Hacia un Simulador Programable de Sistemas Dinámicos

Ing. Diego Martínez Guillén
Instituto Politécnico Nacional
ESIME Culhuacán
Ciudad de México
diegomtzg@hotmail.com

Dr. Domingo Cortés Rodríguez
Instituto Politécnico Nacional
ESIME Culhuacán
Ciudad de México
domingo.cortes@gmail.com

Mtro. Francisco Hernández Salas
Instituto Politécnico Nacional
ESIME Culhuacán
Ciudad de México
fhernandezs094@gmail.com

***Resumen*—Se presenta el desarrollo de un simulador en lisp capaz de especificar sistemas dinámicos con la capacidad de construir nuevas estructuras a partir de estruturas previamente definidas. Para ello se crean las estructuras base que el ingeniero en control pueda manipular de manera sencilla e intuitiva, sin la necesidad de aprender un lenguaje de programación. El simulador permite especificar un gran número de simulaciones mediante la variación iterativa de parámetros y condiciones iniciales. Además puede filtrar aquellas simulaciones que cumplan con un criterio preestablecido. De esta manera se puede simular un sistema bajo un gran número de condiciones sin necesidad de intervención del usuario.**

***Index Terms*—programmable simulation, domain specific language, dynamical systems, differential equations**

I. Introducción

Actualmente existen programas que simulan ecuaciones diferenciales usando diversos métodos de integración y ante diversas condiciones iniciales. Dentro de estos programas se encuentran Matlab/Simulink, Xppaut, Octave, entre otros. Aunque estos programas simulan ecuaciones diferenciales, requieren la atención constante del usuario para definir el sistema dinámico, cambiar parámetros, condiciones iniciales, probar diferentes entradas, cambiar las características del sistema [Gaviño, 2010].

Además, programas como Matlab/Simulink mantienen sus procesos ocultos por lo que modificarlos o comprender el modo en el que están siendo ejecutados puede conllevar una gran dificultad. Muchas de estas tareas podrían ser automatizadas y especificadas al inicio y despues hacer que la computadora realice un gran número de simulaciones una a una [Tewari, 2002]. En general se requiere bastante tiempo para la construcción de un sistema grande e implica gran probabilidad de cometer errores y dificultad al identificarlos.

En el caso de xppaut, aunque permite de forma semiautomática hacer el cambio de parámetros; condiciones iniciales entre otros cambios posibles; no puede programarse con ciclos o condicionales. En el caso de simulink, la necesidad de usar bloques y crear conexiones entre ellos dificulta el manejo de sistemas donde se requieren varias ecuaciones [Ermentrout, 2002].

Por ello es conveniente la construcción de un simulador programable [Vöth, 2006]. Para ello, se requiere de la creación de un lenguaje de dominio específico que facilite especificación de sistemas dinámicos, el cambio de parámetros, de condiciones iniciales y de entradas. Sería conveniente también que se pudieran crear diversas estructuras de sistemas y nombrarlas para crear bloques que puedan reusarse. Todo esto puede ser programable para permitir la simulación de sistemas cambiando sus atributos de forma recursiva y a su vez poder aplicarle otros procedimientos a los resultados. Esto haría que se abriera un gran abanico de posibilidades en las características que se podrán simular [Kirk, 2004]. Al ser programable y libre se permite la creación de nuevas funciones con el uso de las estructuras o procedimientos ya generados y así ser un lenguaje escalable.

II. Características deseables y Estructura del simulador programable

- Instrucciones para la ejecución de tareas complejas como la composición, retroalimentación y filtrado de sistemas.
- Capacidad de nombrar sistemas como un objeto nuevo para reusarlo en conjunto con otras características del programa.
- Expresiones sencillas para la especificación de sistemas.
- Capacidad de usar los resultados fuera del programa.
- Uso de funciones que puedan usarse como argumento en otra función.
- Modularidad en los procedimientos.

II-A. Lenguaje

El programa está siendo desarrollado en scheme [Friedman et al., 1995], para aprovechar las características de la programación funcional. En particular usar funciones que devuelven funciones y estas, a su vez, poder ser usadas como argumento en otras funciones.

El programa está dividido en las siguientes capas.

II-B. Capa 1

II-B1. Agenda: En la capa más baja están los procedimientos para el manejo de colas y el uso de una agenda.

Esta es importante para ordenar el desencadenamiento de acciones ya que ésta rige el tiempo en el que deben ejecutarse cada una de las tareas a través de enviar mensajes a los objetos para realizar la tarea que les corresponde. A su vez cada objeto, puede enviarle a la agenda una nueva acción para ser catalogada y ejecutada en un tiempo determinado. Aunque para un sistema sencillo pudiera ser evidente que una acción es seguida de otra en forma lineal, pudieran combinarse sistemas de diversas formas en las que no fuera tan sencillo definir en qué tiempo se desencadena cada acción y con qué valores evaluar siguientes estados.

La agenda se encarga de poner un orden y evitar posibles errores a la hora de ejecutar las tareas. Esta idea se toma de ejemplos usados con circuitos digitales de [Abelson and Sussman, 1996].

II-C. Capa 2: Funciones para especificar sistemas.

II-C1. Reloj: En esta segunda capa, se encuentran las funciones encargadas de la simulación de las ecuaciones diferenciales. En primer lugar se tiene la estructura del reloj el cual funciona como la agenda. Almacena los datos de tiempo inicial, tiempo final, paso de integración y los procedimientos asociados a los sistemas que se declaren. En cada paso de integración el reloj envía mensajes a los procedimientos que tenga almacenados para que estos se ejecuten sucesivamente hasta recorrer todos los procedimientos en la lista asociada al tiempo en proceso. Así hasta llegar al tiempo final de simulación.

II-C2. Cables: Los cables son otro objetos definidos en esta misma capa. Se encargan de conectar los sistemas, almacenan un valor de señal actual, un valor anterior y una lista de procedimientos asociados a los sistemas entrelazados para que éstos tengan disponibles los datos de manera inmediata sin necesidad de la intervención del reloj.

II-C3. Sistema: En esta misma capa se define la estructura para la creación de un sistema dinámico. Un sistema dinámico está definido por las ecuaciones del sistema dinámico, las condiciones iniciales, los parámetros del sistema, entradas y salidas [Kuo, 1996]. Esta función se encarga de llamar al método de integración y calcular los siguientes estados del sistema, se envían estos a los cables de interconexión a su salida correspondiente y estos, al detectar esta entrada de datos, envían la señal al siguiente sistema para ser evaluado.

De la misma manera, en un tiempo determinado, el reloj al percatarse que se ejecutaron todos los procedimientos, incrementa su tiempo actual y vuelve a barrer con el cálculo de los procedimientos almacenados en su agenda.

II-C4. Scope: Esta función es importante debido a que asigna un marcador a un cable seleccionado y se encarga de almacenar los datos deseados en una estructura de lista como pares de vectores asociando el tiempo y estado correspondientes. Estas listas pueden ser exportadas para ser manejadas por un programa externo o bien dentro del mismo simulador ser transferidas a un graficador o simplemente ser desplegadas como listas de datos.

II-C5. Funciones auxiliares: En esta capa se crean funciones que ayudan a la creación de sistemas complejos a partir de sistemas sencillos. Estas funciones son: la cascada de sistemas, la retroalimentación, sumadores de señales, entre otras estructuras que puedan generar los usuarios. Por la estructura del programa estas son relativamente sencillas de hacer ya que unicamente involucran la conexión de cables y sistemas, al poder nombrar estas conexiones se facilita la construcción de sistemas más grandes y complejos con el uso de estos sistemas menores.

II-D. Capa 3: Graficador

Aunque los datos puedan ser manejados fuera del simulador es deseable contar con un graficador interno el cual apoye la parte visual para la interpretación de los datos recopilados y debe de ser capaz de poder realizar la grafication de los puntos obtenidos de diversas simulaciones en una sola pantalla.

II-E. Capa 4: Interfaz

En esta siguiente capa de abstracción se encuentran las funciones asociadas a la interacción con el usuario, debido a que la programación está basada en scheme y para que el usuario no deba de aprender el lenguaje para el uso del simulador, se crean las funciones necesarias para tener esta comunicación entre usuario y programa. Scheme usa una estructura prefija, esto quiere decir que la definición de la operación o procedimiento se antepone a los argumentos (p.ej: 6+8 se escribe (+ 6 8)), esto ayuda ampliamente a la ejecución del programa pero puede ser motivo de confusión para usuarios no familiarizados con esta sintaxis [Abelson and Sussman, 1996]. Por ello, en esta capa se crean estructuras de traducción para que el usuario pueda introducir las expresiones algebráicas de manera convencional y al ser leídas por el traductor, este se encargue de interpretarlas en código con sintaxis prefija. Se busca que los comandos de el usuario final sean intuitivos en su uso y con una estructura sencilla.

III. Metodología de simulación:

A continuación se describen los pasos necesarios para realizar una simulación en el programa descrito en este artículo:

III-A. Especificar el sistema y las simulaciones en un archivo de texto

Realizar un archivo de texto con la descripción del sistema a simular, con los elementos separados por saltos de línea como se muestra en el Código 1. El nombre del archivo de texto será el nombre del sistema. El orden de los elementos del archivo no es estricto y los espacios entre expresiones no afecta su comportamiento. Los elementos mínimos requeridos para realizar una simulación son tiempo final (“tf=”), paso de integración (“dt=”), estado y su correspondiente ecuación (“’=”), condición

Listing 1: Descripción de un Sistema Dinámico

```
tf= <valor>
dt= <valor>

#<entrada>

<param a> = <valor>
<param b> = <valor>
<param c> = <valor>

<estado 1>' = <ecuación>
<estado 2>' = <ecuación>
<estado n>' = <ecuación>

<estado 1>(0) = <valor>
<estado 2>(0) = <valor>
<estado n>(0) = <valor>

$<salida> = <ecuación>
```

inicial ("(0)=") y salida ("$"). Los parámetros pueden ser omitidos, sin embargo, se recomienda su uso para catalogar los datos de una manera más evidente y abre la posibilidad para que puedan variar de manera automatizada. La entrada, de manera similar, no es estrictamente requerida, pero es recomendable su uso para tener una señal externa e interconectar sistemas.

III-B. Recuperar información

Ejecutar el comando (get-info "«ruta»«nombre del archivo»"). Servirá para obtener los datos del archivo de texto, clasificarlos y almacenarlos en una estructura de datos usada por el programa. Ésta, al contener los datos de cada sistema, es accedida para extraer y traducir a lisp las partes antes descritas.

III-C. Simulación

Ejecutar el comando (simular "¡nombre de archivo»"). Envía las descripciones del sistema almacenadas en la tabla de sistemas al traductor que lo convierte a lenguaje lisp y calcula iterativamente cada uno de los nuevos estados a partir de las condiciones iniciales hasta llegar el tiempo final. La información es graficada o si se utiliza: (simular «nombre de archivo» #t), la información se exportará a un archivo de texto como una lista de pares de vectores con el tiempo y su salida correspondiente de simulación.

IV. Ejemplos

IV-A. Ejemplo 1

En este ejemplo, se simula un sistema con los datos mínimos requeridos para poder realizar una simulación. Los pasos requeridos son los siguientes:

- Se escribe el Código 2 en un archivo de texto con el nombre "primerorden.txt". Este muestra la descripción de un sistema de un primer orden estable.
- Se ejecuta el comando (get-info "«ruta al archivo»primerorden.txt"). En este paso se extraen y almacenan los datos.

Listing 2: Sistema Dinámico de primer orden

```
tf=10
dt=0.001

x'=-2*x+5

x(0)=1

$y=x
```

Listing 3: Parte de resultados de simulación

```
(#(0.0 1.0)
#(0.001 1.003)
#(0.002 1.0059939999999998)
#(0.003 1.008982012)
#(0.004 1.011964047976)
#(0.005 1.0149401198800478)
#(0.006 1.0179102396402877)
#(0.007 1.020874419161007)
#(0.008 1.023832670322685)
...
#(9.9959999999999 2.4999999969452085)
#(9.996999999999899 2.4999999969513182)
#(9.997999999999898 2.4999999969574156)
#(9.998999999999898 2.499999996963501)
#(9.999999999999897 2.499999996969574))
```

- Se ejecuta el comando (simular "primerorden") para graficar el resultado como se observa en la Figura 1 o (simular "primerorden" #t) para generar un archivo externo con el nombre "primerorden.data" con los datos en forma de vectores en una lista como se muestra en el Listado 3.

IV-B. Ejemplo 2

En el Código 4 se muestra la descripción de un péndulo simple como el que se observa en la Figura 2. Note que

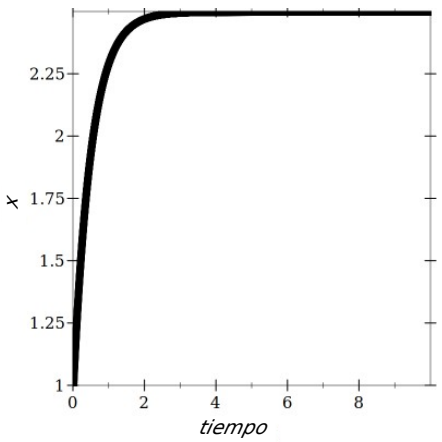


Figura 1: Péndulo Simple.

Listing 4: Descripción de un péndulo simple

```
dt=0.001
tf=4

m=0.6
b=0.5
g=9.81
l=0.3

th'=vel
vel'=(-g/l)*sin[th]-((b*vel)/(m*l))

th(0)=2.5
vel(0)=0

$salida=th
```

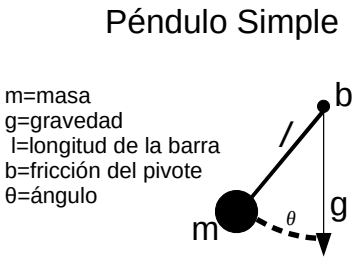


Figura 2: Péndulo Simple.

los argumentos de las funciones trigonométricas deben escribirse entre corchetes. El péndulo simple es un sistema de dos estados que puede ser descrito en pocas líneas. Este sistema descrito en simulink, se muestra en la imagen de la Figura 4.

En la Figura 3 se puede ver que ante la posición inicial de 2.5 radianes, en efecto, el péndulo cae a 0 radianes al igual que el modelo de simulink graficado en la Figura 5.

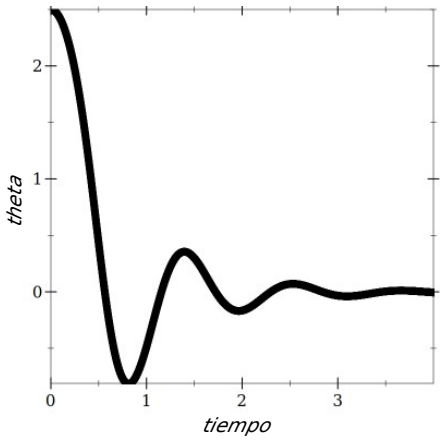


Figura 3: Péndulo Simple.

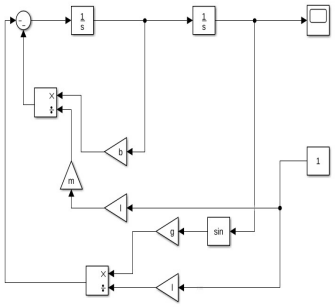


Figura 4: Péndulo Simple descrito en simulink.

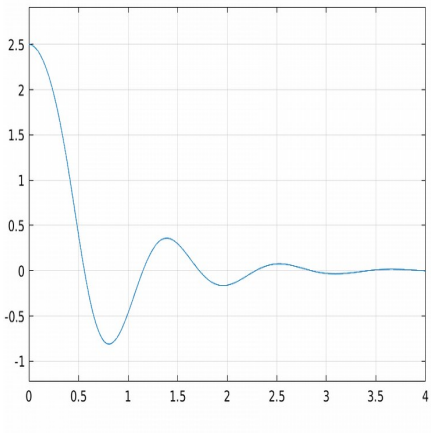


Figura 5: Gráfica de péndulo simple de simulink.

IV-C. Ejemplo 3

En el sistema descrito en Código 5 correspondiente a un péndulo invertido como el expuesto en la Figura 6, puede notarse que es posible desarrollar sistemas de varios estados [Dorf and Bishop, 2005]. Pueden incluir entrada ("#") y diversos parámetros. La simulación del comportamiento dinámico del sistema es, para este caso, la posición del carro y su resultado se expone en la Figura 7.

IV-D. Características notables del simulador programable

IV-D1. Conexión en cascada de sistemas programable: Teniendo dos sistemas como los mostrados en la Figu-

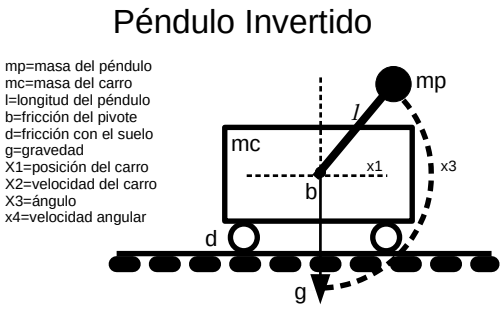


Figura 6: Péndulo Invertido.

Listing 5: Descripción de un péndulo invertido

```
tf=10
dt=0.001

#F

mp=0.3
mc=1.8
l=0.2
b=0.4
d=0.04
g=9.81

x1'=x2
x2'=(F-b*x2+l*mp*sin[x3]*x4*x4+
mp*g*sin[x3]*cos[x3]+((d*x4*mp*
cos[x3])/l))/(Mc+mp-mp*cos[x3]*cos[x3])
x3'=x4
x4'=(-g*sin[x3]-((d*x4)/l)-cos[x3]*
((F-b*x2+l*mp*sin[x3]*x4*x4+mp*g*
sin[x3]*cos[x3]+((d*x4*mp*cos[x3])/l))/
(Mc+mp-mp*cos[x3]*cos[x3])))/l

x1(0)=0
x2(0)=0
x3(0)=3.142
x4(0)=0

$y=x1
```

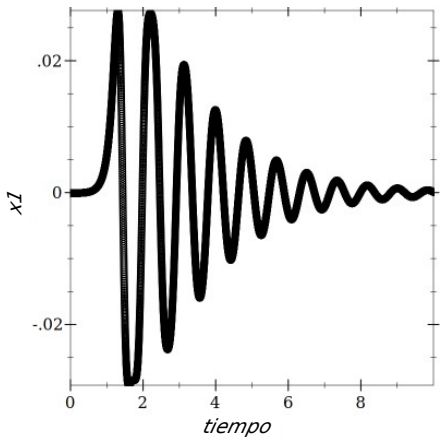
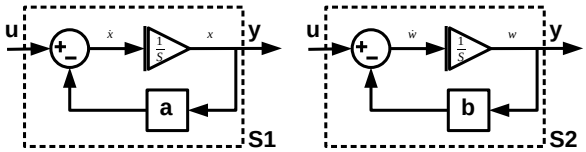


Figura 7: Péndulo Invertido.

ra 8a. Pueden colocarse en cascada mediante el comando: (cascade «sistema 1» «sistema 2»), quedando estos como se representan en la Figura 9. Cada uno puede abarcar cualquier número de estados que se deseen.

Si se aplica el comando al sistema descrito en Código 2 y Código 5 queda: cas=(cascade "primerorden" "penduloinvertido") Al ejecutar "(simular "cas")" se obtiene la gráfica de la Figura 10. Para estos sistemas la acción cascada implica que el péndulo caiga en menor tiempo.

IV-D2. Conexión en retroalimentación programable: Como aparece en la Figura 11 dos sistemas pueden combinarse en conjunto de un sumador para realizar una retroalimentación, aplicando dicha operación a "péndulo_invertido" con el mismo se tiene rpi=(retro "péndulo_invertido" "péndulo_invertido"). Al simularlo (simular "rpi") se genera la gráfica de la Figura 12.



(a) Sistema 1. (b) Sistema 2.

Figura 8: Sistemas.

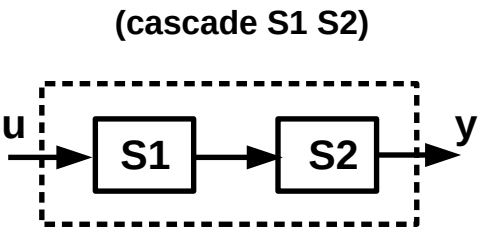


Figura 9: Cascada de Sistemas.

IV-D3. Conexión en cascada iterativa: La composición de un mismo sistema iterativamente está representado en la Figura 13. Para el péndulo invertido podemos aplicar una simulación a dicho comando quedando como sigue: (simular (cascaden "pendulo_invertido" 5)). De tal manera que se genera una cascada de cinco sistemas, uno tras otro, dando por resultado la gráfica mostrada en la Figura 14.

Todas estas estructuras se pueden combinar para crear nuevas y de esta manera describir modelos más complejos,

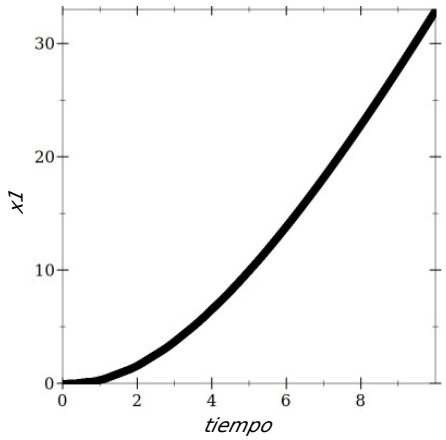


Figura 10: Simulación de cascada de "primerorden" con "penduloinvertido".

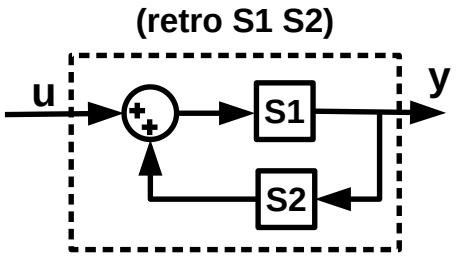


Figura 11: Retroalimentación de Sistemas.

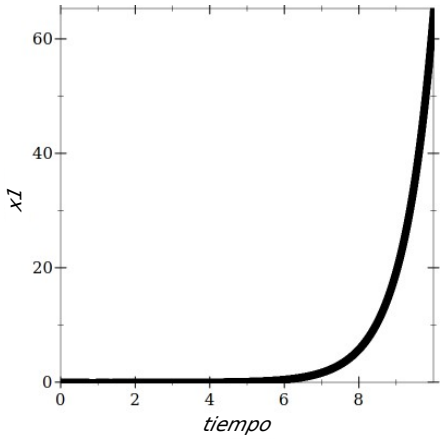


Figura 12: Simulación de retroalimentación de "penduloinvertido" con "penduloinvertido".

con lo que se pueden tener un gran número de posibilidades.

V. Conclusiones y Trabajo a futuro

En este artículo se ha descrito la primera versión de un simulador programable. Dicho simulador no solo permite expresar un sistema dinámico, la entrada y las condiciones iniciales, sino que permite interconectar sistemas de manera sencilla, para crear sistemas complejos a partir de sistemas simples. Ademas permite especificar una simulación con un gran número de entradas, variación

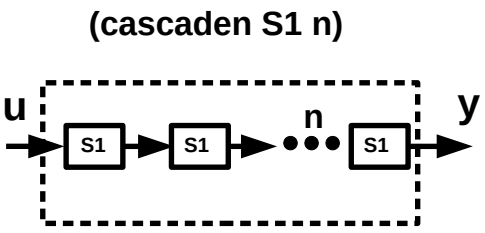


Figura 13: Cascada de n Sistemas.

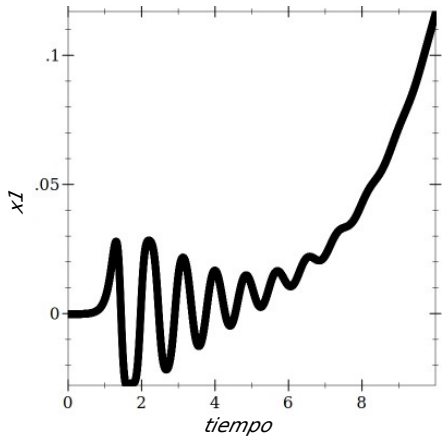


Figura 14: Simulación de Péndulo Invertido en cascada.

de parámetros y condiciones iniciales. De esta manera se podrían especificar de una vez todas las condiciones bajo las cuales se quiere simular un sistema y después la computadora haría una a una cada simulación de manera automática. No solo eso, se pretende que el simulador sea capaz de guardar sólo aquellas simulaciones que cumplan con un criterio preestablecido.

Las partes que hace falta desarrollar en el simulador son:

- Comandos que permitan la variación de parámetros de un sistema.
- Esta automatización será capaz de realizar un barrido de estos dentro de un rango estipulado. El resultado que arroja es una lista de simulaciones.
- Además de los parámetros se pretende que este comando sea capaz de variar condiciones iniciales, entradas y salidas. De modo que se genera una gama amplia para la elección de un sistema y/o controlador adecuado para la tarea requerida.
- Comandos de filtrado que puedan ser aplicados a las listas de simulaciones. De esta forma se obtendrá "el sistema más rápido", "el sistema con menor sobreimpulso", "el sistema que siga de manera más fiel una función de referencia", etc.

Referencias

[Abelson and Sussman, 1996] Abelson, H. and Sussman, G. J. (1996). *Structure and Interpretation of Computer Programs.*

[Dorf and Bishop, 2005] Dorf, R. C. and Bishop, R. H. (2005). *Sistemas de control moderno.*

[Ermentrout, 2002] Ermentrout, B. (2002). *Simulating, Analyzing, and Animating Dynamical Systems.*

[Friedman et al., 1995] Friedman, D. P., Felleisen, M., and Sussman, G. J. (1995). *The Little Schemer.* The MIT Press.

[Gaviño, 2010] Gaviño, R. H. (2010). *Introducción a los sistemas de control: Conceptos, aplicaciones y simulación con MATLAB.*

[Kirk, 2004] Kirk, D. E. (2004). *Optimal control theory: An introduction.*

[Kuo, 1996] Kuo, B. C. (1996). *Sistemas de control automático.*

[Tewari, 2002] Tewari, A. (2002). *Modern Control Design With MATLAB and SIMULINK.*

[Vöth, 2006] Vöth, S. (2006). *Dynamik schwingungsfähiger Systeme.*