TESIS DEFENDIDA POR

Araceli Gárate García

Y aprobada por el siguiente comité:

Dr. Luis Alejandro Márquez Martínez

Director del Comité

Dr. Joaquín Álvarez Gallegos

Miembro del Comité

MC. José Ricardo Cuesta García

Miembro del Comité

Dra. Ana Isabel Martínez García

Miembro del Comité

Dr. Arturo Velázquez Ventura

Coordinador del Programa de Posgrado en Electrónica y Telecomunicaciones

Dr. Edgar G. Pavía López

Director de Estudios de Posgrado

13 de Noviembre de 2006

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN
SUPERIOR DE ENSENADA

POSGRADO EN CIENCIAS
EN ELECTRÓNICA Y TELECOMUNICACIONES

Análisis y control de sistemas no lineales con retardos mediante cálculo simbólico

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

MAESTRO EN CIENCIAS

Presenta:

Araceli Gárate García

Ensenada, Baja California, México. Noviembre de 2006.

RESUMEN de la tesis de **Araceli Gárate García**, presentada como requisito parcial para obtener el grado de MAESTRO EN CIENCIAS en ELECTRÓNICA Y TELECOMUNICACIONES con orientación en INSTRUMENTACIÓN Y CONTROL. Ensenada, Baja California. Noviembre de 2006.

Análisis y control de sistemas no lineales con retardos mediante cálculo simbólico

Resumen aprobado por:

Dr. Luis Alejandro Márquez Martínez

Director de Tesis

En este trabajo se presenta el desarrollo de una utilería informática denominada SAC (por sus siglas en inglés, Symbolic Analysis and Control package), la cual permite determinar propiedades importantes de sistemas no lineales con y sin retardos utilizando un enfoque algebraico de control. La metodología empleada en su construcción implica el uso de varias herramientas de ingeniería de software que han permitido obtener un programa estructurado para su fácil mantenimiento. Este programa es de distribución libre ya que se rige bajo licencia GNU GPL y pretende ser una herramienta de propósito general que pueda incluir soporte para otra clase de sistemas, tales como sistemas no lineales en tiempo discreto.

Palabras clave: Enfoque diferencial-algebraico, sistemas no lineales con retardos, cálculo simbólico.

ABSTRACT of the thesis presented by **Araceli Gárate García**, as a partial requirement to obtain the MASTER IN SCIENCE degree in ELECTRONICS AND TELECOMMUNICATIONS with specialization in INSTRUMENTATION AND CONTROL. Ensenada, Baja California. November 2006.

Analysis and control of non-linear time delay systems through symbolic computation

Abstract approved by:

Dr. Luis Alejandro Márquez Martínez

Thesis director

In this work, the development of a computer toolbox called SAC (Symbolic Analysis and Control package), is shown. This toolbox computes several properties of nonlinear systems with or without time-delays obtained through the so-called algebraic approach. The use of software engineering tools to design SAC permitted to develop a well-structured, easy-to-mantain software system. This software is open-source, and its code is released under the GNU Public License (GPL). The aim is to create a general-purpose toolbox that includes support to other classes of systems (e.g. nonlinear discrete-time systems).

Keywords: Differential-algebraic approach, non-linear time delay systems, symbolic computation.

Dedicado a mis padres

Guadalupe García Contreras
y
Francisco Gárate Velarde.

Por ustedes y para ustedes con todo mi amor.

Agradecimientos

Agradezco primeramente a Dios, por haber puesto a las siguientes personas en mi vida.

A mi familia, por ser mi mayor tesoro. En especial a mis tíos Ana y Fred que son otros padres para mí.

Al Dr. Luis Alejandro Márquez Martínez por ser mi guía, mi apoyo y sobre todo mi amigo a través de este sendero. Gracias por toda la paciencia y por brindarme la oportunidad de trabajar a tu lado.

Al Dr. Joaquín Álvarez Gallegos porque a pesar de todas sus ocupaciones siempre se dá tiempo para transmitir sus conocimientos y experiencia a los estudiantes. Gracias por sus clases, porque sin ellas la maestría no tendría el mismo nivel y por sus aportaciones a este trabajo de tesis.

Al M.C. José Ricardo Cuesta García por el interés mostrado en el tema, por ser un lector cuidadoso y por todas sus útiles observaciones.

A la Dra. Ana Isabel Martínez García por compartir sus conocimientos en el campo de la ingeniería de software, sin los cuales no se hubiera podido llevar a cabo este trabajo de manera adecuada. Gracias por su amabilidad y paciencia.

Al Dr. Fidel Díaz Muñoz, a quien debo mucho. Gracias por haber sembrado en mí la inquietud de mejorar y seguir hasta el final en esta aventura del posgrado; por todas sus palabras de aliento, por la confianza y cariño que siempre me ha brindado. Por ser un ejemplo para todos sus estudiantes.

A Rosy, Dolores, Citlali, Ivonne, Norma, Laura y Aurora por todas sus atenciones.

A Eddie, mi compañero incondicional a lo largo de esta etapa. Gracias por tu tiempo, por tu ánimo y por tu sonrisa que iluminó mi camino en los momentos más difíciles.

Gracias a mis amigos: los de siempre, los que hice en CICESE y en Ensenada. Les agradezco todos los momentos que hemos pasado juntos y el hacerme sentir su apoyo en todo momento. En especial a Carlos, Victor, Alberto, Jesús y Fátima que ahora son parte de mi familia. También al grupo de Evovisión y al Dr. Gustavo Olague por compartir conmigo su amistad y conocimientos. Sin faltar Glenda, Paola, Azalia, Sonia, Lorena, Israel, Neftalí, Ulises y Johanna que siempre están en mi pensamiento.

Al Centro de Investigación Científica y de Educación Superior de Ensenada y al Consejo Nacional de Ciencia y Tecnología, por darme las facilidades para estudiar en esta institución.

Ensenada, México — Araceli Gárate García
13 de Noviembre de 2006.

Tabla de Contenido

Capítulo Página

Tabla de Contenido (Continuación)

Capítulo Página

Lista de Figuras

Lista de Figuras (Continuación)

Lista de Figuras (Continuación)

Lista de Tablas

Capítulo I

Introducción

El comportamiento de los sistemas reales es esencialmente no lineal; sin embargo, al modelarlos generalmente se busca una aproximación mediante sistemas de ecuaciones lineales. Esto se debe a las dificultades encontradas al pasar del campo de la teoría lineal a la no lineal, lo que incrementa la complejidad tanto del modelo como de las herramientas matemáticas necesarias para su análisis. Aproximar el comportamiento de un sistema alrededor de un punto de operación mediante un modelo lineal es una práctica muy común y útil. Sin embargo, cuando el modelo resultante no puede describir satisfactoriamente algunas dinámicas del sistema es necesario recurrir al análisis no lineal. Tal como se describe en Khalil (2002) hay dos limitantes básicas:

- La linealización describe el comportamiento local de un sistema en la vecindad de un punto de operación, por lo que no puede predecir el comportamiento global en el espacio de estado.

- Existen fenómenos exclusivos del comportamiento no lineal que no pueden representarse mediante modelos lineales.

En los sistemas con retardos se presentan problemas similares a los descritos anteriormente cuando los retardos o "tiempos muertos" no son considerados en el análisis. Un sistema con retardos es un sistema dinámico cuya evolución en el tiempo depende no solamente de su estado actual, sino también de su historia pasada, y como se muestra en la literatura, aún cuando estos retardos son pequeños, pueden tener un efecto desestabilizador en un sistema (Abdallah *et al.*, 1993; Fridman, 2002; Hale y Lunel, 1999).

Otro punto importante a considerar es que el tomar en cuenta las no linealidades y los retardos al representar un sistema permite tener un modelo más aproximado a la planta real y por lo tanto es posible obtener un mejor desempeño de un sistema de control si se utilizan las herramientas de diseño adecuadas. Es por esto que desde que aparecieron los primeros trabajos respecto al control de sistemas con retardos, al final de los años 50, numerosos investigadores han sumado esfuerzos para aportar nuevos conocimientos y desarrollar una teoría específica en esta área de investigación (Smith, 1957, 1959).
El interés se ha incrementado en los últimos años, según Richard (2003), debido a cuatro factores principales:

- *Los retardos existen frecuentemente en sistemas reales; varios procesos incluyen este fenómeno en su dinámica interna.* Actuadores, sensores, redes con lazo de retroalimentación usualmente introducen retardos en el sistema, por lo que los encontramos en áreas tan diversas como economía, redes, teleoperación, por mencionar algunas (Manfredi y Fanti, 2004; Qiang Zhang, 2005; Yu-Ping, 2005; Azorin *et al.*, 2004).

- *No es una alternativa realizar aproximaciones de un sistema con retardos a uno de dimensión finita para diseñar controladores clásicos debido al efecto que esto puede ocasionar en términos de causalidad, estabilidad y oscilaciones.* El uso de estrategias de control estándar resulta en expresiones que dependen de valores futuros del estado, por lo que se requiere el uso de predictores.

- *Existen casos en que la introducción intencional de retardos a un sistema puede mejorar el control* (Jalili y Olgac, 1998; Abdallah *et al.*, 1993).

- *La aparición de retardos en una ecuación diferencial puede llevar a la simplificación del modelo, como se explica en Kolmanovskii y Myshkis (1999).*

Como había mencionado con anterioridad, son varias las teorías desarrolladas para analizar los sistemas con retardos; una de las más prometedoras para el caso no lineal se basa en un enfoque algebraico, el cual ha hecho varias aportaciones importantes en temas como accesibilidad, controlabilidad, observabilidad, análisis estructural, equivalencia a la forma triangular, rechazo a perturbaciones, linealización entrada-salida, linealización entrada-estado, entre otros, mostrando su efectividad como herramienta de análisis de estos sistemas. Este enfoque se basa a su vez en el uso de un anillo muy particular de polinomios no conmutativos, lo cual complica los cálculos involucrados en el análisis y el que estos se lleven a cabo manualmente, con mayor razón cuando el sistema consta de un número grande de ecuaciones o éstas son de gran tamaño.

Como sabemos, las computadoras han sido de mucha utilidad en el avance de la ciencia, sobre todo cuando se trata de cálculos muy complejos, extensos o repetitivos, por lo que una solución al problema anteriormente mencionado es el desarrollo de rutinas que realicen las operaciones mediante un paquete de cálculo simbólico, de ahí el desarrollo de este trabajo (Márquez-Martínez, 2004).

I.1 Antecedentes

El concepto de sistema algebraico computacional (CAS, del inglés Computer Algebra System) se remonta al siglo XVIII cuando, en el libro *arithmetica universalis*, Newton menciona métodos para manipular expresiones matemáticas que involucran símbolos y algoritmos para resolverlas (Newton, 1728). Sin embargo, fue hasta 1953 que estos principios encontraron aplicación cuando Kahrimanian y Nolan publicaron sus trabajos de tesis, elaborados de forma independiente, sobre derivación analítica mediante una computadora digital (Kahrimanian, 1953; Nolan, 1953). Sumado a esto, en el MIT John McCarthy desarrolló a principio de los años sesenta un lenguaje de programación de alto nivel, LISP, acrónimo de *list processor*, lo que impulsó la creación de varios CAS.

Surgió entonces una nueva disciplina dedicada a la manipulación, por parte de programas, de expresiones algebraicas no numéricas, conocida como cálculo simbólico o álgebra computacional, con la cual se dieron soluciones exactas a diversos problemas. Algunos CAS trataban temas específicos en el campo de la matemática aplicada, otros contenían un gran número de funciones y procedimientos generales. Algunos ejemplos de estos programas son:

- REDUCE, escrito en LISP en 1963 por Anthony C. Hearn para tratar problemas de física de alta energía.
- MACSYMA, desarrollado en LISP, en el MIT, entre 1969 y 1972. Éste fue la punta de lanza de los programas de cálculo simbólico potentes.
- MAPLE, desarrollado desde 1980 en la universidad de Waterloo, escrito en C.
- SMP, escrito a principios de los ochenta por Stephen Wolfram, el cual constituyó la semilla de MATHEMATICA.
- MATHEMATICA, escrito en C, en 1988.
- Otros como: muPAD, DERIVE, AXIOM, COCOA, MATHCAD, GAUSS, etc.

Estos programas son, entre otros, los que han ido evolucionando y en los últimos años han revolucionado la enseñanza de las matemáticas y de las ciencias que las aplican.
El área de control automático no se ha quedado atrás. Son varios los programas que se han creado para distintas tareas, algunos de cálculo numérico y otros de cálculo simbólico. Entre los programas más completos para sistemas no lineales encontramos el NOLIACPA, creado en el IRCCyN, en Francia, el cual se basa en Maple y calcula varias propiedades de sistemas no lineales; sin embargo, este programa no permite introducir retardos (Vignaud, 1997). Dirigiéndonos hacia el noreste de Europa, en la Universidad

de Tallin en Estonia (TUT), podemos encontrar herramientas de cálculo simbólico basadas en Mathematica para el estudio de sistemas no lineales en tiempo discreto (Kotta y Tõnso, 1998, 2005). Así podemos citar varios paquetes, aunque ninguno para sistemas no lineales con retardos (SNLR).

El no encontrar paquetes disponibles para SNLR se debe desde luego, a la complejidad de los algoritmos, difíciles de implementar en cualquier programa de cálculo simbólico, aún cuando la teoría necesaria para resolver varios de los problemas se encuentra disponible en la literatura. Contribuir al desarrollo de utilerías para facilitar el análisis de sistemas no lineales con retardos es el objetivo general de este trabajo de tesis.

I.2 Objetivo general

Desarrollar un programa para el análisis y el diseño de algoritmos de control de SNLR, mediante un enfoque algebraico, utilizando un lenguaje de cálculo simbólico.

I.3 Objetivos particulares

- Programar diversas pruebas auxiliares y algoritmos para resolver los problemas de:

  Accesibilidad.

  Equivalencia a la forma triangular.

  Observabilidad.

  Linealización por inyecciones aditivas E/S.

  Rechazo de perturbaciones.

- Lograr un producto de fácil comprensión y mantenimiento, mediante la aplicación de herramientas básicas de ingeniería de software.

- Diseñar una documentación consistente y adecuada del software.

I.4 Motivación

Varios son los puntos que motivan el desarrollo de este trabajo:

- Los fenómenos de transporte, así como el tiempo de respuesta de sensores y actuadores introducen tiempos muertos, lo cual origina que los retardos estén presentes en un gran número de sistemas reales.

- No existe actualmente un software de cálculo simbólico que trabaje con sistemas no lineales con retardos.

- La efectividad que ha mostrado el enfoque algebraico al trabajar con sistemas no lineales con retardos.

- Los cálculos involucrados al trabajar bajo dicho enfoque con sistemas con retardos son complejos y difíciles de llevar a cabo manualmente.

I.5 Organización de la tesis

En el capítulo de *preliminares* se abordan los conceptos y teoremas básicos necesarios para comprender la teoría del enfoque algebraico y se especifica el tipo de sistemas que manejan los algoritmos desarrollados a lo largo del trabajo.
En el capítulo de *proceso de construcción del software* se describen los requerimientos básicos del programa SAC, así como el perfil del usuario. Se muestran las herramientas de ingeniería de software utilizadas durante la etapa de análisis para organizar adecuadamente el paquete, tales como los casos de estudio, diagramas de flujo, diccionario de datos y árboles de decisión. También se indican las razones por las que se eligió el lenguaje de cálculo simbólico *Maxima* para el desarrollo de SAC, así como las plataformas

en que trabaja.

El capítulo *implementación y funcionamiento de algoritmos en SAC* es la sección medular de este trabajo, ya que aquí se presentan los distintos problemas de análisis y control de sistemas no lineales con retardos, se describen los algoritmos desarrollados y las estrategias de prueba del software.

Por último se exponen las conclusiones y aportaciones de esta tesis, así como las recomendaciones para trabajo futuro.

Capítulo II

Preliminares

La herramienta utilizada en este trabajo para el análisis de sistemas no lineales con retardos es una teoría matemática que se basa en un anillo de polinomios no conmutativos muy particular. Esta teoría ha adquirido importancia en los últimos años debido a las dificultades encontradas al intentar extender las herramientas del enfoque geométrico de la teoría lineal a la no lineal y en especial a los sistemas no lineales con retardos, problemas cuya solución se ha facilitado con el uso de esta metodología.

En este capítulo se presentan algunos conceptos matemáticos directamente relacionados con el enfoque algebraico y con los sistemas considerados en este trabajo, descritos en la Sección II.2. Para mayor información consultar Márquez-Martínez (2000); Márquez-Martínez y Moog (2001b); Xia *et al.* (2002).

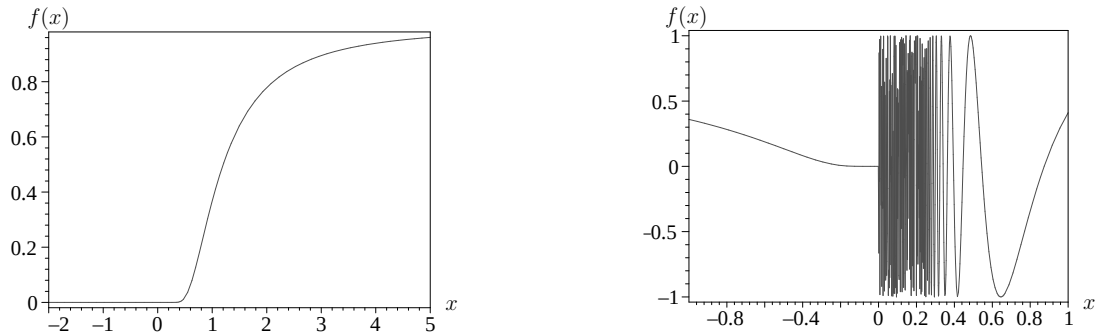
II.1 Formalismo algebraico

II.1.1 Funciones analíticas y funciones meromorfas

En cualquier área de conocimiento el modelar implica una aproximación, y por lo tanto un cierto grado de incertidumbre; por esta razón, al estudiar el control de sistemas no lineales es de interés el considerar situaciones en las cuales las propiedades de un sistema son válidas en la vecindad de un punto x_0.

En términos matemáticos esto se logra al considerar *propiedades genéricas*, es decir, que permanecen válidas en subconjuntos abiertos y densos, dentro del dominio de definición. Sin embargo, esto impone restricciones sobre la clase de funciones consideradas (Conte *et al.*, 1999).

El conjunto de funciones infinitamente derivables, denominadas C^∞, que usualmente son utilizadas en el análisis de sistemas dinámicos, tienen la desventaja de formar un anillo integral, el cual carece de propiedades importantes como la división euclidiana. En su lugar se utilizan funciones meromorfas. Para definirlas formalmente, se hace uso de la noción de función analítica.



(a) $f(x) = \begin{cases} e^{(-1/x^2)}, & \forall\ x > 0 \\ 0, & \forall\ x \leq 0 \end{cases}$

(b) $f(x) = \begin{cases} sen\left(e^{(1/x)}\right), & \forall\ x \neq 0 \\ 0, & \forall\ x = 0 \end{cases}$

Figura 1: Funciones no analíticas.

Definición 1 (Función analítica)**.** Sea $\mathcal{D} \subset I\!R^n$ un dominio abierto de $I\!R^n$. Una función $f : I\!R^n \to I\!R$ es analítica en $x_0 \in \mathcal{D}$ si admite una expansión en series de potencias en la vecindad de x_0. Se dice que f es analítica globalmente si $\mathcal{D} = I\!R^n$.

Estas funciones tienen dos propiedades básicas de interés (Conte *et al.*, 1999):

Proposición 1. *Sea $f : I\!R^n \to I\!R$ una función analítica. Entonces*

1. *f es cero, o*
2. *el conjunto de ceros de f tiene un interior vacío.*

En la Figura 1a se observa que existe un continuo de ceros en f, en tanto que en la Figura 1b se tiene un punto de acumulación, por lo que en ningún caso se trata de una función analítica al no tener los ceros aislados.

Debido a la Proposición 1 se puede hablar de un *rango genérico* de matrices cuyos componentes son funciones analíticas. Este rango es igual al de la máxima submatriz no singular y coincide con el rango de la matriz en los puntos de un subconjunto abierto denso contenido en $\mathbb{R}^n$. Cabe señalar que el rango genérico es igual o mayor que el rango de la matriz en cualquier punto de $\mathbb{R}^n$.

Las funciones analíticas, por lo general, no tienen inversas analíticas, sin embargo en conjunto forman un *anillo integral* $\mathcal{A}_r : \mathbb{R}^r \to \mathbb{R}$.

Al ser un anillo $\mathcal{A}_r$ cuenta con dos leyes de composición interna, la suma y el producto, que satisfacen las condiciones siguientes:

1. Asociatividad aditiva y multiplicativa.
2. Conmutatividad aditiva.
3. Elemento neutro para la suma.
4. Elemento inverso de la suma.
5. Distributividad a izquierda y derecha.

Además al ser un anillo integral cumple que, para todo x, y no nulos $xy \neq 0$.

Podemos extender el enfoque algebraico a otra clase de funciones, llamadas *funciones meromorfas*, si se considera la noción de inverso multiplicativo para cada elemento no nulo. Con este fin de manera natural se puede asociar al anillo de funciones analíticas un campo cociente $\mathcal{K}_r$ cuyos elementos son pares $(f, g) \in \mathcal{A}_r$ tales que $g \neq 0$, módulo la relación de equivalencia $\equiv_R$ definida por $(f, g) \equiv_R (f', g')$ si y sólo si $fg' = gf'$. Escogiendo un representante en dicha clase, un elemento de $\mathcal{K}_r$ se escribirá como f/g.

La suma " + " y el producto " · " están formalmente definidos por:

$$\begin{aligned}(f_1, g_1) + (f_2, g_2) &:= (f_1 g_2 + f_2 g_1, g_1 g_2) \\ (f_1, g_1) \cdot (f_2, g_2) &:= (f_1 f_2, g_1 g_2)\end{aligned}$$

con g_1 y g_2 no idénticamente cero.

El anillo $\mathcal{A}_r$ es un subconjunto de $\mathcal{K}_r$ mapeando cada elemento $f \in \mathcal{A}_r$ a $f/1 \in \mathcal{K}_r$ y dado que es un anillo integral $g_1 \cdot g_2 \neq 0$ en $\mathcal{K}_r$.

Entonces cualquier *función meromorfa* está formada por el cociente de dos funciones analíticas.

II.1.2 Funciones causales

En este trabajo se considera que una función $\varphi(z(t), z(t \pm 1), \dots)$ es no anticipativa o causal[1] si cumple que:

$$\frac{\partial \varphi(\cdot)}{\partial (z(t+\tau))} \equiv 0, \qquad \forall \tau > 0.$$

Los sistemas considerados, como se define en la siguiente sección, se restringen al uso de este tipo de funciones para evitar el uso de predictores en el análisis de los sistemas con retardos.

II.2 Sistemas considerados

La clase de sistemas dinámicos considerados se definen por un conjunto de ecuaciones de la forma:

$$\Sigma : \begin{cases} \dot{x}(t) &= f(x(t-i), u(t-i), i \in \underline{S}) \\ y(t) &= h(x(t-i), i \in \underline{S}) \\ x(t) &= \varphi; u(t) = u_0, \forall t \in [t_0 - s, t_0] \end{cases} \tag{1}$$

[1] Para una definición formal consultar, por ejemplo Khalil (2002).

El estado $x \in \mathbb{R}^n$, la entrada de control $u \in \mathbb{R}^m$, y la salida $y \in \mathbb{R}^p$. Los elementos de f y h son funciones meromorfas de sus argumentos. $\underline{S} := \{0, \tau, \ldots, s\tau\}$, con $\tau, s \in \mathbb{R}^+$, es un conjunto finito de retardos constantes en tiempo, $f(x(t-i), i \in \underline{S}) := f(x(t), x(t-1), \ldots, x(t-s))$ y φ representa una función seccionalmente continua de condiciones iniciales.

La existencia de una solución está garantizada si u es tal que $\dot{x} = \bar{f}(x(t-i), i \in \underline{S})$ y $\bar{f}(x(t-i), i \in \underline{S})$ es continua con respecto a sus argumentos. Si además $\bar{f}(x(t-i), i \in \underline{S})$ es localmente lipschitz esta solución es única[2] (El'sgol'ts y Norkin, 1973).

En algunas situaciones el uso de una representación general, como la (1), lleva a sistemas cuyos elementos no son funciones meromorfas. Un ejemplo es $f(x,u) = sen(u)$: si queremos aplicar una retroalimentación de estado de la forma $u(t) = \alpha(x(t)) + \beta(x(t))v(t)$, se puede elegir $u = \dfrac{1}{x}$ que lleva a $sen\left(\dfrac{1}{x}\right)$ que no es analítica en cero.

En estos casos se consideran los sistemas que tienen una representación afín:

$$\Sigma : \begin{cases} \dot{x}(t) &= f(x(t-i), i \in \underline{S}) \\ & \quad + \sum_{j=0}^{s} g_j(x(t-i), i \in \underline{S}) u(t-j) \\ y(t) &= h(x(t-i), i \in \underline{S}) \\ x(t) &= \varphi; u(t) = u_0, \forall t \in [t_0 - s, t_0] \end{cases} \tag{2}$$

En general se cumplen las mismas condiciones que para el sistema (1). Los elementos de g_j también son funciones meromorfas. En esta clase de representación se hace la hipótesis de que todas las entradas de control $u(t)$ actúan de manera independiente sobre el sistema (2), por lo que $rango \begin{bmatrix} g_1 & g_2 & \ldots & g_s \end{bmatrix} = m$.

Para determinar la evolución de un sistema con retardos es necesario conocer los valores

[2]Aunque el enfoque utilizado está limitado al caso conmensurable en la práctica los retardos se conocen hasta una precisión finita, por lo que los retardos no conmensurables generalmente se pueden aproximar por retardos conmensurables.

De igual manera, retardos distribuidos se pueden aproximar mediante $\int_{t-s\tau}^{t} x(\tau)d\tau \approx \sum_{k=0}^{s} a_i x(t-k\tau)$.

del estado a cada instante dentro del intervalo $[t_0 - s, t_0]$, es decir, se requiere un número infinito de condiciones iniciales, por esta razón su dimensión es infinita.

El conjunto de símbolos $\{x(t-i), u(t-i), \ldots, u^{(k)}(t-i), i \in \underline{S}, k \in \mathbb{N}\}$ son independientes, en el sentido de que no existe ecuación algebraica que relacione una función meromorfa no trivial ϕ de la siguiente manera:

$$\phi(x(t-i), u(t-i), \ldots, u^{(k)}(t-i)) = 0. \tag{3}$$

Cambiando de notación:

$$\begin{aligned} z_{i,j} &:= x_i(t-j) \\ u_{i,j} &:= u_i(t-j) \\ u^{(k)}_{i,j} &:= u^{(k)}_i(t-j). \end{aligned}$$

Se considera el conjunto infinito de variables reales

$$\mathcal{C} := \{z_{1,0}, z_{2,0}, \ldots, z_{1,1}, z_{2,1}, \ldots\}$$

para representar las coordenadas de un punto en $\mathbb{R}^{r\sigma}$, donde $r, \sigma \in \mathbb{N}$ se utilizan los elementos de $\mathcal{C}$ siguientes:

$$z_{i,j}, \qquad i = 1 \ldots r, j = 0 \ldots \sigma - 1. \tag{4}$$

Una función $g : \mathbb{R}^{r\sigma} \to \mathbb{R}$ se describe como una función de $r\sigma$ variables de $\mathcal{C}$ dadas por (4). $\mathcal{K}_{r,\sigma}$ se define como el campo de todas las funciones meromorfas dadas por $r\sigma$ indeterminadas.

El operador derivación actúa naturalmente sobre $\mathcal{K}_{r,\sigma}$, es decir, posee una estructura de campo diferencial:

$$\frac{\partial z_{i,j}}{\partial z_{i',j'}} : \begin{cases} 1 & \text{si} \quad i = i' \ \text{y} \ \text{j} = \text{j}' \\ 0 & \text{en} \quad \text{otro caso} \end{cases}$$

Si $\mathcal{K}$ se define como la unión teórica de los conjuntos $\mathcal{K}_{r,\sigma}$, entonces cada elemento de este campo es una función meromorfa de un número *finito* de elementos de $\mathcal{C}$ y puede denotarse por:

$$a(\mathbf{z}_0, \ldots, \mathbf{z}_{\sigma-1}) \text{ donde } \mathbf{z}_i := \{z_{1,i}, \ldots, z_{r,i}\}.$$

Considerando las diferenciales de los elementos de $\mathcal{C}$, se tiene otro conjunto infinito de símbolos (indeterminados)

$$d\mathcal{C} = \{dz_{i,j}, i = 1, 2, \ldots, \;\; j = 0, 1, \ldots\}.$$

Este conjunto de funciones diferenciales generan sobre el campo $\mathcal{K}$ un espacio vectorial $\mathcal{E}$, esto es,

$$\mathcal{E} = \text{span}_{\mathcal{K}} dC. \tag{5}$$

Cualquier elemento de $\mathcal{E}$, llamado 1-forma, es un vector de la forma

$$v = \sum_{\substack{i\geq 1 \\ l\geq 0}} F_{i,l} dz_{i,l}, \qquad F_{i,l} \in \mathcal{K}$$

donde solamente un número finito de coeficientes $F_{i,l}$ son diferentes de cero. El espacio vectorial $\mathcal{E}$ se puede identificar como un producto cartesiano $\mathcal{K} \times \cdots \times \mathcal{K}$, con vectores unitarios $dz_{i,j}$.

Definimos el operador $d : \mathcal{K} \to \mathcal{E}$ como

$$dF(\mathbf{z}_0, \ldots, \mathbf{z}_{\sigma-1}) = \sum_{\substack{i\leq \text{r} \\ l\leq \sigma-1}} \frac{\partial F(\cdot)}{\partial z_{i,l}} dz_{i,l}.$$

Considérese ahora el conjunto infinito de símbolos:

$$\wedge d\mathcal{C} = \{dz_{i,l} \wedge dz_{i',l'}, \text{ para } i, i' \geq 1, \;\; l, l' \in I\!N\}$$

y sea $\wedge\mathcal{E}$ el espacio vectorial generado sobre $\mathcal{K}$ por los elementos de $\wedge d\mathcal{C}$. Se define la relación de equivalencia siguiente:

$$d\alpha \wedge d\beta \underset{R}{=} -d\beta \wedge d\alpha,$$

por lo que:

$$dz_{i,l} \wedge dz_{i,l} = 0.$$

El espacio vectorial $\wedge\mathcal{E}$ mod R se denota por $\mathcal{E}^2$. Abusando de la notación se define nuevamente un operador $d : \mathcal{E} \to \mathcal{E}^2$ de la siguiente manera:

$$dv \;=\; d(\sum_{\substack{i\geq 1\\ l\geq 0}} F_{i,l}dz_{i,l}) = \sum_{\substack{i\geq 1\\ l\geq 0}} d(F_{i,l}) \wedge dz_{i,l} = \sum_{\substack{i,i'\geq 1\\ l,l'\geq 0}} \frac{\partial F_{i,l}}{\partial z_{i',l'}}\; dz_{i',l'} \wedge dz_{i,l}.$$

Los elementos de $\mathcal{E}^2$ se llaman *2-formas*. De manera similar se definen las s-formas para s > 2.

$\wedge : \mathcal{E}^{(p)}\ \mathcal{E}^{(q)} \to \mathcal{E}^{(p+q)}$ representa el producto exterior de una p-forma ω_p con una q-forma ω_q. Esta operación está dada por:

$$\omega_p \wedge \omega_q = \Big(\sum_{i=1}^{n_p} a_i\theta_i^p\Big)\Big(\sum_{j=1}^{n_q} b_j\theta_j^{(q)}\Big) = \sum_{i,j} a_ib_j\theta_i^{(p)} \wedge \theta_j^{(q)}.$$

Se define un operador $\delta : \mathcal{K} \to \mathcal{K}$:

$$\delta a(\mathbf{z}_0, \ldots \mathbf{z}_s) := a(\mathbf{z}_1, \ldots, \mathbf{z}_{s+1}). \tag{6}$$

En los trabajos de Márquez-Martínez (1999); Mounier y Rudolph (1998) se define un anillo de polinomios a partir del operador δ, el cual actúa sobre los elementos de $\mathcal{E}$ conmutando con el operador de derivación. Sin embargo, como se observa en el siguiente

ejemplo, se puede incurrir en contradicciones.

$$(\delta)(z_{1,0}dz_{1,0}) \quad = \quad z_{1,1}\delta dz_{1,0} \quad = \quad z_{1,1}d\delta z_{1,0} \quad = \quad z_{1,1}dz_{1,1}\delta. \tag{7}$$

Este último resultado no está definido.

Para evitar estos problemas, se introduce un nuevo operador $\nabla : \mathcal{E} \to \mathcal{E}$, definido por:

$$\nabla df(\mathbf{z}_0, \ldots \mathbf{z}_s) := df(\mathbf{z}_1, \ldots, \mathbf{z}_{s+1}). \tag{8}$$

El ejemplo de la Ecuación (7) se puede realizar sin ningún problema:

$$\nabla z_{1,0}dz_{1,0} = z_{1,1}dz_{1,1}.$$

II.3 El anillo $\mathcal{K}[\nabla]$

El operador ∇ permite introducir un anillo de polinomios cuyos coeficientes pertenecen a $\mathcal{K}$, el cual es la base del enfoque utilizado. Este anillo se denomina $\mathcal{K}[\nabla]$ y sus elementos son de la forma

$$a(\nabla) = \sum_{i=0}^{r_a} a_i \nabla^i, \qquad a_i \in \mathcal{K},$$

donde r_a es el grado polinomial de $a(\nabla)$. La suma y el producto están definidos en $\mathcal{K}[\nabla]$ por:

$$\begin{aligned} a(\nabla) + b(\nabla) \quad &:= \textstyle\sum_{i=0}^{max(r_a,r_b)} (a_i + b_i)\nabla^i, \\ a(\nabla)b(\nabla) \quad &:= \textstyle\sum_{i=0}^{r_a} \sum_{j=0}^{r_b} (a_i)(\delta^i b_j)\nabla^{i+j}. \end{aligned}$$

La relación entre los elementos hasta ahora definidos se muestra en la Figura 2, tomada de Márquez-Martínez y Moog (2007).

Defínase ahora el módulo a la izquierda generado por $\mathcal{K}[\nabla]$ sobre los elementos de

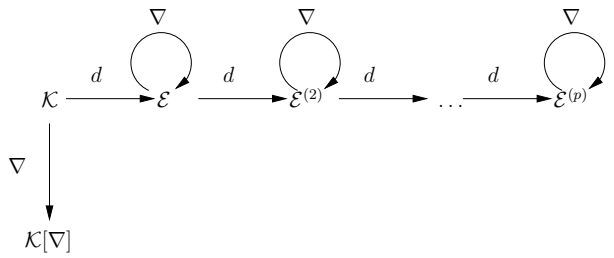


Figura 2: Relación entre los elementos matemáticos utilizados.

$\mathcal{E}$:

$$\mathcal{M} = \text{span}_{\mathcal{K}[\nabla]}\{d\xi \mid \xi \in \mathcal{K}\}.$$

Todo elemento de $\mathcal{M}$ corresponde también a $\mathcal{E}$ y viceversa. Aunque cabe aclarar que $\mathcal{M}$ y $\mathcal{E}$ sólo son iguales como conjuntos, pues tienen una estructura algebraica distinta. Considérese el siguiente ejemplo (Márquez-Martínez (2000)):

Sean dos vectores $\omega_1 := dx_1(t)$ y $\omega_2 := x_1(t)dx_1(t-1)$. Dado que $dx_1(t)$ es independiente sobre $\mathcal{K}$ de $dx_1(t-1)$, entonces $dim_{\{}\text{span}_{\mathcal{K}}\{\omega_1, \omega_2\}\} = 2$. En cambio, sobre $\mathcal{K}[\nabla]$ $\omega_2 = (x_1(t)\nabla)\omega_1$, por lo tanto, $dim\{span_{\mathcal{K}[\nabla]}\{\omega_1, \omega_2\}\} = 1$.

La introducción de este módulo permite relacionar 1-formas que, como se indicó en la Ecuación (3), son independientes sobre $\mathcal{K}$.

Los sistemas con retardos también pueden modelarse como un submódulo de $\mathcal{M}$ tomando la diferencial de $\dot{x}_0$, como se ve a continuación.

II.3.1 Sistema linealizado tangente

Considérese un sistema no lineal descrito por (2). A éste se le asocia el sistema

$$\Sigma_L : \begin{cases} d\dot{x}(t) &= F[\nabla]dx(t) + g[\nabla]du(t) \\ dy(t) &= H[\nabla]dx(t) \end{cases} \tag{9}$$

donde los elementos de $F[\nabla], g[\nabla]$ y $H[\nabla]$ pertenecen a $\mathcal{K}[\nabla]$, y están definidos por

$$\begin{aligned} F[\nabla] &= \sum_{i=0}^{s} \frac{\partial\Big[f(\cdot) + \sum_{j=0}^{s} g_j(\cdot)u(t-j)\Big]\nabla^i}{\partial[x(t-i)]} \\ g[\nabla] &= \sum_{j=0}^{s} g_j(\cdot)\nabla^j \\ H[\nabla] &= \sum_{i=0}^{s} \frac{\partial\Big[h(\cdot)\Big]\nabla^i}{\partial[x(t-i)]} \end{aligned}$$

Σ_L se conoce como el sistema linealizado tangente de Σ.

II.3.2 Propiedades

Una de las características más importantes de $\mathcal{K}[\nabla]$ es ser no conmutativo. Sin embargo, cuenta con otras propiedades convenientes para trabajar con sistemas con retardos. Entre ellas, posee un *algoritmo de división euclidiana* (Márquez-Martínez, 2000).

Teorema 1 (Algoritmo de división por la izquierda)**.** *Para cualquier* $a(\nabla), b(\nabla) \in \mathcal{K}[\nabla]$*,* $b[\nabla] \neq 0$*, existen dos polinomios únicos* $c(\nabla), r(\nabla) \in \mathcal{K}[\nabla]$*, tales que*

$$a(\nabla) = c(\nabla)b(\nabla) + r(\nabla) \text{ con g.p. } r(\nabla) < \text{g.p. } b(\nabla) \tag{10}$$

donde g.p. es el grado polinomial.

Sean dos polinomios $a(\nabla)$ y $b(\nabla) \in \mathcal{K}[\nabla]$. Utilizando el algoritmo de la página 20, se obtienen los polinomios $c_1(\nabla), r_1(\nabla) \in \mathcal{K}[\nabla]$, tales que se satisface (10). Entonces:

$$\begin{bmatrix} 1 & -c_1(\nabla) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a(\nabla) \\ b(\nabla) \end{bmatrix} = \begin{bmatrix} r_1(\nabla) \\ b(\nabla) \end{bmatrix}.$$

Nótese que la matriz de la izquierda es invertible. Dividiendo ahora $b(\nabla)$ entre $r_1(\nabla)$:

$$\begin{bmatrix} 1 & 0 \\ -c_2(\nabla) & 1 \end{bmatrix} \begin{bmatrix} r_1(\nabla) \\ b(\nabla) \end{bmatrix} = \begin{bmatrix} r_1(\nabla) \\ r_2(\nabla) \end{bmatrix},$$

donde el *g.p.* $r_2(\nabla) <$ *g.p.* $r_1(\nabla)$. Repitiendo el procedimiento se obtiene:

$$\begin{bmatrix} P_1(\nabla) & Q_1(\nabla) \\ P_2(\nabla) & Q_2(\nabla) \end{bmatrix} \begin{bmatrix} a(\nabla) \\ b(\nabla) \end{bmatrix} = \begin{bmatrix} d(\nabla) \\ 0 \end{bmatrix} \quad , \tag{11}$$

donde:

- El polinomio $d(\nabla)$ es el máximo común divisor (MCD) por la izquierda.

- Las matrices $\begin{bmatrix} 1 & -c_1(\nabla) \\ 0 & 1 \end{bmatrix}$ y $\begin{bmatrix} 1 & 0 \\ -c_i(\nabla) & 1 \end{bmatrix}$, con $i \in I\!N \geq 2$, son invertibles.

- La matriz $\begin{bmatrix} P_1(\nabla) & Q_1(\nabla) \\ P_2(\nabla) & Q_2(\nabla) \end{bmatrix}$, al ser producto de matrices invertibles, también cuenta con esta propiedad.

Lema 1. *(Márquez-Martínez y Moog, 2007). Para cualquier matriz* $A[\nabla] \in \mathcal{K}^{m\times n}[\nabla]$ *existe al menos una matriz unimodular* $P[\nabla] \in \mathcal{K}^{m\times m}[\nabla]$ *tal que*

$$P[\nabla]A[\nabla] = \begin{bmatrix} \bar{A}[\nabla] \\ 0 \end{bmatrix}, \tag{12}$$

donde $\bar{A}[\nabla] \in \mathcal{K}^{r\times n}[\nabla]$ *tiene rango completo por renglones.*

En ciertos casos es posible encontrar dos polinomios $c'(\nabla), r'(\nabla) \in \mathcal{K}[\nabla]$, tales que

$$a(\nabla) = b(\nabla)c'(\nabla) + r'(\nabla) \text{ con g.p. } r'(\nabla) < \text{g.p. } b(\nabla). \tag{13}$$

Si estos polinomios existen, esta operación se denomina *división euclidiana por la derecha.*

En Xia *et al.* (2002) se ha probado que además de ser un anillo euclidiano $\mathcal{K}[\nabla]$ también es un *anillo izquierdo de Ore.*

Teorema 2. *Para cualquier* $a(\nabla), b(\nabla) \in \mathcal{K}[\nabla]$, *existen polinomios no ambos nulos* $p(\nabla), q(\nabla) \in \mathcal{K}[\nabla]$ *tales que*

$$p(\nabla)a(\nabla) + q(\nabla)b(\nabla) = 0. \tag{14}$$

Algoritmo 1: División de Euclides por la izquierda sobre $\mathcal{K}[\nabla]$

Data: Condiciones iniciales

$$c(\nabla) \leftarrow 0$$
$$r(\nabla) \leftarrow a(\nabla)$$

begin
  Sea l=g.p.$r(\nabla)$-g.p.$b(\nabla)$.
  if $l < 0$ **then**
    Fin del algoritmo
  else

$$p(\nabla) \leftarrow \frac{\bar{r}}{\delta^l \bar{b}} \nabla^l$$

donde las funciones $\bar{r}$ y $\bar{b}$ son los coeficientes líderes de $r(\nabla)$ y $b(\nabla)$ respectivamente. Nótese que $\delta^l \bar{b} \in \mathcal{K}$.

$$c(\nabla) \leftarrow c(\nabla) + p(\nabla)$$
$$r(\nabla) \leftarrow r(\nabla) - p(\nabla) b(\nabla)$$

end

En la Ecuación (11) los polinomios $P_2(\nabla)$ y $Q_2(\nabla)$ representan a $p(\nabla)$ y $q(\nabla)$ respectivamente y se denominan *polinomios de Ore*. Estos polinomios no son ambos nulos ya que forman un renglón de una matriz invertible, sin embargo no son únicos.

Teorema 3. *Sean dos elementos $a(\nabla), b(\nabla) \in \mathcal{K}[\nabla]$ primos entre ellos por la izquierda. Entonces, existen dos polinomios $p(\nabla), q(\nabla) \in \mathcal{K}[\nabla]$ tales que*

$$p(\nabla)a(\nabla) + q(\nabla)b(\nabla) = 1. \tag{15}$$

La prueba se deriva de las ecuaciones (10) y (11). Si en esta última $p(\nabla) = P_1(\nabla)$, $q(\nabla) = Q_1(\nabla)$ y MCD, $d(\nabla) = 1$, entonces $p(\nabla)$ y $q(\nabla)$ se denominan *polinomios de Bezout*.

Corolario 1. *Se definen $a(\nabla)$ y $b(\nabla)$ de igual manera que en el anterior teorema.*

Entonces para todo $f(\nabla) \in \mathcal{K}[\nabla]$*, existen dos coeficientes* $\alpha(\nabla), \beta(\nabla) \in \mathcal{K}[\nabla]$ *tales que*

$$\alpha(\nabla)a(\nabla) + \beta(\nabla)b(\nabla) = f(\nabla).$$

El equivalente al Teorema 3 no es válido para los polinomios primos a la derecha, salvo en casos particulares.

II.3.3 Integrabilidad

La noción de integrabilidad para los casos con y sin retardos es la siguiente:

Definición 2 (Integrabilidad)**.** Un submódulo Ω con vectores base $\{\omega_1, \ldots, \omega_s\}$ es integrable si existen s funciones $\varphi_i \in \mathcal{K}$ tales que, localmente:

$$\text{span}_{\mathcal{K}[\nabla]}\{\omega_1, \ldots, \omega_s\} = \text{span}_{\mathcal{K}[\nabla]}\{d\varphi_1, \ldots, d\varphi_s\}.$$

En los sistemas sin retardos, esta propiedad se calcula mediante el teorema de Frobenius, sin embargo, para el caso con retardos este teorema ya no es válido cuando Ω se expande por más de una 1-forma, como se explica a continuación (Márquez-Martínez y Moog, 2007).

Proposición 2 (Cambio de base)**.** *Sea* $A \subset \mathcal{M}$ *con vectores base* $\omega = \{\omega_1, \ldots, \omega_s\}$*. Entonces* $\bar{\omega} = \{\bar{\omega}_1, \ldots, \bar{\omega}_s\}$ *es también una base de* A *si y sólo si existe una matriz* $T[\nabla] \in \mathcal{K}^{s\times s}[\nabla]$ *tal que*

$$\omega = T[\nabla]\bar{\omega}$$

donde $\omega := [\omega_1 \ldots \omega_s]^T$ *y* $\bar{\omega} := [\bar{\omega}_1 \ldots \bar{\omega}_s]^T$*.*

Partiendo de esta proposición Ω es integrable si y solamente si existen $P', P \in \mathcal{K}^{s\times s}[\nabla]$ tal que $P\omega = d\varphi$ y $P'd\varphi = \omega$.

Cuando un submódulo Ω es expandido por una 1-forma ω, $P \in \mathcal{K}$ por lo que es escalar y unimodular, entonces la integrabilidad de Ω puede ser verificada mediante el siguiente caso particular del Teorema de Frobenius:

Teorema 4. *Una 1-forma* ω *es integrable si y solamente si*

$$d\omega \wedge \omega = 0. \tag{16}$$

El problema de cuando un submódulo expandido por varias 1-formas independientes es integrable, continúa aún sin resolver, es por ello que el reto de las investigaciones más recientes ha sido establecer la integrabilidad, al menos para algunos casos especiales. En el trabajo de Márquez-Martínez y Moog (2007) se encuentran los resultados siguientes:

Teorema 5. *Cualquier submódulo* $A = \text{span}_{\mathcal{K}[\nabla]}\{\omega_1, \ldots, \omega_s\} \in \mathcal{M}$ *es integrable si existe* $k \in \mathbb{N}$ *tal que*

$$d\omega_i \wedge \omega_i \wedge \left(\wedge_{\substack{j=1\ldots i-1 \\ l=1\ldots k}} \nabla^l \omega_j \right) = 0, \quad \forall i = 1, \ldots, s. \tag{17}$$

Corolario 2. *Un submódulo* $\Omega := \text{span}_{\mathcal{K}[\nabla]}\{d\varphi_1, \ldots, d\varphi_s, \bar{\omega}\}$ *es integrable si*

$$d\bar{\omega} \wedge \bar{\omega} \wedge d\bar{\varphi} \wedge \cdots \wedge \nabla^k d\bar{\varphi}, \quad d\bar{\varphi} := d\varphi_1 \wedge \cdots \wedge d\varphi_s. \tag{18}$$

La condición (17) no es necesaria en general. Sin embargo, la condición (18) es necesaria y suficiente en el siguiente caso:

Teorema 6. *Sea* $g_i^{\perp}$ *el aniquilador por la izquierda de una matriz* $g_i[\nabla] \in \mathcal{K}^{n\times m}[\nabla]$, *con* $i \in \mathbb{N}$, *y sean* $A = g_1^{\perp}$ *y* $B = g_2^{\perp}$ *dos submódulos de* $\mathcal{M}$, *con* $A \subset B$ *y* $rango B = rango A + 1$. *Si* A *es integrable, entonces* B *también es integrable, si y sólo si, se cumple (18), con* $A = \text{span}_{\mathcal{K}[\nabla]}\{d\varphi_1, \ldots, d\varphi_s\}$.

II.3.4 Matrices con elementos en $\mathcal{K}[\nabla]$

En esta sección se abordan conceptos básicos de matrices cuyos elementos pertenecen a $\mathcal{K}[\nabla]$, ya que el trabajar con el enfoque algebraico implica el uso de este tipo de elementos matemáticos.

Definición 3 (Rango)**.** El rango por renglones (por columnas) en el anillo $\mathcal{K}[\nabla]$ de una matriz A es el número máximo de renglones (columnas) linealmente independientes a la izquierda (a la derecha) en el anillo.

El rango por columnas y por renglones es distinto en el caso bajo estudio. Las matrices con elementos en $\mathcal{K}[\nabla]$ no poseen determinante, por lo que una matriz unimodular se define de la siguiente manera:

Definición 4 (Matriz unimodular)**.** Una matriz $A \in \mathcal{K}^{n\times n}[\nabla]$ es unimodular si admite una inversa $A' \in \mathcal{K}^{n\times n}[\nabla]$.

Forma de Smith

Toda matriz $A \in I\!R^{n\times m}$, de rango r, definida en un anillo conmutativo R, puede ser llevada a la forma de Smith. Sin embargo, en el caso de un anillo no conmutativo como $\mathcal{K}[\nabla]$ no siempre existe una forma de Smith cuyos elementos sean causales, por lo que se debe tener precaución.

$$Smith(A): \begin{cases} \begin{bmatrix} \triangle & \vdots & 0 \end{bmatrix}, & si \quad r < m \\ \begin{bmatrix} \triangle \end{bmatrix}, & si \quad r = m \\ \begin{bmatrix} \triangle \\ \dots \\ 0 \end{bmatrix} & si \quad r > m \end{cases}$$

con

$$\triangle = \begin{bmatrix} s_1 & & & & & \\ & \ddots & & & & (0) \\ & & s_r & & & \\ & & & 0 & & \\ (0) & & & & \ddots & \\ & & & & & 0 \end{bmatrix}$$

donde s_i divide s_{i+1}, $i = 1, \dots, r-1$. Los polinomios s_i se nombran polinomios invariantes de A.

La forma de Smith se obtiene con la ayuda de tres operaciones elementales, las cuales no alteran el valor del rango de la matriz A.

1. Permutación de dos líneas o dos columnas.

2. Suma de un múltiplo de una línea o columna a otro.

3. Multiplicación de una línea o columna por un elemento no nulo.

Después de aplicar el Algoritmo 2, que se encuentra en la página 26, el resultado son tres matrices:

$$P \cdot A \cdot Q = S,$$

donde P y Q son invertibles al ser formadas por operaciones con matrices elementales y S es la forma de Smith.

La forma de Smith es una herramienta importante para probar si un submódulo es cerrado sobre $\mathcal{K}[\nabla]$. En la Sección II.3 se vió que el módulo $\mathcal{M}$ es cerrado, sin embargo un submódulo de $\mathcal{M}$ puede no tener esta propiedad:

Definición 5 (Cerradura de un submódulo)**.** Sea $\mathcal{M}$ un módulo sobre el anillo $\mathcal{K}[\nabla]$ y S el conjunto de elementos no nulos del anillo. La cerradura de un submódulo $\mathcal{N}$ de $\mathcal{M}$ es el conjunto $\bar{\mathcal{N}}$ definido por

$$cls_{\mathcal{K}[\nabla]}\mathcal{N} := \bar{\mathcal{N}} := \{x \in \mathcal{M} \text{ tal que } \exists s \in S; sx \in \mathcal{N}\}$$

Si $\mathcal{N}$ y $\bar{\mathcal{N}}$ coinciden se dice que $\mathcal{N}$ es cerrado.

La cerradura de un módulo $\mathcal{N}$ tiene las siguientes propiedades:

- Es único.

- Es el mayor submódulo de $\mathcal{M}$ que contiene a $\mathcal{N}$ y tiene igual rango.

Si los polinomios invariantes de la forma de Smith son de grado cero, se asegura que los submódulos que son el kernel izquierdo de una matriz $\mathcal{K}^{n\times m}[\nabla]$ son cerrados, y por lo tanto cuentan con las propiedades anteriormente descritas (Márquez-Martínez y Moog, 2007).

Lema 2. *Sea* $\Omega = \text{span}_{\mathcal{K}[\nabla]}\{T[\nabla]dx\}$ *el aniquilador por la izquierda de una matriz* $g[\nabla] \in \mathcal{K}^{n\times m}[\nabla]$ *(que será denotado por* $\Omega := g^{\perp}$*). Entonces,* $T[\nabla]$ *admite una forma de Smith, siendo todos sus polinomios invariantes escalares.*

Demostración (Márquez-Martínez y Moog, 2007)

Sea $T[\nabla]$ una matriz de rango completo k y una base para $g^{\perp}$. El Lema 1 indica que existe $P[\nabla] \in \mathcal{K}^{n\times n}$ unimodular tal que

$$P[\nabla]g[\nabla] = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} g[\nabla] = \begin{bmatrix} \bar{g} \\ 0 \end{bmatrix}$$

donde la matriz $\begin{bmatrix} p_{21} & p_{22} \end{bmatrix}$, de dimensión $k \times n$, es una base para $g^{\perp}$. La Proposición 2 muestra que existe una matriz unimodular $P'[\nabla] \in \mathcal{K}^{k\times k}[\nabla]$ tal que

$$\begin{aligned} T[\nabla] &= P'[\nabla] \begin{bmatrix} p_{21} & p_{22} \end{bmatrix} \\ &= P'[\nabla] \begin{bmatrix} 0 & \mathbb{I}_k \end{bmatrix} P[\nabla] \\ &= P'[\nabla] \begin{bmatrix} 0 & \mathbb{I}_k \end{bmatrix} \begin{bmatrix} 0 & \mathbb{I}_{n-k} \\ \mathbb{I}_k & 0 \end{bmatrix} \begin{bmatrix} 0 & \mathbb{I}_k \\ \mathbb{I}_{n-k} & 0 \end{bmatrix} P[\nabla] \\ &= P'[\nabla] \begin{bmatrix} \mathbb{I}_k & 0 \end{bmatrix} \begin{bmatrix} 0 & \mathbb{I}_k \\ \mathbb{I}_{n-k} & 0 \end{bmatrix} P[\nabla] \end{aligned}$$

Para terminar la demostración, se define $Q'[\nabla] = \begin{bmatrix} 0 & \mathbb{I}_k \\ \mathbb{I}_{n-k} & 0 \end{bmatrix} P[\nabla]$ y se escribe

$$T[\nabla] = P'[\nabla] \begin{bmatrix} \mathbb{I}_k & 0 \end{bmatrix} Q'[\nabla]$$

Lo cual prueba que $T[\nabla]$ es (i.e. su forma de Smith es) equivalente a una matriz identidad.

II.4 Resumen

En este capítulo se presentaron las herramientas matemáticas que se utilizan para el análisis y control de SNLR, las cuales sirven de base para el desarrollo de las rutinas de cálculo simbólico elaboradas en este trabajo de tesis. En el capítulo siguiente se detalla el proceso de diseño del software.

Algoritmo 2: Forma de Smith con elementos en $K[\nabla]$

Data: Condiciones iniciales
Sea una matriz $A \in \mathcal{K}[\nabla]^{n\times m}$ con elementos $a_{(i,j)} \in A$ donde $i = \{1,\dots,n\}$, $j = \{1,\dots,m\}$
begin
  $P[\nabla] \leftarrow \mathbb{I}_n$
  $Q[\nabla] \leftarrow \mathbb{I}_m$
  for *i=1* ***to*** n **do**

$$M \leftarrow \begin{bmatrix} a_{(i,i)} & \dots & a_{(i,m)} \\ \vdots & \ddots & \vdots \\ a_{(n,i)}, & \dots & a_{(n,m)} \end{bmatrix}$$

    $a_{(i,i)} \leftarrow$ Primer elemento de menor g.p. encontrado en M y asignado mediante permutaciones.
    Con la finalidad de hacer cero otros elementos de la columna i:
    for *k=n-i* ***to*** n **do**
      $p_1, q_1 \leftarrow$ Polinomios de Ore por la izquierda, véase Ecuación (10), con $a(\nabla) = a_{(k,i)}$ y $b(\nabla) = a_{(i,i)}$.
      $P_k[\nabla] \leftarrow$ Matriz con polinomios p_1 y q_1 ubicados de modo que:

$$a_{(k,i)} = 0.$$

$$P[\nabla] \leftarrow P_k[\nabla]P[\nabla]$$

    Con la finalidad de hacer cero otros elementos del renglón i:
    for *r=m-i* ***to*** m **do**
      $p_2, q_2 \leftarrow$ Polinomios de Ore por la derecha, véase Ecuación (13), con $a(\nabla) = a_{(i,r)}$ y $b(\nabla) = a_{(i,i)}$.
      $Q_k[\nabla] \leftarrow$ Matriz con polinomios p_2 y q_2 ubicados de modo que:

$$a_{(i,r)} = 0.$$

$$Q[\nabla] \leftarrow Q[\nabla]Q_k[\nabla]$$

  La forma de Smith (S) se encuentra:
  $S \leftarrow P[\nabla]\, A\, Q[\nabla]$
end

Capítulo III

Proceso de construcción del software

El desarrollo de un sistema es una progresión gradual de las ideas iniciales del cliente acerca del problema y tienen que superarse varias etapas antes de ver el software funcional. Sin embargo, es difícil lograr un programa de calidad sin una metodología y ya que el papel del software en sistemas de todo tipo ha cobrado importancia, también ha aumentado el interés en el desarrollo de tecnologías que hagan más fácil, más rápida y menos cara su construcción y mantenimiento. Así es como surge la ingeniería de software.

En este capítulo se abordan las etapas de análisis y diseño del programa, donde se muestran los requerimientos obtenidos y los diagramas de casos de uso generados a partir de estos requerimientos; además del procedimiento y herramientas utilizadas para modelar el sistema.

III.1 Análisis del programa

El análisis representa una de las fases más importantes en el desarrollo de un proyecto, en esta fase es necesario obtener o descubrir las necesidades de los usuarios finales. Estas necesidades en el área de software se conocen como requerimientos.

Según Pressman (2005), el primer paso para identificar los requerimientos es preguntar a los clientes por sus necesidades o cómo el sistema o producto va a ser utilizado. En el caso de SAC los clientes directos son el grupo de personas que estudian o trabajan sobre sistemas de control no lineal con o sin retardos.

III.1.1 Requerimientos del sistema

Después de varias sesiones de trabajo, se han establecido los requerimientos que, de acuerdo a las necesidades del cliente, debe cumplir SAC. Estos se agrupan en tres categorías:

Requerimientos de información: describen los datos que el sistema necesita para realizar su función.

- *Datos de entrada del sistema.* La información de entrada son las ecuaciones matemáticas que modelan cada sistema de control, las cuales se especifican por ecuaciones de estado de la forma establecida en la Sección II.2.

  Existen tres formas de representación en que el usuario puede ingresar las ecuaciones de estado:

  afín (ver Ecuación (2)),

  general (ver Ecuación (1)),

  diferencial (ver Ecuación (9)).

  El sistema debe ser capaz de calcular las tres para cada sistema de ecuaciones, en caso de que el modelo lo permita.

Requerimientos de interfaz: El programa Maxima es el elegido para desarrollar SAC, ya que es el que se adapta mejor a los siguientes requisitos establecidos por el usuario:

- *Software de manejo simbólico (CAS).* Las herramientas matemáticas utilizadas para analizar sistemas con retardos requieren operaciones algebraicas.

- *Programa de distribución libre.* SAC no se realiza con fines comerciales y se cree que el uso de la licencia GNU GPL facilitará la colaboración entre la comunidad de usuarios, con el fin de mejorar el programa.

- *Multiplataforma*. Se pide que el sistema funcione como mínimo en los sistemas operativos Linux y Windows®.

- *Estructura modular*. El software se divide en componentes con nombres independientes, que es posible abordar en forma individual. Además de practicar la reusabilidad de código, cada uno de estos módulos es una nueva instrucción que se suma al lenguaje de programación existente, de modo que al avanzar en la construcción del sistema se simplifica el diseño de las rutinas.

- *Código estable*. Se refiere a la necesidad de que el programa de cálculo simbólico sea compatible con versiones anteriores y futuras del mismo, es decir, que no sufra cambios significativos en su modo de operación al cambiar de versión.

- *Fácil manejo*. Que su modo de operación sea similar a otros paquetes especializados en el área de control, de modo que el usuario se encuentre familiarizado con el entorno del programa. Esto lleva a definir los conocimientos mínimos que debe tener un usuario de SAC:

  conocimientos de teoría de control no lineal con o sin retardos,

  conocimientos de computación,

  conocimientos básicos en el uso de Maxima.

Requerimientos funcionales: describen los servicios que se espera que el sistema provea.

- *Las operaciones básicas de entrada/salida que el sistema debe ser capaz de realizar son:*

  guardar las ecuaciones de estado de un sistema de control,

  cargar las ecuaciones de estado de un sistema de control.

- *Capaz de manejar varios sistemas de ecuaciones en la misma sesión.* Este requerimiento se establece debido a que algunos problemas de control, como la síntesis de observadores por ejemplo, se pueden tratar como la unión de dos sistemas independientes.

- *En esta primera versión de SAC los problemas de análisis considerados son los siguientes:*

  accesibilidad,

  observabilidad,

  equivalencia con la forma triangular,

  linealización por inyecciones aditivas entrada/salida,

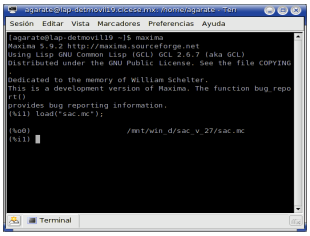
  rechazo de perturbaciones (parcialmente).

III.1.2 Lenguaje simbólico utilizado

Existen varios paquetes de cálculo simbólico, entre los más populares encontramos a Mathematica®, Maple®, muPAD®, REDUCE, YACAS y Maxima. Sin embargo, al revisar estos programas, se eligió a este último para desarrollar SAC por ser el que se adapta mejor a los requisitos establecidos por el cliente.

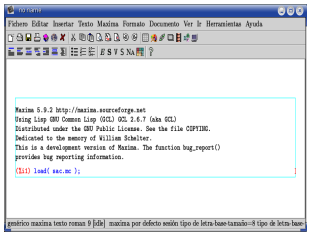
Maxima se rige bajo licencia GNU GPL, por lo que el usuario tiene, entre otros beneficios, la libertad de usar el programa con cualquier propósito, de estudiar cómo funciona y adaptarlo a sus necesidades, de mejorar el programa y hacer públicas las mejoras para que toda la comunidad se beneficie, además de poder distribuir copias de su trabajo. Otra característica interesante es que permite escribir rutinas en LISP, que es un lenguaje especializado en procesamiento de listas, por lo que algunos programas de SAC se han escrito en dicho lenguaje y otros en Maxima aprovechando las ventajas que cada uno ofrece.

Maxima es multiplataforma, trabaja con los sistemas operativos Windows, Linux y MacOS X. En la Figura 3 se muestran algunas de las interfaces, todas funcionan en Linux. Existen entornos como Xmaxima, Wxmaxima y Texmacs, los cuales tienen versiones disponibles para Windows®, aclarando que este último programa es un paquete independiente que permite incluir en sus sesiones a varios programas, entre ellos a Maxima. También es importante mencionar que Maxima es un proyecto vigente, el cual se actualiza constantemente y que un grupo de voluntarios brindan soporte técnico.

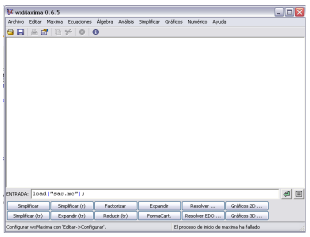


(a) Consola

(b) Xmaxima



(c) Texmacs



(d) Wxmaxima

Figura 3: Maxima como interfaz entre SAC y el usuario.

SAC es un conjunto de bibliotecas, por lo que sus requerimientos dependen directamente de los de Maxima. Estas bibliotecas ocupan 364Kb de espacio en disco duro. Para la generación de SAC se empleó la versión de Maxima 5.9.2, la cual requiere un entorno common-lisp, que también puede obtenerse accesando a la página del proyecto[1].

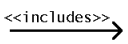
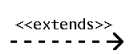
[1]http://maxima.sourceforge.net/

A continuación los requerimientos del sistema se modelan en diagramas UML (Lenguaje Unificado de Modelado). El modelado del análisis con UML comienza con la creación de escenarios que, como se indica en la siguiente sección, sirven para entender la manera en que los usuarios finales y otros actores quieren interactuar con el sistema (Pressman, 2005).

III.1.3 Casos de uso

Un caso de uso describe las interacciones entre el sistema y alguien o algo que utiliza una función o servicio, cada uno de estos casos es una función específica del programa. Esta herramienta es útil en el análisis de un proyecto, ya que permite definir junto con el usuario los servicios, independientemente de las operaciones efectuadas dentro del sistema. En la Tabla I aparecen los elementos que conforman un diagrama de caso de uso, esta representación gráfica ayuda al diseñador junto con su cliente a entender mejor los principales aspectos de todos los casos que afectan al diseño y a elegir qué implementar en cada fase, su utilidad aumenta cuando se tiene un número considerable de casos de uso y es difícil situarlos y relacionarlos (Bennet *et al.*, 2004).

Tabla I: Elementos de un diagrama de casos de uso.

	Elementos
	**Actor** Es el rol que juega un usuario, no representa a una persona en particular, sino la labor que realiza al interactuar con el sistema.
	**Caso de uso** Es una operación/tarea específica que se realiza tras una orden de algún agente externo, ya sea un actor o bien otro caso de uso.
 	**Relaciones entre casos de uso** Pueden ser de dos tipos: 1. Relaciones ≪includes≫. El caso de uso al ser ejecutado *siempre* invoca otro caso de uso. 2. Relaciones ≪extends≫. El caso de uso al ser ejecutado *en ocasiones* invoca otro caso de uso.

En SAC cada función puede considerarse un caso de uso, ya que como se ha mencionado anteriormente, se busca que cada instrucción sea una rutina útil al usuario y a los otros programas. Sin embargo, los casos principales se muestran en la Figura 4.

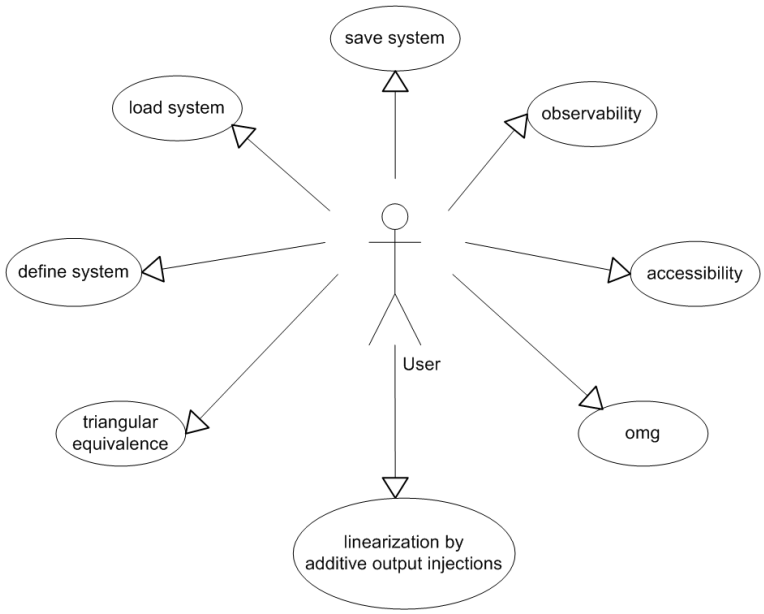


Figura 4: Principales casos de uso en SAC.

El diagrama más importante de esta técnica es el que describe el caso de uso, un formato se muestra en la Tabla II, este documento explica la forma de interactuar entre el sistema y el actor.

Cabe aclarar que el idioma de la documentación utilizada es inglés debido a que se busca que SAC sea un proyecto de código abierto (open source) y eso facilita para que investigadores extranjeros puedan colaborar. Enseguida se muestra el desarrollo de uno de los ocho casos de uso que conforman SAC. Los documentos completos de análisis y diseño del programa se encuentran en el Apéndice A.

Tabla II: Documento de caso de uso.

Caso de uso:	Nombre asignado al caso de uso
Actores:	Especifica si el actor es una persona o un proceso
Propósito:	Indica el objetivo del caso de uso.
Description:	Describe brevemente la interacción actor-sistema.
ACCIÓN DEL ACTOR	RESPUESTA DEL SISTEMA
1. El actor inicia una acción	2. El sistema responde con una primera acción.
	3. Segunda acción del sistema.
4. El actor realiza otra acción	5. El sistema vuelve a responder...

Caso de uso "accesibilidad"

En la Figura 5 se muestra el desarrollo del caso de uso "accesibilidad" y la manera en que se relaciona con otros casos de uso. Al calcular la accesibilidad siempre se ejecuta el caso de uso que obtiene los subespacios $\mathcal{H}_{\mathcal{K}}$, por lo que el tipo de relación de uso es $\ll$*includes*$\gg$, y éste a su vez ejecuta otros casos de uso.

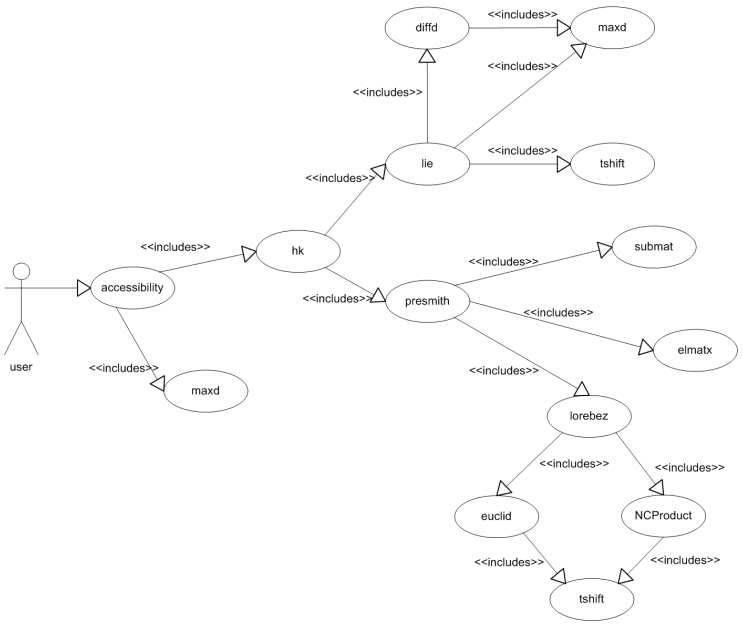


Figura 5: Caso de uso "accesibilidad".

Existe un documento de caso de uso similar al siguiente, por cada uno de los casos de uso mencionados en la Figura 5:

Use case: accessibility.

Actors: user

Purpose: identify if a nonlinear control system with or without delays is accessible.

Description: The user defines the system equations and the routine answers are *true*, *false* or *cannot determinate*.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. isaccessible(system_name)	2. Computes the $\mathcal{H}_{\mathcal{K}}$ submodules.
	3. Verify $\mathcal{H}_{\infty}$:
	- if $\mathcal{H}_{\infty} = 0$ the system is accessible and the answer is *true*;
	- if $\mathcal{H}_{\infty} \neq 0$ and the system does not have delays the answer is *false*;
	- if $\mathcal{H}_{\infty} \neq 0$ and the system has delays there are not enough conditions to determinate if the system is accessible and the routine returns *cannot determinate*.

El diagrama de caso de uso anterior corresponde al caso de uso principal, $\ll$*accessibility*$\gg$. En lo que respecta al desarrollo de SAC, el utilizar esta clase de diagramas ayudó a elegir los módulos que conformarían el programa, así como la interacción entre el usuario y cada una de las rutinas. Sin embargo, los casos de uso son muy limitados para especificar el comportamiento interno de un sistema, por lo tanto, se deben complementar con otras herramientas e ir diseñando con su ayuda cada una de las funciones del programa.

III.2 Diseño del programa

En esta etapa se define la arquitectura del software y se utilizan técnicas de la ingeniería del software que permiten al desarrollador entender mejor el funcionamiento del programa. Las primeras tareas son definir la arquitectura del software.

III.2.1 Arquitectura

La arquitectura de un programa o sistema de cómputo es la estructura del sistema, que incluye los componentes del software, sus propiedades visibles y las relaciones entre ellos (Bass *et al.* (2003)).

Entre las principales características de SAC, Sección III.1.1, se menciona su *estructura modular*, esto es, se practica la *reutilización de código* permitiendo que los programas sean claros, pequeños y ágiles. Este principio es muy útil, ya que una rutina con muchas líneas de código es difícil de leer, difícil de probar y difícil de depurar.

El tener una estructura modular no sólo es formar subrutinas para reutilizar código, cada subrutina o módulo es una nueva instrucción que se suma al lenguaje de programación existente. Cuando se escriben dos o más rutinas, algunos módulos de la primera serán útiles en la segunda y, conforme se avanza en la construcción del sistema, se simplifica el diseño de otras partes del programa y por tanto el esfuerzo y tiempo invertido en su realización disminuye.

Otra ventaja de este tipo de estructura es que ayuda al desarrollador a clarificar las ideas acerca del diseño del programa; si dos componentes de una rutina son similares será más fácil percibir esta similitud y rediseñar los programas. Sin embargo, la labor de identificar y separar las rutinas no es una tarea sencilla y para que se facilite esta labor se utilizan herramientas de la ingeniería de software.

III.2.2 Técnicas de estructuración de programas

Existen varias técnicas de estructuración que nos ayudan a modelar el sistema, su propósito principal es mejorar la comunicación entre el desarrollador y el cliente, además de reducir su complejidad. Cada una de estas herramientas se concentra en ciertos aspectos específicos del programa e ignora otros, es decir, modela parcialmente y para obtener una perspectiva completa se requiere utilizar varias técnicas. Las técnicas de

modelado utilizadas en el diseño de SAC se muestran en la Tabla III (Britton y Doake, 1993).

Tabla III: Herramientas de ingeniería de software utilizadas.

Técnica de modelado	Perspectiva
Diagramas de flujo	Flujo de datos o información a través del programa.
Diccionario de datos	Detalles acerca del contenido de los datos.
Árboles de decisión	Expresan la lógica de los procesos.

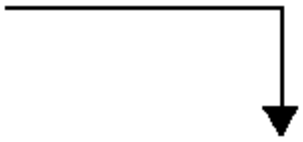
Diagramas de flujo de datos

Estos diagramas identifican el progreso de la información a través del programa y los procesos que la afectan, delimitan la frontera del sistema y además permiten descubrir inconsistencias u omisiones en los datos en una etapa donde este tipo de errores puede ser corregido fácilmente (Britton y Doake, 1993).

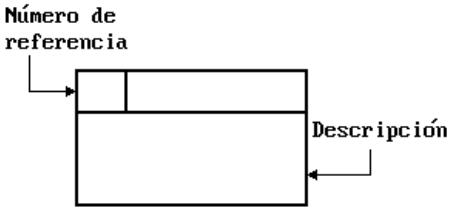
Aún cuando un sistema es pequeño puede tener cientos de procesos, para ordenar esta información de manera clara y coherente se establecen niveles en los diagramas de flujo. Se comienza por el diagrama de nivel 0, posteriormente se expande este diagrama en varias etapas hasta detallar todo el proyecto. La Figura 6, página 38, ilustra la notación básica de un diagrama de flujo.

Los diagramas de flujo de datos son limitados en la información que muestran, no modelan la secuencia en la que ocurren los procesos, ni el tiempo que duran, tampoco detalles acerca de la estructura de los flujos de datos o datos almacenados ni condiciones que gobiernan la ocurrencia de ciertos eventos. Para compensar algunos de estos aspectos SAC utiliza dos herramientas que son el diccionario de flujo de datos y los árboles de decisión, los cuales se explican a continuación.

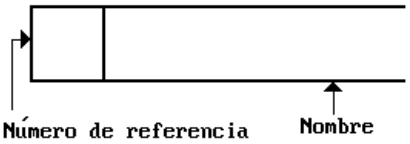
(a) **Entidad externa.** Usuarios o procesos que envían o reciben datos del sistema, pero no son considerados parte de él.



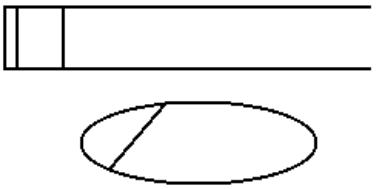
(b) **Flujo de datos.** Modelan el flujo de información dentro del sistema.



(c) **Procesos.** Un proceso modela una acción que transforma a los datos.



(d) **Datos almacenados.** Representa datos que permanecen en memoria para el correcto funcionamiento del sistema.



(e) Objetos con las esquinas cortadas indican que son usados más de una vez en el diagrama.

(f) **Frontera.** Delimita el sistema

Figura 6: Notación básica de un diagrama de flujo.

Árboles de decisión

Expresan la lógica de los procesos que realizan algunas actividades sólo si ciertas condiciones se cumplen. Como se muestra en la Figura 7 el enfoque del árbol de decisión hace posible observar, al menos las principales alternativas y el hecho de que las decisiones posteriormente dependan de acontecimientos en el futuro (Britton y Doake, 1993).

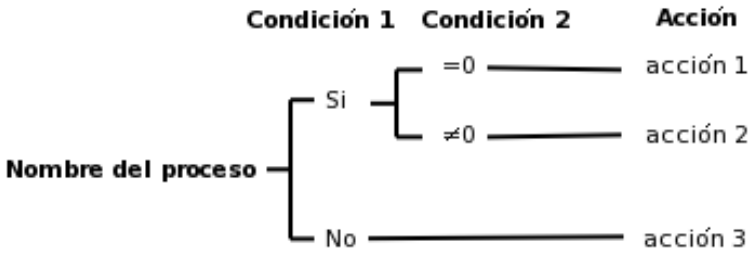


Figura 7: Estructura de un árbol de decisión.

Diccionario de datos

Son especificaciones técnicas detalladas de los datos que fluyen y se almacenan en el programa así como de los procesos que lo conforman. Esta técnica permite describir el desarrollo del sistema en los diagramas por nombres simples y legibles, de manera que cuando el cliente requiere detalles de alguna etiqueta puede buscar en el diccionario de datos. Uno de los mayores beneficios es que resuelve problemas de comunicación cuando varias personas trabajan en un mismo proyecto. La Tabla IV muestra la notación utilizada en el diccionario (Britton y Doake, 1993).

Tabla IV: Notación del diccionario de datos.

Función	**Símbolo**	**Descripción**
Definición	=	El elemento a la izquierda *consiste de* los elementos a la derecha del símbolo.
Secuencia	+	Une las propiedades del objeto de datos (atributos).
Repetición	{ }	Encierra los atributos repetidos.
Opcionales	( )	Encierra los atributos opcionales.
Selección	[ ]	Encierra las opciones de un atributo alternativo.
Separador	|	Separa las alternativas encerradas entre corchetes.
Valor	” ”	Los valores que toma un atributo son dados entre comillas.
Comentario	*. . . *	Comentarios entre asteriscos.

Herramientas de diseño aplicadas al problema de accesibilidad

En la Figura 8 se muestra el diagrama de flujo para el problema de accesibilidad. Este diagrama muestra la interacción con el usuario, es decir, los datos que éste tiene que proporcionar y la respuesta del sistema.

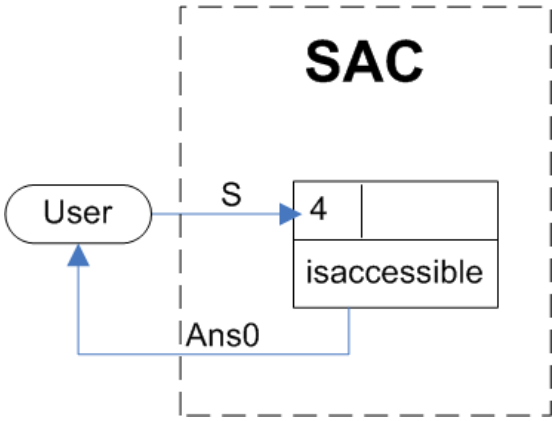


Figura 8: Diagrama de flujo para el problema de accesibilidad, nivel 1.

Consultando en el diccionario de datos de SAC los términos que se muestran se puede tener una idea completa del funcionamiento del programa:

Ans0= {true | false | cannot determinate}
For further information see DecisionTree0 (3.DT0) in the data flow diagrams section.

S= *System name. It could contain only capital or small letters, numbers and the low dash "_". *

En este caso el usuario introduce el nombre del sistema de ecuaciones (S) y la respuesta del sistema ($Ans0$) puede tomar los valores de *true, false* o *cannot determinate*. Se observa que este diagrama se puede relacionar con el documento de caso de uso mostrado en la página 35. A partir de este diagrama general comienza a desarrollarse cada una de las rutinas que se utilizaron para resolver este problema.

El siguiente nivel del diagrama anterior se muestra en la Figura 9. El nombre del sistema (S) se almacena y los procesos que conforman las rutinas del programa pueden accesarlo.

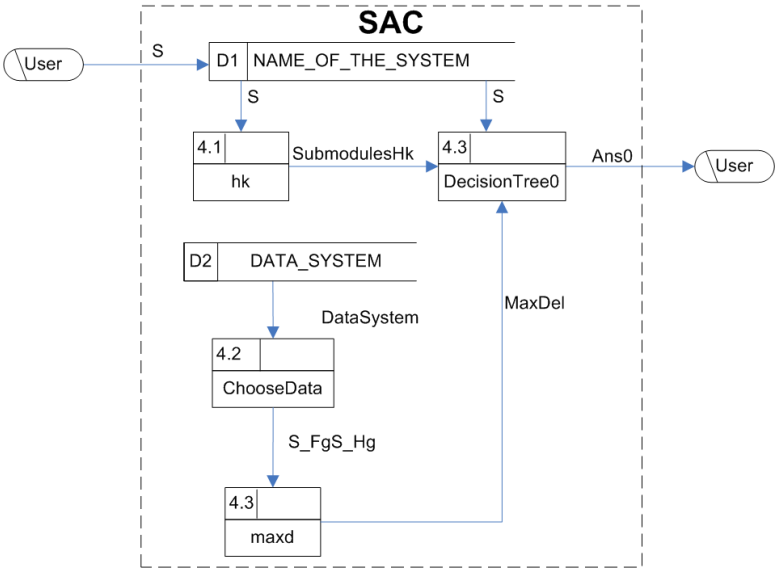


Figura 9: Diagrama de flujo para el problema de accesibilidad, nivel 2.

El número de referencia 3, etiquetado con el nombre *DecisionTree0* corresponde al árbol de decisión mostrado en la Figura 10.

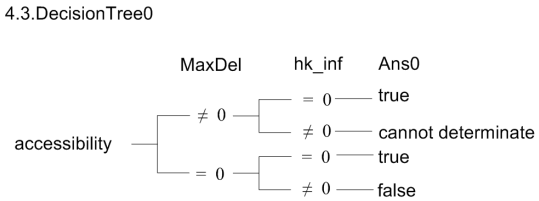


Figura 10: Árbol de decisión que complementa al diagrama de la Figura 9.

Este árbol indica las decisiones lógicas del programa. La primera ramificación por ejemplo, corresponde a un sistema no lineal con retardos, es decir, *MaxDel* distinto de cero, que es accesible, por lo que $\mathcal{H}_\infty$ es cero y la respuesta del sistema (*Ans0*) es *true*. Otros términos del diccionario de datos de SAC que complementan al diagrama de la Figura 9 son los siguientes:

DataSystem= S_dF+S_dG+S_F+S_Fg+S_G+S_var_con+S_var_sta+S_var_out.

DATA_SYSTEM= *Same information that DataSystem data.*

Hk= *Same information that SubmodulesHk.*

MaxDel= $I\!N$.
Maximum system delay.

RepresentationType= {"general" | "affine" | "differential"}
*A system may be defined in three ways within SAC, either as general, affine or differential. *More details in SAC manual.**

S_dF= *Matrix $\in I\!R^{n\times 1}$ in the differential form of representation. *See DataSystem and RepresentationType data.**

S_dG= *Matrix $\in I\!R^{n\times m}$ in the differential form of representation. *For further details see DataSystem data, RepresentationType.**

S_F= *Matrix $\in I\!R^{n\times 1}$ in the affine form of representation. *See DataSystem and RepresentationType data.**

S_Fg= *Matrix $\in \mathcal{K}[\nabla]^{n\times 1}$ equations in the general form of representation. *See RepresentationType and DataSystem data.**

S_FgS_Hg = S_Fg+S_Hg.

S_G= *Matrix $\in I\!R^{n\times m}$ in the affine form of representation. *See DataSystem and RepresentationType data.**

S_Hg= *Matrix $\in I\!R^{p\times 1}$ in the general form of representation, with $S\_Hg = \begin{bmatrix} h_1, & \dots & h_p \end{bmatrix}$.

Shk_i *Subspace $\mathcal{H}_i$ of the S system, where $i = \{1, 2, \dots, n, inf\}$ with n=number of equations in S and inf=∞.*

SubmodulesHk= $Shk\_1 + Shk\_2 + \cdots + Shk\_n + Shk\_inf$.
Filtration $\mathcal{H}_k$.

S_varcon *Control variable base of the S system.*

S_varout *Output variable base of the S system.*

S_varsta *State variable base of the S system.*

S_var_sta *Vector $\in I\!R^{n\times 1}$ with the state variables of the S system.*

S_var_con *Vector $\in I\!R^{m\times 1}$ with the control variables of the S system.*

S_var_out *Vector $\in I\!R^{p\times 1}$ output variable of the S system.*

III.3 Resumen

En este capítulo se presentaron las etapas de análisis y diseño de SAC. Se explicaron diversas técnicas de la ingeniería de software útiles en el desarrollo de un sistema, como son los casos de uso, diagramas de flujo, diccionario de datos y árboles de decisión. Se mostraron ejemplos de estas herramientas aplicadas a SAC.

En el siguiente capítulo se explica a detalle el desarrollo de los algoritmos implementados y las estrategias de prueba del programa.

Capítulo IV

Implementación de algoritmos en SAC

En este capítulo se aborda la teoría necesaria para comprender cada uno de los problemas a resolver en este trabajo y de esta manera describir el método para solucionarlos y establecer el algoritmo de implementación en el sistema SAC.

IV.1 Accesibilidad

Desde la introducción del enfoque geométrico diferencial a la teoría de control, un tema de investigación importante ha sido extender el concepto de controlabilidad a los sistemas no lineales, así surge el concepto de accesibilidad, que ha sido de gran importancia para resolver varios problemas de control en el caso de sistemas no lineales con y sin retardos.

En esta sección se aborda su extensión formal al caso de sistemas no lineales con retardos, la cual fue dada a conocer por el trabajo de Márquez-Martínez y Moog (1999).

IV.1.1 Definición del problema

La noción de controlabilidad, introducida por Kalman en los años sesenta es un concepto básico dentro de la teoría de control y después de cuarenta años sigue siendo un tema de investigación vigente. En el caso lineal esta es una *propiedad estructural*, puesto que un sistema puede ser dividido en un subsistema controlable y uno no controlable.

Para definir la controlabilidad, se requiere la noción de alcanzabilidad.

Definición 6 (Alcanzabilidad)**.** Sea un sistema no lineal Σ dado por

$$\Sigma : \left\{ \begin{array}{rcl} \dot{x}(t) & = & f(x(t-i), u(t-i), i \in \underline{S}), \end{array} \right. \tag{19}$$

cuyos parámetros se definen de manera similar al sistema (1). Se dice que el estado x_1 es alcanzable desde el estado x_0 si existe un tiempo finito T y una función $u(t) : [0, T] \rightarrow I\!R^m$ tal que $x(x_0, u, T) = x_1$ (Conte *et al.*, 1999).

La controlabilidad se define entonces como sigue:

Definición 7 (Controlabilidad)**.** Se dice que el sistema no lineal Σ, dado por (19), es controlable en x_0 si existe una vecindad $V(x_0)$ tal que cualquier estado x_1 es *alcanzable* desde x_0 (Conte *et al.*, 1999).

Sin embargo, en general no se puede esperar que un sistema no lineal sea controlable desde un estado inicial hasta cualquier estado final, debido a su propia dinámica, por lo cual la noción de controlabilidad es un concepto muy fuerte tratándose de esta clase de sistemas. A continuación se presenta un ejemplo que ilustra esta idea.

Ejemplo 1.

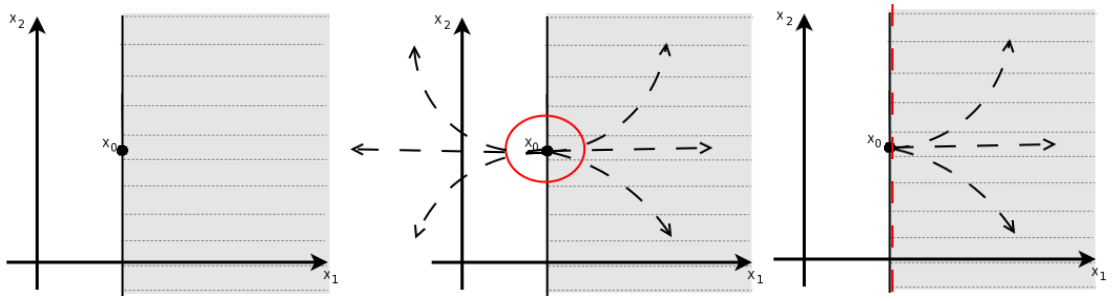
$$\begin{array}{rcl} \dot{x}_1 & = & x_2^2 \\ \dot{x}_2 & = & u. \end{array}$$

Como se observa en la Figura 11b este sistema no es controlable desde cualquier punto inicial x_0 ya que cualquier vecindad contiene un punto x_1 que no se puede alcanzar. Para solventar este problema se introduce la noción de accesibilidad, que es el equivalente estructural de la controlabilidad para el caso de sistemas no lineales, puesto que un sistema no lineal puede dividirse en un subsistema accesible y uno no accesible.

Definición 8 (Accesibilidad)**.** Un sistema es accesible en x_0 si el conjunto de puntos alcanzables desde x_0 es de dimensión n (Sontag, 1991).

De modo que, de acuerdo a esta definición, el sistema del Ejemplo 1 es accesible.

En general, la accesibilidad se asocia con la posibilidad de modificar la dinámica del sistema. Desde el punto de vista algebraico se define en términos de funciones autónomas.



(a) Conjunto de puntos alcanzables. (b) Noción de controlabilidad. (c) Noción de accesibilidad.

Figura 11: Comparación de los conceptos de alcanzabilidad, controlabilidad y accesibilidad.

Definición 9. Una 1-forma no nula w es un elemento autónomo para el sistema Σ si existe un entero v y coeficientes no todos triviales $\alpha_i \in \mathcal{K}[\nabla]$, tales que

$$\alpha_0 w + \cdots + \alpha_v w^{(v)} = 0.$$

Definición 10 (Función autónoma)**.** Una función $\varphi \in \mathcal{K}$ no idénticamente cero es autónoma si $d\varphi$ es un elemento autónomo de Σ.

Una función autónoma es un estado no controlable de un sistema, por lo que la definición formal de accesibilidad es la siguiente:

Definición 11 (Accesibilidad)**.** Un sistema Σ es accesible si no posee ninguna función autónoma.

También puede calcularse si un sistema es accesible en función del grado relativo.

Definición 12 (Grado relativo)**.** Sea $\mathcal{X} = \text{span}_{\mathcal{K}[\nabla]}\{d\mathbf{x}\}$. El grado relativo $r(\omega)$ de una 1-forma $\omega \in \mathcal{X}$ se define como

$$r(\omega) = \min\left\{k \in I\!N \mid \omega^{(k)} \notin \mathcal{X}\right\}.$$

Si, para todo $k \in \mathbb{N}$, $\omega^{(k)} \in \mathcal{X}$, entonces $r(\omega) = \infty$ y ω es una 1-forma autónoma.

$\omega^{(k)}$ es la k-ésima diferencial con respecto al tiempo de la 1-forma ω a lo largo de las trayectorias del sistema Σ.

IV.1.2 Filtración de submódulos $\mathcal{H}_{\mathcal{K}}$

Un método que permite probar con un número reducido de cálculos la accesibilidad de un sistema es la filtración de submódulos $\mathcal{H}_{\mathcal{K}}$, el cual se introdujo para sistemas no lineales sin retardos en Aranda-Bricaire *et al.* (1995); Aranda-Bricaire *et al.* (1996) y para sistemas no lineales con retardos en Márquez-Martínez y Moog (2001a).
Esta filtración $\mathcal{M} \supset \mathcal{H}_0 \supset \mathcal{H}_1 \supset \cdots \supset \mathcal{H}_{\mathcal{K}} \supset \cdots$ de $\mathcal{E}$ clasifica las 1-formas ω con respecto a su grado relativo $r(\omega)$:

$$\begin{aligned} \mathcal{H}_0 &:= \operatorname{span}_{\mathcal{K}[\nabla]} \{ \mathrm{d}x, \mathrm{d}u \} \\ \mathcal{H}_j &:= \operatorname{span}_{\mathcal{K}[\nabla]} \{ \omega \in \mathcal{H}_{j-1} \mid \dot{\omega} \in \mathcal{H}_{j-1} \}, \ j \in \mathbb{N}. \end{aligned} \tag{20}$$

De manera que:

$$\mathcal{H}_j := \left\{ \omega \in \operatorname{span}_{\mathcal{K}[\nabla]} \{ dx \} \mid r(\omega) \geq j \right\}.$$

Ha sido probado en Márquez-Martínez y Moog (2001b) que si una 1-forma tiene grado relativo finito, entonces su valor no es mayor que n.

Lema 3. *Sea $\omega \in \mathcal{X}$ una 1-forma con grado relativo $r(\omega)$. Si $r(\omega) > n$, entonces $r(\omega) = \infty$.*

En otros términos $\mathcal{H}_{n+1} = \mathcal{H}_{\infty}$, por lo que el algoritmo converge a lo más en n pasos.

IV.1.3 Cálculo alterno de $\mathcal{H}_\mathcal{K}$

Un cálculo alterno de $\mathcal{H}_\mathcal{K}$ se encuentra en Márquez-Martínez y Moog (2001b) donde se ha demostrado que $\mathcal{H}_\mathcal{K}$ es el aniquilador por la izquierda de la matriz

$$\mathcal{H}_k = [g_1, \ldots, g_{k-1}]^\perp \tag{21}$$

donde:

$$g_1 = g[\nabla] \tag{22}$$

$$g_{i+1} = F[\nabla]g_i - \dot{g}_i, \tag{23}$$

con $F[\nabla]$ y $g[\nabla]$ definidos por la Ecuación (9).

El aniquilador se calcula utilizando los conceptos básicos de la Sección II.3.4 donde se vió que mediante operaciones elementales sobre una matriz A se puede llegar a la forma de Smith, de manera que:

$$P \cdot A \cdot Q = S.$$

Siendo P y Q matrices formadas por las operaciones elementales sobre A por la izquierda y derecha respectivamente; S es la forma de Smith de la matriz A. Es posible obtener, mediante operaciones similares sobre la matriz A, una matriz denominada *pre-forma de Smith*:

$$\begin{bmatrix} P_{1,1} & \ldots & P_{1,n} \\ \vdots & \ddots & \vdots \\ P_{n,1} & \ldots & P_{n,n} \end{bmatrix} \begin{bmatrix} A_{1,1} & \ldots & A_{1,m} \\ \vdots & \ddots & \vdots \\ A_{n,1} & \ldots & A_{n,m} \end{bmatrix} \begin{bmatrix} Q_{1,1} & \ldots & Q_{1,n} \\ \vdots & \ddots & \vdots \\ Q_{n,1} & \ldots & Q_{n,n} \end{bmatrix} = \left[\begin{array}{ccc} s_{1,1} & \ldots & s_{1,r} \\ & \ddots & \vdots \\ (0) & & s_{r,r} \\ \hline & & (0) \end{array}\right]. \tag{24}$$

Como se observa, una base de la matriz A está formada por los últimos renglones de P, ya que estos mapean A a 0:

$$g_i^{\perp} = \begin{bmatrix} P_{r+1,1} & \dots & P_{r+1,n} \\ \vdots & \ddots & \vdots \\ P_{n,1} & \dots & P_{n,n} \end{bmatrix}. \tag{25}$$

Teorema 7. *Los enunciados siguientes son equivalentes:*

1. *El sistema Σ no contiene elementos autónomos.*

2. *$\mathcal{H}_\infty$ es cero.*

3. *$[g_1, \dots, g_n]^{\perp} = 0$.*

4. *$Rango_{\mathcal{K}[\nabla]}[g_1, \dots, g_n] = n$.*

Estos resultados permiten obtener una condición *suficiente* para la accesibilidad de sistemas no lineales con retardos y una condición *necesaria y suficiente* para sistemas no lineales sin retardos.

IV.1.4 Algoritmo

El siguiente algoritmo indica si un sistema es accesible:

Algoritmo 3: Accesibilidad

Data: Definir sistema de ecuaciones

$$\Sigma_L : \begin{cases} d\dot{\mathbf{x}}(t) &= F[\nabla]d\mathbf{x}(t) + g[\nabla]d\mathbf{u}(t) \\ d\mathbf{y}(t) &= H[\nabla]d\mathbf{x}(t) \end{cases}$$

begin

- $g_1 \leftarrow g[\nabla]$
- **for** *i=1* ***to*** n **do**
  - Calcular el aniquilador

$$\mathcal{H}_{i+1} \leftarrow g_i^{\perp}$$

$$g_{i+1} \leftarrow [g_i, F[\nabla]\, g_i - \dot{g}_i]$$

- **if** $\mathcal{H}_{n+1} = 0$ **then**
  - El sistema es accesible
- **else**
  - **if** *Máximo retardo =0* **then**
    - El sistema no es accesible
  - **else**
    - No se sabe

end

IV.2 Observabilidad

La mayoría de los trabajos consideran que todos los componentes del estado están disponibles y pueden ser usados en el diseño del control. Sin embargo, esto no siempre es posible, es entonces cuando se debe tener en cuenta si el sistema es observable. El término observabilidad se refiere a la propiedad de poder reconstruir el estado a partir de la salida medida, la entrada y eventualmente de sus derivadas temporales. A continuación se verá una definición formal del problema para el caso con retardos.

IV.2.1 Definición del problema

Definición 13 (Observabilidad)**.** Un sistema Σ de la forma (2) es observable cuando el estado *x(t)* puede ser expresado como una función φ de las derivadas de la salida y entrada como sigue:

$$x(t) = \varphi(y^{(k)}(t+\tau), u^{(l)}(t+\rho), k, l, \tau, \rho \in I\!N).$$

Es decir, se recobra el estado a partir de la medición de la salida, la entrada y un número finito de sus derivadas en tiempo. Para los sistemas no lineales esta propiedad se refiere a la existencia de un subconjunto abierto y denso donde el sistema es localmente observable.

IV.2.2 Filtración de submódulos $\mathcal{O}_k$

Para calcular si un sistema es observable se realiza una filtración de submódulos de observabilidad. Dado un sistema de la forma 2 se definen

$$\begin{aligned} \mathcal{X} &= \text{span}_{\mathcal{K}[\nabla]}\{dx\} \\ \mathcal{Y}^{(k)} &= \text{span}_{\mathcal{K}[\nabla]}\{dy^{(j)}, 0 \leq j \leq k\} \\ \mathcal{U} &= \text{span}_{\mathcal{K}[\nabla]}\{du^{(j)}, j \geq 0\}. \end{aligned}$$

Entonces puede establecerse esta filtración como sigue

$$0 \subset \mathcal{O}_0 \subset \mathcal{O}_1 \subset \cdots \subset \mathcal{O}_k \subset \cdots \subset \mathcal{O}_\infty,$$

donde

$$\mathcal{O}_k = \mathcal{X} \cap \{\mathcal{Y}^{(k)} + \mathcal{U}\}.$$

En el caso de un sistema lineal sin excitación, descrito mediante las ecuaciones:

$$\begin{aligned} \dot{x} &= Ax \\ y &= Cx, \end{aligned}$$

este cálculo equivale a

$$\mathcal{O}_k = \text{span}_{\mathcal{K}[\nabla]}\{Cdx, CAdx, \ldots, CA^{k-1}dx\}.$$

El límite de esta filtración es $\mathcal{O}_\infty$ y es llamado *submódulo observable* del sistema Σ

$$\mathcal{O}_\infty = \mathcal{X} \cap \{\mathcal{Y} + \mathcal{U}\}.$$

Definición 14 (Observabilidad genérica)**.** Un sistema Σ de la forma (2) es genéricamente observable si y solamente si

$$\mathcal{O}_\infty = \mathcal{X} \tag{26}$$

$$dim\mathcal{O}_\infty = rank_{\mathcal{K}[\nabla]}\left[\frac{\partial(y, \dot{y}, \ldots, y^{(n-1)})}{\partial x}\right] = n. \tag{27}$$

Mediante la Ecuación (27) se puede establecer un algoritmo computacional más sencillo que mediante la filtración de submódulos $\mathcal{O}_k$ para evaluar si un sistema es observable. El cálculo de las derivadas se explica a continuación.

$$\dot{y} = \frac{\partial h(x(t-i))}{\partial x(t-i)}[f(x(t-i), u(t-i))] = L_f h(x(t-i))$$

donde $L_f h(x(t-i))$ es la derivada de la salida sobre las trayectorias del sistema Σ y se

denomina derivada de Lie. Se tiene que:

$$\begin{aligned} L_f^0 h(x(t-i)) &= h(x(t-i)) \\ L_f^2 h(x(t-i)) &= \frac{\partial L_f h(x(t-i))}{\partial x(t-i)}[f(x(t-i), u(t-i))] \\ &\vdots \quad \vdots \quad \vdots \\ L_f^k h(x(t-i)) &= L_f L_f^{k-1} h(x(t-i)) = \frac{\partial (L_f^{k-1} h(x(t-i)))}{\partial x(t-i)}[f(x(t-i), u(t-i))], \end{aligned}$$

de manera que la matriz de observabilidad genérica puede redefinirse como:

$$\triangle = \sum_{i=1}^{s} \frac{\partial}{\partial x(t-i)} \begin{bmatrix} L_f^{(0)} h(x(t-i)) \\ L_f^{(1)} h(x(t-i)) \\ \vdots \\ L_f^{(n-1)} h(x(t-i)) \end{bmatrix} \nabla^i, \tag{28}$$

donde s es el retardo máximo del sistema Σ.

IV.2.3 Algoritmo

El Algoritmo 4 muestra si un sistema es observable.

Algoritmo 4: Observabilidad

Data: Expresando el sistema Σ descrito por (1) en forma matricial:

$$\begin{bmatrix} \dot{x}_1(t) \\ \vdots \\ \dot{x}_n(t) \end{bmatrix} = \begin{bmatrix} f_1(x(t-i),u(t-i)) \\ \vdots \\ f_n(x(t-i),u(t-i)) \end{bmatrix}, \qquad \begin{bmatrix} y_1(t) \\ \vdots \\ y_p(t) \end{bmatrix} = \begin{bmatrix} h_1(x(t-i)) \\ \vdots \\ h_p(x(t-i)) \end{bmatrix}$$

donde $i \in \underline{S} := \{0,1,\ldots,s\}$

begin

Calcular la matriz de observabilidad genérica (28):

$$\triangle = \sum_{i=0}^{s} \begin{bmatrix} \frac{\partial h_1(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_1(\cdot)}{\partial x_n(t-i)} \\ \vdots & & \vdots \\ \frac{\partial h_p(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_p(\cdot)}{\partial x_n(t-i)} \\ \frac{\partial h_1^{(1)}(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_1^{(1)}(x(t-i))}{\partial x_n(t-i)} \\ \vdots & & \vdots \\ \frac{\partial h_p^{(1)}(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_p^{(1)}(\cdot)}{\partial x_n(t-i)} \\ & \vdots & \\ \frac{\partial h_1^{(n-1)}(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_1^{(n-1)}(\cdot)}{\partial x_n(t-i)} \\ \vdots & & \vdots \\ \frac{\partial h_p^{(n-1)}(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_p^{(n-1)}(\cdot)}{\partial x_n(t-i)} \end{bmatrix} f(\cdot) \cdot \nabla^i$$

donde $f(\cdot) := f(x(t-i),u(t-i))$, $h_j^l(\cdot) := h_j^l(x(t-i))$; $j := \{1,\ldots,p\}$ y $l := \{0,\ldots,n-1\}$

if *rango*$(\triangle)=n$ **then**

El sistema es observable

else

El sistema no es observable

end

IV.3 Linealización por inyecciones aditivas entrada-salida

La teoría lineal cuenta con una amplia gama de herramientas bien definidas para atacar los diversos problemas de control, es por ello que siempre se ha buscado extender su uso a sistemas no lineales. Sin embargo para ello se requiere que el sistema tenga una linealidad inherente, es decir, que su representación pueda modificarse de tal forma que sea equivalente a un sistema lineal.

En esta sección se aborda el problema de linealización por inyecciones aditivas entrada-salida, se explican los conceptos necesarios para entender el algoritmo que resuelve este problema y su implementación en SAC. Dicho algoritmo se plantea en el trabajo de Márquez-Martínez (2000), donde la linealización brinda una condición suficiente para la síntesis de un observador para sistemas no lineales con retardos así como una solución completa al problema de linealización por retroalimentación estática de salida.

IV.3.1 Definición del problema

Definición 15. Supóngase que el sistema (1) es observable, y que $u, y \in I\!R$. Se dice que (1) es linealizable por inyecciones aditivas entrada-salida si la salida puede ser escrita bajo la forma:

$$y_{1,0}^{(\bar{n})} = \sum_{j=1}^{\bar{n}} \Phi_j^{(j-1)}(y, u). \tag{29}$$

Es decir, se busca encontrar, si existe, un cambio de coordenadas $\{\xi_1, \ldots, \xi_n\} = \Phi(x(t-i))$ tales que

$$\text{rango} \frac{\partial \Phi(x(t-i))}{x(t-i)} = n$$

y

$$\begin{aligned} \dot{\xi}_1 &= \xi_2 + \varphi_1(y,u) \\ &\vdots \\ \dot{\xi}_{n-1} &= \xi_n + \varphi_{n-1}(y,u) \\ \dot{\xi}_n &= \varphi_n(y,u) \\ y &= \xi_1. \end{aligned} \tag{30}$$

De modo que si se tiene que $E^0 = 0$, $E^k = \text{span}_{\mathcal{K}[\nabla]}\{dy_{1,0}, \ldots, dy_{1,0}^{(k-1)}, du_{1,0}, \ldots, du_{1,0}^{(k-1)}\}$. Si la relación (29) se satisface, entonces:

$$dy^{(\bar{n})} = d[\Phi_1(y,u)]^{(\bar{n}-1)} + \cdots + d\Phi_{\bar{n}}(y,u), \tag{31}$$

bajo la hipótesis de que:

$$dim_{K[\nabla]} E^{\bar{n}} = 2\bar{n}. \tag{32}$$

La condición (31) puede verificarse en términos de integrabilidad de ciertas 1-formas diferenciales.

IV.3.2 Algoritmo

El Algoritmo 5 permite calcular si un sistema es linealizable por inyecciones aditivas entrada-salida.

Lema 4. *Bajo la hipótesis* (32)*,* $dy^{(\bar{n})} \in \mathcal{E}$ *puede escribirse bajo la forma* (31) *si, y solamente si,* $dy^{(\bar{n})} \in E^{\bar{n}}$ *y*

$$d\bar{\omega}_i = 0, i = 1, \ldots, \bar{n}.$$

Algoritmo 5: Linealización por inyecciones aditivas de entrada-salida

begin

if $dy^{(\bar{n})} \notin E^{\bar{n}}$ **then**

Fin del algoritmo. No es linealizable.

else

Se define $\omega_1 = dy^{(\bar{n})}$

Elegir las funciones $\xi_1, \theta_1 \in \mathcal{K}[\nabla]$ tales que

$$\omega_1 - \xi_1 dy^{(\bar{n}-1)} - \theta_1 du^{(\bar{n}-1)} \in E^{(\bar{n}-1)}. \tag{33}$$

$\bar{\omega}_1 := \xi_1 dy + \theta_1 du$.

if $d\bar{\omega}_1 \neq 0$ **then**

Fin del algoritmo. No es linealizable.

else

for l=2 ***to*** $\bar{n}$ **do**

Sea $\Phi_{l-1}(y, u)$ tal que $d\Phi_{l-1} = \bar{\omega}_{l-1}$.

$\omega_l := \omega_{l-1} - d\Phi_{l-1}^{\bar{n}-l+1}$. Elegir $\xi_l, \theta_l \in \mathcal{K}[\nabla]$ tales que

$$\omega_l - \xi_l dy^{(\bar{n}-l)} - \theta_l du^{(\bar{n}-l)} \in E^{\bar{n}-l}$$

Definir la 1-forma $\bar{\omega}_l$ como

$$\bar{\omega}_l = \xi_l dy - \theta_l du$$

if $d\bar{\omega}_l \neq 0$ **then**

Parar

Fin del algoritmo.

end

IV.4 Equivalencia triangular

La equivalencia a la forma triangular ha sido útil para estudiar propiedades básicas de sistemas no lineales sin retardos, se ha utilizado para resolver problemas en materia de estabilidad, además de ser una condición necesaria para la completa linealización entrada/salida. En el caso de sistemas no lineales con retardos los trabajos de Márquez-Martínez *et al.* (2001); Márquez-Martínez y Moog (2004) abrieron una línea de investigación en la que todavía quedan varios problemas abiertos, por ejemplo el estudio de la estabilidad utilizando la equivalencia a la forma triangular.

En esta sección se definen las condiciones necesarias y suficientes para obtener un sistema equivalente a la forma triangular, así como el algoritmo implementado en SAC.

IV.4.1 Definición del problema

Definición 16 (Cambio de coordenadas)**.** Sea el sistema Σ, con las coordenadas locales de estado x_0. Entonces

$$\xi_0 = \Psi(x),$$

es un cambio de coordenadas de estado para el sistema si existe una función Ψ^{-1} y constantes $\tau_1, \dots \tau_n \in \mathbb{N}$ tales que (localmente)

$$diag\{\nabla^{\tau_i}\} x_0 = \Psi^{-1}(\xi_0).$$

ξ_0 es un cambio de coordenadas bicausal si $\max\{\tau_i\} = 0$, es decir, si $x_0 = \Psi^{-1}(\xi)$.
Con $x := \{x_0, \dots, x_\tau\}$ y $x_i := \{z_{1,i}, \dots, z_{n,i}\}$.

Nótese que el Lema 2 asegura que los subespacios $\mathcal{H}_\mathcal{K}$ admiten una forma de Smith con polinomios invariantes escalares. Esta propiedad les brinda dos características importantes:

- El cambio de coordenadas siempre es bicausal.

- Los subespacios son submódulos cerrados, por lo tanto son únicos.

De manera que se trabaja con los subespacios $\mathcal{H}_\mathcal{K}$ para determinar la equivalencia a la forma triangular.

Definición 17 (Definición del problema)**.** Un sistema Σ de la forma (1), con $u \in I\!R$, es equivalente a la forma triangular si es posible encontrar un cambio de coordenadas bicausal $\bar{x}$ tal que el sistema pueda escribirse de la forma:

$$\begin{aligned} \dot{\bar{x}}_j &= f_j(x_k(t-i), i \in \underline{S}) \text{ con } j = 1, \ldots, n-1, \quad k = 1, \ldots, j+1 \\ \dot{\bar{x}}_n &= f_n(x(t-i), i \in \underline{S}) \end{aligned}$$

donde

$$\underline{S} := \{0, 1, \ldots, s\}.$$

En este trabajo interesa particularmente que el cambio de coordenadas sea bicausal, ya que esta condición evita encontrar soluciones causales en el sistema transformado que impliquen términos no causales en las coordenadas de origen.

Definición 18. (Márquez-Martínez y Moog, 2004) El sistema Σ es equivalente a la forma triangular bajo un cambio de coordenadas si y sólo si

- $\mathcal{H}_\infty = 0$
- $\mathcal{H}_k$ es integrable, para todo $k = 1, \ldots, n$.

Ambas condiciones implican que puedan existir un cambio de coordenadas y un cambio de base (Proposición 2) , ya que si $\mathcal{H}_\infty = 0$ todos los subespacios de la filtración serán independientes y si $\mathcal{H}_\mathcal{K}$ es integrable, como se vió en la Sección II.3.3, entonces existe una matriz de transformación.

Los subespacios $\mathcal{H}_\mathcal{K}$ se determinan mediante el Algoritmo 3 y si estos son o no son integrables se calcula mediante el Teorema 6, tomando $A = \mathcal{H}_{\mathcal{K}+1}$ y $B = \mathcal{H}_\mathcal{K}$ con $A \subset B$, por lo que la Ecuación (18) será una condición necesaria y suficiente para asegurar la integrabilidad.

Retomando el Corolario 2, el submódulo que debe cumplir con la Ecuación (18) es $\Omega := \text{span}_{\mathcal{K}[\nabla]}\{d\varphi_1, \ldots, d\varphi_s, \bar{\omega}\}$ con $A = \mathcal{H}_{\mathcal{K}+1} = \text{span}_{\mathcal{K}[\nabla]}\{d\varphi_1, \ldots, d\varphi_s\}$. Se observa que se comienza por probar si $\mathcal{H}_n$ es integrable y posteriormente se requiere *completar la base* de $\mathcal{H}_{n-1}$ buscando el vector $\bar{\omega}$. Esto se realiza mediante el siguiente teorema:

Teorema 8. *(Márquez-Martínez y Moog, 2007) Sean g_1 y g_2 dos matrices con elementos en $\mathcal{K}[\nabla]$, y sean $A = g_1^{\perp}$ y $B = g_2^{\perp}$ dos submódulos de $\mathcal{M}$, con $A \subset B$. Suponiendo que $\{\omega_i, i = 1, \ldots, r_A\}$ forman una base para A, donde r_A es el rango de A; entonces, existen vectores $\{\omega_j, j = r_A + 1, \ldots, r_B\} \in B$, donde r_B es el rango de B; tales que el conjunto $\{\omega_i, i = 1, \ldots, r_b\}$ es una base para B.*

Demostración(Márquez-Martínez y Moog, 2007)

Dado que $A \subset B$ existe una matriz $T[\nabla] \in \mathcal{K}^{r_a \times r_b}[\nabla]$ tal que

$$\omega = T[\nabla]\bar{\omega}$$

donde los elementos de $\bar{\omega} := [\bar{\omega}_1, \ldots, \bar{\omega}_{r_B}]^T$ son una base arbitraria para B. El Lema 2 nos dice que existen matrices unimodulares $P'[\nabla]$ y $Q'[\nabla]$ tales que $T[\nabla] = P'[\nabla] \begin{bmatrix} \mathbb{I}_{r_A} & 0_{r_A \times (r_B - r_A)} \end{bmatrix} Q'[\nabla]$. De esta relación se definen los siguientes vectores:

$$\tilde{\omega} = \begin{bmatrix} [0_{(r_B - r_A) \times r_A} & \mathbb{I}_{(r_B - r_A)} \end{bmatrix} Q'[\nabla]\bar{\omega}. \tag{34}$$

Entonces, $(\omega, \tilde{\omega})$ *es una base para B,* puesto que

$$\begin{bmatrix} \omega \\ \tilde{\omega} \end{bmatrix} = \begin{bmatrix} P'[\nabla] & 0 \\ 0 & \mathbb{I} \end{bmatrix} \begin{bmatrix} \mathbb{I}_{r_A} & 0_{r_A \times (r_B - r_A)} \\ 0_{(r_B - r_A) \times r_A} & \mathbb{I}_{(r_B - r_A)} \end{bmatrix} Q'[\nabla]\bar{\omega}.$$

La transformación inversa $\bar{\omega} = (Q')^{-1}[\nabla] \begin{bmatrix} (P')^{-1}[\nabla] & 0 \\ 0 & \mathbb{I} \end{bmatrix} \begin{bmatrix} \omega \\ \bar{\omega} \end{bmatrix}$ existe debido a que las matrices $P'[\nabla]$ y $Q'[\nabla]$ son unimodulares.

IV.4.2 Algoritmo

El Algoritmo 6 permite calcular si un sistema es equivalente a la forma triangular.

Algoritmo 6: Equivalencia a la forma triangular

Data: Definir sistema de ecuaciones de la forma (1) con $u \in \mathbb{R}$

begin

if $u \neq 1$ **then**

Sistema no equivalente a la forma triangular. Fin del algoritmo.

else

Calcula los subespacios $\mathcal{H}_{\mathcal{K}}$

if $\mathcal{H}_\infty \neq 0$ **then**

Sistema no equivalente a la forma triangular. Fin del algoritmo.

else

Se calcula si $\mathcal{H}_n = \text{span}_{\mathcal{K}[\nabla]}\{\omega_1\}$ es integrable (16):

$$\Omega := d\omega_1 \wedge \omega_1 = 0$$

if $\Omega \neq 0$ **then**

Sistema no equivalente a la forma triangular. Fin del algoritmo.

else

$\alpha = 2$.

for $l = n - 1$ ***to*** 2 **do**

Sea $\mathcal{H}_l := span_{\mathcal{K}[\nabla]}\{\omega_1, \ldots, \omega_\alpha\}$

$$\mathcal{H}_l = \begin{bmatrix} \dfrac{\partial \omega_1}{\partial x_1} & \cdots & \dfrac{\partial \omega_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial \omega_\alpha}{\partial x_1} & \cdots & \dfrac{\partial \omega_\alpha}{\partial x_n} \end{bmatrix} \mathbf{dx} = A[\nabla] \cdot \mathbf{dx}$$

con $\mathbf{dx} := \left[dx_1, \ldots, dx_n\right]^T$.

Se calcula la forma de Smith de $A[\nabla]$:

$$\begin{aligned} Smith(A[\nabla]) &= P[\nabla] \cdot A[\nabla] \cdot Q[\nabla] = S \\ A[\nabla] &= P'[\nabla] \cdot S \cdot Q'[\nabla] \end{aligned}$$

Completar una base para $\mathcal{H}_l$ (34):

$$\tilde{\omega} = \left[0_{(\alpha-(\alpha-1))\times(\alpha-1)} \quad \mathbb{I}_{(\alpha-(\alpha-1))}\right] Q'[\nabla] \mathcal{H}_l$$

$$\tilde{\omega} \cdot \mathbf{dx} = \begin{bmatrix} \tilde{\omega}_1 \\ \vdots \\ \tilde{\omega}_{(\alpha-(\alpha-1))} \end{bmatrix}$$

De modo que siendo $\mathcal{H}_{l+1} := span_{\mathcal{K}[\nabla]}\{\omega_1, \ldots, \omega_{(\alpha-1)}\}$

$$\mathcal{H}_l := \text{span}_{\mathcal{K}[\nabla]}\{\omega_1, \ldots, \omega_{(\alpha-1)}, \tilde{\omega}_1, \ldots, \tilde{\omega}_{(\alpha-(\alpha-1))}\}$$

Se verifica si $\mathcal{H}_l$ es integrable (18):

$$\Omega := d\tilde{\omega} \wedge \tilde{\omega} \wedge \omega \wedge \cdots \wedge \nabla^k \omega, \quad k \in \mathbb{N}, \ \omega := \omega_1 \wedge \cdots \wedge \omega_{(\alpha-1)}$$

if $\Omega \neq 0$ **then**

Sistema no equivalente a la forma triangular. Fin del algoritmo.

else

$\alpha = \alpha + 1$.

end

IV.5 Rechazo de perturbaciones

Uno de los mayores retos al controlar un sistema es asegurar su funcionamiento a pesar de que existan ruido y perturbaciones que afecten el desempeño del controlador.

El problema de rechazo de perturbaciones consiste en eliminar su influencia sobre las salidas de un sistema Σ, ya sea de manera asintótica o perfecta. Para lograrlo, las perturbaciones pueden incluirse en el modelo como entradas no controlables, aún cuando no sean medibles.

En esta sección se define el problema de rechazo de perturbaciones a partir del enfoque utilizado en este trabajo y la metodología del algoritmo implementado en SAC.

IV.5.1 Definición del problema

La clase de sistemas considerados son modelados por ecuaciones de estado de la forma:

$$\Sigma_p : \begin{cases} \dot{x}(t) &= f(x(t-i), i \in \underline{S}) \\ & \quad + \sum_{j=0}^{s} (g_j(x(t-i), i \in \underline{S})u(t-j) + p_j(x(t-i), i \in \underline{S})q(t-j) \\ y(t) &= h(x(t-i), i \in \underline{S}) \end{cases} \tag{35}$$

El estado $x \in I\!R^n$, la entrada de control $u \in I\!R^m$, y la salida $y \in I\!R^p$. Las entradas de f, g, h y p son funciones meromorfas de sus argumentos. $\underline{S} := \{0, 1, \ldots, s\}$ es un conjunto finito de retardos constantes en tiempo y $q(t-j)$ representa el vector de perturbaciones.

Definición 19 (Problema de rechazo de perturbaciones)**.** Sea el sistema Σ_p, con entradas de control $\mathbf{u}$ y salidas $\mathbf{y}$, afectadas por las perturbaciones $\mathbf{q}$. Elegir, si es posible, un compensador C, tal que las salidas $\mathbf{y}$ no sean afectadas por las perturbaciones $\mathbf{q}$; es decir, tal que el sistema compensado cumpla la relación

$$d\mathbf{y}^{(k)} \in \text{span}_{\mathcal{K}[\nabla]}\{d\mathbf{x}, d\mathbf{v}, \ldots, d\mathbf{v}^{(k-1)}\}, \quad \forall k \in I\!N, \tag{36}$$

donde $\mathbf{v}$ es la nueva entrada de control, como se observa en la Figura 12 (Márquez-Martínez, 2000).

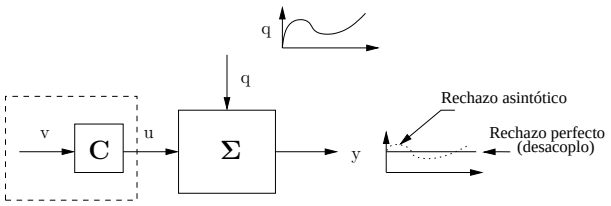


Figura 12: El efecto de la perturbación **q** sobre la salida **y** disminuye o es destruido por el compensador C.

En el caso de sistemas sin retardos, la solución estándar consiste en utilizar retroalimentación para que las salidas sean independientes de aquellos componentes del estado cuya evolución es afectada por la perturbación, lo cual conduce a la máxima pérdida de observabilidad. Sin embargo, cuando existen retardos este procedimiento no siempre se puede aplicar ya que se tiene la restricción de que el compensador sea causal.
Para introducir la idea básica de solución se muestra el siguiente ejemplo. Sea el sistema:

$$\begin{aligned}
\dot{x}_1(t) &= x_2(t-2) + x_3(t-4) - x_4(t) + u_1(t-1) \\
\dot{x}_2(t) &= q(t) \\
\dot{x}_3(t) &= -x_3(t) \\
\dot{x}_4(t) &= x_4(t)x_3(t) \\
y(t) &= x_1(t),
\end{aligned}$$

donde se tiene que

$$\begin{aligned}
\dot{y}(t) &= x_2(t-2) + x_3(t-4) - x_4(t) + u_1(t-1) \qquad (37) \\
\ddot{y}(t) &= q(t-2) - x_3(t-4) - x_4(t)x_3(t) + \dot{u}_1(t-1).
\end{aligned}$$

Se observa que la segunda derivada de la salida es afectada por la perturbación. De la

Ecuación (37) se obtiene:

$$u(t) = v(t) - x_2(t-1) - x_3(t-3) - x_4(t+1)$$

donde aparece el término no causal $x_4(t+1)$, sin embargo, utilizando simplemente el compensador

$$u(t) = v(t) - x_2(t-1)$$

las derivadas de la salida son independientes de la perturbación:

$$\begin{aligned} \dot{y}(t) &= v_1(t-1) + x_3(t-4) - x_4(t) \\ \ddot{y}(t) &= \dot{v}_1(t-1) - x_3(t-4) - x_4(t)x_3(t). \end{aligned}$$

En el ejemplo anterior una parte del estado (x_3 y x_4) no requiere volverse inobservable para rechazar la perturbación. Es así que surge el concepto del submódulo $\Omega \subset \mathcal{M}$ asociado al sistema Σ_d (Moog *et al.*, 2000):

$$\Omega := \{\omega \in \text{span}_{\mathcal{K}[\nabla]}\{d\mathbf{x}\} \quad | \quad \forall k \in I\!N, \quad \omega^{(k)} \in cls_{\mathcal{K}[\nabla]}\{d\mathbf{x}, d\mathbf{y}, d\dot{\mathbf{y}} \dots\}\} \tag{38}$$

Ω representa el estado que no es afectado por ninguna entrada de control o de perturbación, excepto por las salidas. Este subespacio puede ser calculado como el límite de la secuencia

$$\Omega_0 := \text{span}_{\mathcal{K}[\nabla]}\{d\mathbf{x}\} \tag{39}$$

$$\Omega_{k+1} := \{\omega \in \Omega_k \mid \dot{\omega} \in cls_{\mathcal{K}[\nabla]}\{\Omega_k + \frac{d}{dt}(\mathcal{X} \cap \mathcal{Y})\}\} \tag{40}$$

donde

$$\mathcal{X} \cap \mathcal{Y} := \text{span}_{\mathcal{K}[\nabla]}\{d\mathbf{x}\} \cap \text{span}_{\mathcal{K}[\nabla]}\{d\mathbf{y}, d\dot{\mathbf{y}}, \dots\} \tag{41}$$

y $\mathbf{y}$ representa las salidas de Σ_d. En el caso de sistemas sin retardos Σ es considerado un subespacio de controlabilidad.

Este subespacio se utiliza, en conjunto con otras herramientas matemáticas, para resolver el problema de rechazo de perturbaciones, ya sea de manera estática o dinámica. En esta primera versión de SAC nos limitaremos al cálculo de Ω.

IV.5.2 Cálculo del subespacio Ω

El primer paso para obtener la secuencia de subespacios Ω, dados por la ecuaciones (39) y (40) es *el cálculo de* $\mathcal{X} \cap \mathcal{Y}$.

Retomando la Ecuación (41), lo que se busca es la intersección entre el $\text{span}_{\mathcal{K}[\nabla]}\{d\mathbf{x}\}$ y el $\text{span}_{\mathcal{K}[\nabla]}\{d\mathbf{y}, d\dot{\mathbf{y}}, \dots\}$. Este conjunto es distinto al vacío sólo si existe una matriz que permita eliminar la dependencia de las derivadas de la salida con respecto a las variables de entrada. Es decir,

$$[\alpha_0 \dots \alpha_n] \begin{bmatrix} dy \\ \vdots \\ dy^n \end{bmatrix} \in \mathcal{X}. \tag{42}$$

Comenzamos por diferenciar la salida:

$$\begin{aligned} y &= h(x_\tau) \\ dy &= \overline{h} dx \\ dy^{(1)} &= f_1 dx + g_1 du \\ dy^{(2)} &= f_2 dx + g_2 du + g_1 du^{(1)} \\ &\vdots \\ dy^{(n)} &= f_n dx + g_n du + \cdots + g_1 du^{(n-1)} \end{aligned}$$

Entonces se puede reescribir la Ecuación (42):

$$\alpha \left[\begin{bmatrix} \overline{h} \\ f_1 \\ \vdots \\ f_n \end{bmatrix} \mathbf{dx} + \begin{bmatrix} 0 \\ g_1 \\ \vdots \\ g_n \end{bmatrix} \mathbf{du} + \cdots + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ g_1 \end{bmatrix} \mathbf{du^{(n-1)}} \right] = (\cdot)\mathbf{dx}. \tag{43}$$

de donde:

$$\alpha = \begin{bmatrix} 0 & | & & | & 0 \\ g_1 & | & \cdots & | & 0 \\ \vdots & | & & | & \vdots \\ g_n & | & & | & g_1 \end{bmatrix}^{\perp}. \tag{44}$$

El aniquilador se calcula mediante la pre-forma de Smith, (ver ecuaciones (24) y (25)).

Se tiene que

$$\alpha f \in \mathcal{X} \cap \mathcal{Y}$$

con $f = \begin{bmatrix} \overline{h} & f_1 & \dots & f_n \end{bmatrix}^T$.

Para calcular la filtración de Ω se comienza por (39), por lo que en la primera etapa

$$\omega = [\alpha_0 \dots \alpha_n] \begin{bmatrix} dx_1 \\ \vdots \\ dx_n \end{bmatrix}.$$

De la Ecuación (40)

$$\Omega_{k+1} \;:=\; \{\omega \in \Omega_k \mid \dot{\omega} \in cls_{\mathcal{K}[\nabla]}\{\Omega_k + \frac{d}{dt}(\mathcal{X} \cap \mathcal{Y})\}\}. \tag{45}$$

Para el cálculo de (45) se obtiene una base $\mathbf{b} := [b_1 \dots b_r]^T$ para $\phi = cls_{\mathcal{K}[\nabla]}\{\Omega_k + \frac{d}{dt}(\mathcal{X} \cap \mathcal{Y})\}$. Sean los vectores $\begin{bmatrix} \bar{v}_1 & \dots & \bar{v}_z \end{bmatrix}$ una base de Ω_k y $\begin{bmatrix} \bar{f}_1 & \dots & \bar{f}_d \end{bmatrix}$ los elementos de $\frac{d}{dt}(\mathcal{X} \cap \mathcal{Y})$. Se tiene que:

$$\phi = \triangle \bar{\phi}$$

con $\bar{\phi} := \begin{bmatrix} dx_1 & \dots & dx_n & du_1 & \dots & du_m & dq \end{bmatrix}^T$ y

$$\triangle = \sum_{i=0}^{s} \begin{bmatrix} \frac{\partial \bar{v}_1(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial \bar{v}_1(\cdot)}{\partial x_n(t-i)} & \frac{\partial \bar{v}_1(\cdot)}{\partial u_1(t-i)} & \cdots & \frac{\partial \bar{v}_1(\cdot)}{\partial u_m(t-i)} & \frac{\partial \bar{v}_1(\cdot)}{\partial q(t-i)} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots \\ \frac{\partial \bar{v}_z(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial \bar{v}_z(\cdot)}{\partial x_n(t-i)} & \frac{\partial \bar{v}_z(\cdot)}{\partial u_1(t-i)} & \cdots & \frac{\partial \bar{v}_z(\cdot)}{\partial u_m(t-i)} & \frac{\partial \bar{v}_z(\cdot)}{\partial q(t-i)} \\ \frac{\partial \bar{f}_1(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial \bar{f}_1(\cdot)}{\partial x_n(t-i)} & \frac{\partial \bar{f}_1(\cdot)}{\partial u_1(t-i)} & \cdots & \frac{\partial \bar{f}_1(\cdot)}{\partial u_m(t-i)} & \frac{\partial \bar{f}_1(\cdot)}{\partial q(t-i)} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots \\ \frac{\partial \bar{f}_d(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial \bar{f}_d(\cdot)}{\partial x_n(t-i)} & \frac{\partial \bar{f}_d(\cdot)}{\partial u_1(t-i)} & \cdots & \frac{\partial \bar{f}_d(\cdot)}{\partial u_m(t-i)} & \frac{\partial \bar{f}_d(\cdot)}{\partial q(t-i)} \end{bmatrix} \cdot \nabla^i.$$

Utilizando la forma de Smith, definida en la Sección II.3.4 tenemos que:

$$P \triangle Q = S \qquad \Longrightarrow \qquad \triangle = P^{-1} S Q^{-1},$$

con $P \in \mathcal{K}[\nabla]^{(z+d)\times(z+d)}$ y $Q \in \mathcal{K}[\nabla]^{(n+m+1)\times(n+m+1)}$. Dado que S es una matriz de la forma

$$\begin{bmatrix} s_1 & & (0) & & & \\ & \ddots & & & & (0) \\ (0) & & s_r & & & \\ & & & 0 & & \\ (0) & & & & \ddots & \\ & & & & & 0 \end{bmatrix}$$

sus primeros r renglones son linealmente independientes. Si multiplicamos por la matriz P y por $\bar{\phi}$:

$$P \triangle \bar{\phi} = S Q^{-1} \bar{\phi}$$

donde SQ^{-1} sigue siendo una matriz con los primeros r renglones linealmente independientes, por lo que para obtener una base sólo tenemos que multiplicar:

$$\mathbf{b} = \begin{bmatrix} \mathbb{I}_r & 0_{r\times(z+d-r)} \end{bmatrix} P \triangle \bar{\phi} \tag{46}$$

y a partir de

$$\Phi = [\alpha_0 \dots \alpha_n] \begin{bmatrix} \bar{v}_1 \\ \vdots \\ \bar{v}_z \end{bmatrix} - [\beta_0 \dots \beta_n] \begin{bmatrix} b_1 \\ \vdots \\ b_r \end{bmatrix}, \tag{47}$$

se forma un sistema de ecuaciones diferenciando (47):

$$\begin{aligned}
\sum_{k=0}^{s} \frac{\partial \Phi}{x_1(t-k)} \nabla^k &= 0 \\
\vdots \qquad\qquad &\vdots \quad \vdots \\
\sum_{k=0}^{s} \frac{\partial \Phi}{x_n(t-k)} \nabla^k &= 0 \\
\\
\sum_{k=0}^{s} \frac{\partial \Phi}{u_1(t-k)} \nabla^k &= 0 \\
\vdots \qquad\qquad &\vdots \quad \vdots \\
\sum_{k=0}^{s} \frac{\partial \Phi}{u_m(t-k)} \nabla^k &= 0 \\
\\
\sum_{k=0}^{s} \frac{\partial \Phi}{q(t-k)} \nabla^k &= 0
\end{aligned} \tag{48}$$

Resolviendo (48) para las variables α se obtiene un nuevo sistema de ecuaciones que depende de β. Cada una de estas ecuaciones es un término independiente que, ya sea individual o en conjunto con otra ecuación del sistema, multiplica a alguna 1-forma de Ω_k para conformar una 1-forma del submódulo Ω_{k+1}. Las variables β se dejan indicadas ya que pueden tomar cualquier valor, únicamente nos interesan para agrupar los elementos de las 1-formas con más de un término. Estas ecuaciones forman la matriz

$\psi \in \mathcal{K}[\nabla]^{n\times n}$, cuyos elementos se ordenan como se indica a continuación:

$$\begin{array}{c} \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{array} \begin{array}{c} \begin{array}{cccc} \alpha_1 & \alpha_2 & \dots & \alpha_n \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & \dots & a_{nn} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \end{array}. \tag{49}$$

El elemento a_{11} será aquel que es solución de α_1 y contiene a la variable β_1, a_{22} aquel que es solución de α_2 y contiene a la variable β_1. Cuando no existe un término que cumpla ambas condiciones se pone 0 en ese elemento de la matriz. Al formar ψ se sustituyen las variables β por 1. Posteriormente

$$\Omega_{k+1} = \psi\Omega_k. \tag{50}$$

El submódulo Ω se encuentra cuando el rango de Ω_k y de $\Omega_k + 1$ son iguales.

IV.5.3 Algoritmo

Los algoritmos de las páginas 70 y 71 permiten calcular $\mathcal{X} \cap \mathcal{Y}$ y Ω respectivamente.

Algoritmo 7: Cálculo de $\mathcal{X} \cap \mathcal{Y}$

Data: Definir sistema de ecuaciones de la forma (35)
begin

Dado que

$$\mathcal{X} \cap \mathcal{Y} := \text{span}_{\mathcal{K}[\nabla]}\{d\mathbf{x}\} \cap \text{span}_{\mathcal{K}[\nabla]}\{d\mathbf{y}, d\dot{\mathbf{y}}, \dots\}$$

Se diferencian las salidas

$$\begin{aligned} y &= h(x_\tau) \\ dy &= \overline{h}dx \\ dy^{(1)} &= f_1 dx + g_1 du \\ dy^{(2)} &= f_2 dx + g_2 du + g du^{(1)} \\ &\vdots \\ dy^{(n)} &= f_n dx + g_n du + \cdots + g du^{(n-1)} \end{aligned}$$

Se forma la siguiente matriz (43)

$$\alpha \left[\begin{bmatrix} \overline{h} \\ f_1 \\ \vdots \\ f_n \end{bmatrix} \mathbf{dx} + \begin{bmatrix} 0 \\ g_1 \\ \vdots \\ g_n \end{bmatrix} \mathbf{du} + \cdots + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ g_1 \end{bmatrix} \mathbf{du^{(n-1)}} \right] = (\cdot)\mathbf{dx}$$

donde de (44)

$$\alpha = \begin{bmatrix} 0 & | & & | & 0 \\ g_1 & | & \dots & | & 0 \\ \vdots & & & & \vdots \\ g_n & | & & | & g_1 \end{bmatrix}^{\perp} . \tag{51}$$

$$\mathcal{X} \cap \mathcal{Y} = \alpha f$$

con $f = \begin{bmatrix} \overline{h} & f_1 & \dots & f_n \end{bmatrix}^T$

end

Algoritmo 8: Cálculo del submódulo Ω

Data: Definir sistema de ecuaciones de la forma (35)
begin

Se calcula $\mathcal{X} \cap \mathcal{Y}$ mediante el Algoritmo (7). Enseguida se tiene que

$$\Omega_0 := \text{span}_{\mathcal{K}[\nabla]}\{d\mathbf{x}\}$$

Etapa k ($k = 0, 1, \ldots i, i \in I\!N$):

Se obtiene una base $\mathbf{b} := [b_1 \ldots b_r]^T$ para $\phi = \Omega_k + \frac{d}{dt}(\mathcal{X} \cap \mathcal{Y})$. Sean $\begin{bmatrix}\bar{v}_1 & \ldots & \bar{v}_z\end{bmatrix}$ una base de Ω_k y $\begin{bmatrix}\bar{f}_1 & \ldots & \bar{f}_d\end{bmatrix}$ los elementos de $\frac{d}{dt}(\mathcal{X} \cap \mathcal{Y})$.

$$\phi = \triangle\bar{\phi}$$

con $\bar{\phi} := \begin{bmatrix}dx_1 & \ldots & dx_n & du_1 & \ldots & du_m & dq\end{bmatrix}^T$ y

$$\triangle = \sum_{i=0}^{s} \begin{bmatrix} \frac{\partial \bar{v}_1(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial \bar{v}_1(\cdot)}{\partial x_n(t-i)} & \frac{\partial \bar{v}_1(\cdot)}{\partial u_1(t-i)} & \cdots & \frac{\partial \bar{v}_1(\cdot)}{\partial u_m(t-i)} & \frac{\partial \bar{v}_1(\cdot)}{\partial q(t-i)} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots \\ \frac{\partial \bar{v}_z(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial \bar{v}_z(\cdot)}{\partial x_n(t-i)} & \frac{\partial \bar{v}_z(\cdot)}{\partial u_1(t-i)} & \cdots & \frac{\partial \bar{v}_z(\cdot)}{\partial u_m(t-i)} & \frac{\partial \bar{v}_z(\cdot)}{\partial q(t-i)} \\ \frac{\partial \bar{f}_1(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial \bar{f}_1(\cdot)}{\partial x_n(t-i)} & \frac{\partial \bar{f}_1(\cdot)}{\partial u_1(t-i)} & \cdots & \frac{\partial \bar{f}_1(\cdot)}{\partial u_m(t-i)} & \frac{\partial \bar{f}_1(\cdot)}{\partial q(t-i)} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots \\ \frac{\partial \bar{f}_d(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial \bar{f}_d(\cdot)}{\partial x_n(t-i)} & \frac{\partial \bar{f}_d(\cdot)}{\partial u_1(t-i)} & \cdots & \frac{\partial \bar{f}_d(\cdot)}{\partial u_m(t-i)} & \frac{\partial \bar{f}_d(\cdot)}{\partial q(t-i)} \end{bmatrix} \cdot \nabla^i.$$

Se calcula la forma de Smith

$$P \triangle Q = S$$

$$\mathbf{b} = \begin{bmatrix}\mathbb{I}_r & 0_{r\times(z+d-r)}\end{bmatrix} P \triangle \bar{\phi}$$

y a partir de

$$\Phi = [\alpha_0 \ldots \alpha_n] \begin{bmatrix}\bar{v}_1 \\ \vdots \\ \bar{v}_z\end{bmatrix} - [\beta_0 \ldots \beta_n] \begin{bmatrix}b_1 \\ \vdots \\ b_r\end{bmatrix}$$

se forma un sistema de ecuaciones diferenciando Φ:

$$\begin{aligned}
\sum_{k=0}^{s} \frac{\partial \Phi}{x_1(t-k)} \nabla^k &= 0 \\
\vdots \qquad\qquad &\quad \vdots \quad \vdots \\
\sum_{k=0}^{s} \frac{\partial \Phi}{x_n(t-k)} \nabla^k &= 0 \\
\sum_{k=0}^{s} \frac{\partial \Phi}{u_1(t-k)} \nabla^k &= 0 \\
\vdots \qquad\qquad &\quad \vdots \quad \vdots \\
\sum_{k=0}^{s} \frac{\partial \Phi}{u_m(t-k)} \nabla^k &= 0 \\
\sum_{k=0}^{s} \frac{\partial \Phi}{q(t-k)} \nabla^k &= 0
\end{aligned}$$

Se resuelve este sistema para las variables α. Entonces, de acuerdo a (50)

$$\Omega_{k+1} = \psi\Omega_k$$

donde ψ se forma a partir de la solución del sistema, de acuerdo a (49).
if $rank(\Omega_k) = rank(\Omega_{k+1})$ **then**
  $\Omega = \Omega_{k+1}$. Fin del algoritmo.

end

IV.6 Pruebas de requerimientos funcionales

Una estrategia de prueba, según Pressman (2005), integra los métodos de diseño de casos de prueba en una serie bien planeada de pasos que desembocará en la eficaz construcción del software.

Cualquier estrategia de prueba debe incorporar la planeación, el diseño y la ejecución de las mismas, así como la recolección y evaluación de los datos resultantes. También es importante para verificar si el sistema cumple con los requerimientos funcionales establecidos en su diseño.

En esta sección se presentan las pruebas de validación de requerimientos realizadas al programa SAC, las cuales consistieron en la elaboración de un plan de pruebas y de un conjunto de datos de prueba, además de los resultados obtenidos durante este proceso.

IV.6.1 Objetivo y factores de motivación de las pruebas

El objetivo de este tipo de pruebas es poder garantizar un producto final de alta calidad, de acuerdo a las necesidades y expectativas del usuario. En general, se busca encontrar errores de las siguientes categorías:

- funciones incorrectas o ausentes,
- errores de estructuras de datos,
- errores de rendimiento,
- errores de integración entre los distintos módulos del programa.

IV.6.2 Metodología

La metodología seguida para probar la funcionalidad de SAC, así como el cumplimiento de los requerimientos solicitados por el cliente, se describe a continuación:

- Diseñar un plan de pruebas.
- Desarrollar un conjunto de datos de prueba.
- Documentar errores e inconsistencias encontradas al aplicar el plan de pruebas.

Enseguida se describe cada una de estas actividades.

Plan de pruebas

El diseño de un plan de pruebas se requiere para asegurar la funcionalidad del programa, la implementación de los requerimientos de diseño y para proveer un procedimiento de prueba tanto para los módulos individuales como para el sistema completo (Pressman, 2005).

Comenzamos entonces por clasificar los módulos que conforman SAC, de acuerdo a su dependencia con otras rutinas, estos se muestran en la Figura 13 en forma de pirámide.

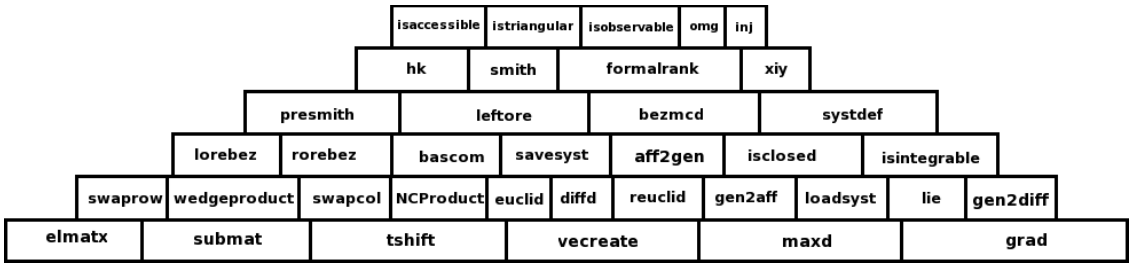


Figura 13: Módulos que conforman SAC.

Los programas que se encuentran en la base no tienen dependencias ni entre ellos, ni con los programas que están en un nivel superior. De igual forma, conforme se

aumenta de nivel las dependencias se dan con los programas que se localizan en niveles inferiores, de forma que los programas en la punta de la pirámide son los principales. Cabe señalar que estos módulos se crearon de acuerdo a los diagramas de caso de uso de cada problema a resolver.

Enseguida se eligieron las pruebas para verificar el cumplimiento de los requerimientos funcionales del sistema SAC, las cuales se muestran en la Tabla V.

Tabla V: Pruebas realizadas.

Clasificación	Tipos de pruebas
Pruebas de caja blanca	1. Pruebas de condición. 2. Pruebas generadas mediante datos aleatorios. 3. Pruebas de integración.
Pruebas de caja negra	4. Comparación de resultados con NOLIACPA. 5. Pruebas de escritorio utilizando sistemas físicos y académicos encontrados en la literatura.

La finalidad de estas pruebas es asegurar el cumplimiento de los diferentes requerimientos funcionales establecidos por el cliente, mostrados en la Tabla VI, así como las pruebas para su verificación.

Tabla VI: Tabla de requerimientos del cliente.

Requerimientos del cliente	Número de prueba
Operaciones básicas de entrada/salida (cargar, guardar ecuaciones de estado).	5
Definir un sistema en cualquier forma de representación (afín, general o diferencial).	3,4,5
Multiplataforma (funcionamiento en Linux y Windows®).	4
Estructura modular.	1,2,3
Capaz de manejar varios sistemas de ecuaciones en la misma sesión.	4,5
Problemas de análisis considerados para SNL con y sin retardos.	1,2,3,4,5

Como se menciona en la Tabla V las pruebas se dividen en dos partes:

- **Pruebas de caja blanca.** Las pruebas de caja blanca utilizadas fueron *pruebas de la estructura de control*, particularmente *pruebas de condición*, en las cuales se verifica cada una de las condiciones lógicas contenidas en un módulo del programa. Estas pruebas evalúan que cada unidad o módulo funcione de manera apropiada, sin embargo, también son útiles para realizar *pruebas de integración ascendente* ya que al empezar la construcción se prueban los módulos de los niveles más bajos y estos componentes se combinan con otros al avanzar por la estructura del programa y vuelven a probarse en cada uno de los niveles siguientes.

- **Pruebas de caja negra.** Las pruebas de caja negra examinan algún aspecto funcional de un sistema, por lo que tiene poca relación con la estructura lógica interna del software. Para definir el tipo de pruebas de caja negra a aplicar primeramente se identificaron las principales funciones que al usuario le interesa realizar, encontrando los siguientes escenarios:

  1. Definir las ecuaciones de un sistema.
  2. Guardar las ecuaciones de un sistema.
  3. Cargar las ecuaciones de un sistema.
  4. Determinar si un sistema es accesible.
  5. Determinar si un sistema es observable.
  6. Determinar si un sistema es equivalente a la forma triangular.
  7. Determinar si un sistema es linealizable por inyecciones aditivas entrada/salida.
  8. Determinar el máximo submódulo que no es afectado por perturbaciones (rechazo de perturbaciones).

Se determinó realizar dos tipos de prueba de caja negra:

- Se compararon resultados con el programa NOLIACPA (**NO**n **LI**near **A**nalysis and **C**ontrol **PA**ckage) para las rutinas de accesibilidad y observabilidad en el caso de sistemas no lineales sin retardos. Este programa fue creado en el IRCCyN (Institut de Recherche en Communications et en Cybernétique de Nantes), en plataforma Windows®, mediante el programa Maple y también utiliza el enfoque algebraico, sin embargo no admite retardos.
- Se realizaron pruebas de escritorio utilizando sistemas físicos y académicos encontrados en la literatura para probar los sistemas no lineales con retardos, así como para el resto de los problemas sin retardos dado que no existe un software que nos ayude a realizar los cálculos y comparar resultados.

Enseguida se explica la elección de los datos de acuerdo al tipo de prueba a realizar.

Desarrollo de datos de prueba

La generación de un conjunto de datos de prueba se elabora con la finalidad de tener datos disponibles para verificar que la funcionalidad del sistema sea la misma que se solicita en los requerimientos.

Los datos de las pruebas de caja blanca, generadas para probar módulos básicos, se especificaron considerando cada una de las rutas de ejecución del programa, así como diversos valores fuera de los intervalos lógicos para observar el comportamiento del sistema cuando el usuario comete un error al introducir los datos. En algunos programas se efectuaron pruebas con rutinas que generan polinomios y matrices aleatorios.

Los datos de las pruebas de caja negra utilizando NOLIACPA se realizaron con ejemplos de sistemas predefinidos en este programa, debido a que precisamente fueron elegidos por su grado de dificultad.

Los datos de las pruebas de escritorio se escogieron de problemas reales, así como de ejemplos académicos disponibles en la literatura. Sin embargo, debido a la complejidad de los cálculos involucrados y la dificultad de realizarlos a mano los sistemas elegidos fueron ejemplos con un menor número de variables que los utilizados en la prueba anterior.

Los ejemplos de estas pruebas se describen en el Apéndice B junto con los datos de prueba considerados y los resultados obtenidos en cada caso.

IV.6.3 Resultados

Las pruebas realizadas al sistema SAC sirvieron para detectar defectos en este. Todas las pruebas resultaron satisfactorias y los defectos encontrados no comprometen la funcionalidad del sistema. Cabe señalar que estas pruebas se realizaron en un ambiente controlado y fueron llevadas a cabo por el desarrollador del sistema, que es la persona con el mejor conocimiento de la herramienta y con experiencia en el uso de esta, además de un experto en el área de sistemas no lineales con retardos.

El número de errores es relativamente bajo, debido a que durante todo el proceso de implementación se realizaron pruebas al sistema, lo que permitió llegar al final con el menor número de defectos. También se atribuye este hecho a que el análisis y diseño de SAC se llevo a cabo siguiendo la metodología de la ingeniería de software.

A continuación se listan las pruebas realizadas y los defectos encontrados.

Tabla VII: Resultados de pruebas de condición.

Defecto	Se detectaron programas en los que hacía falta especificar mensajes de error para datos inválidos.
Solución	Se añadieron los mensajes faltantes.
Comentario	Estos mensajes faltantes no comprometían la integridad de SAC ya que Maxima enviaba un mensaje de error. Sin embargo se considera entre los defectos ya que faltaba claridad a estos mensajes. Ya fue corregido

Tabla VIII: Resultados de pruebas generadas mediante datos aleatorios.

Defecto	Tiempo de ejecución alto en la rutina de división euclidiana *euclid*.
Solución	Debido a que esta rutina depende directamente de la rutina $tshift$ de corrimiento en tiempo, se optó por mejorar el rendimiento de esta cambiándola a lenguaje *lisp*.
Comentario	Este problema es inherente al algoritmo de división euclidiana utilizado, por lo que en un trabajo futuro se sugiere modificar este algoritmo, ya que existen versiones más rápidas disponibles en la literatura. Sin embargo este defecto no afecta la funcionalidad del programa ya que los resultados obtenidos son correctos. No se corrigió. Únicamente se mejoró el tiempo de ejecución.
Defecto	Tiempo de ejecución alto en la rutina $NCProduct$, que calcula el producto no conmutativo $a * \wedge b$.
Solución	El problema es el tiempo de ejecución de la instrucción $freeof$ de Maxima. Se llegó a la conclusión de que esta rutina se requiere únicamente en sistemas con retardos por lo que se modificó el programa, corrigiendo el problema para sistemas sin retardos.
Comentario	La rutina $freeof$ se utiliza para encontrar la diferencial de una variable $dx(t)$ (representada en Maxima como $DEL(x(t))$). Esta instrucción se utiliza en sistemas con retardos por el uso del operador ∇, ya que $\nabla * \wedge dx(t) = dx(t-1)$, a diferencia de $\nabla * \wedge x(t) = x(t-1)\nabla$. Se corrigió para sistemas sin retardos, para problemas con retardos el problema persiste.

Tabla IX: Resultados de pruebas de integración.

Comentario	No se presentó ningún defecto.

Tabla X: Resultados de pruebas de comparación con NOLIACPA.

Comentario	No se presentó ningún defecto en las rutinas de accesibilidad y observabilidad para sistemas sin retardos. El tiempo de ejecución no se puede comparar debido a que NOLIACPA calcula todos los problemas de control en una misma rutina.

Tabla XI: Resultados de pruebas de escritorio.

Comentario	No se presentó ningún defecto.

IV.6.4 Resumen

En este capítulo se relacionaron los conceptos preliminares vistos en el Capítulo II con cada uno de los problemas a resolver en SAC. Se desarrollaron a detalle los algoritmos para resolver problemas de accesibilidad, observabilidad, equivalencia con la forma triangular, linealización por inyecciones aditivas entrada/salida y rechazo de perturbaciones.

También se presentaron las estrategias de prueba para asegurar el correcto funcionamiento del programa, que si bien no fueron exhaustivas, si mostraron el buen funcionamiento de SAC que llegó a resultados correctos en las diversas pruebas.

Capítulo V

Conclusiones

El programa mostrado es la primera versión de una herramienta de ayuda para aplicar los resultados teóricos disponibles para sistemas no lineales con retardos, bajo un enfoque algebraico; resolviendo problemas de análisis como accesibilidad del estado, observabilidad, equivalencia del sistema con la forma triangular, linealización E/S por inyecciones aditivas y el máximo submódulo invariante ante entradas de control y perturbaciones.

SAC fue desarrollado bajo licencia GNU GPL lo cual permite contribuciones de la comunidad y facilita su constante desarrollo ya que se busca que se convierta en una herramienta de propósito general para el análisis y control de sistemas no lineales con y sin retardos.

La elección de cada módulo que conforma la estructura del programa se llevó a cabo a partir de las metodologías de la ingeniería de software, lo que ha permitido una mejor organización.

Las pruebas de caja blanca utilizadas fueron pruebas de la estructura de control, particularmente pruebas de condición, en las cuales se verifica cada una de las condiciones lógicas contenidas en un módulo del programa. Estas pruebas comprenden tanto pruebas de unidad, como de integración y los resultados obtenidos en ambas pruebas fueron satisfactorios.

Las pruebas de caja negra fueron divididas en dos casos, para sistemas no lineales sin retardos y con retardos. En el primer caso fue posible una comparación con el programa NOLIACPA para los problemas en común, la accesibilidad y la observabilidad,

y se obtuvieron iguales resultados. El resto de las rutinas así como los sistemas con retardos se compararon con resultados obtenidos mediante pruebas de escritorio utilizando sistemas físicos y académicos encontrados en la literatura.

En las pruebas aplicadas, aún cuando no fueron exhaustivas, SAC mostró ser un sistema estable al llegar al resultado esperado. En el caso de sistemas no lineales sin retardos tuvo un buen desempeño respecto al tiempo de operación.

V.1 Aportaciones

La principal aportación fue sentar las bases para el inicio de un proyecto ambicioso, el desarrollo de una herramienta completa para análisis y síntesis para sistemas lineales, no lineales con y sin retardos, además de sistemas en tiempo discreto mediante un enfoque algebraico de control.

V.2 Trabajo futuro

Respecto al trabajo futuro se propone implementar un algoritmo de división euclidiana más rápido para sistemas con retardos lo que implica que el tiempo de ejecución de SAC mejore considerablemente, así como realizar pruebas exhaustivas al sistema.

Elaborar rutinas para la síntesis de sistemas no lineales con retardos, tales como linealización entrada/salida, entrada/estado, rechazo de perturbaciones.

Realizar rutinas para el análisis de estabilidad mediante funciones de Lyapunov, discretización en tiempo.

Extender el uso de SAC a otra clase de sistemas, como por ejemplo sistemas en tiempo discreto.

Bibliografía

Abdallah, C., P. Dorato, J. Benitez-Read, y R. Byrne 1993. "Delayed positive feedback can stabilize oscillatory systems". En: "Proc. IEEE American Control Conference", San Francisco, CA., U.S.A. Junio 2-4. 3106–3107 p.

Aranda-Bricaire, E., U. Kotta, y C. Moog 1996. "Linearization of discrete-time systems". SIAM J. Control & Opt., 34(6):1999–2023 p.

Aranda-Bricaire, E., C. Moog, y J.-B. Pomet 1995. "A linear algebraic framework for dynamic feedback linearization". IEEE Trans. on Automatic Control, 40(1):127–132 p.

Azorin, J., O. Reinoso, R. Aracil, y M. Ferre 2004. "Generalized control method by state convergence for teleoperation systems with time delay". Automatica, 40(9):1575–1582 p.

Bass, L., P. Clements, y R. Kazman 2003. "Software architecture in practice". Addisson-Wesley. 452 p.

Bennet, S., J. Skelton, y K. Lunn 2004. "Schaum's outline of uml". Schaum's Outline Series. McGraw-Hill. 380 p.

Britton, C. y J. Doake 1993. "Software system development: a gentle introduction". McGrawHill. 214 p.

Conte, G., C. Moog, y A. Perdon 1999. "Nonlinear control systems: an algebraic setting". Lect. Notes Control and Inf. Sciences. Springer–Verlag. 242:166 p.

El'sgol'ts, L. y S. Norkin 1973. "Introduction to the theory and application of differential equations with deviating arguments". Academic Press, New York. 357 p.

Fridman, E. 2002. "Effects of small delays on stability of singularly perturbed systems". Automatica, 38(5):897–902 p.

Hale, J. y S. Lunel 1999. "Effects of small delays on stability and control". Reporte Técnico WS-528, Vrije University, Amsterdam. 27 p.

Jalili, N. y N. Olgac 4:1998. "Optimum delayed feedback vibration absorber for MDOF mechanical structures". En: "Proc. of the 37th IEEE Conference on Decision and Control", Tampa Fl. U.S.A.. Diciembre 16-18. 4734–4739 p.

Kahrimanian, H. G. 1953. "Analytical differentiation by a digital computer". Tesis de Maestría, Temple Univ., Philadelphia. 48 p.

Khalil, H. 2002. "Nonlinear systems". Prentice Hall, 3ra. edición. 750 p.

Kolmanovskii, V. y A. Myshkis 1999. "Introduction to the theory and applications of functional differential equations.". Ed. Kluwer. 648 p.

Kotta, S. N. Ü. y M. Tõnso 2005. "Realization of nonlinear discrete-time composite systems: computational aspects". En: "Proc. IFAC 16th World Congress", Praga República Checa. Julio 4-8.

Kotta, U. y M. Tõnso 1998. "Towards a symbolic computation toolbox for transforming a generalized state equations into a classical form". En: "Proceedings of the 13th International Conference on Systems Science", Wroclaw, Poland. Diciembre 2-4. 1:121–128 p.

Manfredi, P. y L. Fanti 2004. "Cycles in dynamic economic modelling". Automatica, 21(3):573–594 p.

Márquez-Martínez, L. 1999. "Note sur l'accessibilité des systèmes non linéaires à retards". C.R. Acad. Sci. Paris, 329(6):545–550 p.

Márquez-Martínez, L. 2000. "Analyse et commande de systèmes non linéaires à retards". Tesis de Doctorado, Université de Nantes, Nantes, France. 134 p.

Márquez-Martínez, L. 2004. "Computational aspects of the analysis of nonlinear time-delay systems". En: "Proc. IFAC 2nd Symposium on System, Structure and Control". 198–203 p.

Márquez-Martínez, L. y C. Moog 1999. "New results on the analysis and control of nonlinear time-delay systems". En: "Proc. IEEE Conf. on Dec. Control", Phoenix, USA. Diciembre 7-10. 4240–4244 p.

Márquez-Martínez, L. y C. Moog 2001a. "Accessibility of nonlinear time-delay systems". En: "40th IEEE Conf. on Decision Control", Orlando, Florida. Diciembre 4-7. 4622 – 4627 p.

Márquez-Martínez, L. y C. Moog 2001b. "Trajectory tracking control for nonlinear time–delay systems". Kybernetika, 37(4):453–463 p.

Márquez-Martínez, L. y C. Moog 2004. "New insights on the analysis of nonlinear time-delay systems". En: "Proc. of the 43rd IEEE Conference on Decision Control". 4533–4537 p.

Márquez-Martínez, L., C. Moog, y E. Aranda-Bricaire 2001. "Triangular forms for nonlinear time-delay systems". En: "3rd IFAC workshop on time delay systems", Santa Fe, New Mexico. Diciembre 8-10. 261 – 265 p.

Moog, C., R. Castro-Linares, M. Velasco-Villa, y L. Márquez-Martínez 2000. "Disturbance decoupling for time-delay nonlinear systems". IEEE Trans. on Automatic Control, 48(2):305–309 p.

Mounier, H. y R. Rudolph 1998. "Flatness based control of nonlinear delay systems: a chemical reactor example". Int. J. of Ctrl., 71:871–890 p.

Márquez-Martínez, L. y C. Moog 2007. "New insights on the analysis of nonlinear time-delay systems: application to the triangular equivalence". Systems & Control Letters, 56(2):133–140 p.

Newton, I. 1728. "Arithmetica universalis". London. 332 p.

Nolan, J. 1953. "Analytical differentiation on a digital computer". Tesis de Maestría, Mass. Inst. Technology. 71 p.

Pressman, R. S. 2005. “Ingeniería del software: un enfoque práctico”. McGrawHill. 958 p.

Qiang Zhang, Xiaopeng Wei, J. X. 2005. “An improved result for complete stability of delayed cellular neural networks”. Automatica, 41:333–337 p.

Richard, J.-P. 2003. “Time-delay systems: an overview of some recent advances and open problems”. Automatica, 39:1667–1694 p.

Smith, O. 1957. “Closer control of loops with deadtime”. Chem. Eng. Proc., 53(5):217–221 p.

Smith, O. 1959. “A controller to overcome dead time”. ISA Journal of Instrument Society of America, 6(2):28–33 p.

Sontag, E. D. 1991. “Mathematical control theory: deterministic finite dimensional systems”. Springer. 531 p.

Vignaud, V. 1997. “Calcul symbolique et automatique non lineaire: Noliacpa”. Reporte técnico , Ecole Centrale de Nantes. 125 p.

Xia, X., L. A. Márquez-Martínez, P. Zagalak, y C. Moog 2002. “Analysis of nonlinear time-delay systems using modules over non-commutative rings”. Automatica, 38(9):1549–1555 p.

Yu-Ping, T. 2005. “A general stability criterion for congestion control with diverse communication delays”. Automatica, 41(7):1255–1262 p.

Apéndice A

Documentación de análisis y diseño de SAC.

La siguiente información muestra la documentación completa de análisis y diseño del sistema. Se comienza a partir de los principales casos de uso extraídos del análisis de los requerimientos del cliente y del diagrama de contexto, mostrados en las figuras 4 y 14, respectivamente.

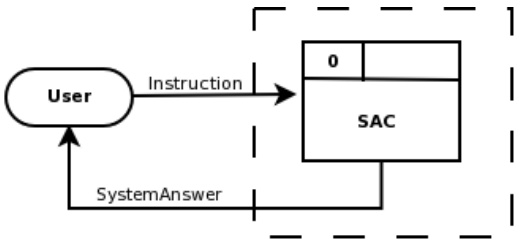
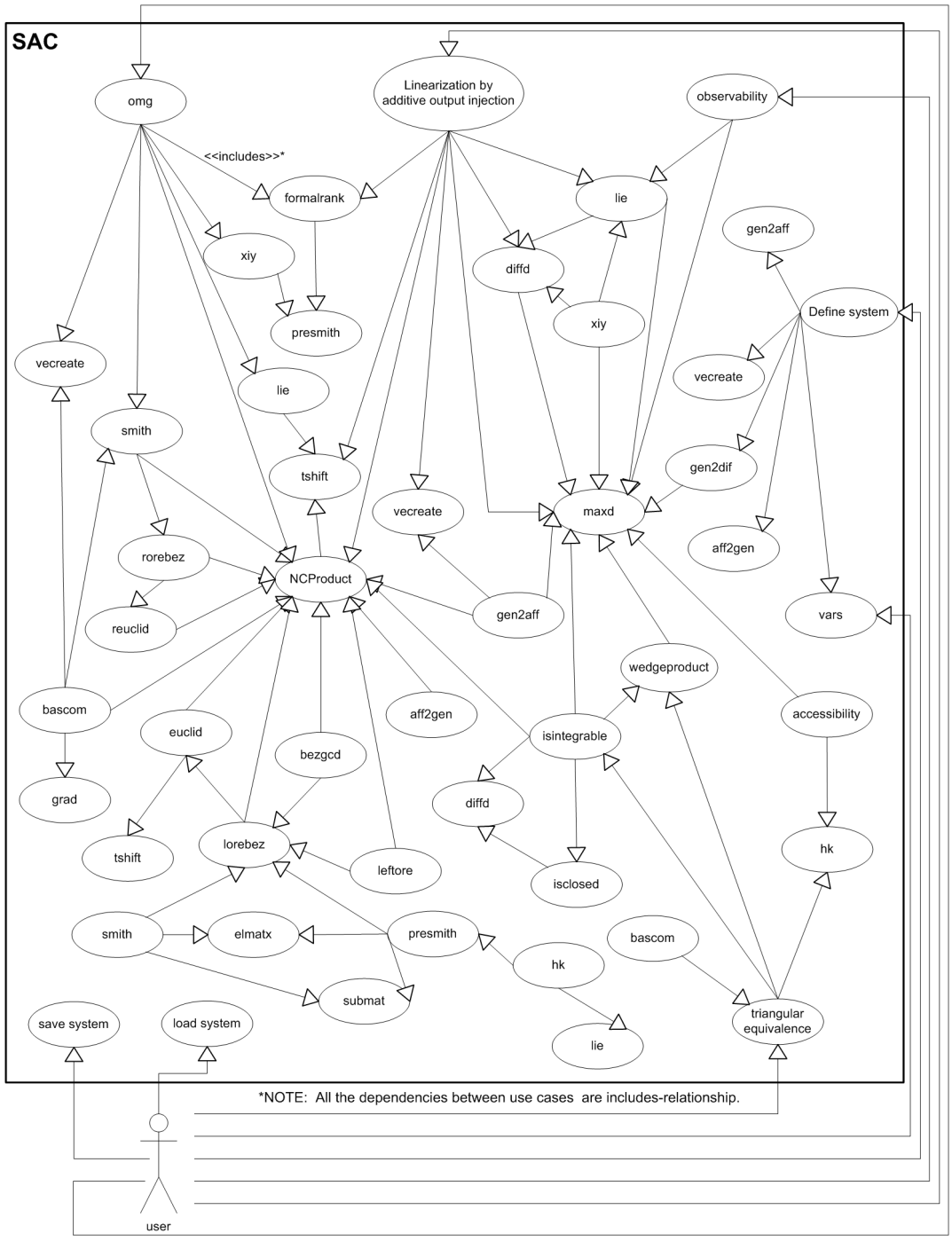


Figura 14: Diagrama de contexto. Nivel 0.

A partir de este diagrama de contexto, que muestra el programa como un todo, se extienden los distintos problemas a resolver, basados en los casos de uso principales. La documentación se ordena por orden alfabético, de acuerdo al tipo de diagrama. Debido a que la intención es colaborar con gente de otros países, esta documentación se hizo en inglés.

A.1 Casos de uso

A.1.1 Diagrama de casos de uso



A.1.2 Documentos de casos de uso

Use case: accessibility.

Actors: user

Purpose: identify if a nonlinear control system with or without delays is accessible.

Description: The user defines the equations system and the routine answers are *true*, *false* or *cannot determinate*.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. isaccessible(system_name)	2. Computes the $\mathcal{H}_{\mathcal{K}}$ submodules. 3. Verify $\mathcal{H}_{\infty}$: - if $\mathcal{H}_{\infty} = 0$ the system is accessible and the answer is *true*; - if $\mathcal{H}_{\infty} \neq 0$ and the system does not have delays the answer is *false*; - if $\mathcal{H}_{\infty} \neq 0$ and the system has delays there are not enough conditions to determinate if the system is accessible and the routine returns *cannot determinate*.

Use case: aff2gen.

Actors: user.

Purpose: computes the variable sys_Fg of the general form of representation (1), starting from sys_F, sys_G of the affine form (2).

Description: the user writes $sys\_F \in \mathbb{R}^{n\times 1}$, $sys\_G \in \mathbb{R}^{n\times m}$ and the control variable ($varcon$). The system computes $sys\_Fg \in \mathbb{R}^{n\times 1}$. The elements of the matrix sys_G could depend on the ∇ operator.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. aff2gen($sys\_F, sys\_G, varcon$)	2. Creates the vector $\begin{bmatrix} varcon_1 & \dots & varcon_n \end{bmatrix}^T$. 3. Multiplies the matrix sys_G for the vector of the step 2 applying the non commutative product ($*\wedge$) and replaces $\nabla = 1$. 4. sys_Fg will be the sum of the answer of the step 3 and the matrix f. 5. Returns sys_Fg.

Use case: bascom.

Actors: user.

Purpose: Computes the basis completion of a submodule.

Description:computes a new vector to complete a basis taking the subspace $\mathcal{H}_k$ (sb), $\mathcal{H}_{k-1}$ (sba) and the state variable basis to derivate (x).

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. bascom(sb, sba, x)	2. Define r as the number of vectors in $\mathcal{H}_k$, $\alpha = r + 1$ and n as the dimensionof the state variable.
	3. Let $\mathcal{H}_{k-1} := span_{\mathcal{K}[\nabla]}\{\omega_1, \ldots, \omega_\alpha\}$. Computes the matrix $A[\nabla]$ such that: $$sba = \begin{bmatrix} \frac{\partial \omega_1}{\partial x_1} & \cdots & \frac{\partial \omega_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \omega_\alpha}{\partial x_1} & \cdots & \frac{\partial \omega_\alpha}{\partial x_n} \end{bmatrix} \mathbf{dx} = A[\nabla] \cdot \mathbf{dx}$$ with $\mathbf{dx} := \begin{bmatrix} dx_1, \ldots, dx_n \end{bmatrix}^T$.
	4. Computes the Smith form of $A[\nabla]$: $$Smith(A[\nabla]) = P[\nabla] \cdot A[\nabla] \cdot Q[\nabla] = S$$ $$A[\nabla] = P'[\nabla] \cdot S \cdot Q'[\nabla]$$
	5. Completes a basis for $\mathcal{H}_{k-1}$ computing the vector $\tilde{\omega}$: $$\tilde{\omega} = \begin{bmatrix} 0_{(\alpha-(\alpha-1)) \times (\alpha-1)} & \mathbb{I}_{(\alpha-(\alpha-1))} \end{bmatrix} Q'[\nabla] \mathcal{H}_l$$ $$\tilde{\omega} \cdot \mathbf{dx} = \begin{bmatrix} \tilde{\omega}_1 \\ \vdots \\ \tilde{\omega}_{(\alpha-(\alpha-1))} \end{bmatrix}$$ such that if $\mathcal{H}_k$ is $$\mathcal{H}_k := span_{\mathcal{K}[\nabla]}\{\omega_1, \ldots, \omega_{(\alpha-1)}\},$$ then the new submodule will be $$\mathcal{H}_{k-1} := \text{span}_{\mathcal{K}[\nabla]}\{\omega_1, \ldots, \omega_{(\alpha-1)}, \tilde{\omega}_1, \ldots, \tilde{\omega}_{(\alpha-(\alpha-1))}\}$$
	6. Returns the new basis $\mathcal{H}_{k-1}$.

Use case: bezgcd.

Actors: user.

Purpose: computes the Bezout polynomials or the polynomials to get the greatest common divisor (GCD) of $a[\nabla], b[\nabla]$.

Description: The user writes the polynomials $a[\nabla], b[\nabla] \in \mathcal{K}[\nabla]$ with $b[\nabla] \neq 0$. The system returns the polynomials $a'[\nabla], b'[\nabla] \in \mathcal{K}[\nabla]$ such that $a'[\nabla]a[\nabla] + b'[\nabla]b[\nabla] = d[\nabla]$. The result $d[\nabla] = 1$ if $a'[\nabla], b'[\nabla]$ are the Bezout polynomials otherwise $d[\nabla]$ =GCD.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. bezgcd($a[\nabla], b[\nabla]$)	2. Computes the PQ matrix. 3. Extracts the polynomials such that $a'[\nabla]a[\nabla] + b'[\nabla]b[\nabla] = d[\nabla]$. 4. Returns $\begin{bmatrix} a'[\nabla] & b'[\nabla] \end{bmatrix}$.

Use case: define system.

Actors: user.

Purpose: assigns a name to the equations system, which could be written, either in general, affine or differential form. When a system is defined the software attempts to compute the other representations and store them temporarily during the Maxima session.

Description: the user specify the representation form (*type*), the system name (*sys*), the equations system (*eq*) and the list with the state, control and output variables (*vars*). The system creates matrixes to store the data of each representation form.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. systdef("type",sys,[eq],[vars])	2. The system computes the data according to the representation form: - 2a. type="affine": the matrixes to store the affine representation form are created (*sys_F*, *sys_G* and *sys_H*) from which the system computes the general form (*sys_Fg* and *sys_Hg*) and finally the differential form (*sys_dF*, *sys_dG* and *sys_dH*). The state, output, control variables are stored in *sys_var_sta sys_var_out* and *sys_var_con* respectively. - 2b. type="general": the matrixes to store the general form (*sys_Fg* and *sys_Hg*) of representation are created from which the system computes the affine form (*sys_F*, *sys_G* and *sys_H*) and finally the differential form of representation (*sys_dF*, *sys_dG* and *sys_dH*). The state, output, control variables are stored in *sys_var_sta sys_var_out* and *sys_var_con* respectively. - 2c. type="differential": the matrixes to store the differential form of representation are created (*sys_dF*, *sys_dG* and *sys_dH*). The state, output and control variables are stored in *sys_var_sta sys_var_out* and *sys_var_con* respectively. 3. The system returns the matrixes according to the representation form.

Use case: diffd.

Actors: user.

Purpose: computes the differential form of f with respect to x and u.

Description: the program depends on the Cartan package which computes the differential form without delays. The user writes f, and the basis variables (x, u). The system returns the differential form of f.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. diffd(f, x, u)	2. Makes the $list1$ with all the basis variables contained in a and b. Remember that $x_1(t)$ is independent of $x_1(t-1)$.
	3. Creates the $list2$ relating each variable of the $list1$ with a single letter and substitutes them in f.
	4. Loads the Cartan package.
	5. Computes the differential form between a and b $(ext\_diff(f))$.
	6. Returns the result obtained in 5 substituting the variables of the $list2$ for its equivalence in the $list1$.

Use case: elmatx.

Actors: user.

Purpose: creates the elementary matrix to interchange two rows or columns in a matrix of dimension $m \times n$.

Description: the user writes the identification number of the rows or columns to interchange (e_1, e_2) and the matrix dimension (dim). The system returns the elementary matrix.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. elmatx(e_1,e_2,dim)	2. Returns the elementary matrix.

Use case: euclid.

Actors: user.

Purpose: computes the Euclidien division by the left between $a[\nabla], b[\nabla] \in \mathcal{K}[\nabla]$.

Description: finds the polynomials $c[\nabla]$ and $r[\nabla] \in \mathcal{K}[\nabla]$, where $a[\nabla] = c[\nabla]b[\nabla] + r[\nabla]$, with $b[\nabla] \neq 0$ and the polynomial degree of $r[\nabla]$ smaller than the polynomial degree of $b[\nabla]$.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. euclid$(a[\nabla], b[\nabla])$	2. Left divides $a[\nabla]$ by $b[\nabla]$.
	3. Returns the quotient $c[\nabla]$ and the reminder $r[\nabla]$.

Use case: formalrank.

Actors: user.

Purpose: computes the generic rank of a matrix.

Description: the user writes the matrix name (*mat*), and the system returns the generic rank.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. formalrank(mat)	2. Gets the formal rank computing the presmith form. 3. Returns the generic rank.

Use case: gen2aff.

Actors: user.

Purpose: computes the affine form of representation, sys_F and sys_G, starting from the general form, sys_Fg.

Description: the user writes $sys\_Fg \in I\!R^{n\times 1}$ and the matrixes with the state and control variables. The system computes $sys\_F \in I\!R^{n\times 1}$ and $sys\_G \in I\!R^{n\times m}$.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. gen2aff($sys\_Fg, varsta, varcon$)	2. Stores the maximum time delay of sys_Fg in r. 3. Computes the matrix: $$sys\_G = \sum_{i=0}^{r} \begin{bmatrix} \frac{\partial sys\_Fg_1}{\partial con_1(t-i)} & \frac{\partial sys\_Fg_1}{\partial con_2(t-i)} & \cdots & \frac{\partial sys\_Fg_1}{\partial con_m(t-i)} \\ \frac{\partial sys\_Fg_2}{\partial con_1(t-i)} & \frac{\partial sys\_Fg_2}{\partial con_2(t-i)} & \cdots & \frac{\partial sys\_Fg_2}{\partial con_m(t-i)} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial sys\_Fg_n}{\partial con_1(t-i)} & \frac{\partial sys\_Fg_n}{\partial con_2(t-i)} & \cdots & \frac{\partial sys\_Fg_n}{\partial con_m(t-i)} \end{bmatrix} \nabla^i$$ 4. $sys\_F = sys\_Fg - sys\_G * \wedge[varcon_1(t), \ldots, varcon_n(t)]^T$. 5. Replace ∇ for $\nabla = 1$ in sys_F. If sys_F has the variable $varcon$ then the system returns "non affine", otherwise goes to 6. 6. Returns the variable $\left[sys\_F \mid sys\_G\right]$ of dimension $\mathbb{R}^{n\times(m+1)}$.

Use case: gen2dif.

Actors: user.

Purpose: computes the differential form of representation (9), sys_dF and sys_dG, starting from the general form (1), sys_Fg.

Description: the user writes $sys\_Fg \in I\!R^{n\times 1}$ and the matrixes with the state and control variables. The system computes $sys\_dF \in I\!R^{n\times n}$ and $sys\_dG \in I\!R^{n\times m}$.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. gen2dif($sys\_Fg, varsta, varcon$)	2. Stores the maximum time delay of sys_Fg in r.
	3. Computes the matrix: $$sys\_dF = \sum_{i=0}^{r} \begin{bmatrix} \frac{\partial sys\_Fg_1}{\partial varsta_1(t-i)} & \frac{\partial sys\_Fg_1}{\partial varsta_2(t-i)} & \cdots & \frac{\partial sys\_Fg_1}{\partial varsta_n(t-i)} \\ \frac{\partial sys\_Fg_2}{\partial varsta_1(t-i)} & \frac{\partial sys\_Fg_2}{\partial varsta_2(t-i)} & \cdots & \frac{\partial sys\_Fg_2}{\partial varsta_n(t-i)} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial sys\_Fg_n}{\partial varsta_1(t-i)} & \frac{\partial sys\_Fg_n}{\partial varsta_2(t-i)} & \cdots & \frac{\partial sys\_Fg_n}{\partial varsta_n(t-i)} \end{bmatrix} \nabla^i$$
	4. Computes the matrix: $$sys\_dG = \sum_{i=0}^{r} \begin{bmatrix} \frac{\partial sys\_Fg_1}{\partial varcon_1(t-i)} & \frac{\partial sys\_Fg_1}{\partial varcon_2(t-i)} & \cdots & \frac{\partial sys\_Fg_1}{\partial varcon_m(t-i)} \\ \frac{\partial sys\_Fg_2}{\partial varcon_1(t-i)} & \frac{\partial sys\_Fg_2}{\partial varcon_2(t-i)} & \cdots & \frac{\partial sys\_Fg_2}{\partial varcon_m(t-i)} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial sys\_Fg_n}{\partial varcon_1(t-i)} & \frac{\partial sys\_Fg_n}{\partial varcon_2(t-i)} & \cdots & \frac{\partial sys\_Fg_n}{\partial varcon_m(t-i)} \end{bmatrix} \nabla^i$$
	5. Returns the variable $\left[sys\_dF \mid sys\_dG\right]$ of dimension $\mathbb{R}^{n\times(n+m)}$.

Use case: gradient.

Actors: user.

Purpose: Computes the gradient of the function f with respect to the variable x.

Description: the user writes the function to derivate and the list of variable basis. The system returns the gradient.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. grad(f, x)	2. Returns the gradient of f with respect to the list of variables contained in x.

Use case: hk.

Actors: user.

Purpose: computes the filtration of $\mathcal{H}_k$ submodules.

$$\mathcal{H}_k = [g_1, \ldots, g_{k-1}]^{\perp}$$

where:

$$\begin{aligned} g_1 &= g[\nabla] \\ g_{i+1} &= F[\nabla]g_i - \dot{g}_i, \end{aligned}$$

$F[\nabla]$ and $g[\nabla]$ are defined by (9).

Description: the user writes the name assigned to the equations system (*sys*) and the word *done* is being displayed when the command computes the calculations. The value of each subspace will be shown specifying the command *syshk_i*, with $i := \{1, 2, \ldots, n, inf\}$.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. hk(*sys*)	2. Gets the matrix $g_1 = g[\nabla]$, $i = 1$. 3. Computes the presmith form of g_i, which returns the matrixes P, Q, R and the formal rank of g_i. 4. The annihilator matrix is obtained on basis of the formal rank of R: - If $rank(R) = n$ then $\mathcal{H}_\infty = 0$ and all the submodules from k (the actual submodule) thru n are zero too. The system goes to 7. - If $rank(R) < n$ then: $\mathcal{H}_{i+1} = g_i^{\perp} = \begin{bmatrix} P_{r+1,1} & \ldots & P_{r+1,n} \\ \vdots & \ddots & \vdots \\ P_{n,1} & \ldots & P_{n,n} \end{bmatrix}$ 5. Computes $\dot{g}_i$ (the lie derivative of g_i). 6. Computes $g_{i+1} = \begin{bmatrix} g_i & F[\nabla]g_i - \dot{g}_i \end{bmatrix}, i = i + 1$. 7. If $i \leq n$ then goes to step 3, else the system returns *done*.
8. syshk_#, with $\# = \{2, \ldots, n, inf\}$.	9. The system returns the submodule $\mathcal{H}_\#$.

Use case: isclosed.

Actors: user.

Purpose: determines if a 1-form is closed.

Description: a 1-form $v \in \mathcal{E}$ is closed if $dv = 0$. The user writes the 1-form and the variable basis to derivate $(varsta, varcon)$, one of this variables could be zero if the user wants to differentiate with respect to only one. The system returns true if the 1-form is closed or false otherwise.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. isclosed$(v, varsta, varcon)$	2. Differentiate v with respect to $varsta$ and $varcon$. 3. Returns true if the result of the step 2 is zero, otherwise returns false.

Use case: isintegrable.

Actors: user.

Purpose: Determines if f is integrable with respect to the variables $varst$ or/and $varco$.

Description: the user writes f, and the basis variables $(varst, varcon)$. The system returns true if the system is integrable and false otherwise.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. isintegrable(f, x, u)	2. Verify if f is a matrix, if it is not goes to 3, otherwise goes to 4. 3. Computes if f is integrable using a particular case of the Frobenius theorem: $\Omega := df \wedge f$. Returns $true$ if $\Omega = 0$, otherwise returns $false$. 4. Let n the number of columns of f. Then creates the vector $\mathbf{dx} := \begin{bmatrix} dx_1, \ldots, dx_n \end{bmatrix}^T$ and multiplies $f = f * \wedge \mathbf{dx}$. 5. Computes the maximum time delayof f and stores it in k. 6. Verify if $f = \begin{bmatrix} \omega_1 & \omega_2 & \ldots & \omega_n \end{bmatrix}^T$ is integrable: $\Omega := d\omega_n \wedge \omega_n \wedge \omega \wedge \cdots \wedge \nabla^k \omega, \quad k \in \mathbb{N}, \ \omega := \omega_1 \wedge \cdots \wedge \omega_{(n-1)}$ Returns $true$ if $\Omega = 0$, otherwise returns $false$.

Use case: leftore.

Actors: user.

Purpose: Computes the left Ore polynomials of $a[\nabla], b[\nabla] \in \mathcal{K}[\nabla]$. These are polynomials $a'[\nabla], b'[\nabla]$, not both zero, such that $a'[\nabla]a[\nabla] = b'[\nabla]b[\nabla]$.

Description: The user writes the polynomials $a[\nabla], b[\nabla] \in \mathcal{K}[\nabla]$ with $b[\nabla] \neq 0$ and the system returns the Ore polynomials.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. leftore$(a[\nabla], b[\nabla])$	2. Computes the PQ matrix. 3. Extracts the polynomials such that $a'[\nabla]a[\nabla] + b'[\nabla]b[\nabla] = 0$. 4. Returns $\begin{bmatrix} a'[\nabla] & b'[\nabla] \end{bmatrix}$.

Use case: lie.

Actors: user.

Purpose: computes the n'th. Lie-derivative of h along the trajectories of the system sys. The default value of n is one if is not specified.

Description: the equations system in matrix form are:

$$\begin{bmatrix} \dot{x}_1(t) \\ \vdots \\ \dot{x}_n(t) \end{bmatrix} = \begin{bmatrix} f_1(x(t-i), u(t-i)) \\ \vdots \\ f_n(x(t-i), u(t-i)) \end{bmatrix}$$

where $i \in \underline{S} := \{0,1,\ldots,s\}$

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. lie(h, sys, n) or lie(h, sys)	2. Stores in r the maximum time delay between the system sys and h. 3. Creates a list of equations shifted from 0 thru r and their differential forms.

$$\begin{aligned}
\dot{x}_1(t) &= f_1(x(t-i), u(t-i)) \\
\dot{x}_1(t-1) &= f_1(x(t-i-1), u(t-i-1)) \\
&\vdots \\
\dot{x}_1(t-r) &= f_1(x(t-i-r), u(t-i-r)) \\
\dot{x}_2(t) &= f_2(x(t-i), u(t-i)) \\
&\vdots \\
\dot{x}_n(t) &= f_n(x(t-i), u(t-i)) \\
\dot{x}_n(t-1) &= f_n(x(t-i-1), u(t-i-1)) \\
&\vdots \\
\dot{x}_n(t-r) &= f_n(x(t-i-r), u(t-i-r)) \\
d\dot{x}_1(t) &= df_1(x(t-i), u(t-i)) \\
d\dot{x}_1(t-1) &= df_1(x(t-i-1), u(t-i-1)) \\
&\vdots \\
d\dot{x}_1(t-r) &= df_1(x(t-i-r), u(t-i-r)) \\
d\dot{x}_2(t) &= df_2(x(t-i), u(t-i)) \\
&\vdots \\
d\dot{x}_n(t) &= df_n(x(t-i), u(t-i)) \\
d\dot{x}_n(t-1) &= df_n(x(t-i-1), u(t-i-1)) \\
&\vdots \\
d\dot{x}_n(t-r) &= df_n(x(t-i-r), u(t-i-r))
\end{aligned}$$

4. Differentiates h and substitutes the equation list created in the step 3.

5. Computes $n = n - 1$. If $n \geq 1$ goes to step 2, otherwise goes to 6.

6. Returns the lie derivative founded in the step 4.

Use case: linearization by additive output injections

Actors: user.

Purpose: computes the linearization by additive output injections.

Description: the user writes the name assigned to the equations system and the system returns *true* if it is linearizable, otherwise returns $false$.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. inj(sys)	2. Returns $false$ if $dy^{(\bar{n})} \notin \mathcal{E}^{(\bar{n})}$, otherwise goes to step 3.
	3. Denote $\omega_1 = dy^{(\bar{n})}$. Picks functions $\xi_1, \theta_1 \in \mathcal{K}[\nabla]$ such that $$\omega_1 - \xi_1 dy^{(\bar{n}-1)} - \theta_1 du^{(\bar{n}-1)} \in \mathcal{E}^{(\bar{n}-1)}.$$ where $$\mathcal{E}^k = span_{\mathcal{K}[\nabla]}\{dy(t), \ldots, dy^{(k-1)}(t), du(t), \ldots, du^{(k-1)}(t)\}.$$
	4. Defines the differential one-form $\bar{\omega}_1$ as $$\bar{\omega}_1 = \xi_1 dy + \theta_1 du.$$
	5. Returns $false$ if $d\bar{\omega}_1 \neq 0$, otherwise goes to 6.
	6. Defines $l = 2$. Let $\Phi_{l-1}(y, u)$ be such that $d\Phi_{l-1} = \bar{\omega}_{l-1}$.
	7. Denotes ω_l as $$\omega_l := \omega_{l-1} - d\Phi_{l-1}^{\bar{n}-l+1}.$$
	8. Choose $\xi_l, \theta_l \in \mathcal{K}[\nabla]$ such that $$\omega_l - \xi_l dy^{(\bar{n}-l)} - \theta_l du^{(\bar{n}-l)} \in E^{\bar{n}-l}.$$
	9. Define the differential one-form $\bar{\omega}_l$ as $$\bar{\omega}_l = \xi_l dy - \theta_l du$$
	10. Returns $false$ if $d\bar{\omega}_l \neq 0$, otherwise goes to step 11.
	11. Returns *true* if $l > \bar{n}$, otherwise set $l = l + 1$ and goes to step 6.

Use case: load system.

Actors: user.

Purpose: loads the state space equations of a control system previously stored in the *FileName.sd* file.

Description: the user assigns a new name to the system stored in FileName.sd, and rename the variables SystemName_X and SystemName_var_Y with NewName_X and NewName_var_Y, where X=F, G, H, Fg, Hg, dF, dG, dH and Y=sta, con, out.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. loadsyst(FileName,NewName)	2. The variables SystemName_X and SystemName_var_Y stored in FileName.sd are loaded temporarily during the Maxima session, but with a new name: NewName_X and NewName_var_Y.

Use case: lorebez.

Actors: user.

Purpose: computes the matrix $PQ \in \mathcal{K}^{2\times 2}[\nabla]$, such that $PQ \begin{bmatrix} a[\nabla] \\ b[\nabla] \end{bmatrix} = \begin{bmatrix} 0 \\ d[\nabla] \end{bmatrix}$. This matrix allows to get the Ore and Bezout polynomials.

Description: The user writes the polynomials $a[\nabla], b[\nabla] \in \mathcal{K}[\nabla]$ with $b[\nabla] \neq 0$ and the system returns the PQ matrix.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. lorebez($a[\nabla], b[\nabla]$)	2. Computes an Euclidien division by the left between $a[\nabla]$ and $b[\nabla]$, so one gets $a[\nabla] = c[\nabla]b[\nabla] + r[\nabla]$, defines $PQ_1 = \begin{bmatrix} 1 & -c[\nabla] \\ 0 & 1 \end{bmatrix}$. 3. Apply the left-Euclidien division algorithm to the quotient $c[\nabla]$ and the reminder $r[\nabla]$. 4. Continues dividing the result of each division until it gets a matrix of the form: $$PQ = \prod PQ_i = \begin{bmatrix} P_1[\nabla] & Q_1[\nabla] \\ P_2[\nabla] & Q_2[\nabla] \end{bmatrix}$$ such that, $$PQ \begin{bmatrix} a[\nabla] \\ b[\nabla] \end{bmatrix} = \begin{bmatrix} 0 \\ d[\nabla] \end{bmatrix}$$. 5. Returns the PQ matrix.

Use case: maxd.

Actors: user.

Purpose: Finds the maximum time delay of the function f.

Description: the user writes the function (f) and the system returns the maximum time delay founded. The minimum time delay detected is zero then the result will be zero for a function that has only non causal terms.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. maxd(f)	2. Checks f, if it is an atom, then return 0 and stop. 3. Makes the variable $mdf = 0$. 4. Extracts the time delay of each term of f and if it is greater than mdf substitutes the new result in this variable. 5. Returns mdf.

Use case: NCProduct.

Actors: user.

Purpose: computes the noncommutative product between a and b.

Description: the user writes a and b, which could be polynomials or matrixes. The system returns the product.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. $a * \wedge b$	2. Stores the maximum ∇ exponent of a in r. 3. If a and b are polynomials goes to 4, otherwise goes to 6. 4. If b is free of derivatives (DEL): - $b \leftarrow b$ shifted r time units. - $b \leftarrow b$ multiplied by ∇^r. Otherwise: - Extracts term by term of b. - Shifts each term r time units. - Multiplies by ∇^r each term, except the one which has DEL. - $b \leftarrow$ the sum of all the terms. 5. Returns the multiplication a*b. 6. If the matrix b is free of derivatives (DEL): - $b \leftarrow b$ shifted r time units. - $b \leftarrow b$ multiplied by ∇^r. Otherwise, it has to compute the next steps to each element of the matrix: - Extracts term by term of the element of b. - Shifts each term r time units. - Multiplies by ∇^r each term, except the one which has DEL. - The element of b will be the sum of all the terms. 7. Returns the multiplication $a \cdot b$.

Use case: observability.

Actors: user.

Purpose: Determines if a system described by (1) is observable. Written the equations system in matrix form:

$$\begin{bmatrix} \dot{x}_1(t) \\ \vdots \\ \dot{x}_n(t) \end{bmatrix} = \begin{bmatrix} f_1(x(t-i), u(t-i)) \\ \vdots \\ f_n(x(t-i), u(t-i)) \end{bmatrix}, \qquad \begin{bmatrix} y_1(t) \\ \vdots \\ y_p(t) \end{bmatrix} = \begin{bmatrix} h_1(x(t-i)) \\ \vdots \\ h_p(x(t-i)) \end{bmatrix}$$

where $i \in \underline{S} := \{0,1,\ldots,s\}$

Description: the user writes the name assigned to the equations system (*sys*) and the system returns true or false.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. isobservable (*sys*)	2. Computes the Lie derivative of $[y \ \dot{y} \ \ldots \ y^{(n-1)}]$. 3. Creates the matrix $\triangle$ through the partial differentiation with respect to x:

$$\triangle = \sum_{i=0}^{s} \begin{bmatrix} \frac{\partial h_1(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_1(\cdot)}{\partial x_n(t-i)} \\ \vdots & & \vdots \\ \frac{\partial h_p(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_p(\cdot)}{\partial x_n(t-i)} \\ \frac{\partial h_1^{(1)}(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_1^{(1)}(x(t-i))}{\partial x_n(t-i)} \\ \vdots & & \vdots \\ \frac{\partial h_p^{(1)}(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_p^{(1)}(\cdot)}{\partial x_n(t-i)} \\ & \vdots & \\ \frac{\partial h_1^{(n-1)}(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_1^{(n-1)}(\cdot)}{\partial x_n(t-i)} \\ \vdots & & \vdots \\ \frac{\partial h_p^{(n-1)}(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_p^{(n-1)}(\cdot)}{\partial x_n(t-i)} \end{bmatrix} f(\cdot) \cdot \nabla^i$$

4. Computes the formal rank of $\triangle$:

- If $rank(\triangle) = n$ the system returns *true*.
- If $rank(\triangle) \neq n$ the system returns*false*.

Use case: omg.

Actors: user.

Purpose: computes the maximum submodule which is not affected by any control input or disturbance. The system is described by equations of the form (35).

Description: the user writes the name assigned to the equations system sys and the system returns the submodule Ω.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. omg(sys)	2. Computes $\mathcal{X} \cap \mathcal{Y}$. 3. Defines $k = 0$ and $\Omega_0 := \text{span}_{\mathcal{K}[\nabla]}\{d\mathbf{x}\}$. 4. Computes a basis $\mathbf{b} := [b_1 \dots b_r]^T$ for $\phi = \Omega_k + \frac{d}{dt}(\mathcal{X} \cap \mathcal{Y})$. Let $\begin{bmatrix}\bar{v}_1 & \dots & \bar{v}_z\end{bmatrix}$ be a basis of Ω_k and $\begin{bmatrix}\bar{f}_1 & \dots & \bar{f}_d\end{bmatrix}$ the elements of $\frac{d}{dt}(\mathcal{X} \cap \mathcal{Y})$. $\phi = \triangle\bar{\phi}$ with $\bar{\phi} := \begin{bmatrix}dx_1 & \dots & dx_n & du_1 & \dots & du_m & dq\end{bmatrix}^T$ and $\triangle = \sum_{i=0}^{s} \begin{bmatrix} \frac{\partial \bar{v}_1(\cdot)}{\partial x_1(t-i)} & \dots & \frac{\partial \bar{v}_1(\cdot)}{\partial x_n(t-i)} & \frac{\partial \bar{v}_1(\cdot)}{\partial u_1(t-i)} & \dots & \frac{\partial \bar{v}_1(\cdot)}{\partial u_m(t-i)} & \frac{\partial \bar{v}_1(\cdot)}{\partial q(t-i)} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots \\ \frac{\partial \bar{v}_z(\cdot)}{\partial x_1(t-i)} & \dots & \frac{\partial \bar{v}_z(\cdot)}{\partial x_n(t-i)} & \frac{\partial \bar{v}_z(\cdot)}{\partial u_1(t-i)} & \dots & \frac{\partial \bar{v}_z(\cdot)}{\partial u_m(t-i)} & \frac{\partial \bar{v}_z(\cdot)}{\partial q(t-i)} \\ \frac{\partial \bar{f}_1(\cdot)}{\partial x_1(t-i)} & \dots & \frac{\partial \bar{f}_1(\cdot)}{\partial x_n(t-i)} & \frac{\partial \bar{f}_1(\cdot)}{\partial u_1(t-i)} & \dots & \frac{\partial \bar{f}_1(\cdot)}{\partial u_m(t-i)} & \frac{\partial \bar{f}_1(\cdot)}{\partial q(t-i)} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots \\ \frac{\partial \bar{f}_d(\cdot)}{\partial x_1(t-i)} & \dots & \frac{\partial \bar{f}_d(\cdot)}{\partial x_n(t-i)} & \frac{\partial \bar{f}_d(\cdot)}{\partial u_1(t-i)} & \dots & \frac{\partial \bar{f}_d(\cdot)}{\partial u_m(t-i)} & \frac{\partial \bar{f}_d(\cdot)}{\partial q(t-i)} \end{bmatrix} \cdot \nabla^i.$ 5. Computes the Smith form $P \triangle Q = S$. $\mathbf{b} = \begin{bmatrix}\mathbb{I}_r & 0_{r\times(z+d-r)}\end{bmatrix} P \triangle \bar{\phi}$ 6. Defines an equations system starting from: $\Phi = [\alpha_0 \dots \alpha_n] \begin{bmatrix}\bar{v}_1 \\ \vdots \\ \bar{v}_z\end{bmatrix} - [\beta_0 \dots \beta_n] \begin{bmatrix}b_1 \\ \vdots \\ b_r\end{bmatrix},$ and computing the differentiation of Φ: $\begin{aligned} \sum_{k=0}^{s} \frac{\partial \Phi}{x_1(t-k)} \nabla^k &= 0 \\ \vdots & \quad \vdots \\ \sum_{k=0}^{s} \frac{\partial \Phi}{x_n(t-k)} \nabla^k &= 0 \\ \sum_{k=0}^{s} \frac{\partial \Phi}{u_1(t-k)} \nabla^k &= 0 \\ \vdots & \quad \vdots \\ \sum_{k=0}^{s} \frac{\partial \Phi}{u_m(t-k)} \nabla^k &= 0 \\ \sum_{k=0}^{s} \frac{\partial \Phi}{q(t-k)} \nabla^k &= 0 \end{aligned}$ 7. Solves the equations system of the step 6 for the variables α. Then, $\Omega_{k+1} = \psi\Omega_k$ where ψ is calculated following the criteria of given by (49). 8. If $rank(\Omega k) = rank(\Omega_{k+1})$ then returns the submodule $\Omega = \Omega_{k+1}$, otherwise $k = k + 1$ and goes to step 4.

Use case: presmith.

Actors: user.

Purpose: computes the presmith form of the input matrix M.

Description: The presmith form (R) is the matrix of the form:

$$PMQ = \begin{bmatrix} d_1 & * & * & 0 \\ 0 & \ddots & * & 0 \\ 0 & 0 & d_n & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Where P, Q are unimodulaires matrixes. The user writes M and the system returns the matrixes P, Q, R and the formal rank of the matrix R.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. presmith(M)	2. Finds the element with lowest polynomial degree (l.p.d.) of M. 3. Place the element of l.p.d. on the top-left corner of the matrix using the elementary matrixes P (by the left) and Q (by the right). 4. Computes the left Ore polynomials between the top-left corner element and the other ones of the first column to form two matrixes that multiplying M by the left and by the right will make the first column elements zero, except the diagonal one. Multiplies these matrixes by P and Q and stores the result in the same variables. 5. Replaces the first row and the first column of the matrix by a zero row and a zero column respectively. 6. Place the element of l.p.d. on the top-left corner of the matrix using elementary matrixes and multiplies them by P and Q saving the result in the same variables. Repeats the steps 3-5 until the matrix has only one non zero element. 7. Computes the presmith form $R = P \cdot M \cdot Q$. 8. The formal rank ($formalrank$) is the number of no-null elements in the diagonal of the matrix R. 9. Returns the list $\begin{bmatrix} P & R & Q & formalrank \end{bmatrix}$.

Use case: reuclid.

Actors: user.

Purpose: computes the Euclidien division by the right between $a[\nabla], b[\nabla] \in \mathcal{K}[\nabla]$.

Description: finds the polynomials $c[\nabla]$ and $r[\nabla] \in \mathcal{K}[\nabla]$, where $a[\nabla] = b[\nabla]c[\nabla] + r[\nabla]$, with $b[\nabla] \neq 0$ and the polynomial degree of $r[\nabla]$ smaller than the polynomial degree of $b[\nabla]$.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. reuclid($a[\nabla], b[\nabla]$)	2. Right-divides $a[\nabla]$ by $b[\nabla]$. 3. Returns the quotient $c[\nabla]$ and the reminder $r[\nabla]$.

Use case: save system.

Actors: user.

Purpose: save the state space equations of a control system storing the variables SystemName_X and SystemName_var_Y in the *FileName.sd* file, where X=F, G, H, Fg, Hg, dF, dG, dH and Y=sta, con, out.

Description: the user defines an equations system with the *systdef* instruction and the name of the file where the data will be stored.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. savesyst(SystemName,FileName)	2. Store the variables SystemName_X and SystemName_var_Y in the file named FileName.sd.

Use case: smith.

Actors: user.

Purpose: computes the smith form of the input matrix M.

Description: The presmith form (S) is the matrix of the form $diag\{d_1 1, \ldots, d_n n\}$. Where P, Q are unimodulaires matrixes. The user writes M and the system returns the matrixes P, S, Q.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. smith(M)	2. Finds the element with lowest polynomial degree (l.p.d.) of M.
	3. Place the element of l.p.d. on the top-left corner of the matrix using the elementary matrix P (by the left) and Q (by the right).
	4. Computes the left Ore polynomials between the top-left corner element and the other ones of the first column to form two matrixes that multiplying M by the left and by the right will make the first column elements zero, except the diagonal one. Multiplies these matrixes by P and Q and stores the result in the same variables.
	5. Computes the right Ore polynomials between the top-left corner element and the other ones of the first row to form two matrix that multiplying M by the left and by the right will make the first row elements zero, except the diagonal one. Multiplies these matrixes by P and Q and stores the result in the same variables.
	6. Deletes the first row and the first column of the matrix and add a last zero row and a last zero column to keep the same matrix dimension.
	7. Place the element of l.p.d. on the top-left corner of the matrix using elementary matrixes and multiplies them by P and Q saving the result in the same variables. Repeats the steps 3-6 until the matrix has only one non zero element.
	8. Computes the smith form $S = P \cdot M \cdot Q$.
	9. Returns the list $\begin{bmatrix} P & S & Q \end{bmatrix}$.

Use case: submat.

Actors: user.

Purpose: Extracts a submatrix.

Description: the user writes the instruction specifying the columns and rows to extract and the original matrix (*mat*). The system returns the given submatrix.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. submat(r_i, r_f, mat, c_i, c_f)	2. Returns the submatrix defined from the row r_i to the row r_f and from the column c_i to the column c_f.

Use case: swapcol.

Actors: user.

Purpose: Interchange two columns in a matrix.

Description: the user writes the identification number of the columns to interchange (c_1, c_2) and the matrix name (*mat*). The system returns the matrix with the columns interchanged.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. swapcol(c_1, c_2, mat)	2. Use lisp instructions to interchange the columns and returns the new matrix.

Use case: swaprow.

Actors: user.

Purpose: Interchange two rows in a matrix.

Description: the user writes the identification number of the rows to interchange (r_1, r_2) and the matrix name (*mat*). The system returns the matrix with the rows interchanged.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. swaprow(r_1, r_2, mat)	2. Use lisp instructions to interchange the rows and returns the new matrix.

Use case: triangular equivalence.

Actors: user.

Purpose: determines if a system is equivalent to a triangular form.

Description: the user writes the name assigned to the equations system. The system returns *true* if the system is equivalent to a triangular form or *false* otherwise. The control variable of the system *sys* must have dimension $\mathbb{R}^{1\times 1}$.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. istriangular(sys)	2. Returns $false$ if the dimension of the control variable is greater than one. Otherwise goes to the step 3. 3. Computes the $\mathcal{H}_k$ submodules. 4. Returns $false$ if $\mathcal{H}_\infty \neq 0$. Otherwise goes to step 5. 5. Computes if $\mathcal{H}_n = \text{span}_{\mathcal{K}[\nabla]}\{\omega_1\}$ is integrable using a particular case of the Frobenius theorem: $$\Omega := d\omega_1 \wedge \omega_1$$ Returns $false$ if $\Omega \neq 0$, otherwise computes $l = n-1, \alpha = 2$ and goes to step 6. 6. Completes a basis for $\mathcal{H}_l := span_{\mathcal{K}[\nabla]}\{\omega_1, \ldots, \omega_\alpha\}$ using the use case *bascom*. The new basis is of the form: $$\mathcal{H}_l := \text{span}_{\mathcal{K}[\nabla]}\{\omega_1,\ldots,\omega_{(\alpha-1)},\tilde{\omega}_1,\ldots,\tilde{\omega}_{(\alpha-(\alpha-1))}\}$$ 7. Verify if $\mathcal{H}_l$ is integrable computing: $$\Omega := d\tilde{\omega}\wedge\tilde{\omega}\wedge\omega\wedge\cdots\wedge\nabla^k\omega, \quad k\in\mathbb{N},\ \omega := \omega_1\wedge\cdots\wedge\omega_{(\alpha-1)}$$ Returns $false$ if $\Omega \neq 0$, which means that is not integrable, otherwise computes $l = l-1, \alpha = \alpha+1$ and goes to 8. If $l > 1$ goes to 6, otherwise returns $true$.

Use case: tshift.

Actors: user.

Purpose: delays f n time units.

Description: the user writes f, which could be a polynomial or a matrix, and the delay to apply it (n). The default value is one when n isn't specified.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. tshift(f,n) or tshift(f)	2. The system substitutes t by $t-n$ in f. 3. Returns f shifted n time units.

Use case: vars.

Actors: user.

Purpose: creates a list with the variables $varsta$ and $varcon$ that are in the function f.

Description: the user writes f, which could be polynomial or matrix, the variables to search. The system returns a list with the variables founded.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. vars$(f, varsta, varcon)$	2. Review term by term of f and create a list with the $varsta_i$ and $varcon_j$ variables, with $i, j \in I\!N$. 3. Returns the list founded in the step 2.

Use case: vecreate.

Actors: user.

Purpose: creates vectors of different forms according to the input arguments.

Description: the user writes the input arguments:

$$args = \begin{bmatrix} var & n & tm \end{bmatrix},$$

where var is the variable basis of the vector; n the vector dimension and tm the parameter that defines the kind of vector to create.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. vecreate$(args)$, with $args = \begin{bmatrix} var & n & tm \end{bmatrix}$.	2. Returns the vector according to the tm variable: tm — Kind of vector "n" — $\begin{bmatrix} var_1, \ldots, var_n \end{bmatrix}^T$ "t" — $\begin{bmatrix} var_1(t), \ldots, var_n(t) \end{bmatrix}^T$ $\mathbb{Z}_+$ — $\begin{bmatrix} del(diff(var_1(t), t, tm)), \ldots, del(diff(var_n(t), t, tm)) \end{bmatrix}^T$ If the optional third argument tm is missing it returns $\begin{bmatrix} del(var_1(t)), \ldots, del(var_n(t)) \end{bmatrix}^T$.

Use case: wedgeproduct.

Actors: user.

Purpose: computes the wedge product of a p-form a and a q-form b.

Description: the program depends on the Cartan package which computes the wedge product without delays. The basis variables to derivate are stored in the variables *_wvarsta* and *_wvarcon* and have as default values x and u, respectively. Both variable values could be changed. The user writes a, the operator "~ $\wedge$" and b. The system returns a (p+q)-form.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. a~ $\wedge b$	2. Makes the $list1$ with all the basis variables contained in a and b. Remember that $x_1(t)$ is independent of $x_1(t-1)$. 3. Creates the $list2$ relating each variable of the $list1$ with a single letter and substitutes them in a and b. 4. Loads the Cartan package. 5. Computes the wedge product between a and b (a~ b). 6. Returns the result obtained in 5 substituting the variables of the $list2$ for its equivalence in the $list1$.

Use case: xiy

Actors: user.

Purpose: computes $\mathcal{X} \cap \mathcal{Y}$ to solve the disturbance rejection problem of a system described by (35).

Description: the user writes the name assigned to the equations system (sys). The system returns $\mathcal{X} \cap \mathcal{Y}$, which is defined: $\mathcal{X} \cap \mathcal{Y} := span_{\mathcal{K}[\nabla]}\{d\mathbf{x}\} \cap \text{span}_{\mathcal{K}[\nabla]}\{d\mathbf{y}, d\dot{\mathbf{y}}, \dots\}$.

ACTION OF THE ACTOR	SYSTEM'S ANSWER
1. xiy(sys)	2. Computes the differential form of the system output y from 0 thru n. $$\begin{aligned} y &= h(x_\tau) \\ dy &= \overline{h}dx \\ dy^{(1)} &= f_1 dx + g_1 du \\ dy^{(2)} &= f_2 dx + g_2 du + g du^{(1)} \\ &\vdots \\ dy^{(n)} &= f_n dx + g_n du + \cdots + g du^{(n-1)} \end{aligned}$$ 3. Computes the follow equation to find $\mathcal{X} \cap \mathcal{Y}$: $$\alpha\left[\begin{bmatrix} \overline{h} \\ f_1 \\ \vdots \\ f_n \end{bmatrix}\mathbf{dx} + \begin{bmatrix} 0 \\ g_1 \\ \vdots \\ g_n \end{bmatrix}\mathbf{du} + \cdots + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ g_1 \end{bmatrix}\mathbf{du^{(n-1)}}\right] = (\cdot)\mathbf{dx}$$ where $$\alpha = \begin{bmatrix} 0 & | & & | & 0 \\ g_1 & | & \cdots & | & 0 \\ \vdots & & & & \vdots \\ g_n & | & & | & g_1 \end{bmatrix}^{\perp}.$$ 4. Finally returns $\mathcal{X} \cap \mathcal{Y}$: $\mathcal{X} \cap \mathcal{Y} = \alpha f$, with $f = \begin{bmatrix} \overline{h} & f_1 & \dots & f_n \end{bmatrix}^T$.

A.2 Diagramas de flujo de datos

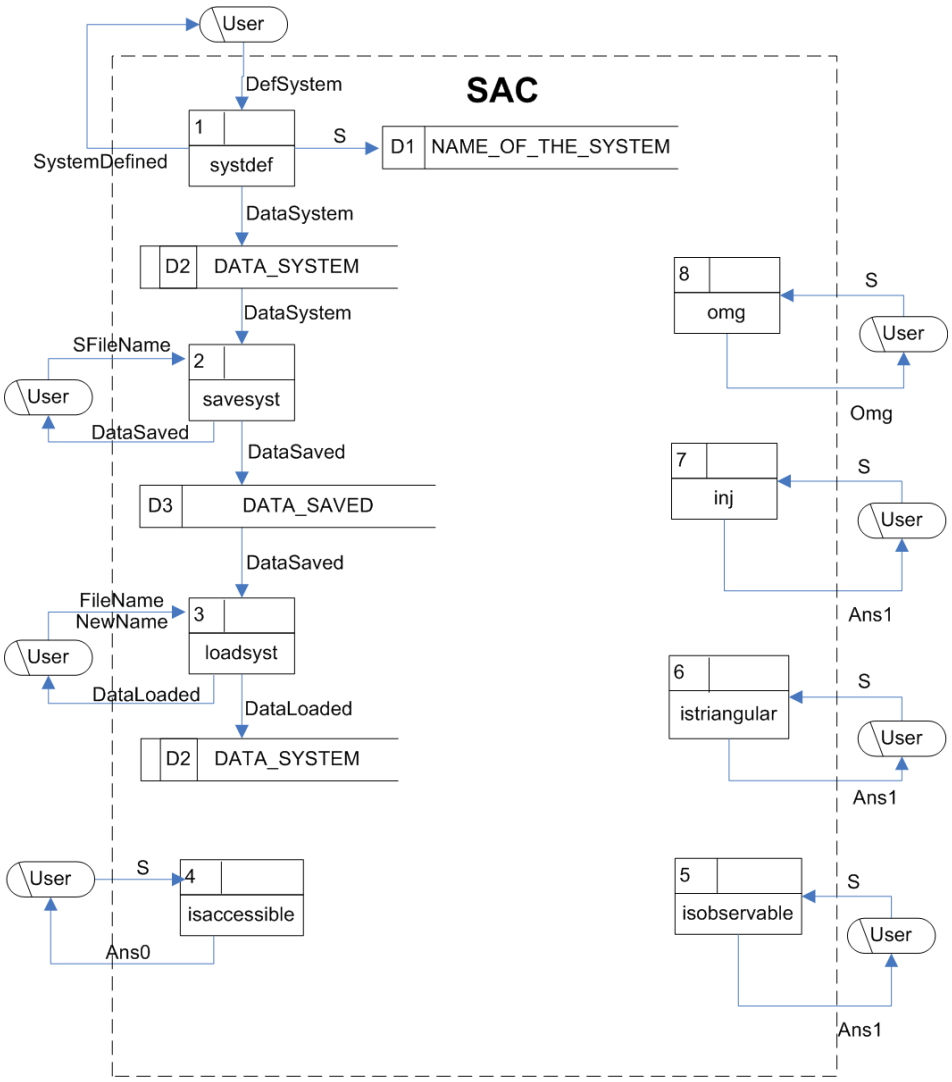


Figura 15: Sistema completo, nivel 1.

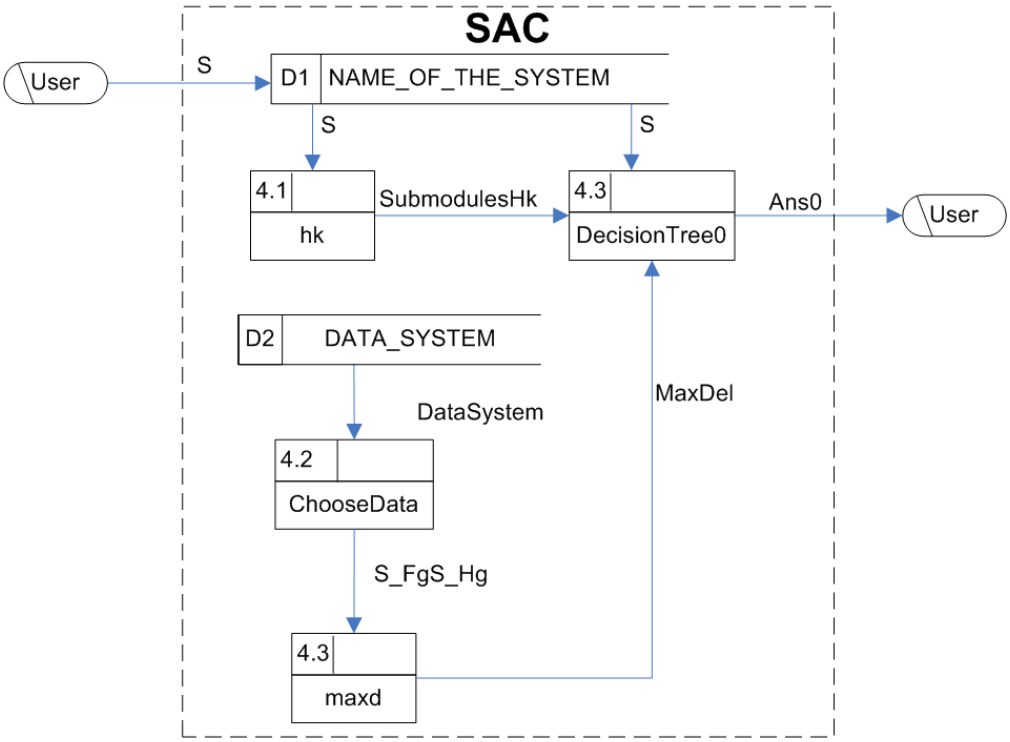


Figura 16: Diagrama de flujo de datos de la rutina que calcula si un sistema de control es accesible. Referencia: caso de uso *accessibility*.

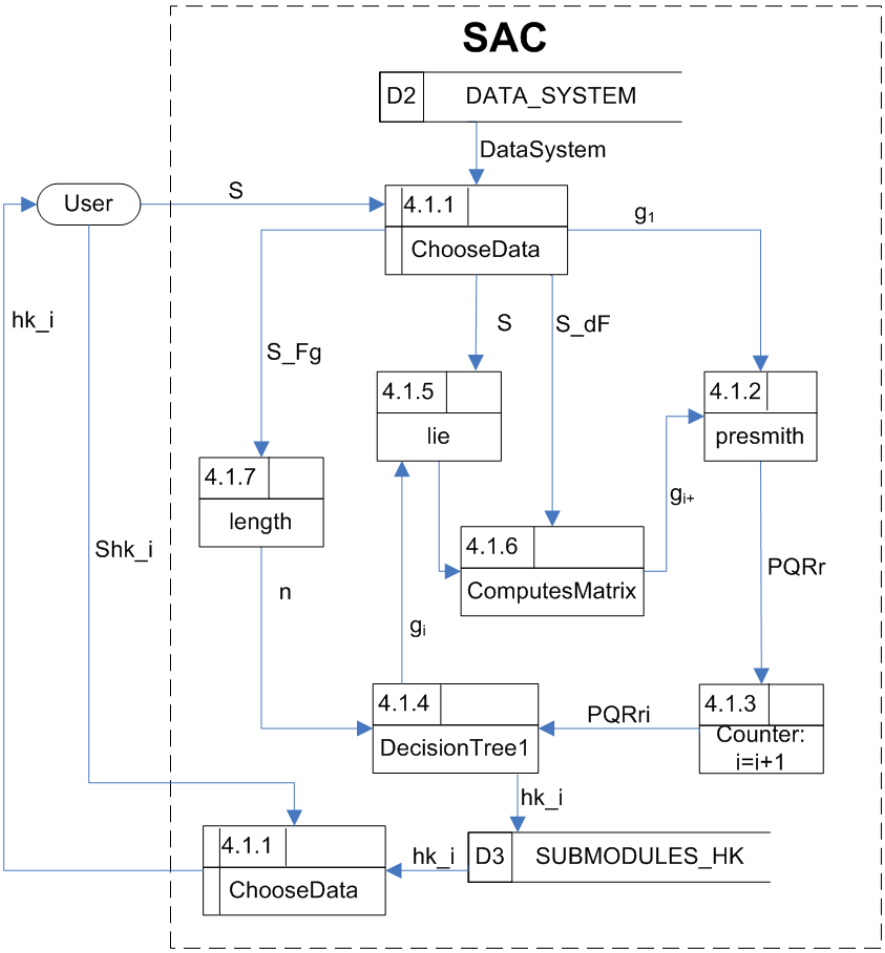


Figura 17: Diagrama de flujo de datos de la rutina que calcula los submódulos $\mathcal{H}_{\mathcal{K}}$. Referencia: caso de uso *hk*.

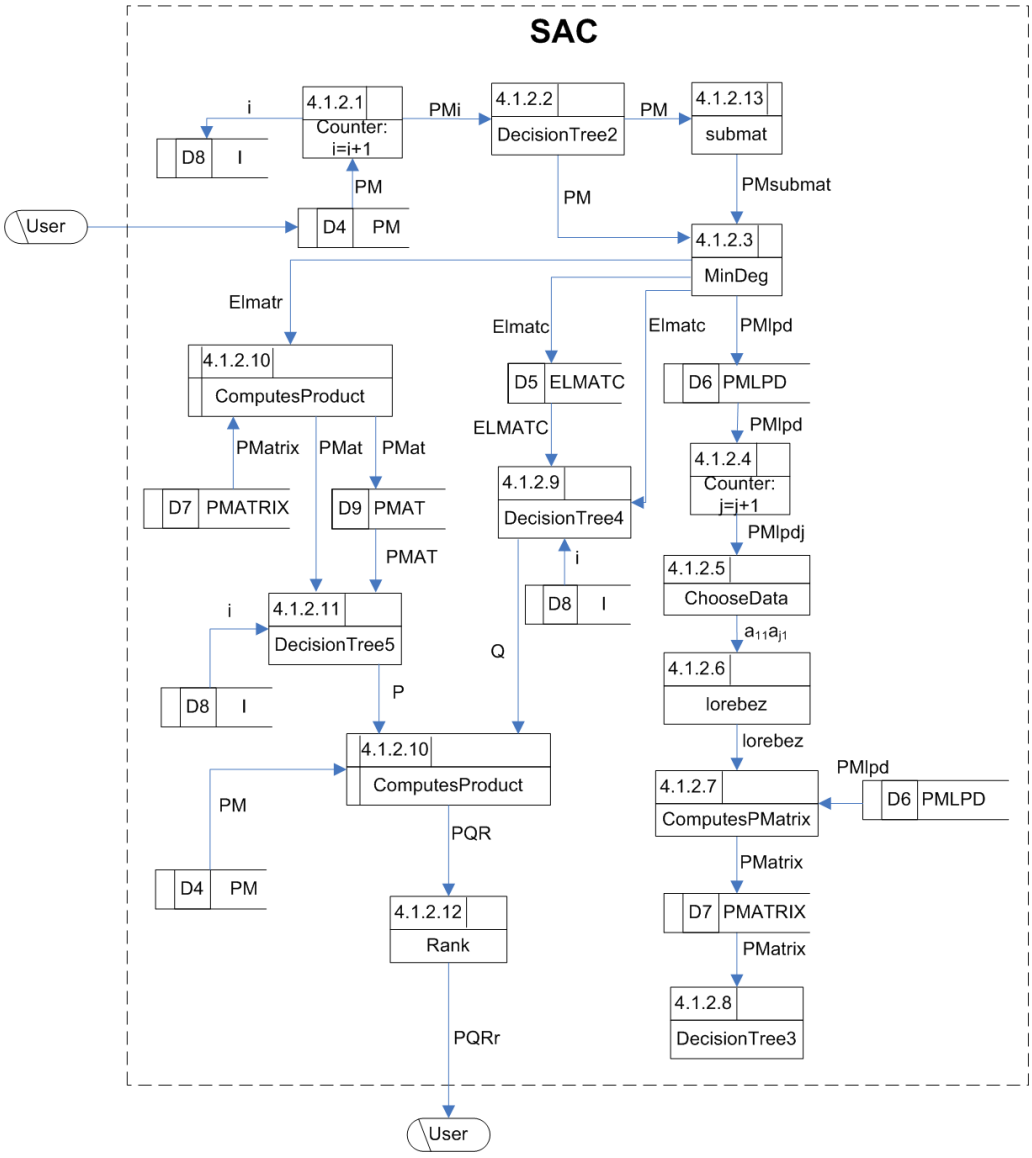


Figura 18: Diagrama de flujo de datos de la rutina que calcula la preforma de Smith. Referencia: caso de uso *presmith*.

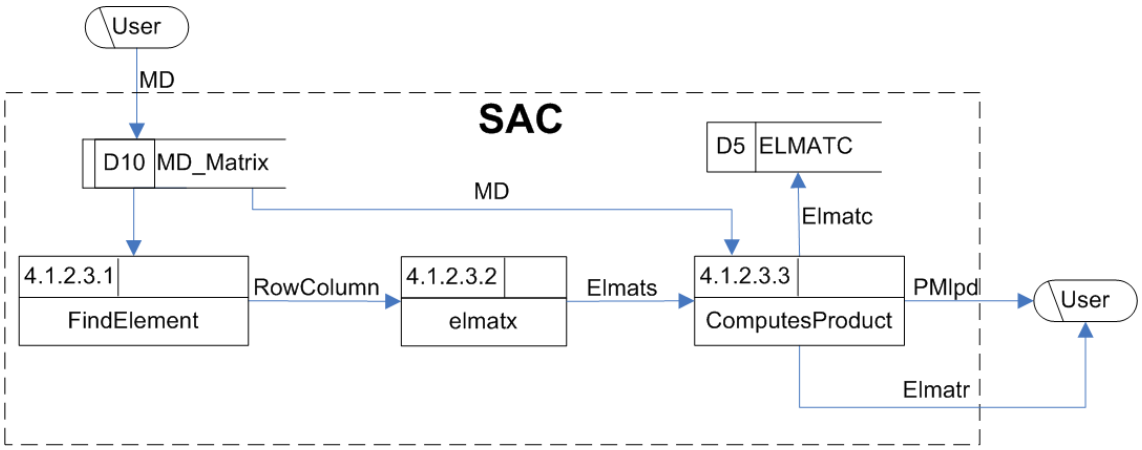


Figura 19: Diagrama de flujo de datos de la rutina que encuentra el elemento de menor grado polinomial en una matriz y lo ubica en el elemento de la esquina superior izquierda. Referencia: caso de uso *presmith*.

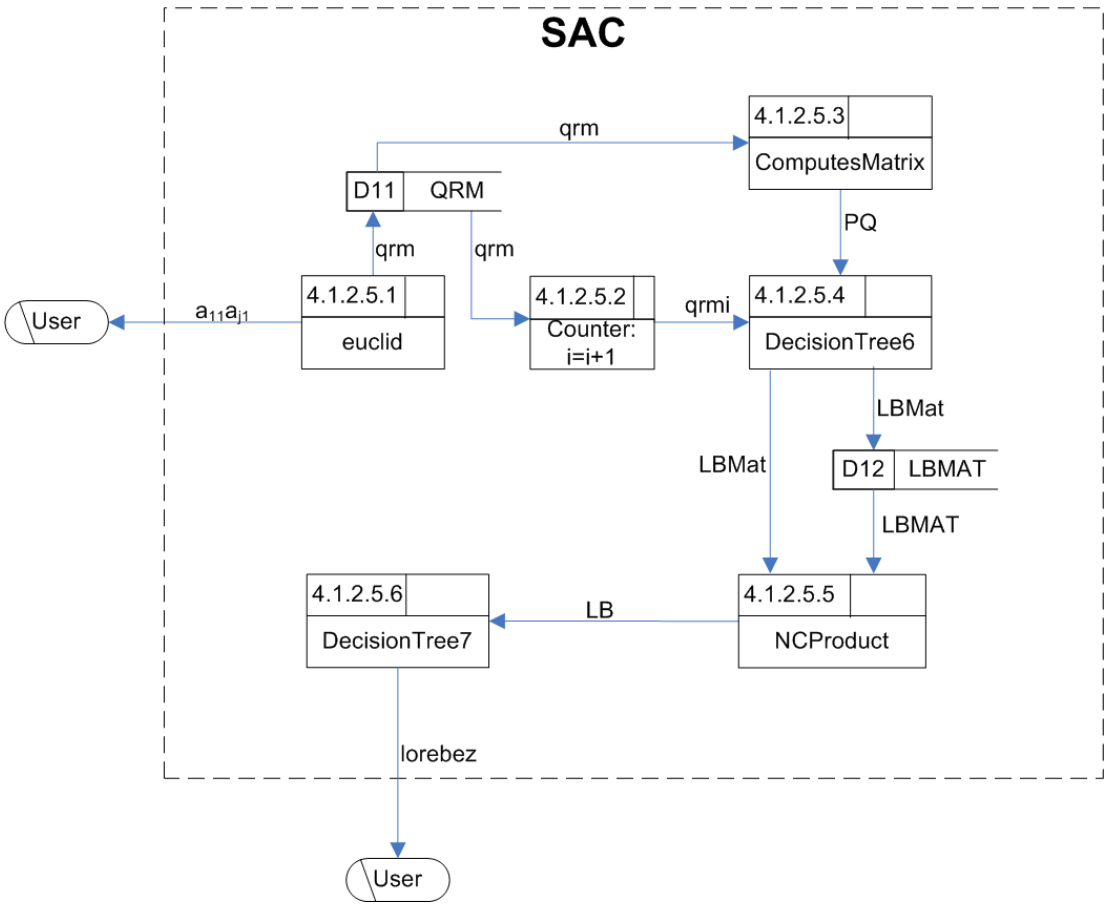


Figura 20: Diagrama de flujo de datos de la rutina que calcula la matriz formada por los polinomios izquierdos de ore y bezout. Referencia: caso de uso *lorebez*.

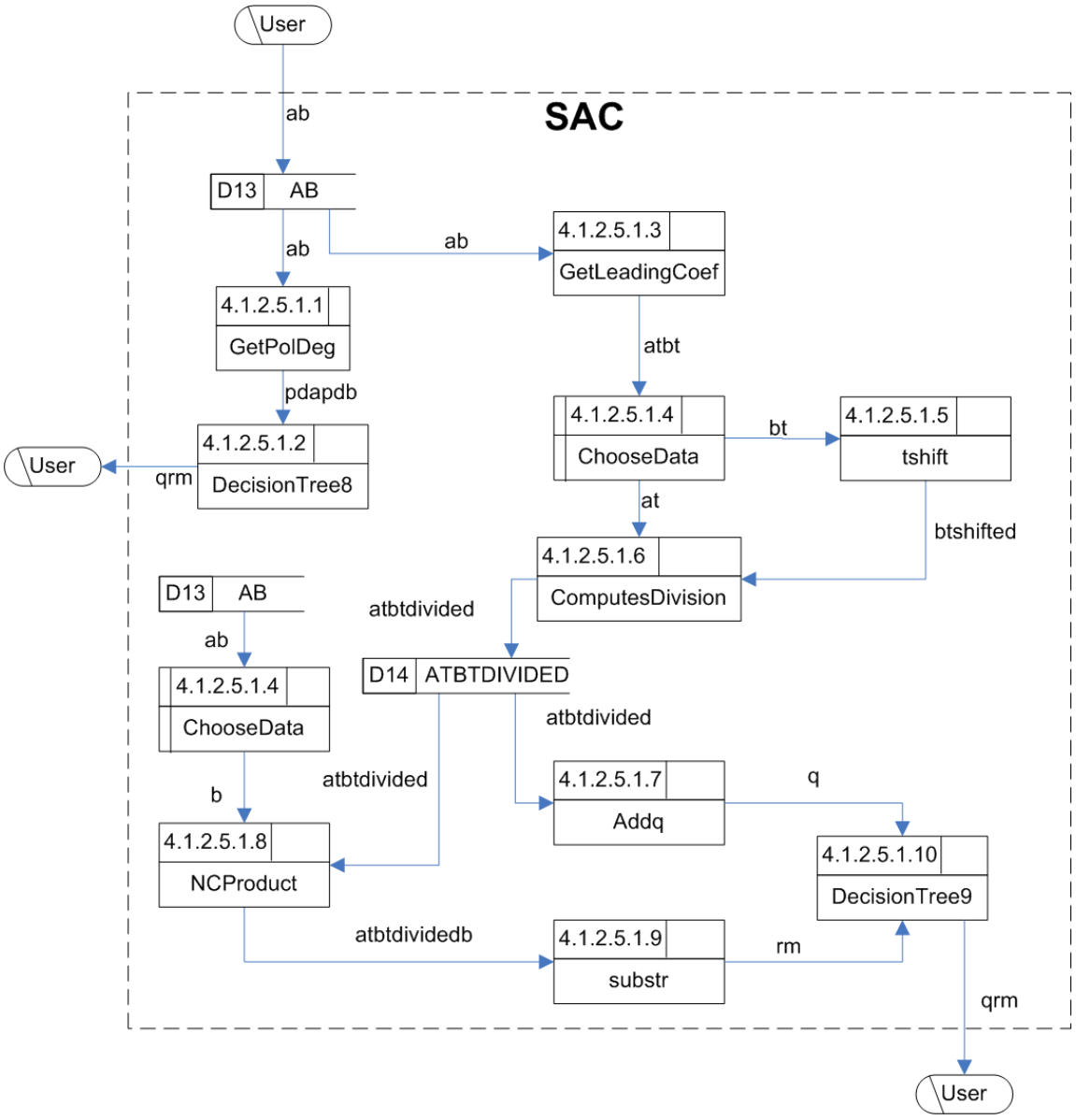


Figura 21: Diagrama de flujo de datos de la rutina que calcula la división Euclidiana. Referencia: caso de uso *euclid*.

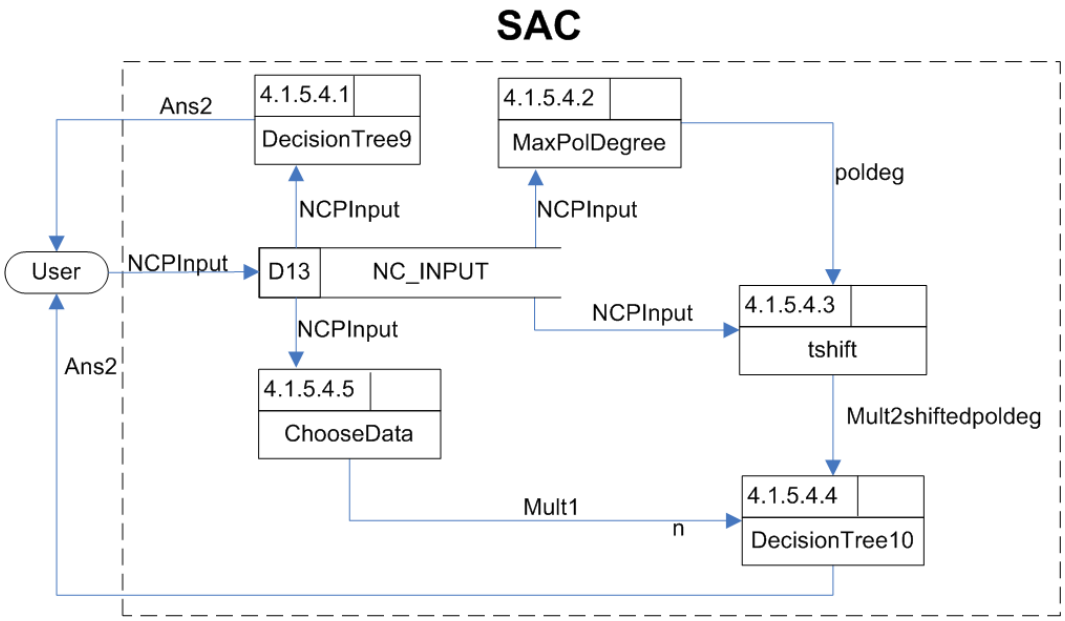


Figura 22: Diagrama de flujo de datos de la rutina que calcula el producto no conmutativo. Referencia: caso de uso *NCProduct*.

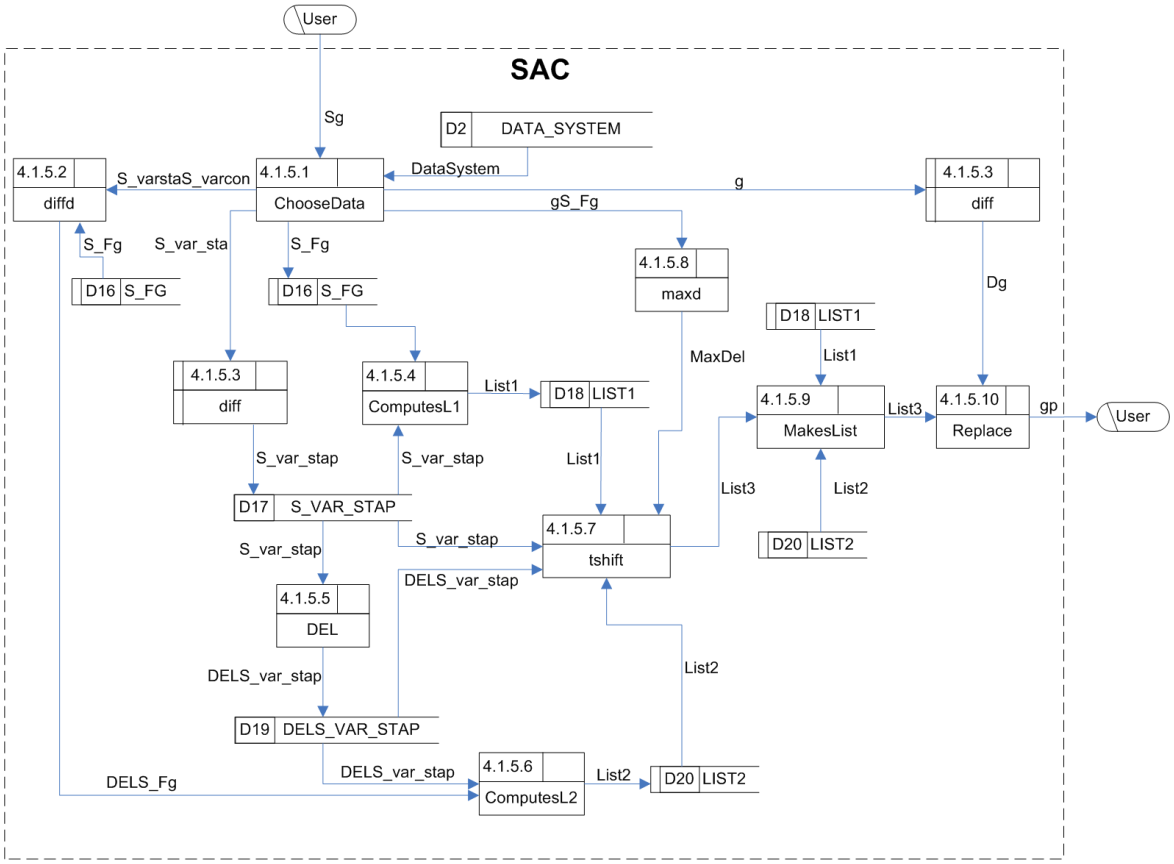


Figura 23: Diagrama de flujo de datos de la rutina que calcula la derivada de Lie. Referencia: caso de uso *lie*.

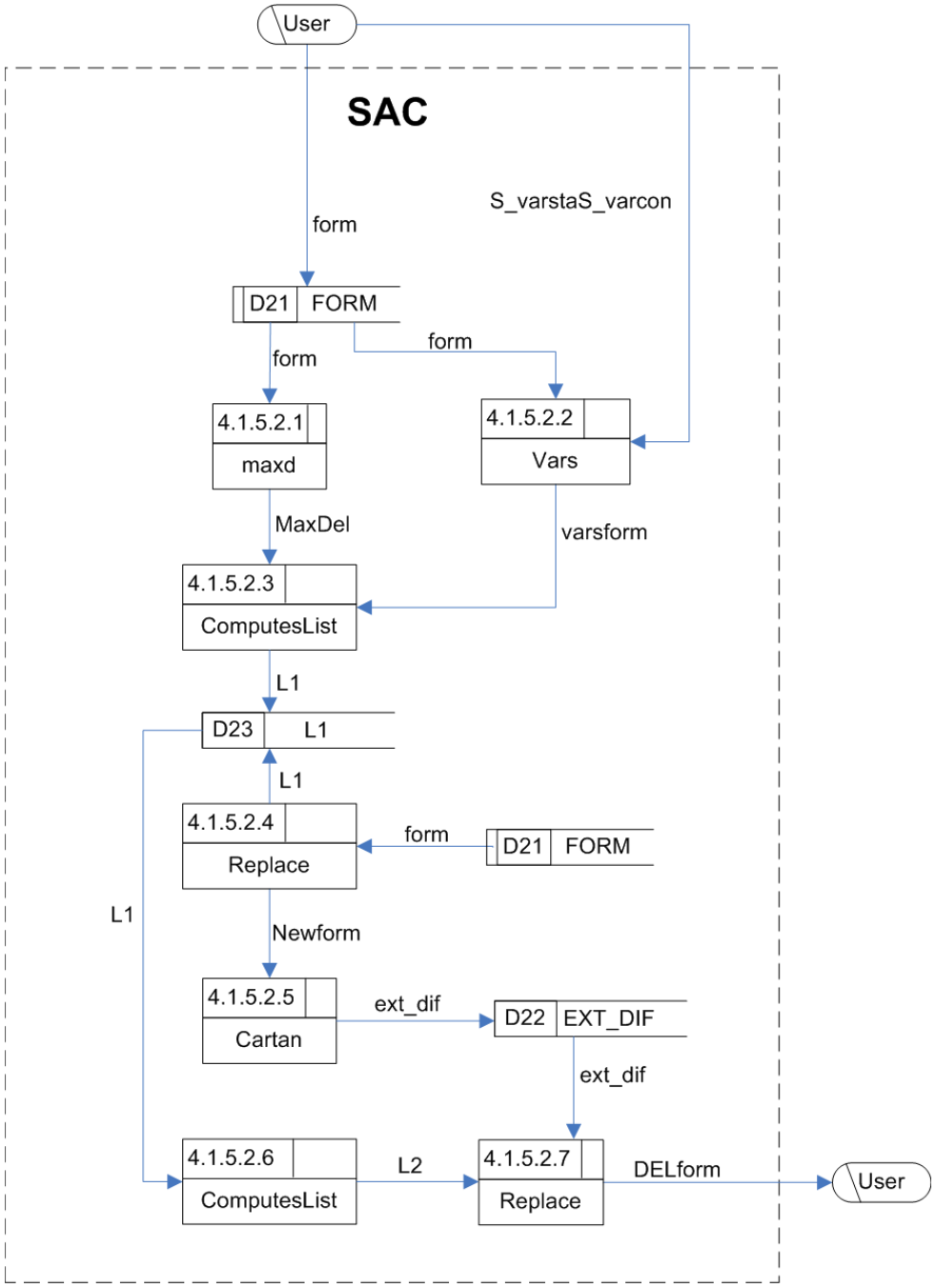


Figura 24: Diagrama de flujo de datos de la rutina que calcula la forma diferencial de un polinomio. Referencia: caso de uso *diffd*.

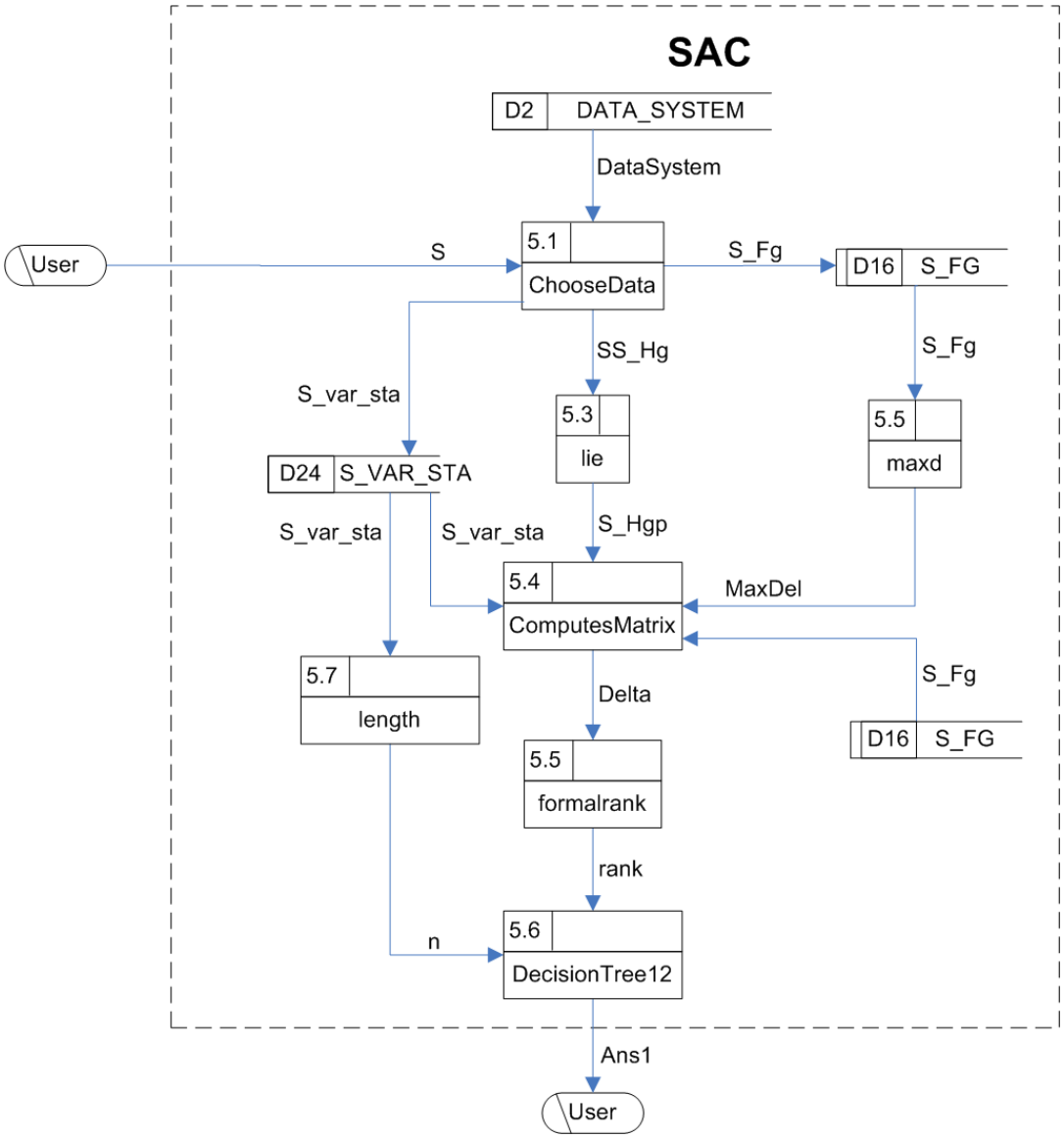


Figura 25: Diagrama de flujo de datos de la rutina que calcula si un sistema de control es observable. Referencia: caso de uso *observability*.

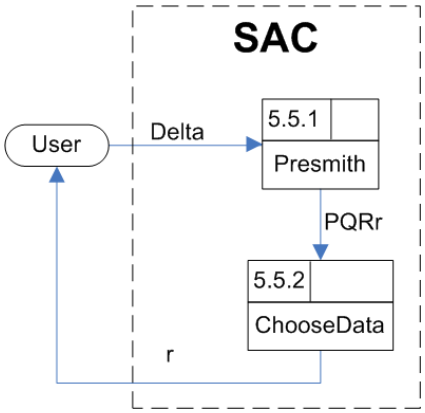


Figura 26: Diagrama de flujo de datos de la rutina que calcula el rango genérico de una matriz. Referencia: caso de uso *formalrank*.

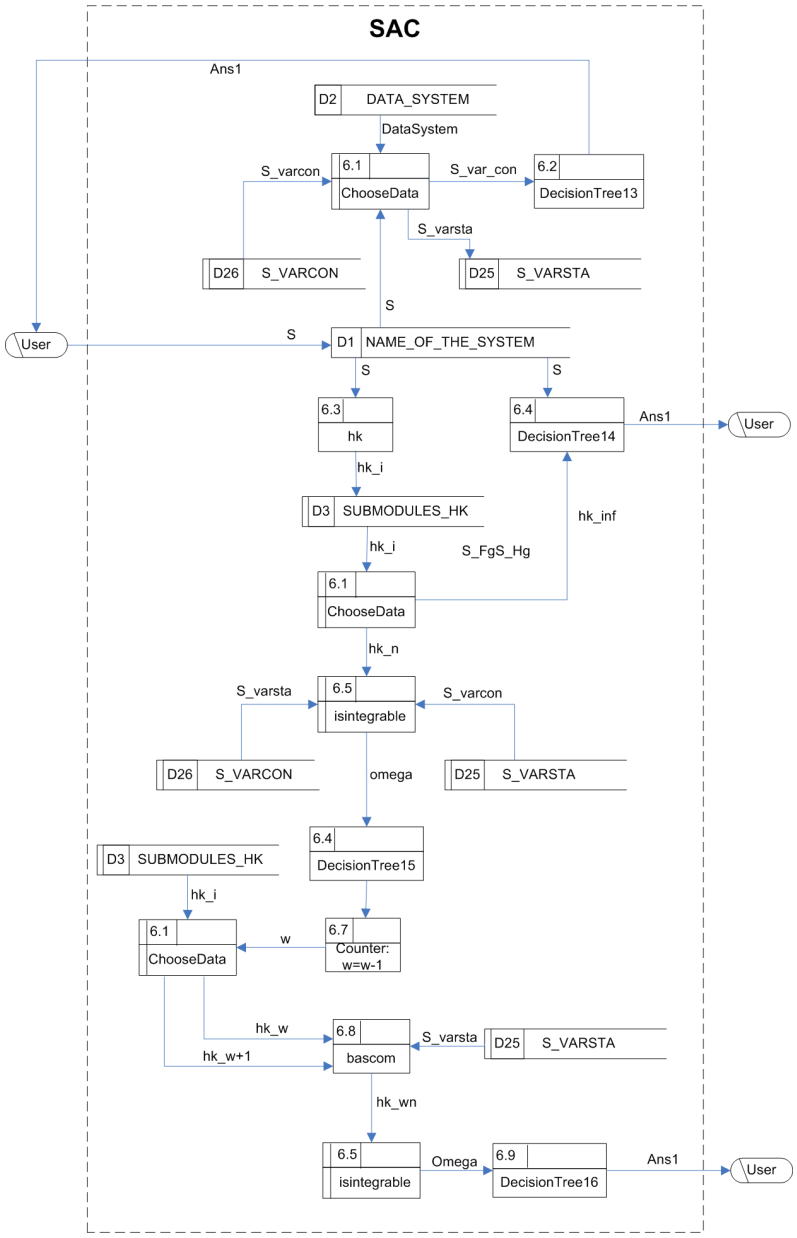


Figura 27: Diagrama de flujo de datos de la rutina que calcula si un sistema es equivalente a la forma triangular. Referencia: caso de uso *triangular equivalence*.

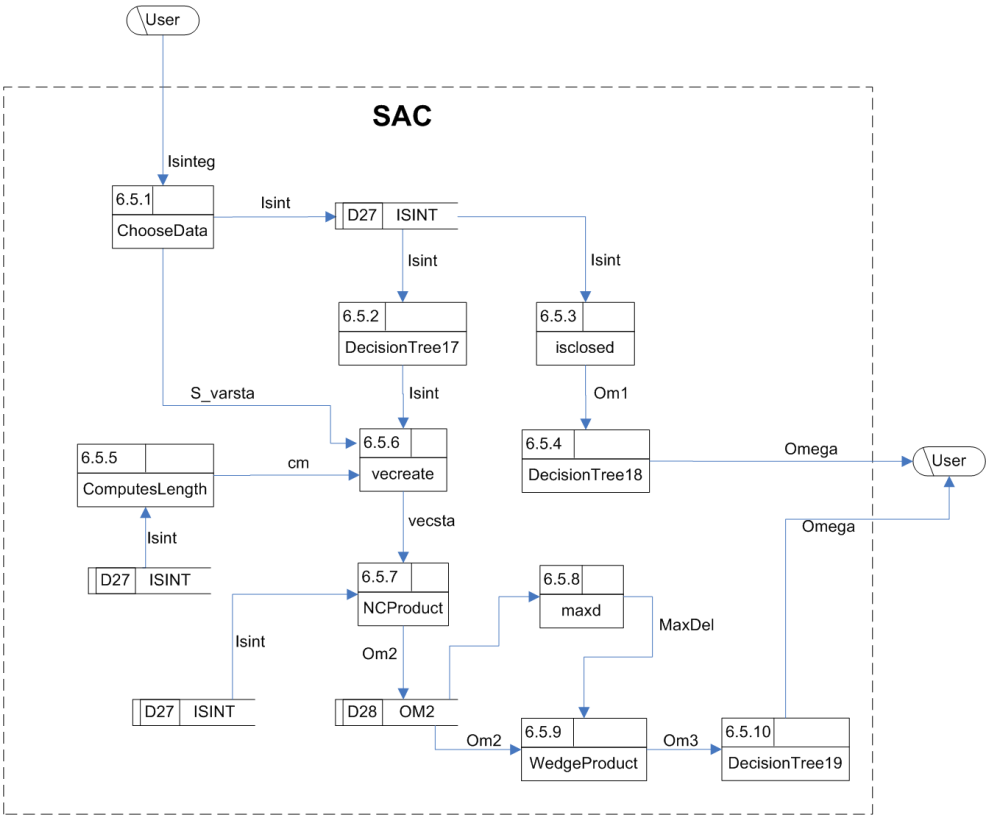


Figura 28: Diagrama de flujo de datos de la rutina que calcula si una 1-forma o submódulo es integrable. Referencia: caso de uso *isintegrable*.

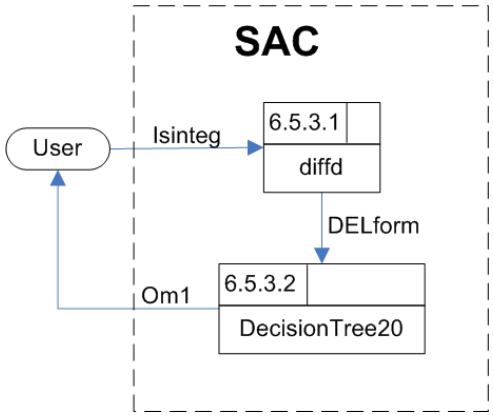


Figura 29: Diagrama de flujo de datos de la rutina que calcula si una 1-forma es cerrada. Referencia: caso de uso *isclosed*.

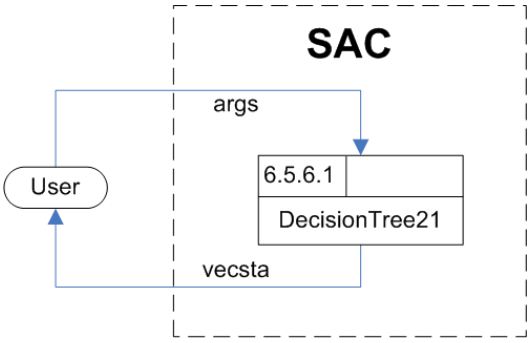


Figura 30: Diagrama de flujo de datos de la rutina para crear vectores. Referencia: caso de uso *vecreate*.

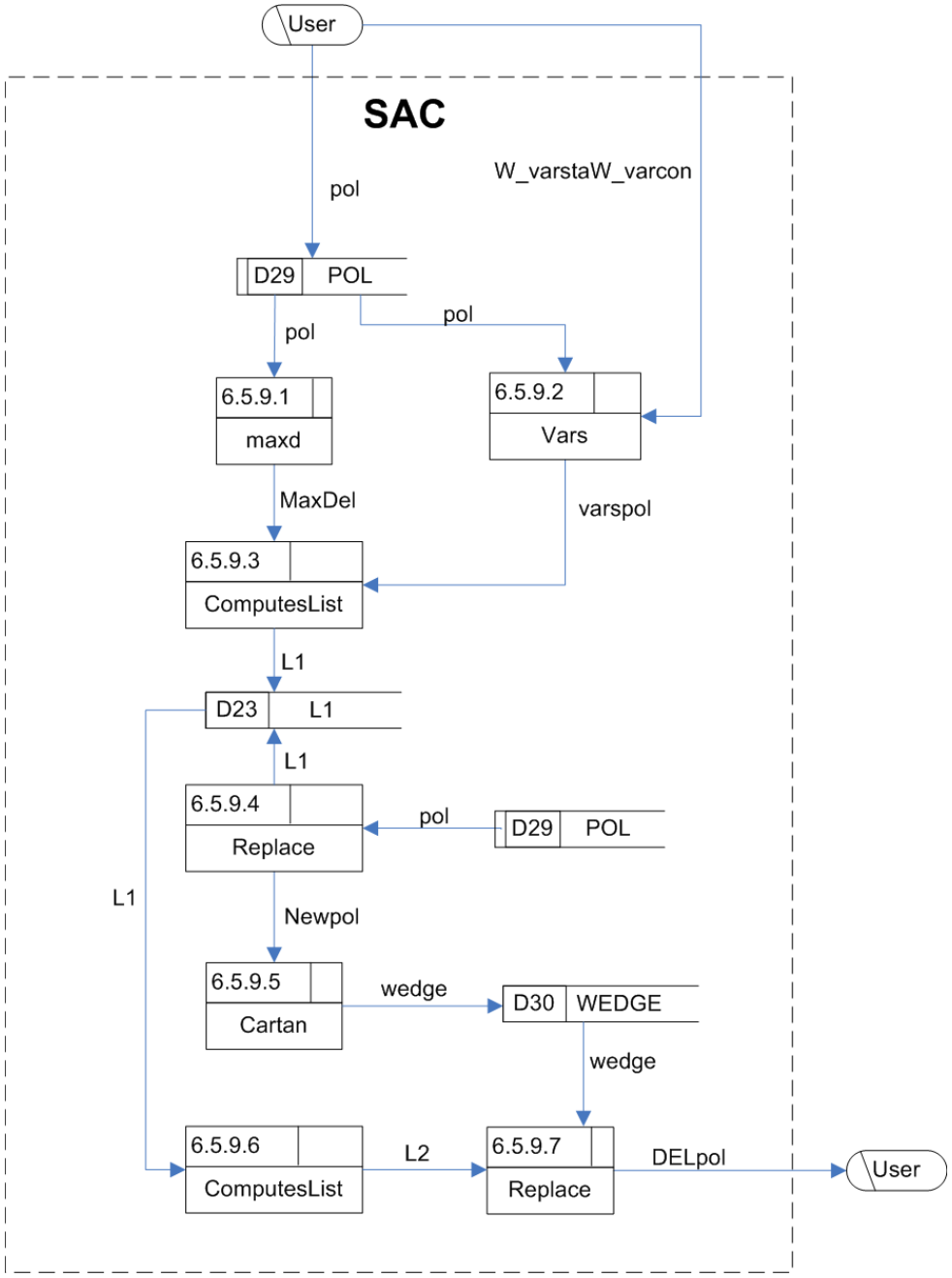


Figura 31: Diagrama de flujo de datos de la rutina que calcula el producto exterior. Referencia: caso de uso *wedgeproduct*.

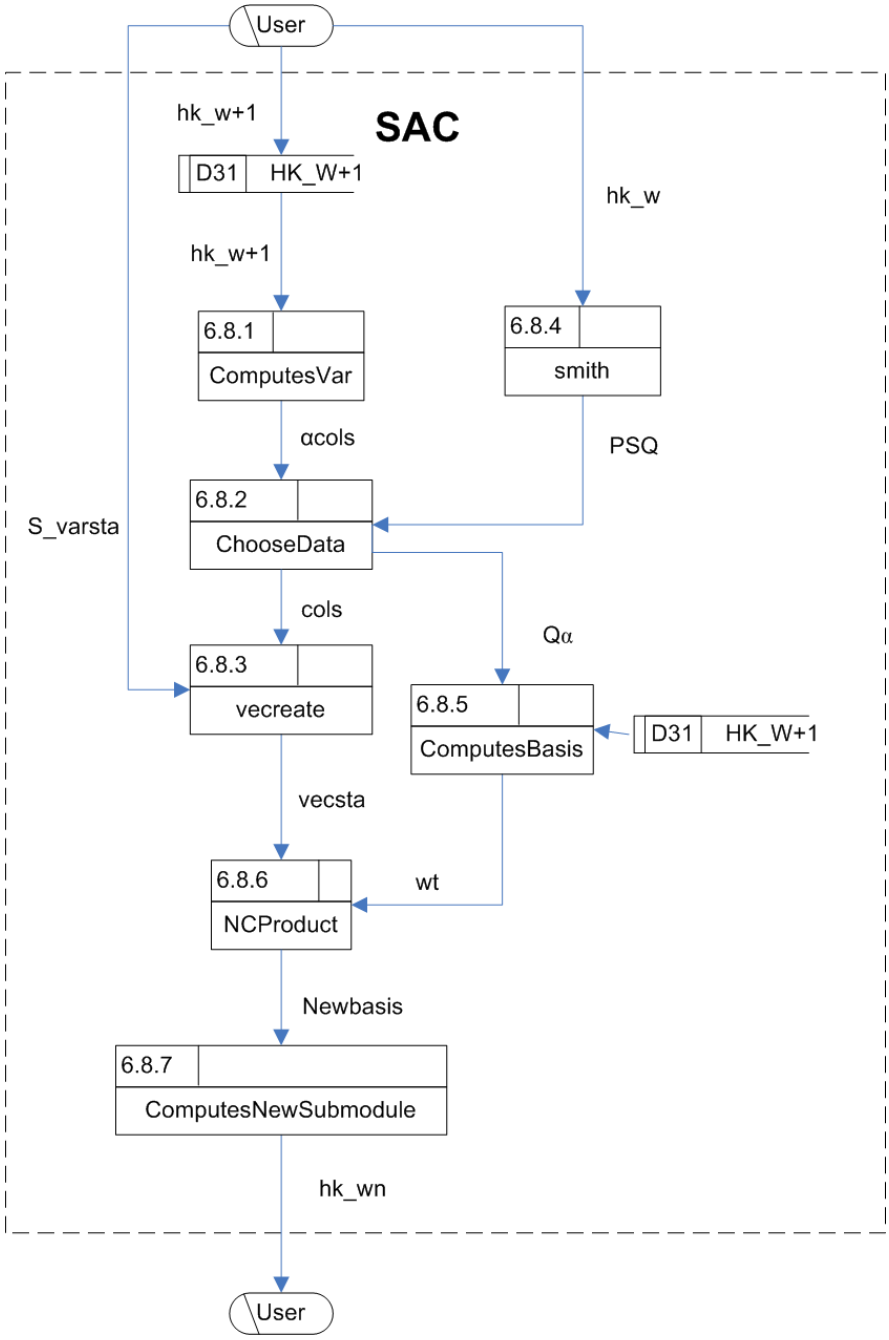


Figura 32: Diagrama de flujo de datos de la rutina que completa una base para un submódulo. Referencia: caso de uso *bascom*.

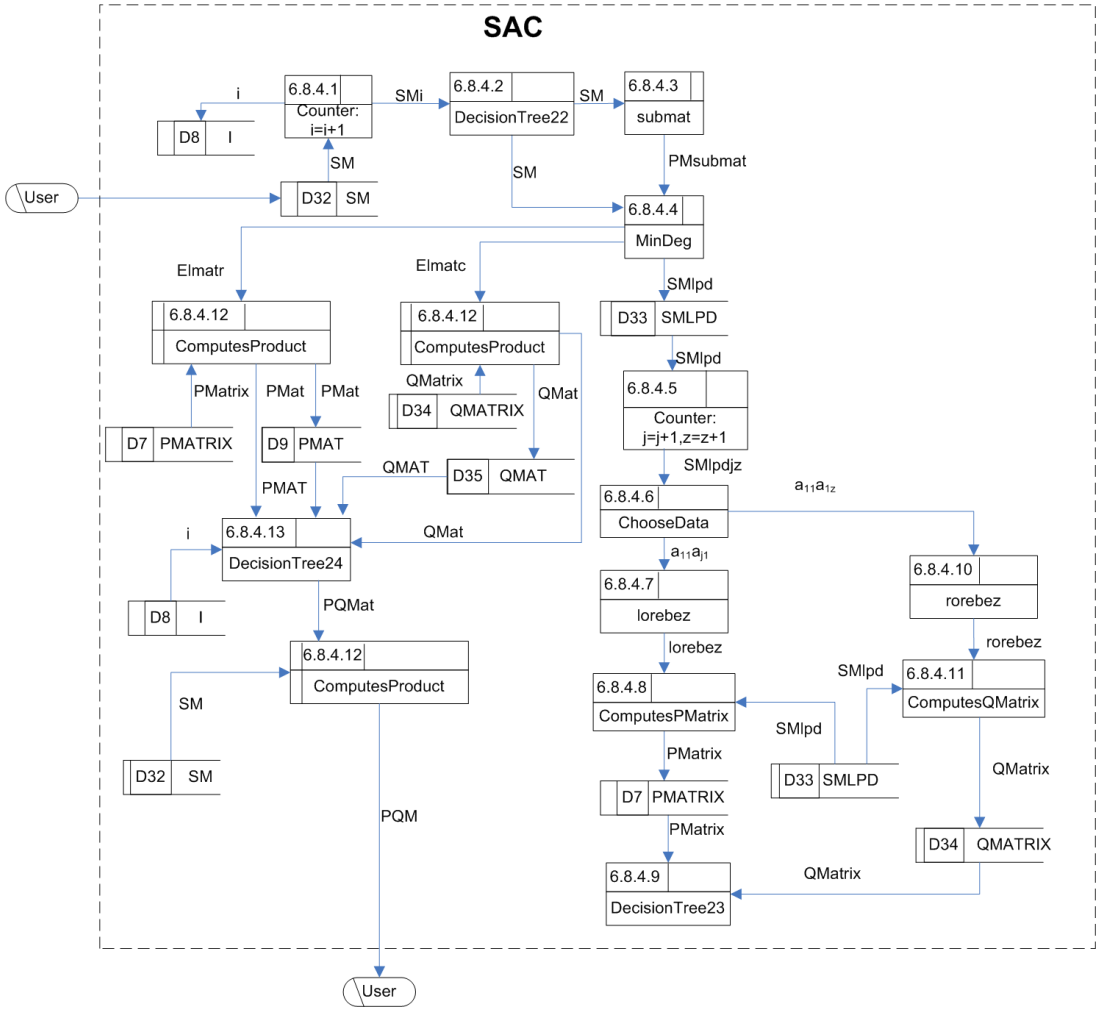


Figura 33: Diagrama de flujo de datos de la rutina que calcula la forma de Smith de una matriz. Referencia: caso de uso *smith*.

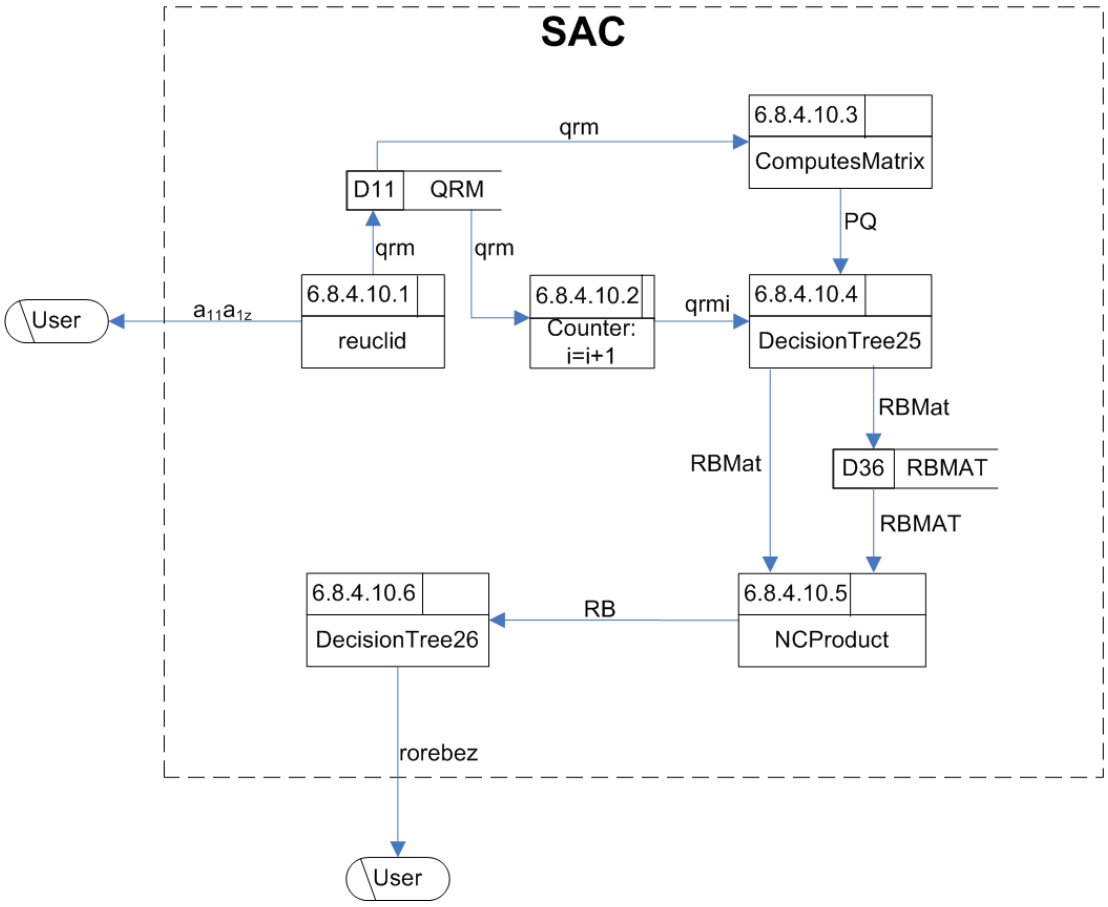


Figura 34: Diagrama de flujo de la rutina que calcula una matriz formada por los polinomios de Ore y de Bezout por la derecha. Referencia: caso de uso *rorebez*.

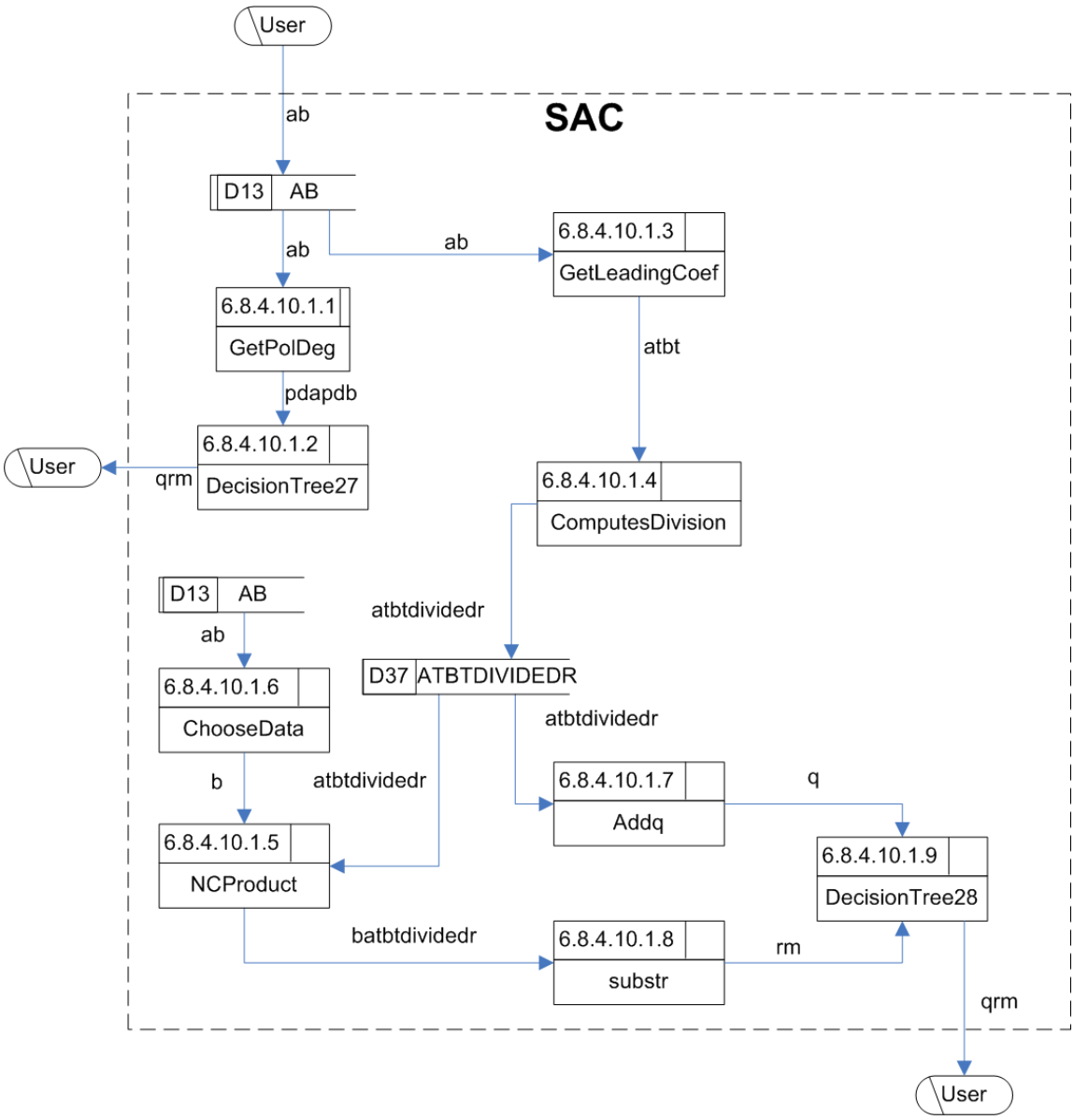


Figura 35: Diagrama de flujo de datos de la rutina que calcula la división Euclidiana por la derecha. Referencia: caso de uso *reuclid*.

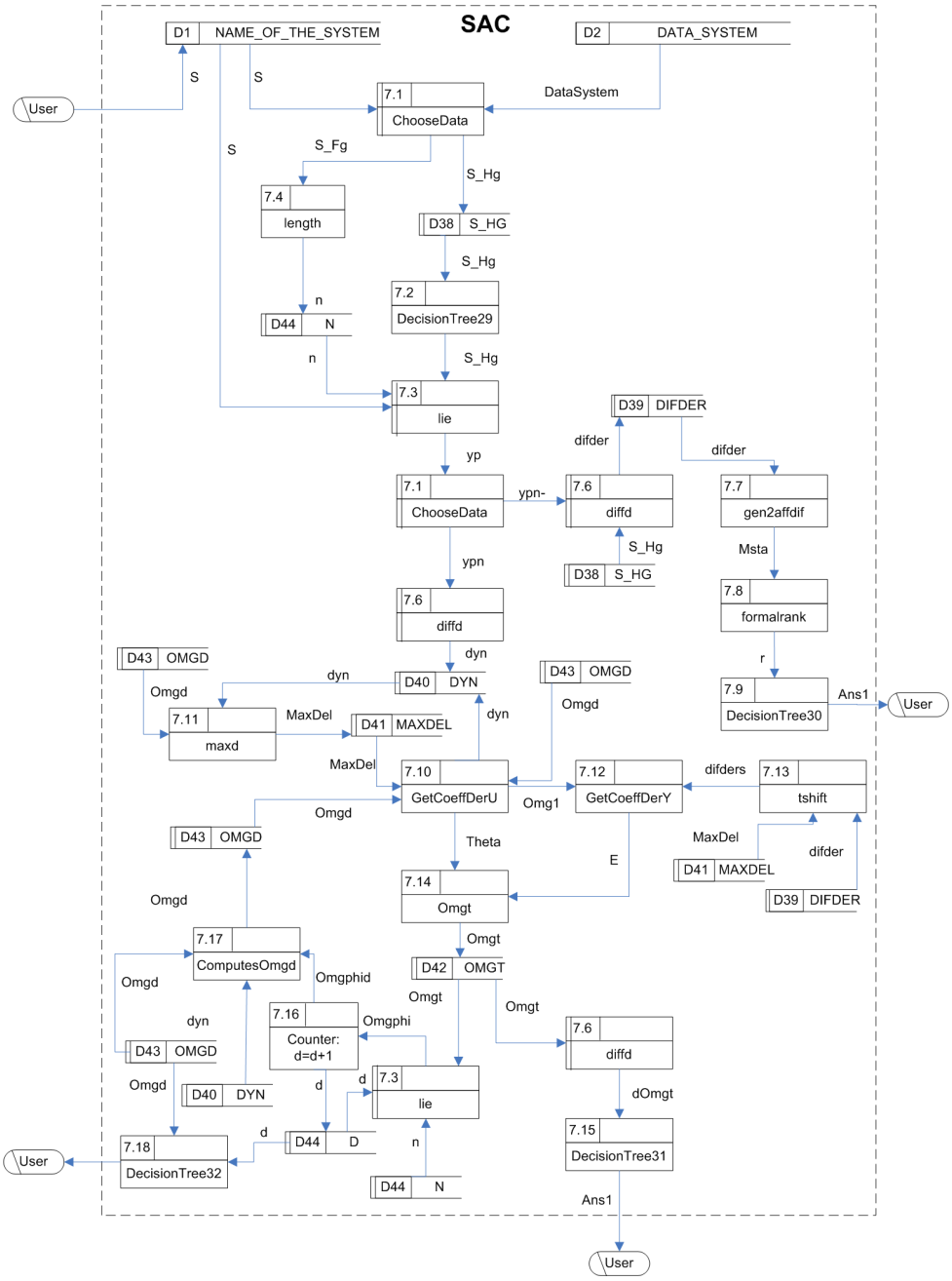


Figura 36: Diagrama de flujo de datos de la rutina que calcula si un sistema se puede linealizar por inyecciones aditivas entrada/salida. Referencia: caso de uso *linearization by additive output injections*.

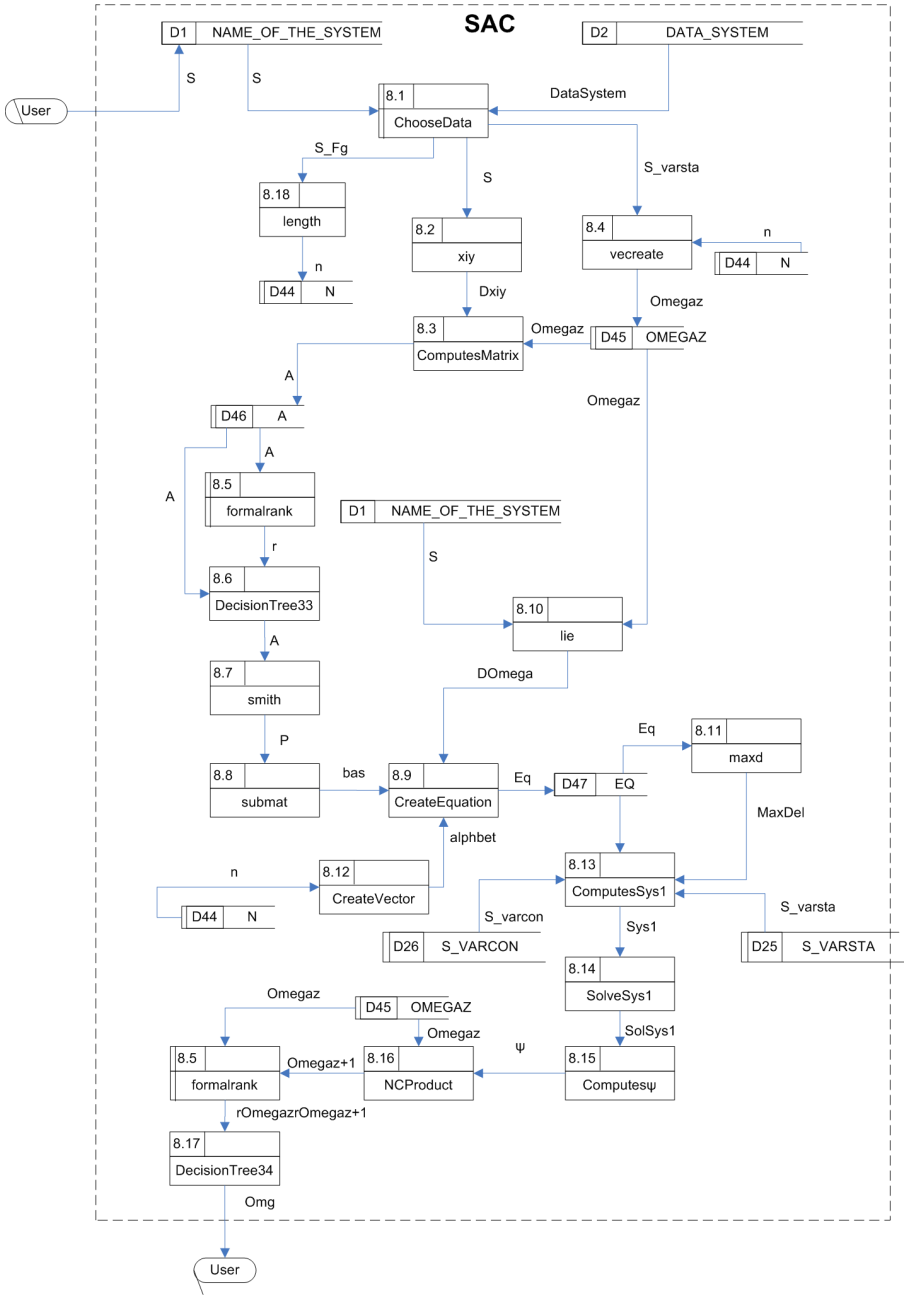


Figura 37: Diagrama de flujo de la rutina que calcula el máximo submódulo que no es afectado por ninguna entrada de control o perturbación. Referencia: caso de uso: *omg*.

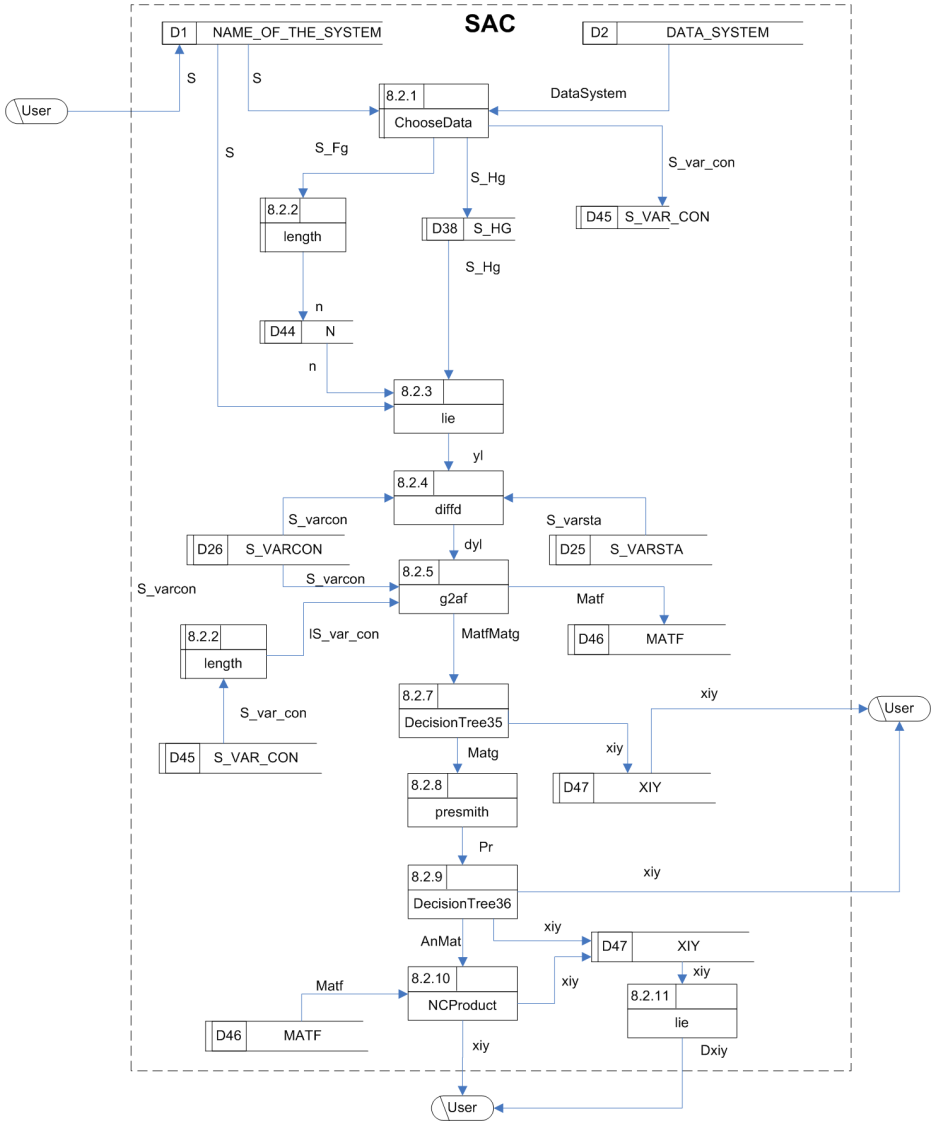


Figura 38: Diagrama de flujo de la rutina que calcula el submódulo $\mathcal{X} \cap \mathcal{Y}$. Referencia: caso de uso *xiy*.

A.3 Diagramas de árboles de decisión

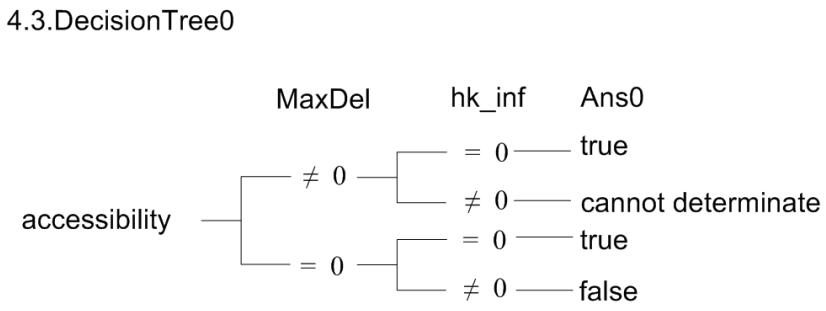


Figura 39: Diagrama de árbol de decisión. Referencia: ver Figura 16.

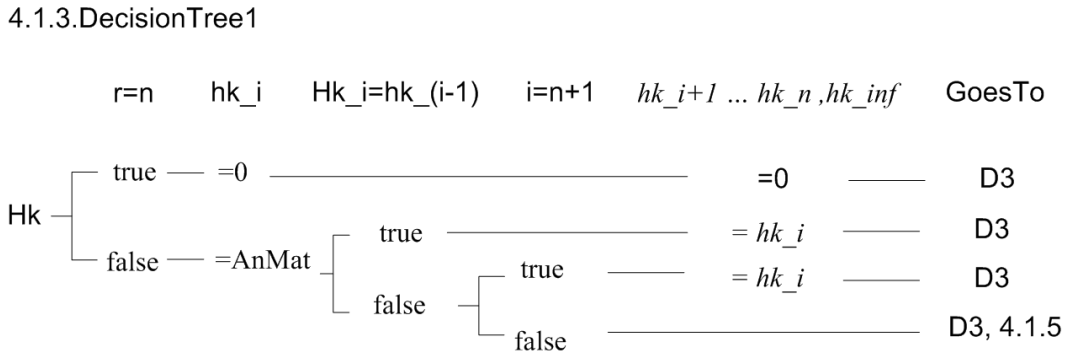


Figura 40: Diagrama de árbol de decisión. Referencia: ver Figura 17.

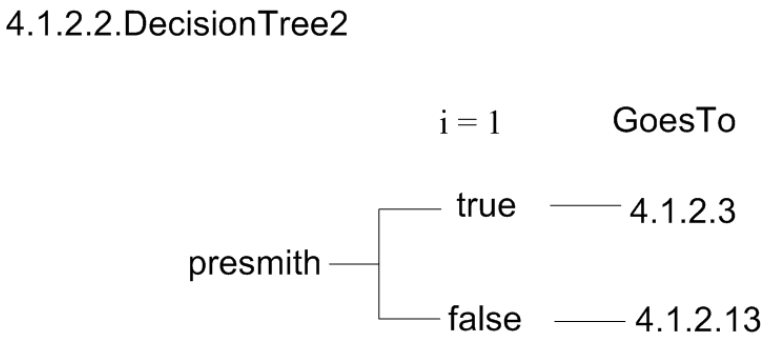


Figura 41: Diagrama de árbol de decisión. Referencia: ver Figura 18.

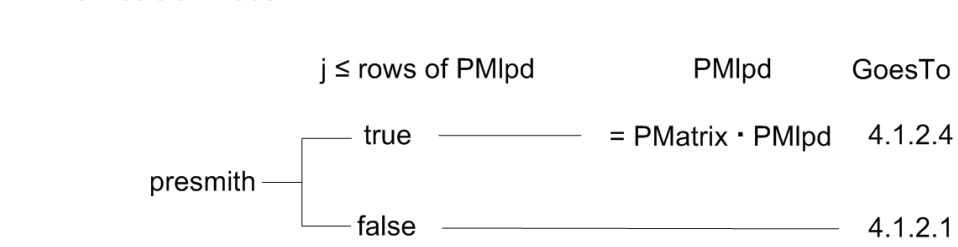


Figura 42: Diagrama de árbol de decisión. Referencia: ver Figura 18.

4.1.2.9.DecisionTree4

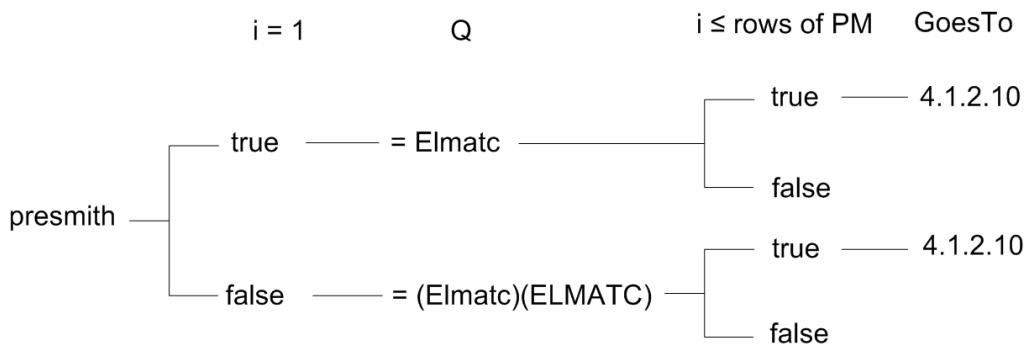


Figura 43: Diagrama de árbol de decisión. Referencia: ver Figura 18.

4.1.2.11.DecisionTree5

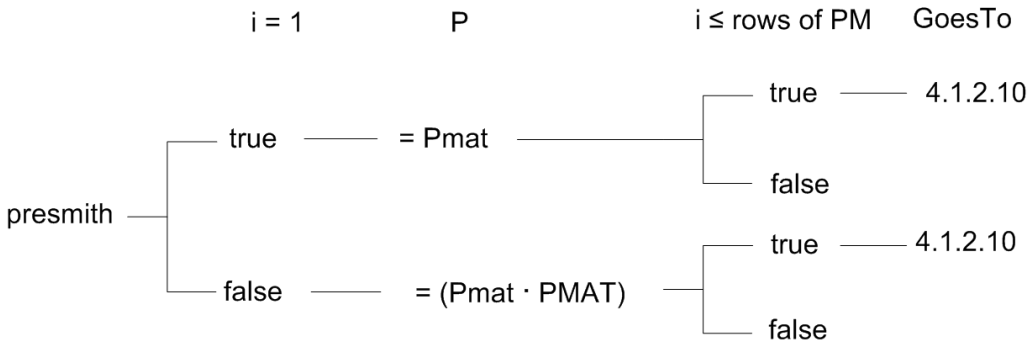


Figura 44: Diagrama de árbol de decisión. Referencia: ver Figura 18.

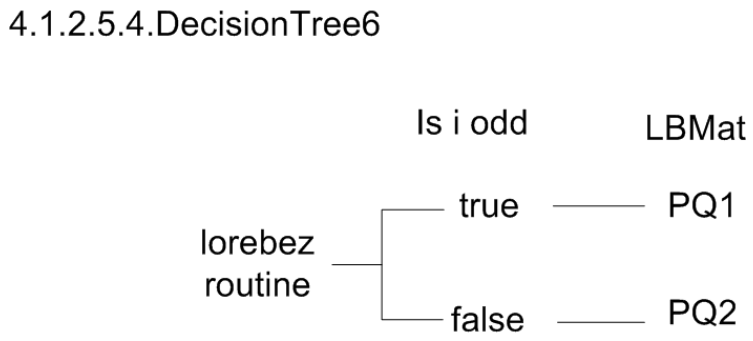


Figura 45: Diagrama de árbol de decisión. Referencia: ver Figura 20.

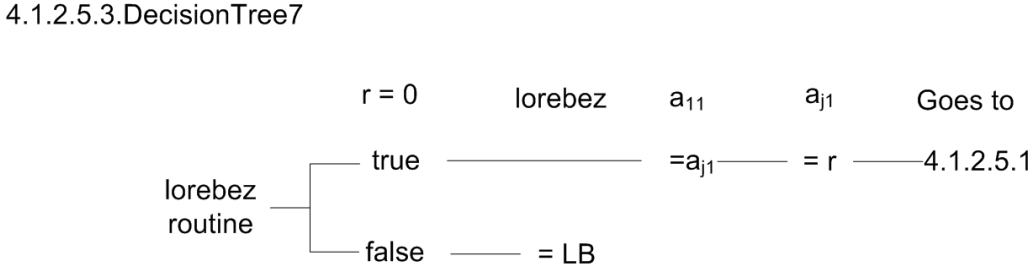


Figura 46: Diagrama de árbol de decisión. Referencia: ver Figura 20.

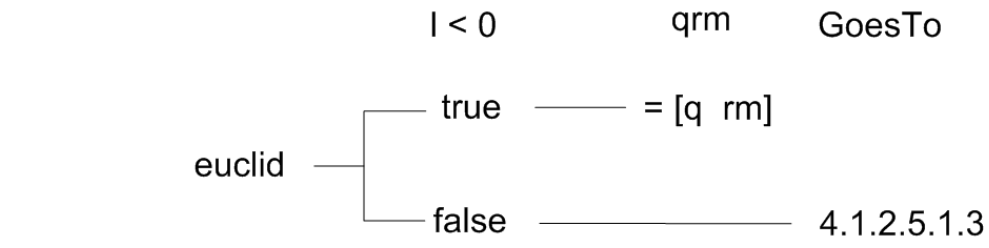


Figura 47: Diagrama de árbol de decisión. Referencia: ver Figura 21.

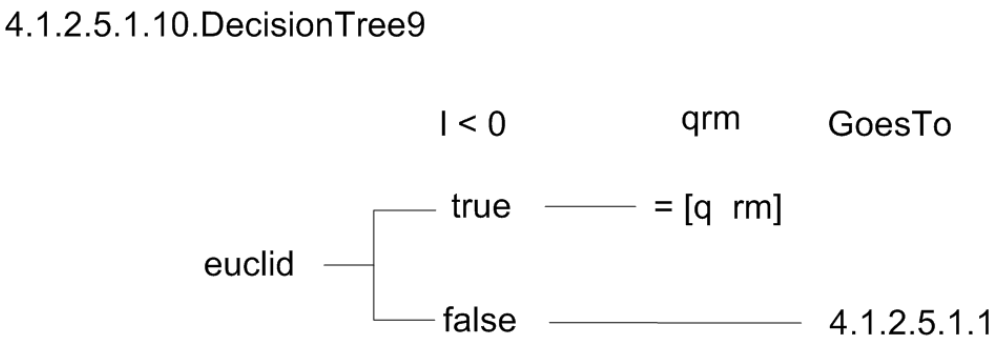


Figura 48: Diagrama de árbol de decisión. Referencia: ver Figura 21.

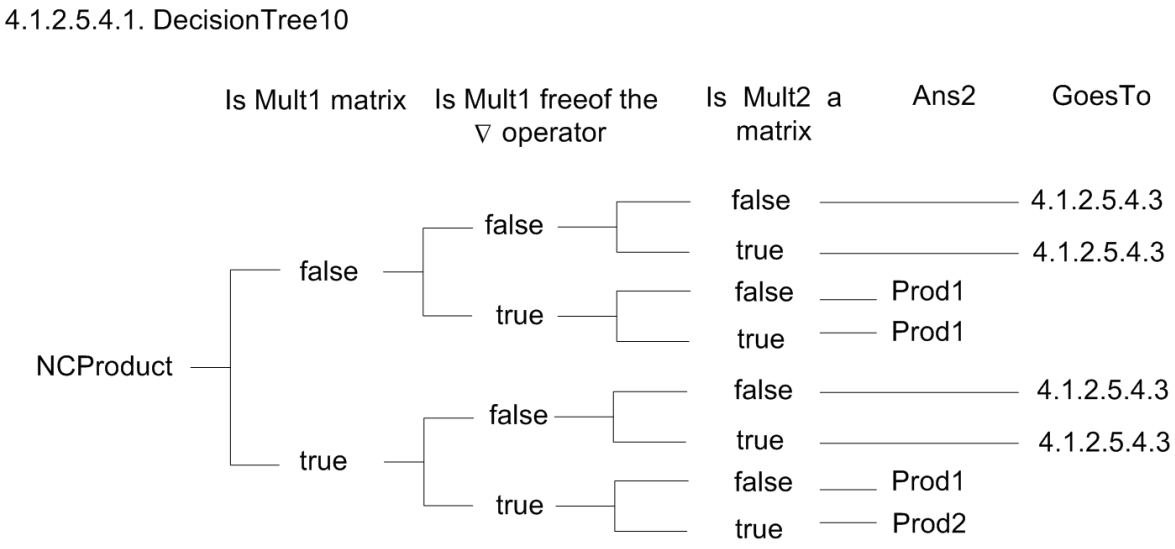


Figura 49: Diagrama de árbol de decisión. Referencia: ver Figura 22.

4.1.2.5.4.4. DecisionTree11

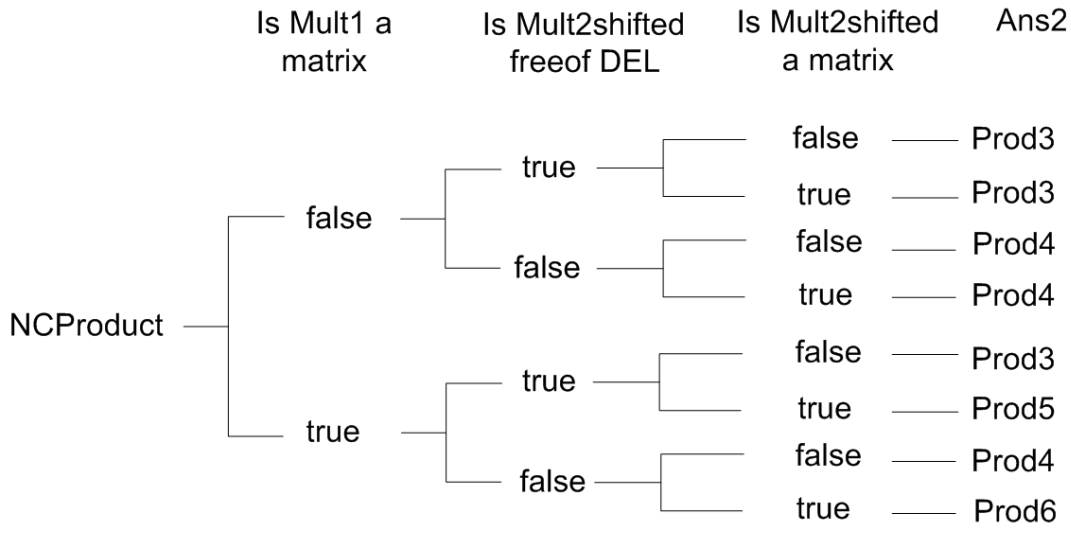


Figura 50: Diagrama de árbol de decisión. Referencia: ver Figura 22.

5.6.DecisionTree12

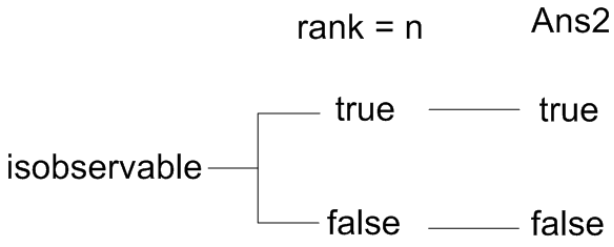


Figura 51: Diagrama de árbol de decisión. Referencia: ver Figura 25.

6.2.DecisionTree13

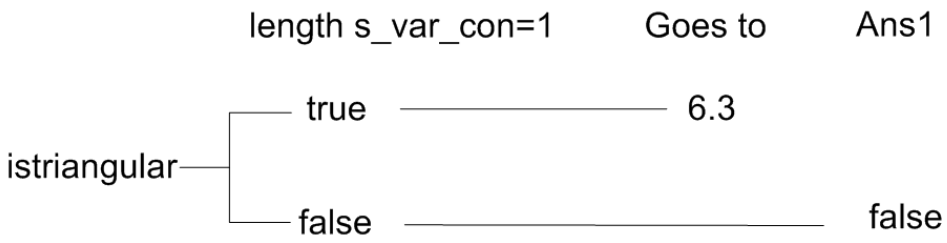


Figura 52: Diagrama de árbol de decisión. Referencia: ver Figura 27.

6.4.DecisionTree14

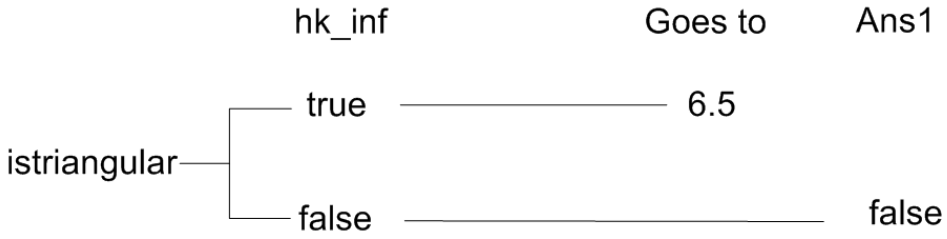


Figura 53: Diagrama de árbol de decisión. Referencia: ver Figura 27.

6.6.DecisionTree15

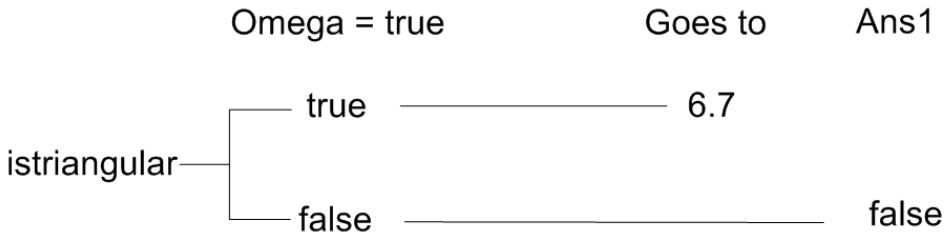


Figura 54: Diagrama de árbol de decisión. Referencia: ver Figura 27.

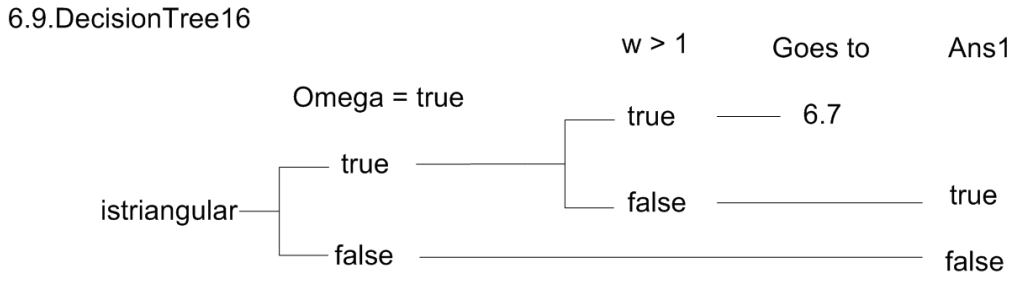


Figura 55: Diagrama de árbol de decisión. Referencia: ver Figura 27.

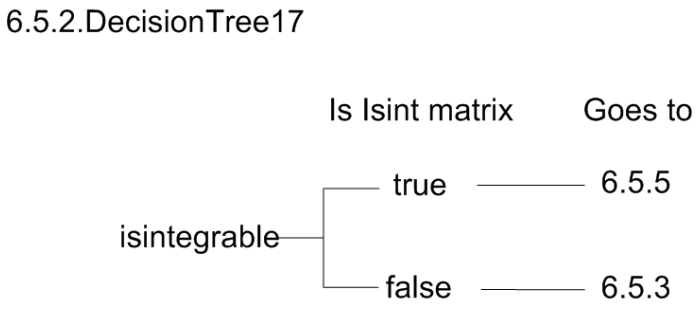


Figura 56: Diagrama de árbol de decisión. Referencia: ver Figura 28.

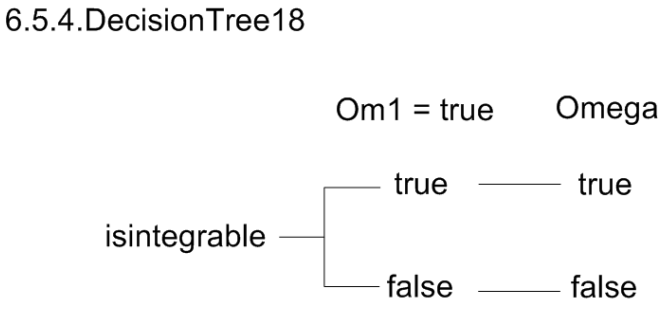


Figura 57: Diagrama de árbol de decisión. Referencia: ver Figura 28.

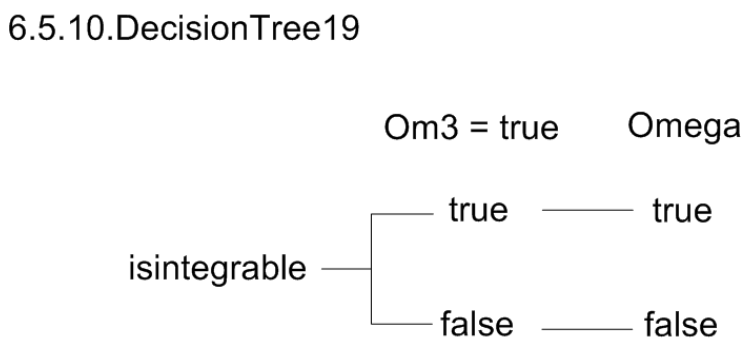


Figura 58: Diagrama de árbol de decisión. Referencia: ver Figura 28.

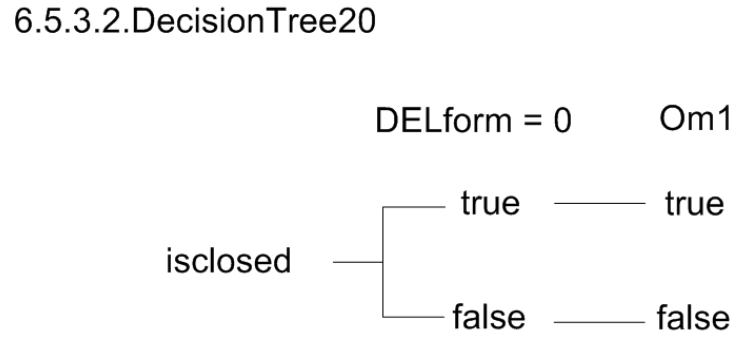


Figura 59: Diagrama de árbol de decisión. Referencia: ver Figura 29.

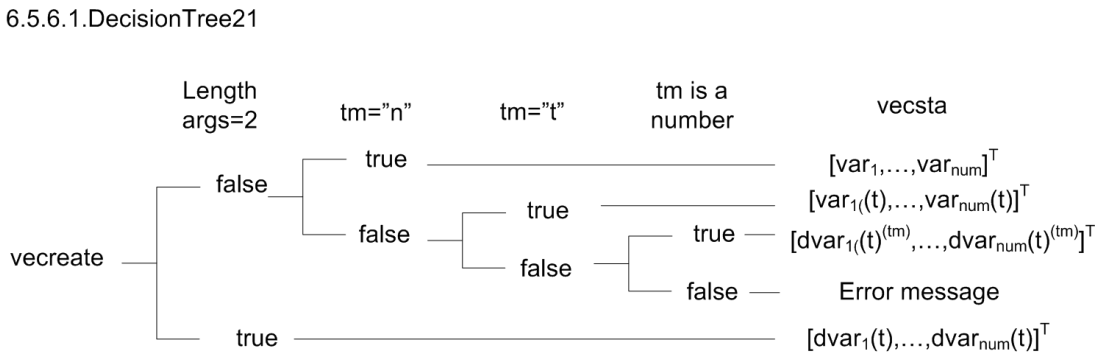


Figura 60: Diagrama de árbol de decisión. Referencia: ver Figura 30.

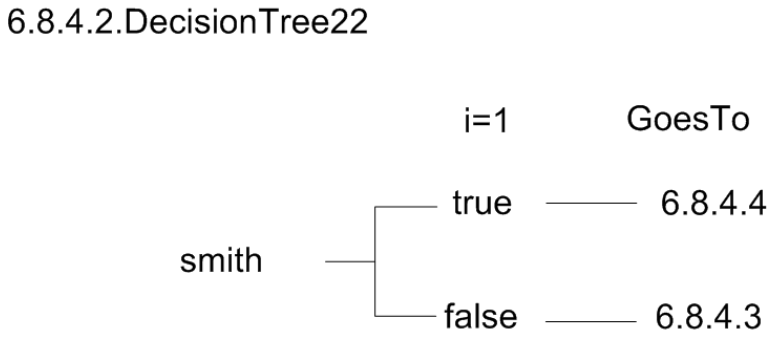


Figura 61: Diagrama de árbol de decisión. Referencia: ver Figura 33.

6.8.4.9.DecisionTree23

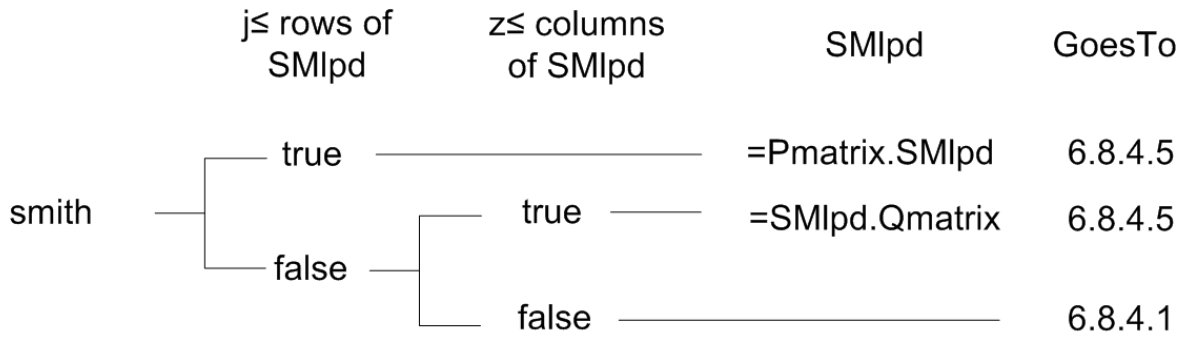


Figura 62: Diagrama de árbol de decisión. Referencia: ver Figura 33.

6.8.4.13.DecisionTree24

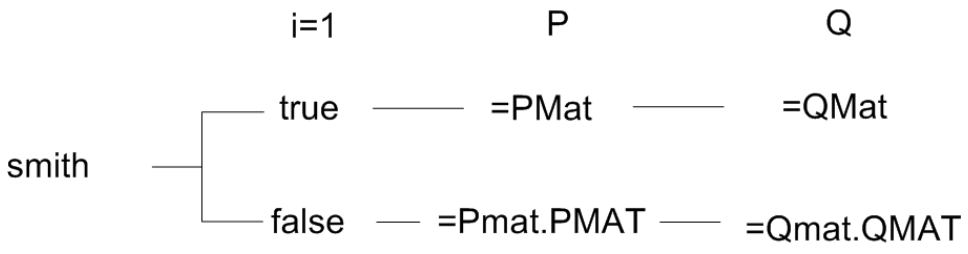


Figura 63: Diagrama de árbol de decisión. Referencia: ver Figura 33.

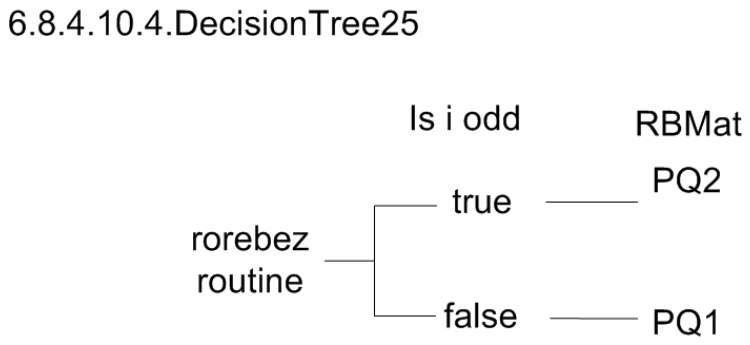


Figura 64: Diagrama de árbol de decisión. Referencia: ver Figura 34.

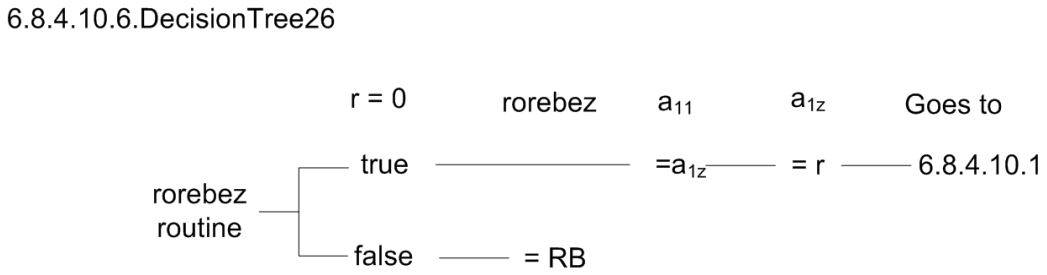


Figura 65: Diagrama de árbol de decisión. Referencia: ver Figura 34.

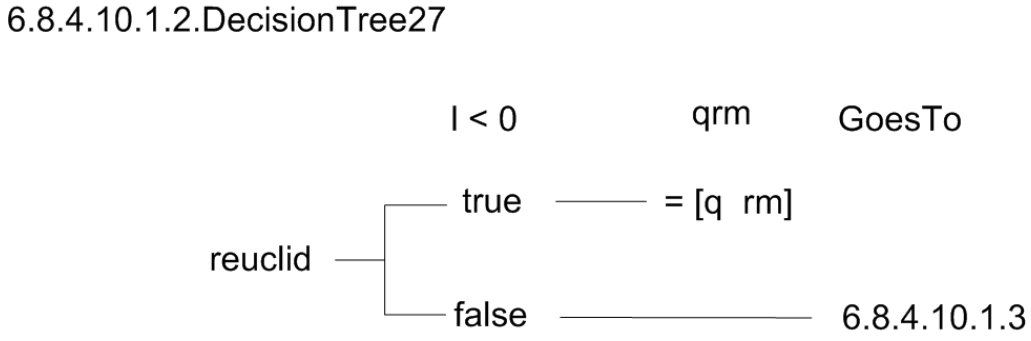


Figura 66: Diagrama de árbol de decisión. Referencia: ver Figura 35.

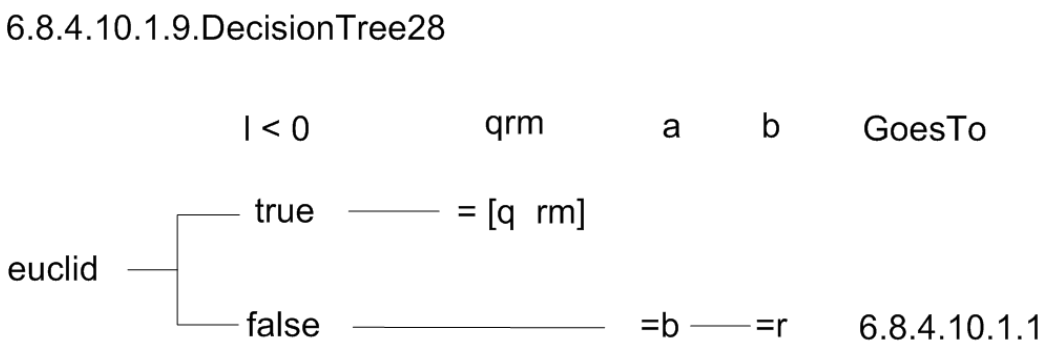


Figura 67: Diagrama de árbol de decisión. Referencia: ver Figura 35.

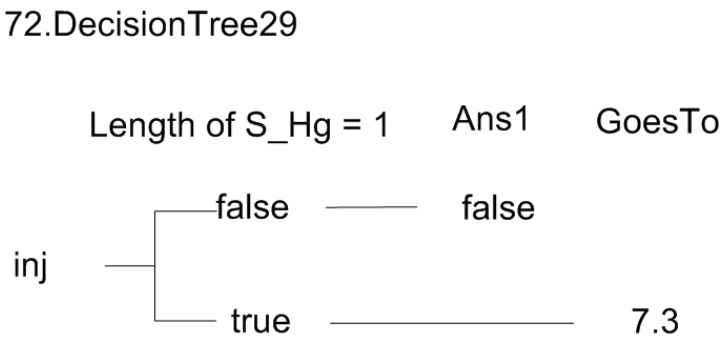


Figura 68: Diagrama de árbol de decisión. Referencia: ver Figura 36.

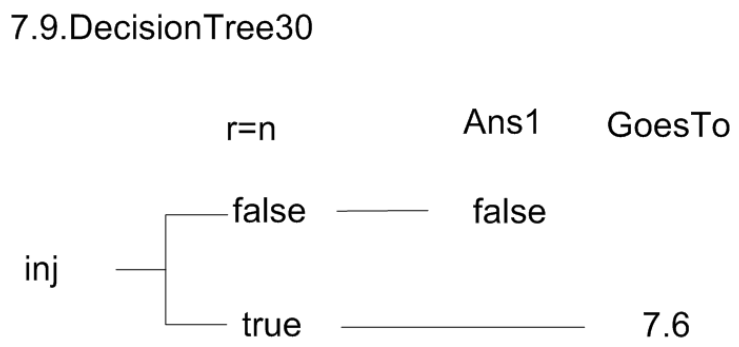


Figura 69: Diagrama de árbol de decisión. Referencia: ver Figura 36.

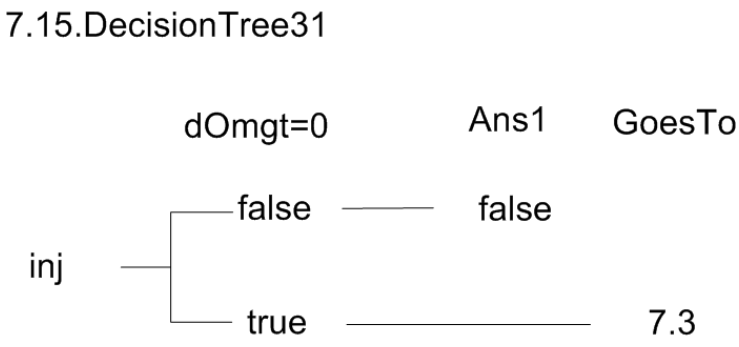


Figura 70: Diagrama de árbol de decisión. Referencia: ver Figura 36.

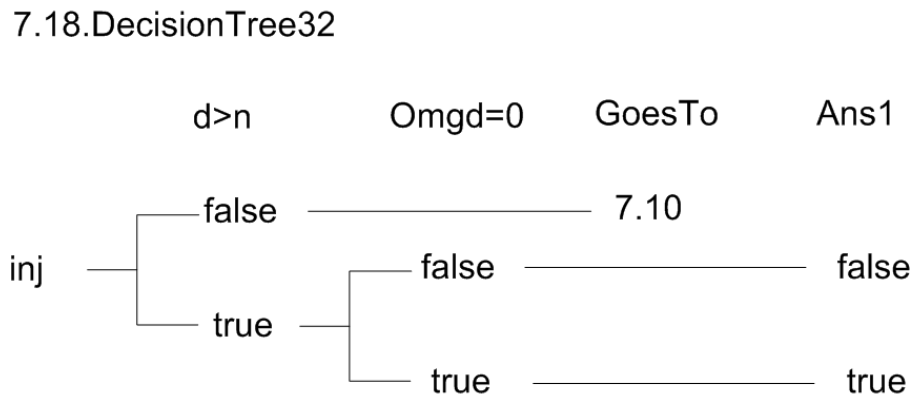


Figura 71: Diagrama de árbol de decisión. Referencia: ver Figura 36.

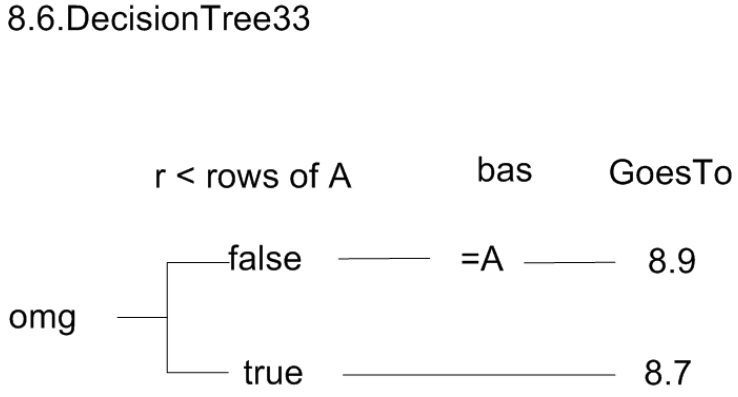


Figura 72: Diagrama de árbol de decisión. Referencia: ver Figura 37.

8.17.DecisionTree34

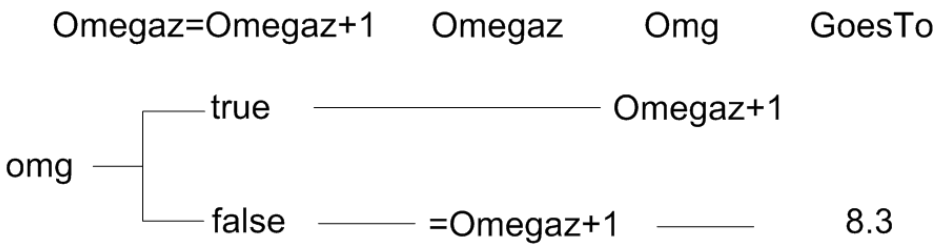


Figura 73: Diagrama de árbol de decisión. Referencia: ver Figura 37.

8.2.7.DecisionTree35

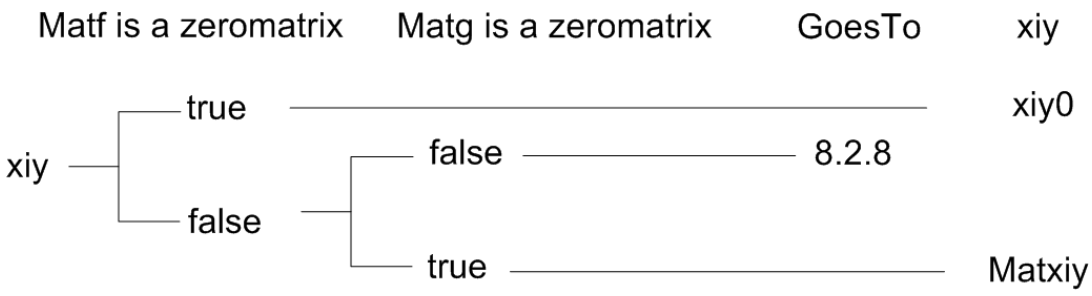


Figura 74: Diagrama de árbol de decisión. Referencia: ver Figura 38.

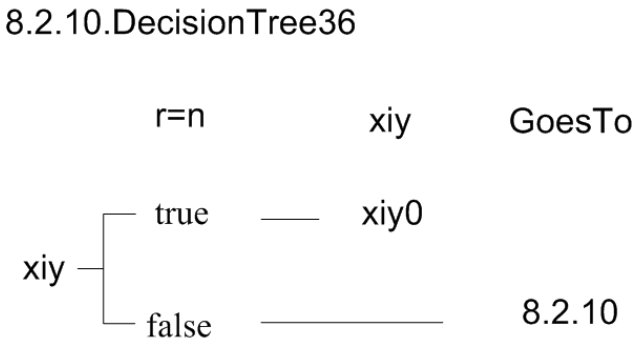


Figura 75: Diagrama de árbol de decisión. Referencia: ver Figura 38.

A.4 Diccionario de flujo de datos

A

$\alpha =$ vec+1.

$\alpha cols =$ α+cols.

$\Phi =$ *Matrix formed to compute the *omg* submodule. *For further details see the omg use case.**

A= *Matrix of the formed by the union of the matrixes *Dxiy* and *Omegaz*: $\begin{bmatrix} Dxiy \\ \dots \\ Omegaz \end{bmatrix}$.*

a= a_{11}.

$a_{11}a_{1z}$= *Elements a_{11} and a_{1z} of a matrix. *For further details see the z data.**

$a_{11}a_{j1}$= *Elements a_{11} and a_{j1} of a matrix. *For further details see the j data.**

AB= *Same information that ab.*

ab= a+b.

alph= $\left[\alpha_1, \alpha_2, \dots, \alpha_n\right]^T$.

alphbet= alph+bet.

AnMat= *Annihilator matrix. It is formed by the last rows of the matrix P.

$$AnMat = \begin{bmatrix} P_{r+1,1} & \dots & P_{r+1,n} \\ \vdots & \ddots & \vdots \\ P_{n,1} & \dots & P_{n,n} \end{bmatrix},$$

*for further details see presmith, smith and P data.**

Ans0= {true | false | cannot determinate}.
For further information see DecisionTree0 (4.3.DecisionTree0), in the decision tree diagrams section.

Ans1= {true | false}.

Ans2= {Prod1 | Prod2 | Prod3 | Prod4 | Prod5 | Prod6}.

args= {var∧num∧tm}.
For further information see DecisionTree20 in the decision trees diagram section.

at= *Leading coefficient of a.*

atbt= at+bt.

ATBTDIVIDED= *Same information that atbtdivided.*

atbtdivided= $\frac{at}{btshifted} \cdot \nabla^l$.
** For further information see l, at and btshifted data.**

atbtdividedb= $atbtdivided * \wedge b$.

ATBTDIVIDEDR= *Same information that atbtdividedr.*

atbtdividedr= $\frac{at}{bt} \cdot \nabla^l$.
* *For further information see l, at and bt data.**

B

b= a_{j1}.

bas= $AnMat$.
It is a basis for the elements of the matrix A.

batbtdividedr= $b * \wedge atbtdividedr$.

bet= $\left[\beta_1, \beta_2, \ldots, \beta_n\right]^T$.

bt= *Leading coefficient of b.*

btshifted= *bt shifted l time units.*

C

cm= *Length of the columns of the matrix Isint.*

cols= *Number of columns in $hk\_w + 1$.*

Column= *Column number of the element of lowest polynomial degree in the matrix MD of the $FindElement$ process. *For further details see the $MinDeg$ diagram in the data flow diagram section.**

D

d= *Counter variable. It starts in $d = 1$.*

DATA_SAVED= *Same information that DataSaved data.*

DATA_SYSTEM= *Same information that DataSystem data.*

DataLoaded= *Same information that DataSaved data, but with the name $NewName$ for the control system instead S.*

DataSaved= *Same information that DataSystem saved in the $FileName.sd$ file.*

DataSystem= S_dF+S_dG+S_dH+S_F+S_Fg+S_G+S_H+S_Hg+S_P+S_var_con+S_var_sta+S_var_out.

DefSystem= RepresentationType+S+SystemEquations+Vars.

DEL= *Maxima uses DEL to indicate a differential.*

DELpol= *Wedge product of the polynomials given in pol.

DELIsinteg= dIsinteg.
Differential form of $Isinteg$.

DELS_dF= dS_dF.
Differential form of S_dF.

DELform= {DELS_dF|DELIsinteg}
*Differential form of the input data of the $diffd$ process.

DELS_Fg= dS_Fg.
Differential form of S_Fg.

DELS_VAR_STAP= *Same information that $DELS\_var\_stap$.*

DELS_var_stap= $\left[d\dot{x}_1, \quad \ldots, \quad d\dot{x}_n\right]^T$.

Delta= *Matrix of the form:

$$\triangle = \sum_{i=0}^{MaxDel} \begin{bmatrix} \frac{\partial h_1(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_1(\cdot)}{\partial x_n(t-i)} \\ \vdots & & \vdots \\ \frac{\partial h_p(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_p(\cdot)}{\partial x_n(t-i)} \\ \frac{\partial h_1^{(1)}(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_1^{(1)}(x(t-i))}{\partial x_n(t-i)} \\ \vdots & & \vdots \\ \frac{\partial h_p^{(1)}(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_p^{(1)}(\cdot)}{\partial x_n(t-i)} \\ & \vdots & \\ \frac{\partial h_1^{(n-1)}(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_1^{(n-1)}(\cdot)}{\partial x_n(t-i)} \\ \vdots & & \vdots \\ \frac{\partial h_p^{(n-1)}(\cdot)}{\partial x_1(t-i)} & \cdots & \frac{\partial h_p^{(n-1)}(\cdot)}{\partial x_n(t-i)} \end{bmatrix} f(\cdot) \cdot \nabla^i.$$

*$f(\cdot)$ represents the vectors of S_Fg. For further information see S_Hgp, $S\_Hg^{(v)}$, S_Hg and the isobservable use case.**

Dg= *Differential of g with respect to t.*

DIFDER= *Same information that $difder$.*

difder= *Matrix of the form:

$$\begin{bmatrix} dh \\ dh^{(1)} \\ dh^{(2)} \\ \vdots \\ dh^{(n-1)} \end{bmatrix}.$$

It is formed by the differential form of the lie derivatives of S_Hg. $S\_Hg = h$ because there is only one output ($p = 1$).*

difders= *List of equations formed by $difder$ shifted from one time unit thru $MaxDel$ time units. *

DOmega= *1st. Lie derivative of Omegaz.*

dOmgt= *Differential form of $Omgt$.*

Dxiy= $\frac{d\mathcal{X} \cap \mathcal{Y}}{dt}$.
It is a matrix. For further details see xiy or omg use case.

dyl= *Differential form of yl.*

dyn= *Differential form of dyp.*

E

E= *Coefficient of the $n-th.$ Lie derivative of dyn if $d = 1$ or of the $(n-d)-th.$ Lie derivative of omgd for $d > 1$.*

ELMATC= *Elmatc previously stored in ELMATC data.*

Elmatc= *Elementary matrix to interchange two columns of a matrix.*

Elmatr= *Elementary matrix to interchange two rows of a matrix.*

Elmats= Elmatr+Elmatc.

Eq= $alph *^{D} omega - bet *^{b} as$.

EXT_DIFF= *Same information that ext_diff data.*

ext_diff= *Differential form of Newform data given by the cartan package. *Further details in diffd use case.**

F

FileName= *Name of the file saved by the function *savesyst* with extension *.sd*.*

FileNameNewName= FileName+NewName.

FORM= *Same information that form.*

form= S_dF.

G

g= $\{g_1 \mid g_{i+}\}$.

g_1**=** S_dG.

g_i**=** *Choose the variable g_i depends on the counter value i.*

g_{i+}**=** *Computes $g_{i+1} = \begin{bmatrix} g_i & g_1 \cdot g_i + \dot{g}_i \end{bmatrix}$. Starting from i=1 thru i=n+1.*

gp= $\dot{g}$.

gS_Fg= g+S_Fg.

H

hk_k= *Submodule $\mathcal{H}_k$ of the S system. The submodule number, k, is specified by the instruction Shk_k. *For further details see k,Shk_k and SubmodulesHk.**

hk_w= *Submodule $\mathcal{H}_w$ of the S system, where w is a counter.*

HK_W+1= *Same information that $hk\_w + 1$.*

hk_w+1= *Submodule $\mathcal{H}_w$ of the S system, where w is a counter.*

hk_wn= *Submodule $\mathcal{H}_w$ with a complete basis. *Further details in bascom use case.**

I

I= *Same information that i.*

i= *Counter variable, starts by default in $i = 0$. Except for g_{i+}.*

inf= *∞ symbol.*

ISINT= *Same information that Isint.*

Isint= {hk_n|hk_wn}.
Input data of isintegrable.*

Isinteg= Isint+S_varsta+S_varcon.

J

j= *Counter variable, starts by default in $j = 0$.*

K

k= $k = \{1, 2, \ldots, n \mid inf\}$.

L

l= pda-pdb.

LB= $LBMat * \wedge LBMAT$.

LBMAT= *LBMat previously stored in LBMAT data.*

LBMat= PQ.
Answer according to the $DecisionTree6$.*

lbez= *Left bezout polynomials or left polynomials to get the greatest common divisor.*

L1= *The input data for the cartan package must be atoms, then $L1$ is a list of equations to relate each variable of $form$ or pol data, which are not atoms because they depends on t, with an atom variable.*

L2= *List of equations $L1$ with the terms interchanged, the right term of each equation is at the left and the left at the right.*

LIST1= *Same information that List1.*

List1= *List equations of the form $\left[\dot{x}_1 = S\_Fg_1, \quad \dot{x}_2 = S\_Fg_2, \quad \ldots, \quad \dot{x}_n = S\_Fg_n\right]$.*

LIST2= *Same information that List2.*

List2= *List equations of the form $\left[d\dot{x}_1 = dS\_Fg_1, \quad d\dot{x}_2 = dS\_Fg_2, \quad \ldots, \quad d\dot{x}_n = dS\_Fg_n\right]$.*

List3= $\left[\dot{x}_1(t) = S\_Fg_1(t), \dot{x}_1(t-1) = S\_Fg_1(t-1), \ldots, \dot{x}_1(t - MaxDel) = S\_Fg_1(t - MaxDel)\right]$
$\left[\dot{x}_2(t) = S\_Fg_2(t), \quad \ldots, \quad d\dot{x}_n(t - MaxDel) = dS\_Fg_n(t - MaxDel)\right]$
List1 and List2 shifted from 1 time unit thru MaxDel time units.*

lore= *Left ore polynomials.*

lorebez= lore+lbez.
*For further information see the use case lorebez.**

lS_var_con= *Length of S_var_con.*

M

M= *Smith form of a matrix A. It is obtained multiplying $P \cdot A \cdot Q = M$.

$$\begin{bmatrix} P_{1,1} & \ldots & P_{1,n} \\ \vdots & \ddots & \vdots \\ P_{n,1} & \ldots & P_{n,n} \end{bmatrix} \begin{bmatrix} A_{1,1} & \ldots & A_{1,m} \\ \vdots & \ddots & \vdots \\ A_{n,1} & \ldots & A_{n,m} \end{bmatrix} \begin{bmatrix} Q_{1,1} & \ldots & Q_{1,n} \\ \vdots & \ddots & \vdots \\ Q_{n,1} & \ldots & Q_{n,n} \end{bmatrix} = diag\{s_{1,1}, \ldots, s_{r,r}\}.$$

m= *Number of control inputs in the system.*

MATF= *Same information that $Matf$.*

Matf= $dyl - Matg * \wedge \left[du_1, du_2, \ldots, du_m\right]^T$.

Matg= *Matrix formed by the differential of dyl with respect to du.*

MatfMatg= Matf+Matg.

Matxiy= *Matrix formed by the union of the matrixes $xiy0$ and $Matf$: $\begin{bmatrix} xiy0 \\ \dots \\ Matf \end{bmatrix}$.*

MAXDEL= *Same information that MaxDel.*

MaxDel= *Maximum time delay of the input data of the maxd process.*

MD= {PM| PMsubmat}.

Msta= *Matrix of the form:

$$Msta = \sum_{i=0}^{MaxDel} \begin{bmatrix} \dfrac{\partial dh}{\partial dx_1(t-i)} & \cdots & \dfrac{\partial dh}{\partial dx_n(t-i)} \\ \dfrac{\partial dh^{(1)}}{\partial dx_1(t-i)} & \cdots & \dfrac{\partial dh^{(1)}}{\partial dx_n(t-i)} \\ \vdots & & \vdots \\ \dfrac{\partial dh^{(n-1)}}{\partial dx_1(t-i)} & \cdots & \dfrac{\partial dh^{(n-1)}}{\partial dx_n(t-i)} \end{bmatrix}$$

Mult1= *Multiplier. It could be polynomial or matrix.*

Mult2= *Multiplicand. It could be polynomial or matrix.*

Mult2shiftedpoldeg= Mult2shifted+poldeg.

Mult2shifted= *$Mult2$ shifted *poldeg* time units.*

N

N= *Same information that n.*

n= *Number of state equations in the control system.*

NAME_OF_THE_SYSTEM= *Same information that S.*

NCP_INPUT= *Same information that NCPInput.*

NCPInput= Mult1+Mult2.

Newbasis= $wt * \wedge vecsta$.

Newform= *Polynomial with the basis variables replaced for atoms variables. *For further details see* $L1$.*

NewName= *Name assigned to the control system saved in the *FileName.sd* file and loaded using the instruction *loadsyst**.

Newpol= *Polynomial with the basis variables replaced for atoms variables. *For further details see* $L1$.*

num= *Number of elements in the vector created by the *vecreate* process.*

O

Om1= {true|false}
Answer of the isclosed process.*

Om2= $Isint * \wedge vecsta$.
It is a vector of the form $\left[\omega_1, \omega_2, \dots, \omega_n\right]^T$.

Om3= $d\omega_n \wedge \omega_n \wedge \omega \wedge \dots \wedge \nabla^{MaxDel}\omega$.
$\omega = \omega_1 \wedge \dots \wedge \omega_{n-1}$. and $k \in I\!N$.*

Omega= {true|false}
Answer of the *isintegrable* process.*

Omega0= $\begin{bmatrix} dx_1 \\ dx_2 \\ \vdots \\ dx_n \end{bmatrix}$.
*It represents $span_{\mathcal{K}[\nabla]}\{dx_1, dx_2, \ldots, dx_n\}$. *For further details see omg use case.**

OMEGAZ= *Same information that Omegaz*.

Omegaz= $\Phi Omega_{z-1}$.
z is a counter. It starts with Omega0.

Omegaz+1= $\Phi Omegaz$.
z is a counter.

Omg= *Maximum submodule which is not affected by any control input or disturbance in a control system. *Further details in the use case omg.**

Omg1= $\{dyn - Theta \mid omgd - Theta\}$.

Omgphi= *(n-d+1)-th. Lie derivative of Omgt.*

Omgphid= Omgphi+d.

Omgt= $E * \wedge dy + Theta * \wedge du$.

OMGD= *Same information that Omgd.*

Omgd= $Omgd_{(d-1)} - Omgt$.
Where $Omgd_0 = dyn$.*

P

P= *Matrix P, it is used to find the presmith and the smith form of a matrix A. It multiplies A by the left. *For further details see the presmith and smith data.**

p= *Number of outputs in the control system.*

pda= *Polynomial degree of a.

pdapdb= pda+pdb.

pdb= *Polynomial degree of b.

PM= g.

PMAT= *PMat previously stored in PMAT data.*

PMat= $Elmatr * \wedge PMatrix$.
Stores the multiplication of the matrix needed to form the matrix P.*

PMATRIX= *Same information that $Pmatrix$.*

Pmatrix= *Matrix that multiplies by the left to $PMlpd$ and makes zero the first column elements of this matrix.*

PMi= PM+i.

PMLPD= *Same information that PMlpd.*

PMlpd= *PM matrix with the element of lowest polynomial degree on the top left corner.*

PMlpdj= PMlpd+j.

PMsubmat= *Submatrix which is formed deleting the first row and column of the input matrix of the *submat* process.*

POL= *Same information that pol.*

pol= *Input data to the *wedgeproduct* process. It must be two polynomials.*

poldeg= *Maximum exponent of ∇ in $Mult1$.*

PQ= {PQ1|PQ2}.

PQ1= *Matrix of the form $PQ1 = \begin{bmatrix} 1 & -c \\ 0 & 1 \end{bmatrix}$.*

PQ2= *Matrix of the form $PQ2 = \begin{bmatrix} 1 & 0 \\ -c & 1 \end{bmatrix}$.*

PQM= P+Q+M.

PQMat= P+Q.

PQR= P+Q+R.

PQRr= P+Q+R+r.

PQRri= P+Q+R+r+i.

Pr= P+r.

Prod1= Mult1*Mult2.

Prod2= Mult1.Mult2.

Prod3= Mult1*Mult2shifted∇^{poldeg}.

Prod4= Mult1*Mult2shifted.

Prod5= Mult1.Mult2shifted∇^{poldeg}.

Prod6= Mult1.Mult2shifted.

Q

Q= *Matrix Q, it is used to find the presmith and the smith form of a matrix A. It multiplies A by the right. *For further details see the presmith and smith data.**

q= {q+atbtdivided|q+atbtdividedr}.
Quotient obtained of the *euclid* or *reuclid* division process respectively. The initial value is 0.*

Qα = Q+α.

QMAT= *QMat previously stored in QMAT data.*

QMat= $QMatrix * \wedge Elmatc$.
Stores the multiplication of the matrix needed to form the matrix Q.*

QMATRIX= *Same information that $Qmatrix$.*

Qmatrix= *Matrix that multiplies by the right to $SMlpd$ and makes zero the first row elements of this matrix.*

QRM= *Same information that qrm.*

qrm= q+rm.

qrmi= q+rm+i.

R

R= *Presmith form of a matrix A. It is obtained multiplying $P \cdot A \cdot Q = R$:

$$\begin{bmatrix} P_{1,1} & \dots & P_{1,n} \\ \vdots & \ddots & \vdots \\ P_{n,1} & \dots & P_{n,n} \end{bmatrix} \begin{bmatrix} A_{1,1} & \dots & A_{1,m} \\ \vdots & \ddots & \vdots \\ A_{n,1} & \dots & A_{n,m} \end{bmatrix} \begin{bmatrix} Q_{1,1} & \dots & Q_{1,n} \\ \vdots & \ddots & \vdots \\ Q_{n,1} & \dots & Q_{n,n} \end{bmatrix} = \left[\begin{array}{ccc} s_{1,1} & \dots & s_{1,r} & \\ & \ddots & \vdots & \\ (0) & & s_{r,r} & (0) \\ \hline & & (0) & \end{array}\right].$$

r= *Formal rank of a matrix A. *Further details in the use case formal rank.**

rank= *Rank of the input matrix of the $formalrank$ process.*

rbez= *Right bezout polynomials or right polynomials to get the greatest common divisor.*

RB= $RBMAT * \wedge RBMat$.

RBMAT= *RBMat previously stored in RBMAT data.*

RBMat= PQ.
Answer according to the $DecisionTree25$.*

RepresentationType= {"general" |"affine" | "differential"}
*A system may be defined in three ways within SAC, either as general, affine or differential. *More details in SAC manual.**

rm= rm-atbtdividedb.
Remainder obtained of the Euclidien division. The initial value is the a_{11} data.*

rOmegaz= *Rank of the submodule Omegaz.*

rOmegaz+1= *Rank of the submodule Omegaz+1.

rOmegazrOmegaz+1= rOmegaz+rOmegaz+1.

rore= *Right ore polynomials.*

rorebez= rore+rbez.
*For further information see the use case rorebez.**

Row= *Row number of the element of lowest polynomial degree in the matrix MD of the $FindElement$ process. *For further details see the $MinDeg$ diagram in the data flow diagram section.**

RowColumn= Row+Column.

S

S= *System name. It could contain only capital or small letters, numbers and the low dash "_". *

S_dF= *Matrix $\in I\!R^{n \times n}$ in the differential form of representation. *For further details see DataSystem and RepresentationType data.**

S_dG= *Matrix $\in I\!R^{n \times m}$ in the differential form of representation. *For further details see DataSystem and RepresentationType data.**

S_dH= *Matrix $\in I\!R^{y \times n}$ in the differential form of representation. *For further details see DataSystem and RepresentationType data.**

S_F= *Matrix $\in I\!R^{n \times 1}$ in the affine form of representation. *For further details see DataSystem and RepresentationType data.**

S_FG= *Same information that S_Fg.*

S_Fg= *Matrix $\in \mathcal{K}[\nabla]^{n\times 1}$ in the general form of representation. *For further details see DataSystem and RepresentationType data.**

S_FgS_Hg= S_Fg+S_Hg.

S_G= *Matrix $\in I\!R^{n\times m}$ in the affine form of representation. *For further details see DataSystem and RepresentationType data.**

S_H= *Matrix $\in I\!R^{p\times 1}$ in the affine form of representation. *For further details see DataSystem and RepresentationType data.**

S_HG= *Same information that S_Hg*.

S_Hg= *Matrix $\in I\!R^{p\times 1}$ in the general form of representation, with $S\_Hg = \begin{bmatrix} h_1, & \dots & h_p \end{bmatrix}$. *For further details see DataSystem and RepresentationType data.**

S_Hg^v**=** *The vth. Lie derivative of S_Hg, with v=[1,2,...,n-1]. *For further details see S_Hg and S_Hgp.**

S_Hgp= $S\_Hg + S\_Hg^{(1)} + S\_Hg^{(2)} + \dots \S\_Hg^{(n-1)}$.

S_P= *Matrix $\in I\!R^{n\times 1}$ in the affine form of representation. It represents the matrix of disturbances in the affine form of representation.*

S_VAR_CON= *Same information that *s_var_con*.

S_var_con= *Vector $\in I\!R^{m\times 1}$ with the control variables of the S system.*

S_var_out= *Vector $\in I\!R^{p\times 1}$ with the output variables of the S system.*

S_VAR_STA= *Same information that S_var_sta.*

S_var_sta= *Vector $\in I\!R^{n\times 1}$ with the state variables of the S system.*

S_VAR_STAP= *Same information that S_var_stap.*

S_var_stap= $\begin{bmatrix} \dot{x}_1, & \dots, & \dot{x}_n \end{bmatrix}^T$.

S_VARCON= *Same information that S_varcon.*

S_varcon= *Control variable base of the S system.*

S_varout= *Output variable base of the S system.*

S_VARSTA= *Same information that *S_varsta*.*

S_varsta= *State variable base of the S system.*

S_varstaS_varcon= S_varsta+S_varcon.

SFileName= S+FileName.

Sg= S+g.

Shk_k= *Specify the number of submodule $\mathcal{H}_k$ of the S system that the user want to see.*

SM= *Input data to the *smith* process.*

SMi= SM+i.

SMLPD= *Same information that SMlpd.*

SMlpd= *SM matrix with the element of lowest polynomial degree on the top left corner.*

SMlpdjz= SMlpd+j+z.

SolSys1= *Solution of $Sys1$ with respect to the α variables.*

SS_Hg= S+S_Hg.

SUBMODULES_HK= *Same information that SubmodulesHk.*

SubmodulesHk= $hk\_1 + hk\_2 + \cdots + hk\_n + hk\_inf$.
Filtration $\mathcal{H}_k$.

Sys1= *Equations system to compute the Omg submodule. It is formed by the differential of Eq with respect to the S_varcon and S_varsta.*

SystemDefined= *The system displays the word "System Defined" when the *systdef* command finish the computations.*

SystemEquations= *It's a list with the equations system, which are represented in matrix form. The elements change according to the representation system. *See the RepresentationType and DataSystem data.**

T

t= *Time variable.*

Theta= *Coefficient of the dth. Lie derivative of dyn.*

tm= {"n"|"t"| $tm \in \mathbb{N}$}.
For further details see the DecisionTree20 in the decision tree diagrams section.

U

u= S_varcon.

V

var= *Variable basis to create the vectors in the *vecreate* process.*

Vars= S_varsta+S_varcon+S_varout
It's a list with these variables.

varsform= *List of variables S_varsta_v and S_varcon_v found in $form$, with $v \in \mathbb{N}$.*

varspol= *List of variables W_varsta_v and W_varcon_v found in pol, with $v \in \mathbb{N}$.*

vec= *Number of vectors in $hk\_w + 1$.*

vecsta= *Vector created with the *vecreate* process.*

W

w= *Counter variable, starts by default in $w = n$.*

wt= $\left[0_{(\alpha-(\alpha-1))\times(\alpha-1)} \quad \mathbb{I}_{(\alpha-(\alpha-1))}\right] Q \cdot hk\_w + 1$.
It is the vector needed to complete the basis for hk_w+1.

W_varcon= *Variable basis to compute the wedgeproduct. It is default value is u.*

W_varsta= *Variable basis to compute the wedgeproduct. It is default value is x.*

W_varstaW_varcon= W_varsta+W_varcon.

WEDGE= *Same information that wedge data.*

wedge= *Differential form of Newpol data given by the cartan package. *Further details in wedgeproduct use case.**

X

x= S_varsta.

xiy0= $\begin{bmatrix} dx_1 \\ dx_2 \\ \vdots \\ dx_n \end{bmatrix}$.
*It represents $span_{\mathcal{K}[\nabla]}\{dx_1, dx_2, \ldots, dx_n\}$.

xiy= $\mathcal{X} \cap \mathcal{Y}$.

Y

yl= $S\_Hg + S\_Hg^{(1)} + S\_Hg^{(2)} + \ldots \S\_Hg^{(n)}$.
The exponent indicates the number of Lie derivative.

yp= $S\_Hg^{(1)} + S\_Hg^{(2)} + \ldots \S\_Hg^{(n)}$.
The exponent indicates the number of Lie derivative.

ypn-= $S\_Hg^{(1)} + S\_Hg^{(2)} + \ldots \S\_Hg^{(n-1)}$.
For further details see yp.

ypn= $S\_Hg^{(n)}$.
For further details see yp.

Z

Z= *Same information that z data.*

z= *Counter variable, starts by default in $z = 0$.*

Apéndice B

Pruebas de caja blanca

En este apéndice se muestran ejemplos de pruebas realizadas a algunas rutinas de SAC. Se explica el propósito de cada una de las pruebas, se diseñan datos de prueba para cada función y se muestran los resultados.

B.1 Diseño de pruebas

B.1.1 Prueba "creación de matrices elementales"

Esta prueba tiene la finalidad de verificar el módulo *elmatx*, el cual genera una matriz elemental que permite permutar renglones o columnas al ser multiplicada por otra matriz.

Sintaxis: `elmatx(`e_1`,`e_2`,dim)` FUNCIÓN

Datos de entrada:

$e_1, e_2 \Rightarrow \mathbb{Z}_+$.
: Renglones o columnas a intercambiar

`dim` $\Rightarrow \mathbb{Z}_+$.
: Dimensión de la matriz elemental

Dependencias: `ninguna`.

Objetivos

Los objetivos de esta prueba son:

1. Verificar que si un usuario ingresa los datos correctos se crea la matriz elemental.
2. Verificar que si un usuario ingresa datos inválidos se genera un mensaje de error.

Requisitos y restricciones

Los datos de entrada deben ser enteros positivos con $e_1, e_2 < dim$.

Descripción de la prueba

Esta prueba tiene cinco posibles escenarios:

- Escenario 1: El usuario ingresa datos válidos con $e_1 > e_2$.
- Escenario 2: El usuario ingresa datos válidos con $e_1 < e_2$.
- Escenario 3: El usuario ingresa datos inválidos con e_1, e_2 o dim nulos o menores que 0.
- Escenario 4: El usuario ingresa datos inválidos con e_1 o e_2 mayores a dim.
- Escenario 5: El usuario ingresa datos inválidos con e_1,e_2 o e_3 no enteros.

Resultados esperados

- Si el usuario ingresa datos válidos el sistema genera la matriz elemental de acuerdo a sus especificaciones.
- Si se ingresan datos inválidos se generan los siguientes mensajes de error:

  Escenario 3. Arguments must be greater than 0.

  Escenario 4. Wrong arguments. e_1 and e_2 must be smaller than 4.

  Escenario 5. Arguments must be integers.

Datos de prueba

Tabla XII: Datos de prueba para elmatx.

Dato de prueba	Escenario	e1	e2	dim
1	1	500	3	1000
2	2	4	6	10
3	3	0	2	10
4	3	1	0	10
5	3	1	2	0
6	3	-1	2	10
7	3	1	-2	10
8	3	1	2	-10
9	4	5	3	4
10	4	3	5	4
11	5	a	3	5
12	5	1.1	3	5

Resultado de los datos de prueba

Escenario 1
El resultado de los *datos de prueba 1* fue correcto, pero no se puede mostrar por legibilidad.
Escenario 2
Resultado de los *datos de prueba 2*.

```
(%i1) load("sac.mc")$
(%i1) elmatx(4,6,10);
         [ 1  0  0  0  0  0  0  0  0  0 ]
         [                               ]
         [ 0  1  0  0  0  0  0  0  0  0 ]
         [                               ]
         [ 0  0  1  0  0  0  0  0  0  0 ]
         [                               ]
         [ 0  0  0  0  0  1  0  0  0  0 ]
         [                               ]
         [ 0  0  0  0  1  0  0  0  0  0 ]
(%o1)    [                               ]
         [ 0  0  0  1  0  0  0  0  0  0 ]
         [                               ]
         [ 0  0  0  0  0  0  1  0  0  0 ]
         [                               ]
         [ 0  0  0  0  0  0  0  1  0  0 ]
         [                               ]
         [ 0  0  0  0  0  0  0  0  1  0 ]
         [                               ]
         [ 0  0  0  0  0  0  0  0  0  1 ]
```

Escenario 3
Resultado de *los datos de prueba 3-8*.

```
(%i2) elmatx(0,2,10);
Arguments must be greater than 0
#0: elmatx(_e1=0,_e2=2,_dim=10)
 -- an error.  Quitting.  To debug this try debugmode(true);

(%i3) elmatx(1,0,10);
Arguments must be greater than 0
#0: elmatx(_e1=1,_e2=0,_dim=10)
 -- an error.  Quitting.  To debug this try debugmode(true);

(%i4) elmatx(1,2,0);
Arguments must be greater than 0
#0: elmatx(_e1=1,_e2=2,_dim=0)
 -- an error.  Quitting.  To debug this try debugmode(true);

(%i5) elmatx(-1,2,10);
Arguments must be greater than 0
#0: elmatx(_e1=-1,_e2=2,_dim=10)
 -- an error.  Quitting.  To debug this try debugmode(true);

(%i6) elmatx(1,-2,10);
Arguments must be greater than 0
#0: elmatx(_e1=1,_e2=-2,_dim=10)
 -- an error.  Quitting.  To debug this try debugmode(true);

(%i7) elmatx(1,2,-10);
Arguments must be greater than 0
#0: elmatx(_e1=1,_e2=2,_dim=-10)
 -- an error.  Quitting.  To debug this try debugmode(true);
```

Escenario 4
Resultado de *los datos de prueba 9-10*.

```
(%i8) elmatx(5,3,4);
Wrong arguments. 5 and 3 must be smaller than 4
#0: elmatx(_e1=5,_e2=3,_dim=4)
 -- an error.  Quitting.  To debug this try debugmode(true);

(%i9) elmatx(3,5,4);
Wrong arguments. 3 and 5 must be smaller than 4
#0: elmatx(_e1=3,_e2=5,_dim=4)
 -- an error.  Quitting.  To debug this try debugmode(true);
```

Escenario 5
Resultado de *los datos de prueba 11-12*.

```
(%i10) a;
(%o11)  a

(%i12) elmatx(a,3,5);
Arguments must be integers
#0: elmatx(_e1=a,_e2=3,_dim=5)
 -- an error.  Quitting.  To debug this try debugmode(true);

(%i13) elmatx(1.1,3,5);
Arguments must be integers
#0: elmatx(_e1=1.1,_e2=3,_dim=5)
 -- an error.  Quitting.  To debug this try debugmode(true);
```

Tiempo de ejecución

El tiempo de ejecución de las rutinas se puede calcular en Máxima mediante la instrucción *timer*() y la información obtenida se muestra mediante la instrucción $timer\_info()$, cuya unidad de tiempo son los segundos o get(nombre de la función,$'runtime$) que utiliza milisegundos.
Antes de ejecutar los datos de prueba de la Tabla XII se especifica:

```
(%i14) timer(elmatx);
(%o14)    [elmatx]

(%i15) elmatx(500,3,1000)$

(%i16) get(elmatx,'runtime);
(%o16)      29
```

Para los datos de prueba 2:

```
(%i17) elmatx(4,6,10)$
(%i18) get(elmatx,'runtime);
(%o18)      0
```

B.1.2 Prueba "extraer una submatriz"

Esta prueba tiene la finalidad de verificar el módulo *submat*, el cual extrae una submatriz.

Sintaxis: `submat`(ii, f, a, ji, jf) FUNCIÓN

Datos de entrada:

$a \Rightarrow$ matriz.
Matriz de dimensión $\mathbb{R}^{n\times m}$, donde m puede ser igual a n.

$ii, f, ji, jf \Rightarrow \mathbb{Z}_+$.
Las variables ii, f son el primer y último del renglón de la nueva matriz, en tanto que ji, jf son la primera y última columna, respectivamente.

Dependencias: `ninguna`.

Objetivos

Los objetivos de esta prueba son:

1. Verificar que si un usuario ingresa los datos correctos se extrae la submatriz indicada.
2. Verificar que si un usuario ingresa datos inválidos se genera un mensaje de error.

Requisitos y restricciones

Las condiciones $ii \leq f \leq$renglones de a y $ji \leq jf \leq$columnas de a se deben satisfacer, con $ii, f, ji, jf \in \mathbb{Z}_+$ y a una matriz.

Descripción de la prueba

Los posibles escenarios son:

- Escenario 1: El usuario ingresa datos válidos con $ii < f$ y con $ji < jf$.
- Escenario 2: El usuario ingresa datos válidos con $ii = f$ y con $ji = jf$.
- Escenario 3: El usuario ingresa datos inválidos con $ii, f, ji, jf \leq 0$.
- Escenario 4: El usuario ingresa datos inválidos a no es una matriz.
- Escenario 5: El usuario ingresa datos inválidos con $ii > f$ o $ji > jf$.
- Escenario 6: El usuario ingresa datos inválidos con $f >$renglones de a o $jf >$columnas de a.
- Escenario 7: El usuario ingresa datos inválidos con ii, f, ji o jf no enteros.

Todos estos escenarios pueden ocurrir siendo a una matriz cuadrada o no cuadrada.

Resultados esperados

- Si el usuario ingresa datos válidos el sistema extrae la submatriz solicitada. En el caso en que $ii = f$ y $ji = jf$ se espera que el resultado sea un solo elemento de la matriz a.

- Si se ingresan datos inválidos se generan los siguientes mensajes de error:

  Escenarios 3 y 7: Arguments $ii\, f\, ji\, jf$ must be integers greater than 0.

  Escenario 4: Argument a must be a matrix.

  Escenario 5: Wrong arguments. ii and ji must be smaller than f and jf respectively.

  Escenario 6: Wrong arguments. f must be equal or smaller than the rows of errexp1 and jf must be equal or smaller than its columns.

Datos de prueba

Siendo la matriz a

```
(%i19) a1:matrix([10,20,30,40,50],[11,21,31,41,51],
                 [12,22,32,42,52],[13,23,33,43,53]);
       [ 10  20  30  40  50 ]
       [                    ]
       [ 11  21  31  41  51 ]
(%o19) [                    ]
       [ 12  22  32  42  52 ]
       [                    ]
       [ 13  23  33  43  53 ]
```

Tabla XIII: Datos de prueba para submat.

Dato de prueba	Escenario	ii	f	ji	jf
1	1	1	3	2	4
2	2	3	3	4	4
3	3	0	2	0	3
4	3	1	0	1	0
5	3	1	-2	2	4
6	5	3	2	1	4
7	5	2	4	3	1
8	6	1	7	2	3
9	6	1	4	2	8
10	7	1	mm	1	4
11	7	1	3	1.5	3

El Escenario 4 se prueba con

```
(%i20) a2:0;
(%o20)      0
(%i21) a3:[10,20,30,40,50];
(%o21)     [10, 20, 30, 40, 50]
(%i22) a4:Araceli;
(%o22)      Araceli
```

Resultado de los datos de prueba

Escenario 1.
Resultado de *los datos de prueba 1.*

```
(%i23) submat(1,3,a1,2,4);
           [ 20  30  40 ]
           [            ]
(%o23)     [ 21  31  41 ]
           [            ]
           [ 22  32  42 ]
```

Escenario 2.
Resultado de *los datos de prueba 2.*

```
(%i24) submat(3,3,a1,4,4);
(%o24)      [ 42 ]
```

Escenario 3.
Resultado de *los datos de prueba 3-5.*

```
(%i25) submat(0,2,a1,0,3);
Arguments 0 2 0 3 must be integers greater than 0.
#0: submat(_ii=0,_f=2,_a=matrix([10,20,30,40,50],[11,21,31,41,51],
    [12,22,32,42,52],[13,23,33,43,53]),_ji=0,_jf=3)(submat.mc line 10)
 -- an error.  Quitting.  To debug this try debugmode(true);

(%i26) submat(1,0,a1,1,0);
Arguments 1 0 1 0 must be integers greater than 0.
#0: submat(_ii=1,_f=0,_a=matrix([10,20,30,40,50],[11,21,31,41,51],
    [12,22,32,42,52],[13,23,33,43,53]),_ji=1,_jf=0)(submat.mc line 10)
 -- an error.  Quitting.  To debug this try debugmode(true);

(%i27) submat(1,-2,2,4);
Too few arguments supplied to submat(_ii, _f, _a, _ji, _jf):
[1, - 2, 2, 4]
 -- an error.  Quitting.  To debug this try debugmode(true);
```

Escenario 4.
Resultado de *los datos de prueba a2, a3 y a4* especificados en las entradas (%i20), (%i21) y (%i22), respectivamente.

```
(%i28) submat(1,3,a2,1.5,3);
Argument 0 must be a matrix
#0: submat(_ii=1,_f=3,_a=0,_ji=1.5,_jf=3)(submat.mc line 6)
 -- an error.  Quitting.  To debug this try debugmode(true);

(%i29) submat(1,2,a3,2,4);
Argument [10, 20, 30, 40, 50] must be a matrix
#0: submat(_ii=1,_f=2,_a=[10,20,30,40,50],_ji=2,_jf=4)(submat.mc line 6)
 -- an error.  Quitting.  To debug this try debugmode(true);
```

```
(%i30) submat(1,2,a4,2,4);
Argument Araceli must be a matrix
#0: submat(_ii=1,_f=2,_a=Araceli,_ji=2,_jf=4)(submat.mc line 6)
 -- an error.  Quitting.  To debug this try debugmode(true);
```

Escenario 5.
Resultado de *los datos de prueba 6-7.*

```
(%i31) submat(3,2,a1,1,4);
Wrong arguments. 3 and 1 must be smaller than 2 and 4 respectively.
#0: submat(_ii=3,_f=2,_a=matrix([10,20,30,40,50],[11,21,31,41,51],
    [12,22,32,42,52],[13,23,33,43,53]),_ji=1,_jf=4)(submat.mc line 12)
 -- an error.  Quitting.  To debug this try debugmode(true);

(%i32) submat(2,4,a1,3,1);
Wrong arguments. 2 and 3 must be smaller than 4 and 1 respectively.
#0: submat(_ii=2,_f=4,_a=matrix([10,20,30,40,50],[11,21,31,41,51],
    [12,22,32,42,52],[13,23,33,43,53]),_ji=3,_jf=1)(submat.mc line 12)
 -- an error.  Quitting.  To debug this try debugmode(true);
```

Escenario 6.
Resultado de *los datos de prueba 8-9.*

```
(%i33) submat(1,7,a1,2,3);
Wrong arguments. 7 must be equal or smaller than the rows of
     errexp1 and 3 must be equal or smaller than its columns.
#0: submat(_ii=1,_f=7,_a=matrix([10,20,30,40,50],[11,21,31,41,51],
    [12,22,32,42,52],[13,23,33,43,53]),_ji=2,_jf=3)(submat.mc line 11)
 -- an error.  Quitting.  To debug this try debugmode(true);

(%i34) submat(1,4,a1,2,8);
Wrong arguments. 4 must be equal or smaller than the rows of
     errexp1 and 8 must be equal or smaller than its columns.
#0: submat(_ii=1,_f=4,_a=matrix([10,20,30,40,50],[11,21,31,41,51],
    [12,22,32,42,52],[13,23,33,43,53]),_ji=2,_jf=8)(submat.mc line 11)
 -- an error.  Quitting.  To debug this try debugmode(true);

(%i35) errexp1;
        [ 10  20  30  40  50 ]
        [                    ]
        [ 11  21  31  41  51 ]
(%o35)  [                    ]
        [ 12  22  32  42  52 ]
        [                    ]
        [ 13  23  33  43  53 ]
```

Escenario 7
Resultado de los datos de prueba 10-11.

```
(%i36) submat(1,mm,a1,1,4);
Arguments 1 mm 1 4 must be integers greater than 0.
#0: submat(_ii=1,_f=mm,_a=matrix([10,20,30,40,50],[11,21,31,41,51],
[12,22,32,42,52],[13,23,33,43,53]),_ji=1,_jf=4)(submat.mc line 9)
 -- an error.  Quitting.  To debug this try debugmode(true);

(%i37) submat(1,3,a1,1.5,3);
Arguments 1 3 1.5 3 must be integers greater than 0.
#0: submat(_ii=1,_f=3,_a=matrix([10,20,30,40,50],[11,21,31,41,51],
[12,22,32,42,52],[13,23,33,43,53]),_ji=1.5,_jf=3)(submat.mc line 9)
 -- an error.  Quitting.  To debug this try debugmode(true);
```

Tiempo de ejecución

Para los datos de prueba válidos de la Tabla XII se tiene:

```
(%i38) timer(submat);
(%o38)     [submat]

(%i39) submat(1,3,a1,2,4)$

(%i40) get(submat,'runtime);
(%o40)       0

(%i41)submat(3,3,a1,4,4)$

(%i42) get(submat,'runtime);
(%o42)       0
```

B.1.3 Prueba "Calcular el máximo retardo en tiempo"

Esta prueba tiene la finalidad de verificar el módulo *maxd*, el cual permite determinar el retardo máximo en tiempo ya sea de un polinomio o de los elementos de una matriz.

Sintaxis: `maxd(f)` FUNCIÓN

Datos de entrada:

`f` ⇒ polinomio o matriz.
Variable a la que se determina el máximo retardo en tiempo.

Dependencias: `ninguna`.

Objetivos

Los objetivos de esta prueba son:

1. Verificar si se determina el retardo máximo correcto del argumento de entrada, ya sea un polinomio o una matriz.
2. Verificar si al ingresar datos inválidos se genera un mensaje de error.

Requisitos y restricciones

El mínimo retardo que detecta la función es 0, así que si el argumento de entrada contiene sólo términos no causales, la respuesta será 0.

Descripción de la prueba

Esta prueba tiene los siguientes escenarios:

- Escenario 1: El usuario ingresa una 1-forma f.
- Escenario 2: El usuario ingresa una matriz f.

Datos de prueba

Se crea un archivo que genera polinomios automáticamente y se forma una matriz con los mismos polinomios. El archivo de prueba es el siguiente:

```
/*<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< S A C   P R O J E C T >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>*/
/*      TEST:    Creates 15 random polynomials and uses them to form a  matrix                  */
/*=====================================================================================================*/
disp("Polinomios                          máximo retardo en tiempo")$
for i:1 thru 15 do (
   b:random(4),
   sum:0,
   for j:1 thru b do (
      nf:random(3)+1,
      for k:1 thru nf do (
         tt:1,
         a:map(random,[10,10,2]),
         tt:tt*(x[a[1]](t-a[2]+2*a[3]))^a[3]
        ),
      sum:sum+(random(100-49)*tt)*_D^(random(6))
      ),
  f[i]:sum
  )$

k:1,
mat:zeromatrix(3,5);
for i:1 thru 3 do(
for j:1 thru 5 do(
mat[i,j]:f[k],k:k+1));
/*=====================================================================================================*/
/*==================================Applying the test to the maxd function============================*/
disp("Matriz")$
for i:1 thru 15 do (
     M[i]:maxd(f[i]),
     print(i,"    ",f[i],"   ",M[i])
       )$

disp(mat)$
disp("Máximo retardo en tiempo de la matriz anterior")$
delay:maxd(mat)$
disp(delay)$
/*=====================================================================================================*/
/*Last modification date: october 5th 2006*/
```

Resultados esperados

- Que el máximo retardo encontrado en los polinomios sea igual al detectado en la matriz.

Resultado de los datos de prueba

Los resultados obtenidos utilizando la rutina *testmaxd* son los siguientes. En la primera columna se muestra el polinomio y enseguida el máximo retardo en tiempo detectado. Se generan 15 polinomios aleatorios y posteriormente se forma una matriz con los mismos, el máximo retardo detectado se indica debajo de esta matriz.

```
(%i43) load("./tests/testmaxd.mc")$
        Polinomios                        máximo retardo en tiempo
                 5        3
1      7 x (t) _D  + 18 _D                              0
         7
           4
2      50 _D                                            0
            5
3      11 _D                                            0
4      0                                                0
                    4
5      2 x (t - 5) _D                                   5
          0
6      0                                                0
            4        2
7      24 _D  + 36 _D                                   0
                     5
8      40 x (t + 2) _D  + 23 x (t - 6) + 30             6
           1                   1
                     2
9      14 x (t + 1) _D                                  0
           4
             3                   2
10      35 _D  + 38 x (t - 1) _D  + 40 x (t - 6)        6
                    0                   3
11      0                                               0
             3
12      20 _D                                           0
13      0                                               0
14      0                                               0
             5                   3
15      48 _D  + 23 x (t - 7) _D  + 4 x (t - 5) _D      7
                    9                 6


                                  Matriz
[       5    3          4                 5                                                4         ]
[ 7x(t)_D+18_D       50 _D             11 _D                     0              2 x (t - 5) _D         ]
[   7                                                                            0                   ]
[                                                                                                    ]
[                  4      2             5                                2        3           2         ]
[      0       24 _D + 36 _D    40x(t + 2)_D+23 x(t - 6)+30   14 x(t + 1)_D    35 _D+38x(t - 1)_D+40x(t - 6)]
[                                  1            1               4                    0           3     ]
[                                                                                                    ]
[                      3                                                  5                        3]
[      0           20 _D                0                      0        48_D+23x(t - 7)_D+4x(t - 5)_D ]
[                                                                                 9          6         ]

                              Máximo retardo en tiempo de la matriz anterior

                                                7
```

Tiempo de ejecución

La función *get*() se utiliza para obtener cualquier dato de la función *timer_info*(). En este caso se muestra la información de las llamadas promedio de la función maxd al ejecutar la rutina de prueba. La matriz devuelta por *timer_info* incluye los nombres de las funciones, tiempo de ejecución en cada llamada, número de veces que ha sido llamada, tiempo total de ejecución y tiempo consumido en la recolección de basura, gctime (del inglés, "garbage collection time") en la versión original de Macsyma, aunque en Máxima, por lo regular, toma el valor constante cero.

```
(%i44) timer(maxd);
(%o44)    [maxd]
```

```
(%i45) load("./Tests/testmaxd.mc")$
(%i46) timer_info()
       [ function         time//call           calls  runtime   gctime ]
       [                                                               ]
(%o46) [  maxd       4.385964912280701E-5 sec   228   0.01 sec    0    ]
       [                                                               ]
       [  total      4.385964912280701E-5 sec   228   0.01 sec    0    ]
```

El número de llamadas es de 228 debido a que se llama a sí misma varias veces durante su ejecución.