo la función objetivo y cómo están definidos los puntos iniciales, la aplicación del algoritmo PSO consiste en inicializar una conjunto de ellos (figura 14) a los que se denominará enjambre (swarm) al mismo tiempo que se inicializan sus velocidades y la mejor posición encontrada por cada uno hasta ese momento (en la primera iteración del algoritmo, coincidirá con el propio punto inicial).

$$X_i = (x_{i1}, x_{i2}, x_{i3}, x_{i4})^T = (x_i, z_i, \dot{x}_\iota, \dot{z}_i)^T \in S, i = 1 \dots N$$

$$V_i = (v_{i1}, v_{i2}, v_{i3}, v_{i4})^T, \qquad i = 1 \dots N.$$

$$P_i = (p_{i1}, p_{i2}, p_{i3}, p_{i4})^T \in S.$$

Figura 14: Expresión formal de puntos, velocidades y mejores posiciones alcanzadas.

Existen dos variantes de la aplicación del algoritmo PSO atendiendo a la manera en la que se comunican la información del mejor punto encontrado hasta el momento:

1. **Variante local:** Cada partícula está asignada a una vecindad de tamaño predefinido de tal manera que la mejor posición encontrada hasta el momento sólo es conocida por los integrantes de la vecindad.

2. **Variante global:** La mejor posición encontrada hasta el momento es conocida por todas las partículas que componen el enjambre, lo que provocará como explicaremos a continuación una convergencia más rápida hacia la solución de la función objetivo.

En el presente trabajo, la variante del algoritmo usada ha sido la global, ya que como se demostrará en la fase de pruebas, efectivamente las tasas de convergencia de puntos son bastante altas y reducen el tiempo de cálculo respecto a la variante local.

Una vez definida la tipología del enjambre y con las partículas inicializadas, cada partícula es evaluada según la función objetivo de la *figura 13* en busca de un minimizador global que nos permita encontrar las condiciones iniciales de la órbita, actualizando a su vez su mejor posición encontrada hasta el momento de la siguiente manera:

$$P_i^{(q+1)} = \begin{cases} X_i^{(q+1)}, & Si\ f(X_i^{(q+1)}) < f(P_i^{(q)}) \\ P_i^{(q)}, & En\ otro\ caso \end{cases}$$

Figura 15 - Actualización de mejor punto encontrado

En caso de no encontrar ningún minimizador de la función entre las partículas evaluadas, se procede con el desplazamiento de las mismas a nuevas zonas, teniendo en cuenta la mejor posición encontrada por cada partícula (P_i) y la mejor posición de todas las partículas evaluadas ($P_{g\,i}$).

$$V_i^{(q+1)} = \chi\left[V_i^{(q)} + c_1 r_1\left(P_i^{(q)} - X_i^{(q)}\right) + c_2 r_2\left(P_{gi}^{(q)} - X_i^{(q)}\right)\right]$$

$$X_i^{(q+1)} = X_i^{(q)} + V_i^{(q+1)}$$

Figura 16: Ecuaciones para la actualización de posición y velocidad de las partículas.

- χ : Factor de constricción
- c1 y c2: Factor cognitivo y factor social respectivamente, cuya suma tiene que ser mayor al valor que tome φ (2,05 ambos).
- r1 y r2: vectores aleatorios con componentes distribuidas en el intervalo [0,1]

Tanto el factor de constricción, como el cognitivo y el social se pueden obtener de la función expresada más abajo, aunque en nuestro trabajo se han tomado valores considerados como óptimos en el artículo tomado de referencia [8]; en concreto, χ = 0,729, φ > 4 y
c1=c2 = 2,05.

$$\chi = \frac{2K}{|2 - \Phi - \sqrt{\Phi - 4\Phi}}$$

Figura 17: Ecuación para el cálculo del factor de constricción

Mediante las ecuaciones reflejadas en la *figura 16* de la presente página, se actualizan las posiciones y velocidades de cada partícula para poder ser evaluadas de nuevo y seguir buscando un minimizador para la función objetivo. En lo referente a la velocidad calculada, se establece un parámetro denominado Vmax que servirá para acotar la velocidad resultante y evitar que tome valores problemáticos (figura 18).

$$v_{ij} = \begin{cases} v_{ij}, & si\ |v_{ij}| \leq V_{max} \\ -V_{max}, & si\ v_{ij} < -V_{max} \\ V_{max}, & si\ v_{ij} > V_{max} \end{cases}$$

Figura 18: Definición de velocidad en función del valor obtenido mediante las ecuaciones

Una vez completado el desplazamiento de los puntos a nuevas posiciones, se vuelve a ejecutar el algoritmo con los nuevos puntos, evaluando de nuevo la función objetivo y deteniendo la exploración en caso de que alguno de los puntos sea minimizador global. La cantidad de iteraciones que se realizarán vienen definidas por un parámetro definido al comienzo de la exploración.

3.5. Diseño e Implementación

De las etapas llevadas a cabo en el plan de trabajo descrito en el apartado 2.2, en esta sección trataremos la implementación completa del algoritmo y el programa que permite su uso, ya que las fases intermedias tuvieron como única finalidad el modularizar el problema y mejorar la comprensión de la solución final más que enfocarlas en obtener resultados representativos.

El proyecto se compila utilizando un fichero *makefile* global que a su vez llama a todos los *makefile* presentes en cada una de las carpetas para compilar todos los módulos. En nuestro caso ha sido necesario crear un *makefile* específico para el módulo *PoExpl* y adaptar al mismo tiempo el del proyecto para que lo referencie.

Podemos destacar tres funciones clave en el desarrollo del algoritmo y afirmar que de ellas depende su efectividad:

1. **Pomex_loop_multi:** función llamada desde el programa principal encargada de la evaluación de la función objetivo.

2. **Pomex_movepoints:** desplazamiento de puntos a nuevas posiciones.

3. **Odec_poincare_fehlberg:** integrador numérico que calcula el siguiente punto de corte de la partícula con la sección de Poincaré.

3.5.1. Tipos de datos

En el momento de la definición de datos, se extrajo del artículo aquellos componentes indispensables que sería necesario codificar en primera instancia y que nos servirían como referencia para elaborar las funciones que describirían su comportamiento.

Atendiendo a la descripción del algoritmo en el artículo tomado como referencia, se consideró que las estructuras donde se almacenaría la información usarían memoria estática (array) dado que el tamaño de la exploración y su tipología estaba fijado previamente a la ejecución y no excederían del tamaño definido.

En la *figura 14*, se muestran los tres tipos de datos principales con los que trabaja el algoritmo y que para nuestro trabajo hemos definido de la siguiente manera:

- Puntos iniciales que se generarán al inicio de la exploración y velocidades asociadas a cada uno de ellos (pDIM=4).

```
typedef struct
{
	double points[pDIM];
	double vel[pDIM];
}PFCpoint;
```

- Mejores posiciones alcanzadas para cada punto de la exploración (pDIM=4).

```
typedef struct
{
	double points[pDIM];
	double distance;
}positions;
```

Ambos tipos de datos creados se almacenaran en sendos vectores con tamaño fijo [20], aunque en las primeras ejecuciones del programa y en la presentación resultados se usarán exploraciones restringidas a 10 puntos.

3.5.2. Librerías heredadas

La propuesta inicial del tutor Juan Carlos, incluía una serie de librerías con las que previamente se habían realizado trabajos por su parte. En ellas, se incluyen diversas funciones matemáticas y físicas que resultan indispensables para poder desarrollar nuestro programa. En algunas de estas funciones fueron necesarias realizar pequeñas mejoras para adaptar su utilidad a nuestro proyecto y cumplir así con los requisitos iniciales.

En la siguiente imagen se resaltan las librerías principales que son usadas en la implementación del algoritmo y que también son usadas por el integrador numérico que se usará en la fase de pruebas.

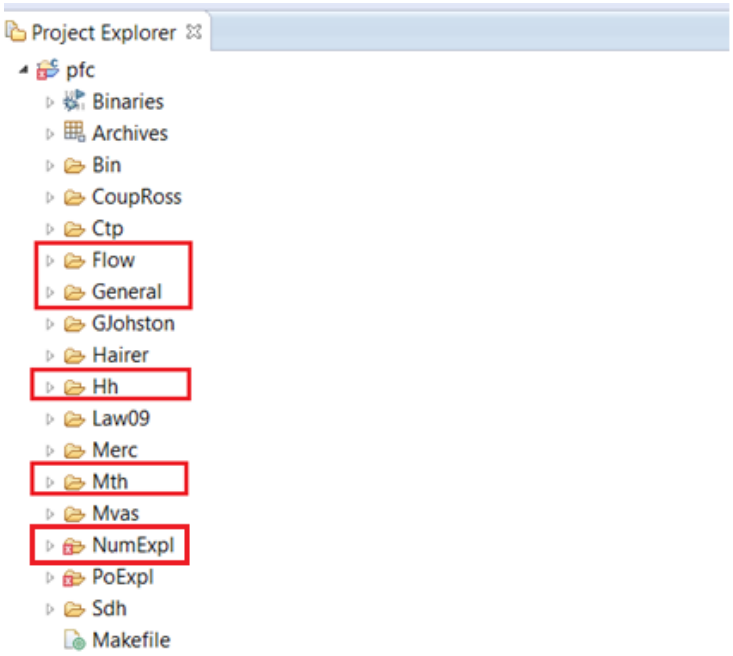


Figura 19: Librerías externas usadas en la implementación

Mth: En este directorio se encuentra la librería matemática en la que se incluyen las funciones necesarias para operar y representar tanto vectores como matrices principalmente. Adicionalmente, también se implementan varias funciones necesarias en el integrador numérico.

Hh: Es la abreviatura de Hénon-Heiles y es el directorio que contiene las funciones de cálculo del hamiltoniano para este sistema así como diferentes parámetros que se usarán durante la implementación del algoritmo.

General: Es el directorio donde se encuentran todas las implementaciones de los integradores matemáticos; en nuestro caso tal y como comentamos, se usará el modelo Runge-Kutta-Fehlberg.

Flow: El directorio Flow es un enlace simbólico al sistema elegido (en nuestro caso Henon-Heiles) por lo que en él se referencian las mismas funciones.

NumExpl: Dentro del directorio NumExpl se incluyen la definición de los tipos datos y la implementación de funciones para el integrador numérico, así como un programa que se usará en la fase de pruebas para obtener la secuencia de cortes de la órbita con una sección de Poincaré durante un tiempo predeterminado.

3.5.3. Librería propia

La librería que incluye todas las funciones desarrolladas y que se han utilizado durante la fase de implementación se encuentran en el directorio PoExpl y se ha denominado *pfc_funcs.h.* En este directorio también se encuentra el código fuente del programa principal, *pomex_main.c* encargado de ejecutar el esqueleto del algoritmo y leer los ficheros de configuración necesarios.

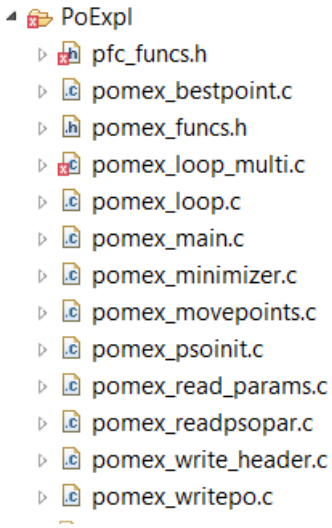


Figura 20: Funciones implementadas para el algoritmo

Pomex_bestpoint: Es la función encargada de buscar el punto con un valor de la función objetivo (*figura 13*) menor y devuelve el índice del array que contiene el mejor punto de la exploración.

- Entrada: vector de mejores posiciones y tamaño de la exploración.
- Salida: índice al array donde se encuentra el mejor punto.

Pomex_loop_multi: Se trata de la función más importante del algoritmo, ya que desde ella se llama al integrador numérico y al cálculo de la función objetivo. Entre los parámetros que se pasan a la función, se encuentra el orden máximo de búsqueda de órbitas periódicas, establecido en 6. Dentro de la función se almacenarán los sucesivos cortes de la órbita con la sección de Poincaré y serán los usados en el cálculo del valor de la función objetivo.

- Entrada: configuración para el integrador, parámetros del algoritmo PSO, vector de mejores posiciones y punto a evaluar.
- Salida: El valor de retorno de la función indica si se ha tenido éxito o no en la búsqueda del minimizador; además, el vector de mejores posiciones se devuelve actualizado tras la evaluación de cada punto.

Pomex_loop: Versión inicial de la implementación, utilizada para las dos primeras etapas de trabajo descritas en la sección 2.2 del presente documento. A partir de la etapa 3, el desarrollo de este módulo se dio por finalizado y se continuó trabajando con el módulo *pomex_loop_multi*, el cual implementa la solución completa.

Pomex_main: Programa principal que implementa el algoritmo y todas las operaciones necesarias especificadas en los requisitos. En el inicio del programa, se leen los ficheros de configuración y se preparan para escritura los ficheros que formarán la salida del programa. Mediante el argumento –o se puede definir el nombre del fichero de salida que se creará con la información y resultado de la ejecución.

Pomex_minimizer: Este módulo implementa la función objetivo representada en la *Figura 13* así como la actualización del vector de mejores posiciones en caso de encontrar un punto mejor. Del mismo modo, el valor de salida de la función indica si se ha encontrado un minimizador global y por tanto es conveniente detener las llamadas del integrador.

- Entrada: punto inicial, punto final, vector de mejores posiciones, mejor punto de toda la exploración y los parámetros específicos del algoritmo.
- Salida: vector de mejores posiciones y mejor punto actualizados y el valor de retorno de la función que indica si se ha encontrado un minimizador global.

Pomex_movepoints: Función encargada de desplazar los puntos a nuevas posiciones en el caso de que no se haya encontrado ningún minimizador para la función objetivo.

- Entrada: vector de puntos y velocidades, índice del mejor punto de la exploración actual, parámetros del integrador y el valor máximo de la velocidad (Vmax).
- Salida: los vectores de posición y velocidad se actualizan con nuevos valores a partir de los anteriores para explorar de nuevo el espacio en busca de un minimizador de la función.

Pomex_psoinit: Este módulo inicializa el vector de puntos y velocidades en función de un punto inicial definido en el fichero de parámetros del algoritmo PSO. Los puntos generados se distribuyen uniformemente alrededor del inicial, considerado como centro de una circunferencia de radio definido en la propia función. La cuarta componente se calcula aplicando la función de la *figura 10* al tratarse de un hamiltoniano en el que la energía es conservativa.

- Entrada: vector de puntos y velocidades, vector de mejores posiciones, parámetros del integrador, del algoritmo PSO y una variable donde se devolverá el tamaño de la exploración.
- Salida: puntos, velocidades, mejores posiciones generados y el tamaño de la exploración.

Pomex_read_params: Función encargada de leer desde fichero los parámetros principales del integrador y de la búsqueda, tales como el orden de búsqueda de órbitas, tamaño del universo y el tiempo de vida del integrador (para detener la integración en caso de no encontrar ningún punto de corte).

- Entrada**:** fichero desde donde se leerán las variables y la estructura donde se almacenarán (definida como ODECParams)
- Salida**:** la estructura con los parámetros leídos de fichero.

Pomex_readpsopar: Lectura de parámetros propios de la implementación del algoritmo PSO tales como origen de la exploración, velocidad máxima de las partículas, condición de igualdad para la función objetivo y el número máximo de iteraciones que se harán desplazando los puntos a nuevas posiciones.

- Entrada: fichero desde donde se leerán las variables y la estructura donde se almacenarán (definida como PFCParams)
- Salida**:** la estructura con los parámetros leídos de fichero.

Pomex_write_header: Esta función escribe en un fichero de trazas solicitado por el tutor la cabecera con la información del integrador que se ha leído con la función *pomex_read_params*. En caso de finalizar con éxito la búsqueda de las condiciones iniciales de la órbita, se guardará una línea con la información del punto y otros aspectos requeridos.

- Entrada: estructura con los parámetros del integrador.
- Salida**:** ninguna (se escribe en un fichero de texto con extensión .po la información de la estructura).

Pomex_writepo: La función genera un fichero de texto con las coordenadas del punto que ha sido considerado como órbita periódica para posteriormente pasárselo al integrador numérico que comprobará si el algoritmo ha funcionado como se esperaba.

- Entrada**:** fichero en el que se escribirá la información, el punto y la dimensión del vector.
- Salida**:** ninguna (se escribe en un fichero de texto con extensión .txt las coordenadas del punto con el formato esperado por el integrador numérico).

3.5.4. Diagrama de flujo del programa principal

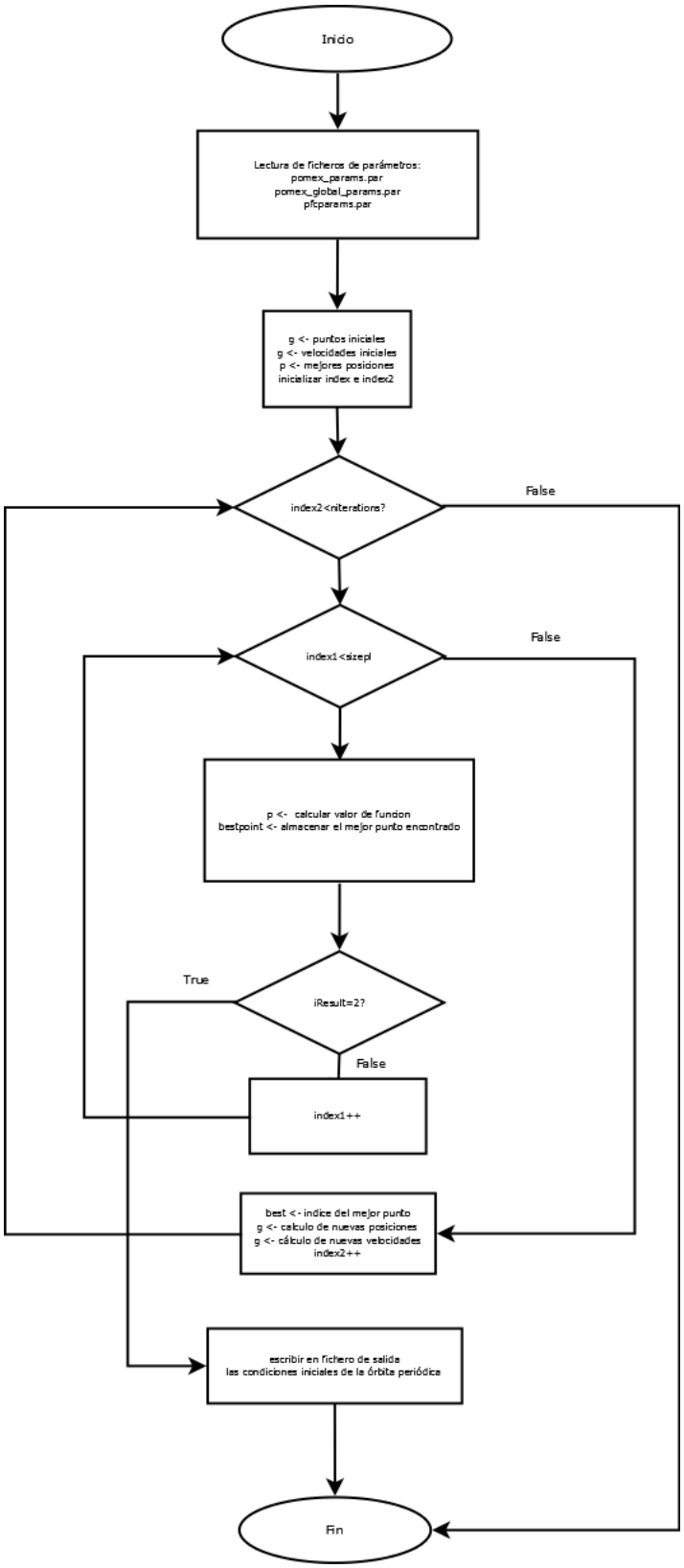


Figura 21: Diagrama de flujo del programa principal

En el diagrama de flujo de la página anterior podemos observar que la parte principal del programa la forman dos bucles encargados de controlar la exploración:

1. El bucle interno realiza llamadas a la función *pomex_loop_multi*, encargada de realizar la integración numérica y calcular el valor de la función objetivo para cada punto calculado.
2. El bucle externo controla las sucesivas exploraciones en caso de no haber encontrado ningún punto con un valor de la función lo suficientemente bajo para considerarlo minimizador. Tras la exploración de todos los puntos del enjambre, se busca el índice del array que indica la mejor posición encontrada (llamando a la función *pomex_bestpoint*) y posteriormente usa esa información para desplazar el enjambre a nuevas posiciones para comenzar de nuevo la exploración (*pomex_movepoints*).

3.6. Otros aspectos de la implementación

Siguiendo los consejos del tutor y cumpliendo con los requisitos de diseño, se mejoró un sistema de trazas ya existente que permite aumentar o disminuir la cantidad y el detalle de los mensajes volcados a uno de los ficheros de salida. El nivel de trazas se puede modificar desde la librería *odec_utils.h* cambiando el valor del parámetro *DEBUGLEVEL* y como referencia, para una ejecución normal se suele establecer a nivel 2 (trazas de información y mensajes importantes de computación de la órbita).

Nivel	Información
0	Sin trazas
1	Errores globales
2	Mensajes del algoritmo PSO y del integrador
3	Información detallada de las funciones matemáticas
4	Nivel de depuración bajo (contenido de algunas matrices y vectores)
5 y 6	Nivel de depuración alto (todas las matrices, funciones y procesos matemáticos son registrados)

Figura 22: Niveles de mensajes proporcionados por el programa

El volcado de la información se realiza al fichero con extensión *.log* y que forma parte de la salida de la ejecución del programa. El nombre del fichero se establece en la llamada al programa, utilizando el argumento ***–o*** y definiendo un nombre específico para los ficheros de salida.

Además del fichero con trazas, se genera otro fichero con el mismo nombre pero extensión *.po* que contiene la configuración del integrador y el resultado en caso de éxito con un formato especificado por el tutor.

Por último, para poder analizar mejor los resultados existe otro tercer fichero a modo de histórico denominado *poout.txt* en el cual se lleva un histórico de órbitas periódicas encontradas en cada ejecución, clasificadas por fecha y hora de la ejecución del programa.

La manera en la que están representados los puntos es compatible con la estructura que espera el integrador numérico que se ejecutará posteriormente para analizar las características de la órbita.

4. Pruebas y evaluación de resultados

En este apartado, se muestran los resultados de distintas ejecuciones del programa así como su posterior evaluación utilizando herramientas auxiliares.

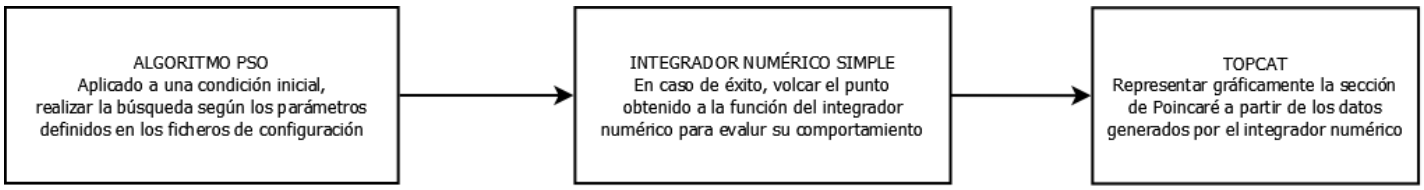


Figura 23: Secuencia seguida para probar el algoritmo

Existe un componente que agrega cierto grado de aleatoriedad en la ejecución del programa y que provoca que difícilmente se repitan dos ejecuciones exactamente iguales. Se trata de la función *srand48*, la cual forma parte de la librería *stdlib.h* y es usada para generar números aleatorios posteriormente con la función *drand48* dentro del rango [0.0 - 1.0]. Estos números se usarán para generar las velocidades de las partículas iniciales y también para generar las dos variables r1 y r2 que serán usadas en el desplazamiento de los puntos a nuevas posiciones (*figura 8*). Para asegurar ese grado de aleatoriedad, el parámetro pasado a la función es la hora del sistema: *srand48(time(null)).*

La configuración del integrador se ha mantenido por defecto, sin modificar ningún parámetro respecto al inicio del proyecto, por lo tanto, el orden de búsqueda de órbitas periódicas será de 6 (definido por el parámetro *iOrderPO* del fichero *pomex_params.par*).

4.1. Configuración del entorno

Al haber trabajado hasta en tres equipos distintos, es necesario comprobar previamente que la variable de entorno *BASEDIR* (definida en el *makefile* principal del proyecto) tiene definida la ruta correcta donde se encuentra el código fuente.

Una vez configurado el directorio base, es preciso crear un enlace simbólico para definir el sistema sobre el que se va a trabajar; como ya comentamos en anteriores apartados, nuestro sistema de trabajo será el de Henon-Héiles y por tanto crearemos un enlace simbólico a dicha carpeta de la siguiente manera:

ln -s Flow Hh

4.2. Ejecución del programa

El fichero que contiene los parámetros del algoritmo PSO se denomina *pfcparams.par* y para esta ejecución contiene los siguientes datos:

x1	-0.0041062588868
x2	0.35370673968447
x3	-0.004367339
equal	0.0000015
maxiterations	20
vmax	0.5

Figura 24: Parámetros del algoritmo PSO

Los tres primeros parámetros, *x1*, *x2* y *x3* definen parcialmente a la partícula inicial de la exploración; el cuarto punto y el resto de condiciones iniciales se calcularán a partir de ellos en la función *pomex_psoinit* descrita anteriormente. A la vista del valor de dicha condición inicial, queda claro que la zona del espacio en la que se buscan las condiciones iniciales de la órbita está próxima al origen de coordenadas.

El parámetro *equal* referencia al minimizador de la función que queremos encontrar, es decir, se analizaran los puntos hasta que alguno tenga un valor de la función menor o igual a este parámetro.

Los otros dos parámetros permiten fijar el número de iteraciones del bucle exterior (*maxiterations*) que desplaza los puntos a nuevas posiciones y la velocidad máxima que puede alcanzar cualquier partícula (*Vmax*), asignándose su valor tal y como se describe en la *figura 18*.

El programa se ejecuta desde una terminal del sistema operativo, accediendo al directorio ***Bin*** del área de trabajo y ejecutando la siguiente sentencia:

./pomex_main –o *fichero*

Como comentamos anteriormente, se generan tres ficheros al terminar la ejecución (dos en caso de no haber encontrado ninguna condición inicial de alguna órbita periódica).

4.2.1. Análisis detallado del resultado

La ejecución realizada con los parámetros definidos anteriormente y con un tamaño de exploración de **10** partículas iniciales arroja los siguientes resultados:

1. El programa ha tenido éxito en la búsqueda de las condiciones iniciales de la órbita periódica (por tanto, ha encontrado un minimizador global de la función que sea igual o mejor al parámetro *equal*).
2. El minimizador ha sido obtenido en el punto número 1 de la 11ª iteración del algoritmo.
3. Las condiciones iniciales describen una órbita periódica de orden 5.

Uno de los indicativos más claros de que el algoritmo realiza su trabajo correctamente es el hecho de que tras cada iteración de los puntos, el mejor minimizador obtenido para la función objetivo es al menos igual o más cercano al valor de *equal*. En la tabla de la siguiente página presentamos la evolución del mejor punto obtenido en cada iteración del algoritmo; dado que el punto inicial con el que hemos iniciado la exploración está en las cercanías de una región con órbitas periódicas, los valores de la función son bajos ya desde la primera iteración:

P	f(x) | 1	f(x) | 2	f(x) | 3	f(x) | 4	f(x) | 5	f(x) | 6	f(x) | 7	f(x) | 8	f(x) | 9	f(x) | 10	f(x) | 11
1	0.00035504	0.00035504	0.00035504	0.00035504	0.00035504	0.00035504	0.00035504	0.00035504	0.00035504	0.00035504	1,92E-07
2	0.00086627	0.00086627	0.00086627	0.00086627	0.00086627	0.00081746	0.00081746	0.00081746	0.00081746	0.00081746	
3	0.00030604	0.00030604	0.00030604	0.00030604	0.00030604	0.00030604	0.00030604	0.00030604	0.00017137	0.00017137	
4	0.00024443	0.00024443	0.00024443	0.00024443	0.00024443	0.00024443	0.00024443	0.00014669	0.00014669	0.00001980	
5	0.00013412	0.00013412	0.00013412	0.00013412	0.00013412	0.00013412	0.00013412	0.00013412	0.00013412	0.00013412	
6	0.00016123	0.00016123	0.00016123	0.00016123	0.00016123	0.00016123	0.00016123	0.00016123	0.00016123	0.00016123	
7	0.00019519	0.00019519	0.00019519	0.00019519	0.00019519	0.00019519	0.00019519	0.00019519	0.00019519	0.00019519	
8	0.00378545	0.00378545	0.00378545	0.00124022	0.00124022	0.00124022	0.00124022	0.00124022	0.00124022	0.00095895	
9	0.00016181	0.00016181	0.00016181	0.00016181	0.00016181	0.00016181	0.00016181	0.00016181	0.00016181	0.00016181	
10	0.00007209	0.00007209	0.00007209	0.00007209	0.00007209	0.00007209	0.00007209	0.00007209	0.00007209	0.00007209	

Figura 25: Evaluación de la función objetivo a lo largo de la ejecución, en azul se representa la mejora de puntos y en verde el minimizador de la función.

El punto obtenido tiene las siguientes coordenadas, que serán volcadas al integrador numérico para analizar el comportamiento de la órbita:

```
x0=  -1.0013214e-07
x1=   0.56204367
x2=  -0.046449619
x3=   0.16084624
```

Una vez obtenido el punto, el siguiente paso es verificar su naturaleza mediante el integrador numérico que se encuentra en el directorio *NumExpl* del área de trabajo. Las coordenadas del punto son incluidas en la función *flow_initpoint* siguiendo el formato de punto que se obtiene en el fichero *poout.txt*.

Tras incluir las condiciones iniciales del punto en la función anterior, se procede a compilar el módulo con esa condición inicial y a su ejecución mediante la siguiente llamada:

./numex_main –o *fichero*

Cuando la ejecución finaliza, se generan tres ficheros de características similares a los generados por el programa que implementa el algoritmo:

- *Fichero.log:* Información de la ejecución, con un sistema de trazas similar al descrito en secciones anteriores para el algoritmo PSO.
- *Fichero.stat:* Información sobre la configuración del integrador numérico
- *Fichero.txt:* En este fichero se guarda la información de los puntos de corte de la órbita con la sección de Poincaré con un formato compatible con el plotter que utilizaremos a continuación para representar gráficamente el resultado.

Dado que la evaluación se realiza sobre 1000 unidades de tiempo, se muestran dos fragmentos de la tabla generada correspondientes a las primeras 100 unidades y a las últimas 100 las cuales son suficientes para analizar la partícula.

Index 0	Index 1	Index 2	Loop	X0	Y0	bOut	Time	Energy	Global Param	Global Param2	Point[0]	Point[1]	Point[2]	Point[3]
-1	-1	-1	1	0.000000	0.000000	0	6.575889	0.112779	0.000000	.000000 -	0.000000	0.385606	-0,048464	0.335771
-1	-1	-1	2	0.000000	0.000000	0	11.715258	0.112779	0.000000	0.000000	-0.000000	-0.414215	-0.028647	0.076056
-1	-1	-1	3	0.000000	0.000000	0	16.694932	0.112779	0.000000	0.000000	-0.000000	0.353640	-0.044143	-0.357817
-1	-1	-1	4	0.000000	0.000000	0	23.263331	0.112779	0.000000	0.000000	-0.000000	0.546025	-0.049969	-0.182887
-1	-1	-1	5	0.000000	0.000000	0	29.109118	0.112779	0.000000	0.000000	-0.000000	0.561596	-0.046167	0.161611
-1	-1	-1	6	0.000000	0.000000	0	35.685217	0.112779	0.000000	0.000000	-0.000000	0.384637	-0.047886	0.336536
-1	-1	-1	7	0.000000	0.000000	0	40.816525	0.112779	0.000000	0.000000	-0.000000	-0.415028	-0.028414	0.069602
-1	-1	-1	8	0.000000	0.000000	0	45.800815	0.112779	0.000000	0.000000	-0.000000	0.355404	-0.043965	-0.356709
-1	-1	-1	9	0.000000	0.000000	0	52.369818	0.112779	0.000000	0.000000	-0.000000	0.546943	-0.049370	-0.181802
-1	-1	-1	10	0.000000	0.000000	0	58.215065	0.112779	0.000000	0.000000	-0.00000	0.560744	-0.046024	0.162945
-1	-1	-1	11	0.000000	0.000000	0	64.791486	0.112779	0.000000	0.000000	-0.00000	0.382955	-0.047267	0.337803
-1	-1	-1	12	0.000000	0.000000	0	69.911619	0.112779	0.000000	0.000000	-0.00000	-0,416133	-0,028216	0.059625
-1	-1	-1	13	0.000000	0.000000	0	74.906251	0.112779	0.000000	0.000000	-0.00000	0.357948	-0.043933	-0.355073
-1	-1	-1	14	0.000000	0.000000	0	81.476068	0.112779	0.000000	0.000000	-0.00000	0.548199	-0.048754	-0.180249
-1	-1	-1	15	0.000000	0.000000	0	87.320673	0.112779	0.000000	0.000000	-0.00000	0.559532	-0.046023	0.164768
-1	-1	-1	16	0.000000	0.000000	0	93.897424	0.112779	0.000000	0.000000	-0.00000	0.380617	-0.046635	0.339520
-1	-1	-1	17	0.000000	0.000000	0	99.003779	0.112779	0.000000	0.000000	-0.00000	-0,41731	2 -0.02805	0.046603
-1	-1	-1	18	0.000000	0.000000	0	104.0102	0.112779	9 0.00000	0 0.00000	-0.00000	0.36110	1 -0.04404	-0,353008

Figura 26: Evolución de intersecciones de la órbita con la sección de Poincaré 0-100 Unidades de Tiempo

Index 0	Index 1	Index 2	Loop	X0	Y0	bOut	Time	Energy	Global Param	Global Param2	Point[0]	Point[1]	Point[2]	Point[3]
-1	-1	-1	155	0.00000	0.00000	0	902,3416	0.112779	0.000000	0.000000	-0.00000	0.559214	-0,046041	0,165237
-1	-1	-1	156	0.00000	0.00000	0	908,9184	0.112779	0.000000	0.000000	-0.00000	0.380005	-0,046493	0.339964
-1	-1	-1	157	0.00000	0.00000	0	914,0198	0.112779	0.000000	0.000000	-0.00000	-0,417564	-0,02803	0.043300
-1	-1	-1	158	0.00000	0.00000	0	919,0294	0.112779	0.000000	0.000000	-0.00000	0.361850	-0,044089	-0,352493
-1	-1	-1	159	0.00000	0.00000	0	925,6003	0.112779	0.000000	0.000000	-0.00000	0.550112	-0,048014	-0,177804
-1	-1	-1	160	0.00000	0.00000	0	931,4442	0.112779	0.000000	0.000000	-0.00000	0.557649	-0,046206	0.167510
-1	-1	-1	161	0.00000	0.00000	0	938,0212	0.112779	0.000000	0.000000	-0.00000	0.376982	-0,04588	0.342132
-1	-1	-1	162	0.00000	0.00000	0	943,1064	0.112779	0.000000	0.000000	-0.00000	-0,418516	-0,027928	0.027436
-1	-1	-1	163	0.00000	0.00000	0	948,1340	0.112779	0.000000	0.000000	-0.00000	0.365513	-0,044364	-0,350065
-1	-1	-1	164	0.00000	0.00000	0	954,7057	0.112779	0.000000	0.000000	-0.00000	0.551885	-0,047453	-0,175472
-1	-1	-1	165	0.00000	0.00000	0	960,5492	0.112779	0.000000	0.000000	-0.00000	0.555887	-0,046496	0.170007
-1	-1	-1	166	0.00000	0.00000	0	967,1261	0.112779	0.000000	0.000000	-0.00000	0.373537	-0,045313	0.344560
-1	-1	-1	167	0.00000	0.00000	0	972,1921	0.112779	0.000000	0.000000	-0.00000	-0,419065	-0,027825	0.010143
-1	-1	-1	168	0.00000	0.00000	0	977,2358	0.112779	0.000000	0.000000	-0.00000	0.369300	-0,044753	-0,347492
-1	-1	-1	169	0.00000	0.00000	0	983,8080	0.112779	0.000000	0.000000	-0.00000	0.553757	-0,046956	-0,17295
-1	-1	-1	170	0.00000	0.00000	0	989,6513	0.112779	0.000000	0.000000	-0.00000	0.554020	-0,046893	0.172592
-1	-1	-1	171	0.00000	0.00000	0	996,2278	0.112779	0.000000	0.000000	-0.00000	0.369826	-0,044815	0.347131
-1	-1	-1	172	0.00000	0.00000	0	1001.276	0.112779	0.000000	0.000000	-0.00000	-0,419102	-0,027872	-0,007681

Figura 27: Evolución de intersecciones de la órbita con la sección de Poincaré entre las unidades de tiempo 900 a 1000

Con las dos tablas representadas anteriormente se puede observar la clara tendencia periódica de la órbita que calculamos con el algoritmo PSO atendiendo a las columnas que representan al punto de corte con la sección de Poincaré (*Point[0]*, *Point[1]*, *Point[2]*, *Point[3]*).

La componente correspondiente a *Point[0]* siempre tiene el valor 0 ya que la sección de Poincaré está definida para x=0; las otras tres componentes toman valores que se repiten cada 5 intersecciones con variaciones razonablemente pequeñas.

Para interpretar gráficamente los resultados, se ha utilizado la aplicación *TopCat* [9], recomendada por los tutores y que permite tratar y representar gráficamente datos con origen en una tabla. De los 4 puntos existentes en la tabla, hemos representado el segundo y el cuarto para poder mostrar la sección de Poincaré.

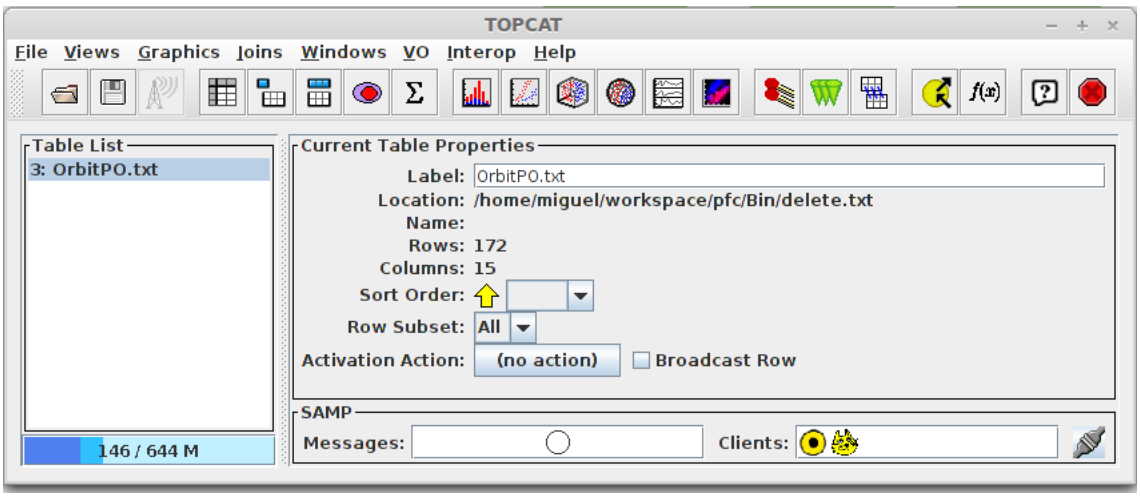
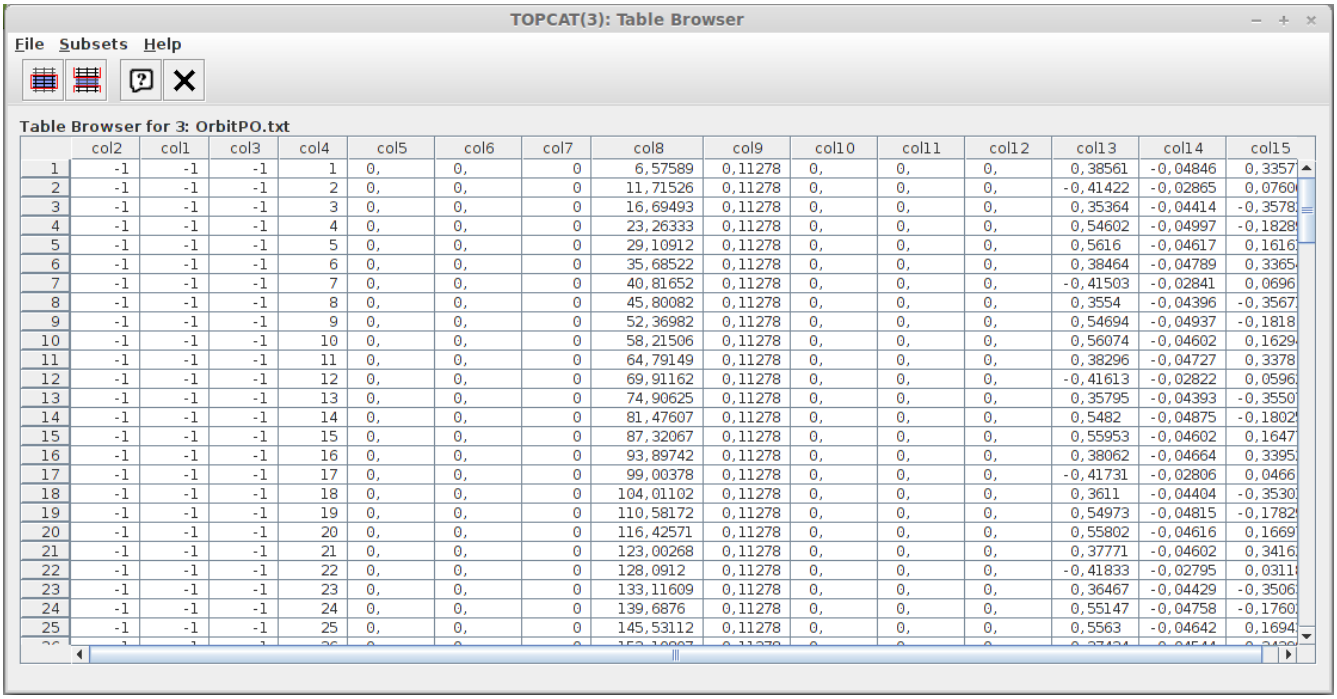


Figura 28: Pantalla principal de la aplicación TopCat tras cargar el fichero generado por el integrador

Al hacer doble click con el ratón sobre el nombre de la tabla (*OrbitPO.txt*) una nueva ventana nos permite explorar el contenido de la misma, tal y como hemos mostrado en las figuras 26 y 27.



	col2	col1	col3	col4	col5	col6	col7	col8	col9	col10	col11	col12	col13	col14	col15
1	-1	-1	-1	1	0,	0,	0	6,57589	0,11278	0,	0,	0,	0,38561	-0,04846	0,3357
2	-1	-1	-1	2	0,	0,	0	11,71526	0,11278	0,	0,	0,	-0,41422	-0,02865	0,0760
3	-1	-1	-1	3	0,	0,	0	16,69493	0,11278	0,	0,	0,	0,35364	-0,04414	-0,3578
4	-1	-1	-1	4	0,	0,	0	23,26333	0,11278	0,	0,	0,	0,54602	-0,04997	-0,1828
5	-1	-1	-1	5	0,	0,	0	29,10912	0,11278	0,	0,	0,	0,5616	-0,04617	0,1616
6	-1	-1	-1	6	0,	0,	0	35,68522	0,11278	0,	0,	0,	0,38464	-0,04789	0,3365
7	-1	-1	-1	7	0,	0,	0	40,81652	0,11278	0,	0,	0,	-0,41503	-0,02841	0,0696
8	-1	-1	-1	8	0,	0,	0	45,80082	0,11278	0,	0,	0,	0,3554	-0,04396	-0,3567
9	-1	-1	-1	9	0,	0,	0	52,36982	0,11278	0,	0,	0,	0,54694	-0,04937	-0,1818
10	-1	-1	-1	10	0,	0,	0	58,21506	0,11278	0,	0,	0,	0,56074	-0,04602	0,1629
11	-1	-1	-1	11	0,	0,	0	64,79149	0,11278	0,	0,	0,	0,38296	-0,04727	0,3378
12	-1	-1	-1	12	0,	0,	0	69,91162	0,11278	0,	0,	0,	-0,41613	-0,02822	0,0596
13	-1	-1	-1	13	0,	0,	0	74,90625	0,11278	0,	0,	0,	0,35795	-0,04393	-0,3550
14	-1	-1	-1	14	0,	0,	0	81,47607	0,11278	0,	0,	0,	0,5482	-0,04875	-0,1802
15	-1	-1	-1	15	0,	0,	0	87,32067	0,11278	0,	0,	0,	0,55953	-0,04602	0,1647
16	-1	-1	-1	16	0,	0,	0	93,89742	0,11278	0,	0,	0,	0,38062	-0,04664	0,3395
17	-1	-1	-1	17	0,	0,	0	99,00378	0,11278	0,	0,	0,	-0,41731	-0,02806	0,0466
18	-1	-1	-1	18	0,	0,	0	104,01102	0,11278	0,	0,	0,	0,3611	-0,04404	-0,3530
19	-1	-1	-1	19	0,	0,	0	110,58172	0,11278	0,	0,	0,	0,54973	-0,04815	-0,1782
20	-1	-1	-1	20	0,	0,	0	116,42571	0,11278	0,	0,	0,	0,55802	-0,04616	0,1669
21	-1	-1	-1	21	0,	0,	0	123,00268	0,11278	0,	0,	0,	0,37771	-0,04602	0,3416
22	-1	-1	-1	22	0,	0,	0	128,0912	0,11278	0,	0,	0,	-0,41833	-0,02795	0,0311
23	-1	-1	-1	23	0,	0,	0	133,11609	0,11278	0,	0,	0,	0,36467	-0,04429	-0,3506
24	-1	-1	-1	24	0,	0,	0	139,6876	0,11278	0,	0,	0,	0,55147	-0,04758	-0,1760
25	-1	-1	-1	25	0,	0,	0	145,53112	0,11278	0,	0,	0,	0,5563	-0,04642	0,1694

Figura 29: Explorador de tablas con la información de la órbita computada

Una vez con los datos cargados, seleccionando la opción "*Scatter plot*" se abre una ventana nueva donde se configuran los parámetros y las columnas que se mostrarán en la gráfica. Para la fase de pruebas, las columnas representadas son la 13 y la 15 que se corresponden con dos de las componentes que describen las condiciones iniciales de la órbita.

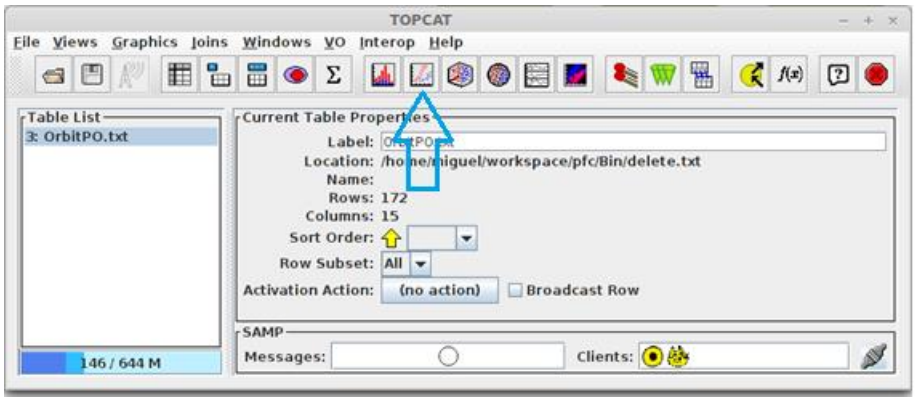


Figura 30: Opción "Scatter plot" de la aplicación TopCat

Ya configurada correctamente las columnas a representar, se observa que la gráfica efectivamente representa 5 agrupaciones de puntos (correspondiéndose con el orden de la órbita periódica calculada con el algoritmo) muy próximos entre ellos.

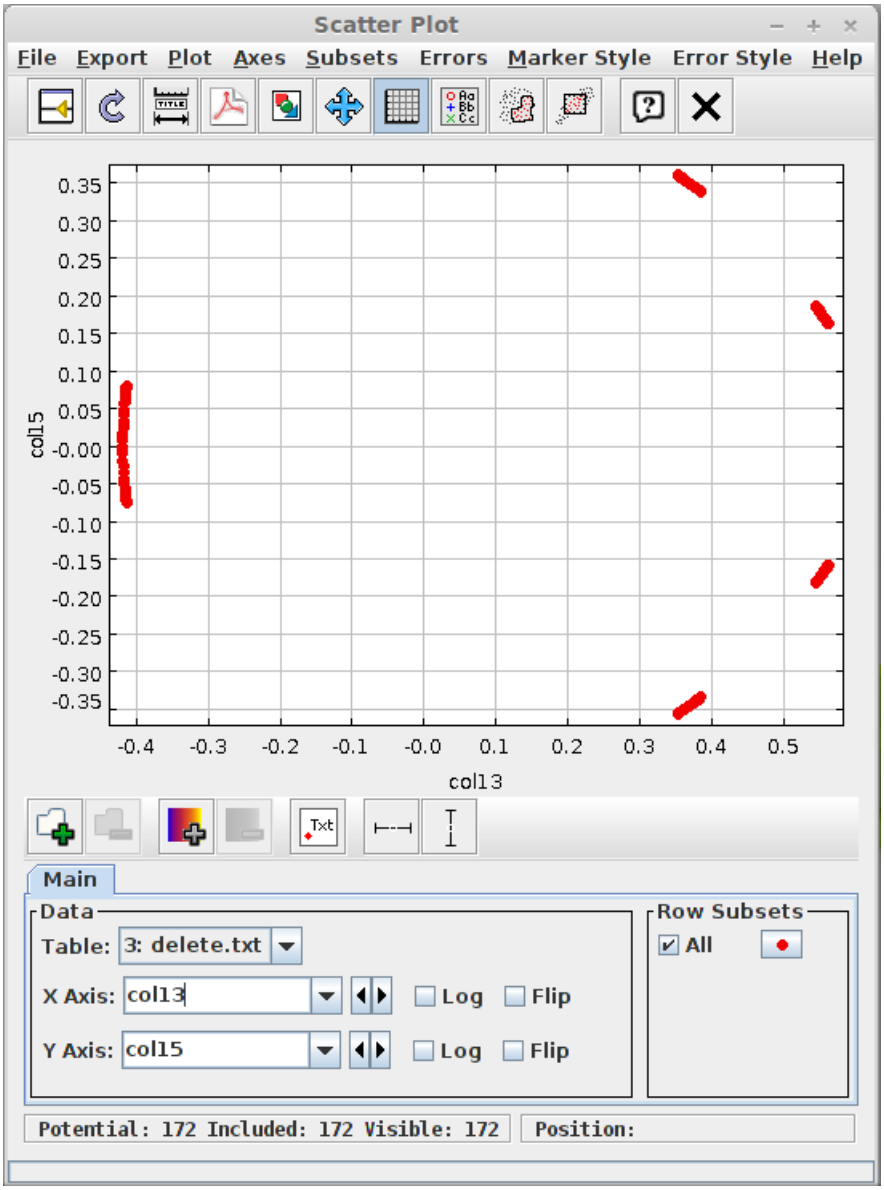


Figura 31: Representación de la sección de Poincaré

Mediante la opción "*Three dimensional scatter plot*" situada justo a la derecha de seleccionada anteriormente, se obtiene una representación en tres dimensiones de los cortes entre la órbita y el plano, aportando una mayor información sobre la misma.

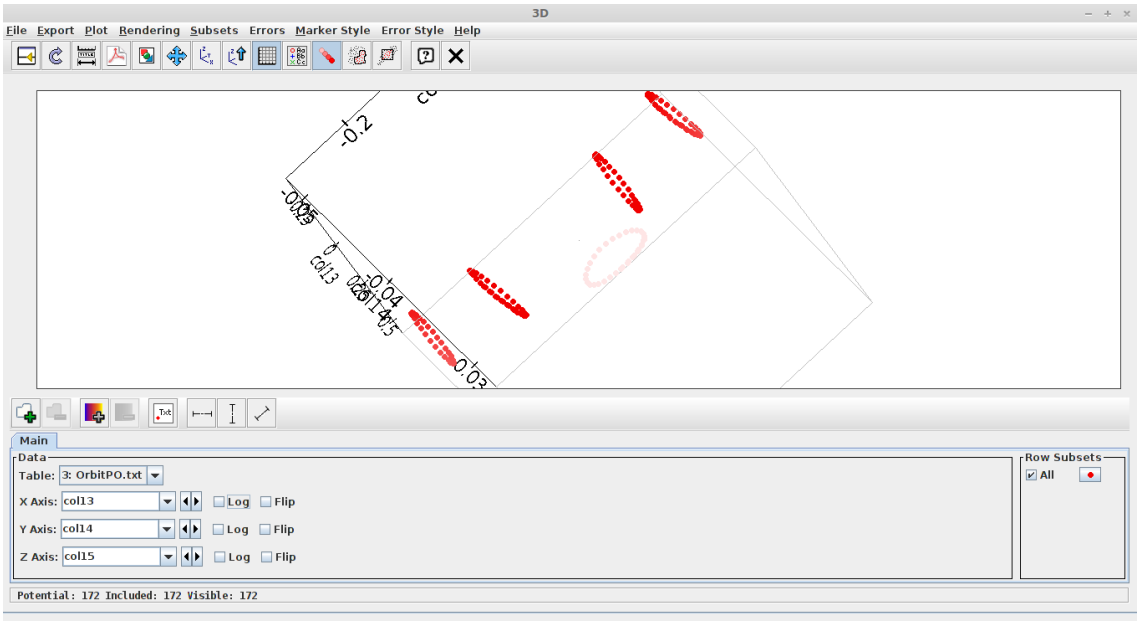


Figura 32: Representación de la sección de Poincaré en 3D

4.2.2. Pruebas adicionales

Además de la explicación extendida del caso práctico anterior, se han realizado sucesivas ejecuciones del algoritmo variando el tamaño de la exploración y cambiando la condición inicial en los parámetros de configuración. En caso de éxito, la mayoría de órbitas obtenidas son de orden 4 o 5, representando secciones muy similares a la descrita anteriormente.

Durante 10 minutos, se realizaron pruebas continuadas de ejecución con unas mismas condiciones iniciales para tratar de establecer una proporción de veces en las que se encontraban las condiciones iniciales de una órbita y las que no, obteniendo los siguientes resultados:

Ejecuciones del algoritmo	**115**
PO encontradas	**98**
PO 1	**9**
PO 2	**0**
PO 3	**5**
PO 4	**44**
PO 5	**40**

Figura 33: Resultados de la ejecución del algoritmo

A la vista de los resultados, podemos cifrar en un 14% los casos en los que nuestro algoritmo no ha sido capaz de identificar las condiciones iniciales de alguna órbita periódica. Este dato hay que considerarlo con cierta prudencia ya que durante la implementación del algoritmo se introdujo una componente aleatoria para generar los puntos que depende de la hora del sistema en la que se ejecuta el programa.

Durante la fase de pruebas se pudieron detectar y corregir varios problemas que afectaban sobre todo al control de la integridad de los puntos y que causaba errores inesperados en la ejecución del programa. El problema más común era obtener como valor en alguna de las componentes de los puntos un valor NaN (Not an Number) por no haber tratado correctamente su valor antes de llamar al integrador numérico.

Cabe destacar entre los resultados obtenidos aquellos identificados como órbitas periódicas de orden 1; las cuales tras su representación gráfica se vieron que forman parte del grupo de las órbitas cuasi-periódicas.

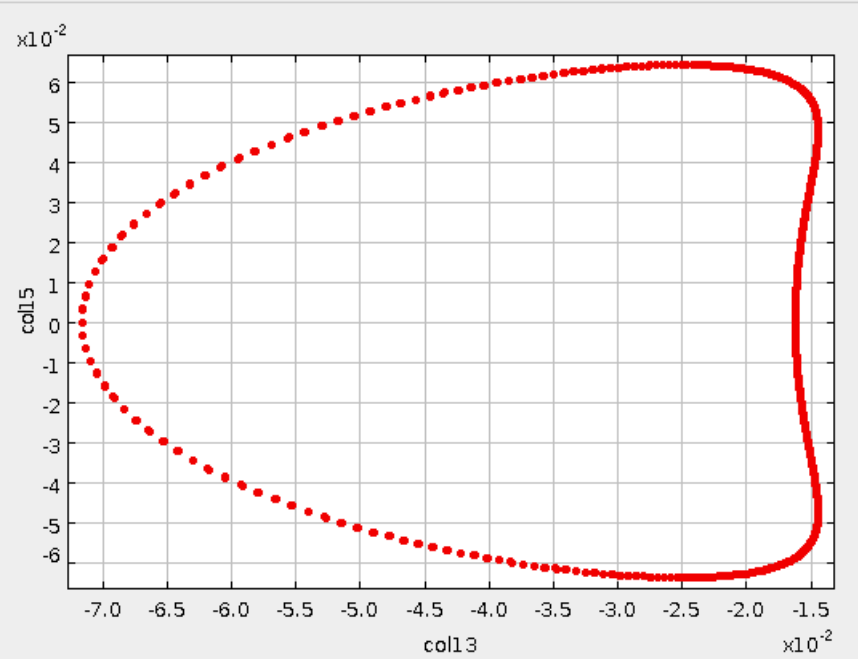


Figura 34: Órbita cuasi-periódica identificada como periódica de orden 1

La obtención de este tipo de órbitas responde a una cuestión de precisión a la hora de definir el umbral para considerar que un punto es minimizador de la función objetivo. Esto provoca que cada punto de corte con la sección de Poincaré sea igual o mejor que el valor establecido por configuración.

En el apartado de conclusiones del trabajo se evaluarán posibles alternativas que permitan optimizar la identificación de órbitas cuasi-periódicas como periódicas.

A continuación se muestra la representación de la sección de Poincaré de algunas de las órbitas encontradas durante la fase de pruebas:

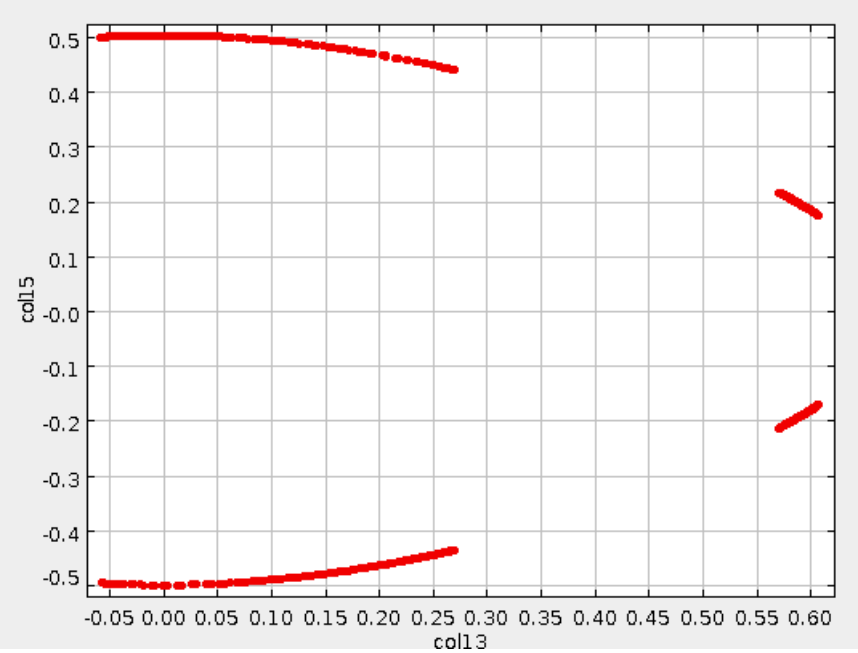


Figura 35: Órbita periódica de orden 4

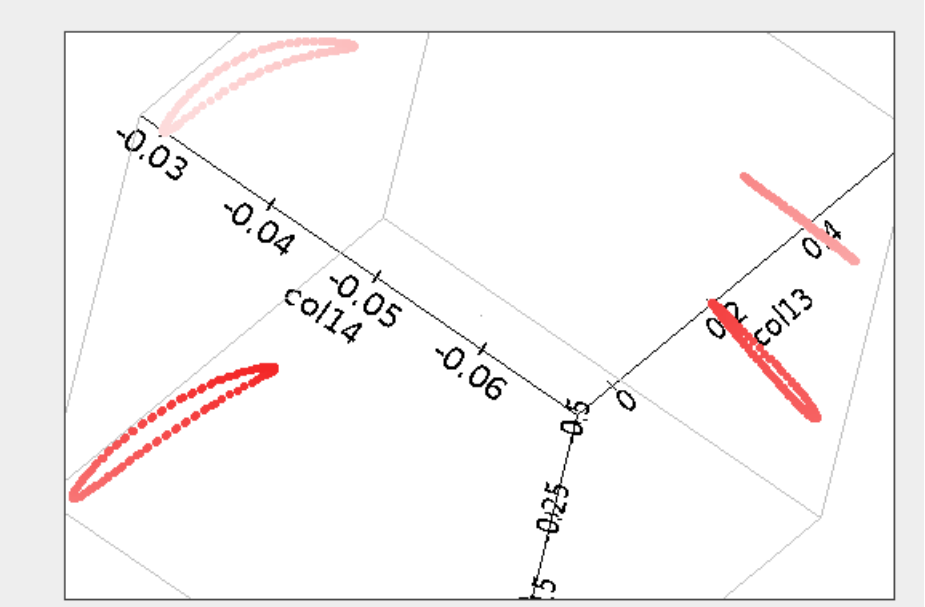


Figura 36: Órbita anterior representada en 3D

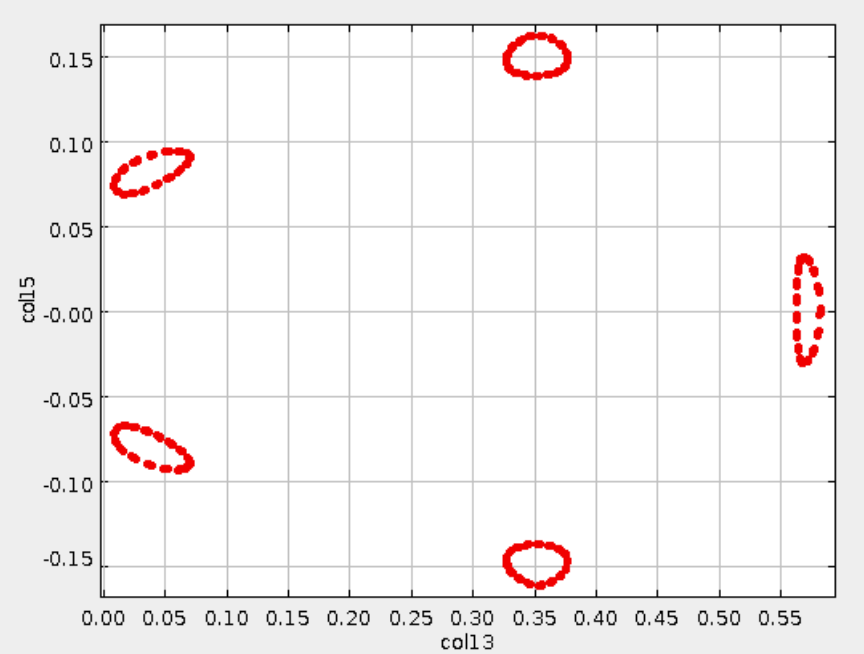


Figura 37: Órbita periódica de orden 5 (Tipo “island chain”)

En este último ejemplo mostrado, se utilizó como condición inicial de la exploración un punto cercano a las condiciones iniciales de una órbita con esa morfología concreta conocida de antemano.

Lo que se pretende demostrar con esta prueba es que el algoritmo resulta eficaz en otras regiones del plano y la convergencia de todos los puntos de la exploración hacia la condición inicial de una órbita periódica se mantiene.

5. Conclusiones y trabajos futuros

Una vez concluida la implementación y la presentación de los resultados obtenidos durante la fase de pruebas del algoritmo, se exponen a continuación las conclusiones obtenidas del trabajo.

Así mismo, se plantean posibles líneas de trabajo futuras que se pueden desarrollar a partir de lo expuesto en este trabajo, y que permitirían profundizar y optimizar la búsqueda de órbitas periódicas en sistemas dinámicos caóticos.

5.1. Conclusiones

El objetivo principal del trabajo consistía en implementar un algoritmo de tipo PSO aplicado a la búsqueda de órbitas periódicas; para ello, se establecieron objetivos secundarios que serán evaluados a continuación:

1. **Estudio de modelos matemáticos y físicos implicados en el proyecto:**

Para una mejor comprensión del algoritmo y su aplicación, fue necesario realizar un estudio previo que abarcase tanto conceptos físicos como matemáticos. Durante la carrera se impartieron dos asignaturas relacionadas con la física, siendo la que se estudia en el primer curso (Fundamentos Físicos de la Informática) la que hacía referencia a algunas ideas elementales que se tratan en el proyecto, principalmente relacionadas con galaxias y ecuaciones de movimiento.

Conceptos tales como la sección de Poincaré, la función del hamiltoniano, el método de integración Runge-Kutta o el propio algoritmo PSO eran absolutamente desconocidos para mí y fue necesario recurrir en primera instancia al conocimiento de los tutores y posteriormente a publicaciones científicas, libros de texto y páginas de internet que permitiesen asentar los conocimientos necesarios.

En paralelo con el estudio conceptual, se analizaron una a una las librerías facilitadas por el tutor Juan Carlos Vallejo para conocer su utilidad y su posible uso en la implementación del algoritmo objetivo.

2. **Desarrollo de las estructuras y funciones necesarias derivadas del análisis de cada apartado del artículo:**

Una vez completado el estudio de todos los temas mencionados anteriormente, se definieron los requisitos del programa y se procedió a su especificación e implementación respetando los ciclos de trabajo fijados en la metodología de trabajo propuesta. De esta manera, a partir de la definición de los tipos de datos a usar, se implementaron las funciones que permitían inicializar los puntos, desplazarlos en caso de no encontrar ninguna condición inicial de órbita periódica, encontrar el mejor punto del enjambre y sobre todo la función que llama al integrador numérico y calcula el valor de la función objetivo.

Durante esta fase, los mensajes generados por la ejecución fueron ampliados y adaptados para evitar que el programa generase más información de la necesaria pero al mismo tiempo cuidando de que cualquier persona que pueda retomar el trabajo pueda seguir de manera clara la ejecución.

3. **Pruebas y aceptación de la aplicación desarrollada:**

Tal y como refleja la tabla de la figura 33, con el algoritmo completamente desarrollado se realizaron más de cien pruebas de ejecución variando el punto inicial de la exploración en las inmediaciones del origen de coordenadas.

Mediante el uso de la herramienta *TopCat*, todas las orbitas generadas han sido representadas en gráficas de 2 y 3 dimensiones; gracias a esta herramienta y usar ambas representaciones, ha permitido observar mejor la naturaleza de las orbitas encontradas y validar el algoritmo implementado.

Por tanto, una vez evaluados los tres objetivos parciales propuestos al inicio del trabajo, podemos concluir que se ha alcanzado de manera satisfactoria el objetivo principal del trabajo.

5.2. Trabajos futuros

El proyecto realizado podría considerarse como una buena base para desarrollar nuevos trabajos de investigación y desarrollo relacionados con la búsqueda de órbitas periódicas en sistemas dinámicos.

Existen dos corrientes principales de avance en las cuales basar esos nuevos trabajos y que se presentan de manera breve a continuación.

5.2.1. Mejoras internas del algoritmo

El algoritmo admite mejoras significativas que podrían aumentar su eficiencia y precisión en la búsqueda de las condiciones iniciales de la órbita.

Uno de los puntos de mejora se encuentra en el método de integración numérica que se usa para calcular las coordenadas de corte de la partícula con la sección de Poincaré. En la implementación descrita en el trabajo se usa el método de Runge-Kutta-Fehlberg (referido como RKF45) de orden 4 y 5 que podría ser optimizado usando la propuesta de Ernst Hairer y que consiste en usar el método de Runge-Kutta de orden 8 (referido como DOP853) para la integración numérica.

Otro punto a considerar para trabajos futuros es adaptar el código para capturar varias condiciones iniciales de órbitas periódicas en la misma ejecución del programa. En el artículo *Particle Swarm Optimization: An efficient method for tracing periodic orbits in 3D galactic potentials* [8], se hace referencia a la técnica de *deflection* que consiste en transformar la función objetivo una vez se encuentra un primer minimizador de ella.

En una línea de trabajo similar, el problema puede adaptarse para implementar la variable local del algoritmo, aquella en la que se definen vecindades entre los puntos y en el desplazamiento de partículas influye la mejor partícula de cada vecindad en lugar de la global. Implementando esta solución permite al usuario poder establecer una comparación fiable sobre el rendimiento y eficiencia de ambos modelos y su conveniencia de uso.

5.2.2. Mejoras externas y de rendimiento

Adicionalmente a las tareas propuestas en el apartado anterior, existe la posibilidad de desarrollar nuevos trabajos que mejoren el rendimiento del programa presentado adaptando su ejecución a las capacidades de las computadoras actuales.

- **Paralelización**: Una posibilidad interesante consiste en adaptar el programa principal y permitir su ejecución en múltiples hilos, pudiendo definir para cada ejecución un conjunto de condiciones iniciales propia y cubrir de esta manera un espacio mucho mayor.
- **Grid**: La computación en grid [5] permite el aprovechamiento coordinado de multitud de recursos para un bien común. En nuestro caso, la mejora consistiría en poder ejecutar el programa desde varias computadoras de tal manera que a medida que vayan concluyendo su ejecución, se comuniquen con un concentrador de datos que evalúe y presente los resultados.

Figura 38: Principio básico de la computación en grid

5.3. Comentario personal

Con estas últimas líneas me gustaría destacar el interés que ha generado en mí el estudio y desarrollo de este proyecto a lo largo del tiempo. Cada nueva etapa desarrollada en el trabajo suponía adquirir nuevos conocimientos tanto teóricos como prácticos.

Tanto es así, que en la medida en que la carrera profesional lo permita, estaría dispuesto a continuar con el trabajo e implementar alguna de las soluciones propuestas, en especial la relacionada con la computación en grid ya que dadas sus características, tal vez fuese posible reutilizar ese conocimiento en el ámbito laboral.

6. Bibliografía

[1] [Beibei,Zhuangkuo,2011],*"Research on Particle Swarm Optimization for Vehicle Routing Problem",* The Contemporary Logistics Journal, http://www.seiofbluemountain.com/en/Journals/info_content/down/3_201103/1321583669356.pdf , Mayo 2013.

[2] [D. Bratton and J. Kennedy, 2007], "*Defining a standard for particle swarm optimization*," in Swarm Intelligence Symposium, 2007. SIS 2007. IEEE,.

[3] [Eclipse13], "*Eclipse Project, Eclipse Juno 3.8"*, http://www.eclipse.org , Enero 2013.

[4] [F.Diacu,1996],*"The solution of the n-body Problem, The Mathematical Intelligencer"*,1996.

[5] [Grid13], *"Grid Computing Info Centre",* Mayo 2013. http://www.gridcomputing.com

[6] [J. Kennedy and R. Eberhart,2001],*"Swarm Intelligence"*. Morgan Kaufmann Publishers, Inc., 2001.

[7] [M. Clerc and J. Kennedy, 2002], "*The particle swarm - explosion, stability, and convergence in a multidimensional complex space*," Evolutionary Computation, IEEE Transactions on, vol. 6, no. 1.

[8] [Skokos Ch., Parsopoulos K. E., Patsis P. A. & Vrahatis M. N., 2005], *Particle Swarm Optimization: An efficient method for tracing periodic orbits in 3D galactic potentials*, Monthly News of Royal Astronomical Society (MNRAS),2005.

[9] [TopCat13], "*Tool for Operations on Catalogues and Tables",* http://star.bris.ac.uk/~mbt/topcat, Abril 2013.


UNIVERSIDAD REY JUAN CARLOS
MCMXCVI