



Universidad de Castilla-La Mancha
Escuela Superior de Ingeniería Informática

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Redes de Petri, lógica difusa y simulación estocástica de rutas metabólicas. Un caso de estudio

Javier Jiménez Ruescas

Septiembre, 2022

TRABAJO FIN DE MÁSTER

Máster Universitario en Ingeniería Informática

Redes de Petri, lógica difusa y simulación estocástica de rutas metabólicas. Un caso de estudio

*Petri Nets, fuzzy logic and stochastic simulation of metabolic
pathways. A case study.*

Autor: Javier Jiménez Ruescas

Tutor: Hermenegilda Macià Soler

Co-Tutor: Edelmira Valero Ruiz

Septiembre, 2022

Declaración de autoría

Yo, Javier Jiménez Ruescas , con DNI 49212747-S, declaro que soy el único autor del trabajo fin de grado titulado “Redes de Petri, lógica difusa y simulación estocástica de rutas metabólicas. Un caso de estudio”, que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual, y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a 5 de Septiembre de 2022

Fdo.: Javier Jiménez Ruescas

Resumen

La simulación de modelos bioquímicos dinámicos, como las rutas metabólicas, puede ser un proceso bastante lento y que consume demasiados recursos de nuestros ordenadores. Esta situación se agrava aún más cuando se busca un mayor grado de realismo en estos modelos, ya que, se deben hacer más complejos, incluyendo un mayor número de ecuaciones diferenciales para su modelado o mayor número de operaciones para la obtención de parámetros de entrada como puede ser la radiación solar.

Este documento pretende ser una continuación del trabajo de Fin de Grado “*Tellurium como herramienta de modelado y testing para rutas metabólicas*”, en el que se propondrán diversas mejoras para los modelos de radiación solar propuestos anteriormente.

Además, se abordan nuevos aspectos como la simulación estocástica y la lógica difusa, surgida a través de la incertidumbre a la hora de modelar nuevos sistemas bioquímicos, validar su correcto funcionamiento y la representación de dichos modelos a través de redes de Petri.

La computación de altas prestaciones también forma parte de este nuevo trabajo, ya que, como se ha comentado anteriormente, la continua mejora de los modelos bioquímicos suponen un incremento del tiempo necesario para realizar pruebas y simulaciones, tanto por parte de las personas que estén involucradas en la parte informática, como de los expertos en el campo de la bioquímica que quieran obtener resultados.

Abstract

The simulation of dynamic biochemical models, such as metabolic pathways, can be a rather slow and resource-intensive process. This situation becomes even worse when a higher degree of realism is required in these models, as they must be made more complex, including a greater number of differential equations for their modelling or a greater number of operations to obtain input parameters such as solar radiation.

This document is intended to be a continuation of the Final Degree work ‘*Tellurium as a modelling and testing tool for metabolic pathways*’, in which several improvements to the previously implemented solar radiation models will be proposed.

In addition, new aspects such as stochastic simulation and fuzzy logic are addressed, arising from uncertainty when modelling new biochemical systems, validating their correct functioning and the representation of these models through Petri nets.

Supercomputing is also part of this new work, because, as mentioned above, the continuous improvement of biochemical models increases the time needed to perform tests and simulations, both for people involved in computer science and for experts in the field of biochemistry who want to obtain results.

Agradecimientos

Quiero dar las gracias a todos aquellos que han formado parte de este año del Máster Universitario de Ingeniería Informática.

En primer lugar a mis tutoras Hermenegilda Macià Soler y Edelmira Valero Ruiz, por confiar en mí, seguir un año más con este proyecto y acogerme en el laboratorio de química de la Agrupación Politécnica de Albacete para la realización de mis prácticas empresariales. También a Rosa y Rebeca, por ayudarme durante dichas prácticas, sin vosotras este trabajo no hubiese sido posible.

También quiero dar las gracias a mi familia y amigos, por apoyarme durante todo el año y darme ánimos para acabarlo, en especial a Sara y Elena, por hacer tan amenas las tardes de clase.

Muchas gracias a todos.

Índice general

1	Introducción	1
1.1	Motivación	1
1.2	Objetivos	2
1.3	Competencias.	2
1.4	Estructura de la memoria	3
2	Antecedentes y estado de la cuestión	5
2.1	Trabajo previo	5
2.1.1	Tellurium	5
2.1.2	Modelos de radiación solar	6
2.1.3	Ciclos metabólicos	11
2.2	Simulación estocástica	14
2.3	Redes de Petri	15
2.4	Lógica difusa y Snoopy	17
3	Metodología y entorno de trabajo	21
4	Desarrollo e implementación	23
4.1	Modelos de radiación solar (Python).	23
4.1.1	Modelo Épsilon-Delta	23
4.1.2	Modelo Envolvente	29
4.2	Modelo Fuzzy (Snoopy)	31
4.3	Modelo Estocástico (Python)	36

5 Experimentos y resultados.....	39
5.1 Pruebas de los modelos de radiación solar	39
5.1.1 Modelo Épsilon-Delta	39
5.1.2 Modelo Envolvente	43
5.2 Pruebas de los modelos Fuzzy y estocásticos	45
6 Conclusiones y propuestas.....	53
A Ficheros Adjuntos	55
Referencia bibliográfica.....	59

Índice de figuras

2.1	Temperaturas 7 primeros días (Albacete)	8
2.2	Temperaturas año completo (Albacete)	8
2.3	Radiación 7 primeros días (Albacete)	9
2.4	Radiación año completo (Albacete)	9
2.5	Radiación obtenida haciendo uso de las funciones 2.6	11
2.6	Diagrama del ciclo de Calvin-Benson	12
2.7	Diagrama del ciclo de GAR	13
2.8	Ejemplo del funcionamiento de una Petri Net	16
2.9	Red de Petri para el modelo del Ciclo de Calvin (Zhu-2009)	17
2.10	Representación gráfica de un número Fuzzy	18
3.1	Diagrama de la metodología SCRUM	21
4.1	Agravamiento del error producido respecto a la latitud	24
4.2	Curva h _{sol} resultante de la interpolación	25
4.3	Radiación Albacete normalizada	26
4.4	Cálculo de la radiación con delta y épsilon	27
4.5	Envolvente radiación Albacete	30
4.6	Menús edición Snoopy 2.6	32
4.7	Fragmento Red de Petri	33
4.8	Opciones del elemento Place	34
4.9	Opciones del elemento Transition	34
4.10	Opciones del elemento Arc	34
4.11	Menú de constantes Snoopy	35
5.1	Fotoperiodo antes del ajuste Albacete	40
5.2	Comparativa ajustes Albacete	40
5.3	Fotoperiodo antes del ajuste Reykjavik	42
5.4	Mejora Reykjavik	42
5.5	Radiación aleatorizada Albacete	43
5.6	Comparativa envolvente Albacete	44

5.7	Comparativa radiación (Albacete)	45
5.8	Comparativa APX (Albacete)	45
5.9	Comparativa ajuste Reykjavik	46
5.10	Gráfica interactiva Tellurium	46
5.11	Comparativa incertidumbre RuBP	47
5.12	Comparativa incertidumbre DPGA	48
5.13	Comparativa simulaciones DPGA	48
5.14	Media de las simulaciones estocásticas vs determinista DPGA	49
5.15	Configuración simulador Snoopy	50
5.16	Resultados simulación Snoopy	50
5.17	Resultados RuBP Snoopy	51

Índice de tablas

2.1	Valores para la ecuación 2.2	7
2.2	Puntos de referencia curva y1 (hsol)	10
2.3	Puntos de referencia radiación solar normalizada	10
5.1	Resultados ajustes Albacete	41

1. Introducción

1.1. Motivación

Tras la elaboración del Trabajo de Fin de Grado se observó que los modelos bioquímicos que normalmente son utilizados para las investigaciones son, en la gran mayoría de los casos, modelos simples o versiones incompletas de los mismos, debido a la complejidad que supone realizar simulaciones de los mismos cuando su grado de complejidad comienza a ser elevado. Por lo general, cuanto más completo es el modelo, más realismo y precisión se obtiene en los resultados obtenidos, pero los tiempos de simulación pueden aumentar considerablemente.

Conociendo este problema, se decidió realizar diferentes versiones de los modelos de radiación creados previamente, dado que el parámetro de la radiación es calculado en cada instante de tiempo de simulación, y mejorando la eficiencia de estos modelos, conseguimos mejorar el tiempo de ejecución necesario para realizar las simulaciones en los nuevos modelos bioquímicos.

Además de esto, se observó que los modelos tienen una cantidad muy elevada de parámetros, que en la gran mayoría de los casos, son los expertos en el campo de la bioquímica los que deben establecer los valores oportunos en función del modelo para obtener resultados válidos. La elección del valor de cada uno de los parámetros puede resultar un verdadero quebradero de cabeza, ya que, un cambio muy pequeño en el valor numérico de un parámetro puede afectar a toda la ruta metabólica al estar gran parte de ella relacionada entre sí.

A estas razones para la elección del desarrollo de este trabajo hay que sumarle el valor práctico personal, en el que se pretenden plasmar los conocimientos adquiridos durante el Máster Universitario de Ingeniería Informática y la mejora personal desde el trabajo presentado anteriormente.

1.2. Objetivos

Los objetivos principales que se desarrollarán a lo largo del Trabajo de Fin de Máster serán los mostrados a continuación, atendiendo a la mejora y ampliación del Trabajo de Fin de Grado mencionado anteriormente:

- Mejora de los modelos de radiación solar con el objetivo de incrementar el rendimiento de las simulaciones.
- Modelado de rutas metabólicas y su simulación estocástica tomando como referencia el Ciclo de Calvin.
- Estudio del comportamiento dinámico del modelo atendiendo a la incorporación de datos reales de radiación solar con simulación estocástica.
- Modelado con redes de Petri difusas del modelo del Ciclo de Calvin, teniendo en cuenta la incertidumbre en parámetros.

1.3. Competencias

Una vez finalizado el Trabajo de Fin de Máster se pretenden obtener las siguientes competencias, las cuales se recogen en la página web del Trabajo de Fin de Máster:

- Capacidad para el análisis de las necesidades de información que se plantean en un entorno y llevarlo a cabo en todas sus etapas el proceso de construcción de un sistema de información.
- Capacidad para comprender y aplicar conocimientos avanzados de computación de altas prestaciones y métodos numéricos o computacionales a problemas de ingeniería.
- Capacidad para aplicar métodos matemáticos, estadísticos y de inteligencia artificial para modelar, diseñar y desarrollar aplicaciones, servicios, sistemas inteligentes y sistemas basados en el conocimiento.
- Realización, presentación y defensa de un ejercicio original realizado individualmente ante un tribunal universitario, consistente en un proyecto integral de Ingeniería en Informática de naturaleza profesional en el que se sintetizan las competencias adquiridas en las enseñanzas.

1.4. Estructura de la memoria

Este documento se encuentra dividido en diferentes capítulos, en los cuales se abordan tanto aspectos teóricos necesarios para comprender la naturaleza del proyecto como prácticos, resultantes de las fase de implementación y análisis del trabajo realizado.

Se ha comenzado realizando una introducción, en la cual se ha explicado la motivación que ha llevado a la elaboración del Trabajo de Fin de Máster, los objetivos y las competencias que se pretenden conseguir con su desarrollo.

Tras este apartado abordaremos el capítulo de *“Antecedentes y estado de la cuestión”*. En dicho capítulo se hará una introducción con un resumen del trabajo previo realizado en el Trabajo de Fin de Grado en el que se basa este nuevo Trabajo de Fin de Máster, haciendo hincapié en la herramienta Tellurium, los modelos de radiación solar implementados y el Ciclo de Calvin-Benson. También se realizará una introducción a las redes de Petri, la simulación estocástica, lógica difusa y a la herramienta Snoopy.

A continuación se describirá la metodología de trabajo utilizada y el entorno de trabajo en el cual se ha desarrollado el Trabajo de Fin de Máster, puesto que este ha sido elaborado junto a un equipo multidisciplinar.

Una vez finalizada la parte teórica, se continuará con el capítulo de *“Desarrollo e implementación”*.

En este capítulo se detallará como se han implementado los nuevos modelos de radiación solar para mejorar el rendimiento en la carga de dichos modelos para su simulación. Por otra parte, se explicará también el desarrollo y metodología seguida a la hora de implementar el nuevo ciclo de Calvin-Benson utilizado para la sección de simulación estocástica y con parámetros Fuzzy y el desarrollo y configuración de este tipo de simulaciones junto a las redes de Petri.

Seguido de esto, se comentarán los resultados obtenidos tras realizar diversas pruebas con los modelos implementados, realizando unas comparaciones respecto a las antiguas implementaciones para ver la evolución del trabajo realizado, y para finalizar la memoria se expondrán las conclusiones obtenidas del desarrollo, así como el planteamiento de futuras mejoras que puedan brindar al trabajo de una mayor funcionalidad.

Al final de esta memoria se incluye un anexo en el cual se pondrá encontrar la información necesaria para la instalación de las herramientas utilizadas en este Trabajo de Fin de Máster y la información relacionada con los ficheros adjuntos a la memoria junto a un enlace de GIT para su descarga.

2. Antecedentes y estado de la cuestión

2.1. Trabajo previo

Como se ha comentado en el capítulo anterior, este Trabajo de Fin de Máster, es una continuación y ampliación del Trabajo de Fin de Grado titulado “*Tellurium como herramienta de modelado y testing para rutas metabólicas*” [Jiménez,].

En dicho trabajo se comenta la complejidad del desarrollo y modelado de sistemas bioquímicos dinámicos, así como la necesidad de usar herramientas especializadas de alto nivel para obtener resultados gráficos. Como solución a esto se propuso el estudio y utilización de la herramienta Tellurium [Medley et al., 2018] y el lenguaje de modelado Antimony [Smith et al., 2009], el cual permitía hacer una representación rápida y sencilla de las reacciones que intervendrían en el modelo bioquímico.

También, haciendo uso de Python, se generaron varios modelos de radiación solar que intervienen en los modelos bioquímicos, los cuales se detallan más adelante debido a que son utilizados nuevamente para este Trabajo de Fin de Máster. Estos modelos de radiación se implementaron con el objetivo de realizar simulaciones para estudiar el efecto de dicha radiación sobre los organismos foto-sintéticos.

2.1.1. Tellurium

Tellurium es un entorno Python creado para el modelado dinámico reproducible de redes biológicas [Tel, 2022]. Este dispone de una versión con interfaz para el usuario, Tellurium Notebook, muy similar aplicaciones como Jupyter, la cual resulta muy sencilla e intuitiva de usar tanto para usuarios del campo de la informática como para aquellos que no lo son.

Una de las grandes ventajas de hacer uso de Tellurium como entorno para programar el modelado de los sistemas bioquímicos es el uso de Antimony [Ant, 2022][Smith et al., 2009]. Este lenguaje derivado de SBM [SBM, 2022], hace uso de modelo de definición de los elemento basado en texto, lo que es mucho más legible y fácil de entender por los expertos

del dominio que trabajen finalmente con los archivos del modelo y no tienen una gran experiencia en el campo de la programación. También nos ofrece la ventaja de eliminar por completo las etiquetas del código SBML, ya que al estar basado en XML, en modelos de gran tamaño, realizar una modificación en el código o simplemente comprender el modelo, puede resultar un tarea bastante complicada.

2.1.2. Modelos de radiación solar

A continuación se explicará de manera resumida el funcionamiento de los modelos de radiación solar desarrollados durante el Trabajo de Fin de Grado, los inconvenientes y ventajas que se encontraron en cada uno de ellos y los cambios realizados (en el caso del modelo de Luz AM).

Cabe destacar que los resultados obtenidos de estos modelos han sido recogidos en el artículo *“Simulating Solar Irradiance through AM Wave Equations for Metabolic Pathways”*, el cual se presentó en la conferencia BIOTECHNO 2022 (Venecia) [Esquinas-Ariza et al., 2022] [Gersbeck-Schierholz, 2022]

Modelo de radiación solar a partir de temperaturas

Este modelo fue propuesto para situaciones en las que únicamente se conocían los datos de temperatura mínima y máxima diaria y que a través de una serie de ecuaciones, mostradas a continuación, se realizaba un cálculo bastante aproximado a la temperatura en cada instante de tiempo para un año completo.

La idea del modelo está basada en el artículo *“New algorithm for generating hourly temperature values using daily maximum, minimum and average values from climate models”* [Chow and Levermore, 2007], en el cual se dispone de las ecuaciones necesarias para su implementación.

Para calcular la temperatura en el instante de tiempo entre la hora de temperatura mínima y máxima de cada día se hacía uso de las siguientes ecuaciones (2.1):

$$\begin{aligned} f_1 &= \frac{\cos\left(\frac{\pi(hrd-H_{min})}{H_{max}-H_{min}}\right) + 1}{2}; \\ f_2 &= 1 - f_1; \\ T_1 &= f_1 \cdot T_{min} + f_2 \cdot T_{max}; \end{aligned} \tag{2.1}$$

donde hrd corresponde a la hora del día, H_{min} y H_{max} a la hora de temperatura mínima y máxima respectivamente, T_{min} y T_{max} a la temperatura mínima y máxima y T_1 a la temperatura calculada en °C en un momento específico de tiempo.

$(\epsilon > 0) \leq hrd \leq hmin$	$hmax \leq hrd \leq 24.0$
$T_{next} = T_{min}(j)$	$T_{next} = T_{min}(j+1)$
$T_{prev} = T_{max}(j-1)$	$T_{prev} = T_{max}(j)$
$H_{next} = H_{min}(j)$	$H_{next} = H_{min}(j+1)$
$H_{prev} = H_{max}(j-1)$	$H_{prev} = H_{max}(j)$

Tabla 2.1: Valores para la ecuación 2.2

Para la zona de la curva de temperatura correspondiente a los instantes de tiempo anteriores a H_{min} y posteriores a H_{max} se hacía uso de la siguiente ecuación (2.2)

$$T_2 = \left(\frac{T_{next} + T_{prev}}{2} \right) - \left[\left(\frac{T_{next} - T_{prev}}{2} \right) \cdot \cos \left(\frac{\pi(hrd - H_{prev})}{H_{next} - H_{prev}} \right) \right] \quad (2.2)$$

donde T_2 corresponderá a la temperatura en °C para el instante de tiempo indicado y el resto de variables las indicadas en la tabla 2.1 en función de si hrd es anterior a H_{min} o posterior a H_{max} , considerando j como el día del año en el que queremos calcular la temperatura.

Estas ecuaciones, nos devolverán como resultado unos datos y curvas de temperatura como las que se aprecian en las siguientes imágenes (2.1 y 2.2) y a través de las cuales podremos calcular otros factores como presión atmosférica, saturación de vapor, etc.¹ que serán necesarios para el cálculo final de la radiación.

En las imágenes mostradas a continuación (2.3 y 2.4) se puede apreciar la radiación obtenida con dicho modelo, en el cual se pueden apreciar los ciclos de día y noche que se producen a lo largo de una semana. Más adelante se desarrollaran nuevos modelos basados en la idea de luz AM, en los cuales se ha trabajado para que el foto-periodo de cada día sea lo más preciso posible.

Este modelo nos ofrece la ventaja de arrojar unos datos de radiación muy similares a los reales, dotando a la simulación de un realismo bastante importante, además de la facilidad que hay actualmente para obtener datos de temperaturas diarias de casi cualquier latitud del planeta frente a los datos de radiación solar.

Sin embargo, cuando se realizaron las primeras pruebas de este modelo basado en temperaturas en el Trabajo de Fin de Máster, se hizo uso del ciclo redox glutation-ascrobato (GAR), el cual podía ser simulado para un año completo en un tiempo de estimado de entre 35 y 40 minutos (incluyendo carga y simulación del modelo).

Para el nuevo modelo, una versión ampliada de este, como veremos más adelante, haciendo uso del computador GALGO se han llegado a superar las 8 horas de tiempo de carga y simulación.

Como se puede apreciar, el coste de un modelo bastante realista, es el tiempo necesario para su simulación, el cual se va agravando cuanto más complejo es el ciclo o ruta metabó-

¹Todas las ecuaciones necesarias para el cálculo de estas variables se encuentran disponibles para su consulta en la memoria del TFG adjunto

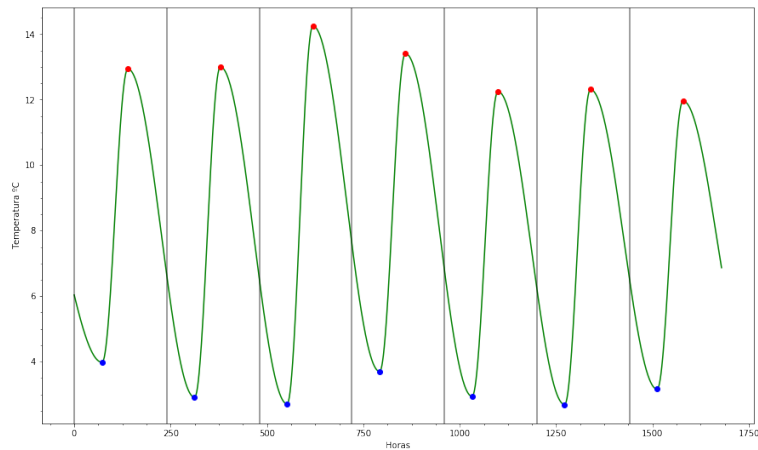


Figura 2.1: Temperaturas 7 primeros días (Albacete)

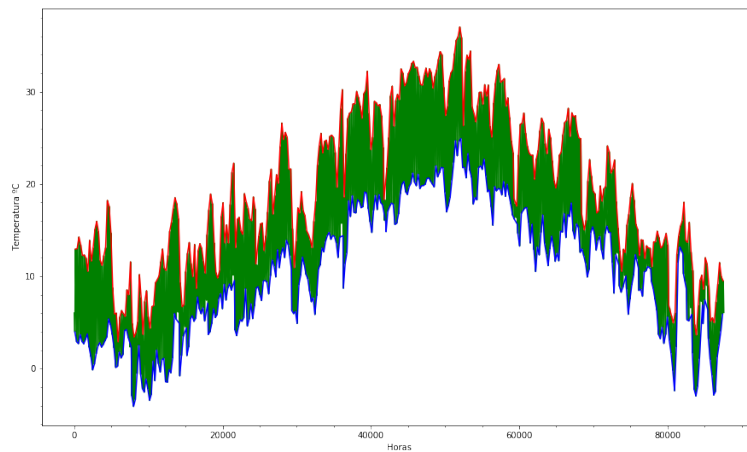


Figura 2.2: Temperaturas año completo (Albacete)

lica que se desea estudiar y puede afectar negativamente a los expertos en bioquímica que desean hacer uso de estas simulaciones para comprobar sus modelos. Para ello se recuperó el trabajo realizado en en modelo de Luz AM mostrado a continuación, con el propósito de mejorarlo y dotarlo de mayor realismo y precisión para que pudiese ser utilizado de la misma manera que este modelo que se acaba de presentar.

Modelo de radiación solar de amplitud modulada (Luz AM)

El modelo de amplitud modulada o Luz AM es un modelo que se diseñó con el propósito de replicar la radiación solar de manera aproximada conociendo una cantidad de valores de radiación muy reducida para una latitud específica, (en este caso se implementó el modelo para Albacete) y los incrementos de duración del día y la noche haciendo uso de una serie de ecuaciones.

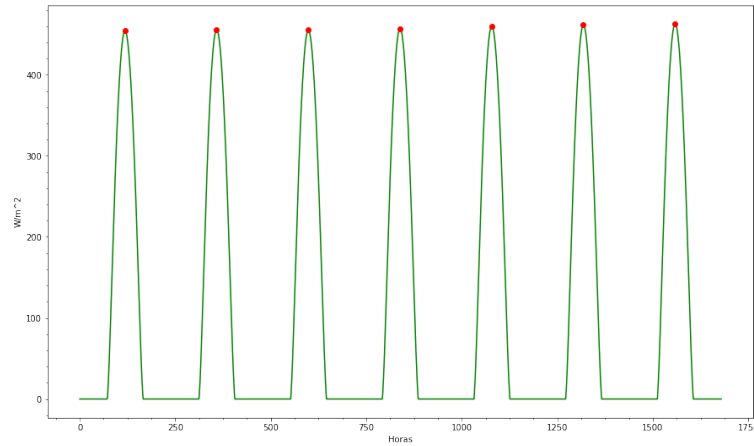


Figura 2.3: Radiación 7 primeros días (Albacete)

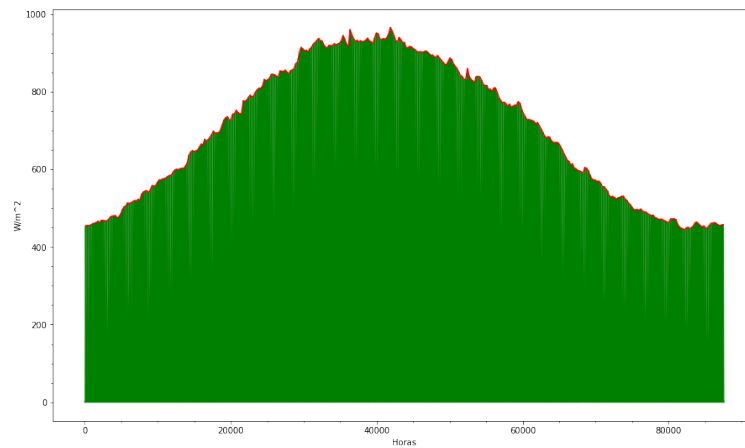


Figura 2.4: Radiación año completo (Albacete)

Los primeros datos que debemos obtener son los valores de una curva, que debe pasar por los siguientes puntos (Tabla 2.2), donde time son las horas del año e y1 el incremento del periodo nocturno. Se puede apreciar que en las horas de invierno obtenemos el valor más alto (0.25), significando que el periodo de noche será más largo que en otros instantes de tiempo. En el caso de los valores 0 nos encontramos con que los periodos de día y noche deberían de tener la misma duración.

Para obtener los valores que indican el cambio del fotoperiodo durante un año completo haremos uso de un ajuste de cosenos, el cual hará uso de la siguiente fórmula (2.3):

$$h_{sol} = a_0 + a_1 \cdot \cos\left(\frac{2\pi}{8760} \cdot time\right) \quad (2.3)$$

siendo $a_0 = 0$ y $a_1 = 0.25$ los valores necesarios para el caso de Albacete.

A continuación se hace uso de unos nuevos puntos de referencia denominados y3, mostrados en la tabla 2.3 los cuales se encuentran en función de time y representan la radiación

time	y1
0	0.25
2190	0
4380	-0.25
6570	0
8760	0.25

Tabla 2.2: Puntos de referencia curva y1 (hsol)

ción normalizada. Con estos valores crearemos una nueva curva de ajuste denominada Ac2 (ecuación 2.4), para la que obtenemos los valores $a_0=0.65$ y $a_1 = -0.1$ la cual se calculará con los valores de $y2 = y1 + y3$ y haciendo uso de la ecuación 2.3 mostrada anteriormente.

time	y3
0	0.3
2190	0.65
4380	1
6570	0.65
8760	0.3

Tabla 2.3: Puntos de referencia radiación solar normalizada

$$Ac2 = 0.65 - 0.1 \cdot \cos\left(\frac{2\pi}{8760} \cdot time\right) \quad (2.4)$$

Con dicha curva declaramos una nueva función s1, la cual tendrá una frecuencia de 24 horas.

$$s1 = Ac2 \cdot \sin\left(\frac{2\pi}{24} \cdot time + \frac{3\pi}{2}\right) \quad (2.5)$$

Conociendo estos datos ya podemos obtener nuestra radiación final haciendo uso de las siguientes fórmulas (2.6). Hay que tener en cuenta que al trabajar con senos y cosenos obtendremos valores negativos, por lo que la fórmula de s se encargará de realizar el ajuste oportuno, tal y como se puede ver en las siguientes imágenes (Figura 2.5).

$$s2 = s1 - h_{sol}$$

$$s = \frac{s2 + |s2|}{2} \quad (2.6)$$

Tras esto solo quedaría aplicar un factor oportuno a la radiación solar obtenida para obtener los datos próximos a los reales.

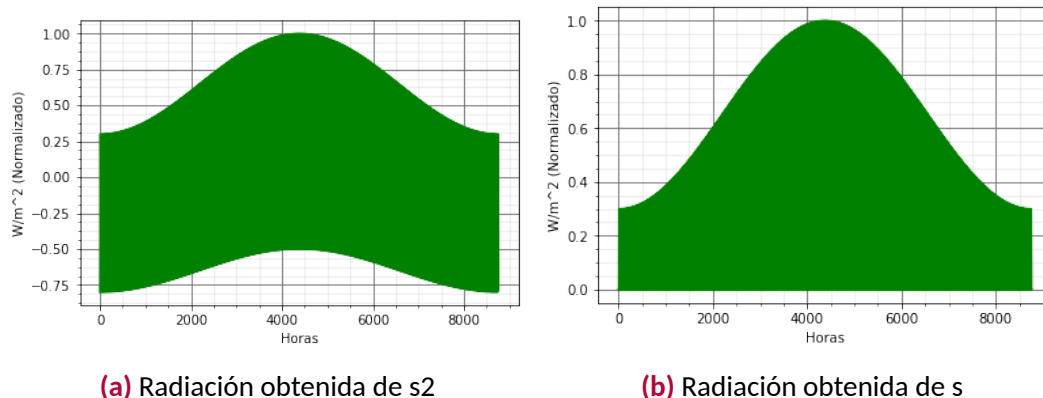


Figura 2.5: Radiación obtenida haciendo uso de las funciones 2.6

La principal ventaja del uso de este modelo es que es bastante simple de calcular y el tiempo necesario para cargar y simular un modelo biológico cuando se encuentra implementado en este es bastante más rápido que el modelo de radiación basado en temperaturas descrito anteriormente.

La parte negativa es que perdemos el realismo obtenido por el anterior modelo, ya que con este se conseguía menor precisión frente a los resultados reales. Además, el modelo de Luz AM es un modelo que está diseñado para funcionar únicamente en una localización, por lo que no se podía hacer pruebas en otras latitudes sin estar cambiando constantemente el modelo. Ante esta problemática se decidió como trabajo futuro en el Trabajo de Fin de Grado realizar mejoras en el realismo de los datos obtenidos en el modelo y automatizar el mismo para que sirva en cualquier latitud. Estas mejoras se detallará más adelante en el apartado de desarrollo e implementación.

2.1.3. Ciclos metabólicos

A continuación se mostrarán los dos ciclos metabólicos que han sido utilizados para el Trabajo de Fin de Máster, los cuales han sido implementados, como se verá en el apartado “Desarrollo e implementación”, a través de modelos matemáticos basados en sistemas de ecuaciones diferenciales ordinarias y traducidas posteriormente al lenguaje de marcado Antimony para su simulación en Tellurium.

Ciclo de Calvin-Benson

Tal como se comentó en el Trabajo de Fin de Grado [Jiménez,], el modelo del Ciclo de Calvin-Benson era una de las propuestas de trabajo futuro y sobre el cual se ha centrado este trabajo en los apartados de simulación estocástica y lógica difusa. En dicho trabajo ya se introdujo la parte teórica de este ciclo, en la cual se comentaba lo siguiente:

El ciclo de Calvin-Benson es una ruta metabólica en la cual, los organismos vegetales fijan el CO_2 asimilado a través de los estomas de las hojas en moléculas orgánicas en forma de glucosa, lo que les permite generar energía.

Dentro de este proceso en la ruta metabólica existen varias etapas, explicadas en el libro “*Concepts of Biology*” [Fowler et al.,] y las cuales se muestran a continuación:

1. **Fijación:** Una molécula de CO_2 se combina con una molécula aceptora de RuBP para dar lugar a un compuesto de seis carbonos. Este compuesto se escinde en dos de tres carbonos, formando así el compuesto 3-PGA. Esta reacción es catalizada por la enzima RuBisCO, localizada en los cloroplastos de la planta.
2. **Reducción:** El compuesto NADPH y el ATP donan electrones y se reducen a un compuesto intermediario de tres carbonos para formar el compuesto GAP (molécula de azúcar de tres carbonos). En este punto del proceso una molécula de CO_2 ya formaría una parte de la molécula de glucosa.
3. **Regeneración:** Las moléculas de GAP que no son convertidas en glucosa se reciclan haciendo uso del compuesto ATP para generar el aceptor RuBP, que se usará de nuevo al comienzo del ciclo y permitirá a la planta terminar de formar la molécula de glucosa.

A continuación, en la siguiente figura (2.6), se puede observar un esquema que resume de manera visual las etapas explicadas anteriormente.

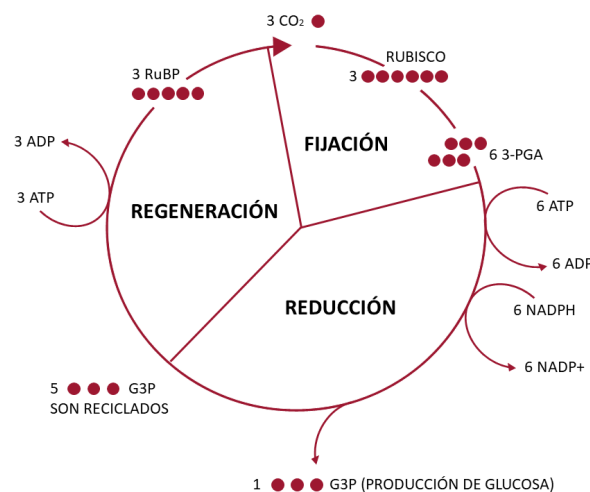


Figura 2.6: Diagrama del ciclo de Calvin-Benson

Tanto este ciclo, como el de GAR que se verá a continuación, han sido desarrollados a partir de un modelo matemático basado en ecuaciones diferenciales ordinarias, con el fin de simular el comportamiento de las especies involucradas en el ciclo. En el caso del ciclo de

Calvin de manera estocástica y en el de GAR de manera deterministas frente a una entrada variable estocástica de radiación.

Ciclo de GAR

El ciclo de Redox Glutation-Ascorbato (GAR) es un red de reacciones espontáneas, fotoquímicas y enzimáticas, cuyo objetivo es realizar una detoxificación del peróxido de hidrógeno en los cloroplastos de los organismos vegetales, actuando así como un mecanismo de defensa ante la oxidación.

El estrés lumínico es un factor bastante importante en este ciclo, dado que la realización de la fotosíntesis está regulada por los ciclos de luz y oscuridad a lo largo del día, y que varía constantemente a lo largo del año. Además, variables como la latitud, altitud, la estación del año o incluso la contaminación que podemos encontrar en el aire pueden afectar a la recepción de la luz solar, haciendo que está varíe constantemente.

Este ciclo ya se encontraba implementado previamente y durante la elaboración de este proyecto ha sufrido diversas modificaciones para ir aumentado su realismo y complejidad. Se hará uso de este ciclo para comprobar el funcionamiento de los nuevos modelos de radiación solar y verificar la correcta implementación del mismo.

Dicho ciclo fue elaborado a partir de la información propuesta por el artículo “*Modeling the ascorbate-glutathione cycle in chloroplasts under light/- dark conditions*” [Valero et al., 2016], en el cual se muestra su diagrama original (Fig 2.7) y modificado en colaboración con el laboratorio de química de la Agrupación Politécnica de Albacete.

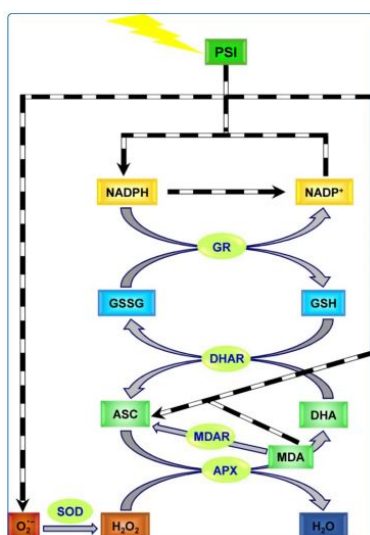


Figura 2.7: Diagrama del ciclo de GAR

2.2. Simulación estocástica

Como se comentaba al principio de este documento, uno de los objetivos de este Trabajo de Fin de Máster era la incorporación de la simulación estocástica al trabajo realizado previamente.

Para comenzar, debemos entender que una simulación estocástica es aquella que presenta un comportamiento no determinista, es decir, el resultado obtenido va a depender de una serie de factores aleatorios que impedirán predecir el comportamiento de nuestro sistema.

En los sistemas biológicos modelados en este trabajo dependemos fuertemente de la radiación calculada para realizar un estudio del mismo, pero esta radiación puede verse alterada por diversos eventos o factores. Algunos de estos eventos pueden ser la aparición de nubes, partículas de polvo suspendidas en el aire, etc. que pueden provocar que la radiación no llegue hasta la superficie terrestre. Esto será representado a través de eventos aleatorios, con los que mediante el uso de un factor en nuestra radiación, modificaremos su valor final.

Todos estos factores hacen que la radiación se convierta en una variable estocástica o aleatoria y al tener una conexión tan fuerte con el resto del modelo, hará que este cambie constantemente al realizar diferentes simulaciones.

Por otra parte, en los sistemas biológicos, si pasamos a considerar el nivel de moléculas, algoritmos como el de “Gillespie” [Gillespie, 1977], destinados a la simulación estocástica, tienen en cuenta posibles fluctuaciones y ruido debido a las moléculas que interactúan con el entorno, haciendo así, un resultado impredecible en cada ejecución al hacer uso de este algoritmo. Estos sistemas se caracterizan por sus elevados niveles de fluctuación al trabajar con un bajo número de moléculas. Este tipo de simulación será implementada más adelante en Tellurium, modificando los parámetros del motor de simulación para que haga uso del algoritmo Gillespie mencionado anteriormente.

Dicho algoritmo fue diseñado para explorar el espacio de estados asociado al sistema de ecuaciones diferenciales, teniendo como base el “algoritmo de Monte Carlo” para generar trayectorias de cadenas de Markov para la interacción de las moléculas dentro de un recipiente de volumen fijo. A continuación se muestran las fases que sigue el algoritmo para su ejecución, las cuales han sido extraídas del documento *“Modelos estocásticos. Algoritmo de Gillespie. Extensiones al marco de los sistemas P”* “del Dpto. de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Sevilla [de J. Pérez Jiménez,].

El algoritmo tomará como entrada un medio m , que albergará diferentes reactivos pertenecientes a las reacciones químicas r_n de las que se conocen sus propensidades p_n (probabilidad de que la reacción se ejecute en un intervalo de tiempo definido). Una vez obtenidos estos datos sigue el siguiente proceso:

1. Calcular $p_o = \sum_{j=1}^n p_j$, siendo p_j las propensidades de las reglas.
2. Generar dos números aleatorios a_1 y a_2 en el intervalo (0,1).
3. Calcular el tiempo de espera para la siguiente reacción mediante $\tau = \frac{1}{p_o} \ln\left(\frac{1}{a_1}\right)$
4. Seleccionar el índice j_0 de la siguiente reacción química, teniendo en cuenta que verifique $\sum_{k=1}^{j_0-1} p_k < a_2 \cdot p_o \leq \sum_{k=1}^{j_0} p_k$
5. Devolver (τ, j_0)

Esto aplicará la regla r_{j_0} , actualizando la cantidad de moléculas en el medio m , dando como salida del algoritmo la reacción química a ejecutar y su tiempo de espera para hallar el siguiente estado del sistema.

2.3. Redes de Petri

Junto a la simulación estocástica se ha decidido hacer uso de Redes de Petri [Peleg et al., 2005]. Esta herramienta gráfica y de modelado de sistemas nos permitirá describir y estudiar los sistemas biológicos implementados, en especial los comentados anteriormente, que implementan características estocásticas.

Dichas redes fueron propuestas en los años 60 por Carl Adam Petri [Petri, 1966], con la finalidad de establecer un formalismo gráfico y matemático destinado al análisis de sistemas distribuidos concurrentes, y que posteriormente sería ampliado a una gran multitud de campos, incluido el de la bioquímica [Chaouiya, 2007], el cual se ha puesto en práctica en este trabajo.

La implementación de las Redes de Petri al trabajo permitirá, además de estudiar el comportamiento de modelo a nivel de partículas en condiciones estocásticas, que los expertos en el campo de la bioquímica comprueben de manera gráfica que el modelo ha sido correctamente implementado y que su funcionamiento es el correcto, sin la necesidad de entrar a entender el lenguaje con el que han sido implementados para su simulación en Tellurium. Estos estudios son posibles ya que tenemos un modelo dinámico de tipo cuantitativo, las cuales se basan principalmente en ecuaciones diferenciales ordinarias.

En la siguiente imagen se puede ver un ejemplo sencillo en el que se ilustra el funcionamiento de una red de Petri básica (Fig. 2.8). En ella se puede observar que un elemento de la sustancia A se convierte en dos del elemento B, ya que viene definido por el peso del arco y en la transición esta establecida esa reacción.

El funcionamiento básico de una red de Petri en los sistemas bioquímicos es el siguiente. Se hace uso de lugares, dibujados como círculos, para representar un reactivo o enzima y su marcado para indicar la cantidad de la misma (número de moléculas), las transiciones para representar las reacciones, que modifican la cantidad de reactivos y los arcos para indicar el

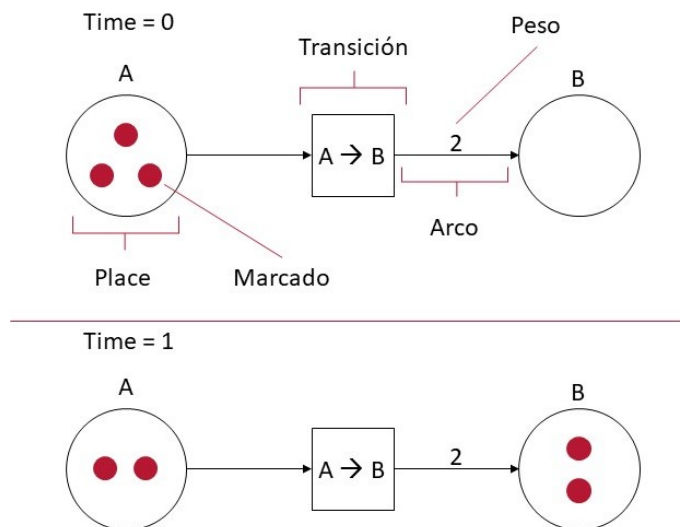


Figura 2.8: Ejemplo del funcionamiento de una Petri Net

sentido de la red, que junto con su peso, que sirve como coeficiente estequiométrico para indicar en que proporción participan los reactivos en la reacción.

Existen multitud de tipos de redes de Petri y, aunque una red estándar ya es más que suficiente para la representación de los sistemas bioquímicos, cuando se desea trabajar a un nivel muy detallado como puede ser a nivel molecular, tenemos la opción de hacer uso de redes de Petri estocásticas, las cuales, si son configuradas para que se haga uso del algoritmo de Gillespie, pueden tener en cuenta el ruido intrínseco debido a los bajos niveles de las concentraciones.

En la siguiente figura (2.9) se puede observar la Red de Petri que se ha diseñado para uno de los modelos del ciclo de Calvin, en concreto, para para el modelo de Zhu [Zhu et al., 2009] diseñado en 2009, siguiendo las premisas presentadas anteriormente del funcionamiento de las redes de Petri en los modelos bioquímicos. Más adelante, en el capítulo de implementación, se desarrollará como ha sido creada dicha red de Petri a partir del código de Antimony inicial.

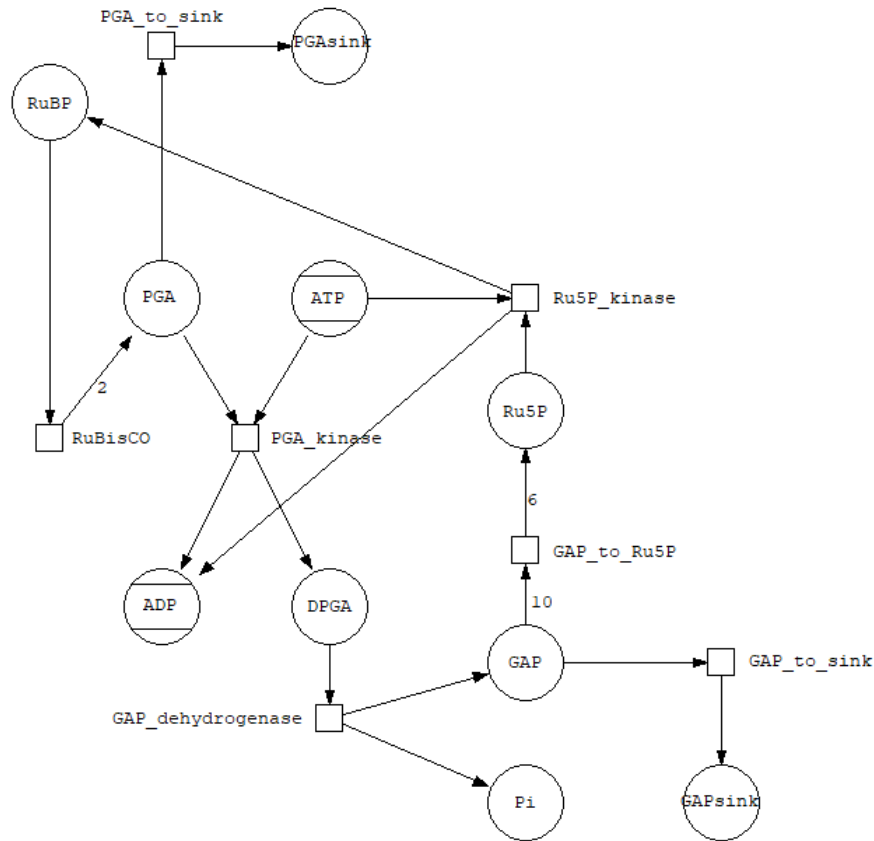


Figura 2.9: Red de Petri para el modelo del Ciclo de Calvin (Zhu-2009)

2.4. Lógica difusa y Snoopy

El desarrollo de sistemas bioquímicos puede ser una tarea bastante compleja, sobre todo a la hora de establecer los valores iniciales a las diferentes especies y variables que componen el modelo, ya que, no existe un valor claro para cada una de estas y va a depender mucho de la representación que se ha hecho del modelo. Actualmente existen diversas páginas web como BRENDA [BED,] o BioNumbers [Sea, 2022], bases de datos que indica al usuario el rango de valores permitido para cada una de las variables. Normalmente se ofrecen rangos amplios, por lo que el valor seleccionado puede no ser el correcto y hacer pruebas para todas las variables puede llevar una gran cantidad de tiempo.

Esto se agrava aún más cuando los modelos comienzan a ser complejos, donde tenemos una gran cantidad de parámetros que establecer y hay una gran relación entre las ecuaciones diferenciales que componen a este. Esto puede provocar que un parámetro que no ha sido establecido correctamente acabe haciendo que la simulación otorgue al usuario resultados carentes de lógica.

Para tratar de solucionar este problema se ha decidido recurrir a la lógica difusa, también conocida como “Fuzzy Logic” [Zadeh, 1988]. Esta se basa en la utilización de conjuntos difusos o borrosos, una generalización de los conjuntos clásicos capaces de manejar la incertidumbre de la que hablábamos anteriormente. Dichos conjuntos se definen dentro de un conjunto universal X por su función, que es característica por tomar valores en un intervalo cerrado $[0,1]$, que indicará el grado de pertenencia del elemento en el conjunto.

También se usaran números difusos [Kacprzyk, 2005] para trabajar con este tipo de lógica. Dichos números son conjuntos borrosos, convexos y normalizados dentro del conjunto X dado por el conjunto de números reales. Estos números, en el caso que se plantea para este Trabajo de Fin de Máster, están conformados por tres valores, el primero de ellos un valor pesimista o con menor grado de pertenencia, otro el que mayor y por último un valor optimista. Estos valores nos devolverán una figura como la que se puede apreciar en la siguiente imagen (2.10), que sirve para representar dicho número difuso.

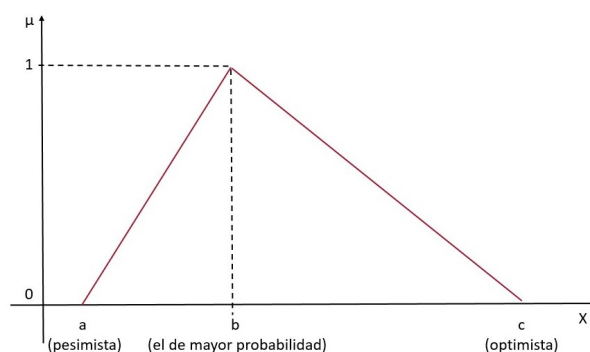


Figura 2.10: Representación gráfica de un número Fuzzy

En el campo de los número difusos también es interesante presentar el concepto de α cut (alfa corte), el cual nos indica un subconjunto acotado del conjunto a examinar, para un valor de $\alpha \in [0, 1]$. De esta manera podremos obtener los valores que tengan un grado de pertenencia superiores o iguales a α .

En este punto, y en relación con el apartado anterior (Redes de Petri), es donde entra en juego Snoopy [Sno, 2022]. Snoopy es una herramienta, aun en desarrollo por el Departamento de Informática de la Universidad de Tecnología de Cottbus, para diseñar y animar diferentes gráficos, entre los que se encuentran las Redes de Petri, las cuales permiten la incorporación de parámetros Fuzzy.

Debido a la complejidad de los modelos bioquímicos desarrollados, actualmente con es-

te tipo de herramientas no se pudo realizar una simulación completa como se realizaba en Tellurium incluyendo la radiación solar, si no que se ha realizado una conversión del modelo original, como se explicará más adelante, a uno diseñado a nivel de partículas y en un volumen mucho más restringido. De esta manera podremos hacer una validación del sistema implementado con la simulación disponible en Snoopy y comprobar el funcionamiento de los parámetros difusos.

Snoopy también hace uso de los alfa cortes mencionados anteriormente, y es que, para realizar las simulaciones es necesario indicar la cantidad de alfa cortes que tendrá nuestro modelo. También será necesario indicar los puntos de discretización, que son puntos tomados dentro del intervalo del alfa corte. De esta manera se realizarán tantas simulaciones como $\text{alfa cortes} \cdot \text{puntos de discretización}$ se hayan establecido y ofrecerán al usuario, como resultado final, la característica banda de valores y el triángulo indicando el grado de pertenencia de cada valor de las simulaciones con parámetros Fuzzy. El uso de estas características puede verse más adelante en la figura 5.15, en el capítulo “*Pruebas de los modelos Fuzzy y estocásticos*” (5.2).

3. Metodología y entorno de trabajo

Para el desarrollo y realización de este Trabajo de Fin de Máster se ha seguido la metodología SCRUM [SCR,], ya que ha sido la más utilizada durante el grado gracias a su flexibilidad en entornos de trabajo de naturaleza cambiante, una sencilla implementación y sobre todo la ventaja de estar enfocada en el trabajo en grupo, por lo que podremos tener un mejor flujo de trabajo entre el desarrollador y el propietario final del producto. Esto es de suma importancia en proyectos complejos debido a que la comunicación es un factor fundamental para la optimización de recursos, sobre todo en etapas tempranas del proyecto.

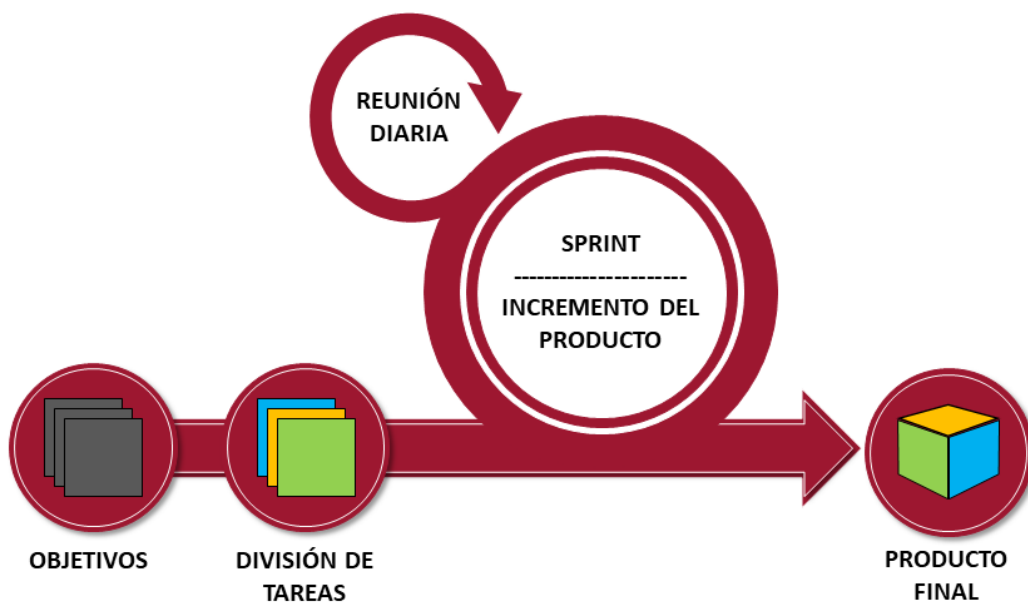


Figura 3.1: Diagrama de la metodología SCRUM

En la figura anterior (3.1) se puede observar un diagrama del funcionamiento de la metodología a lo largo del proyecto. Para comenzar partimos de los objetivos del Trabajo de Fin de Máster, a los cuales se les asociaron una serie de tareas con las que dicho objetivo sería alcanzado y por último estas tareas fueron repartidas en diferentes sprints, los cuales se muestran a continuación.

- **Sprint 0:** Estudio y recopilación del trabajo previo (Modelos y rutas metabólicas implementadas).
- **Sprint 1:** Desarrollo, implementación y testing del ciclo de Calvin-Benson.
- **Sprint 2:** Iniciación en supercomputación (GALGO), optimización y mejora de los modelos.
- **Sprint 3:** Estudio e implementación de simulación estocástica.
- **Sprint 4:** Estudio de lógica difusa e implementación en los modelos
- **Sprint 5:** Redacción del documento Trabajo de Fin de Máster.

Durante la realización todos los sprints relacionados con las etapas de implementación, es decir, sprint del 1 al 4, se han realizado reuniones de seguimiento diario para comprobar el trabajo realizado. De esta manera se conseguía fijar de manera más sencilla el objetivo que se quería conseguir, comprobar si el trabajo realizado era el correcto para lograrlo y poder solventar problemas que apareciesen a lo largo del desarrollo. Tras la finalización de un sprint, finalmente conseguimos un incremento en el proyecto, el cual será el producto o resultado final una vez queden todos los sprints finalizados.

Para mejorar la organización de las tareas durante el proyecto y aumentar el valor de la metodología SCRUM se ha hecho uso de la herramienta web Trello, la cual sirve para la gestión y administración de proyectos de manera colaborativa y que nos permite plasmar de forma gráfica el estado de cada una de las tareas y la información oportuna, como por ejemplo, horas dedicada, problemas encontrados y posibles soluciones, etc.

Por último, se debe destacar que la realización de este trabajo ha sido realizado en conjunto con las prácticas en empresa, adscritas al Instituto de Investigación Informática de Albacete y las cuales han sido desarrolladas en el laboratorio de química de la Agrupación Politécnica de Albacete. Esto ha permitido disponer de un equipo de trabajo multidisciplinar, involucrado durante todo el ciclo de vida del proyecto y a los cuales va dirigido el producto final. De esta manera pueden comprobar el estado del mismo y aportar el conocimiento de la parte de bioquímica que falta para el correcto avance del trabajo.

4. Desarrollo e implementación

4.1. Modelos de radiación solar (Python)

En este apartado se explicará como han sido desarrollados los dos nuevos modelos de radiación solar, ambos basados en el principio de la luz ampliada o luz AM que fue implementada previamente. Ambos modelos tienen como objetivo mejorar el tiempo necesario para realizar las simulaciones, el realismo de los datos obtenidos y la automatización del modelo al seleccionar diferentes ubicaciones.

4.1.1. Modelo Épsilon-Delta

Cuando se recuperó el modelo de Luz Ampliada (Cap 2.1.2) para su posterior mejora y se comprobó su funcionamiento en diferentes latitudes, se observó que, en latitudes extremas, se produce un desfase en las horas de la salida y la puesta del sol, lo que hacía que la radiación, que era el principal dato de entrada del modelo bioquímico, fuese erróneo. En algunos casos este desfase llegaba incluso a producir "huecos" en el modelo, haciendo que en los meses de verano únicamente fuese de día, y por lo tanto, obteniendo datos continuos de radiación. En la siguiente figura (4.1) se muestra el error producido para la ciudad de Oslo (Latitud: 59.91°) y Reykjavik (Latitud: 64.147°).

La idea principal de este modelo es la búsqueda e introducción de dos valores (Épsilon y Delta) durante el proceso de cálculo de la radiación, con el fin de que dichos valores ajusten los puntos de radiación pertenecientes al momento de salida y puesta del sol al momento en el que este se debería de producir. En concreto, el valor de épsilon permite ajustar la parte central del año, entre los equinoccios de primavera y verano, y el delta la parte correspondiente al invierno. De esta manera se mantienen fijos los periodos del año en los que el día y la noche tienen la misma duración. Este proceso, además de ajustar el comienzo y fin de la radiación diaria para obtener más exactitud, permitirá eliminar el "huevo" producido en algunas localizaciones. Más adelante comprobaremos su funcionamiento.

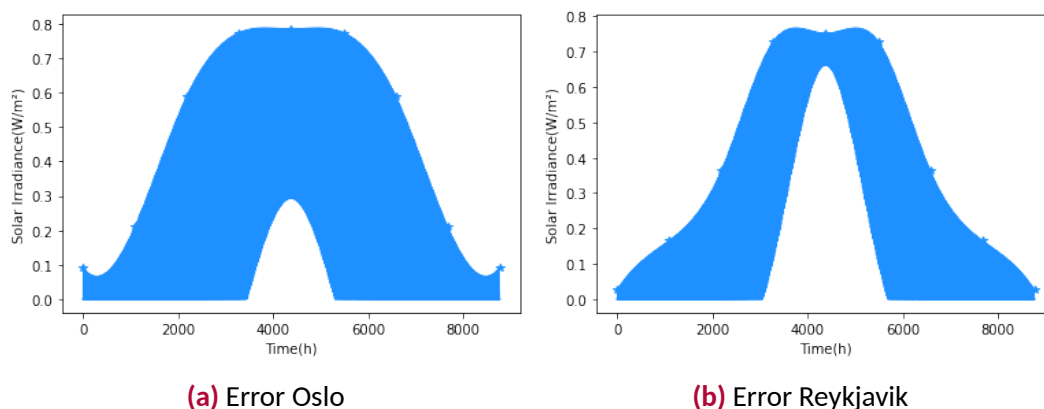


Figura 4.1: Agravamiento del error producido respecto a la latitud

Este modelo podemos encontrarlo en la carpeta RS_EPS_DEL, ejecutando la libreta de Python “RS_Generator.ipynb”. Desde aquí el usuario podrá modificar algunos parámetros para la generación de la radiación solar, los cuales se muestran a continuación.

- **Localización:** Introducir el nombre de una de las ciudades disponibles en el fichero de referencias.
- **Referencia:** Indicar True o False en caso de utilizar o no uno de los ficheros mencionados anteriormente. En caso de no utilizarlos se deberá indicar la latitud, longitud y unos puntos de referencia para el cálculo de la radiación solar en un rango de 0 a 8760 (se recomienda un mínimo de 5 puntos repartidos a través de todo el rango de valores).
- **Aleatorios mínimos y máximos diarios y horarios:** En el caso del diario, se utiliza un factor aleatorio para disminuir o aumentar la radiación y producir diversos picos de esta. En el caso de la horaria, se utiliza para simular nubes u otros efectos atmosféricos que puedan reducir la radiación que llega hasta la superficie. (Se recomienda dejar los valores por defecto).
- **Ajuste:** Indicar True o False para indicar que se quiere realizar la búsqueda y ajuste con los valores Épsilon y Delta.
- **EpsilonValues y DeltaValues:** rango de valores en los que se hace la búsqueda del Épsilon y Delta óptimos.
- **Test:** Indicar True o False para comprobar el funcionamiento del modelo y ver el error cometido.

Una vez definidos los parámetros leeremos los datos necesarios desde el fichero de referencia, entre los cuales se obtienen la latitud y longitud, los puntos de referencia temporales, denominados x1 (por defecto se trabaja con 9) y la radiación de dichos puntos. Este proceso se realiza mediante la función “leeReferencia” del fichero “funcionesAuxiliares.py”. En dicha función también se tiene en cuenta la diferencia producida con el horario UTC en función

de la longitud y se realiza la operación numpy roll para colocar la radiación en su hora correspondiente antes de empezar a trabajar con ella.

A continuación, calculamos el parámetro y_1 , presentado anteriormente en el apartado de Modelos de Radiación Solar (2.2), este parámetro nos servirá para conocer la diferencia en el fotoperiodo respecto a los momentos en los que el día y la noche tiene la misma duración ($y_1 = 0$). Mediante una interpolación con un spline cúbico con los valores de y_1 y x_1 , utilizando el modulo scipy de Python, se obtienen los valores de h_{sol} , la diferencia de fotoperiodo para cada día del año (Figura 4.2).

También es necesario obtener los puntos y_3 , que se toman de la radiación de referencia

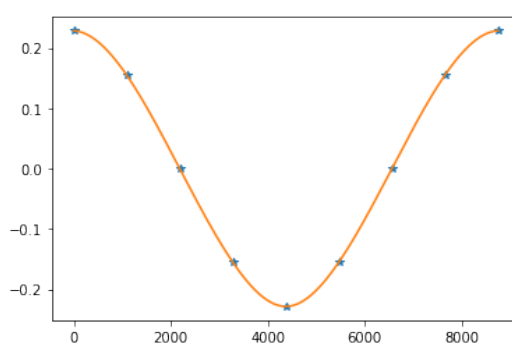


Figura 4.2: Curva h_{sol} resultante de la interpolación

normalizada e y_2 , calculados mediante $y_1 + y_3$, tal como se explicó en el apartado teórico del modelo de radiación. Puesto que estamos trabajando con un modelo de radiación simétrico, a la hora de calcular los puntos de y_3 no es necesario hacer una lectura de todos, si no que se pueden añadir al vector de puntos con la instrucción append como se puede observar en la función “calcula_y23” del fichero “funcionesAuxiliares.py”. Conociendo así los valores de y_2 e y_3 , podemos calcular Ac_2 mediante un spline cúbico de y_2 y x_1 , y finalmente, hacer uso de la función “calculaRadiacion” del fichero “funcionesAuxiliares.py”, lo que nos devolverá los valores de radiación de un año completo normalizados (Figura 4.3).

Test y ajuste

Tras calcular la radiación solar, esta es sometida a un test que ha sido diseñado para comprobar su precisión y solucionar el error del “hueco” comentado anteriormente. Esta prueba lo que hará será comprobar para cada día del año, el fotoperiodo resultante del modelo respecto al que debería de existir realmente. Para las pruebas partimos de la base de que no debemos tener más del 30% de error en una hora, es decir 18 minutos de margen de error en cada día (9 min al comienzo del fotoperiodo y 9 min al final de este).

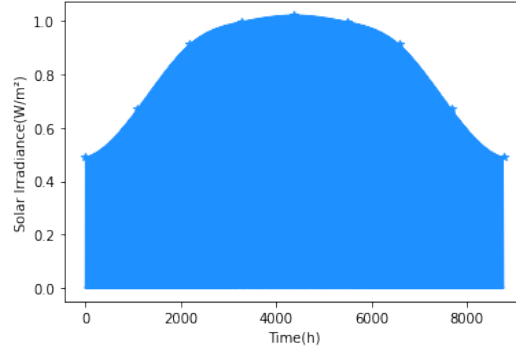


Figura 4.3: Radiación Albacete normalizada

El test es ejecutado por la función “ejecutaTest” del fichero “funcionesTest.py”. Esta función se encargará de calcular el fotoperiodo de cada día en función de la latitud de la localización deseada haciendo uso de las siguientes ecuaciones:

$$\begin{aligned}
 sd &= 23.45 \cdot dr \cdot \sin((2\pi \cdot (dia + 274)) / 365); \\
 ws &= \arccos(-\tan(sd) \cdot \tan(latitud \cdot dr)); \\
 wts &= ws \cdot (180 / (\pi \cdot 15)); \\
 Tr &= 12 - wts; \\
 Ts &= 12 + wts; \\
 Fotoperiodo &= Ts - Tr;
 \end{aligned} \tag{4.1}$$

Donde $dr = \pi/180$ y dia es el número de día del año para el que se está calculando el fotoperiodo.

A continuación para cada 240 valores de radiación calculados, los cuales pertenecen a un día del año, se comprueba la cantidad de valores que son diferentes de 0, es decir, la cantidad de puntos en los que obtenemos radiación y calculamos la diferencia entre el modelo calculado y el real. Si la diferencia es menor que 0.3, el día comprobado se da como válido, en caso contrario como fallo. Finalmente se muestra el resultado obtenido para todo el año. En el caso de Albacete (Figura 4.3), la primera vez que ejecutamos el test obtenemos 125 aciertos y 240 fallos. Por lo tanto, el próximo paso es realizar un ajuste que minimice dichos fallos.

La función “ajusteDeltaEpsilon”, a la cual se le pasa como parámetro una lista con los posibles valores de épsilon, es la encargada de buscar los mejores valores de delta y épsilon capaces de minimizar el error producido en el modelo. Para ello comprobaremos la diferencia del fotoperiodo en dos días del año, pertenecientes al invierno y el verano, ya que si observamos solo el ajuste en una estación, se puede producir un sub-ajuste o sobre-ajuste en la estación contraria.

La elección de la mejor combinación de delta y épsilon es seleccionada por la menor suma de las diferencias del fotoperiodo en invierno y en verano, y estas combinaciones son testeadas de forma paralela, como se verá más adelante, recalculando la radiación con la función “radAjustada” del fichero “funcionesTest.py”. En dicha función se recalcula la radiación solar añadiendo el delta y el épsilon a los valores de y_1 y y_2 tal y como se puede apreciar en la siguiente imagen (Figura 4.4).

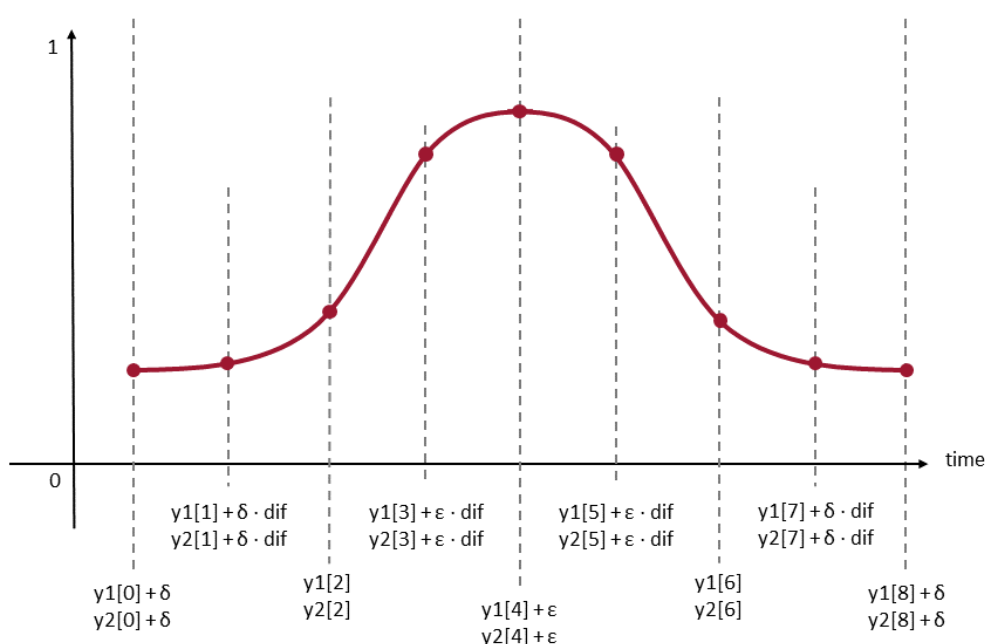


Figura 4.4: Cálculo de la radiación con delta y épsilon

Como se puede observar, al ser un modelo simétrico, facilita bastante el cálculo de los nuevos valores y_1 y y_2 para el ajuste. Para los valores y_1 y y_2 en el punto 0 y 8, correspondientes al invierno, deberemos sumar delta a su valor original. En el punto 4, correspondiente al verano, haremos uso de épsilon y en los puntos 2 y 6, en los que el día y la noche tiene la misma duración no haremos uso de ninguna variable para su ajuste, ya que son considerados puntos de transición.

Aun nos quedan un total de 4 puntos intermedios a los que aplicar un valor de delta o épsilon, el cual no debe de ser el mismo que en los puntos comentados anteriormente, si no que se deberán multiplicar por un factor, denominado dif, para hacer el ajuste oportuno. Para el cálculo de dif se han utilizado las siguientes ecuaciones, que corresponden a una normalización entre 0 y 1 para obtener en porcentaje la distancia del punto a calcular entre dos de referencia de y_1 :

$$dif(cont) = \begin{cases} (y1[cont] - y1[cont + 1]) / (y1[cont - 1] - y1[cont + 1]) & \text{si } cont \text{ es } 1 \\ (y1[cont] - y1[cont - 1]) / (y1[cont + 1] - y1[cont - 1]) & \text{si } cont \text{ es } 3 \\ (y1[cont] - y1[cont + 1]) / (y1[cont - 1] - y1[cont + 1]) & \text{si } cont \text{ es } 5 \\ (y1[cont] - y1[cont - 1]) / (y1[cont + 1] - y1[cont - 1]) & \text{si } cont \text{ es } 7 \end{cases}$$

Siendo *cont* un rango de números enteros de 0 a 8.

De esta manera podremos volver a calcular la radiación solar y la función “ajusteDeltaEpsilon” nos devolverá la mejor suma de las diferencias del fotoperiodo, la mejor delta y el mejor épsilon encontrados para dicha suma.

Dado que tenemos bucles anidados para realizar las diferentes combinaciones y es necesario recalculer toda la radiación en cada una de estas, se ha optado por paralelizar dicha parte del código para así ahorrar tiempo y aprovechar todos los recursos del ordenador en el que se lance la aplicación. Para ello se ha hecho uso de la función `Parallel` de `Joblib`, un conjunto de herramientas disponibles para Python. Dicha función puede ser llamada de la siguiente manera:

```
Parallel(n\_jobs=-1, verbose=1, backend="loky")(map(delayed(
    ajusteDeltaEpsilon), epsilonValues))
```

Donde `n_jobs = -1` indica que se utilizan todas las CPUs disponibles, `verbose = 1` sirve para imprimir el progreso del proceso en pantalla y `backend "loky"` indica el back-end de paralelización, esto servirá para iniciar procesos en CPUs separadas. También se debe indicar la función que queremos paralelizar, la cual es “ajusteDeltaEpsilon”, y la variable que queremos modificar en cada iteración o proceso, “epsilonValues”.

El resultado de cada hilo de ejecución, es decir, la mejor suma, épsilon y delta se va modificando según disminuye la suma de las diferencias en el fotoperiodo, de esta manera podremos usar la mejor combinación posteriormente para crear nuestro modelo final de radiación, llamando de nuevo a la función “radAjustada” y realizando el test para comprobar la precisión con la función “ejecutaTest”.

Para finalizar el modelo, se aleatoriza el modelo de radiación creado. Haciendo uso de `numpy random`, añadimos un factor aleatorio diario dentro de los rangos establecidos previamente por el usuario para aumentar o disminuir la radiación, y otro factor que se aplica cada hora para simular nubes u otros efectos atmosféricos que puedan reducir la radiación solar en momentos puntuales. Estos 87600 nuevos valores de radiación son almacenados en un vector, que será recorrido para posteriormente para escribir los datos en un fichero `txt` (`radiacion_eventos.txt`), el cual será leído mediante eventos en el simulador de Tellurium. Dichos eventos tiene la siguiente estructura:

```
at (time >= h\_inicio && time < h\_final): F1AUX = valor\_rad}
```

Donde h_{inicio} y h_{final} son instantes de tiempo que comprenden el instante temporal en el que se establece el valor de la radiación solar en la variable $F1AUX$, y $valor_{rad}$ el propio valor de la radiación solar en W/m^2 .

Este fichero txt de eventos será unido a uno ya creado previamente en Antimony, en el que se encuentra definido el modelo de la ruta metabólica a simular. Este nuevo fichero ya completo para su lectura se puede encontrar con el nombre de "modeloFinal_v2.txt" para su consulta o modificación por parte de los expertos en el campo de la bioquímica. Haciendo uso de la función "loada" de Tellurium podemos cargar el modelo y almacenarlo en la variable `model`. Posteriormente lo simularemos con la instrucción "`model.simulate`", en la que estableceremos la cantidad de pasos de simulación que deseamos obtener.

Haciendo uso de la función `repr` de Python podemos convertir un objeto a su representación en formato string. De esta manera lo podemos escribir en un fichero txt (*resultados.txt*) y mediante la función `replace` sustituimos caracteres como corchetes y espacios en blanco para que este fichero pueda ser cargado en programas como SigmaPlot para su representación gráfica.

4.1.2. Modelo Envolvente

El modelo envolvente ha sido diseñado con el objetivo de mejorar la envolvente de radiación y solucionar los problemas relacionados con el fotoperiodo diario, que se verán más adelante en los experimentos, y de esta manera obtener un modelo, que aunque algo más simple que el anterior, ofrezca soporte a una mayor cantidad de localizaciones con mayor precisión en cuanto a los periodos de radiación. Esto será implementado teniendo en cuenta la radiación solar máxima diaria y el fotoperiodo real

El proceso inicial para el cálculo de la radiación es bastante similar al método anterior, se comienza leyendo los datos de referencia para obtener la relacionada a la radiación solar de cada instante del año y se establecen n puntos equidistantes para generar nuestra envolvente haciendo uso de la función "`numpy.arange(0, 8760.1, 8760/nPuntos)`", en este caso se han utilizado 13 puntos.

Tras esto, se recorre la lista con los valores tomados como referencia y se almacenan en un array denominado `mediaRAD` los valores más altos de radiación. Se toman los primeros y últimos 10 días para en el caso de los puntos 0 y 13 (primer y último día del año) respectivamente, y los 10 anteriores y siguiente para el resto de puntos. De esta manera obtenemos el pico de radiación de cada día seleccionado y entre estos valores se selecciona el más alto.

Con este valor seleccionado, se utilizará como referencia para realizar una media con los otros 10 o 20 valores seleccionados en función del punto. Se recorren los los valores con un bucle `for` y se comprueba que el valor consultado es igual o mayor que el que tomamos como referencia * 0.75. De esta manera nos aseguramos obtener un valor representativo de la época del año relacionada con el punto calculado y evitamos la posibilidad de obtener

un valor anormal debido a un error en los datos de referencia o una anomalía atmosférica.

Con estas medias calculadas, almacenamos los valores en el array `y3` y realizamos un spline cúbico para interpolar entre los puntos, como el que se utilizó en el modelo ϵ -delta, haciendo uso de la función `ajustey3`, a la que le pasaremos como parámetros los valores de `y3` e `x1`, siendo `x1` los puntos equidistantes tomados previamente. Para la realización de la interpolación se ha hecho uso de la función `splrep()` y `BSpline()` del módulo `Scipy.interpolate` de Python.

Los valores resultantes de dicha interpolación serán almacenados en la variable envolvente y representará el pico de radiación diario para un año completo, por lo que obtendremos 365 valores de radiación (Fig 4.5).

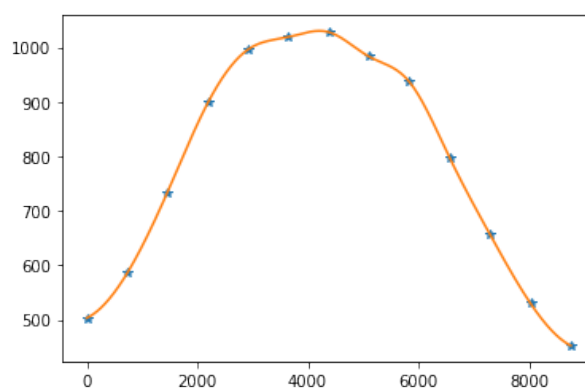


Figura 4.5: Envolvente radiación Albacete

Con estos datos se genera nuestro modelo final para la simulación, que consiste en dos ficheros que se unifican, una parte fija y otra variable en la que se escriben los eventos con aleatorios necesarios para el modelo. Puesto que el cálculo de la radiación es mucho más simple que el modelo anterior, podemos implementar funciones dentro del fichero del modelo con Antimony, en el caso anterior debido a la complejidad no era posible. Lo que se realiza dentro del modelo es la curva de radiación diaria mediante una función seno, forzando el inicio y fin de la radiación calculando la salida y puesta del sol para una localización. A continuación se muestra la función en Antimony que posibilita este cálculo.

```
function fCalculayRS(dia,tm,sr,ss,a1,b1,c1)
    piecewise(a1*sin(b1*(tm-24*(dia-1))+c1),tm <= ss+24*(dia-1) &&
        tm >= sr+24*(dia-1),0)
end
```

Donde `tm` es el instante de tiempo de la simulación, `sr` la hora de salida del sol, `ss` la de

puesta, $a1$ el valor de radiación máximo para ese día (se pasa como evento al modelo), $b1 = \pi/(ss-sr)$ y $c1 = (-\pi/(ss-sr)) \cdot sr$. De esta manera, al calcular la curva de radiación de una manera simple se evita el uso de pasar la radiación de manera horaria como evento, lo cual se tenía que hacer en el modelo anterior e incrementa de manera drástica los tiempos de carga y simulación del modelo.

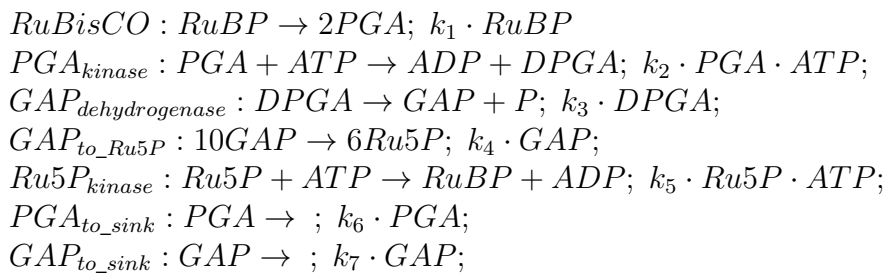
Finalmente se realiza la carga y simulación y se almacenan los resultados y se realiza una limpieza de datos para su posterior consulta en programas como Excel o Sigmaplot.

4.2. Modelo Fuzzy (Snoopy)

A continuación se va a mostrar el desarrollo del modelo para el software Snoopy, en el que se ha modelado a partir de una red de Petri y se ha llevado a cabo la simulación de manera estocástica a nivel de moléculas, a la vez que se introducen parámetros con incertidumbre haciendo uso de números Fuzzy.

El primer paso para la implementación de este modelo y caso de estudio fue la elección del modelo bioquímico del Ciclo de Calvin que se utilizaría. Tras realizar una búsqueda y analizar varios modelos, se decidió usar el modelo Arnold2011-Zhu2009 [Arn, 2011] [Arnold and Nikoloski, 2011], un modelo que representa una parte del ciclo, ya que para su simulación a partir de una red de Petri se necesitaba crear un modelo que no fuese extremadamente complejo. Además, otro de los inconvenientes de este tipo de simulaciones y que se agrava demasiado al trabajar con modelos mínimamente complejos, es que se necesita trabajar a nivel de moléculas, es decir, es necesario hacer una conversión del modelo a moléculas para poder realizar la simulación estocástica con el algoritmo de Gillespie. Esto se explicará más adelante.

La red de Petri ha sido diseñada en Snoopy, programa que permite la creación y simulación de la misma incorporando parámetros Fuzzy, haciendo uso de las siguientes reacciones:



Donde, en primer lugar encontramos el nombre de la reacción, los elementos descritos a la izquierda de la flecha son las especies originales, los elementos a la derecha a los que son transformados, k_n el coeficiente de velocidad a la que se produce dicha reacción, siendo estas del tipo masa-acción y los números enteros la cantidad de moléculas necesarias para

la reacción o como resultado, siendo por defecto 1.

Para el modelado de la red de Petri se ha utilizado los elementos Place, Transition y Arc que podemos encontrar en el menú izquierdo de la herramienta Snoopy (Fig 4.6a) tras abrir un nuevo archivo Fuzzy Stochastic Petri Net, los cuales nos permitirán representar las reacciones descritas anteriormente de manera estocástica y Fuzzy.

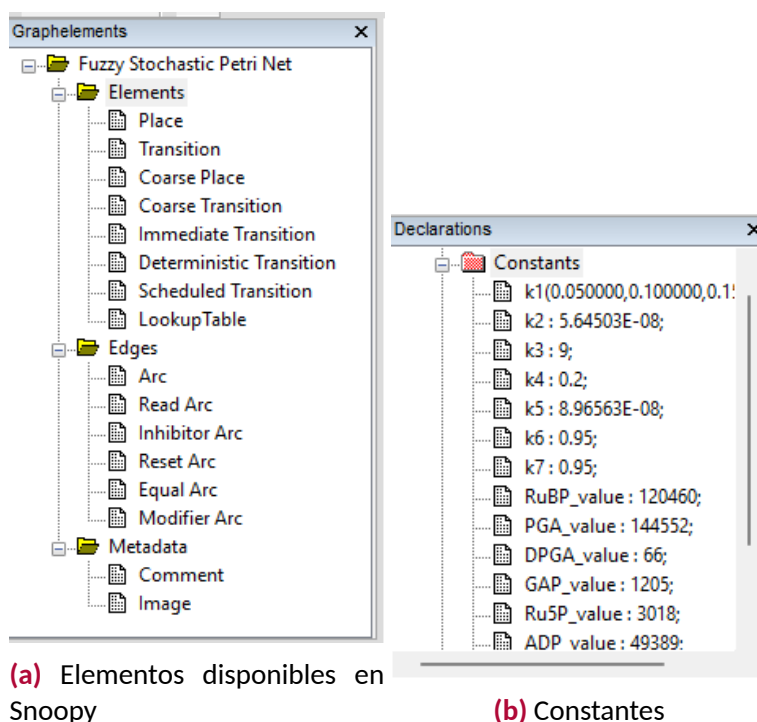


Figura 4.6: Menús edición Snoopy 2.6

Los elementos de tipo Place serán utilizados para las especies, ya que representarán un contenedor en el que albergar la cantidad deseada de estas, en este caso el número de moléculas en el modelo. Los elementos de tipo Transition para representar la velocidad a la que se transforman dichas especies. Por último los de tipo Arc para dar el sentido a las transformaciones, es decir, indicar el recorrido que van a realizar nuestras moléculas a través de la red y su peso indicará las que son necesarias para realizar la transformación.

En la siguiente imagen (Fig. 4.7) se muestra un fragmento de la red modelada en la que se aprecian los diferentes elementos que puede contener la misma.

En la figura podemos apreciar los elementos básicos Place, Transition y Arc, este último con un grado de multiplicidad 2, es decir, 1 partícula de RuBP se transformará en 2 de PGA.

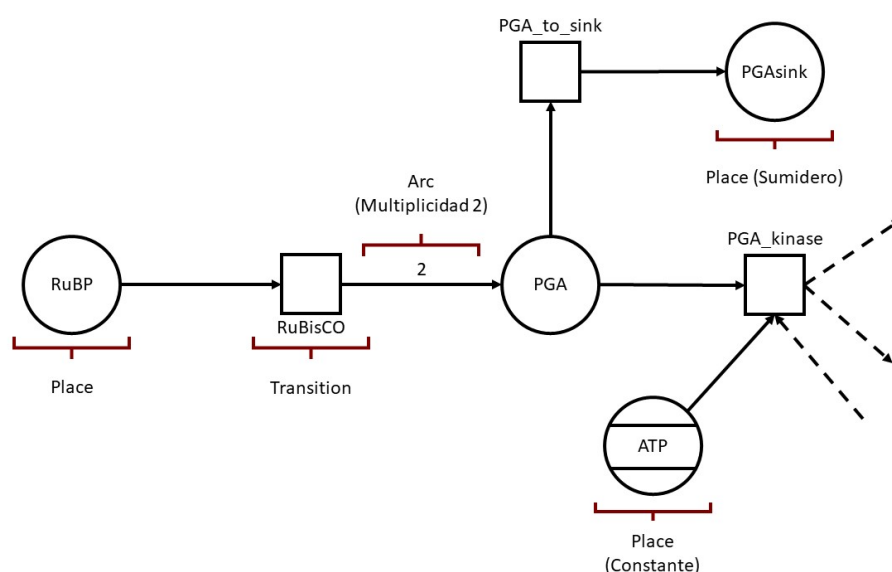


Figura 4.7: Fragmento Red de Petri

También se añaden places adicionales como “PGAsink” que actúan como sumidero para las reacciones “PGAta_sink” y “GAPto_sink”. También encontramos elementos de tipo Place que mantienen constante la cantidad de moléculas, como es el caso del place de la especie ATP, que en este modelo se trata de una constante ya que actúa únicamente como un emisor y para que el resto del modelo siga funcionando se necesita una emisión constante de este elemento.

Una vez colocados los elementos, se editan las propiedades de estos para ajustarlos a nuestro modelo. Esto se realiza haciendo doble click en cada uno de los elementos, en el caso de los de tipo place (Fig. 4.8) nos interesa modificar la opción de “Fixed”, para indicar si queremos que varíe o no su valor con respecto al tiempo de simulación, y la opción de “Marking”, en esta podemos indicar el valor inicial o número de moléculas del elemento o asignarle el nombre de una constante definida previamente.

Para los elementos de tipo Transition (Fig. 4.9) nos iremos hasta la pestaña “Function” y ahí escribiremos la función que defina la velocidad de nuestra reacción. En este caso las funciones serán del tipo $K_n \cdot Elem_{entrante}$, siendo K_n la constante de velocidad de la reacción y $Elem_{entrante}$ la cantidad de la especie entrante al elemento de transición.

Los elementos de tipo Arc (Fig. 4.10) únicamente necesitan modificar su multiplicidad, esto nos servirá para indicar que una molécula del elemento A se puede transformar en n moléculas del elemento B o viceversa.

El resultado final de la red de Petri fue presentada anteriormente (Fig. 2.9) en el capí-

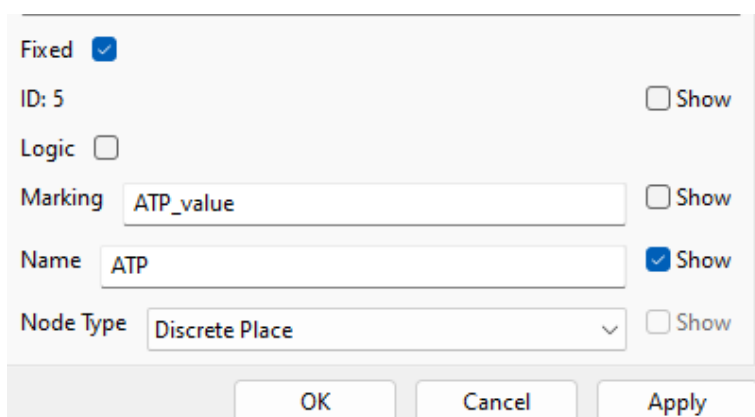


Figura 4.8: Opciones del elemento Place

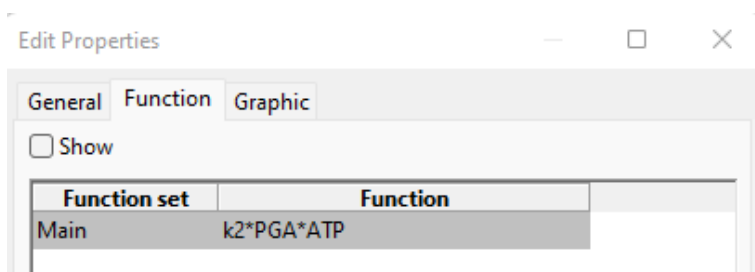


Figura 4.9: Opciones del elemento Transition

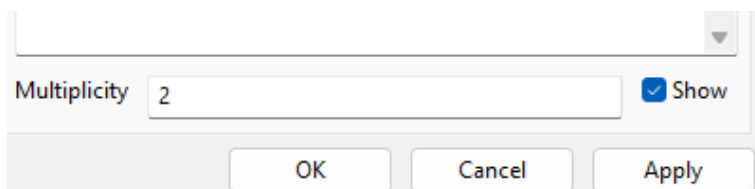
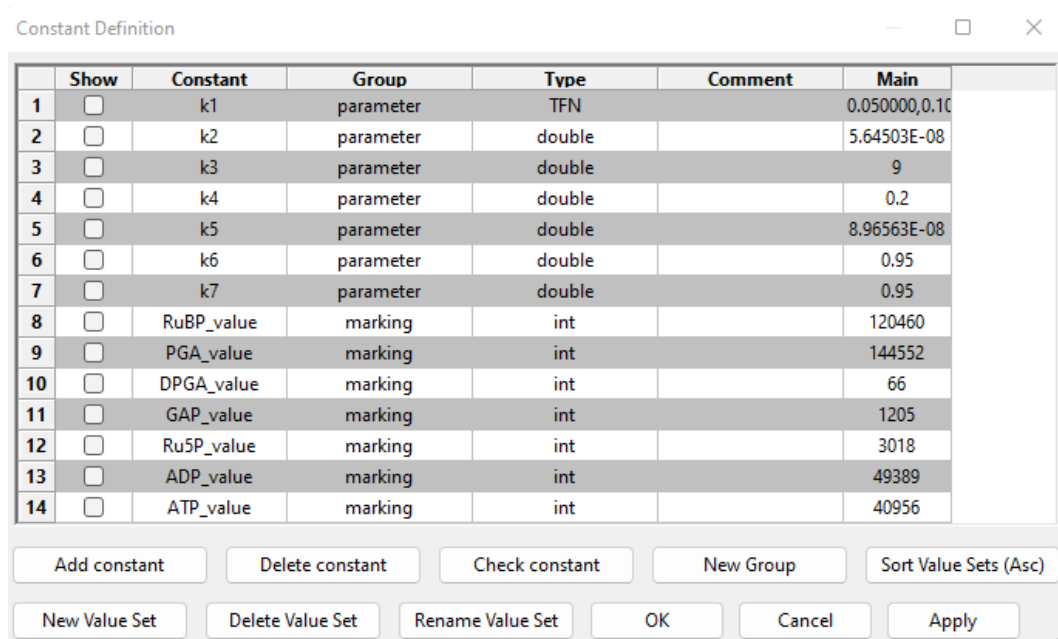


Figura 4.10: Opciones del elemento Arc

tulo 2 de este documento. En dicha figura se encuentran modeladas todas las reacciones propuestas en esta sección (Eq 4.2)

Por último, para definir las constantes nos desplazaremos hasta la sección “Constants”, situada en el menú izquierdo de la herramienta (Fig. 4.6b) y haciendo doble click se nos abrirá una ventana en la que escribir y añadir nuestras constantes. Como se pudo ver en la siguiente imagen (Fig. 4.11) tenemos tres tipos de valores, los de tipo entero, para representar el número de moléculas y que pertenecen al grupo marking, ya que corresponde al nombre que se asignó en el elemento Place (Fig. 4.8); los de tipo double, que son utilizados para asignar los parámetros K_n de las reacciones; y por último los TFN (Triangular Fuzzy

Number), que sirven para establecer un parámetro como un valor Fuzzy. Para este último, tal como se comentó en el capítulo de teoría (Cap. 2.4), se le deberán asignar los tres valores característicos de este tipo de números.



	Show	Constant	Group	Type	Comment	Main
1	<input type="checkbox"/>	k1	parameter	TFN		0.050000,0.10
2	<input type="checkbox"/>	k2	parameter	double		5.64503E-08
3	<input type="checkbox"/>	k3	parameter	double		9
4	<input type="checkbox"/>	k4	parameter	double		0.2
5	<input type="checkbox"/>	k5	parameter	double		8.96563E-08
6	<input type="checkbox"/>	k6	parameter	double		0.95
7	<input type="checkbox"/>	k7	parameter	double		0.95
8	<input type="checkbox"/>	RuBP_value	marking	int		120460
9	<input type="checkbox"/>	PGA_value	marking	int		144552
10	<input type="checkbox"/>	DPGA_value	marking	int		66
11	<input type="checkbox"/>	GAP_value	marking	int		1205
12	<input type="checkbox"/>	Ru5P_value	marking	int		3018
13	<input type="checkbox"/>	ADP_value	marking	int		49389
14	<input type="checkbox"/>	ATP_value	marking	int		40956

Buttons: Add constant, Delete constant, Check constant, New Group, Sort Value Sets (Asc), New Value Set, Delete Value Set, Rename Value Set, OK, Cancel, Apply

Figura 4.11: Menú de constantes Snoopy

En cuanto a las conversiones necesarias para ejecutar el modelo de forma estocástica se han tenido que hacer 2 conversiones diferentes. La primera de ellas, para calcular el número de moléculas de cada especie, se ha pasado de Milimol/Litro a N° de moléculas/Litro. Para ello se ha multiplicado por 0.001 para pasar a moles/Litro y finalmente por el número de Avogadro (6.023×10^{23}). Como esta conversión da un número muy elevado de moléculas para su simulación estocástica, se ha reducido el espacio a simular de 1L a 1×10^{-16} L, por lo que multiplicamos nuevamente por este valor y redondeamos para obtener el número exacto de moléculas, de esta manera la simulación será más estable.

Para las constantes debemos realizar la conversión en factor de que tipo de constante sea. Si solo se implica un reactivo, como puede ser el caso de k1, no es necesario realizar ninguna conversión. Si por el contrario, la constante está implicada en una reacción en la que tenemos más de un reactivo, se deberá de utilizar la siguiente formula:

$$\frac{\text{Valor inicial}}{N_A \cdot V} \quad (4.2)$$

Siendo el valor inicial el valor de la constante K_n , N_A el número de Avogadro y V el volumen

del recipiente en el que se encuentra nuestro modelo, en este caso $1\text{E-}16$. Junto a este documento se dispone de un fichero Excel preparado para realizar las conversiones oportunas para el modelo del ciclo de Calvin (Zhu2009).

4.3. Modelo Estocástico (Python)

Por último se ha desarrollado un cuaderno de Tellurium, en el que haciendo uso del modelo de Zhu2009 mencionado anteriormente se pretende realizar una simulación estocástica del modelo y simular el uso de parámetros Fuzzy para respaldar los resultados obtenidos por el software Snoopy visto previamente.

El primer paso será crear un fichero de texto que contenga nuestro modelo, partiendo de la base de las reacciones vistas en la sección anterior (Eq 4.2). Dicho documento debe presentar la siguiente estructura:

```
model nombre_modelo

    compartment nombre_compartimento; #definición del compartimento
    species especie1, especie2, ...; #definición de las especies
    especie1 in nombre_compartimento, especie2 in ... ; #asignación
        de la especie al compartimento

    #REACCIONES-----
    nombre_reaccion: especie1 -> especie2; kn * especie1; #
        definición de las reacciones
    ...

    #CONCENTRACIONES-----
    especie1 = num_moléculas; #definición cantidad inicial de
        moléculas
    ...

    #CONSTANTES-----
    kn = valor_constante #definición constantes de velocidad
        reacciones
    ...

end
```

A continuación se ha realizado el cuaderno de Tellurium, en el que mediante la función “loadAntimonyModel()” se procede a la carga del modelo txt creado previamente y se almacena en la variable denominada “r”. Tras esto se configura el integrador, en este caso, ya que se quiere ejecutar una simulación estocástica, se debe hacer uso del integrador “Gillespie”, desactivar el tamaño de paso variable y no permitir la integración con negativos. Por

lo que se procede a indicarlo en el cuaderno de la siguiente manera:

```
r.integrator = 'gillespie'
r.integrator.variable_step_size = False
r.integrator.nonnegative = True
```

Tras esta configuración establecemos la cantidad de tiempo que queremos simular (n), en este caso 10 horas y mediante la función “simulate()”, a la cual le pasamos como parámetros el tiempo de inicio (0), el tiempo final (n) y el número de pasos de simulación ($100 \cdot n$). Con esto ya tendríamos preparada nuestra simulación, solo queda visualizar los datos haciendo uso de la función “plot()”

Por último, se explicará como se ha desarrollado el modelo para que simule el uso de un número Fuzzy, esto ayudará a los expertos en ver que parámetros se deben seleccionar como Fuzzy y observar como afecta la incertidumbre de los parámetros a las especies.

Lo primero que hacemos es almacenar en una lista enumerada los parámetros que nos interesa ir modificando, en este caso las variables k_n , a los que podemos acceder con la función “getGlobalParameterIds()”. Al no existir en nuestro código otros parámetros con la letra K podemos obtenerlos con la siguiente instrucción:

```
ID = list(enumerate([x for x in r.getGlobalParameterIds() if ("K"
    in x or "k" in x)]));
```

Si hacemos uso de la instrucción “r.getGlobalParameterValues()” se puede observar que aparece un parámetro más con valor 1 al comienzo. Dicho valor no esta definido en el modelo y aparece siempre al utilizar esta función, por lo que habrá que tener en cuenta esto para hacer uso de los parámetros correctamente.

Para simular el uso de los valores con incertidumbre, se ha creado una función denominada “simulaIncertidumbre”, a la cual se le pasa como parámetro la variable en la que se encuentra almacenado el modelo, el valor de la variable k y el id del parámetro con dicho valor. En esta función se calcula el número difuso, obteniendo un rango calculado a partir del 50 y 150% del valor original y son almacenados en un array para hacer uso de ellos posteriormente. Recorriendo estos valores se establece el valor del parámetro antes de iniciar la simulación, haciendo uso de la instrucción “exec('r.%s = %f' % (parametro,val))”, se indica también la semilla que se esta utilizando para no variar la simulación y se simula. Los resultados de los tres valores se superpondrán en una gráfica, dando como resultado el aspecto similar de una simulación Fuzzy que puede servir para ver como afecta el parámetro a una especie concreta.

Finalmente recorreremos la lista de parámetros k que habíamos obtenido previamente, llamando a la función simulaIncertidumbre() explicada anteriormente, de esta manera conformaremos un grid de gráficas para ver como afectan los 7 parámetros que teníamos inicialmente en nuestro modelo.

Al comienzo de la libreta el usuario dispone de una celda Python en la que poder modificar los siguientes parámetros para la ejecución de la misma:

- int semilla: Semilla para realizar siempre la misma simulación estocástica.
- int n: Intervalo de tiempo a mostrar en horas.
- double diferencia: Porcentaje de diferencia respecto al valor inicial.
- string especie: Especie del modelo que se desea consultar.
- int ID_K: ID del parámetro k, en un rango de 1 a 7, para su consulta individual.

5. Experimentos y resultados

A continuación, en esta sección, se mostrarán los resultados obtenidos para los desarrollos implementados anteriormente y se realizará un análisis de los mismos. Para ello, dividiremos este capítulo en dos apartados, uno para las simulaciones del nuevo modelo de radiación solar y su correspondiente comparativa respecto a los modelos antiguos, y por otra parte, un apartado en el que veremos la simulación Fuzzy y estocástica.

Las pruebas han sido realizadas en un ordenador de escritorio equipado con un Intel Core i5 10600k (6 núcleos - 12 hilos) y 16 Giga Bytes de memoria RAM. Además se ha utilizado también el supercomputador GALGO para la simulación de los modelos en una fase previa del proyecto, donde el tiempo de ejecución de los nuevos modelos bioquímicos con los antiguos de radiación solar era demasiado elevado y se requería de él para su finalización.

5.1. Pruebas de los modelos de radiación solar

5.1.1. Modelo Épsilon-Delta

Como se comentó anteriormente, uno de los problemas encontrados en los modelos creados previamente era la precisión a la hora de ajustar el fotoperiodo de cada día. Comenzaremos viendo los resultados del ajuste proporcionado por el modelo Épsilon-Delta y las ventajas de utilizar procesamiento paralelo a la hora de buscar este ajuste.

Se han seleccionado cuatro días del año para comprobar el ajuste en la localización de Albacete, cada uno correspondiente a una estación del mismo para realizar la comparativa entre el antes y el después del ajuste. En la figura 5.1 se puede observar los resultados obtenidos antes de realizar el ajuste para los cuatro días seleccionados.

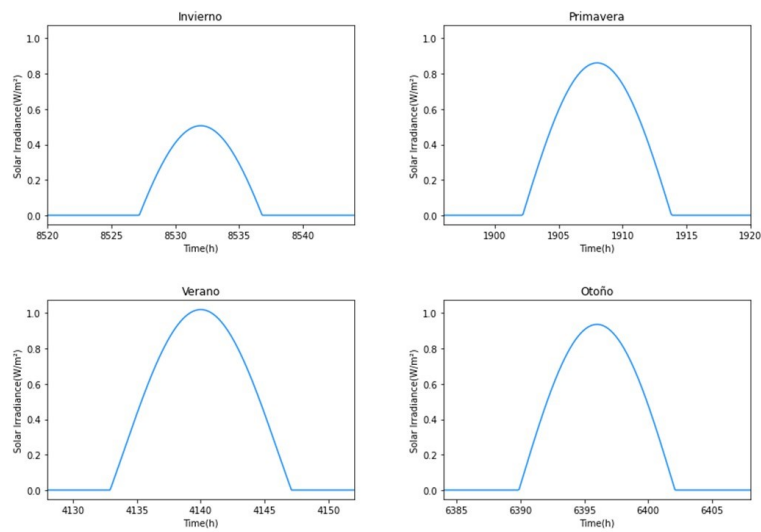


Figura 5.1: Fotoperiodo antes del ajuste Albacete

Mientras tanto, en la figura 5.2 se puede observar el error que se produce en el día de verano, siendo la línea azul el resultado después del ajuste y la gris el producido antes. A simple vista puede parecer un error bajo, pero dado que la radiación solar es el principal parámetro de entrada para los modelos bioquímicos y estos son especialmente sensibles a dicho parámetro, es necesario ajustarlo lo máximo posible a la realidad, ya que un aumento de una hora más o menos puede suponer un gran impacto en los resultados finales o incluso imposibilitar la simulación del modelo.

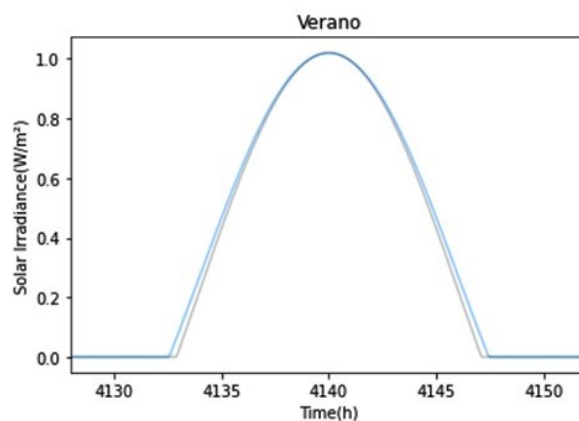


Figura 5.2: Comparativa ajustes Albacete

Para medir la precisión proporcionada se han utilizado las ecuaciones de cálculo del fotoperiodo (4.1) explicadas anteriormente y comprobado la longitud del hueco producido por la curva de radiación, es decir ver cuando esta es diferente de cero. Los resultados se recogen en la tabla 5.1.

	Antes del ajuste	Fotoperiodo real	Después del ajuste
Primavera (Día 79)	11.7 h	11.48 h	11.5 h
Verano (Día 172)	14.1 h	14.69 h	14.9 h
Otoño (Día 266)	12.30 h	12.32 h	12.30 h
Invierno (Día 355)	9.70 h	9.30 h	9.30 h

Tabla 5.1: Resultados ajustes Albacete

En cuanto al tiempo de ejecución de la celda Python, que se encarga de buscar los valores de delta y épsilon mas adecuados para la localización seleccionada, se ha obtenido un tiempo medio de 26 segundos¹ tras realizar 10 ejecuciones de esta y haciendo uso de los 12 hilos del procesador. La búsqueda de estos valores ha sido realizada en un Numpy Arange definido en el intervalo de -1 a 1 con un espaciado de 0.05, y obteniendo como resultados un épsilon de -0.05 y un delta de 0.05. Esto supone una mejora respecto a la primera versión que se implementó, en la que se disponía de dos bucles anidados para comprobar los valores, en los que se obtenía un tiempo de ejecución superior a los 2 minutos 45 segundos para este mismo intervalo de búsqueda. Este proceso solo es necesario la primera vez que se quiere estudiar una localización determinada. Una vez conocido estos valores, el usuario puede acotar el rango o utilizar los últimos mejores encontrados directamente.

Como se comentó, este problema de precisión se agrava conforme nos desplazamos a latitudes más extremas. En las siguientes imágenes (Fig 5.3) se muestra la comparativa de las diferentes estaciones en la localidad de Reykjavik, donde se aprecia bastante más el error cometido en los días de verano, que no llegamos a tener noche en ningún (radiación a 0) momento cuando realmente si que debería ocurrir.

A continuación se muestra una imagen (Fig 5.4) del resultado obtenido para un año completo, donde se puede ver como se ha solucionado el error producido en los días centrales del año (verano) y una comparación para el primer día de verano, donde la gráfica roja representa el fotoperiodo antes del ajuste y la azul después. Sin embargo, el fotoperiodo obtenido para esta latitud en algunos días no es tan exacto como para el caso de Albacete, por lo que haremos uso del modelo basado en la envolvente para dar una posible solución a este inconveniente.

¹Los tiempos de ejecución variarán en función del tamaño del intervalo de búsqueda y la cantidad de valores que se encuentren en este.

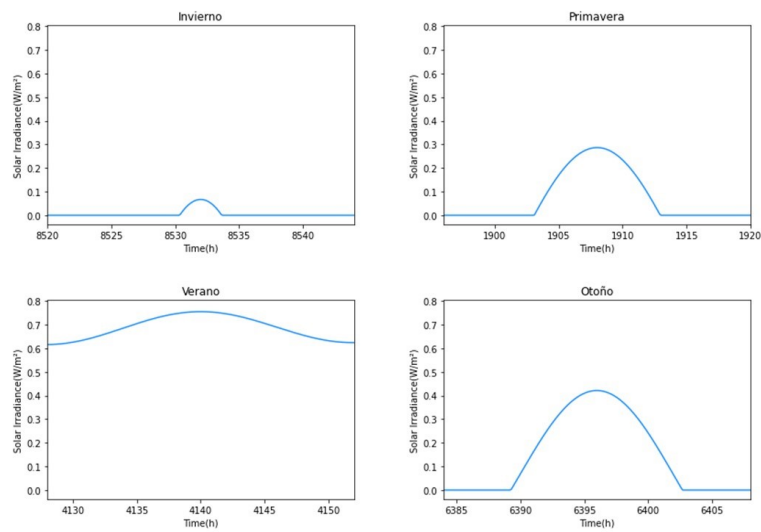


Figura 5.3: Fotoperiodo antes del ajuste Reykjavik

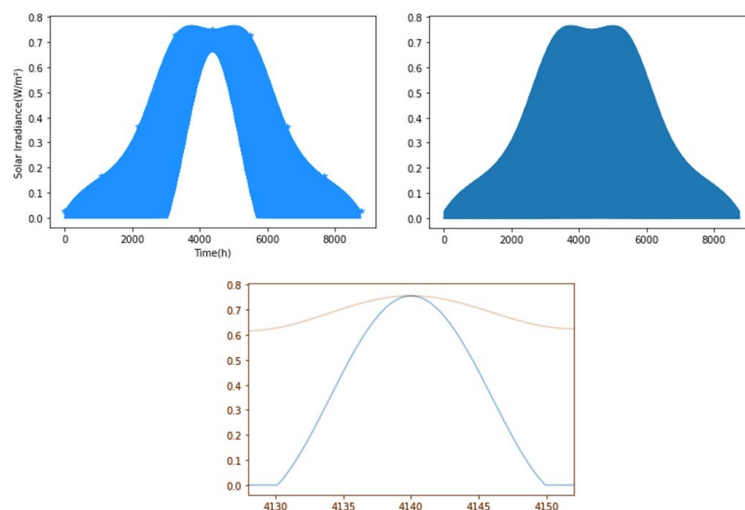


Figura 5.4: Mejora Reykjavik

Continuando con un ejemplo de ejecución para la localidad de Albacete, tras realizar el ajuste oportuno se calcula y muestra al usuario la curva de radiación creada aleatorizada. En la siguiente imagen (Fig 5.5) se puede ver el resultado final para un año completo y para los primeros días de este.

Finalmente se carga y simula el modelo para recibir los datos, el tiempo de ejecución de este modelo en el ordenador de sobremesa destinado a las pruebas ronda los 133 minutos tras una media de 10 ejecuciones (30 minutos de carga + 103 de simulación), mientras que haciendo uso de Galgo el tiempo sobrepasaba en todos los casos los 230 minutos, debido al

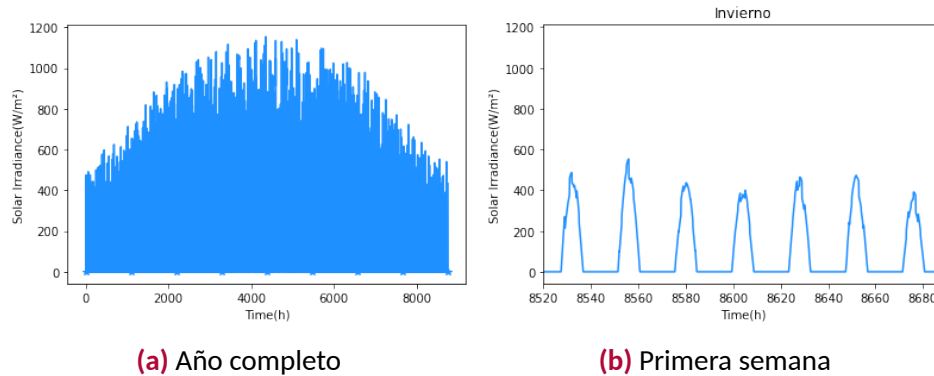


Figura 5.5: Radiación aleatorizada Albacete

retardo en la lectura del modelo y la escritura de los resultados, por lo que para este nuevo modelo se ha podido comprobar que el uso de un supercomputador no es estrictamente necesario como si ocurría antes.

En cualquier caso, la mejora de tiempo de este nuevo modelo frente al antiguo basado en temperaturas propuesto en el Trabajo de Fin de Grado es bastante considerable considerable, ya que se llegaron a obtener tiempos de ejecución superiores a las 7 horas para este mismo modelo del ciclo de GAR.

Una vez finalizada la ejecución de la libreta el usuario puede consultar los resultados obtenido de todas las especies implicadas en el modelo mediante el fichero generado en la carpeta raíz de la libreta y llamado “resultados.txt”².

5.1.2. Modelo Envolvente

En estas pruebas nos centraremos en ver como se consigue solucionar el problema de los fotoperiodos introducido anteriormente y la mejora de rendimiento obtenido al cambiar el método de asignación de la radiación diaria dentro del modelo.

En cuanto a los tiempos de ejecución de este modelo son extremadamente positivos, puesto que, como se ha comentado anteriormente, el número de eventos se ve reducido, se necesita únicamente una media de 28 segundos para la carga del modelo y de 5 minutos para la simulación frente a los 30 minutos de carga del modelo épsilon-delta y más de una hora para su simulación.

En la siguiente imagen (Fig. 5.6) se puede comprobar la envolvente calculada para la lo-

²También se encuentra disponible para su consulta los datos de radiación solar en el fichero “resultadosF1.txt”

calidad de Albacete respecto a la curva de radiación con los datos reales del PVGIS, la cual es usada posteriormente en el fichero del modelo para el cálculo de la radiación diaria.

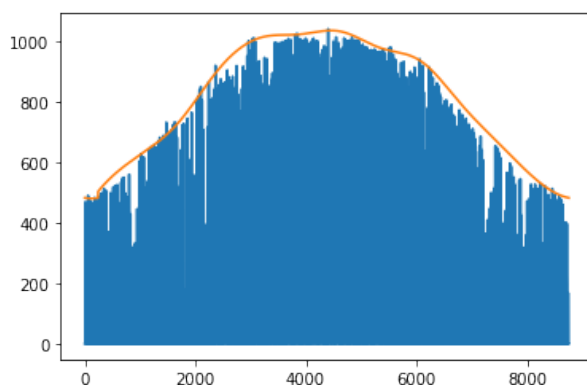


Figura 5.6: Comparativa envolvente Albacete

En cuanto a las pruebas realizadas, en las siguientes imágenes (Fig.5.7) se muestra una comparativa entre la radiación solar obtenida por el modelo ϵ -delta, el modelo envolvente y el PVGIS (Photovoltaic Geographical Information System) [PVG,], donde se puede apreciar que tanto la forma de la gráfica para un año completo como para una semana, muestran niveles y forma parecidas, por lo que prácticamente funcionan de manera muy similar. Cabe destacar que el hecho de añadir eventos aleatorios permite una mayor flexibilidad a la hora de generar la radiación solar, ya que no tiene por qué estar estrictamente ligada a un valor en un determinado momento. Además se puede llegar a obtener mayor precisión que haciendo uso únicamente del PVGIS ya que los datos de este son tomados cada hora.

De igual manera pasa si se comparan las especies involucradas en la ruta metabólica del modelo, en la siguiente imagen (Fig. 5.8) se puede apreciar cómo no hay una gran diferencia respecto un modelo u otro.

En cuanto a la precisión obtenida en el fotoperiodo se ha comparado también los resultados obtenidos en Reykjavik, ya que era una latitud que no terminaba de funcionar correctamente con el modelo anterior. En la figura 5.9 se muestra una comparativa para los días de invierno, otoño, primavera y verano antes de introducir los eventos aleatorios, siendo la línea azul la perteneciente al modelo ϵ -delta después de un primer ajuste y la roja al modelo envolvente. Se puede apreciar que el ajuste es mucho mejor en este último, ya que coincide exactamente con el fotoperiodo calculado al forzar el inicio y fin de la radiación en la salida y puesta del sol.

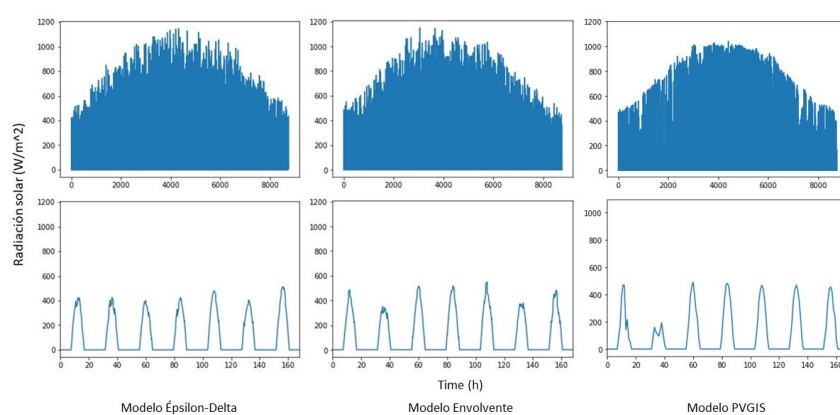


Figura 5.7: Comparativa radiación (Albacete)

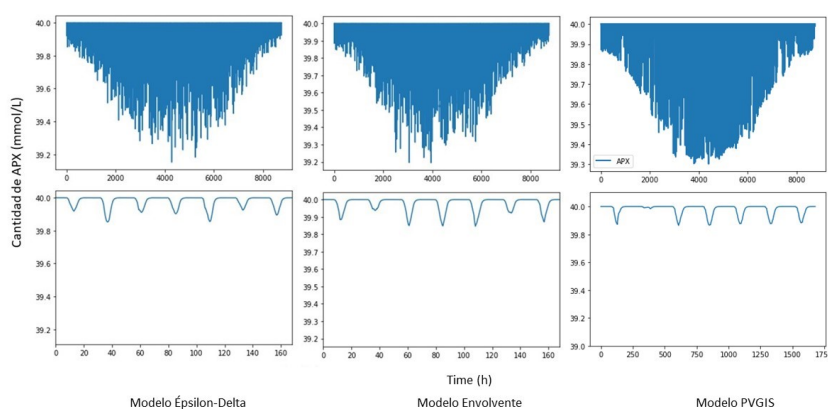


Figura 5.8: Comparativa APX (Albacete)

Se puede apreciar como para los días de otoño e invierno el fotoperiodo calculado en el modelo envolvente debería ser menor al calculado en el modelo épsilon-delta y para los días de verano y primavera debería de ser mayor. En la imagen anterior también se pueden ver diferencias en la cantidad de radiación máxima en los días de invierno y primavera, esto es debido a la manera en la que se toman los puntos máximos de referencia diarios, aunque no es significativo para esta prueba.

5.2. Pruebas de los modelos Fuzzy y estocásticos

Uno de las principales dudas a la hora de realizar simulaciones estocásticas y con parámetros con incertidumbre, es seleccionar que parámetros de nuestro modelo vamos a considerar con incertidumbre, para ello, se ha realizado una prueba en la que a partir de un modelo del ciclo de Calvin, se pueda ver como afecta la modificación de las constantes a las

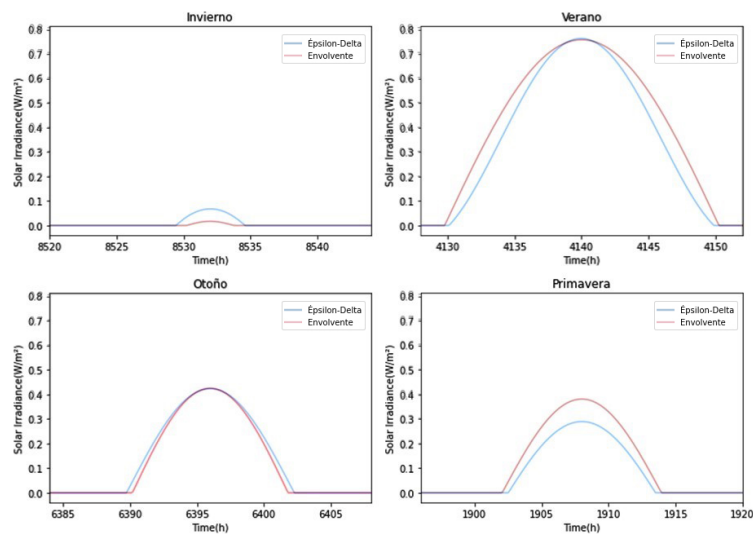


Figura 5.9: Comparativa ajuste Reykjavik

especies de este.

Tras ejecutar la libreta por primera vez veremos una primera gráfica interactiva (Fig. 5.10, desde la que podremos seleccionar las especie a consultar (por defecto se muestran todas), y ver como evoluciona a lo largo de la simulación.

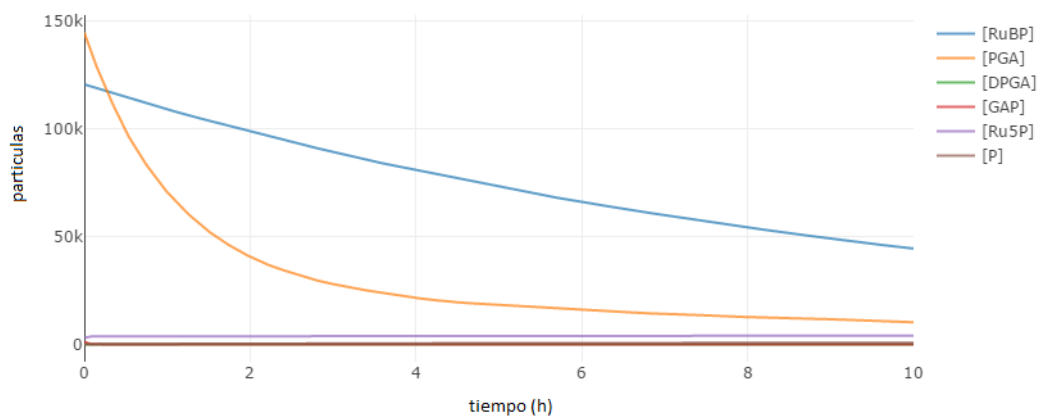


Figura 5.10: Gráfica interactiva Tellurium

Tras esto, la libreta nos proporcionara otras dos figuras, la primera de ellas centrándose

en una variación de una constante k_n y otra en la que compara la especie modificando todas las constantes disponibles. En las siguientes imágenes se muestran algunos ejemplos para las especies RuBP (Fig 5.11) y DPGA (Fig 5.12)³.

Para el RuBP se puede observar que únicamente destaca la modificación de k_1 , quedando prácticamente iguales el resto de simulaciones con los otros parámetros de k . En el caso de la especie DPGA, se puede observar que proporciona la información necesaria para incentivar la realización de un estudio a fondo modificando los valores k_1 , k_3 y k_6 , ya que son los que presentan mayor incertidumbre y sería recomendable hacer uso de un número Fuzzy en su simulación.

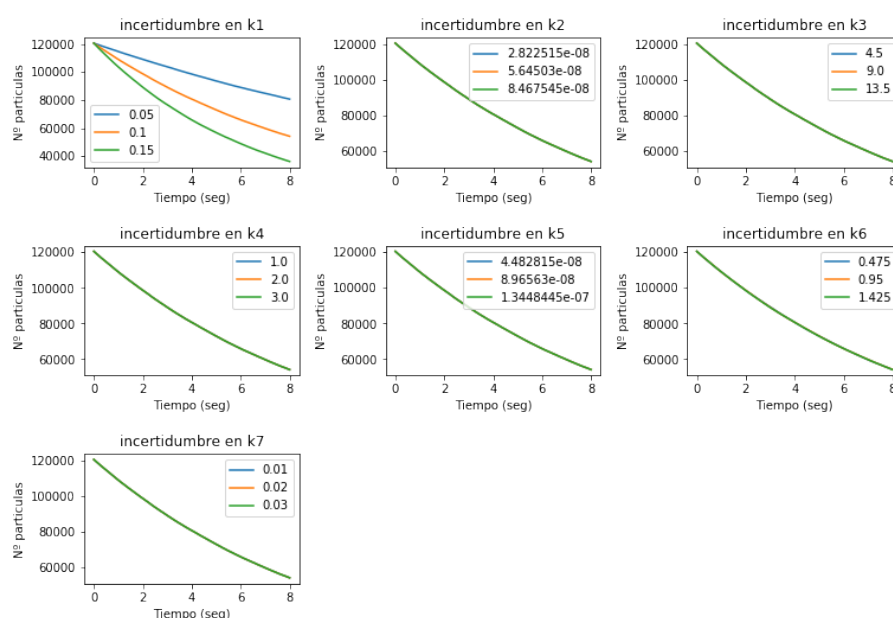


Figura 5.11: Comparativa incertidumbre RuBP

³En los archivos adjuntos se pueden encontrar imágenes para el resto de especies del modelo

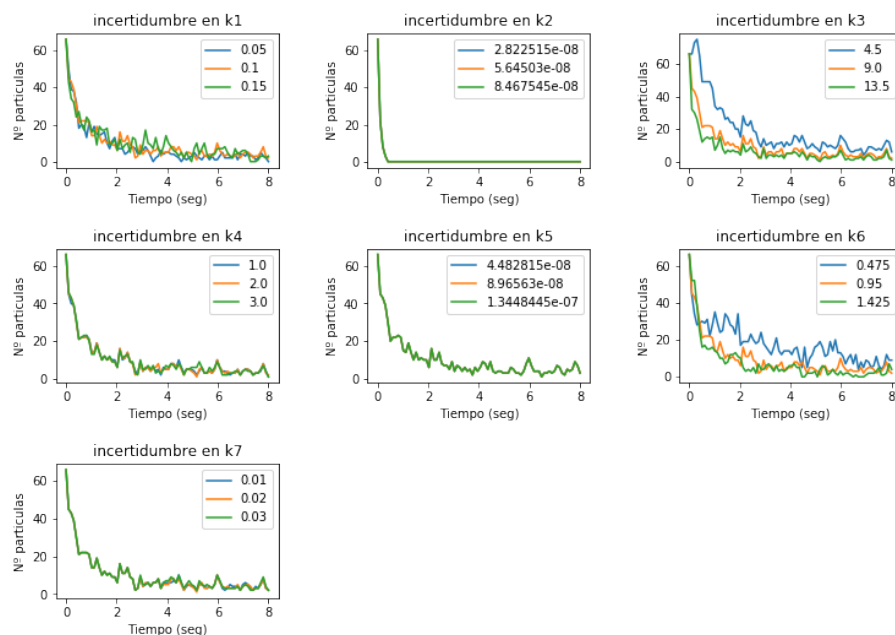


Figura 5.12: Comparativa incertidumbre DPGA

También podemos apreciar el característico resultado de las simulaciones estocásticas, ya que, debido a que se está trabajando con moléculas y su comportamiento no es determinista producen unas gráficas mucho más irregulares, debido a las fluctuaciones, que en casos que si son deterministas. Se puede comprobar que se cumple la teoría de que a menor número de moléculas se producen mayores fluctuaciones, como puede ser el caso del DPGA, en el que tenemos entorno a 70 moléculas de máxima, mientras que en el caso de RuBP estas son casi inapreciables al tener del orden de 120000 de máxima. En la siguiente imagen (Fig. 5.13) se puede ver una comparativa de la especie DPGA en una simulación si fuese determinista y la estocástica.

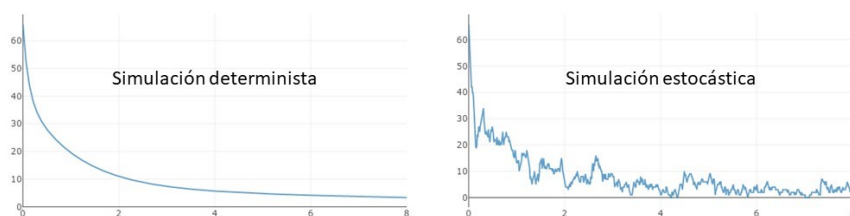


Figura 5.13: Comparativa simulaciones DPGA

Al encontrar un componente de aleatoriedad en las simulaciones estocásticas, se han realizado también varias ejecuciones de estas y se ha podido comprobar que la media de estas simulaciones tiende a ser el resultado de una simulación determinista como se puede ver en la figura 5.14, en la que se superponen 10 ejecuciones estocásticas (azul) y su media (rojo) frente a la determinista (verde).

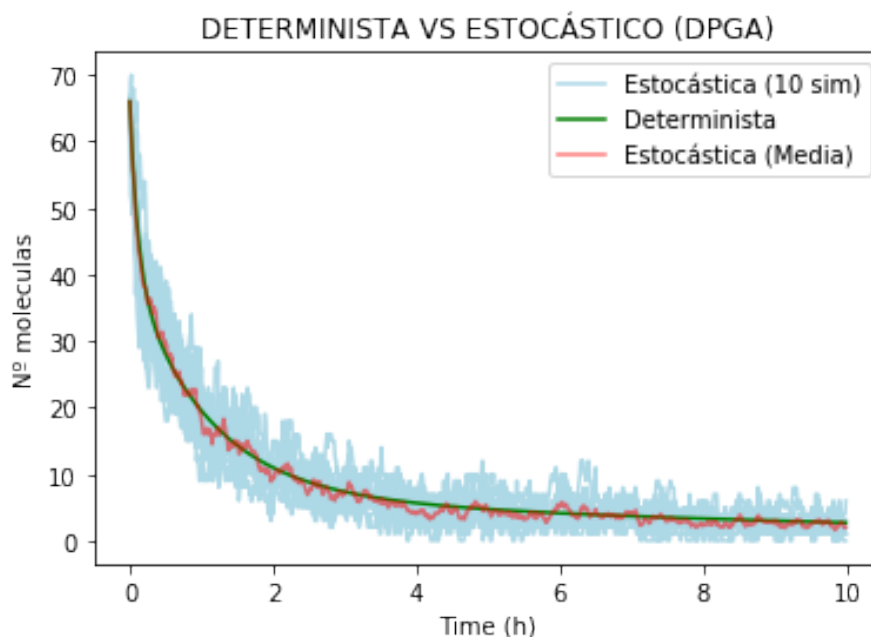


Figura 5.14: Media de las simulaciones estocásticas vs determinista DPGA

Con esta información proporcionada por la libreta Python procedemos a ejecutar el archivo de Snoopy llamado "Ciclo de Calvin Fuzzy (Zhu2009)", en el cual podremos comprobar la simulación, con la particularidad, de que dispondremos de la red de Petri y del gráfico de pertenencia de los parámetros Fuzzy.

En primer lugar debemos modificar las constantes para establecer el valor Fuzzy a nuestra constante con la información previamente obtenida. En este caso, vamos a establecer como parámetro difuso k_1 y veremos que la simulación de Snoopy coincide con la de Tellurium. Para realizar la simulación debemos seleccionar el apartado View y Start Simulation-Mode y establecer la configuración del simulador como la que se muestra en la siguiente imagen (Fig. 5.15).

Una vez configurado comenzamos la simulación, esta tardará en ejecutarse en torno a los

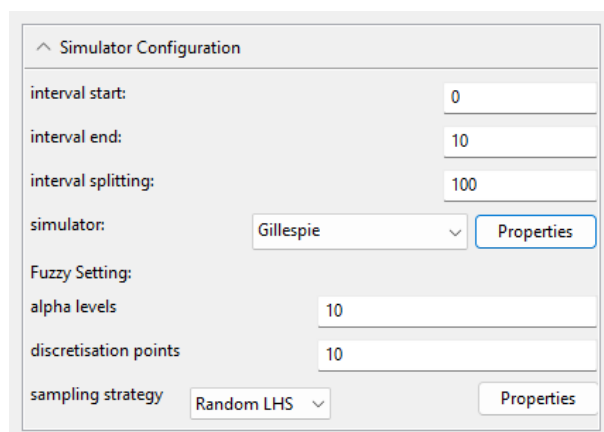


Figura 5.15: Configuración simulador Snoopy

2 minutos⁴, y para ver los resultados seleccionamos Default View. Esto nos abrirá una nueva ventana (Fig. 5.16) en la que interactuar con la gráfica mostrada, dando a elegir las especies a mostrar y exportar los resultados como imagen o CSV. También, uno de los aspectos más interesantes de este programa es la posibilidad de seleccionar el tiempo de simulación y ver como se modifica el triángulo que indica el grado de pertenencia del parámetro haciendo uso del slider en la parte inferior derecha (Fig. 5.17).

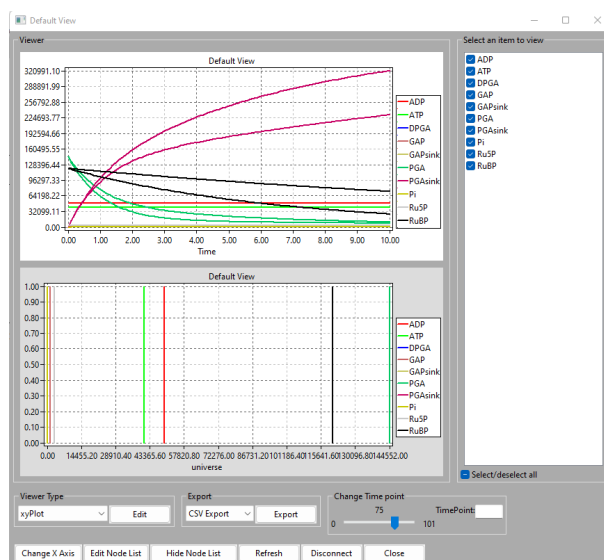


Figura 5.16: Resultados simulación Snoopy

⁴Puede variar en función del equipo que ejecute la simulación

En la figura 5.11 se podía observar que para el último instante de la simulación modificando el parámetro k_1 para la especie RuBP obteníamos un número mínimo de moléculas del orden de 40000, mientras que para el resto de parámetros de 60000. Si observamos los resultados obtenidos por Snoopy (Fig. 5.17) podemos ver como son bastante cercanos. La imagen proporciona los datos del último instante de simulación en el que si nos fijamos en la gráfica inferior, el triángulo nos indica que los valores cercanos a 40000-50000 tienen un grado de pertenencia bastante elevado (en torno al 80%).

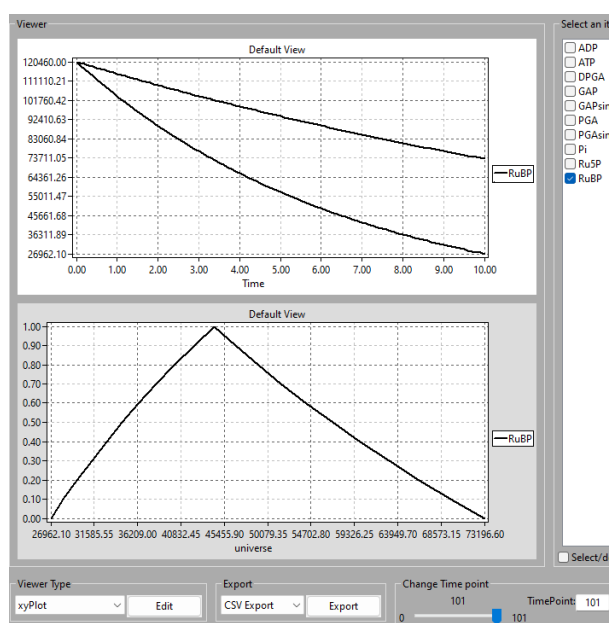


Figura 5.17: Resultados RuBP Snoopy

6. Conclusiones y propuestas

Tal y como se ha podido comprobar a lo largo de este documento y en los ficheros adjuntos, la simulación de sistemas bioquímicos es un proceso cada vez más sencillo de elaborar y que puede ser utilizado por personas ajenas al campo de la informática, pero que se complica conforme se expanden los modelos bioquímicos o se intentan hacer lo más realistas posibles.

En cuanto a los nuevos modelos de radiación solar, se ha podido ver una mejora bastante importante del rendimiento a la hora de la carga y simulación de los sistemas bioquímicos, a pesar de que el sistema en sí era más complejo que en las primeras versiones presentadas en el Trabajo de Fin de Grado. Se ha podido comprobar de la misma manera, que no es necesario un realismo extremo a la hora de modelarlo, si no que con algo más sencillo, pero preciso, podemos obtener resultados bastante prometedores que pueden ser analizados por expertos en el campo de la bioquímica.

Respecto al uso de la supercomputación en la simulación de este tipo de sistemas, de momento no se ha encontrado una total necesidad de su implementación para llevar a cabo esta tarea, ya que, actualmente, por la manera en la que funciona el motor de simulación RoadRunner de Tellurium no se puede aprovechar prácticamente nada de esto. Si es cierto que puede ser de gran ayuda a la hora de ejecutar modelos como el de ϵ -delta, en el que existe paralelización para la búsqueda de parámetros que ayuden a optimizar el modelo, o incluso como soporte para realizar simulaciones de sistemas muy detallados, en los que se requieran tiempos demasiado elevados para su carga y simulación y que no serían viables para un ordenador convencional ya que afectarían a su rendimiento para realizar otras tareas simultáneamente.

También se ha completado el modelo del ciclo de Calvin-Benson, pendiente del trabajo de Fin de Grado y que junto a las redes de Petri han supuesto una mejora sustancial de dicho proyecto.

En la parte de simulación estocástica se han visto resultados muy prometedores, ya que se ha conseguido simular un intervalo de tiempo y con números de moléculas bastante más

elevado de lo que se había visto hasta ahora de una manera estable y con resultados lógicos. Si bien es cierto que no se ha podido implementar con una entrada de radiación debido a la dificultad que esto suponía debido a que era necesario un estudio más en profundidad en el campo de la química y la física, se puede considerar como un excelente punto de partida para futuras investigaciones en este tipo de simulaciones.

En cuanto al uso de números Fuzzy y lógica difusa, se ha podido observar que es una buena manera de ayudar a los expertos a la hora de escoger los parámetros para sus modelos, debido a la gran cantidad de información y a la vez tan discrepante, que se pueden encontrar en las diferentes bases de datos. La implementación realizada tanto en Tellurium como en Snoopy permite obtener unos resultados de manera rápida y muy visuales, que puede ser utilizada para elegir rangos de valores.

Por último, hay recalcar que, tanto las libretas de Python, como archivos de diferentes software y este propio documento, están dirigidos a dar soporte a investigaciones dentro del campo de la bioquímica y se ha comprobado que es muy importante la capacidad de comunicación dentro de los equipos multidisciplinares, para que se puedan elaborar y hacer uso de los materiales entregados, proporcionando una mayor accesibilidad y facilidad de uso al trabajo realizado al resto de la comunidad.

A. Ficheros Adjuntos

En este anexo se listan y se proporciona información sobre los ficheros que han sido adjuntados junto a esta memoria y que forman parte del Trabajo de Fin de Máster “Redes de Petri, lógica difusa y simulación estocástica de rutas metabólicas. Un caso de estudio”. Estos ficheros pueden ser descargados a través del siguiente repositorio GIT:

<https://github.com/JaviJimRue99/TFM2022.git>

- **Modelos Épsilon-Delta:** En esta carpeta se incluyen los ficheros relacionados al capítulo 4.1.1, en el que se detalla la implementación del modelo épsilon-delta.

Encontraremos una libreta Python llamada “RS_Generator.ipynb”, este será el archivo a ejecutar tanto para realizar el ajuste de la radiación, mediante los parametros épsilon y delta, como para realizar la simulación del modelo (Dentro de ella se pueden configurar diversos parámetros).

Existen también diversos ficheros txt, entre los que destacan los de resultados, “resultados.txt” para consultar los resultados obtenidos para cada especie del modelo en la simulación, “resultdosF1.txt”, para ver los resultados de la radiación generada, y por último, “modeloFinal_v2.txt”, que contendrá el modelo ya ensamblado por el código Python para la localización seleccionada.

Se incluyen también diversos archivos Python, los cuales contienen las funciones implementadas para el correcto funcionamiento de la libreta, así como una carpeta llamada “Referencias” en la que se encuentran datos de diferentes localizaciones del planeta para su simulación.

Por último se puede encontrar también una captura de pantalla de los tiempos de carga y simulación orientativos obtenidos por el modelo utilizado.

COMPLETAR CUANDO SE CAMBIE Y REVISE EL APARTADO

Es recomendable ejecutar dicha libreta desde el propio notebook de Tellurium, el cual se puede descargar desde la siguiente URL (<https://github.com/sys-bio/tellurium#front-end-1-tellurium-notebook>) o mediante Visual Studio Code.

- **Modelo Envolvente:** En la carpeta del modelo envolvente se puede encontrar la libreta Python relacionada al capítulo 4.1.2.

Se dispone también de una carpeta en llamada Modelo, en la que se encuentran los txt del modelo generado en Antimony ya completo como sus dos partes por separado y una captura con los tiempos de ejecución de la libreta. También se incluye una carpeta de Referencias en la que se encuentran datos del PVGIS para diferentes localizaciones. Estas localizaciones pueden ser ejecutadas desde la libreta Python cambiando el parámetro Localización.

Por último se dispone de dos ficheros txt en los que se encuentran los resultados para un año completo en la localidad de Albacete, tanto de los valores de radicación como de los reactivos involucrados en el modelo.

- **Estocástico y Fuzzy Tellurium:** Contiene una libreta de Python ejecutable correspondiente al capítulo 4.3 para la comprobación de parámetros con incertidumbre en Tellurium.

También se incluye una carpeta con imágenes de los resultados generados para los diferentes reactivos del modelo y sus parámetros, un archivo txt con el modelo en lenguaje Antimony para su consulta (Arnold2011_Zhu2009_Modelo_Estocastico.txt) y fichero excel para la conversión de los reactivos a moléculas para la simulación estocástica de manera automática.

- **Fuzzy Snoopy:** Contiene el fichero de Snoopy relacionado al capítulo 4.2 para la ejecución de simulaciones estocásticas con parámetros Fuzzy. Para su ejecución es necesario descargar el software Snoopy desde la siguiente URL:

<https://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy#downloads>

Referencia bibliográfica

- [BED,] Brenda-enzymes.org: BRENDA Enzyme Database
<https://www.brenda-enzymes.org/>
(Último acceso: 04/09/2022).
- [PVG,] JRC Photovoltaic Geographical Information System (PVGIS) - European Commission
https://re.jrc.ec.europa.eu/pvg_tools/en/
(Último acceso: 04/09/2022).
- [SCR,] Scrum.org: What is Scrum?
<https://www.scrum.org/resources/what-is-scrum>
(Último acceso: 04/09/2022).
- [Arn, 2011] (2011). BioModels - Arnold2011_Zhu2009_CalvinCycle | BioModels
<https://www.ebi.ac.uk/biomodels/BIONMD0000000388>
(Último acceso: 04/09/2022).
- [SBM, 2022] (2022). SBML.org: What is SBML?
<https://sbml.org/documents/what-is-sbml/>
(Último acceso: 04/09/2022).
- [Sea, 2022] (2022). Bionumbers: Search BioNumbers - The Database of Useful Biological Numbers
<https://bionumbers.hms.harvard.edu/search.aspx?trm=chloroplast>
(Último acceso: 04/09/2022).
- [Sno, 2022] (2022). Dssz.informatikdata: structures and software dependability | Software / Snoopy
<https://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy#generaldescription>
(Último acceso: 04/09/2022).
- [Ant, 2022] (2022). Tellurium: Antimony Reference — tellurium 2.2.2.1 documentation
<https://tellurium.readthedocs.io/en/latest/antimony.html>
(Último acceso: 04/09/2022).

-
- [Tel, 2022] (2022). Tellurium home page | tellurium
<http://tellurium.analogmachine.org/>
(Último acceso: 04/09/2022).
- [Arnold and Nikoloski, 2011] Arnold, A. and Nikoloski, Z. (2011). A quantitative comparison of calvin-benson cycle models. *Trends in Plant Science*, 16:676–683.
- [Chaouiya, 2007] Chaouiya, C. (2007). Petri net modelling of biological networks. *Briefings in Bioinformatics*, 8:210–219.
- [Chow and Levermore, 2007] Chow, D. and Levermore, G. (2007). New algorithm for generating hourly temperature values using daily maximum, minimum and average values from climate models. *Building Services Engineering Research & Technology - BUILD SERV ENG RES TECHNOL*, 28:237–248.
- [de J. Pérez Jiménez,] de J. Pérez Jiménez, M. Modelos estocásticos. algoritmo de gillespie. extensiones al marco de los sistemas p
<https://www.cs.us.es/~marper/docencia/bioinformatics/temas/estocasticos-mezcla-2014-2015.pdf>
(Último acceso: 04/09/2022).
- [Esquinas-Ariza et al., 2022] Esquinas-Ariza, R. M., Jiménez-Ruescas, J., Díaz, B., Macià, H., and Valero, E. (2022). Simulating solar irradiance through am wave equations for metabolic pathways (The Fourteenth International Conference on Bioinformatics, Biocomputational Systems and Biotechnologies - Biotechno 2022).
- [Fowler et al.,] Fowler, S., Roush, R., and Wise, J. 5.3 the calvin cycle - concepts of biology | openstax
<https://openstax.org/books/concepts-biology/pages/5-3-the-calvin-cycle>
(Último acceso: 04/09/2022).
- [Gersbeck-Schierholz, 2022] Gersbeck-Schierholz, B. (2022). *The Fourteenth International Conference on Bioinformatics, Biocomputational Systems and Biotechnologies*
<https://www.iaria.org/conferences2022/BIOTECHNO22.html>
(Último acceso: 04/09/2022).
- [Gillespie, 1977] Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81:2340–2361.
- [Jiménez,] Jiménez, J. Tellurium como herramienta de modelado y testing para rutas metabólicas (Trabajo de Fin de Grado), year = 2021,.
- [Kacprzyk, 2005] Kacprzyk, J. (2005). Springer link: Fuzzy number
https://link.springer.com/chapter/10.1007/3-540-32366-X_5
(Último acceso: 04/09/2022)
. *First Course on Fuzzy Theory and Applications*, pages 129–151.
-

- [Medley et al., 2018] Medley, J. K., Choi, K., König, M., Smith, L., Gu, S., Hellerstein, J., Seal-fon, S. C., and Sauro, H. M. (2018). Tellurium notebooks—an environment for reproducible dynamical modeling in systems biology. *PLOS Computational Biology*, 14:e1006220.
- [Peleg et al., 2005] Peleg, M., Rubin, D., and Altman, R. B. (2005). Using Petri Net tools to study properties and dynamics of biological systems. *Journal of the American Medical Informatics Association*, 12:181–199.
- [Petri, 1966] Petri, C. (1966). *Communication with Automata*. AD-630. Rome Air Development Center, Research and Technology Division.
- [Smith et al., 2009] Smith, L. P., Bergmann, F. T., Chandran, D., and Sauro, H. M. (2009). Antimony: a modular model definition language. *BIOINFORMATICS APPLICATIONS NOTE*, 25:2452–2454.
- [Valero et al., 2016] Valero, E., Macià, H., la Fuente, I. M. D., Hernández, J. A., González-Sánchez, M. I., and García-Carmona, F. (2016). Modeling the ascorbate-glutathione cycle in chloroplasts under light/dark conditions. *BMC systems biology*, 10.
- [Zadeh, 1988] Zadeh, L. A. (1988). Fuzzy logic. *Computer*, 21:83–93.
- [Zhu et al., 2009] Zhu, X. G., Alba, R., and de Sturler, E. (2009). A simple model of the calvin cycle has only one physiologically feasible steady state under the same external conditions. *Nonlinear Analysis: Real World Applications*, 10:1490–1499.