Validación de la correcta definición del modelo de un Sistema Híbrido representado en MLD

Monografía de Grado
Maestría en Ciencias Computacionales

Proponentes:
Carlos Alberto Peña Borrero
cpena@sanchezpolo.com
Jorge Luis Ulloa Candanoza.
J_ulloa_c@hotmail.com
Maria del Socorro David
mariadavidch@gmail.com
Erwin Enrique Pertuz Rudas
erwinper@gmail.com
Carlos Enrique Ramirez Hamburger
carlos.ramirez@estrateg.com

Director:
PhD José Luis Villa
Profesor Universidad Tecnológica de Bolívar

Instituto Tecnológico y de Estudios Superiores de Monterrey
Universidad Autónoma de Bucaramanga
Universidad Tecnológica de Bolívar
Cartagena
2007

Nota de aceptación

_____

_____

_____

_____

_____

_____

Presidente del Jurado

_____

Firma del Jurado

_____

Firma del Jurado

DEDICATORIAS

A Dios quien lo hizo posible, a Sánchez Polo quien lo facilitó, a mis papás por todo su apoyo y motivación, a mis compañeros por su voluntad, y a Lili y Davi por todo el tiempo de esposo y padre al que renunciaron para que esto fuera realidad .

Carlos Peña

A Dios, por ser mi guía en todo momento, a mi madre, Edith, por su tenacidad para sacarme adelante, a mi esposa, Delly, por darme su fortaleza y apoyo en todo momento, a mis hijos Daniel y Edwin David por la alegría que me irradian y a mis compañeros de la tesis por su colaboración.

Edwin Pertuz

Agradezco a mis padres a mi esposa e hijos por brindarme su apoyo durante estos años.

Carlos Ramírez

A Dios por la inteligencia y sabiduría que me dio al nacer y por enseñarme el camino correcto de la vida. A mi adorada esposa Mónica Patricia, quien me ha acompañado en todo momento, por su amor, su paciencia y su compañía incondicional. A mi querida hija Paula Alejandra, por ser mi fuente de alegría y fuerza de superación permanente. A mis padres Eduardo y Lilia quienes con su ejemplo, paciencia e infinito amor, me han hecho la persona humana, integra y profesional que soy hoy en día. A mis hermanos Luz, Freddy y Eduardo, por todas las vivencias y cariño que nos unen.

Jorge Ulloa

A la vida: Dios, Familia, Trabajo, Profesores, Amigos… todos se han conjugado en el logro de este objetivo.

Maria del Socorro David

AGRADECIMIENTOS

A José Luís Villa, nuestro Director de Tesis, por su alma de Maestro, con su inestimable ayuda, dedicación y constante apoyo, ha hecho posible esta Tesis.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ALGORITMOS

LISTA DE ANEXOS

1. INTRODUCCION

Un sistema real es aproximado a través de un modelo matemático con el objetivo de capturar un comportamiento específico del proceso que sea de interés para determinado tipo de análisis o de diseño.

Usualmente, cuando el interés es representar cierto tipo de variables que tienen un comportamiento continuo, tales como temperaturas, presiones, o niveles, el modelo es descrito utilizando el formalismo matemático de ecuaciones diferenciales, ver por ejemplo [1]. Cuando el interés es describir una secuencia de eventos del tipo: se prende una máquina, luego esperar que se caliente, después activar un proceso, este tipo de modelos se denominan Sistemas a Eventos Discretos, algunos de estos formalismos incluyen Autómatas y Redes de Petri, ver por ejemplo [2]. Algunos procesos complejos requieren modelar la interacción de subsistemas continuos con subsistemas a eventos discretos. Por ejemplo, si quisiera estudiar el comportamiento de la temperatura de un horno con una acción de control tipo on/off, necesitaría un modelo para la temperatura representado en ecuaciones diferenciales, y un modelo del control on/off representado a través de un sistema a eventos discretos, como un autómata; ambos modelos interactuarían entre sí para poder representar el comportamiento de dicho horno. Este tipo de sistemas también son llamados Sistemas Dinámicos Híbridos en la literatura especializada del tema ver [3].

En particular, algunos sistemas exhiben discontinuidades en su comportamiento, por lo que deben ser representados en estructuras matemáticas que permitan representar este tipo de comportamientos. Estos comportamientos discontinuos son comunes en la realidad, y su correcto modelamiento es de gran utilidad para abordar problemas de simulación y control.

El grupo de investigación GAICO de la UTB, cuya principal línea de investigación ha sido en modelamiento y control de sistemas industriales, ha abordado el tema a través de lo que se conoce en la literatura técnica como sistemas dinámicos híbridos, y en particular a través de un modelo denominado Sistemas Lógicos y Dinámicos mezclados (*mixed logical and dynamical – MLD systems)*; los sistemas lógicos y dinámicos mezclados pueden ser representados como un sistema aparentemente lineal con desigualdades con forma lineal, pero una parte de sus variables solo pueden tener valores binarios [8].

Ya el grupo ha utilizado esta representación para resolver problemas de simulación y problemas de control predictivo, sin embargo, los algoritmos utilizados requieren enumerar, en cierto modo, las posibles soluciones factibles del problema, y por ello el algoritmo tiene complejidad computacional *No Determinística Polinomial NP*, adicionalmente es difícil encontrar errores de modelamiento en la estructura matemática, debido al potencialmente alto número de restricciones que se pueden requerir para representar el sistema [7].

Debido al potencialmente alto número de restricciones que pueden estar incluidas en el modelo MLD, la representación puede ser de difícil evaluación por parte del usuario final. En nuestro conocimiento no se ha implementado ningún algoritmo que permita verificar la validez del modelo MLD.

El principal objetivo de este trabajo entonces se establece como implementar un programa para validar la correcta definición de un sistema dinámico híbrido, representado con un modelo MLD (Mixed Logical and Dynamic System), por lo que el principal aporte de este trabajo consiste en implementar un algoritmo de validación del modelo MLD, que le permite al diseñador visualizar gráficamente el campo vectorial del sistema.

El trabajo inicia con una definición de las características de un sistema bien definido, y qué se entiende como un sistema mal definido, con lo cual se establecen las especificaciones del

algoritmo, para dar solución al objetivo especifico de determinar las características de funcionamiento del algoritmo de validación.

El algoritmo implementado se basa en la teoría propuesta por Bemporad y Morari en [8], como se explica en el capitulo 4. de este trabajo, esto constituye la solucion del objetivo especifico de implementar y evaluar el algoritmo MILP (Programación Lineal Entera Mixta) con un problema de ejemplo MLD.

Debido al alto costo computacional que el algoritmo propuesto presenta, se desarrolla y propone una versión utilizando computación distribuida, con lo cual la solución propuesta resulta de mayor utilidad, dando de esta forma solución al objetivo especifico de proponer un algoritmo mejorado que reduzca el tiempo de computación en el problema MLD.

Ambos algoritmos se aplican a tres casos de estudio donde se evalúan los ajustes de parámetros del algoritmo y su respectivo tiempo de ejecución, dando solución al objetivo específico de evaluar el comportamiento del algoritmo propuesto en un problema de estudio.

El presente documento esta organizado de la siguiente forma: en el capítulo 2 se revisan las características de los sistemas MLD y las herramientas disponibles para su análisis. En el capítulo 3 se describen las características de los sistemas MLD correctamente definidos("Well Posed") y se elaboran tres casos diferentes de sistemas MLD, sobre los cuales se aplica y evalúa el algoritmo de validación. En el capítulo 4 se describe el algoritmo de validación que se implementa en este proyecto, el cual es optimizado mediante la utilización de un sistema de cómputo distribuido, que se expone en el capítulo 5. Finalmente las conclusiones y proyección de trabajos futuros, son expuestos en el capitulo 6.

2. SISTEMAS LOGICOS Y DINAMICOS MIXTOS - MLD (MIXED LOGICAL AND DYNAMICAL)

En este capitulo se revisan las características principales de los sistemas MLD sobre los cuales se fundamenta esta investigación. También se analizan las herramientas computacionales que están disponibles para sistemas MLD: HYSDEL y MATLAB.

2.1 GENERALIDADES DE SISTEMAS DINAMICOS.

Este aparte trata de algunas formulaciones de la teoría existente sobre sistemas dinámicos, y se basa principalmente en las siguientes referencias [9], [11], [14], [20], [21], [24], [25] y [26].

Un sistema es un conjunto de objetos donde algunos elementos interactúan entre ellos, como resultado de esta interacción se produce un nuevo comportamiento. Un sistema en general produce señales de salidas $y(t)$, las cuales se pueden observar, medir y tiene entradas o señales $u(t)$, las cuales afectan al sistema. El sistema también puede recibir ruidos $e(t)$, usualmente este se representa como un bloque, tal como se observa en la Figura 1. En general, los sistemas tienen un comportamiento "Causal" y "Dinámico", lo que significa que sus salidas son determinadas exclusivamente por lo que ocurrió en el pasado, no por lo que ocurre en el futuro.

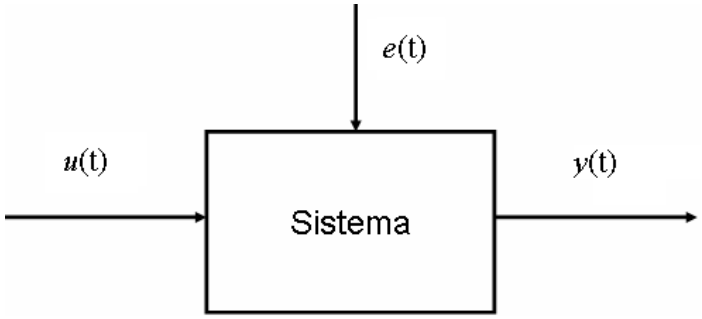

Figura 1 Sistema

Al estudiar o trabajar con sistemas, normalmente se construyen modelos que son nuevos sistemas con un comportamiento equivalente al sistema de estudio. Los modelos matemáticos aproximan el comportamiento de los fenómenos físicos. Considerando los procesos en diferentes niveles de detalles, diferentes modelos del mismo proceso están disponibles en las ciencias aplicadas.

Los modelos no pueden ser tan simples que no tengan en cuenta suficientes detalles del proceso, pero tampoco tan complicados, que no sea posible describir y resolver eficientemente el análisis y síntesis de estos.

Una forma común para describir los sistemas es utilizar modelos de espacio de estado. En esta clase de modelos, la historia del sistema se refleja en el vector de estado x(t). La salida del sistema y(t) depende de x(t), la señal de entrada u(t) y el ruido e(t). Un modelo de espacio de estado en tiempo continuo se puede formular así:

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t), e(t)) \\ y(t) &= g(x(t), u(t), e(t)) \end{aligned} \qquad (2.1)$$

Donde $x(t) \in R^n$, y $\dot{x}(t)$ representa la derivada en el tiempo de x(t).

En el modelo 2.1, las variables en el tiempo son continuas y están definidas en todos los puntos del tiempo.

A menudo las señales son muestreadas con un período regular, por lo cual tiene sentido considerar un modelo en tiempo discreto. Un modelo de espacio de estado en tiempo discreto se puede formular como:

$$\begin{aligned} x(t+1) &= f(x(t), u(t), e(t)) \\ y(t) &= g(x(t), u(t), e(t)) \end{aligned} \qquad (2.2)$$

Con la siguiente forma alternativa:

$$y(t) = f(y(t-1), y(t-2), ..., u(t-1), u(t-2), ..., e(t), e(t-1), ...) \quad (2.3)$$

$y(t)$ Es una función de todas las señales de entrada, salida y ruido anteriores.

Por facilidad utilizaremos la siguiente notación:

$$\begin{aligned} y_{t_1}^{t_2} &\overset{\Delta}{=} \{y(t_1), y(t_1+1), ..., y(t_2)\} \\ Z_{t_1}^{t_2} &\overset{\Delta}{=} \{u_{t_1}^{t_2}, y_{t_1}^{t_2}\} \end{aligned} \qquad (2.4)$$

De forma que 2.3 se puede escribir como:

$$y(t) = f(Z_{-\infty}^{t-1}, e_{-\infty}^{t}) \qquad (2.5)$$

Un supuesto común acerca del ruido, está dado por la clase de sistemas descritos por:

$$y(t) = f(Z_{-\infty}^{t-1}) + e(t) \qquad (2.6)$$

En los sistemas dinámicos podemos encontrar, los sistemas continuos que son aquellos en los cuales las variables de interés pueden tomar valores del conjunto infinito del espacio real y los sistemas a eventos discretos en los cuales las variables de interés pueden tomar valores de un conjunto finito.

En los últimos años, varios científicos de la computación y sistemas de control han investigado modelos que describen la interacción entre sistemas dinámicos continuos, descritos por ecuaciones diferenciales y componentes lógicos descritos por maquinas de

estados finitos, reglas de la forma IF-THEN-ELSE, lógica proposicional y temporal, entre otras representaciones.

Los Sistemas Dinámicos Híbridos, son sistemas que son modelados utilizando sistemas continuos interactuando con sistemas a eventos discretos.

Estos modelos heterogéneos, cambian entre varios modos de funcionamiento, donde cada modo se asocia a una ley dinámica distinta y las transiciones entre los modos son dadas por eventos, como por ejemplo estados que pasan los límites especificados inicialmente.

La importancia práctica de los modelos híbridos es doble. Lo beneficioso de los reguladores basados exclusivamente en lógicas combinatoriales que controlan variables continuas, lo cual aumenta la demanda por las herramientas de análisis y de diseño adecuadas para tratar estos problemas, por ejemplo en la industria automotriz encontramos una aplicación importante en el diseño de cajas de cambio automáticas que maximizan la eficiencia del motor. Por otra parte, muchos fenómenos físicos admiten una descripción híbrida natural, como los circuitos que integran los relays o los diodos, las redes biomoleculares, y las redes de TCP/IP, por ejemplo: [4], [5], [10], [13], [15], [16], [17], [18] y [19].

Los modelos híbridos son necesarios para tratar muchos problemas, como la definición y el cómputo de trayectorias, el análisis de la estabilidad y seguridad, el control y la estimación de estados. La definición de la trayectoria se asocia generalmente a un simulador, una herramienta capaz de calcular la evolución de las variables del sistema. Esto puede parecer simple al principio, no obstante muchos formalismos híbridos introducen comportamientos adicionales como el fenómeno Zeno [22], que complica la definición de trayectorias.

Es importante notar que la simulación permite probar el modelo, mas no permite que las características estructurales del modelo sean determinadas. En realidad, en cualquier

análisis basado en simulación es probable que no se exhiban fenómenos sutiles que el modelo puede generar, especialmente en el caso de modelos híbridos.

Las herramientas de modelamiento de sistemas usualmente son utilizadas para abordar problemas de Especificación, Verificación, Análisis y Control [23].

Varias clases de sistemas híbridos se han propuesto en la literatura, cada clase se adapta generalmente para solucionar un problema en particular. Entre los formalismos mas utilizados se pueden citar los Autómatas Híbridos Discretos – DHA, los Sistemas Afines a Trazos – PWA, y los sistemas Lógicos y Dinámicos Mezclados – MLD.

En la literatura del tema, se pueden encontrar amplios usos de los modelos DHA para problemas de verificación y PWA para problemas de Análisis y Control. Adicionalmente, las propiedades estructurales de estos modelos han sido estudiadas con cierto rigor en **[24]** y **[25**].

Una de las principales propiedades dinámicas a verificar en un sistema no lineal con discontinuidades es la buena definición del modelo matemático. En pocas palabras, un sistema está bien definido, si dadas las condiciones iniciales de los estados y se especifica un valor de las entradas, existe una y solo una solución al modelo [26]. Estas condiciones se denominan Continuidad, Existencia y Unicidad de la solución del modelo.

El modelo utilizado en este trabajo es el modelo MLD, el cual se describe en detalle en la siguiente sección.

2.2 EL MODELO MLD .

En este aparte se describe el modelo MLD. Estos sistemas fueron propuestos por Alberto Bemporad y Manfred Morari en [8], como un modelo conveniente para varias clases de sistemas híbridos. La idea central en este acercamiento es la utilización de las herramientas de programación matemática [12].

Los modelos obtenidos en la programación matemática pueden ser clasificados como Modelos de Programación Lineal, Modelos de Programación No Lineal y Modelos de Programación Entera.

Para entender la técnica de modelamiento utilizada se analiza el siguiente ejemplo:

- Considere un sistema de la siguiente forma:

$$x(k+1) = \begin{cases} 0.8x(k) + u(k) & Si \quad x(k) \geq 0 \\ -0.8x(k) + u(k) & Si \quad x(k) < 0 \end{cases}$$

Donde $x(k) \in [-10, \quad 10]$ y $u(k) \in [-1, \quad 1]$

Los pasos para el modelamiento MLD son los siguientes:

Primer Paso: Asociar una variable binaria $\delta \in \{0,1\}$ con una proposición S, que pueda ser verdadero o falso. δ es 1 si y solo si S es verdadero.

Segundo Paso: reemplazar los productos de funciones lineales y variables lógicas por una nueva variable auxiliar *z*

Tercer Paso: re-escribir el sistema dinámico, las variables binarias y las variables auxiliares en un sistema lineal invariante en el tiempo (LTI).

Aplicando lo anterior al ejemplo obtenemos:

- Se asocia la variable binaria $\delta(k)$ a la condición $x(k) \geq 0$, tal que $[\delta(k) = 1] \Leftrightarrow [x(k) \geq 0]$, lo cual es equivalente a :

$$-m\delta(k) \leq x(k) - m$$

$$-(M+\varepsilon)\delta \leq -x - \varepsilon$$

donde m es un límite inferior para la función f(x) = x(t), y M es un límite superior para la misma función, y ε es un valor de tolerancia pequeño.

- El sistema se puede reescribir como:

$$x(k+1) = 1.6\,\delta(k)\,x(k) - 0.8\,x(k) + u(k)$$

- Al reescribir el sistema en el paso anterior se genera una operación no lineal en el término δ(k)x(k), por ello se reemplaza esta expresión por una nueva variable z(k) que se define como z(k) = δ(k) x(k), que a su vez es equivalente a las siguientes cuatro expresiones

$$z(k) \leq M\,\delta(k)$$

$$z(k) \geq m\,\delta(k)$$

$$z(k) \leq x(k) - m\,(1 - \delta(k))$$

$$z(k) \geq x(k) - M(1 - \delta(k))$$

  donde m es un límite inferior para la función f(x) = x(t), y M es un límite superior para la misma función.

- El sistema original es equivalente a la siguiente representación

$$x(k+1) = 1.6\,z(k) - 0.8\,x(k) + u(k)$$

Sujeto a las restricciones

$$-m\delta(k) \leq x(k) - m$$

$$-(M+\varepsilon)\delta \leq -x - \varepsilon$$

$$z(k) \leq M\,\delta(k)$$

$$z(k) \geq m\,\delta(k)$$

$$z(k) \leq x(k) - m\,(1 - \delta(k))$$

$$z(k) \geq x(k) - M(1 - \delta(k))$$

Un sistema MLD descrito en forma general es representando por las siguientes ecuaciones:

$$\begin{aligned} x(t+1) &= Ax(t) + B_1u(t) + B_2\delta(t) + B_3z(t) \\ y(t) &= Cx(t) + D_1u(t) + D_2\delta(t) + D_3z(t) \end{aligned} \qquad (2.7)$$

$$E_2\delta(t) + E_3 z(t) \leq E_1 u(t) + E_4 x(t) + E_5 \qquad (2.8)$$

Donde $x = \left[x_c^T x_1^T\right] \in \Re^{n_c} \times \{0,1\}^{n1}$,son los estados continuos y binarios,

$u = \left[u_C^T u_1^T\right] \in \Re^{m_c} \times \{0,1\}^{m_1}$ son las entradas,

$y = \left[y_C^T y_1^T\right] \in \Re^{P_E} \times \{0,1\}^{P_l}$ son las salidas y

$\delta \in \{0,1\}^{r_l}$, $z \in \Re^{r_c}$, representan las variables auxiliares binarias y continuas.

El lector interesado puede consultar las referencias [8] y [12] para equivalencias adicionales.

2.3 Equivalencia entre representaciones de Sistemas Híbridos

En esta sección se enfatiza en las equivalencias entre las diferentes formulaciones existentes para sistemas dinámicos híbridos.

La equivalencia entre los diferentes modelos de representación de sistemas híbridos se muestra en [9], en el cual se establecieron las equivalencias entre los siguientes Modelos:

- MLD Sistemas dinámicos de lógica mixta
- LC Sistemas Lineales Complementarios
- ELC Sistemas Lineales complementarios extendidos
- PWA Sistemas de piezas afines a trozos
- MMPS Sistemas de suma-Multiplicación escalar-Máximos-Mínimos

En la Figura 2 se muestran las equivalencias entre los distintos modelos, que se demuestran en [9], en cada nodo se identifica un tipo de modelo, en las flechas entre nodos, se indica la existencia de la equivalencia entre los modelos ubicados en los nodos que se conectan, cada flecha se identifica la existencia de la equivalencia con el número de la proposición

que es demostrada en el articulo. Aquellos enlaces que están marcados con un asterisco (*) son equivalencias sujetas a restricciones.

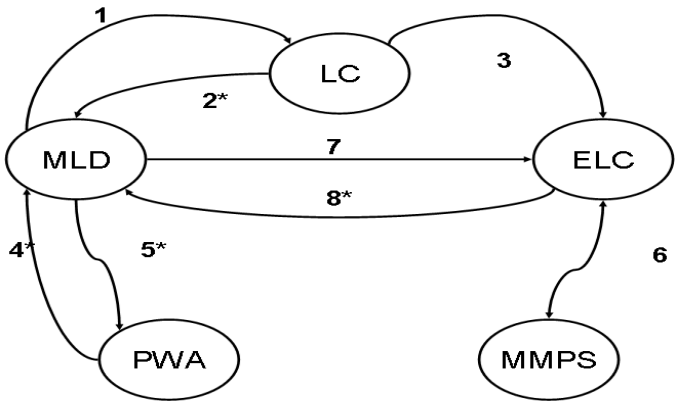

Figura 2 Equivalencia entre modelos híbridos

La importancia de estas equivalencias entre los modelos, permite aprovechar las herramientas desarrolladas en cada uno de ellos, utilizando la migración entre los mismos, lo cual es posible si y solamente si, el modelo esta bien formulado.

Los modelos PWA son apropiados para abordar problemas de análisis mientras que los modelos MLD son apropiados para abordar problemas que pueden ser formulados como problemas de programación matemática (e.g. Control). El principal inconveniente en la representación MLD es la complejidad computacional asociada a los algoritmos que se deben utilizar en programación matemática no lineal.

2.4 HERRAMIENTAS DE ANALISIS PARA SISTEMAS MLD.

Habiendo definido las herramientas matemáticas de base de nuestro trabajo, en este aparte revisamos las diferentes herramientas de software que permiten trabajar actualmente con los MLD.

No cabe duda que por medio de herramientas de software se pueden hacer simulaciones de sistemas híbridos complejos, con lo cual se facilita y simplifica el desarrollo y análisis de un sistema híbrido, además con éstas, se logran ayudas que posibilitan efectuar arduas tareas del diseño y evaluación de sistemas híbridos.

Aunque en este sentido, encontramos que en la actualidad la tecnología de sistemas híbridos se encuentra todavía en etapa de desarrollo en el plano internacional, y existen limitaciones importantes en cuanto a la disponibilidad de metodologías y herramientas ingerentes al diseño óptimo de estos sistemas.

Es así como encontramos que algunas herramientas que permiten la simulación de sistemas híbridos son: HYSDEL y MATLAB.

A continuación se hará una breve descripción de estas.

2.4.1 Hysdel (Hybrid System Description Language)

La construcción de los modelos matemáticos, se hace con el objetivo de representar el comportamiento de los fenómenos físicos, pero en la práctica, no resulta sencillo modelar los sistemas que combinan eventos discretos y continuos.

El problema se basa, en que cuando se modela un sistema híbrido con algún grado de complejidad, se requiere de herramientas que representen de una forma eficaz y eficientemente el modelo, partiendo del marco de MLD (Mixed Logical Dinamical Systems) o del PWA (Piecewise Affine Systems), pues así se podrá alcanzar el objetivo de llegar a la correcta representación de dicho modelo.

En este sentido, HYSDEL (Hybrid Sistem Description Language), proporciona una interfaz intuitiva, y de alto nivel, para modelar la clase de Sistemas Híbridos, los cuales pueden ser

descritos a través de interconexiones con los sistemas lineales dinámicos, autómatas, proposiciones del tipo IF THEN ELSE y reglas proposicionales lógicas[14], Éstos son conocidos como Discrete Hybrid Automata (DHA) [6]. Para este tipo de sistemas, hay técnicas generales para transformar la representación abstracta en un conjunto de ecuaciones lineales, que involucran variables enteras y continuas, y entonces, a través del compilador de HYSDEL se traslada a su forma equivalente en PWA o MLD, para que luego se tome ésta salida como la entrada de un programa en MATLAB.

HYSDEL, puede transformar la descripción del sistema, en una forma MLD, o puede ser un paso intermedio para obtener el PWA (piecewise affine systems), lo que permitiría la correcta validación del modelo. Este modelo MLD también puede ser usado inmediatamente para resolver problemas de control y es posible generar sistemas de complementos lineales (Linear Complemntary LC), o sistemas extendidos de complementos lineales (Extended Linear Complemntary ELC), así como también, sistemas de Máximos y Mínimos con Escalas (Max-Min-Plus-Scaling MMPS).

HYSDEL, se aplica en varias disciplinas para el control y simulación de procesos y ha sido probado de manera exitosa en varias aplicaciones de la industria automotriz.

2.4.2 Matlab (Matrix laboratory)

MATLAB es un software para cálculo científico, en el cual el elemento básico de datos es la matriz.

La versión original de MATLAB fue escrita en lenguaje Fortran por Cleve Moler. Utilizaba programas basados en rutinas de los códigos LINPACK y EINSPACK, máxima expresión

del software de computación matricial, utilizado para resolver numerosos problemas relacionados con los sistemas lineales y el cálculo de autovalores y auto vectores.

El actual programa MATLAB fue escrito en lenguaje C, por Steve Bangert, Steve Kleiman, John Little y Cleve Moler para la empresa The MathWorks Group. Otros muchos autores han ido incorporando una amplia gama de aplicaciones específicas, creadas para MatLab denominadas "toolbox".

Las "toolbox" son librerías de funciones MatLab que permiten resolver determinado tipo de problemas.

Está concebido para resolver problemas con formulación matricial. No necesita la definición de dimensión. La memoria se gestiona automáticamente, donde realiza, directamente, operaciones con vectores y matrices.

Son de destacar sus posibilidades gráficas bi y tridimensionales.

Es uno de los códigos más útiles en la actualidad para realizar prácticas, por su fácil aprendizaje, uso, manejo e implementación en ordenadores personales.

Otro detalle importante es que, por su estructura, cada usuario puede incorporar nuevas subrutinas y ampliar el campo de aplicaciones.

Matlab está conectado con Hysdel, ya que los resultados de Hysdel pueden ser integrados con Matlab. Todo el programa de validación fue realizado en Matlab, pero la salida de Hysdel fue tomada como entrada de Matlab. Esta salida de Hysdel es la estructura en MLD, generada a partir del modelo PWA, ejecutado en Hysdel.

La herramienta de Matlab utilizada para los procesos con MLD es la función *linprog*, la cual resuelve el problema de programación lineal para minimizar una función con las siguientes características:

$$\min_x f^T x$$
$$s.t.$$
$$Ax \leq b$$

La función *linprog*, es ejecutada en Matlab con la siguiente instrucción:

[x, EXITFLAG]= *linprog*(f,A,b),

la cual devuelve la variable EXITFLAG, que describe la condición de terminación de *linprog* así:

- > 0, cuando converge con una solución X.
- 0, cuando *linprog* alcanza el máximo número de iteraciones sin converger.
- < 0, cuando *linprog* falla, el problema no es factible.

3. CASOS DE ESTUDIO DE SISTEMAS MLD

En este capitulo se definirán algunos casos de estudio el siguiente capítulo se definirán algunos casos de estudio que servirán de modelos en la aplicación de los algoritmos de validación. Uno de los principales objetivos o resultados del proceso de validación es el de determinar si un modelo o sistema MLD está correctamente definido ("well posed") . Veamos entonces la definición formal de Sistemas MLD correctamente definidos.

3.1 DEFINICION FORMAL DE SISTEMAS MLD CORRECTAMENTE DEFINIDOS.

Reconsideremos el modelo MLD descrito por las ecuaciones (2.7) y (2.8):

$$x(t+1) = Ax(t) + B_1u(t) + B_2\delta(t) + B_3z(t)$$
$$y(t) = Cx(t) + D_1u(t) + D_2\delta(t) + D_3z(t)$$

$$E_2\delta(t) + E_3z(t) \leq E_1u(t) + E_4x(t) + E_5$$

Sea j_{Bt} que denota el conjunto de todos los índices $i \in \{1,...,r_l\}$, tales que $[B_{2t}]^i \neq 0$, donde $[B_{2t}]^i$ denota la i-ésima columna de B_{2t} . Sea j_{Dt}, J_{Bt}, y sea J_{Dt} definido análogamente por una colección de columnas con posiciones diferentes de cero (0) de D_{2t} , B_{3t} , y D_{3t} respectivamente.

Sea $j_t \underset{\equiv}{\Delta} j_{Bt} \cup j_{Dt}$, $J_t \underset{\equiv}{\Delta} J_{Bt} \cup J_{Dt}$. Un sistema MLD definido por (2.7) se dice que está bien formado ("well posed") si , $\forall t \in Z$,

(i) $x(t), u(t)$ satisfacen la ecuación **(2.8).**

Para algún $\delta(t) \in \{0,1\}^{r_l}$, $z(t) \in \Re^{r_c}$, y $x_l(t+1) \in \{0,1\}^{n_l}$;

(ii) $\forall i \in j_t$ existe una relación $D_{it} : \Re^{n+m} \to \{0,1\}$, tal que el i-ésimo componente $\delta_i(t) = D_{it}(x(t), u(t))$, y $\forall k \in J_t$ existe una relación $Z_{kt} : \Re^{n+m} \to \Re$, tal que $z_k(t) = Z_{kt}(x(t), u(t))$.

Un sistema MLD **(2.7)** se dice que es completamente bien definido ("completely well posed") si además, $j_t = \{1,...,r_l\}$ y $J_t = \{1,...,r_c\}, \forall t \in Z$.

Dicho de una forma simplificada, un sistema MLD está bien definido, si $\forall t, x(t+1)$ y $\delta(t), z(t)$ presentes en las ecuaciones **(2.7)** , están únicamente definidos por $x(t), u(t)$. Y es completamente bien definido si está bien definido y si todas $\delta(t), z(t)$ están presentes en al menos una de las ecuaciones **(2.7)**.

3.2 CASOS DE ESTUDIO

Para la evaluación del algoritmo que se implementará, se definen tres casos de prueba del algoritmo, los cuales contemplan las posibles características de los modelos a evaluar:

- Un sistema bien definido, para cada una de las condiciones de entrada existe una y solo una solución. (ver 3.2.1)
- Un sistema mal definido, en el cual, para un conjunto de valores de entrada no existe ninguna solución. (Ver 3.2.2)

- Un sistema mal definido, en el cual, para un conjunto de valores de entrada existe mas de una solución. (Ver 3.2.3)

Para la definición de los casos, se partió de una representación utilizando el modelo PWA, Se utilizó HYSDEL [14] para la conversión al modelo MLD, que es el que se validará en este trabajo.

Cada caso definido en este aparte, se escribe con la sintaxis requerida por la herramienta HYSDEL, mediante la cual se obtiene el modelo equivalente en MLD. HYSDEL genera un sistema de ecuaciones MLD equivalente. En Anexo A y Anexo B se detalla la representación utilizada y la generada por HYSDEL, cuya notación es la propuesta en [8].

3.2.1 Caso 1 Modelo de un Sistema Bien Definido

Se plantea el modelo para el sistema representado en la Figura 3. en el cual para cada una de los posibles valores de entrada existe una y solo una solución.

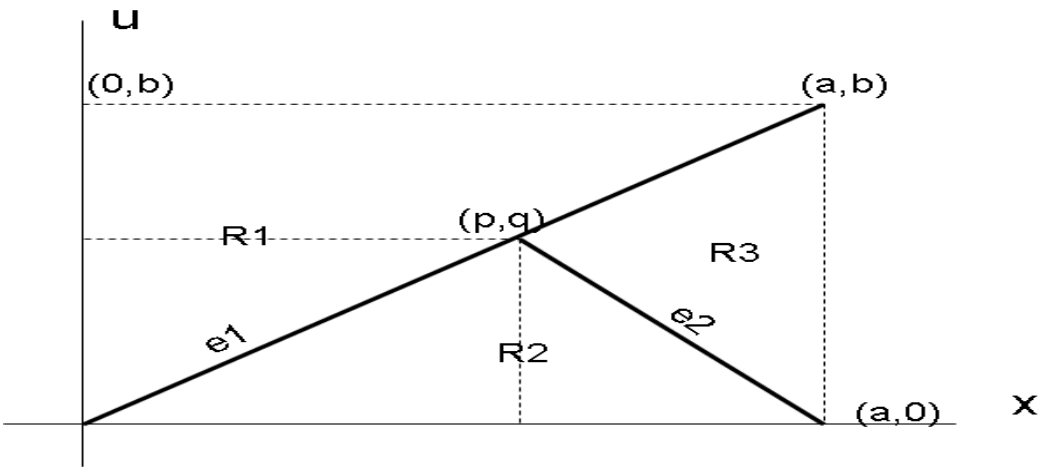

Figura 3. Modelo de un Sistema Bien Definido (Caso 1)

De la figura determinamos las siguientes restricciones para los valores de los puntos:

0<p<a y 0<q<b

Primero se definen las ecuaciones de las rectas e1 y e2.

$$e1 \rightarrow u = \frac{b}{a}x \qquad (3.1)$$

(3.1) Está definido en el intervalo $0 \le x \le a$ y $0 \le u \le b$

(3.1) Se encuentra sujeto a que el punto (p,q) está sobre la recta que va del punto 0,0 al punto (a,b), reemplazando (p,q) en 3.1 obtenemos:

$$q = \frac{b}{a}p \qquad (3.2)$$

$$e2 \rightarrow u = \frac{q}{p-a}(x-a) \qquad (3.3)$$

(3.3) está definido en el intervalo $p \le x \le a$ y $0 \le u \le q$

Reemplazando q en 3.3, por 3.2 obtenemos:

$$e2 \rightarrow u = \frac{\frac{b}{a}p}{p-a}(x-a) = \frac{bp}{a(p-a)}(x-a) \qquad (3.4)$$

Una vez definidas las rectas se definen los modelos para cada una de las áreas R1, R2 y R3:

En la región R1, se cumple

$$u \geq \frac{b}{a}x \quad \text{para los intervalos} \quad \begin{matrix} 0 \leq x \leq a \\ 0 \leq u \leq b \end{matrix} \qquad (3.5)$$

En la región R2 se cumple:

$u \leq \frac{b}{a}x$ para el intervalo $0 \leq x \leq p$ y $u \leq \frac{bp}{a(p-a)}(x-a)$ para el intervalo

$$p \leq x \leq a \qquad (3.6)$$

En la región R3 se cumple:

$u \leq \frac{b}{a}x$ para el intervalo $p \leq x \leq a$ y $u \geq \frac{bp}{a(p-a)}(x-a)$ para el intervalo

$$p \leq x \leq a \qquad (3.7)$$

El modelo está dado por las ecuaciones: 3.5, 3.6 y 3.7

Para evitar el sobrelapamiento de los modelos en las regiones límites, se adicionan las siguientes restricciones:

$R1 \rightarrow\sim R2 \sim R3$

$R2 \rightarrow\sim R1 \sim R3$

$R3 \rightarrow \sim R1 \wedge \sim R2$

3.2.2 Caso 2 Modelo de un sistema sin condición de existencia (ausencia de solución)

Se plantea el modelo para el sistema representado en la Figura 4, en este caso no se representan soluciones para la región R4.

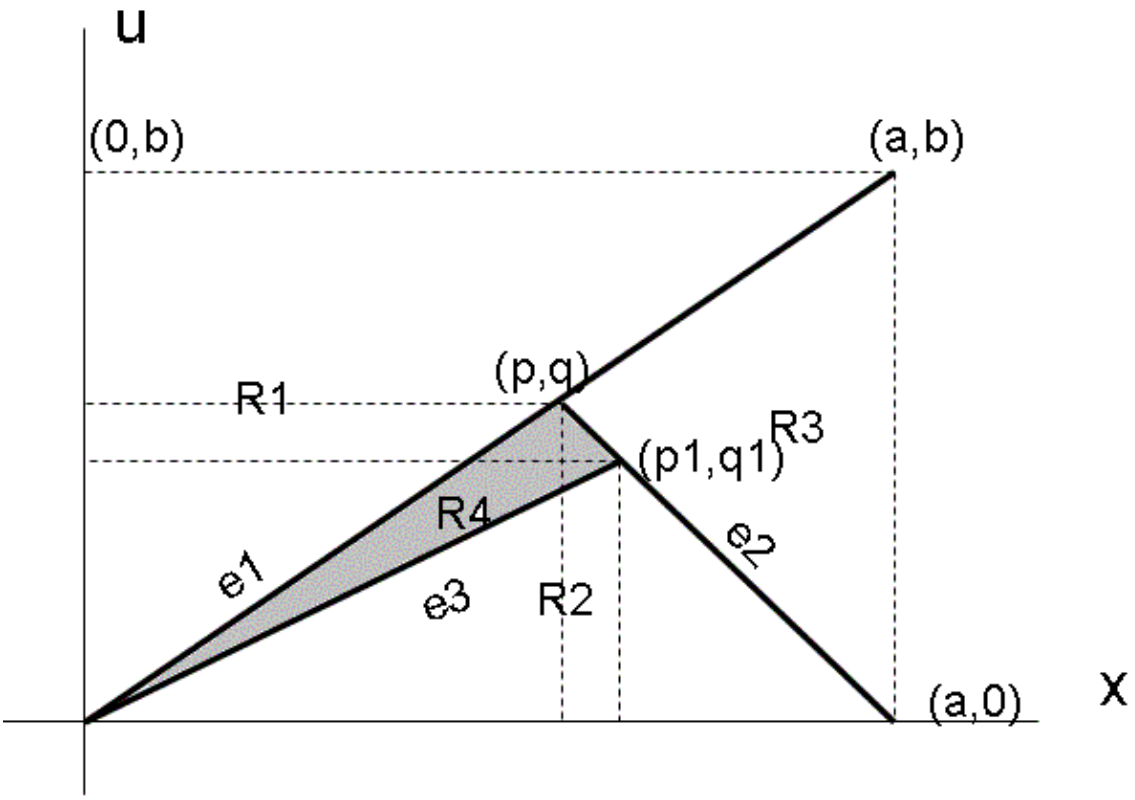

Figura 4. Modelo de un sistema sin condición de existencia (Caso 2)

De la Figura 4 determinamos las siguientes restricciones para los valores de los puntos:

$0 \leq p < p1 \leq a$ y $0 \leq q1 < q \leq b$

Al modelo del caso 1 se le hacen las siguientes variaciones:

- Se agrega la ecuación de la recta e3, que va del punto (0,0) al punto (p1,q1).
- Se reemplaza la región R2 del caso 1, por el área debajo de las líneas e3 y e2, triángulo con los vértices en los puntos: (0,0), (p1,q1) y (a,0) .

Observamos que con el ánimo de dejar incompleto el modelo, no se define la región R4. que es el área (triangulo) delimitada por los puntos (0,0), (p,q) y (p1,q1).

$$\text{e3} \rightarrow u = \frac{q1}{p1}x \text{ para } \quad 0 \le x \le p1 \text{ y } \ 0 \le u \le q1 \qquad (3.8)$$

El punto (p1,q1) se encuentra sobre la recta e2, por lo tanto, reemplazando (p1,q1) en 3.4 obtenemos:

$$q1 = \frac{bp}{a(p-a)}(p1-a) \qquad (3.9)$$

Definimos la ecuación e2' que va de (a,0) a (p1,q1), es igual a 3.4 pero con restricción de los dominios de x y u, así

$$\text{e2'} \rightarrow \quad u = \frac{bp}{a(p-a)}(x-a) \qquad (3.10)$$

(3.10) está definido en el intervalo $p1 \le x \le a$ y $0 \le u \le q1$

En la nueva región R2 se cumple:

$$u \le \frac{q1}{p1} x \text{ para el intervalo } \quad 0 \le x \le p1 \text{ y } \quad 0 \le u \le q1 \qquad (3.11)$$

$$u \le \frac{bp}{a(p-a)}(x-a) \text{ para el intervalo } \quad p1 \le x \le a \text{ y } \quad 0 \le u \le q1 \qquad (3.12)$$

El punto (p1,q1) está sobre la línea e2, con las siguientes propiedades:

1) p1 es mayor que p, suponemos p1= p+ Cte , donde Cte: Constante positiva.

2) De acuerdo con (3.8) q1, satisface la siguiente ecuación:

$q1 = \frac{bp}{a(p-a)}(p1-a)$ se reemplaza p1 por (p+Cte),

Obteniendo: $q1 = \frac{bp}{a(p-a)}(p + Cte - a)$

En el Anexo D se presenta el modelo Hysdel de este caso.

3.2.3 Caso 3 Modelo de un sistema sin condición de unicidad (soluciones múltiples)

Se plantea el modelo para el sistema representado en la Figura 5, en este caso la región R4 tiene múltiples soluciones.

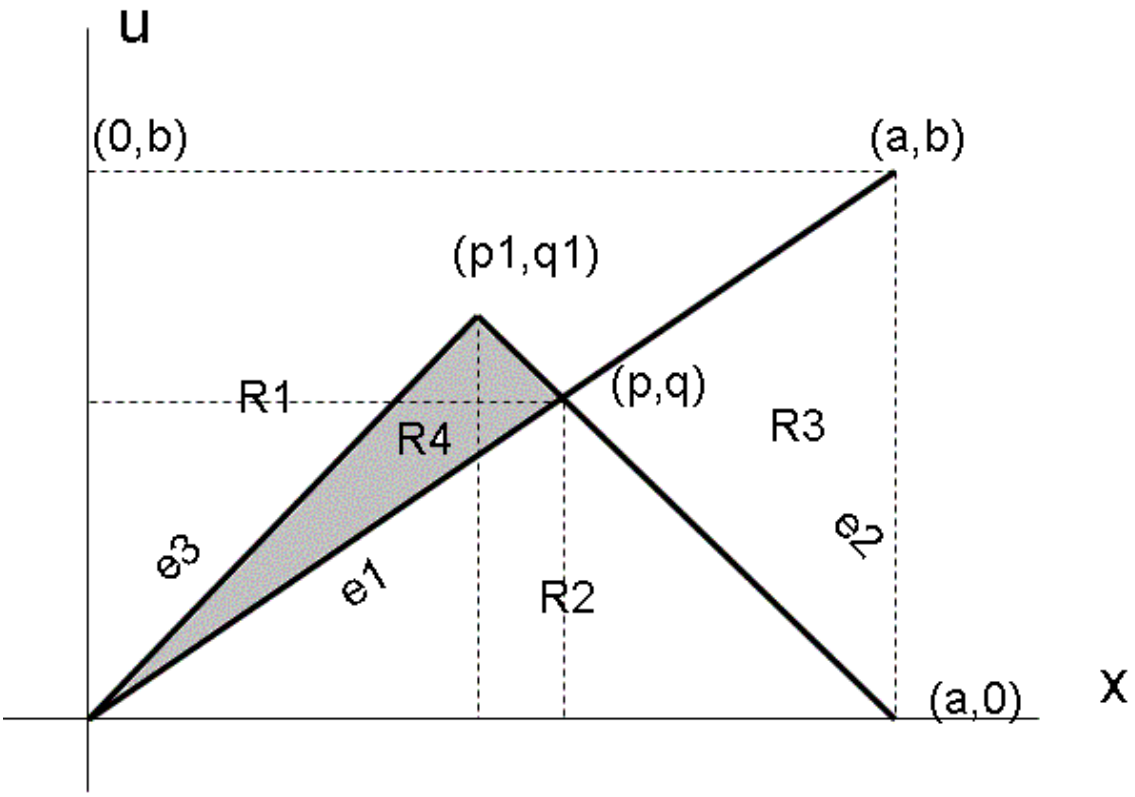

Figura 5. Modelo de un sistema sin condición de unicidad (soluciones múltiples) Caso 3

Para la obtención de este modelo, al modelo del caso 1 se le hacen las siguientes variaciones:

- Se agrega la ecuación de la línea e3 (ver figura 3.3).
- La ecuación de la línea e2 se extiende hasta el punto (p1,q1).
- La región R2 se reemplaza por el área bajo las curvas e3 y e2, dejando las otras dos regiones iguales al caso 1.

Se observa que el área de la región R4 de la figura 3.3, ubicada entre los puntos (0,0), (p,q) y (p1,q1), se encuentra tanto en la región R1 del caso 1, como en la nueva definición de la región R2 de este caso.

$$\text{e3} \rightarrow u = \frac{q1}{p1}x \text{ para } \quad 0 \le x \le p1 \text{ y } 0 \le u \le q1 \qquad (3.13)$$

La nueva representación de la región R2 quedara así:

$$u \le \frac{q1}{p1}x \text{ para } 0 \le x \le p1 \quad \text{y} \quad u \le \frac{bp}{a(p-a)}(x-a) \text{ para } p1 \le x \le a \qquad (3.14)$$

El punto (p1,q1) está sobre la línea e2, con las siguientes propiedades:

1) p1 es menor que p, suponemos p1= p- Cte , donde Cte es una constante positiva.

2) Para deducir el punto q1, reemplazamos e2 de (3.4) obtenemos:

$q1 = \frac{bp}{a(p-a)}(p1-a)$ se reemplaza p1 por (p-Cte),

Obteniendo: $q1 = \frac{bp}{a(p-a)}(p - Cte - a)$

En este caso las siguientes restricciones son eliminadas:

$R1 \rightarrow \sim R2 \wedge \sim R3$

$R2 \rightarrow \sim R1 \wedge \sim R3$

$R3 \rightarrow \sim R1 \wedge \sim R2$

En el Anexo E. se presenta el modelo Hysdel de este caso.

4. ALGORITMO DE VALIDACIÓN

En este capitulo se presenta el algoritmo de validación propuesto. En la primera sección se introduce el algoritmo, el cual requiere a su vez de un algoritmo de programación matemática de variables enteras MILP. Dado que el fundamento de los algoritmos de programación matemática con variables enteras es uno conocido como "Branch and Bound", este se explica en la sección 4.2 . En la sección 4.3 se aplica el algoritmo propuesto (secciones 4.1 y 4.2) a los casos de estudio establecidos en 3. con lo que se pretende validar el algoritmo propuesto. En la sección 4.4 se comenta la complejidad computacional del algoritmo propuesto.

4.1 EL ALGORITMO DE VALIDACIÓN

4.1.1 Existencia de soluciones

Recordemos que los modelos MLD están representados por las siguientes ecuaciones:

$$x(t+1) = Ax(t) + B_1u(t) + B_2\delta(t) + B_3z(t)$$
$$y(t) = Cx(t) + D_1u(t) + D_2\delta(t) + D_3z(t)$$

$$E_2\delta(t) + E_3z(t) \leq E_1u(t) + E_4x(t) + E_5$$

La existencia de la solución implica que para todo x(t) y u(t) existe al menos una solución a las ecuaciones y por tanto un valor de x(t+1), y(t), δ(t) y z(t). De donde se deduce directamente que dados un $x^*(t)$ y $u^*(t)$ especificados, se pueden plantear las siguientes ecuaciones:

$$x(t+1) - B_2\delta(t) - B_3 z(t) = Ax^*(t) + B_2 u^*(t)$$

$$y(t) - D_2\delta(t) - D_3 z(t) = Cx^*(t) + D_1 u^*(t)$$

$$E_2\delta(t) + E_3 z(t) \le E_4 x^*(t) + E_1 u^*(t)$$

El problema se reduce entonces a verificar la factibilidad de las anteriores ecuaciones, las cuales se deben reescribir de acuerdo a la forma general:

$$\min_{X_p} F^T X_p$$

$$s.t.$$

$$A_p X_p \le B$$

Para llegar a dicha formulación se reescriben las ecuaciones en términos de desigualdades de la siguiente forma:

$$x(t+1) - B_2\delta(t) - B_3 z(t) \le Ax^*(t) + B_2 u^*(t)$$

$$-x(t+1) + B_2\delta(t) + B_3 z(t) \le -Ax^*(t) - B_2 u^*(t)$$

$$y(t) - D_2\delta(t) - D_3 z(t) \le Cx^*(t) + D_1 u^*(t)$$

$$-y(t) + D_2\delta(t) + D_3 z(t) \le -Cx^*(t) - D_1 u^*(t)$$

$$E_2\delta(t) + E_3 z(t) \le E_4 x^*(t) + E_1 u^*(t)$$

La función objetivo puede ser cualquier función lineal positiva. De donde se obtiene la siguiente descripción matricial

$$Xp = \begin{bmatrix} x(t+1) \\ y(t) \\ \delta(t) \\ z(t) \end{bmatrix} \quad Ap = \begin{bmatrix} 1 & 0 & -B_2 & -B_3 \\ -1 & 0 & B_2 & B_3 \\ 0 & 1 & -D_2 & -D_3 \\ 0 & -1 & D_2 & D_3 \\ 0 & 0 & E_2 & E_3 \end{bmatrix} \quad B = \begin{bmatrix} Ax^*(t) + B_1 u^*(t) \\ -Ax^*(t) - B_1 u^*(t) \\ Cx^*(t) + D_1 u^*(t) \\ -Cx^*(t) - D_1 u^*(t) \\ E_4 x^*(t) + E_1 u^*(t) \end{bmatrix} \quad F = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Para encontrar Xp se debe utilizar una versión del algoritmo conocido como branch and bound.

Debemos utilizar programación lineal entera mixta porque las variables $\delta_i(t)$ en Xp son variables binarias y las otras variables en Xp son reales.

Al conjunto de restricciones iniciales definidas por MLD agregamos las siguientes restricciones por cada una de las variables binarias: $\delta_i \geq 0$ y $\delta_i \leq 1$.

4.1.2 Unicidad de soluciones

Después de evaluar la factibilidad del problema planteado en la sección anterior, se debe determinar si existe mas de una solución al sistema de ecuaciones planteado.

Para realizar esta tarea, y siguiendo el algoritmo propuesto en [8], se realiza el siguiente planteamiento.

Para todo $v \in X_v$ existe solamente un vector $s \in X_s$, que satisface:

$$s = H_1 w + H_2 v$$

$$K_1 w \leq K_2 v + K_3$$

Donde:

$$s = \begin{bmatrix} x(t+1) \\ y(t) \end{bmatrix} \quad v = \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \quad w = \begin{bmatrix} \delta(t) \\ z(t) \end{bmatrix} \quad H_1 = \begin{bmatrix} B_2 & B_3 \\ D_2 & D_3 \end{bmatrix} \quad H_2 = \begin{bmatrix} A & B_1 \\ C & D_1 \end{bmatrix}$$

$$K_1 = \begin{bmatrix} E_2 & E_3 \end{bmatrix} \quad K_2 = \begin{bmatrix} E_4 & E_1 \end{bmatrix} \quad K_3 = E_5$$

Ahora, consideremos la posibilidad de que *s* no sea único, si dicha situación se diera significaría existiría un $s_- \neq s_+$ tal que el siguiente conjunto de ecuaciones sería factible:

$$s_{-}^{i} = H_1 w_{-} + H_2 v$$

$$s_{+}^{i} = H_1 w_{+} + H_2 v$$

$$K_1 w_{-} \leq K_2 v + K_3$$

$$K_1 w_{+} \leq K_2 v + K_3$$

$$s_{-}^{i} < s_{+}^{i}$$

El problema consiste en verificar si el conjunto de las ecuaciones anteriores son factibles, si la factibilidad existe, significa que *s* tiene mas de un valor y por tanto el sistema estaría mal definido por multiplicidad de soluciones.

Este algoritmo supone que de existir múltiples soluciones, estas se encontrarían en una vecindad de la solución originalmente propuesto.

En los casos de estudio propuestos en el capítulo 3, en particular el caso 3, existen múltiples soluciones que no se ajustan a este supuesto.

Por tal razón se propone el siguiente algoritmo. Partiendo del hecho que dos soluciones diferentes de *s* generan un costo diferente en la función de optimización, siempre y cuando la función $f^T x$ sea convexa, se procede a evaluar el mínimo y el máximo del problema de optimización, si ambos costos difieren o si alguno de los problemas presenta infactibilidad, entonces se deduce que el sistema MLD estaría mal definido.

Este procedimiento se resume en el siguiente algoritmo

Algoritmo 1. ***Validación de Múltiples Soluciones del Modelo MLD***

Paso 0. i=Min x(t), j=Min u(t), inc= ε *, biendefinido=1*

Paso 1. $s_1 = MinF' X_p$

Paso 2. $s_2 = Min - F' X_p$

Paso 3. Si los s en s_1 *son* $\neq$ *a los s en* s_2 *biendefinido=*-1*,vaya a 7*

Paso 4. j=j+inc, Si j<=Max u(t) vaya 1

Paso 5. j=Min u(t)

Paso 6. i=i+inc, Si i<=Max x(t) vaya 1

Paso 7. Detenerse.

En el Anexo G, está la codificación de este algoritmo en MATLAB. Este algoritmo es aplicado a los tres casos de estudio.

4.2 METODO BRANCH & BOUND

Como hemos visto MLD es básicamente un sistema representado por un programa matemático, es decir, una función objetivo con un conjunto de restricciones lineales. Para resolver este problema es necesario utilizar un algoritmo de optimización que incluya variables binarias y variables continuas.

La técnica de Branch & Bound es sin duda el algoritmo de programación entera más conocido y que mas aplicaciones ha tenido. De ahí que la investigación sobre el terreno de Branch & Bound sea de gran importancia, y que sea uno de los algoritmos básicos para este trabajo. Este aparte sigue de las referencias **[27**], **[28**] y **[29**]

Branch and Bound es una técnica basada en el análisis del árbol de estados de un problema, en la enumeración inteligente de todas las soluciones posibles en un problema de optimización combinatoria. Realiza un recorrido sistemático de ese árbol, que no tiene que ser necesariamente en profundidad.

Los algoritmos generados por está técnica son normalmente de orden exponencial en su peor caso, pero su aplicación en casos no muy grandes, ha demostrado ser eficiente.

Planteamiento formal de este método:

Definición 1. Un árbol dirigido con raíz es un grafo dirigido, conexo y acíclico que tiene un vértice distinguido ***s*** al que llamamos raíz tal que para cualquier otro vértice ***v*** hay un camino dirigido de ***s*** a ***v***.

Observación 1. Si (*u; v*) es una rama de un árbol dirigido con raíz entonces (*v; u*) no lo es.

Definición 2. Si (*u; v*) es una rama de un árbol dirigido con raíz diremos que ***u*** es el padre de ***v*** y que ***v*** es el hijo de ***u***. Dado un vértice ***u***, el conjunto de vértices ***v*** tales que existe un camino dirigido de ***u*** a ***v*** se llama la descendencia de ***u***. Se dice que el árbol es binario si cada vértice que no sea una hoja tiene exactamente dos hijos (nótese que las hojas son los vértices que no tienen hijos). Al conjunto de hijos de la raíz lo llamamos el primer nivel, al conjunto de nietos de la raíz (es decir, al conjunto de hijos de los hijos de la raíz) lo llamamos el segundo nivel, etc.

Por ejemplo, la Figura 6 es un árbol dirigido con raíz *s* y cuatro niveles.

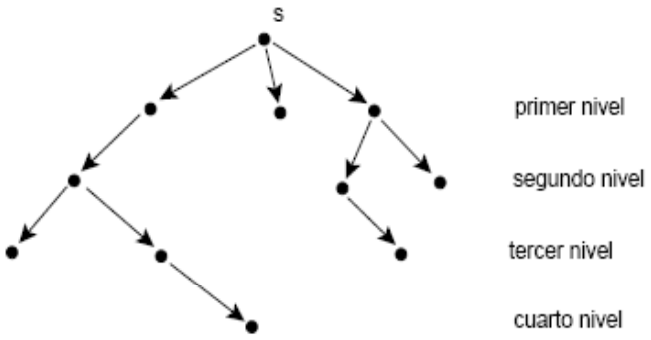

Figura 6. Arbol Dirigido

Observación 2. Si un árbol dirigido con raíz es binario y tiene ***n*** niveles entonces tiene $2n+1 - 1$ vértices.

Consideremos el siguiente problema: dado un árbol dirigido con raíz ***s***, donde cada vértice ***x*** tiene asignado un costo $c(x)$ que satisface $c(x) \leq c(y)$ para toda rama $(x; y)$ queremos hallar una hoja de mínimo costo.

El procedimiento utiliza una mezcla de "backtracking" (volver al vértice anterior para examinar alguno de sus hijos que todavía no ha sido examinado) y un criterio particular de poda que consiste en eliminar toda la descendencia de un vértice ***x*** cuando se satisfaga que $c(x) \geq c(h)$ para alguna hoja ***h*** encontrada previamente (es decir, cuando ninguna hoja descendiente de ***x*** puede ser la solución óptima del problema). El algoritmo quedaría :

Algoritmo 2. Búsqueda de Mínimo (B&B)

Paso 0. $L = \{s\}$, $c = \infty$.

Paso 1. Si ***x*** *es de los elementos de L, el último que ingresó,*

$L = L - \{x\}$. *Calcular c(x).*

Si $c(x) \geq c$, *vaya a 4.*

Paso 2. Si ***x*** *no es una hoja,* $L = L \cup \{$*hijos de* ***x***$\}$, *vaya a 4.*

Paso 3. $h = x$, $c = c(x)$.

Paso 4. Si $L \neq \emptyset$; *vaya a 1.*

Paso 5. DETENGASE.

En el Anexo F. se encuentra la implementación de este algoritmo.

En el caso en que la función c satisfaga $c(x) \geq c(y)$ para toda rama (x , y), podemos resolver el problema de hallar una hoja de máximo costo utilizando el algoritmo que resulta de reemplazar, en el paso 0. , $c = \infty$ por $c = -\infty$ y en el paso 1., la condición $c(x) \geq c$ por la condición $c(x) \leq c$, es decir, el algoritmo quedaría:

Algoritmo 3. Busqueda del Máximo *(BRANCH AND BOUND)*

Paso 0'. L = { s }, c = -∞ .

Paso 1'. Sea x es el último vértice que ingresó en L, L = L – { x }.

Calcular c(x). Si c(x) ≤ c, vaya a 4.

Paso 2'. Si x no es una hoja, L = L { hijos de x }, vaya a 4.

Paso 3'. h = x, c = c(x).

Paso 4'. Si L ≠ ø vaya a 1.

Paso 5'. DETENGASE.

La implementación de este algoritmo en Matlab, esta en el Anexo F.

Tomando este conjunto de restricciones como nuestro *s* inicial aplicamos el Algoritmo 3.

Busqueda del Máximo (*BRANCH AND BOUND).*

Un vértice del árbol será una hoja si la solución optima cumple con la restricción que los δ_i son binarios o la solución no es factible. Cuando un vértice no es una hoja, encontramos la primera δ_i que no cumple con las restricción y se agregan dos restricciones mas a la pila $\delta_i \leq 0$ y $\delta_i \geq 1$.

Ejemplo:

Consideremos el caso de un sistema MLD representado por las siguientes matrices

A=[-0.8]; B1=[1]; B2=[0]; B3=[1.6]; B5=[0];

C=[1]; D1=[0]; D2=[0]; D3=[0]; D5=[0];

$$E_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad E_2 = \begin{bmatrix} 10 \\ -(\varepsilon + 10) \\ -10 \\ -10 \\ -10 \\ 10 \end{bmatrix} \quad E_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \quad E_4 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 1 \\ -1 \end{bmatrix} \quad E_5 = \begin{bmatrix} 10 \\ -\varepsilon \\ 0 \\ 0 \\ 10 \\ 10 \end{bmatrix}$$

Recordemos que dichas matrices hacen referencia a la siguiente formulación de un modelo MLD tenemos que:

$$x(t+1) = Ax(t) + B_1 u(t) + B_2 \delta(t) + B_3 z(t) + B_5$$

$$y(t) = Cx(t) + D_1 u(t) + D_2 \delta(t) + D_3 z(t) + D_5$$

$$E_2 \delta(t) + E_3 z(t) \leq E_1 u(t) + E_4 x(t) + E_5$$

El autómata en la figura 7 representa el sistema expuesto en las anteriores ecuaciones:

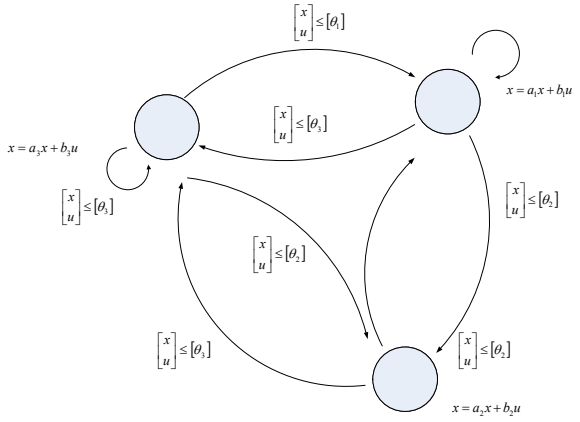

Figura 7. Máquina de Estado Finito

Uno b = $\begin{bmatrix} x \\ u \end{bmatrix} \leq [\theta_1]$ Dos b = $\begin{bmatrix} x \\ u \end{bmatrix} \leq [\theta_2]$ Tres b = $\begin{bmatrix} x \\ u \end{bmatrix} \leq [\theta_3]$

Es continua para:

$$X = Unob(a_1x + b_1u) + Dosb(a_2x + b_2u) + Tresb(a_3x + b_3u)$$

El problema entonces es encontrar X a partir de un valor de *x* y de *u*, la solución implica encontrar cual de las tres variables binarias Unob, Dosb ó Tresb está activa.

Al ejecutar la solución que resulta es:

Xp inicial $[-7.0000 \quad 0.0000 \quad 0.5000 \quad -5.0000]^T$

C(s)= -11.5000

Donde Xp se encuentra codificado como $[\text{Unob Dosb Tresb X}]^T$, y $x = 0$, $u = 0$.

Se agrega al árbol el vértice con la restricción $\delta \leq 0$ y el vértice con la restricción $\delta \geq 1$

Se procesa el vértice >=1 y se obtiene

Xp=$[1.0000 \quad 0 \quad 1.0000 \quad -0.0000]^T$

C(s)= 2.0000

Se procesa vértice <= 0

Xp=$[1.0000 \quad -0.0000 \quad 0.0000 \quad 0.0000]^T$

C(s)= 1.0000

La solución final es que el sistema se encuentra en la región 1 (Unob=1, Dosb=0 y Tresb=0), y X = 0

En el Anexo G, está la codificación del algoritmo de Branch & Bound implementado en MATLAB.

4.3 RESULTADOS Y ANÁLISIS DE RESULTADOS

Al aplicar el algoritmo de validación desarrollado a cada uno de los casos se obtuvieron los siguientes resultados, utilizando un computador con las siguientes características: Procesador Pentium IV de 2.8 Ghz, memoria RAM de 1 Gb, disco duro de 40 Gb.

CASO 1. Modelo de un sistema bien definido

La formulación expresada en el capitulo 3. esta parametrizada permitiendo escoger un amplio rengo de valores, se escogen arbitrariamente teniendo en cuenta la visualización de los resultados , los cuales deben ser fácilmente interpretables.

a1=0.5, b1=0.5, a2=0.4, b2=0.6, a3=0.3, b3=0.7, p=2, a=5, b=5

En los resultados obtenidos al ejecutar el proceso podemos observar que en 94 puntos evaluados el valor del EXIT-FLAG del mínimo y el máximo son diferentes. Analizando estos puntos encontramos lo siguiente:

Tabla 1. Puntos del Caso 1 con ExitFlag Diferentes para Mínimo y Máximo, con exit flag del Mínimo igual a cero.

INDICE	X	U	X(k+1) MIN	Y MIN	FLAG MIN	X(k+1) MAX	Y MAX	FLAG MAX
26	0	2.5	0.00	0	0	1.25	0	1
27	0	2.6	0.00	0	0	1.3	0	1
28	0	2.7	0.00	0	0	1.35	0	1
39	0	3.8	0.70	0	0	1.9	0	1
40	0	3.9	0.85	0	0	1.95	0	1
41	0	4	1.00	0	0	2	0	1
42	0	4.1	1.15	0	0	2.05	0	1
43	0	4.2	1.30	0	0	2.1	0	1
47	0	4.6	1.90	0	0	2.3	0	1

El valor del Exit-Flag del mínimo es 0, porque la función *linprog* de MATLAB alcanza el máximo de iteraciones sin converger, pero el máximo si es factible y es una solución valida.

Para los datos correspondientes a los puntos 20, 970 y 1637 (ver Tabla 2) al minimizar se encuentra que no es factible, mientras que el máximo si lo es.

Tabla 2. Puntos del caso 1, factibles solo al maximizar.

INDICE	X	U	X(k+1) MIN	Y MIN	FLAG MIN	X(k+1) MAX	Y MAX	FLAG MAX
20	0	1.9	0.00	0	-1	0.95	0	1
970	1.9	0	0.00	1.9	-1	0.76	1.9	1
1637	3.2	0.4	0.00	3.2	-1	1.52	3.2	1

Para los otros 82 puntos se encuentra que incrementando y disminuyendo el valor de tolerancia utilizado para aceptar un valor como el entero "uno" o "cero", se logra hacer factible la solución al minimizar y los valores de X(k+1) y Y, coinciden con los valores al maximizar la función.

Tabla 3. Puntos del Caso 1 en que se logra la factibilidad con ajuste al parámetro de tolerancia.

INDICE	X	U	X(k+1) MIN	Y MIN	FLAG MIN	X(k+1) MAX	Y MAX	FLAG MAX
29	0	2.8	0.00	0	-1	1.4	0	1
30	0	2.9	0.00	0	-1	1.45	0	1
31	0	3	0.00	0	-1	1.5	0	1
32	0	3.1	0.00	0	-1	1.55	0	1
34	0	3.3	0.00	0	-1	1.65	0	1
45	0	4.4	1.60	0	-1	2.2	0	1
104	0.2	0.1	0.00	0.2	-1	0.14	0.2	1
362	0.7	0.4	0.00	0.7	-1	0.52	0.7	1
410	0.8	0.1	0.00	0.8	-1	0.38	0.8	1
415	0.8	0.6	0.00	0.8	-1	0.68	0.8	1
437	0.8	2.8	0.00	0.8	-1	1.8	0.8	1
463	0.9	0.3	0.00	0.9	-1	0.54	0.9	1
464	0.9	0.4	0.00	0.9	-1	0.6	0.9	1

465	0.9	0.5	0.00	0.9	-1	0.66	0.9	1
515	1	0.4	0.00	1	-1	0.64	1	1
564	1.1	0.2	0.00	1.1	-1	0.56	1.1	1
566	1.1	0.4	0.00	1.1	-1	0.68	1.1	1
615	1.2	0.2	0.00	1.2	-1	0.6	1.2	1
621	1.2	0.8	0.00	1.2	-1	0.96	1.2	1
622	1.2	0.9	0.00	1.2	-1	1.02	1.2	1
673	1.3	0.9	0.00	1.3	-1	1.06	1.3	1
676	1.3	1.2	0.00	1.3	-1	1.24	1.3	1
719	1.4	0.4	0.00	1.4	-1	0.8	1.4	1
720	1.4	0.5	0.00	1.4	-1	0.86	1.4	1
725	1.4	1	0.00	1.4	-1	1.16	1.4	1
767	1.5	0.1	0.00	1.5	-1	0.66	1.5	1
773	1.5	0.7	0.00	1.5	-1	1.02	1.5	1
776	1.5	1	0.00	1.5	-1	1.2	1.5	1
823	1.6	0.6	0.00	1.6	-1	1	1.6	1
828	1.6	1.1	0.00	1.6	-1	1.3	1.6	1
831	1.6	1.4	0.00	1.6	-1	1.48	1.6	1
872	1.7	0.4	0.00	1.7	-1	0.92	1.7	1
884	1.7	1.6	0.00	1.7	-1	1.64	1.7	1
919	1.8	0	0.00	1.8	-1	0.72	1.8	1
921	1.8	0.2	0.00	1.8	-1	0.84	1.8	1
1027	2	0.6	0.00	2	-1	1.16	2	1
1033	2	1.2	0.00	2	-1	1.52	2	1
1072	2.1	0	0.00	2.1	-1	0.84	2.1	1
1078	2.1	0.6	0.00	2.1	-1	1.2	2.1	1
1082	2.1	1	0.00	2.1	-1	1.44	2.1	1
1125	2.2	0.2	0.00	2.2	-1	1	2.2	1
1140	2.2	1.7	0.00	2.2	-1	1.9	2.2	1
1177	2.3	0.3	0.00	2.3	-1	1.1	2.3	1
1178	2.3	0.4	0.00	2.3	-1	1.16	2.3	1
1179	2.3	0.5	0.00	2.3	-1	1.22	2.3	1
1183	2.3	0.9	0.00	2.3	-1	1.46	2.3	1
1184	2.3	1	0.00	2.3	-1	1.52	2.3	1
1186	2.3	1.2	0.00	2.3	-1	1.64	2.3	1
1228	2.4	0.3	0.00	2.4	-1	1.14	2.4	1
1231	2.4	0.6	0.00	2.4	-1	1.32	2.4	1
1232	2.4	0.7	0.00	2.4	-1	1.38	2.4	1
1237	2.4	1.2	0.00	2.4	-1	1.68	2.4	1
1240	2.4	1.5	0.00	2.4	-1	1.86	2.4	1
1241	2.4	1.6	0.00	2.4	-1	1.92	2.4	1
1283	2.5	0.7	0.00	2.5	-1	1.42	2.5	1
1285	2.5	0.9	0.00	2.5	-1	1.54	2.5	1
1288	2.5	1.2	0.00	2.5	-1	1.72	2.5	1
1290	2.5	1.4	0.00	2.5	-1	1.84	2.5	1
1379	2.7	0.1	0.00	2.7	-1	1.14	2.7	1
1384	2.7	0.6	0.00	2.7	-1	1.44	2.7	1
1386	2.7	0.8	0.00	2.7	-1	1.56	2.7	1
1441	2.8	1.2	0.00	2.8	-1	1.84	2.8	1

1536	3	0.5	0.00	3	-1	1.5	3	1
1582	3.1	0	0.00	3.1	-1	1.24	3.1	1
1594	3.1	1.2	0.00	3.1	-1	1.96	3.1	1
1636	3.2	0.3	0.00	3.2	-1	1.46	3.2	1
1638	3.2	0.5	0.00	3.2	-1	1.58	3.2	1
1639	3.2	0.6	0.00	3.2	-1	1.64	3.2	1
1644	3.2	1.1	0.00	3.2	-1	1.94	3.2	1
1688	3.3	0.4	0.00	3.3	-1	1.56	3.3	1
1689	3.3	0.5	0.00	3.3	-1	1.62	3.3	1
1691	3.3	0.7	0.00	3.3	-1	1.74	3.3	1
1744	3.4	0.9	0.00	3.4	-1	1.9	3.4	1
1745	3.4	1	0.00	3.4	-1	1.96	3.4	1
1792	3.5	0.6	0.00	3.5	-1	1.76	3.5	1
1894	3.7	0.6	0.00	3.7	-1	1.84	3.7	1
1943	3.8	0.4	0.00	3.8	-1	1.76	3.8	1
1944	3.8	0.5	0.00	3.8	-1	1.82	3.8	1
2097	4.1	0.5	0.00	4.1	-1	1.94	4.1	1
2143	4.2	0	0.00	4.2	-1	1.68	4.2	1
2564	5	1.3	0.00	5	-1	2.41	5	1
2582	5	3.1	1.58	5	-1	3.67	5	1

El algoritmo da como resultado que el modelo MLD para este caso, está bien definido, se procesaron 2601 soluciones iniciando con los valores Min x(t)=0.0, Min u(t)=0.0,incrementos de inc=0.1, y valores finales Max x(t)=5.0 y Max u(t)=5.0; El tiempo de ejecución del algoritmo fue 8.7912 Horas.

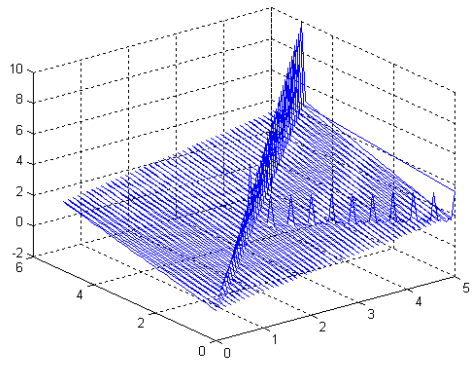

Figura 8 Caso1 $s_2 = Min - F' X_p$ x(t+1)

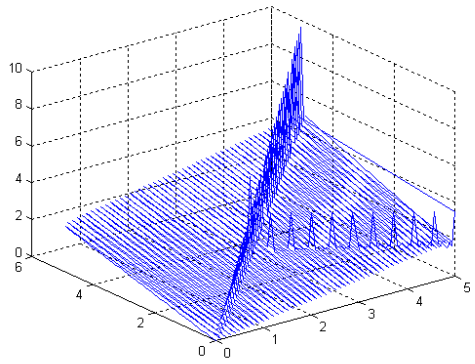

Figura 9. Caso1 $s_1 = MinF'X_p$ x(t+1)

En las gráficas anteriores se puede observar claramente, que todos los puntos cumplen con las condiciones de existencia, continuidad y unicidad, por tanto el modelo está correctamente definido.

CASO2. Modelo de un sistema sin condición de existencia (Ausencia de solución)

La formulación expresada en el capitulo 3. esta parametrizada permitiendo escoger un amplio rengo de valores, se escogen arbitrariamente teniendo en cuenta la visualización de los resultados , los cuales deben ser fácilmente interpretables. Se tomaron los siguientes valores:

a1=0.5, b1=0.5, a2=0.4, b2=0.6, a3=0.3, b3=0.7, p=2, a=5, b=5, c=p/4, p1=p+c, q1= b*p*(p1-a)/(a*(p-a))

En los resultados obtenidos al ejecutar el proceso se puede observar que en 83 puntos evaluados el valor del EXIT-FLAG del mínimo y el máximo son diferentes.

Analizando estos puntos se encuentra en la Tabla 4 lo siguiente:

Tabla 4. Puntos del Caso 2 con ExitFlag Diferentes para Mínimo y Máximo, con exit flag del Mínimo igual a cero.

INDICE	X	U	X(k+1)MIN	Y MIN	FLAG MIN	X(k+1) MAX	Y MAX	FLAG MAX
37	0,0	3,6	0,4000	0,0000	0	1,8000	0,0000	1
44	0,0	4,3	1,4500	0,0000	0	2,1500	0,0000	1
45	0,0	4,4	1,6000	0,0000	0	2,2000	0,0000	1
46	0,0	4,5	1,7500	0,0000	0	2,2500	0,0000	1

El valor del Exit-flag del mínimo es 0 por que la función linprog de MATLAB alcanza el máximo de iteraciones sin converger, pero el máximo si es factible y es una solución válida.

Para los datos correspondientes a los puntos 655, 1135, 1642 y 1740, descritos en la Tabla 5, al minimizar se encuentra que no es factible, mientras que el máximo si lo es.

Tabla 5. Puntos del Caso 2 con mínimo no factible, y máximo factible.

INDICE	X	U	X(k+1)MIN	Y MIN	FLAG MIN	X(k+1) MAX	Y MAX	FLAG MAX
655	1,2	4,2	0,0000	1,2000	-1	2,7000	1,2000	1
1135	2,2	1,2	0,0000	2,2000	-1	1,6000	2,2000	1
1642	3,2	0,9	0,0000	3,2000	-1	1,8200	3,2000	1
1740	3,4	0,5	0,0000	3,4000	-1	1,6600	3,4000	1

Para los otros 75 puntos, relacionados en la. Tabla 6 se encuentra que incrementando y disminuyendo el valor de tolerancia utilizado para aceptar un valor como el entero “uno” o “cero”, se logra hacer factible la solución al minimizar y los valores de X(k+1) y Y, coinciden con los valores al maximizar la función.

Tabla 6. Puntos del Caso 2 en que se logra la factibilidad con ajuste al parámetro de tolerancia.

INDICE	X	U	X(k+1)MIN	Y MIN	FLAG MIN	X(k+1) MAX	Y MAX	FLAG MAX
417	0,8	0,8	0,0000	0,8000	-1	0,8000	0,8000	1
460	0,9	0,0	0,0000	0,9000	-1	0,3600	0,9000	1
461	0,9	0,1	0,0000	0,9000	-1	0,4200	0,9000	1
465	0,9	0,5	0,0000	0,9000	-1	0,6600	0,9000	1
469	0,9	0,9	0,0000	0,9000	-1	0,9000	0,9000	1
569	1,1	0,7	0,0000	1,1000	-1	0,8600	1,1000	1
620	1,2	0,7	0,0000	1,2000	-1	0,9000	1,2000	1
671	1,3	0,7	0,0000	1,3000	-1	0,9400	1,3000	1
721	1,4	0,6	0,0000	1,4000	-1	0,9200	1,4000	1
768	1,5	0,2	0,0000	1,5000	-1	0,7200	1,5000	1
771	1,5	0,5	0,0000	1,5000	-1	0,9000	1,5000	1
774	1,5	0,8	0,0000	1,5000	-1	1,0800	1,5000	1
820	1,6	0,3	0,0000	1,6000	-1	0,8200	1,6000	1
822	1,6	0,5	0,0000	1,6000	-1	0,9400	1,6000	1
824	1,6	0,7	0,0000	1,6000	-1	1,0600	1,6000	1
826	1,6	0,9	0,0000	1,6000	-1	1,1800	1,6000	1
871	1,7	0,3	0,0000	1,7000	-1	0,8600	1,7000	1
879	1,7	1,1	0,0000	1,7000	-1	1,3400	1,7000	1
927	1,8	0,8	0,0000	1,8000	-1	1,2000	1,8000	1
928	1,8	0,9	0,0000	1,8000	-1	1,2600	1,8000	1
978	1,9	0,8	0,0000	1,9000	-1	1,2400	1,9000	1
989	1,9	1,9	0,0000	1,9000	-1	1,9000	1,9000	1
1021	2,0	0,0	0,0000	2,0000	-1	0,8000	2,0000	1
1027	2,0	0,6	0,0000	2,0000	-1	1,1600	2,0000	1
1071	2,0	5,0	1,4000	2,0000	-1	3,5000	2,0000	1
1080	2,1	0,8	0,0000	2,1000	-1	1,3200	2,1000	1
1126	2,2	0,3	0,0000	2,2000	-1	1,0600	2,2000	1
1177	2,3	0,3	0,0000	2,3000	-1	1,1000	2,3000	1
1182	2,3	0,8	0,0000	2,3000	-1	1,4000	2,3000	1
1188	2,3	1,4	0,0000	2,3000	-1	1,7600	2,3000	1
1232	2,4	0,7	0,0000	2,4000	-1	1,3800	2,4000	1
1240	2,4	1,5	0,0000	2,4000	-1	1,8600	2,4000	1
1279	2,5	0,3	0,0000	2,5000	-1	1,1800	2,5000	1
1292	2,5	1,6	0,0000	2,5000	-1	1,9600	2,5000	1
1294	2,5	1,8	0,0000	2,5000	-1	2,0100	2,5000	1
1296	2,5	2,0	0,0000	2,5000	-1	2,1500	2,5000	1
1297	2,5	2,1	0,0000	2,5000	-1	2,2200	2,5000	1
1304	2,5	2,8	0,0000	2,5000	-1	2,6500	2,5000	1
1308	2,5	3,2	0,0000	2,5000	-1	2,8500	2,5000	1
1312	2,5	3,6	0,0000	2,5000	-1	3,0500	2,5000	1
1317	2,5	4,1	0,3800	2,5000	-1	3,3000	2,5000	1
1320	2,5	4,4	0,9200	2,5000	-1	3,4500	2,5000	1
1326	2,5	5,0	2,0000	2,5000	-1	3,7500	2,5000	1
1333	2,6	0,6	0,0000	2,6000	-1	1,4000	2,6000	1
1384	2,7	0,6	0,0000	2,7000	-1	1,4400	2,7000	1
1414	2,7	3,6	3,1500	2,7000	1	7,5000	2,7000	-1
1433	2,8	0,4	0,0000	2,8000	-1	1,3600	2,8000	1
1486	2,9	0,6	0,0000	2,9000	-1	1,5200	2,9000	1
1488	2,9	0,8	0,0000	2,9000	-1	1,6400	2,9000	1
1490	2,9	1,0	0,0000	2,9000	-1	1,7600	2,9000	1
1532	3,0	0,1	0,0000	3,0000	-1	1,2600	3,0000	1
1533	3,0	0,2	0,0000	3,0000	-1	1,3200	3,0000	1

1538	3,0	0,7	0,0000	3,0000	-1	1,6200	3,0000	1
1585	3,1	0,3	0,0000	3,1000	-1	1,4200	3,1000	1
1688	3,3	0,4	0,0000	3,3000	-1	1,5600	3,3000	1
1694	3,3	1,0	0,0000	3,3000	-1	1,9200	3,3000	1
1735	3,4	0,0	0,0000	3,4000	-1	1,3600	3,4000	1
1736	3,4	0,1	0,0000	3,4000	-1	1,4200	3,4000	1
1739	3,4	0,4	0,0000	3,4000	-1	1,6000	3,4000	1
1742	3,4	0,7	0,0000	3,4000	-1	1,7800	3,4000	1
1744	3,4	0,9	0,0000	3,4000	-1	1,9000	3,4000	1
1787	3,5	0,1	0,0000	3,5000	-1	1,4600	3,5000	1
1792	3,5	0,6	0,0000	3,5000	-1	1,7600	3,5000	1
1794	3,5	0,8	0,0000	3,5000	-1	1,8800	3,5000	1
1841	3,6	0,4	0,0000	3,6000	-1	1,6800	3,6000	1
1843	3,6	0,6	0,0000	3,6000	-1	1,8000	3,6000	1
1892	3,7	0,4	0,0000	3,7000	-1	1,7200	3,7000	1
1895	3,7	0,7	0,0000	3,7000	-1	1,9000	3,7000	1
1940	3,8	0,1	0,0000	3,8000	-1	1,5800	3,8000	1
2554	5,0	0,3	0,0000	5,0000	-1	1,7100	5,0000	1
2566	5,0	1,5	0,0000	5,0000	-1	2,5500	5,0000	1
2579	5,0	2,8	1,0400	5,0000	-1	3,4600	5,0000	1
2580	5,0	2,9	1,2200	5,0000	-1	3,5300	5,0000	1
2581	5,0	3,0	1,4000	5,0000	-1	3,6000	5,0000	1
2585	5,0	3,4	2,1200	5,0000	-1	3,8800	5,0000	1

El algoritmo da como resultado que el modelo MLD para este caso está mal definido, se procesaron 2601 soluciones iniciando con los valores Min x(t)=0.0, Min u(t)=0.0,incrementos de inc=0.1, y valores finales Max x(t)=5.0 y Max u(t)=5.0; el tiempo de ejecución del algoritmo fue 7.9663 Horas.

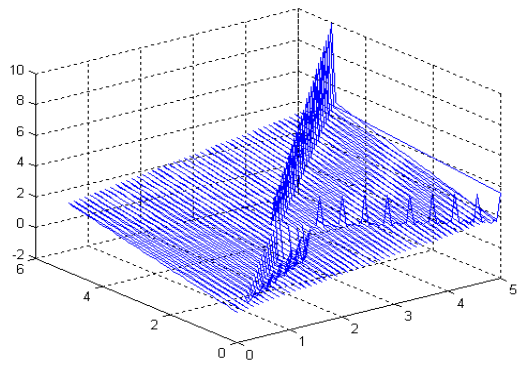

Figura 10. Caso2 $s_2 = Min - F' X_p$ x(t+1)

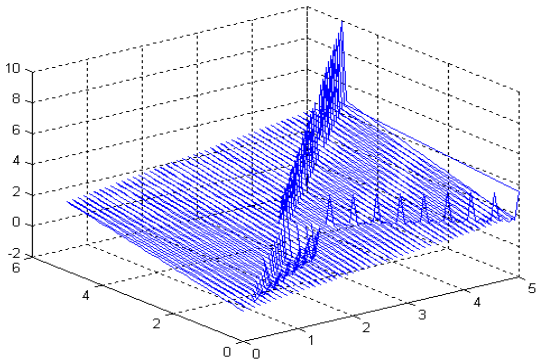

Figura 11 Caso2 $s_1 = MinF'X_p$ x(t+1)

En la Figura 10 y Figura 11 se puede observar una región con inexistencia de soluciones, lo cual representa un hueco en el espacio de soluciones.

Después de correr el algoritmo se aislaron los puntos sobre el límite superior, los cuales presentan no convergencia en el algoritmo solución.

CASO3. Modelo de un sistema sin condición de Unicidad (Soluciones Múltiples)

La formulación expresada en el capitulo 3. esta parametrizada permitiendo escoger un amplio rengo de valores, se escogen arbitrariamente teniendo en cuenta la visualización de los resultados , los cuales deben ser fácilmente interpretables. Se tomaron los siguientes valores:

a1=0.5, b1=0.5, a2=0.4, b2=0.6, a3=0.3, b3=0.7, p=2, a=5, b=5 , c= p/4, p1= p-c, q1= b*p*(p1-a)/(a*(p-a))

En los resultados obtenidos al ejecutar el proceso se puede observar:

En 110 puntos evaluados el valor del Exit-Flag del mínimo y el máximo son diferentes.

Analizados estos puntos (ver Tabla 7) se encuentra lo siguiente:

Tabla 7. Puntos del Caso 3 , factibles, con Exit Flag del Mínimo diferente al Ext Flag del Máximo

INDICE	X	U	X(k+1) MIN	Y MIN	FLAG MIN	X(k+1) MAX	Y MAX	FLAG MAX
24	0	2.3	0.00	0	0	1.15	0	1
28	0	2.7	0.00	0	0	1.35	0	1
29	0	2.8	0.00	0	0	1.4	0	1
30	0	2.9	0.00	0	0	1.45	0	1
31	0	3	0.00	0	0	1.5	0	1
36	0	3.5	0.25	0	0	1.75	0	1
37	0	3.6	0.40	0	0	1.8	0	1
41	0	4	1.00	0	0	2	0	1
42	0	4.1	1.15	0	0	2.05	0	1
43	0	4.2	1.30	0	0	2.1	0	1

El valor del Exit-flag del mínimo es 0 por que la función *linprog* de MATLAB alcanza el máximo de iteraciones sin converger, pero el máximo si es factible y es una solución valida.

Para los siguientes 65 puntos siguientes (ver Tabla 8) , se encuentra que incrementando y disminuyendo el valor de tolerancia utilizado para aceptar un valor como el entero “uno” o

"cero", se logra hacer factible la solución al minimizar y los valores de X(k+1) y Y, coinciden con los valores al maximizar la función.

Tabla 8. Puntos del Caso 2 en que se logra la factibilidad con ajuste al parámetro de tolerancia

INDICE	X	U	X(k+1) MIN	Y MIN	FLAG MIN	X(k+1) MAX	Y MAX	FLAG MAX
105	0.2	0.2	0.00	0.2	-1	0.2	0.2	1
256	0.5	0	0.00	0.5	-1	0.2	0.5	1
262	0.5	0.6	0.00	0.5	-1	0.56	0.5	1
312	0.6	0.5	0.00	0.6	-1	0.54	0.6	1
358	0.7	0	0.00	0.7	-1	0.28	0.7	1
361	0.7	0.3	0.00	0.7	-1	0.46	0.7	1
362	0.7	0.4	0.00	0.7	-1	0.52	0.7	1
411	0.8	0.2	0.00	0.8	-1	0.44	0.8	1
414	0.8	0.5	0.00	0.8	-1	0.62	0.8	1
419	0.8	1	0.00	0.8	-1	0.92	0.8	1
421	0.8	1.2	0.00	0.8	-1	1.04	0.8	1
578	1.1	1.6	0.00	1.1	-1	1.4	1.1	1
616	1.2	0.3	0.00	1.2	-1	0.66	1.2	1
617	1.2	0.4	0.00	1.2	-1	0.72	1.2	1
621	1.2	0.8	0.00	1.2	-1	0.96	1.2	1
622	1.2	0.9	0.00	1.2	-1	1.02	1.2	1
631	1.2	1.8	0.00	1.2	-1	1.56	1.2	1
670	1.3	0.6	0.00	1.3	-1	0.88	1.3	1
676	1.3	1.2	0.00	1.3	-1	1.24	1.3	1
678	1.3	1.4	0.00	1.3	-1	1.36	1.3	1
681	1.3	1.7	0.00	1.3	-1	1.54	1.3	1
724	1.4	0.9	0.00	1.4	-1	1.1	1.4	1
726	1.4	1.1	0.00	1.4	-1	1.22	1.4	1
739	1.4	2.4	0.00	1.4	-1	1.9	1.4	1
769	1.5	0.3	0.00	1.5	-1	0.78	1.5	1
770	1.5	0.4	0.00	1.5	-1	0.84	1.5	1
773	1.5	0.7	0.00	1.5	-1	1.02	1.5	1
776	1.5	1	0.00	1.5	-1	1.2	1.5	1
779	1.5	1.3	0.00	1.5	-1	1.38	1.5	1
823	1.6	0.6	0.00	1.6	-1	1	1.6	1
829	1.6	1.2	0.00	1.6	-1	1.36	1.	1
879	1.7	1.1	0.00	1.7	-1	1.34	1.7	1
881	1.7	1.3	0.00	1.7	-1	1.46	1.7	1
882	1.7	1.4	0.00	1.7	-1	1.52	1.7	1
883	1.7	1.5	0.00	1.7	-1	1.58	1.7	1
929	1.8	1	0.00	1.8	-1	1.32	1.8	1
931	1.8	1.2	0.00	1.8	-1	1.44	1.8	1
981	1.9	1.1	0.00	1.9	-1	1.42	1.9	1

983	1.9	1.3	0.00	1.9	-1	1.54	1.9	1
984	1.9	1.4	0.00	1.9	-1	1.6	1.9	1
985	1.9	1.5	0.00	1.9	-1	1.66	1.9	1
1029	2	0.8	0.00	2	-1	1.28	2	1
1033	2	1.2	0.00	2	-1	1.52	2	1
1049	2	2.8	0.00	2	-1	2.4	2	1
1071	2	5	1.40	2	-1	3.5	2	1
1072	2.1	0	0.00	2.1	-1	0.84	2.1	1
1233	2.4	0.8	0.00	2.4	-1	1.44	2.4	1
1276	2.5	0	0.00	2.5	-1	1	2.5	1
1277	2.5	0.1	0.00	2.5	-1	1.06	2.5	1
1334	2.6	0.7	0.00	2.6	-1	1.46	2.6	1
1353	2.6	2.6	0.00	2.6	-1	2.6	2.6	1
1379	2.7	0.1	0.00	2.7	-1	1.14	2.7	1
1490	2.9	1	0.00	2.9	-1	1.76	2.9	1
1509	2.9	2.9	0.00	2.9	-1	2.9	2.9	1
1589	3.1	0.7	0.00	3.1	-1	1.66	3.1	1
1688	3.3	0.4	0.00	3.3	-1	1.56	3.3	1
1717	3.3	3.3	0.00	3.3	-1	3.3	3.3	1
1787	3.5	0.1	0.00	3.5	-1	1.46	3.5	1
1792	3.5	0.6	0.00	3.5	-1	1.76	3.5	1
1793	3.5	0.7	0.00	3.5	-1	1.82	3.5	1
1821	3.5	3.5	0.50	3.5	-1	3.5	3.5	1
1995	3.9	0.5	0.00	3.9	-1	1.86	3.9	1
2094	4.1	0.2	0.00	4.1	-1	1.76	4.1	1
2450	4.8	0.1	0.00	4.8	-1	1.98	4.8	1
2579	5	2.8	1.04	5	-1	3.46	5	1

Para los datos correspondientes a los siguientes 35 puntos (ver

Tabla 9) , al minimizar se encuentra que no es factible, mientras que el máximo si lo es.

Tabla 9. Puntos del Caso 3 con mínimo no factible, y máximo factible.

INDICE	X	U	X(k+1) MIN	Y MIN	FLAG MIN	X(k+1) MAX	Y MAX	FLAG MAX
27	0	2.6	0.00	0	-1	1.3	0	1
33	0	3.2	0.00	0	-1	1.6	0	1
34	0	3.3	0.00	0	-1	1.65	0	1
35	0	3.4	0.10	0	-1	1.7	0	1
38	0	3.7	0.55	0	-1	1.85	0	1
40	0	3.9	0.85	0	-1	1.95	0	1
309	0.6	0.2	0.00	0.6	-1	0.36	0.6	1
311	0.6	0.4	0.00	0.6	-1	0.48	0.6	1
314	0.6	0.7	0.00	0.6	-1	0.66	0.6	1

315	0.6	0.8	0.00	0.6	-1	0.72	0.6	1
316	0.6	0.9	0.00	0.6	-1	0.78	0.6	1
412	0.8	0.3	0.00	0.8	-1	0.5	0.8	1
473	0.9	1.3	0.00	0.9	-1	1.14	0.9	1
524	1	1.3	0.00	1	-1	1.18	1	1
781	1.5	1.5	0.00	1.5	-1	1.5	1.5	1
831	1.6	1.4	0.00	1.6	-1	1.48	1.6	1
877	1.7	0.9	0.00	1.7	-1	1.22	1.7	1
1038	2	1.7	0.00	2	-1	1.82	2	1
1087	2.1	1.5	0.00	2.1	-1	1.74	2.1	1
1128	2.2	0.5	0.00	2.2	-1	1.18	2.2	1
1129	2.2	0.6	0.00	2.2	-1	1.24	2.2	1
1179	2.3	0.5	0.00	2.3	-1	1.22	2.3	1
1191	2.3	1.7	0.00	2.3	-1	1.94	2.3	1
1286	2.5	1	0.00	2.5	-1	1.6	2.5	1
1382	2.7	0.4	0.00	2.7	-1	1.32	2.7	1
1405	2.7	2.7	0.00	2.7	-1	2.7	2.7	1
1429	2.8	0	0.00	2.8	-1	1.12	2.8	1
1493	2.9	1.3	0.00	2.9	-1	1.94	2.9	1
1537	3	0.6	0.00	3	-1	1.56	3	1
1538	3	0.7	0.00	3	-1	1.62	3	1
1540	3	0.9	0.00	3	-1	1.74	3	1
1541	3	1	0.00	3	-1	1.8	3	1
1592	3.1	1	0.00	3.1	-1	1.84	3.1	1
1643	3.2	1	0.00	3.2	-1	1.88	3.2	1
1665	3.2	3.2	0.00	3.2	-1	3.2	3.2	1

El algoritmo da como resultado que el modelo MLD para este caso está mal definido, se procesaron 2601 soluciones, iniciando con los valores Min x(t)=0.0, Min u(t)=0.0, incrementos de inc=0.1, y valores finales Max x(t)=5.0 y Max u(t)=5.0. El tiempo de ejecución del algoritmo fue 8.1747 Horas.

Nótese que no es necesario hallar todas las soluciones, sin embargo esto es útil para el analista del sistema, quien puede guiarse hacia qué región esta mal definido y cual no.

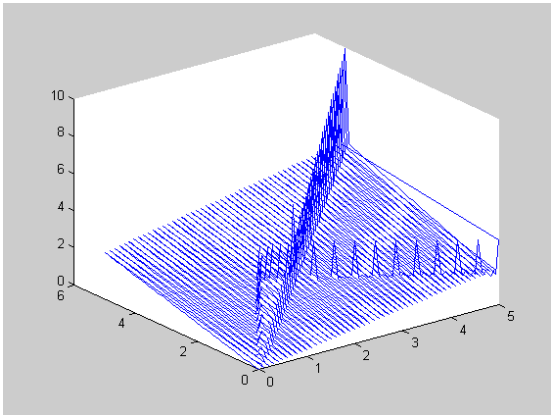

Figura 12. Caso3 $s_2 = Min - F'X_p$ x(t+1)

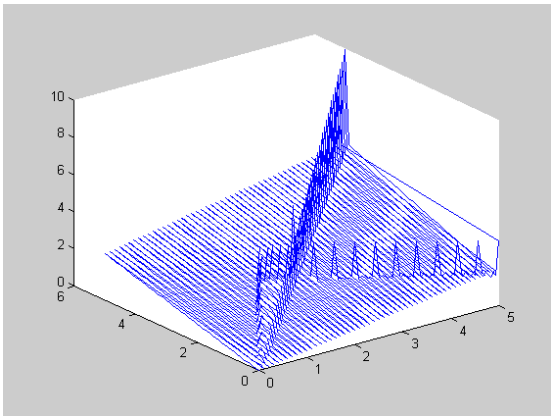

Figura 13 Caso3 $s_1 = MinF'X_p$ **x(t+1)**

De acuerdo al estudio de los 3 casos anteriores se ha observado en general:

La tolerancia general utilizada en el algoritmo es (100.000)*eps, donde eps representa el valor mas pequeño mayor que 0. Para una gran cantidad de puntos enunciados anteriormente en cada caso, los cuales con la tolerancia inicial no alcanzaban la igualdad del máximo y el mínimo de la función en X(k+1), fue posible alcanzarla a partir del ajuste en la tolerancia (aumentando hasta 10^{10} *eps ó disminuyendo hasta 1*eps) el valor inicial utilizado.

Se observa que en la medida que se disminuye el valor de la tolerancia los tiempos de ejecución aumentan considerablemente.

El algoritmo permite identificar si un modelo de un sistema híbrido en representación MLD esta bien o mal definido; el tiempo de ejecución es un poco alto cuando el modelo está bien definido, debido al número de soluciones que se deben calcular.

4.4 Complejidad Computacional

Los algoritmos tipo Branch and Bound se caracterizan por ser algoritmos de complejidad NP, debido a que básicamente se comporta como un algoritmo de enumeración, en el peor de los casos, y dado que las variables consideradas son binarias, el número máximo de iteraciones está dado por $O(2^{nb})$ donde *nb* es el número de variables binarias del problema. En términos prácticos esto significa que, aún para un número reducido de variables, el tiempo de cálculo crece significativamente. Esta evaluación de complejidad computacional hace referencia solamente al algoritmo de Branch and Bound, sin embargo el algoritmo corre como una enumeración en el espacio de variables reales, el cual aumenta el tiempo de ejecución de acuerdo a la siguiente proporción

$$\prod_{k=nr}\left(\frac{\max(n_k)-\min(n_k)}{\Delta_k}\right)$$

Es decir el número de variables reales (*nr*), y la resolución con la cual se enumera el espacio de enumeración de cada variable real (Δ_r).

5. ALGORITMO DE VALIDACIÓN CON COMPUTACIÓN DISTRIBUIDA

En este problema, definido como el desarrollo de un programa para validar la correcta definición de un Sistema Dinámico Hibrido, representado con modelos MLD (Mixed Logical and Dynamical), incluyendo la implementación de un algoritmo de validación, se encuentra que el costo de ejecución del algoritmo de validación, tal como se ha denominado en este trabajo, es muy alto, por lo que se pretende encontrar o desarrollar una versión mejorada, de modo que la solución al mismo sea mas eficiente y rápida que la inicialmente planteada. Mejoras en el algoritmo implican estudiar formas diferentes de abordar el problema, por ejemplo se ha pensado en reconstruir "las formas" descritas por las restricciones, sin embargo el tratamiento matemático elaborado queda por fuera del alcance de este trabajo.

La solución propuesta en esta sección consiste en utilizar el algoritmo ya desarrollado y puesto a punto, en una estructura computacional, que reduzca el tiempo de cómputo desde el punto de vista del usuario final.

Para lo anterior nos basamos en una solución con un algoritmo de validación con computación distribuida, por medio de la cual la solución al problema será realizada en múltiples procesadores interconectados por algún tipo de red. La idea es que N computadores, provean N veces la velocidad computacional de un solo computador, con la expectativa que el problema sería completado en la enésima parte del tiempo; por supuesto esta es la situación ideal, que normalmente no se cumple en la práctica. Las soluciones con Computación Distribuida se han hecho importantes en las ciencias computacionales debido

a la disminución en los costos de procesadores, avances de microelectrónica y en las redes de computadores.

En la sección 5.1 se describe la arquitectura del algoritmo implementado en Java, en la sección 5.2 se describe el algoritmo final de validación basado en computación distribuida, y en la sección 5.3 se presentan los resultados y el análisis de los resultados de la aplicación a los casos definidos en el capitulo 3. Finalmente en la sección 5.4 se comenta la complejidad computacional.

5.1 SOLUCION CON ALGORITMO DISTRIBUIDO IMPLEMENTADO EN JAVA

Para realizar soluciones con computación distribuida se necesitan las siguientes condiciones: Un cálculo debe ser dividido en slots (ranuras). Un slot es básicamente un símbolo asignado a un worker (trabajador) que contiene la información sobre qué parte del cómputo paralelo necesita ser realizada. Un central manager(encargado central) asignará estas ranuras a un número arbitrario de "trabajadores supuestos". Esto se ilustra en la Figura 14.

Las tareas actuales se dividen en los siguientes módulos:

Manager: Un encargado central, escrito enteramente en Java, asigna slots a los workers y acepta resultados. El starter provee al manager con la inicialización de datos. El retriever entrega los resultados al manager.

Worker (Trabajador): Un número arbitrario de workers pide al manager nuevos trabajos. Cuando un nuevo trabajo llega, se realizan los cálculos y los resultados son enviados al manager.

Starter (Inicializador): Después de que todos los trabajadores se hayan registrado con el manager, el starter proveerá los parámetros para iniciar la simulación y el inicio de los cálculos.

Retriever: Cuando finaliza el cálculo, el retriever puede grabar los resultados al manager.

Aborter: Algunos cálculos toman demasiado tiempo o no trabajan de la manera en que fue planeado. El aborter llama al manager para cancelar el cálculo actual

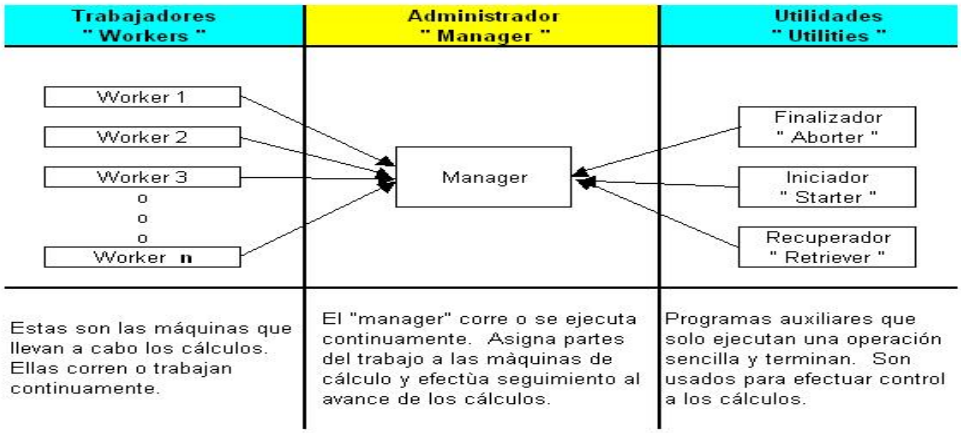

Figura 14. Distribución de Tareas en un Sistema Distribuido

5.2 ALGORITMO FINAL DE VALIDACIÓN BASADO EN COMPUTACIÓN DISTRIBUIDA

Los fuentes de los programas utilizados en la solución de computación distribuida se encuentran en el Anexo F. Básicamente, una parte de los programas fueron desarrollados en Java y otros en Matlab. Los desarrollados en Matlab son en esencia los mismos utilizados para la solución inicial planteada en el capítulo 4, y corresponden al proceso de búsqueda de soluciones del modelo.

La parte desarrollada en Java se encarga de la administración del procesamiento distribuido; estos programas realizan la división del trabajo entre el número de "workers" disponibles, la asignación de tareas con los parámetros específicos, el control de la ejecución de los cálculos, y el control de los resultados así como su consolidación.

En la sección anterior se describieron los distintos módulos que hacen parte de la solución distribuida y su función específica; a continuación se enumeran los programas Java asociados a cada módulo, el código fuente de los mismos se encuentra en el Anexo H:

- **Módulo "Manager" de interface Java (manager.java)**. Objeto remoto de Java para el manager. Como esta clase tiene su propia función main (), se puede ejecutar el manager sin matlab.
- **Módulo "IManager" de interface Java (Imanager.Java).** Interface Remota para el Objeto remoto de Java.
- **Módulo « Slot » de la Interface Java (Slot.java)**. Clase que contiene los datos para un slot.

- **Módulo "Results" de la Interface Java (Results.java)**. Clase que contiene los resultados de un cálculo.

5.3 RESULTADOS Y ANÁLISIS DE RESULTADOS

5.3.1 CASO 1. Modelo de un sistema bien definido

Al comparar los resultados de este proceso distribuido con el proceso no distribuido, podemos observar que los valores obtenidos para X(k+1) MIN, Y MIN, FLAG MIN, X(k+1) MAX, Y MAX, FLAG MAX, son exactamente iguales en cada uno de los pares [X, U].

La columna "Tiempo Dist' representa el tiempo en segundos utilizado por el WORKER para procesar el par [X, U]. El tiempo de ejecución de esta solución fue 1.2688 horas, para la cual se utilizaron 15 máquinas, en cada una de las cuales se colocó un solo WORKER. Esto representa un ahorro de 6.6975 horas, respecto a la solución no distribuida (8.7912), o sea el 85.57% del tiempo. Al sumar el tiempo empleado por cada una de las máquinas en procesar cada par [X, U], tenemos un valor de 11.423 horas, el cual es mayor. Este incremento con respecto al tiempo de la solución no distribuida es debido a que dentro de las 15 máquinas utilizamos varias de menor características que la utilizada para ejecutar la solución no distribuida, además del tiempo utilizado para la comunicación de los trabajadores y el administrador.

Se puede apreciar que a pesar de utilizar maquinas de diferentes características, los valores de las soluciones no se ven afectados, sino que se afecta es el tiempo total de ejecución de la solución distribuida.

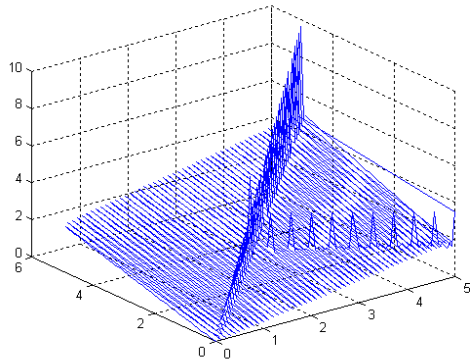

Figura 15 X(k+1).

La Figura 15 muestra la gráfica de los resultados X(k+1) obtenidos (azul).

5.3.2 CASO 2. Modelo de un sistema sin condición de existencia (Ausencia de solución)

Al comparar los resultados de este proceso distribuido con el proceso no distribuido, podemos observar que los valores obtenidos para X(k+1) MIN, Y MIN, FLAG MIN, X(k+1) MAX, Y MAX, FLAG MAX, son exactamente iguales en cada uno de los pares [X, U].

La columna “Tiempo dist.” representa el tiempo en segundos utilizado por el trabajador para procesar el par [X, U]. El tiempo de ejecución de esta solución fue 1.0481 horas, para la cual se utilizaron 15 máquinas, en cada una de las cuales se colocó un solo trabajador.

Esto representa un ahorro de 6.9182 horas respecto a la solución no distribuida (7.9663), que es el 86.84% del tiempo. Al sumar el tiempo empleado por cada una de las máquinas en procesar cada par [X, U] tenemos un valor de 10.2754 horas, el cual es mayor, este incremento con respecto al tiempo de la solución no distribuida es debido a que dentro de las 15 máquinas utilizamos varias de menor características que la utilizada para ejecutar la solución no distribuida, además del tiempo utilizado para la comunicación de los trabajadores y el administrador.

Se puede apreciar que a pesar de utilizar máquinas de diferentes características, los valores de las soluciones no se ven afectados, sino que se afecta es el tiempo total de ejecución de la solución distribuida.

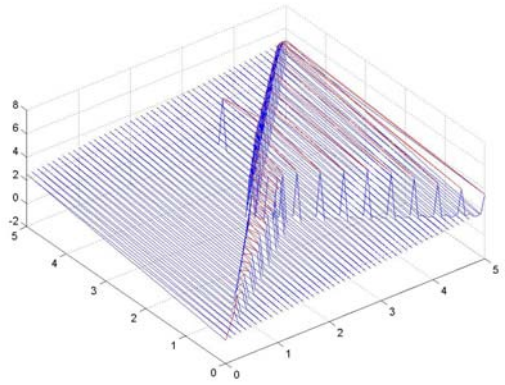

Figura 16 X(k+1).

La Figura 16 muestra la grafica de los resultados X(k+1) obtenidos (azul), además de los puntos que no son factibles en el modelo (rojo).

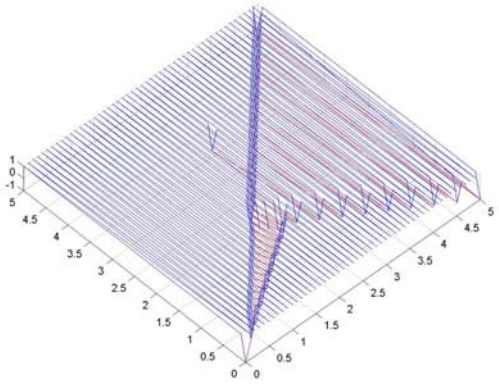

Figura 17 EXITFLAG del Caso 2

La Figura 17 muestra los exitflag igual a “1” (Azul) y los exitflag igual a “-1” (Rojo), que son los puntos no válidos del modelo.

5.3.3 CASO 3. Modelo de un sistema sin condición de Unicidad (Soluciones Múltiples)

Al comparar los resultados de este proceso distribuido con el proceso no distribuido, se puede observar que los valores obtenidos para X(k+1) MIN, Y MIN, FLAG MIN, X(k+1) MAX, Y MAX, FLAG MAX, son exactamente iguales en cada uno de los pares [X, U].

La columna “Tiempo dist.” representa el tiempo en segundos utilizado por el trabajador para procesar el par [X, U]. El tiempo de ejecución de esta solución fue 1.1604 horas, para la cual se utilizaron 15 máquinas, en cada una de las cuales se colocó un solo trabajador. Esto representa un ahorro de 7.0143 horas, respecto a la solución no distribuida (8.1747), que es el 85.80% del tiempo. Al sumar el tiempo empleado por cada una de las máquinas en procesar cada par [X, U] tenemos un valor de 9.60 horas, el cual es mayor, este

incremento con respecto al tiempo de la solución no distribuida, es debido a que dentro de las 15 máquinas utilizamos varias de menor características que la utilizada para ejecutar la solución no distribuida, además del tiempo utilizado para la comunicación de los trabajadores y el administrador.

Se puede apreciar que a pesar de utilizar máquinas de diferentes características, los valores de las soluciones no se ven afectados, sino que se afecta es el tiempo total de ejecución de la solución distribuida.

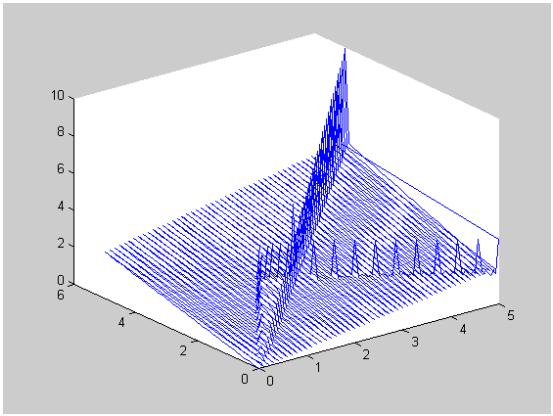

Figura 18 X(k+1)

La Figura 18 muestra la grafica de los resultados X(k+1) obtenidos (azul), además de los puntos que no son factibles en el modelo.

En la siguiente tabla se resumen los diferentes tiempos de ejecución para cada uno de los casos, comparando el algoritmo inicial con el distribuido, en donde se puede apreciar una optimización cercana al 90%.

	Duración Algoritmo Inicial (Horas)	Duración Algoritmo Distribuido (Horas)	Diferencia (Horas)	Variación Tiempos de Ejecución
Caso 1. Modelo de un sistema bien definido.	8.7912	1.2688	7.5224	**85.57%**
Caso 2. Modelo de un sistema sin condición de existencia.	7.9663	1.0481	6.9182	**86.84%**
Caso 3. Modelo de un sistema sin condición de unicidad.	8.1747	1.1604	7.0143	**85.80%**

Tabla 10 Análisis comparativo de las dos soluciones

5.4 COMPLEJIDAD COMPUTACIONAL DEL ALGORITMO DISTRIBUIDO

Tal como se estableció en la introducción de este capitulo, la solución por computación distribuida utiliza el mismo algoritmo que el implementado en el capitulo 4. La complejidad computacional sigue siendo NP, en el peor de los casos es $O(2^{nb})$, donde *nb* es el numero de variables binarias del problema. Si se tienen *m* computadores la complejidad se reduce a $O(2^{nb}/m)$ que aun sigue teniendo un comportamiento exponencial.

6. CONCLUSIONES Y TRABAJOS FUTUROS

6.1 CONCLUSIONES

En el presente trabajo de Investigación se ha desarrollado un algoritmo de validación para un modelo MLD, el cual se ha validado con tres casos de estudio, si tales modelos son factibles o no.

- Caso 1. Modelo de un sistema Bien Definido.
- Caso 2. Modelo de un sistema sin condición de Existencia (Con Ausencia de Puntos - Modelo Mal Definido) y
- Caso 3. Modelo de un sistema sin condición de unicidad (Con Puntos Doblemente Definidos - Modelo Mal Definido).

Para lograr lo anterior, la solución se basa en el algoritmo de Branch and Bound para detectar si en cada pareja de puntos [X,U] del modelo, (donde X representa el estado del Sistema y U representa la Entrada del sistema.) generando subproblemas (hojas del árbol), hasta que al final logramos definir si el punto es factible o infactible. A partir de allí se toma una grilla de valores para [x,u] respecto del caso de estudio y la solución que muestra dicho algoritmo es, si el modelo es factible o infactible.

Teniendo presente lo anterior se concluye si el modelo está bien definido (Well Posed). Esta idea se toma del trabajo realizado por Bemporad y Morari [8], pero como se puede observar en el trabajo, dicho algoritmo fue mejorado para detectar los errores que se plantearon en el modelo de validación para el caso 3. (Sistema con soluciones múltiples).

Surge el planteamiento de mejorar dicha solución y lo que se propone es que se haga el cálculo para el máximo y el mínimo de X(k+1), Y, δ y Z para los valores de X y U, que se encuentran entre los limites de estas variables. Con el cual al no ser factible el máximo y el mínimo o factibles el máximo y el mínimo pero los valores de X(k+1) diferentes el MLD estaría mal definido.

De esta forma, se ejecuta esta solución para los tres casos de prueba y se logra verificar que permite identificar cuando un modelo MLD, está bien o mal definido, en algunos casos se encuentra que el valor máximo es factible, pero el mínimo no, y al examinar el algoritmo se comprueba que esto se debe a la precisión utilizada para aceptar un número como un entero y al modificarla se obtiene una solución factible para el mínimo y los valores de X(k+1), Y, δ y Z coinciden con los valores del máximo.

Finalmente se logra una mejora significativa en los tiempos de ejecución del algoritmo a través de la programación distribuida, se cambia la plataforma de procesamiento y se trabaja con esta filosofía, obteniendo los mismos resultados con una significativa disminución del tiempo de ejecución de dicho algoritmo, en cada uno de los casos.

6.2 TRABAJOS FUTUROS

Aunque se ha conseguido una mejora considerable del rendimiento del algoritmo distribuido de validación, en comparación con la versión convencional, que era uno de los

objetivos del proyecto todavía quedan varias posibilidades de mejora y se abren nuevas líneas de investigación. El principal punto de mejora de este algoritmo, radicaría en disminuir la complejidad computacional, relacionado con la enumeración de las soluciones en el Branch and Bound, y en la "grilla " de soluciones de prueba, lo que abre explícitamente las siguientes dos formulaciones de trabajos futuros:

i) Optimización del algoritmo de validación con una Búsqueda mas inteligente, de tal forma que recorra primero los nodos del árbol mas prometedores con la esperanza que conduzca a mejores soluciones, lo que proporcionará mejores cotas y así podar el árbol de búsqueda mas rápidamente, por tanto no haya que realizar el recorrido por todo el árbol de Restricciones.

ii) Encontrar el Polígono (Polytope) que enmarca la región del modelo de Estudio y de esta forma, encontrar la solución mas óptima para la validación del Modelo, evitando la grilla de soluciones de prueba.

7. BIBLIOGRAFÍA

[1] Oguniev Ke, Batbunde A Process Dynamic, Modeling and Control: Oxford University Press, 1994.

[2] C. G, Cassandras and S. Lafortune, Introduction to Discrete Event Systems. Kluwe Academic Publishes. Boston, USA 1999.

[3] P. Antsaklis, Ed., *Special Issue on Hybrid Systems: Theory and Applications*, ser. Proc. IEEE, July 2000, vol. 88.

[4] R. Alur, C. Belta, F. Ivan¡cic´, V. Kumar, M. Mintz, G. Pappas, H. Rubin, and J. Schug, "Hybrid modeling and simulation of biomolecular networks," in *Hybrid Systems: Comp. and Contr.*. ser. Lecture Notes in Comp. Sc., M. Di Benedetto and A. S. Vincentelli, Eds. New York: Springer-Verlag, 2001, vol. 2034, pp. 19–33.

[5] J. P. Hespanha, S. Bohacek, K. Obraczka, and J. Lee, "Hybrid modeling of TCP congestion control," in *Hybrid Systems: Comp. and Contr.*. ser. Lecture Notes in Comp. Sc., M. Di Benedetto and A. S. Vincentelli,Eds. New York: Springer-Verlag, 2001, vol. 2034, pp. 291–304.

[6] R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid Systems*. ser. Lecture Notes in Comp. Sc., R. Grossman, A. Nerode, A. Ravn, and H. Rischel, Eds. New York: Springer-Verlag, 1993, vol. 736, pp. 209–229.

[7] B. Silva, O. Stursberg, B. Krogh, and S. Engell, "An assessment of the current status of algorithmic approaches to the verification of hybrid systems," in *Proc. 40th IEEE Conf. Decision Contro,* Orlando, FL, Dec.2001, pp. 2867–2874.

[8] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, Mar. 1999.

[9] W. Heemels, B. D. Schutter, and A. Bemporad, "Equivalence of hybrid dynamical models," *Automatica*, vol. 37, no. 7, pp. 1085–1091, July 2001.

[10] A. Bemporad, F. Torrisi, and M. Morari, "Discrete-time hybrid modeling and verification of the batch evaporator process benchmark," *Eur.J. Control*, vol. 7, no. 4, pp. 382–399, 2001.

[11] D. Mignone, "Control and estimation of hybrid systems with mathematical optimization," Ph.D. dissertation, Automatic Control Labotatory, ETH, Zurich, Switzerland, 2002.

[12] H. Williams, *Model Building in Mathematical Programming*, 3rd ed. New York: Wiley, 1993.

[13] R. Raman and I. Grossmann, "Relation between MILP modeling and logical inference for chemical process synthesis," *Comput. Chem. Eng.*, vol. 15, no. 2, pp. 73–84, 1991.

[14] F. Torrisi, A. Bemporad, G. Bertini, P. Hertach, D. Jost, and D.Mignone. (2002) HYSDEL 2.0.5—User Manual. Automatic Control Laboratory, ETH, Zurich, Switzerland. [Online]. Available: http://control.ee.ethz.ch/

[15] A. Bemporad, P. Borodani, and M. Mannelli, "Hybrid control of an automotive robotized gearbox for reduction of consumptions and emissions," in *Hybrid Systems: Computation and Control*. ser. Lecture Notes in Comp. Sc., O. Maler and A. Pnueli, Eds. New York: Springer-Verlag, 2003, pp. 81–96.

[16] F. Borrelli, A. Bemporad, M. Fodor, and D. Hrovat, "A hybrid approach to traction control," in *Hybrid Systems: Computation and Control*. ser. Lecture Notes in Comp. Sci., M. Di Benedetto and A. S. Vincentelli, Eds. New York: Springer-Verlag, 2001, vol. 2034, pp. 162–174.

[17] A. Bemporad, N. Giorgetti, I. Kolmanovsky, and D. Hrovat, "A hybrid systems approach to modeling and optimal control of DISC engines," presented at the 41st IEEE Conf. Decision Control, Las Vegas, NV, Dec.2002.

[18] G. Ferrari-Trecate, E. Gallestey, P. Letizia, M. Spedicato, M. Morari, and M. Antoine, "Modeling and control of co-generation power plants: A hybrid system approach," in *Hybrid Systems: Computation and Control*. ser. Lecture Notes in Comp. Sci., C. J. Tomlin and M. R. Greenstreet, Eds. New York: Springer-Verlag, 2002, vol. 2289, pp. 209–224.

[19] C. Güzelis and I. C. Göknar. A canonical representation for piecewise-affine maps and its applications to circuit análisis. IEEE Transactions on Circuits and Systems, 38(11):1342-1354, Nov.1991.

[20] Heemels, W.P.M.H, Shumacher, J.M., & Weiland, S. (2000). Linear complementary Systems. SIAM Journal of Applied Mathematics, 60(4), 12344-1269.

[21] Van der Shaft, A. J., & Schumacher, J.M (1998)- Complementary modelling of irbid systems. IEEE Transactions on Automatic Control, 43, 483-490.

[22] Michael J Gagen (1985)- Quantum measurement theory and the quantum Zeno effect. PhD Thesis. Chapter 2.

[23] B.H.Krogh (1995), Control Síntesis for DES Using Petri Nets. In: S. Genti (ed). Supervisory Controil of discrete events systems. Summer School of automatic Control of Grenoble, 1995.

[24] Lygeros, Johansson, Simic, Zhang, Sastry. (2003) Dynamical Properties of hybrid Automata, IEEE Trans on Automatic Control, Vol 48, No 1, January 2003.

[25] Imura and Arjan van der Schaft. (2000). Characterization of well-posedness of piecewise-linear systems, IEEE Trans on Automatic Control, Vol 45, No 9, Sept. 2000].

[26] A. F. Filippov. (1988). Differential Equations With Discontinuos Righthand Sides. Dordrecht, The Netherlands: Kluwer.

[27] Papadimitriou, C. and Steiglitz, K. : Combinatorial Optimization : Algorithms and Complexity. Prentice Hall 1982.

[28] Schrage, L. : Linear, Integer and Quadratic Programming with LINDO. Scientific Press 1986.

[29] Williams, H. : Model Solving in Mathematical Programming. Wiley 1993.

Páginas Web de consulta

- Dymola, Dynasim AB. http://www.dynasim.se/

- Ptolemy ii, Department of EECS, UC Berkeley. http://ptolemy.eecs.berkeley.edu/ptolemyII/

- Charon, Deparment of Computer and Information Science, University of Pennsylvania. http://www.cis.upenn.edu/mobies/charon/

- An overview of hybrid simulation phenomena and their support by simulation packages. P. Mosterman. http://robotics.eecs.berkeley.edu/~sastry/ee291e/mosterman.pdf

- UPPAAL. Herramienta de simulación. http://www.uppaal.com/

- Dymola, Dynasim AB. http://www.dynasim.se/

- Ptolemy ii, Department of EECS, UC Berkeley. http://ptolemy.eecs.berkeley.edu/ptolemyII/

- Charon, Deparment of Computer and Information Science, University of Pennsylvania. http://www.cis.upenn.edu/mobies/charon/

- An overview of hybrid simulation phenomena and their support by simulation packages. P. Mosterman. http://robotics.eecs.berkeley.edu/~sastry/ee291e/mosterman.pdf

- The Hybrid Systems Group. http://control.ee.ethz.ch/~hybrid/

- The Hybrid Systems Group. http://control.ee.ethz.ch/~hybrid/

ANEXOS

Anexo A Formato Hysdel para la representación de sistemas MLD.

Los casos de evaluación presentados inicialmente como un modelo PWA, fueron luego representados de forma que pudiese aplicarse sobre ellos la herramienta HYSDEL, y se obtuvo el modelo equivalente en MLD. Hysdel genera un sistema de ecuaciones MLD equivalente al sistema tradicional conocido. A continuación se detalla la representación utilizada o generada por HYSDEL. La notación utilizada es la propuesta en **[8**], donde un sistema MLD es descrito por las siguientes relaciones:

$$\begin{bmatrix} x_r(t+1) \\ x_b(t+1) \end{bmatrix} = \begin{bmatrix} A_{rr} & A_{rb} \\ A_{br} & A_{bb} \end{bmatrix} \begin{bmatrix} x_r(t) \\ x_b(t) \end{bmatrix} + \begin{bmatrix} B_{1rr} & B_{1rb} \\ B_{1br} & B_{1bb} \end{bmatrix} \begin{bmatrix} u_r(t) \\ u_b(t) \end{bmatrix} + \begin{bmatrix} B_{2rb} \\ B_{2bb} \end{bmatrix} d(t) + \begin{bmatrix} B_{3rr} \\ B_{3br} \end{bmatrix} z(t), \qquad \textbf{(a.1)}$$

$$\begin{bmatrix} y_r(t) \\ y_b(t) \end{bmatrix} = \begin{bmatrix} C_{rr} & C_{rb} \\ C_{br} & C_{bb} \end{bmatrix} \begin{bmatrix} x_r(t) \\ x_b(t) \end{bmatrix} + \begin{bmatrix} D_{1rr} & D_{1rb} \\ D_{1br} & D_{1bb} \end{bmatrix} \begin{bmatrix} u_r(t) \\ u_b(t) \end{bmatrix} + \begin{bmatrix} D_{2rb} \\ D_{2bb} \end{bmatrix} d(t) + \begin{bmatrix} D_{3rr} \\ D_{3br} \end{bmatrix} z(t), \qquad \textbf{(a.2)}$$

$$E_2 d(t) + E_3 z(t) \le E_1 \begin{bmatrix} u_r(t) \\ u_b(t) \end{bmatrix} + E_2 \begin{bmatrix} x_r(t) \\ x_b(t) \end{bmatrix} + E_5 \qquad \textbf{(a.3)}$$

Donde $x_r \in \Re^{nx_r}$, $x_b \in \{0,1\}^{nx_b}$, $u_r \in \Re^{nu_r}$, $u_b \in \{0,1\}^{nu_b}$, $y_r \in \Re^{ny_r}$, $y_b \in \{0,1\}^{ny_b}$, $z \in \Re^{n_z}$, y $\delta \in \{0,1\}^{n_\delta}$ todas las matrices E_i tienen n columnas.

Una forma mas general del Sistema MLD es:

$$\begin{bmatrix} x_r(t+1) \\ x_b(t+1) \end{bmatrix} = \begin{bmatrix} A_{rr} & A_{rb} \\ A_{br} & A_{bb} \end{bmatrix} \begin{bmatrix} x_r(t) \\ x_b(t) \end{bmatrix} + \begin{bmatrix} B_{1rr} & B_{1rb} \\ B_{1br} & B_{1bb} \end{bmatrix} \begin{bmatrix} u_r(t) \\ u_b(t) \end{bmatrix} + \begin{bmatrix} B_{2rb} \\ B_{2bb} \end{bmatrix} d(t) + \begin{bmatrix} B_{3rr} \\ B_{3br} \end{bmatrix} z(t) + \begin{bmatrix} B_{5r} \\ B_{5b} \end{bmatrix}, \quad \text{(a.4)}$$

$$\begin{bmatrix} y_r(t) \\ y_b(t) \end{bmatrix} = \begin{bmatrix} C_{rr} & C_{rb} \\ C_{br} & C_{bb} \end{bmatrix} \begin{bmatrix} x_r(t) \\ x_b(t) \end{bmatrix} + \begin{bmatrix} D_{1rr} & D_{1rb} \\ D_{1br} & D_{1bb} \end{bmatrix} \begin{bmatrix} u_r(t) \\ u_b(t) \end{bmatrix} + \begin{bmatrix} D_{2rb} \\ D_{2bb} \end{bmatrix} d(t) + \begin{bmatrix} D_{3rr} \\ D_{3br} \end{bmatrix} z(t) + \begin{bmatrix} D_{5r} \\ D_{5b} \end{bmatrix}, \quad \text{(a.5)}$$

$$E_2 d(t) + E_3 z(t) \le E_1 \begin{bmatrix} u_r(t) \\ u_b(t) \end{bmatrix} + E_2 \begin{bmatrix} x_r(t) \\ x_b(t) \end{bmatrix} + E_5 \quad \text{(a.6)}$$

Donde $x_\bullet, u_\bullet, y_\bullet, z$ y δ son como se definieron anteriormente.

Hay que notar que las matrices B_{5c}, B_{5d}, D_{5c} y D_{5d} no introducen ninguna modificación al modelo MLD pero son útiles para mantener el tamaño de los modelos contenidos.

Anexo B. Descripción de la Estructura MLD generada por Hysdel

A continuación se detalla la estructura utilizada por Hysdel para mostrar la salida o resultado de su ejecución, que no es mas que el sistema MLD.

La Estructura **S** contiene los siguientes campos:

- Arr = A_{rr}, Arb= A_{rb}, Abr = A_{br}, Abb = A_{bb}, A = [Arr, Arb; Abr, Abb].
- B1rr = B_{1rr}, B1rb = B_{1rb}, B1br = B_{1br}, B1bb = B_{1bb}, B1 = [B1rr, B1rb; B1br, B1bb].

- B2rb = B_{2rb}, B2bb = B_{2bb}, B2 = [B2rb; B2bb].
- B3rr = B_{3rr}, B3br = B_{3br}, B3 = [B3rr; B3br].
- B5r = B_{5r}, B5b = B_{5b}, B5 = [B5r; B5b].
- Crr = C_{rr}, Crb = C_{rb}, Cbr = C_{br}, Cbb = C_{bb}, C = [Crr, Crb; Cbr, Cbb]
- D1rr = D_{1rr}, D1rb = D_{1rb}, D1br = D_{1br}, D1bb = D_{1bb}, D1 = [D1rr, D1rb; D1br, D1bb].
- D2rb = D_{2rb}, D2bb = D_{2bb}, D2 = [D2rb; D2bb].
- D3rr = D_{3rr}, D3br = D_{3br}, D3 = [D3rr; D3br].
- D5r = D_{5r}, D5b = D_{5b}, D5 = [D5r; D5b].
- E1 = E_1, E2 = E_2, E3 = E_3, E4 = E_4, E5 = E_5.
- nxr = n_{xr}, nxb = n_{xb}, nx = nxr + nxb, nur = n_{ur}, nub = n_{ub}, nu = nur + nub, nyr = n_{yr}, nyb = n_{yb}, ny = nyr + nyb, nd = n_{δ}, nz = n_z, ne = n_e.
- ul, uu son los límites superior e inferior de u, lo mismo que xl, xu, yl, yu, dl, du, y zl, zu.
- **MLDisvalid**: toma el valor 1 si el modelo ha pasado un chequeo de validación (tamaño de las matrices, entradas nulas,...). Las funciones de usuario pueden confiar esta información.
- **name string:** contiene el nombre del sistema como fué especificado en la declaración de HYSDEL.
- **MLDstructver:** entero que contiene la versión, necesitado para versiones futuras. Todas las versiones, deberían ser compatibles con versiones anteriores. La función

de usuario debería chequear si la nueva versión es suficiente. La versión presentada en este documento es 2. Hysdel versión 1.3.0 soporta la versión 2.

- **MLDsymtable:** toma el valor 1 si la estructura incluye símbolos de la tabla de Información.
- **MLDrowinfo:** toma el valor 1 si la estructura incluye información de depuración para las restricciones generadas.
- **symtable**: arreglo de celdas llamada que contiene los símbolos:
  - **name string**: contiene el nombre del símbolo.
  - **kind {‘**x’|’u’|’y’|’z’|’d’|’p**’}**: nombre del arreglo que contiene la variable; es importante notar que están en minúsculas. La letra ‘p’ significa parámetro.
  - **type {‘r’|’b’}**: tipo de la variable; se escribe en minúscula.
  - **index**: posición de la variable en el arreglo **kind** $_{\textbf{type}}$. En el caso de parámetros (**kind** = ‘p’), el contenido de esta variable no tiene relevancia.
  - **line_of_declaration**: línea donde la variable fue declarada en el código Hysdel. Este índice es –1 para los parámetros predefinidos.
  - **defined**: grupo de restricciones que contienen la definición de la variable (ver mas adelante el campo **rowinfo**).
  - **computable_order**: prioridad del cálculo. Esta información puede ser usada como prioridad de bifurcación si el solucionador entero-mezclado soporta esta opción.
  - **min**: mínimo valor de la variable.
  - **min_computed**: toma el valor 1 si Hysdel calculó el mínimo.
  - **max**: máximo valor de la variable.

- **max_computed**: toma el valor 1 si Hysdel calculó el máximo.
- **Value**: valor numérico del parámetro (este campo está presente solo si kind='p').

- **rowinfo**: estructura que contiene la información de las matrices.
  - **state_upd**: arreglo de celdas que contiene la información de las filas en el estado **update** de las matrices (a.4). El arreglo de celdas tiene el mismo número de ítems que el número de filas de las matrices; el i-ésimo elemento del arreglo provee información sobre la i-ésima restricción. A su vez **state_upd** está conformada por:
    - **section**: nombre de la sección donde el estado **update** fue declarado.
    - **item_type**: sección actual a la cual pertenece el item.
    - **defines**: nombre de la variable que es definida.
    - **depends**: arreglo de celdas de nombres de variable de los que depende la definición.
    - **group, subgroup, subindex**: identificación única de la declaración fuente. Cada instrucción en Hysdel obtiene un número secuencial; esta instrucción es almacenada en **group**. Una instrucción puede corresponder a mas de un elemento básico de Hysdel (por ejemplo, en el caso de introducción automática de variables), y cada uno de los elementos que la constituyen obtiene un número secuencial. Recapitulando, todas las instrucciones que vienen de una instrucción Hysdel comparten el mismo índice **group**, y todas las restricciones que vienen de un elemento básico comparten el mismo índice

subgroup, y dentro del mismo group, subgroup cada restricción obtiene un subíndice (**subindex**) secuencial.

- **source**: código recortado de la declaración Hysdel original.
- **sourceline**: línea del código fuente.
- **human**: expresión leíble para humanos.

- **output**: arreglos de celdas que contienen la información de las celdas en las matrices de salida (a.5). Al igual que antes, el i-ésimo elemento del arreglo provee información sobre la i-ésima restricción, y cada item tiene los mismos campos como en el punto anterior.
- **ineq**: arreglo de celdas que contiene la información de las filas de las restricciones de desigualdades (a.6). Al igual que antes, el i-ésimo elemento del arreglo provee información sobre la i-ésima restricción. Cada item tiene los mismos campos como como los descritos en el punto anterior, mas los siguientes campos :
  - **aff_min**: valor de **m** usado para las transformaciones (3.12) y (3.10) mostradas en [33].
  - **aff_min_computed**: toma el valor 1 si **m** fue calculada, y 0 si es suministrada.
  - **aff_max**: valor de **M** usado para las transformaciones (3.12) y (3.10) mostradas en [33].
  - **aff_max_computed**: toma el valor 1 si **M** fue calculada, y 0 si es suministrada.
  - **aff_cps**: valor de tolerancia $\in$ usado para las transformaciones (3.10) mostrada en [33].

- **aff_cps_computed**: toma el valor 1 si fue utilizado un valor por defecto para $\in$, y 0 si fue provisto o suministrado.

Anexo C. Codificación con la herramienta Hysdel del Caso 1 “Modelo de un sistema bien definido”

```
SYSTEM caso1 {
   INTERFACE {
      PARAMETER {
         REAL  a=5,b=5,  p=2,  a1=0.5,  a2=0.4,  a3=0.3,  b1=0.5,  b2=0.6,
b3=0.7; }
      STATE {
           REAL x [0, a];}
      INPUT {
         REAL u [0, b]; }
      OUTPUT {
         REAL y; }
   } /* end interface */
   IMPLEMENTATION {
      AUX {
         BOOL R1, R2, R3, R11, R12, R13, R14, R15, R21, R22, R23, R24,
R25, R26,
              R31, R32, R33, R34;
           REAL Z1,Z2,Z3;
         }
     AD {
          R11= 0 <= x ;
          R12= x <= a ;
          R13=  0 <= u ;
          R14= u <= b ;
          R15= u >= b*x/a;
```

```
        R21= u<=(b*x)/a;
          R22= 0<=x;
          R23= x<= p;
        R24= u<=b*p*(x-a)/(a*(p-a));
        R25= p<=x;
        R26= x<=a;
        R31= u<=(b*x)/a;
          R32= p<=x;
          R33= x<=a;
        R34= u>=b*p*(x-a)/(a*(p-a));              }
    LOGIC {
            R1= R11 & R12 & R13 & R14 & R15 ;
              R2= (R21 & R22 & R23 ) | ( R24 & R25 & R26 );
            R3= (R31 & R32 & R33 & R34);
        }
    DA {
              Z1 = {IF R1 THEN a1*x+b1*u ELSE 0};
              Z2 = {IF R2 THEN a2*x+b2*u ELSE 0};
              Z3 = {IF R3 THEN a3*x+b3*u ELSE 0};
         }
    CONTINUOUS {
          x =  Z1+Z2+Z3;}
    OUTPUT {
        y = x;}
  } /* end implementation */
} /* end system */
```

Anexo D. Codificación con la herramienta Hysdel del Caso 2 "Modelo de un sistema sin condición de existencia" (ausencia de solución)

```
SYSTEM caso2 {
   INTERFACE {
      PARAMETER {
         REAL  a=5,b=5,  p=2,  a1=0.5,  a2=0.4,  a3=0.3,  b1=0.5,  b2=0.6,
b3=0.7, c= p/4,
              p1=p+c, q1= b*p*(p1-a)/(a*(p-a));}
      STATE {
           REAL x [0, a];}
      INPUT {
         REAL u [0, b]; }
      OUTPUT {
         REAL y; }
   } /* end interface */
   IMPLEMENTATION {
     AUX {
         BOOL  R1,  R2,  R3,  R11,  R12,  R13,  R14,  R15,  R21,  R22,  R23,  R24,
R25, R26,
              R31, R32, R33, R34;
           REAL Z1,Z2,Z3;
         }
     AD {

          R11= 0 <= x ;
          R12= x <= a ;
```

```
      R13=  0 <= u ;
      R14= u <= b ;
      R15= u >= b*x/a;
      R21= u<=(q1*x)/p1;
        R22= 0<=x;
        R23= x<= p1;
      R24= u<=q1*(x-a)/(p1-a);
      R25= p1<=x;
      R26= x<=a;
      R31= u<=(b*x)/a;
        R32= p<=x;
        R33= x<=a;
      R34= u>=b*p*(x-a)/(a*(p-a));
     }
  LOGIC {
         R1= R11 & R12 & R13 & R14 & R15 ;
           R2= (R21 & R22 & R23 ) | ( R24 & R25 & R26 );
         R3= (R31 & R32 & R33 & R34);
     }
  DA {
          Z1 = {IF R1 THEN a1*x+b1*u ELSE 0};
          Z2 = {IF R2 THEN a2*x+b2*u ELSE 0};
          Z3 = {IF R3 THEN a3*x+b3*u ELSE 0};
      }
  CONTINUOUS {
       x =  Z1+Z2+Z3;}
  OUTPUT {
     y = x;}
} /* end implementation */
```

```
} /* end system */
```

Anexo E. Codificación con la herramienta Hysdel del Caso 3 “Modelo de un sistema sin condición de unicidad” (soluciones múltiples)

```
SYSTEM caso3 {
   INTERFACE {
      PARAMETER {
         REAL  a=5,b=5,  p=2,  a1=0.5,  a2=0.4,  a3=0.3,  b1=0.5,  b2=0.6,
b3=0.7, c= p/4,  p1= p-c,
              q1= b*p*(p1-a)/(a*(p-a));}
      STATE {
           REAL x [0, a];}
      INPUT {
         REAL u [0, b]; }
      OUTPUT {
         REAL y; }
   } /* end interface */
   IMPLEMENTATION {
      AUX {
         BOOL  R1,  R2,  R3,  R11,  R12,  R13,  R14,  R15,  R21,  R22,  R23,  R24,
R25, R26,
              R31, R32, R33, R34,R_temp;
           REAL Z1,Z2,Z3;
         }
     AD {
```

```
      R11= 0 <= x ;
      R12= x <= a ;
      R13=  0 <= u ;
      R14= u <= b ;
      R15= u >= b*x/a;
      R21= u<=(q1*x)/p1;
        R22= 0<=x;
        R23= x<= p1;
      R24= u<=b*p*(x-a)/(a*(p-a));
      R25= p1<=x;
      R26= x<=a;
      R31= u<=(b*x)/a;
        R32= p<=x;
        R33= x<=a;
      R34= u>=b*p*(x-a)/(a*(p-a));
   }
LOGIC {
         R1= (R11 & R12 & R13 & R14 & R15) & R_temp ;
           R2= (R21 & R22 & R23 ) | ( R24 & R25 & R26 );
         R3= (R31 & R32 & R33 & R34);
    }
DA {
          Z1 = {IF R1 THEN a1*x+b1*u ELSE 0};
          Z2 = {IF R2 THEN a2*x+b2*u ELSE 0};
          Z3 = {IF R3 THEN a3*x+b3*u ELSE 0};
     }
CONTINUOUS {
       x =  Z1+Z2+Z3;}
OUTPUT {
```

```
          y = x;}
   } /* end implementation */
} /* end system */
```

Anexo F. Codificación "Branch and Bound y programción lineal entera mixta para MLD" en MatLab

```
function [x,fval,exitflag] = linprogmld(f,a,b,nxy,nb)
warning off;
options=optimset;
options.MaxIter=2000;
options.Diagnostics='off';
options.Display='off';
warning off MATLAB:divideByZero;
pila=[a,b,zeros(size(b))];
mi_eps=eps*100000;
mi_costomin = inf;
[mi_topemin,mi_cvar] = size(pila);
mi_topemin = mi_topemin(1);
mi_tope = mi_topemin;
mi_totrest = mi_topemin;
[x,fval,exitflag]=linprog(f,a,b,[],[],[],[],[],options);
mi_respuesta = x;
mi_costo = fval;
mi_exitflag = exitflag;
if exitflag <=0
   mi_costo = inf;
```

```
end
if mi_costo  < (mi_costomin - mi_eps)
%[mi_esunahoja,mi_nentero] = isvecint(x,1,length(x));
[mi_esunahoja,mi_nentero] = isvecint(x,nxy+1,nb+nxy);
if mi_esunahoja == 0
   mi_tope = mi_tope + 1;
   mi_mevalor = floor(x(mi_nentero));
   %mi_mevalor = 0.0;
   mi_restri = arestriccion('I',mi_nentero,mi_cvar,mi_mevalor,mi_totrest);
   pila=[pila;mi_restri];
   mi_tope = mi_tope + 1;
   mi_mavalor = -1*ceil(x(mi_nentero));
   %mi_mavalor = -1.0;
   mi_restri = arestriccion('D',mi_nentero,mi_cvar,mi_mavalor,mi_totrest);
   pila=[pila;mi_restri];
   mi_continuar = 'SI';
   mi_exitflag = -1;
else
   mi_continuar = 'NO';
   mi_costomin = mi_costo;
end
else
   mi_continuar = 'NO';
end
```

```
while mi_continuar=='SI'
   mi_restri = pila(mi_tope,1:mi_cvar);
   mi_tope = mi_tope -1;
   pila = pila(1:mi_tope,:);
   mi_numrest = mi_restri(length(mi_restri));
   a = a(1:mi_numrest,:);
   b = b(1:mi_numrest);
   a = [a;mi_restri(1:mi_cvar-2)];
   b = [b;mi_restri(mi_cvar-1)];
   [x,fval,exitflag]=linprog(f,a,b,[],[],[],[],[],options);
   mi_costo = fval;
   if exitflag <=0
      mi_costo = inf;
   end
   if mi_costo < (mi_costomin - mi_eps)
      %[mi_esunahoja,mi_nentero] = isvecint(x,1,length(x));
      [mi_esunahoja,mi_nentero] = isvecint(x,nxy+1,nb+nxy);
      if mi_esunahoja == 0
         mi_tope = mi_tope + 1;
         mi_mevalor = floor(x(mi_nentero));
         %mi_mevalor = 0.0;
         mi_totrest = size(a);
         mi_totrest = mi_totrest(1);
         mi_restri = arestriccion('I',mi_nentero,mi_cvar,mi_mevalor,mi_totrest);
```

```
        pila=[pila;mi_restri];
        mi_tope = mi_tope + 1;
        mi_mavalor = -1*ceil(x(mi_nentero));
        %mi_mavalor = -1.0;
        mi_restri = arestriccion('D',mi_nentero,mi_cvar,mi_mavalor,mi_totrest);
        pila=[pila;mi_restri];
       else
        mi_costomin = mi_costo;
        mi_respuesta = x;
        mi_exitflag = exitflag;
      end
   end
   if mi_tope <= mi_topemin
     mi_continuar='NO';
   end
end
x=mi_respuesta;
fval = mi_costomin;
exitflag = mi_exitflag;


function [a,b] = isinteger(num)
  mi_epsilon = eps*100000;
  mi_min = floor(num);
```

```
 %mi_min=0.0000;
 mi_max = ceil(num);
 %mi_max=1.0000;
 a = 0;
 if abs(num - mi_min) <= mi_epsilon
    a = 1;
    b = mi_min;
 end
 if abs (num - mi_max) <= mi_epsilon
    a = 1;
    b = mi_max;
 end



function [a,b] = isvecint(x,ini,fin)
 a=1;
 b=0;
 for i = ini:fin
   if isinteger(x(i))==0
      a=0;
      b=i;
      break
    end
 end
```

```
function [a] = arestriccion(lado,posj,ncol,valr,posi)
 if lado == 'I'
    mi_coe = 1.0;
end
if  lado == 'D'
    mi_coe = -1.0;
end
if posj == 1
       a=[mi_coe,zeros(1,ncol-3),valr,posi];
elseif posj == (ncol-2)
       a=[zeros(1,ncol-3),mi_coe,valr,posi];
   else
       a=[zeros(1,posj-1),mi_coe,zeros(1,ncol-2-posj),valr,posi];
end
```

Anexo G. Codificación Algoritmo de Validacion en Matlab

```
tic
caso1
X=0.0;
U=0.0;
XSMI=[];
XWMI=[];
VFLATMI=[];
```

```
XSMA=[];
XWMA=[];
VFLATMA=[];
datos=[];
incre = 0.1;
biendefinido=1;
i=0;
while X <= 5.0 & biendefinido==1
 U=0.0;
 while U <= 5.0 & biendefinido==1
   mi_colx = size(S.A,2);
   mi_coly = size(S.C,2);
   mi_colz = size(S.B3,2);
   [mi_filb,mi_colb]=size(S.E2);
   Ap=[-eye(mi_colx),-zeros(mi_colx,mi_coly),S.B2,S.B3;...
     eye(mi_colx),zeros(mi_colx,mi_coly),-S.B2,-S.B3;...
     -zeros(mi_coly,mi_colx),-eye(mi_coly),S.D2,S.D3;...
     zeros(mi_coly,mi_colx),eye(mi_coly),-S.D2,-S.D3;...
        zeros(mi_filb,mi_colx),zeros(mi_filb,mi_coly),S.E2,S.E3];
   B=[-S.A*X-S.B1*U;...
        S.A*X+S.B1*U;...
        -S.C*X-S.D1*U;...
        S.C*X+S.D1*U;...
        S.E4*X+S.E1*U+S.E5];
```

```
   [mi_fil,mi_col]=size(Ap);
   Ap=[Ap;[zeros(mi_colb,mi_colx),zeros(mi_colb,mi_coly),-
eye(mi_colb),zeros(mi_colb,mi_colz)]];
   B=[B;-zeros(mi_colb,1)];

Ap=[Ap;[zeros(mi_colb,mi_colx),zeros(mi_colb,mi_coly),eye(mi_colb),zeros(mi_colb,mi_
colz)]];
   B=[B;ones(mi_colb,1)];
   F=ones(mi_col,1);
   [Xp,FVAL,EXITFLAG]=linprogmld(F,Ap,B,mi_colx+mi_coly,mi_colb);
   i=i+1;
   XSMI=[XSMI;[Xp(1:mi_colx),Xp(mi_colx+1:mi_colx+mi_coly)]];
   XWMI=[XWMI,[Xp(mi_colx+mi_coly+1:size(Xp,1))]];
   VFLATMI=[VFLATMI;EXITFLAG];
   F=-F;
   [Xp,FVAL,EXITFLAG]=linprogmld(F,Ap,B,mi_colx+mi_coly,mi_colb);
   XSMA=[XSMA;[Xp(1:mi_colx),Xp(mi_colx+1:mi_colx+mi_coly)]];
   XWMA=[XWMA,[Xp(mi_colx+mi_coly+1:size(Xp,1))]];
   VFLATMA=[VFLATMA;EXITFLAG];
   if VFLATMI(i)>0 & VFLATMA(i)>0
     if iseq(XSMI(i,:),XSMA(i,:),eps*1000000)==0
       biendefinido=-1;
     end
  end
```

```
   datos=[datos;[X,U,biendefinido]];
   U = str2double(num2str(U + incre))
  end
  X= str2double(num2str(X + incre))
end
mi_tiempo=toc
save solcaso1
```

Anexo H. Sistema de validación distribuido implementado en Java

Anexo H.1 Modulo "IManager"

```
//Matlab Distributed Computing Tutorial
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//IManager.java:
//        Interface for Manager class.

package mld;
import java.lang.*;
import java.util.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;

public interface IManager extends Remote {
        public void TestCall() throws RemoteException;
        public Slot RequestSlot() throws RemoteException;
        public void DeliverSlot(Slot s) throws RemoteException;
        public void StartCalculation(Parameters p) throws RemoteException;
```

```
        public Results RetrieveResults() throws RemoteException;
    public Resultslot RetrieveSlot(int nslot) throws RemoteException;
        public void AbortCalculation() throws RemoteException;
    public void FinishCalculation() throws RemoteException;
}
```

Anexo H.2 Modulo “Manager”

```
//Matlab Distributed Computing Tutorial
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//Manager.java:
//        Remote object for the manager of the distributed calculation

package mld;
import java.lang.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;


public class Manager extends UnicastRemoteObject implements IManager {
        //keep track of the state of calculations
```

```
        private boolean bCalculating = false;
        private boolean bHaveResults = false;
      private int     numprocess   = 0;

        //data for current calculation
        private int[] slotsAssignment = new int[0]; //remembers whether the slots state:
                                                                    //-1:   unassigned;
0: assigned, but not completed; 1: assigned and completed

        private Slot[] slots = new Slot[0]; //stores the slots of the current calculation
        private Parameters parameters = new Parameters();;          //parameters    for
the current calculation

        ////////////////////////////////////////////////////
        //Manager initializer, constructor, test call

        //allow the Manager as a stand-alone
        public static void main(String args[]) {
                Manager manager;
                try {
            manager = new Manager();
            Naming.rebind("Manager", manager);
            System.out.println("Manager ready.");
```

```
System.out.println("==========================================================
============");
            } catch (Exception e) {
              System.out.println("Exception: " + e.getMessage());
              return;
            }
        }

        public Manager() throws RemoteException {
                super();
                return;
        }

        public void TestCall() throws RemoteException {
                //unecessary utility function for testing the connection
                return;
        }

        ////////////////////////////////////////////////////
        //Interface functions to be called by clients


        public void StartCalculation(Parameters p) throws RemoteException {
```

```
        //start the calculation with a certain set of parameters

        //make sure we're not in the middle of a calculation
        if (!bCalculating) {
                //store the parameters locally
                this.parameters = p;

                //initalize the slots
                SlotsAllocateUnassigned(this.parameters.totalSlots);

                //set the state
                bCalculating = true;
                bHaveResults = false;
        numprocess   = numprocess +1;

                //print something for the user

    System.out.println("==========================================
==========" );
                System.out.println("Calculation STARTED" );
                System.out.println("----------------------------------------------------
---" );
        } else {
```

```
                System.out.println("ERROR: Tried to start a calculation while we're still running another!");
            }

            return;
        }

        public Slot RequestSlot() throws RemoteException {
            //returns a slot object with data to be processed or an empty slot object if there is nothing to do
            Slot slot = new Slot(); //the response object

          if (!bCalculating) {
                //if we can't find an unassigned slot, return a Slot object that says "wait!"
                slot.bWait = true;
                return slot;
            }

            //try to find an unassigned slot and assign it
            //(this needs to be done in a single operation to avoid anyone else
            int slotNum = SlotAssignUnassigned();

            if (slotNum != -1) {
```

```
                //We found an unassigned slot

                //REQUIRED
            slot.process = numprocess;
                slot.slotNum = slotNum;
                slot.totalSlots = parameters.totalSlots;
                slot.bWait = false;

                //APPLICATION-SPECIFIC
                slot.nombarch = parameters.archdat;
                slot.valorx = parameters.vdatos[0][slotNum];
            slot.valoru = parameters.vdatos[1][slotNum];
                //leave the means array alone, it will be filled by the Worker

                System.out.println("Assigned  slot: " + slotNum);
            } else {
                //if we can't find an unassigned slot, return a Slot object that says
"wait!"
                slot.bWait = true;

            }

            return slot;
        }
```

```
public void DeliverSlot(Slot s) throws RemoteException {
        //delivery of a processed slot by the Worker

        //make sure we are currently accepting slots
        if (!bCalculating) {
                System.out.println("Warning: worker tried to deliver slots while no calculation was running!" );
                return;
        }
      if (s.process != numprocess) {
                System.out.println("Warning: worker tried to deliver slots of other process" );
                return;
        }
        //take the slot number out of the slot and store it
        int slotNum = s.slotNum;
        slots[slotNum] = s;

        //mark the slot as delivered
        boolean bFinished = SlotMarkDelivered(slotNum);

        //if the calculation is finished, put us in the respective state
        if (bFinished) {
```

```
                this.bCalculating = false;
                this.bHaveResults = true;
                System.out.println("-------------------------------------------------------
---");
                System.out.println("Calculation FINISHED!" );

        System.out.println("==============================================
==================");
            }

            return;
        }

        public Results RetrieveResults() throws RemoteException {
            //returns the results of the current calculation or an empty Results object if
the calculation is not yet finished.
            Results results = new Results();

            //check if we even have results we can deliver and are not currently
calculating
            if (bHaveResults && (!bCalculating)) {
                System.out.println("Delivering results ..." );
                results.bWait = false;
                results.slots = slots;
```

```
                results.totalSlots = parameters.totalSlots;
        } else {
                //tell the retriever to wait
                results.bWait = true;
        }

        return results;
    }

    public Resultslot RetrieveSlot(int nslot) throws RemoteException {
        //returns the results of the current calculation or an empty Results object if the calculation is not yet finished.
        Resultslot results = new Resultslot();

        //check if we even have results we can deliver and are not currently calculating
        if ( (nslot <= parameters.totalSlots)  && (slotsAssignment[nslot]==1)) {
                System.out.println("Delivering Slot ..." + nslot );
                results.bWait = false;
                results.numSlot = nslot;
                results.rslot = slots[nslot];
        } else {
                //tell the retriever to wait
                results.bWait = true;
```

```
            }

            return results;
        }

        public void AbortCalculation() throws RemoteException {
            //aborts the current calculation, no matter what the state is
            this.parameters = null;
            this.bCalculating = false;
            this.bHaveResults = false;

            SlotsAllocateUnassigned(0);

            System.out.println("------------------------------------------------------------");
            System.out.println("Calculation ABORTED!" );

        System.out.println("==========================================
==================");

            return;
        }

    public void FinishCalculation() throws RemoteException {
```

```
            //aborts the current calculation, no matter what the state is

            this.bCalculating = false;
            this.bHaveResults = true;

        SlotsAllocateUnassigned(0);

            System.out.println("------------------------------------------------------------");
            System.out.println("Calculation FINISHED!" );

        System.out.println("==========================================
==================");

            return;
        }

        ////////////////////////////////////////////////////
        //Private utility functions,
        //mostly handling the SlotAssignment array.
        //All of these functions need to be synchronized, so they
        //can't interfere which other when writing to the array

        public synchronized void SlotsAllocateUnassigned(int totalSlots) {
            slotsAssignment = new int[totalSlots];
```

```
        slots = new Slot[totalSlots];
        for (int i = 0; i < totalSlots; i ++) {
                slotsAssignment[i] = -1; //set all slots to unassigned
        }
}

public synchronized int SlotAssignUnassigned() {
        //assign stationNo to the first unassigned slot
        int i = 0;

        while (i < parameters.totalSlots) {
                if (slotsAssignment[i] == -1) {
                        slotsAssignment[i] = 0;

                        return i;
                }
                i++;
        }

        return -1;
}

public synchronized boolean SlotMarkDelivered(int slotNum) {
        slotsAssignment[slotNum] = 1;
```

```
                //at this point, the calculation might be finished
                for (int i = 0; i < parameters.totalSlots; i++) {
                        if (slotsAssignment[i] <= 0) {
                                return false;
                        }
                }

                return true;
        }
}
```

Anexo H.3 Modulo “Slot”

```
//Matlab Distributed Computing Tutorial
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//Slot.java:
//        Class containing the data for a slot

package mld;
import java.lang.*;
import java.io.*;
```

```
public class Slot implements Serializable {
        //REQUIRED
        public int slotNum;             //number of this slot
        public int totalSlots;  //out of a total number of slots
      public int process;

        public boolean bWait;           //this is set to true if there is no currently unassigned
slot
                                                //and the Worker needs to wait a bit
longer

        //APPLICATION-SPECIFIC
      public String nombarch;
        public double valorx;
        public double valoru;

        public double[] vsmi;           //results we got back from the simulation
        public double[] vwmi;      //(when the Manager sends this out the Worker, this is
empty)
        public double flatmi;
      public double[] vsma;    //results we got back from the simulation
        public double[] vwma;      //(when the Manager sends this out the Worker, this is
empty)
```

```
        public double flatma;
    public double mtime;
}
```

Anexo H.4 Modulo “Resultslot”

```
//Matlab Distributed Computing Tutorial
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//ResultSlot.java:
//        Class containing the results of a calculation

package mld;
import java.lang.*;
import java.io.*;

public class Resultslot implements Serializable {
        //REQUIRED
        public boolean bWait;          //boolean variable signalising to wait
                                                //when the calculation has not yet
completed
        public int numSlot;    //total number of slots
        public Slot rslot;       //the resulting slots
```

}

Anexo H.5 Modulo “Results”

```
//Matlab Distributed Computing Tutorial
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//Results.java:
//        Class containing the results of a calculation

package mld;
import java.lang.*;
import java.io.*;

public class Results implements Serializable {
        //REQUIRED
        public boolean bWait;          //boolean variable signalising to wait
                                                //when the calculation has not yet
completed
        public int totalSlots;   //total number of slots
        public Slot[] slots;     //the resulting slots
}
```

Anexo H.6 Modulo “Parameters”

```
//Matlab Distributed Computing Tutorial
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//Parameters.java:
//        Class containing the initial parameters of a calculation

package mld;
import java.lang.*;
import java.io.*;

public class Parameters implements Serializable {
        //REQUIRED
        public int totalSlots;  //total number of slots that need to be carried
                                                        //out for this calculation

        //APPLICATION-SPECIFIC
        public String archdat;
        public double[][] vdatos;
}
```