Universidad Zaragoza
1542

Trabajo Fin de Grado

Estudio de las características mecánicas de la matriz extracelular en la formación de esferoides tumorales.

Study of the mechanical characteristics in the extracelular matrix in tumour spheroids formation.

Autor:

Alejandro Modrego Bravo

Directores:

José Manuel García Aznar

Francisco Merino Casallo

Escuela de Ingeniería y Arquitectura

Curso 2018/2019

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Alejandro Modrego Bravo,

con nº de DNI 73027310X en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, *por el que se aprueba el Reglamento de los TFG y TFM de la* Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Estudio de las características mecánicas de la matriz extracelular en la formación de esferoides tumorales,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 19 de Noviembre de 2019

Fdo: Alejandro Modrego Bravo

Agradecimientos

Antes de comenzar, me gustaría dar las gracias a mis dos tutores, José Manuel García y Francisco Merino, por el apoyo y toda la ayuda que me han brindado en la realización de este trabajo, además de toda la paciencia y la predisposición que han tenido en todo momento en su elaboración.

También me gustaría dar especial agradecimiento a Alfonso Caiazzo e Ignacio Ramis-Conde, autores del modelo computacional utilizado a lo largo de todo este trabajo.

Resumen

La migración celular en células cancerígenas es un proceso vital a la hora de la formación y proliferación de la masa tumoral. Los hidrogeles de colágeno son los más usados en ensayos *in vitro* debido a su biocompatibilidad y a su capacidad de reproducir las características de los tejidos humanos.

En el presente Trabajo de Fin de Grado (TFG) se va a tratar de replicar una serie de ensayos *in vitro* realizados con células tumorales de pulmón (línea H1299) en hidrogeles con base de colágeno mediante un modelo computacional. Este modelo computacional simula la migración celular en función de las características mecánicas de la matriz extracelular en la que se encuentran, como la rigidez, la viscosidad o el comportamiento elástico.

Se estudiará la precisión del modelo utilizado a la hora de simular diferentes casos en los que la proliferación celular es distinta debido a diferentes parámetros iniciales que determinan, directa o indirectamente, las características mecánicas de la matriz. Las simulaciones que se van a realizar en este estudio se centrarán en analizar los valores de tres características indicadoras de la proliferación de las células tumorales. Estas características que se van a medir son:

- Número de agregados celulares que se forman.
- Tamaño de estos agregados celulares.
- Velocidad de migración de las células y los agregados.

En los ensayos que se van a tratar de replicar, se estudia la influencia de la hormona TGF-β en los resultados. Obteniéndose unos resultados que indican que, en presencia de esta hormona, se facilita la proliferación celular.

Además, este modelo permite la visualización de los resultados iteración por iteración mediante un programa externo llamado Paraview, en este va a ser posible importar las simulaciones realizadas y observar la formación de los agregados celulares a través del tiempo.

Índice de contenido

Agradecimientos **2**

Resumen **3**

Capítulo 1. Introducción General **6**

1.1. Cáncer de pulmón 6

1.1.1. Información General 6

1.1.2. Fases del cáncer de pulmón 7

1.2. Migración celular en un cáncer 8

1.3. Hormona TGF-β 10

Capítulo 2. Antecedentes **11**

2.1. Ensayos de laboratorio 11

2.1. Modelo Computacional 12

Capítulo 3. Simulaciones del modelo computacional **16**

3.1. Parámetros de entrada 16

3.1.1. Parámetros fijos: 16

3.1.2. Parámetros variables: 17

3.2. Software de visualización de resultados Paraview 18

Capítulo 4. Resultados **20**

4.1. Número de clústers 20

4.2. Tamaños de clúster 23

4.2.1. Variación de concentraciones 23

4.2.2. Variación de tiempos. 27

4.3. Velocidades 27

4.4. Simulaciones TGF-β. 30

4.5. Análisis de sensibilidad 32

Capítulo 5. Conclusiones **37**

Capítulo 6. Bibliografía **38**

Capítulo 7. Anexos **40**

Anexo 1. Determinación de cada clúster. 42

Anexo 2. Principales resultados *in vitro* 40

Anexo 3. Posicionamiento de las N células iniciales. 44

Anexo 4. Cálculo de las dimensiones de cada clúster. 48

Anexo 5. Cálculo de velocidad efectiva de las células..... 51
Anexo 6. Edición del fichero de entrada. 52
Anexo 7. Lanzamiento de simulaciones en cola. 55
Anexo 8. Obtención de estadísticas..... 56
Anexo 9. Generación de gráficas. 62

Capítulo 1. Introducción General

1.1. Cáncer de pulmón

1.1.1. Información General

El cáncer de pulmón es uno de los tipos de cáncer más comunes a nivel internacional. En España, según la Asociación Española Contra el Cáncer [1], se detectan más de 28.600 casos nuevos al año.

Dentro del cáncer de pulmón existen dos grandes subgrupos, que se diferencian en función del tamaño de las células que lo forman [1]:

1. Carcinomas de células pequeñas o microcíticos.
   Conforman el 20% de los cánceres de pulmón. Estos se localizan en la parte central del pulmón, pudiendo adherirse a vasos importantes, comprimiéndolos de forma peligrosa. Son muy agresivos y su propagación es muy rápida.

2. Carcinomas no microcíticos.
   Representan el 80 % restante de los cánceres de pulmón. Éstos a su vez se distinguen en tres tipos distintos:
   (a) Carcinoma escamoso o epidermoide.
   Este carcinoma tiene un crecimiento lento en comparación a otros tipos de carcinomas, representa el 40% de los carcinomas no microcíticos y suele localizarse en la parte central de los pulmones.

   (b) Adenocarcinoma pulmonar.
   Representa el 30% de este tipo de carcinomas. Es el más frecuente en fumadores [1], pero el que menos tiene que ver con el consumo de tabaco.

   (c) Carcinoma de células grandes.
   Es el tipo menos frecuente de carcinoma, conforma el 10% restante de este tipo. Como su nombre indica, se caracteriza por estar formado por células de gran tamaño.

Además de estos dos tipos, en los pulmones se pueden encontrar células cancerígenas de otras líneas celulares provenientes de la metástasis de tumores localizados en otras regiones del cuerpo.

Por lo general, la mayoría de los cánceres de pulmón comienzan en las células que revisten los bronquios, los bronquiolos y los alvéolos [2]. Además, otra parte del pulmón que puede verse afectada por las células cancerígenas es la pleura, una membrana que recubre la cavidad torácica y los pulmones y cuyo cáncer es conocido como mesotelioma pleural [1].

La Figura 1 muestra la anatomía básica de los pulmones.

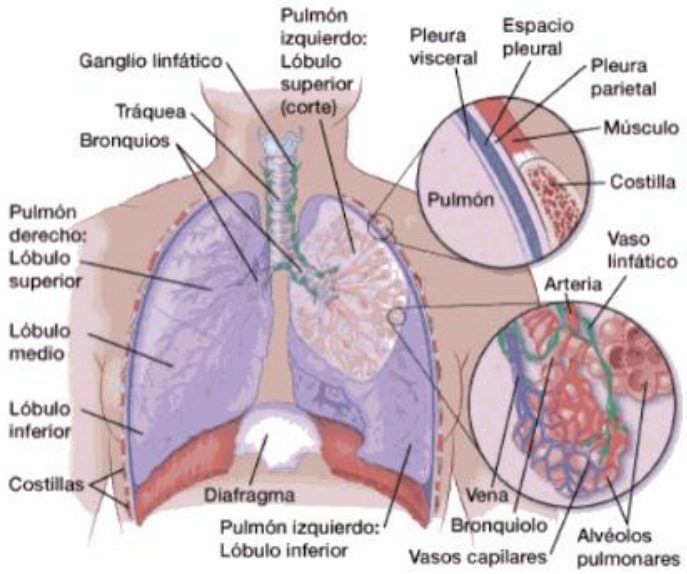

Figura 1. Anatomía básica de los pulmones [2]

1.1.2. Fases del cáncer de pulmón

Las fases de desarrollo de un cáncer de pulmón se recogen en el sistema TNM [1].

Estas siglas hacen referencia a tres factores clave para la evolución del cáncer:

T: Tamaño y localización del tumor.

N: Afectación de los ganglios linfáticos

M: Afectación de otros órganos.

Para cada uno de estos tres factores existen varios subfactores que determinan la gravedad del carcinoma.

La distinción de los diferentes tipos de tumores en función de sus características [1]:

Tumor primario (T):

T0: No hay signos de tumor primario.

TX: Cáncer oculto. Existen células malignas en las secreciones pulmonares, pero no hay otros datos de la existencia del tumor. Además, no se puede demostrar radiológicamente ni en la fibrobroncoscopia.

TIS: Carcinoma in situ.

T1: Tumor menor o igual a 3 cm de dimensión máxima, rodeado por tejido pulmonar o pleura, sin invasión al bronquio lobar.

T2: Tumor cuya dimensión máxima es mayor a 3 cm, o aquel de cualquier tamaño que invade la pleura o que genera una obstrucción de las vías respiratorias (atelectasia) o una neumonitis obstructiva.

T3: Tumor de cualquier tamaño, con extensión directa al diafragma, pleura mediastínica, pared costal o pericardio. No afecta al corazón, vasos grandes, tráquea, esófago o cuerpos cervicales.

T4: Tumor de cualquier tamaño que afecta al corazón, grandes vasos, tráquea, esófago o cuerpos cervicales.

Ganglios Linfáticos Regionales (N):

N0: Sin metástasis en los ganglios linfáticos.

N1: Metástasis en ganglios linfáticos peribronquiales o hiliares ipsilatrales, incluyendo la extensión directa del tumor.

N2: Metástasis en los ganglios mediastínicos o subcarinales ipsilaterales.

N3: Metástasis en los ganglios mediastínicos o hiliares contralaterales, escaleno ipsi o contralateral, o supraclaviculares.

Metástasis a Distancia (M):

M0: Sin metástasis a distancia conocidas.

M1: Metástasis a distancia presentes, especificando su localización (p. e. en el cerebro).

1.2. Migración celular en un cáncer

La migración celular es el proceso mediante el cual una célula se desplaza a través de los tejidos hacia sitios específicos dentro de un sistema.

En el desarrollo de un cáncer, la migración celular tiene un papel fundamental a la hora de la proliferación del propio tumor, siendo la matriz extracelular uno de los factores que más peso tiene en este proceso [3].

Para la realización de este trabajo se han estudiado varios experimentos realizados para comprobar la correlación que existe entre la migración celular y dos factores, las características de la matriz extracelular y la presencia de la hormona TGF-β. Las características principales de esta hormona se explican en el siguiente apartado.

La migración celular se hace más complicada a medida que aumenta la concentración de colágeno de la matriz extracelular, favoreciendo la formación de agregados celulares [3]. Además, en los ensayos en los que se incluye TGF-β, los clústers formados tienen una capacidad invasiva superior, observándose agregados celulares con una forma más alargada que favorece la migración.

En cuanto a la estructura tumoral, se contemplan dos hipótesis alternativas para explicar la aparición y el crecimiento tumoral, iniciándose como una migración de las células a otras zonas del tejido, siendo el resultado de esta migración un frente celular en forma de cerca o muro, formado por un gran número de células y cuya zona interna está compuesta por células en estado de anoxia.

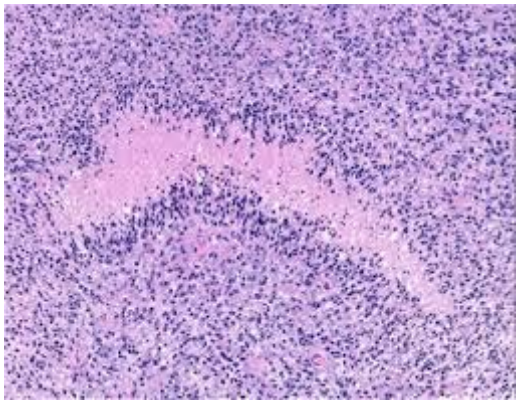

Figura 2. Estructura de cerca formada por la migración celular a distintas regiones del tejido [4].

A pesar de que estas dos hipótesis alternativas difieren en cuanto a las condiciones iniciales, ambas emplean mecánicas celulares muy similares [4].

La primera hipótesis considera que estas zonas necróticas de tejido se forman debido a una cantidad inusualmente alta de células en un espacio. En condiciones normales, las células utilizan un metabolismo aeróbico. Sin embargo, este aumento del número de células en un espacio determinado provoca que algunas de ellas no reciban suficiente oxígeno y sufran un cambio a un metabolismo anaeróbico. De forma prolongada, un metabolismo anaeróbico puede provocar la muerte celular, por este motivo, aquellas células con falta de oxígeno aumentan su capacidad invasiva en busca de zonas de tejido con una cantidad de oxígeno disponible mayor. Estas zonas con mayor cantidad de oxígeno son, normalmente vasos sanguíneos.

Por otra parte, la segunda hipótesis afirma que la formación de este frente celular se debe a un colapso en los vasos sanguíneos. En un inicio, una formación anormal de tejido nuevo crece alrededor de un vaso sanguíneo (neoplasia) y crece hasta un punto en el que ejerce una presión sobre el vaso sanguíneo que resulta en su colapso. En este momento, las células que estaban próximas al vaso sanguíneo mueren debido a una falta de oxígeno y la incapacidad de migrar, formando el centro de la neoplasia [5], [6], [7], [8].

1.3. Hormona TGF-β

Su nombre completo es Factor de Crecimiento Transformante-beta. Son un tipo de proteínas involucradas en varios procesos celulares como la proliferación celular, de especial interés en este trabajo.

Esta proteína tiene efectos muy diversos, incluso opuestos, dependiendo del tipo de tejido, del tipo de células a las que afecte o de la cantidad de hormona que se encuentre en el medio. Mientras que en la mayoría de las células epiteliales actúa como inhibidor del crecimiento, sobre células cancerígenas puede acelerar y facilitar la metástasis a otros tejidos. La presencia de esta hormona genera que los clústers que se forman en diferentes cultivos celulares obtengan una mayor movilidad, aumentando su capacidad invasiva. Esta proteína se relaciona con la neoplasia ósea, ya que se libera al reducir la cantidad de material óseo en una región.

En presencia de TGF-β se forman clústers con una forma más alargada (Figura 3). Sin embargo, en ausencia de esta hormona, los clústers formados tienen una forma más esférica (Figura 4) [3]. Además, la velocidad de las células cancerígenas afectadas por esta proteína es también más alta que en su ausencia.

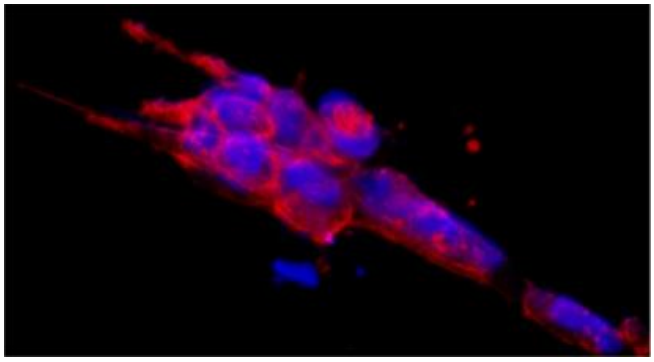

Figura 3. Clúster individual formado por células de la línea H1299 tras 7 días de crecimiento en una matriz de 4 mg/mL con TGF-β. En azul: núcleo celular. En rojo: actina [3]

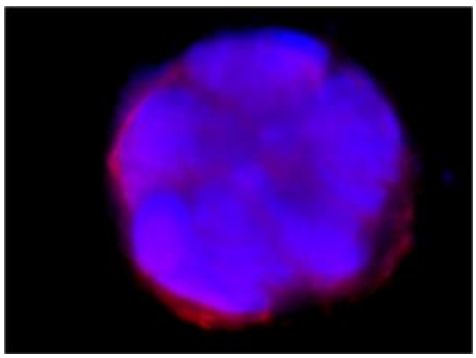

Figura 4. Clúster individual formado por células de la línea H1299 tras 7 días de crecimiento en una matriz de 4 mg/mL sin TGF-β. En azul: núcleo celular. En rojo: actina [3]

Capítulo 2. Antecedentes.

Como se ha mencionado anteriormente, este trabajo trata de simular de manera precisa el comportamiento de un tipo de célula tumoral asociada al cáncer de pulmón (línea H1299).

Mediante simulación se van a estudiar varias características que forman parte en la migración celular, así como su variación en función de las características de la matriz extracelular. Estas características son la velocidad de migración de las células, y el tamaño y número de los clústers formados.

Para la realización del trabajo se dispone del clúster de computación del grupo de investigación M2BE (Multiscale in Biological and Mechanical Engineering). Además, se va a utilizar un modelo computacional basado en mecánica de partículas [4]. A continuación, en el siguiente apartado, se va a proceder a la explicación de los conceptos clave.

La realización del presente Trabajo Fin de Grado viene motivada por la publicación de dos artículos relacionados con la proliferación celular, los cuales tratan sobre el estudio de este proceso en función de las propiedades del sustrato y su modelado computacional.

2.1. Ensayos de laboratorio

En Plou et al. [3], los autores realizan una serie de experimentos in vitro en hidrogeles con distinta concentración de colágeno, con y sin TGF-β, y analizan las diferencias relacionadas con el número de agregados celulares, su tamaño y velocidades de migración entre otros.

En estos ensayos se trabaja con hidrogeles a distinta concentración de colágeno (2.5, 4.0 y 6.0 mg/mL), y se analizan las diferencias en el desarrollo de células tumorales de pulmón (línea H1299).

Estos ensayos realizados en Plou et al. [3] estudian la variación de diferentes parámetros relacionados con la proliferación celular en función de las características mecánicas de la matriz extracelular. Estas propiedades mecánicas que sufren variaciones debido a los cambios en la concentración de colágeno son la viscosidad, la rigidez o el comportamiento elástico de la matriz.

Los datos obtenidos en [3] dan dos premisas clave a la hora de estudiar la migración celular. En primer lugar, con mayor concentración de colágeno se reduce la capacidad de las células para migrar. Como resultado, los agregados celulares que se observan tienen mayor tamaño conforme se aumenta la concentración, además de ser más numerosos. Por otro lado, la presencia de la hormona TGF-β genera una mayor movilidad de las células, facilitando la migración. Además, en presencia de esta hormona, la velocidad de migración de las células aumenta significativamente.

La velocidad de las células es mayor en concentraciones menores de colágeno, debido a que la densidad de la matriz es menor y, por lo tanto, las células tienen mayor facilidad para migrar a otras regiones de la matriz.

En el caso de tener una concentración de 2.5 mg/mL, las células se encuentran más dispersas debido a la mayor facilidad que poseen para la migración. Además, los agregados celulares que se forman no tienen un tamaño suficiente para considerarse clústers.

2.1. Modelo Computacional

Para simular los resultados principales de los experimentos in vitro realizados por Plou et al. [3], se utiliza el modelo computacional propuesto por Caiazzo y Ramis-Conde [4]. En dicho modelo se definen una serie de ecuaciones para simular las interacciones célula-célula y célula-matriz extracelular.

Dos procesos son los que determinan la evolución y proliferación de los organoides tumorales. El primero de ellos está asociado a la dinámica de las células. En particular, al cambio de fenotipo en función de la cantidad de oxígeno disponible. El segundo, está asociado a la difusión de oxígeno, tanto en el tejido sano como en el tejido tumoral, este último depende de la configuración del tumor y del consumo de nutrientes por parte de las células.

Se trata de un modelo multiescala. Por un lado, se incluye un modelo a nivel celular, considerando cada célula como una esfera viscoelástica que interactúa con las demás células mediante fuerzas intercelulares y con el medio que la rodea. Por otro lado, un modelo a nivel de tejido para definir la difusión de oxígeno en el medio.

Existe una interacción entre ambos modelos ya que la difusión de oxígeno depende de la estructura tumoral y de cómo se organicen las células. Esta unión se consigue mediante la simplificación de la distribución espacial de las células junto con la aplicación de una aproximación estacionaria para la difusión de oxígeno.

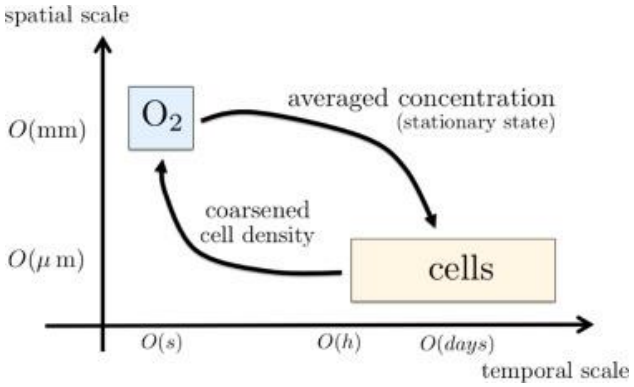

Figura 5. Mapa de diferenciación escalar célula-difusión de oxígeno [4]

Para obtener las ecuaciones que rigen el comportamiento del sistema, se considera principalmente el modelo de Hertz modificado [4], [9], a partir del cual se determinan las fuerzas célula-célula.

La ecuación que rige el movimiento celular es la siguiente:

$$\boldsymbol{\Gamma \dot{x}_i}(t) + \boldsymbol{a^r f_i}(t) = \sum_{j=1}^{Ncells(t)} \boldsymbol{F_{i,j}}(t) \quad (1)$$

En la ecuación (1) se incluyen los términos x, y, z. *"X"* corresponde al producto del tensor que determina la estructura de la matriz y la velocidad de las células, donde *i* es la célula que se está analizando. El término *"Y"* hace referencia a las fuerzas aleatorias que se dan debido a los mecanismos celulares en el proceso de exploración del espacio de la matriz por parte de las células. Donde α^r es la amplitud de las fuerzas aleatorias que ejerce la célula *i* sobre la matriz y $\boldsymbol{f_i}(t)$ es una función normal con media 0 y desviación unidad que determina el valor efectivo de las fuerzas. Por último, *"Z"* hace referencia al término $F_{i,j}$, que es la fuerza que ejercen las células colindantes *j* sobre la célula *i*.

La ecuación (2) define $F_{i,j}$ como una combinación de las fuerzas atractivas y repulsivas entre células.

$$\boldsymbol{F_{i,j} = F_{ij}^{rep} - F_{ij}^{adh}} \quad (2)$$

Por otro lado, la ecuación (3) determina la fuerza atractiva o de adhesión como:

$$\boldsymbol{F_{ij}^{adh}} = \alpha S_{ij}^{adh} \quad (3)$$

Donde α es una constante de adhesión que se mide en N/m^2 y, en segundo lugar, el área de contacto de una célula sobre otra, para la cual se calcula la media entre el área externa de la célula con una altura h_{ij} y el área del círculo que conforma esa altura h_{ij} (Figura 6), esta se obtiene mediante la expresión (4):

$$S_{ij}^{adh} = \frac{1}{2}(2\pi R h_{ij} + \pi(2R - h_{ij})h_{ij}) \quad (4)$$

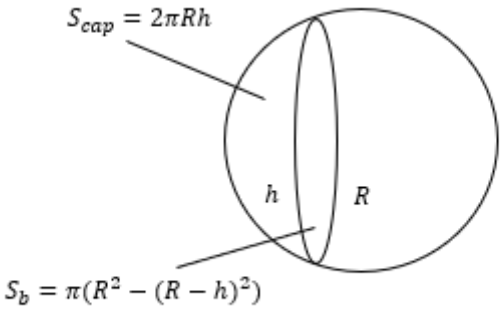

Figura 6. Superficies que se computan en la adhesión celular. [4]

De forma que, tras desarrollar la expresión del área de contacto, se obtiene la ecuación (5), que determina la fuerza de adhesión:

$$\boldsymbol{F_{i,j}^{adh}} = 2\pi\alpha(r_i - \frac{h_{ij}}{4})h_{ij} \quad (5)$$

Las fuerzas repulsivas entre células se calculan mediante la ecuación (6):

$$\boldsymbol{F_{i,j}^{rep}} = \frac{2}{3}(\frac{E}{1-\upsilon^2})h_{ij}^{\frac{3}{2}}\sqrt{\frac{r_i r_j}{r_i+r_j}} \quad (6)$$

De las ecuaciones (5) y (6), se obtiene la ecuación (7) para calcular la fuerza de interacción intercelular:

$$\boldsymbol{F_{i,j}} = \frac{2}{3}\left(\frac{E}{1-\upsilon^2}\right)h_{ij}^{\frac{3}{2}}\sqrt{\frac{r_i r_j}{r_i+r_j}} - 2\pi\alpha(r_i - \frac{h_{ij}}{4})h_{ij} \quad (7)$$

Por último, la ecuación (8) permite calcular la posición de cada célula del sistema en cualquier instante temporal.

$$x_i\left(t^{(n+1)}\right) = x_i(t^n) + \frac{\Delta t_{cells}}{\gamma}(-\boldsymbol{f_i}(t^n) + \sum_{j=1}^{N_{cells}(t^n)}\boldsymbol{F_{i,j}}(t^n) \quad (8)$$

Donde Δt_{cells} representa el tiempo de iteración del modelo de dinámica celular.

La ecuación (9) define las fuerzas repulsivas de las células, que dependen del coeficiente de Poisson y del Módulo de Young de estas.

$$\boldsymbol{F_{ij}^{rep}} = \frac{2}{3}(\frac{E}{1-\nu^2})h_{ij}^{\frac{3}{2}}\sqrt{\frac{r_i r_j}{r_i+r_j}} \quad (9)$$

En el presente trabajo, se aplica el modelo estocástico propuesto por Caiazzo y Ramis-Conde [4], que tiene en cuenta el carácter aleatorio de los procesos biológicos. Esto supone que, para una combinación de parámetros dada, se pueden obtener resultados distintos. Por lo tanto, es necesario realizar un análisis estadístico de los resultados obtenidos, prestando atención a las posibles tendencias del sistema.

Las células del modelo se dividen mediante mitosis, aunque no podrán dividirse en el caso de que se ejerza sobre ellas una fuerza superior a un cierto umbral computado por el modelo.

Además, el modelo contempla tres fenotipos celulares distintos en función de la cantidad de oxígeno disponible para cada célula. Estos tres fenotipos son:

1. **Normoxia**: Los niveles de oxígeno disponibles permiten a la célula mantener su metabolismo aeróbico. Este fenotipo se mantiene hasta que la concentración de oxígeno se sitúa por debajo de un valor umbral (< 7 mm Hg).

2. Hipoxia: Los niveles de oxígeno disponibles son inferiores a los necesarios por la célula para su metabolismo aeróbico (< 0.7 mm Hg). Al no disponer de oxígeno suficiente entra en un estado de metabolismo anaeróbico que, tras un tiempo prolongado (> 2880 s), resulta en la muerte de la célula.

3. Muerte: Tras un tiempo en estado de hipoxia, la célula entra en un estado de anoxia. Su movimiento se produce por consecuencia de las fuerzas mecánicas entre células.

Capítulo 3. Simulaciones del modelo computacional

Empleando el modelo de proliferación celular propuesto por Caiazzo y Ramis-Conde [4] se va a proceder al lanzamiento de varias simulaciones empleando distintas combinaciones de los parámetros de entrada. Como se ha mencionado anteriormente, el objetivo de este trabajo es el de replicar los principales resultados de Plou et al. [3], relativos a experimentos in vitro de proliferación celular y formación de clústers celulares.

3.1. Parámetros de entrada

En primer lugar, el modelo incluye una serie de parámetros de entrada que establecen las características principales tanto de la matriz extracelular como de las propias células. Estos parámetros definen la evolución del sistema a lo largo del ensayo simulado para los distintos escenarios analizados.

Dentro de estos parámetros se pueden diferenciar dos grupos. Por un lado, aquellos que permanecen fijos en todas las simulaciones que se realizan en el estudio. Y por otro lado, los parámetros que varían en las simulaciones, y con los cuales se obtienen datos estadísticos para la comparación con los ensayos in vitro [3]. Además, estos parámetros también están divididos en dos grupos, referentes a las células y referentes a la geometría.

3.1.1. Parámetros fijos:

CÉLULAS		Referencias
Ratio crecimiento	0.1	[4], [10]
Ratio nacimiento	1e-3	[4]
Modulo Young, E (kPa $\mu N/\mu m^2$)	1e-2	[11], [12], [13]
Coeficiente de adhesión, α (N/m^2)	3.1e-5	[4], [12], [13], [14]
Tiempo de iteración (min)	1	[3]
Modulo Poisson, υ	0.5	[11], [12], [13]
Ratio GCM	1	
Límite normoxia → hipoxia ($mm\ Hg$)	7	[7]
Límite hipoxia → muerte ($mm\ Hg$)	0.7	[7]
Ratio de consumo oxígeno hipoxia (s^{-1})	0.	[4]
Número de Iteraciones	1440-7200	[3]

Tabla 1. Parámetros fijos referentes a las células del modelo.

GEOMETRÍA		Referencias
Cajas eje X	180	
Cajas eje Y	180	
Cajas eje Z	90	
Longitud X (μm)	1800	[4]
Longitud Y (μm)	1800	[4]
Longitud Z (μm)	900	[4]
Células máximas	20000	
Dimensión	3	

Tabla 2. Parámetros fijos referentes a la geometría del modelo.

3.1.2. Parámetros variables:

CÉLULAS INICIALES	Rango	Referencias
Viscosidad de la matriz / *GCM* (γ) ($\mu N\ min/\mu m$)	0.3-0.75	[10]
Amplitud de fuerzas intracelulares, α^r (μN)	0.065-0.075	[15]
Posiciones iniciales X (μm)		
Posiciones iniciales Y (μm)		
Posiciones iniciales Z (μm)		

Tabla 3. Parámetros variables del modelo.

Los parámetros fijos son:

- **Coeficientes de nacimiento y crecimiento**: Determinan la velocidad con la que nacen y se desarrollan las células. Cuanto mayor sean esos dos valores, mayor será la proliferación de las células.
- **Coeficiente de Adhesión**: Es utilizado junto con el área de contacto entre células computada por el modelo para calcular las fuerzas de adhesión. Su valor depende del tipo de célula simulada.
- **Factor multiplicador de la viscosidad de la matriz**: Este parámetro es útil en el caso de trabajar con matrices no homogéneas en las que la viscosidad varía en función de la dirección.
- **Datos referentes al consumo de oxígeno**: Los datos de consumo de oxígeno determinan los rangos de concentración de oxígeno que corresponden a un fenotipo u otro. Todas las células en el instante inicial de simulación se encuentran en estado de normoxia.

Referentes a la geometría del modelo:

- **Número de Cajas**: Número de divisiones en los tres ejes cartesianos que hace el modelo en la matriz para determinar de forma más eficiente la localización de cada una de las células y controlar su población.

- **Dimensiones del dominio**: Longitud en los tres ejes cartesianos de la matriz sobre la que se simula el desarrollo celular.

Por último, se añaden a parámetros fijos los valores de tiempo y número de iteraciones, éstos varían en algunas simulaciones, pero no influyen en la migración celular de la misma forma que los parámetros variables que se verán a continuación. Las variaciones en estos parámetros dependerán de si se requiere una simulación con el proceso más detallado o, por el contrario, como es el caso del presente trabajo, interesan en mayor medida los datos finales, por tanto, el tiempo de iteración puede ser mayor.

Los parámetros variables nos permiten simular los distintos escenarios analizados en el trabajo de Plou et al. [3]: hidrogeles con distintas concentraciones de colágeno (2.5, 4.0 y 6.0 mg/mL).

Los parámetros variables más importantes son: la viscosidad de la matriz, que determina la fricción que ejerce sobre las células (*GCM*); y la amplitud de las fuerzas aleatorias que ejercen las células sobre la matriz. Los valores de ambos parámetros son los que determinan la migración de las células simuladas, así como el número y tamaño de clústers que se formen en ella.

3.2. Software de visualización de resultados Paraview

El modelo computacional empleado en el presente trabajo genera una serie de ficheros *VTK* para facilitar la visualización y análisis del caso simulado. Para ello, se ha optado por utilizar *Paraview*, una aplicación multiplataforma de código abierto para la visualización científica. De este modo es posible realizar una visualización de cómo evoluciona el sistema simulado con el paso de las iteraciones.

Para ello, en primer lugar, se carga el fichero en el que aparecen guardados los datos de la matriz o caja y aplicarlo.

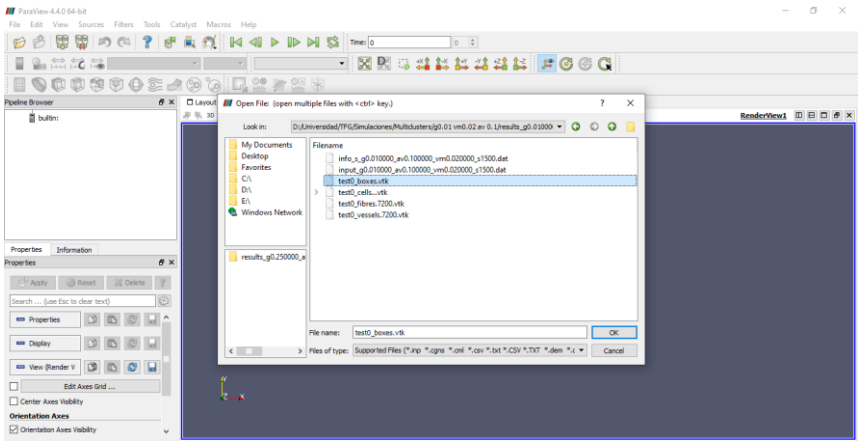

Figura 7. Ejemplo de importación de datos de la matriz a Paraview.

Una vez se han importado los ficheros *VTK* a *Paraview*, se modifica el modo de visualización de este archivo a *"Wireframe"* para poder analizar lo que sucede dentro del dominio del sistema.

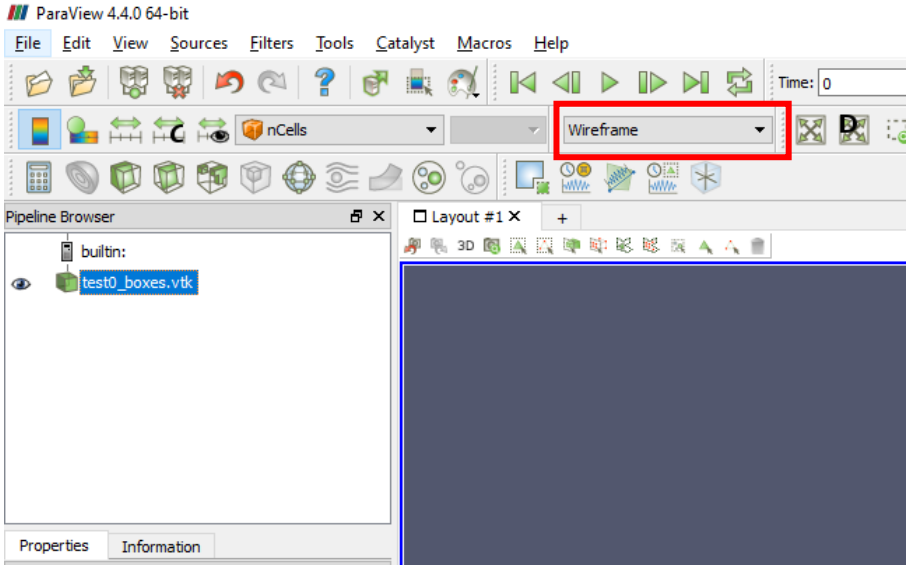

Figura 8. Modificación del modo de visualización del archivo de la matriz.

A continuación, se importan los ficheros *VTK* asociados a las células. En este caso habría tantos ficheros *VTK* como iteraciones indique el parámetro de entrada correspondiente.

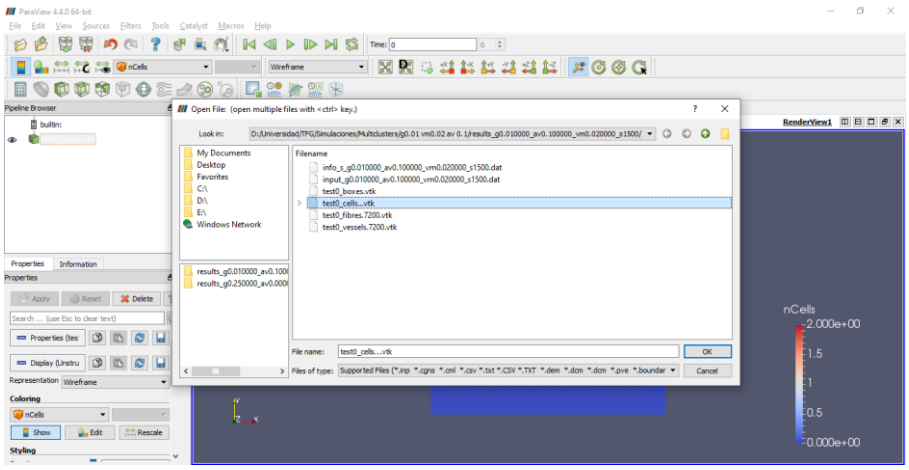

Figura 9. Ejemplo de importación de los archivos correspondientes a las células simuladas.

Por último, se añade un filtro *"Glyph"* sobre el conjunto de ficheros *VTK* asociado a las células. De esta forma, se representa cada célula como una esfera, lo que facilita su posterior análisis de la evolución del sistema simulado. Mediante los controles correspondientes (Play, Pause…) se puede controlar la evolución del sistema.

Capítulo 4. Resultados

Una vez definidos los parámetros de entrada de datos, se va a proceder a las simulaciones para el estudio del comportamiento de las células ante variaciones en la matriz extracelular.

Como se ha mencionado en apartados anteriores, el objetivo de este análisis es el de replicar los principales resultados del trabajo de Plou et al. [3] sobre la proliferación de clústers en distintos sustratos.

Para simular las distintas matrices de colágeno analizadas en los experimentos in vitro que se emplean como referencia [3] es necesario utilizar los datos que aparecen en la Tabla 4.

DATOS	2.5 mg/ml	4 mg/ml	6 mg/ml
Fricción ($\mu N\ min/\mu m$)	0.3	0.7	0.75
Fuerzas aleatorias (μN)	0.065	0.065	0.75
Adhesión (N/m^2)	3.1e-5	3.1e-5	3.1e-5

Tabla 4. Recopilación de los parámetros variables en función del caso.

4.1. Número de clústers

En este apartado se va a analizar el tamaño de los clústers celulares que aparecen en los distintos sustratos considerados en Plou et al. [3] en función de la concentración de colágeno (matrices de 2.5, 4.0 y 6.0 mg/mL). Los parámetros de entrada empleados en las distintas simulaciones son los indicados en la Tabla 4.

A pesar de la gran influencia que tiene el número de células iniciales en el número de clústers que aparecen al final de los ensayos, se va a realizar una comparativa entre los resultados obtenidos por Plou et al. [3] en experimentos in vitro y las simulaciones realizadas en este trabajo.

En el caso del número de clústers, lo que se puede observar en los ensayos in vitro de Plou et al. [3] para una matriz de 2.5 mg/mL es un número de clústers bajo con relación a los otros dos valores de concentración de colágeno. Esto se debe a la poca resistencia que opone la matriz en la migración celular en esta concentración.

El segundo caso tratado corresponde a una matriz de 4 mg/mL de concentración de colágeno, según Plou et al. [3], el número de clústers es mayor en concentraciones más altas. Esto se debe a que las células, al tener una mayor dificultad para migrar que en la matriz del caso anterior, están menos dispersas por la matriz, concentrándose en zonas más concretas que en el caso anterior.

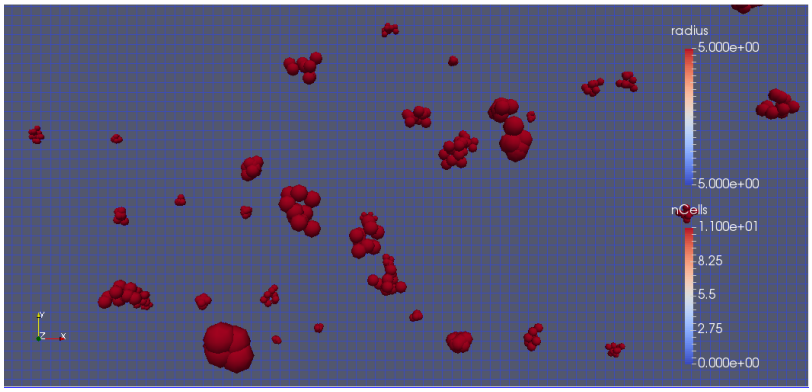

Figura 10. Visualización En Paraview de una simulación lanzada para una matriz de 4 mg/mL tras 5 días.

En las simulaciones lanzadas se confirma esta premisa, apareciendo un mayor número de clústers.

En el último caso de este apartado se simula una matriz de 6.0 mg/mL, según el artículo [3], el resultado esperado es un mayor número de agregados formados por mayor número de células, esto es debido al aumento de la densidad de la matriz. Este aumento de densidad se observa en la prácticamente nula existencia de células solitarias que hayan migrado. Las simulaciones corroboran las observaciones de Plou et al. [3] en experimentos in vitro: un aumento en el número de agregados en geles a 6.0 mg/mL.

A continuación, en las Figuras 11, 12 y 13, se muestran las comparativas para las distintas concentraciones estudiadas entre las simulaciones y los ensayos in vitro [3].

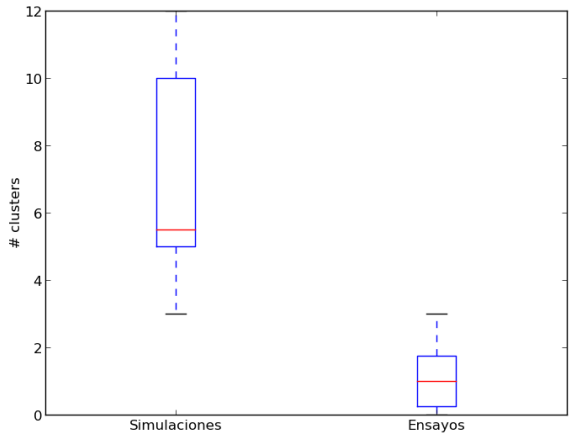

Figura 11. Comparativa de los resultados obtenidos relativos al número de clústers celulares tanto en experimentos in vitro [3] como en las simulaciones realizadas para una matriz a 2.5 mg/mL.

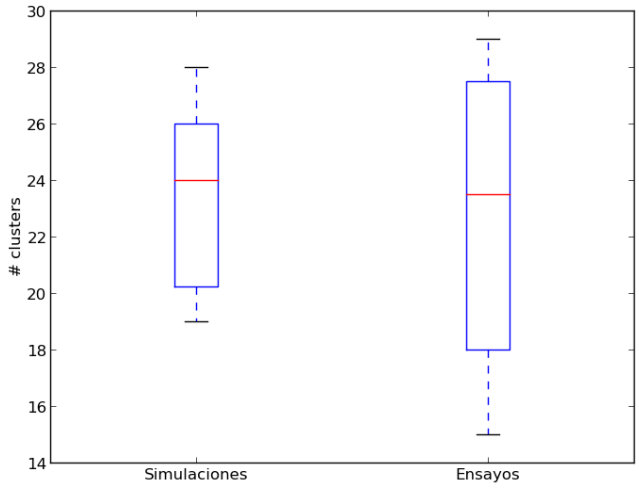

Figura 12. Comparativa de los resultados obtenidos relativos al número de clústers celulares tanto en experimentos in vitro [3] como en las simulaciones realizadas para una matriz a 4.0 mg/mL.

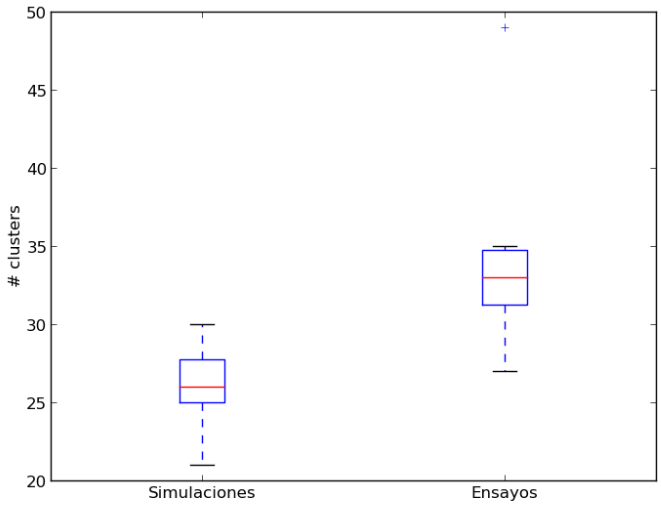

Figura 13. Comparativa de los resultados obtenidos relativos al número de clústers celulares tanto en experimentos in vitro [3] como en las simulaciones realizadas para una matriz a 6.0 mg/mL.

En la Figura 14 se observan gráficamente los resultados de las simulaciones para las distintas condiciones de la matriz. Se obtiene una tendencia similar a la descrita en los ensayos de Plou et al. [3].

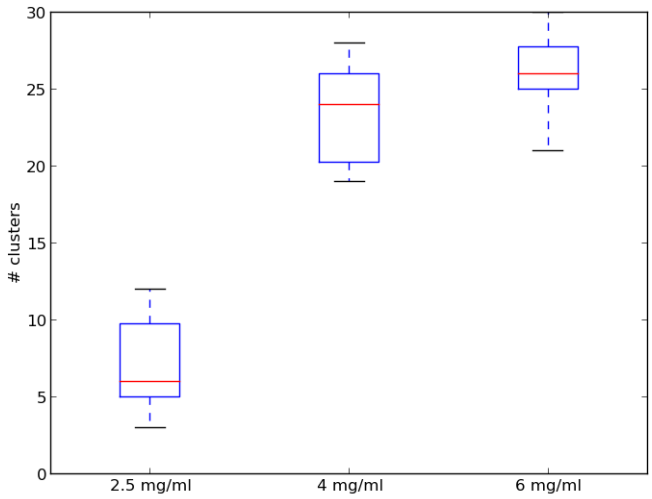

Figura 14. Recopilación estadística de los resultados para el número de clústers obtenidos en las simulaciones

Como se ha mencionado en el inicio de este apartado, el número de clústers depende en gran medida del número de células iniciales. Por lo tanto, un mayor número de células iniciales supone un aumento en el número de clústers.

4.2. Tamaños de clúster

4.2.1. Variación de concentraciones

Para determinar el área de los clústers que aparecen en las distintas simulaciones se ha implementado un algoritmo en C++. Dicho algoritmo determina qué célula forma parte de cada clúster y calcula el diámetro de cada uno de ellos en los tres ejes cartesianos, seleccionando entre ellos el valor máximo.

A lo largo de todas las simulaciones, los diámetros en cada uno de los tres ejes son muy similares para cada matriz respectivamente, lo que quiere decir que los clústers formados tienen forma casi esférica. Estos resultados coinciden con los asociados a los experimentos realizados en Plou et al. [3].

Una vez el diámetro máximo de cada uno de los clústers es determinado, se calcula el área de cada clúster mediante la ecuación (10).

$$Área\ clúster = \pi * radio^2 \qquad (10)$$

El análisis relativo a las áreas de los clústers no tendrá en cuenta aquellos con un área inferior a 1000 μm^2 para replicar el análisis de Plou et al. [3].

El primer caso simulado es el de la proliferación celular en matrices de 2.5 mg/mL. Según los ensayos realizados en Plou et al. [3], para esta concentración, el tamaño de clúster es muy pequeño debido a la baja resistencia de la matriz al movimiento de las células y, por tanto, a su migración. Esto provoca que las células se encuentren dispersas por todo el dominio y, por lo tanto, se obtenga un número muy reducido de clústers con un área superior a 1000 μm^2. Debido a esto, no se dan valores experimentales para la matriz en este caso.

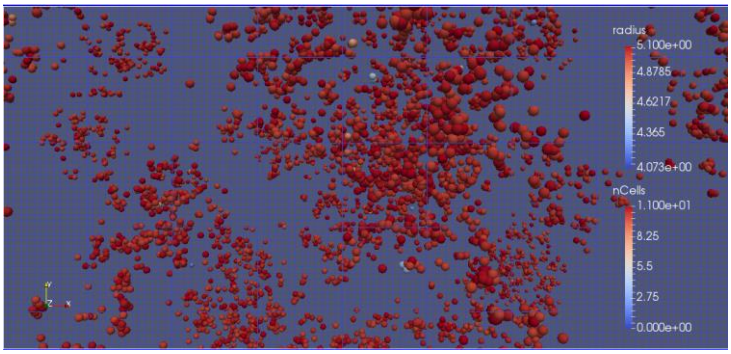

Figura 15. Visualización en Paraview de una simulación tras 120 horas en una matriz de 2.5 mg/mL.

En una matriz de 4 mg/mL de concentración, lo que indican los ensayos en el artículo [3] es la formación de algunos clústers de forma más visible, aunque continúan apareciendo clústers de pequeño tamaño e incluso células solitarias distribuidas por la matriz en menor medida. Las células iniciales, al crecer y dividirse, crean agregados de células de un tamaño mayor que en el caso anterior.

Por último, en una matriz de 6.0 mg/mL la oposición que ejerce el sustrato al movimiento de las células a través de ella es mayor que en los dos casos anteriores. El aumento de densidad implica un aumento de la resistencia por parte de la matriz a la migración de las células.

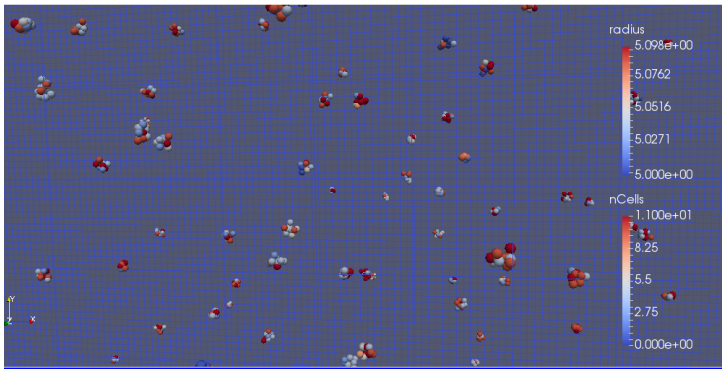

Figura 16. Visualización en Paraview de una simulación tras 120 horas en una matriz de 6.0 mg/mL.

Al tener una mayor dificultad para migrar a otras zonas, se encuentran clústers más formados y grandes y menos células dispersas. Por lo tanto, los clústers formados tienen una mayor densidad de células.

A continuación se muestran los resultados obtenidos para las tres concentraciones: 2.5, 4.0 y 6.0 mg/mL, respectivamente en las Figuras 17, 18 y 19.

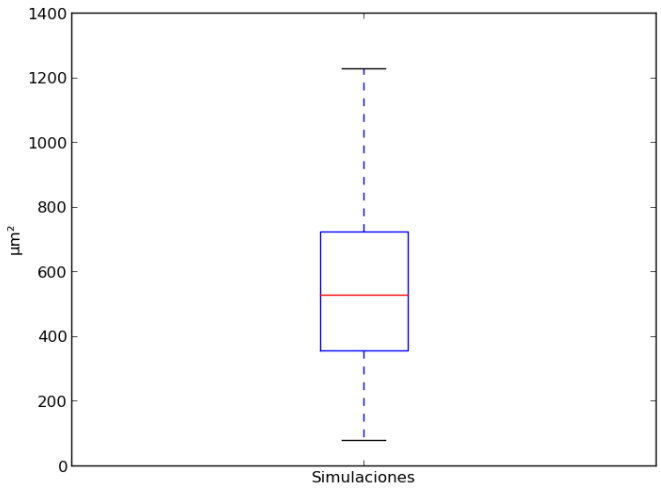

Figura 17. Resultados obtenidos relativos a tamaños de clúster en simulaciones para una matriz de 2.5 mg/mL.

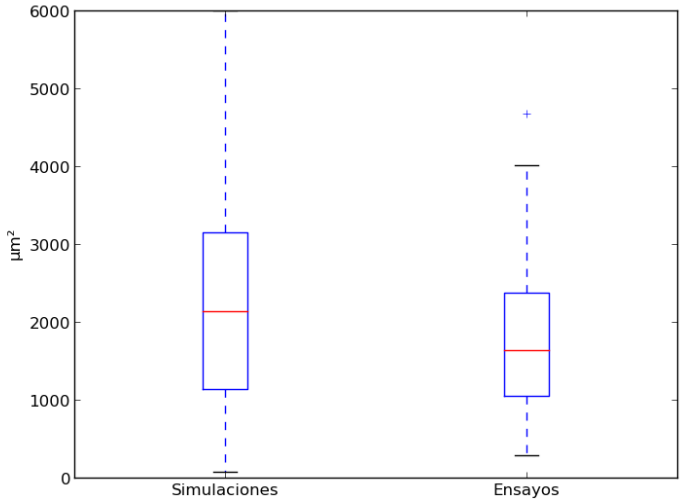

Figura 18. Comparativa de los resultados obtenidos relativos al tamaño de los clústers celulares tanto en experimentos in vitro [3] como en las simulaciones realizadas para una matriz a 4.0 mg/mL.

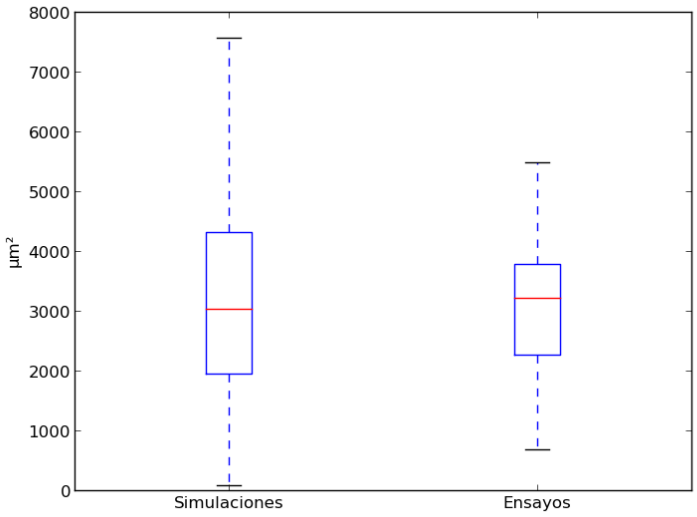

Figura 19. Comparativa de los resultados obtenidos relativos al tamaño de los clústers celulares tanto en experimentos in vitro [3] como en las simulaciones realizadas para una matriz a 6.0 mg/mL.

Como recopilación de los casos simulados en este apartado, la conclusión es que, como se afirma en los ensayos de Plou et al. [3], el tamaño de los clústers depende directamente de la concentración de la matriz, para concentraciones más altas, la fricción matriz-célula es mayor y, por tanto, las células forman clústers más grandes.

En la Figura 20 se puede ver comparativamente la variación entre las áreas proyectadas de los distintos clústers en los 3 casos simulados.

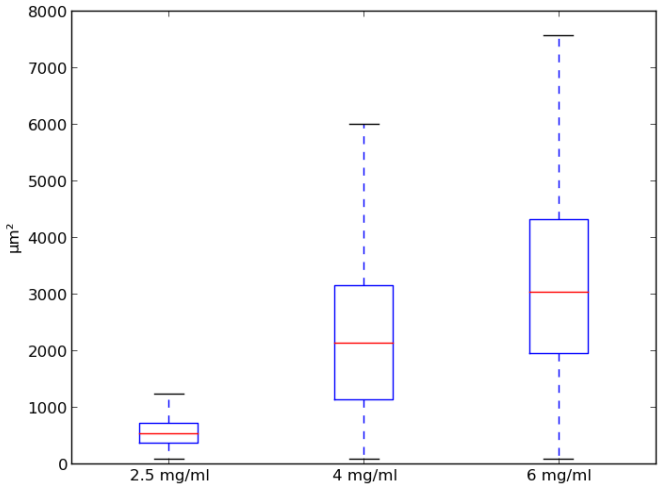

Figura 20. Comparativa de los valores obtenidos referentes a las áreas en los 3 casos simulados.

4.2.2. Variación de tiempos.

Para comprobar las diferencias existentes relacionadas con el tiempo de simulación, se estudia también la variación en las áreas de los clústers en función del tiempo, para cada concentración de matriz. La finalidad de este análisis es observar, de forma similar a los resultados dados en el artículo de Plou et al. [3], las variaciones en el tamaño de los clústers formados tras distintos tiempos.

En este apartado se realizan simulaciones para tres tiempos de simulación diferentes: 24, 72 y 120 horas. Este último tiempo es el utilizado para obtener todos los resultados a lo largo de este trabajo.

Según los resultados obtenidos en Plou et al. [3], a medida que el tiempo de simulación es mayor, los valores para el número de clústers y tamaño de estos aumenta. Esto es debido a que las células tienen más tiempo para nacer y proliferar.

Tal y como muestra la Figura 21, los resultados obtenidos en las simulaciones corroboran los resultados de los ensayos in vitro [3].

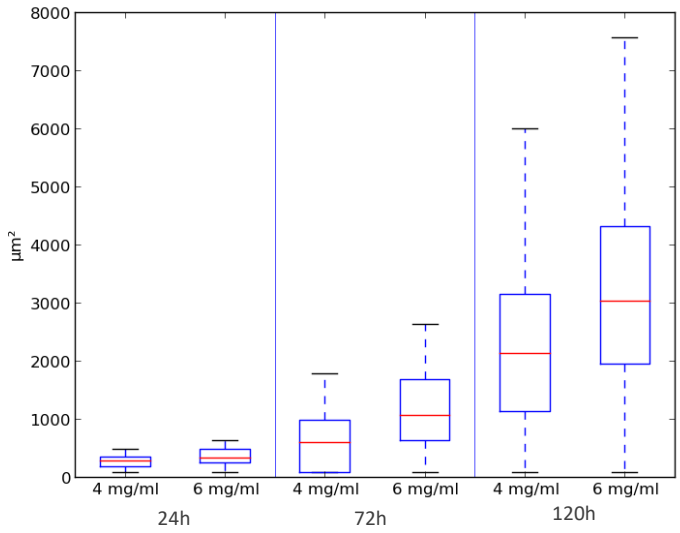

Figura 21. Resumen de la variación de áreas de clúster en función de la concentración de la matriz y del tiempo de desarrollo.

4.3. Velocidades

Respecto a las diferentes velocidades que se obtienen al realizar las simulaciones, en el artículo de Plou et al. [3] se dan las gráficas en relación a las velocidades efectivas de cada una de las células dependiendo de las concentraciones de la matriz. A diferencia de las gráficas para las otras dos características que se han visto anteriormente, las velocidades simuladas se obtienen tras las primeras 48 horas de simulación.

Se observa que, a medida que se aumenta la concentración, las velocidades de las células disminuyen. Esto es debido a que, al ejercer la matriz una mayor fricción sobre las células frena el movimiento de estas, ocasionando así una disminución de la velocidad. Los resultados obtenidos en las simulaciones confirman la tendencia obtenida en los ensayos in vitro de Plou et al. [3], habiendo obtenido unas velocidades de clúster parecidas a las obtenidas en los ensayos.

Mientras que en los ensayos para matrices de 2.5 mg/mL se obtienen velocidades efectivas con una mediana de $0.028\ \mu m/min$, en las simulaciones se obtienen valores bastante similares, aunque con una variabilidad menor, con una mediana de $0.026\ \mu m/min$.

El segundo caso es el correspondiente a la matriz de 4.0 mg/mL. Como se puede observar en la comparativa, las velocidades efectivas obtenidas en las simulaciones tienen una mediana de $0.09\ \mu m/min$, mientras que en los experimentos in vitro [3] la mediana es de $0.1\ \mu m/min$. Se observa una reducción de la mediana de las velocidades efectivas en las células, debido a la mayor oposición de la matriz al movimiento de las células.

En el último caso, para una concentración de 6.0 mg/mL de colágeno en la matriz, se puede observar una disminución de la velocidad efectiva de las células debido al aumento de fricción que supone el aumento de concentración de colágeno. Los datos recogidos en las simulaciones para estas condiciones indican una mediana de valor $0.08\ \mu m/min$, mientras que en los experimentos in vitro [3] se obtiene una mediana de $0.0079\ \mu m/min$, valores inferiores a los obtenidos en los dos casos anteriores.

A continuación, se pueden observar las comparativa entre los resultados obtenidos en los ensayos [3] y los obtenidos en las simulaciones en las Figuras 22, 23 y 24.

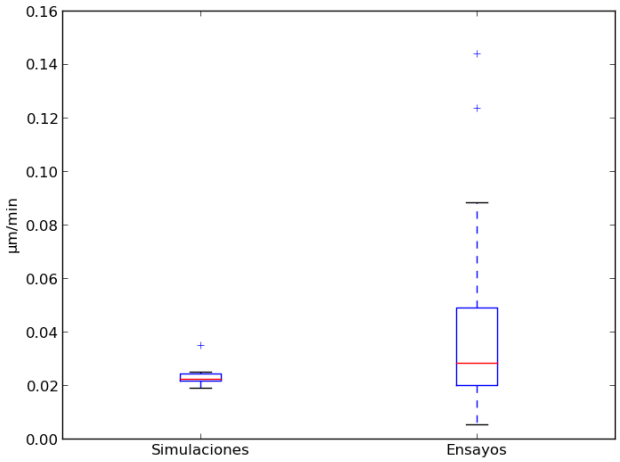

Figura 22. Comparativa de los resultados obtenidos relativos a las velocidades efectivas de las células tanto en experimentos in vitro [3] como en las simulaciones realizadas para una matriz a 2.5 mg/mL.

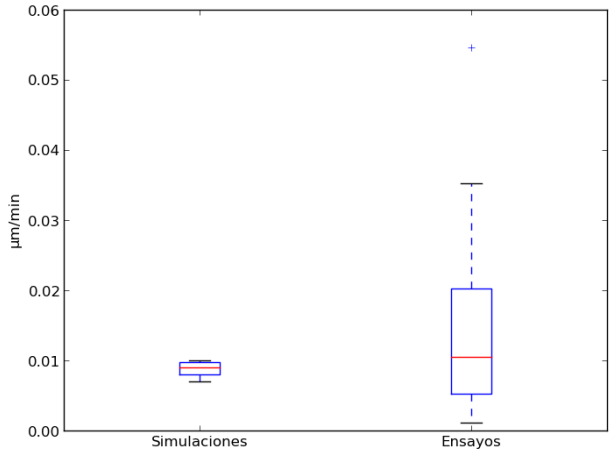

Figura 23. Comparativa de los resultados obtenidos relativos a las velocidades efectivas de las células tanto en experimentos in vitro [3] como en las simulaciones realizadas para una matriz a 4.0 mg/mL.

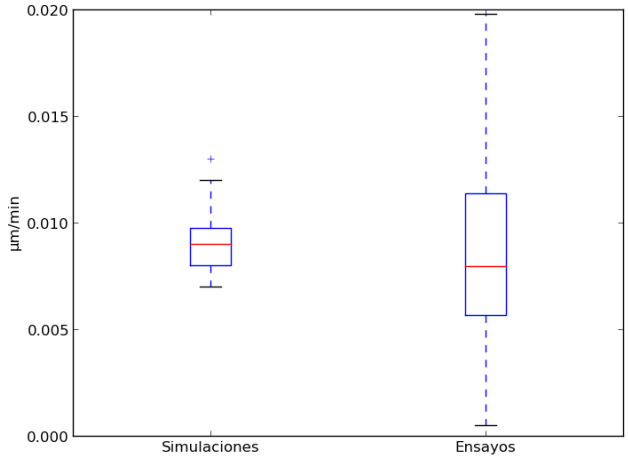

Figura 24. Comparativa de los resultados obtenidos relativos a las velocidades efectivas de las células tanto en experimentos in vitro [3] como en las simulaciones realizadas para una matriz a 6.0 mg/mL.

Como se puede observar a lo largo de los resultados obtenidos en las simulaciones en este apartado, a medida que la densidad de la matriz aumenta, al dificultar la movilidad de las células a través del medio, disminuye la velocidad de estas. En la Figura 25 se puede observar esta disminución de la velocidad al aumentar la densidad de colágeno de la matriz.

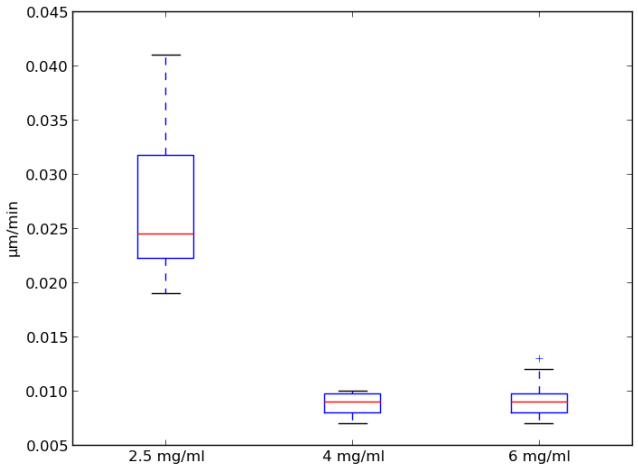

Figura 25. Comparativa de velocidades para distintas concentraciones de colágeno.

A lo largo de los resultados mostrados en este capítulo se han obtenido valores muy similares a los mostrados en Plou et al. [3], sin embargo, se puede observar que, por lo general, los resultados obtenidos en simulaciones tienen una variabilidad menor. Esto es debido a que, en el caso de los ensayos in vitro [3] las propiedades de la matriz no son exactamente las mismas a lo largo del dominio, sino que pueden ir variando. En el caso de las simulaciones, se trabaja con un valor promedio para cada uno de los parámetros, considerando que los valores de estos parámetros no varían a lo largo del dominio de la matriz. Es por ello que la variabilidad es menor en las simulaciones que en los ensayos in vitro [3].

4.4. Simulaciones TGF-β.

En este apartado se va a proceder a estudiar cómo afecta la hormona TGF-β en la velocidad de migración de las células. Según los resultados en los ensayos de Plou et al. [3], las velocidades en las matrices de 2.5 y 6.0 mg/mL se reducen en comparación a las obtenidas en ensayos sin la presencia de TGF-β, mientras que la velocidad en la matriz de 4.0 mg/mL aumenta.

En primer lugar, para simular la presencia de TGF-β en el modelo, se disminuirá el valor de la constante de adhesión del modelo, dado que el principal papel de esta hormona en la proliferación celular es disminuir esta adhesión celular. En la Tabla 5 se muestran los datos que se van a utilizar para las simulaciones de los tres casos de matrices con TGF-β.

DATOS	2.5 mg/mL	4.0 mg/mL	6.0 mg/mL
Fricción ($\mu N\ min/\mu m$)	0.3	0.7	0.75
Fuerzas aleatorias (μN)	0.065	0.065	0.75
Adhesión (N/m^2)	1e-5	1e-5	1e-5

Tabla 5. Resumen de parámetros en simulaciones con TGF-β.

En primer lugar se simulará cómo afecta en el modelo esta reducción de adhesión en una matriz de 2.5 mg/mL. En la Figura 26 se pueden observar los resultados obtenidos en las simulaciones, se da una disminución en la velocidad de las células debida a esta reducción de adhesión.

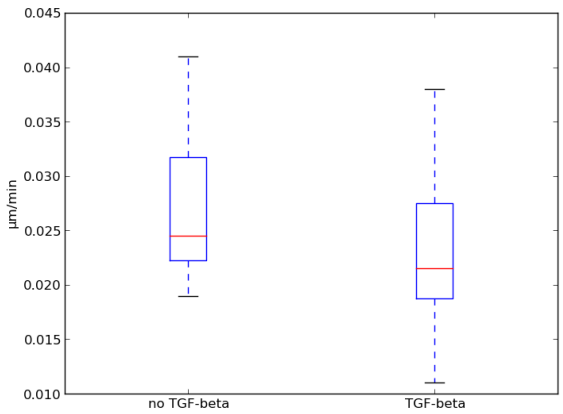

Figura 26. Comparativa de la velocidad de migración celular en una matriz de 2.5 mg/mL sin y con TGF-β.

En el caso de una matriz de 4.0 mg/mL, la velocidad obtenida en los ensayos *in vitro* en Plou et al. [3] aumenta ligeramente. En los resultados obtenidos en las simulaciones, sin embargo, la mediana de las velocidades aumenta en menor medida que en los resultados expuestos por Plou et al. [3]. El valor de media obtenido en las simulaciones aumenta de la misma forma que en los ensayos *in vitro*. Estos resultados obtenidos se pueden ver en la Figura 27.

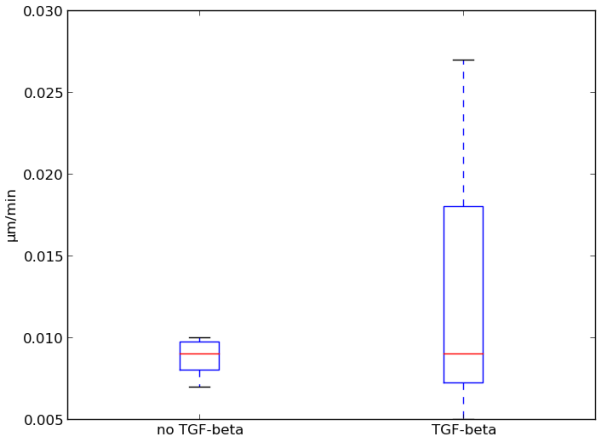

Figura 27. Comparativa de la velocidad de migración celular en una matriz de 4 mg/mL sin y con TGF-β.

Por último, se realizan las simulaciones para la matriz de 6.0 mg/mL. En los ensayos de Plou et al. [3] la tendencia que sigue esta matriz es similar a la observada en el primer caso, para la matriz de 2.5 mg/mL. Los resultados obtenidos por en simulaciones reflejan esta misma tendencia, la reducción de la adhesión celular provocada por la presencia de TGF-β tiene como resultado una disminución en la velocidad de migración de las células. La Figura 28 muestra los resultados obtenidos en las simulaciones.

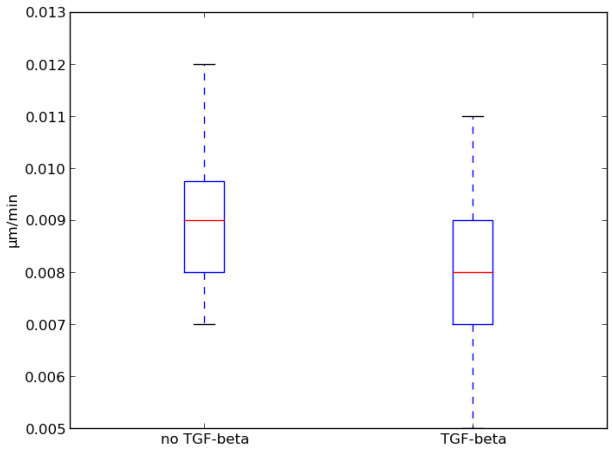

Figura 28. Comparativa de la velocidad de migración celular en una matriz de 6 mg/mL sin y con TGF-β.

4.5. Análisis de sensibilidad

En este apartado se va a realizar un análisis de sensibilidad del modelo. A lo largo de este capítulo se han mostrado los resultados de las simulaciones con unos parámetros mostrados en la Tabla 4 con intención de replicar los ensayos de Plou et al. [3]. A continuación se va a tomar como punto de partida el caso de una matriz de 2.5 mg/mL y se van a lanzar simulaciones variando cada uno de los tres parámetros más importantes que se han tratado en este trabajo de forma individual, para observar cómo varían los resultados obtenidos incrementando y disminuyendo estos parámetros.

Los parámetros que se van a variar en este apartado son la viscosidad de la matriz (γ), la adhesión celular (α) y la amplitud de las fuerzas aleatorias que realizan las células sobre la matriz (α^r).

Partiendo de los parámetros para una matriz de 2.5 mg/mL, se van a incrementar y reducir en un 20 y un 50%.

El primer parámetro que se va a analizar es la viscosidad de la matriz, γ. El valor de este parámetro para una matriz de 2.5 mg/mL es de 0.3 $\mu N\ min/\mu m$.

En la Figura 29 se muestra la tendencia de los tamaños de los agregados formados en función de este parámetro γ.

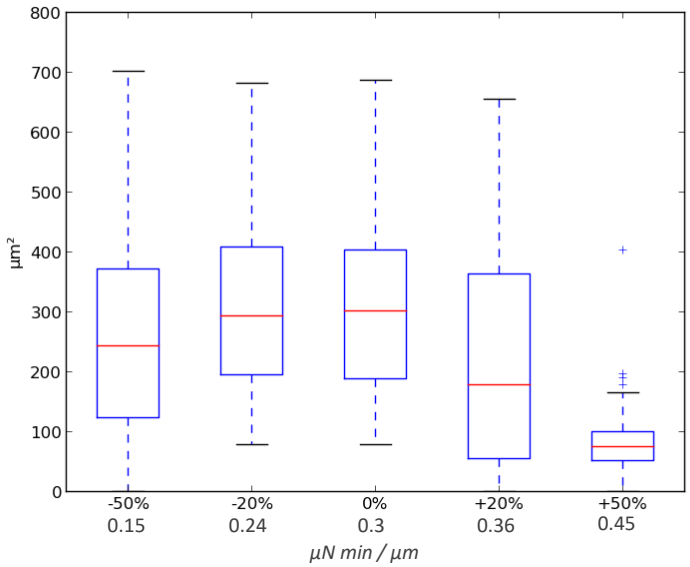

Figura 29. Tendencia del tamaño de los agregados celulares ante la variación de la viscosidad de la matriz (γ).

A continuación, en la Figura 30, se muestra la tendencia que siguen la velocidad de migración de las células ante la variación de la fricción ejercida por la matriz (γ).

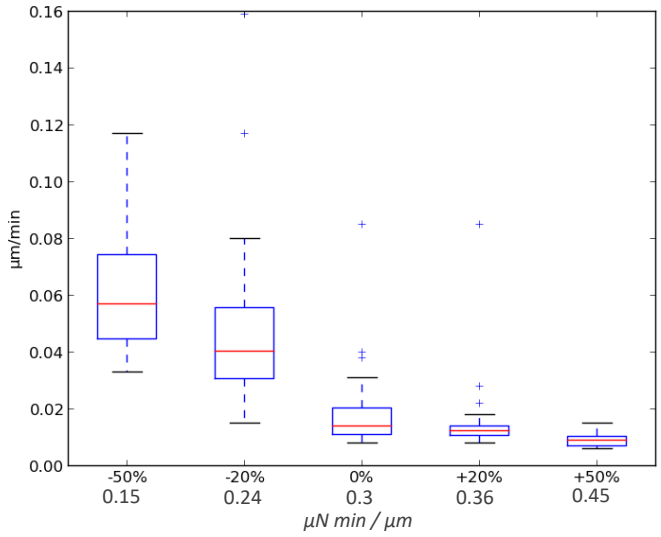

Figura 30. Tendencia de la velocidad de migración de las células ante la variación de la viscosidad de la matriz (γ).

Se observa un descenso de la velocidad de migración conforme la fricción ejercida por la matriz simulada aumenta. Al incrementarse este parámetro γ, la matriz ejerce una resistencia mayor al movimiento de las células, resultando en un descenso de la velocidad de estas.

El siguiente análisis de sensibilidad que se va a estudiar es el referente al incremento y reducción de la adhesión celular α. Este parámetro, como se ha explicado anteriormente, determina, junto al área de contacto, las fuerzas de adhesión que se ejercen entre las células. El valor base para este análisis es el que aparece en la Tabla 4, de *3.1e-5 N/m*^{2}.

De forma similar al caso anterior, se va a incrementar y reducir este parámetro en un 20 y un 50% de su valor de base, estudiando como varían los resultados de velocidad de migración y tamaño de los agregados celulares del modelo. Se puede observar que no existe una variación sustancial en los resultados referentes al tamaño de los agregados celulares. Esto es debido a que el valor de este parámetro α es muy pequeño en su valor base, de forma que al incrementarlo o reducirlo en los porcentajes comentados anteriormente no existe un gran cambio.

En la Figura 31 se puede observar esta tendencia que sigue el tamaño de los agregados con un aumento del coeficiente de adhesión.

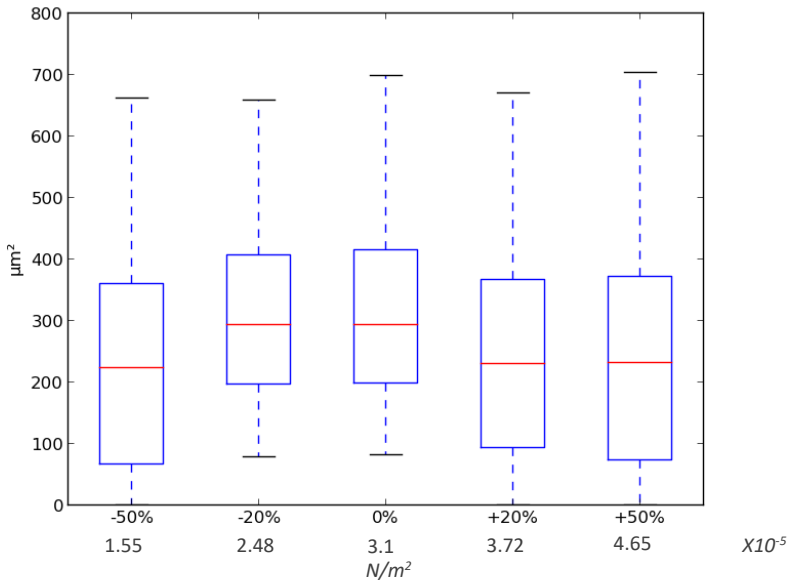

Figura 31. Tendencia del tamaño de los agregados celulares ante la variación de la adhesión celular (α).

Además, también se va a estudiar la tendencia de la velocidad de las células en función de este coeficiente α. A continuación, en la Figura 32 se muestra la tendencia de la velocidad de las células ante incrementos y reducciones en la adhesión celular. Se observa una fluctuación sobre unos valores concretos de velocidad, esto es debido a que, de forma similar al caso del tamaño de los agregados, el valor de la adhesión es muy pequeño, no existen cambios sustanciales al incrementar o reducir este parámetro en los porcentajes vistos.

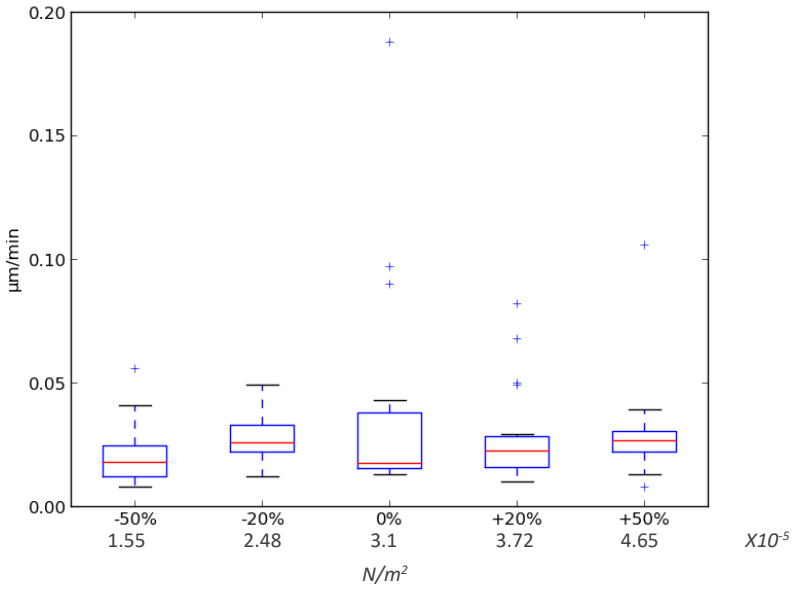

Figura 32. Tendencia de la velocidad de migración de las células ante la variación de la adhesión celular (α).

Por último, se van a analizar cómo afecta a los resultados del modelo la variación de la amplitud de las fuerzas aleatorias que ejercen las células sobre la matriz en su migración (α^r). De forma similar a los casos anteriores, se va a variar el valor base que aparece en la Tabla 4, incrementándolo y reduciéndolo en un 20 y un 50%.

Como en los casos anteriores, se va a comenzar mostrando los resultados obtenidos para los tamaños de los agregados. Se observa que el valor del tamaño de los agregados tiene su máximo en un valor de α^r alrededor del valor base, es decir, de 0.065 μN. Para mayores valores de α^r el tamaño disminuye debido a que este incremento de fuerza hace que las células sean capaces de migrar con mayor facilidad, teniendo como resultado un menor número de agregados con un área superior a $1000\ \mu m^2$.

Algo parecido ocurre al disminuir este parámetro, al reducirse esta fuerza que las células son capaces de hacer, se reduce su capacidad migratoria, lo que impide que el agregado de células pueda expandirse.

En la Figura 33 se puede observar la tendencia del tamaño de los agregados en función de esta fuerza α^r.

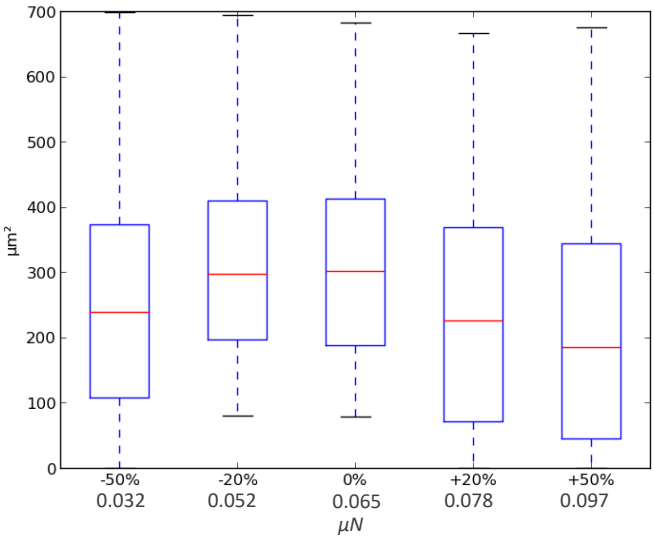

Figura 33. Tendencia del tamaño de los agregados celulares ante la variación de las fuerzas aleatorias ejercidas por las células (α^r).

En cuanto a la velocidad de migración de las células en la matriz, al aumentar esta amplitud de fuerzas se obtiene un aumento de la velocidad de éstas, aunque éste aumento es muy leve, tal y como se puede observar en la Figura 34, en la que se muestra esta tendencia.

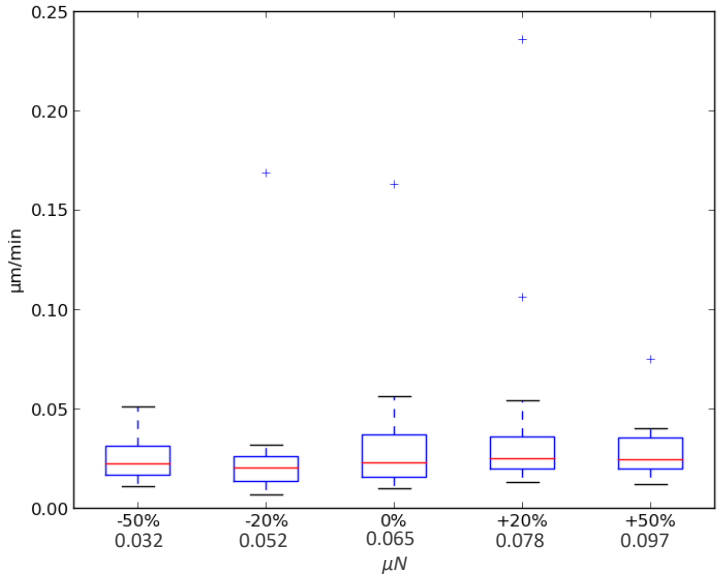

Figura 34. Tendencia de la velocidad de migración de las células ante la variación de la amplitud de las fuerzas aleatorias (α^r).

Capítulo 5. Conclusiones

La migración celular es un proceso que afecta directamente a la formación de agregados celulares. La variación en la concentración de colágeno de la matriz extracelular y los cambios en sus propiedades mecánicas que ésta conlleva es un parámetro clave en las características de esta migración.

A lo largo de este trabajo se ha visto cómo esta concentración de colágeno afecta de forma cuantitativa a parámetros tales como la velocidad de migración de las células o el tamaño y número de los agregados celulares que se forman.

Una concentración de 2.5 mg/mL facilita la migración celular, de forma que, tanto el tamaño de los agregados formados como la cantidad de éstos es menor. Esto se debe a la baja imposición de la matriz al movimiento de las células a través de ésta, teniendo como resultado una mayor velocidad de migración de las células.

En el caso de una matriz de 4.0 mg/mL, la migración se dificulta en mayor medida que en el caso anterior, provocando que se formen mayor número de agregados celulares con un mayor tamaño. La velocidad de las células se reduce de forma sustancial con respecto al caso anterior, siendo un 60% inferior.

En el último caso, en una matriz de 6 mg/mL, los agregados celulares que se observan en las simulaciones son de un tamaño mayor que en los dos casos anteriores, siguiendo el número de agregados formados la misma tendencia ascendente con la concentración. Además, la velocidad de migración también sufre un descenso respecto al caso anterior, aunque proporcionalmente menor que en el paso de 2.5 a 4.0 mg/mL, siendo este descenso de la velocidad de un 20%.

Por último, las simulaciones realizadas con un valor de adhesión inferior tratando de simular la introducción de TGF-β al sistema han demostrado que, tal y como se indica en los ensayos de Plou et al. [3], la velocidad celular disminuye en matrices de 2.5 y 6.0 mg/mL y se incrementa ligeramente en el caso de una matriz de 4.0 mg/mL.

Capítulo 6. Bibliografía

1. Asociación Española Contra el Cáncer.

2. American Cancer Society.

3. Plou, J., Juste-Lanas, Y., Olivares, V., del Amo, C., Borau, C. & García-Aznar, J.M. (2018). From individual to collective 3D cancer dissemination: roles of collagen concentration and TGF-β.

4. Caiazzo, A., & Ramis-Conde, I. (2015). Multiscale modelling of palisade formation in glioblastoma multiforme. Journal of theoretical biology, 383, 145-156.

5. Brat, D.J., Castellano-Sánchez, A.A., Hunter, S.B., Pecot, M., Cohen, C., Hammond, E.H., Devi, S.N., Kaur, B., Van Meir, E.G., 2004. Pseudopalisades in glioblastoma are hypoxic, express extracellular matrix proteases, and are formed by an actively migrating cell population.

6. Brat, D.J., Van Meir, E.G., 2004. Vaso-occlusive and prothrombotic mechanisms associated with tumour hypoxia, necrosis and accelerated growth in glioblastoma.

7. Martínez-González, A., Calvo, G.F., Pérez-Romasanta, L.A., Pérez-García, V.M., 2012. Hypoxic cell waves around necrotic cores in glioblastoma: a biomathematical model and its therapeutic implications.

8. Berens, M.E., Giese, A., 1999. Those left behind, biology and oncology of invasive glioma cells.

9. Ramis-Conde, I., Drasdo, D., Anderson, A.R.A., Chaplain, M.A.J., 2008. Modelling the influence of the e-cadherin-β-catenin pathway in cancer cell invasion.

10. Van Liederke, P., Neitsch, J., Johann, T. Alessandri, K., Nassoy, P., Drasdo, D., (2019). Quantitative cell-based model predicts mechanical stress response of growing tumor spheroids over various growth conditions and cell lines.

11. Guck, J., Ananthakrishnan, R., Mahmood, H., Moon, T.J., Cunningham, C.C., Ks, J., 2001. The optical stretcher: a novel laser tool to micromanipulate cells.

12. Ramis-Conde and Drasdo, 2012. From genotypes to phenotypes: classification of the tumour profiles for different variants of the cadherin adhesion pathway.

13. Galle, J., Drasdo, D., 2005. Modelling the effect of deregulated proliferation and apoptosis on the growth dynamics of epithelial cell populations in vitro.

14. Frisch, T., Thoumine, O., 2002. Predicting the kinetics of cell spreading.

15. Cóndor, M., Mark, C., Gerum, R.C., Grummel, N.C., Bauer, A., García-Aznar, J.M., Fabry, B., 2019. Breast cancer cells adapt contractile forces to overcome steric hindrance.

Capítulo 7. Anexos

Anexo 1. Principales resultados *in vitro*

Este trabajo replica los principales resultados experimentales de Plou et al. [3] mediante un modelo computacional propuesto por Caiazzo y Ramis-Conde [4]. En este primer anexo se comentan los resultados *in vitro* de Plou et al. [3] objeto de estudio en este trabajo: número y diámetro de los agregados celulares, y velocidades efectivas de las células individuales.

En lo que respecta al número de agregados celulares, los resultados de Plou et al. [3] muestran cómo en matrices de colágeno más densas, aparece un mayor número de estos agregados. Los autores consideran agregados celulares a agrupaciones de células con un área mínima de $1000\ \mu m^2$.

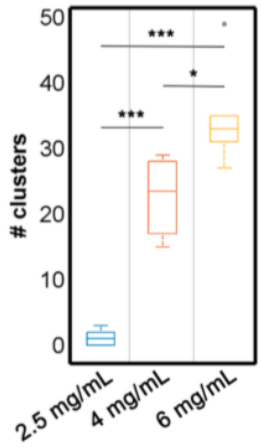

Figura A2.1. Gráfica comparativa de número de clústers en función de la concentración de la matriz [3].

Por otro lado, los experimentos *in vitro* de Plou et al. [3] también ponen de manifiesto un incremento en el tamaño de estos agregados con el paso del tiempo (24, 72 y 120 horas) y en matrices más densas (4.0 vs 6.0 mg/mL).

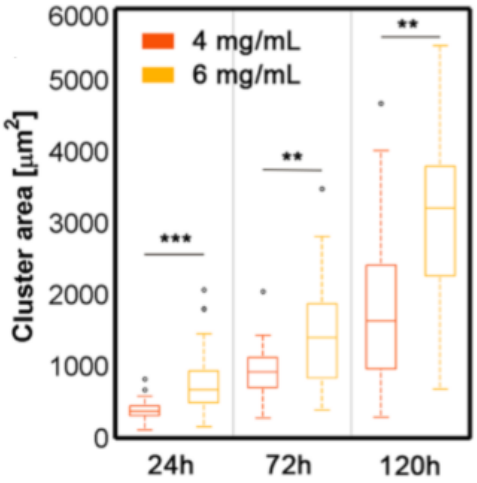

Figura A2.2. Gráfica comparativa de áreas de clúster en función de la concentración de la matriz y del tiempo simulado [3].

Por último, los autores muestran cómo las velocidades efectivas de las células individuales tras 48 horas del inicio de los experimentos se reducen conforme aumenta la densidad de las matrices de colágeno (2.5, 4.0 y 6.0 mg/mL).

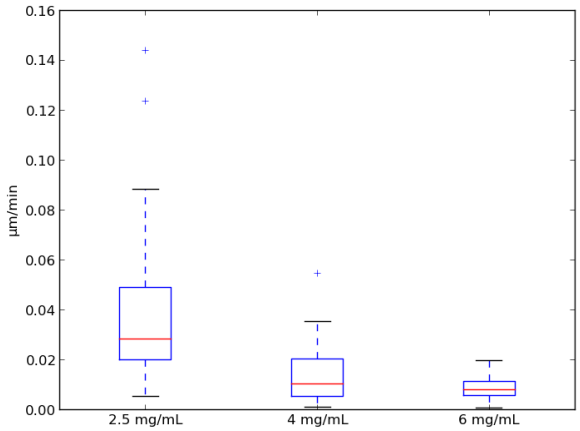

Figura A2.3. Gráfica resumen de los resultados obtenidos en Plou et al. [3] referentes a velocidades de células.

La velocidad efectiva se calcula según la ecuación (A2.1), como la distancia recorrida por la célula desde su nacimiento al final del experimento dividida por el tiempo que ha pasado desde su nacimiento hasta el final del experimento:

$$\boldsymbol{v}_{\boldsymbol{eff}}^{\boldsymbol{i}} = \frac{(posición_{final}^{i} - posición_{inicial}^{i})}{(t_{final}^{i} - t_{inicial}^{i})} \qquad \text{(A2.1)}$$

Anexo 2. Determinación de cada clúster.

En este anexo se explica el algoritmo utilizado para realizar la *clusterización* del modelo. Para ello se utiliza ALGLIB, una librería multiplataforma para análisis numérico y procesamiento de datos. ALGLIB incluye un clasificador que permite establecer a qué agregado celular pertenece cada punto del conjunto especificado. Para ello, es necesario introducir como parámetros de entrada un vector con las posiciones de cada uno de los puntos del conjunto a analizar y la distancia mínima entre agregados. Como parámetro de salida, este clasificador devuelve a qué agregado pertenece cada punto del conjunto dado (en este caso, cada una de las células).

A continuación, se muestra la implementación de este algoritmo:

```
void CoupledModel::clusterization() {

    // Create clusterizer
    clusterizerstate s;
    ahcreport arep;
    // Vector coord donde se almacenan las posiciones de las células
    real_2d_array coord;
    double xyz[20000][3];
    integer_1d_array cz;
    int w = 0;


    // Fill [xyz] vector with each cell position
    for (int k = this->minx; k <= this->maxx; k++) {
        for (int l = this->miny; l <= this->maxy; l++) {
            for (int n = this->minz; n <= this->maxz; n++) {
                for (unsigned int i = 0; i < this->boxes_A[k][l][n].cells.size(); i++) {
                    xyz[w][0] = this->boxes_A[k][l][n].cells[i].position[0];
                    xyz[w][1] = this->boxes_A[k][l][n].cells[i].position[1];
                    xyz[w][2] = this->boxes_A[k][l][n].cells[i].position[2];
                    w++;
                }
            }
        }
    }

    //Turning coord matrix into a 2d array
    const int cols = 3;
    int rows = this->total_no_of_cells;
    coord.setlength(rows, cols);

    // Filling array 2d coord
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            coord(i, j) = xyz[i][j];
        }
    }
```

```
    //Setting cluster
    clusterizercreate(s);

    //Setting points
    clusterizersetpoints(s, coord, 2); // clusterstate, vector con posiciones y metrica(euclidea)

    ae_int_t k; // Variable que almacena el numero de clusters

    clusterizerrunahc(s, arep);

    //Cluster number depending on intercluster distance
    clusterizerseparatedbydist(arep, params.intercluster_dist, k, this->cidx, cz);  //cidx vector that saves cluster index for each cell

    this->n_clusters = k;

    // Allocating cluster number to each cell
    int q = 0;
    for (int k = this->minx; k <= this->maxx; k++) {
        for (int l = this->miny; l <= this->maxy; l++) {
            for (int n = this->minz; n <= this->maxz; n++) {
                for (unsigned int i = 0; i < this->boxes_A[k][l][n].cells.size(); i++) {
                    this->boxes_A[k][l][n].cells[i].cluster_id = this->cidx[q];
                    q++;
                }
            }
        }
    }

}
```

Anexo 3. Posicionamiento de las N células iniciales.

Con el objetivo de simular realistamente el punto de partida de los experimentos *in vitro*, se ha desarrollado un algoritmo en Python para establecer la posición de las N células iniciales. Los parámetros de entrada de dicho algoritmo son los siguientes:

- Número de células iniciales.
- Valor de desviación típica.
- Semilla del generador de números aleatorios.
- Nombre del fichero de entrada para el modelo.

Una vez el algoritmo lee los parámetros de entrada especificados por el usuario a través de la terminal de comandos, procede a la carga del fichero de entrada. Este fichero de entrada almacena todos los valores iniciales necesarios para que el modelo simule correctamente las distintas pruebas. El programa reescribirá el fichero con las posiciones de las células iniciales que ha calculado a partir de los parámetros de entrada especificados por el usuario.

Este fichero de entrada incluye cuatro líneas clave, las cuales el programa modifica según corresponda. El resto del contenido del fichero de entrada queda intacto. Los casos en los que el programa realiza alguna acción son los siguientes:

1. Llega a una línea que comienza por *"n_initial_cells"*, en la cual el programa escribirá el número de células iniciales fijado por el usuario.

2. Llega a una línea que comienza por *"lattice_length_x"*, *"lattice_y"* o *"lattice_z"*, este parámetro da el valor límite de la matriz en cada uno de los ejes. En este caso, el programa almacena el valor de longitud para, posteriormente, compararlo con la posición inicial generada para evitar colocar células fuera del rango.

3. Llega a una línea que comienza por *"ic_cell_x"*, *"ic_cell_y"* o *"ic_cell_z"* en las cuales se almacenan las coordenadas en cada eje de todas las células. En este caso, una vez llega a cada una de estas líneas, se aplica el siguiente método:

   Mediante un generador de números aleatorios se obtiene cada una de las coordenadas. El programa utiliza para generar ese número aleatorio una media, que corresponde a la coordenada media de la caja en el eje correspondiente, además de una desviación típica y la semilla del generador de números aleatorios. Estos dos últimos valores son especificados por el usuario como parámetros de entrada.

   El uso de una semilla conocida para el generador de números aleatorios permite que las simulaciones tengan reproductibilidad, de forma que el generador nos dará el mismo valor aleatorio siempre que se use la misma semilla. El numero aleatorio

que se genera tiene que estar dentro de los límites de la matriz, de no ser así se repite la operación hasta que el valor cumpla esta condición.

A continuación, se muestra el código del programa descrito en este Anexo:

```
# -*- coding: utf-8 -*-
from __future__ import print_function, unicode_literals
from __future__ import absolute_import, division

from io import open
import numpy as np
import click

click.disable_unicode_literals_warning = True

@click.command()
@click.option('--n_celulas', '-n', default=1, help='Numero de celulas iniciales')
@click.option('--desv', '-d', default=1, help='Valor de la desviacion de la distribucion normal')
@click.option('--seed', '-s', help='Valor de la semilla generadora de posicion aleatoria')
@click.option('--input_texto', '-i', help='Fichero con el que trabaja el programa')


def posicionarCelulas(n_celulas, desv, seed, input_texto):

    input = open(input_texto, "r")
    lineas_fichero = input.readlines()  # Lista de lineas
    input.close()

    numsx = []
    numsy = []  # Listas para almacenar la geometria de la matriz
    numsz = []

    valorx = ""
    valory = ""  # Strings donde guardo los limites de la matriz
    valorz = ""

    coordsx = []
    coordsy = []  # Guardamos las coordenadas generadas aleatoriamente
    coordsz = []

    fichero = open(input_texto, "w")
```

```
for linea in lineas_fichero:

    # Encontrar la linea en la que se encuentre el numero de celulas iniciales del modelo

    if linea.startswith("n_initial_cells"):
        linea_nueva = "n_initial_cells = " + str(n_celulas) + "\n"
        fichero.write(linea_nueva)


    #Encontrar los valores limite de la matriz

    elif linea.startswith ("lattice_length_x"): # Coger valor de eje x de la matriz
        for caracter in linea:
            if caracter.isdigit() or caracter == ".":
                numsx.append(caracter)
        valorx = float(''.join(numsx))
        fichero.write(linea)

    elif linea.startswith("lattice_length_y"):  # Coger valor de eje y de la matriz
        for caracter in linea:
            if caracter.isdigit() or caracter == ".":
                numsy.append(caracter)
        valory = float(''.join(numsy))
        fichero.write(linea)

    elif linea.startswith("lattice_length_z"):  # Coger valor de eje z de la matriz
        for caracter in linea:
            if caracter.isdigit() or caracter == ".":
                numsz.append(caracter)
        valorz = float(''.join(numsz))
        fichero.write(linea)

    # Encontrar lineas en las que se encuentre la posicion inicial de las celulas y sobreesc. con numero aleatorio

     elif linea.startswith("ic_cell_x"):
        distribucion = np.random.normal(loc = valorx/2, scale= desv, size= n_celulas)
        for coordenada in distribucion:
            if coordenada > valorx:
                while coordenada > valorx:
                    np.random.seed(seed) # Valor de la semilla
                    coordenada = np.random.normal(loc = valorx, scale= desv)
        for coordenada in distribucion:
            coordsx.append(str(round(coordenada , 1)))
        linea_nueva = "ic_cell_x = " + "'" + ' '.join(coordsx) + "'" + "\n"
        fichero.write(linea_nueva)

     elif linea.startswith("ic_cell_y"):
        distribucion = np.random.normal(loc=valory/2, scale = desv, size = n_celulas)
        for coordenada in distribucion:
            if coordenada > valory:
                while coordenada > valory:
                    np.random.seed(seed) # Valor de la semilla
                    coordenada = np.random.normal(loc=valory/2, scale = desv)
        for coordenada in distribucion:
            coordsy.append(str(round(coordenada , 1)))
        linea_nueva = "ic_cell_y = " + "'" + ' '.join(coordsy) + "'" + "\n"
        fichero.write(linea_nueva)

     elif linea.startswith("ic_cell_z"):
        distribucion = np.random.normal(loc = valorz/2, scale = desv, size = n_celulas)
        for coordenada in distribucion:
            if coordenada > valorz:
                while coordenada > valorz:
                    np.random.seed(seed) #  Valor de la semilla
                    coordenada = np.random.normal(loc = valorz/2, scale = desv)
        for coordenada in distribucion:
            coordsz.append(str(round(coordenada , 1)))
        linea_nueva = "ic_cell_z = " + "'" + ' '.join(coordsz) + "'" + "\n"
        fichero.write(linea_nueva)
```

```
107
108
109             # Escribir el fichero de modo normal
110             else:
111                 fichero.write(linea)
112
113         fichero.close()
114
115         print("Fichero modificado.")
116
117
118 if __name__ == '__main__':
119     posicionarCelulas()
120
```

Anexo 4. Cálculo de las dimensiones de cada clúster.

El método para obtener las dimensiones de cada clúster se explica en este anexo. En primer lugar, es necesario haber realizado la clasificación de las células de nuestro sistema en clústeres. Una vez hecho esto, el programa parte de seis vectores auxiliares, donde almacenar los límites en las coordenadas x, y, z de cada clúster. Para obtener estos límites es necesario consultar la posición de todas las células del sistema.

La implementación de este algoritmo es la siguiente:

```
void CoupledModel::clusterDimension() {

    //Variables where store the max and min position for each cluster
    vector <double> maxdim_x(this->n_clusters, 0.0);
    vector <double> mindim_x(this->n_clusters, 1800.0);

    vector <double> maxdim_y(this->n_clusters, 0.0);
    vector <double> mindim_y(this->n_clusters, 1800.0);

    vector <double> maxdim_z(this->n_clusters, 0.0);
    vector <double> mindim_z(this->n_clusters, 1800.0);

    vector <int> number_of_cells(this->n_clusters);


    //Modifying dimension values
    for (int k = this->minx; k <= this->maxx; k++) {
        for (int l = this->miny; l <= this->maxy; l++) {
            for (int n = this->minz; n <= this->maxz; n++) {
                for (unsigned int i = 0; i < this->boxes_A[k][l][n].cells.size(); i++) {
                    //X axis code
                    if (this->boxes_A[k][l][n].cells[i].position[0] > maxdim_x[this->boxes_A[k][l][n].cells[i].cluster_id]) {
                        maxdim_x[this->boxes_A[k][l][n].cells[i].cluster_id] = this->boxes_A[k][l][n].cells[i].position[0];
                    }

                    if (this->boxes_A[k][l][n].cells[i].position[0] < mindim_x[this->boxes_A[k][l][n].cells[i].cluster_id]) {
                        mindim_x[this->boxes_A[k][l][n].cells[i].cluster_id] = this->boxes_A[k][l][n].cells[i].position[0];
                    }
                    //Y axis code
                    if (this->boxes_A[k][l][n].cells[i].position[1] > maxdim_y[this->boxes_A[k][l][n].cells[i].cluster_id]) {
                        maxdim_y[this->boxes_A[k][l][n].cells[i].cluster_id] = this->boxes_A[k][l][n].cells[i].position[1];
                    }

                    if (this->boxes_A[k][l][n].cells[i].position[1] < mindim_y[this->boxes_A[k][l][n].cells[i].cluster_id]) {
                        mindim_y[this->boxes_A[k][l][n].cells[i].cluster_id] = this->boxes_A[k][l][n].cells[i].position[1];
                    }
                    //Z axis code
                    if (this->boxes_A[k][l][n].cells[i].position[2] > maxdim_z[this->boxes_A[k][l][n].cells[i].cluster_id]) {
                        maxdim_z[this->boxes_A[k][l][n].cells[i].cluster_id] = this->boxes_A[k][l][n].cells[i].position[2];
                    }

                    if (this->boxes_A[k][l][n].cells[i].position[2] < mindim_z[this->boxes_A[k][l][n].cells[i].cluster_id]) {
                        mindim_z[this->boxes_A[k][l][n].cells[i].cluster_id] = this->boxes_A[k][l][n].cells[i].position[2];
                    }

                    number_of_cells[this->boxes_A[k][l][n].cells[i].cluster_id]++;

                }
            }
        }
    }
```

```
    vector <vector<double> > dim(this->n_clusters, vector<double>(3));

    //Setting the max and min position values for each cluster and number of cells
    for (unsigned int a = 0; a < this->n_clusters; a++) {
        dim[a][0] = maxdim_x[a] - mindim_x[a];
        dim[a][1] = maxdim_y[a] - mindim_y[a];
        dim[a][2] = maxdim_z[a] - mindim_z[a];
        this->n_of_cells.push_back(number_of_cells[a]);
    }

    ///////TAKE THE OUTPUT DIRECTORY NAME TO SET THE INERTIA FILE NAME
    string suffix = this->params.outputDirectory;
    suffix.erase(0, 8);
    suffix.erase(suffix.end() - 1, suffix.end());

    int ncellsmin = 50;

    if (params.Gcm >= 0.45) {
        ncellsmin = 30;
    }


    // CREATING THE FILE WITH INERTIA TENSOR AND CELL SPEED
    ofstream ficheroDatos;
;


    this->not_cluster = 0;
    unsigned int real_clusters = this->n_clusters;

    ficheroDatos.open(("info_" + suffix + ".dat").c_str(), ios::out);
    ficheroDatos << "Number of clusters: " << this->n_clusters << endl;
    ficheroDatos << "Dimensiones de clusters:" << endl;
    for (unsigned int a = 0; a < this->n_clusters; a++) {
        ficheroDatos << "Number of cells: " << number_of_cells[a] << endl;

        ficheroDatos << a << ": " << dim[a][0] << ", " << dim[a][1] << ", " << dim[a][2] << "  ";
        if (dim[a][0] > dim[a][1] && dim[a][0] > dim[a][2]) {
            ficheroDatos << "Max. Diameter: " << dim[a][0] << endl;
            if (dim[a][0] <= 36 || number_of_cells[a] < ncellsmin) {
                this->not_cluster++;

            }
        }
        if (dim[a][1] > dim[a][0] && dim[a][1] > dim[a][2]) {
            ficheroDatos << "Max. Diameter: " << dim[a][1] << endl;
            if (dim[a][1] <= 36 || number_of_cells[a] < ncellsmin) {
                this->not_cluster++;
            }
        }
        if (dim[a][2] > dim[a][0] && dim[a][2] > dim[a][1]) {
            ficheroDatos << "Max. Diameter: " << dim[a][2] << endl;
            if (dim[a][2] <= 36 || number_of_cells[a] < ncellsmin) {
                this->not_cluster++;
            }
        }


        ficheroDatos << "\n" << endl;
    }
```

```
    real_clusters = real_clusters - this->not_cluster;

    ficheroDatos << "Real Number of clusters (a > 1000 um^2): " << real_clusters << endl;


    ficheroDatos.close();
}
```

Anexo 5. Cálculo de velocidad efectiva de las células.

El cálculo de la velocidad efectiva de cada célula ha requerido de la definición de un nuevo atributo para los objetos de la clase Cell para almacenar la posición inicial de la célula i. La velocidad efectiva se calcula de acuerdo con la ecuación A6.1.

$$v_{eff}^{i} = \frac{(posición_{final}^{i} - posición_{inicial}^{i})}{(t_{final}^{i} - t_{inicial}^{i})} \qquad \text{(A6.1)}$$

Como puede observarse, esta velocidad no tiene en cuenta las posiciones intermedias por las que haya podido pasar la célula durante la simulación.

El código del algoritmo se ve en detalle a continuación:

```
void CoupledModel::getEffSpeeds() {

    double cell_eff_move = 0;

    vector <double> movement_eff_x;
    vector <double> movement_eff_y;
    vector <double> movement_eff_z;

    vector <int> alive_time;

    for (int k = this->minx; k <= this->maxx; k++) {
        for (int l = this->miny; l <= this->maxy; l++) {
            for (int n = this->minz; n <= this->maxz; n++) {
                for (unsigned int i = 0; i < this->boxes_A[k][l][n].cells.size(); i++) {

                    movement_eff_x.push_back(this->boxes_A[k][l][n].cells[i].position[0] - this->boxes_A[k][l][n].cells[i].position_birth[0]);
                    movement_eff_y.push_back(this->boxes_A[k][l][n].cells[i].position[1] - this->boxes_A[k][l][n].cells[i].position_birth[1]);
                    movement_eff_z.push_back(this->boxes_A[k][l][n].cells[i].position[2] - this->boxes_A[k][l][n].cells[i].position_birth[2]);

                    alive_time.push_back((this->reloj - this->boxes_A[k][l][n].cells[i].birthday) * params.time_step);
                }
            }
        }
    }

    for (int a = 0; a < this->total_no_of_cells; a++) {
        if (alive_time[a] == 0) {
            cell_eff_move += 0;
        }
        else {
            cell_eff_move += sqrt(pow(movement_eff_x[a], 2) + pow(movement_eff_y[a], 2) + pow(movement_eff_z[a], 2)) / alive_time[a];
        }
    }

    this->effective_speed = cell_eff_move / this->total_no_of_cells;
    cout << "effective Speed: " << this->effective_speed << endl;

    //double v_eff = this->effective_speed[params.n_steps - 1];
}
```

Anexo 6. Edición del fichero de entrada.

Este algoritmo se encarga de modificar los valores de los atributos asociados a cada simulación en el fichero de entrada del modelo. Consta de un método que requiere de 9 parámetros de entrada, los cuales son:

1. Fichero plantilla.
2. Fichero específico de la simulación.
3. Carpeta específica de la simulación.
4. Valor de *GCM*.
5. Valor de *ADHESION_VALUE*.
6. Valor de *VARIANCE_MOTION*.
7. Valor de *N_CELLS*.
8. Valor de *DESVIACIÓN*.
9. Valor de *SEED*.

El algoritmo recibe estos valores como parámetros de entrada y, sobre el fichero de la simulación (parámetro de entrada #2), sobrescribe los valores de los atributos *GCM*, *ADHESION_VALUE*, *VARIANCE_MOTION*, *N_CELLS* y la ruta del directorio en el que almacenar los resultados de la simulación (parámetro #3).

Para el posicionamiento de las células, el programa hace una llamada al script de posicionamiento de células de Python descrito anteriormente, al cual se le dan cuatro parámetros de entrada:

1. *N_CELLS*.
2. *DESVIACION*.
3. *SEED*.
4. Fichero específico.

El funcionamiento de este proceso ya ha sido descrito en el Anexo 2.

Los ficheros de entrada y el fichero de salida con todos los resultados se guardan en la carpeta específicamente creada para la simulación junto con cada uno de los ficheros *VTK* y se comprime.

A continuación, se muestra el código de este algoritmo:

```
#!/bin/bash

# Alex, define a function that will modified the contents of the input template.
# Use something like this: https://linuxize.com/post/bash-functions/

set_up_input_file () {
    # Generate cell++'s input file using the values given as input parameters.
    input_template=$1
    input_filename=$2
    results_path=$3
    # cell++ requires decimal numbers to be specified as 'x.x' instead of 'x,x'.
    GCM=$(echo $4 | sed 's/\,/\./g')
    ADHESION_VALUE=$(echo $5 | sed 's/\,/\./g')
    VARIANCE_MOTION=$(echo $6 | sed 's/\,/\./g')
    N_CELLS=$(echo $7 | sed 's/\,/\./g')
    DESV=$(echo $8 | sed 's/\,/\./g')
    SEED=$(echo $9 | sed 's/\,/\./g')

    # Create cell++'s input file from the given template
    cp $input_template $input_filename

    # Sed command to change each variable in the input file.
    sed -i "s/\$gcm/$GCM/g" $input_filename
    sed -i "s/\$av/$ADHESION_VALUE/g" $input_filename
    sed -i "s/\$vm/$VARIANCE_MOTION/g" $input_filename
    sed -i "s/\$ncells/$N_CELLS/g" $input_filename

    # The folder that will store cell++ results needs to be specified in the input file too.
    sed -i "s/\$output/$results_path/g" $input_filename


     # Set up cells initial positions.
     python celulasIniciales.py -n $N_CELLS -d $DESV -s $SEED -i $input_filename
}

# Read input parameters passed by the user through the command line.
# HTCondor will manage it so we have to parse the arguments.
# for i in "$@"
# do
# case $i in
```

```
while [ "$1" != "" ]; do
    # All parameters appearing in python and HTCondor scripts
    case $1 in
        -g | --gcm)
            shift
            GCM=$1
            shift
        ;;

        -a | --av)
            shift
            ADHESION_VALUE=$1
            shift
        ;;

        -v | --vm)
            shift
            VARIANCE_MOTION=$1
            shift
        ;;

        -s)
            shift
            SEED=$1
            shift
        ;;

        -n)
            shift
            N_CELLS=$1
            shift
        ;;

        -d)
            shift
            DESV=$1
            shift
        ;;

          *)
              # unknown option
          ;;
    esac
done
```

```
SUFFIX=$(printf "g%f_av%f_vm%f_s%d" $GCM $ADHESION_VALUE $VARIANCE_MOTION $SEED | sed 's/\,/\./g')

# Declare the variables for the function
input_template=input.dat
input_filename=input_$SUFFIX.dat
results_path=results_$SUFFIX

#Create results directory
mkdir results_$SUFFIX

# Set up cell++'s input file.
set_up_input_file $input_template $input_filename $results_path $GCM $ADHESION_VALUE $VARIANCE_MOTION $N_CELLS $DESV $SEED

# Run simulation.
./cell++ $input_filename

# Move the input file into the
mv $input_filename $results_path
cp info_*.dat $results_path

tar_filename=$(echo $results_path".tar.gz")
tar -czf $tar_filename $results_path *.dat

rm *.txt
```

Anexo 7. Lanzamiento de simulaciones en cola.

Este es el segundo script que se utiliza para automatizar la simulación de un alto número de casos de forma automática. Se encarga de guardar y proporcionar al programa anterior las distintas combinaciones de parámetros de entrada para cada simulación. Esto se consigue mediante la herramienta HTCondor [16].

HTCondor es un sistema basado en colas para distribuir la ejecución de trabajos en entornos de computación de alto rendimiento. De esta manera, es posible reducir de manera considerable el tiempo necesario para ejecutar un elevado número de simulaciones mediante la paralelización.

A continuación, se muestra el código del *Submit File*, el cual incluye las especificaciones de los trabajos a ejecutar en el clúster de computación a través de *HTCondor*:

```
universe     = vanilla
executable   = run_simulation.sh
arguments    = --gcm $(GCM) --vm $(VARIANCE_MOTION) --av $(ADHESION_VALUE) -s $(SEED) -n $(N_CELLS) -d $(DESV)
# requirements = ((Machine == "m2be-intel01") || (Machine == "m2be-intel02") || (Machine == "m2be-intel03") || (Machine == "m2be-intel04"))


error   = condor.$(Process).err
log     = condor.log
output  = condor.$(Process).out

should_transfer_files   = YES
transfer_input_files    = run_simulation.sh, celulasIniciales.py, input.dat, /home/amodrego/.local/bin/cell++
when_to_transfer_output = ON_EXIT_OR_EVICT

queue GCM, VARIANCE_MOTION, ADHESION_VALUE, SEED, N_CELLS, DESV from (
...
...
...
)
```

Anexo 8. Obtención de estadísticas.

El siguiente algoritmo es utilizado para la obtención de estadísticas de las simulaciones mediante Python. Al acabar la simulación de los casos, el modelo devuelve tantos ficheros de texto (*'info_*'*) con la información de la simulación como simulaciones se han ejecutado. Este algoritmo accede a la información de todos estos ficheros y realiza un análisis estadístico, cuyos resultados se almacenan en un nuevo fichero. Este análisis incluye los distintos percentiles, la mediana, la media, la varianza y desviación estándar de las distintas magnitudes objeto de análisis (velocidades efectivas, número y tamaño de los clústeres celulares).

La implementación de este algoritmo se muestra a continuación:

```
# -*- coding: utf-8 -*-
from __future__ import print_function, unicode_literals
from __future__ import absolute_import, division

from io import open
import numpy as np
import glob
import click

click.disable_unicode_literals_warning = True

@click.command()
@click.option('--archivo', '-a', default='info_*.dat', help='archivos a evaluar')

def createStats(archivo):

    statsInput = open("Stats.dat", "w")  # File that stores the data of the different simulations
    iter = 0

####### VARIABLES WHERE STORE ALL THE DATA NECCESARY TO DO THE STATS #########################
    numberClusterStats = []
    diameterStats = []
    speedStats = []
    cellMeanSpeedStats = []
    cellEffSpeedStats = []
    n_cellsStats = []
    real_clustersStats = []
    tx = []
    ty = []
    tz = []

##############################################################################################


```

```
    for simulation_stats in glob.iglob(archivo):

        tens = []
        diam = []
        speed = []
        n_cells = []
        mean_speed = 0
        eff_speed = 0

        t_x = []
        t_y = []
        t_z = []

        input = open(simulation_stats, "r")
        lineas_input = input.readlines()
        input.close()


        for linea in lineas_input:
############### STORING THE TOTAL NUMBER OF CLUSTERS IN EACH SIMULATION ########
            if linea.startswith("Number of clusters:"):
                pos = linea.find(":")
                clusters = int(linea[pos+2:len(linea)+1])

                numberClusterStats.append(clusters)  ############################## TOTAL NUMBER OF CLUSTERS


################ STORING EACH CLUSTER DIAMETER AVERAGE ########################
            if linea[0].isdigit():
                position = linea.find(":")
                compr = True
                for caracter in linea[0:position]:
                    if caracter.isdigit() == False:
                        compr = False
                if compr == True:
                    pos = linea.find("Diameter:")
                    joker = round(float(linea[pos+10:len(linea)+1]), 3)

                    diam.append(joker)
                    diameterStats.append(joker)  ############# ARRAY WITH CLUSTER DIAMETERS


################## STORING ABSOLUTE SPEED OF EACH CLUSTER ######################

            if linea.startswith("CDG Absolut Speed"):
                pos = linea.find(":")
                pos_mic = linea.find("microns")
                joker = round(float(linea[pos+1:pos_mic-1]), 3)

                speedStats.append(joker)  ###################### ARRAY WITH CLUSTER SPEEDS
                speed.append(joker)
```

```
87      ################## STORING MAIN CELLS SPEED #####################################
88
89                  if linea.startswith("Average Mean Speed"):
90                      pos = linea.find(":")
91                      joker = round(float(linea[pos+2:len(linea)+1]), 3)
92
93                      mean_speed = joker
94                      cellMeanSpeedStats.append(joker)
95
96                  if linea.startswith("Average Effective Speed"):
97                      pos = linea.find(":")
98                      joker = round(float(linea[pos+2:len(linea)+1]), 3)
99
100                     eff_speed = joker
101                     cellEffSpeedStats.append(joker)
102
103
104     ################## STORING MAIN VALUES FOR INERTIA TENSOR ########################
105                 if linea.startswith("("):
106
107                     pos = linea.find(",")
108                     pos_num = linea.find("(")
109                     joker = round(float(linea[pos_num + 1:pos]), 3)
110
111                     tens.append(joker)
112
113                 if len(tens) == 3:
114                     tx.append(tens[0])
115                     ty.append(tens[1])
116                     tz.append(tens[2])
117                     t_x.append(tens[0])
118                     t_y.append(tens[1])
119                     t_z.append(tens[2])
120                     tens = []

123     ################## STORING CLUSTER DENSITY #####################################
124
125             if linea.startswith("Number of cells:"):
126                 pos = linea.find(":")
127                 joker = int(linea[pos+2:len(linea)+1])
128                 n_cells.append(joker)
129                 n_cellsStats.append(joker)
130
131             if linea.startswith("Real Number of clusters"):
132                 pos = linea.find(":")
133                 joker = int(linea[pos+2:len(linea)+1])
134
135                 real_clustersStats.append(joker)
136
137
138
139     ################## WRITING DATA IN FILE ##################################
140
141         statsInput.write("\n\nSimulation Number: " + str(iter+1) + "\n")
142         statsInput.write("\nFormed Clusters: " + str(real_clustersStats[iter]) + "\n")
143         statsInput.write("\nAverage Diameters (microns):\n")
144         count = 0
145         while count < len(diam):
146             statsInput.write(str(diam[count]) + "\t\t")
147             if count % 3 == 0 and count != 0:
148                 statsInput.write("\n")
149
150             count += 1
151
```

```
152
153             statsInput.write("\n\nCDG Speeds (microns/min):\n")
154             count = 0
155             while count < len(speed):
156                 statsInput.write(str(speed[count]) + "\t")
157                 if count % 3 == 0 and count != 0:
158                     statsInput.write("\n")
159
160                 count += 1
161
162             statsInput.write("\n\nCell Mean Speeds: " + str(mean_speed) + " microns/min\n")
163
164
165             statsInput.write("\n\nCell Effective Speeds: " + str(eff_speed) + " microns/min\n")
166
167             statsInput.write("\n\nCluster Number of Cells:\n")
168             count = 0
169             while count < len(n_cells):
170                 statsInput.write(str(n_cells[count]) + "\t")
171                 if count % 3 == 0 and count != 0:
172                     statsInput.write("\n")
173
174                 count += 1

176             statsInput.write("\n\nInertias (micr^2):\n")
177             count = 0
178             while count < len(t_x):
179                 statsInput.write(str(t_x[count]) + ",\t" + str(t_y[count]) + ",\t" + str(t_z[count]) + "\n")
180                 count += 1
181
182             iter = iter+1
183

185     ###### ESTADÍSTICAS ################################
186         p_cl = []
187         p_diam = []
188         p_speed = []
189         p_n_cells = []
190         p_tx = []
191         p_ty = []
192         p_tz = []
193         p_meanspeed = []
194         p_effspeed = []
195
196
197         statsInput.write("\n\nEstadísticas\n\n")
198
199         #Numero Clusters
200         statsInput.write("\n\nNúmero de Clústers (> 1000 µm²):\n")
201         count = 0
202         counter = 0
203         while count < 11:
204             p_cl.append(np.percentile(real_clustersStats, counter))
205             statsInput.write("Percentil " + str(counter) + ": " + str(p_cl[count]) + "\n")
206             counter += 10
207             count += 1
208         statsInput.write("\nVarianza: " + str(np.var(real_clustersStats)))
209         statsInput.write("\nDesviación estándar: " + str(np.std(real_clustersStats)) + "\n\n")
210         statsInput.write("\nMedia: " + str(np.mean(real_clustersStats)) + "\n\n")
211
```

```
213         #Diametros
214         statsInput.write("\nDiámetros (µm):\n")
215         count = 0
216         counter = 0
217         while count < 11 :
218             p_diam.append(np.percentile(diameterStats, counter))
219             statsInput.write("Percentil " + str(counter) + ": " + str(p_diam[count]) + "\n")
220             counter += 10
221             count += 1
222         statsInput.write("\nVarianza: " + str(np.var(diameterStats)))
223         statsInput.write("\nDesviación estándar: " + str(np.std(diameterStats)) + "\n\n")
224         statsInput.write("\nMedia: " + str(np.mean(diameterStats)) + "\n\n")
225 
226 
227 
228         #Velocidades CDG
229         statsInput.write("\n\nVelocidad CDG (µm/min):\n")
230         count = 0
231         counter = 0
232         while count < 11:
233             p_speed.append(np.percentile(speedStats, counter))
234             statsInput.write("Percentil " + str(counter) + ": " + str(p_speed[count]) + "\n")
235             counter += 10
236             count += 1
237         statsInput.write("\nVarianza: " + str(np.var(speedStats)))
238         statsInput.write("\nDesviación estándar: " + str(np.std(speedStats)) + "\n\n")
239         statsInput.write("\nMedia: " + str(np.mean(speedStats)) + "\n\n")
240 
243         #Velocidades medias celulas
244         statsInput.write("\n\nVelocidad Media celulas (µm/min):\n")
245         count = 0
246         counter = 0
247         while count < 11:
248             p_meanspeed.append(np.percentile(cellMeanSpeedStats, counter))
249             statsInput.write("Percentil " + str(counter) + ": " + str(p_meanspeed[count]) + "\n")
250             counter += 10
251             count += 1
252         statsInput.write("\nVarianza: " + str(np.var(cellMeanSpeedStats)))
253         statsInput.write("\nDesviación estándar: " + str(np.std(cellMeanSpeedStats)) + "\n\n")
254         statsInput.write("\nMedia: " + str(np.mean(cellMeanSpeedStats)) + "\n\n")
255 
256 
257 
258         #Velocidades efectivas celulas
259         statsInput.write("\n\nVelocidad Efectivas celulas (µm/min):\n")
260         count = 0
261         counter = 0
262         while count < 11:
263             p_effspeed.append(np.percentile(cellEffSpeedStats, counter))
264             statsInput.write("Percentil " + str(counter) + ": " + str(p_effspeed[count]) + "\n")
265             counter += 10
266             count += 1
267         statsInput.write("\nVarianza: " + str(np.var(cellEffSpeedStats)))
268         statsInput.write("\nDesviación estándar: " + str(np.std(cellEffSpeedStats)) + "\n\n")
269         statsInput.write("\nMedia: " + str(np.mean(cellEffSpeedStats)) + "\n\n")
270 
```

```
272         #Numero de celulas
273         statsInput.write("\n\nNúmero de células:\n")
274         count = 0
275         counter = 0
276         while count < 11:
277             p_n_cells.append(np.percentile(n_cellsStats, counter))
278             statsInput.write("Percentil " + str(counter) + ": " + str(p_n_cells[count]) + "\n")
279             counter += 10
280             count += 1
281         statsInput.write("\nVarianza: " + str(np.var(n_cellsStats)))
282         statsInput.write("\nDesviación estándar: " + str(np.std(n_cellsStats)) + "\n\n")
283         statsInput.write("\nMedia: " + str(np.mean(n_cellsStats)) + "\n\n")
284 
285 
286         #Componentes X tensor
287         statsInput.write("\n\nComponentes X Inercias (µm²):\n")
288         count = 0
289         counter = 0
290         while count < 11 :
291             p_tx.append(np.percentile(tx, counter))
292             statsInput.write("Percentil " + str(counter) + ": " + str(p_tx[count]) + "\n")
293             counter += 10
294             count += 1
295         statsInput.write("\n\nVarianza: " + str(np.var(tx)))
296         statsInput.write("\nDesviación estándar: " + str(np.std(tx)) + "\n\n")
297         statsInput.write("\nMedia: " + str(np.mean(tx)) + "\n\n")

300         #Componentes Y tensor
301         statsInput.write("\n\nComponente Y Inercia (µm²):\n")
302         count = 0
303         counter = 0
304         while count < 11 :
305             p_ty.append(np.percentile(ty, counter))
306             statsInput.write("Percentil " + str(counter) + ": " + str(p_ty[count]) + "\n")
307             counter += 10
308             count += 1
309         statsInput.write("\n\nVarianza: " + str(np.var(ty)))
310         statsInput.write("\nDesviación estándar: " + str(np.std(ty)) + "\n\n")
311         statsInput.write("\nMedia: " + str(np.mean(ty)) + "\n\n")
312 
313 
314         #Componentes Z tensor
315         statsInput.write("\n\nComponente Z Inercia (µm²):\n")
316         count = 0
317         counter = 0
318         while count < 11:
319             p_tz.append(np.percentile(tz, counter))
320             statsInput.write("Percentil " + str(counter) + ": " + str(round(p_tz[count], 2)) + "\n")
321             counter += 10
322             count += 1
323         statsInput.write("\n\nVarianza: " + str(np.var(tz)))
324         statsInput.write("\nDesviación estándar: " + str(np.std(tz)) + "\n\n")
325         statsInput.write("\nMedia: " + str(np.mean(tz)) + "\n\n")
```

Anexo 9. Generación de gráficas.

En este anexo se va a explicar y mostrar el código utilizado para la obtención de las figuras con los diagramas de cajas que se muestran a lo largo de esta memoria haciendo uso de *Matplotlib*, una librería para generar gráficas en Python.

En primer lugar, para la obtención de las gráficas comparativas entre los experimentos *in vitro* [3] y las simulaciones, se utiliza un algoritmo que utiliza los resultados del análisis estadístico mencionado en el Anexo 9.

La implementación de este algoritmo se incluye a continuación, tanto para las comparativas entre resultados experimentales y simulaciones, como entre simulaciones de distintos escenarios:

```
404     ######      GRÁFICAS    ######
405         areaStats = []
406         for elemento in diameterStats:
407             areaStats.append(3.14169*pow(elemento/2, 2))
408
409
410         if float(case) == 2.5:
411             plt.boxplot(areaStats)
412             fig1, ax1 = plt.subplots()
413             ax1.boxplot(areaStats)
414             plt.ylabel('µm²')
415             plt.xticks([1], ['Simulaciones'])
416             plt.savefig("Areas 2,5.png")
417
418             data = [cellEffSpeedStats, vel_2_5]
419             fig2, ax2 = plt.subplots()
420             ax2.boxplot(data)
421             plt.ylabel('µm/min')
422             plt.xticks([1, 2], ['Simulaciones', 'Ensayos'])
423
424             plt.savefig("Velocidades 2,5.png")
425
426             data = [real_clustersStats, n_clus_2_5]
427             fig3, ax3 = plt.subplots()
428             ax3.boxplot(data)
429             plt.ylabel('# clusters')
430             plt.xticks([1, 2], ['Simulaciones', 'Ensayos'])
431
432             plt.savefig("Clusters 2,5.png")
433
```

```
435         if float(case) == 4:
436             data = [areaStats, arrayArea_4]
437             fig1, ax1 = plt.subplots()
438             ax1.boxplot(data)
439             plt.ylabel('µm²')
440             plt.xticks([1, 2], ['Simulaciones', 'Ensayos'])
441             plt.savefig("Areas 4.png")
442 
443             data = [cellEffSpeedStats, vel_4]
444             fig2, ax2 = plt.subplots()
445             ax2.boxplot(data)
446             plt.ylabel('µm/min')
447             plt.xticks([1, 2], ['Simulaciones', 'Ensayos'])
448 
449             plt.savefig("Velocidades 4.png")
450 
451 
452             data = [real_clustersStats, n_clus_4]
453             fig3, ax3 = plt.subplots()
454             ax3.boxplot(data)
455             plt.ylabel('# clusters')
456             plt.xticks([1, 2], ['Simulaciones', 'Ensayos'])
457 
458             plt.savefig("Clusters 4.png")
459 

461         if float(case) == 6:
462             data =[areaStats, arrayArea_6]
463             fig1, ax1 = plt.subplots()
464             ax1.boxplot(data)
465             plt.ylabel('µm²')
466             plt.xticks([1, 2], ['Simulaciones', 'Ensayos'])
467             plt.savefig("Areas 6.png")
468 
469             data = [cellEffSpeedStats, vel_6]
470             fig2, ax2 = plt.subplots()
471             ax2.boxplot(data)
472             plt.ylabel('µm/min')
473             plt.xticks([1, 2], ['Simulaciones', 'Ensayos'])
474             plt.savefig("Velocidades 6.png")
475 
476             data = [real_clustersStats, n_clus_6]
477             fig3, ax3 = plt.subplots()
478             ax3.boxplot(data)
479             plt.ylabel('# clusters')
480 
481             plt.xticks([1, 2], ['Simulaciones', 'Ensayos'])
482             plt.savefig("Clusters 6.png")
483 
484 
485 
486 ▶   if __name__ == '__main__':
487         createStats()
488 
```

Para las gráficas comparativas de los distintos casos simulados (matrices a 2.5, 4.0 y 6.0 mg/mL), la implementación del algoritmo es la siguiente:

```
13  def comparativasEnsayos():
14
15
16      speed_25_list = []
17      speed_4_list = []
18      speed_6_list = []
19
20      nclusters_25 = []
21      nclusters_4 = []
22      nclusters_6 = []
23
24      area_25 = []
25      area_4 = []
26      area_6 = []
27
28      tens_x_25 = []
29      tens_y_25 = []
30      tens_z_25 = []
31
32      tens_x_4 = []
33      tens_y_4 = []
34      tens_z_4 = []
35
36      tens_x_6 = []
37      tens_y_6 = []
38      tens_z_6 = []

40      for simulation_stats in glob.iglob("info_s_g0.30*"):
41          tens = []
42
43          input = open(simulation_stats, "r")
44          lineas_input = input.readlines()
45          input.close()
46
47          for linea in lineas_input:
48
49              ################ STORING EACH CLUSTER DIAMETER AVERAGE ##########################
50              if linea[0].isdigit():
51                  position = linea.find(":")
52                  compr = True
53                  for caracter in linea[0:position]:
54                      if caracter.isdigit() == False:
55                          compr = False
56                  if compr == True:
57                      pos = linea.find("Diameter:")
58                      joker = round(float(linea[pos + 10:len(linea) + 1]), 3)
59
60                      if joker >= 10:
61                          area_25.append(3.14159* pow(joker/2, 2))  ############# ARRAY WITH CLUSTER DIAMETERS
62
```

```
                ################## STORING EFFECTIVE SPEED OF EACH CELL ######################

                if linea.startswith("Average Effective Speed"):
                    pos = linea.find(":")
                    joker = round(float(linea[pos + 2:len(linea) + 1]), 3)

                    speed_25_list.append(joker)

                ################# STORING MAIN VALUES FOR INERTIA TENSOR #######################
                if linea.startswith("("):
                    pos = linea.find(",")
                    pos_num = linea.find("(")
                    joker = round(float(linea[pos_num + 1:pos]), 3)

                    tens.append(joker)

                if len(tens) == 3:
                    if tens[0] < 0.04:
                        tens_x_25.append(tens[0])
                    if tens[1] < 0.04:
                        tens_y_25.append(tens[1])
                    if tens[2] < 0.04:
                        tens_z_25.append(tens[2])

                    tens = []

                ################ STORING CLUSTER DENSITY #####################################

                if linea.startswith("Real Number of clusters"):
                    pos = linea.find(":")
                    joker = int(linea[pos + 2:len(linea) + 1])

                    nclusters_25.append(joker)

    for simulation_stats in glob.iglob("info_s_g0.72*"):
        tens = []

        input = open(simulation_stats, "r")
        lineas_input = input.readlines()
        input.close()

        for linea in lineas_input:

            ################ STORING EACH CLUSTER DIAMETER AVERAGE ########################
            if linea[0].isdigit():
                position = linea.find(":")
                compr = True
                for caracter in linea[0:position]:
                    if caracter.isdigit() == False:
                        compr = False
                if compr == True:
                    pos = linea.find("Diameter:")
                    joker = round(float(linea[pos + 10:len(linea) + 1]), 3)

                    if joker >= 10:
                        area_4.append(3.14159 * pow(joker / 2, 2))  ############## ARRAY WITH CLUSTER DIAMETERS
```

```
                ################## STORING EFFECTIVE SPEED OF EACH CELL ######################

                if linea.startswith("Average Effective Speed"):
                    pos = linea.find(":")
                    joker = round(float(linea[pos + 2:len(linea) + 1]), 3)

                    speed_4_list.append(joker)

                ################# STORING MAIN VALUES FOR INERTIA TENSOR #####################
                if linea.startswith("("):
                    pos = linea.find(",")
                    pos_num = linea.find("(")
                    joker = round(float(linea[pos_num + 1:pos]), 3)

                    tens.append(joker)

                if len(tens) == 3:
                    if tens[0] < 0.04:
                        tens_x_4.append(tens[0])
                    if tens[1] < 0.04:
                        tens_y_4.append(tens[1])
                    if tens[2] < 0.04:
                        tens_z_4.append(tens[2])

                    tens = []

                ################ STORING CLUSTER DENSITY #######################################

                if linea.startswith("Real Number of clusters"):
                    pos = linea.find(":")
                    joker = int(linea[pos + 2:len(linea) + 1])

                    nclusters_4.append(joker)

for simulation_stats in glob.iglob("info_s_g0.75*"):
    tens = []

    input = open(simulation_stats, "r")
    lineas_input = input.readlines()
    input.close()

    for linea in lineas_input:

        ################ STORING EACH CLUSTER DIAMETER AVERAGE ########################
        if linea[0].isdigit():
            position = linea.find(":")
            compr = True
            for caracter in linea[0:position]:
                if caracter.isdigit() == False:
                    compr = False
            if compr:
                pos = linea.find("Diameter:")
                joker = round(float(linea[pos + 10:len(linea) + 1]), 3)

                if joker >= 10:
                    area_6.append(3.14159 * pow(joker / 2, 2))  ############## ARRAY WITH CLUSTER DIAMETERS
```

```
            if linea.startswith("Average Effective Speed"):
                pos = linea.find(":")
                joker = round(float(linea[pos + 2:len(linea) + 1]), 3)

                speed_6_list.append(joker)

            ################## STORING MAIN VALUES FOR INERTIA TENSOR ########################
            if linea.startswith("("):
                pos = linea.find(",")
                pos_num = linea.find("(")
                joker = round(float(linea[pos_num + 1:pos]), 3)

                tens.append(joker)

            if len(tens) == 3:
                if tens[0] < 0.04:
                    tens_x_6.append(tens[0])
                if tens[1] < 0.04:
                    tens_y_6.append(tens[1])
                if tens[2] < 0.04:
                    tens_z_6.append(tens[2])

                tens = []

             ################## STORING CLUSTER DENSITY #########################################

             if linea.startswith("Real Number of clusters"):
                pos = linea.find(":")
                joker = int(linea[pos + 2:len(linea) + 1])

                nclusters_6.append(joker)

#Comparativa Areas
data = [area_25, area_4, area_6]
fig1, ax1 = plt.subplots()
ax1.boxplot(data)
plt.ylabel('µm²')
plt.xticks([1, 2, 3], ['2.5 mg/ml', '4 mg/ml', '6 mg/ml'])
plt.savefig("ComparativaAreas.png")


#Comparativa Velocidades
data = [speed_25_list, speed_4_list, speed_6_list]
fig2, ax2 = plt.subplots()
ax2.boxplot(data)
plt.ylabel('µm/min')
plt.xticks([1, 2, 3], ['2.5 mg/ml', '4 mg/ml', '6 mg/ml'])

plt.savefig("ComparativaVelocidades.png")

#Comparativa NClusters
data = [nclusters_25, nclusters_4, nclusters_6]
fig3, ax3 = plt.subplots()
ax3.boxplot(data)
plt.ylabel('# clusters')
plt.xticks([1, 2, 3], ['2.5 mg/ml', '4 mg/ml', '6 mg/ml'])

plt.savefig("ComparativaNClusters.png")
```

```
239         #Comparativas inercias X
240         data = [tens_x_25, tens_x_4, tens_x_6]
241         fig4, ax4 = plt.subplots()
242         ax4.boxplot(data, sym="w")
243         plt.ylabel('µm²')
244         plt.xticks([1, 2, 3], ['2.5 mg/ml', '4 mg/ml', '6 mg/ml'])
245 
246         plt.savefig("InerciasX.png")
247 
248         # Comparativas inercias Y
249         data = [tens_y_25, tens_y_4, tens_y_6]
250         fig5, ax5 = plt.subplots()
251         ax5.boxplot(data, sym="w")
252         plt.ylabel('µm²')
253         plt.xticks([1, 2, 3], ['2.5 mg/ml', '4 mg/ml', '6 mg/ml'])
254 
255         plt.savefig("InerciasY.png")
256 
257         # Comparativas inercias Z
258         data = [tens_z_25, tens_z_4, tens_z_6]
259         fig6, ax6 = plt.subplots()
260         ax6.boxplot(data, sym="w")
261         plt.ylabel('µm²')
262         plt.xticks([1, 2, 3], ['2.5 mg/ml', '4 mg/ml', '6 mg/ml'])
263 
264         plt.savefig("InerciasZ.png")
```

Por último, se incluye el código utilizado para la obtención de las gráficas en función del tiempo, habiendo realizado antes las simulaciones para unos tiempos de 24, 72 y 120 horas:

```
267 def comparativaTiempos():
268     area_4_120 = []
269     area_4_72 = []
270     area_4_24 = []
271 
272     area_6_120 = []
273     area_6_72 = []
274     area_6_24 = []
275 
276 
277 
278     for simulation_stats in glob.iglob("120h/info_s_g0.72*"):
279 
280         input = open(simulation_stats, "r")
281         lineas_input = input.readlines()
282         input.close()
283 
284         for linea in lineas_input:
285 
286             ################ STORING EACH CLUSTER DIAMETER AVERAGE ########################
287             if linea[0].isdigit():
288                 position = linea.find(":")
289                 compr = True
290                 for caracter in linea[0:position]:
291                     if caracter.isdigit() == False:
292                         compr = False
293                 if compr == True:
294                     pos = linea.find("Diameter:")
295                     joker = round(float(linea[pos + 10:len(linea) + 1]), 3)
296 
297                     if joker >= 10:
298                         area_4_120.append(3.14159* pow(joker/2, 2))  ############# ARRAY WITH CLUSTER DIAMETERS
299 
300 
```

```
for simulation_stats in glob.iglob("120h/info_s_g0.75*"):

    input = open(simulation_stats, "r")
    lineas_input = input.readlines()
    input.close()

    for linea in lineas_input:

        ################ STORING EACH CLUSTER DIAMETER AVERAGE ########################
        if linea[0].isdigit():
            position = linea.find(":")
            compr = True
            for caracter in linea[0:position]:
                if caracter.isdigit() == False:
                    compr = False
            if compr == True:
                pos = linea.find("Diameter:")
                joker = round(float(linea[pos + 10:len(linea) + 1]), 3)

                if joker >= 10:
                    area_6_120.append(3.14159 * pow(joker / 2, 2))  ############# ARRAY WITH CLUSTER DIAMETERS


for simulation_stats in glob.iglob("72h/info_s_g0.72*"):

    input = open(simulation_stats, "r")
    lineas_input = input.readlines()
    input.close()

    for linea in lineas_input:

        ################ STORING EACH CLUSTER DIAMETER AVERAGE ########################
        if linea[0].isdigit():
            position = linea.find(":")
            compr = True
            for caracter in linea[0:position]:
                if caracter.isdigit() == False:
                    compr = False
            if compr:
                pos = linea.find("Diameter:")
                joker = round(float(linea[pos + 10:len(linea) + 1]), 3)

                area_4_72.append(3.14159 * pow(joker / 2, 2))  ############# ARRAY WITH CLUSTER DIAMETERS

for simulation_stats in glob.iglob("72h/info_s_g0.75*"):

    input = open(simulation_stats, "r")
    lineas_input = input.readlines()
    input.close()

    for linea in lineas_input:

        ################ STORING EACH CLUSTER DIAMETER AVERAGE ########################
        if linea[0].isdigit():
            position = linea.find(":")
            compr = True
            for caracter in linea[0:position]:
                if caracter.isdigit() == False:
                    compr = False
            if compr:
                pos = linea.find("Diameter:")
                joker = round(float(linea[pos + 10:len(linea) + 1]), 3)

                area_6_72.append(3.14159 * pow(joker / 2, 2))  ############# ARRAY WITH CLUSTER DIAMETERS
```

```
    for simulation_stats in glob.iglob("24h/info_s_g0.72*"):

        input = open(simulation_stats, "r")
        lineas_input = input.readlines()
        input.close()

        for linea in lineas_input:

            ################ STORING EACH CLUSTER DIAMETER AVERAGE ##########################
            if linea[0].isdigit():
                position = linea.find(":")
                compr = True
                for caracter in linea[0:position]:
                    if caracter.isdigit() == False:
                        compr = False
                if compr:
                    pos = linea.find("Diameter:")
                    joker = round(float(linea[pos + 10:len(linea) + 1]), 3)

                    area_4_24.append(3.14159 * pow(joker / 2, 2))  ############## ARRAY WITH CLUSTER DIAMETERS

    for simulation_stats in glob.iglob("24h/info_s_g0.75*"):

        input = open(simulation_stats, "r")
        lineas_input = input.readlines()
        input.close()

        for linea in lineas_input:

            ################ STORING EACH CLUSTER DIAMETER AVERAGE ##########################
            if linea[0].isdigit():
                position = linea.find(":")
                compr = True
                for caracter in linea[0:position]:
                    if caracter.isdigit() == False:
                        compr = False
                if compr:
                    pos = linea.find("Diameter:")
                    joker = round(float(linea[pos + 10:len(linea) + 1]), 3)

                    area_6_24.append(3.14159 * pow(joker / 2, 2))  ############## ARRAY WITH CLUSTER DIAMETERS

    data = [area_4_24, area_6_24, area_4_72, area_6_72, area_4_120, area_6_120]
    fig1, ax1 = plt.subplots()
    ax1.boxplot(data)
    plt.ylabel('µm²')

    plt.xticks([1, 2, 3], ['24h', '72h', '120h'])

    plt.savefig("Comparativa Tiempos.png")




if __name__ == '__main__':
    comparativasEnsayos()
    comparativaTiempos()
```