



DEPARTAMENTO DE INGENIERÍA
DE SISTEMAS Y AUTOMÁTICA

UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS INDUSTRIALES

Máster en Automática e Informática Industrial

TRABAJO FIN DE MÁSTER

REDUCCIÓN SIMBÓLICA DE MODELOS
DINÁMICOS EN SISTEMAS BIOLÓGICOS

Autor: Pablo Carrillo Carrasco

Tutor: Dr. Jesús Picó Marco
Departamento de Ingeniería de
Sistemas y Automática

Tutor: Dr. José Luis Navarro
Herrero
Departamento de Ingeniería de
Sistemas y Automática

Valencia, Septiembre 2019

*En las adversidades sale a
la luz la virtud.*

Aristóteles

Agradecimientos

Quiero agradecer en primer lugar a los compañeros que he tenido el placer de conocer este año, por hacer más fácil y ameno este curso de trabajo y adversidades. A mis tutores Jesús y Jose Luís por brindarme la oportunidad de realizar este TFM, con cierto carácter investigador ofreciendo una salida a la norma, y que permite ampliar capacidades y visión con respecto a la ingeniería.

Por último, y como siempre, a mi familia por estar siempre ahí y darme todas las oportunidades posibles.

Resumen

Este Trabajo Fin de Máster tiene como objetivo el desarrollo e implementación *software* de un algoritmo para reducción de modelos dinámicos como herramienta para el análisis y diseño en Biología Sintética, propuesto en el último artículo de Ayush Pandey y Richard M. Murray, ambos investigadores en el Caltech de California. Para una formalización del *software*, se diseña una GUI (*Guide User Interface*) en Matlab que incluya el algoritmo como parte de una herramienta general enfocada a la ayuda para la investigación.

Se plantea en primer lugar un breve estudio cronológico sobre campos relativos y afines a este proyecto y los fundamentos técnicos que los avalan, plasmados en los capítulos de Estado del Arte y Fundamentos del proyecto.

Continúa con el desarrollo del proyecto *per sé*, cerrando con pruebas y resultados, y unas conclusiones generales.

Palabras clave: *Biología Sintética, Circuitos Biológicos, Reducción Modelos, Algoritmo*

Abstract

This Master's Degree Final Project aims to design and develop software around a reduction algorithm tool for design and analysis on Synthetic Biology, proposed by Caltech researchers Ayush Pandey and Richard M. Murray in their last article. A Guide User Interface (GUI) developed with Matlab completes the software, building a tool focused on research projects.

At first, a chronological study of related fields along technical knowledge is written covering the State of Art and Project Fundamentals chapters.

Then, the proper software is designed and developed. Finally, some tests and results are shown, ending with a general conclusion.

Keywords: *Synthetic Biology, Biological Circuits, Model Reduction, Algorithm*

Índice general

Agradecimientos	III
Resumen	V
Abstract	VII
Índice de siglas	XIII
1. Introducción	1
1.1. Motivación del proyecto	1
1.2. Objetivos	1
1.3. Materiales utilizados	2
1.4. Estructura del documento	2
2. Estado del arte	3
2.1. Biología sintética	3
2.1.1. La Expresión Genética Constitutiva	4
2.1.2. Parámetros de ajuste en la Expresión Genética	8
2.2. Modelos dinámicos en procesos bioquímicos	11
2.2.1. Modelado de la Expresión Genética Constitutiva	11
2.3. Teoría de grafos	15
2.3.1. Estructuras de datos	15
2.3.2. Aplicación a circuitos biológicos	17
3. Fundamentos del proyecto	19
3.1. Reducción de modelos	19
3.1.1. Teoría de la Perturbación Singular	20
3.1.2. Aproximación del Estado Estacionario	20

4. Desarrollo del proyecto	23
4.1. Estudio de la herramienta propuesta	23
4.1.1. Introducción	23
4.1.2. Planteamiento y formulación	25
4.1.3. Algoritmo	28
4.2. Implementación software	30
4.2.1. Modelado dinámico	30
4.2.2. Algoritmo de reducción	34
4.2.3. Grafos	36
4.3. Interfaz de usuario	37
4.3.1. Herramienta GUIDE	38
4.3.2. Diseño de la GUI	39
4.3.3. Desarrollo	40
5. Pruebas y resultados	49
5.1. Dinámica de funcionamiento	49
5.1.1. Selección de sistema de reacciones o creación del mismo	49
5.1.2. Reducción del sistema	50
5.1.3. Dibujo del grafo del sistema	51
5.2. Resultados	52
6. Conclusiones	55
6.1. Conclusión	55
6.2. Trabajos futuros	56
Bibliografía	57
Anexo 1: Fé de erratas en el artículo principal	59

Índice de figuras

2.1. La bacteria <i>Escherichia coli</i> , una de las especies más trabajadas en biología sintética	4
2.2. Transformaciones metabólicas: catabolismo y anabolismo	5
2.3. Interacción entre genoma, proteoma y metaboloma . .	6
2.4. Dogma Central en biología molecular: transcripción y traducción	7
2.5. Transcripción y traducción en una célula procariota . .	8
2.6. Ejemplo de grafo no orientado	15
2.7. Vértices, aristas y matriz de adyacencia de un grafo . .	16
2.8. Ejemplo de grafo orientado en sistemas biológicos . . .	17
4.1. Ejemplo de modelado de la cinética masa-acción de un sistema	24
4.2. Ejemplo de tabla de reacciones de sistema biológico, archivo <i>.csv</i>	30
4.3. Celda con separación de componentes del sistema por cada columna	32
4.4. Ejemplo de interfaz de usuario guiada	37
4.5. IDE de desarrollo de GUIDE	39
4.6. Diseño de la herramienta principal y declaración de los <i>push buttons</i>	39
4.7. Implementación de la herramienta principal en la IDE .	41
4.8. Hilo de funcionamiento al ser ejecutada una interfaz . .	41
4.9. Resultado de implementación de la interfaz de <i>Reaction Editor</i>	44
4.10. Interfaz de <i>Reduction Editor</i>	45
4.11. Ventana emergente con las funciones del modelo reducido	46
4.12. Ventana emergente con la dinámica de las sensibilidades	47
5.1. Presentación de la herramienta principal	49

5.2. Creación de reacciones con <i>React Editor</i>	50
5.3. Funciones de <i>Reduction Editor</i>	50
5.4. Ventana emergente con las funciones del modelo reducido	51
5.5. Ventana emergente con la dinámica de la sensibilidad del sistema	51
5.6. Ejemplo de grafo dibujado	52

Índice de siglas

ADN	Ácido Desoxirribonucleico
ARN	Ácido Ribonucleico
CSV	Comma-Separated Values
GUI	Guide User Interface
GUIDE	Guide User Interface Development Enviroment
IDE	Integrated Development Enviroment
mRNA	ARN mensajero
RBS	Ribosome Binding Site
RNAp	ARN polimerasa
SBML	Systems Biology Markup Language
QSSA	Quasi-Steady State Approximation

Capítulo 1

Introducción

1.1. Motivación del proyecto

Dentro de la Ingeniería Industrial muchos proyectos se llevan a cabo con un fin, valga la redundancia, industrial, en el sentido productivo de la palabra, olvidando a veces el camino investigador, aplicado pero más cercano a la ciencia.

En un punto de convergencia entre la bioquímica y la ingeniería, se puede situar este trabajo, un proyecto de integración software como ayuda para el desarrollo de una investigación a mayor escala, uniendo disciplinas poco menos que interesantes.

1.2. Objetivos

El objetivo principal del proyecto es el desarrollo software de una interfaz de usuario guiada (GUI) con el fin de automatizar la reducción de modelos en circuitos sintéticos biológicos. De forma más concreta:

- Estudio de la herramienta propuesta por Ayush Pandey y Richard M. Murray en su reciente artículo [9].
- Implementación software del algoritmo de reducción de sistemas desarrollado en el artículo previo.
- Desarrollo de la GUI como herramienta de trabajo para sistemas biológicos, integrando el algoritmo de reducción estudiado y proporcionando control paramétrico al usuario.

1.3. Materiales utilizados

Para este proyecto de carácter más investigador se han utilizado:

- Matlab R2018b.
- Ordenador personal.

1.4. Estructura del documento

A continuación y para facilitar la lectura del documento, se explica mínimamente el contenido de cada capítulo.

- En el capítulo 1 se realiza una introducción al proyecto, explicando la motivación, los objetivos y estructura del mismo.
- En el capítulo 2 se expone una pequeña investigación sobre proyectos con objetivos y campos de desarrollo afines a este proyecto.
- En el capítulo 3 se desarrolla de forma sintetizada los fundamentos teóricos que engloban el proyecto.
- En el capítulo 4 se explica detenidamente el desarrollo práctico llevado a cabo en el proyecto.
- En el capítulo 5 se definen los resultados obtenidos y discusión de estos.
- En el capítulo 6 se cierra el proyecto plasmando unas conclusiones con visión personal del mismo y líneas futuras de trabajo.

Capítulo 2

Estado del arte

En este capítulo se expone el recorrido académico y teórico desarrollado en biología sintética, circuitos biológicos y modelado de los procesos bioquímicos.

2.1. Biología sintética

La biología sintética es un campo de estudio aplicado que aporta un marco conceptual a los nuevos sistemas biológicos de ingeniería, basado en los principios de estandarización, modularidad y abstracción de la propia ingeniería. Nuevas funciones extienden la capacidad natural de las células, con aplicaciones potenciales en campos como en energía, agricultura, salud o fabricación [1].

A pesar de esta breve introducción hoy en día no existe un consenso total sobre la definición de biología sintética. Sin embargo, el uso de herramientas y técnicas de biología molecular para realizar *forward-engineering* en el comportamiento celular ha emergido como una identificación generalizada del campo, habiéndose desarrollado un conjunto de enfoques de ingeniería comunes junto a la multiculturalidad de la comunidad [2].

Gran parte del trabajo base fundamental desarrollado en este campo ha sido llevado a cabo con modelos ejemplo de las especies microbiológicas *Escherichia coli* (figura 2.1) y *Saccharomyces cerevisiae*, permaneciendo estas especies como centrales para la mayoría de las áreas de trabajo, incluyendo diseño de circuitos complejos, inge-

niería del metabolismo, construcción genómica mínima, y estrategias terapéuticas basadas en células.

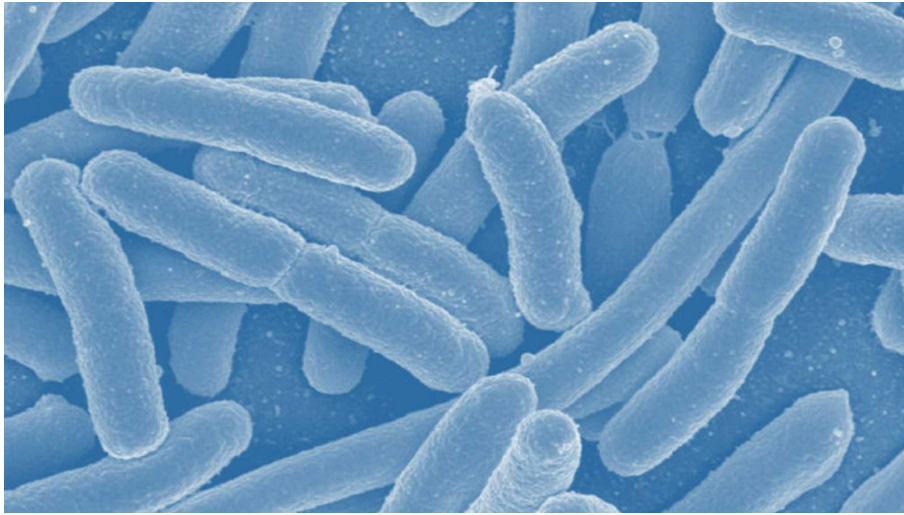


Figura 2.1: La bacteria *Escherichia coli*, una de las especies más trabajadas en biología sintética

Debido a las características del proyecto, en adelante se considerarán las **células procariontas** como la unidad central biológica. El interés principal recae en concreto en la **codificación de genes**, por ejemplo, genes que codifican proteínas. Esto se define bien en la *expresión genética constitutiva* de una célula, cuya explicación se define en el siguiente apartado.

2.1.1. La Expresión Genética Constitutiva

El **dogma central** en biología molecular establece que, para que una célula produzca proteínas, el ADN (DNA en su notación inglesa) de la célula es transcrito al ARN mensajero o mRNA, y este a su vez se traduce en una proteína, es decir la proteína es sintetizada. Aunque este no es siempre el caso, es el caso más general, y este proceso de transcripción seguido de una traducción se denomina *expresión genética* [3].

Metabolismo celular

Células como la *Escherichia coli* son la unidad básica de los organismos vivos, y realizan una amplia variedad de funciones, hacia otra células y hacia ellas mismas, para vivir, crecer y reproducirse. El **metabolismo** es el conjunto de transformaciones o reacciones que tienen lugar en su interior, y se pueden diferenciar dos tipos de procesos metabólicos (figura 2.2):

- **Catabolismo:** componentes orgánicos se rompen para producir energía y bloques de construcción celular.
- **Anabolismo:** uso de esta energía para sintetizar estructuras celulares vitales.

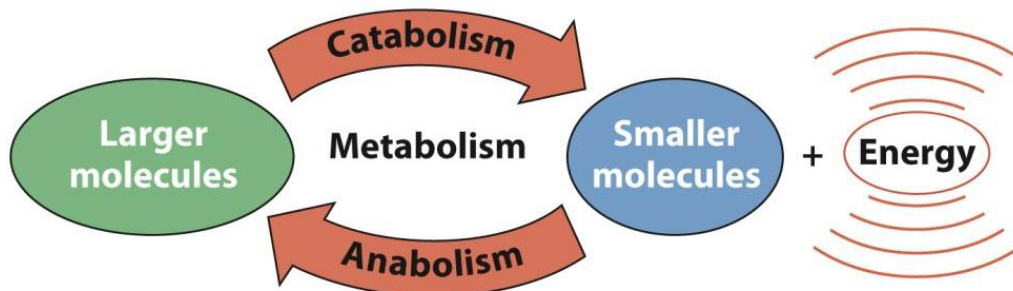


Figura 2.2: Transformaciones metabólicas: catabolismo y anabolismo

Las interacciones metabólicas producidas en una célula pueden resultar en un esquema complejo. Sin embargo, se pueden abstraer estos procesos en 3 niveles funcionales de interacción (figura 2.3):

- **Metaboloma:** conjunto de especies moleculares en la célula, adquiridas por la propia célula de su entorno, o generadas por procesos metabólicos.
- **Proteoma:** es el conjunto de proteínas producidas (expresadas) en la célula. Las proteínas son macromoléculas compuestas por aminoácidos que pueden tener funciones de mecánicas o de estructura, o actuar como enzimas para catalizar procesos metabólicos.
- **Genoma:** conjunto completo de instrucciones requeridas para sintetizar proteínas.

Se puede encontrar por tanto, diferentes actores en la célula (metabolitos, proteínas y genes), denominándose *interactoma* el conjunto de interacciones interrelacionadas.

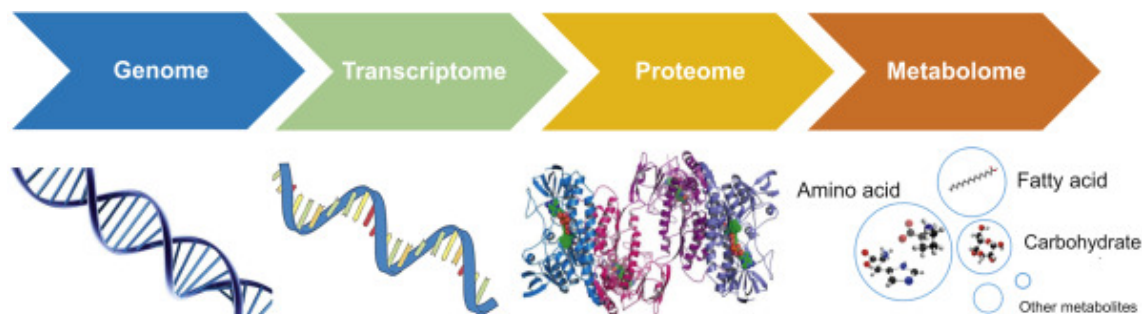


Figura 2.3: Interacción entre genoma, proteoma y metaboloma

Proteínas y el Dogma Central

Las proteínas son macromoléculas compuestas por una o más cadenas de **aminoácidos**, y se diferencian unas a otras según la secuencia de estos. Las características electroquímicas de cada aminoácido, junto con su orden en la secuencia, definen la forma de la proteína en una estructura tridimensional que determina a su vez su actividad.

Todas las proteínas en un organismo son sintetizadas por aminoácidos provenientes de reacciones catabólicas. Para un gen poder ser expresado, ha de ser *transcrito* del alfabeto del ADN, formado por los nucleótidos Adenina (A), Citosina (C), Guanina (G) y Timina (T), al alfabeto del ARN, donde Uracilo (U) reemplaza a la Timina. La molécula ARN es similar al ADN, pero menos estable y más reactiva. La molécula de ARN producida en el proceso de transcripción es denominado ARN mensajero (mRNA). El proceso de sintetizar una proteína desde el mRNA es conocido como *traducción*.

Esta secuencia de eventos es denominada el **Dogma Central** en biología molecular (figura 2.4).

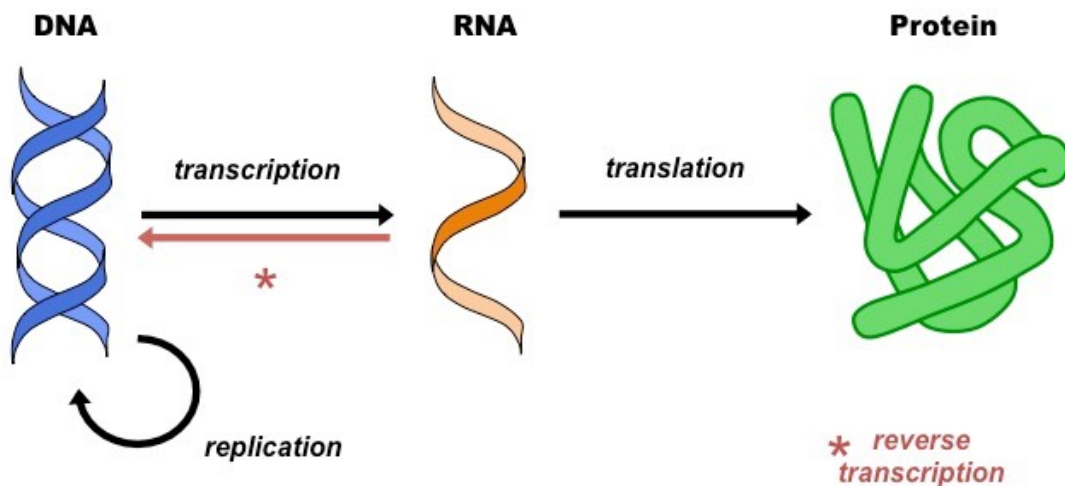


Figura 2.4: Dogma Central en biología molecular: transcripción y traducción

Transcripción genética en células procariotas

Como se ha descrito previamente, para sintetizar una proteína un gen ha de ser transcrito en primer lugar en la molécula del ARN mensajero.

Este proceso se lleva a cabo por una enzima denominada **RNA polimerasa** (RNAP), un complejo formado varias proteínas. Este complejo se *enlaza* (*binding*) a la secuencia apropiada de ADN, al inicio del gen. Esta secuencia se denomina **promotora** (p.e. en los promotores procariotas, el RNAP reconoce dos secuencias cortas a 10 y a 35 nucleótidos del comienzo del gen). Este enlace o *binding* de la proteína RNAP al ADN requiere de la intervención de la denominada proteína factor- σ , anexándose con el propio complejo RNAP para el reconocimiento de las secuencias de inicio.

La transcripción finaliza cuando la proteína RNAP encuentra el denominado **terminador** específico en el gen.

Más adelante, en la sección 3, se verá la importancia del concepto de *binding*, dado que la diferencia en escala de tiempo con el ratio de otras interacciones bioquímicas proporcionará argumentos para la reducción matemática de los sistemas, en técnicas como la *teoría de la perturbación singular*.

Traducción genética en células procariotas

El siguiente proceso para la síntesis de una proteína es la *traducción*. En este proceso, un complejo celular llamado *ribosoma* se une al mRNA, lee su código genético, y sintetiza la correspondiente cadena de aminoácidos. Es decir, el ribosoma *traduce* del lenguaje de los nucleótidos del mRNA al lenguaje de la proteína de aminoácidos.

Para el comienzo de la traducción, el ribosoma se enlaza a la región del mRNA denominada *ribosome binding site* (RBS). Esta posición se define ligeramente antes del punto de comienzo de traducción, y consiste en una secuencia corta, que es detectada por los ribosomas.

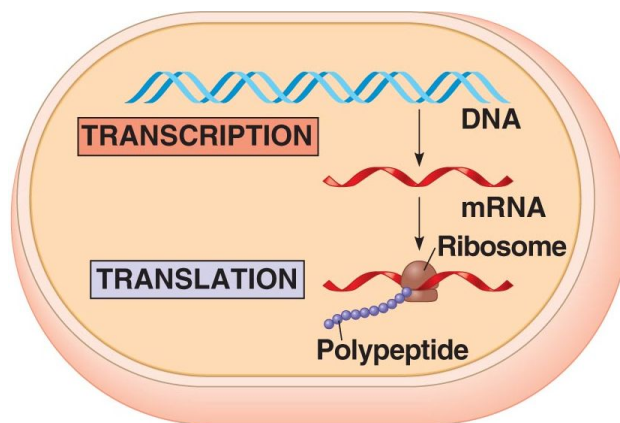


Figura 2.5: Transcripción y traducción en una célula procariota

2.1.2. Parámetros de ajuste en la Expresión Genética

Desde el punto de vista del diseño de circuitos genéticos sintéticos, un aspecto clave es la disponibilidad de parámetros biológicos que pueden ser modificados para aportar al circuito de diseño las características deseadas. Es decir, el diseño requiere de *parámetros de ajuste*.

Se describen a continuación algunas de estas herramientas disponibles para modificar las características del proceso de expresión constitutiva, algunas actuando en la fase de *transcripción*, y otras en la fase de *traducción*.

Fuerza del promotor o afinidad

En las células procariotas la expresión genética se controla principalmente en el nivel de transcripción, resultando que la síntesis del RNA y la síntesis de la proteína suceden simultáneamente. La variación en el ratio de transcripción se debe a variaciones en la *afinidad* del RNAP sobre el promotor, también denominada, su *fuerza*.

La manipulación de la fuerza del promotor se puede realizar de diferentes formas, basadas en la introducción de cambios en la secuencia del promotor. Estos cambios pueden ser introducidos por secuencias de DNA flanqueando el promotor, o entre *motivos* del enlace del promotor.

Eficiencia del terminador

Los terminadores pueden ser no del todo determinantes en su función, es decir, la proteína RNAP puede continuar sintetizando el mRNA a pesar de haber alcanzado el *terminador*. Esto no afecta a la proteína expresada, pero sí al ratio de velocidad, por lo que la transcripción podría resultar más lenta debido a terminadores con baja eficiencia. Además la estabilidad de la molécula de mRNA sintetizada puede depender si la transcripción termina en el punto correcto o no.

Ratio de degradación del mRNA

La degradación del mRNA se produce como resultado del no uso de esta especie después de su función, y puede ser controlada modificando su estructura secundaria en las regiones que aún no han sido traducidas.

Fuerza del RBS

La fuerza del *ribosome binding site* puede ser cambiada modificándose la afinidad de los ribosomas por ejemplo, lo que cambiaría el ratio de inicio de transcripción. Estos cambios en la secuencia RBS puede

cambiar los niveles de expresión en más de tres órdenes de magnitud.

Ratio de degradación de proteína

Las proteína después de ser sintetizadas se degradan en el tiempo, un proceso denominado *proteólisis*. Existen dos formas de degradación: degradación *activa*, forzada por el metabolismo de la célula; y la degradación *pasiva*, debida a la división de la célula. La degradación de la proteína viene dada generalmente por el ratio de crecimiento de la célula.

2.2. Modelos dinámicos en procesos bioquímicos

La **cinética masa-acción** es usada en química e ingeniería química para describir la dinámica de sistemas de reacciones químicas, es decir, de *redes de reacciones*. Estos modelos son una forma especial de los *sistemas compartimentales*, los cuales incluyen relaciones de masa y balance de energías.

A parte de su papel en las aplicaciones de ingeniería química, la cinética masa-acción tiene numerosas propiedades analíticas que son de interés inherente desde una perspectiva de dinámica de sistemas [4].

2.2.1. Modelado de la Expresión Genética Constitutiva

El modelado de procesos bioquímicos en el entorno de las redes genética, como el de la transcripción genética, se puede realizar a diferentes niveles de detalle. Para el propósito de análisis dinámico, los modelos semimecanísticos con algún grado de simplificación son de preferencia, dado que los modelos puramente mecanísticos tienen demasiado parámetros a estimar. Esto, desde el punto de vista biológico, puede generar problemas.

A parte del nivel de abstracción usado en el modelo, se debe tener en cuenta que los procesos bioquímicos son inherentemente *estocásticos*, dado que las reacciones dependen de eventos probabilísticos, por lo que la cuestión del uso de modelos deterministas o estocásticos es relevante.

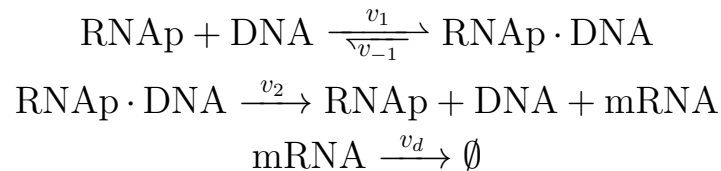
Si el número de moléculas existente en las reacciones es suficientemente grande, el modelo determinístico será válido para el sistema, facilitando las tareas de análisis y simulación. En el caso de nuestro proyecto, este será el nivel de abstracción elegido para los procesos bioquímicos.

Transcripción constitutiva. Redes de reacciones

En la expresión genética constitutiva, la transcripción se puede subdividir en tres procesos:

- **Enlace** de la proteína RNAP al gen promotor.
- **Elongación** sobre el DNA para la creación de una copia mRNA.
- **Degradación** del mRNA.

Estos procesos pueden ser a su vez expresados como reacciones envolviendo las diferentes especies. Se consideran que las siguientes reacciones tienen lugar durante la transcripción:



Donde DNA define el gen, RNAP·DNA representa el complejo molecular resultante de enlazar la proteína RNA polimerasa con el gen promotor, y mRNA, el ARN mensajero.

Los símbolos v_1 , v_{-1} , v_2 y v_d denotan las **velocidades de reacción**. Se toma el enlace de la RNA polimerasa al gen *promotor* como una reacción reversible, mientras que la elongación y la degradación de mRNA son irreversibles. Se asume a su vez que, cuando la RNA polimerasa encuentra el *terminador* y la reacción de elongación finaliza, la RNAP se separa del DNA, por lo que el gen promotor se libera para poder enlazarse con una nueva RNAP.

Transcripción constitutiva. Modelo determinista

Todas las reacciones que tienen lugar en la célula son estocásticas por naturaleza, lo que implica que el modelado de las anteriores reacciones debería ser formulada en términos de *probabilidades*. Sin embargo, los modelos deterministas no tienen en cuenta esta naturaleza probabilística de las reacciones. En su lugar, se asume que la

cantidad de especies transformadas dependen exclusivamente de la cantidad de especies en ese instante, las velocidades a las que suceden las reacciones, y la estequiometría de estas. La cinética masa-acción es el formalismo más usado para expresar las velocidades de las reacciones en una red de ellas.

La ley de masa-acción define que la velocidad de una reacción química es proporcional al producto de las concentraciones de los reactivos, elevado al coeficiente respectivo de la estequiometría de la reacción, y si falta alguno de los productos requeridos la reacción no se produce. La reacción aumentará su velocidad a medida que la concentración de los substratos aumente, es decir, aumentar la concentración de los reactivos (substratos) simplemente aumenta la probabilidad de encuentro o *colisión* entre ellos.

Retomando las reacciones de transcripción comentadas en el apartado anterior, el formalismo de la cinética de masa-acción queda definido considerando que las tres reacciones procederán con velocidades proporcionales a las concentraciones de los reactivos. Denotando $x_1 = [\text{DNA}]$, $x_2 = [\text{RNAP}]$, $x_3 = [\text{RNAP} \cdot \text{DNA}]$, y $x_4 = [\text{mRNA}]$, tenemos:

$$\begin{aligned}v_1 &= k_1 x_1 x_2 \\v_{-1} &= k_{-1} x_3 \\v_2 &= k_m x_3 \\v_3 &= d_m x_4\end{aligned}$$

Donde las constantes proporcionales k_1, k_{-1}, k_m y d_m , son las llamadas *velocidades específicas de reacción*.

Ahora por lo tanto, se puede fijar la dinámica de balance de masas para los cuatro metabolitos. Esta formulación simplemente define que el cambio de la concentración de cada especie con respecto al tiempo es igual a la concentración de entrada menos la concentración de salida. En nuestro modelo queda:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 1 & 0 \\ -1 & 1 & 1 & 0 \\ 1 & -1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = Av = \begin{bmatrix} -k_1x_1x_2 + k_{-1}x_3 + k_mx_3 \\ -k_1x_1x_2 + k_{-1}x_3 + k_mx_3 \\ k_1x_1x_2 - k_{-1}x_3 - k_mx_3 \\ k_mx_3 - d_mx_4 \end{bmatrix}$$

siendo A la **matriz estequiométrica**.

2.3. Teoría de grafos

En matemáticas y en ciencias de la computación, la *teoría de grafos* estudia las propiedades de los grafos. Un grafo es un conjunto no vacío de **vértices** (o *nodos*), y un conjunto de **pares de vértices**, denominados *aristas*. Los grafos se representan mediante una serie de puntos conectados por líneas, y pueden ser orientados (la línea que representa un par de vértices puede tener solo un sentido relacional en el sistema) o no orientado [5][6].

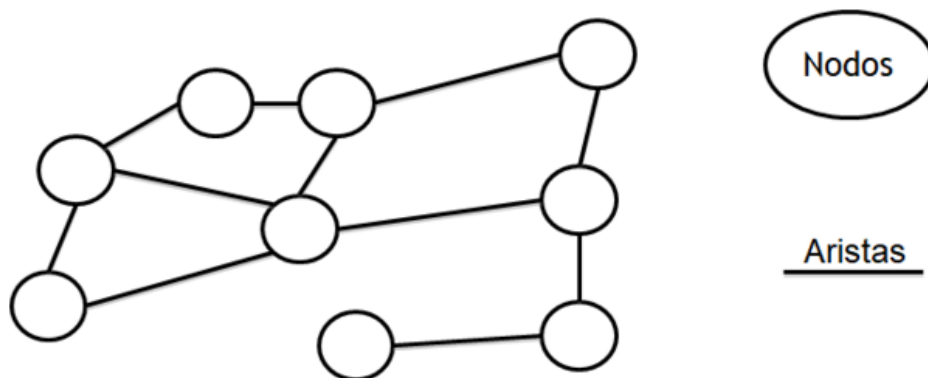


Figura 2.6: Ejemplo de grafo no orientado

2.3.1. Estructuras de datos

A nivel programático, existen dos formas de estructurar los datos, siempre dependiente de las características del grafo y el algoritmo usado para manipularlo. Entre las estructuras más usadas se encuentran las *listas* y las *matrices*.

Estructura de lista

- **Lista de incidencia:** las **aristas** son representadas con un vector de pares, donde cada par representa cada una de las aristas.
- **Lista de adyacencia:** cada **vértice** tiene una lista de vértices los cuales son adyacentes a él. Esto causa redundancia en un grafo no orientado (ya que A existe en la lista de adyacencia de B, y viceversa), pero las búsquedas son más rápidas, a cambio de almacenamiento extra.

En esta estructura de datos la idea es asociar a cada vértice i del grafo una lista que contenga todos aquellos vértices j que sean adyacentes a él.

Estructura de matrices

- **Matriz de incidencia:** el grafo está representado por una matriz de A aristas por V vértices, donde el índice [*arista*, *vértice*] contiene la información de la arista (1 conectado, 0 no conectado).
- **Matriz de adyacencia:** el grafo se representa con una **matriz cuadrada** M de tamaño n^2 , donde n es el número de vértices. si hay una arista entre un vértice x y un vértice y , entonces el elemento $m_{x,y}$ es 1, de lo contrario es 0.

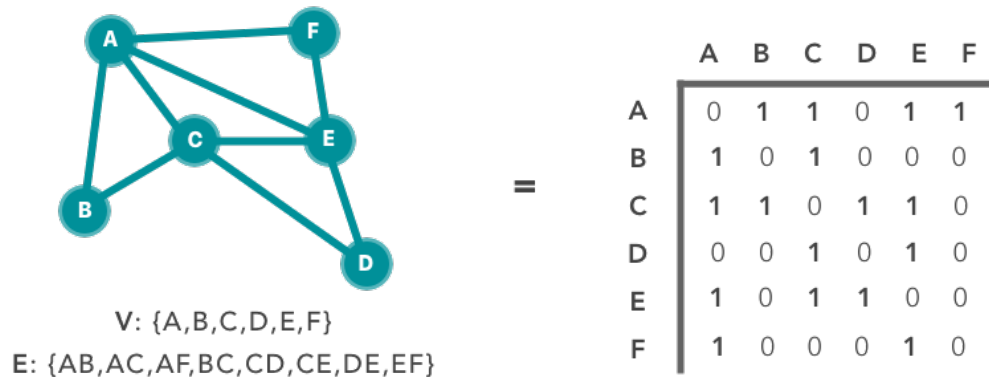


Figura 2.7: Vértices, aristas y matriz de adyacencia de un grafo

2.3.2. Aplicación a circuitos biológicos

Para los sistemas biológicos, la teoría de grafos permite simplificar el estudio de las interacciones bioquímicas de las especies a nivel gráfico, y aporta relaciones analíticas para desarrollar técnicas sobre ellas, a nivel matemático (figura 2.8).

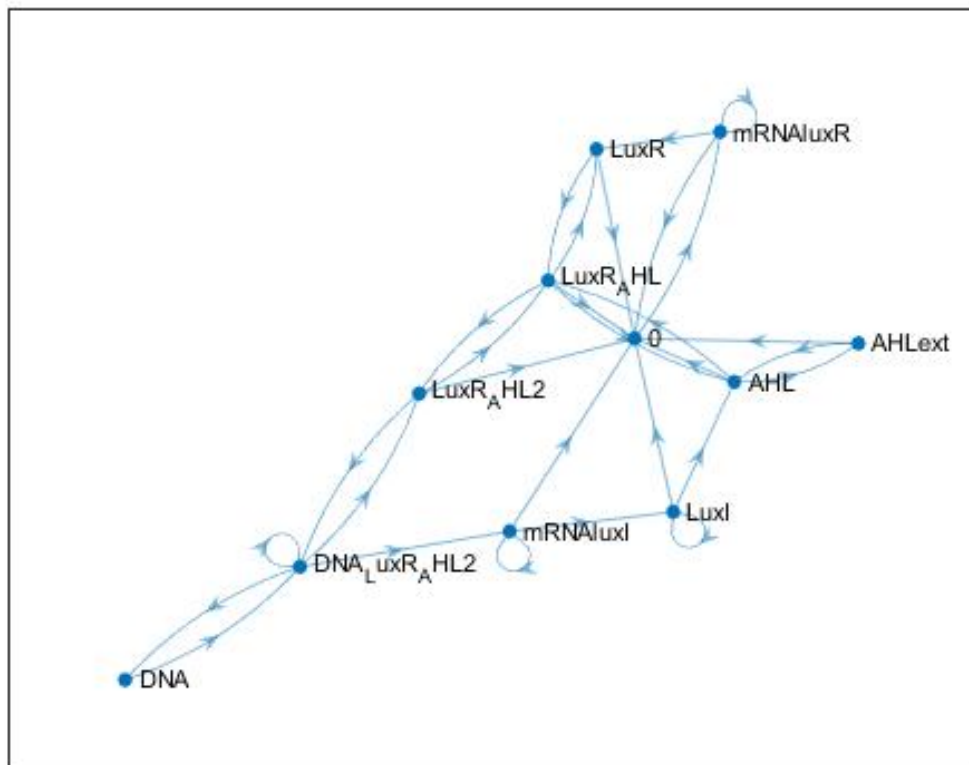


Figura 2.8: Ejemplo de grafo orientado en sistemas biológicos

Capítulo 3

Fundamentos del proyecto

En este capítulo se expone la teoría para la reducción de modelos en sistemas bioquímicos.

3.1. Reducción de modelos

El modelo dinámico presentado en el apartado 2.2 puede ser simplificado más aún. Este proceso de simplificación es llamado *reducción de modelos* dado que tiene como objetivo devolver un modelo con menos variables y sistema diferencial de menor orden. Algunas de las necesidades para reducir un modelo dinámico:

- Los modelos de orden alto suelen tener muchos parámetros, que han de ser obtenidos experimentalmente, un proceso denominado *estimación de parámetros* que conlleva dificultades propiamente experimentales y un elevado coste computacional.
- En la práctica, algunas reacciones suceden a unas velocidades mucho mayores que otras. Por ejemplo, el enlace/desenlace de la proteína RNA polimerasa al gen *promotor* son mucho más rápidas que las de la fase de *elongación*, lo que implica una gran diferencia en las escalas de tiempo entre reacciones. Estas diferencias originan dificultades a la hora de simular la evolución temporal de las reacciones así como el entendimiento de los principios básicos de funcionalidad.

El proceso de reducción debería proporcionar un modelo menos denso para el análisis computacional, evitando a la vez una reducción excesiva que llevaría al sistema a una falta de relevancia biológica.

3.1.1. Teoría de la Perturbación Singular

La teoría de la perturbación singular engloba el estudio de problemas entorno a parámetros para los cuales las soluciones al problema difieren en función del límite del parámetro, por lo que el límite es singular. Las soluciones en general convergen a la solución del problema límite, a medida que el parámetro se aproxima a al valor límite.

Algunas de las áreas de motivación para el desarrollo de la teoría son relativos a problemas físicos. Algunos notables ejemplos:

- Mecánica celeste
- Mecánica de fluidos
- Dinámica oscilatoria y circuitos eléctricos
- Sistemas biológicos y cinética de las reacciones químicas.

Cada una de estas áreas plantea problemas cuyas soluciones se caracterizan por la gran variación de escalas de tiempo o longitud. La filosofía detrás la teoría de perturbación singular reside en aprovecharse de la separación de escalas para obtener problemas reducidos más sencillos que el problema original [7].

3.1.2. Aproximación del Estado Estacionario

La reducción de modelos se puede llevar a cabo con lo que se define como **QSSA** de las especies químicas rápidas. En esencia, QSSA es un método de perturbación singular que considera la separación de escalas de tiempo entre las diferentes dinámicas del sistema. En particular, se asume que las reacciones de enlace suceden muy rápidamente en comparación con aquellas correspondientes con la *transcripción* y la *degradación* [3].

Algunas relaciones algebraicas adicionales se pueden obtener mediante los *invariantes del sistema*. En el caso de la red de reacciones, se puede observar que las especies involucradas mantienen la variación de concentración constante en el tiempo. Estas combinaciones lineales, denominadas *motivos*, se pueden entender como un tipo de *quasi-especies* que permanecen invariantes, es decir mantienen una concentración constantes (no hay variación de concentración, la derivada es 0).

Capítulo 4

Desarrollo del proyecto

En este capítulo se expone el hilo de trabajo, partiendo del estudio de la herramienta propuesta en el artículo hacia su implementación software y desarrollo de la interfaz de usuario.

4.1. Estudio de la herramienta propuesta

4.1.1. Introducción

En el análisis y diseño de sistemas de modelos matemáticos, la reducción de estos modelos es una herramienta indispensable en la mayoría de las aplicaciones de ingeniería. Debido a la complejidad de algunos de estos sistemas, la reducción del orden del sistema ofrece una representación más simple, computacionalmente más rápida, y más fácil de usar para el diseño del sistema.

En los sistemas biológicos no siempre es accesible toda la información de interacción de los sistemas para ser modelado eficientemente, sobretodo en los parámetros biológicos. Por ello resulta de gran importancia la obtención de modelos reducidos.

Existen diferentes formas y escalas en las que los sistemas biológicos son representados. Uno de ellos es la **cinética masa-acción** (comentado en la sección 2.2), una forma de modelado basada en las ecuaciones de las reacciones, donde la dinámica de diferentes procesos se aproxima modelando la concentración de diferentes **especies** [8] (figura 4.1).

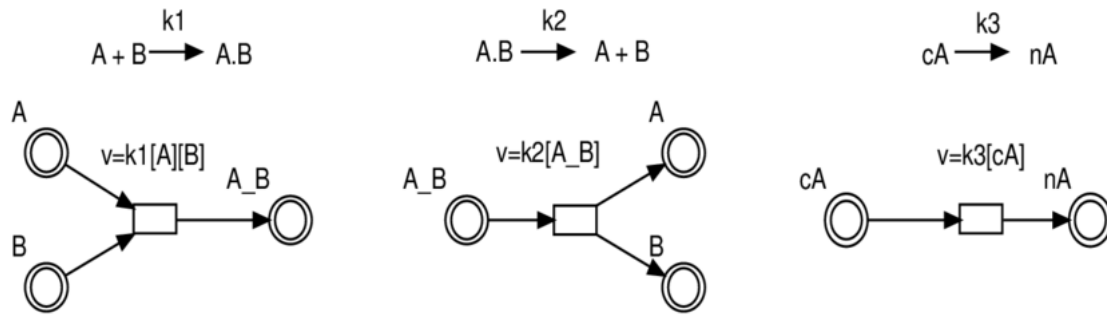


Figura 4.1: Ejemplo de modelado de la cinética masa-acción de un sistema

Con el modelado basado en esta técnica, es posible la creación de modelos de menor dimensión y más simples. La mayoría de los métodos con este objetivo hacen uso de la *separación de escalas de tiempo*, o un *truncado equilibrado* que transforma el modelo original en uno de menor dimensión, respetando el comportamiento entrada-salida.

En la práctica sin embargo, uno de los enfoques más usados es el **QSSA** (*Quasi-Steady State Approximation*). La teoría detrás de esta técnica se deriva de la **teoría de la perturbación singular** de los sistemas dinámicos no lineales. El algoritmo trabajado posteriormente se basa en este método [9].

Entre los factores a tener en cuenta al desarrollar este método, se encuentra el peligro de converger estados de las relaciones algebraicas, para cuyo correcta implementación se ha de resolver en el marco de la teoría de perturbación singular.

4.1.2. Planteamiento y formulación

El elemento más importante en el planteamiento de la herramienta de reducción recae en el el concepto de **coeficientes de sensibilidad**, dado que se buscará una comparación eficiente, basada en el error producido por esta sensibilidad sobre los parámetros del sistema, y escogiendo el modelo reducido que sea menos sensible a este error. Para concretar este concepto, continúan algunas definiciones importantes.

Se define el modelo dinámico de un sistema como:

$$\begin{aligned}\dot{x} &= f(x, \theta) \\ y &= Cx\end{aligned}$$

donde x representa las variables de estado, θ es el conjunto de parámetros, e y representa la salida del sistema. Para cada estado x_i y para cada parámetro θ_j , el coeficiente de sensibilidad s_{ij} para el estado dado se define como:

$$s_{ij} = \frac{\partial x_i}{\partial \theta_j}$$

Definiendo también $S_j = \left[\frac{\partial x_1}{\partial \theta_j} \frac{\partial x_2}{\partial \theta_j} \cdots \frac{\partial x_n}{\partial \theta_j} \right]^T$ como el vector de coeficientes de sensibilidad, se satisface la ecuación lineal diferencial dada:

$$\dot{S}_j = JS_j + Z_j$$

Donde Z_j se compone de derivadas parciales y J es la matriz Jacobiana,

$$Z_j = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_j} \\ \frac{\partial f_2}{\partial \theta_j} \\ \cdot \\ \cdot \\ \frac{\partial f_n}{\partial \theta_j} \end{bmatrix} \quad J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \cdot & \cdot & \cdot & \cdot \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

De igual forma, se puede definir el modelo reducido con $\hat{f}, \hat{x}, \hat{\theta}$.

Extendiendo un poco más el concepto de sensibilidad, se llega a la dinámica de la **sensibilidad del error** de las variables de salida, lo que permite una minimización del mismo para aportar distinción a los diferentes sistemas de reducción posibles que se plantean en el algoritmo.

Sistemas No Lineales

La definición de la salida de la sensibilidad del error para el sistema aumentado (\bar{S} *augmented*) para sistemas no lineales, queda descrita de forma similar a como se ha visto anteriormente:

$$\dot{\bar{S}} = \bar{J}\bar{S} + \bar{Z}$$

Donde $\bar{J} = \begin{bmatrix} J & 0 \\ 0 & \hat{J} \end{bmatrix}$ y $\bar{Z} = \begin{bmatrix} Z \\ \hat{Z} \end{bmatrix}$, y la sensibilidad del error queda como:

$$S_e = \bar{C}\bar{S}$$

donde $\bar{C} = \begin{bmatrix} C & 0 \\ 0 & \hat{C} \end{bmatrix}$, y la dinámica de la sensibilidad del error quedaría:

$$\dot{S}_e = \bar{C}\bar{J}\bar{S} + \bar{C}\bar{Z}$$

Es posible usar estas definiciones para implementar un algoritmo que resolviera la sensibilidad del error integrando las ecuaciones dinámicas de la sensibilidad del error junto con la dinámica del sistema. Sin embargo, acotando superiormente la norma de la sensibilidad del error (definida en la ecuación 4.1) resulta más eficiente en computación que el análisis de sensibilidad en bruto de todos los posibles modelos de orden reducido.

4.1.3. Algoritmo

A continuación se plantea el pseudocódigo del algoritmo en cuestión, basado en las definiciones presentadas anteriormente, para computar la *sensibilidad del error* para todos los modelos reducidos:

```

GetReducedModel(f, x, C, tolE, tolSe)
 $\hat{x} \leftarrow []$ ,  $x_c \leftarrow []$ ,  $\hat{f} \leftarrow []$ ,  $f_c \leftarrow []$ 
R ← Set de los posibles modelos reducidos
f, x ← Contienen el modelo simbólico
for j = 1:length(R) do
  T = R[j]
  for i = 1:length(x) do
    if i in T then
       $x_c[i] \leftarrow x[i]$ 
       $f_c[i] \leftarrow f[i]$ 
    else
       $\hat{x} \leftarrow x[i]$ 
       $\hat{f} \leftarrow f[i]$ 
    end
  end
  for k = 1:length(xc) do
    for j = 1:length( $\hat{f}$ ) do
       $\hat{f}[k] \leftarrow$  Solución de  $f_c[j] = 0$  para  $x_c[j]$ 
    end
  end
  Resolvers EDOs para x y  $\hat{x}$  y computar y y  $\hat{y}$ 
   $e[j] \leftarrow \|y - \hat{y}\|_2$ 
   $S_e[j] \leftarrow \|S_e\|_2$  de la ecuación (4.1)
end
if min(e) > tole and min(Se) > tolSe then
  | indexMin = indexOfMin(e)
  | reducedModel =  $\hat{f}$ [indexMin]
else
  | reducedModel = None
end
return reducedModel

```

donde la norma euclídea de la sensibilidad del error parte de la ecuación:

$$\|S_e\|_2^2 \leq \lambda_{max}(P) + 2Nmax_t \left\| \begin{bmatrix} \frac{\partial f}{\partial \theta} \\ \frac{\partial \hat{f}}{\partial \theta} \end{bmatrix}^T Q_s \begin{bmatrix} \frac{\partial x}{\partial \theta} \\ \frac{\partial x_c}{\partial \theta} \end{bmatrix} \right\|_2 \quad (4.1)$$

cuyo desarrollo, demostración y definición de P y Q_s , puede encontrarse en el propio artículo [9].

El conjunto de R define la combinación de todos las posibles combinaciones de los estados que se pretende reducir. La *matriz de permutación* T del algoritmo consiste en los índices de los estados que se *colapsan* (se juntan).

La complejidad en tiempo del algoritmo es exponencial con la dimensión del modelo n , dado que se busca el mejor modelo reducido para todas las posibles combinaciones de la matriz T .

4.2. Implementación software

La programación del modelo dinámico, el algoritmo y los grafos, se realizan en *scripts* de Matlab, que posteriormente son llamados desde la interfaz de usuario principal.

El hilo de desarrollo software tiene las siguientes pautas:

- Desarrollo de *scripts* que realicen las funciones principales de obtención de los datos del sistema y modelado del mismo.
- Implementación software del algoritmo estudiado.
- Esquematización del sistema biológico en grafos.
- Diseño e implementación de la interfaz de usuario que recoja todas las anteriores funcionalidades.

4.2.1. Modelado dinámico

Uno de los *scripts* de mayor importancia en el software es *getDynamicModel*. Esta función tienen como objetivo la obtención de información del modelo que, como se verá en la sección 4.3, se habrá cargado en el software mediante un archivo **.csv** (*Comma-Separated Values*) que contiene las **reacciones** del sistema biológico, junto con sus **constantes** de velocidad de reacción *forward* y *reverse* (figura 4.2).

	A
1	'Reaction', 'kfwd', 'krev'
2	gR + RNAp = RNAp_gR, k1,k_1
3	RNAp_gR -> RNAp + mR + gR, kmr
4	mR -> mR + R, kr
5	mR -> 0, dmR
6	R -> 0, dR

Figura 4.2: Ejemplo de tabla de reacciones de sistema biológico, archivo *.csv*

A continuación se tratan los diferentes elementos que componen el *script*, a saber:

- Obtención de información del modelo y construcción de celdas y tablas como variables
- Salida de las funciones

Para la explicación del código se toma como ejemplo las reacciones respectivas de la **transcripción constitutiva**.

Construcción de tablas y celdas

En primer lugar se sustituyen las reacciones bidireccionales por dos reacciones unidireccionales, de forma que solo se almacenan constantes de velocidad de reacción de tipo *forward*. Para ello se realiza un *split* de los caracteres =, sustituyéndolos por ->:

```
%% Construct tables
%Split species
s_reac = string(input);
t_reac = array2table(s_reac, 'VariableNames', {'Reaction',
                                             'k fwd',
                                             'k rev'});
% Expand reversible reaction in two
ir=contains(t_reac.Reaction, '=');
t_reacir=t_reac(:,1:2);
irRows={};
for i=find(ir)'
    t_reacir.Reaction(i)=replace(t_reac.Reaction(i),
                                '=', '->');
    aux=regexp(t_reac.Reaction(i), '(=)', 'split');
    irRows=[irRows;{join([aux(2), '->', aux(1)]),
                    t_reac.krev(i)}];
end
```

Una vez recompuesto el sistema con una sola constante de velocidad, se extraen las especies de la misma forma, separando las reacciones por -> y por +.

Construcción de celda dinámica

Se procede a continuación a la formación de una variable de tipo celda, muy manejables en Matlab, con la información del modelo. Para ello primero se obtiene el número de reactivos y productos para conformar las columnas de la celda de forma dinámica, y después se cargan todos los valores quedando de la siguiente forma, en la variable *spa*:

```
6×5 cell array

{'gR'      }    {'RNAp'  }    {'RNAp_gR'}    {0×0 char}    {0×0 char}
{'RNAp_gR'}    {0×0 char}    {'RNAp'      }    {'mR'      }    {'gR'      }
{'mR'      }    {0×0 char}    {'mR'      }    {'R'       }    {0×0 char}
{'mR'      }    {0×0 char}    {'0'       }    {0×0 char}    {0×0 char}
{'R'       }    {0×0 char}    {'0'       }    {0×0 char}    {0×0 char}
{'RNAp_gR'}    {0×0 char}    {'gR'      }    {'RNAp'    }    {0×0 char}
```

Figura 4.3: Celda con separación de componentes del sistema por cada columna

Construcción de las ecuaciones

Se obtienen las variaciones de concentración de cada especie con la cinética de masa-acción, y se conforman las ecuaciones del sistema en variable *v*:

```
%% Set Mass-Action Kinetics
for i=1:size(spa,1)
    current_reaction = spa(i,:);
    vaux = 1;
    for k=1:2
        if(current_reaction(k)=="")
            %skip
        else
            vaux = vaux * str2sym(current_reaction(k));
        end
    end
    vaux = str2sym(t_reacir.kfwd(i))*vaux;
    v(i) = vaux;
end
```



```
%% Dynamic Model Equations
%Reactives
for i=1:length(st_vars)
    for j=1:size(spa,1)
        for k=1:num_r
            if (spa(j,k)==string(st_vars(i)))
                f(i) = f(i) - v(j);
            end
        end
    end

%Products
    for k=num_r+1:num_r+num_p
        if (spa(j,k)==string(st_vars(i)))
            f(i) = f(i) + v(j);
        end
    end
end
end
```

Donde *stvars* hace referencia a un *array* con las variables del sistema, y *numr* y *nump*, al número de reactivos y productos del sistema, respectivamente.

Output

Para que el resto del software pueda operar con las variables construidas en esta función, se retornan los tres siguientes arrays de tipo simbólico:

- Las variables de estado (especies) del sistema.
- Las constantes de velocidad, ahora todas de tipo *forward*.
- Las ecuaciones dinámicas calculadas para el modelo.

4.2.2. Algoritmo de reducción

A continuación se describe la implementación software del algoritmo visto en la sección 4.1.3, encapsulado en la función *reductionAlgorithm*. Es importante resaltar que se han encontrado una serie de **erratas** en el artículo, durante el estudio e implementación del algoritmo, que dejaban al mismo completamente afuncional, por lo que se han notificado en el Anexo 1 y corregido con ayuda de los tutores para su implementación.

La función tiene seis parámetros de entrada y dos variables de retorno:

```
function [output1 output2] =
    reductionAlgorithm(T,f,x,C,tolE,tolSe)
```

- **T**: Matriz de permutación con el conjunto de variables para el análisis de un posible modelo de reducción.
- **f**: Conjunto del sistema de ecuaciones de estado a consecuencia de la obtención del modelo dinámico.
- **x**: Conjunto de variables de estado.
- **C**: Matriz de salida.
- **tolE**: Tolerancia deseada para el error.
- **tolSe**: Tolerancia deseada para la sensibilidad del error.

Y respecto a las variables de salida, ambos arrays:

- Conjunto de funciones del modelo reducido.
- Conjunto de variables de estado en colapso del modelo reducido.

Se declaran las variables necesarias para almacenar todos los vectores de variables de estado y funciones, de *collapse* y *reduced*, y se divide el procedimiento en varias partes.

Para conjunto de modelos reducidos R , se almacenan en *col* los estados y funciones reducibles, y en *red* el resto del sistema.

```

%% Algorithm: Array setting
for j = 1:length(R)
    %Set up collapsed and reduced for each possible
    reduced model
    T = R{j};
    for i = 1:length(x)
        if ismember(i,T)==1
            xcol{col_index} = x(i);
            fcol{col_index} = f(i);
            col_index = col_index + 1;
        else
            xred{red_index} = x(i);
            fred{red_index} = f(i);
            red_index = red_index + 1;
        end
    end
end
end

```

Una vez establecido el sistema, se resuelve el sistema de ecuaciones, igualando a 0 las ecuaciones de estado de las variables no colapsadas (reducidas) y sustituyendo en la funciones de las variables colapsadas, siempre que contengan la propia variable de estado.

```

%% Algoritm: Solve reduced equations
for k = 1:length(xred)
    eqn = fred{k} == 0;
    sol = solve(eqn,xred{k});
    for h=1:length(xcol)
        if has(fcol{h},xred{k})==1
            fsol{sol_index} = subs(fcol{h},xred{k},sol);
            sol_index = sol_index + 1;
        end
    end
end
end

```

La función finaliza retornando los vectores con: las **ecuaciones de estado** reducidas, y el vector de **variables de estado** reducidos.

4.2.3. Grafos

Para el desarrollo del grafo orientado, se hace uso de la función de Matlab *digraph*.

En primer lugar se procede a la construcción celdas y tablas como obtención de la información del sistema biológico, de la misma forma que se ha realizado en el apartado 4.2.1 para la función *getDynamic-Model*.

Una vez obtenida la información se conforman los nodos y las aristas unidireccionales (al estar transformado el sistema de reacciones a solo un sentido y a solo una constante *forward*):

```
%% Set nodes and edges
for i=1:length(isp)
    aux1=[(isp(i,1:2)*aux(:,1:2)')',
          (isp(i,3:4)*aux(:,3:4)')'];
    conn=[conn; aux1];
    if spt.isRev(i)
        conn=[conn;aux1(:, [2,1])];
    end
end
iconn=find(min(conn')>0);
conn=unique(conn(iconn,:), 'rows');
```

Una vez conformado, se genera el grafo con las especies, con la función *digraph* y se dibuja.

```
%Generate species graph and visualize
Ge=digraph(conn(:,1),conn(:,2), [], cellstr(species));
plot(Ge)
```

4.3. Interfaz de usuario

Una aplicación de interfaz de usuario (UI) presenta gráficamente una o más **ventanas de control** (figura 4.4), que contienen lo que se denominan *componentes* (botones, *sliders*, texto, tablas, etc.), que permiten interacción para realización de tareas. La característica principal reside en que el usuario no ha de conocer el código ni las subtareas programadas por debajo de la aplicación [10].

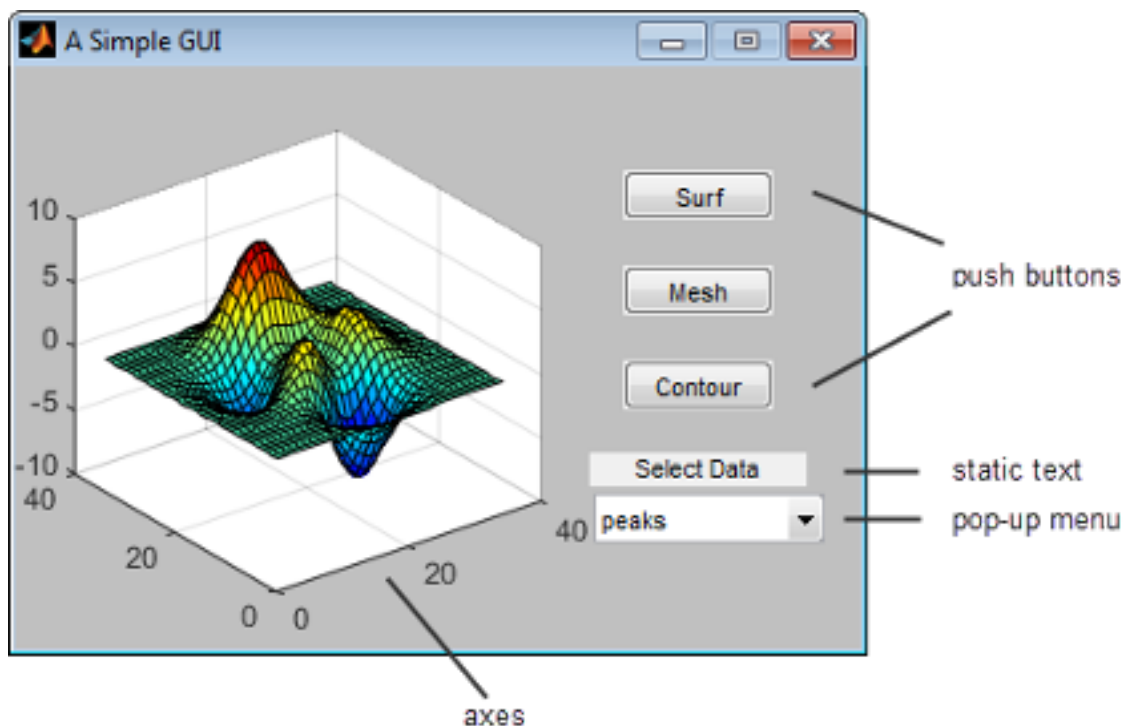


Figura 4.4: Ejemplo de interfaz de usuario guiada

De forma general, la aplicación mantiene un estado de ‘escucha’ y responde a la acción de control. Cada controlable o componente tiene varios *callbacks* asociados que realizan la tarea designada, lanzados durante la interacción del usuario con ellos, lo que es llamado respuesta al **evento**. Esta programación, por ende es referida como *event-driven programming*.

La ejecución de los *callbacks* a raíz de los eventos se realizan asíncronamente, de forma que la aplicación puede responder previamente a otros eventos prioritarios, si es requerido.

4.3.1. Herramienta GUIDE

Para la implementación de la interfaz de usuario se hace uso del *toolbox* de Matlab GUIDE. Esta herramienta posibilita la construcción de una interfaz mediante dos formas:

- **IDE de GUIDE:** esta solución se desarrolla alrededor de una ventana de edición principal en la que los componentes se posicionan gráficamente, así como sus propiedades (figura 4.5). La herramienta crea códigos de archivo que contienen los *callbacks* de interacción con los componentes, desde los que se lanza la aplicación.
- **Programación:** se crea un código base que define todos los componentes, junto con sus propiedades y comportamiento, gestionando las interacciones del usuario.

Ambos códigos generados por la herramienta son muy similares y, como se observará más adelante, se hace uso de ambas formas de trabajo, de forma que se aporta agilidad al proyecto, pero marcando profundidad con una programación más concreta de algunos componentes.

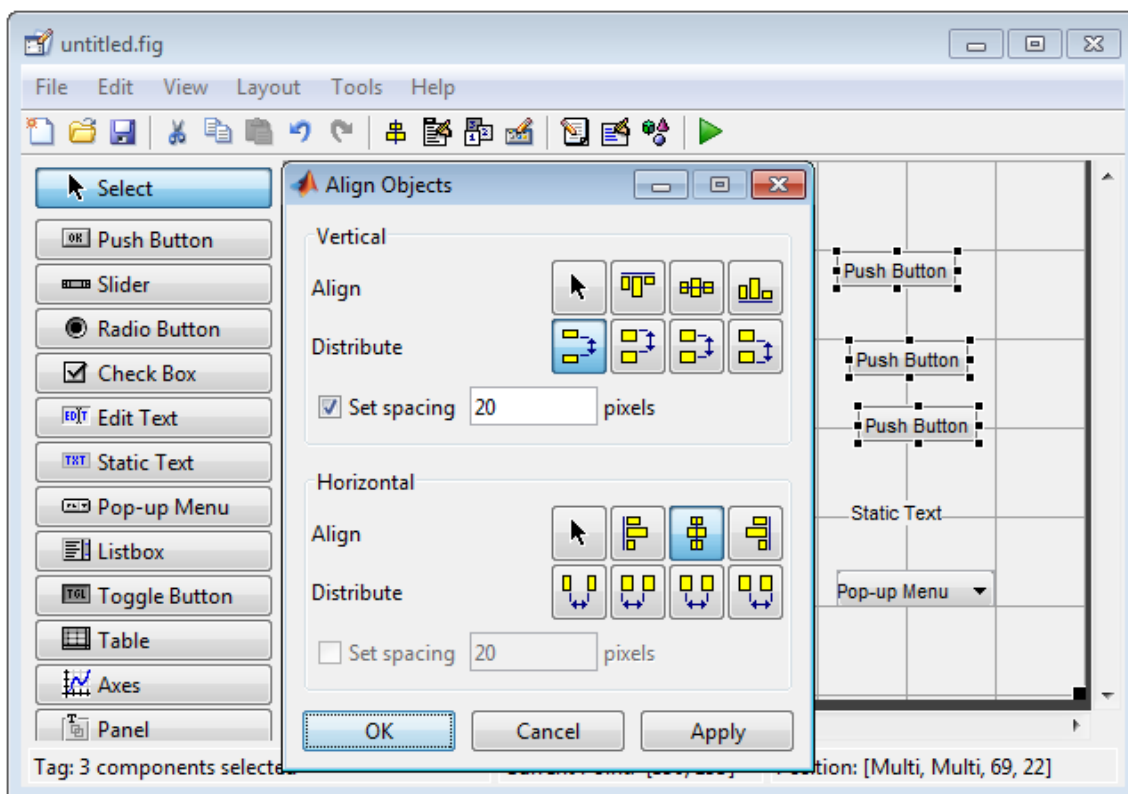
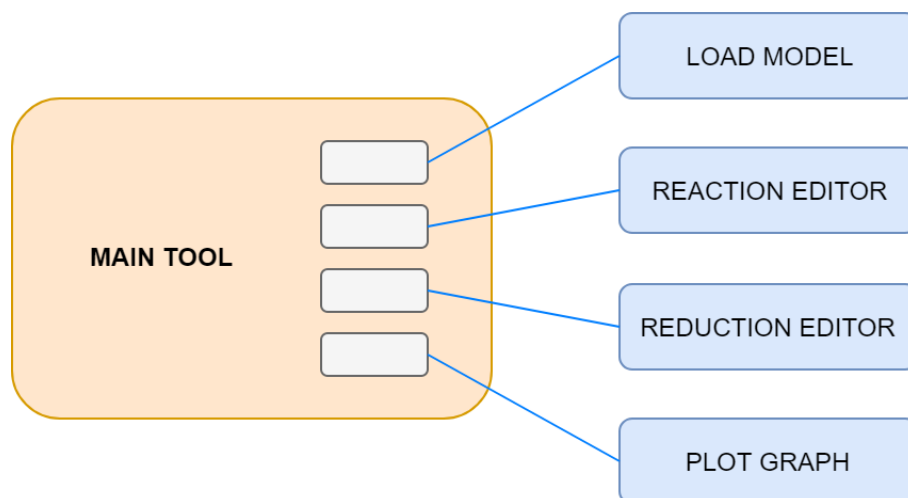


Figura 4.5: IDE de desarrollo de GUIDE

4.3.2. Diseño de la GUI

Para el desarrollo de la interfaz se procura un diseño simple y modular, donde se usan varios *push buttons* para acceder a las diferentes funcionalidades:

Figura 4.6: Diseño de la herramienta principal y declaración de los *push buttons*

- **Load model:** componente que realiza los callbacks necesarios para acceder a la propia interfaz de *open file* del sistema. En este caso en concreto se diseña la GUI para cargar ficheros **.csv** (*Comma-Separated Values*), donde se esperan que estén cargados las reacciones del sistema.
- **Reaction Editor:** este componente realiza una llamada a otra GUI (las nuevas ventanas interaccionables haciendo uso de GUI-DE son nuevas GUIs en sí), y se ofrece al usuario la posible edición de la tabla de reacciones cargada o la creación de una propia.
- **Reduction Editor:** de nuevo este componente también realiza llamada a una nueva GUI para el ajuste de parámetros/rangos, asignación de valores a variables y constantes, y *output* del nuevo modelo reducido.
- **Plot Graph:** se crea una nueva figura de tipo Matlab con el grafo del sistema.

4.3.3. Desarrollo

La implementación de la GUI comienza con el prediseño en la propia IDE de Matlab y posterior programación del *script* generado para manejar la funcionalidad de los componentes. En Matlab, cada ventana abierta, aunque sea llamada desde una ventana anterior, es **una propia GUI**, por lo que a lo largo del desarrollo se diseñan y programan varias GUIs.

A raíz de esto, y como se observa en el diseño de la herramienta en la figura 4.6, la interfaz principal hace las llamadas a las diferentes interfaces, como consecuencia de la activación del *callback* del componente oportuno. En este caso, los eventos son activados por *push buttons* en la interfaz principal.

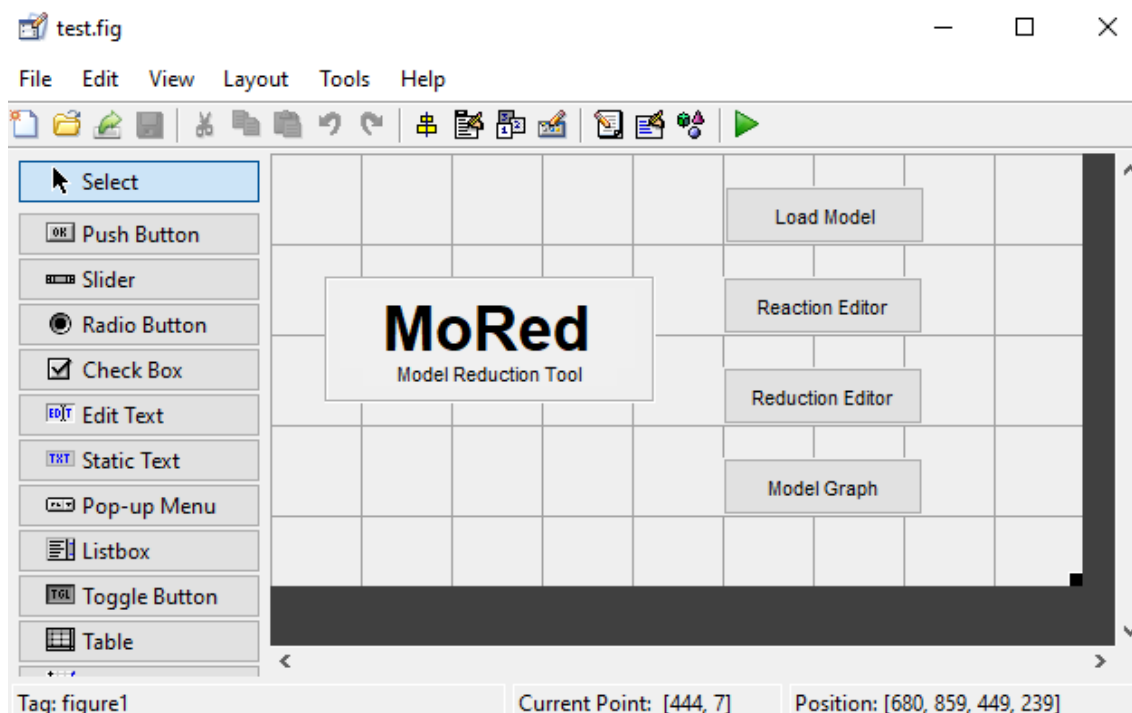


Figura 4.7: Implementación de la herramienta principal en la IDE

Funciones principales

Cada *script* generado por Matlab para las interfaces contiene unas mismas funciones:

- Una función **OpeningFcn** que inicializa la interfaz con las variables principales y la configuración establecida en la IDE.
- La función **OutputFcn** que es llamada por evento cuando finaliza la función general de la interfaz, o se destruye el objeto de la misma.
- Los *callbacks* de todos los componentes de la interfaz.

El hilo de funcionamiento queda reflejado en el siguiente esquema:

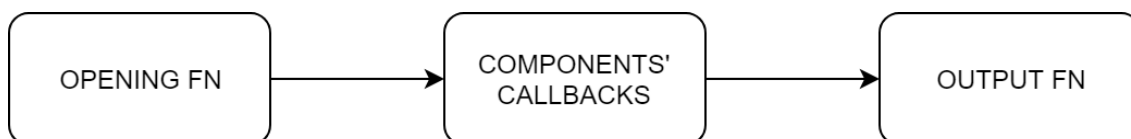


Figura 4.8: Hilo de funcionamiento al ser ejecutada una interfaz

VARIABLES PRINCIPALES

Al generarse el archivo de programación de la interfaz, se declaran e inicializan dos variables importantes:

- **varargin:** variable que contiene los parámetros de entrada de la interfaz, en formato celda.
- **hObject:** manejador del objeto principal en cada evento. Dentro de cada *espacio de nombres* de cada función, *hObject* es la variable del propio objeto del componente.
- **handles:** esta variable es una *estructura de datos* global para todas las *callbacks* de la interfaz. Se actualiza dentro de un espacio de nombres con la función *guidata(hObject,handles)*.
- **varargout:** variable de salida de la interfaz (retornada cuando se destruye el objeto interfaz) en formato celda.

Se detallan a continuación la funcionalidad e implementación de los componentes de **Main Tool**, siendo únicamente *Reaction Editor* y *Reduction Editor* nuevas interfaces.

Load Model

Este componente carga en la variable *handles.reac_table* las columnas en formato *string*, y el conjunto de nodos y aristas para dibujar el grafo, en *handles.reac_plot*, ambos de una tabla en un archivo *.csv*. Existe una función en Matlab para abrir una ventana del sistema y cargar un tipo de archivos específico: *uigetfile()*.

```
function loadModelButton_Callback(hObject, eventdata,
    handles)
% hObject - handle to loadModelButton
% handles - structure with handles and user data (see
    GUIDATA)
[handles.file_name handles.file_path] =
    uigetfile('*.csv');
```

```
if (handles.file_name == 0)
    warning("No file loaded");
else
    model = strcat(handles.file_path,handles.file_name);
    [handles.reac_plot handles.reac_table] =
        getModelData(model);
end

%Save the handles structure
guidata(hObject,handles);
```

En este código se refleja el funcionamiento de este componentes, siendo a su vez el esquema general de *callback* de todos los componentes.

Reaction Editor

Este componente llama a una nueva ventana que permite la edición de las reacciones del sistema cargado previamente o, si no hay ninguno cargado, la creación de nuevas reacciones.

```
function reactionEditorButton_Callback(hObject,
    eventdata, handles)
output = ReactionEditor(handles.reac_table);

%Save the handles structure
guidata(hObject,handles);
```

Se pasa como parámetro de entrada la tabla con la información del sistema biológico y retorna la misma tabla con las modificaciones realizadas por el usuario (figura 4.9).

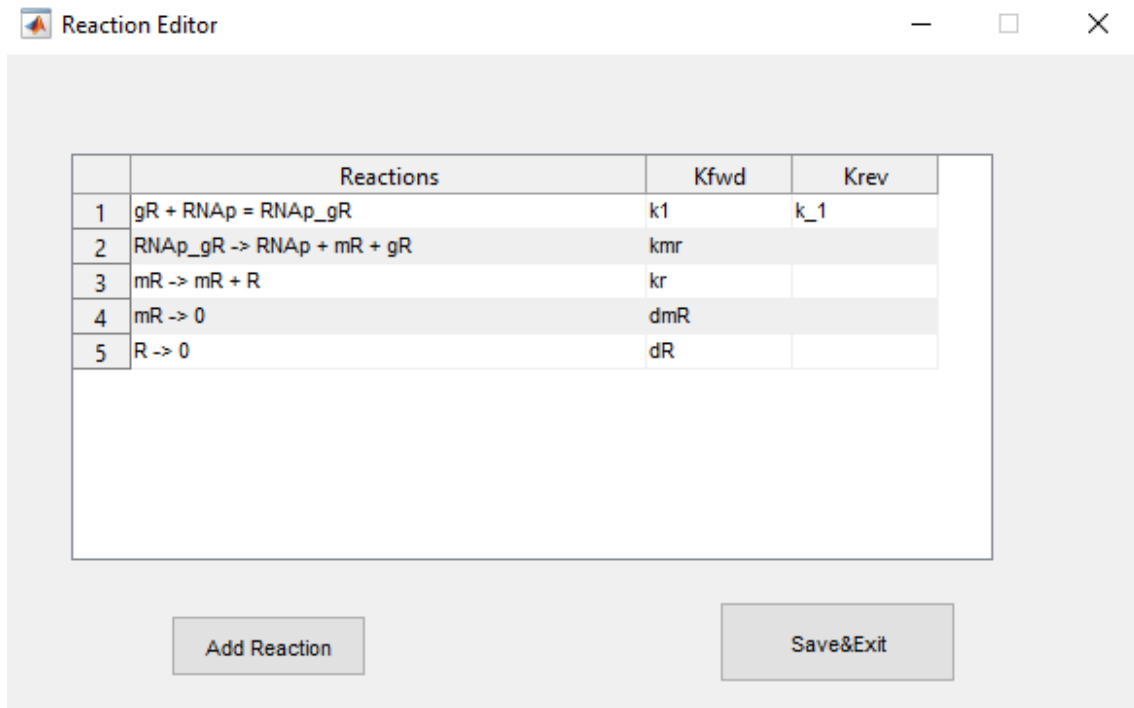


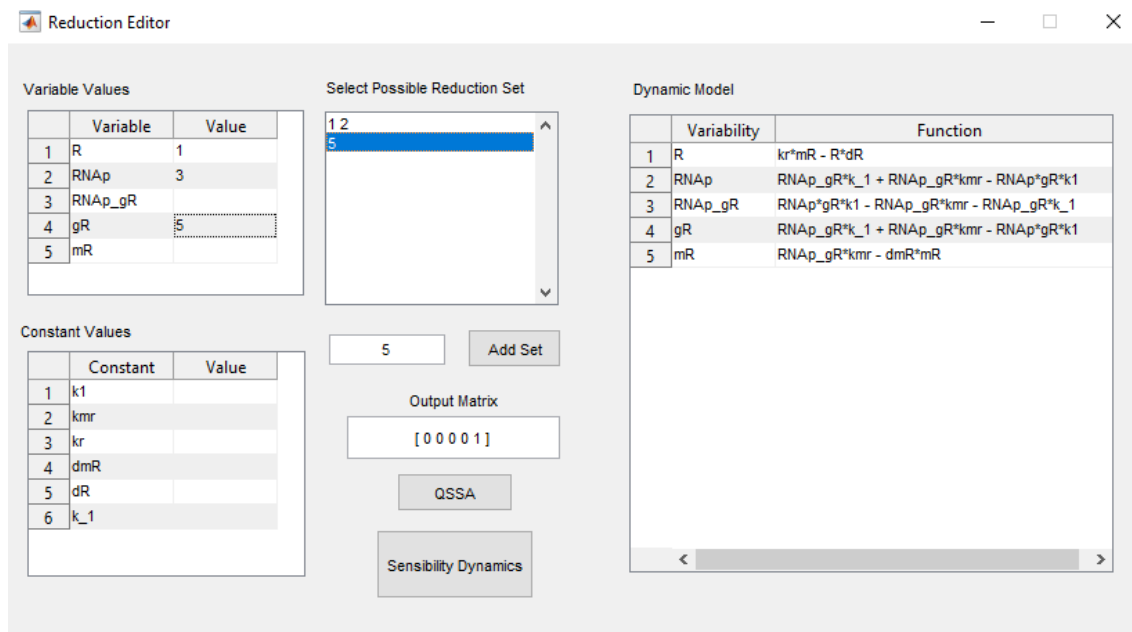
Figura 4.9: Resultado de implementación de la interfaz de *Reaction Editor*

Reduction Editor

Esta interfaz contiene como componentes de ajuste (figura 4.10):

- Dos tablas de datos para la representación de variables y constantes y edición de sus valores.
- Un *listbox* para la selección o creación de conjunto de modelos posibles a reducir.
- Varias celdas editables para la matriz de salida C, y diversos parámetros de ajuste de reducción.

Al iniciar la interfaz con la función *OpeningFcn*, se genera el modelo dinámico del sistema con el parámetro de entrada *handles.reac_table* y la función *getDynamicModel*.

Figura 4.10: Interfaz de *Reduction Editor*

```
function ReductionEditor_OpeningFcn(hObject, eventdata,
    handles, varargin)
    %Variables and Constants separation
    [handles.x handles.kfwd handles.f] =
        getDynamicModel(tab_v);

    %Tables values
    %Print table after openingFn
    set(handles.uitable1, 'Data',
        cellstr(string(handles.x)));

    set(handles.uitable1, 'RowName', num_reacs,
        'ColumnName', {'Variable', 'Value'});

    %Print table after openingFn
    set(handles.uitable2, 'Data',
        cellstr(string(handles.kfwd)));
    set(handles.uitable2, 'RowName', num_reacs,
        'ColumnName', {'Constant', 'Value'});
```

```
%Save data
guidata(hObject,handles);

uiwait
```

Reduction Editor: Output

Por último, tenemos dos *push buttons* para obtener la salida de reducción:

- **QSSA**: se abre una ventana emergente con el resultado de la aplicación del algoritmo como consecuencia del método de reducción QSSA.

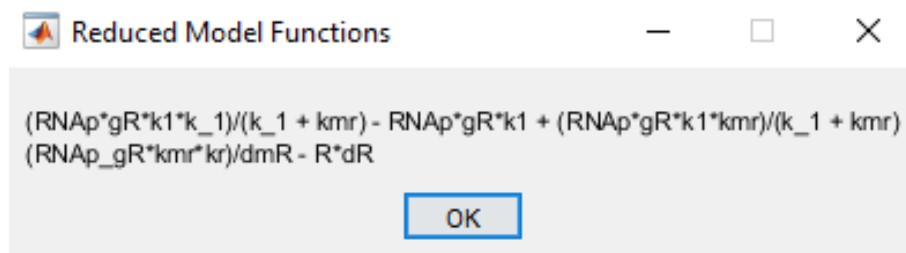


Figura 4.11: Ventana emergente con las funciones del modelo reducido

- **Sensibility Dynamics**: se abre otra ventana emergente con las diferentes tablas que relacionan las variables de estado y su sensibilidad dinámica con respecto a los parámetros.

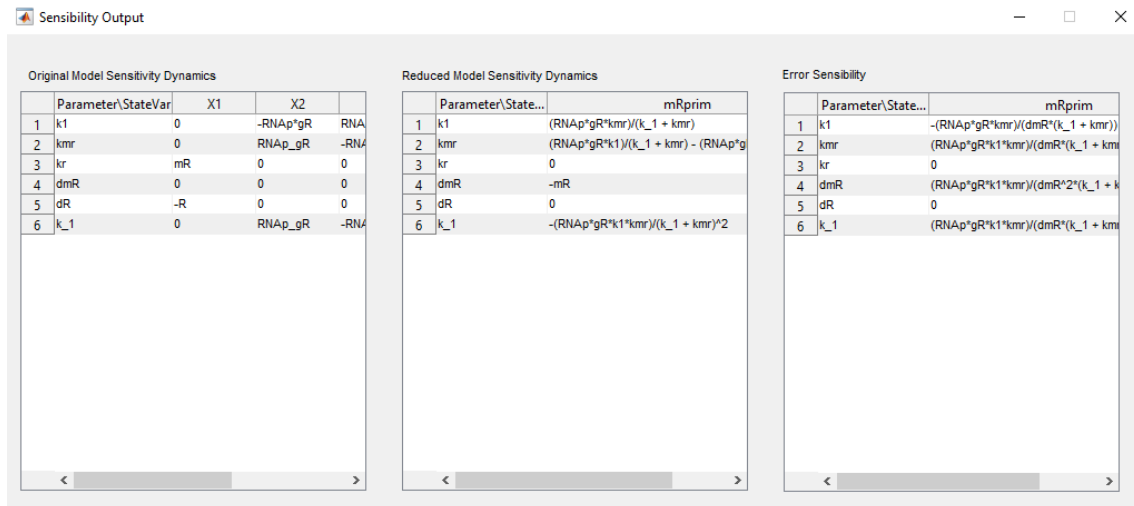


Figura 4.12: Ventana emergente con la dinámica de las sensibilidades

Plot Graph

Este componente dibuja el grafo del sistema biológico, cuya función ya ha sido explicada anteriormente en el apartado 4.2.3. Dentro de la interfaz principal, se pasa como parámetros la variable *handles.reac_plot*, que contiene los nodos y aristas del sistema, como resultado de la función *getModelData*.

```
function plotGraphButton_Callback(hObject, eventdata,
    handles)
% hObject handle to plotGraphButton (see GCBO)
% eventdata reserved - to be defined in a future
    version of MATLAB
% handles structure with handles and user data (see
    GUIDATA)

% Display model graph
figure('Name', "Model Graph");

%Check if reac_plot has been loaded
if exist('handles.reac_plot')
    plot(handles.reac_plot);
end
```

```
guidata(hObject, handles);
```


Capítulo 5

Pruebas y resultados

En este capítulo se exponen algunas pruebas realizadas con la interfaz de usuario.

5.1. Dinámica de funcionamiento

Se describe ahora el correcto uso de la herramienta desarrollada.

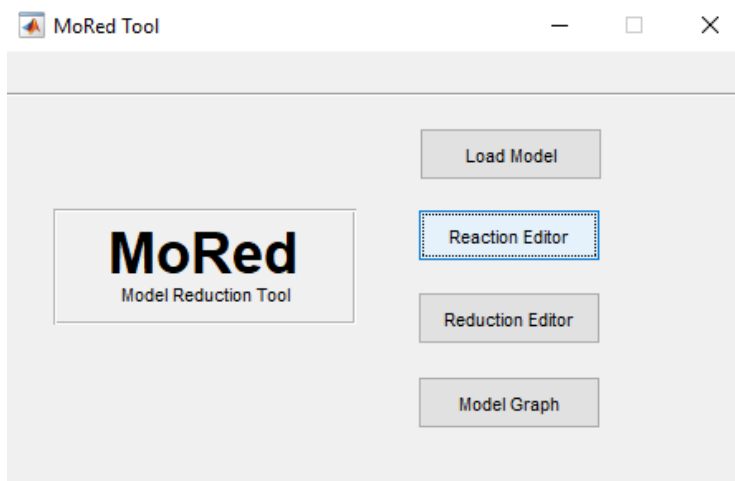


Figura 5.1: Presentación de la herramienta principal

5.1.1. Selección de sistema de reacciones o creación del mismo

En primer lugar se debe plantear un sistema de reacciones de un sistema biológico con el que trabajar. Para ello, como se ha comentado en anteriores apartados, se puede seleccionar un archivo *.csv* con la

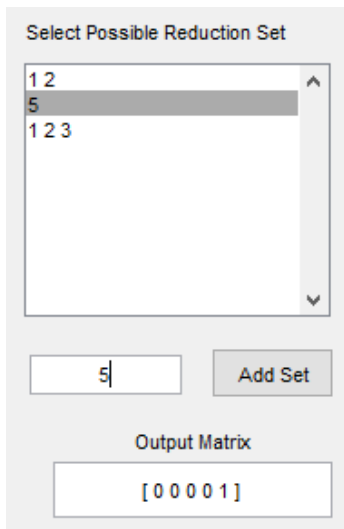
tabla de valores, con *Load Model*. En caso de no disponer de dicha tabla, la funcionalidad *Reactor Editor* permite almacenar en variable toda la edición de la misma (figura 5.2).

	Reactions	Kfwd	Krev
1	A -> B + C	k1	
2	B + R = A + D	k2	k_2
3	R -> 0		
4			

Figura 5.2: Creación de reacciones con *React Editor*

5.1.2. Reducción del sistema

Una vez obtenida la información del sistema, se puede pasar a *Reduction Editor*. En este punto, se ofrece al usuario diversa información y funcionalidad:



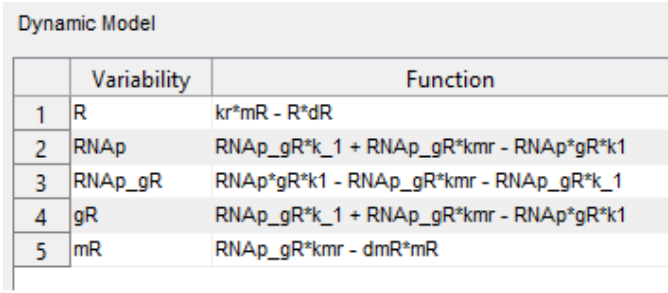
Select Possible Reduction Set

1 2
5
1 2 3

5 Add Set

Output Matrix

[0 0 0 0 1]



Dynamic Model

	Variability	Function
1	R	$kr \cdot mR - R \cdot dR$
2	RNAp	$RNAp_gR \cdot k_1 + RNAp_gR \cdot kmr - RNAp \cdot gR \cdot k1$
3	RNAp_gR	$RNAp \cdot gR \cdot k1 - RNAp_gR \cdot kmr - RNAp_gR \cdot k_1$
4	gR	$RNAp_gR \cdot k_1 + RNAp_gR \cdot kmr - RNAp \cdot gR \cdot k1$
5	mR	$RNAp_gR \cdot kmr - dmR \cdot mR$

(a) Selección de reducción posible y generación de matriz de salida

(b) Funciones dinámicas del sistema

Figura 5.3: Funciones de *Reduction Editor*

- Tablas para la asignación de valores a variables de estado y parámetros del sistema.
- Creación y selección de conjuntos de posibles variables a reducir en el sistema.

- Obtención automática y edición de la matriz de salida del sistema
- Tabla con las funciones dinámicas del sistema para las variables de estado.
- Obtención del modelo reducido y generación de la dinámica de sensibilidades del sistema con respecto a los parámetros del mismo.

El modelo reducido y las tablas con la dinámica de las sensibilidades se muestran en dos ventanas emergentes distintas.

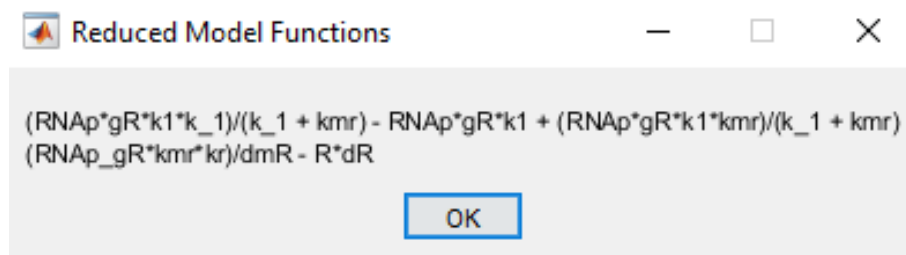


Figura 5.4: Ventana emergente con las funciones del modelo reducido

Reduced Model Sensivity Dynamics			Error Sensibility		
	Parameter\State...	mRprim		Parameter\State...	mRprim
1	k1	$(RNAp^*gR^*kmr)/(k_1 + kmr)$	1	k1	$-(RNAp^*gR^*kmr)/(dmR^*(k_1 + kmr))$
2	kmr	$(RNAp^*gR^*k1)/(k_1 + kmr) - (RNAp^*g$	2	kmr	$(RNAp^*gR^*k1*kmr)/(dmR^*(k_1 + kmr)$
3	kr	0	3	kr	0
4	dmR	-mR	4	dmR	$(RNAp^*gR^*k1*kmr)/(dmR^2*(k_1 + k$
5	dR	0	5	dR	0
6	k_1	$-(RNAp^*gR^*k1*kmr)/(k_1 + kmr)^2$	6	k_1	$(RNAp^*gR^*k1*kmr)/(dmR^*(k_1 + kmr)$

Figura 5.5: Ventana emergente con la dinámica de la sensibilidad del sistema

5.1.3. Dibujo del grafo del sistema

Como complemento de la herramienta, el usuario puede visualizar el grafo del sistema, quedando como resultado el ejemplo siguiente:

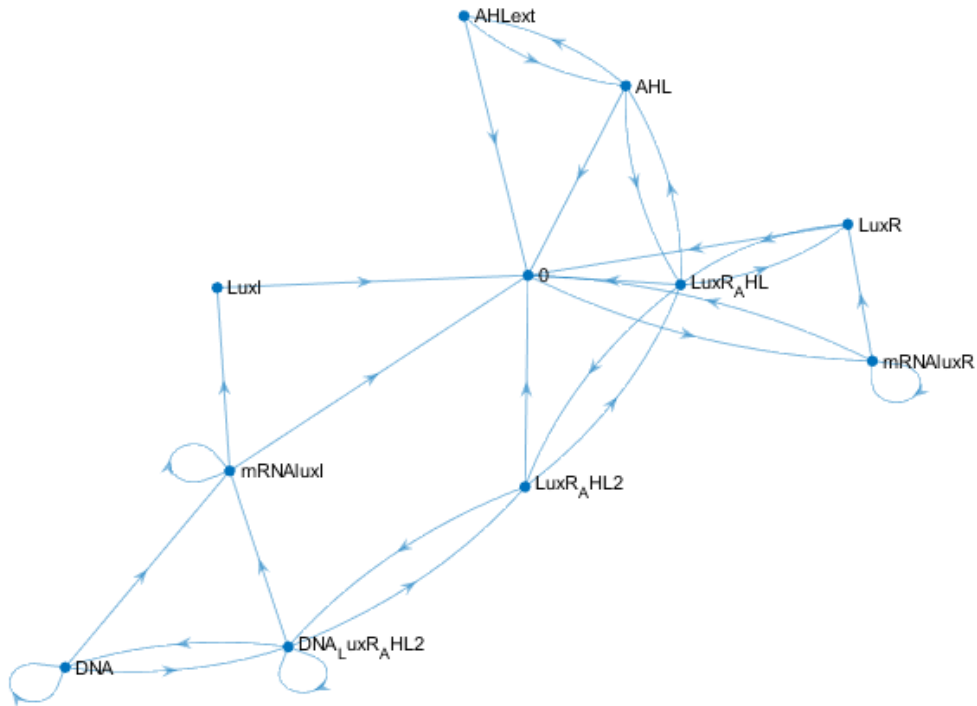


Figura 5.6: Ejemplo de grafo dibujado

5.2. Resultados

Respecto a la reducción del sistema, al seguir una resolución analítica del problema los resultados son los esperados. Sin embargo en la implementación de la interfaz es cierto que se pueden producir algunas acciones no deseadas, como por ejemplo:

- La destrucción de objetos que contiene una interfaz suele conllevar un *warning* o error en la ventana de comandos de Matlab, debido a la asincronía de los *callbacks* queriendo acceder a una variable en un momento dado. Este defecto no repercute en la información guardada o trabajada del usuario.
- Dependiendo de la longitud de la ecuación de reducción resultando, algunos resultados pueden ocultarse en las tablas de visualización si el usuario no interactúa con la tabla.

- En general, la herramienta solo evalúa el sentido matemático del sistema y de las reacciones, pero no el sentido físico, por lo que se relega al usuario gran parte del conocimiento en la materia para el uso de la herramienta.

Estos detalles pueden solventarse realizando una revisión de implementación exhaustiva, que por cuestiones de límite de tiempo y entrega no ha podido realizarse durante el desarrollo de este proyecto.

Capítulo 6

Conclusiones

6.1. Conclusión

La herramienta desarrollada en este proyecto cumple la función que se planteó al inicio del mismo, aportando esa ayuda al usuario a la hora de trabajar en Biología Sintética. Sin embargo, no todos los objetivos intermedios planteados han conseguido cumplirse de la forma pensada originalmente, y es necesario comentar algunos puntos:

- A lo largo del estudio del artículo y el desarrollo de la herramienta, se han observado diversas **erratas** en la formulación del algoritmo de reducción que han obligado a centrarse en los conceptos plasmado en el mismo, en vez de aplicar directamente la solución propuesta. Los errores divisados quedan plasmados en el Anexo 1 del documento.
- A tenor de lo anterior, la sensibilidad del error S_e se iba a llevar a cabo haciendo uso de la acotación superior propuesta en el artículo, descrita en la ecuación 4.1. Sin embargo, por problemas para la resolución simbólica de dicha propuesta, se ha trabajado el error $e = Cy - \hat{C}\hat{y}$ y su derivada con respecto a cada parámetro $\frac{\partial e}{\partial \theta_j}$, obteniendo una relación simbólica de sensibilidad del error del sistema.
- Al obtener una relación simbólica de sensibilidad, las tablas planteadas para la edición de un valor numérico de las variables y parámetros quedan en vacío en la aplicación, pero preparadas y extensibles a posteriores implementaciones del sistema de forma **numérica**.

A pesar de estos inconvenientes, la herramienta logra cumplir el cometido inicial propuesto.

A nivel personal, el proyecto ha resultado de gran interés por su especial carácter dentro de la aplicación industrial en sí. La cercanía de la implementación de la herramienta con la aplicación al mundo físico ha resultado motivador para el aprendizaje obtenido durante el desarrollo del proyecto.

6.2. Trabajos futuros

Algunas de las posibles extensiones de la herramienta podrían ser:

- Desarrollo numérico del sistema reducido obtenido
- Extensión del trabajo sobre la dinámica de sensibilidad del sistema para obtención de mayor información, como una posible simulación del mismo.
- Obtención de las ecuaciones dinámicas del sistema a raíz de modelos SBML, para extender la variedad de carga de modelos
- Optimización software de la herramienta

Bibliografía

- [1] S.J. Moore M.B. Kopniczky and P. Freemont. *Multilevel Regulation and Transactional Switches in Synthetic Biology*. IEEE Transactions on Biomedical Circuits and Systems, 2015.
- [2] C.J. Bashor D.E. Cameron and J.J. Collings. *A Brief History of Synthetic Biology*. Nature Reviews Microbiology, 2014.
- [3] Jesús Picó *et al.* *Synthetic Biology*. John Wiley & Sons, Inc., 2016.
- [4] W.M. Haddad V. Chellaboina, S.P. Bhat and D.S. Bernstein. *Modeling and Analysis of Mass-Action Kinetics*. IEEE Control Systems Magazine, 2009.
- [5] Universidad de Pamplona. *Teoría de Grafos*. unipamplona.edu.co, 2019.
- [6] M. Zaragoza M. Claverol, E. Simó. *Matemática Discreta: Teoría de Grafos*. Departamento Matemática Aplicada EPSEVG, UPC, 2019.
- [7] T. Witelski and M. Bowen. *Singular Perturbation Theory*. Scholarpedia, 2009.
- [8] B. Liu. *Computational Modeling and Analysis of Biological Pathways*. 2006.
- [9] Ayush Pandey and Richard M. Murray. *An Automated Model Reduction Tool to Guide the Design and Analysis of Synthetic Biological Circuits*. BioRxiv, 2019.
- [10] Matlab. *Creating Graphical User Interfaces*. MathWorks, 2015.

Anexo 1: Fé de erratas en el artículo principal

- En el Teorema 4, la ecuación de Lyapunov que resuelve la matriz P , viene dada por la expresión $\bar{J}^T P + P \bar{J} = -\bar{C}^T \bar{C}$.
- En el Algoritmo 1, el bucle que recorre la longitud de las variables de funciones reducidas \hat{f} , no puede iterarse con la misma variable k usada en el bucle en el cual está contenido.
- En el mismo bucle comentado en el punto anterior, la solución de las ecuaciones de *collapse* (f_c), no pueden tener una iteración de la variable i , dado que esta variable almacena un valor no iterativo y de valor alto como resultado de bucles previos. Tampoco la variable x_c .
- En la ecuación 1 del teorema 4 que indica la acotación superior de la sensibilidad del error, la matriz de derivadas parciales a la izquierda de la variable Q_s está transpuesta.