UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA AGRONÓMICA, ALIMENTARIA Y DE BIOSISTEMAS

GRADO EN BIOTECNOLOGÍA

DEPARTAMENTO DE MATEMÁTICA APLICADA

Modelo matemático-computacional del desarrollo y distribución de estomas en hojas de coníferas

TRABAJO FIN DE GRADO

Autor: Rodrigo Monturiol Rolland

Tutor: Juan Carlos Nuño

Noviembre de 2025

UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Técnica Superior De

Ingeniería Agronómica, Alimentaria y de Biosistemas

GRADO DE BIOTECNOLOGÍA

MODELO MATEMÁTICO-COMPUTACIONAL DEL DESARROLLO Y DISTRIBUCIÓN DE ESTOMAS EN HOJAS DE CONÍFERAS

TRABAJO FIN DE GRADO

Rodrigo Monturiol Rolland

MADRID, 2025

Tutor: Juan Carlos Nuño

Dpto. de Matemática Aplicada

TITULO DEL TFG - MODELO MATEMÁTICO-COMPUTACIONAL DEL DESARROLLO Y DISTRIBUCIÓN DE ESTOMAS EN HOJAS DE CONÍFERAS

Memoria presentada por Rodrigo Monturiol Rolland para la obtención del título de Graduado en Biotecnología por la Universidad Politécnica de Madrid

Fdo: Rodrigo Monturiol Rolland

VºBº Tutor y Director del TFG

D. Juan Carlos Nuño
Profesor Titular de Universidad
Dpto. de Matemática Aplicada
Escuela Técnica Superior de Ingeniería de Montes, Forestal y de Medio Natural – Universidad Politécnica de Madrid

Madrid, noviembre de 2025

“He peleado la buena batalla,
he acabado la carrera,
he guardado la fe.”
~ 2 Timoteo 4, 7

AGRADECIMIENTOS

A Dios, por darme la vida y todo lo que hay en ella.

A mis padres, Mamá y Papá, y a mi hermano, Santi, por haber sido siempre todo lo que necesité que fuerais y por enseñarme que lo esencial es invisible a los ojos. Por haber invertido tanto tiempo, dinero y estreses en que tuviera una vida plena y en que consiguiese sacar esta carrera.

Al resto de mi familia, en especial a mis abuelas Vivís y Ana, que tanto me han acompañado, siendo un pilar fundamental de mi vida y una de las causas de ser quien y como soy día a día.

A mis amigos, en especial a Elena, Juan y Nano, que probablemente sepan más que nadie el esfuerzo y sacrificio que ha supuesto a una cabeza como la mía pelearse todos estos años contra un muro llamado Biotecnología. Por hacerme pasar tantos días de risas, antes, ahora y siempre.

A mi tutor, Juan Carlos, por haber estado tan atento y a disposición todos estos meses, por mostrar un interés incluso mayor que el mío por este proyecto, por ayudarme en todos los pasos a dar. A todo el grupo de Montes, a Álvaro, Nacho, Juanma, Salvia y Unai, por haber contribuido cada uno a este proyecto y por haberme hecho sentir parte de la familia desde el primer día.

A todos mis compañeros, en especial a Iván, Melqui y Rubén, y a todos los profesores que me han acompañado a lo largo de estos últimos años, por contribuir cada uno con su granito de arena.

A mí, por no haberme rendido.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

CAPÍTULO 2

CAPÍTULO 3

CAPÍTULO 4

ANEXO B

LISTA DE SÍMBOLOS

***B*:** Pendiente de la función de probabilidad del método de decisión probabilístico

***b*:** Interceptación de las rectas de regresión lineal

***D*:** Distancia interestomática o distancia que separa dos hileras de estomas

D_{media}**:** Valor medio de la distancia *D*, usado en las configuraciones iniciales regular, aleatoria y gaussiana

***d*:** Distancia intraestomática o distancia que separa dos estomas dentro de una misma hilera

***K*:** Constante de decisión

***k*:** Cantidad de 1s asociados al 0 más próximo respecto a la posición evaluada

K_{maximo}**:** Valor máximo de *K* para el que se simulan los resultados de un conjunto de simulaciones

K_{minimo}**:** Valor mínimo de *K* para el que se simulan los resultados de un conjunto de simulaciones

***M*:** Número de columnas de las matrices

***m*:** Pendiente de las rectas de regresión lineal

***N*:** Número de filas de las matrices

R^2**:** Coeficiente de determinación de las rectas de regresión lineal

σ**:** Desviación estándar

LISTA DE ABREVIATURAS

ABA: Ácido abscísico

CO_2: Dióxido de carbono

O_2: Oxígeno

e.g.: *exempli gratia*; por ejemplo

EPF1 y EPF2: Factores de Patrón Epidérmico

MAT: Media Anual de Temperatura

***P*:** *Pinus*

RESUMEN

Los estomas son estructuras microscópicas esenciales para el intercambio gaseoso y la transpiración en las plantas, desempeñando un papel crítico en la adaptación a estreses ambientales. Este estudio desarrolla un modelo matemático-computacional, implementado en Python, para simular el desarrollo y la distribución espacial de estomas en acículas de pino que dan lugar a la aparición de hileras. El modelo representa la superficie de la acícula como una matriz bidimensional, donde la colocación de estomas sigue reglas procedurales inspiradas en procesos biológicos, incluyendo mecanismos de inhibición local y un umbral de decisión (K) que actúa como *proxy* de señales genéticas, hormonales y ambientales.

Las simulaciones incorporan distribuciones iniciales (aleatoria, regular, gaussiana o saturada) y modos de decisión (binario o probabilístico con una función sigmoidea). Se analizan métricas clave —densidad estomática, distancia intraestomática (d), distancia interestomática (D) y número de hileras finales— en función de parámetros variables. Los resultados muestran patrones lineales emergentes, con la densidad estomática y el número de hileras siguiendo una relación de potencia con K, mientras que d permanece estable, alineándose con observaciones empíricas. Configuraciones flexibles replican comportamientos naturales como la no contigüidad y las interrupciones de hileras.

El modelo coincide cualitativamente con las referencias en la literatura sobre densidades y disposiciones en pinos, aunque simplificaciones (e.g. geometría plana) limitan la precisión cuantitativa. Las aplicaciones del modelo son múltiples, desde ser una base a partir de la cual desarrollar herramientas predictivas de resiliencia climática hasta permitir una posible integración de modelos más complejos de integración fotosintéticos o de conductancia, a pesar de las limitaciones que modelos como éste puedan tener, donde se incluyen la escalabilidad computacional y la omisión de dinámicas moleculares. Este enfoque que aúna la biotecnología con la matemática y computación demuestra cómo reglas locales simples generan patrones biológicos complejos, proporcionando una base para generalizaciones futuras a especies vegetales diversas.

ABSTRACT

Stomata are microscopic structures essential for gas exchange and transpiration in plants, playing a critical role in their adaptation to environmental stress. This study presents a mathematical–computational model, implemented in Python, to simulate the development and spatial distribution of stomata in pine needles, where their arrangement leads to the formation of linear rows. The model represents the needle surface as a two-dimensional matrix, with stomatal placement governed by procedural rules inspired by biological processes, including local inhibition mechanisms and a decision threshold (K) that serves as a proxy for genetic, hormonal, and environmental signals.

The simulations incorporate diverse initial distributions (random, regular, Gaussian, or saturated) and decision modes (binary or probabilistic using a sigmoidal function). Key metrics—stomatal density, intra-stomatal distance (d), inter-stomatal distance (D), and the final number of rows—are analyzed as functions of variable parameters. The results reveal emergent linear patterns, with stomatal density and row number following a power-law relationship with K, while d remains stable, consistent with empirical observations. Flexible configurations reproduce natural behaviors such as non-contiguity and row interruptions.

The model qualitatively agrees with literature reports on stomatal densities and arrangements in pines, although simplifications (e.g., planar geometry) limit quantitative accuracy. Potential applications range from serving as a basis for predictive tools in climate resilience to enabling integration with more complex photosynthetic or conductance models, despite inherent limitations such as computational scalability and omission of molecular dynamics. This interdisciplinary approach, combining biotechnology with mathematics and computation, demonstrates how simple local rules can generate complex biological patterns and provides a foundation for future generalizations to diverse plant species.

CAPÍTULO 1

INTRODUCCIÓN

1.1 Contexto biológico y relevancia de los estomas

Los estomas son estructuras microscópicas localizadas en la epidermis de hojas y otros órganos vegetales, esenciales para la regulación del intercambio gaseoso al permitir la entrada del dióxido de carbono (CO_2) necesario para la fotosíntesis y la salida de oxígeno (O_2) y vapor de agua durante la transpiración[1]. Estos estomas, compuestos por dos células oclusivas que rodean un poro central, presentan patrones de distribución específicos, en algunos casos aparentemente de forma aleatoria, pero en otros casos organizados en filas o bandas paralelas, como sucede en pinos (*Pinus spp.*)[2]. Se hipotetiza que estas diferentes disposiciones responden a dos factores: por un lado, a la presión espacial ejercida por células vecinas; y, por otro lado, a factores ambientales. De esta forma, la aparición de unos patrones u otros se debería en parte a una adaptación de las plantas a medios adversos para minimizar la pérdida de agua y optimizar la eficiencia fotosintética[3].

Por tanto, todo factor que influya en los estomas estará también estrechamente ligado a la importancia de ellos en procesos fisiológicos y de adaptación fundamentales en entornos cambiantes[3–6]. Serán especialmente críticos la densidad estomática, definida como el número de estomas por unidad de área, y, como se menciona más arriba, su distribución espacial. En coníferas, la densidad estomática varía dependiendo de la especie y de las condiciones ambientales, habiéndose observado que responde, por ejemplo, a cambios a corto y largo plazo en la concentración de CO_2[7]; mientras que la conductancia estomática se ha relacionado con la altitud y la diferencia de presión de vapor de agua entre el ambiente y la planta[8]. Es precisamente esta estrecha relación entre el ambiente y los estomas la que es de interés para muchas áreas de estudio, como por ejemplo la paleobotánica, al poder servir como indicadores del ambiente en tiempos pasados[9,10]. Además, aparte de responder a factores ambientales, también están regulados por mecanismos genéticos, incluyendo señales de inhibición que evitan la formación de estomas adyacentes, garantizando de esta forma una distribución óptima[11].

Por otro lado, la relevancia de los estomas trasciende la mera fisiología vegetal, ya que su funcionamiento y distribución impacta de forma directa sobre los ciclos globales de carbono y agua. Por ejemplo, los bosques de coníferas, que cubren vastas áreas en regiones templadas y boreales, actúan como importantes sumideros de carbono, secuestrando grandes cantidades de CO_2 a través de la fotosíntesis[2,7]. Además, la densidad y distribución de los estomas influyen directamente en la conductancia estomática (esto es, la capacidad de intercambiar CO_2 y O_2 con el medio) y, por ende, en la capacidad fotosintética de las acículas. Por otro lado, en el contexto del cambio climático, donde los pinos, al igual que otras muchas especies, enfrentan desafíos como el aumento de temperaturas y la disminución de la disponibilidad hídrica, comprender mecanismos básicos de fisiología vegetal es crucial para predecir la resiliencia de los bosques y su contribución al ecosistema. Recientes estudios han puesto sobre todo especial interés en la adaptación de la capacidad de absorber agua ante estreses hídricos, como se observa en *Pinus torreyana*[12] o en la regulación de la densidad estomática a temperaturas más altas[13], demostrando que el aumento de la densidad estomática en respuesta a aumentos de temperaturas puede incrementar la conductancia, pero también el riesgo de pérdida de agua, lo que requiere un equilibrio delicado en la regulación estomática por parte de la planta[13]. Este trabajo, al centrarse en modelar la distribución estomática en acículas, pretende profundizar y entender los diferentes mecanismos y comportamientos que definen su ecofisiología y adaptación.

1.2 Estado del arte en el estudio de la distribución estomática

El estudio de la distribución estomática se desarrolló en gran medida a lo largo del siglo pasado y ha experimentado un nuevo resurgir en estas últimas dos décadas, combinando enfoques experimentales y teóricos, teniendo como sujeto de estudio una gran diversidad de especies y géneros diferentes[3]. En plantas modelo como *Arabidopsis thaliana*, se han identificado mecanismos genéticos que regulan la formación de los estomas, como lo es la acción de los factores de patrón epidérmicos EPF1 y EPF2 a la hora de mediar la inhibición, evitando así la formación de estomas contiguos[1,14]. En otras especies, como es el cacao (*Theobroma cacao L.*), los estudios más recientes se centraron en analizar la coordinación en el desarrollo de los cloroplastos y el de los estomas cuando

las hojas están en una etapa de crecimiento rápido, revelando que existe una estrecha relación y coordinación que es esencial para el crecimiento y adaptación de la planta[6].

Por lo que corresponde a coníferas, especialmente en pinos (dónde nos vamos a centrar en este proyecto), los estudios se han enfocado principalmente en describir patrones espaciales debido a la idoneidad que supone trabajar con desarrollos foliares lineales, precisamente lo que ocurre en las acículas de ciertas especies de esta clase. Se observó que en estas especies predomina la distribución regular en filas, más o menos heterogéneas y densas, influenciadas por factores ambientales[3,15]. También existen otros estudios que buscaban analizar la densidad y distribución estomática en coníferas, proporcionando datos sobre su regulación en diferentes condiciones y ante distintos estreses. Uno de estos proyectos fue el conducido por Brodiribb y colaboradores, donde demostraron que la resistencia a sequía en coníferas sigue dos vías evolutivas, una dependiente de altos niveles de ácido abscísico (ABA) para cerrar estomas (el caso de *Pinaceae* y *Aruacariaceae*) y otra línea basada en la desecación foliar (*Cupressaceae*)[16]. Sin embargo, aún siguen persistiendo lagunas en la integración de factores genéticos, ambientales y espaciales en modelos que expliquen los patrones lineales estomáticos observados en acículas. Este proyecto pretende abordar esta brecha mediante un modelo matemático-computacional que simula la disposición estomática.

Cabe entonces preguntarse por el estado del arte en cuanto al desarrollo de modelos matemático-computacionales en este campo. A lo largo de la última década, éstos han avanzado significativamente en el estudio del desarrollo vegetal, abordando principalmente la morfogénesis y la fisiología foliar. Por ejemplo, Prusinkiewicz y colaboradores utilizaron ecuaciones de reacción-difusión de Turing para modelar patrones estomáticos emergentes mediados por auxina[17]. Otro estudio que merece la pena destacar es el de Buckley, donde desarrolló un modelo de optimalidad que vincula la densidad estomática con variables ambientales, extendiendo el modelo Ball-Woodrow-Berry para predecir respuestas al cambio climático[4]. Por otro lado, Dow y colaboradores propusieron un modelo que relaciona la conductancia estomática máxima con la densidad y tamaño de estomas, integrando factores genéticos y fisiológicos[18]. Por último, Gaur y Drewry aplicaron herramientas relacionadas con el aprendizaje automático para predecir la conductancia estomática en diversos tipos de plantas[19]. Sin embargo, si bien estos enfoques han supuesto un gran avance en la comprensión de procesos morfológicos y

fisiológicos y han sentado precedente a la hora de usar herramientas matemáticas y computacionales para enfrentarse a problemas en esta área de estudio, su foco se dirige a plantas dicotiledóneas y a aspectos funcionales. Nos volvemos a encontrar entonces en una situación donde los modelos que exploran los mecanismos y causas subyacentes a los patrones estomáticos lineales en acículas, como es el caso de los pinos, son escasos. Es en este terreno poco explorado donde pretendemos con este proyecto dar algo más de luz.

1.3 Justificación y objetivos del proyecto

La distribución estomática es un proceso biológico complejo que influye directamente en la capacidad de la planta de adaptarse a entornos[3–6]. Sin embargo, los modelos computacionales existentes no capturan completamente las dinámicas espaciales y temporales de este proceso en especies con patrones lineales, como los pinos. Además, la falta de modelos que integren restricciones biológicas limita la capacidad de predecir cómo estas especies responderán a cambios ambientales, como los asociados al cambio climático. Este proyecto propone un modelo matemático-computacional implementado en Python que simula el desarrollo y distribución de estomas en estas especies, permitiendo evaluar el impacto de parámetros clave en los patrones estomáticos.

El modelo desarrollado se justifica por su potencial para generar diferentes patrones de distribución estomática existentes, basándose en un determinado número de parámetros y variables y regido casi en su totalidad por reglas locales de reparto espacial, permitiendo obtener resultados globales similares a los observados en la naturaleza. Al llevar a cabo simulaciones empleando diferentes distribuciones iniciales y de probabilidad y distintos valores de los parámetros, el modelo permite explorar cómo los patrones estomáticos emergen de reglas locales, ofreciendo una base sobre la que sustentar modelos futuros para comparar con datos empíricos, optimizar modelos de fotosíntesis, eficiencia hídrica o llevar a cabo estudios predictivos acerca de la ecofisiología y el efecto que pueden tener condiciones ambientales cambiantes sobre la planta.

Los objetivos que motivan este trabajo y que pretendemos alcanzar a lo largo de su desarrollo serán, por tanto:

- **Objetivo general:** desarrollar un modelo matemático-computacional para simular el desarrollo y distribución de estomas en acículas de pino.
- **Objetivos específicos:**
    - Implementar un algoritmo que genere patrones estomáticos a partir de diferentes distribuciones iniciales (aleatoria, regular, gaussiana y saturada) para la primera fila de la matriz que representa la hoja.
    - Analizar métricas clave, como la densidad estomática, distancia intraestomática (d), distancia interestomática (D) y el número de hileras finales para caracterizar su efecto sobre los patrones espaciales y su relación con la constante umbral de decisión (K).
    - Generar gráficos que muestren la relación entre los parámetros del modelo y las métricas, facilitando la visualización y validación de resultados y la comparación con datos empíricos de distribución estomática.

CAPÍTULO 2

MATERIALES Y MÉTODOS

2.1 Datos empíricos de referencia

Los estomas son estructuras clave en la regulación del intercambio gaseoso y la transpiración en plantas, y en las coníferas, particularmente en pinos (*Pinus spp.*), su distribución y densidad son fundamentales para la adaptación a entornos variables. Las acículas de pinos, que son hojas con una morfología cilíndrica o semicilíndrica, presentan características morfológicas únicas que influyen en la disposición de los estomas. Estas hojas, agrupadas típicamente en fascículos de dos a cinco acículas según la especie (e.g., dos en *Pinus sylvestris*, tres en *Pinus torreyana*), tienen una superficie recubierta por una cutícula cerosa y estomas organizados en filas longitudinales, lo que se hipotetiza sirve para maximizar la captura de CO_2 y minimiza la pérdida de agua[3,7,15]. La disposición lineal de los estomas en acículas refleja una adaptación a ambientes con limitaciones hídricas o lumínicas, como los ecosistemas mediterráneos o de alta altitud[2,16].

La densidad estomática en pinos varía ampliamente según la especie y las condiciones ambientales. Estudios en *Pinus sylvestris* a lo largo de un transecto europeo muestran densidades estomáticas de 78 a 169 estomas/mm², con una relación positiva con la temperatura media anual (MAT), aumentando aproximadamente 4 estomas/mm² por cada 1°C de incremento en la temperatura[13]. En *Pinus torreyana*, una especie adaptada a climas mediterráneos con niebla, la densidad estomática también se encuentra en rangos similares, y las acículas presentan propiedades hidrofílicas que permiten la absorción directa de agua foliar, contribuyendo al balance hídrico durante los meses secos[12]. Algunos estudios realizados sobre diferentes especies del género *Pinus*, han demostrado que las acículas presentan estomas dispuestos en hileras en las caras adaxial (superior o haz) y abaxial (inferior o envés), con un número de hileras que oscila dependiendo de la especie, como se observa en *P. arizonica*, *P. cooperi*, *P. durangensis*, *P. herrerae*, *P. leiophylla* y *P. teocote*[15]; mientras que otros estudios observaron que existe una estrecha relación entre la densidad estomática y el número de hileras con condiciones ambientales como la altitud[20], la temperatura[13] o la disponibilidad de CO_2[21].

Por otro lado, cabe destacar que el desarrollo estomático en *Pinus* es típicamente mesoperígeno, de forma que se da al menos una división asimétrica en la línea celular que lleva a la formación de las células oclusivas y las anexas, lo que influye en el espaciado regular de ellos a lo largo de la acícula[3]. Estas distancias reflejan la acción de mecanismos de inhibición lateral previamente mencionados, como los mediados por los péptidos EPF1 y EPF2, que evitan la formación de estomas contiguos para optimizar el intercambio gaseoso y el reparto del espacio foliar[1,11].

El desarrollo temprano de los estomas en acículas, observado en la fase B2 y B3 tras la ruptura de la dormancia de los brotes, también proporciona información clave para el modelo. En especies como *Pinus sylvestris*, *P. mugo* y *P. nigra*, los estomas comienzan a diferenciarse en la fase B2 tardía, apareciendo primero cerca de la punta de la acícula y organizándose en filas longitudinales, un patrón que se consolida en la fase B3[22]. Estas observaciones sugieren que los patrones estomáticos emergen de procesos locales de diferenciación celular, influenciados por señales genéticas y ambientales, lo que inspira la incorporación de reglas locales en el modelo computacional. Aunque este trabajo no utiliza datos empíricos directos, los rangos de densidad estomática, la disposición lineal de los estomas en acículas de pinos y la oscilación de los números de hileras dependiente de la especie y de condiciones ambientales, descritos en la literatura[3,12,13,15,16,22], sirvieron como base para diseñar un modelo que replique cualitativa estos patrones biológicos.

2.2 Diseño del modelo matemático-computacional, implementación y simulación

Teniendo en cuenta todo este conocimiento previo acerca del funcionamiento del reparto espacial y de la disposición estomática expuestos hasta ahora, es momento de desarrollar el modelo matemático-computacional y cumplir con los objetivos generales y específicos declarados previamente. El modelo desarrollado representa la superficie foliar como una matriz bidimensional donde los estomas de distribuyen de forma procedural según reglas locales inspiradas en procesos biológicos. Implementado en Python, el modelo permite generar patrones que replican cualitativamente las disposiciones lineales observadas en la naturaleza, surgiendo de forma emergente (sin necesidad de ser forzado por el usuario) comportamientos naturales como la inhibición lateral, la regla de una

célula de separación (*one-cell spacing rule*)[3] o la aparición de hileras[15]. A continuación, se describe la lógica del modelo, las asunciones adoptadas, los parámetros y variables que lo definen, las libertades otorgadas al usuario para la obtención de resultados y el procedimiento para su uso.

La matriz que representa la acícula es una matriz rectangular de dimensiones $N \times M$, donde N corresponde al número de filas y, por tanto, a la longitud de la acícula; y M al número de columnas o al ancho de la acícula. La matriz se irá generando proceduralmete fila por fila, simulando el desarrollo longitudinal de la acícula, y cada celda de la matriz, que no tiene por qué representar únicamente a una sola célula, sino que es más correctamente una parcela espacial de la epidermis foliar, puede tomar uno de dos valores: *0* para la presencia de complejo estomático y *1* para la ausencia de complejo estomático

La lógica del modelo y construcción de la matriz se fundamenta en tres etapas principales:

1. **Inicialización de la primera fila:** la primera fila de la matriz representa el punto de partida del desarrollo estomático y corresponde con el ápice de la acícula. Se genera según una distribución inicial escogida por el usuario, de la cual daremos más detalles en este mismo capítulo. Esta fila determina las columnas donde podrán aparecer estomas en las filas subsiguientes, de forma que esta dictará en gran medida la distribución estomática del modelo.
2. **Desarrollo de filas posteriores:** a partir de la segunda fila la colocación de estomas en las columnas permitidas se rige por una regla de decisión que considera la cantidad de celdas no estomáticas (1s) asociadas al estoma más cercano en filas previas, comparada con un umbral de decisión (K_{ite}). El modelo a su vez incorpora dos modos de evaluación a la hora de decidir si colocar un estoma o no: el modo determinista (o *todo o nada*) y el modo probabilístico. Ambos se explicarán detalladamente después.
3. **Evaluación y visualización:** una vez generada la matriz, se calculan métricas clave como la densidad estomática, la distancia intraestomática (d), la distancia interestomática (D) y el número de hileras finales. Estas métricas

permiten analizar los patrones emergentes y su dependencia de los parámetros del modelo.

El modelo a su vez se basa en varias hipótesis que permiten capturar las características esenciales de la distribución estomática, procurando mantener un equilibrio entre realismo biológico y viabilidad computacional:

- **Geometría simplificada:** la acícula se representa como una matriz bidimensional, asumiendo una superficie plana en lugar de la morfología cilíndrica o semicilíndrica real. Esta simplificación permite modelar las hileras estomáticas como columnas, lo que es coherente con la disposición observada. Además, cada celda no corresponde a una única célula, sino que es una representación del reparto espacial, de forma que cada 1 representa una área determinada del tejido epidérmico y del mesófilo inferior que depende en cierta manera del intercambio gaseoso del estoma más cercano.
- **Diferenciación simplificada:** la diferenciación y aparición de estomas en la acícula se modela mediante la constante de decisión K, que es una representación espacial o geométrica de todos los mecanismos de activación o inhibición genéticos, hormonales y ambientales que influyen o participan en la señalización y diferenciación de las células precursoras en complejos estomáticos. Por tanto, actúa como una caja negra que registra cuándo debe aparecer un nuevo estoma para hacer frente a las exigencias metabólicas y fisiológicas del tejido circundante, sin incorporar ni tener en cuenta explícitamente mecanismos biológicos o de dinámica celular o molecular.
- **Dependencia de la distribución inicial:** la primera fila de la matriz se genera según distribuciones predefinidas, asumiendo que las hileras iniciales determinan las posiciones de estomas futuros. Esto refleja la organización longitudinal de las acículas, y, aunque puede ser demasiado determinista, se le puede hacer frente a esto último usando la distribución sigmoidea de probabilidad a la hora de evaluar posiciones.

La conjunción de todas estas asunciones permite generar patrones estomáticos que se acercan a las observadas en la naturaleza y replicar comportamientos ciertamente complejos usando reglas de distribución espacial sencillas.

En cuanto a los diversos parámetros y variables, el modelo emplea un conjunto configurable de ellos que permiten al usuario personalizar las simulaciones y explorar diferentes escenarios. A continuación, se describen detalladamente todos ellos:

- **Dimensiones de la matriz:** cabe destacar que, si bien unos valores mayores o menores permiten obtener matrices de diferentes tamaños, los resultados obtenidos y el comportamiento general no varía.
  - ***N*:** número de filas de la matriz. Representa la longitud de la acícula
  - ***M*:** número de columnas de la matriz. Representa el ancho de la acícula.
- **Distribución inicial:** define cómo se colocan los estomas en la primera fila.
  - **Aleatoria (1):** los estomas se distribuyen al azar, respetando una distancia interestomática promedio (D_{media}).
  - **Regular (2):** los estomas es colocan a intervalos fijos respetando la D_{media}.
  - **Gaussiana (3):** los estomas se distribuyen según una distribución normal, centrada en D_{media} y con una determinada desviación estándar.
  - **Especial (4):** todas las posiciones en la primera fila son estomas, simulando una distribución saturada. Si bien de comienzo esta es la menos realista biológicamente hablando, es la que va a permitir mayor flexibilidad y unos resultados más correctos al permitir que aparezcan estomas en cualquier posición de la matriz.
- **Distancia interestomática (*D*):** representa la distancia en número de celdas entre estomas de dos hileras diferentes. Un valor medio, D_{media}, debe ser dado por el usuario en caso de usar distribuciones iniciales que no sean la saturada para poder general la primera fila.
- **Distancia intraestomática (*d*):** representa la distancia en número de celdas entre estomas de una misma hilera.
- **Desviación estándar (σ):** usada sólo en la distribución inicial gaussiana. Define la variabilidad en la colocación de estomas en la primera fila.
- **Constante de decisión (*K*):** representa el número máximo de celdas no estomáticas que un estoma puede "controlar" o dar suministro. Es un reflejo de todos los mecanismos que permiten activar o inhibir la diferenciación en complejos estomáticos. En el modelo, un *K* bajo favorece una mayor densidad estomática al implicar que las condiciones ambientales son estresantes o

limitantes y provocan que la planta requiera de un mayor número de estomas para hacerles frente.

- **Modo de decisión:** determina cómo se evalúa la colocación de estomas. Como ya se adelantó anteriormente, existen dos modos de decisión:
  - **Modo determinista o *todo o nada:*** la decisión de colocación de un estoma es estricta. Solo se coloca un 0 si el número de celdas asociadas al estoma más cercano (*k*) supera el umbral *K*. Es un modo más determinista y estricto.
  - **Modo probabilístico:** la decisión de colocar un 0 sigue una distribución de probabilidad sigmoidea dependiente de una pendiente y de los valores de *K* y *k.* Añade estocasticidad y flexibilidad al modelo y permite explorar patrones más realistas y variados, reflejando la variabilidad en la naturaleza. La función sigmoidea es la siguiente:

$$P(0) = \frac{1}{1 + e^{B(1-\frac{k}{K})}}$$

    Donde *P(0)* es la distribución de probabilidad de colocar un 0 en una determinada posición; *B* es la pendiente de la función; *k* es la cantidad de 1s asociados al 0 más cercano; y *K* el umbral o límite de decisión.

  En la Figura 2.1 se muestra la diferencia entre ambos modos de decisión, así como el efecto de aumentar o disminuir la pendiente en la función de probabilidad. Nótese que, en el límite, ambas funciones parecen converger, aunque en el caso de la función de probabilidad, cuando $k = K$ la probabilidad de situar un 0 es de 0.5 y no de 1, que es lo que ocurre en el modo determinista. Como ya hablaremos en el Capítulo 4 de Discusión, una pendiente baja vuelve el modelo más flexible, pero puede aumentar demasiado el ruido; mientras que una pendiente elevada lo vuelve demasiado restrictivo.

- **Otras variables:**
  - **Restricción de no contigüidad:** el usuario puede indicar si desea que la *one-cell spacing rule* se fuerce o no, impidiendo que no puedan aparecer dos complejos estomáticos juntos. Si no se fuerza, el modelo, bajo las condiciones más flexibles y realistas, acaba replicando este comportamiento de forma natural.

- **Restricción lateral:** de igual modo, el usuario puede indicar si dese que los complejos estomáticos no aparezcan en los extremos laterales de la acícula. De nuevo, esta característica surge de forma autónoma en el propio modelo cuando los parámetros son más flexibles.
- **Número de simulaciones:** el usuario puede escoger la cantidad de simulaciones a realizar para un mayor valor estadístico y de validación de los resultados obtenidos.
- **Generación de imágenes:** es decisión del usuario si guardar las matrices asociadas como imágenes para ver el resultado final o si prefiere ahorrar ese tiempo computacional en caso de no ser necesario.

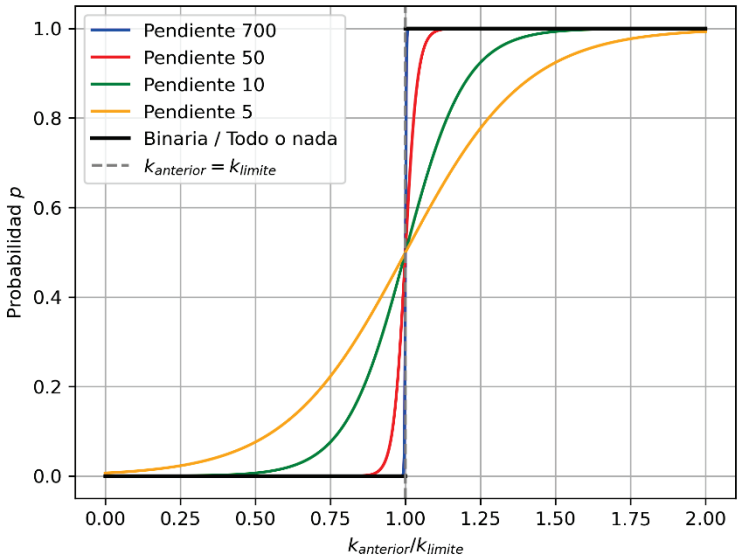


Figura 2.1 – Gráfica con las probabilidades de situar un 0 dependiendo de la relación k y K para las funciones determinista (negro) y de probabilidad con distintas pendientes (azul, rojo, verde y naranja).

El modelo está implementado en Python y se divide en tres *scripts* principales para compartimentar de forma clara y cómoda las distintas funciones y tareas realizadas a lo largo de la simulación:

- **Código 1 – Obtención de posiciones:** genera la matriz y guarda las posiciones de los estomas en un archivo de texto *Posiciones.txt*. Este primer *script* incluye funciones para inicializar la primera fila, desarrollar las filas posteriores según las reglas establecidas previamente y almacenar toda la información necesaria para obtener las métricas y generar las gráficas. El usuario introduce los parámetros a través de una interfaz interactiva especificando los valores de los

parámetros definidos anteriormente. La simulación entonces comenzará y se generarán las matrices desde un valor $K_{mínimo}$ hasta un valor $K_{máximo}$ cada cierta cantidad de valores de K y para un número determinado de simulaciones por cada valor de K (e.g. de $K_{ite} = 5$ hasta $K_{ite} = 95$, aumentando el valor de K en 5 unidades y haciendo 10 simulaciones para cada valor). Nótese que estos valores son escogidos por el usuario a su conveniencia. Además, este primer código también genera las imágenes de las matrices simuladas en caso de que el usuario lo desee.

- **Código 2 – Métricas:** lee el archivo *Posiciones.txt* y calcula las métricas clave (esto es, densidad estomática, d, D y número de hileras finales) para un número de filas finales establecidas por el usuario. Los resultados se guardan en *Metricas.txt*, incluyendo medias y desviaciones por matriz y por cada valor de K_{ite}.
- **Código 3 –** Gráficas: procesa *Metricas.txt* para generar gráficos que muestran la relación entre K y las métricas, en escalas lineal, logarítmica y semilogarítmica. Crea también un archivo *Info.txt* con toda la información relacionada con la simulación y los parámetros usados a modo de registro.

Para usar el modelo por tanto el usuario ejecuta los *scripts* secuencialmente: primero el código 1, especificando los parámetros; luego el código 2, a partir del archivo generado por el primer código, especificando únicamente un número final de filas para obtener las métricas; y finalmente el código 3, donde obtendrá los gráficos para visualizar las métricas en función de K. En consecuencia, el modelo desarrollado es una herramienta versátil, que da libertad y flexibilidad de uso al usuario a la hora de escoger los distintos parámetros para llevar a cabo la simulación. Su diseño modular ofrece al usuario la capacidad de explorar una amplia gama de escenarios y distribuciones, desde las condiciones más restrictivas y deterministas hasta casos más flexibles y con menor o mayor estocasticidad, facilitando de esta manera el análisis de cómo las reglas locales y la modificación de los parámetros afectan a los patrones emergentes. De esta forma, el modelo es capaz de capturar las características esenciales de los patrones y comportamientos observados en la naturaleza, proporcionando una base para comparaciones con datos empíricos y futuras mejoras en la modelización de procesos complejos.

CAPÍTULO 3

RESULTADOS

3.1 Visualización de resultados

Habiendo asentado las bases del modelo, toca realizar las simulaciones para obtener los resultados de interés. Como ya avanzamos en apartados anteriores, la acícula se describe mediante matrices de tamaño N×M coloreadas para facilitar interpretación. En ellas, los complejos estomáticos (0s), se representan como círculos negros sobre el plano y las áreas epidérmicas (1s) asociadas a cada uno de ellos tendrá un color determinado, asociado a cada complejo estomático, usando una paleta de 4 colores (amarillo, azul, verde y rojo). De esta manera, se puede ver de forma clara qué 1s están bajo la influencia de qué 0s, como se puede observar en la Figura 3.1.

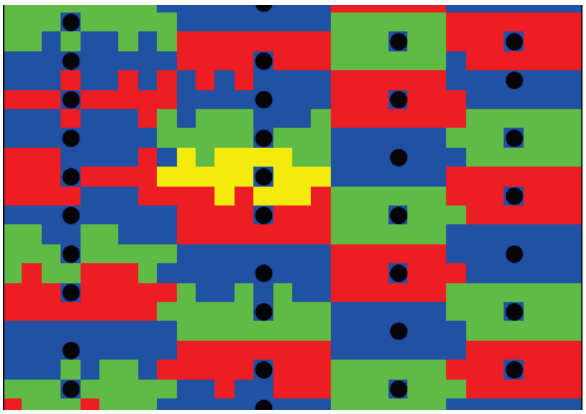


Figura 3.1 - Sección central de una matriz generada. Los círculos negros representan los complejos estomáticos (0s) y los colores las áreas bajo su influencia (1s).

En un primer lugar, partimos de los casos más restrictivos, donde apenas se le da flexibilidad al modelo y la estocasticidad es mínima. Estos casos son menos realistas e interesantes, al forzar a que aparezcan comportamientos naturales como la inhibición lateral, la no contigüidad de estomas o la propia generación de hileras, pero igualmente son de ayuda para comenzar a intuir cual es la relación entre la densidad estomática y los patrones de disposición con la constante *K* de decisión. De esta forma, se presentan en la Figura 3.2 diferentes matrices generadas a partir de unos mismos parámetros iniciales, con la única diferencia siendo la distribución inicial escogida.

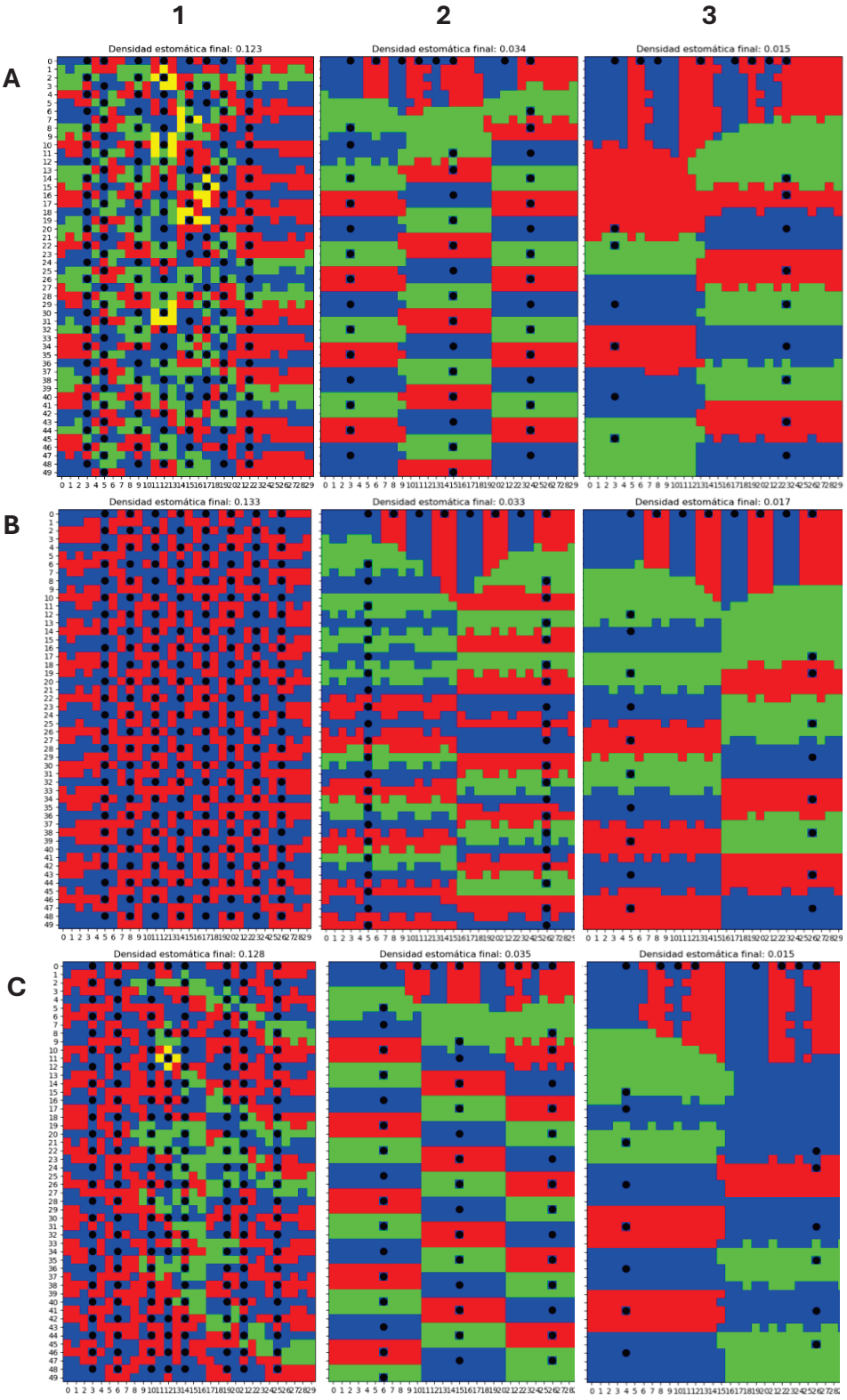


Figura 3.2 – Matrices 50x30 resultado de una simulación con valores $D_{media} = 3$ y valores de $K = 5$ (1), 50 (2) y 95 (3). Se fuerza la regla de no contigüidad y de inhibición lateral y el modo de decisión es determinista. Las distribuciones iniciales son (A) aleatoria, (B) regular y (C) guassiana con una desviación de 1. Sobre cada una se encuentra su densidad estomática.

Por otro lado, si bien estas matrices proporcionan algo de información sobre el comportamiento general y la posible relación entre la constante K y la densidad estomática (por ejemplo, observando cómo, independientemente de la distribución inicial usada, los valores de densidad estomática y el comportamiento es prácticamente idéntico para unos mismos valores de K), es oportuno analizar cómo actúa el modelo ante unas condiciones mucho más flexibles y realistas para comprobar, entre otras cosas, si la generación de hileras o la no contigüidad de estomas aparecen de forma natural sin necesidad de ser forzados. Para ello, se realizaron simulaciones usando la distribución inicial saturada (donde la primera fila estará llena de 0s) y desactivando la inhibición lateral y por contigüidad, de forma que en la práctica todas las posiciones de la matriz serán susceptibles a la aparición de complejos estomáticos. A su vez, se hicieron dos tandas de simulaciones distintas: una manteniendo el sistema de decisión determinista o *todo o nada* (de forma que únicamente se situarán los 0s si y sólo si el número de 1s asociados al 0 anterior supera K); y una segunda tanda para estudiar el efecto de añadir más estocasticidad al proceso al usar la función de probabilidad dependiente del ratio $\frac{k}{K}$ cuya gráfica ya se pudo observar en la Figura 2.1.

Comenzando con la Figura 3.3, se muestra el inicio de dos matrices 300x40 usando las condiciones más flexibles pero el modo de decisión determinista. Se representan giradas 90º para facilitar su visualización y encaje en la página, de modo que las filas iniciales se encuentran ahora a la izquierda de la imagen. Como se puede observar a simple vista, es llamativo cómo el patrón en las primeras filas se vuelve totalmente regular y esto se debe a que, como el modo de decisión es determinista y en todas las posiciones pueden aparecer complejos estomáticos, para unas filas iguales a $K + 1$ y $\frac{3K}{2}$, todas las posiciones cumplen en sincronía la condición de decisión para colocar un 0. Siendo esto así, el modelo entra en lo que hemos denominado un *estado transitorio* donde se mantiene un una gran regularidad y trasladarlo a la naturaleza parece carecer de sentido. Sin embargo, lo que es especialmente interesante es cómo este estado se rompe debido a los desempates de repartición espacial entre estomas cercanos y se acaba alcanzando de forma autónoma y natural un segundo estado que hemos nombrado *estado estabilizado*, a partir del cual, y hasta el final de la matriz, un determinado número de hileras se acaba manteniendo. Todas las métricas que obtiene el Código 2, explicado en

el Capítulo 2.2, tienen en cuenta la existencia de este estado transitorio, de forma que los valores de densidad estomática, número de hileras y distancias inter e intraestomáticas calculadas únicamente se obtienen de las filas finales y, por tanto, cuando el modelo ya ha alcanzado el estado estabilizado. Es también muy interesante observar cómo en ambas matrices de la Figura 3.3, los valores de densidad estomática para K = 10 (A; 0.106) y K = 50 (B; 0.027), son muy parecidos a los observados en los modelos restrictivos de la Figura 3.2 (0.123 para K = 5; 0.016 para K = 50).

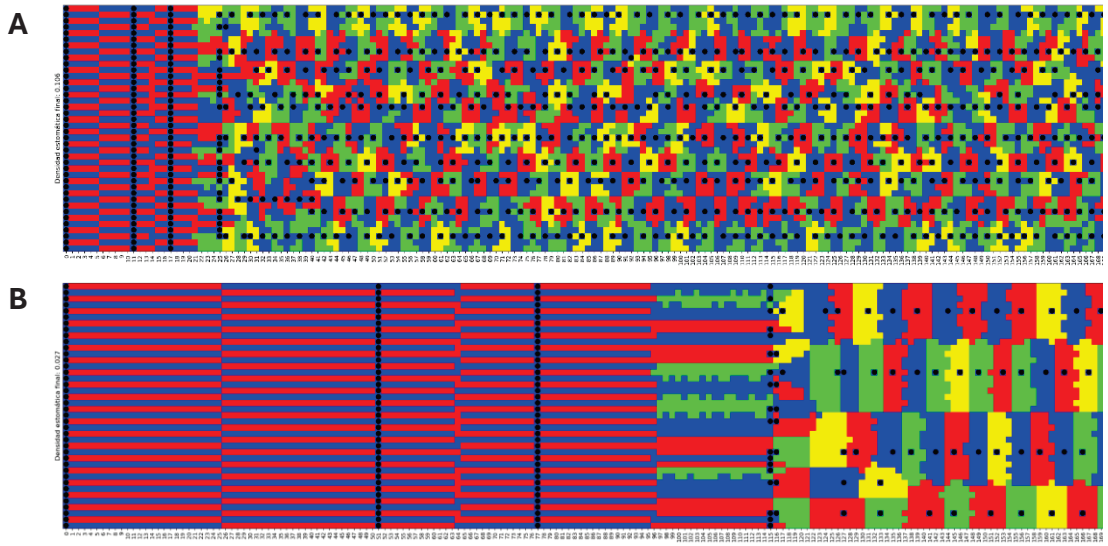


Figura 3.3 – Sección inicial de matrices 300x40 resultado de una simulación sin regla de no contigüidad ni inhibición lateral y usando el modo determinista de decisión. Las matrices están giradas 90º, de forma que las primeras filas se encuentran a mano izquierda, donde se encuentra además la densidad estomática final de cada una. (A) corresponde a un valor K = 10 y (B) a un valor K = 50.

Aunque los resultados comienzan ya a ser mucho más interesantes, idealmente el modelo debería carecer, en la medida de lo posible, de este estado transitorio o, al menos, poder reducirlo a su mínima expresión. Además, otro comportamiento que preferiblemente el modelo debería ser capaz de replicar es la aparición de hileras nuevas o de interrumpir las ya presentes, como se observa en la naturaleza, donde, pese a ser un evento ciertamente raro, ocurre con la suficiente frecuencia como para ser de interés. Por tanto, la idea fue aumentar la estocasticidad del modelo y, como ya se avanzó previamente, usar el modo de decisión probabilístico. Como cabe deducir por su propia definición, la principal repercusión que tendrá esto sobre los resultados vistos hasta ahora es que no siempre que una posición donde se cumpla la condición para situar un 0 se va a poner ese 0; y viceversa, esto es, no siempre que esa condición no se cumpla se va a

poner un 1. De esta manera, se añade un carácter estocástico muy importante que, como ya se indicó, va a tener un mayor o menor efecto según sea la pendiente de la distribución de probabilidad, como se pudo apreciar en la Figura 2.1. Intuitivamente, esto lleva a pensar que, entre otras cosas, ese estado transitorio no tiene por qué producirse ya que ahora no todas las posiciones van a cumplir la condición umbral de forma síncrona y, además, abre la posibilidad de que aparezcan 0s fuera de las hileras finales más estables.

Para ello, se realizó otra tanda de simulaciones, esta vez usando la función probabilística y viendo cómo afecta en las matrices finales una mayor o menor pendiente. Los resultados se muestran en la Figura 3.4, de nuevo volteados para facilitar el encuadre.

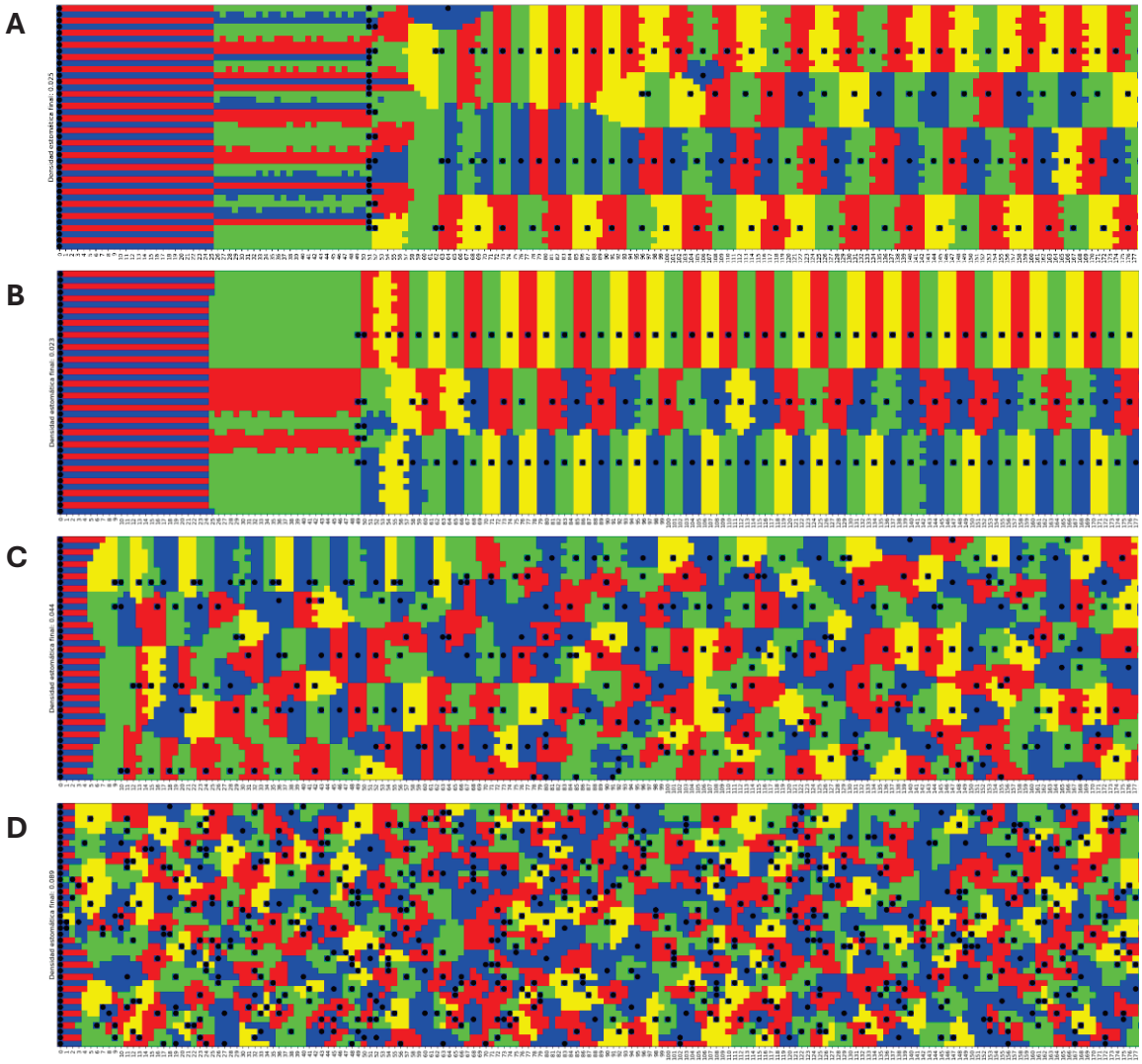


Figura 3.4 – Sección inicial de matrices 300x40 resultado de una simulación sin regla de no contigüidad ni inhibición lateral y usando el modo probabilístico de decisión. Las matrices están giradas 90º, de forma que las primeras filas se encuentran a mano izquierda, donde se encuentra además la densidad estomática final de cada una. El valor de K = 50 y la pendiente de probabilidad varía entre (A) 700, (B) 50, (C) 10 y (D) 3.

Como se puede observar en la figura y como era de esperar, el estado transitorio ha sido modificado o incluso ha desaparecido. En la Figura 3.4 – A, que corresponde a una función de probabilidad de pendiente igual a 700, en la fila $K + 1$ vuelve a aparecer una fila saturada, similar a como ocurría usando la decisión determinista, con la salvedad de que, en este caso, los 0s se encuentran en la mitad de las posiciones y no en todas ellas (como ocurría en el modo determinista y ya se mencionó en el Capítulo 2.2). Esto provoca además que desaparezca esa segunda fila saturada que veíamos en modelos anteriores. Además, podemos observar cómo las hileras se estabilizan de forma natural y los estomas se espacian de forma adecuada y esta vez, gracias a la estocasticidad añadida, pueden aparecer hileras nuevas como ocurre en esa misma imagen. Por otro lado, como ya adelantamos anteriormente, el hecho de disminuir la pendiente de la función hace aún más flexible el modelo bajo el riesgo de aumentar el ruido, de forma que las hileras dejan de ser tan claramente visibles (Figura 3.4 – C) o incluso desaparecer esa identidad lineal y pasar a ser más caótica (Figura 3.4 – D) pero con la ventaja de eliminar completamente el estado transitorio.

3.2 Análisis del modelo ante la variación de parámetros

Los resultados expuestos en el subapartado anterior ya eran más fieles a la realidad y permiten por tanto llevar a cabo análisis más profundos para estudiar cuál y cómo es exactamente esa relación que parecen guardar la constante K con la densidad estomática y su disposición en el espacio. Hasta ahora, todos los resultados obtenidos eran producto del Código 1 y, por tanto, sólo ofrecían información acerca de las posiciones de los estomas. Para realizar un estudio acerca de esta posible dependencia, usaremos el modelo flexible, dándole a la pendiente de la función de probabilidad los valores de 700, 50 y 10, ya que, como se mostró en la Figura 3.4, pendientes más bajas aumentan demasiado el ruido.

Para la obtención de los resultados en este subapartado haremos uso de los Códigos 2 y 3, que como ya expusimos, obtienen datos a partir de las posiciones y realizan gráficas para poder visualizarlos. Los gráficos que nos interesa es ver cómo evolucionan la densidad estomática, el número de columnas finales y las distancias interestomática D e intraestomática d en relación con la constante de decisión K. Para ello realizamos un

conjunto de simulaciones que generan matrices desde un K_{minimo} hasta un K_{maximo} cada cierto número determinado de K y haciendo un número de simulaciones por cada valor de K; todo ello para unos valores de la pendiente de la función de probabilidad de 700, 50 y 10. Los resultados se mostrarán a su vez en su representación lineal y logarítmica en caso de ser relevante. Todo ello está recogido en las Figuras 3.5, 3.6 y 3.7

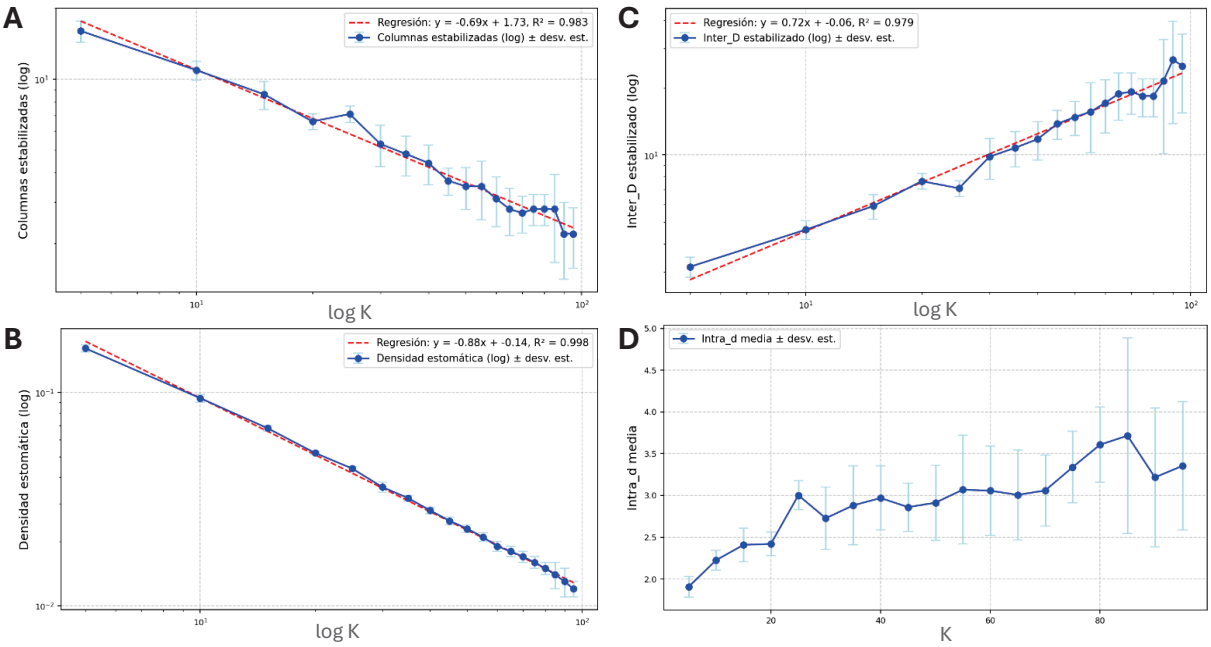


Figura 3.5 – Gráficas de relación entre el número de columnas finales (A), densidad estomática (B), distancia interestomática *D* (C) y distancia intraestomática *d* (D) respecto a *K* para una pendiente de función de probabilidad = 700. Para las imágenes (A), (B) y (C) la gráfica es logarítmica. Para la imagen (D) la gráfica es lineal.

Como se puede observar, todos los parámetros varían considerablemente según aumentan los valores de *K*, algo que concuerda con lo observado en la naturaleza ya que, independientemente de la especie o de la condición ambiental, pese a que pudiese haber más o menos columnas de estomas, la distancia entre ellos dentro de una misma columna es bastante constante. Más concretamente, en los casos con pendientes 700 (Figura 3.5) y 50 (Figura 3.6), se puede ver cómo esa distancia se encuentra entre 2 y 4 y parece ser la única variable que no guarda una relación de potencia con el valor de *K*. Algo interesante también es cómo esa relación sí parece darse cuando la pendiente es baja (Figura 3.7) y el ruido es mayor y la naturaleza lineal tiende a perderse. En cuanto al resto de variables, parecen depender de forma clara de manera potencial respecto a *K*, lo cual, aunque profundizaremos más en ello en el próximo Capítulo 4 de Discusión, nos llevaría a pensar que la densidad estomática y número de hileras de estomas en estas

especies depende de manera potencial de las condiciones ambientales siguiendo una ecuación de tipo $Y = 10^b \cdot K^m$; siendo b la interceptación de la recta de regresión y m su pendiente.

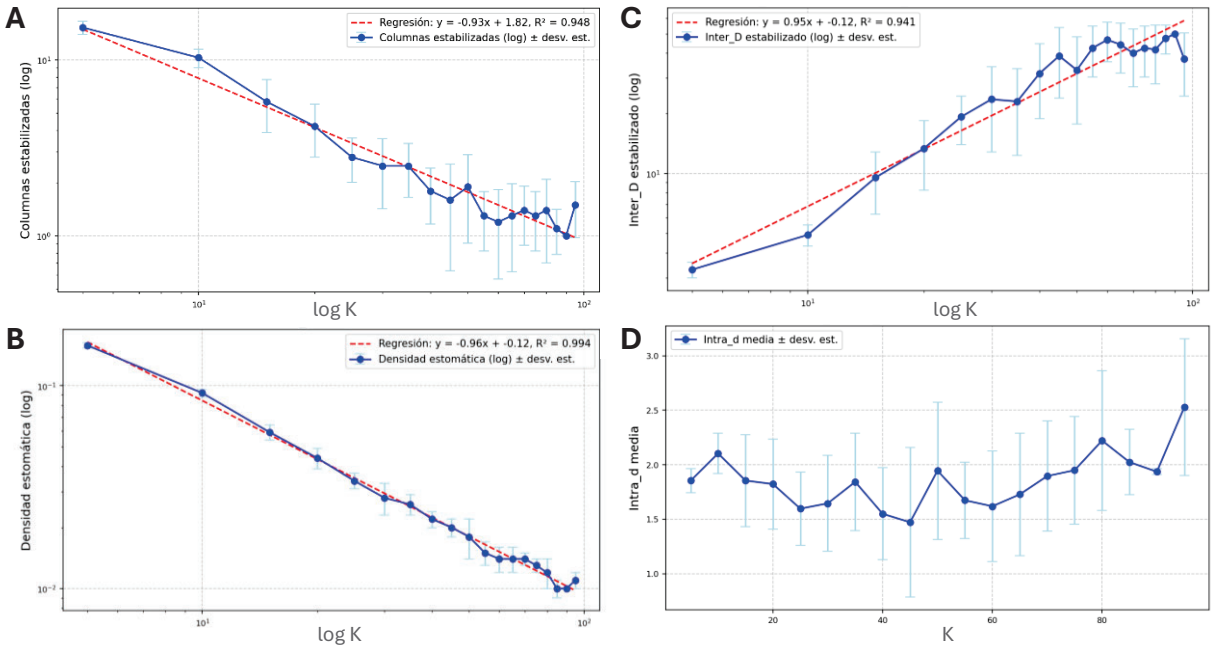


Figura 3.6 – Gráficas de relación entre el número de columnas finales (A), densidad estomática (B), distancia interestomática *D* (C) y distancia intraestomática *d* (D) respecto a *K* para una pendiente de función de probabilidad = 50. Para las imágenes (A), (B) y (C) la gráfica es logarítmica. Para la imagen (D) la gráfica es lineal.

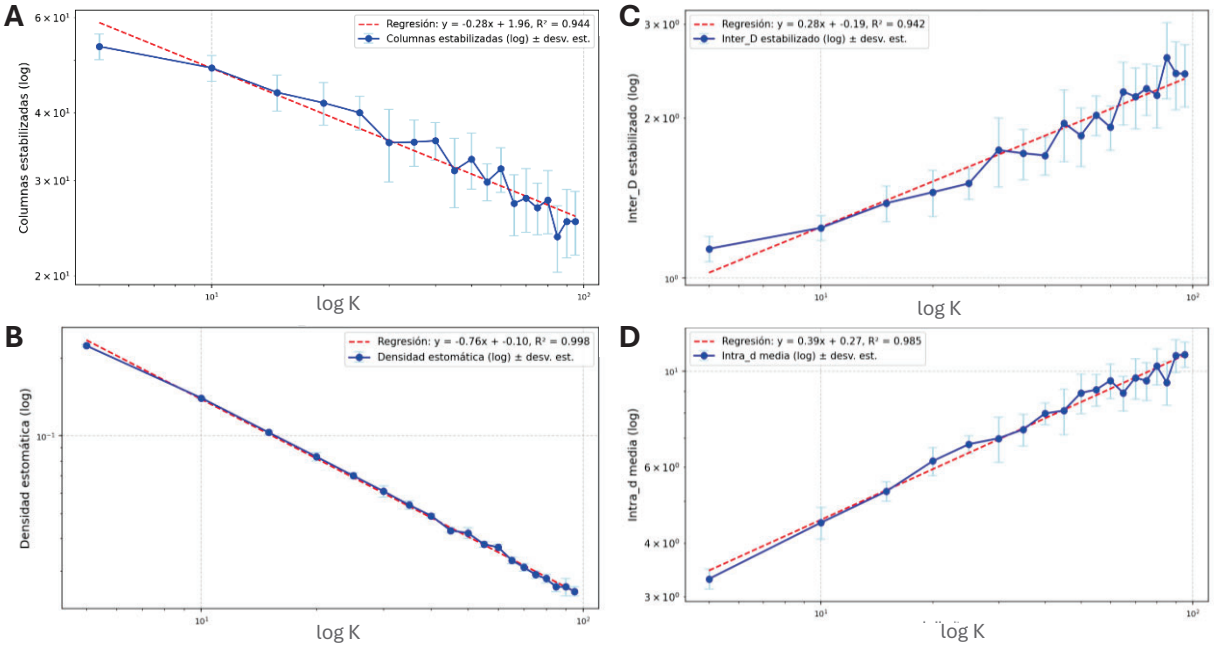


Figura 3.7 – Gráficas de relación entre el número de columnas finales (A), densidad estomática (B), distancia interestomática *D* (C) y distancia intraestomática *d* (D) respecto a *K* para una pendiente de función de probabilidad = 10. Para todas las imágenes la gráfica es logarítmica.

CAPÍTULO 4

DISCUSIÓN

4.1 Interpretación, análisis estadístico y validación de los resultados

Los resultados obtenidos reflejan la capacidad del modelo para replicar patrones estomáticos lineales observados en acículas de pinos, destacando el papel de la constante *K* de decisión como un parámetro clave que simula las demandas locales metabólicas y ambientales de la planta. En simulaciones con valores bajos de *K*, el modelo genera densidades elevadas, lo que se puede traducir como un reflejo de la adaptación al ambiente y a los estímulos, reconociendo el sistema que se requiere una mayor cantidad de estomas para hacer frente a esas condiciones concretas. Por el contrario, para valores altos de *K*, las densidades bajan considerablemente, simulando entornos o bien menos restrictivo o situaciones en la que le es favorable a la planta tener una menor cantidad de estomas. Este comportamiento sugiere que *K* actúa como *proxy* de la regulación biológica, integrando de forma simplificada mecanismos genéticos, de señalización y ambientales que controlan la diferenciación estomática.

Un aspecto notable es cómo emergen y se producen dos estados en el modelo: un *estado transitorio*, caracterizado por patrones regulares en las primeras filas; y un *estado estabilizado*, donde se consolidan un número fijo de hileras estomáticas y cuyo comportamiento es muy parecido al visto en la naturaleza. En el modo determinista con distribución inicial saturada, el estado transitorio surge debido a la sincronía en la evaluación de la colocación de estomas, apareciendo filas saturadas cuando la fila es $K+1$ y $\frac{3K}{2}$, pero este efecto se rompe por desempates en la asignación espacial, alcanzando un el estado estabilizado del que hablábamos. En el modo probabilístico, especialmente con pendientes pronunciadas y moderadas (e.g. 50-700), el estado transitorio se reduce y se generan patrones más realistas al permitirse la interrupción de hileras y la aparición de nuevas, un fenómeno poco común pero observado en acículas reales. Sin embargo, con pendientes bajas, como por ejemplo 10 o menos, la alta estocasticidad introduce ruido excesivo, desdibujando la linealidad de las hileras. Esto nos hace pensar que una pendiente intermedia optimiza el equilibrio entre la flexibilidad y el realismo biológico y

no deja de ser una muestra más del fino control que la planta debe poseer para repartir el espacio de forma adecuada, eficaz y óptima.

Desde el punto de vista estadístico, las métricas calculadas muestran robustez en múltiples simulaciones. Los gráficos generados por el Código 3 (Figuras 3.5-3.7), indican que la densidad estomática y el número de hileras siguen una relación de potencias con K, ajustada por la ecuación $Y = 10^b \cdot K^m$, con coeficientes de determinación (R^2) entre 0.94 y 0.99 en escala logarítmica. Por otro lado, si bien en las gráficas de las Figuras 3.5-3.7 la desviación parece ser muy elevada y poder llevar a pensar que los datos tienen una excesiva variabilidad, hay que recalcar que en esa escala cualquier pequeña variación supone un gran cambio y que, como se puede observar como ejemplo en la Figura 4.1, cuando la representación es lineal, la desviación es mucho menor y en algunos casos, casi despreciable.

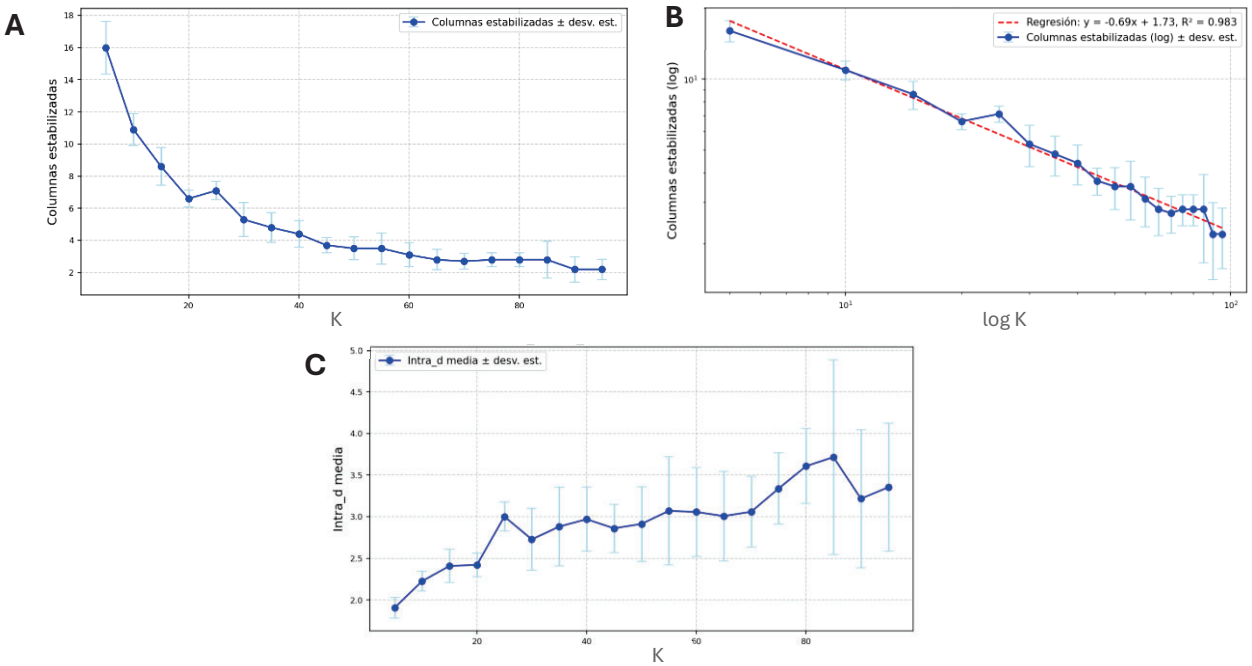


Figura 4.1 – Gráficas de relación entre el número de columnas finales (A y B) y distancia intraestomática *d* (C) respecto a *K* para una pendiente de función de probabilidad = 700. Para las imágenes (A) y (C) la gráfica es lineal. Para la imagen (B) la gráfica es logarítmica.

Si bien esto último es cierto, igualmente las gráficas relacionadas con la distancia intraestomática *d* son donde hay siempre mayor desviación, pese a que en términos generales la variación sea menor dependiendo de *K*, incluso cuando se representa en relación lineal de las variables.

Para validar la robustez del modelo, se realizaron múltiples simulaciones por cada valor de *K* para cada configuración analizada. Las desviaciones estándar de las métricas se reportan en el archivo *Métricas.txt*, del cual se puede encontrar un extracto en al Anexo B. Ahí se comprueba que, en efecto, las desviaciones son bajas, lo que indica una alta reproducibilidad. Además, la consistencia entre distribuciones iniciales (aleatoria, regular, gaussiana y saturada) para un mismo valor de *K* confirma que los resultados dependen principalmente de esa constante y del modo de decisión y no de la configuración inicial. Por otro lado, la no contigüidad de estomas, un comportamiento biológico clave, surge de forma autónoma en el modelo probabilístico con pendiente alta o moderada, sin necesidad de forzar la regla de una célula de separación, lo que refuerza la validez del modelo.

4.2 Comparación con datos empíricos

Dado que este trabajo no incluye datos empíricos directos propios, la validación del modelos se basa en la comparación cualitativa con datos reportados en la literatura científica sobre la distribución estomática en estas especies. Los resultados simulados se alinean con observaciones empíricas en varios aspectos clave, aunque ciertas discrepancias destacan las limitaciones del modelo actual, así como oportunidades para futuras mejoras, de lo cual hablaremos más en profundidad en el subapartado siguiente.

Las densidades estomáticas generadas por el modelo, que oscilan entre 0.01 para valores de K elevados y 0.16 para valores bajos (véase la Figura 4.2), refleja la tendencia comprobada en varios estudios[13,20,21] de que la densidad estomática aumenta ante cambios ambientales como aumentos de temperatura o incrementos en la altitud. Por otro lado, el número de hileras coincide con observaciones en especies como *Pinus arizonica*, *P. cooperi* y otras, que presentan hileras longitudinales variables según la especie y las condiciones[15]; mientras que la relativamente baja variación en la distancia entre estomas de una misma hilera, representado por una distancia d estable entre 2 y 4 en las simulaciones flexibles, se alinea con el espaciado regular observado en el Desarrollo mesoperígeno de pinos, donde divisiones asimétricas garantizan una separación óptima entre estomas[3].

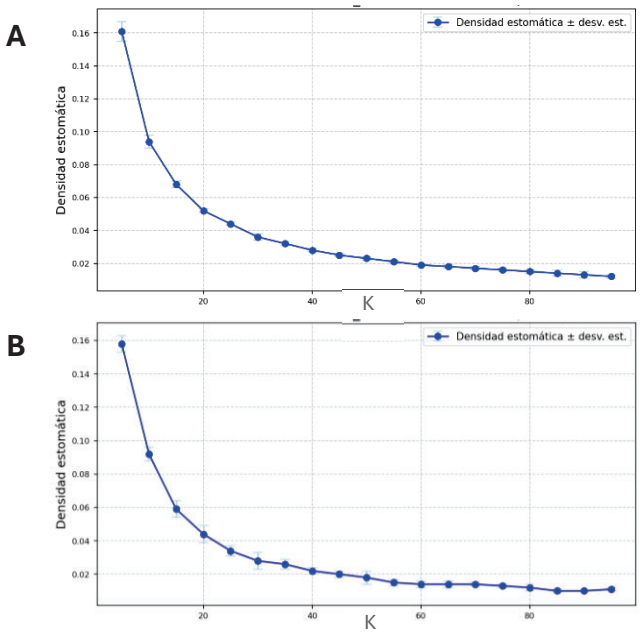


Figura 4.2 – Gráficas de relación lineal entre la densidad estomática respecto a *K* para una pendiente de función de probabilidad = 700 (A) y 50 (B).

4.3 Limitación del modelo, aplicaciones y generalización

A pesar de las bondades que pueda tener el modelo que proponemos en este trabajo, somos conscientes de sus limitaciones, así como de las posibles áreas de mejora. Por un lado, como se indicó en su momento en el Capítulo 2.2, parte de estas limitaciones derivan de sus asunciones simplificadoras. La representación de la acícula como una matriz bidimensional ignora su geometría cilíndrica o semicilíndrica, lo que puede afectar a la precisión de los patrones espaciales, especialmente en especies con una morfología algo más compleja.

Además, la constante *K* actúa como un parámetro global, simplificando los mecanismos biológicos de diferenciación estomática en una sola variable y omitiendo dinámicas moleculares y hormonales complejas y específicas. Un ejemplo de esto es que el modelo tampoco captura de manera explícita el efecto de concentraciones elevadas de CO_2 que se ha comprobado reducen la densidad estomática en especies como *Pinus sylvestris*[21], las fases de diferenciación estomática observadas en esa misma especie y en *P. nigra*.[13] o la influencia de la absorción foliar y conductancia analizadas en *P. torreyana*[12]. Por otro lado, el modelo probabilístico con pendiente baja genera un ruido

excesivo que desdibuja la linealidad estomática, alejándose de estos patrones observados. Sin embargo, esto mismo puede ser una fortaleza del modelo, ya que la flexibilidad que ofrece el variar y modificar esta constante permite pasar de distribuciones en hileras como las observadas en pinos a distribuciones más dispersas y aparentemente aleatorias como las que se producen en especies muy alejadas de las que nos hemos centrado en este proyecto como *Quercus mongolica*[2], lo que permitiría una futura generalización para un rango de especies y distribuciones mucho más amplio. Finalmente, simulaciones con matrices grandes son computacionalmente muy costas y ello restringe su escalabilidad.

A pesar de estas limitaciones, el modelo tiene aplicaciones relevantes. Permite predecir cómo las densidades estomáticas podrían variar bajo condiciones de estrés ambiental, lo que es útil para estudiar la preservación y resistencia de la biodiversidad ante climas cambiantes como las sufridas por el cambio climático; o como punto de apoyo de la posibilidad de emplear técnicas de ingeniería genética vegetal para regular a conveniencia la capacidad estomática de la planta de estudio. A su vez, podría también servir de enlace e integración con modelos fotosintéticos más complejos como el del Ball-Woodrow-Berru[4] para optimizar predicciones de conductancia y eficiencia hídrica. Otro enfoque de la utilidad de este modelo es el potencial educativo que tiene, ilustrando cómo reglas locales sencillas generan patrones complejos de fisiología vegetal; así como el uso de los datos generados para alimentar y entrenar algoritmos de aprendizaje automático para predecir la conductancia, de una manera similar a los enfoques propuestos por Gaur y Drewry[19].

Por último, como indicamos previamente, algunas de las limitaciones del modelo derivadas de su flexibilidad pueden dar pie a la generalización del modelo para adaptarlo a otras especies con distribuciones más diversas[2] modificando las reglas espaciales para patrones no lineales e integrando conceptos como los modelos de optimalidad[18] y de reacción-difusión[17], además de, tal y como está construido el modelo, el análisis cuantitativo con datos reales es fácil de implementar si se cuenta con una gran cantidad de información de referencia.

CAPÍTULO 5

CONCLUSIONES

Este trabajo de fin de grado ha representado un esfuerzo interdisciplinario por unir la biología vegetal, la modelización matemática y la programación computacional, con el objetivo de desentrañar los misterios de la distribución estomática en acículas de pino. A través del desarrollo de un modelo matemático-computacional implementado en Python, hemos logrado no solo cumplir con el objetivo general de simular el desarrollo y la disposición de estomas en estructuras foliarles lineales, sino también avanzar en los objetivos específicos que nos propusimos al comienzo del proyecto: implementar algoritmos para generar patrones a partir de diversas distribuciones iniciales, analizar métricas clave como la densidad o el número de hileras y generar visualizaciones gráficas que revelan la dependencia de estos parámetros con la constante umbral de decisión K.

Los resultados obtenidos demuestran que, mediante reglas locales simples inspiradas en procesos biológicos complejos, el modelo genera patrones emergentes que replican cualitativamente los observados en la naturaleza. En sus configuraciones mas restrictivas, se observan distribuciones mucho más regulares y predecibles, mientras que en escenarios más flexibles y estocásticos, siguen emergiendo comportamientos estables como la formación de hileras mientras que añaden la posibilidad de generación de nuevas hileras o la interrupción de las mismas, así como de la aparición de estomas que rompen la regularidad. La relación de potencias entre K y métricas como la densidad, que se ajustan a una función general $Y = 10^b \cdot K^m$ con coeficientes de determinación entre 0.94 y 0.99 (lo que indica un buen ajuste) subraya cómo factores ambientales y genéticos, representados de forma simplificada por esta misma constante K, podrían modular la adaptación de las plantas a entornos cambiante. Notablemente, la distancia d se mantiene relativamente estable en rangos biológicamente plausibles independientemente de K, lo que refuerza la robustez del modelo al alinearse con los datos de referencia observados en diferentes especies de pino.

Estas simulaciones no solo validan conceptos clave de fisiología vegetal, como la optimización del reparto espacial para el intercambio gaseoso y la conductancia hídrica,

sino que también abre la puerta a futuras herramientas prácticas para predecir respuestas vegetales a cambios de clima y ambiente. En un mundo donde los bosques boreales y templados, con una alta presencia de estas especies, son imprescindibles como sumideros de carbono, comprender y modelar el funcionamiento de estructuras que son fundamentales en el intercambio gaseoso puede sustentar futras estrategias de conservación, empleando tal vez herramientas de ingeniería genética y manejo forestal sostenible. Las limitaciones identificadas no restan valor al modelo, sino que abren puertas a mejoras y desarrollos futuros incorporando datos empíricos precisos, variables ambientales dinámicas o generalizan y extendiendo el modelo a otras especies con patrones no lineales, como ocurre generalmente en dicotiledóneas.

En última instancia, este proyecto ilustra el poder de la modelización computacional para comprender y trabajar con procesos biológicos complejos, demostrando que patrones aparentemente caóticos o arbitrarios en la naturaleza surgen de reglas locales elegantes, sencillas y eficientes. Al simular esta “clase de danza” microscópica de los estomas en las acículas de pinos, solo esperamos haber contribuido al avance del conocimiento en biología vegetal y a investigaciones que fortalezcan la resiliencia de los ecosistemas frente a desafíos globales. Este trabajo culmina con la convicción personal de que, mediante la fusión de ciencia y tecnología, podemos no solo entender mejor el mundo vegetal, sino también protegerlo para generaciones venideras. Así sea.

CAPÍTULO 6

BIBLIOGRAFÍA

1. Richardson, L. G. L. & Torii, K. U. Take a deep breath: peptide signalling in stomatal patterning and differentiation. *J. Exp. Bot.* **64**, 5243–5251 (2013).
2. Liu, C. *et al.* Stomatal Arrangement Pattern: A New Direction to Explore Plant Adaptation and Evolution. *Front. Plant Sci.* **12**, 655255 (2021).
3. Rudall, P. J., Hilton, J. & Bateman, R. M. Several developmental and morphogenetic factors govern the evolution of stomatal patterning in land plants. *New Phytol.* **200**, 598–614 (2013).
4. Haefner, J. W., Buckley, T. N. & Mott, K. A. A spatially explicit model of patchy stomatal responses to humidity. *Plant Cell Environ.* **20**, 1087–1097 (1997).
5. Fontana, M., Labrecque, M., Collin, A. & Bélanger, N. Stomatal distribution patterns change according to leaf development and leaf water status in Salix miyabeana. *Plant Growth Regul.* **81**, 63–70 (2017).
6. Baek, I. *et al.* Spatial patterning of chloroplasts and stomata in developing cacao leaves. *Commun. Biol.* **8**, 554 (2025).
7. Brodribb, T. & Hill, R. S. Imbricacy and Stomatal Wax Plugs Reduce Maximum Leaf Conductance in Southern Hemisphere Conifers. *Aust. J. Bot.* **45**, 657–668 (1997).
8. Woodruff, D. R., Meinzer, F. C. & McCulloh, K. A. Height-related trends in stomatal sensitivity to leaf-to-air vapour pressure deficit in a tall conifer. *J. Exp. Bot.* **61**, 203–210 (2010).
9. García-Amorena, I., Wagner, F., van Hoof, T. B. & Gómez Manzaneque, F. Stomatal responses in deciduous oaks from southern Europe to the anthropogenic atmospheric

CO2 increase; refining the stomatal-based CO2 proxy. *Rev. Palaeobot. Palynol.* **141**, 303–312 (2006).

10. García Álvarez, S., Morla Juaristi, C., Solana Gutiérrez, J. & García-Amorena, I. Taxonomic differences between Pinus sylvestris and P. uncinata revealed in the stomata and cuticle characters for use in the study of fossil material. *Rev. Palaeobot. Palynol.* **155**, 61–68 (2009).
11. Pillitteri, L. J. & Torii, K. U. Mechanisms of stomatal development. *Annu. Rev. Plant Biol.* **63**, 591–614 (2012).
12. Tianshi, E. & Chau, P. C. Foliar water uptake in the needles of Pinus torreyana. *Plant Ecol.* **223**, 465–477 (2022).
13. Marek, S. *et al.* Stomatal density in Pinus sylvestris as an indicator of temperature rather than CO2: Evidence from a pan-European transect. *Plant Cell Environ.* **45**, 121–132 (2022).
14. Serna, L., Torres-Contreras, J. & Fenoll, C. Clonal Analysis of Stomatal Development and Patterning in Arabidopsis Leaves. *Dev. Biol.* **241**, 24–33 (2002).
15. De La Paz Pérez Olvera, C. & Ceja-Romero, J. Anatomía de la hoja de seis especies de Pinus del estado de Durango, México. *Madera Bosques* **25**, (2019).
16. Brodribb, T. J., McAdam, S. A. M., Jordan, G. J. & Martins, S. C. V. Conifer species adapt to low-rainfall climates by following one of two divergent pathways. *Proc. Natl. Acad. Sci. U. S. A.* **111**, 14489–14493 (2014).
17. Prusinkiewicz, P. & Runions, A. Computational models of plant development and form. *New Phytol.* **193**, 549–569 (2012).
18. Dow, G. J., Bergmann, D. C. & Berry, J. A. An integrated model of stomatal development and leaf physiology. *New Phytol.* **201**, 1218–1226 (2014).

19. Gaur, S. & Drewry, D. T. Explainable machine learning for predicting stomatal conductance across multiple plant functional types. *Agric. For. Meteorol.* **350**, 109955 (2024).

20. Tiwari, S. P., Kumar, P., Yadav, D. & Chauhan, D. K. Comparative morphological, epidermal, and anatomical studies of Pinus roxburghii needles at different altitudes in the North-West Indian Himalayas. *Turk. J. Bot.* https://doi.org/10.3906/bot-1110-1 (2013) doi:10.3906/bot-1110-1.

21. Lin, J., Jach, M. E. & Ceulemans, R. Stomatal density and needle anatomy of Scots pine (Pinus sylvestris) are affected by elevated CO2. *New Phytol.* **150**, 665–674 (2001).

22. Guzicka, M., Marek, S., Gawlak, M. & Tomaszewski, D. Micromorphology of Pine Needle Primordia and Young Needles after Bud Dormancy Breaking. *Plants* **12**, 913 (2023).

ANEXO A

CÓDIGOS DEL MODELO

A.0 – Módulos y librerías

```
import random
import math
import os
import shutil
import time
from statistics import mean, stdev
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib.patches import Circle
from PIL import Image
import io
import numpy as np
import re
```

A.1 – Código 1 – Posiciones

```
def format_time(seconds):
    """Formatea el tiempo en minutos y segundos."""
    minutes = int(seconds // 60)
    seconds = int(seconds % 60)
    return f"{minutes} minutos y {seconds} segundos"

def generate_first_row(m, inter_D_media, distribution_type, sigma=None
, lateral_distance=0, force_non_contiguity=True, force_lateral_restric
tion=True):
    """
    Genera la primera fila según el tipo de distribución elegido.
    """
    if force_lateral_restriction and (lateral_distance < 0 or lateral_
distance > m // 2):
        raise ValueError(f"lateral_distance debe estar entre 0 y {m //
2}.")

    if not force_lateral_restriction:
        lateral_distance = 0

    row = [1] * m
    zero_cols = set()

    if distribution_type == 1:  # Aleatoria
        num_zeros = max(1, int((m - 2 * lateral_distance) / inter_D_me
dia))
        available_positions = list(range(lateral_distance, m - lateral
_distance))
        zeros_placed = 0
        for _ in range(num_zeros):
            if not available_positions:
                print(f"Advertencia: No se pudieron colocar {num_zeros
- zeros_placed} ceros debido a restricciones.")
                break
            pos = random.choice(available_positions)
            row[pos] = 0
            zero_cols.add(pos)
            zeros_placed += 1
            if force_non_contiguity:
                available_positions = [p for p in available_positions
if p not in {pos, pos-1, pos+1}]
            else:
                available_positions.remove(pos)

    elif distribution_type == 2:  # Regular
        if inter_D_media <= 1:
            pos = lateral_distance
```

```
            while pos < m - lateral_distance:
                row[pos] = 0
                zero_cols.add(pos)
                pos += 2 if force_non_contiguity else 1
        else:
            inter_D_media = max(2 if force_non_contiguity else 1, int(
round(inter_D_media)))
            if inter_D_media >= m - 2 * lateral_distance:
                pos = lateral_distance + (m - 2 * lateral_distance) //
2
                row[pos] = 0
                zero_cols.add(pos)
            else:
                pos = lateral_distance + max(0, inter_D_media - 1)
                while pos < m - lateral_distance:
                    row[pos] = 0
                    zero_cols.add(pos)
                    pos += inter_D_media

    elif distribution_type == 3:  # Gaussiana
        num_zeros = max(1, int((m - 2 * lateral_distance) / inter_D_me
dia))
        if sigma is None:
            sigma = inter_D_media / 3
        available_positions = list(range(lateral_distance, m - lateral
_distance))
        if num_zeros == 1:
            pos = lateral_distance + (m - 2 * lateral_distance) // 2
            row[pos] = 0
            zero_cols.add(pos)
        else:
            for _ in range(num_zeros):
                if not available_positions:
                    print(f"Advertencia: No se pudieron colocar todos
los ceros debido a restricciones.")
                    break
                first_pos = random.gauss(mu=inter_D_media - 1 + latera
l_distance, sigma=sigma)
                first_pos = max(lateral_distance, min(m - 1 - lateral_
distance, int(round(abs(first_pos)))))
                if first_pos in available_positions:
                    row[first_pos] = 0
                    zero_cols.add(first_pos)
                    if force_non_contiguity:
                        available_positions = [p for p in available_po
sitions if p not in {first_pos, first_pos-1, first_pos+1}]
                    else:
                        available_positions.remove(first_pos)

    elif distribution_type == 4:  # Especial
        for pos in range(lateral_distance, m - lateral_distance):
            row[pos] = 0
```

```
            zero_cols.add(pos)

    else:
        raise ValueError("Tipo de distribución inválido. Use 1 (aleato
ria), 2 (regular), 3 (gaussiana) o 4 (especial).")

    return row, zero_cols

def initialize_matrix(n, m, zero_cols):
    """Inicializa la matriz con 1s en columnas no permitidas y -1 en c
olumnas permitidas."""
    matrix = [[1 if j not in zero_cols else -1 for j in range(m)] for
i in range(n)]
    return matrix

def assign_ones_to_zeros(matrix, zero_positions, max_row):
    """Asigna cada 1 al 0 más cercano (distancia euclidiana) hasta max
_row."""
    associations = {zero: [] for zero in zero_positions}
    n, m = max_row + 1, len(matrix[0])

    for i in range(n):
        for j in range(m):
            if matrix[i][j] != 1:
                continue
            distances = [(math.sqrt((i - zi)**2 + (j - zj)**2), (zi, z
j))
                         for zi, zj in zero_positions if zi <= max_row
]
            if not distances:
                continue
            min_dist = min(distances, key=lambda x: x[0])[0]
            closest_zeros = [z for d, z in distances if d == min_dist]
            chosen_zero = random.choice(closest_zeros)
            associations[chosen_zero].append((i, j))

    return associations

def evaluate_row(matrix, row_idx, zero_cols, k_limite, decision_mode="
binary", slope=5, force_non_contiguity=True):
    """Evalúa una fila para decidir 0s o 1s en posiciones vacantes."""
    m = len(matrix[0])
    zero_positions = [(i, j) for i in range(row_idx) for j in range(m)
if matrix[i][j] == 0]
    associations = assign_ones_to_zeros(matrix, zero_positions, row_id
x - 1) if zero_positions else {}

    for j in zero_cols:
        is_valid = True
        if force_non_contiguity:
            if row_idx > 0 and matrix[row_idx-1][j] == 0:
                is_valid = False
```

```
            if j > 0 and matrix[row_idx][j-1] == 0:
                is_valid = False
            if j < m-1 and matrix[row_idx][j+1] == 0:
                is_valid = False

        if force_non_contiguity and not is_valid:
            matrix[row_idx][j] = 1
            continue

        k_anterior = 0
        for i in range(row_idx - 1, -1, -1):
            if matrix[i][j] == 0:
                k_anterior = len(associations.get((i, j), []))
                break

        if decision_mode == "binary":
            matrix[row_idx][j] = 0 if k_anterior >= k_limite else 1
        else:
            B = slope
            p = 1 / (1 + math.exp(B * (1 - k_anterior / k_limite)))
            matrix[row_idx][j] = 0 if random.random() < p else 1

def plot_matrix(matrix, associations, current_row, total_rows):
    """Genera una imagen de la matriz con 0s como círculos negros y si
n texto ni leyenda."""
    n, m = len(matrix), len(matrix[0])
    zero_positions = [(i, j) for i in range(current_row + 1) for j in
range(m) if matrix[i][j] == 0]

    colors = ['#0000FF', '#FF0000', '#00FF00', '#FFFF00', '#FFA500', '
#800080']
    cmap = mcolors.ListedColormap(colors)

    color_map = {}
    for zero_pos in zero_positions:
        i, j = zero_pos
        used_colors = set()

        last_zero_in_column = None
        for k in range(i - 1, -1, -1):
            if matrix[k][j] == 0:
                last_zero_in_column = (k, j)
                break
        if last_zero_in_column in color_map:
            used_colors.add(color_map[last_zero_in_column])

        closest_left_zero = None
        min_left_distance = float('inf')
        for zi, zj in zero_positions:
            if zi <= i and zj < j:
                distance = math.sqrt((i - zi)**2 + (j - zj)**2)
                if distance < min_left_distance:
```

```
                    min_left_distance = distance
                    closest_left_zero = (zi, zj)
        if closest_left_zero in color_map:
            used_colors.add(color_map[closest_left_zero])

        closest_right_zero = None
        min_right_distance = float('inf')
        for zi, zj in zero_positions:
            if zi <= i and zj > j:
                distance = math.sqrt((i - zi)**2 + (j - zj)**2)
                if distance < min_right_distance:
                    min_right_distance = distance
                    closest_right_zero = (zi, zj)
        if closest_right_zero in color_map:
            used_colors.add(color_map[closest_right_zero])

        available_colors = [i for i in range(len(colors)) if i not in
used_colors]
        color_map[zero_pos] = available_colors[0] if available_colors
else 0

    color_matrix = np.full((n, m), -1, dtype=float)
    for zero_pos, ones in associations.items():
        for one_pos in ones:
            color_matrix[one_pos[0]][one_pos[1]] = color_map[zero_pos]

    for i in range(n):
        for j in range(m):
            if matrix[i][j] == 1 and color_matrix[i][j] == -1:
                color_matrix[i][j] = -2  # Gris claro para 1s no asoci
ados
            if matrix[i][j] == -1:
                color_matrix[i][j] = -3  # Blanco para celdas vacantes

    cmap_with_extras = mcolors.ListedColormap(colors + ['#D3D3D3', '#F
FFFFF'])

    cell_size = 0.2
    fig_width = max(5, m * cell_size)
    fig_height = max(3, n * cell_size)

    fig, ax = plt.subplots(figsize=(fig_width, fig_height))
    ax.imshow(color_matrix, cmap=cmap_with_extras, vmin=0, vmax=len(co
lors) + 1)

    for i in range(n):
        for j in range(m):
            if matrix[i][j] == 0:
                circle = Circle((j, i), 0.4, color='black', zorder=10)
                ax.add_patch(circle)

    zero_count = sum(row.count(0) for row in matrix[:current_row + 1])
```

```
    total_elements = (current_row + 1) * m
    title = f"Densidad estomática: {zero_count / total_elements:.3f} (
Fila {current_row + 1}/{total_rows})"
    ax.set_title(title, fontsize=12)
    ax.set_yticks(range(n))
    ax.set_xticks(range(m))

    buf = io.BytesIO()
    plt.savefig(buf, format='png', bbox_inches='tight')
    plt.close(fig)
    buf.seek(0)
    img = Image.open(buf).convert('RGB')

    return img

def plot_final_matrix(matrix, associations, save_path=None):
    """Guarda la gráfica estática de la matriz final con 0s como círcu
los negros."""
    n, m = len(matrix), len(matrix[0])
    zero_positions = [(i, j) for i in range(n) for j in range(m) if ma
trix[i][j] == 0]

    colors = ['#0000FF', '#FF0000', '#00FF00', '#FFFF00', '#FFA500', '
#800080']
    cmap = mcolors.ListedColormap(colors)

    color_map = {}
    for zero_pos in zero_positions:
        i, j = zero_pos
        used_colors = set()

        last_zero_in_column = None
        for k in range(i - 1, -1, -1):
            if matrix[k][j] == 0:
                last_zero_in_column = (k, j)
                break
        if last_zero_in_column in color_map:
            used_colors.add(color_map[last_zero_in_column])

        closest_left_zero = None
        min_left_distance = float('inf')
        for zi, zj in zero_positions:
            if zi <= i and zj < j:
                distance = math.sqrt((i - zi)**2 + (j - zj)**2)
                if distance < min_left_distance:
                    min_left_distance = distance
                    closest_left_zero = (zi, zj)
        if closest_left_zero in color_map:
            used_colors.add(color_map[closest_left_zero])

        closest_right_zero = None
        min_right_distance = float('inf')
```

```
        for zi, zj in zero_positions:
            if zi <= i and zj > j:
                distance = math.sqrt((i - zi)**2 + (j - zj)**2)
                if distance < min_right_distance:
                    min_right_distance = distance
                    closest_right_zero = (zi, zj)
        if closest_right_zero in color_map:
            used_colors.add(color_map[closest_right_zero])

        available_colors = [i for i in range(len(colors)) if i not in
used_colors]
        color_map[zero_pos] = available_colors[0] if available_colors
else 0

    color_matrix = np.full((n, m), -1, dtype=float)
    for zero_pos, ones in associations.items():
        for one_pos in ones:
            color_matrix[one_pos[0]][one_pos[1]] = color_map[zero_pos]

    for i in range(n):
        for j in range(m):
            if matrix[i][j] == 1 and color_matrix[i][j] == -1:
                color_matrix[i][j] = -2
            if matrix[i][j] == -1:
                color_matrix[i][j] = -3

    cmap_with_extras = mcolors.ListedColormap(colors + ['#D3D3D3', '#F
FFFFF'])

    cell_size = 0.2
    fig_width = max(5, m * cell_size)
    fig_height = max(3, n * cell_size)

    fig, ax = plt.subplots(figsize=(fig_width, fig_height))
    ax.imshow(color_matrix, cmap=cmap_with_extras, vmin=0, vmax=len(co
lors) + 1)

    for i in range(n):
        for j in range(m):
            if matrix[i][j] == 0:
                circle = Circle((j, i), 0.4, color='black', zorder=10)
                ax.add_patch(circle)

    zero_count = sum(row.count(0) for row in matrix)
    total_elements = n * m
    title = f"Densidad estomática final: {zero_count / total_elements:
.3f}"
    ax.set_title(title, fontsize=12)
    ax.set_yticks(range(n))
    ax.set_xticks(range(m))

    if save_path:
```

```
            plt.savefig(save_path, format='png', bbox_inches='tight')
            plt.close(fig)
        else:
            plt.close(fig)

def generate_matrix_and_save_positions(n, m, inter_D_media, k_limite,
distribution_type, decision_mode="binary", slope=5, sigma=None, sim_nu
m=1, output_file="Posiciones.txt", lateral_distance=0, force_non_conti
guity=True, force_lateral_restriction=True, generate_images=False, gen
erate_gif=False):
    """Genera una matriz, guarda posiciones de 0s y, opcionalmente, ge
nera imágenes/GIFs."""
    start_time = time.time()

    first_row, zero_cols = generate_first_row(m, inter_D_media, distri
bution_type, sigma, lateral_distance, force_non_contiguity, force_late
ral_restriction)
    matrix = initialize_matrix(n, m, zero_cols)
    matrix[0] = first_row

    zero_positions_by_column = [[] for _ in range(m)]
    for j in range(m):
        if matrix[0][j] == 0:
            zero_positions_by_column[j].append(0)

    images = [] if generate_gif else None
    if generate_gif or generate_images:
        zero_positions = [(0, j) for j in range(m) if matrix[0][j] ==
0]
        associations = assign_ones_to_zeros(matrix, zero_positions, 0)
        if generate_gif:
            img = plot_matrix(matrix, associations, current_row=0, tot
al_rows=n)
            images.append(img)

    for row_idx in range(1, n):
        evaluate_row(matrix, row_idx, zero_cols, k_limite, decision_mo
de, slope, force_non_contiguity)
        for j in range(m):
            if matrix[row_idx][j] == 0:
                zero_positions_by_column[j].append(row_idx)
        if generate_gif:
            zero_positions = [(i, j) for i in range(row_idx + 1) for j
in range(m) if matrix[i][j] == 0]
            associations = assign_ones_to_zeros(matrix, zero_positions
, row_idx)
            img = plot_matrix(matrix, associations, current_row=row_id
x, total_rows=n)
            images.append(img)

    with open(output_file, "a") as f:
        f.write(f"\nSimulación {sim_num}, k_limite = {k_limite}\n")
```

```
            f.write("Posiciones de 0s por columna (filas):\n")
            for j in range(m):
                f.write(f"Columna {j}: {zero_positions_by_column[j]}\n")

    if generate_images or generate_gif:
        zero_positions = [(i, j) for i in range(n) for j in range(m) i
f matrix[i][j] == 0]
        associations = assign_ones_to_zeros(matrix, zero_positions, n
- 1)
        if generate_images:
            image_path = f"matrix_images/matrix_k_{k_limite}_sim_{sim_
num}.png"
            plot_final_matrix(matrix, associations, save_path=image_pa
th)
        if generate_gif:
            gif_filename = f"matrix_images/matrix_evolution_k_{k_limit
e}_sim_{sim_num}.gif"
            images[0].save(gif_filename, save_all=True, append_images=
images[1:], duration=500, loop=0)

    end_time = time.time()
    sim_time = end_time - start_time
    with open(output_file, "a") as f:
        f.write(f"Tiempo de simulación {sim_num} para k_limite = {k_li
mite}: {format_time(sim_time)}\n")

    return matrix, sim_time

def main_generate_positions():
    """Función principal para generar matrices y guardar posiciones de
0s."""
    start_time = time.time()

    if os.path.exists("Posiciones.txt"):
        os.remove("Posiciones.txt")
    if os.path.exists("matrix_images"):
        shutil.rmtree("matrix_images")
    os.makedirs("matrix_images", exist_ok=True)

    print("Generación de matrices y registro de posiciones de 0s.")

    while True:
        try:
            distribution_type = int(input("Elija el tipo de distribuci
ón para la primera fila (1=aleatoria, 2=regular, 3=gaussiana, 4=especi
al): "))
            if distribution_type in [1, 2, 3, 4]:
                break
            print("Por favor, ingrese 1, 2, 3 o 4.")
        except ValueError:
            print("Entrada inválida. Ingrese un número (1, 2, 3 o 4)."
)
```

```
    inter_D_media = None
    if distribution_type != 4:
        while True:
            try:
                inter_D_media = float(input("Ingrese la distancia prom
edio entre estomas (inter_D_media): "))
                if inter_D_media > 0:
                    break
                print("inter_D_media debe ser positivo.")
            except ValueError:
                print("Entrada inválida. Ingrese un número.")

    sigma = None
    if distribution_type == 3:
        try:
            sigma_input = input("Ingrese la desviación estándar para l
a distribución gaussiana (sigma, presione Enter para usar inter_D_medi
a/3): ")
            sigma = float(sigma_input) if sigma_input else inter_D_med
ia / 3
            if sigma <= 0:
                print("sigma debe ser positivo. Usando inter_D_media/3
.")
                sigma = inter_D_media / 3
        except ValueError:
            print("Entrada inválida. Usando inter_D_media/3.")
            sigma = inter_D_media / 3

    while True:
        try:
            n = int(input("Ingrese el número de filas (n): "))
            if n > 0:
                break
            print("Las dimensiones deben ser positivas.")
        except ValueError:
            print("Entrada inválida. Ingrese números enteros.")

    while True:
        try:
            m = int(input("Ingrese el número de columnas (m): "))
            if m > 0:
                break
            print("Las dimensiones deben ser positivas.")
        except ValueError:
            print("Entrada inválida. Ingrese números enteros.")

    while True:
        try:
            k_min = int(input("Ingrese el valor mínimo de k_limite (k_
min): "))
            if k_min >= 0:
```

```
                break
            print("k_min debe ser no negativo.")
        except ValueError:
            print("Entrada inválida. Ingrese un número entero.")

    while True:
        try:
            k_max = int(input("Ingrese el valor máximo de k_limite (k_
max): "))
            if k_max >= k_min:
                break
            print("k_max debe ser mayor o igual que k_min.")
        except ValueError:
            print("Entrada inválida. Ingrese un número entero.")

    while True:
        try:
            k_step = float(input("Ingrese el paso para k_limite (k_ste
p): "))
            if k_step > 0:
                break
            print("k_step debe ser positivo.")
        except ValueError:
            print("Entrada inválida. Ingrese un número.")

    while True:
        try:
            num_simulations = int(input("Ingrese el número de simulaci
ones por valor de k_limite: "))
            if num_simulations > 0:
                break
            print("El número de simulaciones debe ser positivo.")
        except ValueError:
            print("Entrada inválida. Ingrese un número entero.")

    while True:
        decision_mode = input("Elija el modo de decisión para colocar
0s (1=determinista, 2=probabilístico): ")
        if decision_mode in ['1', '2']:
            decision_mode = "binary" if decision_mode == '1' else "pro
babilistic"
            break
        print("Por favor, ingrese 1 o 2.")

    slope = 5
    if decision_mode == "probabilistic":
        try:
            slope_input = input("Ingrese la pendiente para la función
probabilística (presione Enter para usar 5): ")
            slope = float(slope_input) if slope_input else 5
            if slope <= 0:
                print("La pendiente debe ser positiva. Usando 5.")
```

```
                slope = 5
            except ValueError:
                print("Entrada inválida. Usando pendiente=5.")
                slope = 5

        while True:
            force_non_contiguity = input("¿Desea forzar la regla de no con
tiguidad (no permitir 0s adyacentes)? (s/n): ").lower()
            if force_non_contiguity in ['s', 'n']:
                force_non_contiguity = (force_non_contiguity == 's')
                break
            print("Por favor, ingrese 's' (sí) o 'n' (no).")

        while True:
            force_lateral_restriction = input("¿Desea forzar una restricci
ón de distancia lateral desde los bordes? (s/n): ").lower()
            if force_lateral_restriction in ['s', 'n']:
                force_lateral_restriction = (force_lateral_restriction ==
's')
                break
            print("Por favor, ingrese 's' (sí) o 'n' (no).")

        lateral_distance = 0
        if force_lateral_restriction:
            while True:
                try:
                    lateral_distance = int(input(f"Ingrese la distancia la
teral desde los bordes (0 a {m//2}): "))
                    if 0 <= lateral_distance <= m // 2:
                        break
                    print(f"La distancia lateral debe estar entre 0 y {m//
2}.")
                except ValueError:
                    print("Entrada inválida. Ingrese un número entero.")

        while True:
            generate_images = input("¿Desea generar imágenes de las matric
es en mapas de colores? (s/n): ").lower()
            if generate_images in ['s', 'n']:
                generate_images = (generate_images == 's')
                break
            print("Por favor, ingrese 's' (sí) o 'n' (no).")

        while True:
            generate_gif = input("¿Desea generar GIFs de la evolución de l
as matrices? (s/n): ").lower()
            if generate_gif in ['s', 'n']:
                generate_gif = (generate_gif == 's')
                break
            print("Por favor, ingrese 's' (sí) o 'n' (no).")

        dist_name = {1: "aleatoria", 2: "regular", 3: "gaussiana", 4: "esp
```

```
ecial"}
    with open("Posiciones.txt", "w") as f:
        f.write(f"Análisis de posiciones de 0s ({num_simulations} simu
laciones por valor de K, distribución {dist_name[distribution_type]})\
n")
        f.write(f"Parámetros: n={n}, m={m}, inter_D_media={inter_D_med
ia}, k_min={k_min}, k_max={k_max}, k_step={k_step}, sigma={sigma}, pen
diente={slope}, lateral_distance={lateral_distance}, force_non_contigu
ity={force_non_contiguity}, force_lateral_restriction={force_lateral_r
estriction}, generate_images={generate_images}, generate_gif={generate
_gif}\n")

    k_values = [k for k in range(k_min, k_max + 1, int(k_step)) if k <
= k_max]
    total_sim_time = 0
    for k_limite in k_values:
        for sim in range(1, num_simulations + 1):
            _, sim_time = generate_matrix_and_save_positions(
                n, m, inter_D_media, k_limite, distribution_type, deci
sion_mode, slope, sigma, sim,
                output_file="Posiciones.txt", lateral_distance=lateral
_distance,
                force_non_contiguity=force_non_contiguity, force_later
al_restriction=force_lateral_restriction,
                generate_images=generate_images, generate_gif=generate
_gif
            )
            total_sim_time += sim_time
            print(f"Completada simulación {sim} para k_limite = {k_lim
ite} en {format_time(sim_time)}")

    end_time = time.time()
    total_time = end_time - start_time
    with open("Posiciones.txt", "a") as f:
        f.write(f"\nTiempo total de generación: {format_time(total_tim
e)}\n")
    print(f"Tiempo total de generación: {format_time(total_time)}")

if __name__ == "__main__":
    main_generate_positions()
```

A.2 – Código 2 – Métricas

```
def format_time(seconds):
    """Formatea el tiempo en minutos y segundos."""
    minutes = int(seconds // 60)
    seconds = int(seconds % 60)
    return f"{minutes} minutos y {seconds} segundos"

def parse_positions_file(filename, n, m, stable_rows):
    """Lee Posiciones.txt y organiza las posiciones de 0s por k_limite
y simulación."""
    positions_data = {}
    current_k = None
    current_sim = None
    current_col = None
    with open(filename, "r") as f:
        lines = f.readlines()
        for line in lines:
            line = line.strip()
            if line.startswith("Simulación"):
                sim_num, k_value = line.split(", k_limite = ")
                sim_num = int(sim_num.split()[-1])
                k_value = int(k_value)
                current_k = k_value
                current_sim = sim_num
                if current_k not in positions_data:
                    positions_data[current_k] = {}
                positions_data[current_k][current_sim] = [[] for _ in
range(m)]
            elif line.startswith("Columna"):
                col = int(line.split()[1].strip(":"))
                current_col = col
                positions = eval(line.split(": ")[1])
                positions_data[current_k][current_sim][col] = position
s

    return positions_data

def calculate_metrics_from_positions(positions_data, n, m, stable_rows
):
    """Calcula métricas a partir de las posiciones de 0s."""
    stable_start = n - stable_rows
    metrics_data = {}

    for k_limite, sims in positions_data.items():
        metrics_data[k_limite] = []
        density_per_sim = []
        intra_d_per_sim = []
        columns_per_sim = []
```

```
        inter_d_per_sim = []

        for sim, zero_positions_by_column in sims.items():
            stabilized_columns = sum(1 for col_positions in zero_posit
ions_by_column
                                     if any(row >= stable_start for row
in col_positions))

            inter_d_stabilized = float('inf') if stabilized_columns ==
0 else m / stabilized_columns

            zero_count = sum(sum(1 for row in col_positions if row >=
stable_start)
                             for col_positions in zero_positions_by_co
lumn)
            stomatal_density = zero_count / (stable_rows * m) if stabl
e_rows > 0 else None

            intra_d_by_column = [[] for _ in range(m)]
            for j in range(m):
                zero_rows = [row for row in zero_positions_by_column[j
] if row >= stable_start]
                if len(zero_rows) >= 1:
                    for k in range(len(zero_rows) - 1):
                        intra_d = zero_rows[k + 1] - zero_rows[k]
                        intra_d_by_column[j].append(intra_d)
                    last_zero_row = zero_rows[-1]
                    intra_d = n - 1 - last_zero_row
                    intra_d_by_column[j].append(intra_d)

            all_distances = []
            for distances in intra_d_by_column:
                all_distances.extend(distances)

            intra_d_mean = mean(all_distances) if all_distances else N
one
            intra_d_stdev = stdev(all_distances) if len(all_distances)
> 1 else None

            column_means = []
            column_stdevs = []
            for distances in intra_d_by_column:
                if distances:
                    column_means.append(mean(distances))
                    column_stdevs.append(stdev(distances) if len(dista
nces) > 1 else 0)
                else:
                    column_means.append(None)
                    column_stdevs.append(None)

            metrics_data[k_limite].append({
                'sim': sim,
```

```
                'zero_positions': zero_positions_by_column,
                'intra_d_by_column': intra_d_by_column,
                'column_means': column_means,
                'column_stdevs': column_stdevs,
                'intra_d_mean': intra_d_mean,
                'intra_d_stdev': intra_d_stdev,
                'stabilized_columns': stabilized_columns,
                'inter_d_stabilized': inter_d_stabilized,
                'stomatal_density': stomatal_density
            })

            columns_per_sim.append(stabilized_columns)
            if inter_d_stabilized != float('inf'):
                inter_d_per_sim.append(inter_d_stabilized)
            if stomatal_density is not None:
                density_per_sim.append(stomatal_density)
            if intra_d_mean is not None:
                intra_d_per_sim.append(intra_d_mean)

        metrics_data[k_limite].append({
            'columns_mean': mean(columns_per_sim) if columns_per_sim e
lse None,
            'columns_stdev': stdev(columns_per_sim) if len(columns_per
_sim) > 1 else None,
            'inter_d_mean': mean(inter_d_per_sim) if inter_d_per_sim e
lse None,
            'inter_d_stdev': stdev(inter_d_per_sim) if len(inter_d_per
_sim) > 1 else None,
            'density_mean': mean(density_per_sim) if density_per_sim e
lse None,
            'density_stdev': stdev(density_per_sim) if len(density_per
_sim) > 1 else None,
            'intra_d_mean': mean(intra_d_per_sim) if intra_d_per_sim e
lse None,
            'intra_d_stdev': stdev(intra_d_per_sim) if len(intra_d_per
_sim) > 1 else None
        })

    return metrics_data

def write_metrics_file(metrics_data, n, m, stable_rows, parameters, nu
m_simulations, output_file="Metricas.txt", total_time=None):
    """Escribe las métricas en Metricas.txt."""
    dist_name = {1: "aleatoria", 2: "regular", 3: "gaussiana", 4: "esp
ecial"}
    with open(output_file, "w") as f:
        f.write(f"Análisis de métricas ({num_simulations} simulaciones
por valor de K, distribución {dist_name[parameters['distribution_type'
]]})\n")
        f.write(f"Parámetros: n={n}, m={m}, inter_D_media={parameters[
'inter_D_media']}, k_min={parameters['k_min']}, k_max={parameters['k_m
ax']}, k_step={parameters['k_step']}, sigma={parameters['sigma']}, pen
```

```
diente={parameters['pendiente']}, lateral_distance={parameters['latera
l_distance']}, force_non_contiguity={parameters['force_non_contiguity'
]}, force_lateral_restriction={parameters['force_lateral_restriction']
}, stable_rows={stable_rows}, generate_images={parameters['generate_im
ages']}, generate_gif={parameters['generate_gif']}\n\n")

        for k_limite, metrics in sorted(metrics_data.items()):
            f.write(f"k_limite = {k_limite}:\n")
            for metric in metrics[:-1]:
                f.write(f"\nSimulación {metric['sim']}:\n")
                f.write(f"Número de columnas con al menos un 0: {metri
c['stabilized_columns']}\n")
                f.write("Posiciones de 0s por columna (filas):\n")
                for j in range(m):
                    f.write(f"Columna {j}: {metric['zero_positions'][j
]}\n")
                f.write("\nDistancias intra_d por columna (en filas es
tables):\n")
                for j in range(m):
                    if metric['intra_d_by_column'][j]:
                        f.write(f"Columna {j}: {metric['intra_d_by_col
umn'][j]}\n")
                    else:
                        f.write(f"Columna {j}: Sin distancias intra_d\
n")
                f.write("\nMedias y desviaciones estándar por columna:
\n")
                for j in range(m):
                    if metric['column_means'][j] is not None:
                        f.write(f"Columna {j}: Media = {metric['column
_means'][j]:.3f}, Desviación estándar = {(metric['column_stdevs'][j] i
f metric['column_stdevs'][j] is not None else 0):.3f}\n")
                    else:
                        f.write(f"Columna {j}: Media = N/A, Desviación
estándar = N/A\n")
                if metric['intra_d_mean'] is not None:
                    f.write(f"Media intra_d de la matriz: {metric['int
ra_d_mean']:.3f}\n")
                    f.write(f"Desviación estándar intra_d de la matriz
: {(metric['intra_d_stdev'] if metric['intra_d_stdev'] is not None els
e 0):.3f}\n")
                else:
                    f.write("Media intra_d de la matriz: N/A\n")
                    f.write("Desviación estándar intra_d de la matriz:
N/A\n")
                f.write(f"Inter_D estabilizado: {metric['inter_d_stabi
lized']:.3f}\n")
                f.write(f"Densidad estomática: {metric['stomatal_densi
ty']:.3f}\n")

            summary = metrics[-1]
            f.write(f"\nResumen para k_limite = {k_limite}:\n")
```

```
            f.write(f"Columnas estabilizadas: Media = {summary['column
s_mean']:.3f}, Desviación estándar = {(summary['columns_stdev'] if sum
mary['columns_stdev'] is not None else 0):.3f}\n")
            if summary['inter_d_mean'] is not None:
                f.write(f"Inter_D estabilizado: Media = {summary['inte
r_d_mean']:.3f}, Desviación estándar = {(summary['inter_d_stdev'] if s
ummary['inter_d_stdev'] is not None else 0):.3f}\n")
            else:
                f.write("Inter_D estabilizado: Sin columnas estabiliza
das\n")
            if summary['density_mean'] is not None:
                f.write(f"Densidad estomática: Media = {summary['densi
ty_mean']:.3f}, Desviación estándar = {(summary['density_stdev'] if su
mmary['density_stdev'] is not None else 0):.3f}\n")
            else:
                f.write("Densidad estomática: Sin 0s en el estado esta
ble\n")
            if summary['intra_d_mean'] is not None:
                f.write(f"Intra_D: Media = {summary['intra_d_mean']:.3
f}, Desviación estándar = {(summary['intra_d_stdev'] if summary['intra
_d_stdev'] is not None else 0):.3f}\n")
            else:
                f.write("Intra_D: Sin distancias suficientes\n")
            f.write("\n")

        f.write(f"Tiempo total de cálculo de métricas: {format_time(to
tal_time)}\n")

def main_calculate_metrics():
    """Función principal para calcular métricas a partir de Posiciones
.txt."""
    start_time = time.time()

    if not os.path.exists("Posiciones.txt"):
        print("Error: El archivo Posiciones.txt no existe.")
        return

    print("Cálculo de métricas a partir de Posiciones.txt.")

    parameters = {}
    with open("Posiciones.txt", "r") as f:
        lines = f.readlines()
        header_line = lines[0].strip()
        match_sim = re.search(r'\((\d+) simulaciones por valor de K',
header_line)
        if match_sim:
            num_simulations = int(match_sim.group(1))
        else:
            print("Error: No se pudo extraer num_simulations del encab
ezado.")
            return
```

```
        match_dist = re.search(r'distribución (\w+)', header_line)
        if match_dist:
            dist_name = match_dist.group(1)
            dist_map = {"aleatoria": 1, "regular": 2, "gaussiana": 3,
"especial": 4}
            parameters['distribution_type'] = dist_map.get(dist_name,
4)
        else:
            print("Advertencia: No se pudo extraer distribution_type.
Usando 4 (especial).")
            parameters['distribution_type'] = 4

        param_line = lines[1].strip()
        param_parts = param_line.split(", ")
        for part in param_parts:
            key, value = part.split("=")
            key = key.split(": ")[1] if ": " in key else key
            try:
                parameters[key] = eval(value)
            except:
                parameters[key] = value

    n = parameters['n']
    m = parameters['m']

    while True:
        try:
            stable_rows = int(input("Ingrese el número de filas establ
es (e.g., 50): "))
            if 0 < stable_rows <= n:
                break
            print(f"El número de filas estables debe estar entre 1 y {
n}.")
        except ValueError:
            print(f"Entrada inválida. Usando n//2 = {n//2} filas estab
les.")
            stable_rows = n // 2

    positions_data = parse_positions_file("Posiciones.txt", n, m, stab
le_rows)
    metrics_data = calculate_metrics_from_positions(positions_data, n,
m, stable_rows)

    end_time = time.time()
    total_time = end_time - start_time
    write_metrics_file(metrics_data, n, m, stable_rows, parameters, nu
m_simulations, total_time=total_time)

    print(f"Tiempo total de cálculo de métricas: {format_time(total_ti
me)}")
```

```
if __name__ == "__main__":
    main_calculate_metrics()

def format_time(seconds):
    """Formatea el tiempo en minutos y segundos."""
    minutes = int(seconds // 60)
    seconds = int(seconds % 60)
    return f"{minutes} minutos y {seconds} segundos"

def parse_metrics_file(filename):
    """Lee Metricas.txt y organiza las métricas para graficar."""
    metrics_data = {}
    current_k = None
    current_sim = None
    with open(filename, "r") as f:
        lines = f.readlines()
        for line in lines:
            line = line.strip()
            if line.startswith("k_limite ="):
                current_k = int(line.split("=")[1].strip().strip(":"))
                metrics_data[current_k] = []
            elif line.startswith("Simulación"):
                current_sim = int(line.split()[1].strip(":"))
                metrics_data[current_k].append({'sim': current_sim, 'm
etrics': {}})
            elif line.startswith("Número de columnas con al menos un 0
:"):
                metrics_data[current_k][-1]['metrics']['stabilized_col
umns'] = int(line.split(": ")[1])
            elif line.startswith("Inter_D estabilizado:") and not line
.startswith("Inter_D estabilizado: Media ="):
                value = line.split(": ")[1]
                metrics_data[current_k][-1]['metrics']['inter_d_stabil
ized'] = float(value) if value != "Sin columnas estabilizadas" else No
ne
            elif line.startswith("Densidad estomática:") and not line.
startswith("Densidad estomática: Media ="):
                metrics_data[current_k][-1]['metrics']['stomatal_densi
ty'] = float(line.split(": ")[1])
            elif line.startswith("Media intra_d de la matriz:"):
                value = line.split(": ")[1]
                metrics_data[current_k][-1]['metrics']['intra_d_mean']
= float(value) if value != "N/A" else None
            elif line.startswith("Desviación estándar intra_d de la ma
triz:"):
                value = line.split(": ")[1]
                metrics_data[current_k][-1]['metrics']['intra_d_stdev'
] = float(value) if value != "N/A" else None
            elif line.startswith("Resumen para k_limite"):
                current_k = int(line.split("=")[1].strip(":"))
                metrics_data[current_k].append({'summary': {}})
            elif line.startswith("Columnas estabilizadas:"):
```

```
                parts = line.split(", ")
                metrics_data[current_k][-1]['summary']['columns_mean']
= float(parts[0].split("=")[1].strip())
                metrics_data[current_k][-1]['summary']['columns_stdev'
] = float(parts[1].split("=")[1].strip())
            elif line.startswith("Inter_D estabilizado: Media ="):
                parts = line.split(", ")
                metrics_data[current_k][-1]['summary']['inter_d_mean']
= float(parts[0].split("=")[1].strip())
                metrics_data[current_k][-1]['summary']['inter_d_stdev'
] = float(parts[1].split("=")[1].strip())
            elif line.startswith("Densidad estomática: Media ="):
                parts = line.split(", ")
                metrics_data[current_k][-1]['summary']['density_mean']
= float(parts[0].split("=")[1].strip())
                metrics_data[current_k][-1]['summary']['density_stdev'
] = float(parts[1].split("=")[1].strip())
            elif line.startswith("Intra_D: Media ="):
                parts = line.split(", ")
                metrics_data[current_k][-1]['summary']['intra_d_mean']
= float(parts[0].split("=")[1].strip())
                metrics_data[current_k][-1]['summary']['intra_d_stdev'
] = float(parts[1].split("=")[1].strip())

    return metrics_data

def plot_metric(k_values, means, stdevs, ylabel, title, filename, dist
_name, log=False, semilog=False):
    """Genera gráficas de métricas vs k_limite con recta de regresión
para log y semilog."""
    if not any(mean is not None for mean in means):
        print(f"No se generó la gráfica para {ylabel}: sin datos válid
os.")
        return
    fig_width = max(8, len(k_values) * 0.5)
    fig, ax = plt.subplots(figsize=(fig_width, 5))
    valid_k = [k for k, mean in zip(k_values, means) if mean is not No
ne]
    valid_means = [max(mean, 1e-10) if log else mean for mean in means
if mean is not None]
    valid_stdevs = [stdev for mean, stdev in zip(means, stdevs) if mea
n is not None]

    ax.errorbar(valid_k, valid_means, yerr=valid_stdevs, fmt='o-', col
or='blue', ecolor='lightblue', capsize=5, label=f'{ylabel} ± desv. est
.')

    if log or semilog:
        x_data = np.log10(valid_k) if log or semilog else np.array(val
id_k)
        y_data = np.log10(valid_means) if log else np.array(valid_mean
s)
```

```
        coeffs = np.polyfit(x_data, y_data, 1)
        slope, intercept = coeffs
        y_pred = slope * x_data + intercept

        y_mean = np.mean(y_data)
        ss_tot = np.sum((y_data - y_mean) ** 2)
        ss_res = np.sum((y_data - y_pred) ** 2)
        r_squared = 1 - (ss_res / ss_tot) if ss_tot != 0 else 0

        if log:
            x_plot = np.logspace(np.log10(min(valid_k)), np.log10(max(
valid_k)), 100)
            y_plot = 10 ** (slope * np.log10(x_plot) + intercept)
        else:
            x_plot = np.logspace(np.log10(min(valid_k)), np.log10(max(
valid_k)), 100)
            y_plot = slope * np.log10(x_plot) + intercept

        ax.plot(x_plot, y_plot, 'r--', label=f'Regresión: y = {slope:.
2f}x + {intercept:.2f}, R² = {r_squared:.3f}')

    ax.set_xlabel('k_limite', fontsize=12)
    ax.set_ylabel(ylabel, fontsize=12)
    ax.set_title(title.format(dist_name=dist_name), fontsize=14)
    ax.grid(True, linestyle='--', alpha=0.7)
    ax.legend(fontsize=10)
    ax.tick_params(axis='both', labelsize=8)
    if len(valid_k) > 20:
        ax.set_xticks(valid_k[::len(valid_k)//10 or 1])
    if log:
        ax.set_xscale('log')
        ax.set_yscale('log')
    elif semilog:
        ax.set_xscale('log')
    plt.savefig(f"Imágenes/{filename}", format='png', bbox_inches='tig
ht', dpi=150)
    plt.close(fig)

def generate_plots(metrics_data, parameters):
    """Genera gráficas a partir de Metricas.txt."""
    os.makedirs("Imágenes", exist_ok=True)
    dist_name = {1: "aleatoria", 2: "regular", 3: "gaussiana", 4: "esp
ecial"}
    dist_name = dist_name[parameters['distribution_type']]

    k_values = sorted(metrics_data.keys())
    density_means = []
    density_stdevs = []
    intra_d_means = []
    intra_d_stdevs = []
    columns_means = []
```

```
    columns_stdevs = []
    inter_d_means = []
    inter_d_stdevs = []

    for k in k_values:
        summary = metrics_data[k][-1]['summary']
        density_means.append(summary['density_mean'])
        density_stdevs.append(summary['density_stdev'])
        intra_d_means.append(summary['intra_d_mean'])
        intra_d_stdevs.append(summary['intra_d_stdev'])
        columns_means.append(summary['columns_mean'])
        columns_stdevs.append(summary['columns_stdev'])
        inter_d_means.append(summary['inter_d_mean'])
        inter_d_stdevs.append(summary['inter_d_stdev'])

    if any(mean is not None for mean in density_means):
        plot_metric(k_values, density_means, density_stdevs, 'Densidad
estomática',
                    'Densidad estomática vs k_limite (Distribución {di
st_name})',
                    'density_vs_k.png', dist_name)
        plot_metric(k_values, density_means, density_stdevs, 'Densidad
estomática (log)',
                    'Densidad estomática vs k_limite (log) (Distribuci
ón {dist_name})',
                    'density_vs_k_log.png', dist_name, log=True)
        plot_metric(k_values, density_means, density_stdevs, 'Densidad
estomática',
                    'Densidad estomática vs k_limite (semilog) (Distri
bución {dist_name})',
                    'density_vs_k_semilog.png', dist_name, semilog=Tru
e)

    if any(mean is not None for mean in intra_d_means):
        plot_metric(k_values, intra_d_means, intra_d_stdevs, 'Intra_d
media',
                    'Intra_d vs k_limite (Distribución {dist_name})',
                    'intra_d_vs_k.png', dist_name)
        plot_metric(k_values, intra_d_means, intra_d_stdevs, 'Intra_d
media (log)',
                    'Intra_d vs k_limite (log) (Distribución {dist_nam
e})',
                    'intra_d_vs_k_log.png', dist_name, log=True)
        plot_metric(k_values, intra_d_means, intra_d_stdevs, 'Intra_d
media',
                    'Intra_d vs k_limite (semilog) (Distribución {dist
_name})',
                    'intra_d_vs_k_semilog.png', dist_name, semilog=Tru
e)

    if any(mean is not None for mean in columns_means):
        plot_metric(k_values, columns_means, columns_stdevs, 'Columnas
```

```
estabilizadas',
                      'Columnas estabilizadas vs k_limite (Distribución
{dist_name})',
                      'columns_vs_k.png', dist_name)
        plot_metric(k_values, columns_means, columns_stdevs, 'Columnas
estabilizadas (log)',
                      'Columnas estabilizadas vs k_limite (log) (Distrib
ución {dist_name})',
                      'columns_vs_k_log.png', dist_name, log=True)
        plot_metric(k_values, columns_means, columns_stdevs, 'Columnas
estabilizadas',
                      'Columnas estabilizadas vs k_limite (semilog) (Dis
tribución {dist_name})',
                      'columns_vs_k_semilog.png', dist_name, semilog=Tru
e)

    if any(mean is not None for mean in inter_d_means):
        plot_metric(k_values, inter_d_means, inter_d_stdevs, 'Inter_D
estabilizado',
                      'Inter_D estabilizado vs k_limite (Distribución {d
ist_name})',
                      'inter_d_stabilized_vs_k.png', dist_name)
        plot_metric(k_values, inter_d_means, inter_d_stdevs, 'Inter_D
estabilizado (log)',
                      'Inter_D estabilizado vs k_limite (log) (Distribuc
ión {dist_name})',
                      'inter_d_stabilized_vs_k_log.png', dist_name, log=
True)
        plot_metric(k_values, inter_d_means, inter_d_stdevs, 'Inter_D
estabilizado',
                      'Inter_D estabilizado vs k_limite (semilog) (Distr
ibución {dist_name})',
                      'inter_d_stabilized_vs_k_semilog.png', dist_name,
semilog=True)

def write_info_file(parameters, output_file="Info.txt", total_time=Non
e):
    """Escribe los parámetros en Info.txt."""
    dist_name = {1: "aleatoria", 2: "regular", 3: "gaussiana", 4: "esp
ecial"}
    with open(output_file, "w") as f:
        f.write(f"Parámetros de la simulación (distribución {dist_name
[parameters['distribution_type']]})\n")
        for key, value in parameters.items():
            f.write(f"{key}: {value}\n")
        f.write(f"Tiempo total de generación de gráficas: {format_time
(total_time)}\n")

def main_generate_plots():
    """Función principal para generar gráficas a partir de Metricas.tx
t."""
    start_time = time.time()
```

```
    if not os.path.exists("Metricas.txt"):
        print("Error: El archivo Metricas.txt no existe.")
        return

    print("Generación de gráficas a partir de Metricas.txt.")

    parameters = {}
    with open("Metricas.txt", "r") as f:
        lines = f.readlines()
        header_line = lines[0].strip()
        match_dist = re.search(r'distribución (\w+)', header_line)
        if match_dist:
            dist_name = match_dist.group(1)
            dist_map = {"aleatoria": 1, "regular": 2, "gaussiana": 3,
"especial": 4}
            parameters['distribution_type'] = dist_map.get(dist_name,
4)
        else:
            print("Advertencia: No se pudo extraer distribution_type.
Usando 4 (especial).")
            parameters['distribution_type'] = 4

        param_line = lines[1].strip()
        param_parts = param_line.split(", ")
        for part in param_parts:
            key, value = part.split("=")
            key = key.split(": ")[1] if ": " in key else key
            try:
                parameters[key] = eval(value)
            except:
                parameters[key] = value

    metrics_data = parse_metrics_file("Metricas.txt")
    generate_plots(metrics_data, parameters)

    end_time = time.time()
    total_time = end_time - start_time
    write_info_file(parameters, total_time=total_time)

    print(f"Tiempo total de generación de gráficas: {format_time(total
_time)}")

if __name__ == "__main__":
    main_generate_plots()
```

A.3 – Código 3 – Gráficas

```
def format_time(seconds):
    """Formatea el tiempo en minutos y segundos."""
    minutes = int(seconds // 60)
    seconds = int(seconds % 60)
    return f"{minutes} minutos y {seconds} segundos"

def parse_metrics_file(filename):
    """Lee Metricas.txt y organiza las métricas para graficar."""
    metrics_data = {}
    current_k = None
    current_sim = None
    with open(filename, "r") as f:
        lines = f.readlines()
        for line in lines:
            line = line.strip()
            if line.startswith("k_limite ="):
                current_k = int(line.split("=")[1].strip().strip(":"))
                metrics_data[current_k] = []
            elif line.startswith("Simulación"):
                current_sim = int(line.split()[1].strip(":"))
                metrics_data[current_k].append({'sim': current_sim, 'm
etrics': {}})
            elif line.startswith("Número de columnas con al menos un 0
:"):
                metrics_data[current_k][-1]['metrics']['stabilized_col
umns'] = int(line.split(": ")[1])
            elif line.startswith("Inter_D estabilizado:") and not line
.startswith("Inter_D estabilizado: Media ="):
                value = line.split(": ")[1]
                metrics_data[current_k][-1]['metrics']['inter_d_stabil
ized'] = float(value) if value != "Sin columnas estabilizadas" else No
ne
            elif line.startswith("Densidad estomática:") and not line.
startswith("Densidad estomática: Media ="):
                metrics_data[current_k][-1]['metrics']['stomatal_densi
ty'] = float(line.split(": ")[1])
            elif line.startswith("Media intra_d de la matriz:"):
                value = line.split(": ")[1]
                metrics_data[current_k][-1]['metrics']['intra_d_mean']
= float(value) if value != "N/A" else None
            elif line.startswith("Desviación estándar intra_d de la ma
triz:"):
                value = line.split(": ")[1]
                metrics_data[current_k][-1]['metrics']['intra_d_stdev'
] = float(value) if value != "N/A" else None
            elif line.startswith("Resumen para k_limite"):
                current_k = int(line.split("=")[1].strip(":"))
```

```
                metrics_data[current_k].append({'summary': {}})
            elif line.startswith("Columnas estabilizadas:"):
                parts = line.split(", ")
                metrics_data[current_k][-1]['summary']['columns_mean']
= float(parts[0].split("=")[1].strip())
                metrics_data[current_k][-1]['summary']['columns_stdev'
] = float(parts[1].split("=")[1].strip())
            elif line.startswith("Inter_D estabilizado: Media ="):
                parts = line.split(", ")
                metrics_data[current_k][-1]['summary']['inter_d_mean']
= float(parts[0].split("=")[1].strip())
                metrics_data[current_k][-1]['summary']['inter_d_stdev'
] = float(parts[1].split("=")[1].strip())
            elif line.startswith("Densidad estomática: Media ="):
                parts = line.split(", ")
                metrics_data[current_k][-1]['summary']['density_mean']
= float(parts[0].split("=")[1].strip())
                metrics_data[current_k][-1]['summary']['density_stdev'
] = float(parts[1].split("=")[1].strip())
            elif line.startswith("Intra_D: Media ="):
                parts = line.split(", ")
                metrics_data[current_k][-1]['summary']['intra_d_mean']
= float(parts[0].split("=")[1].strip())
                metrics_data[current_k][-1]['summary']['intra_d_stdev'
] = float(parts[1].split("=")[1].strip())

    return metrics_data

def plot_metric(k_values, means, stdevs, ylabel, title, filename, dist
_name, log=False, semilog=False):
    """Genera gráficas de métricas vs k_limite con recta de regresión
para log y semilog."""
    if not any(mean is not None for mean in means):
        print(f"No se generó la gráfica para {ylabel}: sin datos válid
os.")
        return
    fig_width = max(8, len(k_values) * 0.5)
    fig, ax = plt.subplots(figsize=(fig_width, 5))
    valid_k = [k for k, mean in zip(k_values, means) if mean is not No
ne]
    valid_means = [max(mean, 1e-10) if log else mean for mean in means
if mean is not None]
    valid_stdevs = [stdev for mean, stdev in zip(means, stdevs) if mea
n is not None]

    ax.errorbar(valid_k, valid_means, yerr=valid_stdevs, fmt='o-', col
or='blue', ecolor='lightblue', capsize=5, label=f'{ylabel} ± desv. est
.')

    if log or semilog:
        x_data = np.log10(valid_k) if log or semilog else np.array(val
id_k)
```

```
        y_data = np.log10(valid_means) if log else np.array(valid_mean
s)

        coeffs = np.polyfit(x_data, y_data, 1)
        slope, intercept = coeffs
        y_pred = slope * x_data + intercept

        y_mean = np.mean(y_data)
        ss_tot = np.sum((y_data - y_mean) ** 2)
        ss_res = np.sum((y_data - y_pred) ** 2)
        r_squared = 1 - (ss_res / ss_tot) if ss_tot != 0 else 0

        if log:
            x_plot = np.logspace(np.log10(min(valid_k)), np.log10(max(
valid_k)), 100)
            y_plot = 10 ** (slope * np.log10(x_plot) + intercept)
        else:
            x_plot = np.logspace(np.log10(min(valid_k)), np.log10(max(
valid_k)), 100)
            y_plot = slope * np.log10(x_plot) + intercept

        ax.plot(x_plot, y_plot, 'r--', label=f'Regresión: y = {slope:.
2f}x + {intercept:.2f}, R² = {r_squared:.3f}')

    ax.set_xlabel('k_limite', fontsize=12)
    ax.set_ylabel(ylabel, fontsize=12)
    ax.set_title(title.format(dist_name=dist_name), fontsize=14)
    ax.grid(True, linestyle='--', alpha=0.7)
    ax.legend(fontsize=10)
    ax.tick_params(axis='both', labelsize=8)
    if len(valid_k) > 20:
        ax.set_xticks(valid_k[::len(valid_k)//10 or 1])
    if log:
        ax.set_xscale('log')
        ax.set_yscale('log')
    elif semilog:
        ax.set_xscale('log')
    plt.savefig(f"Imágenes/{filename}", format='png', bbox_inches='tig
ht', dpi=150)
    plt.close(fig)

def generate_plots(metrics_data, parameters):
    """Genera gráficas a partir de Metricas.txt."""
    os.makedirs("Imágenes", exist_ok=True)
    dist_name = {1: "aleatoria", 2: "regular", 3: "gaussiana", 4: "esp
ecial"}
    dist_name = dist_name[parameters['distribution_type']]

    k_values = sorted(metrics_data.keys())
    density_means = []
    density_stdevs = []
    intra_d_means = []
```

```
    intra_d_stdevs = []
    columns_means = []
    columns_stdevs = []
    inter_d_means = []
    inter_d_stdevs = []

    for k in k_values:
        summary = metrics_data[k][-1]['summary']
        density_means.append(summary['density_mean'])
        density_stdevs.append(summary['density_stdev'])
        intra_d_means.append(summary['intra_d_mean'])
        intra_d_stdevs.append(summary['intra_d_stdev'])
        columns_means.append(summary['columns_mean'])
        columns_stdevs.append(summary['columns_stdev'])
        inter_d_means.append(summary['inter_d_mean'])
        inter_d_stdevs.append(summary['inter_d_stdev'])

    if any(mean is not None for mean in density_means):
        plot_metric(k_values, density_means, density_stdevs, 'Densidad
estomática',
                    'Densidad estomática vs k_limite (Distribución {di
st_name})',
                    'density_vs_k.png', dist_name)
        plot_metric(k_values, density_means, density_stdevs, 'Densidad
estomática (log)',
                    'Densidad estomática vs k_limite (log) (Distribuci
ón {dist_name})',
                    'density_vs_k_log.png', dist_name, log=True)
        plot_metric(k_values, density_means, density_stdevs, 'Densidad
estomática',
                    'Densidad estomática vs k_limite (semilog) (Distri
bución {dist_name})',
                    'density_vs_k_semilog.png', dist_name, semilog=Tru
e)

    if any(mean is not None for mean in intra_d_means):
        plot_metric(k_values, intra_d_means, intra_d_stdevs, 'Intra_d
media',
                    'Intra_d vs k_limite (Distribución {dist_name})',
                    'intra_d_vs_k.png', dist_name)
        plot_metric(k_values, intra_d_means, intra_d_stdevs, 'Intra_d
media (log)',
                    'Intra_d vs k_limite (log) (Distribución {dist_nam
e})',
                    'intra_d_vs_k_log.png', dist_name, log=True)
        plot_metric(k_values, intra_d_means, intra_d_stdevs, 'Intra_d
media',
                    'Intra_d vs k_limite (semilog) (Distribución {dist
_name})',
                    'intra_d_vs_k_semilog.png', dist_name, semilog=Tru
e)
```

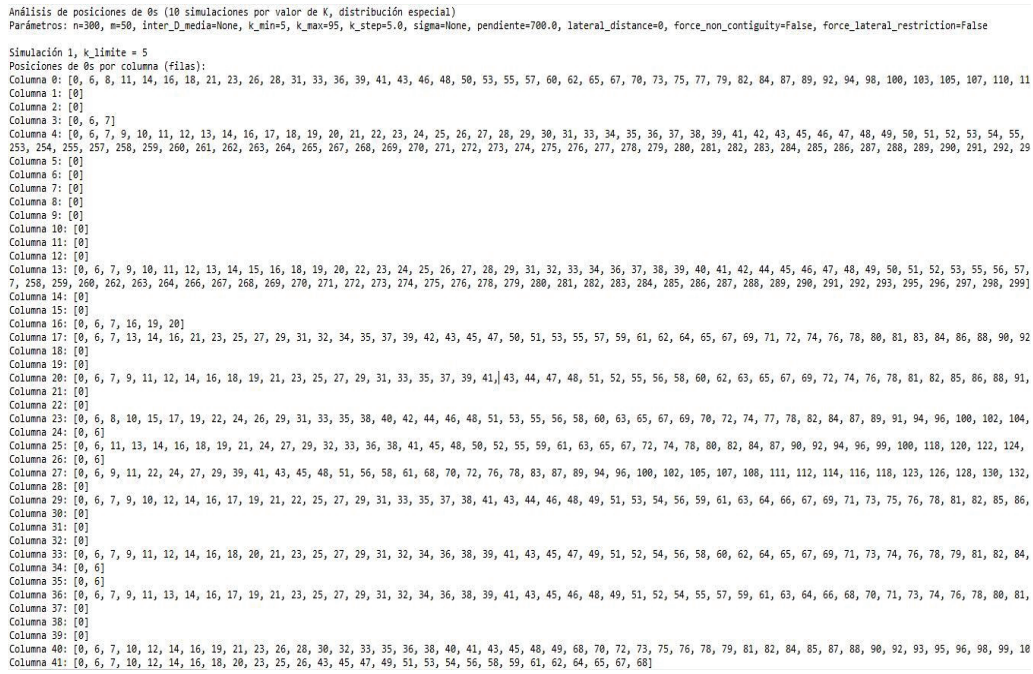
```
    if any(mean is not None for mean in columns_means):
        plot_metric(k_values, columns_means, columns_stdevs, 'Columnas
estabilizadas',
                    'Columnas estabilizadas vs k_limite (Distribución
{dist_name})',
                    'columns_vs_k.png', dist_name)
        plot_metric(k_values, columns_means, columns_stdevs, 'Columnas
estabilizadas (log)',
                    'Columnas estabilizadas vs k_limite (log) (Distrib
ución {dist_name})',
                    'columns_vs_k_log.png', dist_name, log=True)
        plot_metric(k_values, columns_means, columns_stdevs, 'Columnas
estabilizadas',
                    'Columnas estabilizadas vs k_limite (semilog) (Dis
tribución {dist_name})',
                    'columns_vs_k_semilog.png', dist_name, semilog=Tru
e)

    if any(mean is not None for mean in inter_d_means):
        plot_metric(k_values, inter_d_means, inter_d_stdevs, 'Inter_D
estabilizado',
                    'Inter_D estabilizado vs k_limite (Distribución {d
ist_name})',
                    'inter_d_stabilized_vs_k.png', dist_name)
        plot_metric(k_values, inter_d_means, inter_d_stdevs, 'Inter_D
estabilizado (log)',
                    'Inter_D estabilizado vs k_limite (log) (Distribuc
ión {dist_name})',
                    'inter_d_stabilized_vs_k_log.png', dist_name, log=
True)
        plot_metric(k_values, inter_d_means, inter_d_stdevs, 'Inter_D
estabilizado',
                    'Inter_D estabilizado vs k_limite (semilog) (Distr
ibución {dist_name})',
                    'inter_d_stabilized_vs_k_semilog.png', dist_name,
semilog=True)

def write_info_file(parameters, output_file="Info.txt", total_time=Non
e):
    """Escribe los parámetros en Info.txt."""
    dist_name = {1: "aleatoria", 2: "regular", 3: "gaussiana", 4: "esp
ecial"}
    with open(output_file, "w") as f:
        f.write(f"Parámetros de la simulación (distribución {dist_name
[parameters['distribution_type']]})\n")
        for key, value in parameters.items():
            f.write(f"{key}: {value}\n")
        f.write(f"Tiempo total de generación de gráficas: {format_time
(total_time)}\n")

def main_generate_plots():
    """Función principal para generar gráficas a partir de Metricas.tx
```

```
t."""
    start_time = time.time()

    if not os.path.exists("Metricas.txt"):
        print("Error: El archivo Metricas.txt no existe.")
        return

    print("Generación de gráficas a partir de Metricas.txt.")

    parameters = {}
    with open("Metricas.txt", "r") as f:
        lines = f.readlines()
        header_line = lines[0].strip()
        match_dist = re.search(r'distribución (\w+)', header_line)
        if match_dist:
            dist_name = match_dist.group(1)
            dist_map = {"aleatoria": 1, "regular": 2, "gaussiana": 3,
"especial": 4}
            parameters['distribution_type'] = dist_map.get(dist_name,
4)
        else:
            print("Advertencia: No se pudo extraer distribution_type.
Usando 4 (especial).")
            parameters['distribution_type'] = 4

        param_line = lines[1].strip()
        param_parts = param_line.split(", ")
        for part in param_parts:
            key, value = part.split("=")
            key = key.split(": ")[1] if ": " in key else key
            try:
                parameters[key] = eval(value)
            except:
                parameters[key] = value

    metrics_data = parse_metrics_file("Metricas.txt")
    generate_plots(metrics_data, parameters)

    end_time = time.time()
    total_time = end_time - start_time
    write_info_file(parameters, total_time=total_time)

    print(f"Tiempo total de generación de gráficas: {format_time(total
_time)}")

if __name__ == "__main__":
    main_generate_plots()
```

ANEXO B

FORMATO DE ARCHIVOS DE RESULTADOS

B.1 – Posiciones



```
Análisis de posiciones de 0s (10 simulaciones por valor de K, distribución especial)
Parámetros: n=300, m=50, inter_D_media=None, k_min=5, k_max=95, k_step=5.0, sigma=None, pendiente=700.0, lateral_distance=0, force_non_contiguity=False, force_lateral_restriction=False

Simulación 1, k_límite = 5
Posiciones de 0s por columna (filas):
Columna 0: [0, 6, 8, 11, 14, 16, 18, 21, 23, 26, 28, 31, 33, 36, 39, 41, 43, 46, 48, 50, 53, 55, 57, 60, 62, 65, 67, 70, 73, 75, 77, 79, 82, 84, 87, 89, 92, 94, 98, 100, 103, 105, 107, 110, 11
Columna 1: [0]
Columna 2: [0]
Columna 3: [0, 6, 7]
Columna 4: [0, 6, 7, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 33, 34, 35, 36, 37, 38, 39, 41, 42, 43, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
253, 254, 255, 257, 258, 259, 260, 261, 262, 263, 264, 265, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 29
Columna 5: [0]
Columna 6: [0]
Columna 7: [0]
Columna 8: [0]
Columna 9: [0]
Columna 10: [0]
Columna 11: [0]
Columna 12: [0]
Columna 13: [0, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 31, 32, 33, 34, 36, 37, 38, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 55, 56, 57,
7, 258, 259, 260, 262, 263, 264, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 295, 296, 297, 298, 299]
Columna 14: [0]
Columna 15: [0]
Columna 16: [0, 6, 7, 16, 19, 20]
Columna 17: [0, 6, 7, 13, 14, 16, 21, 23, 25, 27, 29, 31, 32, 34, 35, 37, 39, 42, 43, 45, 47, 50, 51, 53, 55, 57, 59, 61, 62, 64, 65, 67, 69, 71, 72, 74, 76, 78, 80, 81, 83, 84, 86, 88, 90, 92
Columna 18: [0]
Columna 19: [0]
Columna 20: [0, 6, 7, 9, 11, 12, 14, 16, 18, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 44, 47, 48, 51, 52, 55, 56, 58, 60, 62, 63, 65, 67, 69, 72, 74, 76, 78, 81, 82, 85, 86, 88, 91,
Columna 21: [0]
Columna 22: [0]
Columna 23: [0, 6, 8, 10, 15, 17, 19, 22, 24, 26, 29, 31, 33, 35, 38, 40, 42, 44, 46, 48, 51, 53, 55, 56, 58, 60, 63, 65, 67, 69, 70, 72, 74, 77, 78, 82, 84, 87, 89, 91, 94, 96, 100, 102, 104,
Columna 24: [0, 6]
Columna 25: [0, 6, 11, 13, 14, 16, 18, 19, 21, 24, 27, 29, 32, 33, 36, 38, 41, 45, 48, 50, 52, 55, 59, 61, 63, 65, 67, 72, 74, 78, 80, 82, 84, 87, 90, 92, 94, 96, 99, 100, 118, 120, 122, 124,
Columna 26: [0, 6]
Columna 27: [0, 6, 9, 11, 22, 24, 27, 29, 39, 41, 43, 45, 48, 51, 56, 58, 61, 68, 70, 72, 76, 78, 83, 87, 89, 94, 96, 100, 102, 105, 107, 108, 111, 112, 114, 116, 118, 123, 126, 128, 130, 132,
Columna 28: [0]
Columna 29: [0, 6, 7, 9, 10, 12, 14, 16, 17, 19, 21, 22, 25, 27, 29, 31, 33, 35, 37, 38, 41, 43, 44, 46, 48, 49, 51, 53, 54, 56, 59, 61, 63, 64, 66, 67, 69, 71, 73, 75, 76, 78, 81, 82, 85, 86,
Columna 30: [0]
Columna 31: [0]
Columna 32: [0]
Columna 33: [0, 6, 7, 9, 11, 12, 14, 16, 18, 20, 21, 23, 25, 27, 29, 31, 32, 34, 36, 38, 39, 41, 43, 45, 47, 49, 51, 52, 54, 56, 58, 60, 62, 64, 65, 67, 69, 71, 73, 74, 76, 78, 79, 81, 82, 84,
Columna 34: [0, 6]
Columna 35: [0, 6]
Columna 36: [0, 6, 7, 9, 11, 13, 14, 16, 17, 19, 21, 23, 25, 27, 29, 31, 32, 34, 36, 38, 39, 41, 43, 45, 46, 48, 49, 51, 52, 54, 55, 57, 59, 61, 63, 64, 66, 68, 70, 71, 73, 74, 76, 78, 80, 81,
Columna 37: [0]
Columna 38: [0]
Columna 39: [0]
Columna 40: [0, 6, 7, 10, 12, 14, 16, 19, 21, 23, 26, 28, 30, 32, 33, 35, 36, 38, 40, 41, 43, 45, 48, 49, 68, 70, 72, 73, 75, 76, 78, 79, 81, 82, 84, 85, 87, 88, 90, 92, 93, 95, 96, 98, 99, 10
Columna 41: [0, 6, 7, 10, 12, 14, 16, 18, 20, 23, 25, 26, 43, 45, 47, 49, 51, 53, 54, 56, 58, 59, 61, 62, 64, 65, 67, 68]
```

Figura B.1 – Encabezado y sección inicial de la información almacenada en el archivo *Posiciones.txt* obtenido por el Código 1.

B.2 – Métricas

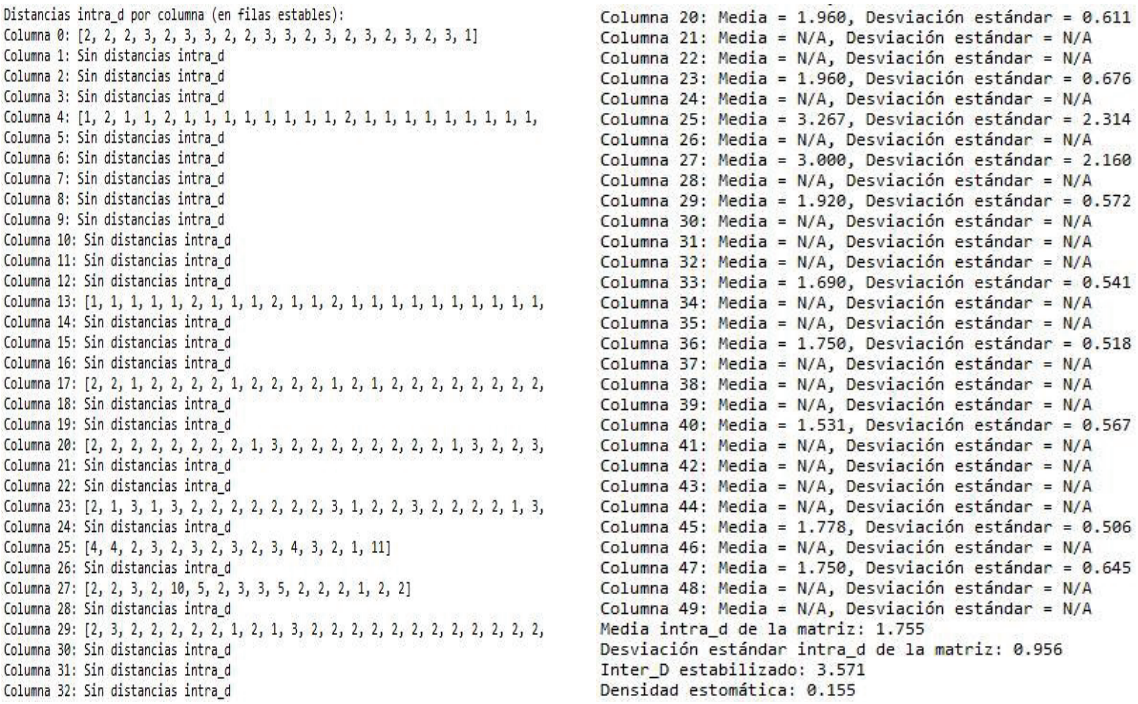


Figura B.2 – Secciones parciales de la información almacenada en el archivo *Metricas.txt* obtenido por el Código 2. Entre otras cosas, recoge también las desviaciones de cada valor para cada conjunto de simulaciones de un mismo *K* y de cada matriz

B.3 – Log de la simulación

```
Parámetros de la simulación (distribución especial)
distribution_type: 4
n: 300
m: 50
inter_D_media: None
k_min: 5
k_max: 95
k_step: 5.0
sigma: None
pendiente: 700.0
lateral_distance: 0
force_non_contiguity: False
force_lateral_restriction: False
stable_rows: 50
Tiempo total de generación de gráficas: 0 minutos y 22 segundos
```

Figura B.3 – Información recogida en el registro de la simulación *Info.txt*. El tiempo total de generación de gráficas se corresponde únicamente al tiempo que tarda en obtenerse los resultados del Código 3. Para el tiempo empleado en los demás códigos, sus respectivos archivos *ouput* cuentan también con un registro temporal.