



UNIVERSIDAD DE ALMERÍA

ESCUELA POLITÉCNICA SUPERIOR Y FACULTAD DE CIENCIAS EXPERIMENTALES

ESTUDIO DE ALGORITMOS  
MATEMÁTICOS IMPLEMENTADOS EN  
LIBRERÍAS DE CÓDIGO ABIERTO.  
ECUACIONES DIFERENCIALES: MÉTODOS  
DE RUNGE-KUTTA.

TRABAJO FIN DE GRADO PRESENTADO POR  
ALBA CÁNOVAS PÉREZ

DIRECTORA Y CODIRECTOR  
MERCEDES MARTÍNEZ DURBÁN Y JULIO BARÓN MARTÍNEZ

2014

GRADO EN MATEMÁTICAS

DEPARTAMENTO DE INFORMÁTICA

---

# Índice general

<b>1. Introducción</b>	<b>4</b>
<b>2. Objetivos</b>	<b>5</b>
<b>3. Antecedentes bibliográficos</b>	<b>6</b>
3.1. Métodos de Runge-Kutta . . . . .	6
3.2. Pares encajados de métodos Runge-Kutta . . . . .	11
3.3. Instalación de la librería <i>Apache Commons Math</i> . . . . .	12
3.3.1. Introducción a Apache Commons . . . . .	12
3.3.2. Commons Math: The Apache Commons Mathematics Library . . . . .	12
<b>4. Resultados y discusión</b>	<b>17</b>
4.1. El brazo robótico . . . . .	19
4.1.1. Resolución con Java . . . . .	20
4.1.2. Resolución con Matlab . . . . .	24
4.2. El problema de la órbita de los 3 cuerpos . . . . .	26
4.2.1. Resolución con Java . . . . .	27
4.2.2. Resolución con Matlab . . . . .	30
4.3. Discusión . . . . .	33
<b>5. Conclusión</b>	<b>35</b>
<b>Bibliografía</b>	<b>36</b>
<b>Apéndices</b>	<b>37</b>
<b>A. Programación en Java y entorno Eclipse</b>	<b>38</b>
A.1. Programación en Java . . . . .	38
A.1.1. Origen del lenguaje de programación en Java . . . . .	38
A.1.2. Características de Java . . . . .	38
A.1.3. El entorno de desarrollo integrado Eclipse . . . . .	39
A.1.4. Subversion . . . . .	42

---

# Índice de figuras

3.1.	Descargas de librerías . . . . .	13
3.2.	Archivos en carpeta lib . . . . .	14
3.3.	Propiedades del ProyectoJava . . . . .	14
3.4.	Incluyendo las fuentes . . . . .	15
3.5.	Incluyendo Javadoc . . . . .	16
4.1.	Vista de ODE en la página de la <i>Apache Commons</i> . . . . .	17
4.2.	Paquetes de ode . . . . .	18
4.3.	Paquete nonstiff . . . . .	18
4.4.	Descripción del paquete ode . . . . .	19
4.5.	Descripción del paquete nonstiff . . . . .	19
4.6.	Esquema del péndulo simple . . . . .	20
4.7.	Solución del péndulo simple con Java . . . . .	24
4.8.	Solución del péndulo simple con Matlab . . . . .	26
4.9.	Solución del problema de la órbita de los tres cuerpos con Java . . . . .	30
4.10.	Solución del problema de la órbita de los tres cuerpos con Matlab . . . . .	32
4.11.	Diagrama de clases UML . . . . .	33
4.12.	Visión de la librería <i>Commons Math</i> en el repositorio . . . . .	34
A.1.	Pantalla inicial de <i>Eclipse</i> . . . . .	39
A.2.	Perspectiva inicial orientada al desarrollo de proyectos Java2 SE. . . . .	40
A.3.	Perspectivas . . . . .	41
A.4.	Perspectiva Depurador . . . . .	41
A.5.	Nuevo Repositorio . . . . .	42

---

# Capítulo 1

## Introducción

La dificultad que presentan algunas ecuaciones diferenciales a la hora de su resolución nos lleva a utilizar métodos numéricos para obtener una aproximación. Entre todos los métodos existentes, estudiaremos y utilizaremos la familia de los métodos Runge-Kutta, la cual presentaremos más adelante. Esta familia es muy amplia, nos centraremos en los métodos de paso fijo, concretamente en el método Runge-Kutta clásico de orden 4.

Para la implementación de dicho método se ha utilizado el lenguaje de programación Java con el entorno de programación Eclipse (Ver [1]) y las librerías *Apache Commons* (Ver [2]), las cuales son de código abierto. Esto nos permite acceder a dicho código no sólo para su entendimiento y aprendizaje, sino también nos da la posibilidad de modificar, y por tanto incorporar aquellas mejoras que consideremos necesarias.

Todo la documentación se ha elaborado utilizando la herramienta  $\text{\LaTeX}$  y JabRef para la elaboración de la bibliografía (Ver [3], [4]).

---

# Capítulo 2

## Objetivos

A continuación se relacionan los objetivos principales que se persiguen con el presente trabajo fin de grado:

1. Conocer y estudiar la familia de los métodos Runge-Kutta para la resolución numérica tanto de ecuaciones diferenciales ordinarias como de problemas de valores iniciales. Presentar algunos de los ejemplos más conocidos de esta familia, entre ellos el método de Runge-Kutta Clásico.
2. Estudiar e instalar la librería *Apache Commons Math*, así como conocer su funcionamiento y características.
3. Resolver problemas mediante métodos numéricos utilizando la librería *Apache Commons Math*, comprobando los resultados obtenidos con los obtenidos utilizando una implementación con Matlab.
4. Obtener conclusiones acerca de las posibles ventajas que puede tener un matemático con sólidos conocimientos de programación de ordenadores.

---

# Capítulo 3

## Antecedentes bibliográficos

En este capítulo vamos a introducir algunos conceptos teóricos sobre los métodos Runge-Kutta (Ver [5], [6], [7]) y algunos ejemplos de métodos conocidos.

También explicaremos cómo conectar la librería Commons Math en el entorno Eclipse para poder utilizarla posteriormente en el capítulo 4. (Ver [8], [9], [10]).

### 3.1. Métodos de Runge-Kutta

#### Definición 1

Los **métodos de Runge-Kutta de s etapas**<sup>a</sup> son aquellos que se pueden escribir en la forma

$$\begin{cases} k_i = f(x_n + c_i h, y_n + h \sum_{j=1}^s a_{ij} k_j) \\ y_{n+1} = y_n + h \sum_{i=1}^s b_i k_i \end{cases} \quad (3.1)$$

Cada una de las evaluaciones de  $k_i$  es una etapa.

<sup>a</sup>A menos que especifiquemos lo contrario, vamos a considerar el paso  $h$  fijo.

El método (3.1) se representa por medio de su *tablero de Butcher*:

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_n \end{array}$$

Si escribimos  $c = (c_1, c_2, \dots, c_s)^T$ ,  $b = (b_1, b_2, \dots, b_s)^T$ ,  $A = (a_{ij})$ , el tablero queda de la

siguiente manera:

$$\frac{c}{b^T} \mid \begin{array}{c} A \\ b^T \end{array}$$

Si  $a_{ij} = 0$  para  $j \geq i, i = 1, 2, \dots, s$ , es decir, si la matriz  $A$  es triangular inferior estricta, entonces cada uno de los  $k_i$  viene dado explícitamente en términos de los anteriormente calculados  $k_j, j = 1, 2, \dots, i - 1$ . En este caso, el método es **explícito**. Al escribir su tablero se suelen omitir los ceros sobre la diagonal principal y por encima de ella.

### Ejemplo 1

Nuestro primer método, padre de alguna manera de todos los demás que vamos a estudiar, es el **método de Euler**:

$$y_{n+1} = y_n + hf(x_n, y_n), \quad n = 0, \dots, N - 1. \quad (3.2)$$

Es un método de Runge-Kutta de una etapa ya que podemos reescribirlo de la siguiente manera:

$$\begin{cases} k_1 = f(x_n, y_n) \\ y_{n+1} = y_n + hk_1 \end{cases}$$

Su tablero de Butcher es:

$$\frac{0}{1} \mid \begin{array}{c} 0 \\ 1 \end{array}$$

### Ejemplo 2

El **método de Euler modificado**:

$$y_{n+1} = y_n + hf\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}f(x_n, y_n)\right) \quad n = 0, \dots, N - 1. \quad (3.3)$$

Es un método Runge-Kutta de dos etapas:

$$\begin{cases} k_1 = f(x_n, y_n) \\ k_2 = f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}f(x_n, y_n)\right) \\ y_{n+1} = y_n + hk_2 \end{cases}$$

Su tablero de Butcher es:

$$\frac{0}{1/2} \mid \begin{array}{cc} 0 & 1/2 \\ 0 & 1 \end{array}$$

**Ejemplo 3**

El **método de Euler mejorado**:

$$y_{n+1} = y_n + \frac{h}{2}(f(x_n, y_n) + f(x_{n+1}, y_{n+1})) \quad n = 0, \dots, N - 1. \quad (3.4)$$

Es un método Runge-Kutta de dos etapas:

$$\begin{cases} k_1 = f(x_n, y_n) \\ k_2 = f(x_n + h, y_n + hf(x_n, y_n)) \\ y_{n+1} = y_n + h\left(\frac{k_1}{2} + \frac{k_2}{2}\right) \end{cases}$$

Su tablero de Butcher es:

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 1/2 & 1/2 \end{array}$$

Si el método no es explícito, es **implícito**. En general, es necesario entonces resolver en cada paso un sistema no lineal para calcular los  $k_i$ . Vamos a ver ejemplos de métodos implícitos.

**Ejemplo 4**

El **método RK-Radau IA** de dos etapas, de tablero:

$$\begin{array}{c|cc} 0 & 1/4 & -1/4 \\ 2/3 & 1/4 & 5/12 \\ \hline & 1/4 & 3/4 \end{array}$$

es un método de Runge-Kutta implícito.

Hay una situación intermedia: si  $a_{ij} = 0$  para  $j > i, i = 1, 2, \dots, s$ , entonces cada  $k_i$  está individualmente definido por:

$$k_i = f\left(x_n + c_i h, y_n + h \sum_{j=1}^i a_{ij} k_j\right), \quad i = 1, 2, \dots, s.$$

Estos métodos se llaman **semi-implícitos**. Al escribir sus tableros se omiten los ceros por encima de la diagonal principal.

**Ejemplo 5**

El **método RK-Lobato IIIA** de dos etapas, de tablero:

$$\begin{array}{c|cc} 0 & 0 & \\ 1 & 1/2 & 1/2 \\ \hline & 1/2 & 1/2 \end{array}$$

es un método de Runge-Kutta semi-implícito.

Para que un método Runge-Kutta implícito esté bien determinado, el sistema que define las etapas debe tener solución única. Esto es cierto si  $h$  es pequeño.

Ahora vamos a presentar un conocido método RK, que surge al plantear la estimación de la integral de la ecuación diferencial  $y' = f(t, y)$  mediante la regla de Simpson.

**Ejemplo 6**

El **método de Runge-Kutta clásico**:

$$\left\{ \begin{array}{l} y_{n+1} = y_n + h \left( \frac{1}{6}k_1 + \frac{2}{6}k_2 + \frac{2}{6}k_3 + \frac{1}{6}k_4 \right) \\ k_1 = f(t_n, y_n) \\ k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right) \\ k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right) \\ k_4 = f(t_n + h, y_n + hk_3) \end{array} \right. \quad (3.5)$$

Su tablero de Butcher es:

$$\begin{array}{c|cccc} 0 & 0 & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Salvo alguna excepción, de poca importancia práctica, todos los métodos RK satisfacen que:

$$\sum_j a_{ij} = c_i, \quad i = 1, 2, \dots, s.$$

En el siguiente ejemplo vamos a resolver numéricamente un problema de ecuaciones diferenciales ordinarias con condiciones iniciales usando el método de Runge-Kutta clásico 3.5.

**Ejemplo 7**

Usar el método Runge-Kutta clásico para aproximar  $y(0,5)$  dada la siguiente ecuación diferencial:

$$y' = 2xy, \quad y(0) = 1.$$

*Solución:*

Primero, identificamos las condiciones iniciales, el intervalo y la función:

$$\begin{cases} x_0 = 0 \\ y_0 = 1 \\ h = 0,1 \\ f(x, y) = 2xy \end{cases}$$

Para poder calcular el valor de  $y_1$ , debemos calcular primero los valores de  $k_1$ ,  $k_2$ ,  $k_3$  y  $k_4$ . Tenemos entonces que para la primera iteración:

$$x_1 = x_0 + h = 0,1$$

$$k_1 = hf(x_0, y_0) = 0$$

$$k_2 = hf\left(x_0 + \frac{1}{2}h, y_0 + \frac{1}{2}k_1\right) = 0,01$$

$$k_3 = hf\left(x_0 + \frac{1}{2}h, y_0 + \frac{1}{2}k_2\right) = 0,01005$$

$$k_4 = hf(x_0 + h, y_0 + k_3) = 0,020201$$

Por tanto,

$$y_1 = 1 + \frac{1}{6}(0 + 0,02 + 0,02010 + 0,020201) = 1,01005$$

El proceso debe repetirse hasta, en este caso, obtener  $y_5$ . Resumimos los resultados en la siguiente tabla:

<b>n</b>	$x_n$	$y_n$
0	0	1
1	0,1	1,01005
2	0,2	1,04081
3	0,3	1,09417
4	0,4	1,17351
5	0,5	1,28403

Nuestra solución buscada es:  $y_5 = 1,28403$ .

### 3.2. Pares encajados de métodos Runge-Kutta

Hasta ahora los métodos que hemos visto han sido considerando un paso  $h$  inicial fijo, pero esta condición es poco práctica ya que hay puntos  $x_n$  donde el error toma valores mucho más grandes que en otros donde el paso  $h$  hubiese sido más pequeño.

Hoy día, el procedimiento que se ha probado como más eficaz es el de los *pares encajados de métodos Runge-Kutta*. Estos pares están formados por dos métodos Runge-Kutta que comparten coeficientes  $c$  y  $A$ ,

$$\begin{array}{c|c} c & A \\ \hline & \hat{b}^T \\ \hline & \tilde{b}^T \end{array}$$

donde  $b$  corresponde al método de orden  $p$  que es el usado para calcular el valor  $y_{n+1}$  (avanzar), y  $\hat{b}$  al de orden  $\hat{p}$ , el usado para estimar el error local. Originalmente se tomaba  $\hat{p} > p$ . Hoy día, sin embargo, se toma  $p > \hat{p}$ , pues se ha comprobado que es una opción más eficaz en la práctica.

#### Ejemplo 8

El método Runge-Kutta-Fehlberg RKF2(3), con tablero de Butcher:

$$\begin{array}{c|cc} 0 & 0 & \\ 1 & 1 & \\ 1/2 & 1/4 & 1/4 \\ \hline b_i & 1/2 & 1/2 & 0 \\ \tilde{b}_i & 1/6 & 1/6 & 4/6 \end{array}$$

con órdenes  $p = 2$  y  $\hat{p} = 3$ . El método de orden 2 que se usa para avanzar es en realidad de dos etapas, mientras que el de orden 3 que se usa para estimar, es de tres etapas.

#### Ejemplo 9

El método Runge-Kutta-Fehlberg RKF2(3)B, con tablero de Butcher:

$$\begin{array}{c|ccc} 0 & 0 & & \\ 1/4 & 1/4 & & \\ 27/40 & -189/800 & 729/800 & \\ 1 & 214/891 & 1/33 & 650/891 \\ \hline b_i & 214/891 & 1/33 & 650/891 & 0 \\ \tilde{b}_i & 533/2106 & 0 & 800/1053 & -1/78 \end{array}$$

El método de avance es de 3 etapas y de orden  $p = 2$ , y el de estimación es de 4 etapas y orden  $\hat{p} = 3$ .

### 3.3. Instalación de la librería *Apache Commons Math*

En esta sección vamos a introducir e instalar la librería *Apache Commons Math*, la cual usaremos para resolver problemas de ecuaciones diferenciales, centrándonos especialmente en los que no tienen solución analítica.

#### 3.3.1. Introducción a Apache Commons

Commons es un proyecto de Apache centrado en todos los aspectos de componentes reutilizables de Java, es de código abierto. El proyecto Apache Commons está compuesto de tres partes:

1. The Commons Proper; un repositorio de componentes reutilizables de Java.
2. The Commons Sandbox; un espacio de trabajo para el desarrollo de los componentes de Java.
3. The Commons Dormant; un repositorio de componentes que habitualmente se encuentran inactivos.

#### The Commons Proper

The Commons Proper está dedicada a un objetivo principal: creación y mantenimiento de componentes reutilizables de Java. Dicha aplicación funciona como lugar para la colaboración e intercambio, donde los colaboradores y creadores de este software de todas partes de la comunidad de Apache pueden trabajar juntos sobre proyectos.

#### The Commons Sandbox

Este proyecto también contiene un espacio de trabajo que es abierto a todo el conjunto de usuarios de Apache. Esto es un lugar para probar nuevas ideas y prepararse para la inclusión en la parte de The Commons Proper del proyecto o en otro proyecto de Apache. Los usuarios son libres de experimentar con los componentes desarrollados en The Commons Sandbox, pero los componentes de The Commons Sandbox no necesariamente serán mantenidos, en particular en su estado corriente.

#### The Commons Dormant

Está formado por los componentes de The Commons Sandbox que han sido considerados inactivos ya que ellos han visto poca actividad de desarrollo reciente. Si se desea usar cualquiera de estos componentes, se debe construirlos por uno mismo.

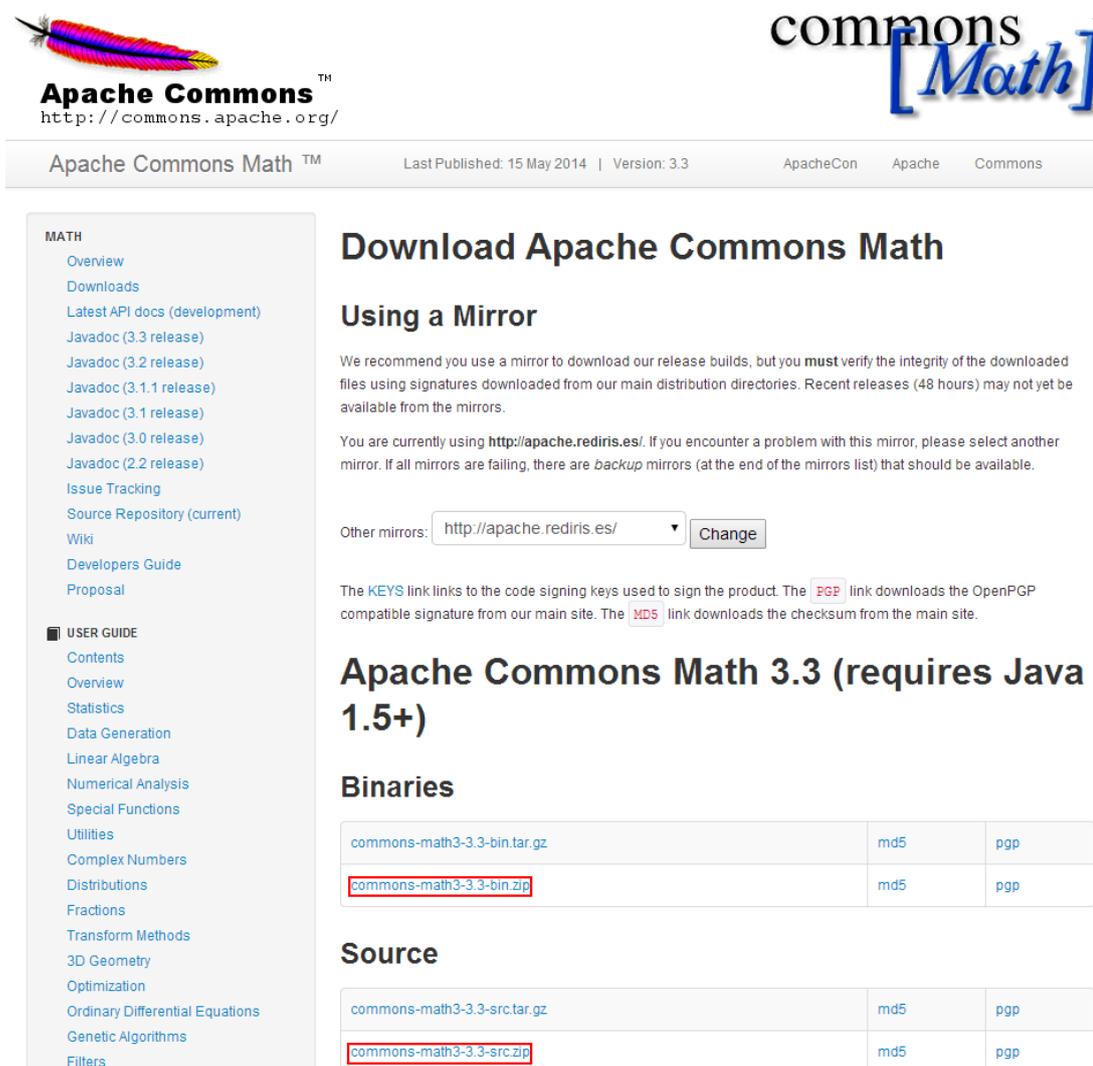
#### 3.3.2. Commons Math: The Apache Commons Mathematics Library

Commons Math es una librería ligera e independiente de componentes software que implementan algoritmos matemáticos y estadísticos que aborda los problemas más comunes que no están disponibles en el lenguaje de programación Java o Commons Lang.

## Instalación librería

Para poder usar esta librería debemos seguir los siguientes pasos:

1. Descargar los archivos `commons-math3-3.3-bin.zip` y `commons-math3-3.3-src.zip` de la página web [11] (Ver Fig. 3.1)



The screenshot shows the Apache Commons Math website. The header includes the Apache Commons logo and the Commons Math logo. The main content area is titled "Download Apache Commons Math" and includes a section "Using a Mirror" with instructions on how to download the software. Below this, there are two tables: "Binaries" and "Source". Both tables list download links for different file formats (tar.gz and zip) and provide links for md5 and pgp signatures. The links for the zip files are highlighted with red boxes.

**Apache Commons Math**™ Last Published: 15 May 2014 | Version: 3.3 ApacheCon Apache Commons

**MATH**

- Overview
- Downloads
- Latest API docs (development)
- Javadoc (3.3 release)
- Javadoc (3.2 release)
- Javadoc (3.1.1 release)
- Javadoc (3.1 release)
- Javadoc (3.0 release)
- Javadoc (2.2 release)
- Issue Tracking
- Source Repository (current)
- Wiki
- Developers Guide
- Proposal

**USER GUIDE**

- Contents
- Overview
- Statistics
- Data Generation
- Linear Algebra
- Numerical Analysis
- Special Functions
- Utilities
- Complex Numbers
- Distributions
- Fractions
- Transform Methods
- 3D Geometry
- Optimization
- Ordinary Differential Equations
- Genetic Algorithms
- Filters

## Download Apache Commons Math

### Using a Mirror

We recommend you use a mirror to download our release builds, but you **must** verify the integrity of the downloaded files using signatures downloaded from our main distribution directories. Recent releases (48 hours) may not yet be available from the mirrors.

You are currently using <http://apache.rediris.es/>. If you encounter a problem with this mirror, please select another mirror. If all mirrors are failing, there are *backup* mirrors (at the end of the mirrors list) that should be available.

Other mirrors:

The [KEYS](#) link links to the code signing keys used to sign the product. The [PGP](#) link downloads the OpenPGP compatible signature from our main site. The [MD5](#) link downloads the checksum from the main site.

## Apache Commons Math 3.3 (requires Java 1.5+)

### Binaries

<a href="#">commons-math3-3.3-bin.tar.gz</a>	<a href="#">md5</a>	<a href="#">pgp</a>
<a href="#">commons-math3-3.3-bin.zip</a>	<a href="#">md5</a>	<a href="#">pgp</a>

### Source

<a href="#">commons-math3-3.3-src.tar.gz</a>	<a href="#">md5</a>	<a href="#">pgp</a>
<a href="#">commons-math3-3.3-src.zip</a>	<a href="#">md5</a>	<a href="#">pgp</a>

Figura 3.1: Descargas de librerías

2. Creamos una carpeta, que llamaremos lib, dentro de nuestro proyecto (ProyectoJava).
3. Descomprimos los archivos que descargamos, en ellos buscamos los archivos commons-math3-3.3-javadoc.jar, commons-math3-3.3-sources.jar y commons-math3-3.3.jar y los copiamos en nuestra carpeta lib. (Ver Fig. 3.2)

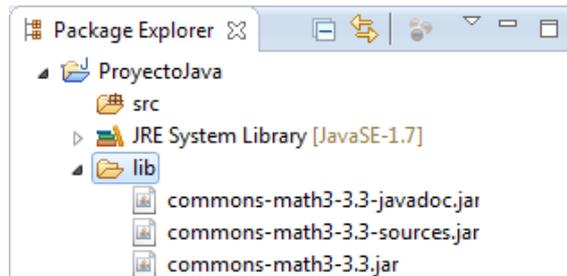


Figura 3.2: Archivos en carpeta lib

4. Sobre nuestro proyecto ProyectoJava, hacemos click botón derecho y seleccionamos **Properties** → **Java Build Path** → **Libraries**. Nos aparecerá una ventana como la de la Fig. 3.3:

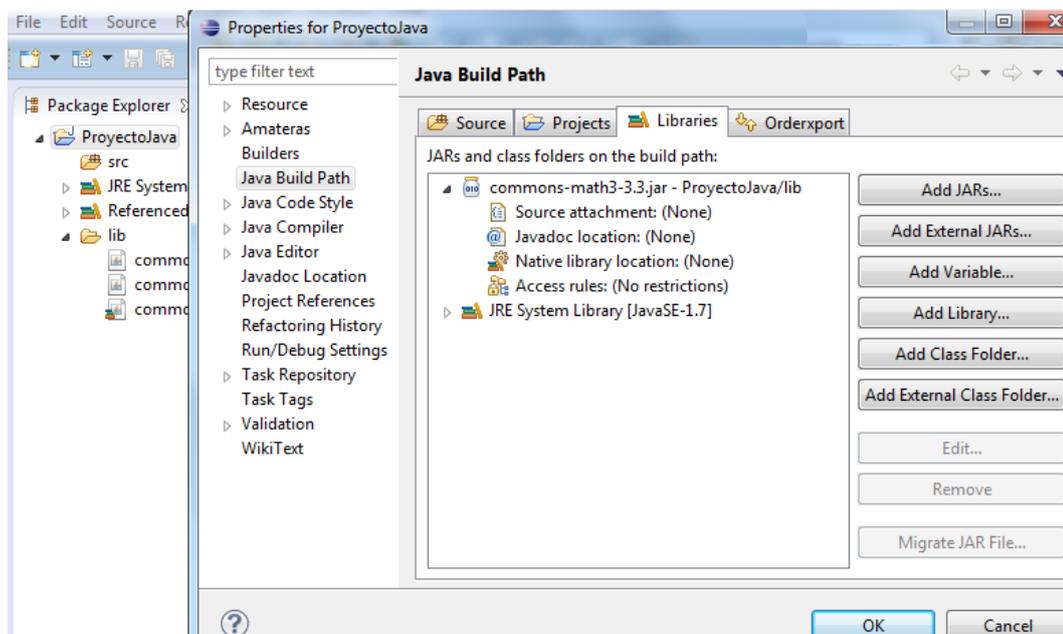


Figura 3.3: Propiedades del ProyectoJava

5. Seleccionamos **Source attachment:(None)** y editamos en la opción **Edit**, situada en la parte derecha. En la ventana abierta, marcamos **Workspace location** y al examinar (**Browse**) dentro de la carpeta **lib**, seleccionamos el archivo **commons-math3-3.3-sources.jar** (Ver 3.4).

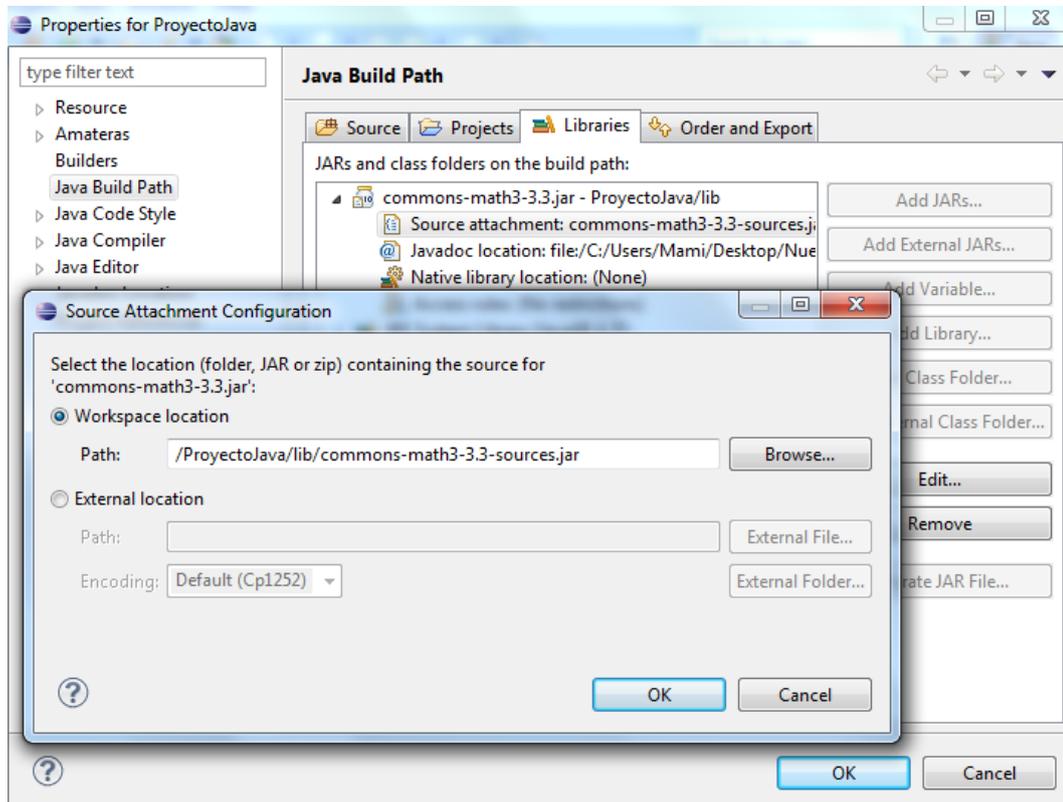


Figura 3.4: Incluyendo las fuentes

6. Seleccionamos **Javadoc location:(None)** y editamos igual que el paso anterior. En la ventana abierta, marcamos **Javadoc in archive** → **External file** y en la opción **Archive path** examinamos (**Browse**) y seleccionamos dentro de la carpeta **lib** el archivo **commons-math3-3.3-javadoc.jar** (Ver Fig 3.5).

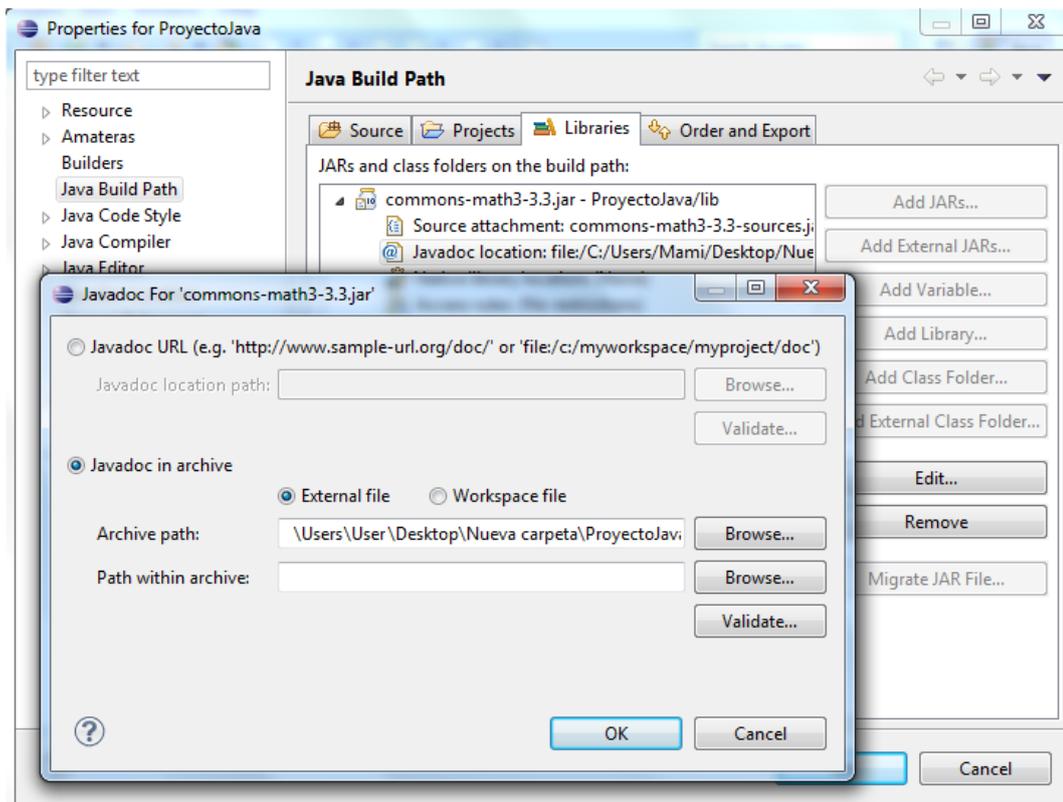


Figura 3.5: Incluyendo Javadoc

Así concluimos la instalación de esta librería.

---

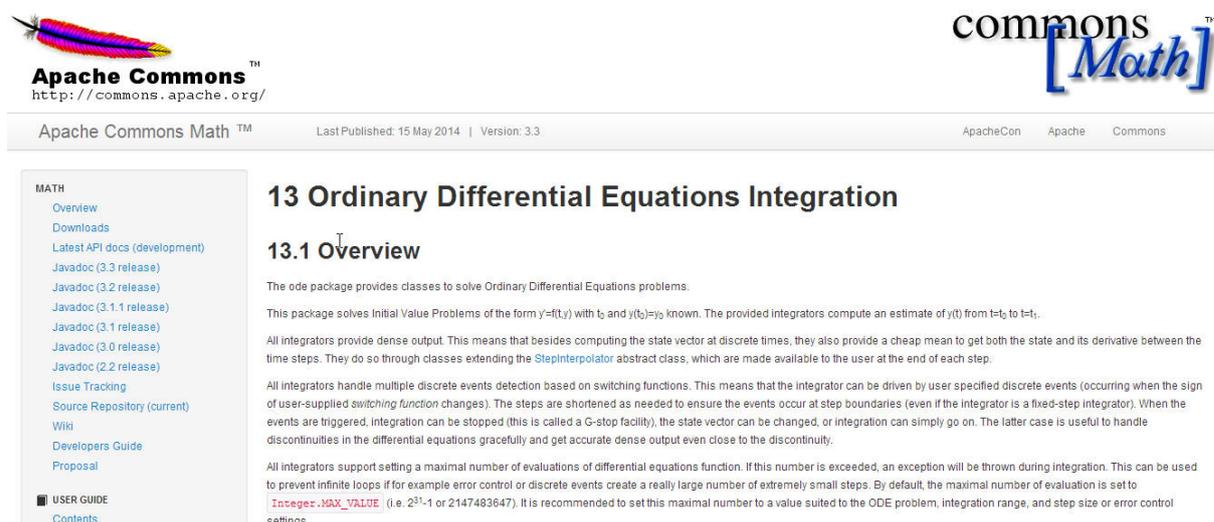
# Capítulo 4

## Resultados y discusión

En este capítulo resolveremos 2 problemas de ecuaciones diferenciales que no tienen solución analítica conocida:

1. El brazo robótico (Ver [12])
2. El problema de la órbita de los tres cuerpos (Ver [13])

Para ello utilizaremos la librería de Commons Math. En esta librería hay diversidad de paquetes que permiten resolver problemas de diferentes campos de las matemáticas. Nosotros usaremos los que necesitamos para nuestro trabajo: **Ordinary Differential Equations** (Ver [14])



The screenshot shows the Apache Commons Math website. At the top left is the Apache Commons logo with the URL <http://commons.apache.org/>. At the top right is the Commons Math logo. Below the logos is a navigation bar with the text "Apache Commons Math™", "Last Published: 15 May 2014 | Version: 3.3", and "ApacheCon Apache Commons". The main content area is titled "13 Ordinary Differential Equations Integration" and "13.1 Overview". The overview text describes the package's purpose: "The ode package provides classes to solve Ordinary Differential Equations problems." It further explains that the package solves Initial Value Problems of the form  $y'=f(t,y)$  with  $t_0$  and  $y(t_0)=y_0$  known. It also mentions that all integrators provide dense output and handle multiple discrete events detection based on switching functions. A note at the bottom states that all integrators support setting a maximal number of evaluations of differential equations function, with a default value of `Integer.MAX_VALUE` (i.e.  $2^{31}-1$  or 2147483647).

Figura 4.1: Vista de ODE en la página de la *Apache Commons*

En la librería Commons Math, en el apartado de Ordinary Differential Equation, encontramos varios paquetes en los cuales se encuentran diversas clases para resolver ecuaciones diferenciales (Ver Figura 4.2).

- ▶ org.apache.commons.math3.ode 25
- ▶ org.apache.commons.math3.ode.events 25
- ▶ org.apache.commons.math3.ode.nonstiff 25
- ▶ org.apache.commons.math3.ode.sampling 25

Figura 4.2: Paquetes de ode

Para los problemas que vamos a resolver trabajaremos con algunas clases del paquete `org.apache.commons.math3.ode.nonstiff` (Ver Figura 4.3).

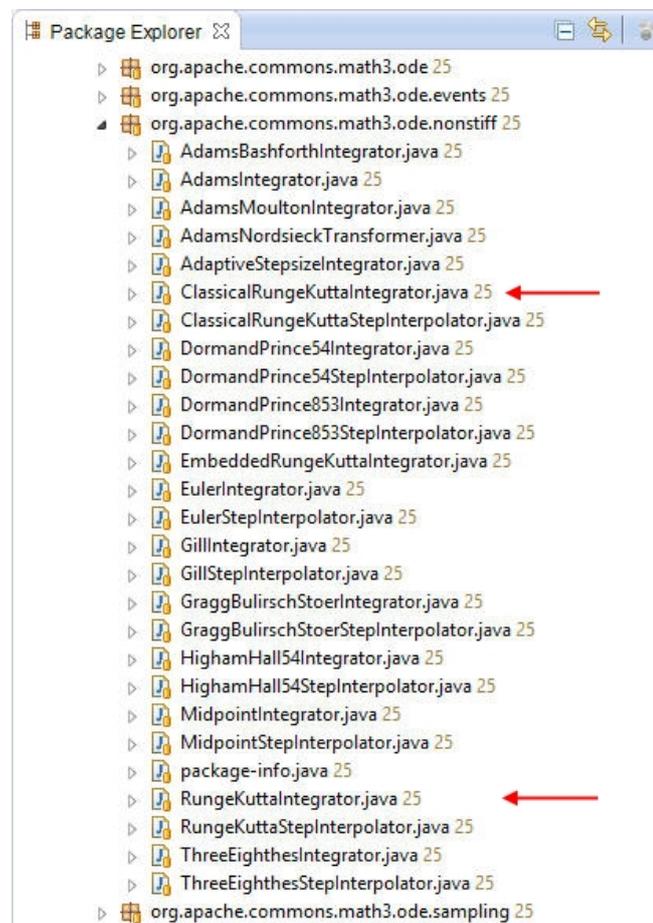


Figura 4.3: Paquete nonstiff

En las siguientes figuras vemos una breve descripción de las clases que tiene implementadas los paquetes que estamos usando:

**Package org.apache.commons.math3.ode**  
 This package provides classes to solve Ordinary Differential Equations problems.  
 See: Description

Interface Summary	
Interface	Description
FirstOrderDifferentialEquations	This interface represents a first order differential equations set.
FirstOrderIntegrator	This interface represents a first order integrator for differential equations.
MainStateJacobianProvider	Interface expanding <code>FirstOrderDifferentialEquations</code> in order to compute exactly the main state jacobian matrix for partial derivatives equations.
MultistepIntegrator.NordsieckTransformer	Transformer used to convert the first step to Nordsieck representation.
ODEIntegrator	This interface defines the common parts shared by integrators for first and second order differential equations.
Parameterizable	This interface enables to process any parameterizable object.
ParameterizableODE	Interface to compute by finite difference Jacobian matrix for some parameter when computing partial derivatives equations.
ParameterJacobianProvider	Interface to compute exactly Jacobian matrix for some parameter when computing partial derivatives equations.
SecondaryEquations	This interface allows users to add secondary differential equations to a primary set of differential equations.
SecondOrderDifferentialEquations	This interface represents a second order differential equations set.
SecondOrderIntegrator	This interface represents a second order integrator for differential equations.

Class Summary	
Class	Description
AbstractIntegrator	Base class managing common boilerplate for all integrators.
AbstractParameterizable	This abstract class provides boilerplate parameters list.
ContinuousOutputModel	This class stores all information provided by an ODE integrator during the integration process and build a continuous model of the solution from this.
EquationMapper	Class mapping the part of a complete state or derivative that pertains to a specific differential equation.
ExpandableStateHolder	This class represents a combined set of first order differential equations, with at least a primary set of equations expandable by some sets of secondary equations.
FirstOrderConverter	This class converts second order differential equations to first order ones.
JacobianMatrixes	This class defines a set of secondary equations to compute the Jacobian matrices with respect to the initial state vector and, if any, to some parameters of the primary ODE set.
MultistepIntegrator	This class is the base class for multistep integrators for Ordinary Differential Equations.

Exception Summary	
Exception	Description
JacobianMatrixes.MismatchedEquations	Special exception for equations mismatch.
UnknownParameterException	Exception to be thrown when a parameter is unknown.

Figura 4.4: Descripción del paquete ode

**Package org.apache.commons.math3.ode.nonstiff**  
 This package provides classes to solve non-stiff Ordinary Differential Equations problems.  
 See: Description

Class Summary	
Class	Description
AdamsBashforthIntegrator	This class implements explicit Adams-Bashforth integrators for Ordinary Differential Equations.
AdamsIntegrator	Base class for Adams-Bashforth and Adams-Moulton integrators.
AdamsMoultonIntegrator	This class implements implicit Adams-Moulton integrators for Ordinary Differential Equations.
AdamsNordsieckTransformer	Transformer to Nordsieck vectors for Adams integrators.
AdaptiveStepsizeIntegrator	This abstract class holds the common part of all adaptive stepsize integrators for Ordinary Differential Equations.
ClassicalRungeKuttaIntegrator	This class implements the classical fourth order Runge-Kutta integrator for Ordinary Differential Equations (it is the most often used Runge-Kutta method).
DormandPrince54Integrator	This class implements the 5(4) Dormand-Prince integrator for Ordinary Differential Equations.
DormandPrince853Integrator	This class implements the 8(5,3) Dormand-Prince integrator for Ordinary Differential Equations.
EmbeddedRungeKuttaIntegrator	This class implements the common part of all embedded Runge-Kutta integrators for Ordinary Differential Equations.
EulerIntegrator	This class implements a simple Euler integrator for Ordinary Differential Equations.
GillIntegrator	This class implements the Gill fourth order Runge-Kutta integrator for Ordinary Differential Equations.
GraggBulirschStoerIntegrator	This class implements a Gragg-Bulirsch-Stoer integrator for Ordinary Differential Equations.
HighamHall54Integrator	This class implements the 5(4) Higham and Hall integrator for Ordinary Differential Equations.
LutherIntegrator	This class implements the Luther sixth order Runge-Kutta integrator for Ordinary Differential Equations.
MidpointIntegrator	This class implements a second order Runge-Kutta integrator for Ordinary Differential Equations.
RungeKuttaIntegrator	This class implements the common part of all fixed step Runge-Kutta integrators for Ordinary Differential Equations.
ThreeEightsIntegrator	This class implements the 3/8 fourth order Runge-Kutta integrator for Ordinary Differential Equations.

Figura 4.5: Descripción del paquete nonstiff

## 4.1. El brazo robótico

El brazo idealizado de un robot puede modelarse como un péndulo de masa  $m$ , suspendido de un brazo de longitud  $l$  (Ver Figura 4.6). En un momento  $t$ , su posición se describe por el ángulo  $\theta(t)$  (en radianes) que forma el brazo con la vertical. Si asumimos que la fricción del aire es proporcional a la velocidad angular del péndulo, obtenemos la EDO (ecuación diferencial ordinaria):

$$ml\theta'' + c\theta' + mg \sin(\theta) = u(t), \quad t \geq 0 \tag{4.1}$$

donde  $g$  es la aceleración de la gravedad y  $c$  es la constante de fricción. El miembro derecho  $u(t)$  define la fuerza externa aplicada al péndulo en el momento  $t$ .

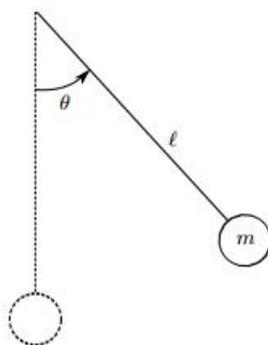


Figura 4.6: Esquema del péndulo simple

Nosotros vamos a considerar que no existe fuerza externa, es decir  $u \equiv 0$ . Para tratar con un problema estable consideramos  $c \neq 0$ , en particular usaremos  $c = 0,5$ .

### Problema 1

Consideremos el péndulo simple que satisface la EDO 4.1, con los datos  $m = 1$  kg,  $l = 1$  m,  $c = 0,5$ ,  $g = 9,81$  m/s<sup>2</sup> y  $u \equiv 0$ . Vamos a obtener y representar la solución de dicha EDO.

#### 4.1.1. Resolución con Java

A continuación, los pasos realizados durante el diseño del programa son:

1. Diseñamos una clase `PendoloSimple` que implementa la interface `FirstOrderDifferentialEquation`, donde se define la ecuación diferencial que se va a resolver.
2. Creamos un objeto de la clase `FirstOrderIntegrator` y elegimos el integrador con el que vamos a trabajar.
3. Creamos un objeto de la clase `PendoloSimple`.
4. Introducimos los valores de nuestro PVI (problema de valores iniciales).
5. Aplicamos al objeto creado en **2.** el método de objeto `integrate`.

Como podemos ver en los siguientes listados de código.

```

1 package org.tfg.ejemplos;
2
3 import org.apache.commons.math3.exception.DimensionMismatchException;
4 import org.apache.commons.math3.exception.MaxCountExceededException;
5 import org.apache.commons.math3.ode.FirstOrderDifferentialEquations;

```

```
6
7 public class PenduloSimple implements FirstOrderDifferentialEquations
8     {
9         final double G = 9.81;
10        private double longitud;
11        private double masa;
12        private double friccion;
13
14        public PenduloSimple(double longitud, double masa, double
15            friccion) {
16            super();
17            this.longitud = longitud;
18            this.masa = masa;
19            this.friccion = friccion;
20        }
21
22        public double getLongitud() {
23            return longitud;
24        }
25
26        public double getMasa() {
27            return masa;
28        }
29
30        public double getFriccion() {
31            return friccion;
32        }
33
34        public double getG() {
35            return G;
36        }
37
38        public String toString() {
39            return "Longitud=" + longitud + " m, Masa=" + masa
40                + " Kg, Friccion=" + friccion + " ";
41        }
42
43        public int getDimension() {
44            return 2;
45        }
46
47        public void computeDerivatives(double t, double[] y, double[]
48            yDot)
49            throws MaxCountExceededException,
50                DimensionMismatchException {
51            yDot[0] = y[1];
```

```

49         yDot[1] = -(G/longitud)*Math.sin(y[0])-(friccion/(
50             masa*longitud))*y[1];
51     }

```

En el siguiente listado ponemos a prueba el diseño mediante el programa `TestPenduloSimple` para resolver el problema con los datos ya específicos de masa, longitud y constante de fricción:

```

1  package org.tfg.ejemplos;
2
3  import java.io.File;
4  import java.io.PrintWriter;
5  import java.io.IOException;
6
7
8  import org.apache.commons.math3.ode.FirstOrderDifferentialEquations;
9  import org.apache.commons.math3.ode.FirstOrderIntegrator;
10 import org.apache.commons.math3.ode.nonstiff.
    ClassicalRungeKuttaIntegrator;
11
12 public class TestPenduloSimple {
13
14     public static void main(String[] args) throws IOException {
15         // Introducimos el paso h
16         FirstOrderIntegrator ps = new
            ClassicalRungeKuttaIntegrator(0.003);
17
18         // Creamos un objeto de la clase PenduloSimple
19         FirstOrderDifferentialEquations ode = new
            PenduloSimple(1.0, 1.0, 0.5);
20         PenduloSimple ps1 = new PenduloSimple(1.0, 1.0, 0.5);
21         System.out.println("Características del p'endolo
            simple:");
22         System.out.println(ps1.toString());
23
24         // Datos del PVI
25         double[] y0 = { Math.PI / 4, 0 };
26         double[] y = { Math.PI / 4, 0 }; //Inicializamos a
            cualquier valor, por ejemplo a y0.
27         double a = 0.0;
28         double b = 30.0;
29         double t = 0.0;
30         double h = 0.003;
31
32         // Creamos un archivo con los puntos para despu'es
            representarlos.

```

```
33     String directorioEntrada = System.getProperty("user.dir");
34     System.out.println("user.dir: " + directorioEntrada);
35
36     directorioEntrada = directorioEntrada + File.separator + "src"
37         + File.separator + "org" + File.separator + "tfg"
38         + File.separator + "ejemplos";
39     System.out.println(directorioEntrada);
40     String archivoDondeEsta = directorioEntrada + File.separator
41         + "PenduloSimple.txt";
42     System.out.println(archivoDondeEsta);
43     File file = new File(archivoDondeEsta);
44
45     // Borra lo que hubiera en el archivo existente
46     PrintWriter pw = new PrintWriter(file);
47
48     // Escribe una cabecera en el archivo
49     pw.println("Tiempo t\t\tAngulo theta ");
50
51     while (t <= b) {
52         t = t + h;
53         // Integramos con el método integrate
54         ps.integrate(ode, a, y0, t, y);
55         // Salida archivo. Guardamos t e y[0]. Nombre del archivo
56         // PenduloSimple.txt,
57         // guardado en mi proyecto.
58         pw.print(t+"\t"+y[0] + "\t");
59         pw.println();
60     }
61
62     // Cerramos el archivo
63     pw.close();
64 }
65 }
```

La ejecución del programa genera el archivo `PenduloSimple.txt` que contiene una serie de puntos que se han obtenido con el algoritmo Runge-Kutta Clásico.

Para la representación de la solución vamos a utilizar un programa externo a Eclipse, **mjograph** [15], utilizando los puntos obtenidos anteriormente. (Ver Fig. 4.7)

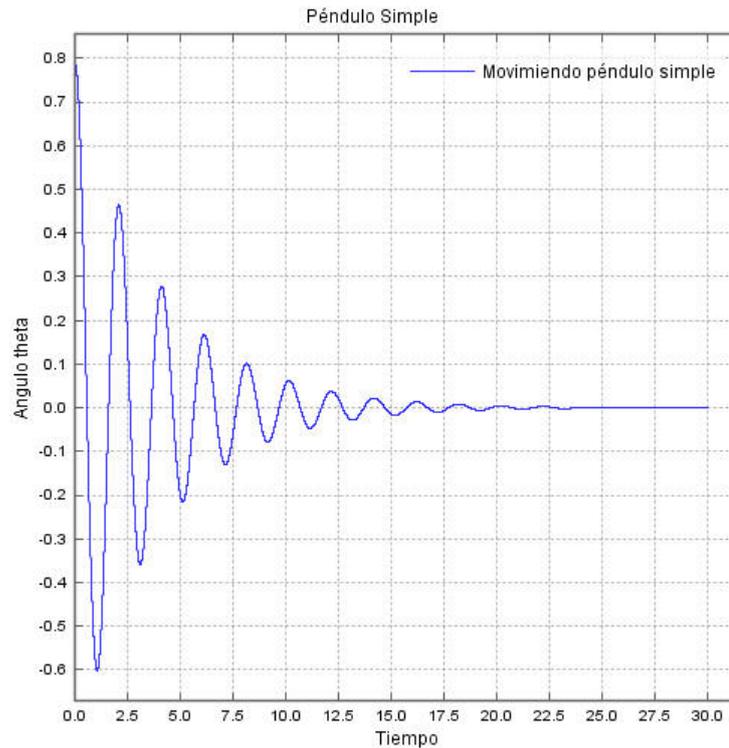


Figura 4.7: Solución del péndulo simple con Java

### 4.1.2. Resolución con Matlab

En este apartado resolveremos el mismo problema usando la implementación del método de Runge-Kutta Clásico 3.5.

Matlab tiene varias funciones ya implementadas para resolver un PVI, pero no tiene implementado el método que nosotros vamos a utilizar. Utiliza los *pares encajados de Métodos Runge-Kutta* que vimos en el capítulo 3.

A continuación ejecutaremos el archivo `PendoloSimple.m` para resolver nuestro problema:

```

1 % Datos pendulo
2 m = 1;
3 l = 1;
4 c= 0.5;
5 G = 9.81;
6
7 % PVI
8 f = @(x,y) [y(2); -(G/l)*sin(y(1))-(c/(m*l))*y(2)];
9
10
11 % Datos PVI

```

```
12 a = 0;
13 b = 30;
14 y0 = [pi/4;0];
15 N = 10000;
16
17 % Paso para los nodos
18 h = (b-a)/N;
19
20 % Tamano del sistema
21 m = length(y0);
22
23 % Construimos los nodos
24 x = a+h.*[0:1:N];
25 y(:,1)=y0;
26
27 % Aplicamos el metodo Runge-Kutta
28 for i=1:N
29     k1 = feval(f,x(i),y(:,i));
30     k2 = feval(f,x(i)+h/2,y(:,i)+(h/2)*k1);
31     k3 = feval(f,x(i)+h/2,y(:,i)+(h/2)*k2);
32     k4 = feval(f,x(i)+h,y(:,i)+h*k3);
33     y(:,i+1) = y(:,i)+(h/6)*(k1+2*k2+2*k3+k4);
34 end
35
36 % Representamos la solucion
37 grid on
38 plot(x,y(1,:), 'r')
39 legend('Movimiento pendulo simple')
40 title('Pendulo Simple')
41 xlabel('Tiempo')
42 ylabel('Angulo \theta')
43
44 % Guardar datos en archivo
45 fi = fopen('salidaPenduloSimple.txt', 'w')
46
47 for k =1:10001
48     fprintf(fi, '%f %f\n', x(1,k),y(1,k));
49 end
50 fclose(fi)
```

Con la ejecución de este código, obtenemos la representación de los puntos obtenidos mediante el método (Ver Fig. 4.8).

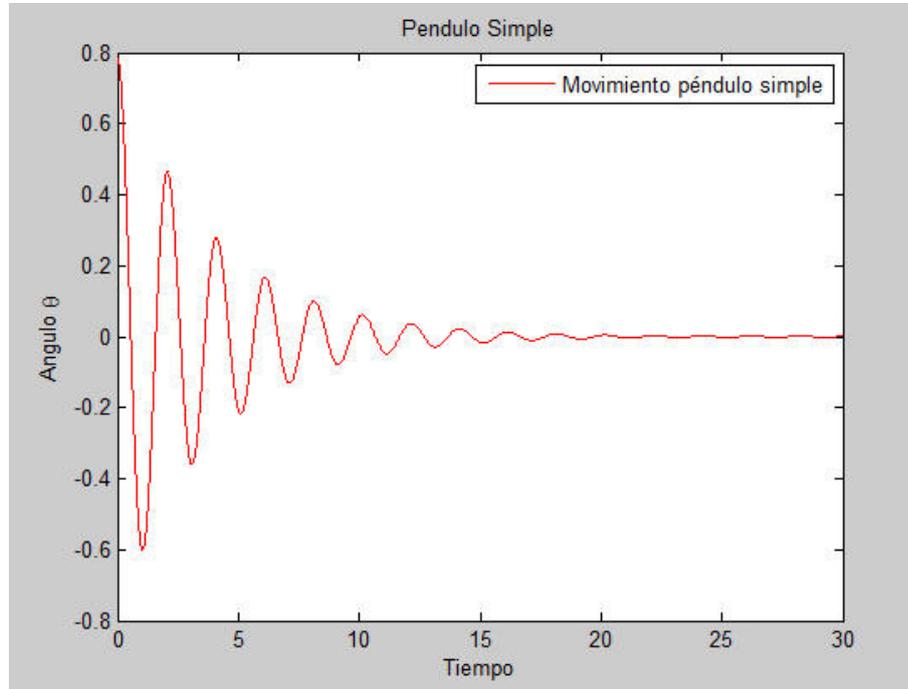


Figura 4.8: Solución del péndulo simple con Matlab

## 4.2. El problema de la órbita de los 3 cuerpos

El problema de la órbita de los 3 cuerpos es un PVI que representa el sistema de un satélite que orbita a un planeta que a su vez orbita a un tercer cuerpo en dos dimensiones (semejante al sistema Luna-Tierra-Sol). Las condiciones iniciales del sistema son las posiciones y velocidades iniciales del satélite. El intervalo de tiempo (*time span*) es el periodo de tiempo que lleva al planeta orbitar alrededor del tercer cuerpo una vez.

Se puede describir matemáticamente este problema con la siguiente EDO:

$$\begin{cases} \dot{y}_1 = y_3 \\ \dot{y}_2 = y_4 \\ \dot{y}_3 = y_1 + 2y_4 - \frac{\mu_2(y_1 + \mu_2)}{d_1} - \frac{\mu_1(y_1 - \mu_2)}{d_2} \\ \dot{y}_4 = y_2 - 2y_3 - \frac{\mu_2 y_2}{d_1} - \frac{\mu_1 y_2}{d_2} \end{cases} \quad (4.2)$$

con

$$\begin{cases} d_1 = ((y_1 + \mu_1)^2 + y_2^2)^{3/2} \\ d_2 = ((y_1 - \mu_2)^2 + y_2^2)^{3/2} \\ \mu_1 = 0,012277471 \\ \mu_2 = 1 - \mu_1 \end{cases}$$

donde  $y_1$  e  $y_2$  forman la posición del satélite en dos dimensiones,  $y_3$  e  $y_4$  son las

velocidades de este satélite,  $d_1$  y  $d_2$  son los cálculos de las distancias relativas y  $\mu_1$  y  $\mu_2$  son las masas del satélite y del planeta, respectivamente.

### Problema 2

Consideremos los siguientes valores iniciales del PVI:

$$\begin{cases} y_1(t_0) = 0,994 \\ y_2(t_0) = 0 \\ y_3(t_0) = 0 \\ y_4(t_0) = -2,00158510637908252240537862224. \end{cases}$$

El intervalo de tiempo del PVI es:

$$\begin{cases} t_0 = 0 \\ t_f = 17,065216501579625588917206249. \end{cases}$$

Vamos a obtener y representar la solución de la EDO 4.2.

#### 4.2.1. Resolución con Java

A continuación, los pasos realizados durante el diseño del programa son:

1. Diseñamos una clase `TresCuerpos` que implementa la interface `FirstOrderDifferentialEquation`, donde se define la ecuación diferencial que se va a resolver.
2. Creamos un objeto de la clase `FirstOrderIntegrator` y elegimos el integrador con el que vamos a trabajar.
3. Creamos un objeto de la clase `TresCuerpos`.
4. Introducimos los valores de nuestro PVI (problema de valores iniciales).
5. Aplicamos al objeto creado en el apartado **2.** el método de objeto `integrate`.

como podemos ver en los siguientes listados de código.

```

1 package org.tfg.ejemplos;
2
3 import org.apache.commons.math3.exception.DimensionMismatchException;
4 import org.apache.commons.math3.exception.MaxCountExceededException;
5 import org.apache.commons.math3.ode.FirstOrderDifferentialEquations;
6
7 public class TresCuerpos implements FirstOrderDifferentialEquations {
8
9     final double Mu1 = 0.012277471;

```

```

10     final double Mu2 = 1 - Mu1;
11
12     public int getDimension() {
13         return 4;
14     }
15
16     public void computeDerivatives(double t, double[] y, double[]
17         yDot)
18         throws MaxCountExceededException,
19             DimensionMismatchException {
20
21         double d1 = Math.pow(Math.pow((y[0] + Mu1), 2) + y
22             [1]*y[1], 1.5);
23         double d2 = Math.pow(Math.pow((y[0] - Mu2), 2) + y[1]*y[1],
24             1.5);
25
26         yDot[0] = y[2];
27         yDot[1] = y[3];
28         yDot[2] = y[0] + 2*y[3] - Mu2*(y[0] + Mu1)/d1 - Mu1*(y[0] -
29             Mu2)/d2;
30         yDot[3] = y[1] - 2*y[2] - Mu2*y[1]/d1 - Mu1*y[1]/d2;
31     }
32 }

```

En el siguiente listado ponemos a prueba el diseño mediante el programa `TestTresCuerpos` para resolver el problema con los datos ya específicos del PVI:

```

1 package org.tfg.ejemplos;
2
3 import java.io.File;
4 import java.io.PrintWriter;
5 import java.io.IOException;
6
7 import org.apache.commons.math3.ode.FirstOrderDifferentialEquations;
8 import org.apache.commons.math3.ode.FirstOrderIntegrator;
9 import org.apache.commons.math3.ode.nonstiff.
10     ClassicalRungeKuttaIntegrator;
11
12 public class TestTresCuerpos {
13
14     public static void main(String[] args) throws IOException {
15         // Introducimos el paso h
16         FirstOrderIntegrator e3c = new
17             ClassicalRungeKuttaIntegrator(0.0017065);
18         // Creamos un objeto de la clase TresCuerpos
19         FirstOrderDifferentialEquations ode = new TresCuerpos
20             ();

```

```
18
19 //Datos del PVI
20 double [] y0 = { 0.994, 0, 0,
21                -2.00158510637908252240537862224 };
22 double [] y = { 0.994, 0, 0,
23                -2.00158510637908252240537862224 }; //
24 // Inicializamos a cualquier valor, por ejemplo a y0.
25 double a = 0.0; // t0
26 double b = 17.0650000000002602;
27 double t = 0;
28 double h = 0.0017065;
29
30 // Creamos un archivo con los puntos para despu\`es
31 // representarlos.
32 String directorioEntrada = System.getProperty("user.
33 dir");
34 System.out.println("user.dir: " + directorioEntrada);
35
36 directorioEntrada = directorioEntrada + File.
37 separator + "src"
38 + File.separator + "org" + File.
39 separator + "tfg"
40 + File.separator + "ejemplos";
41 System.out.println(directorioEntrada);
42 String archivoDondeEsta = directorioEntrada + File.
43 separator
44 + "TresCuerpos.txt";
45 System.out.println(archivoDondeEsta);
46 File file = new File(archivoDondeEsta);
47
48 // Borra lo que hubiera en el archivo existente
49 PrintWriter pw = new PrintWriter(file);
50
51 // Escribe cabecera en el archivo
52 pw.println("Coordenada x"+"\\t"+"Coordenada y ");
53 while (t <= b) {
54     t = t + h;
55     // Integramos con el m\`etodo integrate
56     e3c.integrate(ode, a, y0, t, y);
57     // Salida archivo. Guardamos y[0], y[1].
58     // Nombre del archivo
59     // TresCuerpos.txt,
60     // guardado en mi proyecto.
61     pw.println(y[0]+ "\\t" + y[1]);
62 }
63 // Cerramos el archivo
64 pw.close();
65 }
```

57 }

Al igual que en el caso del brazo robótico, la ejecución del programa genera el archivo `TresCuerpos.txt` que contiene una serie de puntos que se han obtenido con el algoritmo Runge-Kutta Clásico, que utilizaremos para representar la solución con el programa **mjograph**, usando en este caso este archivo de texto. (Ver Fig 4.9)

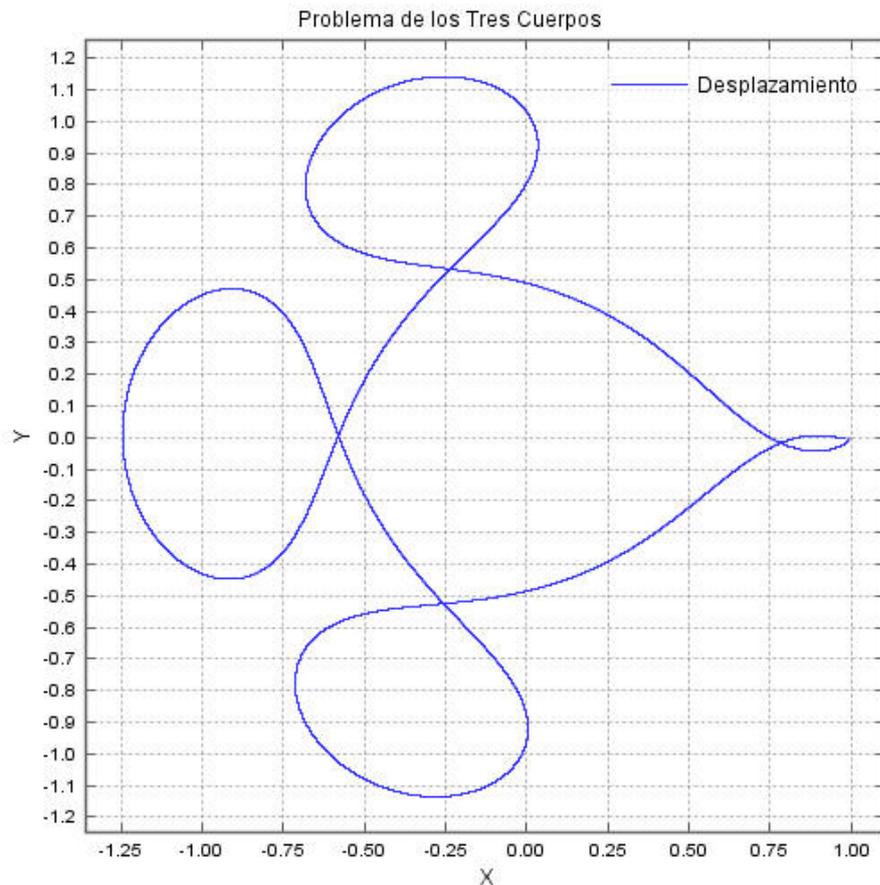


Figura 4.9: Solución del problema de la órbita de los tres cuerpos con Java

### 4.2.2. Resolución con Matlab

A continuación ejecutaremos el archivo `TresCuerpos.m` para resolver nuestro problema:

```
1 % Constantes
2 mu1 = 0.012277471;
3 mu2 = 1-mu1;
4
5 % PVI
```

```

6  f = @(x,y) [y(3);y(4);y(1)+2*y(4)-((mu2*(y(1)+mu1))./(((y(1)+mu1)
   .^2+...
7     y(2).^2).^2).^2))-((mu1*(y(1)-mu2))./(((y(1)-mu2).^2+...
8     y(2).^2).^2).^2));y(2)-2*y(3)-((mu2.*y(2))./(((y(1)+mu1).^2+...
9     y(2).^2).^2).^2))-((mu1.*y(2))./(((y(1)-mu2).^2+y(2).^2).^2).^2))]
   ];
10
11
12  % Datos
13  a = 0;
14  b = 17.0650000000002602;
15  y0 = [0.994;0;0;-2.00158510637908252240537862224];
16  N = 10000;
17
18  % Paso para los nodos
19  h = (b-a)/N;
20
21  % Tamano del sistema
22  m = length(y0);
23
24  % Construimos los nodos
25  x = a+h.*[0:1:N];
26  y(:,1)=y0;
27
28  % Aplicamos el metodo Runge-Kutta
29  for i=1:N
30     k1 = feval(f,x(i),y(:,i));
31     k2 = feval(f,x(i)+h/2,y(:,i)+(h/2)*k1);
32     k3 = feval(f,x(i)+h/2,y(:,i)+(h/2)*k2);
33     k4 = feval(f,x(i)+h,y(:,i)+h*k3);
34     y(:,i+1) = y(:,i)+(h/6)*(k1+2*k2+2*k3+k4);
35  end
36
37  % Representamos la solucion
38  plot(y(1,:),y(2,:), 'r')
39  title('Problema de los tres cuerpos')
40  xlabel('x')
41  ylabel('y')
42  legend('Desplazamiento')
43
44  % Guardar datos en archivo
45  fi = fopen('salidaTresCuerpos.txt', 'w')
46
47  for k =1:10001
48  fprintf(fi, '%f %f\n', y(1,k),y(2,k));
49  end
50  fclose(fi)

```

Al igual que en el problema anterior, tras la ejecución del archivo obtenemos un conjunto de puntos con los que obtenemos la representación de la solución, la cual podemos ver en la Figura 4.10.

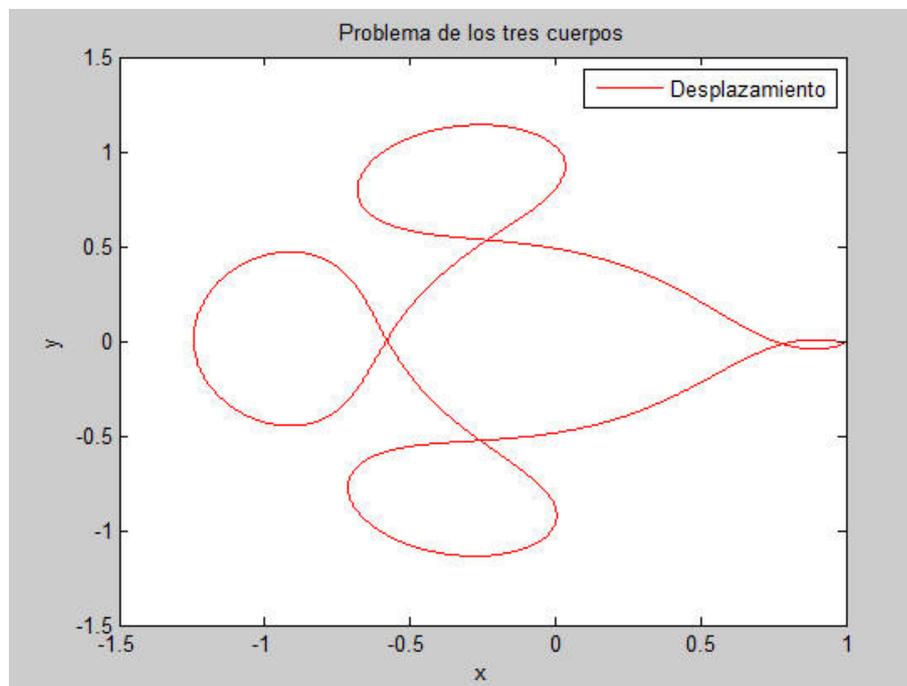


Figura 4.10: Solución del problema de la órbita de los tres cuerpos con Matlab

### 4.3. Discusión

Se ha trabajado en la resolución de problemas con solución numérica utilizando dos alternativas distintas que nos permitiesen ir comprobando los resultados obtenidos, así como el modo de abordarlos en cada caso.

Como hemos podido ver en ambos ejemplos, utilizamos las mismas clases adaptándolas al problema en particular que tratemos en cada momento. En la Figura 4.11 mostramos un diagrama de clases UML (Unified Modeling Language) donde representamos las clases que hemos usado, así como la relación que existe entre ellas. En el paquete `org.apache.commons.math3.ode.nonstiff` tenemos la clase `ClassicalRungeKuttaIntegrator` que a su vez hereda de una clase abstracta, `AbstractIntegrator`, del paquete `org.apache.commons.math3.ode`.

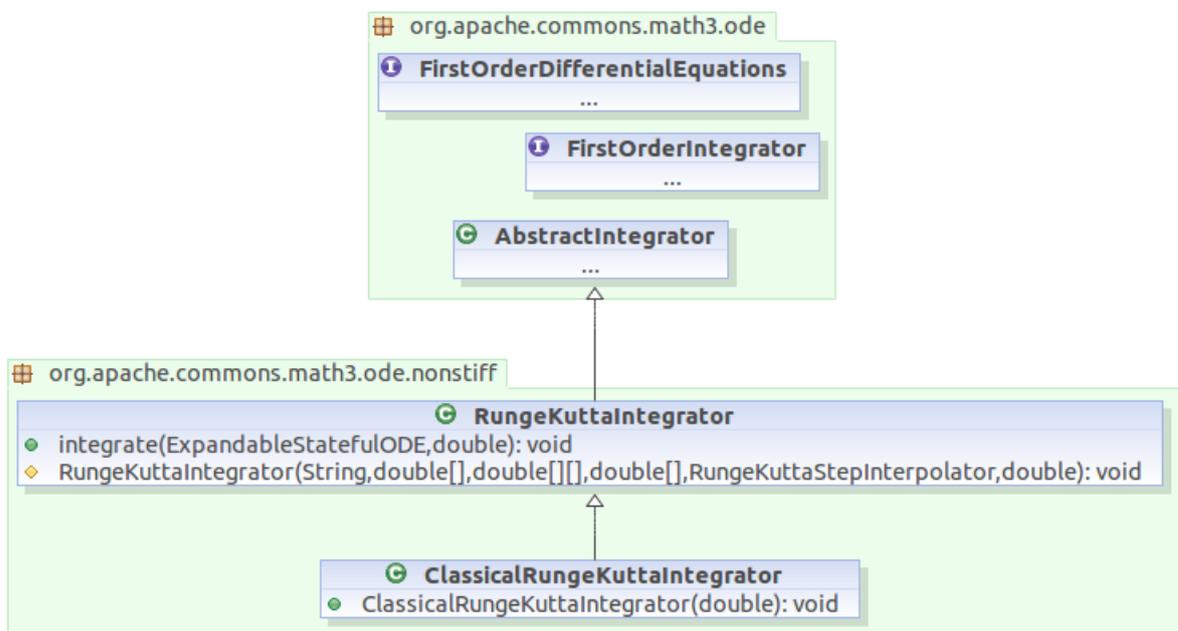


Figura 4.11: Diagrama de clases UML

Se ha hecho una aproximación al estudio de la multitud de clases que están disponibles para resolver ecuaciones diferenciales, abriendo un horizonte de estudio más profundo en un futuro dada su complejidad con nuestro nivel de conocimiento.

La codificación me ha permitido un entendimiento más profundo del problema y un control más al detalle del mismo. Ésta es una de las grandes ventajas de la librería *Apache Commons Math*: su carácter de código abierto (“open source”). Cualquier programador puede tener acceso al código de la librería, estudiarlo y reutilizarlo para la elaboración de proyectos propios.

En la siguiente imagen vamos a ver la vista que cualquier usuario podría tener una vez descargada y conectada la librería a su repositorio, que es la misma que tendría un programador encargado de actualizar y modificar la *Apache Commons Math*.

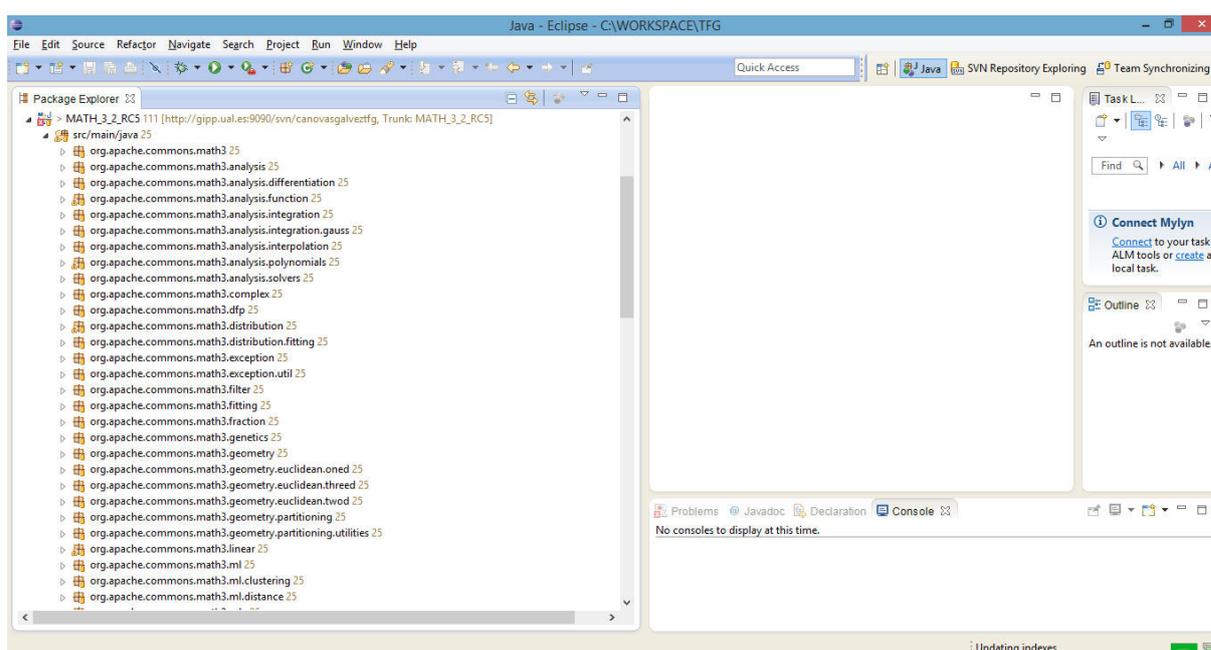


Figura 4.12: Visión de la librería *Commons Math* en el repositorio

Otra gran ventaja, es que no existe dependencia por parte de la *Apache Commons Math* con otras librerías. Una vez se esté trabajando dentro de esa librería no es necesario recurrir a otras para el correcto funcionamiento de sus clases.

Al ser código abierto pueden surgir aportaciones o correcciones de errores existentes en el código por parte de colaboradores con la Commons Apache, lo cual hace que se vaya actualizando la librería. La versión más reciente es la 3.3.

Un buen conocimiento matemático junto con un adecuado nivel de programación proporciona un amplio abanico de posibilidades laborales en el mundo de la programación.

---

# Capítulo 5

## Conclusión

Tras el estudio realizado se ha llegado a las siguientes conclusiones en base a los objetivos propuestos:

1. Se ha estudiado la familia de los métodos Runge-Kutta para la resolución de problemas de ecuaciones diferenciales. Dicho estudio nos ha permitido comprobar la amplitud y la importancia de esta familia.
2. La reutilización de librerías matemáticas de código abierto, como la utilizada en este trabajo: *Apache Commons Math*, que están construidas utilizando herramientas y procesos de desarrollo software actuales, e implementadas sobre plataformas bien establecidas y documentadas como lo es Java, permiten programar problemas de ecuaciones diferenciales de forma sencilla.
3. En este trabajo se han presentado varios ejemplos de reutilización (problema del brazo robótico y el problema de los tres cuerpos) y se han comprobado los resultados con los obtenidos con otra herramienta, privativa, como lo es Matlab.
4. Se ha podido comprobar que un usuario de estas librerías del perfil: matemático con suficientes, pero sólidos, conocimientos de programación puede acceder a la reutilización, y también , a la creación de nuevas librerías que implementan algoritmia matemática.

---

# Bibliografía

- [1] Página web de eclipse. URL <http://www.eclipse.org>.
- [2] Página web de la librería apache commons. URL <http://commons.apache.org/>.
- [3] Walter F. Alexander, A. *Edición de textos científicos L<sup>A</sup>T<sub>E</sub>X*. Instituto tecnológico de Costa Rica., 2013.
- [4] Página web jabref. URL <http://jabref.sourceforge.net/>.
- [5] J.C Butcher. *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and general methods*. John Wiley & Sons, 2008.
- [6] F. Quirós. Apuntes de métodos numéricos ii, . URL [http://www.uam.es/personal\\_pdi/ciencias/fquiros/Numerico2\\_05\\_06](http://www.uam.es/personal_pdi/ciencias/fquiros/Numerico2_05_06).
- [7] F. Quirós. El problema del valor inicial: Métodos runge-kutta, . URL [http://www.uam.es/personal\\_pdi/ciencias/fquiros/Numerico2\\_03\\_04/capitulo2.pdf](http://www.uam.es/personal_pdi/ciencias/fquiros/Numerico2_03_04/capitulo2.pdf).
- [8] Gutiérrez A. López M. Luri X. Prades D. Gómez, J. *Programación en Java para físicos e ingenieros*. Universitat de Barcelona, 2012.
- [9] Granell C. Erdozain M.C Belmonte, O. *Desarrollo de proyectos informáticos con tecnología Java*. Universitat Jaume, 2012.
- [10] Página web de la librería commons math. URL <http://commons.apache.org/proper/commons-math/>.
- [11] Link de descarga de la librería apache commons math. URL [http://commons.apache.org/proper/commons-math/download\\_math.cgi](http://commons.apache.org/proper/commons-math/download_math.cgi).
- [12] Ramos D. Martínez, A. *Apuntes de Simulación Numérica*. 2014.
- [13] M. Patterson. *Implementing Runge-Kutta solvers in Java*. PhD thesis, Acadia University, 2003.
- [14] Página web de commons math ordinary differential equations. URL <http://commons.apache.org/proper/commons-math/userguide/ode.html>.
- [15] Página web del editor gráfico mjograph. URL <http://www.ochiaailab.dnj.ynu.ac.jp/mjograph/index.html>.

---

# Apéndices

---

# Apéndice A

## Programación en Java y entorno Eclipse

### A.1. Programación en Java

#### A.1.1. Origen del lenguaje de programación en Java

El lenguaje de programación Java tiene sus orígenes en un lenguaje de programación anterior, llamado *Oak* (roble en inglés), que nació de un proyecto interno en *Sun Microsystems* en el año 1991 llamado *Green project*.

*Oak* fue creado con el objetivo de ser el lenguaje de programación con el que programar dispositivos electrónicos domésticos, en particular aparatos de televisión inteligentes e interactivos.

El proyecto de televisión inteligente e interactivo nunca se materializó. De modo simultáneo, a principios de la década de los 90 surgió *Internet* y con ella, la aparición de los primeros *navegadores web*. Los líderes del *Green project* fueron conscientes de la importancia que iba a tener *Internet* y orientaron su lenguaje de programación *Oak* para que programas escritos en este lenguaje se pudiesen ejecutar dentro del navegador web *Mozilla*. Y éste fue el inicio de *Java*, así llamado porque se intentó registrar el nombre *Oak* y éste ya estaba registrado.

La ventaja de que un programa escrito en Java se puede ejecutar en una gran cantidad de plataformas ha hecho de él un interesante lenguaje de programación por su “universalidad”.

#### A.1.2. Características de Java

Java es un lenguaje de programación orientado a objetos y de propósito general que toma de otros lenguajes de programación algunas ideas fundamentales, en particular toma de *Smalltalk* el hecho de que los programas Java se ejecutan sobre una máquina virtual. Y del lenguaje de programación C++ toma su sintaxis. De entre las muchas características que Java posee destacamos:

- Java es multiplataforma.

- Java es seguro.
- Java tiene un amplio conjunto de bibliotecas estándar.
- Java incluye una biblioteca portable para la creación de interfaces gráficas de usuario (AWT en Java 1.0/1.1, JFC/Swing en Java 2 y Java FX).
- Java simplifica algunos aspectos a la hora de programar.

### A.1.3. El entorno de desarrollo integrado Eclipse

Un entorno integrado de desarrollo o IDE de sus siglas en inglés (Integrated Develop Enviroment) nos permite escribir código de un modo cómodo. La comodidad reside en que los entornos de desarrollo integrado son mucho más que un simple editor de textos.

Eclipse reúne todas las características comunes a los modernos IDE. Además posee un sistema de *plug-ins* con los que se pueden añadir nuevas funcionalidades. Por ejemplo, mediante un *plug-in* nos podemos conectar al sistema de control de versiones *Subversion*.

#### Descarga e instalación de Eclipse

Eclipse se puede descargar desde el sitio web <http://www.eclipse.org>. Existen versiones para las principales plataformas y sistemas operativos.

Una particularidad de Eclipse es que no necesita instalación. Una vez descargado el fichero comprimido, lo único que debemos hacer para empezar a utilizarlo es descomprimirlo en algún directorio y seguidamente ejecutar el archivo correspondiente. La Figura A.1 muestra la página de inicio de Eclipse. Los iconos que muestra esta pantalla son enlaces a sitios de información sobre Eclipse.

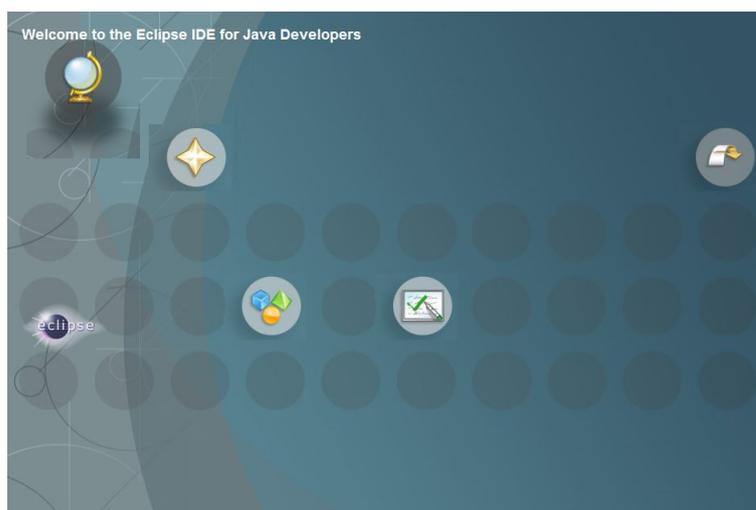


Figura A.1: Pantalla inicial de *Eclipse*

En la Figura A.2 vemos la perspectiva inicial de Eclipse y las diferentes partes de esta ventana principal:

1. Explorador de paquetes.
2. Editor de código.
3. Área compuesta por múltiples vistas que podremos seleccionar según nuestras necesidades. Entre ellas destacamos “Problems”, que mostrará mensajes de error en tiempo de compilación, “Console”, que será donde se muestren las salidas de ejecución.

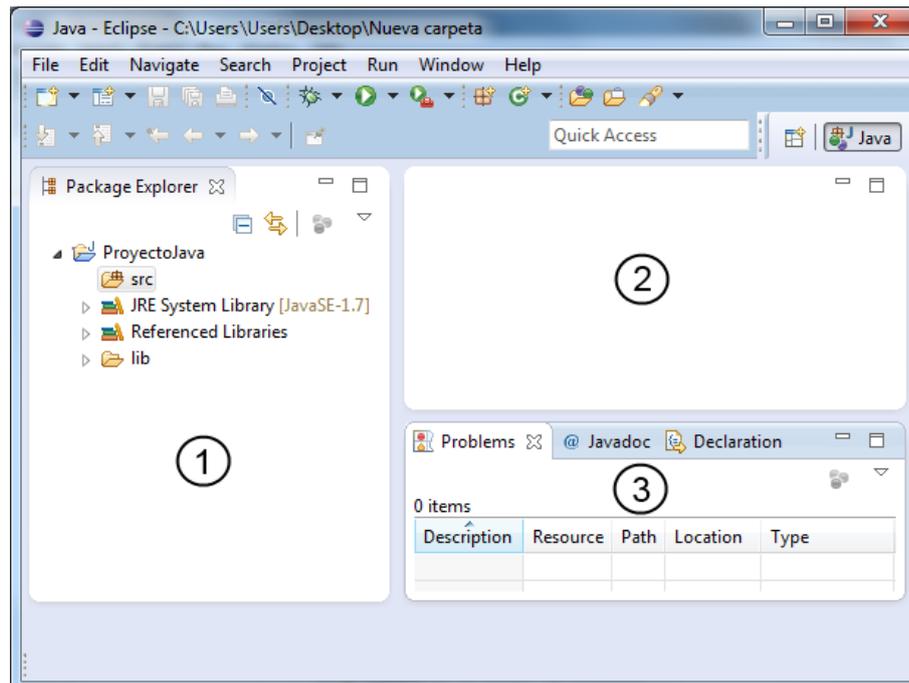


Figura A.2: Perspectiva inicial orientada al desarrollo de proyectos Java2 SE.

## Perspectivas

Una perspectiva de Eclipse es una agrupación de vistas y editores de manera que den apoyo a una actividad completa del proceso de desarrollo software. Los editores normalmente permiten realizar una tarea completa, las vistas proporcionan funciones de apoyo o auxiliares tales como mostrar información, requerir datos, etc. Las perspectivas pueden seleccionarse haciendo clic en los iconos de perspectiva en la esquina superior derecha o eligiendo **Window**→**Open Perspective** del menú.



Figura A.3: Perspectivas

Nosotros hemos trabajado con las siguientes perspectivas:

- **Java:** esta perspectiva se centra en tareas de programación, mostrando paquetes, clases, métodos y atributos en sus vistas asociadas.
- **Debug:** esta perspectiva se abre cuando lanzamos una ejecución con el depurado. Un depurador es una herramienta que permite seguir el programa línea a línea y ver cómo cambian los valores de las variables y expresiones que tengamos seleccionadas durante la ejecución de un programa. Permite, por tanto, controlar y analizar la ejecución del programa. Ayuda a localizar diferentes tipos de errores.

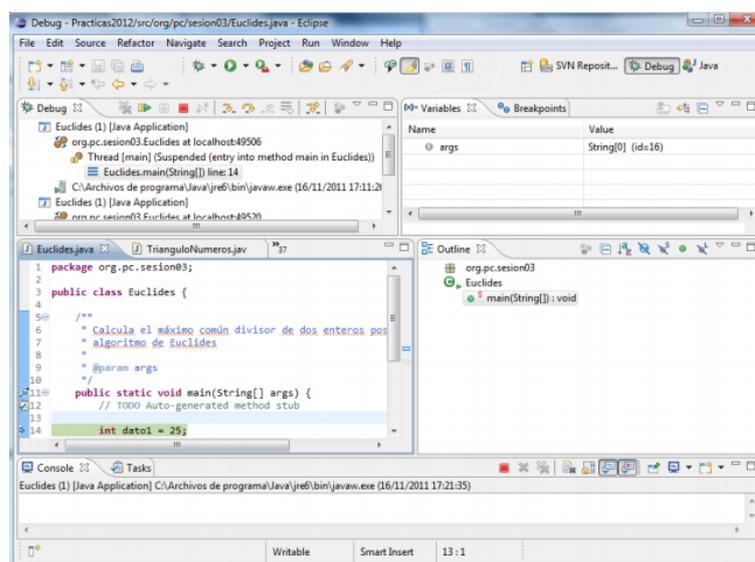


Figura A.4: Perspectiva Depurador

- **SVN:** esta perspectiva se centra en el trabajo con repositorios. Permitirá establecer conexiones con diferentes repositorios, acceder a los mismos, navegar por ellos, etc... En el siguiente apartado desarrollaremos esta perspectiva.

## A.1.4. Subversion

### ¿Qué es un sistema de control de versiones?

Un sistema de control de versiones es una herramienta software que, de manera automática, se encarga de facilitar la gestión de las versiones del código de un proyecto de manera centralizada.

### Principales características de *Subversion*

El concepto central en *Subversion* es el *Repositorio*. Por repositorio se entiende la última versión del proyecto que existe en el sistema de control de versiones.

El paradigma que Subversion utiliza es *Copia-Modificación-Fusión* (Copy-Modify-Merge en inglés). En este paradigma, cada uno de los miembros del equipo, cuando empieza a trabajar en el proyecto, hace una copia local del contenido del repositorio; modifica su copia local y finalmente fusiona sus modificaciones locales con el código del repositorio. Al finalizar esta fase, se dice que se ha creado una nueva versión del proyecto en el repositorio.

Una de las características principales de *Subversion* es que las actualizaciones en el repositorio son incrementales, sólo se actualizan los ficheros que se han modificado respecto a la versión anterior. Otra característica es relativa a la numeración de la versión del repositorio, cada vez que se realiza una modificación en el repositorio, se actualiza la versión de todos los ficheros existentes en el repositorio y no únicamente de los ficheros que se han modificado.

### Trabajar con Repositorios

Debemos abrir la perspectiva de **SVN Repository Exploring** y crear el repositorio con el icono de nuevo repositorio. Para hacer la conexión daremos la dirección, autenticación y clave necesaria para acceder al repositorio. (Ver Figura A.5)

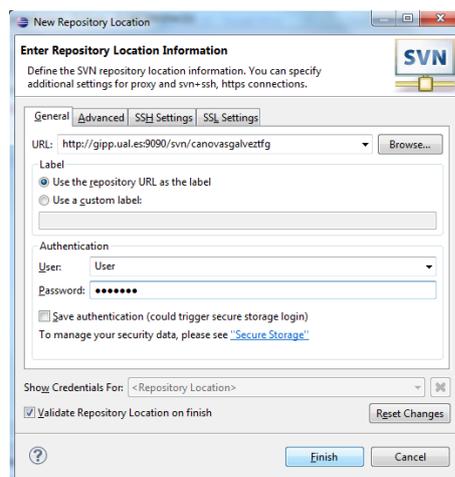


Figura A.5: Nuevo Repositorio

Ahora vamos a ver una serie de opciones que nos permite trabajar con los repositorios:

- **Check out:** Para traernos un proyecto Java desde la perspectiva del repositorio a nuestra perspectiva Java. En la perspectiva SVN: **Botón derecho sobre el proyecto** → **Check out**.
- **Commit:** Una vez trabajado en nuestro proyecto en la perspectiva Java, para subirlo al repositorio tenemos que hacer un Commit. En la perspectiva Java: **Botón derecho sobre el proyecto** → **Team** → **Commit**.
- **Update:** Actualiza el proyecto. Trae las modificaciones del repositorio al proyecto. En la perspectiva Java: **Botón derecho sobre el proyecto** → **Team** → **Update**.