https://doi.org/10.17979/spudc.9788497498081.0121

LIBRERÍA JAVA PARA ANÁLISIS Y SIMULACIÓN DE SISTEMAS LINEALES

Jordi Blanch Costa, Ramon Costa Castelló
Escola Tècnica Superior d'Enginyeria Industrial de Barcelona (ETSEIB)
Universitat Politècnica de Catalunya (UPC)
jordi.blanch.costa@gmail.com, ramon.costa@upc.edu

Resumen

Este proyecto presenta el diseño, implementación y algunos ejemplos de uso de una librería JAVA pensada para el análisis y simulación de sistemas lineales. Se ha diseñado pensando en que sea fácilmente integrable con Easy Java Simulations. El uso combinado de ambas herramientas facilita el desarrollo de laboratorios virtuales y remotos.

Palabras Clave: Sistemas lineales, laboratorios virtuales, librería de cálculo.

1 INTRODUCCIÓN

En los últimos tiempo ha habido un auge en el desarrollo de herramientas interactivas para el diseño y la docencia de sistemas de control. Ello ha sido propiciado por la aparición de diferentes herramientas que facilitan su desarrollo, entre otras cabe destacar Sysquake y Easy Java Simulations (EJS). Sysquake está más orientado al desarrollo de herramientas interactivas mientras que EJS está más orientado al desarrollo de laboratorios virtuales y remotos. EJS es un generador de código JAVA que ofrece una interfaz totalmente gráfica e interactiva que permite diseñar las aplicaciones sin apenas conocer JAVA.

Actualmente, EJS no dispone de una librería JAVA para poder trabajar directamente con sistemas de control lineales. Para expandir este software hacia estos campos, era necesario crear una librería capaz de trabajar no solo con sistemas de control lineales, sino también con números complejos, ecuaciones y funciones de transferencia [7].

Antes de empezar con el proyecto, fue necesaria una búsqueda de diferentes opciones que ya existían. De las opciones encontradas, la librería Flanagan [6] (creada por Michael Thomas Flanagan) era una de las más completas y disponía ya de clases JAVA para poder trabajar con números complejos y funciones.

Partiendo con esta base sólida, ya sólo faltaba definir, diseñar, crear y depurar la librería para trabajar con funciones de transferencia (tanto en tiempo continuo como en tiempo discreto).

La clase resultante, será llamada JLSL (Java Lineal Systems Library)

2 IMPLEMENTACIÓN DE LA LIBRERÍA

2.1 ESTRUCTURA DE LA LIBRERÍA

Para crear una librería que trabaje con sistemas de control lineales, es necesario crear una librería que sea capaz de trabajar con funciones de transferencia. Para esto, hay que entender como puede un computador interpretar los diferentes componentes de una función de transferencia y ser capaz de trabajar con ellos.

Por facilidad y eficiencia, se ha creado una clase JAVA principal que usa el diminutivo de TF que proviene de Transfer Function. Esta clase es la responsable de definir la estructura de función de transferencia y poder trabajar con ella (modificar raíces, encontrar elementos, resolver por ciertos valores, etc.).

Dado que las funciones de transferencia pueden definirse tanto en tiempo continuo como tiempo discreto se ha definido una clase JAVA abstracta, de la cual van a derivar las clases de tiempo continuo (CTF, Continuous Transfer Fucntion) y tiempo discreto (DTF, Discrete Transfer Fucntion) como se puede ver en la Figura 1.

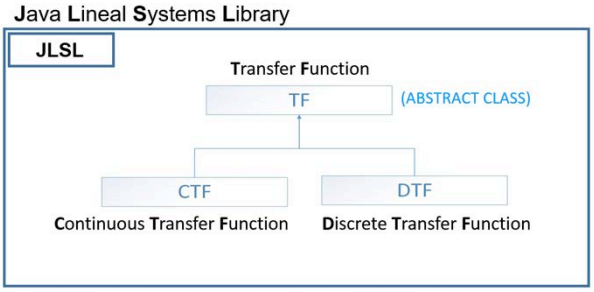

Figura 1 : Diagrama de las clases.

2.2 ESTRUCTURA BASE INTERNA DE LAS CLASES

Las funciones de transferencia pueden estar definidas de dos maneras; de forma polinómica o de forma factorizada.

$$\frac{a_1 + a_2X + a_3X^2 + \cdots + a_mX^{m-1}}{b_1 + b_2X + b_3X^2 + \cdots + b_nX^{n-1}} \quad (1)$$

$$k \cdot \frac{(X - z_1)^a \cdot (X - z_2)^b \cdot \ldots \cdot (X - z_m)^c}{(X - p_1)^d \cdot (X - p_2)^e \cdot \ldots \cdot (X - p_n)^f} \quad (2)$$

Cuando se trata de hablar en términos de control la forma factorizada proporciona más información a simple vista. Además, si los programas deben modificar polos, ceros, ganancias y retrasos; esta representación reduce los tiempos de cómputo además de minimizar la pérdida de precisión numérica. Tomada ya esta decisión, hace falta traducir a una clase JAVA la función de transferencia (Figura 2).

Aprovechando las clases de números complejos y polinomios que proporciona la librería Flanagan, la descomposición y traducción óptima de una función de transferencia factorizada a variables y clases JAVA, sería la siguiente:

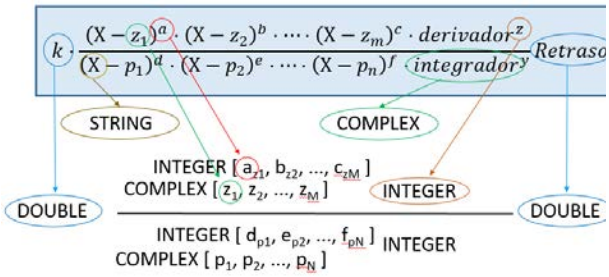

Figura 2 : Estructura interna de la clase.

La Tabla 1 muestra los componentes de la clase TF (y las clases CTF y DTF).

OBJETO	NOMBRE
Complex[]	ceros;
Complex[]	polos;
int[]	multiplicidadCeros;
int[]	multiplicidadPolos;
int	integradores;
int	derivadores;
double	ganancia;
double	retraso;
String	variable;
Complex	raizSingular;

Tabla 1 : Estructura interna de un objeto TF.

Se ha introducido los elementos "número de Integradores y derivadores" que son variables tipo enteras básicas, pues se considera que integradores y derivadores son elementos singulares que deben tratarse diferenciadamente, de forma similar se guarda la multiplicidad de cada elemento para evitar que la pérdida de precisión extravíe esta información

2.3 MÉTODOS

Las tres clases, TF, CTF y DTF, disponen, mayoritariamente, de los mismos métodos públicos para tratar con sistemas lineales (a excepción de algunos casos, como transformadas bilineales, sólo accesible en DTF).
TF define los métodos compartidos de manera abstracta, obligando así que las clases derivadas (CTF y DTF) deban implementarlas adaptadas a sus características propias.
Se han desarrollado un gran número de clases privadas y protegidas, que realizan operaciones intermedias o adaptaciones de valores a representar/devolver.
Los métodos públicos (los accesibles por los usuarios), se podrían separar en dos grupos; los que sirven para tratar, modificar y obtener elementos de las funciones de transferencia, y los que sirven para realizar cálculos enfocados al control de sistemas.
Entre la lista de métodos del primer grupo, existen:

- Añadir, eliminar y modificar elementos (polos, ceros, ganancias, retrasos,…)
- Funciones/señales básicas (Dirac, Heaviside, rampa, cosenos, etc.)
- Interconnexión de sistemas:
  - Serie
  - Paralelo positivo y negativo
  - Realimentación positiva y negativa
- Transformación en fracciones simples
- Escritura por pantalla en LaTeX [1]:
  - Escritura factorizada
  - Escritura multiplicidad
  - Escritura polinómica

La lista de métodos del segundo grupo, se destaca:

- Estabilidad de sistemas
- Evaluar por un valor
- Constantes de tiempo, velocidad y aceleración
- Respuesta frecuencial (dB, fase, fase en grados)
- Transformada Bilineal
- Anti-transformada Bilineal
- Diagrama de Bode
- Diagrama de Bode asintótico
- Diagrama de Nyquist
- Diagrama de Nichols
- Respuesta temporal

3 EJEMPLOS DE USO

Para hacer un ejemplo de uso de la librería, sin el EJS, se va a calcular la respuesta temporal del siguiente sistema delante una entrada en forma de escalón de amplitud 12:

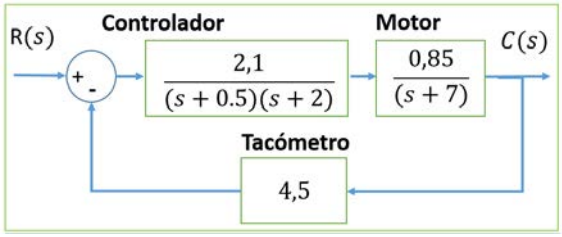

Figura 3 : Ejemplo de sistema de lazo cerrado.

Para empezar, primero de todo se deben crear los objetos "controlador", "motor" y "tacómetro".

Una vez hecho, se debe crear el objeto que represente el sistema total y, esto se realizará mediante los métodos de construcción de sistemas.

Finalmente, construido el sistema total, se procederá a calcular la respuesta temporal del sistema frente a un escalón de valor 12.

A continuación, se muestran las líneas de código Java que permitirían crear estos objetos y realizar los pasos mencionados.

3.1 CREACIÓN DE OBJETO CONTROLADOR

Como en toda clase JAVA, existen distintos constructores que permiten crear los objetos de maneras distintas.
Los principales constructores de una clase DTF se pueden ver la Figura 4.

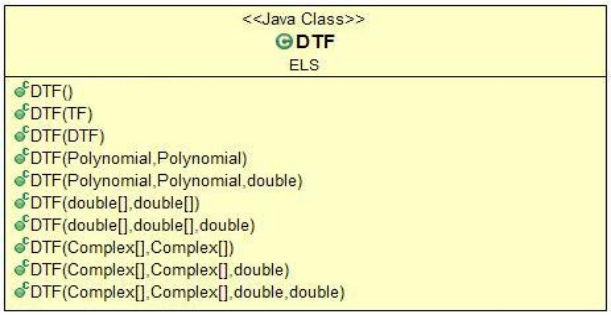

Figura 4 : Constructores de una clase DTF.

Empezando creando el controlar, el código JAVA sería el siguiente:

```
Complex[] poles = new Complex[2];
Poles[0] = new Complex(-0.5, 0);
Poles[1] = new Complex(-2, 0);
CTF control = new CTF(null, poles, 2.1, 0);
```

OBJETO	VALOR
Complex[]	null
Complex[]	{(-7,0)}
int[]	null
int[]	(1)
int	0
int	1
double	0.85
double	0
String	s
Complex	(0,0)

Tabla 2 : Construcción del controlador

3.2 CREACIÓN DE OBJETO MOTOR

```
CTF motor = new CTF();
motor.addPole(new Complex(-7,0));
motor.setGain(0,85);
```

OBJECTO	VALOR
Complex[]	null
Complex[]	{(-7,0)}
int[]	null
int[]	{(1)}
int	0
int	1
double	1
double	0
String	s
Complex	(0,0);

Tabla 3 : Construcción del motor

3.3 CREACIÓN DE OBJETO TACÓMETRO

```
CTF tacho = new CTF(null, null, 4.5, 0);
```

OBJECTO	VALOR
Complex[]	null
Complex[]	null
int[]	null
int[]	null
int	0
int	0
double	1
double	0
String	s
Complex	(0,0);

Tabla 4 : Construcción del tacómetro.

3.4 CREACIÓN DEL SISTEMA ENTERO

Para la creación del sistema de lazo cerrado a partir de sus componentes, se han incorporado métodos que permiten generar sistemas de orden superior a partir

de sistemas sencillos. En particular se ha incorporado la conexión serie, paralelo y en realimentación.

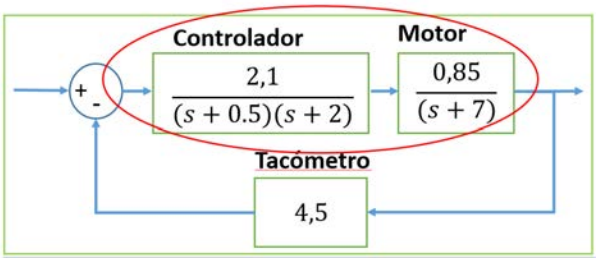

Figura 5 : Sistema con controlador y motor en serie.

Por ejemplo si se desea obtener el sistema obtenido de la combinación en serie del control y el controlador, que se muestran en la Figura 5, el código que debería utilizarse es:

```
CTF system = motor.getSeries(control);
```

El contenido de objeto retornado se muestra en la Tabla 5.

OBJECTO	VALOR
Complex[]	null · null
Complex[]	{(-0.5, 0); (-2, 0); (-7, 0)}
int[]	null
int[]	{1; 1; 1}
int	0 + 0
int	0 + 0
double	2.1 x 0.85 = 1.78
double	0 + 0
String	s
Complex	(0,0);

Tabla 5 : Controlador y motor en serie

Para obtener la función de transferencia del sistema que se muestra en la Figura 5 es necesario aplicar la realimentación negativa, la operación interna que realiza la clase es la de trabajar con numeradores y denominadores (polinomios):

$$\frac{C(s)}{R(s)} = \frac{CM(s)}{1 + CM(s)T(s)} = \frac{NCM(s) \cdot DT(s)}{DCM(s) \cdot DT(s) + NCM(s) \cdot NT(s)} \quad (3)$$

$$\frac{1{,}785}{(s + 1.14 - 0.89j)(s + 1.14 + 0.89j)(s + 7{,}23)} \quad (4)$$

El código usado sería el siguiente:

```
CTF system = system.getNegativeFeedback(tacho);
```

3.5 CÁLCULO DE LA RESPUESTA TEMPORAL

Para el cálculo de la respuesta temporal frente un escalón de valor 12, primero se debe crear la función escalón. Para esto, se pueden usar los métodos ya integrados en la librería:

```
CTF escalon = new CTF().heaviside[12,0];
```

El primer valor de entrada (12) es el valor final y el segundo (0) corresponde al instante de tiempo en que se produce el escalón.
Una vez hecho esto, se deben indicar los instantes de tiempo en que se calculará la respuesta temporal. Para ello, se crea un vector de valores del tipo *double* y se rellana con los valores en los que calcular la respuesta temporal. En este caso, ser crea un vector de 8 segundos, con precisión de 0,01 segundos:

```
double time[] = new double[numSamples];
Time = {0, 0.01, 0.02, 0.03, 0.04, ….., 8}
```

En este momento, ya se puede llamar al método de respuesta temporal, dándole como entrada el escalón más el vector de tiempo y el método te retorna un vector *double* correspondiente al valor de la salida en los instantes seleccionados:

```
double[] response = system.temporalResponse(r, time);
```

Internamente, este método calcula la respuesta temporal mediante la transformación en fracciones simples y luego anti transformada directa.
A modo de ejemplo, si se desea calcula la transformada de Laplace inversa de:

$$\frac{12}{s} \cdot \frac{1{,}785}{(s + 1.14 - 0.89j)(s + 1.14 + 0.89j)(s + 7{,}23)} \quad (5)$$

Primeramente se descompone en fracciones sencillas de la forma:

$$\frac{B_i{}^r}{(s - p_i)^r} + \frac{B_i{}^{r-1}}{(s - p_i)^{r-1}} + \cdots + \frac{B_i{}^1}{(s - p_i)^1} \quad (6)$$

donde se obtienen de la siguiente forma:

$$B_i{}^k = \left[\frac{d^{r-k}}{dz^{r-k}}[(z - p_i)^r system(z)]\right] z = p_i \quad (7)$$

Teniendo los coeficientes, se crea la matriz resultante de 4 filas y número de columnas igual al número de polos del sistema dónde, en la fila 0 guarda los polos, en la fila 1 guarda las veces que este polo se ha repetido anteriormente, en la fila 3 la multiplicidad de este polo y en la fila 4 el coeficiente resultante para descomponer en fracciones simples (ver Figura 6).

$$Matriz[0][num\ polos] = \{-1.14 - 0.89j,\ -1.14 + 0.89j,\ -7.23,\ 0\}$$

$$Matriz[1][num\ polos] = \{1,\ 1,\ 1,\ 1\}$$

$$Matriz[2][num\ polos] = \{1,\ 1,\ 1,\ 1\}$$

$$Matriz[3][num\ polos] = \{-0.67 - 1.17j,\ -0.67\ 1.17j,\ -0.08,\ 1.41\}$$

Figura 6 : Matriz resultante de la descomposición en fracciones sencillas.

El resultado obtenido es el siguiente:

$$\frac{-0.67-1.17j}{(s+1.14+0.89j)^1}+\frac{-0.67+1.17j}{(s+1.14-0.89j)^1}+\frac{-0.08}{(s+7.23)^1}+\frac{1.41}{s} \quad (8)$$

A partir de esta matriz, directamente se obtiene la respuesta temporal aplicando las antitransformadas conocidas.

4 APLICACIONES

4.1 EDITOR DE POLOS Y CEROS

Como se ha comentado al inicio, esta librería está enfocada a poder ser usada en EJS. Para demostrar su funcionalidad, utilizad y potencial, se han creado 4 aplicaciones con EJS usando esta librería.

Las dos primeras han sido llamadas Editores de polos y ceros. Estas aplicaciones permiten de manera fácil, precisa, rápida, gráfica e intuitiva, definir un sistema a partir de sus polos y ceros. De esta manera se evita tener que escribir el código JAVA que formarían los sistemas y ganar tiempo. Dado que los lugares geométricos son diferentes en tiempo continuo y tiempo discreto se han desarrollado dos editores diferentes (ver Figura 7 y Figura 8).
La zona amarilla de los editores, indica el lugar geométrico de estabilidad.
Para añadir polos y ceros, sólo hace falta arrastrar un polo (X) o un cero (O) des del repositorio superior (Figura 9). Para eliminarlo, sólo hace falta devolverlo al repositorio (Figura 10).
Existen dos repositorios, de colores diferentes, cada uno representa un sistema diferente (que pueden estar en serie, paralelo, retroalimentados, indicado en otro apartado de la aplicación).

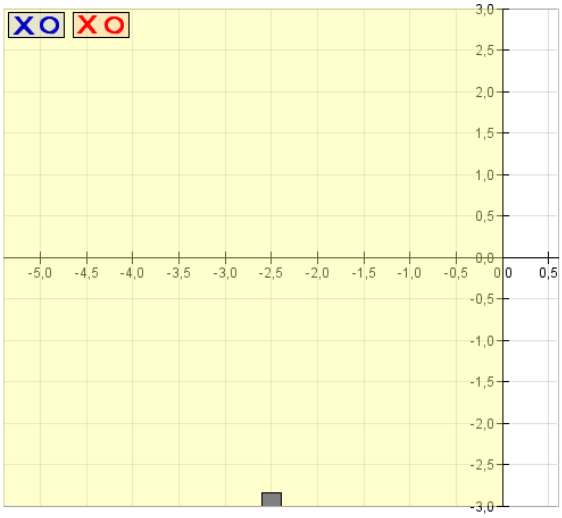

Figura 7 : Editor de polos y ceros en tiempo continuo.

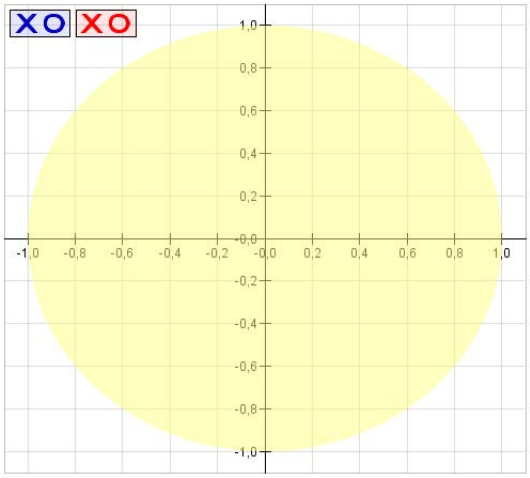

Figura 8 : Editor de polos y ceros en tiempo discreto.

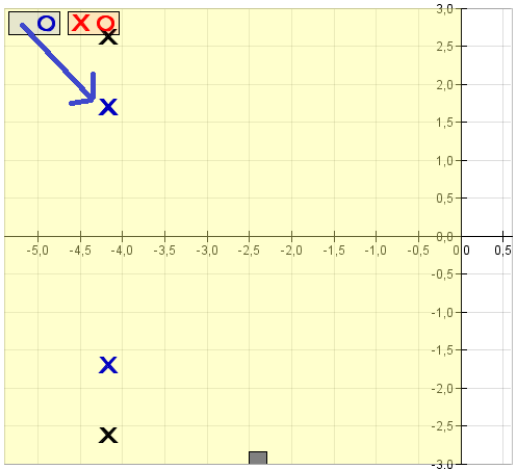

Figura 9 : Añadir raíces al sistema definido.

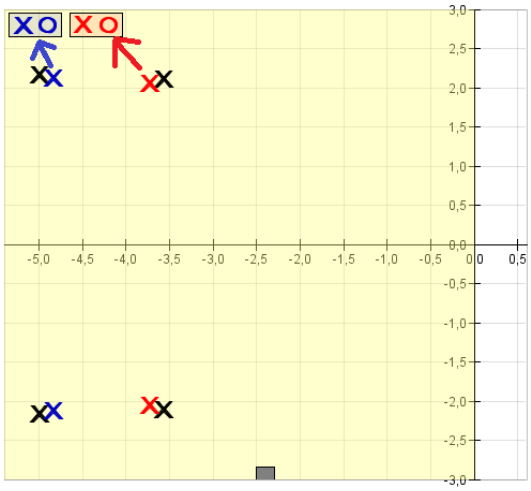

Figura 10 : Eliminar polos/ceros del sistema

Una vez en la zona de los ejes, estos se pueden modificar arrastrándolos en posiciones distintas y se puede ver su posición exacta en la parte inferior (Figura 11).

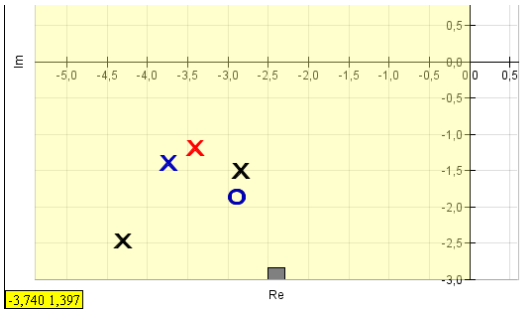

Figura 11 : Posición de los polos y ceros del sistema.

Los editores, disponen de diferentes ayudas para el posicionamiento de las raíces. Las primeras ayudas, se tratarían de zonas de influencia donde, si se deja el polo o cero en esa zona, se posiciona automáticamente en el punto crítico. Estos puntos críticos, se trata de los ejes, los puntos [0,0] y, en caso de trabajar en tiempo discreto, los puntos de módulo 1 (ver Figura 12 y Figura 13).

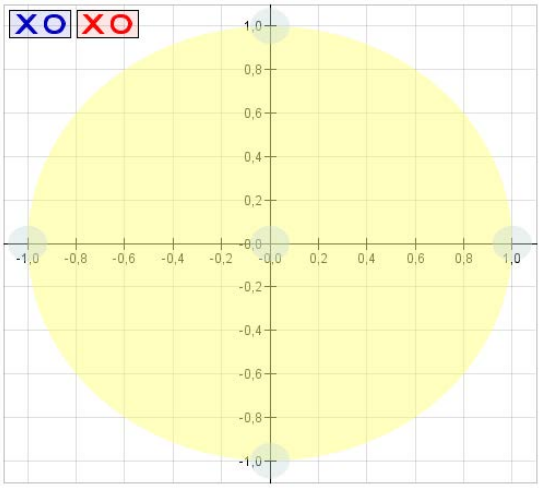

Figura 12 : Elementos de ayuda del editor de polos/ceros.

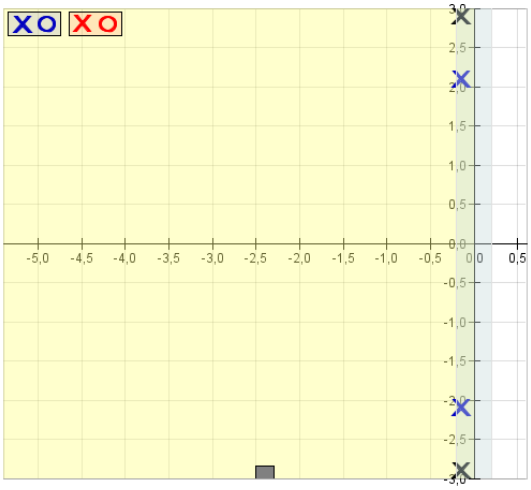

Figura 13 : Elementos de ayuda del editor de polos/ceros.

Finalmente, también se puede hacer doble selección a una raíz y posicionar exactamente cada polo y cero indicando la posición (Figura 14).

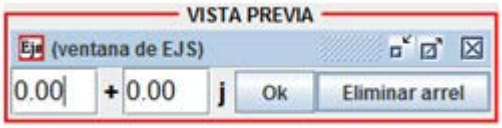

Figura 14 : Ventana de posicionamiento exacto.

4.2 APLICACIONES DE ANÁLISIS

Con el objetivo de validar la librería presentada y poner a prueba su potencial, se han creado dos aplicaciones, una para tiempo continuo y otra para tiempo discreto, que sirven para analizar funciones de transferencia y estudiar su comportamiento en diferentes dominios. Estas aplicaciones pueden ser de interés en si mismas para fines docentes.
La Figura 15 muestra la ventana principal de la aplicación de tiempo continuo. Esta dispone de una pestaña donde se puede escoger gráficamente la configuración del sistema con el que se trabaja.

Una vez escogida la configuración es posible definir los elementos que componen cada una de las funciones de transferencia utilizando el editor de polos y ceros, que ha sido integrado en la aplicación.

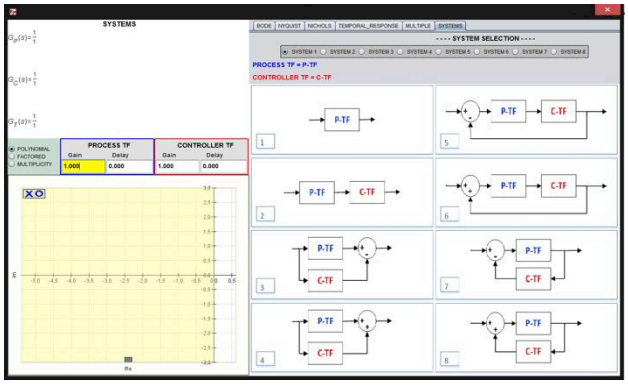

Figura 15 : Selección de la configuración con la que se trabaja.

Además de las representación mediante el mapa de polos y ceros es posible visualizar textualmente las diferentes funciones de transferencia. Esta visualización se genera automáticamente por las funciones que componen la librería. Esta visualización puede ofrecerse en diferentes formatos, la sea polinomialmente o en formato factorizado (ver Figura 16 y Figura 17).

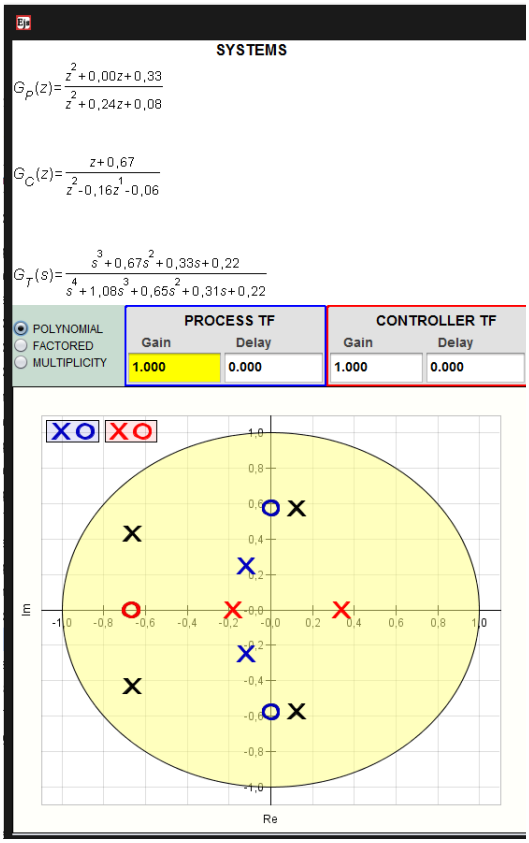

Figura 16 : Visualización textual (función de transferencia).

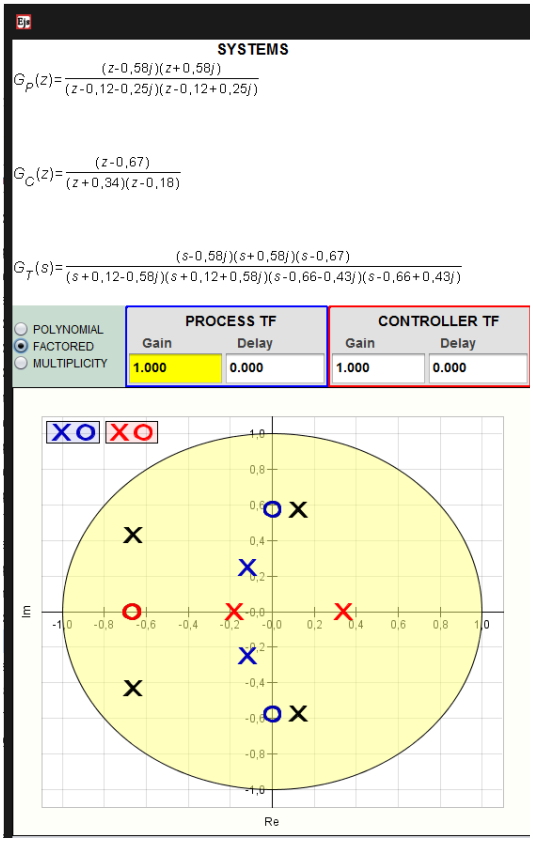

Figura 17 : Visualización textual (formato factorizado).

La aplicación dispone también de campos de texto, perfectamente integrados, que permiten definir las ganancias y retardos de los diferentes elementos.

Una vez definidas las configuración del sistema y todos sus componentes es posible analizar automáticamente el comportamiento del sistema en diferentes dominios. Por un lado el dominio frecuencial permite visualizar los diagrama de Bode, Nyquist y Nichols (Figura 18).

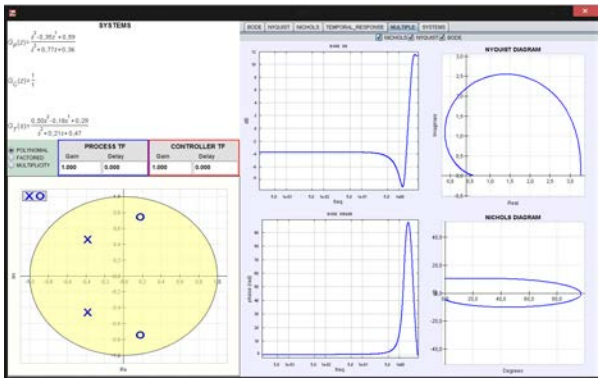

Figura 18 : Análisis de la respuesta frecuencial.

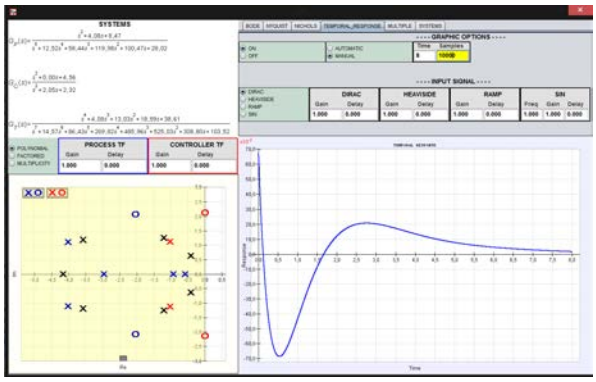

Figura 19 : Análisis de la respuesta temporal.

De forma similar es posible analizar la respuesta temporal del sistema frente a diferentes señales de entrada (Figura 19). Todas estas funcionalidades se actualizan automáticamente cuando el usuario modifica interactivamente la configuración del sistema o el valor de sus componentes. La aplicación ha sido validada con sistemas de diferentes órdenes, para sistemas de orden bajo (hasta 5 o 6) y con una resolución razonable de las respuestas en frecuencia y temporales el sistema responde interactivamente y apenas sin retraso.

En ocasiones interesa desarrollar aplicaciones con mayor precisión, por ello la librería incorpora unas funciones que permiten modifican la resolución, el número de muestras en las respuesta frecuencial y temporal. La Figura 20 y Figura 21 muestran el mismo diagrama de Nyquist con un número de muestras diferente. El número de muestras

seleccionado debe elegirse realizando un compromiso entre la precisión necesaria en la aplicación y el tiempo de respuesta de la aplicación. Números de muestras reducidos mejoran la interactividad aunque reducen la calidad de las representaciones. En las mayor parte de aplicaciones se puede llegar a decisiones de compromiso que garantizan una interactividad bueno con una precisión razonable.

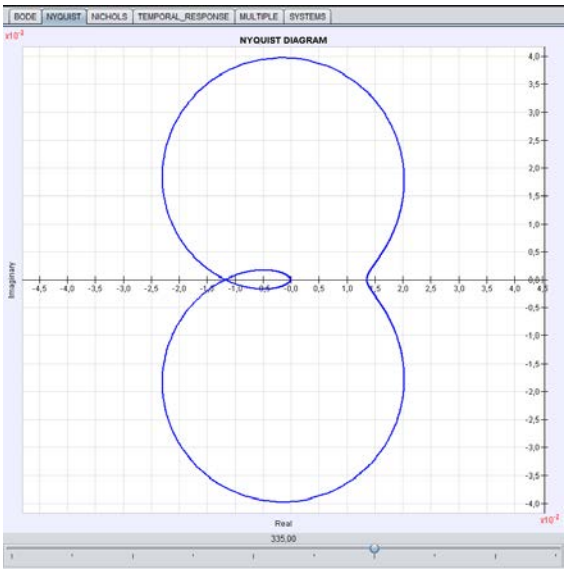

Figura 20 : Diagrama de Nyquist con alta resolución.

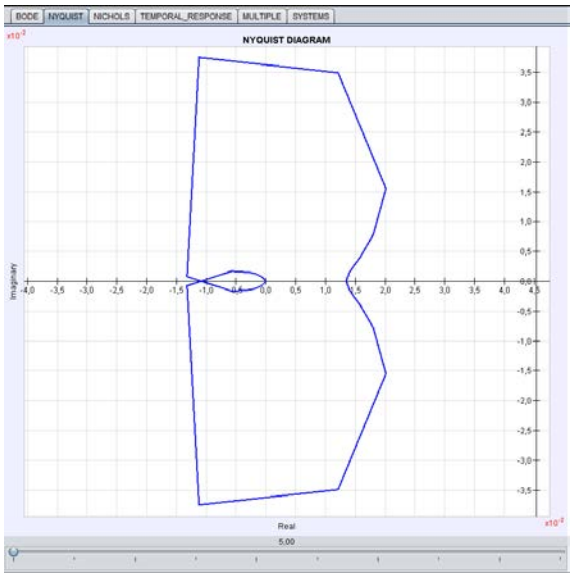

Figura 21 : Diagrama de Nyquist con baja resolución.

5 CONCLUSIONES

En este trabajo se ha presentado una librería JAVA pensada para gestionar y analizar sistemas lineales. Esta librería ha sido especialmente pensada para que sirva de complemento a Easy JAVA Simulations.

Gracias a esta librería, las posibilidades del EJS han aumentado y se pueden crear aplicaciones con que trabajen con sistemas lineales, muy intuitivas, resultando en un fácil y rápido manejo.

Dada esta nueva funcionalidad, se pueden hacer laboratorios virtuales y remotos para trabajar con sistemas de control lineales. Esto permitiría, por ejemplo, a profesores preparar prácticas universitarias para que los estudiantes las realicen en casa, crear exámenes a distancia y corregidos en tiempo real, simulaciones, etc.

También es importante mencionar la importancia de la agilidad, rapidez e intuición. Estos elementos ayudan mucho al rápido aprendizaje y a entender lo que se pretende enseñar.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto DPI2015-69286-C3-2-R (MINECO/FEDER) y el proyecto 2014 SGR 267 de la AGAUR (agencia de la Generalitat de Catalunya).

Referencias

[1] LaTeX Project. A document preparation system. [http://latex-project.org/, 04/06/2013].

[2] OGATA, K. Ingeniería de Control Moderna. Pearson Educación, S.A. (Prentice-Hall), 4ª Edición.

[3] OGATA, K., Sistemas de control en tiempo discreto. Prentice Hall Hispanoamericana, S.A., 2ª Edición.

[4] Ogata, Katsuhiko. Ingeniería de control moderna. 3ª ed. México D.F: Prentice Hall, 1998. ISBN 9701700481.

[5] ORACLE. Technology Network. Resources for Java Developers. [http://www.java.com/es/, 28/01/2013].

[6] UCL. DEPARTMENT OF ELECTRONIC AND ELECTRICAL ENGINEERING. Michael Thomas Flanagan's Java Scientific Library [http://www.ee.ucl.ac.uk/~mflanaga/java/, 08/06/2012].

[7] UNIVERSIDAD DE MURCIA. Departamento de matemáticas. Ejs Wiki. 2011. [http://www.um.es/fem/EjsWiki/Es/HomePage, 10/06/2012].