UNIVERSIDAD AUTÓNOMA CHAPINGO

POSGRADO EN INGENIERÍA AGRÍCOLA Y USO INTEGRAL DEL AGUA

SISTEMA COMPUTACIONAL PARA EL ANÁLISIS DE SENSIBILIDAD LOCAL AUTOMÁTICO DE MODELOS DINÁMICOS

TESIS

QUE COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE: DOCTOR EN INGENIERÍA AGRÍCOLA Y USO INTEGRAL DEL AGUA

PRESENTA:

FELIPE JOSÉ ANTONIO PEDRAZA OROPEZA

CHAPINGO, ESTADO DE MÉXICO
FEBRERO DE 2019

SISTEMA COMPUTACIONAL PARA EL ANÁLISIS DE SENSIBILIDAD LOCAL AUTOMATICO DE MODELOS DINÁMICOS

Tesis realizada por Felipe José Antonio Pedraza Oropeza bajo la dirección del Comité Asesor indicado, aprobada por el mismo y aceptada como requisito parcial para obtener el grado de:

DOCTOR EN INGENIERÍA AGRÍCOLA Y USO INTEGRAL DEL AGUA

DIRECTOR: _____
DR. IRINEO LORENZO LÓPEZ CRUZ

ASESOR: _____
DR. OSCAR LUIS PALACIOS VÉLEZ

ASESOR: _____
DR. AGUSTÍN RUIZ GARCÍA

LECTOR EXTERNO: _____
DR.MARTIN A. BOLAÑOS GONZALES

AGRADECIMIENTOS

A mi hija Rosa Samantha Pedraza Eriza, el mejor regalo que me ha dado la vida.

A mi esposa Ma Elba Eriza Salmerón con todo mi amor, respeto y admiración.

A la memoria de mi madre Rosa Oropeza Vargas.

A la memoria de mis profesores de vida Samuel Trueba Coronel, Donaciano Ojeda Ortega, Jesús Muñoz Vázquez y José Luis Oropeza Mota.

A mis compañeros y guías en la fascinante travesía del doctorado Irineo López Cruz, Oscar Palacios Vélez, Agustín Ruiz García, Martín Bolaños Gonzales y Enrique Palacios Vélez.

A mis compañeros de vida que para mí fortuna *esta dedicatoria es demasiado pequeña para contenerlos a todos*, y entre los que se encuentran:

Lourdes Sánchez Arellano
Gabriela Garay Arreola
Mercedes Peralta Espinosa
Guadalupe Mora Vital
María de Jesús Velasco Salas
Jorge Valdés Carrasco
Luis Piñón Sosa
Enrique Mejía Saenz
Ramón Lobato Silva
José Reyes Sánchez
Juan Ávila Alcívar
José Luis Chávez Zárate
Antonio Rosales Cervantes
Felipe Zapata Pérez
Raúl Medrano Gordillo
Armando Méndez Muñoz

DATOS BIOGRÁFICOS

El autor es originario de la localidad de San Juan Epatlán, cabecera del municipio de Epatlán en el estado de Puebla.

Cursó la licenciatura de ingeniero agrónomo especialista en zonas áridas en la Unidad Regional Universitaria de Zonas Áridas de la Universidad Autónoma Chapingo, formando parte de la primera generación de dicha Unidad Regional.

Obtuvo la maestría en Hidrociencias en el campus Montecillo del Colegio de Postgraduados.

Cuenta con el grado de Master of Science in Spatial Information Science and Engineering por la Universidad de Maine EE. UU.

Actualmente colabora como Investigador Titular en el posgrado de Hidrociencias del colegio de Postgraduados.

Sus áreas de interés son: Matemáticas; Estadística; Programación de computadoras; Percepción Remota; Sistemas de Información Geográfica y Modelación, Simulación y Control de Biosistemas.

AUTOMATIZACIÓN DEL ANÁLISIS DE SENSIBILIDAD LOCAL DE MODELOS DINÁMICOS

Autor: Felipe José Antonio Pedraza Oropeza
Director: Irineo Lorenzo López Cruz

RESUMEN

El análisis de sensibilidad es una valiosa herramienta que nos permite cuantificar el efecto de los parámetros de un modelo dinámico sobre las variables de estado del mismo. Conocer la importancia de los parámetros permite poner más atención a los más importantes y descuidar e incluso ignorar los menos importantes.

El análisis de sensibilidad local se basa en la obtención y la integración numérica de las llamadas ecuaciones de sensibilidad, las cuales son un conjunto de ecuaciones diferenciales ordinarias (ODEs) adicionales a las ecuaciones diferenciales del modelo dinámico. Tanto la postulación de las ecuaciones de sensibilidad como su solución es un gran reto matemático ya que en general si el modelo dinámico consta de n estados y q parámetros, entonces se tienen que resolver un total de $n + (n \times q)$ ODEs.

Llevar a cabo este proceso manualmente conduce fácilmente a errores, por lo que en el presente trabajo se desarrollaron dos herramientas computacionales, que generan de manera automática el código requerido para llevar a cabo el análisis de sensibilidad local, la primera de ellas en Matlab/Simulink y la otra en Matlab/Simulink/Lenguaje C.

AUTOMATION OF THE LOCAL SENSITIVITY ANALYSIS OF DYNAMIC MODELS

Author: Felipe José Antonio Pedraza Oropeza
Advisor: Irineo Lorenzo López Cruz

ABSTRACT

Sensitivity analysis is a valuable tool that allows to quantify the effect of the parameters of a dynamic model on its state variables. Knowing the importance of the parameters allows you to pay more attention to the most important ones and neglect and even ignore the least important.

The local sensitivity analysis is based on the obtaining and the numerical integration of the so-called sensitivity equations, which are a set of ordinary differential equations (ODEs) in addition to the differential equations of the dynamic model. Both the postulation of the sensitivity equations and their solution is a great mathematical challenge since in general if the dynamic model consists of n states and q parameters, then a total of $n + (n \times q)$ ODEs must be solved.

Carrying out this process manually leads easily to errors, so in the present work two computational tools were developed, which automatically generate the code required to carry out the local sensitivity analysis, the first one in Matlab/Simulink and the other in Matlab/Simulink/C Language.

Contenido

1. INTRODUCCIÓN.

Estamos viviendo una nueva era al borde de una economía basada en datos. Una de las palabras de moda de nuestro tiempo es '*big data*'. El 18 de febrero de 2015, el presidente de EE. UU., Barack Obama, nominó a D.J. Patil como el primer jefe oficial de datos de la Casa Blanca. La toma de decisiones es cada vez más cuantitativa en una variedad de campos. Nos encontramos constantemente hablando de análisis de negocios y del uso de información numérica para tomar decisiones (Borgonovo, 2017).

El primer paso en el proceso moderno de toma de decisiones es identificar el problema, el segundo paso es identificar las alternativas disponibles, el tercer paso es implementar un modelo de ayuda a la decisión y el cuarto paso es usar el modelo para evaluar las alternativas (Borgonovo, 2017).

Un sistema es una parte limitada de la realidad que contiene elementos interrelacionados, un modelo es una representación simplificada de un sistema y la simulación puede definirse como el arte de construir modelos matemáticos y el estudio de sus propiedades en referencia a las de los sistemas (de Wit, 1982).

Un modelo es una representación simple de un fenómeno complejo. Es una abstracción, y por lo tanto no contiene todas las características del sistema real. Sin embargo, un modelo comprende todas las características esenciales para el problema a resolver o describir (Soetaert and Herman, 2009).

Los modelos matemáticos son ampliamente utilizados en diversas disciplinas, observándose en los últimos años un crecimiento en su número de variables y parámetros, dado que el refinamiento de la percepción física conlleva a que los modelos se vuelvan más sofisticados, aunado a que la capacidad de las computadoras crece constantemente lo que permite manejar modelos complejos con mayor facilidad. En estos modelos complejos no queda claro

cuáles son los parámetros más importantes, lo cual hace necesario un análisis de sensibilidad (Saltelli *et al*, 2000).

El modelado matemático es uno de los métodos más importantes y poderosos para estudiar los fenómenos que ocurren en nuestro universo. En términos generales, los modelos utilizados se componen de una o varias ecuaciones a partir de las cuales se determinan una o varias incógnitas de interés, en términos de otras cantidades conocidas. Las incógnitas son en muchos casos funciones de un conjunto de variables de entrada y parámetros. Dado que con frecuencia las leyes físicas o empíricas que rigen los procesos evolutivos implican las tasas de cambio de estas funciones con respecto a sus variables, y dado que las tasas de cambio se expresan matemáticamente mediante derivadas, es importante saber resolver ecuaciones que contienen las funciones desconocidas y sus derivadas, denominadas ecuaciones diferenciales (Constanda, 2017).

El análisis de sensibilidad es el estudio de cómo la variación en las salidas producidas por un modelo (numéricas o de otro tipo) puede ser distribuida, cualitativa o cuantitativamente, a diferentes fuentes de variación, y de cómo el modelo dado depende de la información alimentada (Saltelli *et al*., 2000).

El análisis de sensibilidad es útil en varias etapas del modelado, la sensibilidad a los parámetros del modelo se utiliza para detectar a qué parámetros es más sensible el resultado del modelo. Una vez encontrados, estos parámetros son candidatos para calibración o experimentación adicional (Van Straten, 2012).

Existen dos tipos de análisis de sensibilidad, el análisis de sensibilidad global el cual, como su nombre lo indica, explora los factores de entrada en espacios, o rangos, mediante el cálculo de radios incrementales en diferentes puntos en el espacio de los factores de entrada, mientras que el análisis de sensibilidad local calcula derivadas parciales en puntos específicos de los factores de entrada, denominados puntos de operación, dado que la derivada parcial

$\sigma Y_j / \sigma X_i$, de una salida Y_j contra una entrada X_i puede considerarse como la definición matemática de la sensibilidad de Y_j contra X_i (Saltelli *et al*., 2008).

Los modelos se pueden clasificar de diferentes maneras como determinísticos y estocásticos, distribuidos y no distribuidos, estáticos y dinámicos, continuos y discretos, lineales y no lineales, invariantes en el tiempo y variables en el tiempo, empíricos y mecanicistas (Tangirala, 2014).

La metodología desarrollada en el presente trabajo funciona para modelos dinámicos, continuos, lineales, invariantes en el tiempo y definidos en espacio de estados, lo cual podemos resumir de la siguiente manera:

1) **Modelo dinámico**. La categorización de los modelos en estáticos y dinámicos se basa en la respuesta. Los modelos estáticos, también conocidos como modelos de estado estacionario, relacionan cantidades instantáneas o, en el mejor de los casos, describen los efectos de las entradas retardadas. Los modelos de estado estacionario no tienen una "memoria" asociada a ellos. Los modelos dinámicos son una clase más general de modelos que describen el comportamiento transitorio de un proceso (Tangirala, 2014).
2) **Modelo continuo**. Según el tipo de datos disponibles y la descripción del proceso, las dos categorías principales de modelos matemáticos son modelos continuos y discretos. Los modelos continuos operan con variables continuas, mientras que los modelos discretos operan con variables discretas. Más específicamente, un modelo discreto implica un número finito de las variables escalares desconocidas, mientras que un modelo continuo utiliza una variables continuas (escalares o vectoriales) definida en algún dominio D, que por lo general es de los números reales de dimensión n (Hritonenko y Yatsenko, 2013).
3) **Modelo linea**l. Un sistema se considera lineal si cumple con la propiedad de superposición, o principio de superposición, que establece que la respuesta neta causada por dos o más estímulos es la suma de las respuestas que habría sido causada por cada estímulo

individualmente. De modo que si la entrada A produce la respuesta X y la entrada B produce la respuesta Y, entonces la entrada (A + B) produce la respuesta (X + Y). Es importante señalar que la propiedad de superposición es la combinación de la propiedad de aditividad y la propiedad de homogeneidad (Chen, 2013).

4) **Modelo invariante en el tiempo**. Se dice que un sistema es invariante en el tiempo si sus características no cambian con el tiempo. Un cohete no es un sistema invariante en el tiempo, porque su masa disminuye rápidamente. Si bien todos los sistemas físicos se deterioran con el paso del tiempo, un gran número de ellos se pueden modelar como invariantes en el tiempo durante un período de tiempo limitado (Chen, 2013).
5) **Espacio de estados.** Los sistemas dinámicos pueden definirse mediante una convolución, una función de transferencia, una ecuación diferencial de alto orden y en espacio de estados. La función de transferencia utiliza la transformada de Laplace por lo que es una descripción del dominio de transformación, mientras que las otras tres son función del tiempo por lo que son descripciones de dominio de tiempo. Las dos representaciones más utilizadas son la función de transferencia y el espacio de estados, pero dado que la primera es una descripción externa, o de caja negra, el espacio de estados es una descripción interna porque también describe las variables internas del sistema lo cual le da una gran ventaja (Chen, 2013).
6) **Modelo lineal.** Cada sistema LTI con entrada $\boldsymbol{u}(t)$ y salida y $\boldsymbol{y}(t)$ se puede describir mediante un conjunto de ecuaciones de forma:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\,\boldsymbol{x}(t) + \boldsymbol{B}\,\boldsymbol{u}(t) \qquad (1)$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\,\boldsymbol{x}(t) + \boldsymbol{D}\,\boldsymbol{u}(t) \qquad (2)$$

En donde $\dot{\boldsymbol{x}}(t) = d\boldsymbol{x}(t)/dt$. Si el sistema tiene n variables de estado, entonces $\boldsymbol{x}$ es un vector de dimensión $n \times 1$. Para que las matrices en las ecuaciones (1) y (2) sean compatibles, $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ y $\boldsymbol{D}$ deben ser de dimensiones $n \times n$, $n \times 1$, $1 \times n$ y 1×1. Todas las entradas de las

cuatro matrices son números reales, independientes del tiempo. La ecuación (1), llamada ecuación de estado, en realidad consiste en un conjunto de n ecuaciones diferenciales de primer orden. La ecuación (2), llamada ecuación de salida, es una ecuación algebraica. La constante D se denomina parte de transmisión directa. El conjunto de estas dos ecuaciones se denomina ecuación de espacio de estados n-dimensional (Chen, 2013). Según (Chen, 2013) para un sistema con p inputs, q outputs y n variables de estado: A es una matriz de $n \times n$, B es una matriz de $n \times p$, C es un matriz $q \times n$ y D es una matriz $q \times p$.

7) **Modelo no lineal.** Un modelo dinámico no lineal en espacio de estados se define mediante un sistema de ecuaciones diferenciales (ecuación de estado) y un sistema de ecuaciones algebraicas (ecuación de salida), que matricialmente se expresan como:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{\theta}), \;\; \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \qquad (3)$$

$$\boldsymbol{y}(t) = \boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{\theta}) \qquad (4)$$

En donde:

$\boldsymbol{x}(t)$ = Vector de variables de estado

$\boldsymbol{y}(t)$ = Vector de variables de salida

$\boldsymbol{u}(t)$ = Vector de variables de entrada

$\boldsymbol{\theta}$ = Vector de parámetros del modelo

$\boldsymbol{x}_0$ = Vector de valores iniciales

8) **Matriz de sensibilidad local de primer orden**. Se define como:

$$\boldsymbol{S} = \frac{\partial \boldsymbol{x}(t)}{\partial \boldsymbol{\theta}} \qquad (5)$$

La dificultad para su cálculo se debe a que por lo general no conocemos $\boldsymbol{x}(t)$ sino $\dot{\boldsymbol{x}}(t)$, por lo que a menos que el modelo tenga solución analítica, la tenemos que calcular mediante el método directo (Turányi y Tomlin, 2014) basado en la solución del sistema de ecuaciones diferenciales siguiente:

$$\dot{\boldsymbol{S}} = \boldsymbol{A}\,\boldsymbol{S} + \boldsymbol{M}, \;\; \boldsymbol{S}(0) = 0 \qquad (6)$$

En donde:

$\boldsymbol{A} = \partial \boldsymbol{f}/\partial \boldsymbol{x}$ = Matriz Jacobiana de dimensión $n \times n$

$\boldsymbol{M} = \partial \boldsymbol{f}/\partial \boldsymbol{\theta}$ = Matriz Jacobiana paramétrica de dimensión $n \times m$

$\boldsymbol{S} = \partial \boldsymbol{x}/\partial \boldsymbol{\theta}$ = Matriz de sensibilidad local de dimensión $n \times m$

Por lo que el cálculo de estas matrices para un modelo de n estados y m parámetros requiere de $n \times n + n \times m$ derivadas. Por ejemplo el modelo NICOLET con 5 variables de estado y 18 parámetros requiere de 5×5+5×18=115 derivadas.

2. REVISIÓN DE LITERATURA.

El análisis de sensibilidad es hoy un elemento crucial en el desarrollo de modelos y en cualquier análisis de decisión práctico, y puede desempeñar cualquiera de varios roles en el proceso de análisis de decisión (Felli y Hazen, 2003).

El término análisis de sensibilidad define una colección de métodos matemáticos que se pueden utilizar para explorar las relaciones entre los valores de los parámetros de entrada de un modelo matemático y sus soluciones (Turányi y Tomlin, 2014).

Las etapas en el proceso moderno de toma de decisiones son la identificación del problema, la identificación de las alternativas de solución, la implementación del modelo, la evaluación de las alternativas de solución, el análisis de sensibilidad y la implementación de la mejor alternativa resultante del proceso iterativo anterior (Borgonovo, 2017), como se ilustra en la Figura 1.

Existen muchos métodos de análisis de sensibilidad, y su idoneidad depende de: (1) el modelo utilizado y su complejidad, (2) los recursos disponibles para llevar a cabo el análisis de sensibilidad y (3) la pregunta científica que se desea contestar. Los métodos van desde simplemente cambiar un parámetro del modelo hasta cambiar simultáneamente muchos parámetros del modelo y evaluar el cambio en los resultados del modelo (Petropoulos y Srivastava, 2017).

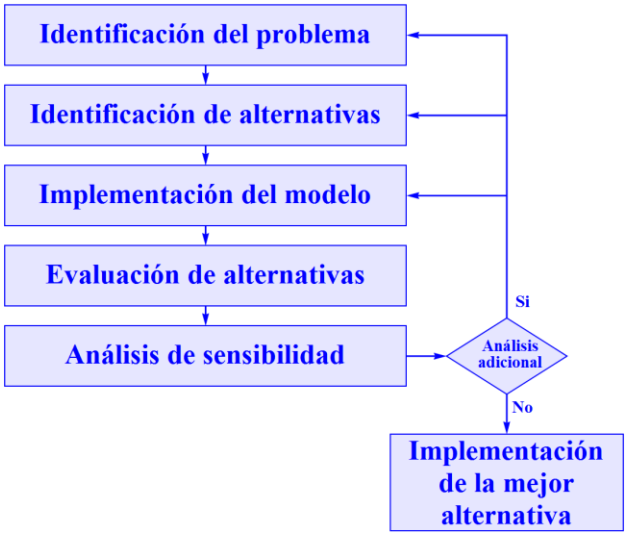

Figura 1. Etapas en el proceso moderno de toma de decisiones (Borgonovo, 2017)

Los modeladores y filósofos de la ciencia han debatido largamente el tema de la indeterminación de los modelos. La mayoría de los modeladores de hoy en día probablemente estarían de acuerdo en que un modelo no puede ser validado, en el sentido de "ser probado como verdadero". Es más defendible y correcto decir que un modelo ha sido ampliamente corroborado, lo que significa que el modelo ha sobrevivido a una serie de pruebas, ya sean formales, de consistencia interna o relativas a la capacidad del modelo para explicar o predecir el 'mundo' de una manera convincente y parsimoniosa, lo cual se conoce como evaluación del modelo (Saltelli *et al*, 2008).

Hasta hace muy poco tiempo, el análisis de sensibilidad se concibió y a menudo se definió como una medida local del efecto de un insumo dado en un producto dado. Un ejemplo típico es la determinación de las constantes cinéticas o potenciales mecánicos cuánticos a partir de la tasa de rendimiento de un proceso químico. De acuerdo a Saltelli la mayoría de los análisis de sensibilidad que se pueden encontrar en las revistas de ciencias físicas son análisis de sensibilidad local (Saltelli *et al*, 2004).

La combinación de datos computacionales y experimentales de diferentes fuentes implica una propagación ponderada de las incertidumbres de los

parámetros, utilizando las derivadas funcionales locales de las respuestas del sistema con respecto a los parámetros del modelo de entrada. Estas derivadas de respuesta se llaman habitualmente sensibilidades de respuesta o simplemente sensibilidades. En un nivel fundamental, la necesidad de calcular tales sensibilidades proviene de la imposibilidad de medir o calcular cualquier cantidad física con precisión absoluta, como se ha discutido anteriormente (Cacuci, 2018).

El análisis de sensibilidad puede proporcionar la identificación posterior de los parámetros más importantes que impulsan la incertidumbre del modelo. Estos métodos pueden formar una parte clave del proceso de evaluación y mejora del modelo (Turányi y Tomlin, 2014).

La matriz de sensibilidad local muestra el efecto de un cambio unitario de los valores de los parámetros en los resultados del modelo. Esto puede proporcionar información útil sobre la influencia relativa de los parámetros cerca de sus valores nominales y también puede ser útil para estimar cómo la incertidumbre en estos valores de parámetros puede propagarse a la incertidumbre predictiva en los resultados del modelo (Turányi y Tomlin, 2014).

El análisis de incertidumbre estima el intervalo de confianza en el subconjunto de parámetros identificables y también en las predicciones del modelo (Ferrari *et al*, 2015).

El análisis de sensibilidad es de particular importancia tanto desde el punto de vista cualitativo como numérico. Esto último implica la justificación de una solución numérica exitosa al tratar las perturbaciones como errores que ocurren típicamente en los cálculos, y también como una herramienta para determinar una tasa de convergencia de algoritmos de solución (Mordukhovich, 2013).

El objetivo del análisis de sensibilidad es determinar qué tan sensible es la salida de un modelo con respecto a los elementos del modelo que están sujetos a incertidumbre. Para los modelos dinámicos, el análisis de

sensibilidad está estrechamente relacionado con el estudio de la propagación de errores. Generalmente se distinguen dos tipos de análisis de sensibilidad, el análisis de sensibilidad local y el análisis de sensibilidad global. El análisis de sensibilidad local se centra en el impacto local de factores de incertidumbre en los resultados del modelo y se lleva a cabo calculando derivadas parciales de las variables de salida con respecto a los factores de entrada. El análisis de sensibilidad global considera el dominio completo de la incertidumbre de los factores del modelo incierto. En el análisis de sensibilidad global, los factores de incertidumbre pueden variar dentro de su rango completo de variación (Wallach *et al*, 2014).

El análisis de sensibilidad es particularmente útil para verificar la funcionalidad del modelo y para validar simulaciones de salida en casos donde tales simulaciones no pueden ser probadas contra un conjunto robusto de datos empíricos y también puede revelar dónde se introduce la incertidumbre en un modelo, y qué insumos deben recibir especial atención en el diseño y la ejecución de modelos (Brouwer-Burg *et al*, 2016).

El análisis de sensibilidad es un enfoque bien establecido para investigar cuánto depende una variable dada dentro de un modelo de las cantidades dentro de ese modelo. El análisis de sensibilidad de un modelo de sistema complejo puede ayudar a identificar posibles áreas problemáticas que pueden dar lugar a altas dependencias de productos en elementos específicos del subsistema. Esto es de fundamental importancia para los procedimientos de identificación del sistema destinados a proporcionar estimaciones de parámetros fiables en la etapa de calibración del modelo y también para la validación y actualización de modelos (Murray-Smith, 2015).

Existe un vínculo directo entre el análisis de sensibilidad y el análisis de incertidumbre directa. En efecto, ambos son tipos de exploración del espacio modelo basados solo en las suposiciones previas sobre la naturaleza de ese espacio hechas por el modelador (Beven, 2009).

Una aplicación importante del análisis de sensibilidad es la reducción del modelo, es decir, la búsqueda de modelos más simples que se ajustan a los datos. La reducción de modelos basada en la teoría de la sensibilidad es un enfoque particularmente útil para modelos de biología de sistemas moleculares complejos de vías de transducción de señales celulares a gran escala y otras redes bioquímicas (Di Stefano, 2013).

La eficacia de la calibración de parámetros se mejora si se concentrar el esfuerzo en aquellos parámetros a los que los resultados de la simulación del modelo son más sensibles. Esto requiere un enfoque para evaluar la sensibilidad del parámetro dentro de una estructura de modelo complejo. La sensibilidad se evalúa con respecto a alguna medida de rendimiento, y pueden pensarse en términos de su superficie de respuesta en el espacio de los parámetros. Una definición de la sensibilidad de los resultados de la simulación del modelo a un parámetro particular es el gradiente local de la superficie de respuesta en la dirección del eje del parámetro elegido (Beven, 2012).

En un análisis de sensibilidad local, se estima el efecto de un cambio en los parámetros en una región infinitesimalmente pequeña. Esto se puede hacer por medio de una función de sensibilidad. Estas funciones representan la sensibilidad de las variables del modelo a los parámetros individuales, ponderados para la escala de las variables. Luego se puede calcular la sensibilidad media de todas las variables del modelo con respecto a cada parámetro y esta información se usa para clasificar los parámetros del modelo de acuerdo con el efecto que tienen en la salida del modelo. Por lo tanto, este tipo de análisis se usa para determinar los parámetros más influyentes de un modelo. No tiene sentido invertir grandes cantidades de tiempo tratando de encontrar buenos valores para parámetros que solo tienen un efecto pequeño en el resultado del modelo (Soetaert y Herman, 2009).

El análisis de sensibilidad es un método importante en la biología de sistemas. La determinación de las sensibilidades es una herramienta importante en la clasificación de la importancia de los parámetros cinéticos y también en la

estimación de parámetros. Un método bien conocido para el análisis de las sensibilidades es la teoría del control metabólico. Define los coeficientes que conectan las características locales de los pasos de reacción individuales con las características globales del módulo como el flujo de estado estable a través de la red (Kremling, 2014).

Una famosa cita sobre el análisis de sensibilidad es de Fuerbringer, quien escribe: "Análisis de sensibilidad para modeladores: ¿irías a un ortopedista que no usa rayos X?" (Saltelli, 2000).

Es importante señalar que la ecuación (6) es una complicación necesaria para el cálculo de las funciones de sensibilidad local. Decimos que es una complicación dado que si un modelo tiene n variables de estado y m parámetros, las funciones de sensibilidad local se pueden calcular derivando cada una de las n variables de estado entre cada uno de los m parámetros, o sea mediante $n \times m$ derivadas, mientras que la ecuación (6) requiere de la solución de un sistema de $n \times m$ ecuaciones diferenciales simultaneas, también llamadas acopladas, con $n \times m$ incógnitas (las ecuaciones de sensibilidad local).

Pasar del cálculo de $n \times m$ derivadas a la solución de un sistema de $n \times m$ ecuaciones diferenciales simultáneas con $n \times m$ incógnitas es una verdadera complicación, pero es necesaria ya que como dijimos anteriormente, por lo general no conocemos las funciones de las variables de estado sino sus primeras derivadas (las ecuaciones de estado).

Si un modelo tiene solución analítica la ecuación (6) es innecesaria ya que calculamos las funciones de sensibilidad local en dos pasos simples, primero resolvemos las ecuaciones de estado, o sea un sistema de n ecuaciones diferenciales simultaneas con n incógnitas (las variables de estado) y una vez que tenemos las funciones analíticas de las variables de estado, las derivamos con respecto a cada parámetro, que es lo que se hace en el apéndice 2.

Si no contamos con la solución analítica de un modelo dinámico, el análisis de sensibilidad local puede hacerse por otros procedimientos como el método de fuerza bruta, el método de la función de Green o el método directo (Turányi y Tomlin, 2014).

El método directo de solución del análisis de sensibilidad local se basa en la solución de la ecuación (6), para lo cual se requiere de los siguientes procedimientos:

a) Resolver la ecuación de estado del sistema (3), dado que no se cuenta con su solución analítica, esto debe de hacerse mediante un algoritmo de solución de las ecuaciones diferenciales ordinarias acorde al tipo de modelo, si el modelo es rígido en Matlab se puede utilizar ode23s o ode45 en caso contrario.
b) Calcular la matriz Jacobiana (A), ya sea mediante diferenciación analítica, usando el Symbolic Math Toolbox, diferenciación numérica mediante diferencias finitas como la fórmula de diferencia media.
c) Calcular la matriz Jacobiana paramétrica (M), ya sea mediante diferenciación analítica, usando el Symbolic Math Toolbox, diferenciación numérica mediante diferencias finitas como la fórmula de diferencia media.
d) Generar las ecuaciones de sensibilidad local de primer orden mediante la ecuación (6)

Los procedimientos anteriores se pueden implementar mediante el modelo Simulink mostrado en la Figura 3.

Las funciones de sensibilidad local de primer orden obtenidas mediante las primeras derivadas de las variables de estado con respecto a los parámetros, se denominan funciones de sensibilidad local absolutas y por lo general tienen diferentes unidades y no pueden compararse entre sí, para evitar esto es común normalizarlas con el fin de volverlas adimensionales (Turányi y Tomlin, 2014).

Las funciones de sensibilidad local de primer orden son adimensionales y se conocen como funciones de sensibilidad local relativa o normalizada, existen diferentes maneras de normalizar las funciones de sensibilidad local como multiplicarlas por el cociente del parámetro entre la variable de estado correspondientes, siempre y cuando esta última no valga cero (Turányi y Tomlin, 2014).

Los datos son un elemento fundamental en el análisis de biosistemas, y en otras disciplinas científicas, por lo que su documentación es fundamental a fin de poder hacer un uso adecuado de los mismos. De la misma manera en que no podemos hacer un uso adecuando de un sistema de cómputo si no contamos con su documentación, para poder utilizar de manera correcta los datos de entrada de un sistema dinámico es necesario contar con su respectiva documentación, la cual se conoce como metadatos.

Los metadatos se definen comúnmente como "datos sobre datos", de acuerdo con su etimología, y su objetivo es describir una propiedad o un conjunto de propiedades útiles de un recurso u objeto de información (Sicilia, 2014) y su generación se tratará en el apéndice 1.

3. MATERIALES Y MÉTODOS.

3.1. MODELOS SIMULINK PARA SENSIBILIDAD LOCAL

Los análisis de sensibilidad local realizados con Matlab/Simulink que se encontraron en la revisión bibliográfica, utilizan dos enfoques:

a) Generación de código en lenguaje de programación C, el cual se compila y se encadena con Matlab mediante un modelo Simulink con tres bloques, un puerto de entrada (Inport), un bloque función (S-Function) y un puerto de salida (Outport).
b) Generación de un modelo Simulink con todos los bloques necesarios para realizar el análisis de sensibilidad local.

Para generar un modelo, e incluso para entender un modelo existente, empleando lenguaje C se requieren conocimientos de programación de computadoras y de la instalación de un compilador de lenguaje C compatible con Matlab, pero el empleo de lenguaje C acelera el procesamiento de modelos complejos, por lo cual en el presente trabajo se optó por implementar ambas opciones aunque se recomienda la primera ya que un modelo gráfico es más fácil de entender e incluso modificar, dado que como se dijo anteriormente una imagen dice más que mil palabras.

Se encontraron tres modelos implementados en Simulink, los cuales utilizan un enfoque algebraico lo cual los hace si no complejos si extensos, como se muestra en la tabla 1.

MODELO	VARIABLES DE ESTADO	PARÁMETROS	BLOQUES
Seginer *et al.*, 1991	1	10	74
Van Straten 2008	2	4	106
Garduño, 2012	3	8	361

Tabla 1. Número de bloques de diferentes modelos en Simulink

El número de bloques de los modelos de la Tabla 1, se obtuvo mediante el comando de Matlab **sldiagnostics ('modelo','CountBlocks')**, con el cual se obtiene el número total de bloques y los subtotales para cada tipo de bloque,

el primer modelo de la Tabla 1 se tiene en un archivo denominado **modelseginer911sa.mdl**, por lo que mediante el comando **sldiagnostics ('modelseginer911sa','CountBlocks')**, se obtiene lo siguiente:

```
'Finished counting blocks in 'modelseginer911sa'.
 Found 74 blocks.
      modelseginer911sa Total blocks :   74
                    Fcn :   27
                 Inport :    2
             Integrator :   11
                    Mux :   22
                Outport :   12
```

Lo anterior significa que el modelo **modelseginer911sa.mdl** cuenta con un total de 74 bloques (27 funciones, 2 puertos de entrada, 11 integradores, 22 multiplexores y 12 puertos de salida).

Como se puede ver en la Figura 2 la implementación de un modelo Simulink utilizando un enfoque algebraico da lugar a la repetición de grupos de bloques con estructura similar que se repiten, incrementando el tamaño del modelo de manera geométrica. Cabe recalcar que la Figura 2 se incluye no para analizarla, dado que el tamaño del texto es demasiado pequeño, sino únicamente para mostrar sus dimensiones.

El modelo mostrado en la Figura 2 (Seginer *et al.*, 1991) es bastante sencillo ya que cuenta con:

a) Una variable de estado: materia seca de la lechuga (W)
b) Dos variables de entrada: radiación fotosintéticamente activa (PAR) y temperatura ambiente (T)
c) Diez parámetros: fracción de la parte aérea respecto a la materia seca total (m), eficiencia fotosintética (épsilon), coeficiente de iluminación (K), coeficientes de la tasa de respiración de mantenimiento (Rr) y (b), relación de área foliar (Lambda), temperatura de referencia (Tref), ordenada al origen de la fotosíntesis bruta (Br), coeficiente lineal de la

fotosíntesis bruta (Alfa), coeficiente cuadrático de la fotosíntesis bruta (Beta)

En el mismo artículo (Seginer *et al.*, 1991), los autores proponen un modelo con una variable de estado adicional, índice de área foliar (L), por lo que siguiendo este esquema algebraico el modelo Simulink para el análisis de sensibilidad local prácticamente se duplicaría al pasar de 74 a 138 bloques, no se cuenta con este modelo pero su tamaño es fácil de calcular ya que el modelo para una variable de estado de la Figura 2 cuenta con 10 bloques comunes y 64 particulares para la variable de estado W, por lo que el tamaño del modelo para dos variables de estado sería igual a 10 + 2*64=138.

Con base a lo anterior si tuviéramos el modelo anterior pero con 10 variables de estado, en número de bloque necesario seria de 10+10*64=670 bloques, este rápido crecimiento del número de bloques mediante el enfoque algebraico, hizo necesario desarrollar un enfoque matricial más compacto, el cual se describe en la siguiente sección (3.2).

Como se verá más adelante este enfoque matricial compacto es el que se utiliza en los programas desarrollados, LSAGs y LSAGc, y es una de las principales aportaciones del presente trabajo, ya que facilita la implementación de código Matlab/Simulink para el análisis de sensibilidad local aún de manera manual.

Los modelos Simulink se pueden guardar en dos formatos, un formato tipo texto con extensión mdl y un formato binario con extensión slx, el cual es mucho más compacto que el primero ya que por ejemplo el archivo **modelseginer911sa.mdl** tiene un tamaño de 72 KB mientras que si lo guardamos como **modelseginer911sa.slx** su tamaño se reduce a 31 KB, razón por la cual los programas generados LSAGs y LSAGc generan archivos tipo slx.

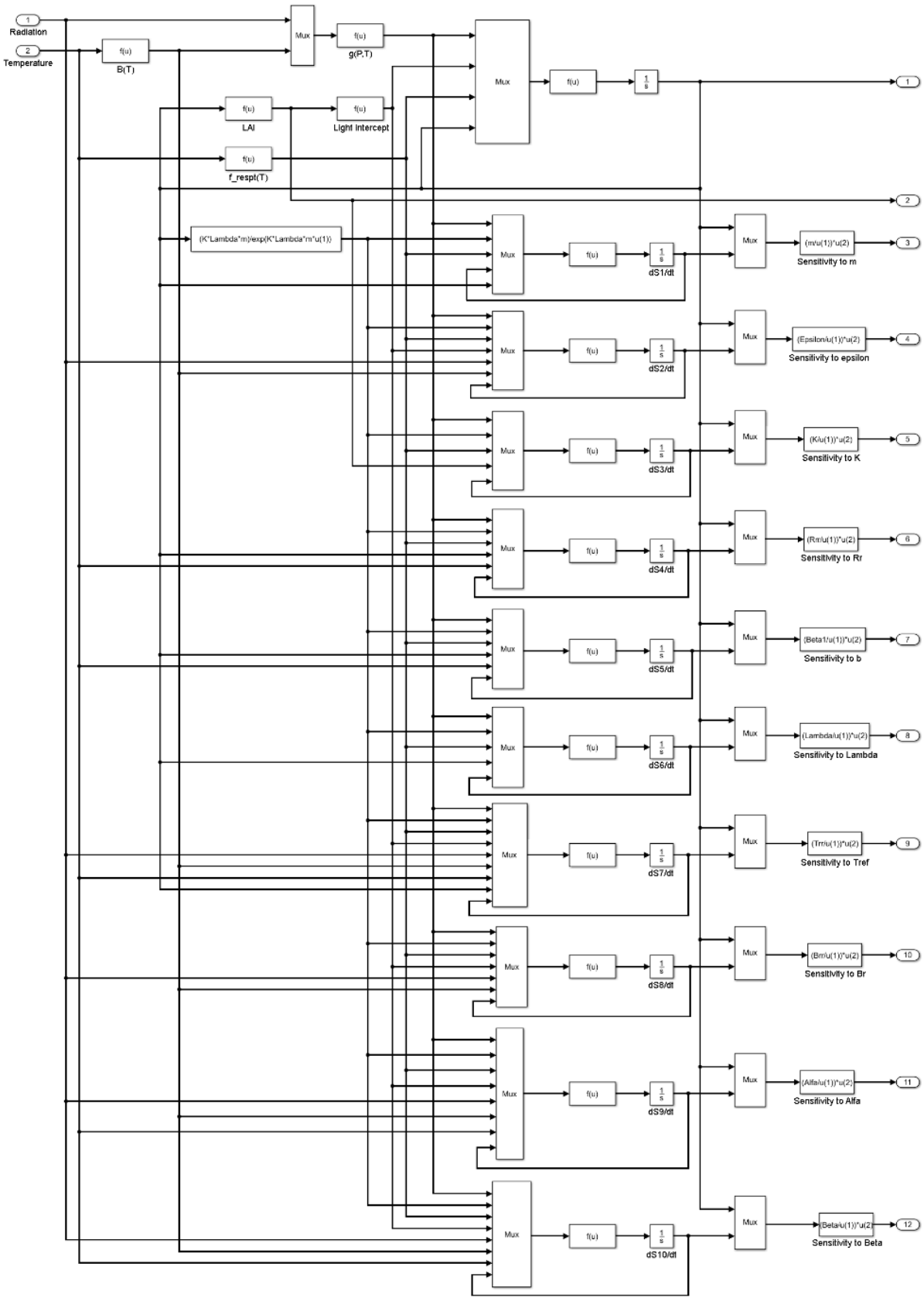

Figura 2. Modelo Seginer *et al*., 1991 implementado en Simulink

3.2. ENFOQUE MATRICIAL DE MODELO SIMULINK PARA ASL

El problema del crecimiento exponencial de los modelos Simulink conforme se incrementa su número de variables de estado y de parámetros, es muy similar al incremento del tamaño de las fórmulas algebraicas de los modelos de regresión lineal en relación a su número de variables independientes, para un modelo de regresión lineal simple, una variable independiente, tenemos que las fórmulas algebraicas para calcular los coeficientes de regresión, expresados como sumas de productos, son:

$$\beta_0 = \frac{\Sigma X \, \Sigma XY - \Sigma XX \, \Sigma Y}{\Sigma X \, \Sigma X - n \, \Sigma XX} \qquad (7)$$

$$\beta_1 = \frac{-n \, \Sigma XY + \Sigma X \, \Sigma Y}{\Sigma X \, \Sigma X - n \, \Sigma XX} \qquad (8)$$

Las ecuaciones (7) y (8), así como las contenidas en la Tabla 3 se generaron en Mathematica a partir de la solución de las ecuaciones normales correspondientes y difieren de las que aparecen en los libros de estadística, pero se comprobó que son equivalentes, la notación empleada es de sumas de productos por lo que ΣXX corresponde a la suma de cuadrados de X.

Como podemos apreciar las ecuaciones (7) y (8) tienen cuatro elementos en el numerador y cuatro en el denominador, incluyendo sumatorias y n, dando un total de 16 términos a evaluar para el cálculo de los dos coeficientes de regresión, pero conforme se incrementa el número de variables independientes, el número de elementos se incrementa como se puede apreciar en las Tablas 2 y 3.

Se cuenta con las fórmulas para un modelo de regresión lineal con cuatro variables independientes, pero su tamaño es 5 veces mayor que las del modelo de regresión lineal con tres variables independientes por lo que la fórmula de la ordenada al origen requiere de una tabla de más de dos páginas de extensión, también es importante señalar que no fue posible generar las fórmulas para un modelo de regresión lineal con 5 variables independientes, usando Mathematica.

Los números de la Tabla 2 pueden variar de acuerdo al grado de simplificación de las fórmulas, pero dan una idea clara del incremento exponencial del número de términos totales involucrados en el cálculo de los coeficientes de regresión en función del número de variables independientes. Para evitar el uso de estas fórmulas algebraicas, es más práctico utilizar la fórmula matricial indicada en la ecuación (9) la cual es general, por lo que no depende del número de variables independientes.

VARIABLES INDEPENDIENTES	TÉRMINOS TOTALES
1	16
2	83
3	512
4	2793

Tabla 2. Número de términos para el cálculo de los coeficientes de regresión

VARIABLES	FÓRMULA
1	$(\Sigma X\ \Sigma XY-\Sigma XX\ \Sigma Y)/(\Sigma X^2-n\ \Sigma XX)$
2	$(-\Sigma X1X2\ (\Sigma X1\ \Sigma X2Y+\Sigma X1Y\ \Sigma X2)+\Sigma X1\ \Sigma X1Y\ \Sigma X2X2+\Sigma X1X1\ \Sigma X2\ \Sigma X2Y-\Sigma X1X1\ \Sigma X2X2\ \Sigma Y+(\Sigma X1X2)^2\ \Sigma Y)/(n\ ((\Sigma X1X2)^2-\Sigma X1X1\ \Sigma X2X2)+(\Sigma X1)^2\ \Sigma X2X2-2\ \Sigma X1\ \Sigma X1X2\ \Sigma X2+\Sigma X1X1\ (\Sigma X2)^2)$
3	$-\Sigma X1X3\ (-\Sigma X1\ \Sigma X2X2\ \Sigma X3Y+\Sigma X1\ \Sigma X2X3\ \Sigma X2Y+\Sigma X1X2\ \Sigma X2\ \Sigma X3Y-2\ \Sigma X1X2\ \Sigma X2X3\ \Sigma Y+\Sigma X1X2\ \Sigma X2Y\ \Sigma X3+\Sigma X1Y\ \Sigma X2\ \Sigma X2X3-\Sigma X1Y\ \Sigma X2X2\ \Sigma X3)+\Sigma X1\ (-\Sigma X1X2\ \Sigma X2X3\ \Sigma X3Y+\Sigma X1X2\ \Sigma X2Y\ \Sigma X3X3-\Sigma X1Y\ \Sigma X2X2\ \Sigma X3X3+\Sigma X1Y\ (\Sigma X2X3)^2)+\Sigma X1X1\ \Sigma X2\ \Sigma X2X3\ \Sigma X3Y-\Sigma X1X1\ \Sigma X2\ \Sigma X2Y\ \Sigma X3X3-\Sigma X1X1\ \Sigma X2X2\ \Sigma X3\ \Sigma X3Y+\Sigma X1X1\ \Sigma X2X2\ \Sigma X3X3\ \Sigma Y-\Sigma X1X1\ (\Sigma X2X3)^2\ \Sigma Y+\Sigma X1X1\ \Sigma X2X3\ \Sigma X2Y\ \Sigma X3+(\Sigma X1X2)^2\ \Sigma X3\ \Sigma X3Y-(\Sigma X1X2)^2\ \Sigma X3X3\ \Sigma Y+\Sigma X1X2\ \Sigma X1Y\ \Sigma X2\ \Sigma X3X3-\Sigma X1X2\ \Sigma X1Y\ \Sigma X2X3\ \Sigma X3+(\Sigma X1X3)^2\ (\Sigma X2\ \Sigma X2Y-\Sigma X2X2\ \Sigma Y)/(-\Sigma X3X3\ (n\ ((\Sigma X1X2)^2-\Sigma X1X1\ \Sigma X2X2)+(\Sigma X1)^2\ \Sigma X2X2-2\ \Sigma X1\ \Sigma X1X2\ \Sigma X2+\Sigma X1X1\ (\Sigma X2)^2)+2\ \Sigma X1X3\ (n\ \Sigma X1X2\ \Sigma X2X3-\Sigma X1\ \Sigma X2\ \Sigma X2X3+\Sigma X1\ \Sigma X2X2\ \Sigma X3-\Sigma X1X2\ \Sigma X2\ \Sigma X3)-n\ \Sigma X1X1\ (\Sigma X2X3)^2+(\Sigma X1X3)^2\ ((\Sigma X2)^2-n\ \Sigma X2X2)+(\Sigma X1)^2\ (\Sigma X2X3)^2-2\ \Sigma X1\ \Sigma X1X2\ \Sigma X2X3\ \Sigma X3+2\ \Sigma X1X1\ \Sigma X2\ \Sigma X2X3\ \Sigma X3-\Sigma X1X1\ \Sigma X2X2\ (\Sigma X3)^2+(\Sigma X1X2)^2\ (\Sigma X3)^2)(-\Sigma X1X3\ (-\Sigma X1\ \Sigma X2X2\ \Sigma X3Y+\Sigma X1\ \Sigma X2X3\ \Sigma X2Y+\Sigma X1X2\ \Sigma X2\ \Sigma X3Y-2\ \Sigma X1X2\ \Sigma X2X3\ \Sigma Y+\Sigma X1X2\ \Sigma X2Y\ \Sigma X3+\Sigma X1Y\ \Sigma X2\ \Sigma X2X3-\Sigma X1Y\ \Sigma X2X2\ \Sigma X3)+\Sigma X1\ (-\Sigma X1X2\ \Sigma X2X3\ \Sigma X3Y+\Sigma X1X2\ \Sigma X2Y\ \Sigma X3X3-\Sigma X1Y\ \Sigma X2X2\ \Sigma X3X3+\Sigma X1Y\ (\Sigma X2X3)^2)+\Sigma X1X1\ \Sigma X2\ \Sigma X2X3\ \Sigma X3Y-\Sigma X1X1\ \Sigma X2\ \Sigma X2Y\ \Sigma X3X3-\Sigma X1X1\ \Sigma X2X2\ \Sigma X3\ \Sigma X3Y+\Sigma X1X1\ \Sigma X2X2\ \Sigma X3X3\ \Sigma Y-\Sigma X1X1\ (\Sigma X2X3)^2\ \Sigma Y+\Sigma X1X1\ \Sigma X2X3\ \Sigma X2Y\ \Sigma X3+(\Sigma X1X2)^2\ \Sigma X3\ \Sigma X3Y-(\Sigma X1X2)^2\ \Sigma X3X3\ \Sigma Y+\Sigma X1X2\ \Sigma X1Y\ \Sigma X2\ \Sigma X3X3-\Sigma X1X2\ \Sigma X1Y\ \Sigma X2X3\ \Sigma X3+(\Sigma X1X3)^2\ (\Sigma X2\ \Sigma X2Y-\Sigma X2X2\ \Sigma Y))$

Tabla 3. Formulas algebraicas de la ordenada al origen

$$\boldsymbol{\beta} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{Y} \qquad (9)$$

Si bien las dimensiones de los elementos de la ecuación (9) varían de acuerdo al número de variables independientes y de observaciones, la ecuación es la misma para cualquier número de variables independientes.

Lo mismo ocurre con el modelo matricial Simulink mostrado en la Figura 3, el cual realiza el ASL de cualquier modelo dinámico independientemente de su número de estados y de parámetros, aunque lo mismo que en la ecuación (9) cambian las dimensiones de los bloques System, M Matrix y A matrix.

La parte superior del modelo matricial Simulink de la Figura 3, resuelve numéricamente la ecuación de estado utilizando un solo bloque de integración independientemente del número de ecuaciones diferenciales del sistema.

La parte inferior de dicho modelo, resuelve numéricamente la ecuación de sensibilidad local utilizando un solo bloque de integración independientemente del número de estados y parámetros del sistema, utilizando un bloque multiplicador de matrices y un bloque de suma de matrices a fin de generar las el sistema de ecuaciones diferenciales de la matriz de sensibilidad local, ecuación (6).

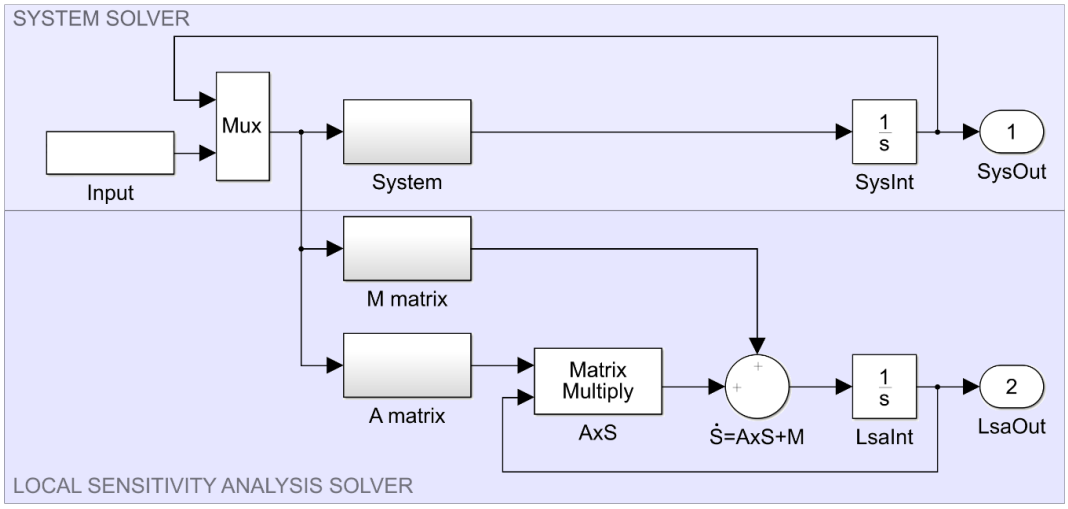

Figura 3. Modelo Simulink matricial para el análisis de sensibilidad local

Las matrices Jacobianas, se generan analíticamente mediante el Symbolic Math Toolbox de Matlab, para lo cual el usuario solamente tiene que proporcionar el archivo de modelo, en formato Matlab y el archivo de datos, en formato tscollection, de acuerdo a como se indica en el manual de usuario (apéndice 1).

Se seleccionó la sintaxis Matlab para el archivo del modelo dado que si un usuario está utilizando un programa implementado en dicha plataforma, se asume que tiene los conocimientos necesarios para generar dicho archivo del modelo, evitando que tenga que aprender otros formatos como el SBML (Systems Biology Markup Language) que es el que utilizan la mayoría de los programas disponibles para realizar el ASL.

Por otro lado el formato para los datos de entrada, tscollection, se seleccionó por facilitar la documentación de dichos datos, y permitir su validación a fin de evitar que se utilice un archivo de datos con una estructura no válida para el modelo de analizado.

3.4. IMPLEMENTACIÓN DEL PROGRAMA.

Un programa en Matlab se puede implementar de diversas maneras como son:

a) Utilizando la herramienta de Matlab **App Designer**, la cual se ejecuta mediante el comando **HOME/New/App/App Designer** o tecleando **appdesigner** en la consola de comandos
b) Creando la interface de usuario mediante la herramienta GUIDE (**G**raphical **U**ser **I**nterface **D**evelopment **E**nvironment) la cual se ejecuta mediante el comando **HOME/New/App/GUIDE** o tecleando **guide** en la consola de comandos. Una vez creada la interface mediante esta herramienta hay que exportarla en forma de código Matlab e incluirla en la aplicación.
c) Generar la interface de usuario programáticamente, lo cual da un mayor control sobre el diseño y la programación de la aplicación, razón por la cual se seleccionó en el presente trabajo.

La interface de usuario generada y su utilización se detallan en el apéndice 1.

El cálculo de la matriz A y la matriz M se llevaron a cabo mediante la utilización de la caja de herramientas de matemáticas simbólicas de Matlab (Symbolic Math Toolbox). Dado que Matlab no es un programa orientado a la matemática simbólica las funciones obtenidas mediante Symbolic Math Toolbox distan mucho de ser eficientes y compactas.

El programa Mathematica que está más orientado a la Mathematica simbólica y cuenta con dos comandos para simplificar expresiones, uno Simplify el cual da resultados parecidos a los del Symbolic Math Toolbox, y el otro FullSimplify con el cual se obtienen expresiones mucho más compactas y eficientes, como se muestra en el apéndice 3, por lo que para el análisis de modelos complejos sería muy útil desarrollar un sistema que utilizara ambos programas.

Desafortunadamente R que es uno de los programas más utilizados en la actualidad, no cuenta con una librería de matemática simbólica por lo que no es factible migrar los sistemas generados, pero existe un programa

denominado SciLab (https://www.scilab.org/), el cual es compatible con Matlab, tiene un módulo denominado Xcos similar a Simulink y cuenta con una caja de herramientas (SciMax) que le permite realizar operaciones de matemática simbólica a través del CAS Maxima, que es sin duda el programa matemático libre más robusto ya que se viene desarrollando desde el año de 1982, lo que lo convierte en un buen candidato para migrar los programas generados en el presente trabajo.

Es importante señalar que en el código en lenguaje C que se genera mediante el programa LSAGc, las matrices Jacobiana y Jacobiana paramétrica se denominan A_ y M_ respectivamente, con la finalidad de que los nombres A y M estén disponibles para el usuario, pero por tal motivo el usuario no puede utilizar los nombres A_ y M_.

El diagrama de bloques del proceso se indica en la Figura 4, y como se puede apreciar los principales pasos son:

a) Se solicita el archivo a procesar. Dado que los programas LSAGs y LSAGc cuentan con una interface gráfica esto se facilita ya que se muestran al usuario los archivos con extensión lsag, la cual se seleccionó dado que no se encontraron archivos en Windows con dicha extensión lo cual evita posibles confusiones.

b) Se ejecuta el archivo seleccionado. Dado que los programas LSAGs y LSAGc se ejecutan en Matlab, no tiene ningún sentido desarrollar una rutina para la interpretación del archivo lsag, puesto que los archivos lsag utilizan sintaxis Matlab por lo que se procesan directamente mediante el comando run de Matlab.

c) Se valida el Workspace. Después de ejecutar el archivo lsag en Matlab, se deben de generar las siguientes variables en el Workspace, auxiliary_cts, data_source, input_var, model_def, model_par, out_lang, owf, project_name, sim_time y state_var. De no ser así seguramente se le indica al usuario la lista de variables faltantes y se cancela el proceso,

a fin de que el usuario revise las secciones correspondientes a las variables faltantes y haga las correcciones pertinentes que por lo general corresponden a errores en el nombre de las mismas. Es importante señalar que las variables indicadas se pueden colocar en cualquier orden pero con la finalidad de estandarizar los archivos lsag se recomienda utilizar el orden más lógico que es: project_name, data_source, sim_time, out_lang, owf, auxiliary_cts, input_var, state_var, model_par y model_def (como se puede ver en la Figura A1.4 del apéndice 1).

d) Se determinan las unidades de tiempo. Este proceso se ilustra en la Figura 5 y consiste en verificar si existe un archivo de datos en el que se definan estas unidades, de no haber archivo de datos o unidades definidas en el mismo se analizan las unidades del archivo lsag a fin de determinarlas, y de no hallarse tampoco de esta manera se asume que las unidades del tiempo de simulación son días. Por lo anterior es muy importante que de utilizarse un archivo de datos se definan en este las unidades del tiempo de simulación y además colocar comentarios en el archivo lsag indicando las unidades de cada una de los identificadores utilizados, como se puede ver en la Figura A1.4 del apéndice 1.

e) Consistencia del tiempo de simulación y el archivo de datos. Esta validación se muestra en la Figura 6, y básicamente consiste en validar que si el modelo no tiene un archivo de entrada, el usuario indico data_source='none', no debe de haber tampoco variables de entrada definidas y se debe definir un tiempo de simulación válido ya que no tiene ningún sentido utilizar sim_time='all' ya que esto significa utilizar todos los tiempos del dato de entrada cuando este existe. Y se verifica que el archivo de datos sea de tipo tscollection, que es el único formato que aceptan los programas ya que es el único que permite el uso de metadatos.

f) Validación de las expresiones. En esta parte se verifica que los identificadores contenidos en el archivo lsag tengan nombres válidos, que no haya nombres repetidos y que las asignaciones no contengan errores de sintaxis ni elementos no definidos.

g) Generación de condiciones iniciales. Para cada variable de estado del modelo se genera una variable con sufijo cero a la cual se le asigna el valor inicial de dicha variable, por lo que no es necesario que el usuario las defina como en la sección de constantes auxiliares.

h) Cálculo de las matrices A y M. En los programas LSAGs y LSAGc la matriz A corresponde a la matriz Jacobiana y se calcula derivando cada ecuación de estado con respecto a cada variable de estado, mediante el Symbolic Math Toolbox de Matlab, mientras que la matriz M es la matriz Jacobiana paramétrica y se calcula derivando cada ecuación de estado con respecto a cada parámetro utilizando la misma herramienta. Por esta razón en un sistema con n variables de estado y m parámetros, la dimensión de la matriz A es n^2 y la de la matriz M de $m*n$.

i) Generación del archivo de bitácora. Todos los archivos generados por los sistemas LSAGs y LSAGc tienen un prefijo a fin de identificar el archivo con el cual se generaron, este prefijo es ls para LSAGs y lc para LSAGc. El nombre del archivo de bitácora tiene la forma prefijo_ project_name_lf.txt, en donde el prefijo depende del programa generador, project_name es el nombre indicado por el usuario en archivo lsag y lf (ele efe) indica que es un log file. Este archivo contiene el resultado de varias validaciones hechas por el sistema. Se verifica que no hayan variables auxiliares en el modelo expandido, que todos los parámetros se encuentren en el modelo, las variables de estado que no aparecen en cada una de las ecuaciones de estado las cuales dan lugar a ceros en la matriz A y los parámetros que no aparecen en cada una de las ecuaciones de estado los cuales dan lugar a ceros en la matriz M.

j) Generación del archivo Simulink. El programa LSAGs genera un archivo Simulink complejo el cual contiene todo el análisis de sensibilidad, mientras que el programa LSAGc genera un pequeño archivo Simulink con únicamente tres bloques; un puerto de entrada (Inport), un bloque función (S-Function) y un puerto de salida (Outport); el cual solo sirve como una interface entre Matlab y el programa compilado a partir del código en lenguaje C. El nombre de este archivo tiene la forma prefijo_project_name_mf en donde mf significa model file.

k) Generación del archivo conteniendo el código en lenguaje C. Este archivo solo se genera por el programa LSAGc y tiene la estructura de una S-FUNCTION en lenguaje C, la cual se compila para generar un MEX-file (MATLAB executable file). El nombre de este archivo tiene la forma lc_project_name_sf en donde sf significa source file.

l) Generación del archivo Matlab. Este archivo contiene a grandes rasgos tres secciones; una con valores por omisión generados a partir de las opciones seleccionadas por el usuario en la interface gráfica de los programas LSAGs y LSAGc, las cuales pueden ser cambiadas por el usuario; una segunda con la simulación del modelo, mediante el comando sim de Matlab, y finalmente una sección que genera una serie de gráficas y reportes de los resultados (las gráficas en formato PNG una tabla de áreas bajo las curvas de las funciones de sensibilidad local en formato csv, comma separated values, y la matriz de resultados de la simulación, variables de estado y funciones de sensibilidad, en formato xlsx de Excel). El nombre de este archivo tiene la forma prefijo_project_name_cf en donde cf significa code file.

m) Mensaje de fin de proceso. Al terminar la generación del programa para generar el análisis de sensibilidad local, los programas LSAGs y LSAGc mandan un mensaje indicando que el proceso concluyo con éxito e indicando el tiempo utilizado.

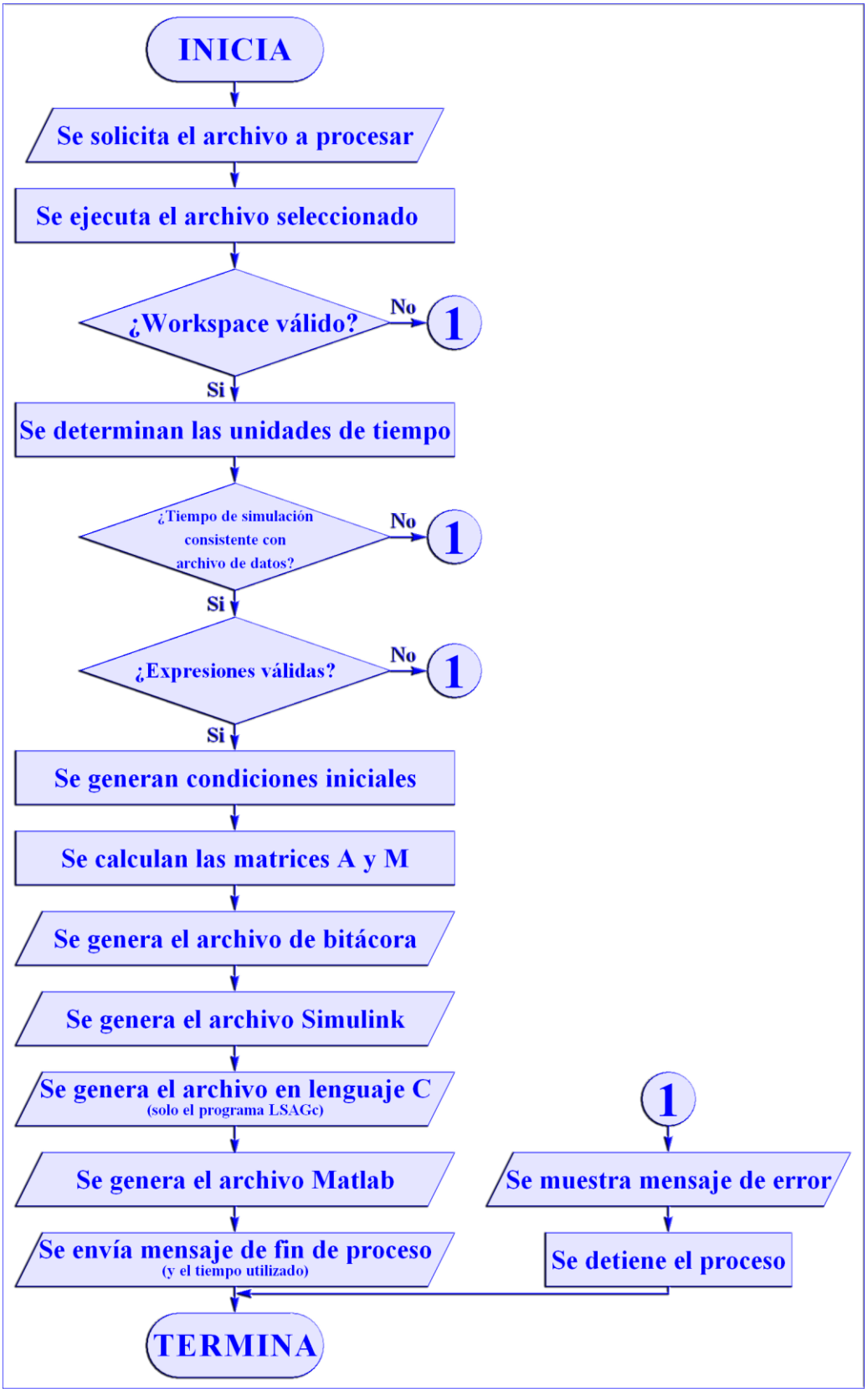

Figura 4. Diagrama de bloques del proceso

Si bien Matlab cuenta con el comando ccode, el cuál convierte una expresión simbólica a lenguaje C, el programa LSAGc utiliza una función propia denominada m2c, Matlab to C, la cual convierte directamente una cadena de caracteres en sintaxis Matlab en una cadena de caracteres en sintaxis de lenguaje C, lo cual facilita hacer pruebas rápidas ya que no se tiene que declarar nada como simbólico. Esto también permitió generar una función m2w la cual convierte una cadena de caracteres en sintaxis Matlab en una cadena de caracteres en lenguaje Wolfram, que es el que utiliza Mathematica, la cual fue de gran ayuda para las validaciones realizadas en el apéndice 3.

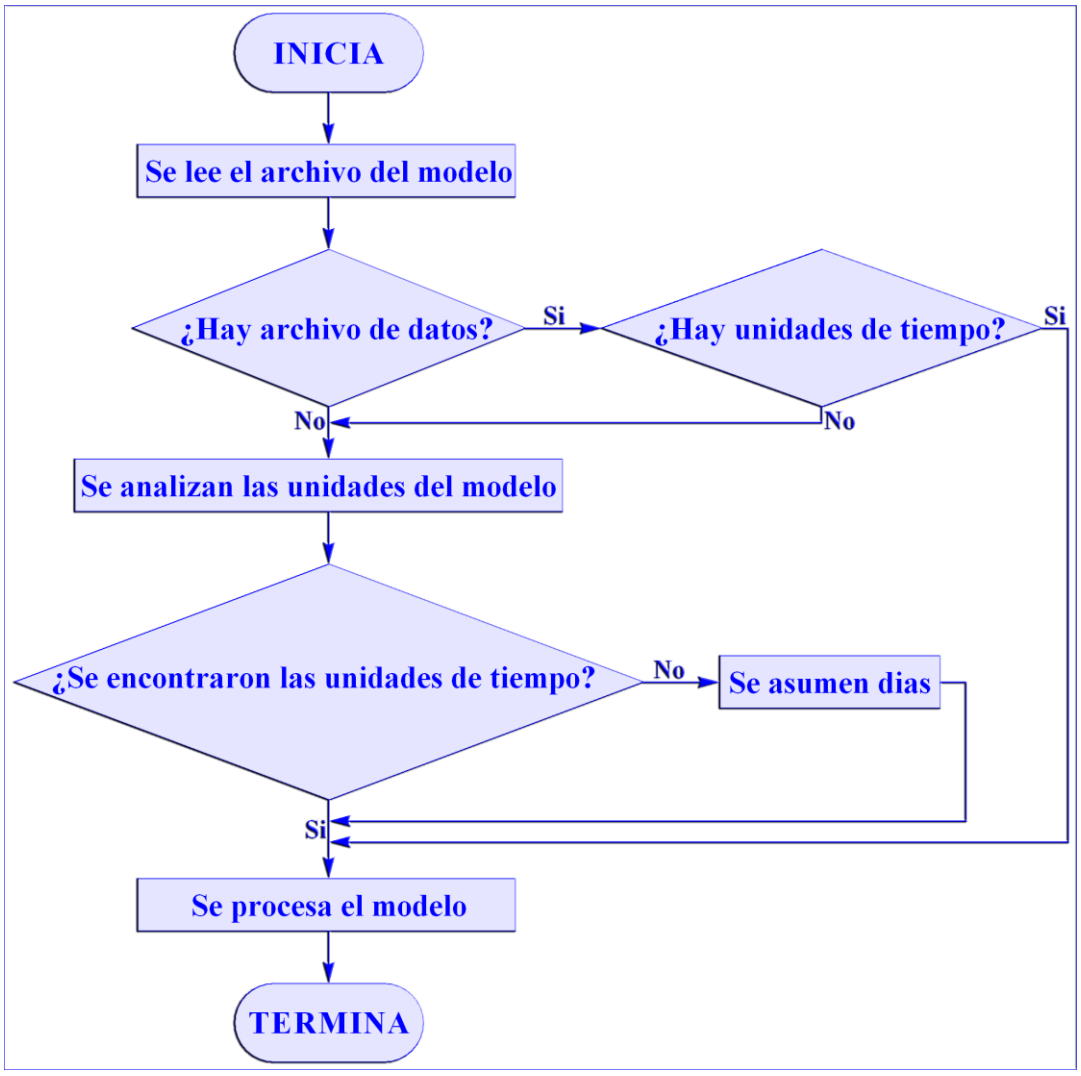

Figura 5. Determinación de las unidades del tiempo de simulación

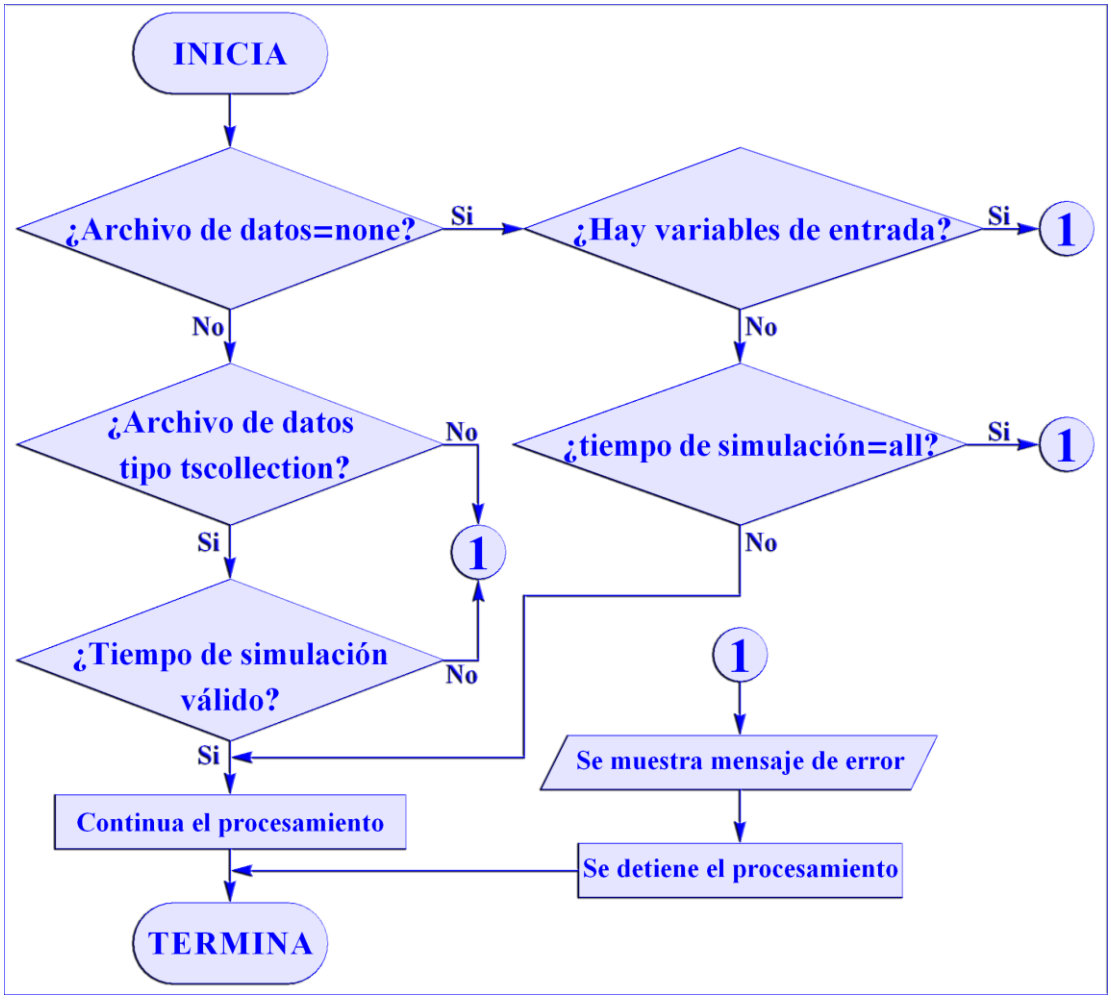

Figura 6. Validación de la consistencia entre el archivo de entrada y el tiempo de simulación.

4. RESULTADOS Y DISCUSIÓN.

El resultado del presente trabajo fue la implementación de dos programas en código Matlab, que a partir de un archivo de modelo, también con sintaxis Matlab y con extensión lsag, generan de manera automática el código necesario para realizar el análisis de sensibilidad local de un modelo dinámico en espacio de estados.

El primer programa se denominó LSAGs, Local Sensitivity Analysis Generator in Simulink, y el segundo LSAGc, Local Sensitivity Analysis Generator in C, y para facilidad del usuario ambos comparten la misma interface gráfica de usuario y el mismo archivo de modelo, los cuales se detallan en el apéndice 1.

Al simplificar el cálculo del análisis de sensibilidad local, el usuario puede concentrarse en el ***para que*** de este análisis y olvidarse del ***cómo***, con lo cual el análisis de sensibilidad local deja de ser un fin y se convierte en un medio para entender y utilizar mejor los modelos dinámicos.

Ambos programas requieren que las variables de entrada se encuentren en un archivo tipo tscollection, timeseries collection, el cual permite nombrar y documentar cada una de las variables de entrada incluyendo: sus unidades, su fecha y hora de adquisición, los sensores e instrumentos utilizados para su captura, la agencia y persona colectora, etc., lo cual se detalla en el apéndice 1.

Con lo anterior se pretende inculcar en el usuario:

a) Una cultura de metadatos, los cuales se definen como los “datos de los datos”, ya que la documentación de los datos incrementa considerablemente su utilidad.

b) Una cultura de sistematización de los modelos, ya que los archivos de modelo utilizados, con extensión lsag, resumen de una manera sistemática los elementos que configuran el modelo dinámico, los cuales en los documentos fuente pueden estar distribuidos en decenas de páginas. La idea es que este archivo de modelo, junto con información adicional, pueda ser utilizado en otro tipo de análisis, como análisis de sensibilidad global, calibración, evaluación, control, etc.

El programa LSAGs genera, a partir de las opciones seleccionadas por el usuario en la interface gráfica y los datos contenidos en el archivo de modelo, los siguientes archivos (todos con el prefijo ls_):

a) Un archivo de código con sufijo _cf (code file), el cual contiene el código Matlab para realizar el análisis de sensibilidad local.

b) Un archivo de modelo con sufijo _mf (model file), el cual contiene el modelo Simulink para realizar el análisis de sensibilidad local.

c) Un archivo de bitácora con sufijo _lf (log file), el cual contiene los mensajes generados por el sistema durante el proceso.

Por su parte el programa LSAGc genera lo tres archivos anteriores, pero con el prefijo lc_, más un cuarto archivo de código fuente en lenguaje C con sufijo _sf (source file), el cual contiene el programa en lenguaje C para realizar el análisis de sensibilidad local.

A manera de ejemplo vamos a trabajar con un modelo simple, ¿del artículo What can Systems and Control Theory do for Agricultural Science? (Van Straten, 2008), el cual se detalla en el Apéndice 2, para lo cual hacemos lo siguiente:

a) Generamos el archivo de datos de entrada, tipo tscollection, mediante el código Matlab mostrado en la Figura A1.1, al cual nombramos ti.mat.

b) Generamos el archivo de modelo, el cual se ilustra en la Figura A1.4, al cual nombramos vs2.lsag.

c) Ejecutamos el programa LSAGs, el cual muestra la interface gráfica de usuario mostrada en la Figura A1.5

d) En la interface gráfica de usuario oprimimos el botón **Load model**

e) En la ventana de modelos existentes, mostrada en la figura A1.6 seleccionamos el modelo vs2.lsag y oprimimos el botón **Abrir**, o damos **doble-click** al nombre del modelo.

f) Lo anterior inicia la creación del código Matlab/Simulink para realizar el análisis de sensibilidad local, y al terminar el programa muestra el mensaje de fin de proceso mostrado en la Figura A1.7.

g) Para realizar el análisis de sensibilidad local ejecutamos el programa Matlab generado, cuyo nombre es ls_vs2_cf.m.

Para usar el programa LSAGc, realizamos el mismo procedimiento pero entre los pasos f y g, hay que compilar el programa en lenguaje C generado, de nombre lc_vs2_cf.c.

Además de la verificación cruzada de los programas LSAGs y LSAGc, a fin de contar con una validación contundente de los programas generados, se compararon sus resultados con los de la solución analítica de un modelo LTI simple, la cual se detalla en el apéndice 2, a manera de resumen podemos subrayar lo siguiente:

a) Los resultados de LSAGs y LSAGc tuvieron una diferencia absoluta mínima de cero una diferencia absoluta máxima de 1.42109E-14 y una diferencia absoluta promedio de 1.15477E-15, lo cual concuerda con la precisión de las variables de doble precisión utilizadas en los cálculos.

b) Las diferencias absolutas entre los resultados del programa LSAGs y LSAGc con respecto a la solución analítica del modelo tuvieron en ambos casos un valor mínimo de cero, un valor máximo de 2.8463E-08 y un valor promedio de 1.90288E-09.

Por lo anterior podemos concluir que la diferencia entre los valores promedio de los incisos anteriores, 1.90288E-09 - 1.15477E-15 = 1.9029E-09, es el error asociado a la solución numérica del modelo.

Los tiempos de generación del código para el análisis de sensibilidad local, utilizando LSAGs y LSAGc, así como el tiempo de ejecución de dicho código, para varios modelos se muestran en la Tabla 4.

De los datos de la Tabla 4, podemos deducir lo siguiente:

a) Los tiempos de generación del código para el análisis de sensibilidad local, son prácticamente los mismos para LSAGs y LSAGc.

b) Los tiempos de ejecución del código generado es prácticamente el mismo para el modelo Matlab/Simulink que para Matlab/Simulink/Lenguaje C en modelos simples, pero en modelos complejos se nota un mejor desempeño del código con lenguaje C, por lo que a medida que se incrementa la complejidad de los modelos más conveniente es el uso de LSAGc. Para el modelo Seginer, 2003 (NICOLET con 5 variables de estado y 18 parámetros) el código en lenguaje C se ejecuta en el 44% del tiempo utilizado por el modelo completamente en Simulink.

	LSAGs		LSAGc	
MODELO	GENERACIÓN	ANÁLISIS	GENERACIÓN	ANÁLISIS
Van Straten, 2008	1.69	2.17	1.65	2.74
Garduño, 2012	2.04	2.27	1.99	2.21
Seginer, 2003	9.85	22.65	10.21	9.92

Tabla 4. Tiempo en segundos de la generación y ejecución de varios modelos

5. CONCLUSIONES.

El análisis de sensibilidad local es una herramienta valiosa y necesaria para poder cuantificar el efecto que tienen los parámetros sobre las variables de estado y las variables de salida de un modelo dinámico, pero los métodos para llevarlo a cabo con que se cuenta actualmente, código en lenguaje C y modelo Simulink, lo convierten en un proceso laborioso y propenso a error por lo que son pocos los trabajos de investigación que hacen uso del mismo, de una revisión de los trabajos de tesis del posgrado en Ingeniería Agrícola y Uso Integrad del Agua en los que se utilizan modelos dinámicos, solamente en uno se hace análisis de sensibilidad local.

Dado que la solución analítica del análisis de sensibilidad de un modelo dinámico solo es factible en un número reducido de casos (modelos lineales, o linealizados, sin variables de entrada o con variables de entrada analíticas en forma de funciones integrables), la única manera de facilitar este tipo de análisis es mediante la generación de una herramienta computacional que lo automatice, que fue el objetivo del presente trabajo.

Si bien se automatizó la generación de código en lenguaje C, en general es más fácil la utilización de un modelo Simulink, dado que el usuario no requiere tener habilidades de programación de computadoras ni contar con un compilador compatible con Matlab.

En este trabajo, se pudo automatizar la generación de modelos con enfoque algebraico, que son los más utilizados actualmente, pero se optó por generar modelos con enfoque matricial por ser más eficientes y compactos.

El modelo Simulink matricial implementado en el presente trabajo es una de sus principales aportaciones, ya que permite realizar el análisis de sensibilidad local de cualquier modelo con solo 11 bloques principales, independientemente del número de parámetros y variables de estado del modelo analizado.

El programa generado permite calcular el análisis de sensibilidad local de un modelo dinámico con 5 variables de estado y 18 parámetros en tan solo 13 minutos en una computadora portátil estándar.

6. REFERENCIAS.

Beven, K. 2009. Environmental Modelling: An Uncertain Future?. Routledge, Taylor & Francis Group. Oxon, UK. 310 pp.

Beven, K. 2012. Rainfall-Runoff Modelling. John Wiley & Sons Ltd. Chichester, UK. 457 pp.

Borgonovo, E. 2017. Sensitivity Analysis: An Introduction for the Management Scientist. Springer International Publishing. Cham, Switzerland. 294 pp.

Brouwer-Burg, M.; Peeters, H. y Lovis, W. A. 2016. Uncertainty and Sensitivity Analysis in Archaeological Computational Modeling. Springer International Publishing. Cham, Switzerland. 175 pp.

Cacuci, D. G. 2018. The Second-Order Adjoint Sensitivity Analysis Methodology. CRC Press, Taylor & Francis Group. Boca Raton, Florida. USA. 305 pp.

Chen, Chi-Tsang. 2013. Linear system theory and design. Oxford University Press. New York, USA. 386 pp.

Constanda, C. 2017. Differential Equations: A Primer for Scientists and Engineers. Springer International Publishing. Cham, Switzerland. 297 pp.

De Wit, C.T. 1982. Simulation of living systems. In Simulation of plant growth and crop production. Eds. F.W.T. Penning de Vries and H.H. van Laar. Simulation Monographs. PUDOC. Wageningen. The Netherlands. pp. 3-8.

DiStefano III, J. 2013. Dynamic Systems Biology: Modeling and Simulation. Academic Press, Elsevier Inc. London, UK. 859 pp.

Felli, J. y Hazen, G. 2004. Javelin Diagrams: A Graphical Tool for Probabilistic Sensitivity Analysis, Decision Analysis 1(2), 93–107.

Ferrari, A.; Gutiérrez, S. & Sin, G. 2015. A comprehensive sensitivity and uncertainty analysis of a milk drying process. 25th European Symposium on Computer Aided Process Engineering. Copenhagen, Denmark

Garduño-García, A. 2012. Simulación del proceso de fermentación e cerveza artesanal. Tesis (Maestro en Ingeniería Agrícola y Uso Integral del Agua). México, Universidad Autónoma Chapingo: 84 pp.

Kremling, A. 2014. Systems Biology: Mathematical Modeling and Model Analysis. CRC Press, Taylor & Francis Group. Oxfordshire, UK. 359 pp.

Mordukhovich, B. S. 2013. Variational Analysis and Generalized Differentiation I: Basic Theory. Springer International Publishing. Cham, Switzerland. 579 pp.

Petropoulos, G. P. y Srivastava, P. K. 2017. Sensitivity Analysis in Earth Observation Modelling. Elsevier Inc. Amsterdam, Netherlands. 422 pp.

Saltelli, A.; Chan, K. y Scott, E. M. 2000. Sensitivity Analysis. John Wiley & Sons Ltd. Chichester, UK. 475 pp.

Saltelli, A.; Ratto, M.; Andres, T.; Campolongo, F.; Cariboni, J.; Gatelli, D.; Saisana, M. y Tarantola, S. 2008. Global Sensitivity Analysis: The Primer. John Wiley & Sons Ltd. Chichester, UK. 292 pp.

Saltelli, A.; Tarantola, S.; Campolongo, F. y Ratto, M. 2004. Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models. John Wiley & Sons Ltd. Chichester, UK. 219 pp.

Seginer, I. 2003. A Dynamic Model for Nitrogen-stressed Lettuce. Annals of Botany. 91: 623-635.

Seginer, I., G. Shina, L.D & Albright, L.S. March. 1991. Optimal temperature setpoints for greenhouse lettuce. Journal of Agricultural Engineering Research. 49: 209-22.

Sicilia, M. 2014. Handbook of metadata, semantics and ontologies. World Scientific Publishing Co. Toh Tuck, Singapore. 570 pp.

Soetaert, K. y Herman, M. J. 2009. A Practical Guide to Ecological Modelling. Springer International Publishing. Cham, Switzerland. 372 pp.

Tangirala, A. K. 2014. Principles of System Identification: Theory and Practice. CRC Press, Taylor & Francis Group. Boca Raton, Florida. USA. 828 pp.

Turányi, T. y Tomlin, A. S. 2014. Analysis of Kinetic Reaction Mechanisms. Springer International Publishing. Cham, Switzerland. 363 pp.

Van Straten, G. 2008. What can Systems and Control Theory do for Agricultural Science?. AUTOMATIKA 49(2008), 105-117.

Wallach, D.; Makowski, D.; Jones, J. W. y Brun, F. 2014. Working with Dynamic Crop Models: Methods, Tools, and Examples for Agriculture and Environment. Elsevier Inc. Amsterdam, Netherlands. 487 pp.

APÉNDICE 1. MANUAL DE USUARIO DEL PROGRAMA

INTRODUCCIÓN

Uno de los errores más comunes en el desarrollo de todo tipo de código computacional, desde programas formales hasta simples macroinstrucciones, es revolver la lógica de los datos con la de su procesamiento, uno de los libros más influyentes en el área de programación de computadoras es sin duda ***Algoritmos + Estructuras de datos = Programas***, escrito por el prominente científico informático suizo Niklaus Wirth, creador de varios lenguajes de programación incluyendo Pascal, cuyo nombre no deja ninguna duda sobre la independencia de estos elementos, por lo que en los programas generados, LSAGs y LSAGc, estos dos elementos son igualmente relevantes pero independientes.

DOCUMENTACION DE LOS DATOS

Para el uso correcto de los datos utilizados en biosistemas, es necesario contar con los metadatos descriptivos que indiquen su naturaleza, sus unidades, su fecha y hora de adquisición, instrumentos utilizados para su captura, agencia y persona colectora, etc.

Por lo anterior se seleccionó la estructura de datos de Matlab denominada colección de series de tiempo, *Time Series Collection o tscollection*, ya que permite un manejo eficiente de los datos así como su documentación como se muestra en la Figura A1.1.

```
% Se borra el espacio de trabajo
clear all;
% Se cierran todas las figuras
close all;
% Se define el tiempo de muestreo
Time = 0:0.00001:24;

% Se crea una serie de tiempo
% con los datos de temperatura
gte=sin(Time/(24/(2*pi)))*5 + 12;
Te=timeseries(gte',Time,'name', 'Te');
% Se definen las unidades de la
% temperatura (grados centígrados)
Te.DataInfo.Units = 'C';
% Se define información adicional sobre
% la temperatura (sensor utilizado)
Te.UserData = ['sensor: Generadas con la '...
 'función  sin(t/(24/(2*pi)))*5 + 12'];

% Se crea una serie de tiempo con
% los datos de radiación (I)
gi=sin(Time/(24/(2*pi)))*147 + 53;
gi(gi<0.0)=0.0;
I=timeseries(gi',Time,'name', 'I');
```

```
% Se definen las unidades de
% la PAR (Watt/metro cuadrado)
I.DataInfo.Units = 'W/m^2';
% Se define información adicional
% sobre la PAR (sensor utilizado)
I.UserData = ['sensor:  Generadas con la '...
 'función sin(t/(24/(2*pi)))*147 + 53'];

% Se crea la colección de series de tiempo
ti = tscollection({Te I},'name','ti');
% Se definen las unidades del tiempo (horas)
ti.TimeInfo.Units = 'hours';
% Se define el intervalo de tiempo (horas)
ti.TimeInfo.Increment = 0.00001;
% Se definen información adicional de los datos
ti.TimeInfo.UserData = ['(Lactuca sativa L.'];
% Se definen fecha y hora iniciales
ti.TimeInfo.StartDate = '01-Jan-2019 00:00:00';
% Se definen el formato de la fecha
ti.TimeInfo.Format = 'yyyy-mmm-dd';

% Se graba el archivo de datos
save ti -v7.3 ti
```

Figura A1.1 Código Matlab para generar una colección de series de tiempo con metadatos

Los metadatos facilitan el correcto uso de los datos, ya que por ejemplo el comando de Matlab ***plot(lett_uach.PAR)*** genera una gráfica con las fechas de los datos en el eje x y el nombre de la variable y sus unidades en el eje y, todo lo cual se toma de los metadatos, como se puede ver en la Figura A1.2.

Es común que en el eje x de las gráficas se prefiera poner un número secuencial en lugar de la fecha para hacer esto es necesario asignar una cadena de caracteres nula al campo StartDate mediante el comando ***Archivo_de_datos.TimeInfo.StartDate=’’***; que para el caso del ejemplo mostrado en la Figura A1.2 es ***lett_uach .TimeInfo.StartDate=***''; después de lo cual el comando ***plot(lett_uach.PAR)*** genera la gráfica mostrada en la Figura A1.3.

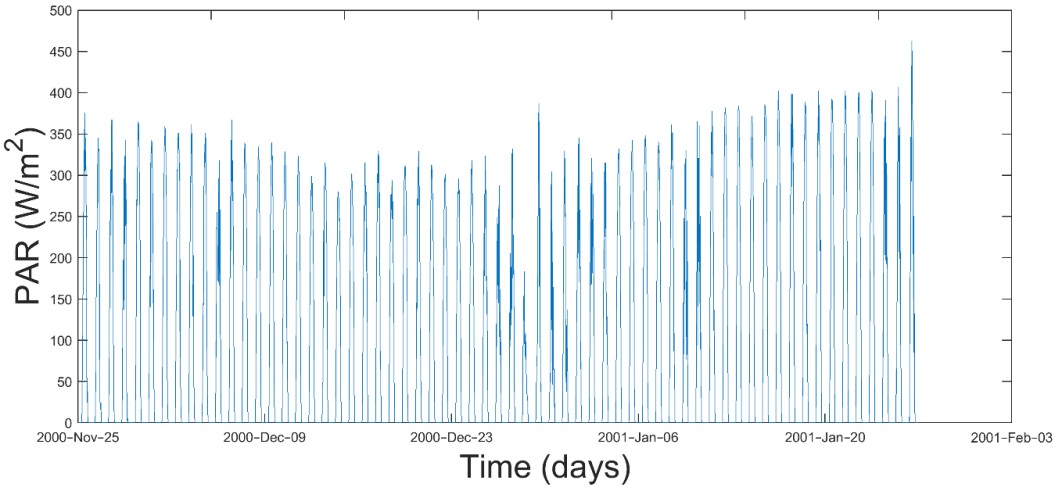

Figura A1.2. Gráfica de Matlab con las fechas y las unidades de los metadatos

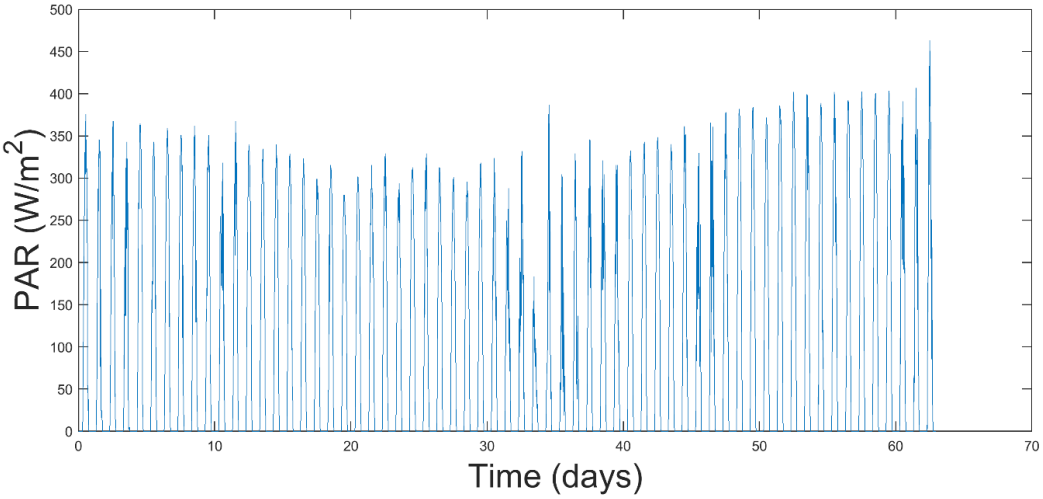

Figura A1.3. Gráfica de Matlab con números secuenciales en vez de fechas

Para recuperar la fecha inicial solo se vuelven a cargar los datos, mediante el comando ***load***, ya que la modificación se hace en la memoria y no en el archivo original, aunque también es posible salvarla a una variable temporal y volverla a definir, como se muestra a continuación:

```
tmp =lett_uach.TimeInfo.StartDate;        % Se almacena StartDate
lett_uach.TimeInfo.StartDate= '';         % Se elimina StartDate

…
lett_uach.TimeInfo.StartDate=tmp;         % Se recupera StartDate
```

Se recomienda que en las unidades se utilice el formato del Aerospace Toolbox a fin de facilitar su conversión mediante los comandos de esta herramienta.

Los programas generados, LSAGs y LSAGc, solo procesa datos en formato tscollection ya que sólo de esta manera se pueden incluir los metadatos, lo que permite validar que las variables de entrada definidas por el usuario, en el archivo lsag, correspondan a las contenidas en el archivo de datos, y en futuras versiones incluso se podrán verificar las unidades de todos los elementos del modelo mediante su análisis dimensional.

Si bien la generación del archivo de datos le podrá parecer al usuario una tarea laboriosa y lenta, en realidad es la única manera de documentar sus datos, facilitando su comprensión y aumentando la utilidad de los mismos, ya que los datos sin documentación en un principio solo los entiende dios y la persona que los creó, posteriormente solo dios y finalmente nadie.

DEFINICION Y PROCESAMIENTO DEL MODELO

Para realizar un análisis de sensibilidad local de un modelo dinámico en espacio de estados, el usuario tiene que indicar los datos del modelo y de las salidas deseadas mediante un archivo con sintaxis de Matlab pero con extensión ***lsag*** (Local Sensitivity Analysis Generator) como el mostrado en la Figura A1.4, se optó por no utilizar la extensión de Matlab (.m) con el fin de facilitar la identificación de los mismos, evitando que se confundan con los miles de archivos .m contenidos en una computadora con Matlab, los elementos requeridos en este archivo con extensión lsag y al que llamaremos archivo de modelo son los siguientes:

a) Nombre del proyecto. Es un nombre seleccionado por el usuario y el cual se utiliza como identificador de todos los archivos generados por el sistema.

b) Archivo de datos. Es el nombre del archivo de datos, ya que se asume .mat que es la extensión de los archivos tscollection, o ***none*** en caso de que no hayan datos de entrada, este archivo debe de contener una colección de series de tiempo con una serie de tiempo para cada variable de entrada del sistema. Los nombres de las variables de entrada deben coincidir con los utilizados en la sección de variables de estado que se analiza más adelante.

c) Tiempo de simulación. Debe ser una lista con los tiempos en que se desea realizar la simulación o la palabra ***all*** en caso de que se deseen simular todos los datos registrados, lo cual es muy tardado e

innecesario, ya que se recomienda muestrear con la máxima frecuencia soportada por los sensores y el equipo de almacenamiento para evitar errores por interpolación (cuando se utilizan métodos de integración con paso variable), pero simular a intervalos más espaciados, de preferencia múltiplos del intervalo de muestreo a fin de simular en tiempos para los cuales se tienen datos de entrada, disminuyendo así los errores por interpolación.

d) Idioma de salida. Es el idioma en que se generan los títulos y las etiquetas de las gráficas generadas por el sistema. Los textos se colocan en el código generado como valores por omisión que pueden ser modificados por el usuario, por el momento solo están disponibles i=Inglés y cualquier otro valor asume español.

e) Sobrescribir los archivos de salida. Es un valor lógico, 0=false 1=verdadero, que indica si el sistema pregunta si se desean sobrescribir los archivos de salida.

f) Constantes auxiliares. Lista de constantes auxiliares utilizadas por el modelo indicadas en forma de ***nombre***=***valor***.

g) Variables de entrada. Es una lista de las variables de entrada utilizadas en el modelo, para cada variable se debe indicar un coeficiente opcional por el cual se multiplican los valores de entrada por ejemplo para cambiar de unidades, un nombre que debe coincidir con el del archivo de entrada y su símbolo en sintaxis LaTEX.

h) Variables de estado. Es una lista de las variables de estado utilizadas en el modelo, para cada variable se debe indicar su valor inicial, su nombre y su símbolo en sintaxis LaTEX.

i) Parámetros del modelo. Es una lista de los parámetros del modelo, para cada parámetro se debe indicar su valor, su nombre y su símbolo en sintaxis LaTEX.

j) Definición del modelo. Es una lista con definiciones de variables auxiliares así como las ecuaciones diferenciales ordinarias de primer orden de la ecuación(es) de estado, las cuales se indican con el prefijo d (de derivada).

Como se puede apreciar en la Figura A1.4 el archivo de modelo con extensión lsag, tiene 10 variables, las primeras cinco son simples, auxiliary_cts y model_def son arreglos simples y las restantes son arreglos de arreglos y la definición de cada variable termina en punto y coma, para evitar que el resultado de la asignación se muestre en la ventana de comandos de Matlab.

Para facilitar la creación de los archivos de modelo, con extensión lsag, el contenido de todas las variables es del mismo tipo, cadenas de caracteres también conocidas como strings, lo que permite expresiones complejas por ejemplo '0:0.01:19' para el tiempo de simulación (sim_time), '12*shp' para el valor nominal de un parámetro o 'U_{mg}' para el símbolo LaTex de una variable de estado.

Dado que tanto los valores de las variables simples como los elementos de los arreglos de las variables de tipo arreglo son cadenas de caracteres, todos deben de ir entre comillas sencillas, esto puede pareces excesivo pero evita que el usuario tenga que recordar el tipo de dato de cada uno de los elementos del modelo.

Existen dos valores especiales para las variables, si un modelo no utiliza variables de entrada se indica mediante **data_source='none';** y si se quiere simulara en cada intervalo del archivo de datos se indica mediante **sim_time='all';**.

Si un modelo no utiliza constantes auxiliares se indica mediante **auxiliary_cts={};** y si no hay variables de entrada mediante **input_var={};.**

En cuanto al orden de las 10 variables, si bien es posible que los programas puedan operar con distintas combinaciones de las mismas, se recomienda que siempre se utilice el orden mostrado en la Figura A1.4 a fin de estandarizar los archivos generados y facilitar su manejo.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MODELO DE LA TEMPERATURA DEL AIRE Y DE LOS MATERIALES SÓLIDOS DE UN INVERNADERO %
%    tomado de: Van Straten, G. 2008. What can Systems and Control Theory do for  %
%               Agricultural Science?. Automatika 49(2008), 105-117.              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

project_name='vs2';             % Nombre del proyecto
data_source='ti';               % Archivo de datos
sim_time='0:1:13.4';            % Tiempo de simulación
out_lang='i';                   % Idioma de salida (i=inglés, otro=español)
owf='1';                        % Sobreescribe archivos

auxiliary_cts={                 % Constantes auxiliares
'sph=3600;'                     %   sph  = número de segundos por hora
};

input_var={                     % Variables de entrada (Coeficiente, nombre, LaTEX)
{'','Te','$T_e$'},              % Temperatura exterior (°C)
{'sph','I','$I$'}};             % Radiación (W m^-2)

state_var={                     % Variables de estado (valor inicial, nombre, LaTEX)
{'12.0','Tg','$T_{g}$'},        %   Temperatura del invernadero (°C)
{'12.0','Tm','$T_{m}$'}};       %   Temperatura de los materiales sólidos del invernadero (°C)

model_par={                     % Parámetros del modelo  (valor nominal, nombre, LaTEX)
{'6000','Cg','$C_{g}$'},        %   Capacidad de calor del aire del invernadero (J m^−2 K^−1)
{'16000','Cm','$C_{m}$'},       %   Capacidad de calor de los materiales sólidos
                                %   del invernadero (J m^−2 K^−1)
{'12.0*sph','Uge','$U_{ge}$'},  %   Coeficiente de transferencia del aire
                                %   del invernadero (W m^−2 K^−1)
{'25.0*sph','Umg','$U_{mg}$'}}; %   Coeficiente de transferencia de los materiales sólidos
                                %   del invernadero (W m^−2 K^−1)

model_def={                     % Definición del modelo
% ECUACION(ES) DE ESTADO DEL MODELO DINÁMICO
%  Primera derivada de la temperatura del aire del invernadero
'dTg = (1/Cg)*(Umg*(Tm-Tg)-Uge*(Tg-Te));',
%  Primera derivada de la temperatura de los materiales sólidos del invernadero
'dTm = (1/Cm)*(-Umg*(Tm-Tg)+I);'
};
```

Figura A1.4. Archivo ***lsag*** de un modelo simple

INTERFACE DE USUARIO

Una vez que se cuenta con los datos de entrada en formato tscollection y la definición del modelo en formato Matlab, archivo con terminación lsag, lo cual se explicó en las secciones previas, es posible generar el código Matlab/Simulink para realizar el análisis de sensibilidad local mediante el programa LSAGs, Local Sensitivity Analysis in Simulink o bien el código

Matlab/Simulink/C mediante el programa LSAGc, Local Sensitivity Analysis in C.

Es importante señalar que los programas LSAGs y LSAGc comparten la misma interface de usuario, ya que solo cambia el nombre del programa mostrado en el encabezado de la misma

Al ejecuta el programa ***LSAGs*** o ***LSAGc*** se muestra la interface de usuario, en pantalla completa, mostrada en la Figura A1.5 y en la cual podemos activar o desactivar las siguientes opciones:

a) Sensibilidad relativa. Esta opción indica si el programa calcula índices de sensibilidad absoluta o relativa, se recomienda utilizar índices de sensibilidad relativa ya que son adimensionales y pueden comparar entre ellos. Esta opción aplica para las gráficas generadas y la tabla resumen de áreas bajo las funciones de sensibilidad, pero el archivo Excel generado siempre contiene los valores de las variables de estado y las funciones de sensibilidad local absolutas.

b) Áreas absolutas. Como las funciones de sensibilidad local de un modelo dinámico varían en el tiempo, es común utilizar el área bajo la curva, o trayectoria, como un índice de sensibilidad local. Dado que las funciones de sensibilidad local se obtienen numéricamente, el área bajo la trayectoria se calcula mediante el comando de Matlab ***trapz***, el cual calcula la integración numérica trapezoidal, de la forma ***trapz(t,y)***. Esto hace que las áreas positivas y negativas se cancelen disminuyendo la sensibilidad total, por lo que es conveniente calcular áreas absolutas mediante el comando ***trapz(t,abs(y))*** lo cual da un mejor estimador total de la sensibilidad, para lo cual el usuario tiene que habilitar esta opción.

c) Gráficas de las variables de entrada. Esta opción determina si se generan las gráficas de las variables de entrada, de habilitarse esta opción se generan gráficas en formato PNG, sin pérdida de información,

a 300 dpi que son las requeridas por la mayoría de las revistas científicas.

d) Gráficas de las variables de estado. Esta opción determina si se generan las gráficas de las variables de estado, con las características descritas en el inciso c.

e) Gráficas de las funciones de sensibilidad local. Esta opción determina si se generan las gráficas de las funciones de sensibilidad local, con las características descritas en el inciso c.

f) Tabla de índices de sensibilidad local. Esta opción determina si se genera una tabla con los índices de sensibilidad local, áreas bajo la curva, y por el momento no se puede deshabilitar.

Al presionar el botón **Load model** pueden ocurrir dos cosas:

a) Si se encuentra activo Simulink se visualiza la ventana mostrada en la Figura A1.6

b) Si no se encuentra activo Simulink se mostrará el mensaje ***Loading Simulink*** y al finalizar se visualiza la ventana mostrada en la Figura A1.6.

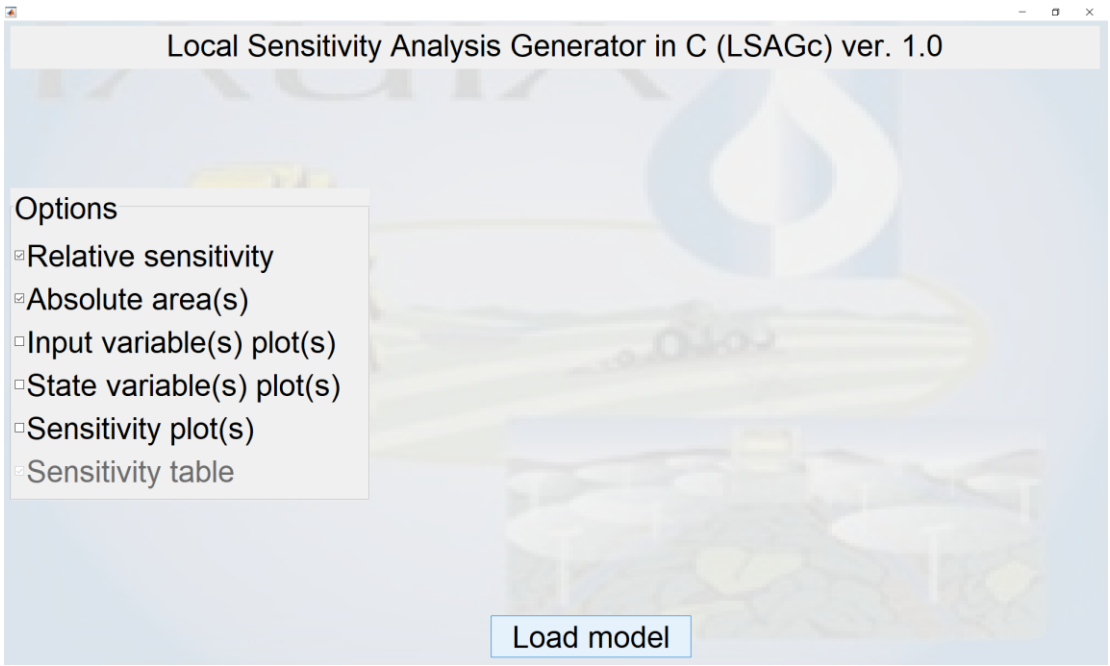

Figura A1.5. Interface de usuario de los programas LSAGs y LSAGc

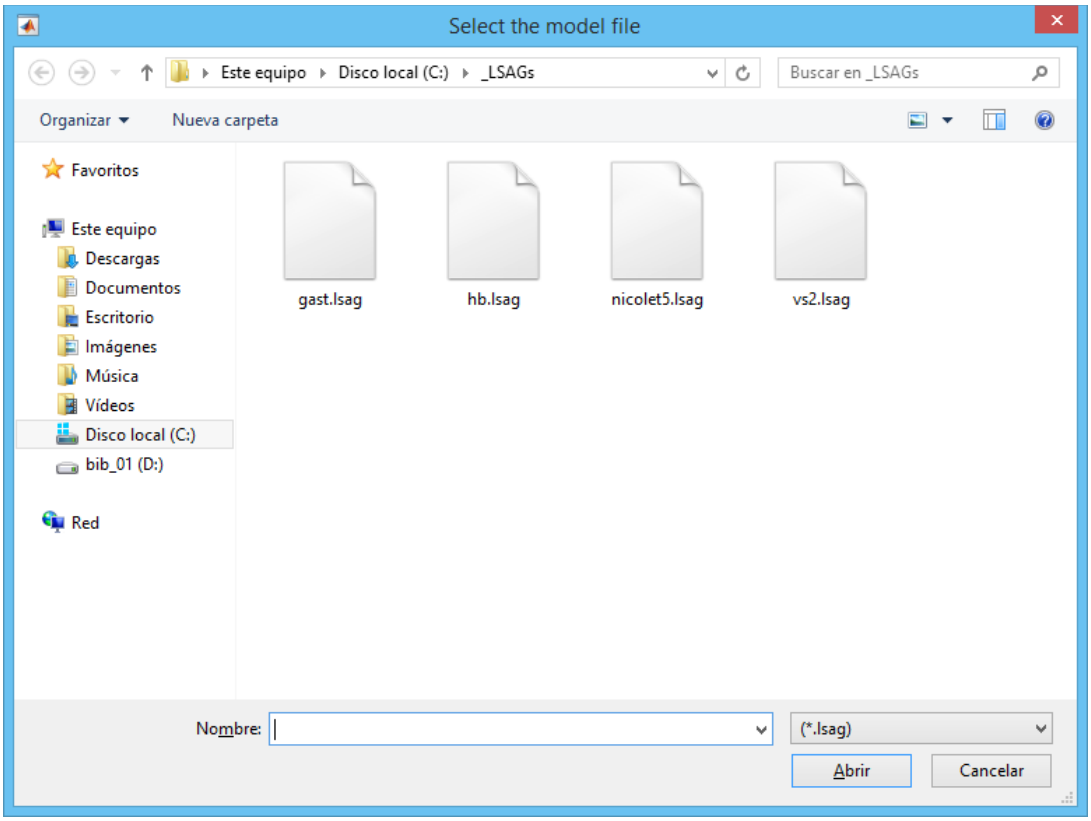

Figura A1.6. Ventana de selección del modelo a procesar

Al seleccionar un modelo y oprimir el botón Abrir, o dar doble click sobre el modelo, se inicia la generación del modelo de análisis de sensibilidad local y al finalizar se muestra el botón mostrado en la Figura 11:

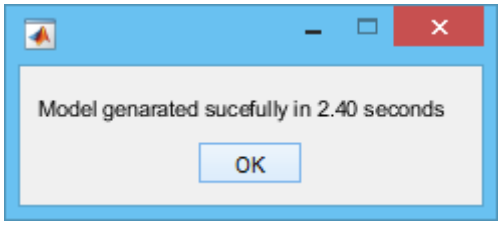

Figura A1.7. Mensaje de fin de proceso

ARCHIVOS GENERADOS POR EL PROGRAMA LSAGS

Como se explicó anteriormente, para utilizar el programa LSAGs, o LSAGc, se requiere el archivo del modelo, con extensión lsag, si el modelo cuenta con variables de salida se requiere de un archivo de datos de tipo tscollection, colección de series de tiempo, con extensión mat.

Después de concluir satisfactoriamente el proceso del programa LSAGs se generan, en el directorio de trabajo, los siguientes archivos:

a) ls_project_name_cf.m: Archivo de código (code file) contiene el código Matlab para realizar el análisis de sensibilidad local.

b) ls_project_name_mf.slx: Archivo de modelo (model file) contiene el modelo Simulink para realizar el análisis de sensibilidad local.

c) ls_project_name_lf. txt: Archivo de bitácora (log file) contiene los mensajes generados por el sistema durante el proceso.

ARCHIVOS GENERADOS POR EL PROGRAMA LSAGC

Después de concluir satisfactoriamente el proceso del programa LSAGc se generan, en el directorio de trabajo, los siguientes archivos:

a) lc_project_name_cf.m: Archivo de código (code file) contiene el código Matlab para realizar el análisis de sensibilidad local.
b) lc_project_name_mf.slx: Archivo de modelo (model file) contiene el modelo Simulink para realizar el análisis de sensibilidad local.
c) lc_project_name_lf.txt: Archivo de bitácora (log file) contiene los mensajes generados por el sistema durante el proceso.
d) lc_project_name_sf.c: Archivo de código fuente en lenguaje C (source file) contiene el código en lenguaje C, para realizar el análisis de sensibilidad local.

Como se puede apreciar se utiliza un prefijo para diferenciar los archivos generados por el programa LSAGs, prefijo ls_, de los generados por el programa LSAGc, prefijo lc_.

Los archivos de código son prácticamente iguales en ambos casos, existiendo solo una pequeña diferencia en la forma de utilizar el comando **sim**.

Los archivos de modelo son muy deferentes ya que el que genera LSAGs contiene toda la lógica para el análisis de sensibilidad local, mientras que el genera LSAGc es solamente un archivo que enlaza el programa compilado. mex file, con el archivo de código, por lo que solamente tiene tres bloques, un puerto de entrada (Inport), un bloque función (S-Function) y un puerto de salida (Outport).

Los archivos de bitácora son exactamente iguales en ambos casos y el archivo de código solamente es generado por el programa LSAGc.

ARCHIVOS GENERADOS EN EL ANÁLISIS DE SENSIBILIDAD LOCAL

Los programas LSAGs y LSAGc no realizan el análisis de sensibilidad local del modelo, solo generan los archivos necesarios para hacerlo. Para realizar dicho análisis hay que ejecutar el archivo de código, y en caso de LSAGc hay que compilar previamente el archivo C mediante el comando mex sorce_file.c lo cual, de contar con un compilador de C compatible con Matlab debidamente configurado, genera un archivo tipo mex (MATLAB executable file).

Al ejecutar el archivo de código se genera un archivo tipo csv (comma separated values) con nombre lc_project_name_sft.csv, el cual contiene una tabla con el resumen de las áreas bajo la curva de las funciones de sensibilidad local, y otro archivo de tipo Excel con nombre lc_project_name.xlsx, el cual contiene una tabla con los valores de las variables de estado y las funciones de sensibilidad local generadas durante la simulación.

De manera opcional el usuario puede seleccionar la generación de gráficas de las variables de entrada, las variables de estado y las funciones de sensibilidad local, mediante los cuadros de selección contenidos un la interface de usuario de los programas LSAGs y LSAGc.

ERRORES MÁS COMUNES AL USAR LOS PROGRAMAS

Durante el desarrollo y prueba de los programas LSAGs y LSAGc, los errores más frecuente fueron:

a) Error en los nombres de las 10 variables contenidas en el archivo del modelo, extensión lsag, lo cual se traduce en el siguiente mensaje de error, **Error: Missing variables (project_name, auxiliary_cts)** y la consiguiente cancelación del proceso. Para evitar este tipo de error se recomienda generar los archivos de modelo. Extensión lsag, a partir de una ya validado y cambiar los valores de las variables, pero no sus nombres.

b) Error en los nombres de las variables de entrada. Los nombres, de las variables de entrada, utilizados en el archivo de modelos, con extensión lsag, deben de coincidir con los nombres utilizados en el archivo de datos, de no ser así se muestra el siguiente mensaje, **Error: data source don´t match with input variables** y se cancela el proceso.

c) Error en el tipo del archivo de datos de entrada. Los programas LSAGs y LSAGc solamente aceptan archivos de datos de tipo tscollection,

extensión mat, de no ser así se muestra el mensaje, **Error: data source is not a tscollection,** cancelándose el proceso.

d) Archivo de datos vacío. Si el archivo de datos es de tipo tscollection pero no tiene registros, se muestra el mensaje, **Error: data source is empty** y se cancela el proceso.

e) Variables de entrada en un modelo que carece de ellas. Si la variable data source se define como none, mediante **data_source='none'**, el arreglo de variables de enterada se debe definir como vacío mediante **input_var={}**, de no ser así se muestra el mensaje, **Error: Not input variables are allowed** y se cancela el proceso.

f) Tiempo de simulación all en un modelo sin variables de entrada. Si un modelo tiene variables de entrada se puede similar en todos los tiempos del archivo de datos, mediante el comando **sim_time='all'** en el archivo de modelo con terminación lsag, pero en un modelo sin variables de entrada y por consiguiente sin archivo de datos, si utilizamos esta opción se muestra el mensaje **Error: Numeric simulation time required** y se cancela el proceso. En los archivos de modelo utilizados en el presente trabajo esta situación es fácil de detectar ya que las variables data_source y sim_time son contiguas y se puede validar que no aparezca esta combinación (data_source='none' y sim_time='all').

g) Caracteres no válidos en el tiempo de simulación: Si la variable sim_time, simulation time, contiene uno o más caracteres no válidos se muestra el mensaje **Error: invalid character(s) in sim_time**, los únicos caracteres permitidos son prefijo +, prefijo -, punto, dos puntos y los dígitos de 0 a 9.

h) Errores de sintaxis en el archivo de modelo. Como se explicó anteriormente el archivo de modelo, extensión lsag, se ejecuta mediante el comando run, a fin de que sea interpretado por Matlab, de

existir un error de sintaxis el mensaje generado es capturado mediante la estructura try, catch de Matlab y mostrado al usuario de la siguiente forma, **Error: wrong syntax in model file (error_message)**, en donde error_message es el error reportado por Matlab al ejecutar el archivo de modelo, uno de los errores más comunes es ***Dimensions of matrices being concatenated are not consistent***, y se genera cuando una de las variables de entrada (input_var), variables de estado (state_var) o parámetros del modelo (model_par) tienen un número de elementos diferente a tres que es lo correcto.

EJEMPLO DE ANALISIS DE SENSIBILIDAD LOCAL

Para validar los programas generados, LSAGs y LSAGc, se utilizó el modelo mostrado en la Figura A1.8, el cuál es una solución del modelo (Van Straten, 2008), pero resuelto con diferentes condiciones iniciales, parámetros y funciones de radiación y temperatura, de las que utilizó el autor en su artículo. Como las funciones de radiación y temperatura utilizadas en este caso no son integrables, el modelo no se puede resolver analíticamente.

Esta solución se tomó del curso IA-748 CE-SIMULACIÓN AVANZADA impartido por el Dr. Irineo López Cruz en el semestre de otoño del año 2016, en el Posgrado de Ingeniería Agrícola y Uso Integral del Agua de la Universidad Autónoma Chapingo.

Al comparar los resultados obtenidos con el programa del curso antes mencionado, con los de los programas LSAGs y LSAGc se observaron algunos hechos interesantes como:

a) El código del curso mezcla la generación de los datos de entrada, con el análisis de sensibilidad local, por lo que es no solo más complejo sino también menos eficiente, ya que los datos de entrada se tienen que generar cada vez que se ejecuta el programa.

b) Al simular 10 intervalos de tiempo, las funciones de generación de los datos de entrada no se ejecutan 10 veces sino miles de veces, dependiendo del criterio de convergencia utilizado en la integración numérica, en nuestro caso con una tolerancia absoluta de 1e-5 se ejecutan 3,116 veces, mientras que con una tolerancia absoluta de 1e-10 se ejecutan 6,272 veces. Lo anterior se debe a que la integración numérica se lleva a cabo mediante un método de tamaño de paso variable (ODE45: Fórmula de Runge-Kutta (4,5) con el método de Dormand y Prince).

c) Lo anterior confirma que los datos obtenidos mediante funciones se deben de generar una sola vez, documentarse y usarse tantas veces como sea necesario, siguiendo la máxima de créalo una vez y úsalo muchas veces (Create once, use many).

d) La integración numérica mediante el método ODE45, requiere que los datos de entrada del modelo se midan a intervalos muy pequeños de tiempo, del orden de 1e-7, por lo que al medir dichas variables, o generarlas mediante fórmulas como en nuestro caso, se debe utilizar el menor tamaño de intervalo posible, para evitar errores por interpolación. En este caso se probaron varios intervalos, y el mayor intervalo que no generó errores de fue de 0.0001 días, aproximadamente cada 9 segundos, si bien en condiciones normales no se pueden registrar datos con este intervalo, se recomienda utilizar el mínimo permitido por los sensores y el equipo de registro utilizados, para minimizar los errores por interpolación.

e) Otro factor que influye en los errores de redondeo es el intervalo de simulación, para el modelo utilizado, con un intervalo de 0.004 días (aproximadamente 6 minutos) se obtuvieron los mismos resultados que con una función continua, con cuatro decimales.

f) De lo anterior se puede concluir que los intervalos óptimos de simulación y de muestreo, para el modelo analizado, varían en un orden de magnitud.

g) Las Tablas A1.1 y A1.2 muestran que las áreas bajo las curvas de las funciones de sensibilidad local, son exactamente las mismas si se calculan con el modelo mostrado en la Figura A1.8, estructura algebraica, o el mostrado en la Figura A1.9, estructura matricial, pero este último se ejecuta en menos de un sexto del tiempo del primero (menos de 2 segundos del matricial contra más de 12 segundos del algebraico).

h) Las áreas bajo las curvas generadas por el modelo de la Figura A1.8 no son absolutas, lo cual se puede deducir por los signos negativos de las Tablas A1.1 y A1.2. Este tipo de valores pueden cancelarse entre sí, lo que subestima los índices de sensibilidad local, por lo que es altamente recomendable utilizar las áreas absolutas bajo las curvas de las funciones de sensibilidad local, para hacer eso en el Modelo de la Figura A1.8 hay que modificar manualmente el código del mismo, pero si se usan los programas desarrollados esto solamente requiere activar la segunda opción de la interface de usuario, Absolute área(s), como se puede apreciar en la Figura A1.5, obteniéndose las áreas mostradas en la Table A1.3, que como se puede apreciar estiman mejor los índices de sensibilidad local.

i) Los resultados obtenidos por los programas LSAGs y LSAGc son exactamente los mismos, y dada la simplicidad del modelo utilizado, el tiempo de procesamiento es prácticamente igual. No obstante lo anterior existen claras diferencias entre ambos programas ya que al trabajar con LSAGc se requiere contar con un compilador de lenguaje C compatible con Matlab y obviamente se requiere un paso adicional en el proceso, que es la compilación del archivo generado, mientras que

al utilizar LSAGs la principal desventaja es que en los bloques de Función de Simulink se pierden los nombres de los identificadores utilizados, ya que se tiene que expresar como entradas mediante el identificador u, como u(1), u(2), etc., lo cual si bien se hace de manera automática si dificulta entender las expresiones.

j) Por lo anterior si un usuario tiene instalado un compilador C compatible con Matlab, es recomendable que utilice el programa LSAGc, siendo mejor utilizar LSAGs en caso contrario.

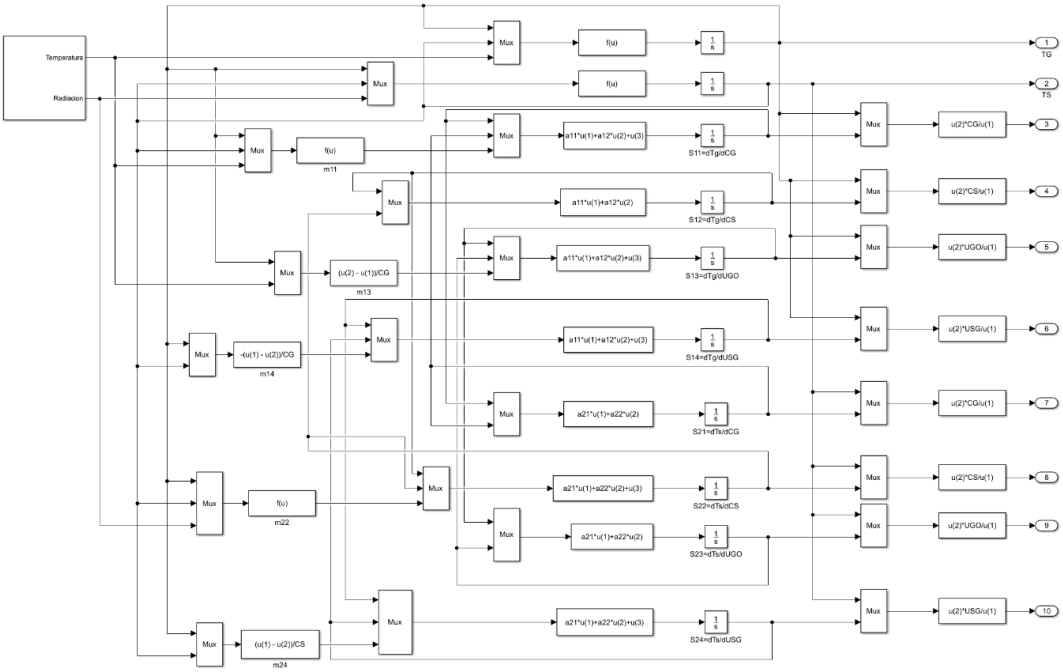

Figura A1.8. Modelo Simulink con estructura algebraica

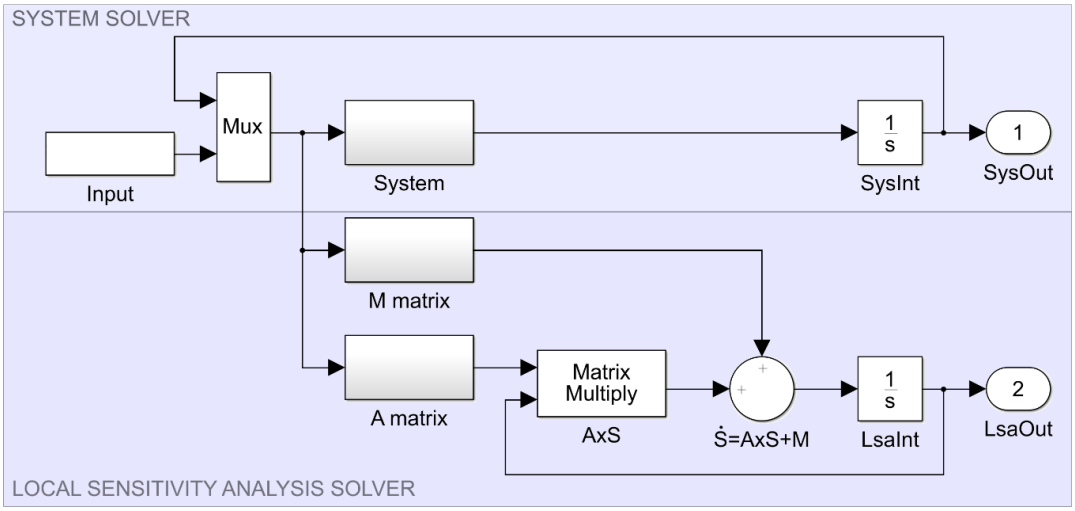

Figura A1.9. Modelo Simulink con estructura matricial

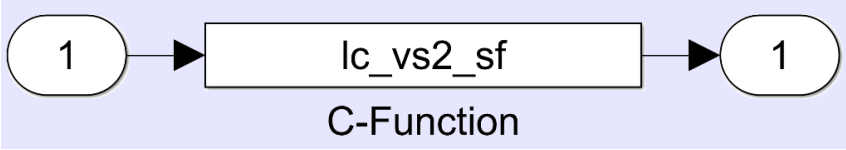

Figura A1.10. Modelo Simulink para enlazar código C

```
% Generated using LSAGs by Felipe Pedraza-Oropeza (fpedraza@colpos.mx)
%**********************************************************************
% Local Sensitivity Analysis in Simulink (LSAGs)
% Model file used: vs2.lsag
% Generation date: 27-Dec-2018
%**********************************************************************

Auxiliary variable(s) in model:
 Not auxiliary variable(s) in model

Searching parameter(s) in model
 All parameters are in the model

Searching missing state variable(s) in function(s) (zeros in A matrix)

Search missing parameter(s) in function(s) (zeros in M matrix)
   1) Parameter Cm missing in function 1
   2) Parameter Cg missing in function 2
   3) Parameter Uge missing in function 2
```

Figura A1.11. Archivo de bitácora

	SENSIBILIDAD RELATIVA	
PARÁMETROS	**Tg**	**Ts**
CG	-0.0042	-0.0066
CS	-0.0110	-0.0472
UGO	0.0227	0.0357
USG	-0.0077	0.0180

Tabla A1.1. Índices de sensibilidad local relativa del modelo con estructura algebraica

	SENSIBILIDAD RELATIVA	
PARÁMETROS	**Tg**	**Ts**
CG	-0.0042	-0.0066
CS	-0.0110	-0.0472
UGO	0.0227	0.0357
USG	-0.0077	0.0180

Tabla A1.2. Índices de sensibilidad local relativa del modelo con estructura matricial

	SENSIBILIDAD RELATIVA	
PARÁMETROS	**Tg**	**Ts**
CG	0.1452	0.1356
CS	0.5251	0.7795
UGO	0.6968	0.6542
USG	0.1742	0.2569

Tabla A1.3. Índices de sensibilidad local relativa (áreas absolutas)

APÉNDICE 2. SOLUCIÓN ANALÍTICA DE UN MODELO LTI

INTRODUCCIÓN

Un sistema LTI (Linear Time Invariant) en espacio de estados se expresa mediante las ecuaciones:

$$\dot{x}(t) = A\,x(t) + B\,u(t)$$

$$y(t) = C\,x(t) + D\,u(t)$$

En donde la primera es la ecuación de estado, la segunda es la ecuación de salida, $\dot{x}(t)$ es el vector de las derivadas con respecto al tiempo de las variables de estado, $x(t)$ es el vector de estados, $u(t)$ es el vector de entrada, $y(t)$ es el vector de salida, A es la matriz de estado, B es la matriz de entrada, C es la matriz de salida, D es la matriz de transmisión directa y su solución analítica es igual a:

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\tau)}B\,u(\tau)d\tau$$

El primer término representa el efecto de las condiciones iniciales y el segundo el efecto de las variables de entrada, es evidente que para poder utilizar esta solución, se requiere que el sistema sea LTI y que los datos de entrada sean funciones integrables, lo cual no ocurre normalmente en Biosistemas, en donde los modelos utilizados son altamente no lineales y los datos de entrada no son funciones integrables sino tablas de observaciones discretas medidas a cierto intervalo de tiempo.

Las limitaciones anteriores explican por qué en Biosistemas se utilizan soluciones numéricas, pero a fin de contar con una validación analítica del sistema implementado se decidió resolver un sistema LTI, con variables de entrada en forma de funciones integrables (Van Straten, 2008), el cual simula la temperatura del aire y de los materiales sólidos de un invernadero y cuya descripción se da a continuación.

DESCRIPCIÓN DEL MODELO

El modelo tiene dos variables de estado:

- Tg: Temperatura del aire del invernadero
- Tm: Temperatura de los materiales sólidos del invernadero

Dos variables de entrada, ver Figura A1.1:

- Te: Temperatura exterior
- I: Radiación exterior

Cuatro parámetros

- Cg: Capacidad de calor del aire del invernadero
- Cm: Capacidad de calor de los materiales sólidos del invernadero
- Uge: Coeficiente de transferencia del aire del invernadero
- Umg: Coeficiente de transferencia de los materiales sólidos del invernadero

Las ecuaciones de estado son:

$$\frac{d(Tg)}{dt} = \frac{1}{Cg}(Umg(Tm - Tg) - Uge(Tg - Te))$$

$$\frac{d(Tm)}{dt} = \frac{1}{Cm}(-Umg(Tm - Tg) + I)$$

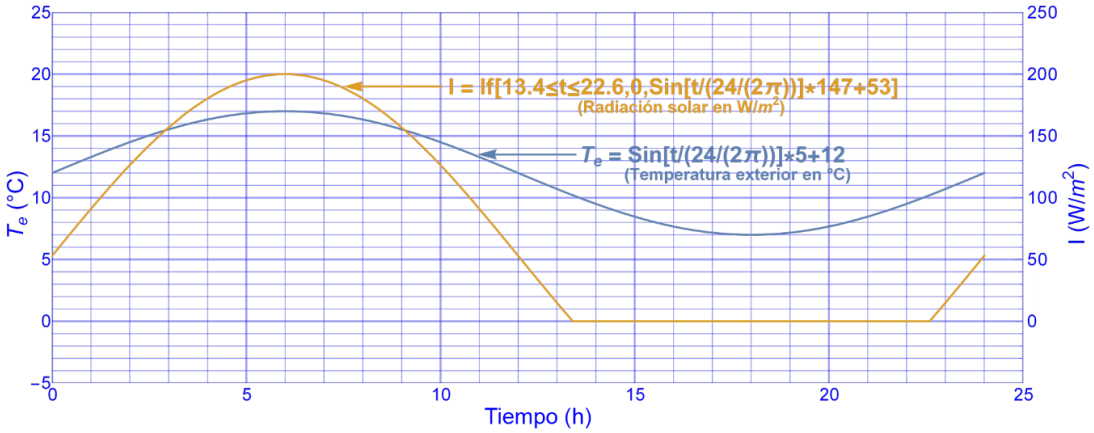

Figura A2.1. Variables de entrada del modelo (Van Straten, 2008)

De la Figura A2.1 se puede ver que el modelo (Van Straten, 2008), solo tiene solución analítica en el rango de tiempo en que las dos funciones son

continuas, y cuyo límite se puede encontrar resolviendo la ecuación sin(t/(24/(2 Pi))*147+53=0 con lo cual obtenemos las raíces cíclicas:

$$t_1 = \frac{12\left(\pi + \sin^{-1}\left(\frac{53}{147}\right) + 2\pi\, n\right)}{\pi}$$

$$t_2 = \frac{12\left(-\sin^{-1}\left(\frac{53}{147}\right) + 2\pi\, n\right)}{\pi}$$

Para cualquier n entero, lo cual para el primer día es igual a:

$$t_1 = \frac{12\left(\pi + \sin^{-1}\left(\frac{53}{147}\right) + 2\pi(0)\right)}{\pi} \approx 13.408908$$

$$t_2 = \frac{12\left(-\sin^{-1}\left(\frac{53}{147}\right) + 2\pi(1)\right)}{\pi} \approx 22.59109186$$

GENERALIDADES DE UN CAS (COMPUTER ALGEBRA SYSTEM)

Un CAS (Computer Algebra System), o Sistema de Álgebra Computacional, es una herramienta informática orientada al procesamiento simbólico de expresiones matemáticas, aunque la mayoría de los programas de este tipo realizan también cálculos matemáticos numéricos.

Entre los CAS más conocidos están:

- Comerciales
  - MATHEMATICA
  - MAPLE
  - MATLAB
- Software libre
  - SAGE (System for Algebra and Geometry Experimentation)
  - MATHICS
  - MAXIMA
  - GNU-OCTAVE
- Calculadoras gráficas
  - TI-92 (Texas Instruments, 1995)
  - Voyage 200 (Texas Instruments, 2002)
  - TI-Nspire™ CAS (Texas Instruments, 2006)
  - HP Prime (Hewlett-Packard, 2013)

Matlab es un CAS orientado a soluciones numéricas, aunque posee un ToolBox (Symbolic Math Toolbox) que le da ciertas capacidades simbólicas, mientras que Mathematica es un CAS orientado a la manipulación simbólica, aunque algunos de sus comandos (los cuales inician con N) realizan cálculos numéricos como NSolve (resuelve, numéricamente, sistemas de ecuaciones algebraicas) y NDSolve (resuelve, numéricamente, sistemas de ecuaciones diferenciales).

Por lo anterior, se utilizó Mathematica para la solución analítica del sistema.

GENERALIDADES DE MATHEMATICA

El desarrollo del CAS Mathematica es conducido por el destacado científico inglés Stephen Wolfram (Londres, 1959) fundador de la compañía Wolfram Research y famoso por escribir su primer artículo científico a los 15 años, recibir su doctorado en física teórica a los 20 años por el Caltech y obtener el premio MacArthur (también conocido como el premio de los genios) a los 22 años.

Algunas características de la sintaxis de Mathematica son:

- Un comando se ejecuta mediante Shift+Enter
- Dos elementos adyacentes en una expresión se consideran como factores (2a=2*a pero a2 es un identificador)
- Los comandos y funciones inician con mayúscula (Sin, Con, Simplify, Pi, etc.)
- La mayoría de las funciones operan en listas
- Los argumentos de las funciones van entre corchetes (Sin[x], Simplify[a a])
- La multiplicación se puede indicar con *
- Las potencias se pueden indicar con el símbolo ^
- Una lista se indica por elementos separados por comas encerrados entre llaves (por ejemplo:{1,2,3,4,5})

- Las ecuaciones utilizan el operador de relación “es igual a” (==) y no el de asignación (=) por ejemplo: 6x^2-5x+1==0
- Los sistemas de ecuaciones se definen como una lista de ecuaciones (por ejemplo: { x + 4y – z – 6 == 0, 2x + 5y - 7z + 9 == 0, 3x - 2y + z -2 ==0}
- Pi denota a la constante matemática π
- El operador -> se denomina regla, dado que genera una regla de conversión en la cual el elemento a la izquierda del operador se transforma en el elemento de la derecha (por ejemplo x->5)
- Las raíces de una ecuación, o sistema de ecuaciones, se devuelven en forma de reglas
- El operador /. (Replace All) evalúa una expresión remplazando todas las ocurrencias de uno o varios elementos indicados por una regla o por una lista e reglas (por ejemplo: 2x+5y+7z /. {x->3, y->5, z->6}
- Un punto y coma al final de una instrucción hace que no se visualice el resultado de la misma, con excepción del comando Print, y sirve para visualizar únicamente los resultados relevantes de un proceso, y como separador para incluir varios comandos en una misma línea.

Algunas de las funciones y comandos utilizados en el presente trabajo son:

ClearAll borra todos los valores, definiciones, atributos, mensajes y valores predeterminados asociados con los símbolos.

DSolve Resuelve una ecuación o sistema de ecuaciones diferenciales.

Export Exporta datos a un archivo, convirtiéndolos al formato correspondiente a la extensión del mismo.

First Devuelve el primer elemento de una expresión

Simplify Simplifica una expresión.

Table Genera una tabla de valores.

MATRICES DEL SISTEMA

Para resolver la ecuación de estados de un sistema LTI necesitamos las matrices A y B, las cuales en Mathematica se determina de la siguiente manera:

a) Se genera una lista con las ecuaciones del sistema a la que llamaremos **eq.**
b) Se genera una lista con las variables de estado a la que llamaremos **sv.**
c) Se genera una lista con las variables de entrada a la que llamaremos **iv**.
d) Se ejecuta el comando **A** = Thread[Simplify[Coefficient[Expand[**eq**], #]] & /@ **sv**], para calcular la matriz **A**.
e) Se ejecuta el comando **B** = Thread[Simplify[Coefficient[Expand[**eq**], #]] & /@ **iv**], para calcular la matriz **B**.

Para nuestro ejemplo el código de Mathematica queda como:

```
(*****************************************************************************)
(* Generación de las matrices A y B del modelo (Van Straten, 2008)           *)
(* Lista de las ecuaciones del modelo*)
eq = {(1/Cg)*(Umg*(Tm - Tg) - Uge*(Tg - Te)), (1/Cm)*(-Umg*(Tm - Tg) + I)};
(* Lista de las variables de estado del modelo*)
sv = {Tg, Tm};
(* Lista de las variables de entrada del modelo*)
iv = {Te, I};
(* Cálculo de las matrices*)
A = Thread [Simplify [Coefficient [Expand [eq], #]] & /@ sv];
B = Thread [Simplify [Coefficient [Expand [eq], #]] & /@ iv];
(* Visualización de las matrices*)
Print [StandardForm["A="], A // MatrixForm, StandardForm[";"]];
["B=", B // MatrixForm, ";"];
Print ["x [t_]=", (#[t] & /@ sv) // MatrixForm, ";"];
Print ["u [t_]=",(#[t]&/@iv)//MatrixForm,";"];
(*****************************************************************************)
```

Y los resultados obtenidos son:

$$A = \begin{pmatrix} -\frac{Uge+Ugm}{Cg} & \frac{Umg}{Cg} \\ \frac{Umg}{Cm} & -\frac{Umg}{Cm} \end{pmatrix}$$

$$B = \begin{pmatrix} \frac{Uge}{Cg} & 0 \\ 0 & \frac{1}{Cm} \end{pmatrix}$$

$$x[t\_] = \begin{pmatrix} Tg[t] \\ Tm[t] \end{pmatrix}$$

$$u[t\_] = \begin{pmatrix} Te[t] \\ I[t] \end{pmatrix}$$

SOLUCIÓN ANALÍTICA DEL SISTEMA

Para resolver una ecuación diferencial, o un sistema de ecuaciones diferenciales, en Mathematica se utiliza el comando **DSolve**, el problema se puede plantear de manera general o si se trata de un modelo LTI con la notación correspondiente como se muestra a continuación.

Código usando el formato general

```
(*************************************************************************)
(* Solución del modelo (Van Straten, 2008) con formato general           *)
(* Se borran las definiciones de las variables                           *)
ClearAll [shp, Cg, Cm, Uge, Umg, Tg0, Tm0];
(* Se definen las funciones de las variables de entrada
        *)
Te [t_] =Sin [t/(24/(2\[Pi]))]*5+12;
I [t_] = (Sin[t/(24/(2\[Pi]))]*147+53)*shp;
(* Se define el lado derecho de la primera ecuación                      *)
dTg = (1/Cg)*(Umg*(Tm[t]-Tg[t])-Uge*(Tg[t]-Te[t]));
(* Se define el lado derecho de la segunda ecuación                      *)
dTm = (1/Cm)*(-Umg*(Tm[t]-Tg[t])+I[t]);
(* Se resuelve el sistema                                                *)
ar=DSolve[{Tg'[t]==dTg,Tm'[t]==dTm,Tg[0]==Tg0,Tm[0]==Tm0},{Tg[t],Tm[t]},t]
(*************************************************************************)
```

Código usando el formato de modelo LTI

```
(*****************************************************************************)
(* Solución del modelo (Van Straten, 2008) con formato LTI                   *)
(* Se borran las definiciones de las variables                               *)
ClearAll [sph, Cg, Cm, Uge, Umg, Tg0, Tm0];
(* Se definen las funciones de las variables de entrada                      *)
Te [t_] =Sin [t/ (24/ (2\ [Pi]))]*5+12;
I [t_] = (Sin [t/ (24/ (2\ [Pi]))]*147+53)*sph;
(* Se definen las matrices A, B, u (t) y x (t)                               *)
```

$$A = \begin{pmatrix} -\frac{Uge+Ugm}{Cg} & \frac{Umg}{Cg} \\ \frac{Umg}{Cm} & -\frac{Umg}{Cm} \end{pmatrix};$$

$$B = \begin{pmatrix} \frac{Uge}{Cg} & 0 \\ 0 & \frac{1}{Cm} \end{pmatrix};$$

$$u[t\_] = \begin{pmatrix} Te[t] \\ I[t] \end{pmatrix};$$

$$x[t\_] = \begin{pmatrix} Tg[t] \\ Tm[t] \end{pmatrix};$$

```
 (* Se resuelve el sistema                                                   *)
ar=DSolve[{x'[t]==A.x[t]+B.u[t],Tg[0]==Tg0,Tm[0]==Tm0},{Tg[t],Tm[t]},t]
(*****************************************************************************)
```

En cualquiera de los casos al final tenemos las funciones analíticas de Tg[t] y Tm[t], en forma de reglas, en la variable ar, analytical root, y continuamos el análisis con el siguiente código de Mathematica, utilizando comentarios para explicar las operaciones realizadas.

```
(* Se obtienen las funciones Tg[t_] y Tm[t_] a partir    del     *)
(* resultado de DSolve, el cual está en forma de reglas        *)
{Tg[t_],Tm[t_]}=First[{Tg[t],Tm[t]}/.ar];

(* Se definen los valores de las constantes auxiliares,        *)
(* parámetros del modelo y          condiciones iniciales en   *)
(* forma de reglas   (cr=change rules)                         *)
cr = {sph->3600, Cg->6000, Cm->16000, Uge->12*sph,
        Umg->25*sph, Tg0->12, Tm0->12};

(* Se calculan las funciones de sensibilidad local,            *)
(* derivando las variables  de estado con respecto             *)
(* a cada parámetro                                            *)
TgCg[t_]=Simplify[D[Tg[t],Cg]//. cr];
TgCm[t_]=Simplify[D[Tg[t],Cm]//. cr];
TgUge[t_]=Simplify[D[Tg[t],Uge]//. cr];
TgUmg[t_]=Simplify[D[Tg[t],Umg]//. cr];
TmCg[t_]=Simplify[D[Tm[t],Cg]//. cr];
TmCm[t_]=Simplify[D[Tm[t],Cm]//. cr];
TmUge[t_]=Simplify[D[Tm[t],Uge]//. cr];
TmUmg[t_]=Simplify[D[Tm[t],Umg]//. cr];

(* Se simplifican las funciones de las variables de estado     *)
Tg[t_]=Simplify[Tg[t]//. cr];
Tm[t_]=Simplify[Tm[t]//. cr];

(* Se genera una tabla con las variables de estado y los       *)
(* funciones de sensibilidad local para diferentes tiempos     *)
(* en los cuales las funciones de temperatura y radiación      *)
(* son continuas (ver Figura A1.1)                             *)
flds={Tg[t],Tm[t],TgCg[t],TmCg[t],TgCm[t],TmCm[t],
        TgUge[t],TmUge[t],TgUmg[t],TmUmg[t]};

tb=Table[First[NumberForm[N[flds],{12,10}]],{t,0,13,1}];

(* Se exporta la tabla generada a un archivo Excel             *)
Export["c:/vs2w.xlsx",tb];
```

La Tabla A2.1 muestra los resultados obtenidos con la solución analítica, la Tabla A2.2 muestra los resultados obtenidos por el programa LSAGs y la Tabla A2.3 muestra los resultados obtenidos por el programa LSAGc.

t	Tg	Tm	Tg_{Cg}	Tm_{Cg}	Tg_{Cm}	Tm_{Cm}	Tg_{Uge}	Tm_{Uge}	Tg_{Umg}	Tm_{Umg}
0	12.00000	12.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1	17.65603	20.08712	-0.00010	-0.00010	-0.00014	-0.00021	-0.00006	-0.00005	0.00001	-0.00002
2	22.30172	26.33345	-0.00010	-0.00010	-0.00014	-0.00021	-0.00014	-0.00013	0.00001	-0.00004
3	26.28830	31.69303	-0.00009	-0.00009	-0.00013	-0.00019	-0.00021	-0.00020	0.00001	-0.00005
4	29.54688	36.07990	-0.00007	-0.00008	-0.00011	-0.00015	-0.00028	-0.00027	0.00000	-0.00007
5	31.89881	39.25492	-0.00005	-0.00006	-0.00008	-0.00011	-0.00033	-0.00032	0.00000	-0.00008
6	33.19313	41.01454	-0.00003	-0.00003	-0.00005	-0.00007	-0.00036	-0.00036	0.00000	-0.00008
7	33.34361	41.24157	0.00000	-0.00001	-0.00001	-0.00001	-0.00038	-0.00038	0.00000	-0.00009
8	32.34044	39.92115	0.00002	0.00002	0.00002	0.00004	-0.00038	-0.00038	0.00000	-0.00009
9	30.25206	37.14337	0.00005	0.00004	0.00006	0.00009	-0.00035	-0.00036	0.00000	-0.00008
10	27.22082	33.09757	0.00007	0.00007	0.00009	0.00013	-0.00032	-0.00032	0.00000	-0.00007
11	23.45329	28.05947	0.00009	0.00008	0.00011	0.00017	-0.00026	-0.00027	-0.00001	-0.00006
12	19.20622	22.37240	0.00010	0.00010	0.00013	0.00019	-0.00020	-0.00021	-0.00001	-0.00004
13	14.76906	16.42394	0.00010	0.00010	0.00014	0.00020	-0.00013	-0.00014	-0.00001	-0.00003

Tabla A2.1 Resultados obtenidos mediante la solución analítica

t	Tg	Tm	Tg_{Cg}	Tm_{Cg}	Tg_{Cm}	Tm_{Cm}	Tg_{Uge}	Tm_{Uge}	Tg_{Umg}	Tm_{Umg}
0	12.00000	12.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1	17.65603	20.08712	-0.00010	-0.00010	-0.00014	-0.00021	-0.00006	-0.00005	0.00001	-0.00002
2	22.30172	26.33345	-0.00010	-0.00010	-0.00014	-0.00021	-0.00014	-0.00013	0.00001	-0.00004
3	26.28830	31.69303	-0.00009	-0.00009	-0.00013	-0.00019	-0.00021	-0.00020	0.00001	-0.00005
4	29.54688	36.07990	-0.00007	-0.00008	-0.00011	-0.00015	-0.00028	-0.00027	0.00000	-0.00007
5	31.89881	39.25492	-0.00005	-0.00006	-0.00008	-0.00011	-0.00033	-0.00032	0.00000	-0.00008
6	33.19313	41.01454	-0.00003	-0.00003	-0.00005	-0.00007	-0.00036	-0.00036	0.00000	-0.00008
7	33.34361	41.24157	0.00000	-0.00001	-0.00001	-0.00001	-0.00038	-0.00038	0.00000	-0.00009
8	32.34044	39.92115	0.00002	0.00002	0.00002	0.00004	-0.00038	-0.00038	0.00000	-0.00009
9	30.25206	37.14337	0.00005	0.00004	0.00006	0.00009	-0.00035	-0.00036	0.00000	-0.00008
10	27.22082	33.09757	0.00007	0.00007	0.00009	0.00013	-0.00032	-0.00032	0.00000	-0.00007
11	23.45329	28.05947	0.00009	0.00008	0.00011	0.00017	-0.00026	-0.00027	-0.00001	-0.00006
12	19.20622	22.37240	0.00010	0.00010	0.00013	0.00019	-0.00020	-0.00021	-0.00001	-0.00004
13	14.76906	16.42394	0.00010	0.00010	0.00014	0.00020	-0.00013	-0.00014	-0.00001	-0.00003

Tabla A2.2 Resultados obtenidos mediante el programa LSAGs

t	Tg	Tm	Tg_{Cg}	Tm_{Cg}	Tg_{Cm}	Tm_{Cm}	Tg_{Uge}	Tm_{Uge}	Tg_{Umg}	Tm_{Umg}
0	12.00000	12.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1	17.65603	20.08712	-0.00010	-0.00010	-0.00014	-0.00021	-0.00006	-0.00005	0.00001	-0.00002
2	22.30172	26.33345	-0.00010	-0.00010	-0.00014	-0.00021	-0.00014	-0.00013	0.00001	-0.00004
3	26.28830	31.69303	-0.00009	-0.00009	-0.00013	-0.00019	-0.00021	-0.00020	0.00001	-0.00005
4	29.54688	36.07990	-0.00007	-0.00008	-0.00011	-0.00015	-0.00028	-0.00027	0.00000	-0.00007
5	31.89881	39.25492	-0.00005	-0.00006	-0.00008	-0.00011	-0.00033	-0.00032	0.00000	-0.00008
6	33.19313	41.01454	-0.00003	-0.00003	-0.00005	-0.00007	-0.00036	-0.00036	0.00000	-0.00008
7	33.34361	41.24157	0.00000	-0.00001	-0.00001	-0.00001	-0.00038	-0.00038	0.00000	-0.00009
8	32.34044	39.92115	0.00002	0.00002	0.00002	0.00004	-0.00038	-0.00038	0.00000	-0.00009
9	30.25206	37.14337	0.00005	0.00004	0.00006	0.00009	-0.00035	-0.00036	0.00000	-0.00008
10	27.22082	33.09757	0.00007	0.00007	0.00009	0.00013	-0.00032	-0.00032	0.00000	-0.00007
11	23.45329	28.05947	0.00009	0.00008	0.00011	0.00017	-0.00026	-0.00027	-0.00001	-0.00006
12	19.20622	22.37240	0.00010	0.00010	0.00013	0.00019	-0.00020	-0.00021	-0.00001	-0.00004
13	14.76906	16.42394	0.00010	0.00010	0.00014	0.00020	-0.00013	-0.00014	-0.00001	-0.00003

Tabla A2.3 Resultados obtenidos mediante el programa LSAGc

Como se puede apreciar en las tablas anteriores los resultados obtenidos son iguales, al menos con cinco decimales, por lo que se procedió a calcular las estadísticas de sus diferencias absolutas, obteniéndose lo siguiente:

a) Diferencia entre los resultados analíticos y los del programa LSAGs. Diferencia mínima=0, diferencia máxima=2.8463E-08 y diferencia promedio=1.90288E-09.

b) Diferencia entre los resultados analíticos y los del programa LSAGc. Diferencia mínima=0, diferencia máxima=2.8463E-08 y diferencia promedio=1.90288E-09.

c) Diferencia entre los resultados del programa LSAGs y LSAGc. Diferencia mínima=0, diferencia máxima=1.42109E-14 y diferencia promedio=1.15477E-15.

Con base en lo anterior se puede concluir que no hay ninguna diferencia entre los resultados obtenidos por los programas LSAGs y LSAGc por lo que se pueden utilizar de manera indistinta, ya que sus diferencias están dentro de la

precisión de las variables de doble precisión utilizadas que es de 15 dígitos decimales.

La conclusión más importante es que los programas generados, LSAGs y LSAGc, funcionan correctamente dado que sus resultados concuerdan con los que se obtendrían de manera analítica, la cual es sin duda la prueba más contundente a la que se pueden someter este tipo de programas computacionales.

Podemos concluir también, que la diferencia entre la doble precisión y la precisión de los programas generados, LSAGs y LSAGc, que es aproximadamente de 1E-6 se debe al uso del enfoque numérico que utilizan ambos programas, en lo que respecta al modelo analizado.

APÉNDICE 3. SIMPLIFICACIÓN CON MATLAB Y MATHEMATICA

INTRODUCCIÓN

Matlab es un CAS (Computer Algebra System), o Sistema de Álgebra Computacional, orientados al procesamiento numérico y solo cuenta con una caja de herramientas, código externo, para el procesamiento simbólico Symbolic Math Toolbox.

Por otro lado Mathematica es un CAS orientado al procesamiento simbólico y cuenta con dos comandos para simplificar expresiones Simplify que genera resultados similares a los del Symbolic Math Toolbox y FullSimplify con el que se obtienen resultados más compactos y eficientes como se muestra a continuación.

A manera de ejemplo expandimos los lados derechos de las 5 ecuaciones de estado del modelo NICOLET (Seginer, 2003), este modelo se define mediante variables auxiliares de la siguiente manera:

```
G = Lambda*Piv + BetaN*r + BetaC*(1 + Theta)
GCv = (BetaC*MCv)/(Lambda*Piv*MCs)
fs = 1 - exp(-a*MCs)
pI = (Epsilon*PAR*Sigma*CO2)/(Epsilon*PAR + Sigma*CO2)

hp = 1/(1 + ((1 - bp)/(1 - GCv))^sp)
hg = 1/(1 + (bg/GCv)^sg)
Avs = hg
Ap = hp + (1 - hp)*Xi
Ave = (1 - hp)*Xi
Aev = (1 - hg)*Xi
eTa = k*exp(c*(TEMP - Tstar))
gTa = Nu*eTa
FCp = pI*fs*Ap
FCvs = gTa*fs*Avs
FCg = Theta*FCvs
FCm = eTa*fs
FCve = pI*fs*Ave
FCev = gTa*fs*Aev*(1 + Theta)
FNu = (G*FCvs - BetaC*(FCp - FCm - FCve + FCev))/BetaN
```

FNvs = r*FCvs

dCV = FCp - FCvs - FCg - FCm - FCve + FCev
dCS = FCvs
dCE = FCve - FCev
dNV = FNu - FNvs
dNS = FNvs

En este modelo las 5 variables de estado son: Carbono en vacuolas (CV), Carbono estructural (CS), Carbono en excedencia (CE), Nitrógeno en vacuolas (NV) y Nitrógeno estructural (NS).

Mediante la notación anterior, las ecuaciones de estado lucen engañosamente simples, pero al expandirlas quedan de la siguiente manera:

MODELO OBTENIDO CON MATLAB

dCVm=k*Exp[TEMP*c-MCs*a-Tstar*c]-k*Exp[c*(TEMP-Tstar)]+Nu*Xi*k*Exp[c*(TEMP-Tstar)]-(Nu*k*Exp[c*(TEMP-Tstar)])/(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1)-Nu*Xi*k*Exp[TEMP*c-MCs*a-Tstar*c]+(Nu*k*Exp[TEMP*c-MCs*a-Tstar*c])/(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1)+Nu*Theta*Xi*k*Exp[c*(TEMP-Tstar)]-(Nu*Theta*k*Exp[c*(TEMP-Tstar)])/(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1)-(Nu*Xi*k*Exp[c*(TEMP-Tstar)])/(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1)-Nu*Theta*Xi*k*Exp[TEMP*c-MCs*a-Tstar*c]+(Nu*Theta*k*Exp[TEMP*c-MCs*a-Tstar*c])/(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1)+(Nu*Xi*k*Exp[TEMP*c-MCs*a-Tstar*c])/(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1)+(Nu*Theta*Xi*k*Exp[TEMP*c-MCs*a-Tstar*c])/(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1)+(CO2*Epsilon*PAR*Sigma)/((((Lambda*MCs*Piv*(bp-1))/(BetaC*MCv-Lambda*MCs*Piv))^sp+1)*(Epsilon*PAR+CO2*Sigma))-(Nu*Theta*Xi*k*Exp[c*(TEMP-Tstar)])/(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1)-(CO2*Epsilon*PAR*Sigma*Exp[-MCs*a])/((((Lambda*MCs*Piv*(bp-1))/(BetaC*MCv-Lambda*MCs*Piv))^sp+1)*(Epsilon*PAR+CO2*Sigma));

dCSm=-(Nu*k*Exp[-MCs*a-Tstar*c]*(Exp[TEMP*c]-Exp[MCs*a+TEMP*c]))/(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1);

dCEm=Nu*Xi*k*Exp[TEMP*c-MCs*a-Tstar*c]-Nu*Xi*k*Exp[c*(TEMP-Tstar)]-Nu*Theta*Xi*k*Exp[c*(TEMP-Tstar)]+(Nu*Xi*k*Exp[c*(TEMP-Tstar)])/(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1)+Nu*Theta*Xi*k*Exp[TEMP*c-MCs*a-Tstar*c]-(Nu*Xi*k*Exp[TEMP*c-MCs*a-Tstar*c])/(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1)-(Nu*Theta*Xi*k*Exp[TEMP*c-MCs*a-Tstar*c])/(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1)+(CO2*Epsilon*PAR*Sigma*Xi)/(Epsilon*PAR+CO2*Sigma)+(Nu*Theta*Xi*k*Exp[c*(TEMP-Tstar)])/(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1)-(CO2*Epsilon*PAR*Sigma*Xi*Exp[-MCs*a])/(Epsilon*PAR+CO2*Sigma)-(CO2*Epsilon*PAR*Sigma*Xi)/((((Lambda*MCs*Piv*(bp-1))/(BetaC*MCv-Lambda*MCs*Piv))^sp+1)*(Epsilon*PAR+CO2*Sigma))+(CO2*Epsilon*PAR*Sigma*Xi*Exp[-MCs*a])/((((Lambda*MCs*Piv*(bp-1))/(BetaC*MCv-Lambda*MCs*Piv))^sp+1)*(Epsilon*PAR+CO2*Sigma));

dNVm=(BetaC*k*Exp[c*(TEMP-Tstar)])/BetaN-(BetaC*k*Exp[TEMP*c-MCs*a-Tstar*c])/BetaN+(BetaC*Nu*k*Exp[c*(TEMP-Tstar)])/(BetaN*(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1))+(BetaC*Nu*Xi*k*Exp[TEMP*c-MCs*a-Tstar*c])/BetaN-(BetaC*Nu*k*Exp[TEMP*c-MCs*a-Tstar*c])/(BetaN*(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1))-(BetaC*Nu*Xi*k*Exp[c*(TEMP-Tstar)])/BetaN-(BetaC*Nu*Theta*Xi*k*Exp[c*(TEMP-Tstar)])/BetaN+(BetaC*Nu*Theta*k*Exp[c*(TEMP-Tstar)])/(BetaN*(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1))+(BetaC*Nu*Xi*k*Exp[c*(TEMP-Tstar)])/(BetaN*(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1))+(Lambda*Nu*Piv*k*Exp[c*(TEMP-Tstar)])/(BetaN*(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1))+(BetaC*Nu*Theta*Xi*k*Exp[TEMP*c-MCs*a-Tstar*c])/BetaN-(BetaC*Nu*Theta*k*Exp[TEMP*c-MCs*a-Tstar*c])/(BetaN*(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1))-(BetaC*Nu*Xi*k*Exp[TEMP*c-MCs*a-Tstar*c])/(BetaN*(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1))-(Lambda*Nu*Piv*k*Exp[TEMP*c-MCs*a-Tstar*c])/(BetaN*(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1))-(BetaC*Nu*Theta*Xi*k*Exp[TEMP*c-MCs*a-Tstar*c])/(BetaN*(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1))-

(BetaC*CO2*Epsilon*PAR*Sigma)/(BetaN*(((Lambda*MCs*Piv*(bp-1))/(BetaC*MCv-Lambda*MCs*Piv))^sp+1)*(Epsilon*PAR+CO2*Sigma))+(BetaC*Nu*Theta*Xi*k*Exp[c*(TEMP-Tstar)])/(BetaN*(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1))+(BetaC*CO2*Epsilon*PAR*Sigma*Exp[-MCs*a])/(BetaN*(((Lambda*MCs*Piv*(bp-1))/(BetaC*MCv-Lambda*MCs*Piv))^sp+1)*(Epsilon*PAR+CO2*Sigma));

dNSm=-(Nu*k*r*Exp[-MCs*a-Tstar*c]*(Exp[TEMP*c]-Exp[MCs*a+TEMP*c]))/(((Lambda*MCs*Piv*bg)/(BetaC*MCv))^sg+1);

MODELO OBTENIDO CON MATHEMATICA

dCVw=(CO2*Epsilon*PAR*Sigma)/((1+(((-1+bp)*Lambda*MCs*Piv)/(BetaC*MCv-Lambda*MCs*Piv))^sp)*(Epsilon*PAR+CO2*Sigma))-(CO2*Epsilon*PAR*Sigma)/(Exp[a*MCs]*(1+(((-1+bp)*Lambda*MCs*Piv)/(BetaC*MCv-Lambda*MCs*Piv))^sp)*(Epsilon*PAR+CO2*Sigma))+Exp[c*(TEMP-Tstar)]*k*(-1+Nu*Xi+Nu*Theta*Xi-(Nu*(1+Theta)*(1+Xi))/(1+((bg*Lambda*MCs*Piv)/(BetaC*MCv))^sg))+(Exp[-(a*MCs)+c*TEMP-c*Tstar]*k*(1+Nu+Nu*Theta+((bg*Lambda*MCs*Piv)/(BetaC*MCv))^sg*(1-Nu*(1+Theta)*Xi)))/(1+((bg*Lambda*MCs*Piv)/(BetaC*MCv))^sg);

dCSw=(Exp[-(a*MCs)+c*TEMP-c*Tstar]*(-1+Exp[a*MCs])*k*Nu)/(1+((bg*Lambda*MCs*Piv)/(BetaC*MCv))^sg);

dCEw=((CO2*Epsilon*PAR*(1-1/(1+(((-1+bp)*Lambda*MCs*Piv)/(BetaC*MCv-Lambda*MCs*Piv))^sp))*Sigma)/(Epsilon*PAR+CO2*Sigma)+(CO2*Epsilon*PAR*(-1+1/(1+(((-1+bp)*Lambda*MCs*Piv)/(BetaC*MCv-Lambda*MCs*Piv))^sp))*Sigma)/(Exp[a*MCs]*(Epsilon*PAR+CO2*Sigma))-(Exp[c*(TEMP-Tstar)]*k*Nu*((bg*Lambda*MCs*Piv)/(BetaC*MCv))^sg*(1+Theta))/(1+((bg*Lambda*MCs*Piv)/(BetaC*MCv))^sg)+(Exp[-(a*MCs)+c*TEMP-c*Tstar]*k*Nu*((bg*Lambda*MCs*Piv)/(BetaC*MCv))^sg*(1+Theta))/(1+((bg*Lambda*MCs*Piv)/(BetaC*MCv))^sg))*Xi;

dNVw=((-1+Exp[a*MCs])*(-((BetaC*CO2*Epsilon*PAR*Sigma)/((1+(((-1+bp)*Lambda*MCs*Piv)/(BetaC*MCv-Lambda*MCs*Piv))^sp)*(Epsilon*PAR+CO2*Sigma)))+(Exp[c*(TEMP-Tstar)]*k*(Lambda*Nu*Piv+BetaC*(1+Nu+Nu*Theta+((bg*Lambda*MCs*Piv)/(BetaC*MCv))^sg*(1-Nu*(1+Theta)*Xi))))/(1+((bg*Lambda*MCs*Piv)/(BetaC*MCv))^sg)))/(BetaN*Exp[a*MCs]);

dNSw=(Exp[-(a*MCs)+c*TEMP-c*Tstar]*(-1+Exp[a*MCs])*k*Nu*r)/(1+((bg*Lambda*MCs*Piv)/(BetaC*MCv))^sg);

En este modelo las 5 variables de estado son: Carbono en vacuolas (CV), Carbono estructural (CS), Carbono en excedencia (CE), Nitrógeno en vacuolas (NV) y Nitrógeno estructural (NS).

La diferencia es notable como se puede apreciar en la siguiente tabla:

	TAMAÑO		DIFERENCIA	
DERIVADA	MATLAB	MATHEMATICA	CARACTERES	PORCENTAJE
dCV	1,021	487	534	47.70
dCS	100	92	8	92.00
dCE	795	487	308	61.26
dNV	1,421	318	1,103	22.38
dNS	102	94	8	92.16
TOTAL	3,439	1,478	1,961	42.98

Tabla A3.1 Comparación de los resultados en Matlab y Mathematica

La diferencia es tan significativa que se procedió a hacer una validación, convirtiendo las expresiones anteriores a la sintaxis de Mathematica, mediante una rutina de conversión, y se generó un notebook, archivo de Mathematica, con las expresiones obtenidas y el comando:

FullSimplify [{dCVw-dCVm, dCSw - dCSm, dCEw – dCEm, dNVw - dNVm, dNSw - dNSm}]

Para probar que dos expresiones matemáticas son equivalentes utilizando un CAS, se simplifica la diferencia entre las mismas, si el resultado es cero las expresiones son equivalentes y en caso contrario hay dos posibilidades, que las expresiones no sean equivalentes o que sean demasiado complejas para ser simplificadas correctamente, es por eso que Mathematica es una herramienta efectiva ya que cuenta con el comando FullSimplify el cuál puede simplificar correctamente expresiones complejas.

Dado que el resultado obtenido fue una lista de cinco ceros, concluimos que las expresiones son equivalentes, no obstante que el tamaño total de las expresiones en Mathematica es de solo el 42.98% con respecto a Matlab y en particular llama la atención la derivada del Nitrógeno en vacuolas, dNV, la cual

tiene la quinta parte de tamaño en Mathematica con respecto a Matlab, como se puede apreciar en la Tabla 1.