UNIVERSIDAD
REY JUAN CARLOS

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Curso Académico 2010/2011

Proyecto de Fin de Carrera

SIMULACIÓN MEDIANTE APPLETS DE JAVA DE SISTEMAS DINÁMICOS:

DINÁMICA Y CONTROL DEL OSCILADOR DE HELMHOLTZ

Autor: IGNACIO GARCÍA MÁRQUEZ

Tutores: Inés Pérez Mariño
Jesús M. Seoane Sepúlveda

Resumen

Este documento contiene la memoria del Proyecto de Fin de Carrera, el cual permite simular el comportamiento físico de un sistema dinámico, y en concreto el oscilador de Helmholtz, que es un sistema físico con aplicaciones en Ciencia e Ingeniería.

Se trata de un proyecto desarrollado en el Departamento de Física de la Universidad Rey Juan Carlos para la carrera de Ingeniería Técnica en Informática de Gestión, por lo cual el objetivo es desarrollar una aplicación informática que nos permita estudiar y trabajar más fácilmente con el sistema físico en cuestión, así como la presentación de sus resultados.

Este sistema se implementa mediante un applet de JAVA y permite simular la representación y el comportamiento del sistema dinámico del oscilador de Helmholtz a partir de unos valores de entrada y las funciones de cálculo correspondientes. Dicha representación se puede mostrar de manera gráfica o mediante una animación.

La aplicación se desarrolla mediante un applet de JAVA, que es uno de los lenguajes de programación orientada a objetos más potentes y usados en la actualidad. Para la implementación de esta aplicación con dicho lenguaje de programación, se ha optado por realizarlo con la ayuda del entorno de desarrollo integrado (IDE) de NetBeans, que es un proyecto de código abierto creado por la empresa Sun MicroSystems, que hace más cómoda la programación de aplicaciones en JAVA al permitir desarrollarlas mediante módulos.
Las principales ventajas presentadas por dicho sistema de representación radican en que los applets se pueden ejecutar en cualquier sistema operativo (Unix, Mac OS, Windows, etc.) y mediante cualquiera de los navegadores web disponibles (Mozilla Firefox, Chrome, Safari, Opera, Windows Explorer, etc.), factores que dotan a la aplicación de una gran portabilidad.
Por ello se ha decidido integrar dicho applet en una página Web para obtener el producto final.

Esto se traduce en que la aplicación posea una gran portabilidad y usabilidad y, de este modo, que puedan acceder a ella un gran número de usuarios. Dado el carácter de portabilidad de los applets, cualquier usuario podría ejecutar la aplicación en cualquier ordenador, ya sea en universidades, colegios, institutos, o en cualquier otro ordenador personal. El applet podría ser útil para la docencia, pues muestra de forma clara el comportamiento del sistema dinámico del oscilador de Helmholtz facilitando así su comprensión. El usuario podrá comprobar de una forma visual y rápida la representación deseada.

Índice

1. Introducción

Las oscilaciones son muy abundantes en la naturaleza, y por lo tanto el movimiento oscilatorio constituye un concepto fundamental y básico de la física en general. La idea intuitiva de oscilación es la evolución en el tiempo de un sistema caracterizado por un movimiento de vaivén, ya sea este regular o no. La evolución en el tiempo de estos sistemas puede ir desde un comportamiento totalmente periódico a uno totalmente irregular, en el que las oscilaciones nunca se repiten, denominado caótico. Esto puede dar una idea de que los comportamientos oscilantes se encuentran por todas las partes en la naturaleza, y en consecuencia su estudio ha sido y sigue siendo indispensable en física. Para estudiar cómo el comportamiento de un oscilador varía en función de los distintos tipos de fuerzas que pueden actuar sobre él, se pueden usar nociones elementales de la teoría de los sistemas dinámicos.

Un sistema dinámico es un tipo de sistema que presenta un cambio o una evolución de su estado con el paso del tiempo. En general, los sistemas dinámicos pueden clasificarse en lineales **[1]** y no lineales dependiendo de la respuesta que presenten ante un estímulo externo. En los sistemas lineales el efecto producido es proporcional a la causa y en los no lineales no, por lo cual tienen un comportamiento impredecible a largo alcance.

En primer lugar, nos encontramos de menor a mayor complejidad con el oscilador lineal libre en el que la elongación es directamente proporcional a la fuerza recuperadora Este movimiento es conocido como armónico simple. Se trata de un caso ideal, cualquier caso real llevaría consigo que todo oscilador acabe deteniéndose debido a la existencia de fuerzas de rozamiento o de fricción que se oponen a su movimiento. Por lo tanto la existencia de rozamiento hará que las oscilaciones cesen, a no ser que haya un aporte externo de energía en cada oscilación que compense las pérdidas pro rozamiento. Supongamos que una fuerza externa periódica de amplitud F_0 y frecuencia ω_f actúa sobre el oscilador. Esto significa que por cada periodo de la oscilación se introduce un aporte de energía al oscilador que produce como efecto un mantenimiento de las oscilaciones del sistema, incluso en presencia de amortiguamiento. En particular, si el oscilador es lineal presentará un comportamiento periódico característico que dependerá de la frecuencia de la fuerza externa oscilante.

El grado de complejidad en un oscilador proviene de la no linealidad de la fuerza que actúa sobre él. En general, todo oscilador se comportará como un oscilador no lineal si se aleja lo suficiente de la posición de equilibrio. De este modo se introduce el estudio de los sistemas no lineales. Mientras que en los osciladores lineales la presencia de rozamiento y de una fuerza periódica externa únicamente puede producir una respuesta periódica, en un oscilador no lineal la respuesta puede llegar a ser caótica para cierto rango de valores de algún parámetro característico que modifiquemos.

Para representar las soluciones de un sistema en general disponemos de la conocida como representación “espacio-tiempo”, que consiste en dibujar la coordenada desplazamiento

a lo largo del eje de las ordenadas y el tempo en el eje de abscisas. Otra representación consistiría en mostrar en un espacio bidimensional la velocidad en función de la posición. Esta representación, introducida en Física a comienzos del siglo XX por el físico americano Josiah Willard Gibas, se conoce como espacio de fases y nos permite representar el estado del sistema en cada instante. Así, si para caracterizar completamente un sistema hemos de dar sus coordenadas de posición y velocidad en el tiempo, en un espacio bidimensional este par de coordenadas irán describiendo una trayectoria que nos da idea de cómo evoluciona el sistema en el tiempo. Por tanto, una trayectoria cerrada nos indicaría que el sistema evoluciona de forma periódica.

Como se ha explicado anteriormente, los sistemas dinámicos son aquellos sistemas complejos en los que su comportamiento o propiedades varían en función del tiempo. Un ejemplo de dicho tipo de sistemas es el sistema dinámico del oscilador de Helmholtz.

Además de todos los aspectos mencionados anteriormente, también se debe estudiar la forma de evitar fugas en los sistemas dinámicos en presencia de disipación y otras fuerzas, como ocurre en situaciones físicas realistas. Utilizaremos como modelo prototipo el oscilador de Helmholtz, que es el más simple oscilador no lineal con fugas. Para algunos valores de los parámetros de entrada, este oscilador presenta un valor crítico del forzamiento de todas las partículas, que escapan de su trayectoria normal. Usando la técnica de control de la fase, cambiando suavemente la forma del potencial a través de una perturbación periódica de la fase adecuada, se evitan las fugas en las diferentes regiones del espacio de fases. Este método puede ser útil para evitar fugas en las situaciones físicas más complicadas.

Los sistemas dinámicos libres son típicos en la naturaleza. En un sistema dinámico libre, hay una región del espacio de fases donde casi todas las trayectorias divergen de forma asintótica a infinito. Estas trayectorias han atraído una gran atención en el contexto de caos transitorio y, en particular, en los problemas de dispersión caótica, entre otros. El carácter generalizado de este tipo de sistemas dinámicos sugiere que hay situaciones en que podríamos estar interesados en evitar estas divergencias hasta el infinito, es decir, evitar escapes. Con el fin de definir lo que es un escape podemos imaginar el siguiente escenario. Supongamos que una partícula está bajo la influencia de algún objeto o potencial enorme. Bajo esta situación, decimos que un sistema dinámico tiene un escape siempre que esta partícula cruza un límite determinado y nunca regresa. También, desde un punto de vista práctico, será de gran interés poder controlar este tipo de comportamiento.

Desde el trabajo pionero del control del caos, debido a Ott, Grebogi, y Yorke (OGY) **[2]**, se han propuesto diferentes esquemas de control que permiten obtener una respuesta deseada de un sistema dinámico mediante la aplicación de algunas pequeñas pero precisas perturbaciones seleccionadas. En este contexto, se han propuesto algunas técnicas que permiten evitar los escapes en sistemas dinámicos libres que presentan caos transitorio, con aplicaciones a muchas situaciones diferentes en la física y la ingeniería.

Los métodos indicados para controlar el caos se pueden clasificar dependiendo de cómo interactúan con el sistema. Uno de los métodos de control del caos consiste en estabilizar las órbitas inestables utilizando pequeñas perturbaciones dependientes del estado del sistema. Sin embargo, en implementaciones experimentales, la respuesta rápida que estos métodos requieren no puede ser proporcionada por lo general. Para estas situaciones, existe un método más útil utilizado para suprimir el caos en sistemas dinámicos impulsados periódicamente. Entre ellos una amplia clase está representada por los osciladores no lineales cuya ecuación clásica general es:

$$\ddot{x} + \mu\dot{x} + \frac{dV}{dx} = F\cos(\omega t)$$

donde μ es el coeficiente de amortiguamiento, $V(x)$ es la función potencial responsable de la fuerza restauradora que actúa sobre el sistema, y $F\cos(\omega t)$ es un forzamiento externo periódico. Obviamente, en función del potencial $V(x)$ tenemos diferentes tipos de osciladores.

La idea principal de estos métodos es aplicar una perturbación armónica, ya sea a algunos de los parámetros del sistema o como un forzamiento adicional, siendo su eficacia demostrada numérica y experimentalmente en diferentes obras. Se observó que la diferencia de fase Φ entre el forzamiento periódico y la perturbación tenía cierta influencia en el comportamiento dinámico del sistema. Por otra parte, los autores han demostrado que Φ juega un papel crucial en la dinámica global del sistema. Esta técnica de control, donde Φ actúa como un parámetro de control fijo, se llama "control de caos por la fase". El objetivo es mostrar que el método de control de fase se puede aplicar para evitar escapes en sistemas dinámicos libres. El oscilador de Helmholtz se usa como paradigma de este tipo de sistemas, y demuestra que el método de control por la fase es también una poderosa herramienta para controlar y evitar fugas.

A continuación se presenta el oscilador de Helmholtz, que es un ejemplo paradigmático de un sistema dinámico con fugas, y se da una descripción de la implementación del control por la fase **[3]**. Esta es la forma más sencilla para modelar fenómenos físicos que presentan la capacidad de escapar de un pozo de potencial. Este oscilador no lineal describe el movimiento de una unidad de partícula de masa en un potencial cúbico $V(x)=ax^2/2+bx^3/3$, la cual puede ser eventualmente perturbada externamente por un movimiento sinusoidal. Al añadir una fuerza disipativa lineal, la ecuación de movimiento es:

$$\ddot{x} + \mu\dot{x} + ax + bx^2 = F\cos(\omega t)$$

donde μ representa el coeficiente de amortiguamiento, F la amplitud de la fuerza externa, ω la frecuencia de dicha fuerza, y a y b determinan la forma del potencial.
Fijamos los parámetros $\mu=0.1$, $\omega=1$, y $a=b=-1$, por lo cual el potencial es:

$$V(x) = -\frac{x^2}{2} - \frac{x^3}{3}$$

Este potencial tiene un máximo en $x=0$ y un mínimo en $x=-1$ como se muestra en la *Figura 1*.

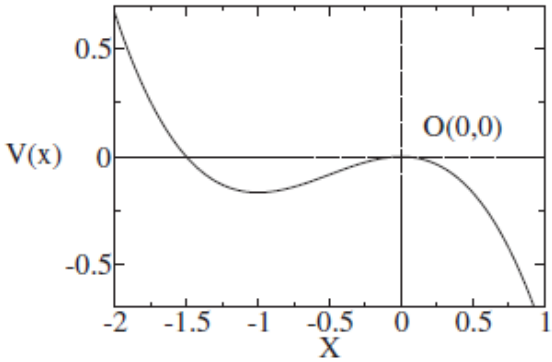

Figura 1: *Gráfica del potencial cúbico* $V(x) = -\frac{x^2}{2} - \frac{x^3}{3}$.

Para esta elección de valores de los parámetros la ecuación de movimiento es:

$$\ddot{x} + 0.1\dot{x} - x - x^2 = F\cos(t)$$

Este sistema presenta comportamientos diferentes en función del valor del único parámetro libre, que es la amplitud *F*. Por ejemplo, en la *Figura 2* podemos ver una gráfica de las cuencas de atracción de este sistema para *F=0,12(a)* y *F=0.21 (b).*

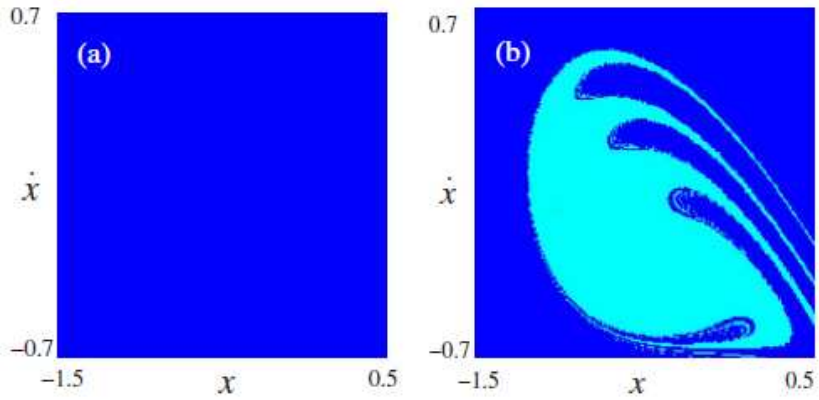

Figura 2: *a) Cuenca de atracción del oscilador de Helmholtz* $\ddot{x} + 0.1\dot{x} - x - x^2 = 0.21\,cos(t)$. *Los puntos azules indican el conjunto de puntos que se escapan del pozo de potencial, y los puntos cian indican los puntos que corresponden a los atractores. Hay que tener en cuenta que todas las condiciones iniciales escapan después de un periodo de tiempo.*

b) Cuenca de atracción del oscilador de Helmholtz $\ddot{x} + 0.1\dot{x} - x - x^2 = 0.12\,cos(t)$. *Aquí las cuencas del espacio de fases tienen una estructura fractal, donde los puntos cian denotan el conjunto de puntos que caen en los atractores.*

La imagen de la *Figura 2(b)* corresponde a la cuenca de atracción para *F=0.12*. Esta representación corresponde a una órbita limitada donde coexisten puntos dentro del pozo de potencial y puntos que se escapan. Sin embargo, la cuenca de atracción de la *Figura 2(a)* muestra que para este valor de la fuerza todas las trayectorias escapan y por lo tanto, la trayectoria resultante se aleja hasta el infinito. En general, un punto se escapa de la trayectoria del pozo cuando cruza con velocidad positiva el máximo del potencial, situado en *x=0*, como vimos en la *Figura 1*, y nunca regresa. El tiempo empleado por una partícula determinada que

se encuentra en el interior del pozo hasta que se cruza con el máximo del potencial se llama tiempo de escape *T*.

En la *Figura 3* podemos observar la representación gráfica de las representaciones indicadas anteriormente.

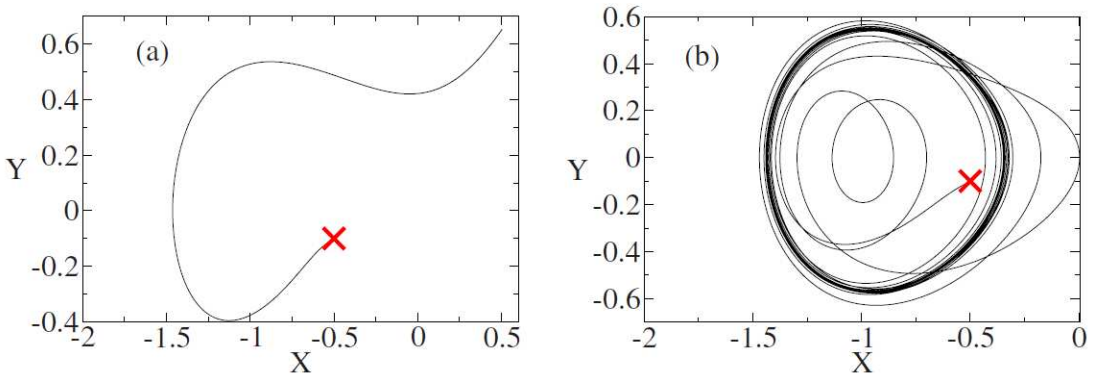

Figura 3: *(a) Trayectoria sencilla para el oscilador de Helmholtz* $\ddot{x} + 0.1\dot{x} - x - x^2 = 0.21\,cos(t)$, *con condición inicial en el punto* (x_0,$\dot{x}_0$)= (-0, 5,-0,1). *La partícula escapa después de un tiempo determinado. (b)Trayectoria sencilla para el oscilador de Helmholtz con control* $\ddot{x} + 0.1\dot{x} - x - [1 + 0.05\,cos(t + \pi)]x^2 = 0.21cos(t)$. *La perturbación mantiene la partícula en la cuenca para siempre.*

Las cuencas de atracción para *F=0.21* mostradas en la *Figura 2(a) y Figura 3(a)* es la que queremos controlar, y este es el valor de la fuerza *F* que consideraremos en el resto de la introducción. Nuestro objetivo principal es evitar el escape para el mayor número de condiciones iniciales mediante el método de control por la fase.

Hay diferentes maneras de implementar el control por la fase, ya sea mediante la adición de una segunda fuerza periódica o perturbando armónicamente uno de los parámetros del sistema. Después de esto, sólo tenemos que variar la fase Φ en busca del comportamiento dinámico deseado. Ahora presentamos algunos argumentos con el fin de comprender mejor lo que es la mejor estrategia en nuestro caso. Supongamos que tenemos partículas oscilando en un potencial y una fuerza externa está actuando. Estas partículas pueden permanecer en el interior del pozo o escapar de ella en función de la amplitud de forzamiento. Una forma adecuada para modificar el comportamiento dinámico de las partículas en el interior del pozo es mediante la modificación de la forma del pozo de potencial. Siguiendo este razonamiento, introducimos un parámetro de perturbación en el término cuadrático de la ecuación de movimiento:

$$\ddot{x} + 0.1\dot{x} - x - [1 + \varepsilon\cos(t + \Phi)]x^2 = F\cos(\omega t)$$

donde ε es la modulación de amplitud y Φ es la diferencia de fase con el forzamiento.

Tengamos en cuenta que estamos utilizando para la perturbación paramétrica una frecuencia resonante con la amplitud de forzamiento, lo cual es un supuesto común para este tipo de métodos de control.

2. Objetivos y metodología

A continuación se enumeran los objetivos del Proyecto Fin de Carrera:

1. En primer lugar deberemos adquirir unos conocimientos básicos de física acerca del sistema que queremos simular. Para ello deberemos de conocer las funciones que usaremos a lo largo del Proyecto y las gráficas que vamos a representar, así como la explicación final del resultado gráfico obtenido dependiendo de los diferentes parámetros de entrada.

2. También deberemos adquirir los conceptos básicos de la programación orientada a objetos **[4]** y conocer el lenguaje de programación de JAVA, ya que esta será la metodología y lenguaje empleados en nuestro Proyecto. En concreto nos centraremos en el estudio de los applets de JAVA y el entorno donde desarrollaremos su implementación será en NetBeans. Por ello deberemos de aprender todos estos conceptos, su funcionalidad y su forma de uso para alcanzar los objetivos buscados.

3. El principal objetivo es la implementación mediante un applet de JAVA del sistema dinámico del oscilador de Helmholtz, de manera que el usuario pueda estudiar su comportamiento en diferentes situaciones. Dicha simulación se llevará a cabo mediante una aplicación informática consistente en un applet con una interfaz sencilla, completa e intuitiva, para que cualquier usuario no informático y con unos conocimientos básicos en física pueda usar y estudiar la representación de una forma fácil y cómoda a partir de unos valores de entrada proporcionados por el mismo.

4. La implementación buscada debe funcionar correctamente y ofrecer la posibilidad de obtener diferentes tipos de representaciones y características respectivas para cada una. Las principales opciones de representación serán las siguientes:
   - Representación gráfica o mediante una animación.
   - Representación de una o dos órbitas.
   - Representaciones sin o con variables de control de escape.
   - Representación estática o dinámica.
   - Representación con o sin ejes de coordenadas.
   - Modificación del color de representación de la/s gráficas.

   Además, se podrá iniciar, pausar o detener la representación en cualquier momento. También podremos cortar o borrar la representación en cualquier momento.

5. Finalmente se deberá generar una memoria que contenga toda la información correspondiente al Proyecto. Este documento deberá contener tanto la información previa del sistema a implementar como los pasos seguidos para lograr el objetivo. Además deberá explicarse el objetivo del applet obtenido junto con la descripción de todas sus partes y el manual de usuario para su utilización. Finalmente se hará referencia a las posibles aplicaciones o usos futuros del applet.

2.1. Descripción del problema

El sistema dinámico del oscilador de Helmholtz se describe mediante diferentes ecuaciones dependiendo de si le insertamos o no los parámetros de control:

- Sin parámetros de control

$$\ddot{x} + \mu\dot{x} + ax + bx^2 = F\cos(\omega t)$$

- Con parámetros de control

$$\ddot{x} + \mu\dot{x} + ax + bx^2 S(t) = F\cos(\omega t)$$

donde $S(t) = [1 + \varepsilon\cos(t + \Phi)]$

Con respecto a las ecuaciones anteriormente mostradas:

- *x* representa la variable de espacio,
- $\dot{x}$ representa la derivada primera de x con respecto al tiempo (se corresponde con la variable y)
- $\ddot{x}$ representa la derivada segunda de x con respecto al tiempo (se corresponde con la derivada de y),
- *S(t)* representa la ecuación de control del sistema,
- *a, b, μ, F, ω, t, ε, Φ* representan los parámetros insertados por el usuario.

La técnica de control utilizada para este sistema en concreto es la de control por la fase, en la cual se añade un término de control obtenido mediante diferentes parámetros insertados por el usuario.

Por lo tanto, al insertar los parámetros o variables de control en la ecuación normal, el resultado de la misma estará formado por valores dentro de un rango normal de representación y se evitará que los resultados calculados tiendan a infinito.

En la representación gráfica se mostrará la gráfica resultante al dibujar todos los puntos calculados a lo largo de la ejecución del programa. En el caso de la animación mediante el dibujo, se mostrará el movimiento de oscilación o rotación provocado por los puntos calculados sobre un elemento. En concreto, el barco oscilará, flotará o se hundirá dependiendo de los puntos calculados y el rumbo que va tomando la representación gráfica.

2.2. Estudio de alternativas

En este apartado se muestran los diferentes métodos estudiados para lograr entender el problema físico que queremos mostrar en el applet, es decir, los pasos previos a la obtención de las ecuaciones que implementaremos en el programa para la resolución de los sistemas dinámicos del oscilador de Helmholtz.

Los movimientos que presentan comportamientos caóticos se expresan mediante ecuaciones diferenciales. El problema de las ecuaciones diferenciales es que en algunas de las ocasiones no es posible encontrar una solución de forma analítica debido a la complejidad de estas. Para poder conseguir una solución, aunque aproximada, se pueden utilizar varios métodos de integración numéricas **[5]**. Entre ellos los que explicaremos a continuación:

Método de Euler

El método de Euler **[6]** de primer orden es un método para la resolución de ecuaciones diferenciales ordinarias y las soluciones que proporciona son aproximaciones numéricas (*Figura 4*). La idea de dicho método está basada en el significado geométrico de la pendiente de una función en un punto inicial dado para, posteriormente, poder hallar el punto siguiente. Dicha ecuación sería la siguiente:

$$y_{n+1} = y_n + hf(x_n, y_n) + O(h^2)$$

donde *h* es el tamaño del paso temporal.

Este método es más preciso cuanto menor sea el tamaño de paso o *h*, y presenta un margen de error elevado respecto al verdadero valor de las funciones. No es muy recomendable el uso de este tipo de método para realizar aproximaciones ya que no es muy preciso en comparación con otros métodos con el mismo valor de *h*, ni tampoco es muy estable.

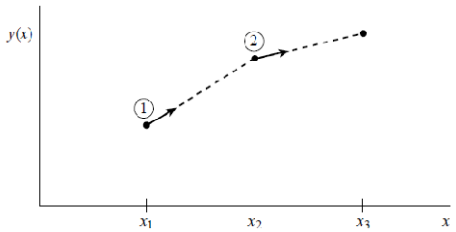

Figura 4: *Método de Euler: es el más simple y menos preciso de los métodos de aproximación de ecuaciones diferenciales ordinarias. La pendiente o derivada en el punto inicial (①) es extrapolado para obtener el siguiente valor en la función (②). El método tiene precisión de primer orden.*

Con el objetivo de remediar lo errores anteriormente mencionados, se desarrollaron nuevos métodos de aproximación, como el método de Euler mejorado.

Método de Euler mejorado

La diferencia de este método *(Figura 5)* respecto al método anteriormente mencionado consiste en que el método de Euler usa la pendiente de la función en el punto inicial del intervalo para extrapolar el valor del punto siguiente. Por el contrario, en el método de Euler mejorado, dicho proceso no será suficiente para hallar el valor del siguiente punto y se deberá acompañar de un paso intermedio. Es decir, se debe hallar la derivada de la función en un punto intermedio del intervalo para, a continuación, extrapolar el valor en el punto final del mismo. Por dicho motivo, este método es también conocido como el método del punto medio.

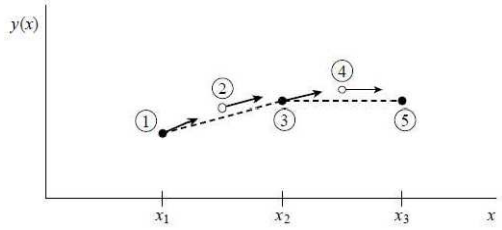

Figura 5: *Método del punto medio: se aumenta la precisión usando el valor de la pendiente en el punto inicial (①) para encontrar los puntos intermedios de los intervalos (②, ④). Una vez hallados, se usará la pendiente en los puntos intermedios a lo largo de toda la función para poder hallar los siguientes puntos. En la imagen, los puntos coloreados en negro (①. ③, ⑤) representan los valores finales de la función, mientras que los puntos no coloreados (②, ④) representan los puntos de la función que han sido descartados una vez que su derivada o pendiente ha sido calculada y usada.*

Las ecuaciones del método serian las del siguiente conjunto:

$$k1 = hf(x_n, y_n)$$
$$k2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk1\right)$$
$$y_{n+1} = y_n + k2 + O(h^3)$$

Método de Runge-Kutta

Fue creado en 1900 por los matemáticos C. Runge y W. Kutta **[7]** y es uno de los procedimientos más usados en la actualidad debido a la alta precisión de sus aproximaciones. Como procedimiento de Runge-Kutta (*Figura 6*) se entiende no solo un método, sino que realmente son un conjunto de métodos iterativos cuya función es la de aproximar el valor de ecuaciones diferenciales ordinarias.

Como mencionamos anteriormente, el método de Euler se mueve a lo largo de la tangente de una cierta curva que está "cerca" a la curva desconocida o buscada, es decir, cerca del valor real de la función en un punto. Los métodos Runge-Kutta extienden esta idea geométrica al utilizar varias derivadas o tangentes intermedias, en lugar de solo una, para aproximar la función desconocida. Para realizar este método se deben calcular cuatro valores en cada intervalo *k1, k2, k3* y *k4*, correspondientes a cuatro pendientes intermedias, y se debe elegir un paso de integración *h*. Las ecuaciones del método de Runge-Kutta de cuarto orden son las siguientes:

$$k1 = hf(x_n, y_n)$$
$$k2 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k1}{2}\right)$$
$$k3 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k2}{2}\right)$$
$$k4 = hf(x_n + h, y_n + k3)$$
$$y_{n+1} = y_n + \frac{k1}{6} + \frac{k2}{3} + \frac{k3}{3} + \frac{k4}{6} + O(h^5)$$

Para la implementación del applet, se ha usado el algoritmo de Runge-Kutta de cuarto orden. Las aproximaciones del método de Runge-Kutta de cuatro orden es mucho mayor que la de los métodos de Euler anteriormente mencionados, por lo que fue el método usado para la implementación del applet sobre el sistema dinámico del oscilador de Hemlholtz.

Figura 6: *Método de Runge-Kutta de cuarto orden. Para cado anchura de paso "h" se debe calcular la pendiente cuatro veces: una en el punto inicial (1) y otras en los puntos intermedios (2, 3) y otro en el punto final del intervalo (4). A Partir de estas derivadas se calcula el valor final de la función (mostrado con un punto negro relleno).*

2.3. Metodología empleada

La ingeniería del software es la disciplina o área de la informática que ofrece métodos y técnicas para desarrollar y mantener software de calidad. Las técnicas de desarrollo que ofrece tienen como objetivo mejorar la productividad y la calidad del software estableciendo metodologías sistemáticas o predecibles. Todas las metodologías existentes, así como la que se ha elegido, se basan en la metodología clásica o en cascada.

En el caso del desarrollo de este applet se ha optado por la metodología incremental **[8]**, ya que se decidió desarrollar inicialmente las partes básicas del applet para, posteriormente, ir añadiendo sucesivamente nuevas características que completasen la interfaz y funcionalidad del mismo (incrementos). En la *Figura 7* se presenta el esquema que representa la metodología para el desarrollo incremental de proyectos software.

Una metodología incremental presenta la ventaja de ser dinámica y flexible. Permite usar las salidas de las etapas precedentes, como entradas en las etapas sucesivas, y facilita corregir cualquier error detectado o llevar a cabo mejoras en los distintos productos que se generan a lo largo de su aplicación.

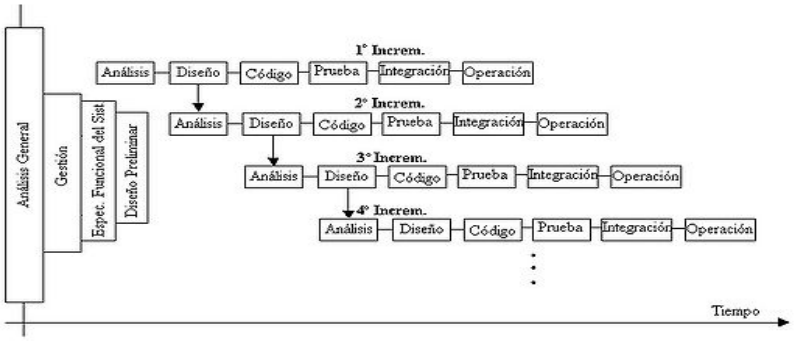

Figura 7: *Método incremental. Se muestra el esquema general correspondiente a la metodología incremental, tanto las etapas de la base inicial como las etapas de los sucesivos incrementos.*

La idea principal de la metodología iterativa de desarrollo de software es implementar las aplicaciones de manera incremental, permitiendo al usuario sacar ventaja de lo aprendido a lo largo del desarrollo anterior incrementando, sucesivamente, versiones de la aplicación. Los pasos claves en el proceso son comenzar con una implementación simple de los requerimientos del sistema, e iterativamente mejorar la secuencia evolutiva de versiones hasta que el sistema completo este implementado. En cada iteración se realizan cambios en el diseño y se agregan nuevas funcionalidades y capacidades al sistema. En cuanto a las diferentes etapas de la metodología iterativa, son las siguientes:

1. **Análisis de requisitos:** En esta etapa se definen las necesidades y condiciones a satisfacer por el proyecto o incremento nuevo. Es trascendental que el análisis de requisitos alcance un estado óptimo antes de alcanzar la fase de “Diseño”. Los buenos requisitos no deben tener ambigüedades ni contradicciones. En esta etapa se tuvo que comprender el sistema dinámico del oscilador de Helmholtz y el método de aproximación establecido para resolverlo y crear un esquema o idea elemental de cómo implementar la interfaz gráfica del applet, idea que fue cambiando progresivamente en las sucesivas iteraciones del desarrollo del applet.

2. **Diseño:** En esta etapa se define una solución que convierta los requisitos en la aplicación deseada. El análisis de requisitos se centraba en que se debía hacer en la aplicación, mientras que la fase de diseño hace hincapié en cómo se debe realizar. En esta etapa se diseñó el diagrama de clases y se decidió la función que tendría cada una de las clases de la aplicación desarrollada.

3. **Programación:** En esta fase se codifica, es decir, se lleva a código fuente las clases definidas en la etapa anterior de “Diseño”. En esta etapa se codificaron las clases, diseñadas en la etapa anterior de análisis, en el lenguaje de programación JAVA con la ayuda del entorno NetBeans.

4. **Pruebas:** Consiste en comprobar que el software realice correctamente las tareas indicadas en la fase de análisis de requisitos. En esta fase se comprobaron, entre otras tareas, que la animación de las graficas se reflejaba correctamente o que se pudiera desplazar la representación hacia los lados.

5. **Mantenimiento:** El mantenimiento de software es el proceso de control, mejora y optimización del software ya desarrollado. Incluye depuración de errores y defectos que puedan haberse filtrado de la fase de pruebas. Esta fase es la última antes de iterar, según el modelo iterativo incremental, que se aplica al ciclo de vida del desarrollo de software. En esta etapa, entre otras acciones, se mejoró el aspecto de la interfaz de la aplicación y se fueron incorporando las nuevas funcionalidades al applet.

3. Descripción informática

Para el desarrollo e implementación del sistema descrito en este proyecto de fin de carrera se ha recurrido a un applet de JAVA **[9]**. Podríamos definir los applets como programas escritos en JAVA, que se descargan desde un servidor y se ejecutan en el navegador del cliente, es decir, en el navegador de la persona que quiera ejecutar dicho programa. Para añadir el applet en una página Web, se usa una página HTML, que será donde se añadirá la aplicación desarrollada en JAVA. El navegador del usuario es el que se encargará de ejecutar dicho documento HTML.

Para ejecutar un applet se puede recurrir al AppletViewer, desarrollado por la compañía Sun MicroSystems, o por medio de un navegador web, como por ejemplo Internet Explorer, Mozilla Firefox, Chrome, Netscape, Opera o Safari entre otros. Entre las características principales de los applets se pueden citar las siguientes:

- Los applets poseen un esquema de seguridad que restringe el acceso a partes sensibles de la máquina en la que se ejecutan. Como ejemplo de este tipo de acciones podemos citar, por ejemplo, el acceso a archivos del usuario para escribir en ellos o modificar información. Por el contrario, si se desea que los applets puedan realizar dicho tipo de acciones, se les deberán entregar los permisos de ejecución correspondientes.
- La gran ventaja de los applets es su portabilidad. Son multiplataforma, pues los applets de JAVA se pueden ejecutar en cualquier navegador disponible y, además, en cualquiera de los sistemas operativos que dispongan de JVM, entre otros, Windows, Linux o Mac OS.
- Como desventaja que tienen los applets podríamos decir que los navegadores que los vayan a ejecutar necesitan un plug-in de JAVA, que no siempre está disponible en todos los navegadores. Algunas organizaciones solo permiten la instalación de software a los administradores. Como resultado, muchos usuarios, sin privilegios para instalar el plug-in en su navegador, no pueden ver los applets.

3.1. Especificación

Podríamos definir la especificación de requisitos como el proceso en el que se describe el comportamiento esperado en el software una vez desarrollado. Gran parte del éxito de un proyecto de software radicará en la identificación de los requisitos. Los requisitos se pueden clasificar en funcionales y no funcionales. Por un lado, los requisitos funcionales son aquellas características que debe incorporar el sistema o aplicación a desarrollar, como acciones que este deberá ser capaz de desempeñar. Por otro lado, los requisitos no funcionales están más enfocados al diseño o la implementación de la aplicación. Son aquellos requisitos que ni describen información a guardar ni las funciones que la aplicación debe desarrollar: calidad, eficiencia, disponibilidad, etc.

- Requisitos funcionales:
  - La aplicación debe representar las gráficas obtenidas a partir de las variables insertadas por el usuario
  - La aplicación podrá mostrar una animación mediante un dibujo en la que se pueda ver una representación de los resultados obtenidos sobre el movimiento de un barco.
  - El applet debe representar la gráfica o el dibujo del oscilador de Helmholtz teniendo en cuenta o no las variables de control (ε y Φ).
  - La aplicación permitirá seleccionar la representación de una sola gráfica o la representación de dos gráficas simultáneamente para poder compararlas.
  - El programa dispondrá de los elementos necesarios que recojan y comprueben los parámetros de entrada introducidos por el usuario.
  - El programa deberá poder mostrar la representación elegida por el usuario directamente o mediante una animación, es decir, de manera estática o dinámica.
  - La aplicación dispondrá de los controles necesarios para que el usuario pueda pausar, reanudar, interrumpir o iniciar una representación.
  - El programa permitirá variar la velocidad de representación de las animaciones.
  - El programa poseerá controles que permitan que el usuario pueda mover en todas las direcciones, acercarse o alejarse de las representaciones.
  - La aplicación permitirá cambiar el color de las representaciones.
  - La aplicación permitirá mostrar u ocultar los ejes de representación.
  - La aplicación permitirá cortar la gráfica en cualquier momento durante su representación dinámica y limpiar la pantalla de dibujo en cualquier otro momento.

- Requisitos no funcionales:
  - El programa debe ser desarrollado en un applet de JAVA.
  - La interfaz de la aplicación debe ser fácil de usar e intuitiva, a la vez que consistente.
  - La interfaz de la aplicación debe ser atractiva para el usuario.
  - La aplicación debe responder a los órdenes de usuario de una forma efectiva y rápida.
  - La aplicación no permitirá realizar operaciones innecesarias dependiendo del momento en que se encuentre el programa.
  - El programa deberá controlar la introducción de datos erróneos que pudiesen provocar errores de ejecución de la aplicación.
  - La aplicación deberá ser eficiente y por lo tanto tendrá que ocuparse de controlar todos los posibles errores durante la ejecución para que el usuario no lo perciba.

3.2. Diseño

El lenguaje de programación de JAVA proporciona las clases AWT y Swing, que son muy útiles para implementar interfaces gráficas de usuario. En la implementación del applet se ha dado prioridad al uso de la clase Swing, ya que proporciona una serie de ventajas frente

a la AWT, a pesar de que hemos utilizado las dos: posibilita la inclusión de iconos en los botones; se pueden elegir varios tipos de bordes disponibles para los elementos; los botones pueden tener una forma no rectangular; etc. Todo esto permite crear una interfaz mucho más vistosa y elegante. Swing proporciona muchas clases prefabricadas para hacer más fácil la implementación de interfaces gráficas.

A continuación, explicaremos brevemente los elementos que han sido utilizados en el diseño gráfico:

- ***JPanel:*** es un contenedor que sirve para agrupar otros elementos del mismo o diferente tipo. Facilita el diseño y la organización de los componentes en la interfaz de usuario. En este applet hemos usado varios JPanel, entre ellos las tres grandes superficies rectangulares: la zona central de dibujo y las que están a la izquierda y derecha de la zona de representación de las gráficas. Dentro de estos paneles, todos los conjuntos de elementos se encuentran organizados en respectivos paneles. A su vez, todos los elementos que componen el applet se encuentran contenidos en el panel general del applet.
- ***JTabbedPanel:*** es un contenedor que se emplea para agrupar otros elementos en pestañas. Este elemento da la posibilidad al usuario de acceder a diferentes elementos o variables seleccionando las diferentes pestañas correspondientes a cada grupo. Este elemento lo hemos usado para poder contener las variables correspondientes a las gráficas 1 y 2, cada una en una pestaña diferente.
- ***ButtonGroup:*** permite contener varios JRadioButton relacionados de tal manera que solo se pueda seleccionar uno a la vez. Lo usamos para contener de forma excluyente los botones de selección del tipo de representación estática o dinámica.
- ***JRadioButton:*** es un botón que puede ser seleccionado o no dependiendo de la funcionalidad buscada. Permite la elección entre un conjunto de elementos. Se usa junto a la clase JButtonGroup, que asociada a un conjunto de JRadioButton, posibilita que solo sea posible la elección de uno de los JRadioButton. Los usaremos para poder elegir entre la representación dinámica o estática de la gráfica.
- ***JComboBox:*** consiste en una lista desplegable de cajas de texto. Se puede elegir una de dichas opciones de las cajas de texto y estas son excluyentes. Inicialmente, sólo se puede ver el valor de una de las opciones de la caja de texto y, para ver las demás se debe hacer clic en la pestaña pequeña situado en su extremo derecho. Dicha acción desplegará una lista con todas las opciones de las cajas de texto a elegir. En este applet, se usa para poder elegir el tipo de gráfica a representar, es decir, para elegir si queremos representar la gráfica sin o con control o si preferimos mostrar la animación con el dibujo del barco, igualmente sin o con control.
- ***JLabel:*** es una etiqueta de texto. Lo usamos para informar al usuario a qué variable corresponde cada uno de los JTextFields con que se corresponden; para indicar el tipo de gráfico seleccionado junto con su fórmula; y como contenedor para insertar la imagen representativa de la universidad.

- ***JTextField:*** es un componente que permite la entrada de texto por parte del usuario. Lo hemos usado para que el usuario introduzca los diferentes valores de las variables de las gráficas.
- ***JCheckBox:*** se trata de un componente que se emplea para seleccionar o no una opción determinada. Se usa para indicar si queremos que se muestren los ejes de coordenadas en la representación o, en el JTabbedPanel de las gráficas, para indicar si queremos representar una o dos gráficas.
- ***JButton:*** es un componente en el que el usuario hace clic para desencadenar una acción específica. En este applet se usa para iniciar, pausar o detener la reproducción de una gráfica mediante los botones de "Play", "Pause" y "Stop" respectivamente; cambiar el color en el que se representa la gráfica; en el caso de que se esté representado dinámicamente, el icono con la tijera, para borrar la trayectoria anteriormente dibujada, y en cualquier caso en el que la representación no esté en curso sirve para borrar el lienzo de dibujo; los botones de movimiento sirven para desplazar la representación en cualquiera de las direcciones o restablecer la posición original; y, por último, los botones del panel de zoom se usan para alejar, acercarse o restablecer el zoom a la gráfica. En cada uno de estos botones se ha insertado un icono representativo de la acción que desencadenan.
- ***JProgressBar:*** consiste en una barra de progreso que permite ver de una forma gráfica la evolución de una tarea. Nos será útil para poder ver la evolución de la reproducción de las gráficas, es decir, nos mostrará el porcentaje completado de la ejecución de la representación. Mostrará también de forma numérica el porcentaje de progreso.
- ***JSlider:*** permite la selección de un valor de una forma gráfica, que debe oscilar entre un máximo y un mínimo, mediante el deslizamiento de una barra. Este elemento lo emplearemos para aumentar o disminuir la velocidad de representación de la gráfica en caso de que se haya elegido la representación dinámica.

3.2.1. Diseño del interfaz

A continuación, nos centraremos en las partes significativas de la interfaz del programa, en la cual se encuentran todos los elementos anteriormente descritos:

1. Panel general del applet

Este panel es el contenedor principal del applet y está compuesto por tres paneles contenedores para separar las diferentes partes del applet, ordenados mediante "Border Layout". Este contenedor está diseñado mediante un "Etched Border" y sus medidas por defecto reflejan las medidas generales del applet en la página web que son de 850*600.

1.1. Panel del dibujo

Este es el panel central del applet y está formado por tres paneles interiores ordenados mediante "Border Layout" que forman la parte principal del applet, ya que indican el sistema dinámico representado y contienen la representación en sí. Este panel está diseñado mediante un "Etched Border" y sus dimensiones predefinidas son 500*500.

1.1.1. Panel de información

Este panel *(Figura 8)* se encuentra en la parte central superior del applet y está formado por dos jLabels que contienen el título de la representación elegida y la función que la representa respectivamente. Dichas etiquetas están ordenadas mediante "Grid Layout" y su diseño se ha realizado mediante un "Etched Border". El contenido de estos jLabel puede modificarse a lo largo de la ejecución del programa dependiendo de la representación elegida. Por defecto en el applet el título y función elegida son los correspondientes a la representación gráfica "Sin control".

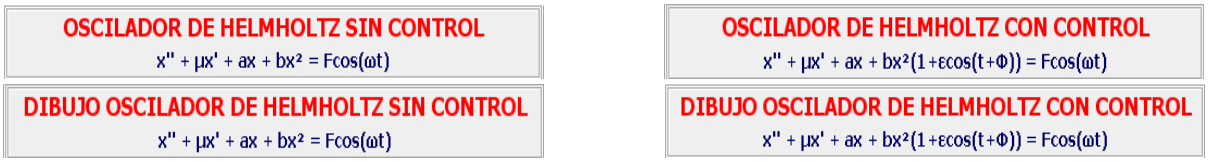

Figura 8: *Paneles de información que muestran las diferentes ecuaciones con sus respectivas funciones.*

1.1.2. Panel del lienzo

Este panel *(Figura 9)* se encuentra en la parte central del applet y es el más importante, ya que contiene el lienzo de dibujo donde se mostrará la representación deseada. El lienzo se encuentra centrado en dicho panel mediante "Border Layout" y para su diseño se ha elegido un "Titled Border" en el cual el título es "GRÁFICA".

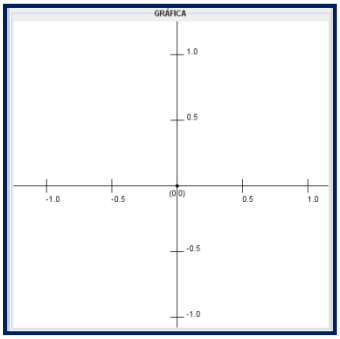

Figura 9: *Panel del lienzo donde se mostrará la representación seleccionada.*

1.1.3. Panel de progreso

Este panel *(Figura 10)* se encuentra situado en la zona central inferior del applet y contiene un solo elemento organizado mediante "Border Layout", una JProgressBar, que es la barra de progreso en la que se muestra la evolución de la representación. El diseño del panel es mediante un "Titled Border" con el texto "Progreso". La JProgressBar muestra en su interior el porcentaje de avance de la representación de manera tanto gráfica como numérica. Este número se encuentra centrado y el color de relleno de la barra es azul.

Por defecto en el applet, el valor inicial de esta barra de progreso es el mínimo (0).

Figura 10: *Panel donde se encuentra la barra de progreso de la representación (inicial y finalizada).*

1.2. Panel de datos

Este panel se encuentra situado en la parte izquierda del applet y contiene tres paneles ordenados mediante "Border Layout" con las funcionalidades y variables principales del programa. Está diseñado mediante un "Etched Border" y sus dimensiones predefinidas son 250*600.

1.2.1. Panel de variables

Este panel *(Figura 11)* se encuentra en la parte izquierda superior del applet y se encarga de contener y mostrar todas las variables para la representación de la gráfica o gráficas. Está compuesto por dos paneles ordenados mediante "Border Layout" y su diseño se ha realizado mediante un "Titled Border" con el título "Gráficas".

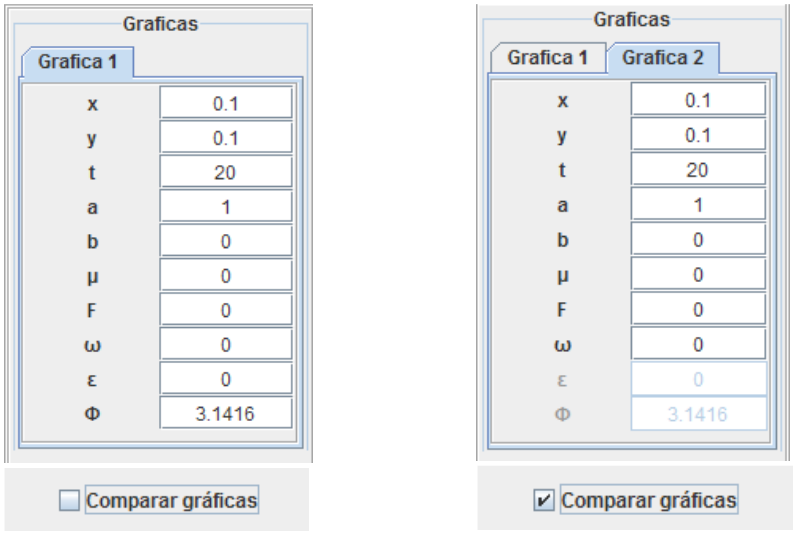

Figura 11: *Panel donde se muestran los parámetros de entrada a insertar por el usuario y la opción de representar una o dos órbitas.*

1.2.1.1. Panel gráficas

Este panel es un JTabbedPanel que contiene los dos paneles necesarios para todas las variables de una o dos gráficas. Estos dos paneles interiores se encuentran cada uno en su pestaña correspondiente, marcadas mediante los títulos "Gráfica 1" y "Gráfica 2". Las variables estarán habilitadas o deshabilitadas dependiendo del transcurso de la ejecución.

1.2.1.1.1. Panel gráfica1

Este panel está en la primera pestaña del JTabbedPanel y contiene un grupo de JLabel con sus respectivos JTextField, utilizados para que el usuario pueda insertar todas las variables necesarias para la representación de la primera gráfica. Estos elementos están ordenados mediante "Grid Layout". El diseño de este panel se ha realizado mediante un "Etched Border".

Las diferentes variables se encuentran indicadas en los diferentes jLabel y sus valores se encuentran rellenados con un valor por defecto al iniciar el applet. Además, las variables correspondientes al sistema "Con control" están inicialmente deshabilitadas.

1.2.1.1.2. Panel gráfica2

Este panel está en la segunda pestaña del JTabbedPanel y contiene un grupo de JLabel con sus respectivos JTextField, utilizados para que el usuario pueda insertar todas las variables necesarias para la representación de la segunda gráfica. Dicho panel permanecerá o no dependiendo de si se desea mostrar una o dos gráficas y se controla desde el JCheckBox situado debajo de estos paneles. Estos elementos están ordenados mediante "Grid Layout". El diseño de este panel se ha realizado mediante un "Etched Border".

Las diferentes variables se encuentran indicadas en los diferentes jLabel y sus valores se encuentran rellenados con un valor por defecto al iniciar el applet. Además, las variables correspondientes al sistema "Con control" están inicialmente deshabilitadas.

1.2.1.2. Panel común

Este panel se encuentra debajo del panel de las variables de las gráficas y contiene un JCheckBox organizado mediante "Border Layout", para poder seleccionar si deseamos mostrar la representación de una o dos gráficas. Esto se consigue seleccionando o deseleccionando dicho elemento. Al mismo tiempo, también podemos conseguir mostrar u ocultar el panel de las variables de la segunda gráfica.

Por defecto al iniciar el applet esta casilla no se encuentra seleccionada.

1.2.2. Panel representativo

Este panel se encuentra en la parte izquierda central del applet y contiene las principales opciones de representación. Está formado por tres paneles ordenados mediante "Border Layout" y está diseñado mediante un "Etched Border".

1.2.2.1. Panel estática/dinámica

Este panel *(Figura 12)* se utiliza para contener un ButtonGroup, que a su vez contiene dos RadioButton, usados para poder elegir el tipo de representación: estática o dinámica. La ventaja de usar estos elementos es que sólo nos dejará elegir un tipo de representación, es decir, que son excluyentes y si seleccionamos uno implica que se deselecciona automáticamente el otro. El diseño elegido para este panel es "Titled Border" con el título "Representación".

Por defecto al arrancar el applet se encuentra seleccionada la opción de estática.

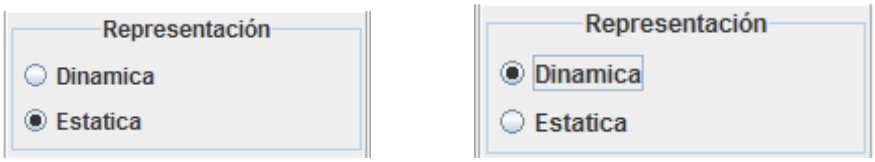

Figura 12: *Panel para seleccionar si se desea una representación estática o dinámica.*

1.2.2.2. Panel ejes

Este panel *(Figura 13)* está diseñado mediante "Titled Border" con el título "Ejes" y contiene el JCheckBox para poder seleccionar si queremos que se muestren o no los

ejes de representación. Esto se consigue marcando o desmarcando la caja, respectivamente.
Por defecto al iniciar el applet esta casilla está deseleccionada.

Figura 13: *Panel para seleccionar si se desean mostrar o no los ejes de coordenadas.*

1.2.2.3. Panel velocidad

Este panel *(Figura 14)* contiene una JSlider usada para modificar la velocidad de representación dinámica. El valor mínimo de dicha barra de velocidad se sitúa a la izquierda y el valor de máxima velocidad a la derecha. Dicho panel se encuentra diseñado mediante "Titled Border" con el título "Velocidad".
Por defecto al iniciar el applet el valor de esta velocidad será el mínimo, es decir, que la velocidad de representación será la más lenta posible.

Figura 14: *Panel para seleccionar la velocidad de representación dinámica.*

1.2.3. Panel funcional

Este panel *(Figura 15)* se encuentra en la parte izquierda inferior del applet y contiene los tres JButton necesarios para el funcionamiento del applet que son "Play", "Pause" y "Stop" ordenados mediante "Border Layout". La funcionalidad de dichos botones es la de iniciar o continuar, pausar y parar la ejecución del programa, respectivamente. Dichos botones tienen un icono representativo en su interior. El diseño del applet se ha realizado mediante un "Titled Border" con el título "Función".
Por defecto al iniciar el applet los botones de "Pause" y "Stop" están deshabilitados.

Figura 15: *Panel que contiene los botones para iniciar, pausar, reanudar o finalizar la representación.*

1.3. Panel de preferencias

Este panel se encuentra en la parte derecha del applet y contiene las funciones auxiliares y preferencias de representación del sistema. Está formado por tres paneles ordenados mediante "Border Layout". Está diseñado mediante un "Etched Border" y sus dimensiones predefinidas son 100*600.

1.3.1. Panel de control

Este panel *(Figura 16)* se encuentra en la parte derecha superior del applet y está diseñado mediante un "Titled Border" con título "Control". Contiene un único elemento organizado mediante "Border Layout" que es del tipo JComboBox y se utiliza para poder seleccionar el tipo de representación deseada. Este combo nos permite elegir entre cuatro opciones de representación "gráfica sin control", "gráfica con control", "dibujo sin control" o "dibujo con control".

Por defecto al iniciar el applet aparecerá seleccionada la opción de "Sin control".

Figura 16: *Panel donde se pueden seleccionar las diferentes representaciones deseadas..*

1.3.2. Panel extra

Este panel se encuentra en la parte derecha central del applet y contiene tres paneles con funciones auxiliares del applet ordenados mediante "Border Layout". El diseño de este panel se ha realizado mediante un "Etched Border".

1.3.2.1. Panel de movimiento

Este panel *(Figura 17)* está diseñado mediante un "Titled Border" con título "Movimiento". Contiene los botones necesarios para poder mover la representación en cualquier dirección ("Arriba", "Abajo", "Derecha", "Izquierda") y poder volver a la posición original ("Reset"). Dichos botones contienen un icono representativo. Para la correcta colocación de dichos botones hemos insertado varios paneles auxiliares.

Al iniciar el applet, el botón correspondiente a la opción de "Reset" se encuentra deshabilitado.

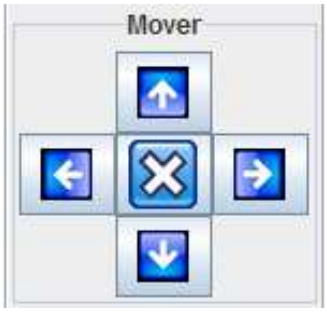

Figura 17: *Panel donde se pueden seleccionar los diferentes movimientos del lienzo de dibujo.*

1.3.2.2. Panel de zoom

Este panel *(Figura 18)* está diseñado mediante un "Titled Border" con título "Zoom". Contiene los botones necesarios para poder alejar o acercar la representación y poder volver a la escala original. Hay que resaltar que los zoom, tanto para alejar como para acercar son infinitos por lo cual están siempre habilitados. Dichos botones contienen un icono representativo.

Al iniciar el applet, el botón correspondiente a la opción de "Reset" se encuentra deshabilitado.

Figura 18: *Panel donde se puede ampliar, reducir o restaurar el zoom aplicado sobre el lienzo de dibujo.*

1.3.2.3. Panel de opciones

Este panel *(Figura 19)* está diseñado mediante un "Titled Border" con título "Opciones". Contiene dos botones ordenados mediante "Border Layout" que sirven para cortar la gráfica en ejecución o limpiarla en cualquier otro momento y para elegir el color de la gráfica representada, respectivamente. Dichos botones contienen un icono representativo.

Al cliquear el botón de la paleta de colores, se abrirá una nueva pantalla del applet para poder seleccionar el color deseado para la gráfica o las gráficas representadas. En dicha paleta seleccionaremos el color deseado y a continuación el botón "Aceptar". Podremos cerrar esta ventana pinchando sobre el símbolo de "X". Esta paleta está formada por dos paneles que contienen el conjunto de colores permitidos y los botones de "Aceptar gráfica1" y "Aceptar gráfica2" que permiten seleccionar el color de la gráfica correspondiente. Si solo estamos representando una gráfica, el botón de color de la segunda gráfica permanece deshabilitado.

Figura 19: *Panel donde se puede cortar o borrar la representación o seleccionar el color en que queremos que se muestren las gráficas.*

1.3.3. Panel del logo

Este panel *(Figura 20)* se encuentra en la parte derecha inferior del applet y contiene un jLabel utilizado para insertar un icono, que corresponde al icono del escudo de la

URJC. Dicho icono se encuentra centrado en el JLabel, y al estar ordenado en el panel con un "Border Layout", ocupa prácticamente la totalidad del panel.

Figura 20: *Panel donde se muestra el escudo de la Universidad Rey Juan Carlos.*

Todos estos paneles tienen un tamaño predefinido específico correspondiente a su importancia dentro del applet. En la *Figura 21* se muestra el diseño final de la interfaz del applet:

Figura 21: *Interfaz inicial del applet. Es la interfaz que se muestra inicialmente al ejecutar el applet.*

3.2.2. Diseño de clases

A continuación se muestra *(Figura 22)* el esquema seguido para las clases del proyecto de final de carrera junto con una breve explicación de cada una de ellas. Con el fin de que el diagrama de clases sea lo más entendible posible, se ha optado por simplificarlo:

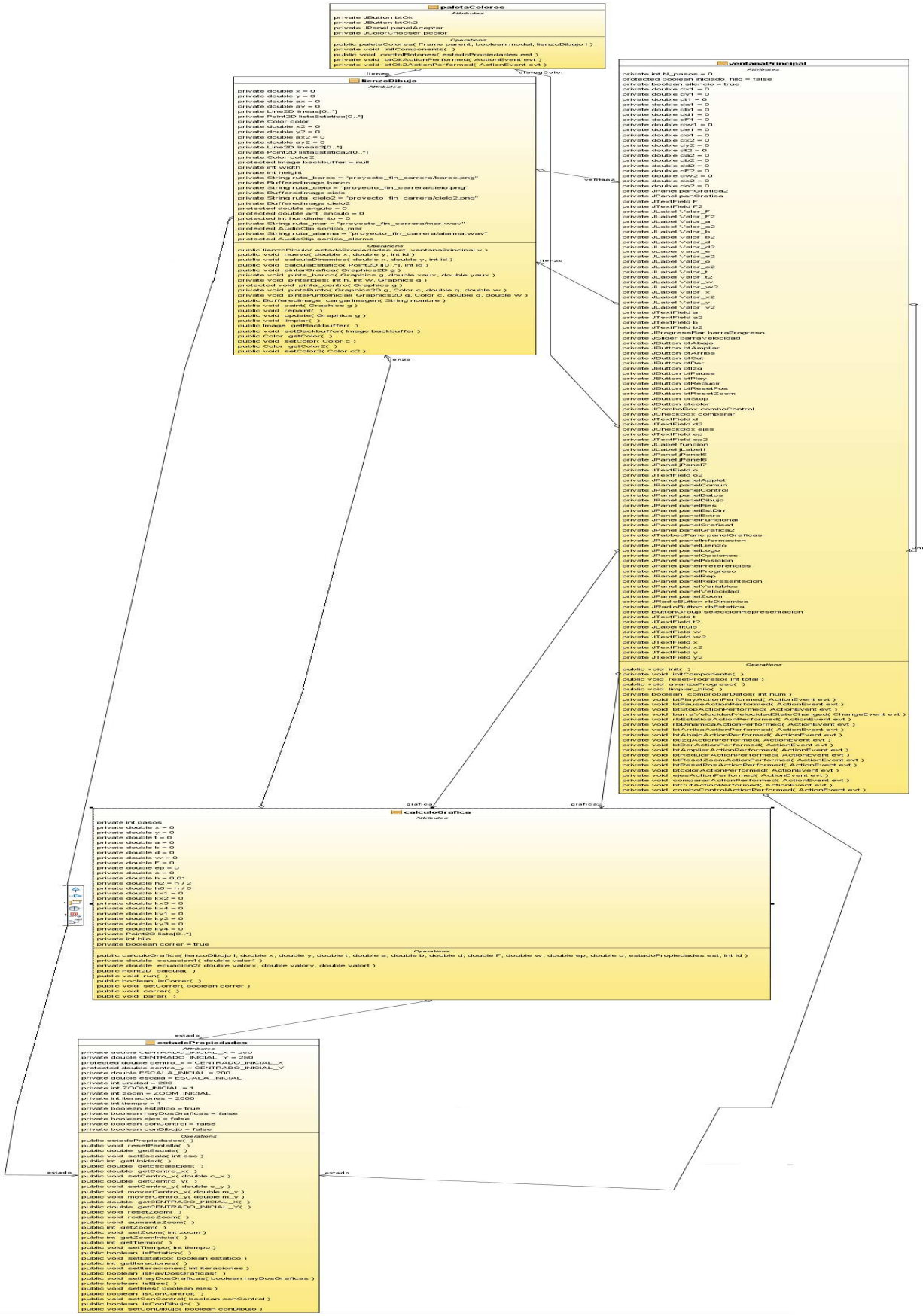
paletaColores
Attributes
private JButton btOk
private JButton btOk2
private JPanel panelAceptar
private JColorChooser pcolor
Operations
public paletaColores(Frame parent, boolean modal, lienzoDibujo l)
private void initComponents( )
public void controlBotones(estadoPropiedades est)
private void btOkActionPerformed(ActionEvent evt)
private void btOk2ActionPerformed(ActionEvent evt)
lienzo
dialogColor
lienzoDibujo
Attributes
private double x = 0
private double y = 0
private double ax = 0
private double ay = 0
private Line2D lineas[0..*]
private Point2D listaEstatica[0..*]
private Color color
private Image backbuffer = null
private int width
private int height
private String ruta_barco = "proyecto_fin_carrera/barco.png"
private BufferedImage barco
private String ruta_cielo = "proyecto_fin_carrera/cielo.png"
private BufferedImage cielo
private String ruta_cielo2 = "proyecto_fin_carrera/cielo2.png"
private BufferedImage cielo2
protected double angulo = 0
protected double ant_angulo = 0
protected int hundimiento = 0
private String ruta_mar = "proyecto_fin_carrera/mar.wav"
protected AudioClip sonido_mar
private String ruta_alarma = "proyecto_fin_carrera/alarma.wav"
protected AudioClip sonido_alarma
Operations
public lienzoDibujo(estadoPropiedades est, ventanaPrincipal v)
public void nuevo(double x, double y, int id)
public void calculaDinamico(double x, double y, int id)
public void calculaEstatico(Point2D t[0..*], int id)
public void pintaGrafica(Graphics2D g)
private void pinta_barco(Graphics g, double xaux, double yaux)
private void pintarEjes(int h, int w, Graphics g)
protected void pinta_centro(Graphics g)
private void pintaPunto(Graphics2D g, Color c, double q, double w)
private void pintaPuntoInicial(Graphics2D g, Color c, double q, double w)
public BufferedImage cargarImagen(String nombre)
public void paint(Graphics g)
public void repaint( )
public void update(Graphics g)
public Image getImage( )
public Image getBackbuffer( )
public void setBackbuffer(Image backbuffer)
public Color getColor( )
public void setColor(Color c)
public Color getColor2( )
public void setColor2(Color c2)
ventana
lienzo
lineas
ventanaPrincipal
Attributes
private int N_pasos = 0
protected boolean iniciado_hilo = false
private boolean silencio = true
private double dx1 = 0
private double dy1 = 0
private double dF1 = 0
private double da1 = 0
private double db1 = 0
private double dd1 = 0
private double dF1 = 0
private double dw1 = 0
private double de1 = 0
private double ds1 = 0
private double dx2 = 0
private double dy2 = 0
private double dt2 = 0
private double da2 = 0
private double db2 = 0
private double dd2 = 0
private double dF2 = 0
private double dw2 = 0
private double de2 = 0
private double ds2 = 0
private JPanel panGrafica2
private JPanel panGrafica
private JTextField F
private JTextField F2
private JLabel Valor_F
private JLabel Valor_F2
private JLabel Valor_a
private JLabel Valor_a2
private JLabel Valor_b
private JLabel Valor_b2
private JLabel Valor_d
private JLabel Valor_d2
private JLabel Valor_e
private JLabel Valor_e2
private JLabel Valor_o
private JLabel Valor_o2
private JLabel Valor_t
private JLabel Valor_t2
private JLabel Valor_w
private JLabel Valor_w2
private JLabel Valor_x
private JLabel Valor_x2
private JLabel Valor_y
private JLabel Valor_y2
private JTextField a
private JTextField a2
private JTextField b
private JTextField b2
private JProgressBar barraProgreso
private JSlider barraVelocidad
private JButton btAbajo
private JButton btAmpliar
private JButton btArriba
private JButton btCal
private JButton btDer
private JButton btIzq
private JButton btPause
private JButton btPlay
private JButton btReducir
private JButton btResetPos
private JButton btResetZoom
private JButton btStop
private JButton btcolor
private JComboBox comboControl
private JCheckBox comparar
private JTextField d
private JTextField d2
private JCheckBox ejes
private JTextField ep
private JTextField ep2
private JLabel funcion
private JLabel jLabel1
private JPanel jPanel5
private JPanel jPanel6
private JPanel jPanel7
private JTextField o
private JTextField o2
private JPanel panelApplet
private JPanel panelComun
private JPanel panelControl
private JPanel panelDatos
private JPanel panelDibujo
private JPanel panelEjes
private JPanel panelEntrada
private JPanel panelF
private JPanel panelFuncional
private JPanel panelGrafica1
private JPanel panelGrafica2
private JTabbedPane panelGraficas
private JPanel panelInformacion
private JPanel panelLienzo
private JPanel panelLogo
private JPanel panelOpciones
private JPanel panelPosicion
private JPanel panelPreferencias
private JPanel panelProgreso
private JPanel panelRep
private JPanel panelRepresentacion
private JPanel panelVariables
private JPanel panelVelocidad
private JPanel panelZoom
private JRadioButton rbDinamica
private JRadioButton rbEstatica
private ButtonGroup seleccionRepresentacion
private JTextField t
private JTextField t2
private JLabel titulo
private JTextField w
private JTextField w2
private JTextField x
private JTextField x2
private JTextField y
private JTextField y2
Operations
public void init( )
private void initComponents( )
public void resetProgreso(int total)
public void setProgreso( )
public void limpiar_hilo( )
private boolean comprobarDatos(int num)
private void btPlayActionPerformed(ActionEvent evt)
private void btPauseActionPerformed(ActionEvent evt)
private void btStopActionPerformed(ActionEvent evt)
private void barraVelocidadVelocidadStateChanged(ChangeEvent evt)
private void rbEstaticaActionPerformed(ActionEvent evt)
private void rbDinamicaActionPerformed(ActionEvent evt)
private void btArribaActionPerformed(ActionEvent evt)
private void btAbajoActionPerformed(ActionEvent evt)
private void btIzqActionPerformed(ActionEvent evt)
private void btDerActionPerformed(ActionEvent evt)
private void btAmpliarActionPerformed(ActionEvent evt)
private void btReducirActionPerformed(ActionEvent evt)
private void btResetZoomActionPerformed(ActionEvent evt)
private void btResetPosActionPerformed(ActionEvent evt)
private void btcolorActionPerformed(ActionEvent evt)
private void ejesActionPerformed(ActionEvent evt)
private void compararActionPerformed(ActionEvent evt)
private void btnCalActionPerformed(ActionEvent evt)
private void comboControlActionPerformed(ActionEvent evt)
grafica
grafica2
calculoGrafica
Attributes
private int pasos
private double x = 0
private double y = 0
private double t = 0
private double a = 0
private double b = 0
private double d = 0
private double w = 0
private double F = 0
private double ep = 0
private double o = 0
private double h = 0.01
private double h2 = h / 2
private double h6 = h / 6
private double kx1 = 0
private double kx2 = 0
private double kx3 = 0
private double kx4 = 0
private double ky1 = 0
private double ky2 = 0
private double ky3 = 0
private double ky4 = 0
private Point2D lista[0..*]
private int hilo
private boolean correr = true
Operations
public calculoGrafica(lienzoDibujo l, double x, double y, double t, double a, double b, double d, double F, double w, double ep, double o, estadoPropiedades est, int id)
private double ecuacion1(double valor)
private double ecuacion2(double valorx, double valory, double valort)
public Point2D calcula( )
public void run( )
public boolean isCorrer( )
public void setCorrer(boolean correr)
public void correr( )
public void parar( )
estado
estadoPropiedades
Attributes
private double CENTRADO_INICIAL_X = 260
private double CENTRADO_INICIAL_Y = 260
protected double centro_x = CENTRADO_INICIAL_X
protected double centro_y = CENTRADO_INICIAL_Y
private double ESCALA_INICIAL = 200
private double escala = ESCALA_INICIAL
private int unidad = 200
private int ZOOM_INICIAL = 1
private int zoom = ZOOM_INICIAL
private int iteraciones = 2000
private int tiempo = 1
private boolean estatico = true
private boolean hayDosGraficas = false
private boolean ejes = false
private boolean conControl = false
private boolean conDibujo = false
Operations
public estadoPropiedades( )
public void resetPantalla( )
public double getEscala( )
public void setEscala(int esc)
public int getUnidad( )
public double getEscalaEjes( )
public double getCentro_x( )
public void setCentro_x(double c_x)
public double getCentro_y( )
public void setCentro_y(double c_y)
public void moverCentro_x(double m_x)
public void moverCentro_y(double m_y)
public double getCENTRADO_INICIAL_X( )
public double getCENTRADO_INICIAL_Y( )
public void resetZoom( )
public void reducirZoom( )
public void aumentaZoom( )
public int getZoom( )
public void setZoom(int zoom)
public int getIteraciones( )
public int getTiempo( )
public void setTiempo(int tiempo)
public boolean isEstatico( )
public void setEstatico(boolean estatico)
public int getIteraciones( )
public void setIteraciones(int iteraciones)
public boolean isHayDosGraficas( )
public void setHayDosGraficas(boolean hayDosGraficas)
public boolean isEjes( )
public void setEjes(boolean ejes)
public boolean isConControl( )
public void setConControl(boolean conControl)
public boolean isConDibujo( )
public void setConDibujo(boolean conDibujo)
estado
estado
estado

Figura 22: *Representación del diagrama de clases obtenido tras la implementación del programa.*

Las clases mostradas en el diagrama de clases son las siguientes:

1. Clase ventanaPrincipal:

Esta clase es la clase principal del proyecto. Hereda de la interfaz JApplet, necesaria para todas las clases que quieran implementar un applet.

Las funciones principales de la clase "ventanaPrincipal" son la de iniciar la ejecución del programa y la crear y establecer la disposición de todos los elementos del applet en la interfaz gráfica. Además, desde esta clase se definen las acciones que se desencadenarán cuando el usuario interactúa con los distintos elementos de la interfaz, acciones reflejadas en el diagrama de clases mediante los distintos métodos y propiedades.

Dicha clase también crea los objetos necesarios para desarrollar la aplicación buscada. Esto se realiza en el método principal "init()".

Estas funcionalidades son asociadas a los elementos en el método "init()" y sirven, por ejemplo, para que cuando se pulse el botón con el símbolo de "Play", se dibuje la representación seleccionada.

2. Clase lienzoDibujo:

Esta clase se encarga de controlar todas las representaciones gráficas sobre el lienzo de dibujo para lo cual hereda de la clase "Canvas" **[10]**. Recibe los valores de los puntos calculados y ajusta los puntos para que se representen en el plano de dibujo.

La clase "lienzoDibujo" se encargará de pintar los puntos de la grafica. Para ello, recibirá los puntos de la clase "calculoGrafica".

3. Clase calculoGrafica:

Esta clase calcula los puntos necesarios para pintar las distintas graficas del applet. Estos puntos se calculan mediante el método de aproximación de Runge-Kutta, definido en el método "calcula ()".

Un aspecto a destacar por su importancia en el applet son los threads. Los threads **[11]** o hilos de ejecución son segmentos de código de un programa que se ejecutan secuencialmente de modo independiente de las otras partes del programa. Esto contribuye decisivamente en el rendimiento de las aplicaciones. En nuestro caso los hilos nos son de gran ayuda, ya que la JAVA Virtual Machine (Maquina Virtual de JAVA) es un sistema multitarea, es decir, permite la realización de varias tareas al mismo tiempo.

En esta implementación es necesario el uso de threads y sin ellos no se podrían representar las animaciones de las graficas, pausarlas, o cambiar la velocidad de reproducción de las mismas mediante una barra de velocidad. Para ello, esta clase hereda de la interfaz Thread. Recibe los valores de la clase "ventanaPrincipal".

4. Clase estadoPropiedades:

Esta clase contiene las variables globales del applet, así como los métodos necesarios para acceder y/o modificarlas desde cualquier otro lugar del proyecto.

5. Clase paletaColores:

Esta clase contiene una paleta de colores que utilizaremos para seleccionar los colores de representación de las gráficas. Gracias a esta clase podremos elegir el color deseado y pasárselo a la clase correspondiente mediante sus respectivos métodos.

3.3. Implementación

En este apartado trataremos de explicar los aspectos más importantes y relevantes de las clases de las que consta la implementación del applet.
Respecto al formato elegido para explicar las partes del código de la implementación, se mostrarán las cabeceras de todos los métodos y, posteriormente se incluirá una pequeña sección explicativa del funcionamiento de dicho código.

En primer lugar habría que decir que todas las clases se encuentran dentro del paquete "proyecto_fin_carrera", que es el paquete que contiene todo el proyecto.

- **Clase ventanaPrincipal:** La cabecera de la clase es la siguiente:

  public class ventanaPrincipal extends javax.swing.JApplet {...}

Dicha clase heredará de la clase JApplet, ya que lo que se querrá implementar será un applet de JAVA. Para indicar que la clase hereda de una clase se debe poner la palabra "extends" y a continuación el nombre de la clase heredada.

Esta clase llevará a cabo varias funciones importantes para el applet, entre ellas destacan las siguientes:

- Crear los objetos iniciales necesarios para la ejecución del programa e inicializar las variables y diseño de los diferentes componentes del interfaz del applet.
- Controlar los eventos o el comportamiento de las distintas funcionalidades de la interfaz del applet. Es decir, la clase "ventanaPrincipal" describirá cuales son las acciones que se deberán desencadenar cuando se utilicen los diferentes elementos de la interfaz. Ejemplos de dichas acciones serian la acción que se producirá si un usuario pulsa el botón del "Play", o lo que pasara si un usuario desplaza la barra de velocidad durante la reproducción dinámica de una grafica.

En cuanto a los elementos de la clase "ventanaPrincipal", podemos destacar tanto propiedades como métodos. Las propiedades declaradas representan a las diferentes clases del programa, así como a diferentes variables para controlar la ejecución del applet. En cuanto a métodos, se explican a continuación:

- *public void init() {*

```
try {
   java.awt.EventQueue.invokeAndWait(new Runnable() {
      public void run() {
         initComponents();
         ...
      }
   });
} catch (Exception ex) {
   ex.printStackTrace();
}
}
```

Este es el primero y más importante de los métodos encontrados en la clase. La declaración de este método se realiza automáticamente al crear el applet y es obligatoria ya que es el método principal. Se encarga de iniciar la ejecución del applet inicializando las diferentes componentes de la interfaz y creando los objetos que se corresponden con las diferentes clases del proyecto. En concreto crearemos el objeto correspondiente al lienzo de dibujo, situando dicho lienzo en el centro del panel de representación; el objeto correspondiente a la clase que contiene las propiedades globales del applet; y el objeto correspondiente a la clase que contiene la paleta de colores para la selección del color de las gráficas representadas. Por lo tanto, se asocian las diferentes variables declaradas con las correspondientes clases. También se ocultará el panel correspondiente a las variables para la segunda gráfica, ya que inicialmente y por defecto estará seleccionada la representación de una sola gráfica.

- *public void init Components() {…}*

Este método que se llama desde el método anterior, sirve para establecer la disposición de todos los elementos de la interfaz gráfica en el applet, es decir, donde estarán situados. También se encarga de establecer las condiciones iniciales y por defecto que se mostrarán al arrancar el applet. Todas las instrucciones contenidas en este método se generan automáticamente según actuamos sobre la interfaz a la hora de la programación.

- *public void mostrarVariables()) {…}*

Este método se encarga de habilitar los campos de texto donde el usuario deberá introducir las variables de entrada. Este procedimiento se invoca cada vez que se finaliza o detiene definitivamente una representación. Así el usuario solo podrá modificar las variables de entrada para iniciar una nueva representación cuando haya finalizado la anterior.

- *public void ocultarVariables()) {…}*

Este método se encarga de deshabilitar los campos de texto donde el usuario deberá introducir las variables de entrada. Este procedimiento se invoca cada vez que se inicia una representación y permanecerán deshabilitadas hasta que finalice la misma. Así el usuario solo

podrá modificar las variables de entrada para iniciar una nueva representación cuando haya finalizado la anterior.

- *public void resetProgreso(int total) {…}*

Este método se encarga de resetear la barra de progreso del applet cada vez que iniciamos una nueva representación o borramos el lienzo de dibujo. Se ocupa de definir los valores mínimo y máximo (recibido como parámetro) que tiene la barra de progreso y establecerlo en el mínimo. Dicho parámetro de entrada nos permitirá diferenciar el valor máximo que puede tomar la barra de progreso dependiendo de si se va a representar una o dos gráficas. Por lo tanto, dicho método situará el valor de la barra de progreso en el 0%.

- *public void avanzaProgreso() {…}*

Este método se encarga de aumentar el valor de la barra de progreso durante la ejecución del programa. Dicho valor aumentará en una unidad cada vez que es invocado. Además dicho método nos permitirá iniciar la reproducción del sonido de alarma cuando nos encontremos en la situación de la representación del dibujo y el barco se hunda, es decir, cuando en la representación del dibujo la gráfica se escape. También se hará la comprobación del valor actual de la barra de progreso y en caso de ser el máximo significará que hemos llegado al final de la representación, por lo cual inhibiremos o activaremos los elementos correspondientes de la interfaz para el correcto funcionamiento del programa. También se habilitarán los campos de texto de las variables de entrada para permitir cambiar los valores e iniciar una nueva representación. Finalmente, en este caso daremos por concluidos los diferentes hilos de representación y las reproducciones de sonidos, en caso de estar activas.

- *public void limpiar_hilo() {…}*

Este método se encarga de parar la ejecución de los hilos creados para las representaciones. Detiene el hilo iniciado para la representación de una gráfica o los hilos iniciados para las dos gráficas respectivamente. En caso de la representación del dibujo detiene la reproducción de los sonidos correspondientes que se encuentren activos.

- *private boolean comprobarDatos(int num) {…}*

Este procedimiento se utiliza para comprobar la corrección de los valores insertados para las diferentes variables para la representación de la/s gráfica/s. La comprobación se realizará para las variables correspondientes a la primera gráfica o para las variables correspondientes para las variables de la primera y segunda gráficas, dependiendo de la representación elegida. El parámetro de entrada de la función nos indica el número de gráficas a representar. Para cada una de dichas variables se comprobará si el valor insertado por el usuario se corresponde el formato de valor esperado, es decir, si la variable en cuestión es un número real, se comprobará que realmente el valor insertado es un número real. En caso de encontrarse algún valor erróneo, se mostrará un mensaje de error en forma de ventana indicando la variable correspondiente al error encontrado, además de reproducirse un sonido de “bip” y establecer el fondo del cuadro de texto donde está el valor erróneo de color rojo. Dichas acciones se producirán para cada uno de los errores encontrados. Además, el valor devuelto por la función “comprobarDatos” será “false”. En caso de comprobar la corrección de todos los valores

insertados, no se realizará ninguna acción y el valor devuelto por la función "comprobarDatos" será "true".

- *private void btPlayActionPerformed(java.awt.event.ActionEvent evt) {…}*

Este método se ejecuta al pulsar el botón "Play" de la interfaz del applet.

En primer lugar se comprueba si el valor de la barra de progreso es el máximo en cuyo caso la representación anterior habrá finalizado y podremos establecer la variable correspondiente indicando que el hilo de representación no está activo.

A continuación comprobaremos si la representación está pausada o si la representación está finalizada ya que este botón de "Play" se utilizará tanto para iniciar una nueva representación como para reanudar una representación pausada.

En caso de tener que reanudar una representación, simplemente reanudaremos el curso del hilo o de los hilos necesarios y en caso de ser necesario, reanudaremos la reproducción de los sonidos correspondientes. Además activaremos o inhibiremos los botones y elementos necesarios para proseguir la representación.

Si la acción requerida es la de iniciar una nueva representación, habrá que asegurarse de que está finalizada cualquier representación anterior y a continuación comprobar la corrección de los valores insertados por el usuario para las variables correspondientes. En caso de producirse algún error en la entrada de datos, se generarán los mensajes de error oportunos y el programa quedará a la espera de que el usuario corrija dichos errores. Para volver a iniciar la representación una vez esté todo correcto, el usuario deberá de volver a pulsar el botón de "Play".

Si los valores insertados por el usuario son correctos, se iniciará la representación seleccionada. Para ello, se crearán tantos objetos de "calculoGrafica" como gráficas se quieran representar (una o dos dependiendo de si hemos marcado la caja de comparar gráficas) pasándole los parámetros correspondientes a los valores de las variables insertadas por el usuario. Para iniciar la representación, deberemos iniciar el/los hilo/s necesarios llamando al método "start" de la clase "calculoGrafica".

Finalmente dependiendo de si la representación seleccionada es "Estática" o "Dinámica", se activarán o inhibirán los botones o elementos necesarios del applet.

En caso de la representación estática, se mostrará la representación final obtenida directamente, situando la barra de progreso en el máximo y dando por finalizada la representación.

En el caso de representación dinámica, la propia ejecución del programa irá estableciendo los valores de las variables que se modifiquen. Simplemente en el caso de estar seleccionada la representación de dibujo se iniciará la reproducción del sonido correspondiente.

También se deshabilitarán los campos de texto de las variables de entrada para evitar que se modifiquen sus valores hasta que no se finalice la representación actual.

- *private void btPauseActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al pulsar el botón "Pause". Se encarga de detener la ejecución que se encuentre en curso. Dicha detención o pausa consistirá en la suspensión momentánea de las acciones realizadas por los hilos activos y también la detención de las reproducciones de

sonido activas. También se activarán o inhibirán los elementos correspondientes del interfaz del applet.

- *private void btStopActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al pulsar el botón "Stop". Se encargará de detener totalmente la ejecución del applet que se encuentre en curso y activar o inhibir los elementos del interfaz correspondientes. También se habilitarán los campos de texto de las variables de entrada para permitir cambiar los valores e iniciar una nueva representación. Para ello detendrá definitivamente la ejecución de los hilos que se estuvieran ejecutando. Una vez pulsado dicho botón, no se podrá continuar con la representación anterior desde el punto en el que se encontraba, sino que ya sólo se podrá iniciar una nueva representación. Sin embargo, podremos realizar diferentes actividades sobre la representación que se acabe de detener como ampliarla, moverla, mostrar ejes, etc. Dicha representación sólo se perderá definitivamente al iniciar una nueva con el botón "Play" o al borrar dicha representación con el botón "Cortar".

- *private void barraVelocidadVelocidadStateChanged(javax.swing.event.ChangeEvent evt) {...}*

Este método se ejecuta automáticamente siempre que cambie el estado de la barra de velocidad. Simplemente se encarga de obtener el valor representado en la barra de velocidad para establecer el valor del tiempo que la ejecución del programa debe esperar entre el cálculo de los diferentes puntos de la representación, es decir, dependiendo del valor de la velocidad elegida en la barra la representación se realizará más despacio o más rápido. Dicha variable de control del tiempo se encuentra en la clase "estadoPropiedades" y se modifica desde este método.

- *private void rbEstaticaActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al seleccionar la opción de representación estática. Establece el valor correspondiente de la variable global de la clase "estadoPropiedades" para que el programa pueda utilizarla y el estado de los diferentes elementos de la interfaz del applet para proseguir con la ejecución correcta del programa. Para ello inicializa e inhibe la barra de velocidad y activa o inhibe los componentes correspondientes. También se habilitarán los campos de texto de las variables de entrada para permitir cambiar los valores e iniciar una nueva representación. En caso de tratarse de la representación del dibujo, detiene la reproducción de los sonidos que estuvieran en curso. Al seleccionar esta opción de representación estática, si la representación se encuentra en curso, directamente se mostrará el resultado final de la ejecución con sus respectivas características, y en caso de estar la representación parada, simplemente se actualizarán los valores de las diferentes variables necesarias. Para ello, la representación se repintará con las nuevas opciones seleccionadas.

- *private void rbDinamicaActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al seleccionar la opción de representación dinámica. Establece el valor correspondiente de la variable global de la clase "estadoPropiedades" para que el programa pueda utilizarla y el estado de los diferentes elementos de la interfaz del applet para proseguir con la ejecución correcta del programa. Al pasar de una representación estática a

una dinámica seleccionando dicho botón de dinámica, el lienzo de dibujo se limpia, ya que dentro de dicho método se repinta la representación.

- *private void btArribaActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al pulsar el botón de "Arriba". Se encarga de modificar el valor de la variable contenida en la clase "estadoPropiedades" que contiene la posición vertical del lienzo. En este caso modifica el valor de la variable que controla el eje de las "y" para mover la representación un valor constante hacia arriba en el lienzo. En caso de que el valor obtenido tras el movimiento sea el estado original del lienzo (centrado), se inhibirá el botón correspondiente al reseteo de posición. Finalmente se repinta la representación para mostrar el nuevo estado de la misma.

- *private void btAbajoActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al pulsar el botón de "Abajo". Se encarga de modificar el valor de la variable contenida en la clase "estadoPropiedades" que contiene la posición vertical del lienzo. En este caso modifica el valor de la variable que controla el eje de las "y" para mover la representación un valor constante hacia abajo en el lienzo. En caso de que el valor obtenido tras el movimiento sea el estado original del lienzo (centrado), se inhibirá el botón correspondiente al reseteo de posición. Finalmente se repinta la representación para mostrar el nuevo estado de la misma.

- *private void btIzqActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al pulsar el botón de "Izquierda". Se encarga de modificar el valor de la variable contenida en la clase "estadoPropiedades" que contiene la posición horizontal del lienzo. En este caso modifica el valor de la variable que controla el eje de las "x" para mover la representación un valor constante hacia la izquierda en el lienzo. En caso de que el valor obtenido tras el movimiento sea el estado original del lienzo (centrado), se inhibirá el botón correspondiente al reseteo de posición. Finalmente se repinta la representación para mostrar el nuevo estado de la misma.

- *private void btDerActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al pulsar el botón de "Derecha". Se encarga de modificar el valor de la variable contenida en la clase "estadoPropiedades" que contiene la posición horizontal del lienzo. En este caso modifica el valor de la variable que controla el eje de las "x" para mover la representación un valor constante hacia la derecha en el lienzo. En caso de que el valor obtenido tras el movimiento sea el estado original del lienzo (centrado), se inhibirá el botón correspondiente al reseteo de posición. Finalmente se repinta la representación para mostrar el nuevo estado de la misma.

- *private void btAmpliarActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al pulsar el botón de "Ampliar". Se encarga de modificar el valor de la variable contenida en la clase "estadoPropiedades" que contiene el zoom de representación en el lienzo. En este caso modifica el valor de la variable que controla el zoom para aumentarle un valor constante. En caso de que el valor obtenido tras el movimiento sea

el del zoom original del lienzo (escala inicial), se inhibirá el botón correspondiente al reseteo del zoom. Finalmente se repinta la representación para mostrar el nuevo estado de la misma.

- *private void btReducirActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al pulsar el botón de "Reducir". Se encarga de modificar el valor de la variable contenida en la clase "estadoPropiedades" que contiene el zoom de representación en el lienzo. En este caso modifica el valor de la variable que controla el zoom para reducirle un valor constante. En caso de que el valor obtenido tras el movimiento sea el del zoom original del lienzo (escala inicial), se inhibirá el botón correspondiente al reseteo del zoom. Finalmente se repinta la representación para mostrar el nuevo estado de la misma.

- *private void btResetZoomActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al pulsar el botón de "Reset Zoom". Se encarga de modificar el valor de la variable contenida en la clase "estadoPropiedades" que contiene el zoom de representación en el lienzo. En este caso modifica el valor de la variable que controla el zoom para asignarle el valor inicial. Al mismo tiempo se ocupa de inhibir dicho botón. Finalmente se repinta la representación para mostrar el nuevo estado de la misma.

- *private void btResetPosActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al pulsar el botón de "Reset Posición". Se encarga de modificar el valor de la variable contenida en la clase "estadoPropiedades" que contiene la posición de la representación en el lienzo. En este caso modifica el valor de las variables que controlan el eje de las "x" y el de las "y" para situar la representación en la posición inicial en el centro del lienzo y por lo tanto del panel de representación. Al mismo tiempo se ocupa de inhibir dicho botón. Finalmente se repinta la representación para mostrar el nuevo estado de la misma.

- *private void btcolorActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al pulsar el botón de selección del color de la/s gráfica/s. Simplemente se encarga de hacer visible la paleta de colores definida en la clase "paletaColores".

- *private void ejesActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al seleccionar la opción de mostrar u ocultar ejes. Simplemente se encarga de modificar la variable correspondiente de la clase "estadoPropiedades" dependiendo de si dicha opción está o no seleccionada. Finalmente se repinta la representación para mostrar el nuevo estado de la misma.

- *private void compararActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al seleccionar la opción de comparar gráficas. Simplemente se encarga de modificar la variable correspondiente de la clase "estadoPropiedades" dependiendo de si dicha opción está o no seleccionada. En caso de que esté seleccionada la opción de comparar gráficas, se mostrarán los paneles que contienen las variables correspondientes a las dos gráficas que se quieran representar. También se activarán o inhibirán las variables correspondientes en caso de estar seleccionada la opción de

representación con control. Si la opción de comparar gráficas no está seleccionada, simplemente se mostrará el panel que contiene las variables necesarias para la representación de una sola gráfica.

- *private void btCutActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al pulsar el botón para cortar la representación o limpiar el lienzo de dibujo. Este método se encarga de eliminar todos los datos existentes de la representación hasta el momento, es decir, que al pulsar dicho botón se borran todos los puntos calculados hasta el momento y si prosiguiera la representación, lo haría como si fuera una representación nueva iniciándola desde el punto actual. Hay tres casos diferentes dependiendo del estado de la representación en el momento de pulsar el botón:
En caso de estar la representación en curso, se borra el contenido de la representación en el lienzo de dibujo hasta el momento y se prosigue con la representación desde ese punto actual. Dicho punto se muestra como el inicial en la representación.
En caso de estar pausada la representación, se borra el contenido de la representación en el lienzo de dibujo hasta el momento y simplemente se muestra el punto en el que se ha detenido la misma.
En caso de estar finalizada la representación (o estar seleccionada la opción de estática) se borra toda la representación limpiándose el lienzo de dibujo. También se actualizará el estado de la barra de progreso situándolo en el mínimo.
Para poder mostrar el nuevo estado de representación se repinta la representación.

- *private void comboControlActionPerformed(java.awt.event.ActionEvent evt) {...}*

Este método se ejecuta al modificar la opción de sistema de representación. Se pueden seleccionar cuatro tipos de representación y este método de encarga de realizar las acciones necesarias para cada una de ellas. En concreto muestra la información en el applet de la representación seleccionada junto con su fórmula correspondiente. Además se ocupa de activar o inhibir los diferentes elementos del interfaz para que el usuario solo pueda modificar o seleccionar las acciones permitidas en cada una de ellas. Por ejemplo, si las representaciones son con control activará las variables de control de los paneles de las variables y si seleccionamos alguna de las representaciones del dibujo, se inhibirán los botones de la paleta de colores o la posibilidad de representar dos gráficas.
Estas acciones permitirán al usuario el mejor manejo de la interfaz.

Finalmente, esta clase contiene la declaración de todos los elementos que forman parte de la interfaz del applet. Dicha declaración se genera de forma automática al ir diseñando el applet y no se puede modificar directamente, sino solo a través del interfaz.

- **Clase lienzoDibujo:** La cabecera de la clase es la siguiente:

public class lienzoDibujo extends java.awt.Canvas {...}

Dicha clase heredará de la clase “Canvas”, que nos permite realizar y mostrar la representación de gráficas y trabajar con imágenes y dibujos sobre un lienzo de dibujo. Para indicar que la clase hereda de una clase se debe poner la palabra “extends” y a continuación el nombre de la clase heredada.

Esta clase llevará a cabo varias funciones importantes para el applet, entre ellas destacan las siguientes:

- Mostrar la representación seleccionada en el lienzo de dibujo, bien dibujando la/s gráfica/s calculadas, o bien controlando las diferentes imágenes necesarias para mostrar la representación del dibujo. En conclusión, se encarga de pintar sobre el lienzo todos los elementos que deben aparece en la representación.
- Generar y manejar el lienzo de dibujo, así como anclarlo y situarlo en el centro de un panel de representación.
- Cargar y controlar las diferentes imágenes y sonidos utilizados en las representaciones.

En cuanto a los elementos de la clase “lienzoDibujo”, podemos destacar tanto propiedades como métodos. Las propiedades declaradas representan a las clases del programa que se utilizan en esta clase así como variables necesarias para controlar la representación de las gráficas o dibujos correspondientes. También se definen variables para controlar el lienzo de dibujo en el cual se plasmarán las representaciones y las diferentes imágenes y sonidos que forman parte del mismo. Dichas imágenes se corresponden con la de un día soleado o un día lluvioso y oscuro. Los sonidos se corresponden con el de las olas del mar y el de una sirena de barco. Estas imágenes y sonidos se usarán según corresponda. En cuanto a métodos, se explican a continuación:

- *public lienzoDibujo(estadoPropiedades est, ventanaPrincipal v) {...}*

Este es el primero y más importante de los métodos encontrados en la clase, ya que es el constructor de la misma. La declaración y redefinición de este método se realiza para inicializar las variables utilizadas por la clase, así como para establecer el color de fondo del lienzo de dibujo y cargar las imágenes y sonidos utilizados a lo largo de la representación.
Los parámetros recibidos se corresponden con los objetos creados de las otras clases del proyecto y permiten asignar dichos objetos a las variables correspondientes de la clase.

- *public void nuevo(double x, double y, int id) {...}*

Este método se encarga de vaciar la lista que contiene los puntos de las gráficas a representar y seleccionar el punto de origen de la misma para su posterior representación en el lienzo de dibujo. Este punto se recibe como parámetro del procedimiento. También se recibe como parámetro el hilo al que corresponde ese punto inicial. Así se puede diferenciar si se trata de un punto de la primera gráfica o de la segunda en caso de representarse ambas. Este procedimiento se usa solamente en el caso de representación dinámica.

- *public void calculaDinamico(double x, double y, int id) {...}*

Este método corresponde a la representación dinámica y se encarga de asignar los valores de los puntos recibidos como parámetro a las variables globales de la clase que contienen dicha información. Al igual que en el procedimiento anterior se recibe un parámetro de entrada para diferenciar el hilo al cual corresponde el punto, es decir, si se trata de un punto de la primera o de la segunda gráfica. Al final del procedimiento se realiza la llamada al procedimiento correspondiente de la clase "ventanaPrincipal" para aumentar el valor de la barra de progreso y por lo tanto de la representación. La barra de progreso se corresponde con el número de iteraciones por lo cual avanzarán simultáneamente. Una vez se tienen los puntos recibidos asignados en las variables globales de la clase, se realiza la llamada oportuna para su representación en el lienzo de dibujo.

- *public void calculaEstatico(Vector<Point2D> l, int id) {...}*

Este método corresponde a la representación estática y recibe como parámetro de entrada la lista que contiene todos los puntos calculados en la representación. Simplemente se encarga de asignar dichos valores a la variable global de la clase que contiene los puntos a representar. Al igual que los procedimientos anteriores, recibe el identificador de la gráfica a la que corresponde la lista de puntos. Una vez tenemos la lista completa de puntos, se realiza la llamada oportuna para su representación en el lienzo de dibujo.

- *public synchronized void pintarGrafica(Graphics2D g) {...}*

Este método se encarga de dibujar la representación elegida en el lienzo de dibujo, por lo cual es el más importante de la clase. Será invocado desde el método principal "paint()" cada vez que se quiera pintar el lienzo de dibujo, por lo cual deberemos de controlar si se tiene que dibujar de nuevo o si por el contrario no es preciso ejecutar el código de este método. Para ello comprobamos si el hilo de ejecución está en curso o si se trata de la representación estática. En caso contrario no se ejecutará nada, ya que no hay elementos nuevos que dibujar, sino que ya está mostrándose la representación deseada.
Este procedimiento comprobará si la representación seleccionada es dinámica o estática.
En caso de ser dinámica se dibujan los puntos contenidos en la lista correspondiente. Hay que tener en cuenta que esta lista de puntos irá aumentando según avance la ejecución del programa por lo cual cada vez contendrá más puntos, y por lo tanto, cada vez se dibujarán más puntos. Así conseguimos una representación progresiva y animada. También hay que indicar que los puntos contenidos en esta lista no siempre se dibujarán explícitamente sobre el lienzo de dibujo, sino que en caso de querer realizar la representación del dibujo, dichos puntos serán sustituidos por la imagen del barco en la posición correspondiente. Para ello llamaremos al método encargado de hacer esta transformación de los puntos. Al igual que se hace para una gráfica, cuando la representación seleccionada corresponde a dos gráficas, el proceso es exactamente igual al anterior, pero dibujando los puntos de las dos gráficas simultáneamente. Para la mejor visibilidad de la representación gráfica, se incluirá en el lienzo de dibujo el punto inicial desde donde se ha comenzado a dibujar la gráfica y un círculo indicativo para el punto actual que se está dibujando. Las gráficas se dibujan en el color seleccionado por el usuario en la paleta y por defecto será rojo y azul, para la primera y segunda gráfica respectivamente.

En caso de ser estática el proceso es similar al de dinámica pero teniendo en cuenta que solo se llama una única vez a este método por cada representación, es decir, que cuando se dibuje la representación, la lista de puntos será directamente la lista completa.
Es muy importante diferenciar la representación de las gráficas o del dibujo, por lo cual o bien se dibuja el punto o bien se dibuja la imagen resultante a dicho punto, pero no ambas a la vez.
Hay que indicar también que los puntos no se pueden representar directamente en el lienzo de dibujo según los calculamos, sino que siempre habrá que realizar el cálculo correspondiente a la escala de representación, es decir, que el punto a dibujar será un múltiplo del punto real por la escala de representación.
Para evitar que se produzcan discontinuidades en la representación de dos puntos consecutivos de la lista, se ha optado por dibujar líneas en lugar de puntos. Esto nos asegura la continuidad de la representación.

- *private void pinta_barco(Graphics g, double xaux, double yaux) {...}*

Este método nos permitirá realizar la representación del dibujo. Se llama desde el procedimiento anterior cuando se desee mostrar el dibujo representativo en lugar de la gráfica correspondiente. La imagen dibujada y su posición dependen de los parámetros de entrada ya que el barco rotará o se desplazará en función de los mismos.
En primer lugar se realiza el escalado de las imágenes que se mostrarán en el lienzo de dibujo para ajustarlas al tamaño del mismo. A continuación se calcula el grado de rotación del barco, el cual depende de los parámetros de entrada, y más en concreto de la "x". Si los valores de estas variables son correctos, se mostrará la representación del barco con la rotación calculada sobre la imagen de fondo de un día soleado.
En caso de que alguno de los parámetros sean infinitos, es decir, que la gráfica calculada se escape, o el ángulo de rotación calculada sea mayor de 90 grados en valor absoluto, el barco se hundirá. Para ello el ángulo de rotación avanzara hasta lograr que la imagen haya girado 180 grados y esté dada la vuelta. También se modificará la imagen de fondo que pasará a ser la de un día oscuro y lluvioso. Al mismo tiempo la posición del barco irá descendiendo según el eje de las "y" simulando el hundimiento. Si la representación es estática, la imagen aparecerá dada la vuelta y hundida totalmente.
En caso de una representación estática se mostrará el resultado del último punto calculado, mientras que si se trata de una representación dinámica se irán mostrando los resultados de los diferentes puntos por lo cual se obtiene una imagen animada.
Mientras el barco esté navegando se reproducirá el sonido de las olas del mar y en caso de que se hunda, se sumará el sonido de la alarma de peligro o sirena del barco.

- *private void pintarEjes(int h, int w, Graphics g) {...}*

Este método se encarga de dibujar los ejes de coordenadas en el lienzo de dibujo en las representaciones gráficas. Para ello lo primero que se hace es establecer el color en que se mostrarán tanto los ejes, como el punto inicial y como las diferentes marcas a lo largo de los ejes indicando el valor correspondiente.
Para ello en primer lugar se dibujan las líneas de los ejes de coordenadas, tanto el eje de las "x" como el de las "y". Dichos ejes están perfectamente centrados en el lienzo de dibujo y se muestra el punto central (0,0).

A continuación se van recorriendo dichos ejes de coordenadas para ir marcando los diferentes puntos que nos indicarán el valor de la coordenada en dicho punto. En cada una de las unidades señalizadas se mostrará una línea perpendicular al eje y el valor que le corresponde formateado. Evidentemente dichos valores mostrados dependerán del escalado de la representación y por lo tanto el valor mostrado en un punto dependerá del valor real que represente dependiendo de la escala elegida.

- *protected void pinta_centro(Graphics g) {...}*

Este método es llamado desde el procedimiento anterior y simplemente se encarga de mostrar el punto inicial o (0,0) en el centro de la representación. Este punto se mostrará en el mismo color que los ejes de coordenadas.

- *private void pintaPunto(Graphics2D g, Color c, double q, double w) {...}*

Este método será llamado desde el procedimiento "pintarGrafica" y se encarga de dibujar un circulito correspondiendo con el punto actual que se esté dibujando en la representación. Este punto se recibe como parámetro y el color de representación será el mismo que la gráfica de la que forma parte. Simplemente nos sirve para ver más fácilmente el curso de la representación dinámica o cual ha sido el último punto dibujado.

- *private void pintaPuntoInicial(Graphics2D g, Color c, double q, double w) {...}*

Este método será llamado desde el procedimiento "pintarGrafica" y se encarga de mostrar en el lienzo de dibujo y entre paréntesis el valor del punto inicial en el que se ha comenzado la representación. En caso de cortarse la representación dinámica, dicho punto será el primero calculado tras el corte. Este punto se mostrará junto con el punto al que se corresponde y nos permite conocer explícitamente en qué punto se ha iniciado la representación. Para ello, este punto se mostrará con su valor real, es decir, sin aplicarle ninguna escala, y por supuesto formateado para que tenga una longitud adecuada. El color de dicho punto será el mismo que el de la gráfica a la que corresponde.

- *public BufferedImage cargarImagen(String nombre) {...}*

Este método se encarga de cargar en el programa una imagen de disco y en caso de no encontrarla o producirse algún error, devolver el mensaje correspondiente. Se usa para poder cargar las imágenes tanto del barco como de los fondos cuando se trata de representaciones con dibujo. Para su buen funcionamiento la imagen buscada debe encontrarse dentro del directorio del proyecto.

- *public void paint(Graphics g) {...}*

Este es el método principal para las representaciones gráficas. Se encarga de manejar el lienzo de dibujo y su correcto funcionamiento así como plasmar sobre él las representaciones o dibujos deseados. En primer lugar se creará el lienzo de dibujo con su tamaño correspondiente y se centrará en el panel de dibujo. En el centro de dicho lienzo se colocará el centro inicial de representación, es decir el punto de coordenadas (0,0). A continuación se llamará al método principal "pintarGrafica" para que se dibuje sobre el mismo la

representación deseada. También se encarga de controlar si se deben o no mostrar los ejes de coordenadas.

- *public void repaint() {...}*

Este método se utiliza para repintar la representación. Será el método invocado desde la mayoría de los métodos del proyecto para realizar la representación y se ocupa básicamente de repintar o actualizar la representación mostrada. Este método será utilizado cada vez que se quiera mostrar un nuevo punto en la representación o actualizar la vista del mismo al realizar alguna acción como ampliar, mover, etc.

- *public void update(Graphics g) {...}*

Este método llama al método "paint()" y sirve para actualizar la representación mostrada en el lienzo de dibujo.
Este método será utilizado cada vez que se quiera mostrar un nuevo punto en la representación o actualizar la vista del mismo al realizar alguna acción como ampliar, mover, etc.

- *public void limpiar() {...}*

Este método se invoca cuando se pulsa el botón de cortar en la interfaz del applet y se encarga básicamente de vaciar las listas que contienen los puntos de la representación.
Se vacían las listas correspondientes a las gráficas que se están representando.
Así se consigue que las listas queden vacías y cuando se prosiga con la ejecución del programa, la representación solo se realice a partir del siguiente punto. Esto se utiliza para limpiar la pantalla una vez pausada o parada la representación o para cortar la representación anterior e iniciar una nueva a partir del punto actual en las representaciones dinámicas.

- *public Image getBackbuffer() {...}*
- *public void setBackbuffer(Image backbuffer) {...}*

Estos métodos nos permiten devuelve y establecer el "backbuffer" o imagen sobre la que se asentará la representación, respectivamente. Se tratará de una imagen que actúa como lienzo de dibujo y sobre la cual se mostrarán el resto.

- *public Color getColor() {...}*
- *public void setColor(Color c) {...}*

Estos procedimientos se utilizan dentro de la clase para obtener y establecer el color en el que el usuario desea representar la primera gráfica, respectivamente. El primer método será llamado desde la paleta de colores y el color seleccionado en la misma por el usuario se pasará a esta clase a través de dicho método.

- *public Color getColor2() {...}*
- *public void setColor2(Color c2) {...}*

Estos procedimientos permiten a la clase obtener y establecer el color en el que el usuario desea representar la segunda gráfica, respectivamente. El segundo método será llamado desde

la paleta de colores y el color seleccionado en la misma por el usuario se pasará a esta clase a través de dicho método.

- **Clase calculoGrafica:** La cabecera de la clase es la siguiente:

  public class calculoGrafica extends java.lang.Thread{...}

Dicha clase heredará de la clase "Thread", ya que es necesario utilizar hilos para realizar los cálculos concurrentemente al transcurso del programa y del resto de funciones. Esto permite poder realizar simultáneamente las acciones deseadas sobre la interfaz y el programa mientras se realizan los cálculos. Si no se utilizaran estos hilos, el programa se bloquearía, ya que solo se podría realizar una tarea cada vez. A su vez, todos los métodos que se ejecutan de manera concurrente contienen en su declaración la palabra "synchronized". Para indicar que la clase hereda de una clase se debe poner la palabra "extends" y a continuación el nombre de la clase heredada.

Esta clase llevará a cabo varias funciones importantes para el applet, entre ellas destacan las siguientes:

- Realizar los cálculos necesarios para obtener los puntos necesarios para realizar las representaciones.
- Pasar los datos calculados a la clase "lienzoDibujo" para su posterior representación. En el caso de la representación estática también se encargará de generar la lista o vector de puntos directamente para su posterior representación en la clase correspondiente.

En cuanto a los elementos de la clase "calculoGrafica", podemos destacar tanto propiedades como métodos. Las propiedades declaradas representan a las clases del programa que se utilizan para el control del cálculo de los puntos y a la clase que se encargará en sí de la representación de los puntos calculados y a la cual se deben pasar los mismos. En cuanto a métodos, se explican a continuación:

- *public calculoGrafica(lienzoDibujo l, double x, double y, double t, double a, double b, double d, double F, double w, double ep, double o, estadoPropiedades est, int id) {...}*

Este es el primero y más importante de los métodos encontrados en la clase, ya que es el constructor de la misma. La declaración y redefinición de este método se realiza para iniciar las acciones y variables necesarias al crear un objeto de dicha clase desde cualquier otro punto del programa.
Todas las variables inicializadas en este constructor se reciben como parámetro y se corresponden con los valores insertados por el usuario, así como el indicador del hilo a ejecutar dependiendo de la gráfica correspondiente y las variables que identifican a las clases del proyecto utilizadas en esta clase.

- *private double ecuacion1(double valor1) {...}*

Este método se encarga de calcular el valor de la primera ecuación auxiliar necesaria para realizar los cálculos. Dicho valor se corresponderá matemáticamente con el valor de la derivada primera de "x" ("y"). Como indica el método de descomposición de funciones que se va a utilizar, el valor devuelto será el mismo que el recibido como parámetro.

- *private double ecuacion2(double valorx, double valory, double valort) {...}*

Este método se encarga de calcular el valor de la segunda ecuación auxiliar necesaria para realizar los cálculos. Dicho valor se corresponderá matemáticamente con el valor de la derivada segunda de "x" (derivada segunda de "y"). El valor devuelto se calcula en función de los parámetros de entrada de la función y utilizando las variables correspondientes de la clase. Hay que tener en cuenta si la representación deseada es con o sin control ya que en el primer caso deberemos realizar los cálculos teniendo en cuenta las variables de control.

- *public synchronized Point2D calcula() {...}*

Este método es el que realiza los cálculos de la función del oscilador de Helmholtz en sí. Para ello implementa el método de Runge Kutta para aproximar la solución y obtener el valor correspondiente. El valor devuelto por esta función es un punto, por lo cual tiene el valor de "x" y de "y". Este método se ejecutará múltiples veces al ser invocado desde un bucle, por lo cual, al no recibir parámetros de entrada, modifica directamente los valores de las variables globales de la clase. En caso de tratarse de una representación estática, únicamente devuelve el punto calculado, pero en caso de tratarse de una representación dinámica, además de calcular el punto y devolverlo, llama al método correspondiente de la clase "lienzoDibujo" para la representación del punto calculado, pasándole dicho punto. También tiene en cuenta el hilo al cual corresponde el cálculo realizado a la hora de pasar el punto a la clase de "lienzoDibujo".
Finalmente hay que indicar que en la representación dinámica hay que tener en cuenta la velocidad de representación seleccionada. Para controlar este aspecto disponemos de las sentencias necesarias para que el hilo espere el tiempo correspondiente.

- *public synchronized void run() {...}*

Este es el método principal de los hilos y se ejecutan de manera continua mientras el hilo en cuestión esté activo. Al iniciar su ejecución inicializa a vacío la lista con el conjunto de puntos a representar e inserta el punto inicial donde se comenzará la representación y que se obtiene de las variables globales de la clase. Dichas variables que representan el punto inicial obtienen su valor en el constructor de la clase y se corresponde con el valor insertado por el usuario en el interfaz del applet.
A continuación se calculan todos los puntos necesarios mediante un bucle ejecutado continuamente hasta completar el número de iteraciones o cálculos especificados por el usuario. Todos los puntos calculados se van insertando en la lista correspondiente. Dicho bucle, a pesar de ejecutarse continuamente, contiene una condición en su interior que evita que se realicen nuevos cálculos mientras la representación está pausada o parada. Así conseguimos que la ejecución y cálculo de puntos solo se realice cuando la representación está en curso.

Una vez concluida la ejecución del bucle por haber completado el número de iteraciones y en caso de tratarse de la representación estática, se pasa la lista con los puntos a la clase "lienzoDibujo" para su representación en el lienzo. En el caso de tratarse de la representación dinámica no es necesario ya que dichos puntos se han ido pasando en el método de cálculo anterior.

- *public synchronized boolean isCorrer() {...}*

Este método devuelve un valor booleano que indica si la representación está en curso y por lo tanto se debe de proseguir calculando nuevos puntos. En caso de estar pausada o parada la representación, dicho método devolverá "false" y en caso contrario devolverá "true". Gracias a este método podemos controlar las sentencias ejecutadas en el bucle del método anterior "run".

- *public synchronized void setCorrer(boolean correr) {...}*

Este método permite establecer si se deben seguir calculando nuevos puntos o no gracias al parámetro recibido. En concreto, dicho método modificará la variable correspondiente que indica si la representación está en curso o si por el contrario está pausada o parada. Si la representación está activa establecerá el valor "true" y si está pausada o parada establecerá el valor "false".

- *public synchronized void correr() {...}*

Este método será invocado al iniciar o reanudar la representación desde la clase ventanaPrincipal y se encarga de establecer el valor "true" en la variable que controla el desarrollo de los nuevos cálculos de puntos.

- *public synchronized void parar() {...}*

Este método será invocado al pausar o parar la representación desde la clase ventanaPrincipal y se encarga de establecer el valor "false" en la variable que controla el desarrollo de los nuevos cálculos de puntos.

- **Clase estadoPropiedades:** La cabecera de la clase es la siguiente:

public class estadoPropiedades {...}

Dicha clase no hereda de ninguna otra clase ya que simplemente se utilizará para contener las variables globales del programa y contendrá os métodos necesarios para poder modificar su valor y acceder al mismo desde cualquier otra clase del programa.
Esta clase llevará a cabo varias funciones importantes para el applet, entre ellas destacan las siguientes:

- Contener las propiedades o variables globales del programa para poder acceder a ellas desde cualquier clase del programa.
- Contener los métodos necesarios para acceder a dichas propiedades y modificar globalmente su valor.

En cuanto a los elementos de la clase “estadoPropiedades”, podemos destacar tanto propiedades como métodos. Las propiedades declaradas serán las necesarias para controlar las variables globales a todas las clases del proyecto así como alguna variable y constante que usaremos para controlar la corrección de las mismas. En cuanto a los métodos, se explican a continuación:

- *public estadoPropiedades() {...}*

Este es el primero y más importante de los métodos encontrados en la clase, ya que es el método constructor de la misma. La declaración y redefinición de este método se realiza para iniciar las acciones necesarias al crear un objeto de dicha clase desde cualquier otro punto del programa. Simplemente se encarga de inicializar las principales variables globales del programa como son la escala inicial de representación o el zoom, así como definir cuál es la posición inicial del lienzo de dibujo dentro del panel de representación.

- *public void resetPantalla() {...}*

Este método será llamado cuando pulsemos el botón de resetear la posición en la interfaz del applet y nos permitirá que el lienzo de dibujo se sitúe en la posición inicial, es decir, en el centro del applet.

- *public double getEscala() {...}*
- *public void setEscala(int esc) {...}*

Estos métodos nos permiten devolver y establecer el valor de la escala de representación seleccionada, respectivamente. Esta escala es necesaria para poder mostrar la representación en el lienzo de dibujo con el tamaño y resolución adecuada al tamaño del mismo.

- *public int getUnidad() {...}*

Este método permite conocer el valor de la unidad de la escala de representación, es decir, el valor por el que debemos de multiplicar un valor dado para pasarlo a la escala de representación que deseamos.

- *public double getEscalaEjes() {...}*

Este método se encarga de devolver el valor correspondiente a la escala de representación para los ejes de coordenadas del lienzo de dibujo. Este valor será el número que nos permitirá calcular el valor de los ejes de coordenadas (“x” e “y”) para adaptarlos a nuestra representación.

- *public double getCentro_x() {...}*
- *public void setCentro_x(double c_x) {...}*

Estos métodos nos permiten devolver y establecer el valor de la posición del eje de las “x” en la representación, respectivamente. Dicho valor se corresponderá con el desplazamiento horizontal que hayamos efectuado a la representación. En el caso del estado inicial, dicho valor se corresponderá con el centro del lienzo de dibujo.

- *public double getCentro_y() {...}*
- *public void setCentro_y(double c_y) {...}*

Estos métodos nos permiten devolver y establecer el valor de la posición del eje de las "y" en la representación, respectivamente. Dicho valor se corresponderá con el desplazamiento vertical que hayamos efectuado a la representación. En el caso del estado inicial, dicho valor se corresponderá con el centro del lienzo de dibujo.

- *public void moverCentro_x(double m_x) {...}*
- *public void moverCentro_y(double m_y) {...}*

Estos métodos nos permiten desplazar horizontal y verticalmente la representación en el lienzo de dibujo, respectivamente. El desplazamiento se producirá según el eje de las "x" o las "y" y su valor se recibe como parámetro. Simplemente se encarga de aumentar o reducir el valor de la posición del eje de las "x" o de las "y" para desplazar la representación hacia la izquierda, derecha, arriba o abajo del lienzo de dibujo, según corresponda.

- *public double getCENTRADO_INICIAL_X() {...}*
- *public double getCENTRADO_INICIAL_Y() {...}*

Estos métodos nos permiten conocer el valor inicial de la posición del eje de las "x" y de las "y", respectivamente. Dichos valores están contenidos en constantes, por lo cual estos métodos simplemente nos devuelven los valores de las mismas.

- *public void resetZoom() {...}*

Este método será llamado cuando pulsemos el botón de resetear el zoom en la interfaz del applet y nos permitirá que el lienzo de dibujo se muestre con el zoom inicial, es decir, que se muestre la representación sin ampliar ni reducir. Por lo tanto, además de resetear el valor de la variable del zoom, se encarga de asignar el valor inicial a la variable que controla la escala de representación.

- *public void reduceZoom() {...}*
- *public void aumentaZoom() {...}*

Estos métodos se encargan de reducir y ampliar el zoom de representación, respectivamente. Este zoom se reducirá o aumentará de forma constante por lo cual simplemente deberemos reducirlo o aumentarlo en un valor constante, que en este caso es una unidad. Al modificar el valor del zoom, se modifica también el valor de la nueva escala de representación en función del valor del mismo. También se controla el caso en el que el zoom tome el valor de 0, en cuyo caso se asigna un nuevo valor al mismo para que la ejecución no se vea afectada.

- *public int getZoom() {...}*
- *public void setZoom(int zoom) {...}*

Estos métodos nos permiten devolver y establecer el valor del zoom en la representación actual.

- *public int getZoomInicial() {...}*

Este método devuelve el valor del zoom inicial de la representación, es decir, el valor inicial por defecto del zoom. Dicho valor se encuentra en una constante por lo cual simplemente se encarga de devolver dicho valor.

- *public int getTiempo() {...}*
- *public void setTiempo(int tiempo) {...}*

Estos métodos nos permiten devolver y establecer el valor de tiempo que debemos esperar entre los cálculos durante la ejecución del programa, respectivamente. Este valor dependerá del valor indicado por la barra de velocidad. Cuanto mayor sea la velocidad contenida en la barra de velocidad, menor tiempo de espera se tendrá. Se considera que el valor devuelto en la barra de velocidad se corresponde con el tiempo de espera. Además se debe tener en cuenta que el valor del tiempo no puede ser 0.

- *public boolean isEstatico() {...}*
- *public void setEstatico(boolean estatico) {...}*

Estos métodos nos permiten devolver y establecer un valor booleano que indica si la representación seleccionada es estática o dinámica, respectivamente.

- *public int getIteraciones() {...}*
- *public void setIteraciones(int iteraciones) {...}*

Estos métodos nos permiten devolver y establecer el número de iteraciones o pasos que se van a llevar a cabo en la representación, respectivamente. Este valor se usa para conocer el número de cálculos que se van a realizar para obtener la obtener la representación deseada. Dicho valor dependerá del tiempo introducido por el usuario

- *public boolean isHayDosGraficas() {...}*
- *public void setHayDosGraficas(boolean hayDosGraficas) {...}*

Estos métodos nos permiten devolver y establecer, respectivamente, un valor booleano que indica si se ha seleccionado la opción de representación de dos gráficas o si solo se desea representar una. En caso de que sean dos las gráficas seleccionadas, devolverá “true” y en caso de que solo esté seleccionada la representación de una, devolverá “false”.

- *public boolean isEjes() {...}*
- *public void setEjes(boolean ejes) {...}*

Estos métodos nos permiten devolver y establecer, respectivamente, un valor booleano que indica si se ha seleccionado la opción de mostrar u ocultar los ejes de coordenadas en la representación. En caso de que haya que dibujarlos, devolverá “true”, y en caso contrario devolverá “false”.

- *public boolean isConControl() {...}*
- *public void setConControl(boolean conControl) {...}*

Estos métodos nos permiten devolver y establecer, respectivamente, un valor booleano que indica si se va a realizar la representación teniendo en cuenta las variables de control o

no. Dicho valor es necesario conocerlo ya que si seleccionamos las representaciones con control, se tendrán en cuenta las variables de control en la fórmula de cálculo, y sino no. En caso de que se seleccione el control, devolverá “true”, y en caso contrario devolverá “false”. Esta opción de control es necesaria para evitar el escape de las gráficas obtenidas con la fórmula sin control.

- *public boolean isConDibujo() {...}*
- *public void setConDibujo(boolean conDibujo) {...}*

Estos métodos nos permiten devolver y establecer, respectivamente, un valor booleano que indica si se va a realizar la representación de las gráficas o la representación del dibujo representativo. Dicho valor es necesario conocerlo ya que se inhibirán o activarán diferentes elementos del interfaz del applet dependiendo de la opción seleccionada para el correcto manejo del programa. En caso de que se elija la representación del dibujo, devolverá “true”, y en caso contrario devolverá “false”.

- **Clase paletaColores:** La cabecera de la clase es la siguiente:

public class paletaColores extends javax.swing.JDialog {...}

Dicha clase heredará de la clase JDialog, ya que lo que se querrá implementar será una ventana de diálogo para interactuar con el usuario. En este caso se desea mostrar una paleta de colores para que el usuario pueda seleccionar el color deseado de representación. Para indicar que la clase hereda de una clase se debe poner la palabra “extends” y a continuación el nombre de la clase heredada.

Esta clase llevará a cabo varias funciones importantes para el applet, entre ellas destacan las siguientes:

- Mostrar toda la gama de posibles colores a seleccionar para la representación. Se podrá elegir entre tres pestañas dentro de la ventana según el formato en que se prefiera que se nos muestre la paleta de colores.
- Permite seleccionar el color elegido por el usuario y utilizarlo en la representación pasándolo a la clase lienzoDibujo

En cuanto a los elementos de la clase “paletaColores”, podemos destacar tanto propiedades como métodos. La única propiedad declarada representa el objeto de la clase “lienzoDibujo” para poder modificar directamente el color en la representación del applet. En cuanto a los métodos, se explican a continuación:

- *public paletaColores(java.awt.Frame parent, boolean modal, lienzoDibujo l) {...}*

Este es el primero y más importante de los métodos encontrados en la clase, ya que es el método constructor de la misma. La declaración y redefinición de este método se realiza para iniciar las acciones necesarias al crear un objeto de dicha clase desde cualquier otro punto del programa. Simplemente se encarga de iniciar el nuevo formulario de ventana inicializando sus

componentes y recibir la variable que contiene el lienzo de dibujo al cual queremos pasarle el color seleccionado en esta ventana.

- *private void initComponents() {...}*

  Este método se llama desde el constructor de la clase y simplemente se encarga de inicializar todas las componentes del interfaz del applet. Este código se genera automáticamente al diseñar la internar durante la programación.

- *public void contolBotones(estadoPropiedades est) {...}*

  Este método recibe como parámetro una variable con la clase "estadoPropiedades" para comprobar si se están representando una o dos gráficas y activa o inhibe los botones correspondientes para la selección del color para cada una de ellas. En caso de representarse una gráfica activa solo el botón de selección de la primera gráfica y, en caso de representarse dos, activa los dos botones para la selección del color de cada una.

- *private void btOkActionPerformed(java.awt.event.ActionEvent evt) {...}*

  Este método se ejecuta al pulsar el botón de "Gráfica 1". Se encarga de pasar el color seleccionado a la clase que controla el lienzo de dibujo para poder realizar la representación de la primera gráfica con el color seleccionado. Finalmente se repinta el lienzo de dibujo para mostrar el nuevo estado de la representación.

- *private void btOk2ActionPerformed(java.awt.event.ActionEvent evt) {...}*

  Este método se ejecuta al pulsar el botón de "Gráfica 2". Se encarga de pasar el color seleccionado a la clase que controla el lienzo de dibujo para poder realizar la representación de la segunda gráfica con el color seleccionado. Finalmente se repinta el lienzo de dibujo para mostrar el nuevo estado de la representación.

Finalmente esta clase contiene la declaración de todos los elementos que forman parte de la interfaz del formulario. Dicha declaración se genera de forma automática al ir diseñando la ventana de diálogo y no se puede modificar directamente, sino solo a través del interfaz.
Una vez seleccionados los colores deseados, se podrá cerrar la ventana de diálogo pulsando la "X" de la parte superior derecha de la misma, al igual que si se cerrara una ventana normal de Windows.

3.4. Modo de uso

El programa creado en este Proyecto de Fin de Carrera, al ser un applet, se puede ejecutar desde un navegador Web. Una vez ejecutado, se mostrará la interfaz del applet para poder interactuar con él.

Esto es debido a que la primera vez que ejecutamos el applet, se ejecuta el método init() el cual se encarga de crear los objetos necesarios y de inicializar todos los componentes. De esta forma se consigue cargar la interfaz del applet.

Una vez que se ha cargado e iniciado el applet, podremos comenzar a trabajar con él y a generar las representaciones deseadas, tantas veces como queramos. Las posibles representaciones a generar se encuentran “Gráfica sin control”, “Gráfica con control”, “Dibujo sin control” y “Dibujo con control”. La opción seleccionada por defecto en el JComboBox del panel de control es “Gráfica sin control”. Para poder cambiar el tipo de representación, simplemente deberíamos pulsar la pestaña de dicho JComboBox y seleccionar la opción deseada. Otro aspecto a destacar es que para que podamos cambiar el tipo de representación, la ejecución del applet debe estar parada.

El resto de opciones de representación también pueden modificarse al inicio de la representación actuando sobre el elemento que lo controla. Por defecto, la representación será de una sola gráfica, estática y no se mostrarán los ejes de coordenadas. Hay que indicar que para el correcto funcionamiento del applet y para evitar que el usuario intente realizar acciones no permitidas, todos los elementos del applet se habilitarán o deshabilitarán según las necesidades del propio applet y los requerimientos de las representaciones deseadas.

Para iniciar la representación de cualquiera de las gráficas se deberá pulsar el botón con el icono de “Play”. Los botones de “Pausa” y “Stop” estarán deshabilitados por defecto al inicio de cualquier representación. A lo largo de la ejecución, estos tres botones funcionales se deshabilitarán o habilitarán según corresponda. Para pausar la representación deberemos pulsar el botón de “Pause” y para reanudarla el botón de “Play” de nuevo. Para detenerla definitivamente deberemos pulsar el botón de “Stop”. Todos los botones anteriormente mencionados se encuentran en el panel de “Función”.

En primer lugar deberemos de fijarnos en el panel que contiene los parámetros de entrada. Para dichos parámetros se muestra un valor por defecto, por lo cual deberemos de modificarlos para cumplir las condiciones de la representación que queremos generar. En caso de encontrarse un error en los valores insertados por el usuario, el programa mostrará un mensaje de error indicando el fallo existente. También se reproducirá un sonido de aviso. No se podrá iniciar ninguna representación mientras exista algún error en la entrada de los valores. Los valores a insertar dependerán de la representación que queremos (con o sin control y una o dos gráficas). En caso de no ser necesario insertar algún valor de parámetro, el campo correspondiente al mismo permanecerá deshabilitado.

Podremos elegir entre la representación de una sola gráfica o de dos, para lo cual deberemos marcar la casilla “Comparar gráficas” y a continuación insertar los valores deseados para cada una de ellas. En el caso de representaciones del dibujo, no se ofrecerá esta posibilidad ya que solo se podrá representar el dibujo para una órbita.

La representación, tanto de las gráficas como del dibujo, puede ser dinámica o estática. La representación estática hace que se muestre la gráfica o el dibujo directamente, es decir, como quedaría reflejada en su estado final. Por otro lado, la representación dinámica representa la gráfica o el dibujo mediante una animación. Durante la ejecución, al cambiar de estática a dinámica la representación se borrará, y al pasar de dinámica a estática, se dibujará

la representación final correspondiente. Para elegir entre uno de los dos tipos de reproducción se deberá hacer “clic” en el JRadioButton “Estática” o en el JRadioButton “Dinámica” del panel de “Representación”. Dichas opciones son excluyentes y siempre habrá un tipo de representación seleccionado. Por otro lado, la representación dinámica representa la gráfica o el dibujo mediante una animación. Dicha representación se puede pausar, pulsando el botón de “Pause” del panel de “Función” y para reanudar la reproducción el usuario deberá volver a pulsar el botón de “Play”.

El color de representación de la gráfica o gráficas también está por defecto. Podemos modificar los colores de representación en cualquier momento y las veces que se deseen, simplemente pinchando en el botón correspondiente y seleccionando el nuevo color en la ventana de diálogo que se muestra. Para las representaciones del dibujo, esta opción permanecerá deshabilitada al no ser necesaria.

La velocidad de representación sólo será modificable cuando la representación sea dinámica y su valor por defecto es el mínimo. Durante la ejecución dicha velocidad puede ser modificada desplazando la barra de velocidad hacia la izquierda o derecha según queramos reducirla o aumentarla respectivamente.

También podremos cortar las gráficas en cualquier momento de su reproducción dinámica mediante el botón de “Cortar” en la parte derecha del applet. Mediante este botón también podremos limpiar el lienzo de dibujo y por lo tanto la representación mostrada en cualquier momento, incluso cuando la ejecución esté parada.

Para mejorar la visualización de las representaciones, podremos mover y ampliar o reducir la representación dentro del lienzo de dibujo con sus respectivos botones. Todas estas acciones pueden ejecutarse con la representación en curso o con la representación pausada o parada.

El applet también nos ofrece la posibilidad de mostrar u ocultar los ejes de coordenadas en cualquier momento de la representación de las gráficas. Para ello bastará con seleccionar o deseleccionar el elemento correspondiente. Esta posibilidad no tiene cabida cuando la representación seleccionada es la del dibujo.

La utilidad de la barra de progreso es la de mostrar el progreso de las representaciones. Por lo tanto al iniciar una nueva, tendrá el valor mínimo de 0% y cuando se finaliza o detiene una representación tendrá el valor de 100%. Este valor irá aumentando progresivamente según el avance de la representación.

Por último, el panel que hay encima del panel de dibujo alojará un campo de texto. Dicho campo de texto dará información al usuario del tipo de representación seleccionada, junto con su correspondiente fórmula de cálculo.

3.5. Movimientos representativos del sistema

En esta sección se mostrarán algunas imágenes correspondientes a diferentes ejecuciones llevadas a cabo mediante el applet. En dichas ejecuciones se mostrarán diferentes representaciones y con diferentes opciones seleccionadas.

- En la *Figura 23* se presenta la ejecución de una representación estática de una sola gráfica con control. La representación está finalizada y no se representan los ejes de coordenadas.

Figura 23: *Representación estática de una gráfica con control.*

- En la *Figura 24* se presenta ejecución de una representación dinámica de dos gráficas sin control. La representación se encuentra en curso a una velocidad alta. Además la representación se muestra ampliada y con los ejes de coordenadas dibujados.

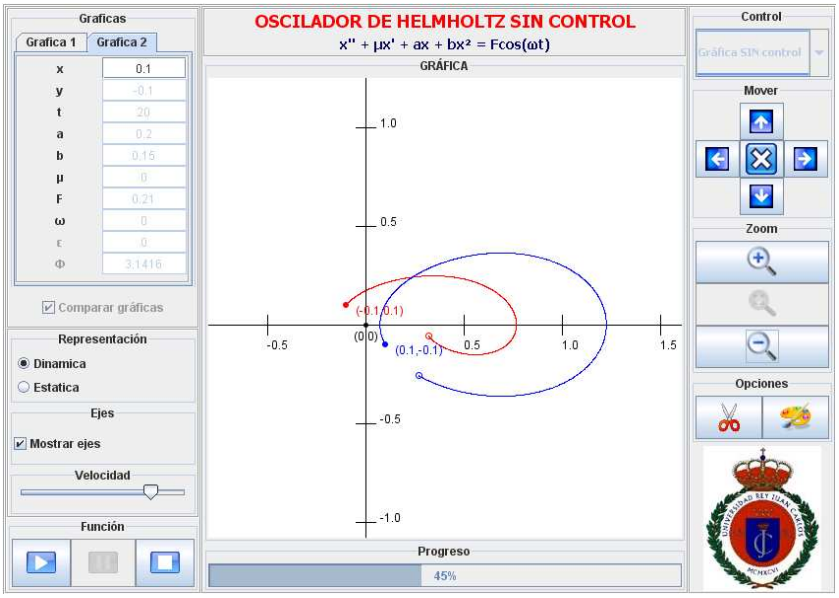

Figura 24: *Representación dinámica de dos gráficas sin control.*

- En la *Figura 25* se presenta la ejecución de una representación dinámica del dibujo del barco sin control. En este momento el barco comienza su hundimiento y la representación está pausada, ya que no ha completado su progreso.

Figura 25: *Representación dinámica del dibujo sin control en la cual el barco se hunde.*

- En la *Figura 26* se presenta la ejecución de una representación estática del dibujo del barco con control. En este momento la representación ha finalizado y el barco no se ha hundido.

Figura 26: *Representación estática del dibujo con control.*

- En la *Figura 27* se presenta el resultado ofrecido por el applet al intentar insertar un valor erróneo para los parámetros de entrada. El parámetro erróneo se indica mediante un mensaje de error específico, además de colorear de rojo el campo de texto correspondiente. La representación no podrá comenzar hasta que todos los valores sean correctos.

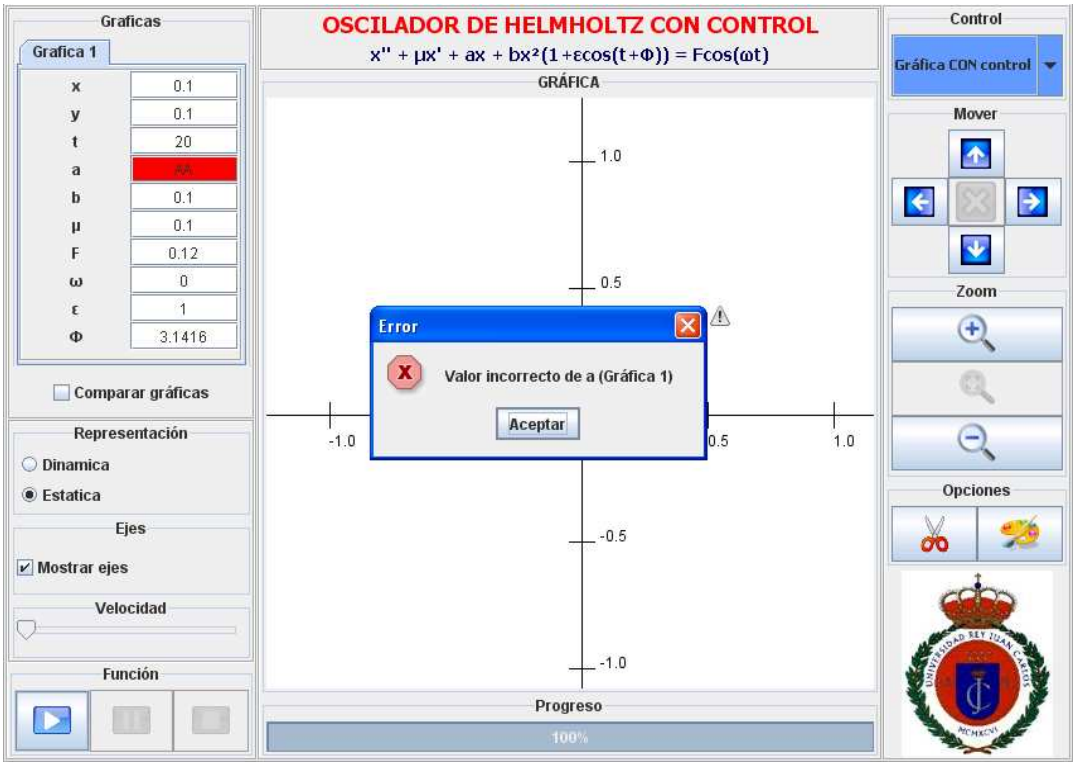

Figura 27: *Resultado de la ejecución al insertar un valor erróneo para los parámetros de entrada.*

- En la *Figura 28* se presenta la ventana de diálogo que se muestra en el applet para seleccionar el color de representación de la/s gráfica/s a representar.

Figura 28: *Ventana de diálogo para la modificación del color de las gráficas representadas.*

4. Resultados y conclusiones

A continuación se enumeran los resultados y objetivos alcanzados en el Proyecto Fin de Carrera:

1. En primer lugar deberemos indicar que se ha conseguido entender todos los conceptos de física necesarios para comprender el problema que se va a tratar en este Proyecto, es decir, el sistema del oscilador de Helmholtz. Para ello se han adquirido conocimientos importantes sobre los sistemas físicos en general, así como sus posibles métodos de resolución.

2. También deberemos indicar que se ha aprendido todo lo necesario para la programación orientada a objetos en JAVA. En particular hemos adquirido una especial destreza en el desarrollo e implementación de applets de JAVA. Para ello se ha utilizado el entorno de desarrollo NetBeans, el cual también es conocido.

3. El principal objetivo conseguido ha sido la implementación mediante un applet de JAVA del sistema dinámico del oscilador de Helmholtz, de manera que el usuario pueda estudiar su comportamiento en diferentes situaciones. Dicha simulación se ha llevado a cabo mediante una aplicación informática consistente en un applet con una interfaz sencilla, completa e intuitiva, para que cualquier usuario no informático y con unos conocimientos básicos en física pueda usar y estudiar la representación de una forma fácil y cómoda a partir de unos valores de entrada proporcionados por el mismo.

4. A raíz del resultado obtenido en el punto anterior, hay que indicar que el applet desarrollado funciona perfectamente y cumple todos los requisitos y funcionalidades previstos al inicio de su desarrollo.

5. Finalmente se ha generado una memoria que contiene toda la información y proceso seguido a lo largo del Proyecto. Esta memoria está redactada de forma clara y gracias a ella podremos entender y conocer todas las características del Proyecto realizado. También nos permitirá conocer el funcionamiento detallado del mismo, así como su utilidad y posibles aplicaciones futuras.

4.1. Logros principales conseguidos

En cuanto a los logros personales conseguidos que considero más importantes, serian:

- Conseguir completar con éxito el desarrollo e implementación del applet del Proyecto de Fin de Carrera para el estudio del sistema del oscilador de Helmholtz.

La implementación del applet supuso un reto personal porque inicialmente poseía unos conocimientos muy básicos de programación en JAVA, pero no había trabajado con applets específicamente. En un principio, el desarrollo del applet de JAVA resultó ser una tarea ardua pero, a medida que iba incluyendo nuevas funcionalidades en la interfaz y al ver la mejora que iba adquiriendo progresivamente, aumento mi motivación o autoestima.

- Poder plasmar en una memoria completa todo el trabajo desarrollado, así como los estudios auxiliares realizados para llegar a tal fin.

- Además de los objetivos especificados anteriormente, me enorgullezco especialmente de haber conseguido que todas las especificaciones y requisitos solicitados para mi proyecto funcionen correctamente y que el tiempo de realización no haya superado al requerido.

4.2. Aplicaciones futuras

Los applets de JAVA tienen la ventaja de que se pueden añadir a páginas Web. Por tanto, el applet desarrollado se podría incluir en una página Web de forma que la página quedase mucho más completa. En concreto este applet podría formar parte de páginas Web de física para poder tener una herramienta sencilla para mostrar los resultados. Este applet se podría incluir con más applets físicos y de sistemas similares.

También podríamos utilizar el código fuente de este proyecto para representar otros sistemas dinámicos similares, ya que simplemente tendríamos que modificar la parte del código en la que se realizan los cálculos. También podríamos añadirle funcionalidades.

Otra posible funcionalidad que se le podría añadir al applet seria que todos las etiquetas de texto de información del applet se pudiesen cargar en varios idiomas diferentes, dotando así al applet de un carácter más internacional y que fuera accesible por un número mayor aun de personas.

Por último, otra funcionalidad que se podría incorporar en un futuro seria que se pudiesen guardar en un fichero los puntos de los distintos gráficos, así como poder también guardar las graficas generadas para su posterior estudio.

5. Experiencia personal y agradecimientos

La experiencia personal una vez finalizado el proyecto de carrera ha sido muy buena en varios aspectos.

Me ha permitido aprender a enfocar y resolver un determinado objetivo desde el principio hasta el final del mismo.

En lo personal, tras la finalización del proyecto de final de carrera sentí una gran sensación de satisfacción, porque pude resolver correctamente todos los problemas que se me plantearon durante el desarrollo del applet. En un principio, no sabía por dónde comenzar la interfaz y tuve que esforzarme para poder encontrar solución a todos los inconvenientes que surgían, pero con trabajo conseguí superarlos.

Por otro lado, logré profundizar en el lenguaje de programación JAVA, lenguaje que es muy utilizado en la actualidad y que podrá ser de gran utilidad en mi carrera profesional.

En cuanto a los agradecimientos, me gustaría nombrar especialmente la ayuda prestada por los tutores de mi proyecto, los Profesores Inés Pérez y Jesús M. Seoane, del Departamento de Física de la Universidad Rey Juan Carlos, por haberme ofrecido la posibilidad de realizar el Proyecto de Fin de Carrera bajo su dirección y por el tiempo y ayuda prestados durante las tutorías que necesité para desarrollar el mismo.

También me gustaría hacer referencia a todos los profesores y compañeros que me han acompañado a lo largo de la carrera y en especial a aquellos que han formado parte de mi equipo en la realización de las prácticas de las diferentes asignaturas.

Finalmente, no puedo olvidarme de mi familia y amigos y de su apoyo incondicional y constante, ya que a pesar de no poder ayudarme en el aprendizaje de las materias, siempre han estado conmigo y en especial en los momentos más difíciles y en los cuales me encontré desalentado.

Por todo ello, sólo puedo decir, ¡muchas gracias!

6. Bibliografía

[1] I. Sendiña Nadal, M. A. F. Sanjuán, Sistemas lineales y no lineales: del oscilador armónico al oscilador caótico, Revista Española de Física 16(3), 2002.

[2] E.Ott, C.Grebogi and J.A.Yorke; "Controlling Chaos",
Phys. Rev. Lett. 64, 1196 (1190).

[3] J.M.Seoane et al. "Avoiding escapes in open dynamical systems using phase control",
Phys. Rev. E78, 016205 (2008).

[4] Jose F. Vélez Serrano, Angel Sánchez Calle, Alfredo Casado. Bernárdez, Santiago Doblas Alvarez. Técnicas avanzadas de diseño de software: Orientación a objetos, UML, patrones de diseño y Java. Universidad Rey Juan Carlos.

[5] William H. Press, Saul A. Teukolsky, William T. Vettering, Brian P. Flannery, Numerical Recipes in C: The Art of Scientific Computing, Cambridge University Press; 1992.

[6] William H. Press, Saul A. Teukolsky, William T. Vettering, Brian P.Flannery, Cambridge University Press; 1992 Numerical Recipes in C: The art of science computing

[7] Richard L. Burden, J. Douglas Faires, Numerical Analysis, ITP, 1997.

[8] Desarrollo iterativo y creciente:
http://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente

[9] Bruce Eckel; 2002 Piensa en Java , Cambridge Educación (2a Edición)

[10] Implementaciones gráficas mediante la clase canvas:
http://sunsite.dcc.uchile.cl/java/docs/JavaTut/Cap4/canvas.html

[11] Creación y control de un thread:
http://sunsite.dcc.uchile.cl/java/docs/JavaTut/Cap7/creath.html