



UNIVERSIDAD
NACIONAL
DE LA PLATA

Facultad de Ciencias Exactas

Departamento de Ciencias Biológicas

Regeneración de tejidos

Una aproximación de biología de sistemas

Tesista: Lic. Emanuel Cura Costa

Director: Prof. Dr. Osvaldo Chara

Año 2022

Trabajo realizado en el Systems Biology Group (SysBio)
Instituto de Física de Líquidos y Sistemas Biológicos (IFLySiB)

Calle 59 Número 789 (entre 10 y 11)
B1900BTE La Plata - Buenos Aires
Argentina

<https://sysbioiflysib.wordpress.com/>

<http://iflysib.unlp.edu.ar/>

Resumen

El axolotl es un vertebrado que posee la inigualable capacidad de regenerar completamente la médula espinal luego de una amputación. Pese a que estas capacidades regenerativas de esta salamandra se conocen desde hace más de 250 años en el ámbito científico, poco se sabe aún sobre los mecanismos que subyacen a la regeneración de la médula espinal. El objetivo general de esta tesis consiste en entender mecanísticamente la regeneración de la médula espinal del axolotl en términos de procesos celulares y moleculares o de señalización. Anteriormente nuestro grupo encontró que la amputación de la cola del axolotl conduce a la reactivación de un programa similar al del desarrollo en las células endodermales de la médula espinal (Rodrigo Albors et al., 2015. eLife. 4:e10230), caracterizado por una zona de alta proliferación que emerge 4 días después de la amputación (Rost et al., 2016. eLife. 5:e20357). Sin embargo, se desconocía qué subyace a este patrón espacio-temporal de proliferación celular. En esta tesis, utilizamos modelado matemático computacional estrechamente vinculados a datos experimentales (en una aproximación de biología de sistemas), para estudiar la regeneración de la médula espinal del axolotl en dos escalas: una escala celular y una escala de señalización. En una primera aproximación estudiamos el proceso regenerativo con un modelo de esferas duras en una dimensión, que incluye información precisa de la posición de las células en su ciclo celular, y demostramos que la respuesta regenerativa es consistente con una señal que recluta células endodermales durante ~ 85 horas después de la amputación dentro de $\sim 830\mu\text{m}$ de la médula espinal anteriores al plano de amputación. Luego, nos valemos de la tecnología del indicador de ciclo celular basado en la ubiquitinación fluorescente (FUCCI) que adaptamos a los axolotls (AxFUCCI) para visualizar los ciclos celulares in vivo y determinamos la distribución espacio-temporal de las células de la médula espinal en las fases del ciclo celular, confirmando el tiempo de aparición predicho y el tamaño de la zona de reclutamiento inducida por la amputación revelando, además, sincronía del ciclo celular entre las células endodermales. Con la intención de comenzar a abordar aspectos mecánicos de la interacción entre las células endodermales, desarrollamos un paquete computacional de un *celular vertex model* (SysVert) que nos permitió la implementación de un modelo en dos dimensiones de la médula espinal del axolotl durante el proceso regenerativo. Los resultados obtenidos en el marco de este trabajo de tesis nos acercan más a la comprensión del proceso regenerativo de la médula espinal del axolotl.

Publicaciones durante la tesis

En el marco de esta tesis doctoral se publicó el siguiente artículo en una revista internacional con referato:

E Cura Costa, L Otsuki, AR Albors, EM Tanaka & O Chara. 2021. *Spatiotemporal control of cell cycle acceleration during axolotl spinal cord regeneration*. eLife 2021;10:e55665 doi: 10.7554/eLife.55665.

Asimismo se aloja en un repositorio de *gitlab.com* el código del *software* SysVert que prontamente será publicado y liberado.

Adicionalmente y durante el lapso del doctorado, el tesista publicó estos artículos también en revistas internacionales con referato:

N König, C Fiehn, C Wolf, M Schuster, **E Cura Costa**, V Tüngler, HA Alvarez, O Chara, K Engel, R Goldbach-Mansky, C Günther & MA Lee-Kirsch. 2017. *Familial chilblain lupus due to a gain-of-function mutation in STING*. Ann Rheum Dis. 2017. Feb;76(2):468-472. doi: 10.1136/annrheumdis-2016-209841. Epub 2016 Aug 26. PMID: 27566796.

N Berndt, C Wolf, K Fischer, **E Cura Costa**, P Knuschke, N Zimmermann, F Schmidt, M Merkel, O Chara, M Lee-Kirsch & C Günther. *Photosensitivity and cGAS-dependent type I IFN activation in lupus patients with TREX1 deficiency*. J Invest Dermatol. 2021. Aug 14:S0022-202X(21)01686-9. doi: 10.1016/j.jid.2021.04.037. Epub ahead of print. PMID: 34400195.

Agradecimientos

En primer lugar y por sobre todo quiero agradecer al director de esta tesis doctoral, Prof. Dr. Osvaldo Chara, por haber confiado en mi, por compartirme su sabiduría, por contagiarme su pasión por la ciencia y por su infinita paciencia.

Agradezco también a mis compañeros de SysBio. Especialmente al Lic. Alberto Caccarelli y Dr. Ariel Alvarez por los momentos de trabajo conjunto y las risas compartidas.

De igual manera agradezco a toda la gente en el IFLySiB. Gracias a todos y cada uno por hacerme sentir en casa, por los asados y por las charlas durante el almuerzo.

Esta tesis es fruto no sólo de mi trabajo sino también de proyectos y esfuerzos de otras personas. Por esta razón quiero agradecer a Dra. Aida Rodrigo Albors, Dr. Fabian Rost, Dr. Leo Otsuki y Prof. Dra. Elly Tanaka. Gracias Fabian por tu simpatía y las discusiones siempre tan valiosas, y gracias Leo por tu amabilidad y entusiasmo por este proyecto.

También quiero agradecer a CONICET y a la Universidad Nacional de La Plata por otorgarme el financiamiento y las herramientas que hicieron posible este trabajo.

Asimismo quiero agradecer a la Technische Universität Dresden por permitirme viajar a Dresden a través del *great! ipid4all group2group exchange for academic talents* y, además, al Prof. Dr. Andreas Deutsch, Prof. Dra. Min Ae Lee-Kirsch y Dra. Victoria Tüngler por hacer posible mi viaje y agradable mi estancia.

La elaboración de esta tesis me demandó mucho tiempo, esfuerzo y trabajo. Muchas veces tuve que decir que no a alguna juntada con amigos, me olvidé de levantar el teléfono para hablar con mi familia o postergué alguna visita a Junín. Quiero agradecer profundamente a ellos, a mi familia y a mis amigos, por su apoyo, por su acompañamiento, por su energía, por su paciencia y su comprensión. Sin su apoyo este trabajo nunca se habría terminado y por eso considero que esta tesis es también de ustedes.

A todos, muchas gracias.

Índice general

Resumen	i
Publicaciones durante la tesis	iii
Agradecimientos	v
Índice	vii
Índice de figuras	ix
Índice de cuadros	xii
Índice	xiii
1 Introducción	1
1.1 Modelando la complejidad biológica: biología de sistemas	1
1.2 Regeneración de tejidos animales	2
1.3 Modelo de estudio: axolotl (<i>ambystoma mexicanum</i>)	3
1.4 Potencial regenerativo de las células ependimales en la médula espinal del axolotl	4
1.5 Aceleración del ciclo celular durante la regeneración de la médula espinal del axolotl	5
1.6 Objetivos de este trabajo de tesis	5
2 Modelado de la regeneración de la médula espinal del axolotl en 1D	7
2.1 Modelo de médula espinal en desarrollo	7
2.2 Modelo de regeneración de la médula espinal	9
2.3 Detalles del modelado de desarrollo y regeneración de la médula espinal axolotl	13
2.4 El crecimiento regenerativo de la médula espinal se puede explicar por una señal que actúa durante las 85 horas después de la amputación y recluta células dentro de los $828\mu m$ anteriores al plano de amputación	16
2.5 Crecimiento de la médula espinal cuando se impide la aceleración del ciclo celular	19
2.6 El acortamiento de la fase S es suficiente para explicar el crecimiento inicial de la médula espinal en regeneración	22

3	Determinación del tránsito a través del ciclo celular mediante FUCCI en axolotl <i>in vivo</i>	25
3.1	Visualización de la progresión del ciclo celular en axolotl <i>in vivo</i> usando FUCCI	25
3.2	AxFUCCI discrimina fielmente las fases del ciclo celular <i>in vivo</i>	26
3.3	Medición de la zona de reclutamiento <i>in vivo</i> usando AxFUCCI	28
3.4	Las células en regeneración tienen una alta sincronía del ciclo celular <i>in vivo</i>	29
3.5	Una convergencia en la respuesta regenerativa de distintas líneas de base	36
4	Sincronización inducida por el reclutamiento y sincronización pre-existente a la amputación: predicciones del modelo 1D	41
4.1	Sincronización en la fase G1 al momento del reclutamiento	41
4.2	Sincronización en la condición inicial	43
5	Implementación computacional del modelo de vértices: SysVert	49
5.1	Modelo de vértices	50
5.2	Estructura de SysVert	79
5.3	¿Cómo se corre y que datos se pueden obtener de una simulación en SysVert?	104
6	Modelado de la regeneración de la médula espinal del axolotl en 2D	105
6.1	Configuración de la simulación del tejido en 2D en SysVert	105
6.2	Resultados de simulación	109
7	Discusión	119
A	Materiales y métodos	133
A.1	Métodos computacionales	133
A.2	Materiales y métodos experimentales	135
	Bibliografía	133
B	Otros resultados experimentales de FUCCI	141

Índice de figuras

1.1	Axolotl (<i>ambystoma mexicanum</i>)	4
2.1	Modelo de crecimiento de la médula espinal ilesa y en regeneración basado en el acortamiento de las fases <i>G1</i> y <i>S</i> de las células	8
2.2	Distribución espacio-temporal de la proliferación celular durante la regeneración médula espinal axolotl	11
2.3	Explicación adicional de la transformación de coordenadas del ciclo celular invocada por el modelo.	12
2.4	Aproximación de cálculo bayesiano (ABC) ajuste del modelo de límite de reclutamiento a los datos experimentales del <i>switchpoint</i>	17
2.5	El modelo contempla el mecanismo de empuje de la célula: cuanto más posterior está una célula, cuanto más rápido se mueve	18
2.6	La incorporación de la variabilidad de la longitud de la célula a lo largo del eje <i>AP</i> no afecta la predicción del crecimiento de la médula espinal del axolotl	20
2.7	Una señal hipotética recluta células ependimales hasta $828\mu m$ antes del plano de amputación durante las 85 horas posteriores a la amputación, recapitulando el crecimiento regenerativo de la médula espinal <i>in vivo</i>	21
2.8	El tiempo máximo de reclutamiento (τ) y la longitud máxima de reclutamiento (λ) determinan el crecimiento de la médula espinal del axolotl	22
3.1	AxFUCCI: un reportero del ciclo celular transgénico para axolotl	27
3.2	El tamaño de la zona de reclutamiento previsto se observa en las colas Ax-FUCCI después de la amputación	30
3.3	Estimación de los porcentajes anterior y posterior de células que expresan <i>G0/G1</i> -AxFUCCI y <i>S/G2</i> -AxFUCCI ajustado un modelo de dos zonas	32
3.4	Ajuste individual del modelo de dos zonas a los perfiles <i>AP</i> experimentales de los porcentajes de <i>G0/G1</i> -AxFUCCI y <i>S/G2</i> -AxFUCCI que expresan las células	33
3.5	Las células ependimales exhiben sincronía del ciclo celular durante la regeneración de la médula espinal	34
3.6	Convergencia de la respuesta del ciclo celular entre las simulaciones del modelo y los datos de AxFUCCI	38
3.7	Modelado de distribuciones espaciotemporales de las divisiones de células ependimales durante la regeneración de la médula espinal del axolotl	39
3.8	Distribuciones espacio-temporales de las células AxFUCCI de transición durante la regeneración de la médula espinal del axolotl	40

4.1	Modelo de crecimiento de la médula espinal ilesa y en regeneración basado en el acortamiento de las fases <i>G1</i> y <i>S</i> de las células para las dos variantes del modelo: sincronización parcial de la fase <i>G1</i> y mapeo de la fase <i>G1</i>	43
4.2	Tiempo ahorrado por las células en fase <i>G1</i> para el modelo de mapeo en comparación con el modelo de sincronización parcial de la fase <i>G1</i>	44
4.3	Mapas de calor de proporción de células en <i>G1/G0</i> y <i>S/G2</i> en función de la posición espacial para el modelo de mapeo y sincronización parcial de <i>G1</i> en comparación a los datos experimentales.	45
4.4	Crecimiento de la médula espinal para distintos grados de sincronización en la posición del ciclo celular al inicio de la simulación (distintos valores del parámetro <i>p</i>)	46
4.5	Mapas de calor de proporción de células en <i>G1/G0</i> y <i>S/G2</i> en función de la posición espacial para distintos grados de sincronización en la posición del ciclo celular al inicio de la simulación (distintos valores del parámetro <i>p</i>) . .	48
5.1	Esquema aproximado de cómo se modela a las células en de un tejido en 2D en un modelo de vértices	51
5.2	Representación gráfica de un tejido modelado mediante el modelo de vértices	59
5.3	Esquema del intercambio T1	60
5.4	Esquema del intercambio T2	67
5.5	Esquema del intercambio T3	69
5.6	Esquema de la implementación de la división celular.	76
6.1	Mapeo de las simulación 2D a un cilindro que representa la médula espinal .	106
6.2	Simulación de la médula espinal del axolotl en 2D mediante SysVert sin presencia de la señal	110
6.3	El modelo 2D reproduce la reducción del crecimiento experimental cuando se impide la aceleración de la proliferación celular	111
6.4	Simulación de la médula espinal del axolotl en 2D mediante SysVert en presencia de la señal	112
6.5	El modelo 2D reproduce los datos experimentales de la médula espinal del axolotl en regeneración	112
6.6	Las células en el escenario con reclutamiento sufren una mayor compresión que las células en el escenario sin reclutamiento	114
6.7	Área de las células como función del tiempo y el espacio para los dos escenarios	115
6.8	Cuando las células se hacen más deformables el tejido se comprime por debajo de lo reportado experimentalmente	116
6.9	Cuando las células se comprimen por debajo del área reportada experimentalmente el modelo no es capaz de reproducir el crecimiento experimental de la médula espinal	117
B.1	Determinación de fragmentos de Cdt1 y Gmnn para el reportero AxFUCCI .	141
B.2	Determinaciones de intensidad de fluorescencia de AxFUCCI a través del ciclo celular	142

B.3	Cuantificación de ADN por citometría de flujo para validar la funcionalidad AxFUCCI	143
B.4	Validación de la funcionalidad AxFUCCI mediante análisis de tejidos <i>in vivo</i>	144
B.5	El protocolo de limpieza de tejidos no altera la longitud de la médula espinal	146
B.6	Las células ependimales exhiben sincronía del ciclo celular durante la regeneración de la médula espinal	147

Índice de cuadros

2.1	Parámetros del modelo	24
-----	---------------------------------	----

Índice de código

5.1	Definiciones del cálculo de fuerzas de tensión de línea (<i>vertex_line_force</i>), área (<i>vertex_area_force</i>) y perímetro (<i>vertex_perimeter_force</i>) del tejido en el archivo <i>forces.py</i>	52
5.2	Archivo <i>solver.py</i> donde se realiza cada iteración temporal de las simulaciones, llamando a las funciones responsables del cálculo de las fuerzas, las transiciones y las divisiones	54
5.3	Definiciones de los intercambios T1 del tejido en el archivo <i>swaps.py</i> . . .	60
5.4	Definiciones de los intercambios T2 del tejido en el archivo <i>swaps.py</i> . . .	67
5.5	Definiciones de los intercambios T3 del tejido en el archivo <i>swaps.py</i> . . .	70
5.6	Implementación de la división celular en <i>cell_division_random_angle.py</i>	75
5.7	Definición de la clase Lattice en el archivo <i>lattice.py</i>	79
5.8	Definición de la clase Cell en el archivo <i>cells.py</i>	85
5.9	Definición de la clase Edge en el archivo <i>edges.py</i>	96
5.10	Definición de la clase Vertex en el archivo <i>vertex.py</i>	100
6.1	Párametros de la geometría del tejido en el archivo <i>jwrite.py</i>	106
6.2	Párametros de los tipos celulares del tejido en el archivo <i>jwrite.py</i>	107
6.3	Tensiones de línea del tejido en el archivo <i>jwrite.py</i>	108
6.4	Párametros de la señal en el archivo <i>jwrite.py</i>	108
6.5	Párametros generales de la simulación en el archivo <i>jwrite.py</i>	109

CAPÍTULO 1

Introducción

Nuestros antepasados observaron la complejidad y la belleza de la vida, y comprendieron que tenía que existir un gran creador.

—Carl Sagan

1.1 Modelando la complejidad biológica: biología de sistemas

Tradicionalmente la ciencia fue dividida en diferentes disciplinas. Esto se debe a que ante la complejidad de los diferentes procesos que tienen lugar en la naturaleza fue necesario dividirlos de algún modo. Al dividirla de esta manera, se comenzaron a estudiar fenómenos simples. El organismo vivo era descompuesto en células, sus actividades en procesos fisiológicos y por último fisicoquímicos. Existen aspectos de la naturaleza que no pueden ser explicados solamente utilizando una disciplina. En cambio, la concepción organísmica (o sistémica) es básica para la biología moderna. Es necesario estudiar no sólo las partes y procesos aislados, sino también resolver los problemas decisivos hallados en la organización y el orden que los unifican, resultantes de la interacción dinámica de partes y que hacen al diferente comportamiento de éstas cuando se estudian aisladas o dentro del todo. En muchos fenómenos biológicos, pero también de las ciencias sociales y del comportamiento, resultan aplicables expresiones y modelos matemáticos que son implementados en lenguajes de programación computacional para poder ser resueltos. La similitud estructural entre estos modelos y su isomorfismo en diferentes campos se tornaron ostensibles, y en el centro quedaron precisamente problemas de orden, organización, totalidad, teleología, etc., excluidos programáticamente de la ciencia mecanicista. Tal fue, la idea de la “teoría general de los sistemas” (Von Bertalanffy, 1976). En este marco la biología, la física y la computación se fueron combinando de forma de poder explicar ciertos fenómenos que previamente no fueron explicados, dando así lugar a la biología de sistemas.

Durante el siglo veinte, podemos encontrar ejemplos notables de esfuerzos intelectuales que combinaron biología y física. Un ejemplo destacable son los trabajos de D’Arcy Thompson quien sostenía en su libro “*On growth and form*” que para estudiar la forma final de los organismos no es suficiente con estudiar los aspectos genéticos del proceso sino que además es necesario considerar los efectos de las leyes físicas (Thompson *et al.*,

1917) Nicolas Rashevsky es otro antecedente, quien aplicó métodos matemáticos para el estudio de la dinámica del transporte en tejidos (Rashevsky *et al.*, 1938). En el campo de la ecología podemos encontrar a Alfred Lotka o Vito Volterra (Murray *et al.*, 2003) que propusieron sus famosas ecuaciones para modelar la dinámica predador-presa de poblaciones de animales. En el campo de la neurociencia podemos mencionar a Alan Lloyd Hodgkin y Andrew Fielding Huxley (Hodgkin, *et al.*, 1952) y su modelo de potenciales de acción en células excitables. Finalmente, resaltaremos los trabajos de Alan Turing (Turing, 1952), Hans Meinhardt y Alfred Gierer (Gierer & Meinhardt, 1972) en el estudio de patrones durante el desarrollo o regeneración de tejidos. Turing fue el primero en proponer un modelo para la formación de patrones a partir de un mecanismo de reacción-difusión. Luego Meinhardt y Gierer desarrollaron un modelo similar aunque en forma independiente. Los trabajos mencionados fueron utilizados para estudiar diferentes aspectos inherentes a la formación y el establecimiento de tejidos en los seres vivos. Sin embargo, las características dinámicas y estructurales de los tejidos durante el desarrollo y la regeneración fueron estudiadas mayoritariamente mediante aproximaciones de biología cualitativa hacia fines del siglo pasado.

En años más recientes, se han empezado a estudiar utilizando abordajes cuantitativos diferentes procesos, tanto de señalización como mecánicos en tejidos durante el desarrollo o durante procesos regenerativos. Los avances en métodos experimentales, tales como la microscopia confocal, microscopia de superresolución, edición génica, etc., permitieron obtener información con alta resolución espacial y temporal en los tejidos. Esto dio lugar a la aparición de diferentes modelos computacionales capaces de explicar los resultados experimentales.

En esta tesis presentaré dos modelos matemáticos: uno de ellos, espacialmente unidimensional, inspirado en el modelo de esferas duras y el otro, espacialmente bidimensional, de un modelo computacional llamado *celular vertex model*, que en esta tesis llamaré “modelo de vértices” (Fletcher *et al.*, 2013). Estos modelos me permitieron modelar la regeneración de tejidos animales, en particular la regeneración de la médula espinal del axolotl.

1.2 Regeneración de tejidos animales

Todos los organismos han evolucionado desarrollando estrategias que les posibilitan mantener su forma y funciones a lo largo de su vida adulta. El término regeneración es usado para referirse a muchos fenómenos que van desde la reconstitución de las células sanguíneas hasta el restablecimiento completo del cuerpo de organismos como la planaria. En esta tesis, con regeneración me referiré al proceso que permite a un organismo recuperar la función de un órgano o estructura que ha sido dañado/a. Una pregunta fascinante, aún sin respuesta satisfactoria es por qué la amputación de un tejido o de un órgano induce una respuesta regenerativa en algunos organismos.

La comprensión de los mecanismos moleculares que regular los procesos regenerativos mediante el estudio de los modelos animales poseedores de estas sorprendentes capacida-

des, podrían permitir dar un salto en el campo de la medicina regenerativa en humanos. Si bien los mecanismos y los tipos celulares involucrados en el proceso de regeneración de estos organismos son de una naturaleza variada, es evidente que hay señales comunes. Un conjunto de señales provistas por el sistema inmune parecen actuar como una primera respuesta a la injuria. Posteriormente, se requiere la activación de redes de señalización conocidas por su importancia en la regulación del desarrollo embrionario (Chernoff, 1996; Clarke and Ferretti, 1998), y aunque algunas cascadas de señalización han sido involucradas en estos procesos regenerativos (como BMP o Wnt, por ejemplo), aún no se han identificado mecanismos claros que conecten la amputación con la respuesta celular y molecular en estos tejidos. Es razonable pensar que los mecanismos de señalización responsables de la formación de patrones, proliferación y diferenciación subyacentes a los procesos de regeneración recapitulen los eventos que tienen lugar en el desarrollo embrionario. De esta forma, la capacidad de mantener o re-acceder a patrones de las siguientes señales químicas y factores de crecimiento que son cruciales durante el desarrollo embrionario podrían ser fundamentales para generar una respuesta regenerativa exitosa a una injuria tisular.

Concretamente, el resultado más común de las lesiones de la médula espinal y el cerebro en mamíferos adultos, incluidos los humanos, es la pérdida significativa de tejido. Esta pérdida conduce a condiciones paralizantes ya que, una vez que se pierde el tejido neural, no se puede reemplazar debido a la capacidad aparentemente limitada del sistema nervioso central (SNC) de los mamíferos para repararse a sí mismo (Banfi *et al.*, 2000; Huibregtse *et al.*, 2000). Los requisitos clave para una regeneración exitosa de la médula espinal son la presencia de un número adecuado de neuronas y glía, ya sea como resultado de la producción *de novo* a partir de células madre neurales o de una supervivencia celular significativa después de una lesión, y la capacidad de estas neuronas para desarrollar axones que pueden re-enervar con éxito sus objetivos.

1.3 Modelo de estudio: axolotl (*ambystoma mexicanum*)

El axolotl (Figura 1.1) tiene la notable capacidad de regenerar la médula espinal lesionada (Freitas, Yandulskaya & Monaghan, 2019; Tazaki *et al.*, 2017; Chernoff *et al.*, 2003), y por lo tanto representa un sistema único para estudiar los mecanismos de regeneración exitosa de la médula espinal. La médula espinal de las salamandras es esencialmente similar a la de otros vertebrados. Consiste de materia gris, conteniendo los somas de las células ependimales, neuronas y astrocitos; y la materia blanca, rodeando la materia gris, conteniendo fibras ascendentes y descendentes, así como oligodendrocitos. Quizá una de las características más salientes de la médula espinal de las salamandras y actores clave en el proceso de regeneración son las células ependimales, que recubren el canal central y conservan el potencial de las células madre neurales durante toda la vida (Becker, Becker y Hugnot, 2018).



Figura 1.1: axolotl (*Ambystoma mexicanum*).

1.4 Potencial regenerativo de las células endodermales en la médula espinal del axolotl

El epitelio endodermario permanece proliferativo y sufre neurogénesis continuamente bajo condiciones homeostáticas en el adulto. Desde el punto de vista morfológico, las células endodermales mantienen su polaridad apico-basal y conforman un neuroepitelio pseudo-estratificado similar al que hay en el tubo neural embrionario de los mamíferos. A nivel molecular, estas células expresan marcadores de células madre tales como SOX2, Musashi-116 y GFAP, así como marcadores de pluripotencia como OCT4 (Fahmy and Moftah, 2010; Hunter *et al.*, 1991; Moftah *et al.*, 2008; O'Hara and Chernoff, 1994; O'Hara *et al.*, 1992; Zaky and Moftah, 2014). Más allá de sus similitudes morfológicas se presume que estas células tendrían funciones similares a la de la glía radial. Podría especularse que en el mantenimiento de este fenotipo de alguna forma inmaduro subyacería su plasticidad celular y su potencial regenerativo.

1.5 Aceleración del ciclo celular durante la regeneración de la médula espinal del axolotl

En estudios anteriores de nuestro grupo, se encontró que la lesión de la médula espinal en el axolotl gatilla la reactivación de un programa similar al del desarrollo en las células endodurales, provocando una transición de divisiones celulares lentas y neurogénicas a rápidas y proliferativas (Rodrigo Albors *et al.*, 2015). Se demostró que en la médula espinal ilesa y en la región no regenerativa de la médula espinal lesionada, las células endodurales se dividen lentamente, completando un ciclo celular en $14,2 \pm 1,3$ días. Por el contrario, las células endodurales en la zona de regeneración aceleran su ciclo celular y se dividen cada $4,9 \pm 0,4$ días (Rodrigo Albors *et al.*, 2015; Rost *et al.*, 2016). Mediante el uso de un enfoque de modelado matemático, se mostró que la aceleración del ciclo celular es el principal impulsor del crecimiento regenerativo de la médula espinal, y que otros procesos como el influjo de células, los re-ordenamientos celulares y la activación de células madre neurales desempeñan papeles más pequeños (Rost *et al.*, 2016). Un análisis cuantitativo de la proliferación celular en el espacio y el tiempo permitió identificar una zona de alta proliferación que emerge 4 días después de la amputación dentro de los $800\mu m$ adyacentes al sitio de la lesión y se desplaza hacia el extremo posterior con el tiempo a medida que crece la médula espinal en regeneración (Rost *et al.*, 2016) (Fig. 2.1). Sin embargo, qué subyace a este patrón espacio-temporal preciso de proliferación celular continúa siendo un enigma. Los fenómenos de formación de patrones que ocurren durante el desarrollo pueden reproducirse cuantitativamente mediante la invocación de señales morfogénicas que se propagan desde fuentes localizadas (Morelli *et al.*, 2012). Por tanto, es concebible que la amputación de la cola desencadene una señal que se propague o difunda a lo largo de la médula espinal lesionada para acelerar el ciclo celular de las células residentes.

1.6 Objetivos de este trabajo de tesis

El objetivo general de este proyecto se constituye en entender mecánicamente la regeneración de la médula espinal del axolotl en términos de procesos celulares y moleculares o de señalización. Para ello, en esta tesis se implementa una aproximación de biología de sistemas que implica llevar a cabo modelado matemático alimentado por el análisis de datos experimentales previos y actuales de nuestros colaboradores experimentales. En particular, el propósito de la tesis es modelar las fases tempranas de la regeneración de la médula espinal del axolotl en dos niveles: un nivel celular y un nivel de señalización.

1. En una primera aproximación me propongo estudiar el proceso regenerativo con un modelo de esferas duras en una dimensión que incluya información precisa de la

posición de las células en su ciclo celular, a los fines de capturar las generalidades del proceso regenerativo.

2. A continuación me propongo validar este modelo con datos experimentales de la respuesta regenerativa de la médula espinal del axolotl, haciendo énfasis en la distribución espacio-temporal de la proliferación en dicha respuesta.
3. Luego me propongo indagar en la existencia o no de procesos de sincronización durante la respuesta regenerativa.
4. Con la intención de comenzar a abordar aspectos mecánicos de la interacción entre las células, me propongo desarrollar un modelo de vértices celular en dos dimensiones de la médula espinal del axolotl durante la regeneración.
5. Con este objetivo me propongo, además, desarrollar un paquete computacional que permita la implementación del modelo en dos dimensiones mencionado.

La estructura de esta tesis es la siguiente: en el capítulo 2, se presenta la metodología y los métodos empleados. En el capítulo 3, “Modelado de la regeneración de la médula espinal del axolotl en 1D”, desarrollo y exploro un modelo espacialmente unidimensional capaz de capturar cuantitativamente los resultados experimentales de las cinéticas de crecimiento de la médula espinal del axolotl en regeneración. En el capítulo 4, “Determinación del tránsito a través del ciclo celular mediante Fucci en axolotl *in vivo*”, presento los resultados de las determinaciones experimentales de las distribuciones espacio-temporales de las células de la médula espinal en las fases del ciclo celular obtenidas mediante AxFucci (una adaptación *de novo* de la técnica de Fucci para el axolotl) y los comparo con las predicciones del modelo 1D. En el capítulo 5, “Sincronización inducida por el reclutamiento y sincronización pre-existente a la amputación: predicciones del modelo 1D”, analizo la probable necesidad así como las consecuencias de las sincronizaciones asumidas por el modelo 1D. En el capítulo 6, “Implementación computacional del modelo de vértices: SysVert”, describo mi implementación de un modelo de vértices celular, SysVert, que posibilita el modelado de tejidos en dos dimensiones. En el capítulo 7, “Modelado de la regeneración de la médula espinal del axolotl en 2D”, utilizando la parametrización del modelo 1D, modelo la regeneración de la médula espinal del axolotl en dos dimensiones (mediante SysVert) y recupero los resultados experimentales. Adicionalmente, llevo a cabo un análisis de la compresión del tejido a nivel celular e hipotetizo su eventual papel en la interrupción del proceso de reclutamiento celular durante la respuesta regenerativa. Finalmente, discuto los resultados obtenidos que en conjunto aportan nuevos indicios acerca de donde y cuando buscar las señales responsables de dirigir la regeneración de la médula espinal.

CAPÍTULO 2

Modelado de la regeneración de la médula espinal del axolotl en 1D

La simplicidad es la máxima sofisticación.

—Leonardo da Vinci

2.1 Modelo de médula espinal en desarrollo

Teniendo en cuenta la simetría del tubo endodermal y que las células endodermales se organizan como un epitelio pseudoestratificado (Joven & Simon, 2018), modelé el eje antero-posterior (AP) de la médula espinal como una fila de células endodermales (consulte la sección de Métodos computacionales para obtener más detalles). Modelé las células endodermales como esferas rígidas de diámetro uniforme y asumí que pueden encontrarse ciclando o quiescentes y definí la fracción de células que se encuentran ciclando como la fracción de crecimiento, GF. Modelé la dinámica de proliferación de las células que se encuentran ciclando de la siguiente manera: asumí que en la condición inicial, cada ciclo la célula está en una coordenada aleatoria a lo largo de su ciclo celular, donde la coordenada inicial del ciclo celular y la longitud del ciclo celular siguen una distribución exponencial y lognormal, respectivamente. En la médula espinal, tras la división celular, i) las células hijas heredan la longitud del ciclo celular de la distribución lognormal de la madre y ii) las células hijas se trasladan posteriormente, desplazando las células posteriores a ellas. Esta última característica del modelo es la implementación de lo que definimos anteriormente como “mecanismo de empuje celular” (Rost *et al.*, 2016). Este modelo predice que después de un tiempo de aproximadamente la duración de un ciclo celular, ocurrirán eventos mitóticos a lo largo del eje AP y contribuirán al crecimiento de la médula espinal (Figura 2.1A).

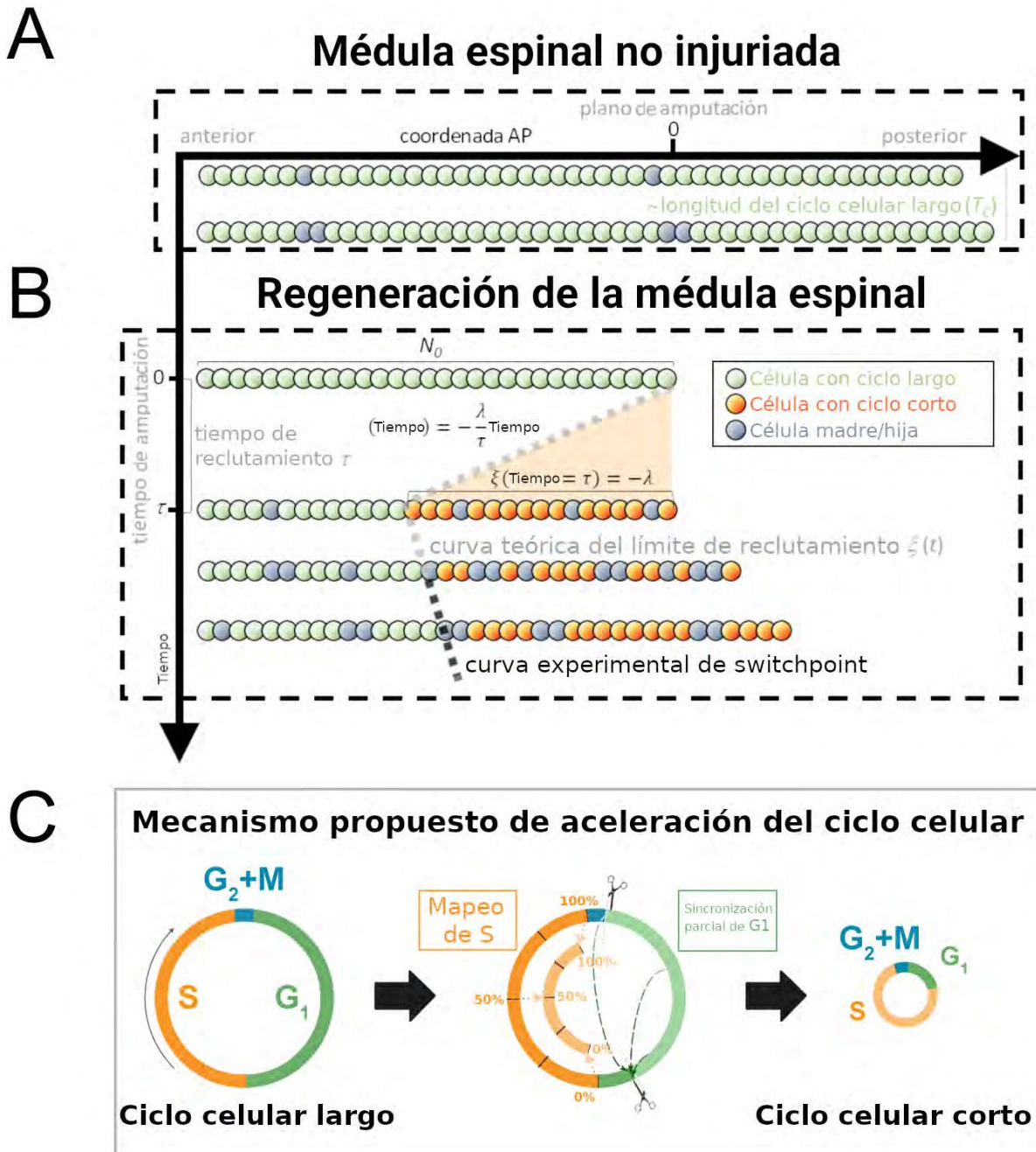


Figura 2.1: Modelo de crecimiento de la médula espinal ilesa y en regeneración basado en el acortamiento de las fases G1 y S de las células.

Figura 2.1: A) Médula espinal ilesa. Modelo 1D de células endodermiales alineadas a lo largo del eje AP . En el tejido sano, las células endodermiales ciclan con una duración larga del ciclo celular (ver Figura 2.2) y cuando se dividen las células “empujan” posteriormente y el tejido de la médula espinal crece. Como ejemplo, dos células madre (en azul) en la fila superior dan lugar a cuatro células hijas en la segunda fila (también en azul) después de una división. Esto da como resultado un crecimiento de dos diámetros celulares dentro de un período de tiempo de aproximadamente un ciclo celular largo. **B) Regenerador de la médula espinal.** Después de la amputación (coordenada AP y tiempo igual a cero) se libera una señal anteriormente desde el plano de amputación durante un tiempo τ y se propaga mientras se reclutan células endodermiales residentes hasta el límite de reclutamiento teórico ξ ubicado en $-\lambda\mu m$ desde el plano de la amputación. Después de cierto tiempo, el límite de reclutamiento ξ se superpone al *switchpoint* experimental (ver Figura 2.2). **C) Los mecanismos propuestos de acortamiento del ciclo celular consisten en la omisión parcial de la fase G1 y el mapeo proporcional entre fases S largas y cortas.** En el panel central, se representan dos ejemplos (flechas verdes discontinuas) de células que estaban en la fase G1 larga (inmediatamente antes del reclutamiento) que se sincronizan (inmediatamente después del reclutamiento). Además, hay tres ejemplos (flechas naranjas punteadas) de células que estaban en la fase S larga (inmediatamente antes del reclutamiento) que se mapean proporcionalmente (inmediatamente después del reclutamiento). Todos estos ejemplos se muestran en detalle en la Figura 2.3. El diámetro de los círculos es aproximadamente proporcional a la longitud del ciclo celular de la célula..

2.2 Modelo de regeneración de la médula espinal

A continuación, eliminé las células más posteriores del tejido para modelar la amputación de la cola y la respuesta regenerativa en las restantes (N_0) células in silico (consulte la Figura 2.1B y la sección de métodos computacionales para más detalles). Asumí que la amputación desencadena la liberación de una señal que se propaga en sentido anterior desde el sitio de la lesión con velocidad constante a lo largo del eje AP , reclutando células al inducir un cambio en su ciclo celular. Establecí que el reclutamiento celular se detiene al tiempo τ , reclutando $\lambda\mu m$ de células anteriores al plano de amputación. Anoto la posición AP de la célula más anterior reclutada por la señal como $\xi(t)$, y llamo a esto el límite de reclutamiento, tal que $\xi(t = \tau) = -\lambda$. En el modelo, todas las células anteriores a las células ubicadas en $\xi(t)$ no se reclutan y continúan ciclando lentamente durante las simulaciones (Figura 2.2). Por el contrario, las células posteriores a $\xi(t)$ se reclutan en un tiempo t dentro del intervalo $0 \leq t \leq \tau$ y modifican irreversiblemente su

ciclo de acuerdo con su fase del ciclo celular en el momento del reclutamiento.

Debido a que previamente demostramos que la duración de las fases $G2$ y M no cambia con la amputación (Rodrigo Albors *et al.*, 2015), asumí que las células en $G2$ o M dentro de la zona de reclutamiento continuará ciclando como antes (Figura 2.1C, sección Métodos computacionales). En contraste, dado que demostramos que las células en regeneración pasan por fases $G1$ y S más cortas que las células no regenerativas ($G1$ se acorta de 152 ± 54 horas a 22 ± 19 horas; S acorta de 179 ± 21 horas a 88 ± 9 horas, Rodrigo Albors *et al.*, 2015), razoné que la señal instruye a las células reclutadas para acortar $G1$ y S , acortando de manera efectiva su ciclo celular.

Para explicar cómo las células endodermales pueden acortar las fases $G1$ y S en respuesta a la señal de lesión, concibo un mecanismo de acortamiento de $G1$ en el que se salta cierta parte de esta fase del ciclo celular. Implementé este mecanismo de la siguiente manera (Figura 2.1C, Figura 2.3, Computacional métodos sección 1.1.1): si en el momento del reclutamiento la célula está en $G1$, hay dos posibles transformaciones de sus coordenadas. Si la coordenada del ciclo celular está ubicada antes de la diferencia entre la longitud $G1$ larga (no regenerativa) y la longitud $G1$ acortada (regenerativa), el reloj celular se reinicia; es decir, su coordenada transformada del ciclo celular será el comienzo de la fase $G1$ acortada en el siguiente paso de simulación (inmediatamente después de las fases $G2 + M$). Por el contrario, si la coordenada original del ciclo celular de la célula se encuentra situada después de la diferencia entre la longitud $G1$ larga y la longitud $G1$ acortada, no hay cambio en el tiempo para entrar en fase S . La diferencia entre las longitudes $G1$ largas y cortas constituye una especie de umbral. Si la célula en $G1$ se encuentra antes de este valor, salta y si está después, continúa ciclando como antes. Este mecanismo de salto en $G1$ predice una sincronización parcial del ciclo celular a medida que las células transitan a través de $G1$ (Figura 2.1 - Figura 2.7) - una implicación importante que probamos experimentalmente más tarde.

Debido a que todo el ADN debe duplicarse antes de la división celular, consideré un mecanismo diferente para el modelado del acortamiento de la fase S : si la coordenada del ciclo celular pertenece a S en el momento del reclutamiento, la nueva coordenada del ciclo celular de esta célula se mapeará proporcionalmente a la coordenada correspondiente de una fase S reducida en el siguiente paso de simulación (Figura 2.1C, Figura 2.3, sección Métodos computacionales). Por ejemplo, si la célula reclutada está en un 40% en su fase S (larga) cuando la señal llega, estará en el 40% de su fase S corta en el siguiente paso de simulación. Las células hijas de las células reclutadas heredan fases $G1$ y S cortas de sus progenitores y en consecuencia, tienen longitudes de ciclo celular cortas (Figura 2.1C). Finalmente, asumí que el reclutamiento de una célula inactiva induciría su progreso de $G0$ a $G1$ después de un retraso arbitrario (métodos computacionales). Para parametrizar las duraciones de la fase celular de las células cíclicas reclutadas y no reclutadas, la fracción de crecimiento y geometría celular, utilizamos nuestros datos experimentales anteriores de regiones de regeneración y no regeneración de la médula espinal del axolotl (Rodrigo Albors *et al.*, 2015) (sección de métodos computacionales).

El modelo predice que si esperamos un tiempo similar a la duración del ciclo celular corto, observaremos mayor proliferación posterior a ξ que anterior a él. En particular, si este modelo es correcto, la predicción para ξ (Figura 2.1B) debe coincidir con la

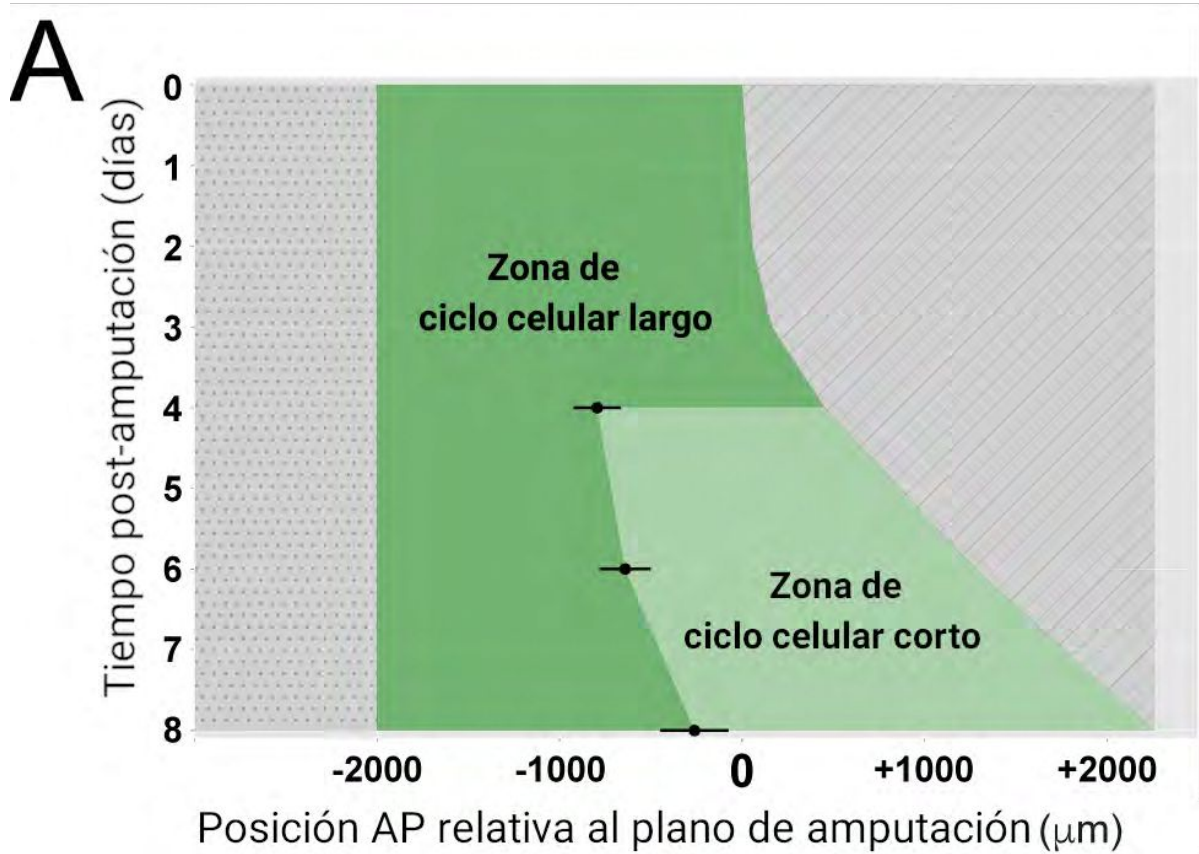


Figura 2.2: Distribución espacio-temporal de la proliferación celular durante la regeneración médula espinal axolotl. *switchpoint* experimental (puntos negros) que separa áreas de baja proliferación celular (zona de ciclo celular largo, verde oscuro) debido a una alta proliferación celular (zona de ciclo celular corto, verde claro) a lo largo del eje anterior-posterior (AP) (representado horizontalmente) de la médula espinal axolotl. La región rayada marca el espacio fuera del embrión mientras la región punteada marca la parte no afectada del embrión (adaptado de la Fig.2 F'' de Rost *et al.*, 2016). Cada valor del *switchpoint* se determinó como el borde que separa dos zonas espacialmente homogéneas asumidas por un modelo matemático, que se ajustó a la distribución espacial experimental de la fracción de crecimiento y el índice mitótico a lo largo de la médula espinal del axolotl (Rost *et al.*, 2016).

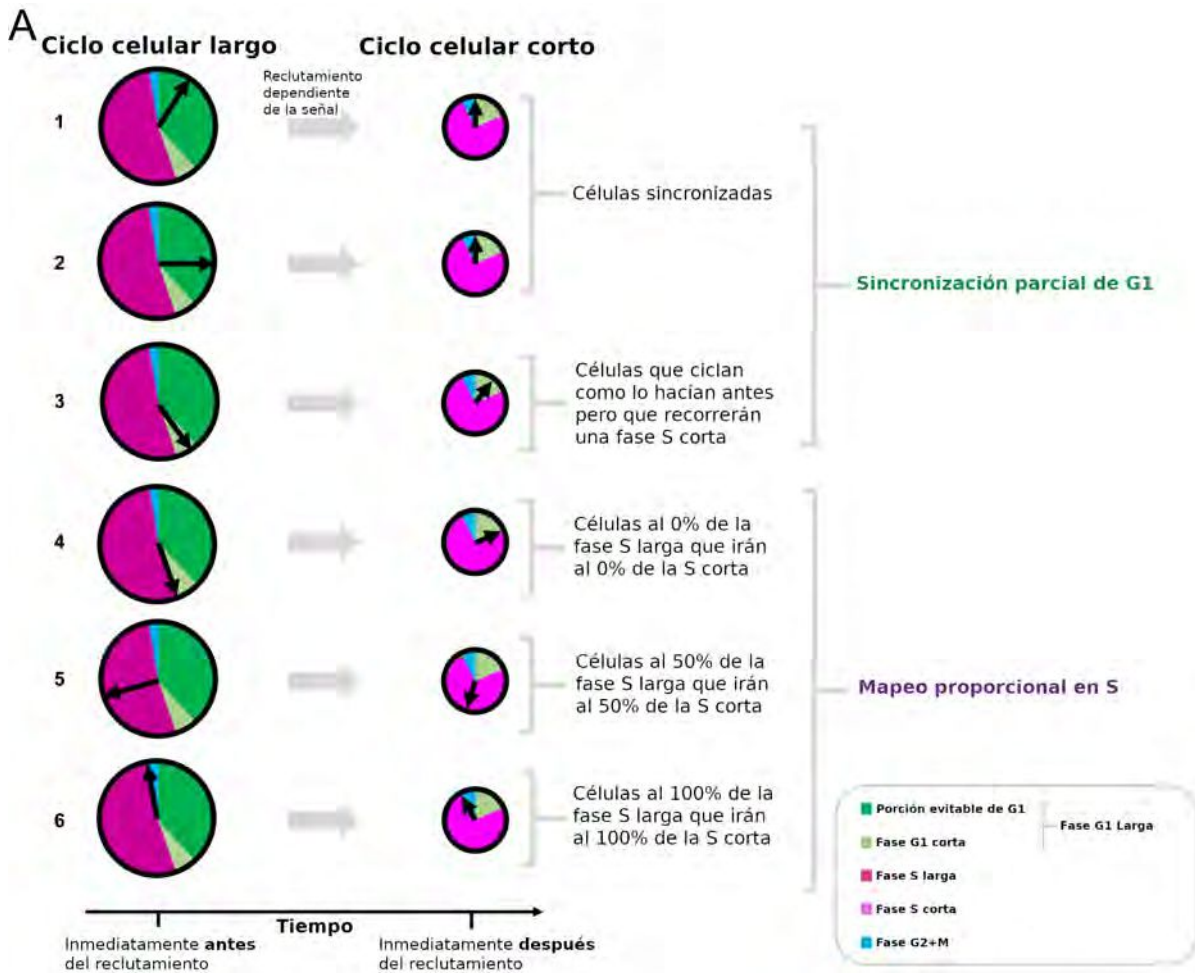


Figura 2.3: Explicación adicional de la transformación de coordenadas del ciclo celular invocada por el modelo. Las células en la fase G1 al momento del reclutamiento pueden omitir el resto de la fase, dependiendo de dónde están ubicados en la fase G1 larga. Si la coordenada de su ciclo celular está antes de la diferencia entre las fases G1 larga y corta, se saltan esta parte inicial de la fase larga G1 y se sincronizan al comienzo de la fase G1 corta (1, 2). Si la coordenada de su ciclo celular es igual o posterior a la diferencia entre las fases G1 larga y corta, continúan ciclando como antes, pero luego pasarán por una fase S corta (3). Esto genera un mecanismo de sincronización parcial (Ver sección Métodos computacionales). Si las células están en fase S al momento del reclutamiento, las células siguen un mapeo proporcional (4 - 6). Esencialmente, las coordenadas del ciclo celular dentro de S se escalan a la fase S corta (consulte la sección de Métodos computacionales).

curva experimental del *switchpoint* que separa la región de alta proliferación de la de baja proliferación a lo largo del eje AP de la médula espinal del axolotl durante la regeneración (Figura 2.2).

2.3 Detalles del modelado de desarrollo y regeneración de la médula espinal axolotl

Modelé la médula espinal como una fila densamente empaquetada de células endodermales. Dado que asumo todas las células como esferas rígidas idénticas, el modelo implica efectivamente sólo una dimensión espacial: eje antero-posterior (AP) de la médula espinal. Asumí que las células se encuentran ciclando o inactivas, donde la fracción de células que se encuentran transitando el ciclo es la fracción de crecimiento GF . Consideré que cada célula ubicada en la posición x_i en el tiempo t prolifera con una cierta longitud de ciclo celular aleatoria distribuido lognormalmente $T_i(x_i, t)$ y tiene una cierta edad dentro de su ciclo celular, $C_i(x_i, t)$, definida como una coordenada a lo largo del ciclo celular o reloj ($0 \leq C_i(x_i, t) < T_i(x_i, t)$). En la condición inicial, cada célula ciclando tiene una edad aleatoria $C_i(x_i, t = 0) = C_i^0$ a lo largo de la longitud del ciclo celular en particular

$T_i(x_i, t)$, donde la distribución de C_i^0 viene dada por $(\frac{\ln 2}{T_i(x_i, t)})2^{1 - \frac{2C_i^0}{T_i(x_i, t)}}$. A medida que avanza el tiempo, cada célula que cicla aumenta su reloj $C_i(x_i, t)$ de forma determinista hasta que alcanza su correspondiente valor $T_i(x_i, t)$. En este preciso momento, la célula se divide y una hija hereda la coordenada AP de su madre mientras que el otro está intercalado entre la primera hija y la célula vecina posterior. Esta última característica del modelo es la implementación de lo que definimos anteriormente como “mecanismo de empuje celular” (Rost *et al.*, 2016). Después de la división celular, las células hijas reinician sus relojes y $C_i(x_i, t) = 0$. Este modelo predice que después de un tiempo de aproximadamente una duración de ciclo celular, los eventos mitóticos ocurrirán a lo largo del eje AP, contribuyendo al crecimiento de la médula espinal durante el desarrollo (Figura 2.1A).

Para estudiar la evolución del tejido en un escenario regenerativo, me centré en la respuesta del tejido a una amputación modelada simplemente eliminando las células más posteriores. Modelé la respuesta regenerativa en las N_0 células restantes asumiendo que la amputación desencadena la liberación de una señal, que se propaga con velocidad constante en dirección anterior sobre el eje AP mientras recluta células endodermales. Supuse que el reclutamiento de células se detiene a tiempo τ , dejando $\lambda\mu m$ de células anteriores al plano de amputación reclutadas y una velocidad de reclutamiento $-\lambda/\tau$ durante el intervalo $0 \leq t \leq \tau$. Llamé a la posición AP de la célula más anterior reclutada por la señal como $\xi(t)$, el límite de reclutamiento, tal que, $\xi(t = \tau) = -\lambda$.

Dado que las células endodermales regenerativas acortan las fases G1 y S sin alterar

las fases $G2$ y M que conducen a una aceleración del ciclo celular (Rodrigo Albors *et al.*, 2015), asumí que el reclutamiento inducido por señales instruye a las células endodimales en regeneración con precisión para reducir las fases $G1$ y S , acortando eficazmente su ciclo celular (Figura 2.1 C). Represento aquí como $G1^{long}$ y S^{long} ($G1^{short}$ y S^{short}) la duración de las fases correspondientes para las células endodimales de animales ilesos (animales en regeneración). Anoto con T^{long} y T^{short} a la longitud del ciclo celular de las células endodimales ilesas y regenerativas de axolotl, respectivamente.

Note que una célula i que se encuentra ciclando cuya posición x_i es anterior al límite de reclutamiento ($x_i < \xi(t)$) no es reclutada y tiene una longitud de ciclo celular $T_i(x_i, t)$ igual a T^{long} , es decir, continua ciclando lentamente durante las simulaciones (Figura 2.2B). En contraste, una célula ciclando i cuya posición x_i es posterior al límite de reclutamiento ($x_i \geq \xi(t)$) dentro del intervalo de tiempo $0 < t \leq \tau$ es reclutada y en consecuencia, tiene una longitud de ciclo celular $T_i(x_i, t)$ igual a T^{short} . La progenie de las células reclutadas (células no reclutadas) tienen una longitud de ciclo celular extraída de la misma distribución lognormal de T^{short} (T^{long}) (Figura 2.1C).

Supuse que el reclutamiento de una célula i ubicada en la posición x_i en el tiempo t induce un efecto de transformación en la coordenada de su ciclo celular $C_i(x_i, t) \rightarrow C'_i(x_i, t)$, donde $C_i(x_i, t)$ y $C'_i(x_i, t)$ son las coordenadas del ciclo celular original y transformado, respectivamente. Esto significa que el ciclo celular de estas células experimentan una transformación de coordenadas, modificando su ciclo de acuerdo con la fase del ciclo celular en la que se encuentran en el momento del reclutamiento, como describimos a continuación en las siguientes subsecciones (Figura 2.2B):

2.3.1 Cuando las células están en la fase $G1$ en el momento del reclutamiento

Supusimos que si en el momento de la amputación t , una celda i se encuentra en una coordenada del ciclo celular $C_i(x_i, t)$ dentro de $0 \leq C_i(x_i, t) < G1^{long} - G1^{short}$, la nueva coordenada del ciclo celular es la siguiente (Figura 2.1C y Figura 2.3):

$$C'_i(x_i, t) = 0 \quad (2.1)$$

lo que induciría a una sincronización. Por el contrario, si en el momento de la amputación t , una célula i se encuentra en una coordenada del ciclo celular $C_i(x_i, t)$ dentro de $G1^{long} - G1^{short} \leq C_i(x_i, t) \leq G1^{long}$, la nueva coordenada del ciclo celular es:

$$C'_i(x_i, t) = C_i(x_i, t)(G1^{long} - G1^{short}) \quad (2.2)$$

Es decir, estas células continúan ciclando como antes. En conjunto, las células en $G1$ resultan parcialmente sincronizados (Figura 2.1C y Figura 2.3).

2.3.2 Cuando las células están en la fase S en el momento del reclutamiento

Teniendo en cuenta que en la fase S todo el ADN debe estar duplicado para que ocurra la división celular, consideré un mecanismo diferente para modelar el acortamiento de la fase S basado en un mapeo proporcional. La nueva coordenada del ciclo celular de esta célula se asigna proporcionalmente a la coordenada correspondiente de una fase S abreviada en el siguiente paso de simulación. Por lo tanto, asumí que si en el momento de la amputación t , una célula i estaría en la fase S, es decir, en una coordenada del ciclo celular $C_i(x_i, t)$ dentro de $G1^{long} \leq C_i(x_i, t) \leq G1^{long} + S^{long}$, la coordenada transformada del ciclo celular es relativa a la longitud de la fase S es invariante (Figura 2.1C y Figura 2.3):

$$\frac{C'_i(x_i, t)G1^{short}}{S^{short}} = \frac{C_i(x_i, t)G1^{long}}{S^{long}} \quad (2.3)$$

Como consecuencia, la coordenada del ciclo celular transformada es la siguiente:

$$C'_i(x_i, t) = (C_i(x_i, t)G1^{long})\frac{S^{short}}{S^{long}} + G1^{short} \quad (2.4)$$

2.3.3 Cuando las células están en la fase G0 en el momento del reclutamiento

Supuse que si una célula i reclutada está inactiva en el momento t de reclutamiento, es decir, en G0, pasa de esta fase a la fase G1-corta después de un cierto retardo t_{G0-G1} .

$$C'_i(x_i, t + t_{G0G1}) = 0 \quad (2.5)$$

2.4 El crecimiento regenerativo de la médula espinal se puede explicar por una señal que actúa durante las 85 horas después de la amputación y recluta células dentro de los $828\mu m$ anteriores al plano de amputación

Para evaluar si el modelo podría explicar el crecimiento regenerativo del tubo ependimal y estimar los parámetros libres, ajusté $\xi(t)$ al *switchpoint* experimental (Rost *et al.*, 2016). Específicamente, seguí un método de cálculo bayesiano aproximado de inferencia, simplificado por el uso del paquete pyABC (Klinger, Rickert & Hasenauer, 2018). El modelo reprodujo con éxito el *switchpoint* experimental con los mejores parámetros de ajuste $N0 = 196 \pm 2$ células, $\lambda = 828 \pm 30\mu m$ y $\tau = 85 \pm 12$ horas (Figura 2.7A; las distribuciones posteriores de los parámetros obtenidas después de la convergencia se muestran en la Figura 2.4 y consulte la sección de métodos computacionales para obtener más detalles). Curiosamente, un análisis clonal del modelo muestra que mientras que las células más anteriores están ligeramente desplazadas de su posición original, las células ubicadas cerca al plano de amputación terminan en el extremo posterior de la médula espinal regenerada (Figura 2.5A), de acuerdo con las trayectorias celulares experimentales observadas durante la regeneración de la médula espinal del axolotl (Rost *et al.*, 2016). Además, la velocidad de un clon aumenta monótonicamente con su coordenada AP (Figura 2.5B), también en línea con los datos experimentales (Rost *et al.*, 2016). Estos resultados sugieren que las células conservan su posición relativa a lo largo del eje AP. Por lo tanto, al graficar la posición relativa de cada clon al crecimiento del tejido correspondiente menos el límite de reclutamiento $\xi(t)$, observó que esta cantidad normalizada se conserva en el tiempo, una huella del comportamiento de escalado característica de la regeneración (Figura 2.5C). Es importante destacar que con la parametrización que conduce al mejor ajuste del *switchpoint* experimental, predijimos cuantitativamente la evolución temporal del crecimiento regenerativo que se observó in vivo (Rost *et al.*, 2016) (Figura 2.7B).

Mi modelo asume que las células son esferas rígidas de diámetro uniforme fijado a partir de la longitud media de células ependimales medidas a lo largo del eje AP (Rost *et al.*, 2016). Para probar si esta simplificación podría afectar nuestra predicción del crecimiento de la médula espinal regenerativa, repetimos las simulaciones pero reemplazando la longitud media de las células por longitudes de células más grandes y más pequeñas posibles dentro de un intervalo de 99% de confianza basado en datos anteriores (Rost *et al.*, 2016). El crecimiento de la médula espinal predicho bajo estos dos escenarios extremos difícilmente podrían distinguirse de la predicción anterior (Figura 2.6 A, B). Obtuve

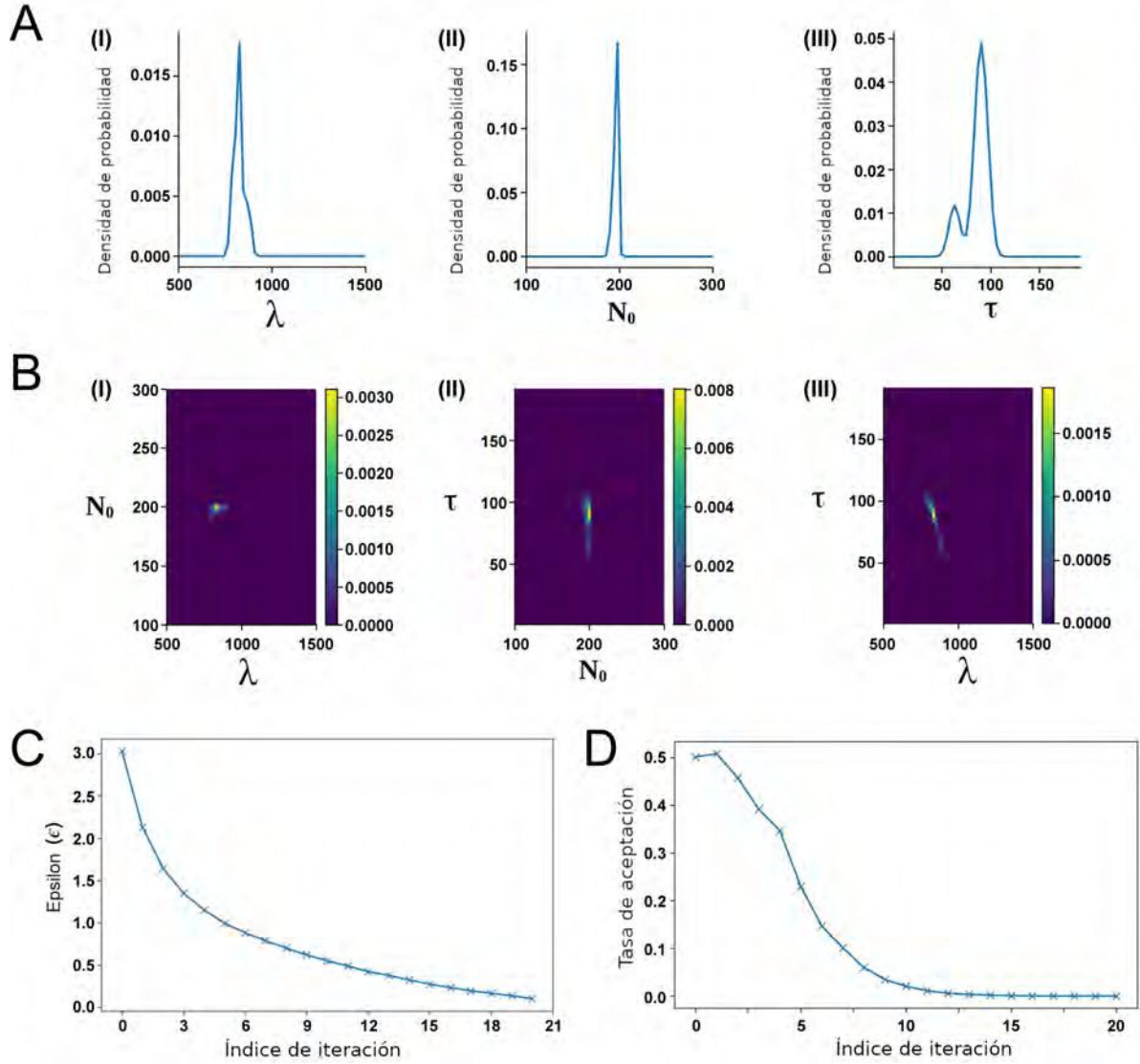


Figura 2.4: Aproximación de cálculo bayesiano (ABC) ajuste del modelo de límite de reclutamiento a los datos experimentales del *switch-point*. (A) Distribuciones posteriores de los parámetros λ (I), N_0 (II) y τ (III). (B) Distribuciones posteriores combinadas de N_0 y λ (I), τ y N_0 (II) así como τ y λ (III). (C y D) Convergencia del ajuste representada por la tolerancia de ajuste ϵ (B) y la tasa de aceptación (D) versus el índice de iteración. (Ver Métodos computacionales para más detalles).

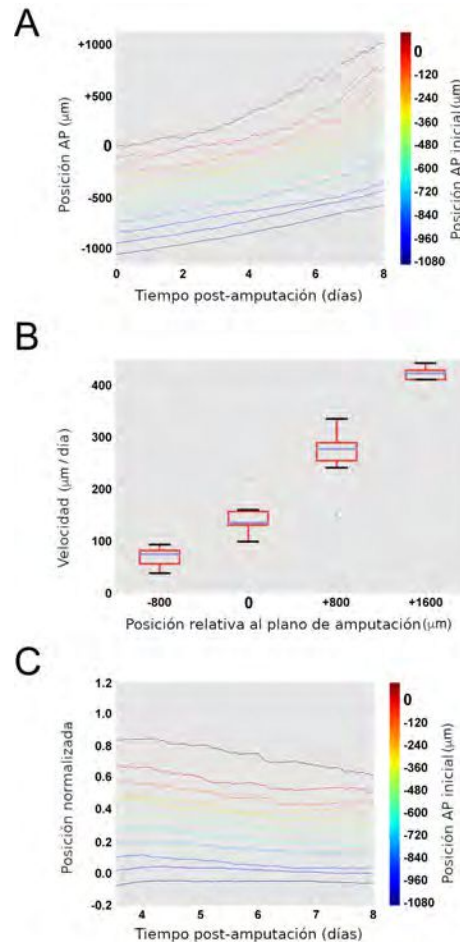


Figura 2.5: El modelo contempla el mecanismo de empuje de la célula: cuanto más posterior está una célula, cuanto más rápido se mueve. A) Las células ubicadas cerca del plano de amputación terminan en el extremo posterior del tejido regenerado modelado. B) La velocidad de los clones aumenta monótonicamente con la coordenada AP. Representación de diagrama de caja que muestra la mediana de las velocidades de los clones agrupadas cada $800\mu m$ C) **Comportamiento escalado:** las células clonadas conservan su orden espacial original. La posición relativa de cada clon al tejido altamente proliferante delimitado por su el crecimiento y el límite de reclutamiento se mantienen constantes en el tiempo. La figura muestra 11 simulaciones.

resultados similares cuando asumí que las células endodurales no tienen una longitud constante a lo largo del eje *AP* sino que son extraídas de una distribución normal parametrizada de la datos experimentales sobre las longitudes de las células endodurales a lo largo del eje *AP* (Figura 2.6 C).

Cuando simulo un proceso de reclutamiento rápido reduciendo el tiempo máximo de reclutamiento τ a un día mientras se mantiene la longitud máxima de reclutamiento λ constante, encuentro que el crecimiento predicho por el modelo sobrestima el crecimiento experimental (Figura 2.7 - Figura 3.2 A). Por el contrario, cuando disminuyo la velocidad de reclutamiento aumentando τ a 8 días manteniendo λ constante, observo un crecimiento menor que el observado experimentalmente (Figura 2.7 - Figura 3.2 A). La reducción de la distancia de reclutamiento máxima λ a cero imita un caso hipotético en el que la señal es incapaz de reclutar a las células anteriores al plano de amputación (Figura 2.8 B). Aumentar λ en aproximadamente un 100% sin cambiar τ (es decir, aumentar la velocidad de reclutamiento) da como resultado más células endodurales reclutadas y un crecimiento de la médula espinal más rápido que el observado in vivo (Figura 2.8 B). Estos resultados apuntan a un mecanismo de reclutamiento celular espacio-temporal preciso subyacente a la respuesta de crecimiento tisular durante la regeneración de la médula espinal del axolotl

2.5 Crecimiento de la médula espinal cuando se impide la aceleración del ciclo celular

Luego pregunté cuánto crecería la médula espinal si la aceleración del ciclo celular instruida por la señal está bloqueada. Hice uso de mi modelo y predije el crecimiento del tejido cuando las longitudes de *G1* y *S* permanecieron inalterados después de la amputación. En esta condición, todas las células se dividirían con las duraciones de las fases del ciclo celular informadas en condiciones de no regeneración (Rodrigo Albors *et al.*, 2015). Los resultados muestran que el bloqueo del reclutamiento y, por tanto, la aceleración del ciclo celular, ralentiza el crecimiento de tejido, lo que lleva a un reclutamiento de $694 \pm 77 \mu\text{m}$ en lugar de los $1.127 \pm 103 \mu\text{m}$ observados a día 6 (Figura 2C). Este resultado es consistente con reducir a cero la longitud máxima de reclutamiento λ (Figura 2.8). Curiosamente, esta consecuencia predicha por el modelo está en acuerdo con la que se informó sobre el crecimiento experimental en axolotl knock-out de Sox2, en el que la aceleración del ciclo celular no tiene lugar después de la amputación (Figura 2.7 C, Fei *et al.*, 2014).

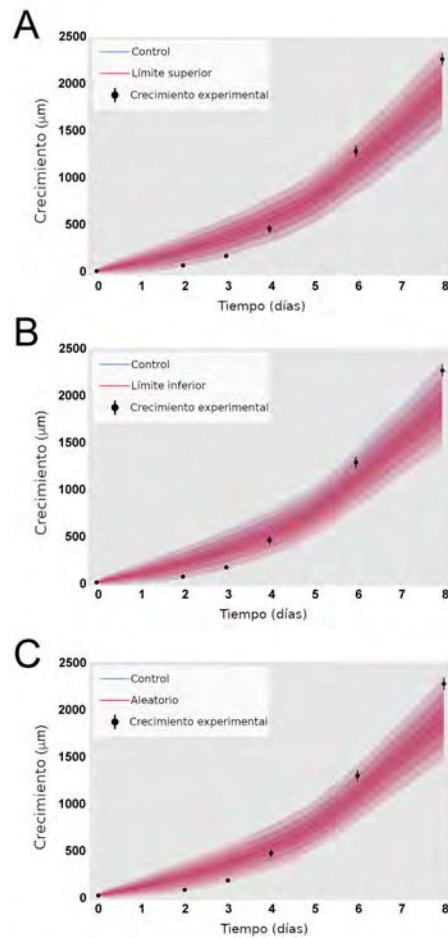


Figura 2.6: La incorporación de la variabilidad de la longitud de la célula a lo largo del eje *AP* no afecta la predicción del crecimiento de la médula espinal del axolotl. (A) Crecimiento predicho a partir del modelo que supone longitudes de las células endodimales iguales al valor medio experimental más tres sigma de la longitud medida a lo largo del eje *AP* (en rojo). (B) Crecimiento predicho a partir del modelo suponiendo longitudes de células endodimales iguales a la media experimental menos tres sigma de su longitud medida a lo largo del eje *AP* (en rojo). (C) Crecimiento predicho a partir del modelo asumiendo longitudes de células endodimales extraídas de una distribución normal parametrizada a partir de la longitud de las células medidas a lo largo del eje *AP* (en rojo). En los tres paneles se representa el crecimiento predicho del modelo asumiendo longitudes de células endodimales iguales al valor medio experimental de la longitud de las células medida a lo largo el eje *AP* (Control, en azul, mismas simulaciones que se muestran en la Figura 2.7B). La longitud experimental de las células endodimales a lo largo del eje *AP* se caracteriza por $\mu = 13,2\mu m$, $\sigma = 0,1\mu m$.

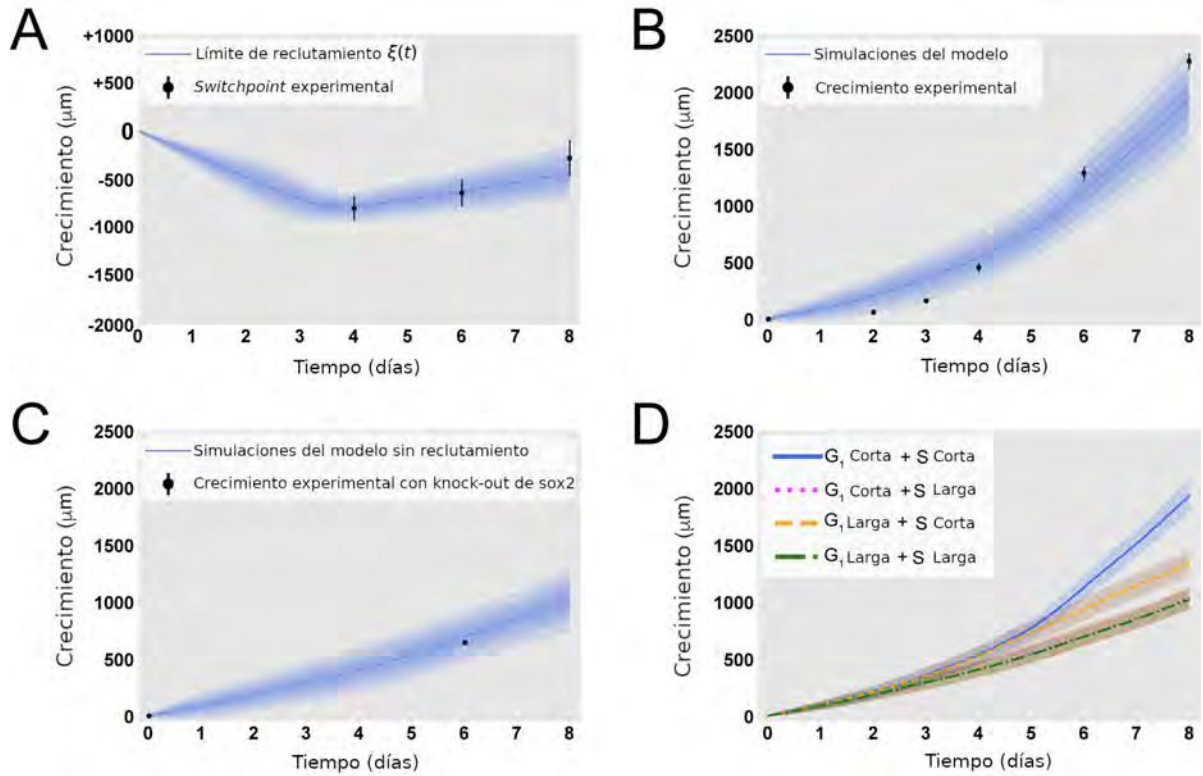


Figura 2.7: A) El límite de reclutamiento modelado se ajusta con éxito a la curva del *switchpoint* experimental. Simulaciones de mejor ajuste del modelo de límite de reclutamiento previsto $\xi(t)$ se superpone a la curva del *switchpoint* experimental. Los mejores parámetros de ajuste son $N_0 = 196 \pm 2$ células, $\lambda = 828 \pm 30 \mu\text{m}$ y $\tau = 85 \pm 12$ horas. B) El modelo coincide cuantitativamente con la cinética de crecimiento experimental de la médula espinal del axolotl (Rost *et al.*, 2016). C) El modelo reproduce la reducción del crecimiento experimental cuando se impide la aceleración de la proliferación celular. Predicción del modelo asumiendo que ni las longitudes de la fase S ni la de la fase G1 se acortaron, superpuestas con las cinéticas de crecimiento experimental en las que la aceleración del ciclo celular fue evitada al noquear a Sox2 (Fei *et al.*, 2014). Las líneas en A, B y C muestran las medias, mientras que las áreas sombreadas en azul corresponden a intervalos de confianza de 68, 95 y 99,7%, de más oscuro a más claro, calculados a partir del mejor ajuste de 1.000 simulaciones. D) El acortamiento de la fase S domina la aceleración del ciclo celular durante la regeneración de la médula espinal del axolotl. Cinética de crecimiento de la médula espinal predicha por el modelo suponiendo un acortamiento de las fases S y G1 (línea azul), solo acortamiento de la fase S (línea discontinua naranja), solo acortamiento de la fase G1 (línea discontinua magenta) y ni acortamiento de S ni Acortamiento G1 (línea verde). La línea magenta y la línea verde se superponen entre sí. Las medias son representado como líneas y cada área sombreada corresponde a un sigma de 1.000 simulaciones.

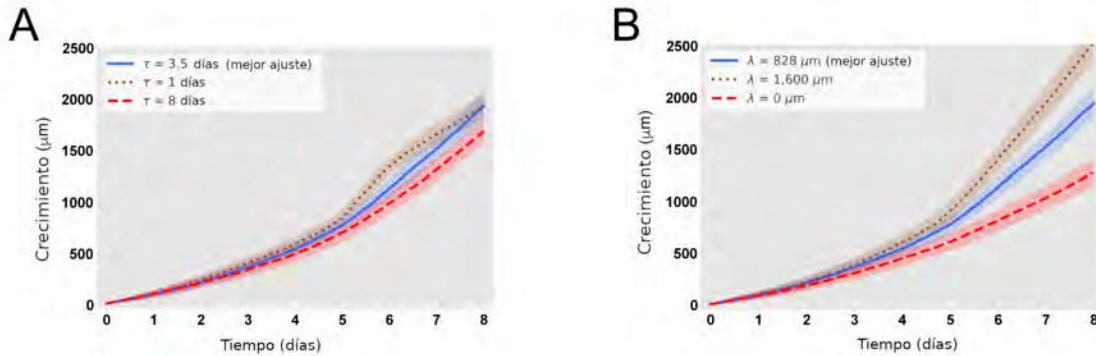


Figura 2.8: El tiempo máximo de reclutamiento (τ) y la longitud máxima de reclutamiento (λ) determinan el crecimiento de la médula espinal del axolotl. **A)** El aumento (disminución) de τ reduce (aumenta) el crecimiento de la médula espinal predicho por el modelo. Evolución temporal del crecimiento de la médula espinal predicha por el modelo al variar τ de 1 día (marrón) a 8 días (rojo). En azul, la predicción del modelo asumiendo $\tau = 85 \pm 12$ horas. $N_0 = 196 \pm 2$ células y $\lambda = 830 \pm 30 \mu m$. **B)** El aumento (disminución) de λ aumenta (reduce) el crecimiento de la médula espinal predicho por el modelo. Evolución temporal del crecimiento de la médula espinal predicho por el modelo al variar λ de cero (rojo) a $1.600 \mu m$ (marrón). En azul, la predicción del modelo asumiendo $\lambda = 828 \pm 30 \mu m$. $N_0 = 196 \pm 2$ células y $\tau = 85 \pm 12$ horas. Las simulaciones representadas en las curvas azules en A y B son las mismas simulaciones que se muestran en la Fig. 2.7B. Las medias son representados como líneas, mientras que las áreas sombreadas corresponden a intervalos de confianza del 68%, respectivamente, calculados a partir de 1.000 simulaciones.

2.6 El acortamiento de la fase S es suficiente para explicar el crecimiento inicial de la médula espinal en regeneración

Las contribuciones relativas del acortamiento de la fase G1 vs S al crecimiento de la médula espinal son un factor importante desconocido que es técnicamente difícil de interrogar *in vivo*. Hice uso de mi modelo para responder a esta pregunta *in silico*. Para ello, mantuve la misma parametrización obtenida con la que se recapituló el crecimiento de la médula espinal (Figura 2.7 A y B) pero modifiqué el modelo de tal manera que las células reclutadas acortan la fase S pero no la fase G1 (es decir, dejando la fase

G1 inalterada) o viceversa. Curiosamente, mis resultados indican que el acortamiento únicamente de la fase S, puede explicar el crecimiento explosivo de la médula espinal observado *in vivo*, independientemente del acortamiento de G1, hasta el día 4 (Figura 2.7 D, línea azul y línea naranja). En contraposición, el acortamiento de la fase G1 solamente, tiene un impacto leve en el crecimiento inicial, ya que da como resultado crecimiento casi idéntico al caso en el que no se redujeron ni la fase G1 ni la S (Figura 2.7 D, línea magenta versus línea verde). Sin embargo, desde el día 4, el acortamiento de solo la fase S no es capaz de recapitular el crecimiento (Figura 2.7 D, línea azul y línea naranja) y, de hecho, es el acortamiento de las fases S y G1 que devuelve el mismo crecimiento que el observado *in vivo*. Estas predicciones de modelado son una consecuencia de i) la proximidad de la fase S a la siguiente división celular en comparación con la fase G1 y ii) el hecho de que la fase S representa 7,5 días del total 14 días del ciclo celular largo, que se reduce a 3,7 días en el ciclo celular corto de 5 días y iii) la ventana de tiempo del crecimiento investigado fue de 8 días. En conclusión, estos resultados indican que, hasta el día 4, el acortamiento de la fase S puede explicar la el crecimiento regenerativo de la médula espinal en el axolotl, mientras que el efecto del acortamiento de G1 se manifiesta a partir del día 5.

Párametro del modelo	Valor/explicación	Fijo/libre
Fase G1 no regenerativa (promedio)	152 horas	Parámetros fijos, extraídos de Rodrigo Albors <i>et al.</i> , 2015.
Fase G1 no regenerativa (sigma)	54 horas	
Fase S no regenerativa (promedio)	179 horas	
Fase S no regenerativa (sigma)	21 horas	
Fases G2 + M no regenerativa (promedio)	9 horas	
Fases G2 + M no regenerativa (sigma)	6 horas	
Fase G1 regenerativa (promedio)	22 horas	
Fase G1 regenerativa (sigma)	19 horas	
$G1_r$	130 horas	
Fase S regenerativa (promedio)	88 horas	
Fase S regenerativa (sigma)	9 horas	
Fases G2 + M regenerativa (promedio)	9 horas	
Fases G2 + M regenerativa (sigma)	2 horas	
Longitud de la célula a lo largo del eje AP	$13,2\mu m$	
N_0	Número inicial de células a lo largo del eje AP, anterior al the plano de amputación.	Parámetros libres (determinados en esta tesis).
λ	Región desde el plano de amputación reclutada por la señal (μm).	
τ	Retraso entre el momento de la amputación y el reclutamiento (días u horas).	

Cuadro 2.1: Parámetros del modelo.

CAPÍTULO 3

Determinación del tránsito a través del ciclo celular mediante FUCCI en axolotl *in vivo*

Tiempo es la medida del movimiento entre dos instantes.

—Aristóteles

3.1 Visualización de la progresión del ciclo celular en axolotl *in vivo* usando FUCCI

Mi modelo hace suposiciones bien definidas sobre cómo las fases del ciclo celular se acortan para dar como resultado una aceleración del ciclo celular durante 85 horas dentro de los 828 μm del lugar de la lesión. Buscamos validar el modelo determinando la cinética de esta respuesta rigurosamente, utilizando una herramienta que distingue las fases del ciclo celular *in vivo* preservando el contexto espacio-temporal. Para ello, adaptamos la tecnología basada en el indicador de ciclo celular basado en la ubiquitinización fluorescente (FUCCI) para axolotl (Figura 3.1 A) (Zielke & Edgar, 2015). FUCCI es un reportero codificado genéticamente que distingue las fases del ciclo celular capitalizando la actividad oscilatoria exclusiva de dos ubiquitin ligasas (Sakaue-Sawano *et al.*, 2008). SCFSkp2 está activo en S y G2 del ciclo celular, cuando se dirige al factor de licencia de ADN Cdt1 para la degradación proteolítica. Por el contrario, APC/CCdh1 está activo desde mediados de M hasta G1; durante estas fases, apunta a Geminin (Gmnn; a Cdt1 inhibidor) para la degradación. Fusionando los motivos que apuntan a la degradación (degrons) en Cdt1 y Gmnn proteínas a dos fluoróforos distintos pone la abundancia de fluoróforos bajo el control de SCFSkp2 y actividad de APC / CCdh1 y permite utilizar la fluorescencia como lectura para la fase del ciclo celular. Es importante destacar que el

análisis de FUCCI no requiere disociación celular (preservando así el contexto espacial dentro del tejido), inmunotinción o medición del contenido de ADN.

FUCCI se ha adaptado con éxito a varios organismos modelo, incluidos ratones, peces cebra, *Drosophila* y células humanas (revisado por Zielke y Edgar, 2015). Diseñamos axolotl FUCCI *de novo* al extraer las secuencias que albergan degren de las proteínas axolotl Cdt1 y axolotl Gmn. Esto era importante ya que encontramos que el N-terminal de la proteína Cdt1, que alberga el degren PIP, es divergente a través de modelos animales (Figura B.1 A). Definimos los fragmentos relevantes de la proteína Cdt1 de axolotl (que alberga el motivo PIP degren y Cy) y la proteína Gmn de axolotl (que alberga la caja D degren) usando alineamiento por homología y comparación con el pez cebra FUCCI (suplementos de figura 3.1 1A, B, C, D) (Sugiyama *et al.*, 2009, Bouldin *et al.*, 2014). El fragmento de axolotl Cdt1 [aa1-128] se fusionó a la proteína fluorescente mVenus y el fragmento de axolotl Gmn [aa1-93] a la proteína fluorescente mCherry. Se utilizó el promotor ubicuo de CAGG y la secuencia viral T2A para coexpresar el Cdt1 [aa1-128]-mVenus y fusiones Gmn [aa1-93]-mCherry en una transcripción. El reportero del ciclo celular específico de axolotl resultante se refiere como AxFUCCI (Figura 3.1A).

3.2 AxFUCCI discrimina fielmente las fases del ciclo celular *in vivo*

Realizamos imágenes de células vivas, análisis de contenido de ADN y caracterizaciones basadas en inmunofluorescencia para validar la capacidad de AxFUCCI para informar las fases del ciclo celular. Primero, electroporamos una línea de cultivo de células de axolotl immortalizada (células AL1) con plásmido AxFUCCI y se realizaron imágenes *in vivo*. Como era esperado, la fluorescencia AxFUCCI osciló durante el ciclo celular en el orden mVenus, Doble positivo, mCherry, Doble positivo, mVenus (Figura 3.1 B, Figura B.2). Confirmamos que las células positivas para mVenus eran células G0/G1 y que las células positivas para mCherry eran S/G2 usando un citómetro de flujo para analizar el contenido de ADN (Figura B.3 A). Las células son transitoriamente AxFUCCI doble positivo entre G0/G1 y S/G2 (Figura 3.1 B y Figura B.2 A, B); inferimos que estas células están en el límite G1/S, como se observa en FUCCI de ratón (Abe *et al.* 2013). Curiosamente, y en contraste con FUCCI en otros organismos modelo, observamos una segunda ventana doble positiva AxFUCCI entre S/G2 y G0/G1, correspondiente al límite G2/M y células M (Figura 3.1 B y Figura B.2A, B). Por lo tanto, AxFUCCI discrimina las siguientes fases del ciclo celular en axolotl: G0/G1 (solo mVenus); S/G2 (solo mCherry); Transición G1/S y transición G2/M (doble positivo) (Figura 3.1 B). Es importante destacar que la capacidad de AxFUCCI para etiquetar puntos de referencia definidos en el ciclo celular (es decir, la transición G1/S y G2/M) nos permitió probar más tarde la sincronización del ciclo celular, una característica de nuestro modelo.

Generamos axolotls AxFUCCI transgénicos estables usando transgénesis mediada por I-SceI, los criamos a la madurez sexual y utilizó la progenie F1 (transmitida por la

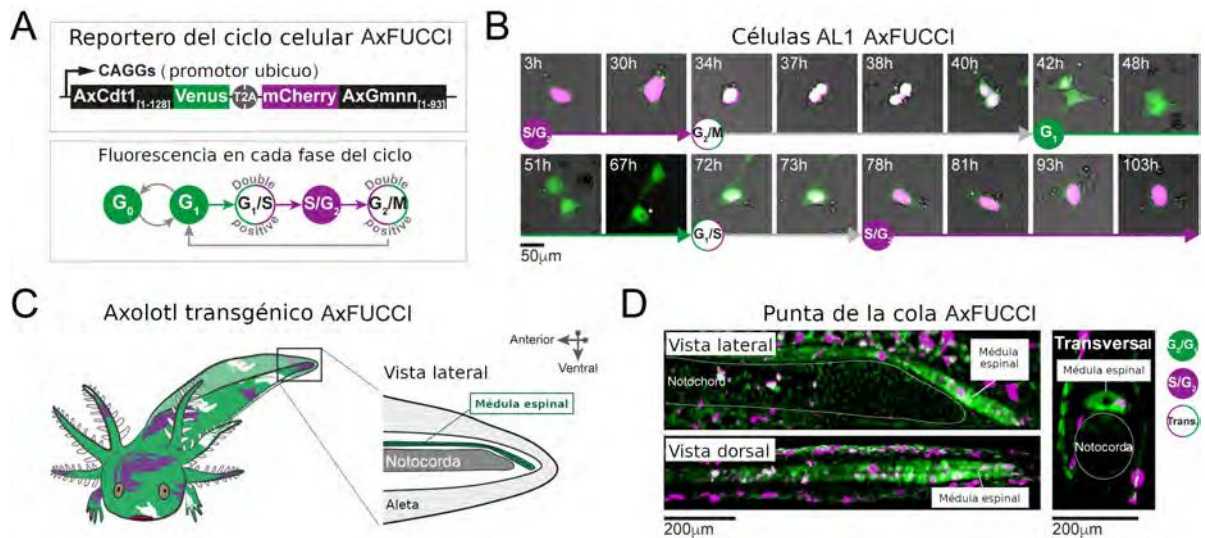


Figura 3.1: A) **Panel superior: diseño AxFUCCI.** Un promotor ubicuo (CAGG) impulsa la expresión de dos sondas AxFUCCI (AxCdt1 [1-128] -Venus y mCherry-AxGmnn [1-93]) en una transcripción. Las dos sondas AxFUCCI se separan cotraduccionalmente en virtud de la secuencia de péptidos T2A viral “autoescindible”. Panel inferior: combinaciones de fluorescencia de AxFUCCI en cada fase del ciclo celular. B) **Obtención de imágenes de una sola célula axolotl electroporada con AxFUCCI *in vitro*.** Cada panel es un solo marco adquirido a la hora indicada (h) después del inicio de la sesión de imágenes. Se representa un ciclo celular (desde S/G₂ hasta el siguiente S/G₂). A las 67 h, las dos células hijas mitóticas se separaron; la celda hija con un asterisco se representa los paneles restantes. C) **Establecimiento de axolotls transgénicos AxFUCCI.** La ubicación de la médula espinal se indica en el contexto de la cola. D) **Una cola AxFUCCI fija, aclarada y fotografiada en su totalidad utilizando microscopía de láminas de luz.** Los datos 3D permiten el corte digital *post-hoc* de la misma médula espinal en vistas lateral, dorsal o transversal. Las imágenes muestran proyecciones de máxima intensidad a través de 50μm (vistas lateral y transversal) o 150μm (dorsal vista) de tejido. Trans.: Transición-AxFUCCI (transición G₁/S o G₂/M).

línea germinal) para validaciones posteriores (Figura 3.1 C). Los animales AxFucci se desarrollaron a un ritmo similar al de sus hermanos no transgénicos, no difirieron en su proliferación celular (Figura B.4 A) y tejido de la cola amputado regeneró con similar cinética a los animales d/d utilizados en nuestro estudio anterior (Figura B.4 B). Células disociadas de las colas de axolotl AxFucci y analizadas usando un citómetro de flujo exhibieron relaciones de fluorescencia/contenido de ADN esperadas (Figura B.4 C). Como segundo ensayo, se prepararon secciones de tejido de la médula espinal de axolotls AxFucci y se comparó el contenido de ADN (según lo evaluado por Fluorescencia DAPI) en mVenus frente a células positivas para mCherry. Como era de esperar, las células positivas para mCherry albergaron significativamente más ADN que las células positivas para mVenus (Figura B.4 D, E). En tercer lugar, los axolotl AxFucci fueron inyectados intraperitonealmente con EdU, un análogo de timidina que se incorpora en ADN durante la fase S. Después de un pulso de EdU de 8 horas, las células positivas para mCherry, pero no positivas para mVenus, deberían ser positivas para EdU y este fue realmente el caso (Figura B.4 F). Finalmente, co-secciones de tejido de médula espinal fueron teñidas con AxFucci con marcadores específicos del tipo celular. Neuronas que expresan NeuN, localizadas en la periferia de la médula espinal, son células diferenciadas posmitóticas (G0) y deben ser mVenus-positivo (y nunca mCherry-positivo). De hecho, encontramos que el 100% de las neuronas que expresan AxFucci fueron mVenus-positivo (Figura B.4 G). Por el contrario, Sox2-expresando en las células endodermales, que son células proliferativas, expresaron mVenus, mCherry o ambos fluoróforos (Figura B.4 H).

Basándonos en estas validaciones, y para simplificar, nos referimos a la fluorescencia de mVenus como “G0/ G1-AxFucci” (verde en las Figuras 3.1 - 3.6), fluorescencia de mCherry como “S/G2-AxFucci” (magenta en las Figuras 3.1 - 3.6) y doble positividad como “Transition-AxFucci” (blanco en las Figuras 3.1 - 3.6).

3.3 Medición de la zona de reclutamiento *in vivo* usando AxFucci

Usamos animales AxFucci para medir el tamaño de la zona de reclutamiento de células endodermales *in vivo*. Colas de AxFucci fueron amputadas a 5mm de las puntas de las colas y colas regeneradas fueron recolectadas diariamente hasta 5 días después de la amputación. Implementamos un conducto para limpiar ópticamente y fijar la imagen del tejido de la cola AxFucci en montura completa, lo que permitió volver a seccionar digitalmente a *posteriori* los datos de imágenes en cualquier orientación para una medición precisa (Figura 3.1 D) (Pende & Vadiwala *et al.*, 2020). Es importante destacar que este conducto no alteró la longitud del tejido de la cola (Figura B.5). Se encontró que, en la amputación, la mayoría de las células endodermales expresaban G0/G1-AxFucci y solo una minoría expresó S/G2-AxFucci (Figura 3.2 A). En los cinco días siguientes a la amputación de la punta de la cola, la proporción de células que expresan S/G2-AxFucci aumentaron localmente y de forma pronunciada en el sitio de la amputación,

luego se propagaron anteriormente a lo largo de la médula espinal, consistente con la aparición de una zona de reclutamiento (Figura 3.2 A y Figura 2.1B).

Como primer paso, cuantificamos el porcentaje de células endodurales que expresan *G0/G1-AxFUCCI* solamente o *S/G2-AxFUCCI* únicamente dentro de los 1000 – 1600 μm de médula espinal anterior al sitio de la amputación a cada día después de la amputación. Una longitud de 1600 μm debe abarcar no solo la zona de reclutamiento sino también células endodurales ubicadas más anteriormente que no son reclutadas por la señal de lesión y que continúan en el ciclo lento (Figura 2.1 B). Realizamos nuestras cuantificaciones en bins adyacentes de 100 μm para preservar la información espacial, utilizando la punta de la notocorda cortada para denotar el plano de amputación. Probamos estadísticamente si las células que expresan *G0/G1-AxFUCCI* y *S/G2-AxFUCCI* se encuentran heterogéneamente distribuidos a lo largo del eje *AP* en la médula espinal en regeneración (es decir, si se puede detectar una zona de reclutamiento), seguimos un enfoque similar al que usamos anteriormente para determinar el *switchpoint* (Rost *et al.*, 2016). Ajusté los perfiles *AP* espaciales medidos del porcentaje de *G0/G1-AxFUCCI* y *S/G2-AxFUCCI* en cada animal con un modelo matemático asumiendo dos zonas espaciales homogéneas adyacentes, separadas por un borde antero-posterior (borde *AP*), que asumimos fue el mismo para los datos *G0/G1* y *S/G2-AxFUCCI*. Para cada animal y a cada tiempo, probé si el porcentaje medio de células que expresan *G0/G1-AxFUCCI* o *S/G2-AxFUCCI* en la región anterior frente a la zona posterior fueron significativamente diferentes al ejecutar una prueba de Kolmogorov-Smirnov. Hasta 3 días después de la amputación, no se detectó significación estadística entre anterior y posterior en el *G0/G1-AxFUCCI* y *S/G2-AxFUCCI* (Figura 3.2 B, Figura 4 - Figura 3.3 A-D y Figura 3.4 A-D). Por el contrario, a los 4 y 5 días después de la amputación, los datos de *G0/G1-AxFUCCI* y *S/G2-AxFUCCI* revelaron una diferencia significativa entre las zonas anterior y posterior, consistente con la aparición de una zona de reclutamiento (Figura 3.2 B, Figura 3.3 E, F y Figura 3.4 E, F).

Crucialmente, medí el borde *AP* de los datos *AxFUCCI* obteniendo $-717 \pm 272\mu\text{m}$ en relación con el plano de amputación a día 4 y $-446 \pm 112\mu\text{m}$ a día 5, superponiéndose dentro de dos sigma con $-782 \pm 50\mu\text{m}$ y $-710 \pm 62\mu\text{m}$, que fueron los límites de reclutamiento predichos por nuestro modelo (Figura 3.2 B). Además, la aparición de la zona de reclutamiento entre los días 3 y 4 después de la amputación se condice con el reclutamiento de 85 horas en nuestro modelo. Por tanto, los animales *AxFUCCI* confirmaron el tiempo de aparición y el tamaño de la zona de reclutamiento previsto.

3.4 Las células en regeneración tienen una alta sincronía del ciclo celular in vivo

Mi modelo de acortamiento *G1* (Figura 2.1 C y Figura 2.3 2) predice que las células endodurales en la zona de reclutamiento deben exhibir una alta sincronía entre sí en el ciclo celular durante la regeneración temprana, una propiedad que no ha sido investigada.

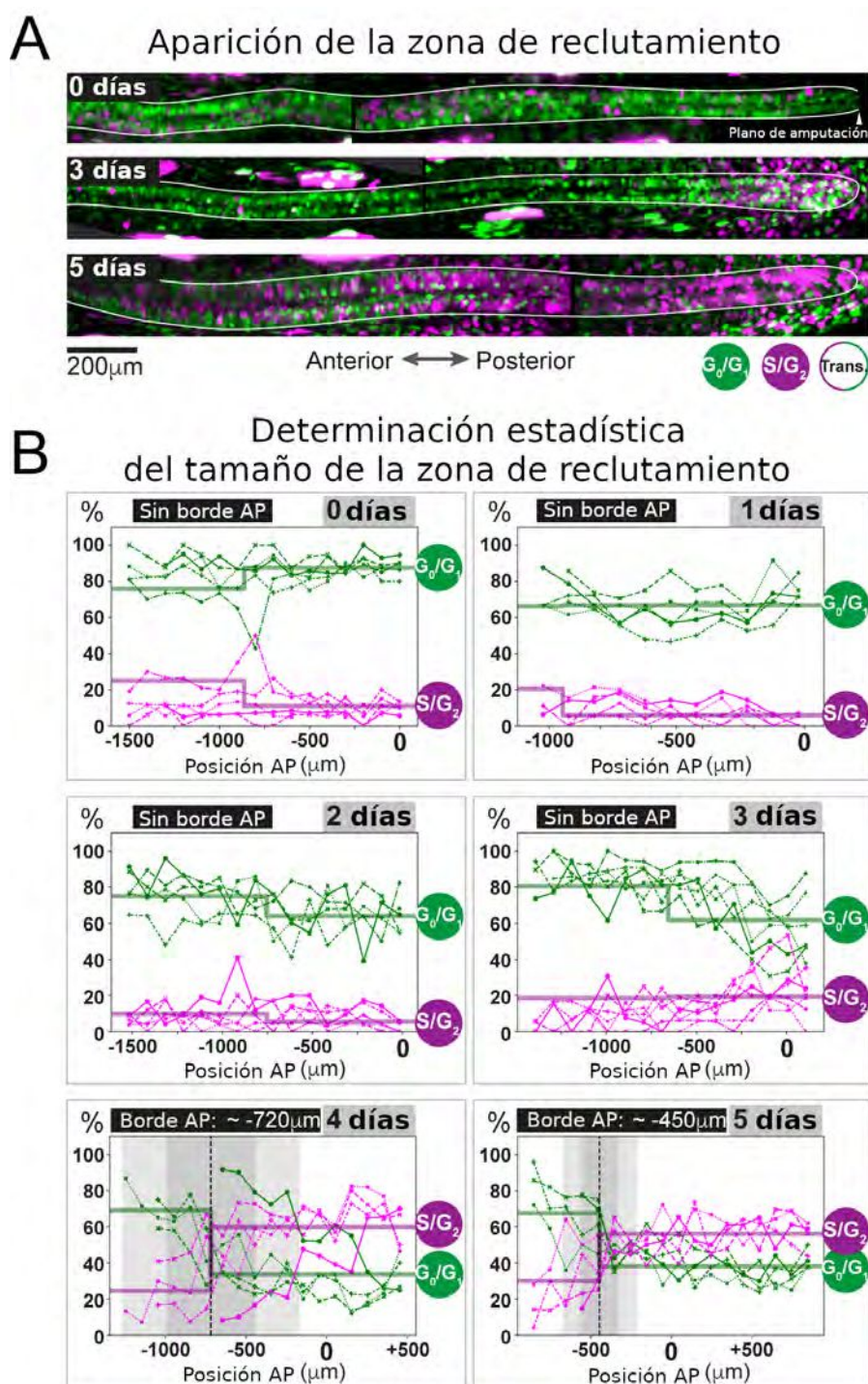


Figura 3.2: El tamaño de la zona de reclutamiento previsto se observa en las colas AxFucci después de la amputación.

Figura 3.2: A) Visualización de la zona de reclutamiento en colas AxFUCCI. En la amputación, la mayoría de las células ependimales son positivas para G0/G1-AxFUCCI (verde). Después de la amputación, se observa una disminución en G0/G1-AxFUCCI y un aumento en la expresión de S/G2-AxFUCCI (magenta) anterior al plano de amputación y esta zona aumenta de tamaño anteriormente hasta el día 5 después de la amputación. La médula espinal está delineada. Las imágenes representan proyecciones de máxima intensidad a través de $30\mu m$ de tejido y son compuestas de dos campos de visión adyacentes. **B) Cuantificación del borde AP que delimita la zona de reclutamiento en los datos de G0/G1 y S/G2-AxFUCCI.** Porcentaje de células que expresan G0/G1 (en verde) y S/G2 (en magenta) -AxFUCCI cuantificado en contenedores de $100\mu m$ a lo largo del eje anteroposterior de la médula espinal. Un modelo matemático que asume dos zonas adyacentes espacialmente homogéneas separadas por un borde anteroposterior (borde AP) se ajustó a los datos de G0/G1 y S/G2-AxFUCCI para cada animal. Se observaron diferencias significativas entre las zonas anterior y posterior detectadas solo en el día 4 y 5 (prueba de Kolmogorov-Smirnov $p = 0,0286$). La media del borde AP y dos sigmas son representado como una línea discontinua negra y áreas sombreadas en gris, respectivamente. La posición AP se define con respecto al plano de amputación ($0\mu m$). $n = 4-6$ colas por punto de tiempo, aproximadamente 300 células cada una. Los diferentes símbolos representan diferentes animales; cada línea representa un animal. Los valores de mejor ajuste del modelo con respecto al porcentaje anterior y posterior de los datos de AxFUCCI se encuentran en la Figura 3.3. Los ajustes individuales se encuentran en la Figura 3.4. Para más detalles, consulte la sección de Materiales y métodos.

Los axolotls AxFUCCI nos permitieron evaluar la posible sincronía del ciclo celular ependimal en la regeneración de la médula espinal.

Realicé una cuantificación más rigurosa de la distribución del ciclo celular de nuestra cantidad total datos en los que nos centramos en los $600\mu m$ de médula espinal inmediatamente anterior al plano de amputación, dentro de la zona de reclutamiento, y incluyendo también Transiciones-AxFUCCI (células de transición G1/S y G2/M) y células de fase M. En el momento de la amputación, $85 \pm 5\%$ de las células ependimales expresaron G0/G1-AxFUCCI y $11 \pm 6\%$ expresó S/G2-AxFUCCI (Figura 3.5 A). Observamos que esta línea de base difiere de la que informamos anteriormente en animales de control d/d más pequeños, pero encontramos esta tasa de proliferación basal más baja para ser consistente entre los animales en este estudio independientemente del genotipo (Figura B.4 A) (Rodrigo Albors *et al.*, 2015). Restricciones en la alimentación y/o cambios en el manejo de animales durante la pandemia de COVID-19 podría explicar esta diferencia. Como esperábamos, nuestro modelo podría ser robusto al perfil del ciclo celular de línea de base, recopilamos mediciones en estos animales para más análisis posteriores a este trabajo.

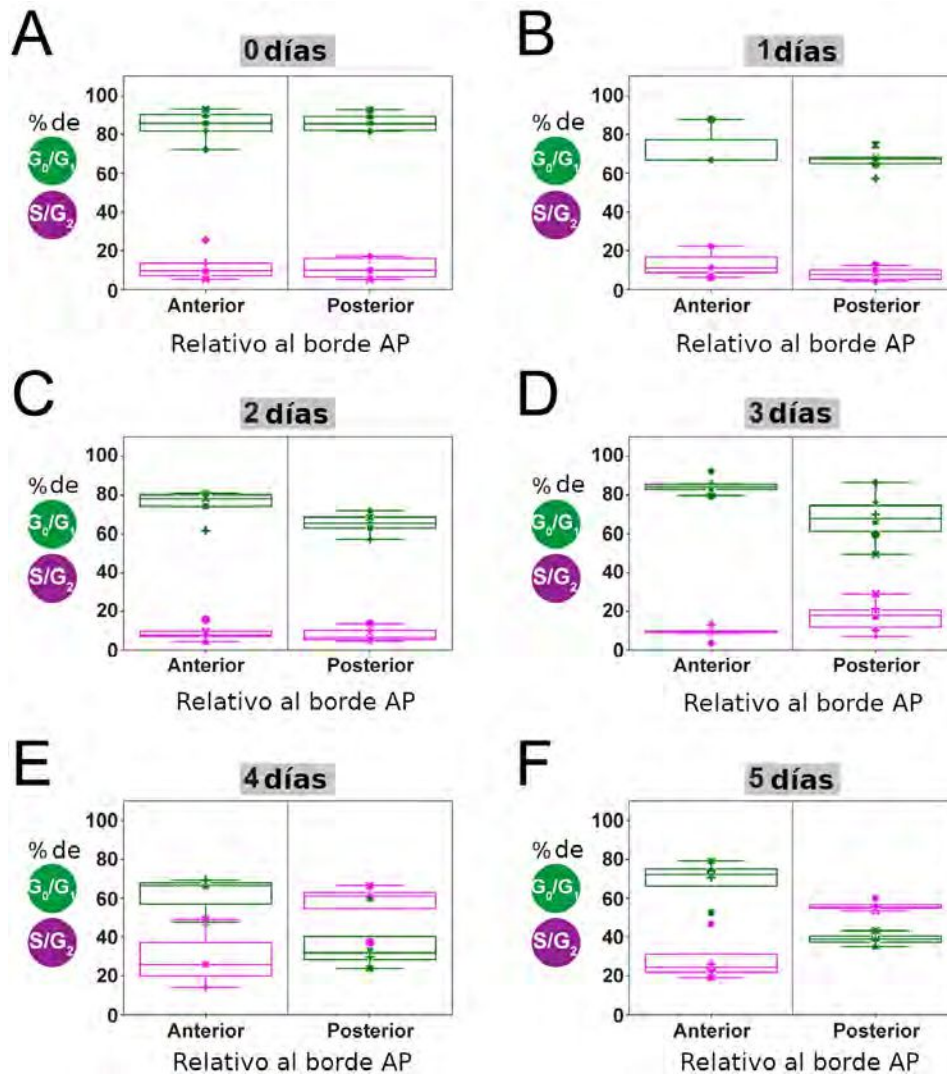


Figura 3.3: Estimación de los porcentajes anterior y posterior de células que expresan G0/G1-AxFucci y S/G2-AxFucci ajustado un modelo de dos zonas. Los puntos corresponden al valor de mejor ajuste de los parámetros $g0g1_a$, $g0g1_p$, $sg2_a$ y $sg2_p$ a los perfiles AP espaciales experimentales de los porcentajes G0/G1-AxFucci (en verde) y S/G2-AxFucci (en magenta) que expresan las células para cada animal, en cada punto temporal (A-E). Cada símbolo identifica a un animal diferente (mismo criterio que en la Figura 3.2). Los diagramas de caja muestran que en el día 0 a 3 días después de la amputación, los valores anterior y posterior no son significativamente diferentes entre sí (A - D). Por el contrario, de día 4 y 5, los valores anterior y posterior son significativamente diferentes (E, F) (prueba de Kolmogorov-Smirnov $p = 0,0286$) (Materiales y método para más detalles).

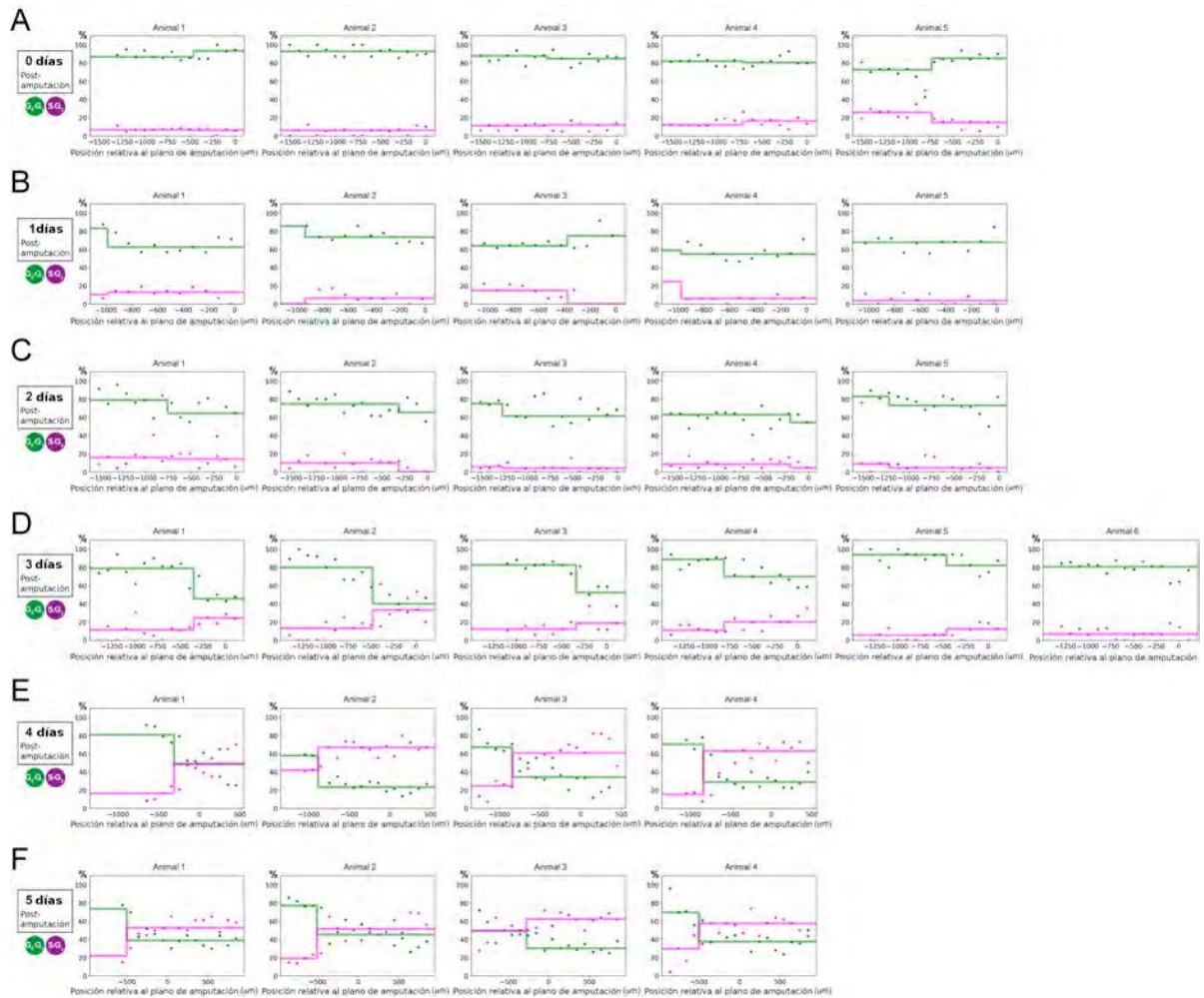


Figura 3.4: Ajuste individual del modelo de dos zonas a los perfiles *AP* experimentales de los porcentajes de G0/G1-AxFUCCI y S/G2-AxFUCCI que expresan las células. Porcentaje de G0/G1 (en verde) y S/G2 (en magenta) de las células expresando AxFUCCI cuantificadas en contenedores de $100\mu\text{m}$ a lo largo del eje anteroposterior de la médula espinal. Modelo matemático asumiendo dos zonas adyacentes espacialmente homogéneas separadas por un borde antero-posterior (borde *AP*) que se ajustó a los datos de G0/G1 y S/G2-AxFUCCI para cada animal a tiempo 0 (A), 1 (B), 2 (C), 3 (D), 4 (E) y 5 (F) días después de la amputación. La posición anteroposterior se define con respecto al plano de amputación ($0\mu\text{m}$). Los mejores valores de ajuste del modelo con respecto al porcentaje anterior y posterior de los datos AxFUCCI se encuentran en la Figura 3.3 (Para obtener más detalles, consulte la sección de Materiales y métodos).

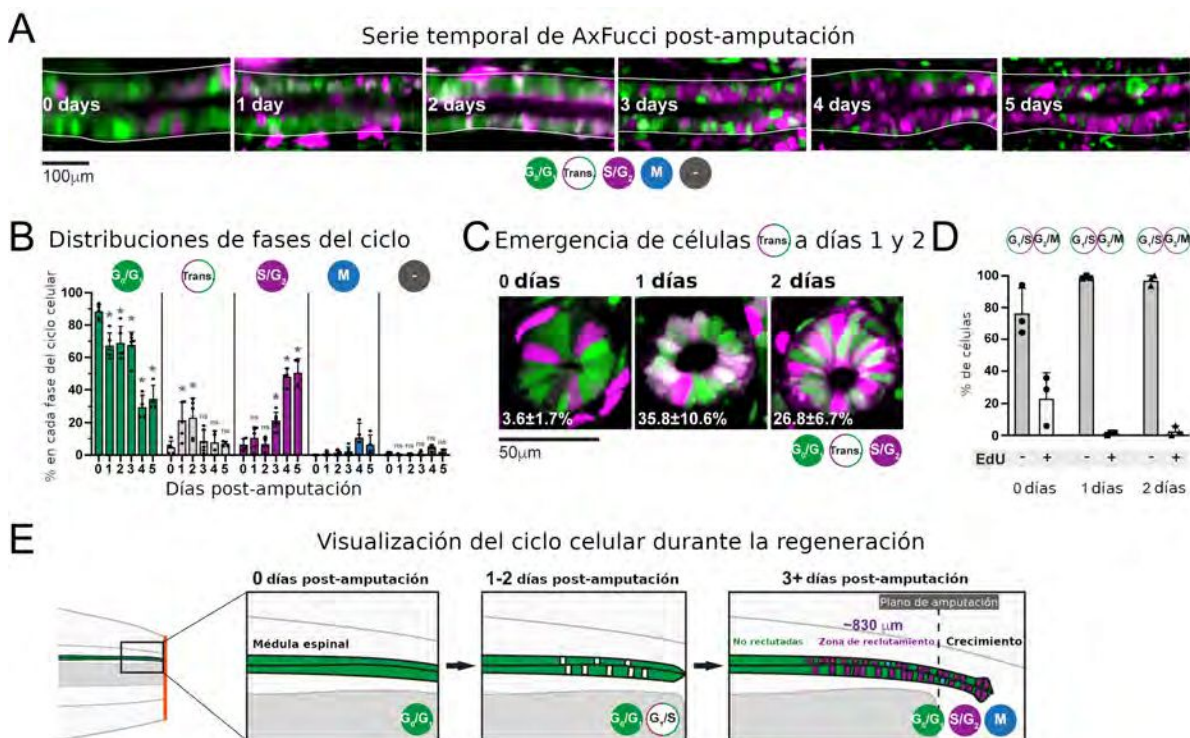


Figura 3.5: Las células endodiales exhiben sincronía del ciclo celular durante la regeneración de la médula espinal.

Después de la amputación, detectamos una caída significativa en las células endodiales que expresan *G0/G1-AxFucci* ya en el día 1. Recíprocamente, el número de células que expresan *S/G2-AxFucci* aumentó significativamente a partir de los 3 días y alcanzó el 50% en los días 4 y 5 después de la amputación, en comparación con una línea de base porcentaje por debajo del 10% (Figuras 3.5 A y B). Las células de la fase *M* comenzaron a aparecer notablemente desde el día 4, aunque no fue posible realizar análisis estadístico debido a la ausencia de células de fase *M* en muestras a tiempo 0 (Figura 3.5 B). Curiosamente, observamos una “ráfaga” breve y transitoria de células *AxFucci* de transición en los días 1 y 2 después de la amputación. El porcentaje de células *Transition-AxFucci* aumentó a $21 \pm 12\%$ y $23 \pm 12\%$ respectivamente durante esta ráfaga, antes de volver a disminuir al nivel de línea de base de $5 \pm 4\%$ a los 3 días (Figura 3.5 B). Confirmamos la precisión de estas cuantificaciones mediante la preparación y cuantificación de secciones de tejido de réplicas de médula espinal *AxFucci* (Figura 3.5 C y Figura B.6 A).

Transition-AxFucci podría corresponder a la transición *G1/S* o la transición *G2/M* (Figura 3.1 A). Se planteó la hipótesis de que las células en *Transition-AxFucci* a 1 y 2 días después de la amputación residían en la transición *G1/S*, ya que aparecieron entre la disminución de las células *G0/G1-AxFucci* en el día 1 y el aumento de células *S/G2-*

Figura 3.5: A) Distribuciones del ciclo celular de las células endodermales durante los primeros cinco días de regeneración en los 600 μ m mas posteriores. Proyecciones de máxima intensidad a través de 25 μ m de médula espinal orientada de anterior a izquierda y posterior a derecha. Médula espinal delineada. El lumen está en el centro. Las células en mitosis (M) se contaron independientemente de AxFUCCI y se identificaron en función de su condensación de cromatina revelada por tinción con DAPI (no mostrado). “-” indica células AxFUCCI-negativas, que fueron insignificantes en todos los momentos. $n = 4-6$ colas por punto de tiempo, aproximadamente 100 células cada una. Las barras de error indican desviaciones estándar.

B) Datos AxFUCCI separados por fase del ciclo celular. Los datos cuantitativos del panel A. Datos a día 0 (inmediatamente después de la amputación) se tomaron como referencia y se realizaron análisis estadísticos contra estas líneas de base. Las pruebas de Kruskal-Wallis seguidas de las pruebas de suma de rangos de Wilcoxon revelaron una disminución significativa en G0/G1-AxFUCCI desde 1 día después de la amputación ($p = 0,02$) y un aumento significativo en S/G2-AxFUCCI a partir de 3 días después de la amputación ($p = 0,02$). ANOVA unidireccional seguido de la prueba HSD de Tukey reveló un aumento significativo en las células Transición-AxFUCCI 1 y 2 días después de la amputación ($p = 0,04$ y $0,02$ respectivamente), pero no en los días posteriores a la amputación ($p > 0,99$). Células mitóticas ausentes en las muestras de 0 días. El porcentaje de células AxFUCCI-negativas fue insignificante y no cambió en ningún momento ($p = 0,40$, suma de rangos Wilcoxon). Cada punto representa datos de una cola. Las barras de error indican desviaciones estándar. ns: estadísticamente no significativo.

C) Se confirmó la aparición de células Transición-AxFUCCI 1 y 2 días después de la amputación en secciones de tejido. Imágenes confocales de un solo plano de secciones transversales de tejido de 10 μ m de espesor fijadas en el tiempo indicado después de la amputación. Los porcentajes indican el porcentaje de células Transición-AxFUCCI en cada punto de tiempo (media $\pm$ desviación estándar). $n = 3$ colas por punto de tiempo, aproximadamente 400 células contadas en cada una, lo que corresponde a aproximadamente 750 μ m de médula espinal.

D) Las células que expresan AxFUCCI de transición 1 y 2 días después de la amputación residen en la transición G1/S. Después de 8 horas (0 días) o pulso EdU de 24 horas (1 y 2 días), solo las células G2/M deben etiquetarse con EdU. $> 97\%$ de las células Transición-AxFUCCI fueron negativas para EdU en 1 y 2 días después de la amputación, lo que indica que residen en la transición G1/S. $n = 3$ colas por punto de tiempo. Un total de 36, se ensayaron 442 y 315 células Transición-AxFUCCI 0, 1 y 2 días después de la amputación, respectivamente.

E) Dinámica del ciclo celular durante la regeneración de la médula espinal axolotl.

AxFUCCI en el día 3. Para confirmar esto, sometimos a los animales AxFUCCI a un pulso de EdU durante 24 horas inmediatamente antes de la recolección de las cola a los 0, 1 o 2 días después de la amputación. En este ensayo, las células Transition-AxFUCCI que incorporan EdU deben estar en G2/M, mientras que las que no incorporan EdU deben estar en G1/S (Figura B.6 B). De acuerdo con nuestras expectativas, las células Transition-AxFUCCI a 1 y 2 días después de la amputación fueron casi en su totalidad negativos para EdU y, por lo tanto, residían en G1/S (Figura 3.5 D y B.6 C, D).

En resumen, AxFUCCI reveló la siguiente dinámica del ciclo celular durante la regeneración (resumida en Figura 3.5 E). La mayoría de las células endodermales de la médula espinal ilesa residen en la fase G0/G1 del ciclo celular. Después de la amputación de la cola, las células endodermales comienzan a salir de G0/G1 dentro del primer día de la amputación, transitan por G1/S en los días 1 y 2, ingresan en S/G2 desde el día 3 en adelante y sufren mitosis desde el día 4. El hecho de que estos comportamientos se observen fácilmente a nivel de población indican un alto nivel de sincronía entre células endodermales en la zona de reclutamiento. La transición G1/S actúa como un discreto hito en el ciclo celular en el que esta sincronía se puede inferir de forma fiable. Las células de Transition-AxFUCCI son muy raras (5%) en el momento de la amputación. Tomamos el aumento de 4,5 veces en las células que expresan Transition-AxFUCCI en días 1 y 2 después de la amputación como una fuerte indicación de sincronía del ciclo celular durante la regeneración de la médula espinal temprana, una predicción clave de nuestro modelo.

3.5 Una convergencia en la respuesta regenerativa de distintas líneas de base

Las cuantificaciones de AxFUCCI validaron predicciones clave de mi modelo en términos del tamaño de la zona de reclutamiento de células endodermales y la demostración de alta sincronía del ciclo celular durante la respuesta regenerativa. Nos intrigó que estos acuerdos ocurrieran a pesar de una diferencia significativa en las condiciones iniciales del ciclo celular (día 0) entre los animales AxFUCCI de nuestros experimentos y mi modelo, que se basó en datos descrito en (Rodrigo Albors *et al.* 2015). Los animales AxFUCCI tenían un tamaño de 5,5cm desde el hocico hasta la cola y, el día de la amputación, $11 \pm 6\%$ de las células endodermales expresaron S/G2-AxFUCCI. En contraste, mi modelo fue parametrizado utilizando medidas adquiridas de 3cm, axolotl no transgénico, en el que el porcentaje inicial de células en fase S fue cuatro veces mayor, como se infiere del marcador acumulado BrdU (Rodrigo-Albors *et al.*, 2015). A pesar de estas diferencias, encontramos que la velocidad de regeneración del tejido de la médula espinal es similar en los dos conjuntos de datos (Figura B.4 B). Esto podría reflejar una convergencia en la respuesta regenerativa a nivel del ciclo celular.

Ahondé aún mas la investigación sobre la cinética de esta convergencia utilizando nuestro modelo para generar simulaciones diarias de la distribución espacial de las células

en la fase *G1* o *S* y comparar estas simulaciones con las cuantificaciones de la fase del ciclo celular realizadas a partir de los animales AxFUCCI (Figura 3.6 A-D). Validé las simulaciones probando dónde y cuándo las células ependimales dejarían la fase *G1* para ingresar a fase *S* y se descubrió que, como era esperado, las transiciones simuladas de *G1* a *S* eran más frecuentes después del límite de reclutamiento derivado experimentalmente (Figura 3.6 E), que en el día 4 y 5 se superponen con los límites *AP* determinados. En consecuencia, las divisiones celulares simuladas (fase *M*) exhiben un patrón similar (Figura 3.7).

Comparé las simulaciones del modelo con los datos de AxFUCCI. Como se señaló, las condiciones del ciclo celular en día 0 difieren entre los experimentos AxFUCCI y mi modelo, pero coinciden cuantitativamente en el día 4 después de la amputación (Figura 3.6 A-D). Descubrimos que la apareición de la zona de reclutamiento ocurre más tarde en los animales AxFUCCI (días 4-5) que en las simulaciones (día 2). Sin embargo, una vez que la zona de reclutamiento evidente en los datos de AxFUCCI, su tamaño en el día 4 y 5 es comparable al predicho por las simulaciones. Por lo tanto, la zona de reclutamiento en la médula espinal AxFUCCI se manifiesta más sincrónica y rápida que en las simulaciones, en las que aumenta gradualmente de tamaño en una dirección anterior-a-posterior de los días 2 a 4. Esto es probablemente es una consecuencia de la mayor cantidad de células *G0/G1* en los animales AxFUCCI en los días 0-2 en comparación con el modelo, que incurre en un colectivo (pero relativamente sincrónico) retraso en la entrada de la fase *S* (manifestación de la zona de reclutamiento). Nuestros datos revelan dos trayectorias contrastando hacia el logro de una salida regenerativa común.

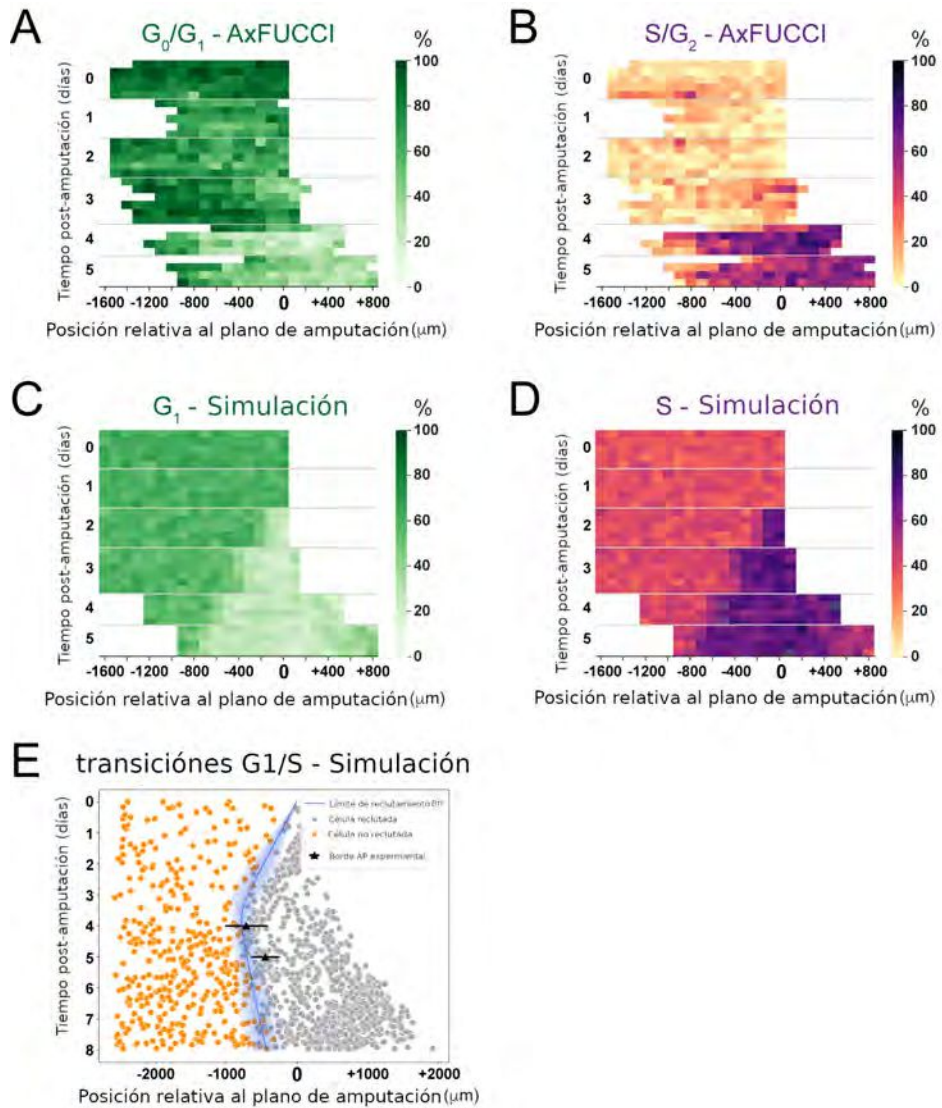


Figura 3.6: Mapas de calor que representa la distribución espacio-temporal de las células G_0/G_1 -AxFUCCI (A), células S/G_2 -AxFUCCI (B), células predichas por el modelo en la fase G_1 (C) y fase S (D). Los ejes x representan la posición AP con respecto al plano de amputación (posición $AP = 0\mu\text{m}$). Los ejes y representan el tiempo (días) luego de la amputación y las réplicas tanto experimentales como simuladas dentro de cada punto de tiempo. Los códigos de color corresponden al porcentaje de células en la fase correspondiente. Los datos de las células Transición-AxFUCCI están en Figura 3.8. (E) La ocurrencia de transiciones G_1 - S predicha por el modelo es más frecuente dentro de las células reclutadas (círculos grises) en comparación con las células no reclutadas (cruces naranjas). En los días 4 y 5, el límite de reclutamiento se superponen con los bordes AP . Se muestran simulaciones de 10 semillas independientes y el modelo está parametrizado como en la Figura 2.7.

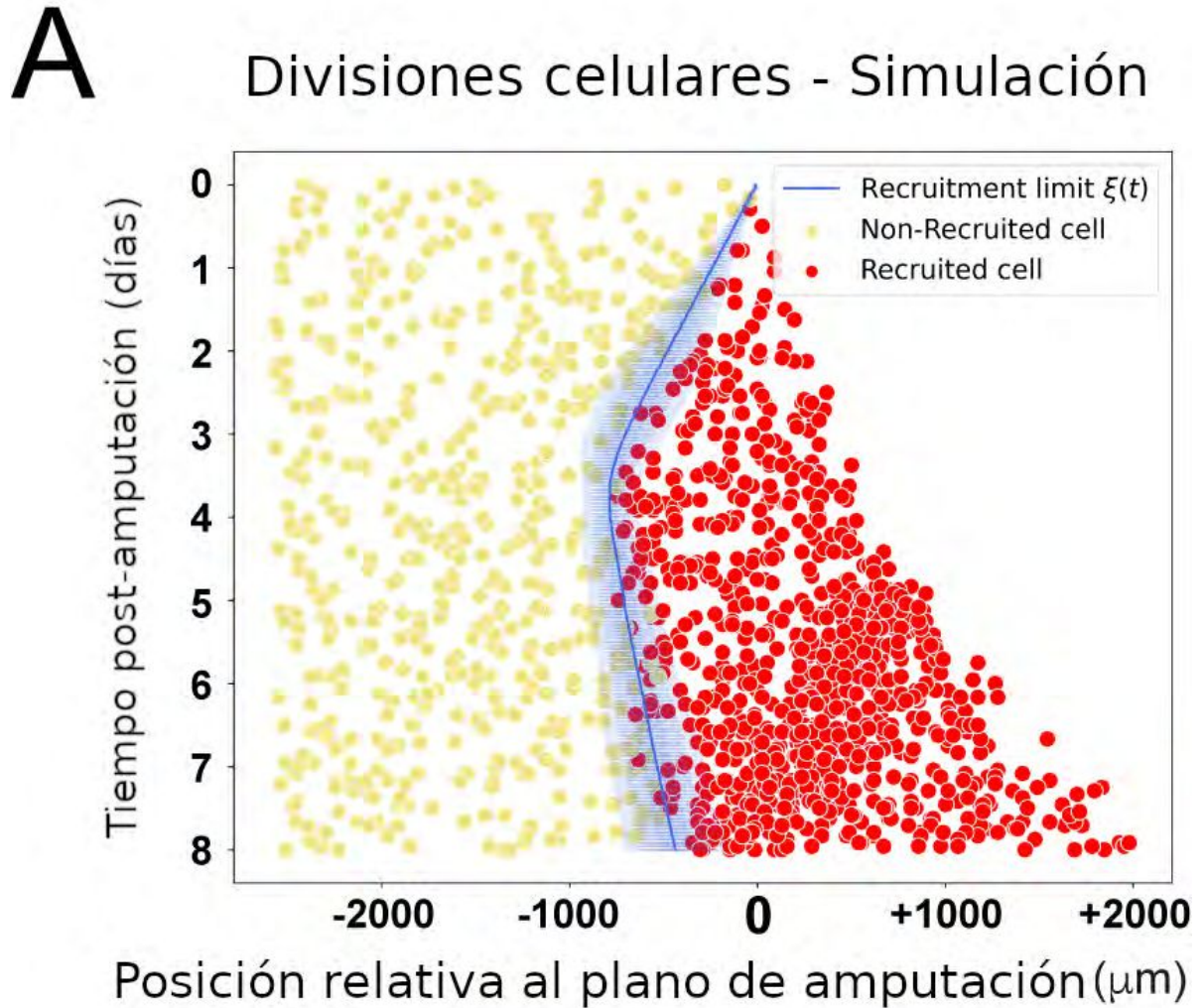


Figura 3.7: Modelado de distribuciones espaciotemporales de las divisiones de células endodermales durante la regeneración de la médula espinal del axolotl. La ocurrencia de divisiones celulares predicha por el modelo es más a menudo dentro de los reclutados (rojo cruces) en comparación con las células no reclutadas (círculos amarillos), donde las células reclutadas y no reclutadas son separadas por el límite de reclutamiento $\xi(t)$. El modelo está parametrizado como en la Figura 2.7 y la Figura 3.6C-E.

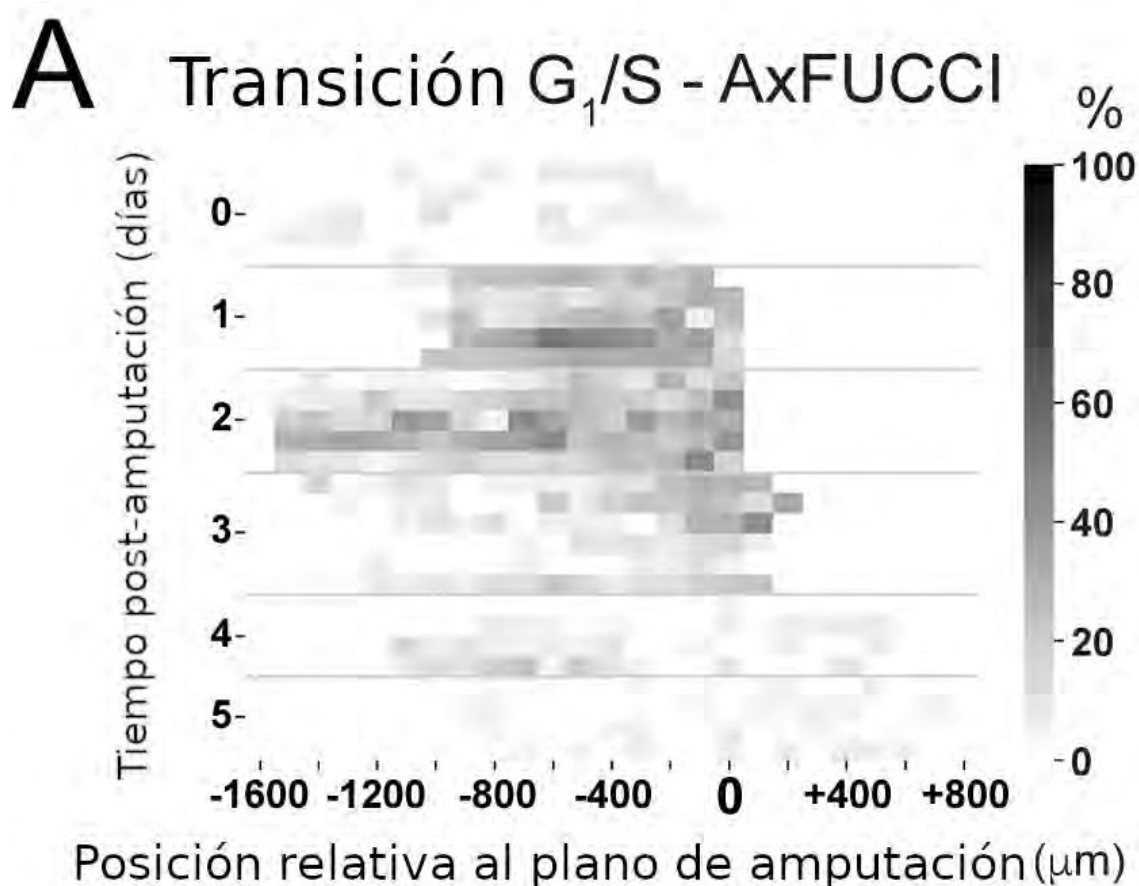


Figura 3.8: Distribuciones espacio-temporales de las células AxFUCCI de transición durante la regeneración de la médula espinal del axolotl. Mapas de calor que representan la distribución espacio-temporal experimental de las células Transición AxFUCCI. La posición anteroposterior (AP) se representa horizontalmente con respecto al plano de amputación (posición $AP = 0\mu m$). En el eje vertical, se muestra el tiempo luego la amputación (tiempo = 0 días) y las réplicas experimentales dentro de cada tiempo. El código de color corresponde al porcentaje de células Transición-AxFUCCI. El aumento de las células AxFUCCI de transición en el día 1 y 2 después de la amputación se observa fácilmente.

CAPÍTULO 4

Sincronización inducida por el reclutamiento y sincronización pre-existente a la amputación: predicciones del modelo 1D

Un minuto que pasa es irrecuperable. Conociendo esto, ¿cómo podemos malgastar tantas horas?

—Mahatma Gandhi

4.1 Sincronización en la fase $G1$ al momento del reclutamiento

En el capítulo anterior se validaron las predicciones del modelo unidimensional de la médula espinal del axolotl a través de la técnica Fucci. Como emergente del modelo propuesto se tiene una sincronización de las células que se encuentran en la fase $G1$ temprana y media del ciclo largo al momento del reclutamiento. Al momento en que la señal haga efecto sobre las células, estas se sincronizarán al principio de la fase $G1$ en el ciclo corto. Con el objetivo de estudiar el papel de esta sincronización, realicé una nueva variante del modelo 1D, en la cual, en lugar de experimentar una sincronización a nivel de la fase $G1$, las células son mapeadas proporcionalmente, en un mecanismo similar al que implemento en la fase S . En la figura 4.1 reporto el resultado de las simulaciones del crecimiento de la médula espinal para esta nueva variante del modelo y lo comparo

con la variante del modelo de sincronización parcial en la fase G1 y mapeo proporcional de *S* junto, además, con el escenario sin reclutamiento celular. Los resultados indican que no existen diferencias significativas entre esta nueva variante del modelo de mapeo proporcional en la fase G1 y la anterior variante del modelo de sincronización parcial en G1 dentro de un desvío estándar, concluyendo que la sincronización parcial a nivel de la fase G1 no es necesaria para explicar, al menos, el crecimiento experimental de la médula espinal del axolotl.

Si nos ponemos en la situación de una célula al principio de la fase G1 en el ciclo largo, iremos a parar, luego del reclutamiento, al principio de la fase G1 en el ciclo corto. Esto ocurre en ambas variantes del modelo, tanto para el mapeo proporcional de G1 como para la sincronización parcial de G1. Si una célula se encuentra al final de la porción evitable de la fase G1 en la variante de sincronización parcial terminará igualmente al principio del ciclo corto de la fase G1. Por otra parte, si una célula se encuentra en esa misma posición en la variante del modelo con mapeo proporcional de la fase G1, esta será mapeada proporcionalmente al ciclo corto, obteniéndose una mayor aceleración de la fase G1 respecto a la variante de sincronización parcial. De esta manera, uno podría obtener cuanto tiempo se ahorrará una célula en las dos variantes del modelo dependiendo de la posición inicial en el ciclo celular largo (Figura 4.2 A) y, asimismo, calcular la diferencia de aceleración que se espera entre las dos variantes del modelo (Figura 4.2 B). Cuando se comparan de esta forma las dos variantes del modelo, nos encontramos con un ahorro considerable en el tiempo de tránsito por la fase G1 que, por otro lado, no se ve reflejado en el crecimiento de la médula espinal como se muestra en la Figura 4.1.

Luego estudié cómo es la distribución de células en fase G1 y en fase *S* (Figura 4.3A y B, respectivamente) como función del tiempo y el espacio. Se evidencia que las células salen antes de G1 entre día 0 y día 1 para el modelo de mapeo proporcional, mientras que en el modelo de sincronización las células salen de G1 recién para el día 2. En el mismo sentido, las células entran entre día 0 y día 1 a fase *S* en el modelo de mapeo proporcional mientras que en el modelo de sincronización parcial entran a fase *S* recién a día 2. Por otra parte, las células más cercanas al extremo posterior de la médula espinal a día 5 vuelven a entrar en fase G1 en forma sincronizada en el modelo de mapeo proporcional mientras que continúan en fase *S* en el modelo de mapeo proporcional. Los datos experimentales, por su parte, reflejan una situación similar a la observada en el modelo de sincronización parcial: las células salen de la fase G1 para entrar en la fase *S* recién entre día 2 y 3. Adicionalmente, a día 5, las células en el extremo posterior de la médula espinal continúan en fase *S*, no llegando a entrar nuevamente en G1 antes que finalice el día 5. La observación de los resultados parecen reflejar que es necesaria una sincronización a nivel del ciclo celular durante el reclutamiento para explicar el número de células que se encuentran la fase G1 y la fase *S* del ciclo celular en tiempo y espacio en los primeros 5 días de la regeneración de la médula espinal del axolotl.

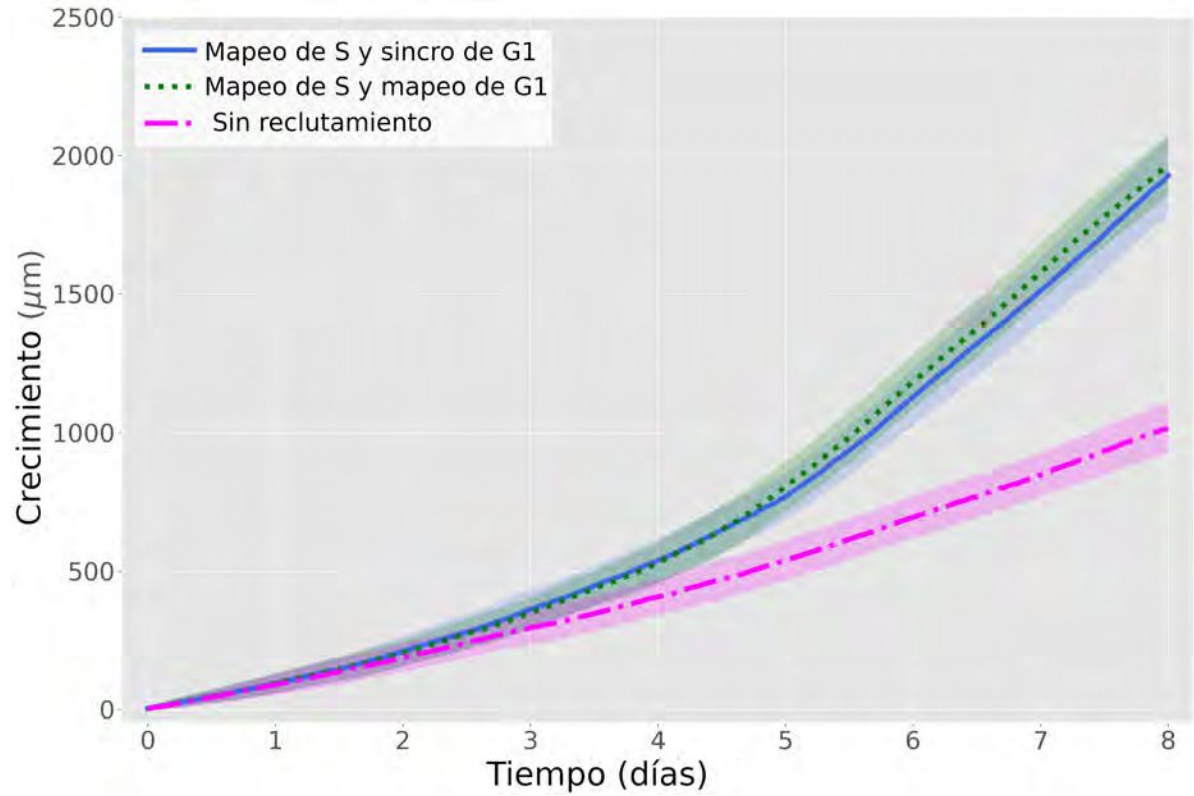


Figura 4.1: No existen diferencias significativas entre las dos variantes de acortamiento de la fase G1 para el modelo dentro de un desvío estándar. El crecimiento para la variante de mapeo de la fase G1 se encuentra por encima del crecimiento para la variante de sincronización parcial de la fase G1 a partir de día 5. De cualquier manera, no existen diferencias significativas para las dos variantes dentro de un desvío estándar.

4.2 Sincronización en la condición inicial

En la sección anterior exploré el efecto de la sincronización como consecuencia del reclutamiento de las células por la señal. Mi modelo asume otra sincronización, además, en la condición inicial dada por la posición en la que se encuentran las células a lo largo de su ciclo celular inmediatamente después de la amputación. En otras palabras, cada célula tiene una cierta edad dentro de su ciclo celular, $C_i(x_i, t)$, definida como una coordenada a lo largo del ciclo celular o reloj ($0 \leq C_i(x_i, t) < T_i(x_i, t)$). En la condición inicial, cada célula que se encuentra ciclando tiene una edad aleatoria $C_i(x_i, t = 0) = C_i^0$ a lo largo de la longitud del ciclo celular en particular, donde la distribución de C_i^0 viene dada por $(\frac{\ln 2}{T_i(x_i, t)})2^{1 - \frac{pC_i^0}{T_i(x_i, t)}}$. En esta sección exploraré las consecuencias de modificar esta condición inicial variando el parámetro p , el cuál permite regular esta sincronización al inicio de

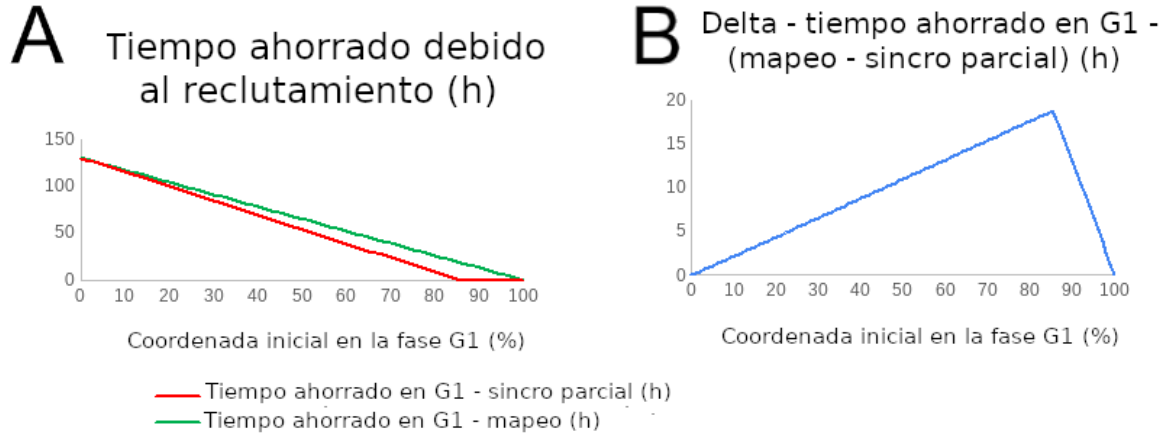


Figura 4.2: Las células en la fase G1 experimental presentan una mayor aceleración para el modelo de mapeo en comparación con el de mapeo proporcional de la fase G1. A) El tiempo ahorrado (en horas) para una célula en función de la posición inicial en la fase G1 del ciclo celular es mayor para el modelo de mapeo de la fase G1 cuando se lo compara con el modelo de sincronización parcial. Esta diferencia se incrementa hasta un máximo (85% de la coordenada inicial de la fase G1 larga) correspondiente al inicio de la fase evitable de la G1 en el modelo de sincronización parcial. B) Aceleración de una célula en la fase G1 para el modelo de mapeo respecto al modelo de sincronización parcial.

las simulaciones (Figura 4.4). En el extremo de $p = 0$ se tiene una distribución uniforme, mientras que a medida que se aumenta el valor de p la distribución se vuelve exponencial, con una caída más pronunciada a medida que p se hace más grande. Finalmente, $p = 1$ corresponde al caso de una distribución de estado estacionario.

Cuando $p = 0$, se obtiene un escenario donde no existe sincronización en la condición inicial que corresponde a una distribución uniforme de las células a lo largo del ciclo celular en todo el tejido. En este escenario, el crecimiento de la médula espinal es sobrestimado por el modelo para todos los tiempos (Figura 4.4). En los casos $p = 1$ y $p = 2$, la predicción mejora: $p = 1$ reproduce mejor los datos experimentales a tiempos largos pero sobrestima demasiado el crecimiento a tiempos cortos, en cambio $p = 2$, genera una predicción del crecimiento más fiel a los datos experimentales, tanto para tiempos cortos como para tiempos largos (Figura 4.4). En estos dos casos existe sincronización de las células: un mayor número de ellas se encuentran en fase G1 dado que la distribución es más pesada más al principio del ciclo celular que al final de este. Por último, cuando se tiene $p = 3$, la sincronización es aún mayor. En esta variante del modelo hay un gran número de células en G1 y un número muy pequeño de células en fase S en la condición inicial. Como consecuencia de esto, el número de células que dividen al principio de la simulación, es decir tiempos cortos, será más pequeño, reproduciendo

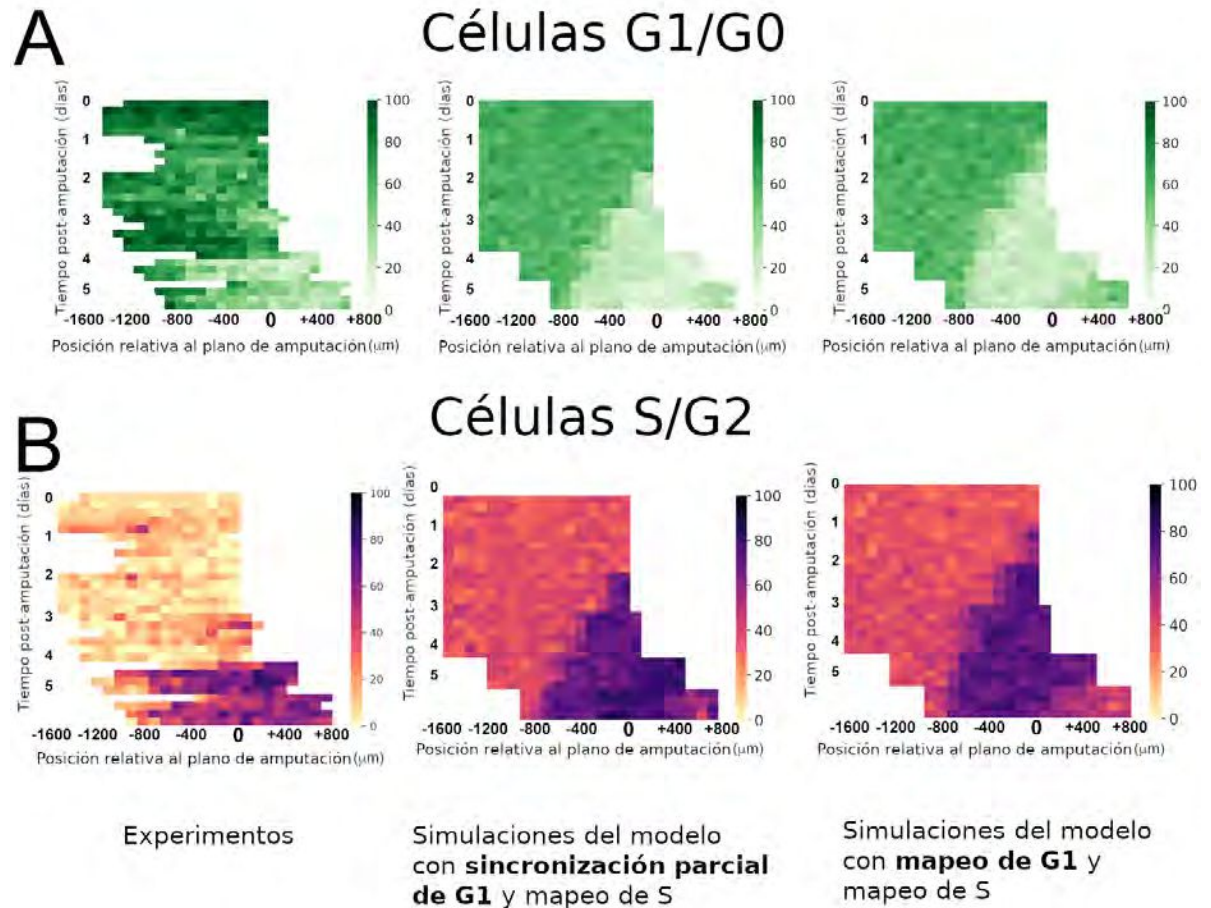


Figura 4.3: El modelo de sincronización parcial de G1 refleja mejor que el modelo de mapeo de G1 los datos experimentales. A y B) Al comparar con los datos experimentales con el modelo de mapeo de la fase G1 se observa que las células salen de la fase G1 y entran en fase S prematuramente (día 1) y, como consecuencia de esto, salen de fase S e ingresan nuevamente a fase G1 también antes de lo previsto (día 5). La predicción del modelo de sincronización parcial de la fase G1 resulta ser más fiel a lo esperado por los datos experimentales.

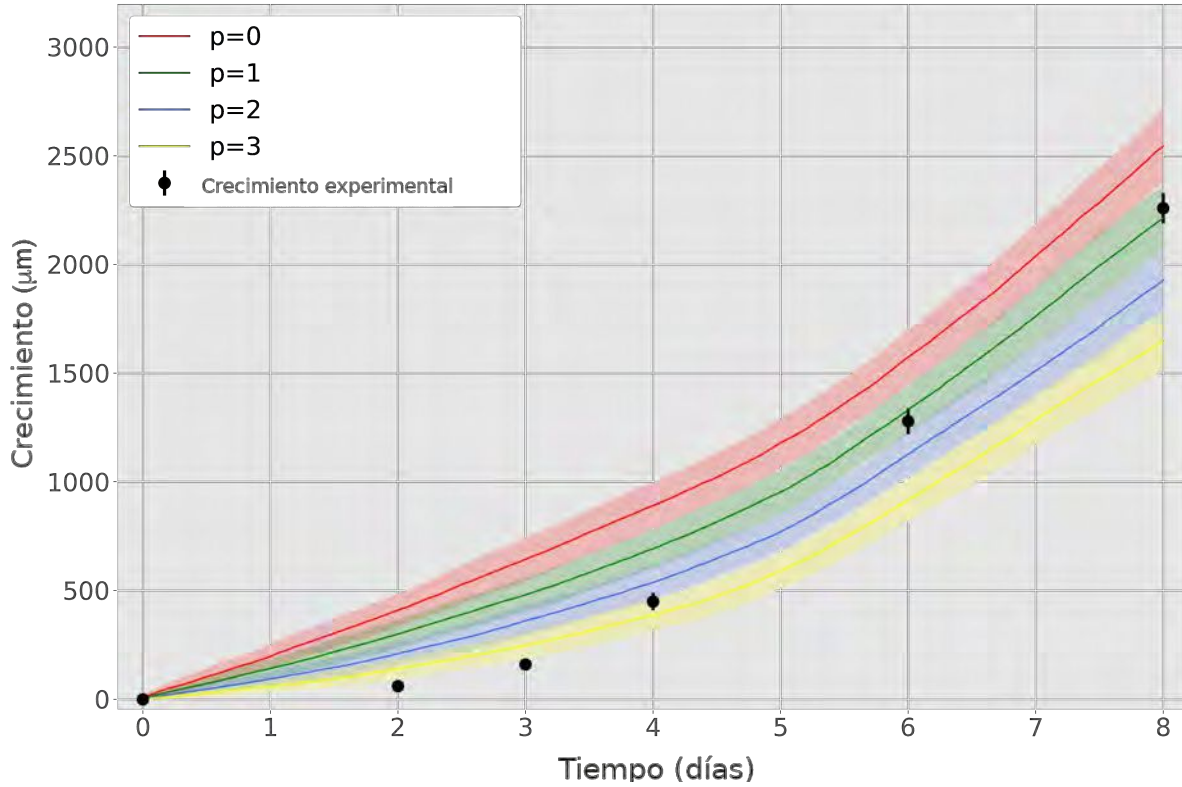


Figura 4.4: Una sincronización nula o muy baja ($p = 0$ y $p = 1$) llevan a una sobrestimación del outgrowth a días cortos, mientras que una sincronización alta ($p = 3$) lleva a una subestimación del outgrowth a días largos. Cuando las células se encuentran muy sincronizadas en la fase G1 el número de células que llegan a dividir hasta los 8 días es demasiado bajo, no alcanzando a recuperar el crecimiento a 8 días. En el otro extremo, si la sincronización en la fase G1 es baja nos encontramos con un mayor número de células en la fase S, las cuales dividen muy pronto dando un crecimiento superior al esperado entre los días 1 y 4. De esta situación de compromiso se obtiene que un valor de $p = 2$ es el que mejor predice el crecimiento en los 8 días simulados.

mejor el crecimiento a tiempos cortos que en los casos anteriores. Por otra parte, dado que es muy grande el número de células que se encuentran inicialmente en fase *G1* y que no llegarán a dividir en los tiempos de simulación, se observa una subestimación grande del crecimiento a tiempos largos (Figura 4.4). En resumen, podemos concluir que es necesaria una sincronización en la condición inicial pero que esta sincronización no debe ser excesivamente alta de forma de poder reproducir exitosamente los datos experimentales del crecimiento la médula espinal del axolotl.

A continuación me pregunté cuales son las consecuencias de la sincronización en la condición inicial sobre el número de células en fase *G1* y fase *S* (figura 4.5) en tiempo y espacio. Los resultados más interesantes tienen que ver con el comportamiento de la región no reclutada, es decir, alejado del plano de amputación. $p = 0$, me encontré con que el número de células en fase *G1* aumenta con el tiempo en esta región. En esta situación, estamos lejos de la situación de estado estacionario: el número de células en fase *G1* es menor que la del estado estacionario y el número de células en fase *S* es mayor a la del estado estacionario. Como consecuencia de esto, puede verse un aumento de la proporción de células en *G1* y una disminución de la proporción de células en *S*. Para las variantes con $p = 1$ y $p = 2$, las células en la zona no reclutada se encuentran cerca del estado estacionario y como consecuencia la proporción relativa de células en fase *G1* y fase *S* no varía demasiado. Por último, en el caso de $p = 3$ nos encontramos con un escenario opuesto al de $p = 0$, el número de células en la zona no reclutada en fase *G1* es muy grande y comienza a disminuir a medida que avanza la simulación mientras que el número de células en fase *S* aumenta. Será interesante aprender de nuevos datos experimentales en condiciones de crecimiento y a tiempos más largos que ocurre con la proporción de células en fase *G1* y fase *S*, con el objeto de saber si las células en la zona no reclutada se encuentran en estado estacionario, o si por el contrario, existe algún tipo de oscilación en la proporción de las fases lejos del plano de amputación.

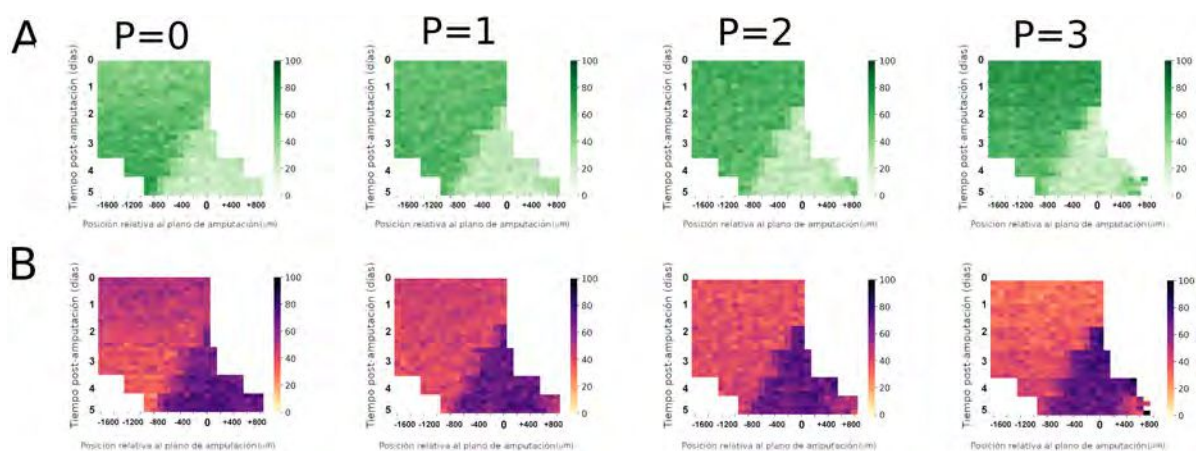


Figura 4.5: La sincronización en la condición inicial afecta las proporciones de células en la fase G1/G0 y S/G2 a lo largo de la simulación. A y B) Un grado de sincronización muy bajo en la condición inicial $p = 0$ lleva a un cambio notable en las proporciones de células tanto en (A) como en (B) especialmente en la zona región no reclutada, las células salen de la fase S para entrar en G1 a lo largo de la simulación. Análogamente en el extremo de alta sincronización $p = 3$, las células salen de G1 para entrar en fase S en la zona no reclutado a medida que transcurre la simulación. Por su parte, en los casos de $p = 1$ y $p = 2$ nos encontramos en la zona no reclutada con una situación más cercana a un estado estacionario.

CAPÍTULO 5

Implementación computacional del modelo de vértices: SysVert

“¿Qué es real? ¿Cómo defines lo real? Si estás hablando de lo que puedes sentir, lo que puedes oler, lo que puedes saborear y ver, entonces lo real son simplemente señales eléctricas interpretadas por tu cerebro.”

—Morfeo (Matrix)

En esta tesis me propuse desarrollar un modelo que incluya aspectos mecánicos de la interacción entre células, y para ello realicé una implementación de un modelo de vértices celular. Este tipo de modelos se ha utilizado en distintos trabajos para estudiar cambios de topología de tejidos en *Drosophila*, *Pez Zebra*, *Hydra* y *Xenopus* (Aigouy *et al.*, 2010; Salbreux *et al.*, 2012; Farhadifar *et al.*, 2007; Aegerter-Wilmsen *et al.*, 2012). Los modelos se desarrollan para resolver un problema biológico específico y son difíciles de adaptar a otro sistema, por distintas razones. Sin embargo, existe alguna excepción como el paquete de simulaciones Chaste (Cooper *et al.*, 2020), que propone una biblioteca C++ de código abierto para modelar poblaciones de células o cómo surgen eventos específicos a nivel del sistema. Sin embargo, Chaste no cuenta con una interfaz gráfica ni resulta lo suficientemente intuitivo para personas con pocos o escasos conocimientos de programación y en particular de C++, por el contrario, se requiere mucho tiempo para lograr comprender su lógica y lograr simular algunos de los ejemplos que ofrece al nuevo usuario.

Durante mi tesis he desarrollado junto a otros miembros del grupo *SysVert*, una implementación de un modelo de vértices que tiene como objetivo ser lo suficientemente general para estudiar problemas biológicos diversos permitiendo, a su vez, a usuarios con escasos conocimientos en programación simular un tejido mediante un modelo de vértices. Python es el segundo lenguaje mas usado en el año 2021, sólo por detrás de JavaScript, que por otro lado está orientado a desarrollo web. Adicionalmente Python está pensado para ser un lenguaje intuitivo y con una curva de aprendizaje muy apropiada para nuevos programadores. Por estas razones decidimos incursionar en una implementación

de un modelo de vértices en Python que logre ser amigable con usuarios provenientes de disciplinas relacionadas a las ciencias biológicas y médicas que, en general, tienen escasos o nulos conocimientos en programación, a la vez que por su naturaleza de código abierto permita a usuarios con algún conocimientos en programación modificar el código o agregar módulos para adaptar el software a problemas más específicos y contribuir a su desarrollo.

5.1 Modelo de vértices

El modelo de vértices es un modelo de simulación computacional basado en agentes utilizado para modelar tejidos compactos. El mismo es un modelo del tipo “fuera de la red” (del inglés off-lattice). Esto significa que sus componentes espaciales no tienen posiciones limitadas dentro de una red sino que se pueden mover libremente por todo el espacio.

El modelo de vértices consiste en modelar a las células como polígonos. Se le asigna a cada vértice una posición en el espacio y en base a ella se realizan cálculos para determinar sus nuevas posiciones de acuerdo a sus interacciones, modeladas por medio de un Hamiltoniano efectivo, a medida que el sistema evoluciona dinámicamente en el tiempo. Hasta aquí las capacidades del programa permiten estudiar sistemas en dos dimensiones. Modelaremos a las células como prismas cuyas caras son polígonos, tal como muestra la figura 5.1. Debido a que existe una simetría con respecto a la altura del prisma, es suficiente estudiar lo que le sucede en un plano horizontal que corta a las células para saber lo que sucede en todo el tejido. Este tipo de aproximación resulta de mucha utilidad para el estudio de tejidos en los cuáles la región apical del mismo es prácticamente plana.

5.1.1 Descripción de la implementación del modelo de vértices

En primer lugar, se modela a las células como polígonos. Luego a cada vértice se le asigna una posición en el espacio. Esta información determina el estado del sistema a un tiempo dado (configuración inicial). Para que el sistema avance a otro tiempo, se definen fuerzas que actuarán sobre los vértices y cambiarán su posición, y por ende la morfología de las células. Con estas posiciones podremos definir el estado del sistema en el nuevo tiempo y repetir este proceso para obtener su comportamiento en tiempos sucesivos. Definiremos las fuerzas a partir de la energía total del sistema E . Primero debemos proponer un modelo para el comportamiento de la energía del sistema. Supondremos que la energía depende de las posiciones de los vértices. Luego simplemente calcularemos la fuerza resultante sobre un vértice como:

$$\sum_i \vec{F}_i = -\vec{\nabla}_i E \quad (5.1)$$

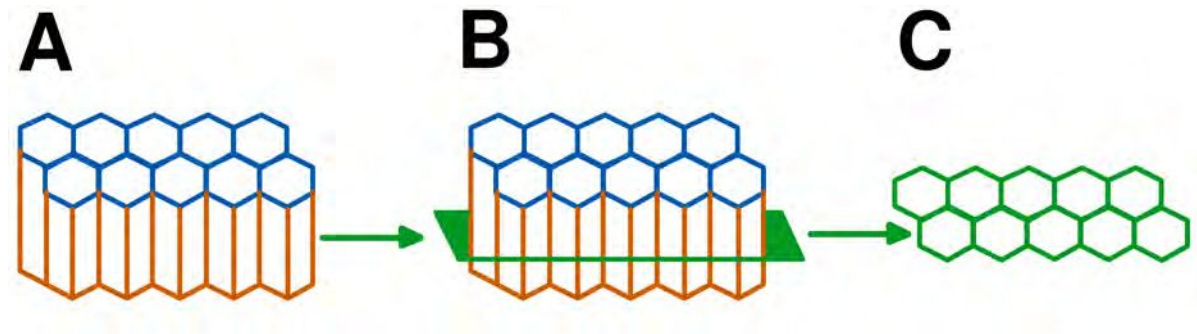


Figura 5.1: Esquema aproximado de cómo se modela a las células en de un tejido en 2D en un modelo de vértices. La figura A es el modelo tridimensional. En la figura B se realiza un corte sobre un plano paralelo a la región apical (superior en este esquema). En la figura C se muestra el resultado de la intersección entre ese plano y el tejido.

Donde el subíndice i se refiere al vértice i . La sumatoria suma sobre las fuerzas que actúan sobre el vértice i . Una vez que tenemos calculadas las fuerzas, debemos establecer cómo el sistema utiliza esta información para mover los vértices. Partiremos de la segunda ley de Newton:

$$\sum_i \vec{F}_i = m\vec{a}_i \quad (5.2)$$

Donde nuevamente el subíndice i se refiere al vértice i .

Supondremos que existe una fuerza viscosa y que esta es muy grande (sistema sobre-amortiguado), y que afecta a los vértices y que tiene la forma $-\lambda\vec{v}_i$, donde λ es una constante (coeficiente de viscosidad) y $\vec{v}_i$ es la velocidad del vértice i . En este régimen sobre-amortiguado la aceleración es muy pequeña y por lo tanto podemos despreciarla. Con estas aproximaciones obtenemos:

$$\sum_i \vec{F}_i - \lambda\vec{v}_i = 0 \quad (5.3)$$

$$\sum_i \vec{F}_i = \lambda\vec{v}_i \quad (5.4)$$

$$\frac{d\vec{x}_i}{dt} = \frac{\sum_i \vec{F}_i}{\lambda} \quad (5.5)$$

Donde $\vec{x}_i$ es la posición del vértice i . Hemos obtenido la expresión para la velocidad a la que se mueven los vértices. A partir de la velocidad, utilizaremos el método de Euler para obtener la nueva posición de los vértices:

$$\Delta \vec{x}_i = \frac{d\vec{x}_i}{dt} \Delta t \quad (5.6)$$

$$\vec{x}_i(t + \Delta t) = \vec{x}_i(t) + \frac{d\vec{x}_i}{dt} \Delta t \quad (5.7)$$

$$\vec{x}_i(t + \Delta t) = \vec{x}_i(t) - \frac{\vec{F}}{\lambda} \Delta t \quad (5.8)$$

Entonces dadas las posiciones del sistema a un tiempo t y una definición de la energía total del sistema podemos calcular cuáles serán las nuevas posiciones a un tiempo $t + \Delta t$.

La actualización de las nuevas posiciones de los vértices en *SysVert* es llevada a cabo por la función *vertex_forces* definida e invocada en el archivo *forces.py*.

```

1 def vertex_line_force(configuration, vID, lineConstant='default'):
2     """ Calculate the line forces on a vertex """
3     linex = 0
4     liney = 0
5     for edgeID in configuration.getVertices()[vID].getEdgesBelonging():
6         edge = configuration.getEdges()[edgeID]
7         if lineConstant == 'default':
8             c = edge.getLineConstant()
9         else:
10            c = float(lineConstant)
11         if vID == edge.getVerticesBelonging()[0]:
12             deltaX, deltaY, modulus = edge.getDistances(configuration)
13             if modulus != 0:
14                 linex -= c * deltaX/modulus
15                 liney -= c * deltaY/modulus
16             else:
17                 linex -= c * deltaX/modulus
18                 liney -= c * deltaY/modulus
19         else:
20             deltaX, deltaY, modulus = edge.getDistances(configuration)
21             linex += c * deltaX/modulus
22             liney += c * deltaY/modulus
23     return linex, liney
24
25
26 def vertex_area_force(configuration, vID):
27     """ Calculate the area forces on a vertex """
28     boxX = configuration.get_box_size('x')
29     boxY = configuration.get_box_size('y')
30     areaX = 0
31     areaY = 0
32     # Go through all the cells of that vertex
33     cellsBelonging = configuration.getVertices()[vID].getCellsBelonging()
34     for cellID in cellsBelonging:
35         cell = configuration.getCells()[cellID]
36         area = cell.getArea(configuration)
37         areaTarget = cell.getTargetArea()

```

```

38     rho = cell.get_area_elasticity_constant()
39     h0 = cell.get_cell_height()
40     cellVertices = cell.modified_vertices
41     theVertex = cell.getVertexOrder(vID)
42     if theVertex == len(cellVertices)-1:
43         deltaY = cellVertices[0][1] - cellVertices[theVertex-1][1]
44         deltaX = cellVertices[0][0] - cellVertices[theVertex-1][0]
45     elif theVertex == 0:
46         deltaY = cellVertices[theVertex+1][1] - cellVertices[len(
47             cellVertices)-1][1]
48         deltaX = cellVertices[theVertex+1][0] - cellVertices[len(
49             cellVertices)-1][0]
50     else:
51         deltaY = cellVertices[theVertex+1][1] - cellVertices[theVertex
52             -1][1]
53         deltaX = cellVertices[theVertex+1][0] - cellVertices[theVertex
54             -1][0]
55     if abs(deltaX) > boxX/2:
56         deltaX = deltaX - np.sign(deltaX) * boxX
57     if abs(deltaY) > boxY/2:
58         deltaY = deltaY - np.sign(deltaY) * boxY
59     areaX -= (rho) * (h0 ** 2) * (area-areaTarget) * deltaX
60     areaY += (rho) * (h0 ** 2) * (area-areaTarget) * deltaY
61     return areaX, areaY
62
63 def vertex_perimeter_force(configuration, vID):
64     boxX = configuration.get_box_size('x')
65     boxY = configuration.get_box_size('y')
66     perimeterx = 0
67     perimetery = 0
68     cellsBelonging = configuration.getVertices()[vID].getCellsBelonging()
69     for cellID in cellsBelonging:
70         cell = configuration.getCells()[cellID]
71         perimeter = cell.getPerimeter(configuration)
72         perimeterTarget = cell.getTargetPerimeter()
73         beta = cell.get_perimeter_elasticity_constant()
74         cellVertices = cell.modified_vertices
75         theVertex = cell.getVertexOrder(vID)
76         if theVertex == len(cellVertices)-1:
77             deltaxPrev = cellVertices[theVertex][0] - cellVertices[
78                 theVertex-1][0]
79             deltayNext = cellVertices[theVertex][0] - cellVertices[0][0]
80             deltayPrev = cellVertices[theVertex][1] - cellVertices[
81                 theVertex-1][1]
82             deltayNext = cellVertices[theVertex][1] - cellVertices[0][1]
83         elif theVertex == 0:
84             deltaxPrev = cellVertices[theVertex][0] - cellVertices[len(
85                 cellVertices)-1][0]
86             deltayNext = cellVertices[theVertex][0] - cellVertices[
87                 theVertex+1][0]
88             deltayPrev = cellVertices[theVertex][1] - cellVertices[len(

```

```

82         cellVertices) - 1][1]
            deltaxNext = cellVertices[theVertex][1] - cellVertices[
                theVertex + 1][1]
83
84     else:
85         deltaxPrev = cellVertices[theVertex][0] - cellVertices[
            theVertex - 1][0]
86         deltaxNext = cellVertices[theVertex][0] - cellVertices[
            theVertex + 1][0]
87         deltaxPrev = cellVertices[theVertex][1] - cellVertices[
            theVertex - 1][1]
88         deltaxNext = cellVertices[theVertex][1] - cellVertices[
            theVertex + 1][1]
89     if abs(deltaxPrev) > boxX/2:
90         deltaxPrev = deltaxPrev - np.sign(deltaxPrev) * boxX
91     if abs(deltaxNext) > boxX/2:
92         deltaxNext = deltaxNext - np.sign(deltaxNext) * boxX
93     if abs(deltayPrev) > boxY/2:
94         deltayPrev = deltayPrev - np.sign(deltayPrev) * boxY
95     if abs(deltayNext) > boxY/2:
96         deltayNext = deltayNext - np.sign(deltayNext) * boxY
97
98     modPrev = np.sqrt(deltaxPrev**2 + deltayPrev**2)
99     modNext = np.sqrt(deltaxNext**2 + deltayNext**2)
100    perimeterx += -2*beta*(perimeter - perimeterTarget)*(deltaxPrev /
        modPrev + deltaxNext / modNext)
101    perimetry += -2*beta*(perimeter - perimeterTarget)*(deltayPrev /
        modPrev + deltayNext / modNext)
102    return perimeterx, perimetry

```

Listing 5.1: Definiciones del cálculo de fuerzas de tensión de línea (*vertex_line_force*), área (*vertex_area_force*) y perímetro (*vertex_perimeter_force*) del tejido en el archivo *forces.py*.

```

1 def nextTime(configuration, dt, cellTypes, ctlc, signal, step):
2     madeT1 = []
3     viscosity = 1
4     divisionp = {}
5     deterministicDiv = {}
6     cellCycleLength = {}
7     cellCycleTime = {}
8     apoptp = {}
9     apoptc = {}
10    for cType, data in cellTypes.items():
11        deterministicDiv[cType] = data['deterministicDiv']
12        cellCycleLength[cType] = data['cellCycleLength']
13        cellCycleTime[cType] = data['cellCycleTime']
14        divisionp[cType] = data['divisionp']
15        apoptp[cType] = data['apoptoticProbability']
16        apoptc[cType] = data['apoptoticConstant']
17
18    changeMade = False

```

```

19     vertices = configuration.getVertices()
20     cells = configuration.getCells()
21     edges = configuration.getEdges()
22
23     lineEnergy = 0
24     areaEnergy = 0
25     perimeterEnergy = 0
26
27     threshold = signal.get_threshold()
28
29     for cellID, cell in configuration.getCells().items():
30         cell.calculate_periodic_positions(configuration)
31         # Comment for non-recruited cells
32         posX, posY = cell.get_cell_center(configuration)
33         if posX > threshold and cell.get_cell_typeID() == 0:
34             cell.set_cell_type(cellTypes["1"])
35             cell.cell_cycle_shortening()
36
37     # Calculates line, perimeter and area contributions to forces
38     vertices_forces = {}
39     for vertexID, vertex in vertices.items():
40
41         vertexID = vertex.getVertexID()
42         lx, ly = forces.vertex_line_force(configuration, vertexID)
43         px, py = forces.vertex_perimeter_force(configuration, vertexID)
44         ax, ay = forces.vertex_area_force(configuration, vertexID)
45         ex, ey = forces.vertex_expression_force(configuration, vertexID)
46         vertices_forces[vertexID] = [(lx + px + ax), (ly + py + ay)]
47
48     # Calculates delta positions
49     deltaX = ((lx + px + ax + ex)*dt) / viscosity
50     deltaY = ((ly + py + ay + ey)*dt) / viscosity
51
52     new_x = vertex.getX() + deltaX
53     new_y = vertex.getY() + deltaY
54
55     # Calculates new positions
56     if new_x > configuration.get_box_limits('right'):
57         new_x -= configuration.get_box_size('x')
58     elif new_x < configuration.get_box_limits('left'):
59         new_x += configuration.get_box_size('x')
60     if new_y > configuration.get_box_limits('top'):
61         new_y -= configuration.get_box_size('y')
62     elif new_y < configuration.get_box_limits('bottom'):
63         new_y += configuration.get_box_size('y')
64
65     if new_x > configuration.get_box_fixed_limits('right'):
66         vertex.fixed = True
67         new_x = configuration.get_box_fixed_limits('right')
68     elif new_x < configuration.get_box_fixed_limits('left'):
69         vertex.fixed = True
70         new_x = configuration.get_box_fixed_limits('left')

```

```

71     if new_y > configuration.get_box_fixed_limits('top'):
72         vertex.fixed = True
73         new_y = configuration.get_box_fixed_limits('top')
74     elif new_y < configuration.get_box_fixed_limits('bottom'):
75         vertex.fixed = True
76         new_y = configuration.get_box_fixed_limits('bottom')
77
78     if vertex.fixed:
79         new_x = vertex.getX()
80         new_y = vertex.getY()
81
82     # Virtual movement
83     vertex.set_new_x(new_x)
84     vertex.set_new_y(new_y)
85
86     # Saves the forces values to lattice
87     configuration.save_forces(vertices_forces)
88
89     # Updates positions
90     for vertexID, vertex in vertices.items():
91         vertices[vertexID].update()
92
93     # Division and apoptosis random selection
94     for dividingCell in list(cells):
95
96         if deterministicDiv[str(cells[dividingCell].get_cell_typeID())]:
97             # Deterministic division
98             cells[dividingCell].cellCycleTimeUpdate()
99             if cells[dividingCell].get_cell_cycle_length() <= cells[
100                 dividingCell].get_cell_cycle_time():
101                 vertices, edges, cells = cell_division(configuration,
102                     dividingCell, ctlc, cellTypes)
103                 changeMade = True
104
105         else:
106             # Probabilistic division
107             if np.random.uniform(0,1) < divisionp[str(cells[dividingCell].
108                 get_cell_typeID())]:
109                 vertices, edges, cells = cell_division(configuration,
110                     dividingCell, ctlc)
111                 changeMade = True
112
113         if np.random.uniform(0,1) < apoptp[str(cells[dividingCell].
114             get_cell_typeID())]:
115             # Apoptosis
116             cells[dividingCell].set_apoptotic()
117             cells[dividingCell].apoptotic_time_foward(apoptc[str(cells[
118                 dividingCell].get_cell_typeID())])
119
120     # T3 swaps
121     changeMadeT3 = False
122     edgesMedium = []

```



```

117     verticesMedium = set([])
118     for edgeID, edge in edges.items():
119         if len(edge.get_cells_belonging(vertices)) == 1:
120             edgesMedium.append(edgeID)
121             for vertexID in edge.getVerticesBelonging():
122                 verticesMedium.add(vertexID)
123     for vertexID in verticesMedium:
124         intersectingPosition, intersectingEdgeID = swaps.check_T3(
125             configuration, vertexID, edgesMedium, step)
126         if intersectingEdgeID:
127             vertices, edges, cells = swaps.swap_T3(configuration,
128                 vertexID, intersectingEdgeID, intersectingPosition)
129             changeMade = True
130             changeMadeT3 = True
131
132     # T1 swaps
133     for edgeID in list(edges.keys()):
134         checkResT1 = swaps.check_T1(configuration, edgeID)
135         if type(checkResT1) == int and checkResT1 not in madeT1 and not
136             changeMadeT3:
137             madeT1.append(checkResT1)
138             vertices, edges, cells = swaps.swap_T1(configuration,
139                 checkResT1, ctle)
140             changeMade = True
141
142     # T2 swaps
143     listT2 = swaps.check_T2(configuration)
144     if listT2 != []:
145         for cellID in listT2:
146             vertices, edges, cells = swaps.swap_T2(configuration, cellID)
147             changeMade = True
148
149     # Creates the new configuration
150     if changeMade:
151         return Lattice(configuration.data, vertices, edges, cells)
152     else:
153         return configuration

```

Listing 5.2: Archivo *solver.py* donde se realiza cada iteración temporal de las simulaciones, llamando a las funciones responsables del cálculo de las fuerzas, las transiciones y las divisiones.

5.1.2 Energía del sistema

Finalmente, introduciremos un modelo para la energía. El mismo consistirá en tres contribuciones: La energía de deformación E_D , la contractilidad E_C y la adhesión entre células E_A . La energía del sistema será simplemente la suma de estas tres contribuciones:

$$E = E_D + E_A + E_C \quad (5.9)$$

La energía de deformación intenta lograr que cada célula alcance en equilibrio un área objetivo. La misma será:

$$E_D = \sum_k \lambda_D (A_k - A_{kT})^2 \quad (5.10)$$

Donde λ_D es el coeficiente de deformación, A_k es el área de la célula k (ver Figura 5.2), A_{kT} es el área objetivo (target) de la célula k y la sumatoria se realiza sobre todas las células. Esta energía también puede ser vista como una energía que depende del volumen en lugar del área. Debido a que estamos considerando a las células como prismas, simplemente podemos reescribir al λ_D como:

$$\lambda_D = \lambda_{DV} h_0^2 \quad (5.11)$$

Con λ_{DV} el coeficiente de deformación volumétrico y h_0 la altura de la célula. Como el volumen será el área del prisma por su altura, obtenemos:

$$E_D = \sum_k \lambda_{DV} (V_k - V_{kT})^2 \quad (5.12)$$

donde V_k y V_{kT} son el volumen y el volumen objetivo de la célula k respectivamente.

Con respecto al área objetivo, Nagai y Honda (Nagai & Honda, 2001) proponen dos modelos. Uno donde las células censan el área de las células con las que están en contacto y el área objetivo es el promedio de ellas. El otro consiste en definir al área objetivo como una constante. En este trabajo utilizaremos el último modelo, siguiendo también el trabajo de Farhadifar y colaboradores (4).

El término de adhesión está asociado a tensiones debido a las uniones entre las células. Se pueden definir distintos tipos de células, es decir, células con distintas propiedades en sus membranas y obtener por lo tanto distintos tipos de tensiones entre ellas. La misma será de la forma:

$$E_A = \sum_j \lambda_{Aj} d_j \quad (5.13)$$

Donde λ_{Aj} es la tensión de línea de la arista j , d_j es el largo de la arista j (ver Figura 5.2) y la sumatoria se realiza sobre todas las aristas del tejido.

Se ha encontrado en trabajos anteriores (Schilling *et al.*, 2011; Landsberg *et al.* 2009) que una inhomogeneidad en la tensión de línea es suficiente para explicar por qué los compartimientos anterior y posterior del disco imaginal del ala de *Drosophila Melanogaster* (mosca de fruta, el modelo biológico más utilizado para este tipo de estudios) permanecen separados durante el desarrollo. Es importante que estos compartimientos permanezcan separados porque en estadios posteriores del desarrollo los mismos darán lugar a tejidos diferentes.

Siguiendo con el modelo de energía utilizado en este trabajo, el término de contracción será parecido al de deformación pero en este caso el perímetro ocupará el lugar del área:

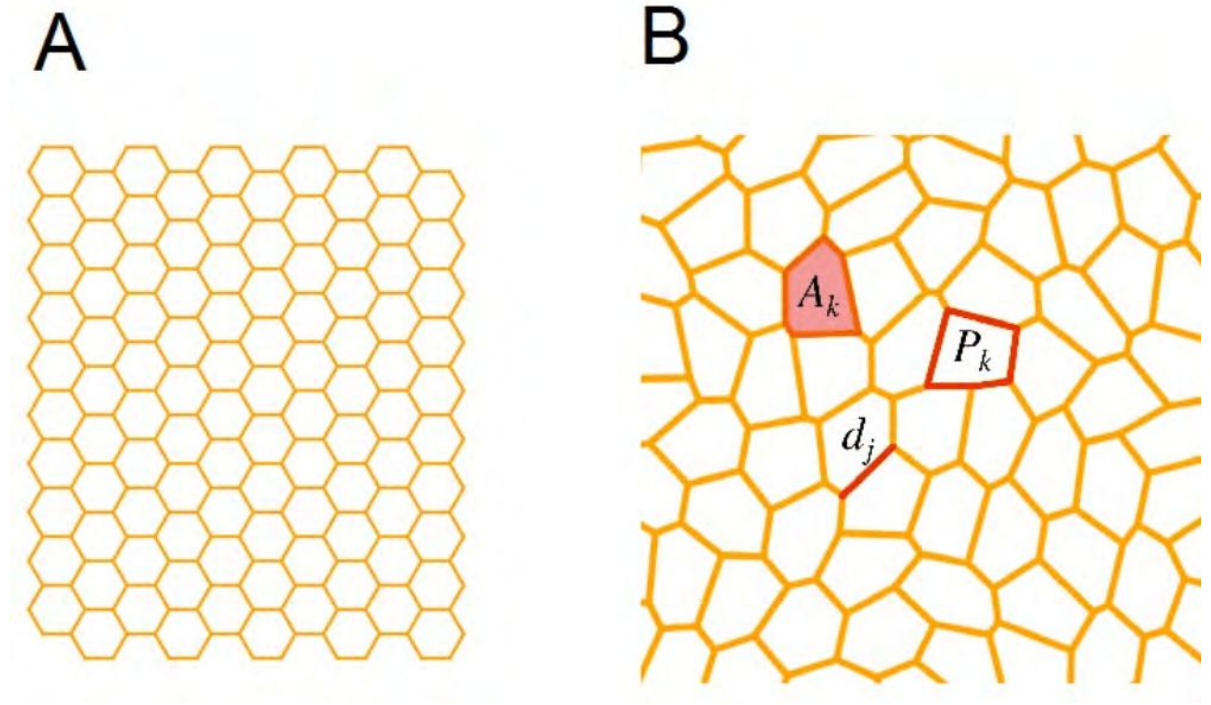


Figura 5.2: Representación gráfica de un tejido modelado mediante el modelo de vértices. El panel A muestra como el tejido en el instante inicial está formado por células con forma hexagonal (honeycomb). El panel B es un esquema que muestra a qué nos referimos cuando hablamos del área de una célula (A_k), de perímetro de una célula (P_k) y del largo de una arista (d_j).

$$E_C = \sum_k \lambda_C (P_k - P_{kT})^2 \quad (5.14)$$

Donde λ_C es el coeficiente de contractilidad, P_k es el perímetro de la célula k (ver Figura 5.2), P_{kT} es el perímetro objetivo de la célula k y la sumatoria se realiza sobre todas las células. Este término es importante para lograr que las células tengan una forma convexa.

5.1.3 Intercambios T (T-Swaps)

Como es usual en este modelo, introduciremos los intercambios T (del inglés “swaps”). Los mismos son cambios topológicos que se dan en el sistema por distintas razones, que en general remedan el comportamiento topológico del tejido que se modela. En esta implementación utilizaremos los tipos de intercambios conocidos como T1, T2 y T3. Estos intercambios han sido introducidos en estudios de modelos de vértices en

espumas, que son los que dan origen al modelo de vértices celular (6). Otras elecciones de intercambios son posibles.

5.1.3.1 Intercambios T1

Durante el intercambio T1 las células se deforman como muestra la Figura 5.3. El intercambio T1 sucede cuando un lado de un polígono se vuelve menor a una cantidad ΔL , previamente establecida. El lado desaparece y otro nuevo, perpendicular al anterior y con un largo L_0 predefinido, lo reemplaza. Durante este intercambio hay un re-ordenamiento de vértices entre las células. Dos células obtienen un vértice extra (A y D en la Figura 5.3) y dos células pierden uno, produciéndose adicionalmente un intercambio de vecinos (B y C en la Figura 5.3).

La implementación de los intercambios T1 se implementa en la función *swap_T1* que puede encontrarse en el archivo *swaps.py*.

```

1 def swap_T1(configuration, edgeID, cellTypesLineConstant):
2     """
3     From the vertices, edges and cells passed it applies T1 and T2 swaps.
4     It returns the new vertices, edges and cells dictionaries.
5     t1: array with possible edges for a T1 swap.
6     """
7
8     vertices = configuration.getVertices()
9     edges = configuration.getEdges()
10    cells = configuration.getCells()

```

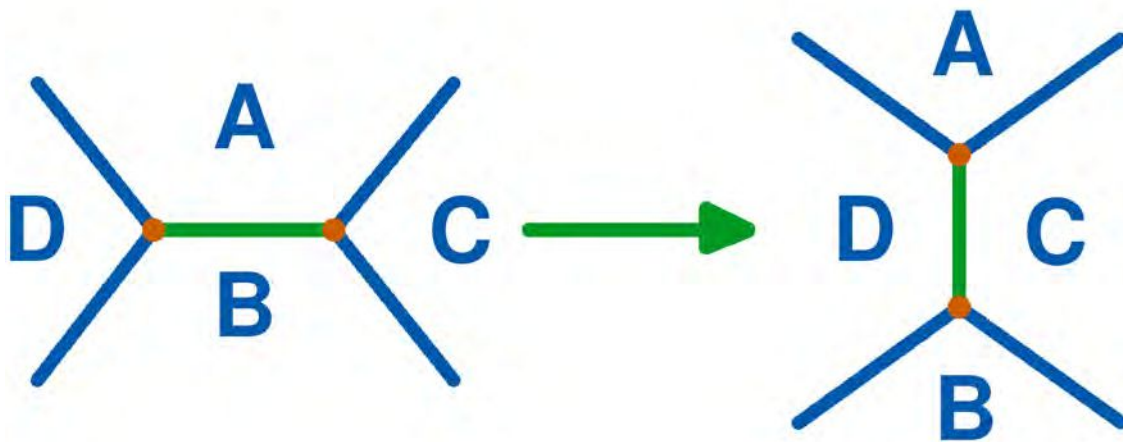


Figura 5.3: Esquema del intercambio T1. El lado en verde tiene un largo menor al valor límite y sufre un intercambio T1. Cuatro células (A, B, C y D) participan en el mismo.

```

11
12     sets = []
13     for givenVertex in edges[edgeID].getVerticesBelonging():
14         sets.append(vertices[givenVertex].getCellsBelonging())
15     sharedCells = np.intersect1d(sets[0], sets[1])
16     for eachCell in sharedCells:
17         if cells[eachCell].getVertexNumber() <= 3:
18             return vertices, edges, cells
19     # Check if still is necessary to do the swap
20     deltaX, deltaY, distance = edges[edgeID].getDistances(configuration)
21     cellB = []
22     cellID = []
23     if(round(distance, 4) > edges[edgeID].minimumDistance):
24         return vertices, edges, cells
25     # Do the swap
26     ksep = 1.5
27     newDist = edges[edgeID].minimumDistance * ksep
28     # Charge the vertices where we will operate
29     vi = vertices[edges[edgeID].getVerticesBelonging()[0]]
30     vj = vertices[edges[edgeID].getVerticesBelonging()[1]]
31
32     # Calculate the middle point
33     periodicX, periodicY = edges[edgeID].periodic_edge(configuration)
34
35     boxX = configuration.get_box_size('x')
36     boxY = configuration.get_box_size('y')
37
38     if periodicX:
39         posX = (vi.getX()+vj.getX() - boxX)/2
40         if posX < configuration.get_box_limits('left'):
41             posX = posX + boxX
42     else:
43         posX = (vi.getX()+vj.getX())/2
44     if periodicY:
45         posY = (vi.getY()+vj.getY() - boxY)/2
46         if posY < configuration.get_box_limits('bottom'):
47             posY = posY + boxY
48     else:
49         posY = (vi.getY()+vj.getY())/2
50     middle = [posX, posY]
51
52     existEdge = True
53     # Check if there is only one cell at play
54     if len(vi.getCellsBelonging()) == 1 and len(vj.getCellsBelonging()) ==
55         1:
56         # join vertices
57         newx = middle[0]
58         newy = middle[1]
59
60         for currentEdge in vj.getEdgesBelonging():
61             if currentEdge != edgeID:
62                 edges[currentEdge].set_vertex_from_ID(vi, vj.getVertexID())

```

```

62
63     # Destroy edge == edgeID
64     vi.remove_edges_belonging(edgeID)
65     del edges[edgeID]
66     existEdge = False
67     # remove vertex from cell
68     cells[vj.getCellsBelonging()[0]].remove_vertex(vj.getVertexID())
69     # save new positions
70     vertices[vi.getVertexID()].set_new_x(newx)
71     vertices[vi.getVertexID()].set_new_y(newy)
72
73     vertices[vi.getVertexID()].update()
74
75     del vertices[vj.getVertexID()]
76     # Check if you have only two cells (Cell B and Cell C)
77     elif (np.asarray(vi.getCellsBelonging() == vj.getCellsBelonging())).
78         all() or np.asarray((np.flip(vi.getCellsBelonging(), axis=0) == vj.
79         getCellsBelonging())).all():
80         # Join vertices
81         newx = middle[0]
82         newy = middle[1]
83         for currentEdge in np.asarray(vj.getEdgesBelonging()):
84             if currentEdge != edgeID:
85                 edges[currentEdge].set_vertex_from_ID(vi, vj.getVertexID())
86
87     # Destroy edge == edgeID
88     vi.remove_edges_belonging(edgeID)
89     del edges[edgeID]
90     existEdge = False
91     # Remove vertex from cell
92     cells[vj.getCellsBelonging()[0]].remove_vertex(vj.getVertexID())
93     cells[vj.getCellsBelonging()[1]].remove_vertex(vj.getVertexID())
94     # Save new positions
95     vertices[vi.getVertexID()].set_new_x(newx)
96     vertices[vi.getVertexID()].set_new_y(newy)
97
98     vertices[vi.getVertexID()].update()
99
100     del vertices[vj.getVertexID()]
101     # Else will do the standar swap algorithm
102     else:
103         # Virtually move the points to the new origin
104         if abs(vi.getX() - middle[0]) > boxX/2:
105             if vi.getX() - middle[0] > 0:
106                 newy0 = -vi.getX() + middle[0] - boxX
107             elif vi.getX() - middle[0] < 0:
108                 newy0 = -vi.getX() + middle[0] + boxX
109         else:
110             newy0 = -vi.getX() + middle[0]
111
112         if abs(vj.getX() - middle[0]) > boxX/2:
113             if vj.getX() - middle[0] > 0:

```

```

112         newy1 = -vj.getX() + middle[0] - boxX
113         elif vj.getX() - middle[0] < 0:
114             newy1 = -vj.getX() + middle[0] + boxX
115     else:
116         newy1 = -vj.getX() + middle[0]
117
118     if abs(vi.getY() - middle[1]) > boxX/2:
119         if vi.getY() - middle[1] > 0:
120             newx0 = vi.getY() - middle[1] - boxX
121         elif vi.getY() - middle[1] < 0:
122             newx0 = vi.getY() - middle[1] + boxX
123     else:
124         newx0 = vi.getY() - middle[1]
125
126     if abs(vj.getY() - middle[1]) > boxX/2:
127         if vj.getY() - middle[1] > 0:
128             newx1 = vj.getY() - middle[1] - boxX
129         elif vj.getY() - middle[1] < 0:
130             newx1 = vj.getY() - middle[1] + boxX
131     else:
132         newx1 = vj.getY() - middle[1]
133
134     # Edge shrinking such that it have norm equal to 1
135     newiv = (newx0, newy0)/(np.sqrt(newx0**2+newy0**2))
136     newjv = (newx1, newy1)/(np.sqrt(newx1**2+newy1**2))
137
138     # Enlarge the edge to the desired norm
139     newi = newiv * newDist/2 + middle
140     newj = newjv * newDist/2 + middle
141     #if vertex i perioc
142     if newi[0] < configuration.get_box_limits('left'):
143         newi[0] += boxX
144     elif newi[0] > configuration.get_box_limits('right'):
145         newi[0] -= boxX
146     if newi[1] < configuration.get_box_limits('bottom'):
147         newi[1] += boxY
148     elif newi[1] > configuration.get_box_limits('top'):
149         newi[1] -= boxY
150     #if vertex j perioc
151     if newj[0] < configuration.get_box_limits('left'):
152         newj[0] += boxX
153     elif newj[0] > configuration.get_box_limits('right'):
154         newj[0] -= boxX
155     if newj[1] < configuration.get_box_limits('bottom'):
156         newj[1] += boxY
157     elif newj[1] > configuration.get_box_limits('top'):
158         newj[1] -= boxY
159
160
161     # Finds cells B and C (at least one exists) but dont know which is
162     # which
163     exist2 = False

```

```

163     cell1 = list(set(vi.getCellsBelonging()).intersection(vj.
164                   getCellsBelonging()))[0]
165
166     # Checks if the other cell ("cell2") exists by counting the number
167     # of cells owned by verticies vi and vj
168     if len(set(vi.getCellsBelonging()).intersection(vj.
169             getCellsBelonging())) == 2:
170         cell2 = list(set(vi.getCellsBelonging()).intersection(vj.
171             getCellsBelonging()))[1]
172         exist2 = True
173
174     existA = existB = existC = existD = False
175     existV1 = existV2 = existV3 = existV4 = False
176
177     # Determines which is cell B and which is cell C by looking at
178     # vertices order of vi and vj inside the cells (taking into
179     # account that vi would be 0)
180     if (cells[cell1].getVertexOrder(vi.getVertexID()) > cells[cell1].
181         getVertexOrder(vj.getVertexID()) and not ((cells[cell1].
182             getVertexOrder(vi.getVertexID()) == cells[cell1].
183             getVertexNumber()-1 and cells[cell1].getVertexOrder(vj.
184             getVertexID()) == 0)) or (cells[cell1].getVertexOrder(vi.
185             getVertexID()) == 0 and cells[cell1].getVertexOrder(vj.
186             getVertexID()) != 1):
187         cellC = cell1
188         existC = True
189         if exist2:
190             cellB = cell2
191             existB = True
192     else:
193         cellB = cell1
194         existB = True
195         if exist2:
196             cellC = cell2
197             existC = True
198
199     # Determines if cell A exists
200     neighborsA = []
201     if existB:
202         neighborsA.append(cellB)
203     if existC:
204         neighborsA.append(cellC)
205
206     cellsVi = set(vi.getCellsBelonging()).symmetric_difference(
207         neighborsA)
208
209     if len(cellsVi) == 1:
210         existA = True
211         cellA = list(cellsVi)[0]
212     elif len(cellsVi) > 1:
213         print("Error: Trying to do a T1 swap with more than one cell A
214               ")

```



```

201         print("Shared cells:", sharedCells)
202         print("cellsVi:", cellsVi)
203         exit()
204
205     # Determines if cell D exists
206     neighborsD = []
207     if existB:
208         neighborsD.append(cellB)
209     if existC:
210         neighborsD.append(cellC)
211
212     cellsVj = set(vj.getCellsBelonging()).symmetric_difference(
213         neighborsD)
214
215     if len(cellsVj) == 1:
216         existD = True
217         cellD = list(cellsVj)[0]
218     elif len(cellsVj) > 1:
219         print("Error: Trying to do a T1 swap with more than one cell D")
220         exit()
221
222     # Assigns vertices 1 and 4
223     if existC:
224         vertex1 = cells[cellC].next_vertex(vi.getVertexID())
225         existV1 = True
226         vertex4 = cells[cellC].previous_vertex(vj.getVertexID())
227         existV4 = True
228     else:
229         if existA:
230             vertex1 = cells[cellA].previous_vertex(vi.getVertexID())
231             existV1 = True
232         if existD:
233             vertex4 = cells[cellD].next_vertex(vj.getVertexID())
234             existV4 = True
235
236     # Assigns vertices 2 and 3
237     if existB:
238         vertex2 = cells[cellB].previous_vertex(vi.getVertexID())
239         existV2 = True
240         vertex3 = cells[cellB].next_vertex(vj.getVertexID())
241         existV3 = True
242     else:
243         if existA:
244             vertex2 = cells[cellA].next_vertex(vi.getVertexID())
245             existV2 = True
246         if existD:
247             vertex3 = cells[cellD].previous_vertex(vj.getVertexID())
248             existV3 = True
249
250     # Vertices exchanges
251     if existC:

```

```

251         # Removes vi from C
252         cells[cellC].remove_vertex(vi.getVertexID())
253         vi.remove_cells_belonging(cellC)
254     if existD:
255         # D gains a new vertex, vi, in the order of vj
256         cells[cellD].add_vertex(vi, cells[cellD].getVertexOrder(vj.getVertexID()))
257         vi.addCellsBelonging(cellD)
258     if existB:
259         # Removes vj from B
260         cells[cellB].remove_vertex(vj.getVertexID())
261         vj.remove_cells_belonging(cellB)
262     if existA:
263         # A gains a new vertex, vj, in the order of vi
264         cells[cellA].add_vertex(vj, cells[cellA].getVertexOrder(vi.getVertexID()))
265         vj.addCellsBelonging(cellA)
266     if existV3:
267         for edgeNum in vj.getEdgesBelonging():
268             if vertex3.getVertexID() in edges[edgeNum].getVerticesBelonging():
269                 edges[edgeNum].set_vertex_from_ID(vi, vj.getVertexID())
270     if existV1:
271         for edgeNum in vi.getEdgesBelonging():
272             if vertex1.getVertexID() in edges[edgeNum].getVerticesBelonging():
273                 edges[edgeNum].set_vertex_from_ID(vj, vi.getVertexID())
274
275     vertices[edges[edgeID].getVerticesBelonging()[0]].set_new_x(newi[0])
276     vertices[edges[edgeID].getVerticesBelonging()[0]].set_new_y(newi[1])
277
278     vertices[edges[edgeID].getVerticesBelonging()[1]].set_new_x(newj[0])
279     vertices[edges[edgeID].getVerticesBelonging()[1]].set_new_y(newj[1])
280
281     vertices[edges[edgeID].getVerticesBelonging()[0]].update()
282     vertices[edges[edgeID].getVerticesBelonging()[1]].update()
283
284     #Change line constant if it is in the border
285     if existEdge:
286         cellsBelonging = edges[edgeID].get_cells_belonging(vertices)
287         if len(cellsBelonging) == 1:
288             edges[edgeID].set_line_constant(cellTypesLineConstant[str(
289                 cells[cellsBelonging[0]].get_cell_typeID())+'M'])
290         else:
291             edges[edgeID].set_line_constant(cellTypesLineConstant[str(
292                 cells[cellsBelonging[0]].get_cell_typeID())+str(cells[
293                     cellsBelonging[1]].get_cell_typeID())])

```

292 `return vertices , edges , cells`

Listing 5.3: Definiciones de los intercambios T1 del tejido en el archivo *swaps.py*.

5.1.3.2 Intercambios T2

Un intercambio T2 sucede cuando una célula tiene tres lados y un área menor a un valor límite ΔA . Podría ocurrir que células de 4 o 5 lados pasen por un intercambio T2 pero la probabilidad de que eso ocurra es mucho menor que en el caso de células de 3 lados (Stavans *et al.*, 1993). Por ello nos limitaremos a células de tres lados. Durante un intercambio T2, la célula es eliminada y sus tres vértices se unen en el centro de la misma. Las tres células que la rodean pierden un vértice durante el intercambio. Es importante aclarar que este intercambio no está relacionado necesariamente con la muerte celular (apoptosis). Si se desea que el modelo tenga en cuenta la tasa de muerte celular, esta debe ser introducida aparte. Lo que sucede en el intercambio T2 es que la célula eliminada sale del plano que estamos estudiando debido a que es empujada por sus vecinas. Un esquema del intercambio T2 se puede ver en la figura 5.4.

La implementación de los intercambios T2 se implementa en la función *swap_T2* que puede encontrarse en el archivo *swaps.py*.

```
1 def swap_T2(configuration , cellID):
2     """
3     If a cell has less than 4 vertices and the area is less than a
4     threshold,
5     the cell is removed from the mesh by this function. (Fletcher 2013)
```

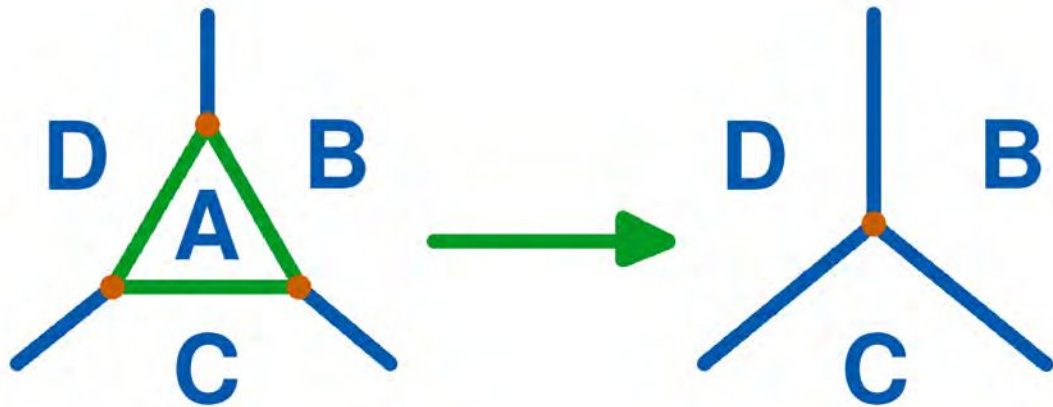


Figura 5.4: Esquema del intercambio T2. En el mismo la célula A desaparece y sus lados se unen en el centro de la misma. Las células B, C y D pierden un vértice en el proceso.

```

5      """
6      vertices = configuration.getVertices()
7      edges = configuration.getEdges()
8      cells = configuration.getCells()
9
10     # Calculates cell center
11     centerx, centery = cells[cellID].get_cell_center(configuration)
12     verticesNumber = max(vertices.keys()) + 1
13     vertices[verticesNumber] = Vertex(centerx, centery)
14     vertices[verticesNumber].setVertexID(verticesNumber)
15
16     # Assigns cells to the new vertex
17     myVertices = cells[cellID].get_cell_vertices()
18     edgesToRemove = [] # edges of the cell that will be deleted
19     edgesToReplace = [] # edges that will join at the center of the cell
20                          # with the new vertex
21     for vertexID in myVertices:
22         # Goes through edges of each vertex
23         for edgeID in vertices[vertexID].getEdgesBelonging():
24             intersect = np.intersect1d(edges[edgeID].getVerticesBelonging(), myVertices)
25             # If only one vertex of the edge belong to the cell then one
26             # vertex of the edge will be replaced
27             if len(intersect) == 1:
28                 edgesToReplace.append((edgeID, intersect[0]))
29             # If both vertices of the edge belong to the cell then that
30             # edge will be deleted
31             elif len(intersect) == 2:
32                 if edgeID not in edgesToRemove:
33                     edgesToRemove.append(edgeID)
34             else: ## Debug
35                 print("T2 swap, cell with weird number of vertex/edges")
36                 exit()
37     for cellID2 in vertices[vertexID].getCellsBelonging():
38         for vertexID2 in np.intersect1d(cells[cellID2].get_cell_vertices(), myVertices):
39             oldVerticesOrder = cells[cellID2].getVertexOrder(vertexID2)
40             cells[cellID2].remove_vertex(vertexID2)
41             if verticesNumber not in cells[cellID2].get_cell_vertices():
42                 cells[cellID2].add_vertex(vertices[verticesNumber],
43                                           oldVerticesOrder)
44             if cellID2 != cellID:
45                 vertices[verticesNumber].addCellsBelonging(cellID2)
46     del vertices[vertexID]
47     for changedEdge in edgesToReplace:
48         edges[changedEdge[0]].set_vertex_from_ID(vertices[verticesNumber],
49                                                    changedEdge[1])
50     del cells[cellID]
51     for edgeDel in edgesToRemove:
52         del edges[edgeDel]
53     return vertices, edges, cells

```

Listing 5.4: Definiciones de los intercambios T2 del tejido en el archivo *swaps.py*.

5.1.3.3 Intercambios T3

Los intercambios T3 permiten la adhesión de dos o más células en el tejido que se encuentran separadas por medio extracelular. Cuando el tejido crece por divisiones celulares esta transición permite que se rellenen los huecos intercelulares. Esta transición tiene, por lo tanto, gran relevancia por ejemplo, en la cicatrización de heridas. Si un vértice de una célula se encuentra con el lado de otra, entonces el vértice se incorpora en esta última. Existen cuatro clases principales de intercambios T3 (Figura 5.5), dependiendo del número de células a las que pertenezca el vértice que colisiona con la célula y si el vértice se conecta o no con la célula sobre la que colapsa a través de un lado.

Los nuevos vértices creados en los reordenamientos estarán siempre a una distancia de al menos d_{sep} de los vértices existentes. Si un lado es demasiado corto para acomodar el nuevo vértice entonces los vértices en los extremos del lado se separan para hacerle espacio. La inclusión de este proceso de reordenamiento celular permite la simulación de problemas de extremos de tejido libres y móviles (Fletcher *et al.*, 2013). En el próximo capítulo veremos un caso particular de un tejido con extremo libre y móvil: el crecimiento de la médula espinal del axolotl amputada.

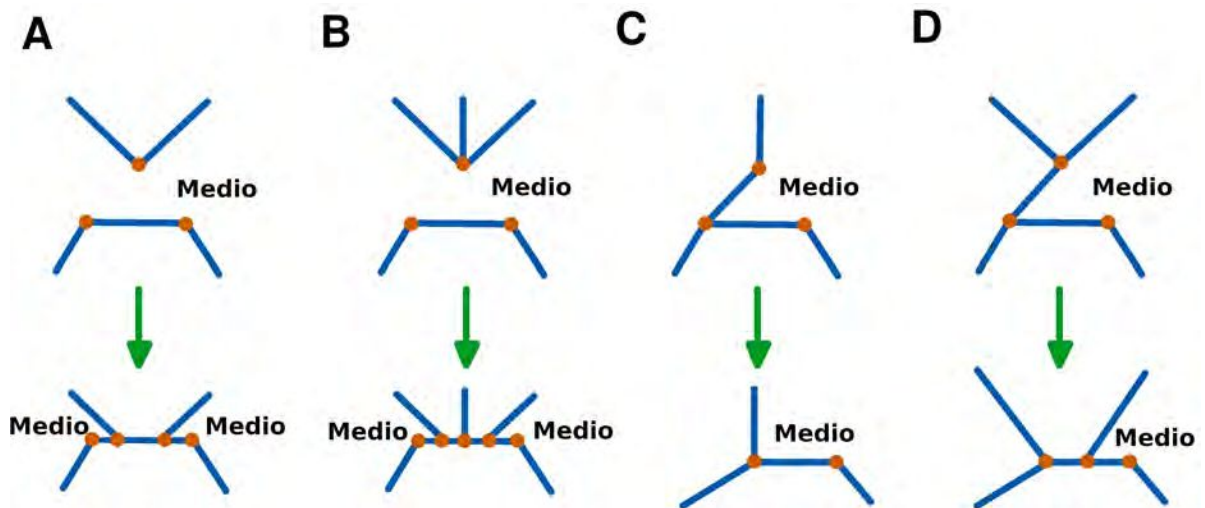


Figura 5.5: Esquema del intercambio T3. Existen cuatro (A-D) casos de intercambios T3 dependiendo de si están conectados el vértice de la célula que colapsa con el lado de la célula en la que colapsa y a cuantas células pertenece el vértice que colapsa. Los vértices se encuentran denotados con color naranja y los lados con color azul. *Void* corresponde al medio extracelular.

A continuación se detalla la implementación de los intercambios T3 en el código de SysVert:

```

1 def swap_T3(configuration, vertexID, intersectingEdgeID,
  intersectingPosition):
2     vertices = configuration.getVertices()
3     edges = configuration.getEdges()
4     cells = configuration.getCells()
5
6     colapsingEdge = edges[intersectingEdgeID]
7     colapsingVertex = vertices[vertexID]
8
9     colapsingVertexEdges = colapsingVertex.getEdgesBelonging() # Edges of
  the colapsing vertex
10    colapsingEdgeVertices = colapsingEdge.getVerticesBelonging() #
  Vertices of the colapsing edge
11
12    edgesList1 = vertices[colapsingEdgeVertices[0]].getEdgesBelonging()
13    edgesList2 = vertices[colapsingEdgeVertices[1]].getEdgesBelonging()
14    colapsingEdgeEdges = set(edgesList1).union(set(edgesList2)) # Edges
  connected to the colapsing edge
15
16    colisionPoint = intersectingPosition
17
18    if len(colapsingVertex.getCellsBelonging()) == 1:
19        if np.intersect1d(list(colapsingEdgeEdges), colapsingVertexEdges).
  size > 0:
20            # Case C
21            vertices, edges, cells = t3caseC(configuration, vertices, edges,
  cells, colisionPoint, colapsingVertex, colapsingEdgeVertices,
  colapsingVertexEdges)
22        else:
23            # Case A
24            vertices, edges, cells = t3caseA(vertices, edges, cells,
  colapsingVertex, colapsingEdge)
25    elif len(colapsingVertex.getCellsBelonging()) >= 2:
26        if np.intersect1d(list(colapsingEdgeEdges), colapsingVertexEdges).
  size > 0:
27            # Case D
28            vertices, edges, cells = t3caseD(vertices, edges, cells,
  colisionPoint, colapsingVertex, colapsingEdge,
  colapsingEdgeVertices, colapsingVertexEdges)
29        else:
30            # Case B
31            vertices, edges, cells = t3caseB(vertices, edges, cells,
  colapsingVertex, colapsingEdge)
32    else:
33        raise Exception('The colapsing vertex in the T3 swap belongs to
  more than 2 cells (not implemented)')
34
35    return vertices, edges, cells
36
37 def t3caseA(vertices, edges, cells, colapsingVertex, colapsingEdge):

```

```

38     vl = colapsingVertex
39     eb = colapsingEdge
40     cb = cells[colapsingEdge.get_cells_belonging(vertices)[0]]
41     verticesBottom = eb.getVerticesBelonging()
42     if cb.getVertexOrder(verticesBottom[0]) > cb.getVertexOrder(
43         verticesBottom[1]):
44         vrr = verticesBottom[0]
45         vll = verticesBottom[1]
46     else:
47         vll = verticesBottom[0]
48         vrr = verticesBottom[1]
49     vlNewX, vlNewY = moveVertexOverEdge(vertices, edges, vll, vrr, eb.getEdgeID
50         ())
51     vl.set_new_x(vlNewX) # sets vl new position
52     vl.set_new_y(vlNewY) # sets vl new position
53     vl.update()
54
55     vlEdges = vl.getEdgesBelonging()
56     verticesTop = []
57     for eachEdge in vlEdges:
58         edgeVertices = edges[eachEdge].getVerticesBelonging()
59         verticesTop.append(edgeVertices[0])
60         verticesTop.append(edgeVertices[1])
61     verticesTop = set(verticesTop)
62     verticesTop.remove(vl.getVertexID())
63     verticesTop = list(verticesTop)
64
65     ct = cells[vl.getCellsBelonging()[0]]
66
67     if ct.getVertexOrder(verticesTop[0]) > ct.getVertexOrder(verticesTop
68         [1]):
69         vtr = verticesTop[0]
70         vtl = verticesTop[1]
71     else:
72         vtl = verticesTop[0]
73         vtr = verticesTop[1]
74
75     if vtr in edges[vlEdges[0]].getVerticesBelonging():
76         er = edges[vlEdges[0]]
77     elif vtr in edges[vlEdges[1]].getVerticesBelonging():
78         er = edges[vlEdges[1]]
79     else:
80         raise Exception('T3 case A edges problem, please report')
81
82     vrNewX, vrNewY = moveVertexOverEdge(vertices, edges, vrr, vll, eb.getEdgeID
83         ())
84     verticesNumber = max(vertices.keys()) + 1
85     vr = vertices[verticesNumber] = Vertex(vrNewX, vrNewY)
86     vertices[verticesNumber].setVertexID(verticesNumber)
87     eb.set_vertex_from_ID(vl, vrr) # Replaces Eb with vc
88     er.set_vertex_from_ID(vr, vl.getVertexID()) # Replaces El with vl

```

```

86 edgesNumber = max(edges.keys()) + 1
87 edges[edgesNumber] = Edge(edgesNumber, vl, vr, eb.getLineConstant(), eb.
    minimumDistance)
88 edgesNumber += 1
89 edges[edgesNumber] = Edge(edgesNumber, vr, vertices[vrr], eb.
    getLineConstant(), eb.minimumDistance)
90
91 vlOrderBot = max([cb.getVertexOrder(vll), cb.getVertexOrder(vrr)])
92 cb.add_vertex(vl, vlOrderBot)
93 cb.add_vertex(vr, cb.getVertexOrder(vl.getVertexID()))
94 vrOrderTop = ct.getVertexOrder(vl.getVertexID())+1
95 ct.add_vertex(vr, vrOrderTop)
96
97 vl.addCellsBelonging(cb.get_cellID())
98 vr.addCellsBelonging(cb.get_cellID())
99 vr.addCellsBelonging(ct.get_cellID())
100
101 return vertices, edges, cells
102
103 def t3caseB(vertices, edges, cells, colapsingVertex, colapsingEdge):
104     vl = colapsingVertex
105     vcX = colapsingVertex.getX()
106     vcY = colapsingVertex.getY()
107     eb = colapsingEdge
108     cb = cells[colapsingEdge.get_cells_belonging(vertices)[0]]
109     verticesBottom = eb.getVerticesBelonging()
110     if cb.getVertexOrder(verticesBottom[0]) > cb.getVertexOrder(
        verticesBottom[1]):
111         vrr = verticesBottom[0]
112         vll = verticesBottom[1]
113     else:
114         vll = verticesBottom[0]
115         vrr = verticesBottom[1]
116     vlNewX, vlNewY = moveVertexOverEdge(vertices, edges, vll, vrr, eb.getEdgeID
        ())
117     vl.set_new_x(vlNewX) # sets vl new position
118     vl.set_new_y(vlNewY) # sets vl new position
119     vl.update()
120
121     vrNewX, vrNewY = moveVertexOverEdge(vertices, edges, vrr, vll, eb.getEdgeID
        ())
122     verticesNumber = max(vertices.keys()) + 1
123     vr = vertices[verticesNumber] = Vertex(vrNewX, vrNewY)
124     vertices[verticesNumber].setVertexID(verticesNumber)
125     eb.set_vertex_from_ID(vl, vrr) # Replaces Eb with vc
126
127     verticesNumber += 1
128     vc = vertices[verticesNumber] = Vertex(vcX, vcY)
129     vertices[verticesNumber].setVertexID(verticesNumber)
130
131     ct1 = cells[vl.getCellsBelonging()[0]]
132     ct2 = cells[vl.getCellsBelonging()[1]]

```



```

133     if ct1.getVertexOrder(vtc) > ct1.getVertexOrder(vl.getVertexID()):
134         ct1 = ct1
135         ctr = ct2
136         vtl = ct1.getVertexOrder(vl.getVertexID()-1)
137         vtr = ct2.getVertexOrder(vl.getVertexID()+1)
138     else:
139         ct1 = ct2
140         ctr = ct1
141         vtl = ct2.getVertexOrder(vl.getVertexID()-1)
142         vtr = ct1.getVertexOrder(vl.getVertexID()+1)
143
144     verticesTop = vl.getEdgesBelonging()
145     for edgeID in verticesTop:
146         if vtr.getVertexID() in edges[edgeID].getVerticesBelonging():
147             er = edges[edgeID]
148         elif vtl.getVertexID() in edges[edgeID].getVerticesBelonging():
149             el = edges[edgeID]
150         else:
151             ec = edges[edgeID]
152             ecVertices = set(ec.getVerticesBelonging())
153             ecVertices.remove(vl.getVertexID())
154             vtc = list(ecVertices)[0]
155
156     er.set__vertex__from__ID(vr, vl.getVertexID()) # Replaces Er with vr
157     ec.set__vertex__from__ID(vc, vl.getVertexID()) # Replaces Er with vr
158
159     edgesNumber = max(edges.keys()) + 1
160     edges[edgesNumber] = Edge(edgesNumber, vl, vc, ec.getLineConstant(), ec.
        minimumDistance)
161     edgesNumber += 1
162     edges[edgesNumber] = Edge(edgesNumber, vc, vr, ec.getLineConstant(), ec.
        minimumDistance)
163     edgesNumber += 1
164     edges[edgesNumber] = Edge(edgesNumber, vr, vertices[vrr], eb.
        getLineConstant(), ec.minimumDistance)
165
166     vllOrderBottom = cb.getVertexOrder(vll)
167     cb.add__vertex__(vl, vllOrderBottom)
168     cb.add__vertex__(vc, vllOrderBottom)
169     cb.add__vertex__(vr, vllOrderBottom)
170     vtcOrderBottom = ct1.getVertexOrder(vtc)
171     ct1.add__vertex__(vc, vtcOrderBottom)
172     vtrOrderBottom = ct1.getVertexOrder(vtr)
173     ctr.add__vertex__(vc, vtrOrderBottom)
174     ctr.add__vertex__(vr, vtrOrderBottom)
175
176     vl.addCellsBelonging(cb.get__cellID__())
177     vc.addCellsBelonging(cb.get__cellID__())
178     vc.addCellsBelonging(ctr.get__cellID__())
179     vc.addCellsBelonging(ct1.get__cellID__())
180     vr.addCellsBelonging(cb.get__cellID__())
181     vr.addCellsBelonging(ctr.get__cellID__())

```

```

182     vr.addCellsBelonging(ctl.get_cellID())
183
184     return vertices, edges, cells
185
186 def t3caseC(configuration, vertices, edges, cells, colisionPoint,
187             colapsingVertex, colapsingEdgeVertices, colapsingVertexEdges):
188     vc = colapsingVertex
189     ct = cells[vc.getCellsBelonging()[0]]
190     colapsingVertexVertices = []
191     for eachEdge in colapsingVertexEdges:
192         for eachVertex in edges[eachEdge].getVerticesBelonging():
193             colapsingVertexVertices.append(eachVertex)
194     vl = vertices[np.intersect1d(colapsingEdgeVertices,
195                                 colapsingVertexVertices)[0]]
196     for edgeID in colapsingVertexEdges:
197         if vl.getVertexID() in edges[edgeID].getVerticesBelonging():
198             ec = edges[edgeID]
199         else:
200             edgeVerticesSet = set(edges[edgeID].getVerticesBelonging()))
201             edgeVerticesSet.remove(vc.getVertexID())
202             vt = vertices[list(edgeVerticesSet)[0]]
203             ec.set_vertex_from_ID(vt, vc.getVertexID()) # Replaces Ec with vt
204             configuration.removeVertex(vc.getVertexID()) # Remove vc from the
205                 lattice
206     vl.set_new_x(colisionPoint[0]) # sets vl new position
207     vl.set_new_y(colisionPoint[1]) # sets vl new position
208     vl.update()
209     ct.remove_vertex(vc.getVertexID())
210     del vc
211     del ec
212
213     return vertices, edges, cells
214
215 def t3caseD(vertices, edges, cells, colisionPoint, colapsingVertex,
216             colapsingEdge, colapsingEdgeVertices, colapsingVertexEdges):
217     vc = colapsingVertex
218     for vertexID in colapsingEdgeVertices:
219         for vcEdgeID in vc.getEdgesBelonging():
220             if vertexID in edges[vcEdgeID].getVerticesBelonging():
221                 vl = vertices[vertexID]
222     for vcEdgeID in vc.getEdgesBelonging():
223         for vcCellID in edges[vcEdgeID].get_cells_belonging(vertices):
224             if vcCellID in vl.getCellsBelonging() and vl.getVertexID() not
225                 in edges[vcEdgeID].getVerticesBelonging():
226                 el = edges[vcEdgeID]
227
228     eb = colapsingEdge
229     colapsingEdgeVerticesCopy = colapsingEdgeVertices[:]
230     colapsingEdgeVerticesCopy.remove(vl.getVertexID())
231     vrID = colapsingEdgeVerticesCopy[0]
232     cb = cells[colapsingEdge.get_cells_belonging(vertices)[0]]

```

```

229     vc.set_new_x(colisionPoint[0]) # sets vl new position
230     vc.set_new_y(colisionPoint[1]) # sets vl new position
231     vc.update()
232     for eachCellID in el.get_cells_belonging(vertices):
233         if vl.getVertexID() in cells[eachCellID].get_cell_vertices():
234             cl = cells[eachCellID]
235         else:
236             ct = cells[eachCellID]
237
238     vlOrder = max([ct.getVertexOrder(el.getVerticesBelonging()[0]), ct.
239                   getVertexOrder(el.getVerticesBelonging()[1])])
239     eb.set_vertex_from_ID(vc, vl.getVertexID()) # Replaces Eb with vc
240     el.set_vertex_from_ID(vl, vc.getVertexID()) # Replaces El with vl
241     vc.addCellsBelonging(cb.get_cellID())
242     vcOrder = max([cb.getVertexOrder(vl.getVertexID()), cb.getVertexOrder(
243                   vrID)])
243     cb.add_vertex(vc, vcOrder)
244     ct.add_vertex(vl, vlOrder)
245     vl.addCellsBelonging(ct.get_cellID())
246     vc.remove_cells_belonging(cl.get_cellID())
247     cl.remove_vertex(vc.getVertexID())
248
249     return vertices, edges, cells

```

Listing 5.5: Definiciones de los intercambios T3 del tejido en el archivo *swaps.py*.

5.1.4 División celular

Un aspecto importante durante el estudio de tejidos en desarrollo o regeneración es la división celular. Diversos modelos pueden ser introducidos sobre división celular. En esta implementación utilizaremos un modelo sencillo en el cual a cada paso se le asigna a cada célula una probabilidad, $P_{divisin}$, de que lleve a cabo una división celular. Se elige al azar una recta que pasa por el centro de la célula. En la intersección entre la misma y los lados de la célula se crean dos vértices. Entre los nuevos vértices se crea un lado que servirá de división entre ambas células. Se puede ver un esquema del mismo en la figura 5.6. La célula hija heredaré las propiedades de la célula madre.

La implementación de la división celular se encuentra en la función *cell_division* que puede encontrarse en el archivo *cell_division_random_angle.py*.

```

1 def cell_division(configuration, cellID, cellTypesLineConstant):
2     cells = configuration.getCells()
3     edges = configuration.getEdges()
4     vertices = configuration.getVertices()
5
6     cellLen = cells[cellID].getVertexNumber()
7     if cellLen > 3: # check if this is necessary for all the
8         cases
9
10        randomAngle1 = random.uniform(0, 2*np.pi)
11        randomAngle2 = randomAngle1 + np.pi

```

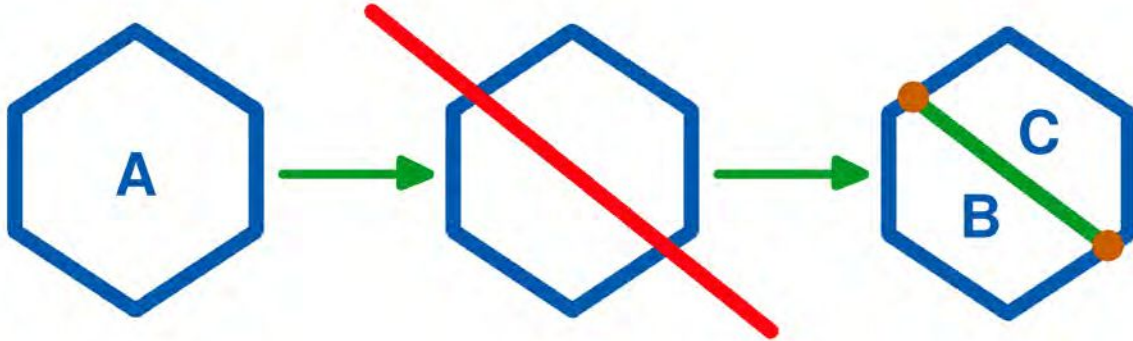


Figura 5.6: Esquema de la implementación de la división celular. La célula A es atravesada por un plano al azar que pasa por el centro de la misma (línea roja). En la intersección entre el plano y los lados de la célula se crean dos vértices y se crea un lado entre ellos. Finalmente se crean las nuevas células (B y C).

```

11
12     if randomAngle2 > 2*np.pi:
13         randomAngle2 = randomAngle2 - 2*np.pi
14
15     # This find the vertices angles previous and next to the
16         randomAngle
17     orderRange1 = find_range(configuration, cellID, randomAngle1)
18     orderRange2 = find_range(configuration, cellID, randomAngle2)
19
20     idOrder1 = (cells[cellID].getVertexFromOrder(orderRange1[0]).
21                 getVertexID(), cells[cellID].getVertexFromOrder(orderRange1[1]).
22                 getVertexID())
23     idOrder2 = (cells[cellID].getVertexFromOrder(orderRange2[0]).
24                 getVertexID(), cells[cellID].getVertexFromOrder(orderRange2[1]).
25                 getVertexID())
26
27     # This find the new vertices position
28     newVertex1pos = intersection(configuration, cellID, randomAngle1,
29                                  idOrder1)
30     newVertex2pos = intersection(configuration, cellID, randomAngle2,
31                                  idOrder2)
32
33     cells, edges, vertices, newOrder1 = vertex_and_edges_rearrangement(

```

```

        cellID , cells , edges , vertices , idOrder1 , newVertex1pos ,
        cellTypesLineConstant)
27 newOrder1ID = cells [ cellID ].getVertexFromOrder(newOrder1).
    getVertexID()
28 cells , edges , vertices , newOrder2 = vertex_and_edges_rearrangement(
    cellID , cells , edges , vertices , idOrder2 , newVertex2pos ,
    cellTypesLineConstant)
29
30 r1 = cells [ cellID ].getVertexOrder(newOrder1ID)
31 vertex1 = cells [ cellID ].getVertexFromOrder(r1)
32 r2 = newOrder2
33 vertex2 = cells [ cellID ].getVertexFromOrder(r2)
34
35 if r1 < r2:
36     newVertices = [ cells [ cellID ].getVertexFromOrder(r1).
37         getVertexID() ]
38     for vertex in range(r1+1,r2):
39         newVertices.append( cells [ cellID ].getVertexFromOrder(r1+1).
40             getVertexID() )
41         vertices [ cells [ cellID ].getVertexFromOrder(r1+1).
42             getVertexID() ].remove_cells_belonging( cellID )
43         cells [ cellID ].remove_vertex( cells [ cellID ].
44             getVertexFromOrder(r1+1).getVertexID() )
45         newVertices.append( cells [ cellID ].getVertexFromOrder(r1+1).
46             getVertexID() )
47     else:
48         newVertices = [ cells [ cellID ].getVertexFromOrder(r2).
49             getVertexID() ]
50         for vertex in range(r2+1,r1):
51             newVertices.append( cells [ cellID ].getVertexFromOrder(r2+1).
52                 getVertexID() )
53             vertices [ cells [ cellID ].getVertexFromOrder(r2+1).
54                 getVertexID() ].remove_cells_belonging( cellID )
55             cells [ cellID ].remove_vertex( cells [ cellID ].
56                 getVertexFromOrder(r2+1).getVertexID() )
57             newVertices.append( cells [ cellID ].getVertexFromOrder(r2+1).
58                 getVertexID() )
59
60 newEdgeID = len(edges)+1
61 while newEdgeID in edges:
62     newEdgeID += 1
63 edges [newEdgeID] = Edge(newEdgeID , vertex1 , vertex2 ,
64     cellTypesLineConstant [str( cells [ cellID ].
65         get_cell_typeID() ) *2] ,
66     minDist=list( edges . values () ) [0] .
67         minimumDistance)
68
69 vertices [ vertex1 .getVertexID() ] .addEdgesBelonging( newEdgeID)
70 vertices [ vertex2 .getVertexID() ] .addEdgesBelonging( newEdgeID)
71
72 newVerticesObj = {}
73 if r1 < r2:
74     for vertex , vertexOrder in zip( newVertices , range(0 , len(

```

```

        newVertices))) :
        newVerticesObj[vertexOrder] = vertices[vertex]
    else:
        for vertex, vertexOrder in zip(newVertices, range(0, len(
            newVertices))) :
            newVerticesObj[vertexOrder] = vertices[vertex]

    newCellID = len(cells)+1
    while newCellID in cells:
        newCellID += 1

    cells[newCellID] = Cell(newVerticesObj, newCellID, cells[cellID].
        cellType)
    cells[newCellID].set_cell_cycle_length(cells[cellID].
        get_cell_cycle_length())

    for eachVertex, vertexOrder in zip(newVertices, range(0, len(
        newVertices))) :
        vertices[eachVertex].addCellsBelonging(newCellID)

    for edgeID, edge in edges.items():
        cellsBelonging = edge.get_cells_belonging(vertices)
        if len(cellsBelonging) == 1:
            edges[edgeID].set_line_constant(cellTypesLineConstant[str(
                cells[cellsBelonging[0]].get_cell_typeID())+'M'])
        else:
            edges[edgeID].set_line_constant(cellTypesLineConstant[str(
                cells[cellsBelonging[0]].get_cell_typeID())+str(cells[
                cellsBelonging[1]].get_cell_typeID())])

    return vertices, edges, cells

```

Listing 5.6: Implementación de la división celular en *cell_division_random_angle.py*.

5.1.5 Condiciones iniciales

Las funciones necesarias para crear la condición inicial se encuentran en el archivo *voronoi.py*, que permite crear un reticulado regular del tipo panal de abejas. Además, el archivo *initial_geometry.py* da la posibilidad de elegir entre una serie de geometrías predefinidas para la distribución de los tipos celulares cuando se define más de uno.

5.1.6 Condiciones de contorno

Un aspecto importante al realizar una simulación computacional es definir las condiciones de contorno que se utilizarán en la misma. En *SysVert* se implementaron condiciones periódicas y absorbentes. La implementación de estas condiciones se realiza en diversos lugares del código. Siempre que se realice un cambio en las posiciones de los vértices se debe chequear si se encuentra o no dentro de la caja para corregir su posición

si fuera necesario. Las funciones esenciales que determinan las dimensiones de la caja entre otras cosas se encuentran en el archivo *boundary_conditions.py*.

5.2 Estructura de SysVert

La estructura del programa tiene un núcleo que consiste en cuatro clases de objetos: *Vertex*, *Edge*, *Cell* y *Lattice*, que son sencillamente abstracciones de los vértices, lados, células del modelo de vértices y de la caja de simulación. Las clases mencionadas se encuentran definidas en *vertex.py*, *edge.py*, *cells.py* y *lattice.py*, respectivamente. En la clase *Lattice* “viven” el resto de los objetos del tejido simulado: vértices, lados y células; al mismo tiempo esta clase contiene métodos que permiten obtener los datos que emergen de la simulación del tejido. A cada paso de tiempo el objeto *lattice* es actualizado registrando los cambios en el resto de los objetos.

5.2.1 Clase Lattice

Los atributos de esta clase son:

- vertices (diccionario): donde cada clave corresponde a un número que identifica al vértice (*vertexID*) y el valor a un objeto vértice.
- edges (diccionario): donde cada clave corresponde a un número que identifica al lado (*edgeID*) y el valor a un objeto lado.
- cells (diccionario): donde cada clave corresponde a un número que identifica a la célula (*cellID*) y el valor a un objeto célula.

```

1 def cell_division(configuration, cellID, cellTypesLineConstant, cellTypes)
2     :
3     cells = configuration.getCells()
4     edges = configuration.getEdges()
5     vertices = configuration.getVertices()
6
7     cellLen = cells[cellID].getVertexNumber()
8     if cellLen > 3: # check if this is necessary for all the
9         cases
10
11         randomAngle1 = np.random.uniform(0, 2*np.pi)
12         randomAngle2 = randomAngle1 + np.pi
13
14         if randomAngle2 > 2*np.pi:
15             randomAngle2 = randomAngle2 - 2*np.pi
16
17         # This find the vertices angles previous and next to the
18         randomAngle
19         orderRange1 = find_range(configuration, cellID, randomAngle1)

```

```

17     orderRange2 = find_range(configuration, cellID, randomAngle2)
18
19     idOrder1 = (cells[cellID].getVertexFromOrder(orderRange1[0]).
20                 getVertexID(), cells[cellID].getVertexFromOrder(orderRange1[1]).
21                 getVertexID())
22     idOrder2 = (cells[cellID].getVertexFromOrder(orderRange2[0]).
23                 getVertexID(), cells[cellID].getVertexFromOrder(orderRange2[1]).
24                 getVertexID())
25
26     # This find the new vertices position
27     newVertex1pos = intersection(configuration, cellID, randomAngle1,
28                                  idOrder1)
29     newVertex2pos = intersection(configuration, cellID, randomAngle2,
30                                  idOrder2)
31
32     cells, edges, vertices, newOrder1 = vertex_and_edges_rearrangement(
33         cellID, cells, edges, vertices, idOrder1, newVertex1pos,
34         cellTypesLineConstant)
35     newOrder1ID = cells[cellID].getVertexFromOrder(newOrder1).
36                 getVertexID()
37     cells, edges, vertices, newOrder2 = vertex_and_edges_rearrangement(
38         cellID, cells, edges, vertices, idOrder2, newVertex2pos,
39         cellTypesLineConstant)
40
41     r1 = cells[cellID].getVertexOrder(newOrder1ID)
42     vertex1 = cells[cellID].getVertexFromOrder(r1)
43     r2 = newOrder2
44     vertex2 = cells[cellID].getVertexFromOrder(r2)
45
46     if r1 < r2:
47         newVertices = [cells[cellID].getVertexFromOrder(r1).
48                         getVertexID()]
49         for vertex in range(r1+1, r2):
50             newVertices.append(cells[cellID].getVertexFromOrder(r1+1).
51                                 getVertexID())
52             vertices[cells[cellID].getVertexFromOrder(r1+1).
53                       getVertexID()].remove_cells_belonging(cellID)
54             cells[cellID].remove_vertex(cells[cellID].
55                                           getVertexFromOrder(r1+1).getVertexID())
56             newVertices.append(cells[cellID].getVertexFromOrder(r1+1).
57                                 getVertexID())
58     else:
59         newVertices = [cells[cellID].getVertexFromOrder(r2).
60                         getVertexID()]
61         for vertex in range(r2+1, r1):
62             newVertices.append(cells[cellID].getVertexFromOrder(r2+1).
63                                 getVertexID())
64             vertices[cells[cellID].getVertexFromOrder(r2+1).
65                       getVertexID()].remove_cells_belonging(cellID)
66             cells[cellID].remove_vertex(cells[cellID].
67                                           getVertexFromOrder(r2+1).getVertexID())
68             newVertices.append(cells[cellID].getVertexFromOrder(r2+1).

```



```

getVertexID()
49
50 newEdgeID = len(edges)+1
51 while newEdgeID in edges:
52     newEdgeID += 1
53
54 edges[newEdgeID] = Edge(newEdgeID, vertex1, vertex2,
55                         cellTypesLineConstant[str(cells[cellID].
56                                                 get_cell_typeID()*2),
57                                                 minDist=list(edges.values())[0].
58                                                 minimumDistance)
59
60 vertices[vertex1.getVertexID()].addEdgesBelonging(newEdgeID)
61 vertices[vertex2.getVertexID()].addEdgesBelonging(newEdgeID)
62
63 newVerticesObj = {}
64 if r1 < r2:
65     for vertex, vertexOrder in zip(newVertices, range(0, len(
66                                     newVertices))):
67         newVerticesObj[vertexOrder] = vertices[vertex]
68 else:
69     for vertex, vertexOrder in zip(newVertices, range(0, len(
70                                     newVertices))):
71         newVerticesObj[vertexOrder] = vertices[vertex]
72
73 newCellID = len(cells)+1
74 while newCellID in cells:
75     newCellID += 1
76
77 cells[newCellID] = Cell(newVerticesObj, newCellID, cells[cellID].
78                         cellType)
79 mCellType = str(cells[cellID].get_cell_typeID())
80 cells[newCellID].set_cell_cycle_length(cellTypes[mCellType]['
81     cellCycleLength'](cellTypes[mCellType]['cellCycleLengthMean'],
82     cellTypes[mCellType]['cellCycleLengthStd']))
83 cells[newCellID].cellCycleReset()
84 cells[cellID].set_cell_cycle_length(cellTypes[mCellType]['
85     cellCycleLength'](cellTypes[mCellType]['cellCycleLengthMean'],
86     cellTypes[mCellType]['cellCycleLengthStd']))
87 cells[cellID].cellCycleReset()
88
89 for eachVertex, vertexOrder in zip(newVertices, range(0, len(
90     newVertices))):
91     vertices[eachVertex].addCellsBelonging(newCellID)
92
93 for edgeID, edge in edges.items():
94     cellsBelonging = edge.get_cells_belonging(vertices)
95     if len(cellsBelonging) == 1:
96         #edges[edgeID].set_line_constant(cellTypesLineConstant[str
97         (cells[cellsBelonging[0]].get_cell_typeID()*2))
98         edges[edgeID].set_line_constant(cellTypesLineConstant[str
99         (cells[cellsBelonging[0]].get_cell_typeID())+'M'])
100     else:

```

```

88         edges[edgeID].set_line_constant(cellTypesLineConstant[str(
            cells[cellsBelonging[0]].get_cell_typeID())+str(cells[
            cellsBelonging[1]].get_cell_typeID())])
89
90     return vertices, edges, cells
91
92 def find_range(configuration, cellID, randomAngle):
93     cells = configuration.getCells()
94     vertices = configuration.getVertices()
95     centerX, centerY = cells[cellID].get_cell_center(configuration)
96     allVertices = cells[cellID].get_cell_vertices()
97
98     anglesDict = {}
99     for eachVertex in allVertices:
100         boxX = configuration.get_box_size('x')
101         boxY = configuration.get_box_size('y')
102         newX = vertices[eachVertex].getX() - centerX
103         newY = vertices[eachVertex].getY() - centerY
104         if abs(newX) > boxX/2:
105             newX = newX - np.sign(newX) * boxX
106         if abs(newY) > boxY/2:
107             newY = newY - np.sign(newY) * boxY
108         vertexAngle = np.arctan2(newY, newX)
109         if vertexAngle < 0:
110             vertexAngle = vertexAngle + 2*np.pi
111         anglesDict[vertexAngle] = eachVertex
112
113     if randomAngle in anglesDict.keys():
114         randomAngle = randomAngle + 0.0001
115     anglesDictSorted = sorted(anglesDict.items())
116     allKeys = []
117     allValues = []
118     for eachKey in anglesDictSorted:
119         allKeys.append(eachKey[0])
120     for eachValue in anglesDictSorted:
121         allValues.append(eachValue[1])
122
123     l = bs.bisect_left(allKeys, randomAngle)
124     r = bs.bisect_right(allKeys, randomAngle)
125     if l == len(allKeys):
126         l = 0
127     if r == 0:
128         r = len(allKeys)
129
130     idMinus = anglesDictSorted[l][1]
131     idPlus = anglesDictSorted[r-1][1]
132
133     return cells[cellID].getVertexOrder(idMinus), cells[cellID].
        getVertexOrder(idPlus)
134
135 def intersection(configuration, cellID, randomAngle, idOrder):
136     boxX = configuration.get_box_size('x')

```

```

137     boxY = configuration.get_box_size('y')
138
139     cells = configuration.getCells()
140     vertices = configuration.getVertices()
141
142     vertex1 = vertices[idOrder[0]]
143     vertex2 = vertices[idOrder[1]]
144
145     newX1 = vertex1.getX()
146     newY1 = vertex1.getY()
147
148     newX2 = vertex2.getX()
149     newY2 = vertex2.getY()
150
151     centerX, centerY = cells[cellID].get_cell_center(configuration)
152
153     if abs(newX1-centerX) > boxX/2:
154         newX1 = newX1 - np.sign(newX1-centerX) * boxX
155     if abs(newY1-centerY) > boxY/2:
156         newY1 = newY1 - np.sign(newY1-centerY) * boxY
157
158     if abs(newX2-centerX) > boxX/2:
159         newX2 = newX2 - np.sign(newX2-centerX) * boxX
160     if abs(newY2-centerY) > boxY/2:
161         newY2 = newY2 - np.sign(newY2-centerY) * boxY
162
163     deltaX = newX2-newX1
164     deltaY = newY2-newY1
165
166     if deltaX != 0:
167         m = (deltaY)/(deltaX)
168         mp = np.tan(randomAngle)
169         if (mp-m) != 0:
170             newX = (newY1 - centerY - m*newX1 + mp*centerX) / (mp-m)
171         else:
172             newX = (newY1 - centerY - m*newX1 + mp*centerX) / (mp-m)
173         newY = newY1 + m*(newX-newX1)
174     else:
175         newX = newX1
176         mp = np.tan(randomAngle)
177         newY = centerY + mp*(newX-centerX)
178
179     if newX < configuration.get_box_limits('left'):
180         newX = newX + boxX
181     elif newX > configuration.get_box_limits('right'):
182         newX = newX - boxX
183     if newY < configuration.get_box_limits('bottom'):
184         newY = newY + boxY
185     elif newY > configuration.get_box_limits('top'):
186         newY = newY - boxY
187
188     return (newX, newY)

```

```

189
190 def vertex_and_edges_rearrangement( cellID , cells , edges , vertices , idRange ,
    newVertexPos , cellTypesLineConstant ):
191     orderRange0 = cells [ cellID ].getVertexOrder( idRange [0])
192     orderRange1 = cells [ cellID ].getVertexOrder( idRange [1])
193     orderRange = ( orderRange0 , orderRange1 )
194     newVertexID = len( vertices )+1
195     while newVertexID in vertices :
196         newVertexID += 1
197     vertices [ newVertexID ] = Vertex( newVertexPos [0] , newVertexPos [1])
198     vertices [ newVertexID ].setVertexID( newVertexID )
199     if ( len( cells [ cellID ].get_cell_vertices())-1,0) == ( orderRange [0] ,
        orderRange [1]) or ( len( cells [ cellID ].get_cell_vertices())-1,0) == (
        orderRange [1] , orderRange [0]) :
200         newOrder = 0
201     else :
202         newOrder = max( orderRange [0] , orderRange [1])
203     cells [ cellID ].add_vertex( vertices [ newVertexID ] , newOrder )
204
205     # Add cells to vertex
206     vertices [ newVertexID ].addCellsBelonging( cellID )
207     toAdd = np.intersect1d( vertices [ idRange [0] ].getCellsBelonging() ,
        vertices [ idRange [1] ].getCellsBelonging() )
208     toAdd = np.delete( toAdd , np.argwhere( toAdd==cellID ) )
209     if len( toAdd ) != 0 :
210         toAdd = int( toAdd )
211         vertices [ newVertexID ].addCellsBelonging( toAdd )
212         toAddOrderRange0 = cells [ toAdd ].getVertexOrder( idRange [0])
213         toAddOrderRange1 = cells [ toAdd ].getVertexOrder( idRange [1])
214         if ( len( cells [ toAdd ].get_cell_vertices())-1,0) == (
            toAddOrderRange0 , toAddOrderRange1 ) or ( len( cells [ toAdd ].
            get_cell_vertices())-1,0) == ( toAddOrderRange1 , toAddOrderRange0
            ):
215             newOrder2 = 0
216         else :
217             newOrder2 = max( toAddOrderRange0 , toAddOrderRange1 )
218         cells [ toAdd ].add_vertex( vertices [ newVertexID ] , newOrder2 )
219
220     if newOrder == len( cells [ cellID ].get_cell_vertices())-1:
221         replacePlus = 0
222     else :
223         replacePlus = newOrder+1
224
225     if newOrder == 0:
226         replaceMinus = len( cells [ cellID ].get_cell_vertices())-1
227     else :
228         replaceMinus = newOrder-1
229
230     newEdgeID = len( edges )+1
231     while newEdgeID in edges :
232         newEdgeID += 1
233     vertex2 = cells [ cellID ].getVertexFromOrder( replacePlus )

```

```

234     edges[newEdgeID] = Edge(newEdgeID, vertices[newVertexID], vertex2,
235                             cellTypesLineConstant[str(cells[cellID].
236                                                     get_cell_typeID())*2],
237                             # minDist=list(edges.items())[0].minDist)
238                             minDist=list(edges.values())[0].
239                             minimumDistance)
240
241     vertices[newVertexID].addEdgesBelonging(newEdgeID)
242     vertices[vertex2.getVertexID()].addEdgesBelonging(newEdgeID)
243
244     edgesList1 = vertices[cells[cellID].getVertexFromOrder(replaceMinus).
245                         getVertexID()].getEdgesBelonging()
246     edgesList2 = vertices[cells[cellID].getVertexFromOrder(replacePlus).
247                         getVertexID()].getEdgesBelonging()
248     edgeID = np.intersect1d(edgesList1, edgesList2)
249
250     edges[int(edgeID)].set_vertex_from_ID(vertices[newVertexID], cells[
251                                         cellID].getVertexFromOrder(replacePlus).getVertexID())
252
253     return cells, edges, vertices, newOrder

```

Listing 5.7: Definición de la clase Lattice en el archivo *lattice.py*.

5.2.2 Clase Cell

Esta clase define métodos y atributos para las células. El constructor de la clase recibe los vértices de la células, el *cellID* y el tipo celular.

Uno de los métodos a destacar de la clase es el *periodic_conditions*, que calcula las imágenes periódicas de los vértices que cruzan los límites de la caja cuando se utilizan condiciones periódicos de contorno en la simulación.

```

1 class Cell:
2     """This class defines methods and attributes for cells.
3         The class constructor receives cell vertices, cell IDs and cell
4         Types,
5         and generates the initial condition for all cells.
6
7         Args:
8             cellVerticesPassed: Dictionary of the cell's vertices. \n
9             cellID: key that identifies each cell. \n
10            cellTypePassed: All cell parameters.
11            """
12     cellVertices = {}
13
14     def __init__(self, cellVerticesPassed, cellID, cellTypePassed,
15                cellCycleLengthPassed=None, cellCycleTimePassed=None):
16         self.cellVertices = cellVerticesPassed
17         self.modified_vertices = None
18
19         self.cellType = cellTypePassed
20         self.set_cellID(cellID)

```

```

19         self.set_cell_type(cellTypePassed)
20
21         self.deterministicDiv = True
22         self.cellCycleLength = cellCycleLengthPassed
23         self.cellCycleTime = cellCycleTimePassed
24
25         self.expression = {}
26         self.expression_color = {}
27
28         for vID, v in self.cellVertices.items():
29             v.addCellsBelonging(self.get_cellID())
30
31     def __del__(self):
32         """
33         Delete vertices from a cell with a given cellID, that has been deleted.
34         """
35         pass
36
37     def periodic_positions(self, configuration):
38         """
39         If in the initial condition then calculates the periodic positions
40
41         Args:
42             configuration (obj)
43         Returns:
44             modified_vertices (dict)
45             """
46             if self.modified_vertices == None:
47                 return self.calculate_periodic_positions(configuration)
48             else:
49                 return self.modified_vertices
50
51     def calculate_periodic_positions(self, configuration):
52         """
53         Calculates periodic images of vertices that cross boundary limits when
54         using Periodic Boundary Conditions in order to recover entire cells.
55
56         Args:
57             configuration (obj)
58         Returns:
59             modified_vertices (dict)
60             """
61             modified_vertices = {}
62             boxX = configuration.get_box_size('x')
63             boxY = configuration.get_box_size('y')
64             arrayX = []
65             arrayY = []
66
67             for vertexID, vertex in self.cellVertices.items():
68                 arrayX.append(vertex.getX())
69                 arrayY.append(vertex.getY())
70             minX = np.min(arrayX)

```

```

70     minY = np.min(arrayY)
71
72     for vertexID, vertex in self.cellVertices.items():
73         newx = vertex.getX()
74         if configuration.get_periodic_conditions('x') and newx-minX >
           boxX/2:
75             newx = vertex.getX()-boxX
76         newy = vertex.getY()
77         if configuration.get_periodic_conditions('y') and newy-minY >
           boxY/2:
78             newy = vertex.getY()-boxY
79         modified_vertices[vertexID] = (newx,newy)
80     self.modified_vertices = modified_vertices
81     return self.modified_vertices
82
83     def getVertexNumber(self):
84         """
85         Get the total number of vertices of the cell.
86
87         Returns:
88             verticesNumber (int)
89         """
90         verticesNumber = len(self.cellVertices)
91         return verticesNumber
92
93
94     def getArea(self, configuration):
95         """
96         Calculates the area of the cell by means of 'Gauss formula'.
97
98         Returns:
99             area (float)
100         """
101         area1 = 0
102         area2 = 0
103         positions = self.periodic_positions(configuration)
104         for k in range(0, self.getVertexNumber()-1):
105             area1 += positions[k][0]*positions[k+1][1]
106             area2 += positions[k+1][0]*positions[k][1]
107         t1 = positions[self.getVertexNumber()-1][0]*positions[0][1]
108         t2 = positions[0][0]*positions[self.getVertexNumber()-1][1]
109         area = 0.5 * abs(area1+t1-area2-t2)
110         return area
111
112     def set_area_target(self, newTargetArea):
113         """
114         Modifies the target area of the cell.
115
116         Args:
117             newTargetArea (float)
118         """
119         self.targetArea = newTargetArea

```

```

120
121 def get_area_sign(self , configuration):
122     """
123     Gives the area sign of the cell (related to order of vertices) for
124         further calculation.
125
126     Returns:
127         -1 for clockwise ordered vertices. \n
128         1 for counterclockwise ordered vertices.
129     """
130     area1 = 0
131     area2 = 0
132     positions = self.periodic_positions(configuration)
133     for k in range(0, self.getVertexNumber() - 1):
134         area1 += positions[k][0] * positions[k+1][1]
135         area2 += positions[k+1][0] * positions[k][1]
136     t1 = positions[self.getVertexNumber() - 1][0] * positions[0][1]
137     t2 = positions[0][0] * positions[self.getVertexNumber() - 1][1]
138     area = 0.5 * (area1 + t1 - area2 - t2)
139     return np.sign(area)
140
141 def getPerimeter(self , configuration):
142     """
143     Calculates the cell perimeter
144
145     Returns:
146         perimeter (float)
147     """
148     perimeter = 0
149     positions = self.periodic_positions(configuration)
150     for k in range(0, self.getVertexNumber() - 1):
151         perimeter += np.sqrt((positions[k+1][0] - positions[k][0])**2 +
152                               (positions[k+1][1] - positions[k][1])**2)
153     perimeter += np.sqrt((positions[0][0] - positions[self.
154         getVertexNumber() - 1][0])**2 + (positions[0][1] - positions[self.
155         getVertexNumber() - 1][1])**2)
156     return perimeter
157
158 def set_perimeter_target(self , newTargetPerimeter):
159     """
160     Modifies the target perimeter of the cell.
161
162     Args:
163         newTargetPerimeter (float)
164     """
165     self.targetPerimeter = newTargetPerimeter
166
167 def get_perimeter_target(self):
168     """
169     Gives the target perimeter of the cell.

```



```

168         Args:
169             targetPerimeter (float)
170         """
171         return self.targetPerimeter
172
173     def getVertexFromOrder(self, vertexCellID):
174         """
175         Given a vertex order in the cell, returns the corresponding vertex
176         object.
177
178         Args:
179             vertexCellID (int)
180
181         Returns:
182             toReturn (obj)
183         """
184         if vertexCellID == -1:
185             # I mean the last one
186             toReturn = self.cellVertices[len(self.cellVertices) - 1]
187         elif vertexCellID == -2:
188             # I mean the last one
189             toReturn = self.cellVertices[len(self.cellVertices) - 2]
190         elif vertexCellID == len(self.cellVertices):
191             toReturn = self.cellVertices[0]
192         elif vertexCellID == len(self.cellVertices)+1:
193             toReturn = self.cellVertices[1]
194         else:
195             toReturn = self.cellVertices[vertexCellID]
196         return toReturn
197
198     def get_cell_edges(self):
199         # Returns a list with all the edges ID of the cell
200         edgesList = set()
201         cellEdges = []
202         for vertexID, vertex in self.cellVertices.items():
203             for edge in vertex.getEdgesBelonging():
204                 if edge in edgesList:
205                     cellEdges.append(edge)
206                 edgesList.add(edge)
207         return cellEdges
208
209     def getVertexOrder(self, vertexID):
210         """
211         Given a vertexID, get position in cellVertices()
212
213         Args:
214             vertexID (int)
215
216         Returns:
217             vertexOrder (int)
218         """

```

```

219         for vertexOrder, vertex in self.cellVertices.items():
220             if vertexID == vertex.getVertexID():
221                 return vertexOrder
222
223     def get_cell_vertices(self):
224         """
225         Gives a list of the vertexIDs in the cell.
226
227         Returns:
228             verticesOut (list)
229         """
230         verticesOut = []
231         for vertexOrder, vertex in sorted(self.cellVertices.items()):
232             verticesOut.append(vertex.getVertexID())
233         return verticesOut
234
235     def cell_vertices_reorder(self, configuration):
236         """
237         Reorder cell vertices when disordered (anti-clockwise).
238
239         Args:
240             configuration (obj)
241         """
242         if self.get_area_sign(configuration) < 0:
243             newCellVerticesOrder = {}
244             for cellOrder, vertex in self.cellVertices.items():
245                 newCellVerticesOrder[self.getVertexNumber()-1-cellOrder] =
                     vertex
246             self.cellVertices = newCellVerticesOrder
247
248     def getTargetArea(self):
249         """
250         Gives the Target area of the cell.
251
252         Returns:
253             targetArea (float)
254         """
255         return self.targetArea
256
257     def getTargetPerimeter(self):
258         """
259         Gives the Target perimeter of the cell.
260
261         Returns:
262             targetPerimeter (float)
263         """
264         return self.targetPerimeter
265
266     def setVertexFromOrder(self, orderPassed, vertexPassed):
267         """
268         Given a vertex, assigns an order to it in the corresponding cell.
269

```

```

270         Args:
271             orderPassed (int) \n
272             vertexPassed (obj)
273         """
274         self.cellVertices[orderPassed] = vertexPassed
275
276     def get_cell_center(self, configuration):
277         """
278         Gives X and Y coordinates of the cell center.
279
280         Returns:
281             X position of cell center (float) \n
282             Y position of cell center (float)
283         """
284         sumX = 0
285         sumY = 0
286         positions = self.modified_vertices
287         vertexNumber = len(positions)
288         for vertexID, vertex in positions.items():
289             sumX += positions[vertexID][0]
290             sumY += positions[vertexID][1]
291         if sumX/vertexNumber < configuration.get_box_limits('left'):
292             newx = sumX/vertexNumber + configuration.get_box_size('x')
293         elif sumX/vertexNumber > configuration.get_box_limits('right'):
294             newx = sumX/vertexNumber - configuration.get_box_size('x')
295         else:
296             newx = sumX/vertexNumber
297         if sumY/vertexNumber < configuration.get_box_limits('bottom'):
298             newy = sumY/vertexNumber + configuration.get_box_size('y')
299         elif sumY/vertexNumber > configuration.get_box_limits('top'):
300             newy = sumY/vertexNumber - configuration.get_box_size('y')
301         else:
302             newy = sumY/vertexNumber
303         return newx, newy
304
305     def set_cellID(self, cellID):
306         """
307         Assigns cellID to a cell.
308
309         Arg:
310             cellID (int)
311         """
312         self.cellID = cellID
313
314     def get_cellID(self):
315         """
316         Gives cellID of a cell.
317
318         Returns:
319             cellID (int)
320         """
321         return self.cellID

```

```

322
323 def add_vertex(self, vertex, vertexOrder):
324     """
325     Adds a vertex to a cell.
326
327     Args:
328         vertex (obj) \n
329         vertexOrder (int)
330     """
331     for i in reversed(range(vertexOrder, len(self.cellVertices))):
332         self.cellVertices[i+1] = self.cellVertices.pop(i)
333     self.cellVertices[vertexOrder] = vertex
334
335 def remove_vertex(self, vertexID):
336     """
337     Remove a vertex with key vertexID from a cell.
338
339     Args:
340         vertexID (int)
341     """
342     popOrder = self.getVertexOrder(vertexID)
343     if popOrder+1 == len(self.cellVertices):
344         self.cellVertices.pop(popOrder)
345     else:
346         for i in range(popOrder, len(self.cellVertices)-1):
347             self.cellVertices[i] = self.cellVertices.pop(i+1)
348
349 def set_cell_type(self, cellTypePassed):
350     """
351     Set the cell type for a given cell.
352
353     Args:
354         cellTypePassed (dict)
355     """
356     self.cellType = cellTypePassed
357     self.set_cellID(self.cellID)
358     self.targetArea = cellTypePassed['targetArea']
359     self.targetPerimeter = cellTypePassed['targetPerimeter']
360     self.typeID = cellTypePassed['typeID']
361     self.areaElasticityConstant = cellTypePassed['
362         areaElasticityConstant']
363     self.perimeterElasticityConstant = cellTypePassed['
364         perimeterElasticityConstant']
365     self.cellHeight = cellTypePassed['cellHeight']
366     self.apoptotic = False
367     self.apoptoticTime = None
368     self.preApoptoticTargetArea = None
369     self.preApoptoticTargetPerimeter = None
370     self.minimumArea = cellTypePassed['minimumArea']
371
372 def cell_cycle_shortening(self):

```

```

372     if 0 <= cycle_position <= g1_reduction:
373         # cell in the G1 skip
374         # G1 skipping part 1 (partial synchronization implementation
            part 1)
375         self.cellCycleTime = cycle_position-cycle_position
376     elif g1_reduction < cycle_position <= g1_length:
377         # cell in the rest of G1
378         # G1 skipping part 2 (partial synchronization implementation
            part 2)
379         self.cellCycleTime = cycle_position-g1_reduction
380     elif g1_length < cycle_position <= g1_length+s_length:
381         # cell in S phase
382         # S mapping (proportional mapping implementation)
383         self.cellCycleTime = int(((cycle_position-g1_length)*((s_length
            -s_reduction)/s_length)+(g1_length-g1_reduction))
384     elif g1_length+s_length < cycle_position <= g1_length+s_length+
            g2m_length+2:
385         # cell in G2/M
386         self.cellCycleTime = cycle_position-g1_reduction-s_reduction
387         self.cellCycleLength = cycle_length-g1_reduction-s_reduction
388
389     def get_cell_typeID(self):
390         """
391         Gives cell typeID for a cell.
392
393         Returns:
394             typeID (int)
395         """
396         return self.typeID
397
398     def get_area_elasticity_constant(self):
399         """
400         Gives area Elasticity Constant for a cell.
401
402         Returns:
403             area Elasticity Constant (float)
404         """
405         return self.areaElasticityConstant
406
407     def get_perimeter_elasticity_constant(self):
408         """
409         Gives perimeter Elasticity Constant for a cell.
410
411         Returns:
412             perimeter Elasticity Constant (float)
413         """
414         return self.perimeterElasticityConstant
415
416     def get_cell_height(self):
417         """
418         Gives the cell height.
419

```

```

420     Returns:
421         cell Height (float)
422     """
423     return self.cellHeight
424
425 def next_vertex(self, vID):
426     """
427     Gives the next vertex from the one with ID vID.
428
429     Args:
430         vID (int)
431     Returns:
432         Next vertex in the cell (obj)
433     """
434     return self.getVertexFromOrder(self.getVertexOrder(vID) + 1)
435
436 def previous_vertex(self, vID):
437     """
438     Gives the previous vertex from the one with ID vID.
439
440     Args:
441         vID (int)
442     Returns:
443         Previous vertex in the cell (obj)
444     """
445     try:
446         return self.getVertexFromOrder(self.getVertexOrder(vID) - 1)
447     except TypeError:
448         print("TypeError: Cell ", self.get_cellID(), " doesn't have
449             vertex", vID )
450         print("it has vertices: ")
451         for id, v in self.cellVertices.items():
452             print("Local ID: ", id, "Global ID: ", v.getVertexID())
453         print("Vertex order:", self.getVertexOrder(vID))
454         exit()
455
456 def periodic_cell(self, configuration):
457     # Check if cell is periodic in x and/or y asking if the edges are
458     periodic
459     cellPeriodicX = False
460     cellPeriodicY = False
461     edges = configuration.getEdges()
462     for edgeID in self.get_cell_edges():
463         periodicX, periodicY = edges[edgeID].periodic_edge(
464             configuration)
465         if periodicX:
466             cellPeriodicX = True
467         if periodicY:
468             cellPeriodicY = True
469     return cellPeriodicX, cellPeriodicY
470
471 def get_cell_cycle_time(self):

```

```

469         """
470         Gives the "position" of the cell in the cell cycle.
471
472         Returns:
473             typeID (int)
474         """
475         return self.cellCycleTime
476
477     def get_cell_cycle_length(self):
478         """
479         Gives the length of the cell cycle.
480
481         Returns:
482             typeID (int)
483         """
484         return self.cellCycleLength
485
486     def set_cell_cycle_time(self, cellCycleTime):
487         self.cellCycleTime = cellCycleTime
488
489     def set_cell_cycle_length(self, cellCycleLength):
490         self.cellCycleLength = cellCycleLength
491
492     def get_cell_typeID(self):
493         """
494         Gives cell typeID for a cell.
495
496         Returns:
497             typeID (int)
498         """
499         return self.typeID
500
501     def cellCycleTimeUpdate(self):
502         self.cellCycleTime += 1
503
504     def cellCycleReset(self):
505         self.cellCycleTime = 0
506
507     def is_apoptotic(self):
508         """
509         Say if a cell is apoptotic or not.
510
511         Returns:
512             True if apoptotic, false if not (bool)
513         """
514         return self.apoptotic
515
516     def set_apoptotic(self):
517         """
518         Set a cell as an apoptotic one if it is not.
519         """
520         if not self.is_apoptotic():

```

```

521         self.apoptotic = True
522         self.apoptoticTime = 0
523         self.preApoptoticTargetArea = self.getTargetArea()
524         self.preApoptoticTargetPerimeter = self.getTargetPerimeter()
525
526
527     def apoptotic_time_foward(self, apoptoticConstant):
528         """
529         Changes cell properties during the cell apoptotic process.
530
531         Args:
532             apoptotic Constant (float)
533         """
534         if self.is_apoptotic():
535             self.apoptoticTime += 1
536             newTargetArea = self.preApoptoticTargetArea*np.exp(-
537                             apoptoticConstant*self.apoptoticTime)
538             newTargetPerimeter = self.preApoptoticTargetPerimeter*np.exp(-
539                                 apoptoticConstant*self.apoptoticTime)
540             self.set_area_target(newTargetArea)
541             self.set_perimeter_target(newTargetPerimeter)

```

Listing 5.8: Definición de la clase Cell en el archivo *cells.py*.

5.2.3 Clase Edge

Esta clase define métodos y atributos para los lados. El constructor de la clase recibe el *edgeID*, ambos vértices que conforman el lado (*vértice1* y *vértice2*), la constante de línea para calcular la contribución del borde a la energía (*lineConstant*) y la distancia mínima a la que se produce la transición topológica T1 (*minDist*). Cuando un lado es creado, los vértices correspondientes se asocian a ese lado.

```

1 class Edge:
2     """This class defines methods and attributes for edges.
3     The class constructor receives the edgeID, both vertices from the
4     edge (vertex1 and vertex2), the Line Constant
5     to calculate the contribution of the edge to the Energy (
6     lineConstant), and the distance at which the T1
7     topological transition occurs (minDist).
8     When an edge is created, the corresponding vertices are associated
9     to that edge.
10
11     Args:
12         edgeID: key that identifies each edge (int).\n
13         vertex1: One of the vertices of the edge (obj).\n
14         vertex2: The other vertex (obj).\n
15         lineConstant: Line Constant to calculate the contribution of
16         the edge to the Hamiltonian (float).\n
17         minDist: The distance at which the T1 topological transition
18         occurs (float).\n
19
20     Raises:

```



```

15         error: If vertex1 is equal to vertex2
16     """
17
18     def __init__(self, edgeID, vertex1, vertex2, lineConstant, minDist):
19         # Establish edge vertices
20         self.vertex1 = vertex1
21         self.vertex2 = vertex2
22         self.lineConstant = lineConstant
23         self.edgeID = edgeID
24         self.minimumDistance = minDist
25         # Create the link between those vertices:
26         self.vertex1.addEdgesBelonging(self.edgeID)
27         self.vertex2.addEdgesBelonging(self.edgeID)
28
29         if self.vertex1 == self.vertex2:
30             log.error("Trying to create an edge with the same two vertices
31                        .")
32             exit()
33
34     def __del__(self):
35         pass
36
37     def getEdgeID(self):
38         """
39         Gives edgeID of an edge.
40
41         Returns:
42             edgeID (int)
43         """
44         return self.edgeID
45
46     def getDistances(self, configuration):
47         """
48         Calculates Distance between vertices (edge length).
49
50         Args:
51             configuration (obj)
52
53         Returns:
54             deltaX (float): X component of the vector between vertex1 and
55                             vertex2.\n
56             deltaY (float): Y component of the vector between vertex1 and
57                             vertex2.\n
58             modulus (float): Distance between vertices (edge length).
59         """
60         deltaX = self.vertex1.getX() - self.vertex2.getX()
61         deltaY = self.vertex1.getY() - self.vertex2.getY()
62         boxX = configuration.get_box_size('x')
63         boxY = configuration.get_box_size('y')
64         if abs(deltaX) > boxX/2:
65             deltaX = deltaX - np.sign(deltaX) * boxX
66         if abs(deltaY) > boxY/2:

```

```

64         deltaY = deltaY - np.sign(deltaY) * boxY
65     modulus = np.sqrt(deltaX**2+deltaY**2)
66     if modulus == 0:
67         print("EdgeID:", self.getEdgeID())
68         print("vertex1:", self.vertex1.getVertexID())
69         print("vertex2:", self.vertex2.getVertexID())
70     return deltaX, deltaY, modulus
71
72     def getVerticesBelonging(self):
73         """
74         Gives the IDs of the vertices of the edge.
75
76         Returns:
77             vertex1.getVertexID() (int) and vertex2.getVertexID() (int)
78         """
79         return [self.vertex1.getVertexID(), self.vertex2.getVertexID()]
80
81     def getLineConstant(self):
82         """
83         Gives the Line Constant.
84
85         Returns:
86             lineConstant (float)
87         """
88         return self.lineConstant
89
90     def getLineX(self):
91         """
92         Gives the X components of both vertices.
93
94         Returns:
95             vertex1.getX() (float) and vertex2.getX() (float)
96         """
97         return (self.vertex1.getX(), self.vertex2.getX())
98
99     def getLineY(self):
100         """
101         Gives the Y components of both vertices.
102
103         Returns:
104             vertex1.getY() (float) and vertex2.getY() (float)
105         """
106         return (self.vertex1.getY(), self.vertex2.getY())
107
108     def set_vertex_from_ID(self, vertexNew, vertexOldID):
109         """
110         Given an ID, replaces the vertex by a new one (used on T-Swaps).
111
112         Args:
113             vertexNew: new vertex to assign (obj).\n
114             vertexOldID: ID of the old vertex (int).
115         """

```

```

116         if self.vertex1.getVertexID() == vertexOldID:
117             self.vertex1.remove_edges_belonging(self.edgeID)
118             self.vertex1 = vertexNew
119             self.vertex1.addEdgesBelonging(self.edgeID)
120         elif self.vertex2.getVertexID() == vertexOldID:
121             self.vertex2.remove_edges_belonging(self.edgeID)
122             self.vertex2 = vertexNew
123             self.vertex2.addEdgesBelonging(self.edgeID)
124
125     def get_cells_belonging(self, vertices):
126         """
127         Gives the IDs of the cells to which the edge belongs.
128
129         Returns:
130             cellsBelonging (list).
131         """
132         edgeVertices = self.getVerticesBelonging()
133         cellsBelonging = np.intersect1d(vertices[edgeVertices[0]].
134                                         getCellsBelonging(), vertices[edgeVertices[1]].getCellsBelonging()
135                                         ())
136         return cellsBelonging
137
138     def set_line_constant(self, lineContantPassed):
139         """
140         Sets a new Line Constant for the edge.
141
142         Args:
143             lineContantPassed (float).
144         """
145         self.lineConstant = lineContantPassed
146
147     def periodic_edge(self, configuration):
148         """
149         Identifies if the edge abandons one side of the simulation box and
150         enters the other side, and in which direction.
151
152         Args:
153             configuration (obj).
154
155         Returns:
156             periodicX (bool) and periodicY (bool)
157         """
158         periodicX = False
159         periodicY = False
160         deltaX = self.vertex1.getX() - self.vertex2.getX()
161         deltaY = self.vertex1.getY() - self.vertex2.getY()
162         modulus = np.sqrt(deltaX**2+deltaY**2)
163         boxX = configuration.get_box_size('x')
164         boxY = configuration.get_box_size('y')
165         if abs(deltaX) > boxX/2 and boxX != 0:
166             periodicX = True
167         if abs(deltaY) > boxY/2 and boxY != 0:

```

```

165         periodicY = True
166     return periodicX, periodicY
167
168     def previous_periodic_edge(self, configuration):
169         """
170         Identifies if the edge abandons one side of the simulation box and
171         enters the other side, and in which direction.
172
173         Args:
174             configuration (obj).
175
176         Returns:
177             periodicX (bool) and periodicY (bool)
178         """
179         periodicX = False
180         periodicY = False
181         deltaX = self.vertex1.getPreviousX() - self.vertex2.getPreviousX()
182         deltaY = self.vertex1.getPreviousY() - self.vertex2.getPreviousY()
183         modulus = np.sqrt(deltaX**2 + deltaY**2)
184         boxX = configuration.get_box_size('x')
185         boxY = configuration.get_box_size('y')
186         if abs(deltaX) > boxX/2 and boxX != 0:
187             periodicX = True
188         if abs(deltaY) > boxY/2 and boxY != 0:
189             periodicY = True
190     return periodicX, periodicY

```

Listing 5.9: Definición de la clase Edge en el archivo *edges.py*.

5.2.3.1 Clase Vertex

Esta clase define métodos y atributos para los vértices. El constructor de clases recibe las componentes x e y de la posición del vértice. Cuando se crea una célula o un lado, la clase correspondiente se asociará automáticamente al vértice.

```

1 class Vertex:
2     """This class defines methods and attributes for vertices.
3     The class constructor receives x and y components of the vertex
4     position. When a cell or an edge is created,
5     the corresponding class will automatically associate them to the
6     vertex, filling cellsBelonging and
7     edgesBelonging lists with the given cellID and edgeID.
8
9     Args:
10         xPosPassed: X component of the vertex.\n
11         yPosPassed: Y component of the vertex.
12     """
13     def __init__(self, xPosPassed, yPosPassed):
14         self.xPos = xPosPassed
15         self.yPos = yPosPassed
16         self.xPreviousPos = xPosPassed

```

```
15         self.yPreviousPos = yPosPassed
16         self.cellsBelonging = [] # list with cellIDs of the cells to which
17                                 # the vertex belongs.
18         self.edgesBelonging = [] # list with edgeIDs of the edges to which
19                                 # the vertex belongs.
20         self.fixed = False
21
22     def __del__(self):
23         pass
24
25     def getVertexID(self):
26         """
27         Gives vertex ID.
28
29         Returns:
30             vertexID (int)
31         """
32         return self.vertexID
33
34     def getX(self):
35         """
36         Gives X component of vertex position.
37
38         Returns:
39             xPos (float)
40         """
41         return self.xPos
42
43     def getY(self):
44         """
45         Gives Y component of vertex position.
46
47         Returns:
48             yPos (float)
49         """
50         return self.yPos
51
52     def getCellsBelonging(self):
53         """
54         Gives the IDs of the cells to which the vertex belongs.
55
56         Returns:
57             cellsBelonging (list)
58         """
59         return self.cellsBelonging
60
61     def getEdgesBelonging(self):
62         """
63         Gives the IDs of the edges to which the vertex belongs.
64
65         Returns:
66             edgesBelonging (list)
```

```

65         """
66         return self.edgesBelonging
67
68     def setVertexID(self, vertexID):
69         """
70         Sets the vertex ID.
71
72         Args:
73             vertexID (int)
74         """
75         self.vertexID = vertexID
76
77     def addCellsBelonging(self, cellID):
78         """
79         Adds a cellID to the CellsBelonging list.
80
81         Args:
82             cellID (int)
83
84         Warns:
85             Gives a warning if the cellID is already in the list.
86         """
87         if cellID in self.cellsBelonging:
88             pass #print("*** WARNING *** Cell ID already in array: ",
89                   cellID, " not added to ", self.getVertexID())
90         else:
91             self.cellsBelonging.append(cellID)
92
93     def remove_cells_belonging(self, cellID):
94         """
95         Removes a cellID from the cellsBelonging list.
96
97         Args:
98             cellID (int)
99         """
100         self.cellsBelonging.remove(cellID)
101
102     def addEdgesBelonging(self, edgeID):
103         """
104         Adds an edgeID to the edgesBelonging list.
105
106         Args:
107             edgeID (int)
108
109         Warns:
110             Gives a warning if the edgeID is already in the list.
111         """
112         if edgeID in self.edgesBelonging:
113             pass #print("*** WARNING *** Edge ID already in array: ",
114                   edgeID, " not added to ", self.getVertexID())
115         else:
116             self.edgesBelonging.append(edgeID)

```

```

115
116     def remove_edges_belonging(self , edgeID):
117         """
118         Removes an edgeID from the edgesBelonging list.
119
120         Args:
121             edgeID (int)
122         """
123         self.edgesBelonging.remove(edgeID)
124
125     def set_new_x(self , xPos):
126         """
127         Sets (updates) the new X component of vertex position.
128
129         Args:
130             xPos (float)
131         """
132         self.xPosNew = xPos
133
134     def set_new_y(self , yPos):
135         """
136         Sets (updates) the new Y component of vertex position.
137
138         Args:
139             yPos (float)
140         """
141         self.yPosNew = yPos
142
143     def update(self):
144         """
145         Sets (updates) the new X and y components of vertex position.
146         """
147         self.xPreviousPos = self.xPos
148         self.yPreviousPos = self.yPos
149         self.xPos = self.xPosNew
150         self.yPos = self.yPosNew
151
152     def getPreviousX(self):
153         """
154         Gives X component of vertex position in the previous step.
155
156         Returns:
157             xPreviousPos (float)
158         """
159         return self.xPreviousPos
160
161     def getPreviousY(self):
162         """
163         Gives Y component of vertex position in the previous step.
164
165         Returns:
166             yPreviousPos (float)

```

```
167         """  
168         return self.yPreviousPos
```

Listing 5.10: Definición de la clase Vertex en el archivo *vertex.py*.

5.3 ¿Cómo se corre y que datos se pueden obtener de una simulación en SysVert?

Para correr una simulación basta con definir los parámetros en el archivo *jwrite.py* y correrlo, creando de esta forma un archivo *model.json* (formato JSON) conteniendo los parámetros de la corrida. Luego llamar a *jread.py* para dar inicio a la simulación. El resultado de las simulaciones podrá encontrarse en la carpeta *simulations*. Dentro de este directorio el usuario se encontrará con una carpeta que responde al nombre que se le ha dado al proyecto en el archivo *jwrite.py* y dentro de ella, con un número de imágenes en formato *.png* a distintos tiempos de simulación (el número de archivos queda determinado por el número total de pasos de simulación y una frecuencia de *plotteo* definidos también en *jwrite.py*). En el directorio del proyecto habrá cuatro archivos “*.log*” que contienen diversos tipos de datos de las células, lados y vértices del tejido, que pueden encontrarse en *cells.log*, *edges.log* y *vertices.log* respectivamente; y datos de distintas propiedades emergentes de la simulación del tejido en el archivo *lattice.log*.

CAPÍTULO 6

Modelado de la regeneración de la médula espinal del axolotl en 2D

Si eres flexible, te mantendrás recto.

—Lao Tse

El modelo de la médula espinal del axolotl en 1D que se presentó previamente tiene la ventaja de ser un modelo sencillo que permitió una parametrización confiable dado el reducido número de parámetros, algunos obtenidos a través de experimentos previos y otros ajustados a uno de los *sets* de datos experimentales de los que disponíamos, a la vez que ofrece un costo computacional muy bajo por las mismas razones. El modelo permitió realizar predicciones interesantes que sirvieron de inspiración para realizar los experimentos con AxFUCCI y así entender como la dinámica de crecimiento del tejido, la médula espinal del axolotl, es conducida por una regulación espacio-temporal del ciclo celular de las células endodermiales. Las fuerzas que experimenta un tejido *in vivo* están íntimamente relacionadas al citoesqueleto, que a su vez, se relaciona con una serie de redes de señalización que regulan el ciclo celular. En esta sección estudiaré como es la distribución y la evolución de las fuerzas en el tejido a través de un nuevo modelo de dos dimensiones implementado en el paquete SysVert buscando posibles indicios sobre el comportamiento y la naturaleza de la señal hipotética.

6.1 Configuración de la simulación del tejido en 2D en SysVert

Los parámetros de corrida en SysVert se definen en el archivo *jwrite.py*. Detallaré en esta sección la parametrización de la simulación de la médula espinal del axolotl en 2D a través del modelo de vertex implementado en SysVert.

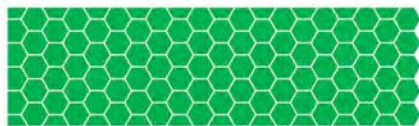
6.1.1 Geometría del tejido simulado

Las células endodermiales que forman el canal central de la médula espinal del axolotl se disponen en un tejido pseudo-estratificado. A raíz de esta disposición modelamos el tejido como un rectángulo con condiciones periódicas de contorno en el eje y para obtener el equivalente a un cilindro. En $-x$ la caja de simulación tiene condiciones absorbentes para capturar el hecho de que el tejido no puede crecer en esa dirección. Finalmente en $+x$ el tejido puede crecer libremente, recapitulando la amputación del tejido.

Las dimensiones del tejido simulado fueron escaladas para reducir los tiempos computacionales de las simulaciones individuales, $1/10$ en el eje y de manera que tendremos 3 células en esta dirección y $1/40$ en el eje x dejándonos 5 células en esta dirección. A la izquierda se encontraría la cabeza del animal (lado anterior) y a la derecha el plano de amputación (lado posterior).

```
1 # Lattice parameters
2 lattice = {
3   'ncellx': round(196/escaladoEspacio), # scaled 1/40
4   'ncelly': round(30/10), # scaled 1/10
5   'chl': cellRadio,
6   'periodicX': False,
7   'periodicY': True,
8   'fixed': {'left': True,
9             'right': False,
10            'bottom': False,
11            'top': False}}
```

A



B



Figura 6.1: Mapeo de la simulación 2D a un cilindro que representa la médula espinal. **(A) Condición inicial.** Todas las células del tejido poseen una longitud de ciclo celular largo (células verdes). **(B) Mapeo de la condición inicial** Mapeo del tejido simulado a un cilindro mediante un software de edición de imágenes graficando como se abstrae la geometría cilíndrica de la médula espinal a nuestro modelo en 2D.

Listing 6.1: Parámetros de la geometría del tejido en el archivo *jwrite.py*.

En la condición inicial entonces, tendremos un tejido confluyente de 3 células en la dirección del eje y (*ncellx*) y 5 células en la dirección del eje x (*ncelly*). La longitud de los lados que formarán los hexágonos regulares representando a la células ependimales están normalizados a 1 (*chl*). Como se menciona más arriba el tejido tendrá condiciones periódicas en el eje y (*periodicY*) y condiciones absorbentes en $-x$

6.1.2 Tipos celulares

En la condición inicial la mayoría de las células serán de tipo no reclutadas (*Non-recruited*), es decir, con un ciclo celular largo (no alcanzadas por la señal). La asignación de las posiciones en el ciclo celular se realizaron de la misma forma que para el modelo 1D presentado en el capítulo 3.

```

1 celltype = {
2     'typeName': 'Non-recruited',
3     'typeID': 0,
4     'targetArea': regularArea,
5     'targetPerimeter': regularPerimeter,
6     'areaElasticityConstant': areaElasticityConstant,
7     'perimeterElasticityConstant': perimeterElasticityConstant,
8     'minimumArea': minimumArea,
9     'cellHeight': cellHeight,
10    'divisionp': 0,
11    'deterministicDiv': True,
12    'cellCycleLength': lognormal,
13    'cellCycleLengthMean': long_cycle_mean,
14    'cellCycleLengthStd': long_cycle_std,
15    'cellCycleTime': tc_distribution,
16    'apoptoticConstant': 0,
17    'apoptoticProbability': 0}
18 cell["0"] = celltype
19
20 celltype = {
21     'typeName': 'Recruited',
22     'typeID': 1,
23     'targetArea': regularArea,
24     'targetPerimeter': regularPerimeter,
25     'areaElasticityConstant': areaElasticityConstant,
26     'perimeterElasticityConstant': perimeterElasticityConstant,
27     'minimumArea': minimumArea,
28     'cellHeight': cellHeight,
29     'divisionp': 0,
30     'deterministicDiv': True,
31     'cellCycleLength': lognormal,
32     'cellCycleLengthMean': short_cycle_mean,
33     'cellCycleLengthStd': short_cycle_std,
34     'cellCycleTime': tc_distribution,

```

```

35     'apoptoticConstant': 0,
36     'apoptoticProbability': 0}
37 cell["1"] = celltype

```

Listing 6.2: Parámetros de los tipos celulares del tejido en el archivo *jwrite.py*.

Luego las células alcanzadas por la señal en nuestra simulación cambiarán su tipo celular (implementación en la clase *Cell*), acortando su ciclo celular siguiendo el mismo modelo de sincronización parcial de la fase G1 y mapeo proporcional de la fase S que se uso en el modelo 1D.

6.1.3 Tensiones de línea

Las tensiones de línea se definen de a pares, en una suerte de matriz a la que llamamos *cellTypesLineConstant* en SysVer: *M* representa al medio, *0* a las células de tipo no recultadas y *1* a las células de tipo recultadas.

Luego se define la distancia mínima (*minDist*) para que sea posible un intercambio de tipo 1 y la constante de tensión de línea (*ctlc*).

```

1 # 'M' stands for Medium
2 cellTypesLineConstant = {'M0': 0, '00': 0,
3                           'M1': 0, '01': 0, '11': 0}
4
5 edges = {
6     'minDist': 0.1,
7     'ctlc': cellTypesLineConstant}

```

Listing 6.3: Tensiones de línea del tejido en el archivo *jwrite.py*.

Todas las tensiones de a pares fueron definidas con un valor nulo como primera aproximación. El fin de esto es evitar la presencia de fuerzas de interacción entre células o con el medio, para que el modelo sea similar al modelo unidimensional pero incorporando a este modelo 2D la capacidad de deformación y cambio de área de las células.

6.1.4 La señal

Aprovechando los resultados de las predicciones de mi modelo 1D, validadas con distintos *sets* de datos de los que disponíamos en primer lugar, y posteriormente confirmados por los primeros experimentos con la técnica FUCCI realizados en axolotl, parametrice la dinámica de distribución espacio-temporal de la señal que fue obtenida a través del ajuste al *switchpoint* experimental de Rost *et al.*, 2016 y descripta por los parámetros λ (*lambda*) (escalado de manera de ser coherente con la geometría escalada) y τ (*tau*).

```

1 # Signal parameters
2 signal = {
3     'lambda': 830/escaladoEspacio,    # scaled
4     'tau': 5100*escaladoTiempo,      # minutes}

```

Listing 6.4: Parámetros de la señal en el archivo *jwrite.py*.

6.2 Resultados de simulación

Las simulaciones se corrieron por un total de 11520 pasos (*nstep*) equivalentes a 8 días y a pasos de *1min* (*dt*). Cada 100 pasos se exportaron datos de la simulación para los vértices, lados, células y propiedades emergentes del tejido (*logVerticesFrequency*, *logEdgesFrequency*, *logCellsFrequency* y *logLatticeFrequency*).

```

1 # Simulations parameters
2 simu = {
3     'nstep': 11520*scaladoTiempo # 8 days in minutes
4     , 'dt': dt
5     , 'checkPointFrequency': 20
6     , 'runFromCheckPoint': False
7     , 'checkPointFile': None
8     , 'plotFrequency': 20
9     , 'folder': 'simulations/' + str(simulation_name)
10    , 'logVerticesFrequency': 20
11    , 'logEdgesFrequency': 20
12    , 'logCellsFrequency': 20
13    , 'logLatticeFrequency': 20
14    , 'timeSplittedLogs': False
15 }
```

Listing 6.5: Parámetros generales de la simulación en el archivo *jwrite.py*.

Parametricé los multiplicadores de Lagrange correspondientes a las fuerzas de área y fuerzas de perímetro de manera que sean lo mas altas posibles manteniendo un tejido coherente, es decir, que no se rompa debido a que las fuerzas eran demasiado grandes. Esta parametrización me permitió tener un punto de partida lo mas cercano posible al modelo de esferas duras para explorar posteriormente otras configuraciones de fuerzas y estudiar como es la respuesta en la dinámica de crecimiento del tejido y en la distribución de los tipos celulares en el tejido.

6.2.1 Modelo de médula espinal sin reclutamiento celular

En primera instancia se realizaron 33 simulaciones del tejido en ausencia de la señal, por lo que no hay reclutamiento en ninguna célula del tejido en todo el tiempo de simulación. En la Figura 6.2 se muestra un ejemplo.

Utilizando las 33 simulaciones generadas a partir de 33 semillas aleatorias se calculo el crecimiento en cada una de ellas tomando el centro de la célula que se encontraba mas a la derecha (lado posterior) del tejido simulado. Los resultados de crecimiento predichos

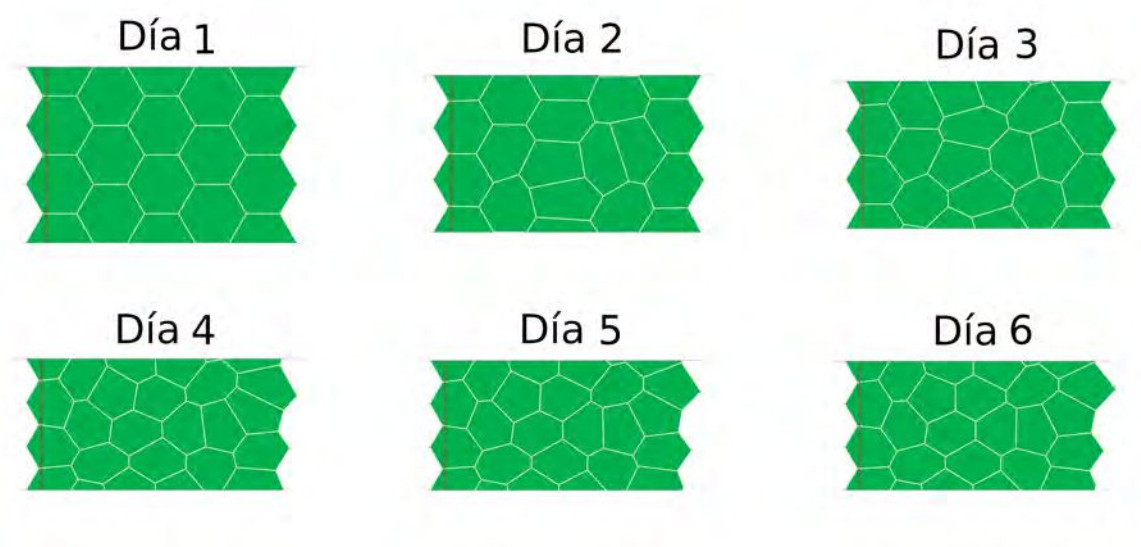


Figura 6.2: Simulación de la médula espinal del axolotl en 2D mediante Sys-Vert sin presencia de la señal. Ejemplo de una simulación sin presencia de la señal y por lo tanto sin reclutamiento. Las células son verdes en todo el tejido y para todo tiempo, es decir, células no reclutadas.

por este modelo se comparan con los datos experimentales de las cinéticas de crecimiento en las que la aceleración del ciclo celular fue evitada al noquear a Sox2 (Figura 6.3).

El modelo es capaz de recapitular los datos experimentales en estas condiciones.

6.2.2 Modelo de médula espinal con reclutamiento celular

En segunda instancia, se incorporó al modelo la señal. Nuevamente se tiene una velocidad de reclutamiento lineal con el tiempo y bajo la acción de esta señal las células reclutadas dividen mas rápidamente. En la Figura 6.4 se muestra un ejemplo.

Utilizando las 33 simulaciones generadas a partir de 33 semillas aleatorias se calculo el crecimiento en cada una de ellas tomando el centro de la célula que se encontraba mas a la derecha (lado posterior) del tejido simulado. La predicción del crecimiento para este modelo se muestra superpuesto a la cinética de crecimiento de la médula espinal en condiciones regenerativas (Figura 6.5 B). Asimismo se grafica el *switchpoint* experimental con el recruitment limit predicho por el modelo (Figura 6.5 A).

El límite de reclutamiento modelado se ajusta con éxito a la curva del *switchpoint* experimental y la predicción del modelo coincide además con la cinética de crecimiento experimental de la médula espinal del axolotl.

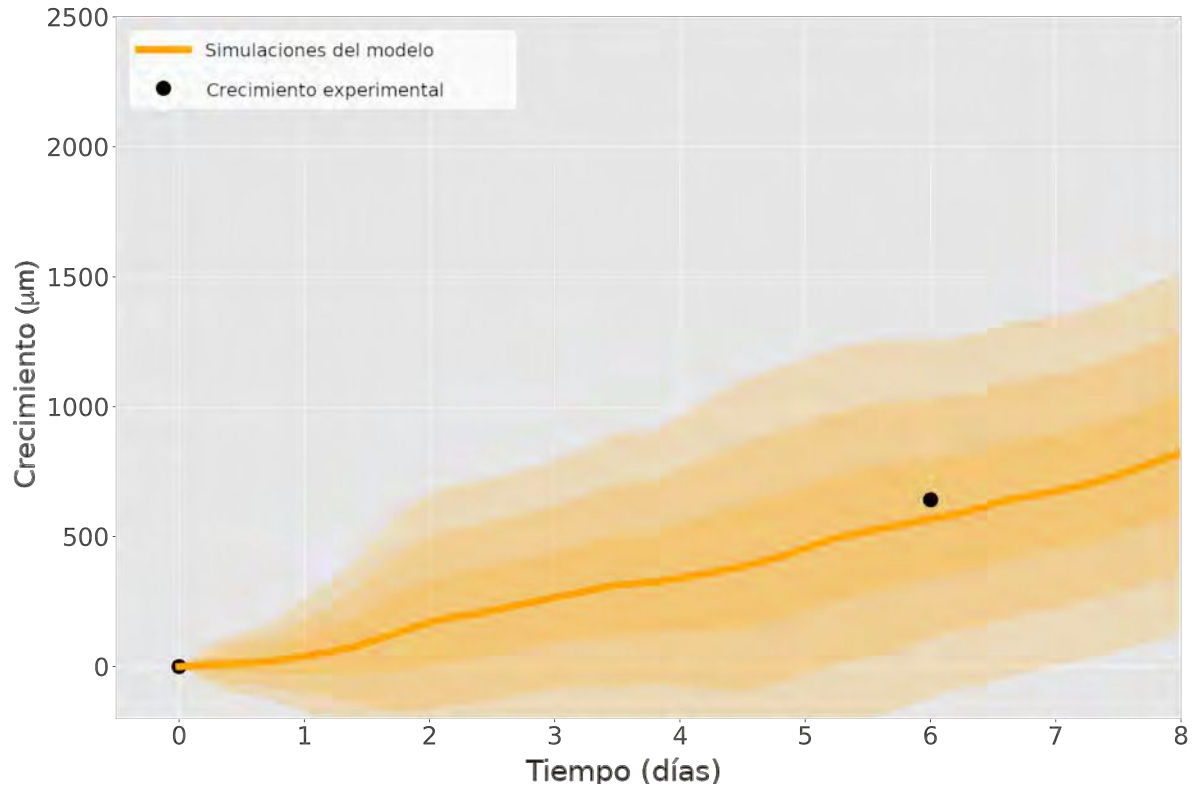


Figura 6.3: El modelo 2D reproduce la reducción del crecimiento experimental cuando se impide la aceleración de la proliferación celular. Predicción del modelo asumiendo que ni las longitudes de la fase S ni la de la fase G1 se acortaron, superpuestas con las cinéticas de crecimiento experimental en las que la aceleración del ciclo celular evitada al noquear a Sox2 (Fei et al., 2014). La línea naranja corresponde a la media, mientras que las áreas sombreadas corresponden a intervalos de confianza de 68, 95 y 99, 7%, de más oscuro a más claro, calculados a partir de 33 simulaciones.

6.2.3 Mecanismo hipotético de interrupción del reclutamiento

A continuación utilicé los dos modelos 2D (con y sin reclutamiento) validados a partir de las predicciones previas para analizar que ocurre con el área de las células a medida que el tejido crece. Para esto se calculó el área promedio en todo el tejido para cada tiempo en ambos modelos (Figura 6.6). Conforme avanza la simulación, y se suceden las divisiones celulares, el tejido sufre compresión, es decir, no alcanzan su área objetivo (A_{kt}). Esta compresión como consecuencia de las divisiones se hace más notable en las simulaciones con reclutamiento que en las que no existe reclutamiento (como es de esperar debido a la diferencia en el número de divisiones). Concretamente

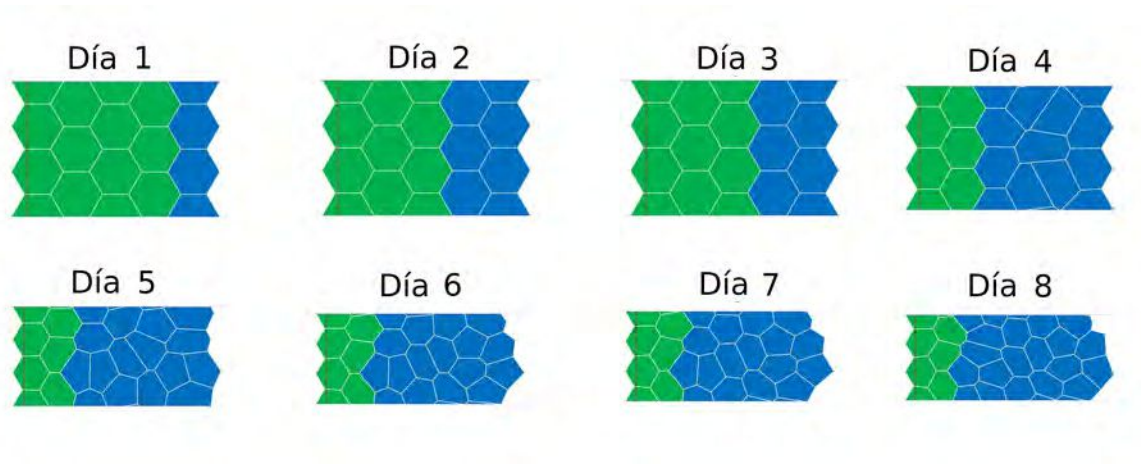


Figura 6.4: Simulación de la médula espinal del axolotl en 2D mediante Sys-Vert en presencia de la señal. Ejemplo de una simulación sin presencia de la señal y por lo tanto con reclutamiento. Las células son verdes en la condición inicial y son reclutadas linealmente con el tiempo a medida que avanza la señal desde el extremo posterior hacia el lado posterior tornándose color azul.

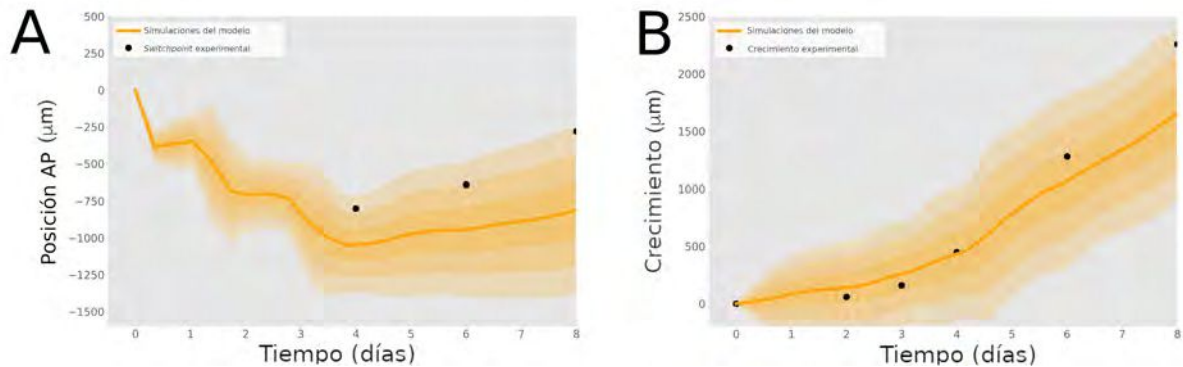


Figura 6.5: El modelo 2D reproduce los datos experimentales de la médula espinal del axolotl en regeneración A) El límite de reclutamiento modelado se ajusta con éxito a la curva de *switchpoint* experimental. B) El modelo coincide cuantitativamente con la cinética de crecimiento experimental de la médula espinal del axolotl (Rost et al., 2016). Las líneas naranjas corresponde a las medias, mientras que las áreas sombreadas corresponden a intervalos de confianza de 68, 95 y 99, 7%, de más oscuro a más claro, calculados a partir de 33 simulaciones.

esta diferencia comienza a ser evidente entre los días 3 y 4. Notablemente el modelo 1D me había arrojado la predicción de que la señal debería extenderse en el tejido hasta las 85 horas post-amputación (3,6 días). Este resultado me permite formular la hipótesis de que la señal podría detenerse como consecuencia de que las células experimenten una compresión en el tejido a medida que avanza el proceso regenerativo.

Para explorar con mayor detalle esta hipótesis se graficó el área promedio de las células en las 33 simulaciones como función del tiempo y el espacio (en bins de $500\mu m$) en la Figura 6.7. Adicionalmente en la figura 6.7B se superpone el *swithpoint* experimental y el límite de reclutamiento promedio en las simulaciones.

En la Figura 6.7A no se ven grandes cambios de área en el tiempo, y donde se observan los cambios no son muy bruscos. Por otra parte en la Figura 6.7B puede observarse que a partir de día 3 comienza a comprimirse levemente el tejido en la región mas anterior y esta compresión aumenta con el tiempo y se mueve posteriormente. Por debajo del límite de reclutamiento (línea naranja) se encuentran las células reclutadas que entre día 3 y 4 comienzan a reducir su área. Esto sugiere que un modelo en el cual el reclutamiento de las células se detenga por acción de la compresión en el tejido podría ser coherente con los resultados de modelado y experimentales.

6.2.4 Cota en la compresibilidad de las células durante el proceso regenerativo

Los resultados obtenidos de las simulaciones anteriores corresponden a una parametrización en la cuál los multiplicadores de Lagrange del modelo de vértices son los más grandes posibles (células lo menos deformables posibles) que hayan permitido modelar un tejido sin que las fuerzas simuladas hagan que el tejido se rompa. Para explorar una condición en la cuál el área de las células fuera todavía más deformable, disminuí los multiplicadores de Lagrange del área y del perímetro en un orden de magnitud respecto de los valores correspondientes a las simulaciones mostradas previamente. Realicé simulaciones con reclutamiento y, como era de esperarse, las células se comprimieron en promedio con el transcurso de la simulación (y por tanto, con el aumento de las proliferaciones acumuladas), llegando a valores por debajo de las $100\mu m^2$ a día 8 (Figura 6.8). Estos valores de área celular están por debajo de lo esperable para los experimentos si se tiene en cuenta que el desvío estándar en el largo de las células a lo largo del eje *AP* es de $0,1\mu m$: propagando el error del área de la célula se obtiene que el 99% de las células debería tener un área entre 107 y $119\mu m^2$, por lo tanto, las simulaciones en las que redujimos los multiplicadores de Lagrange para el área y el perímetro conducen a área celulares predichas significativamente por debajo de las informadas experimentalmente.

La predicción para el crecimiento de esta variante del modelo refleja la compresión del tejido, prediciendo un crecimiento muy inferior a la esperada por los datos experimentales (Figura 6.9).

Esta última variante del modelo permite validar la parametrización de los multiplicadores de Lagrange al indicar una cota inferior sobre los valores que estos pueden

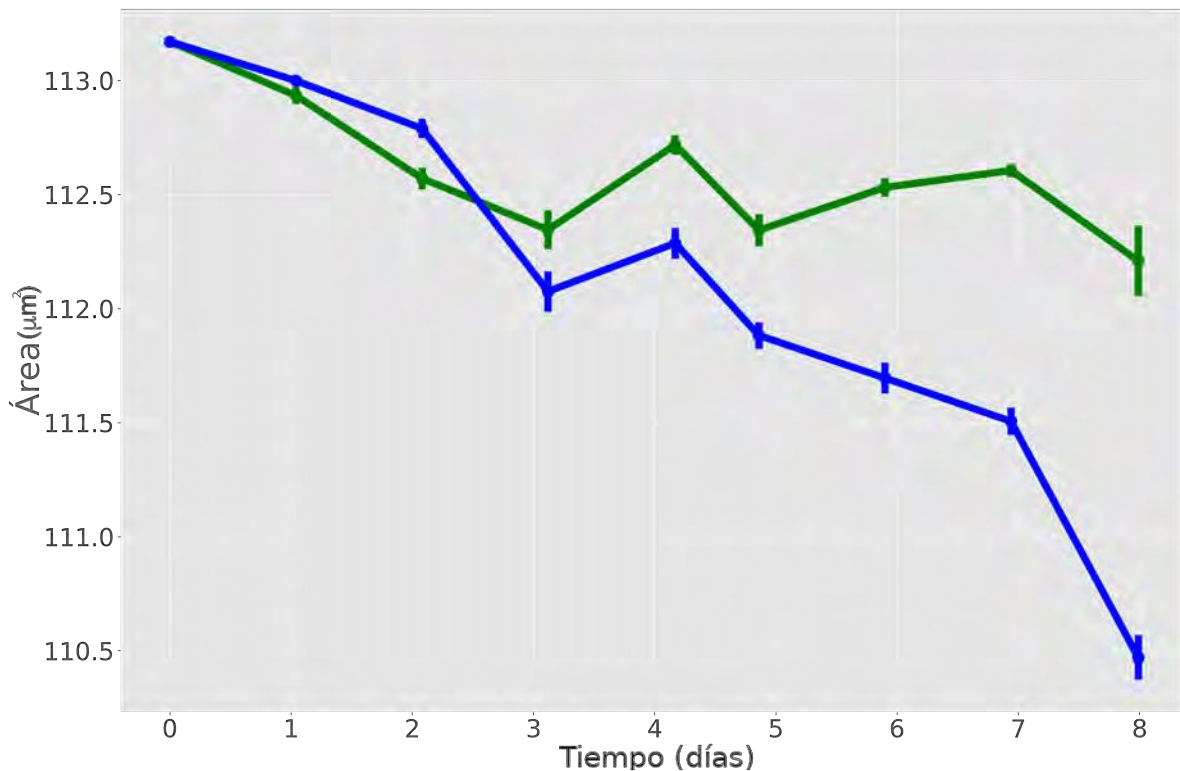


Figura 6.6: Las células en el escenario con reclutamiento sufren una mayor compresión que las células en el escenario sin reclutamiento. Cuando las células son reclutadas la probabilidad de división es mayor y como consecuencia de esto, las células tienen menos espacio para acomodarse (al menos transitoriamente) porque se tiene una mayor población de células. La diferencia en el área entre los dos escenarios comienza a hacerse evidente entre día 3 y 4.

tomar, evitando así una compresión muy alta del tejido que no permita reproducir el crecimiento experimental observado. Es de notar que este resultado, por otra parte, es consistente con las determinaciones experimentales de la densidad celular, que sea informado no cambia demasiado ($2,3 \pm 0,6 \text{ cel}/\mu\text{m}$) en el espacio ni en el tiempo en el tejido en regeneración (Rost *et al.* 2016).

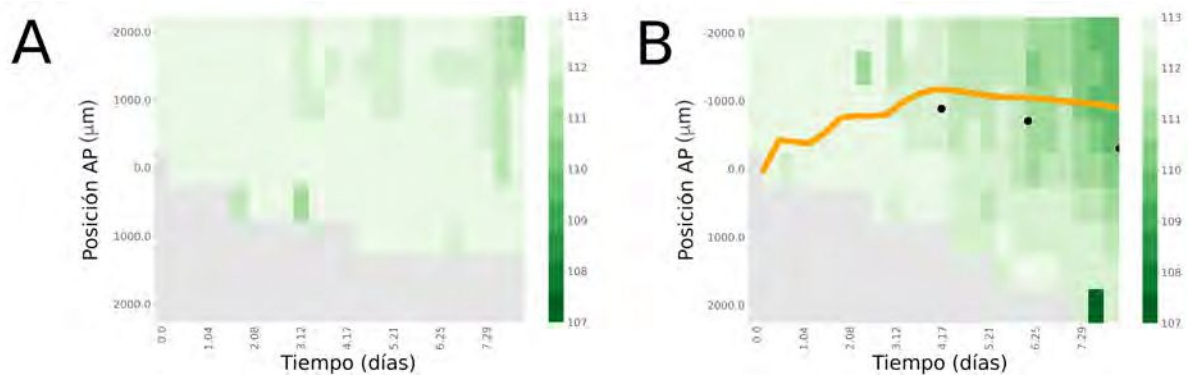


Figura 6.7: Área de las células como función del tiempo y el espacio para los dos escenarios. **A) Escenario sin reclutamiento.** No existen grandes cambios del área de las células en el tiempo ni en el espacio, los cambios observados son relativamente pequeños. **B) Escenario con reclutamiento.** Entre día 3 y 4 se observa una disminución del área de las células que comienza en el lado mas posterior del tejido y aumenta a medida que avanza la simulación (y se tiene mas divisiones) hacia regiones anteriores. El espacio fue *binneado* de a $500\mu m$ y se promedió el área de las células en cada *bin* y en las 33 simulaciones.

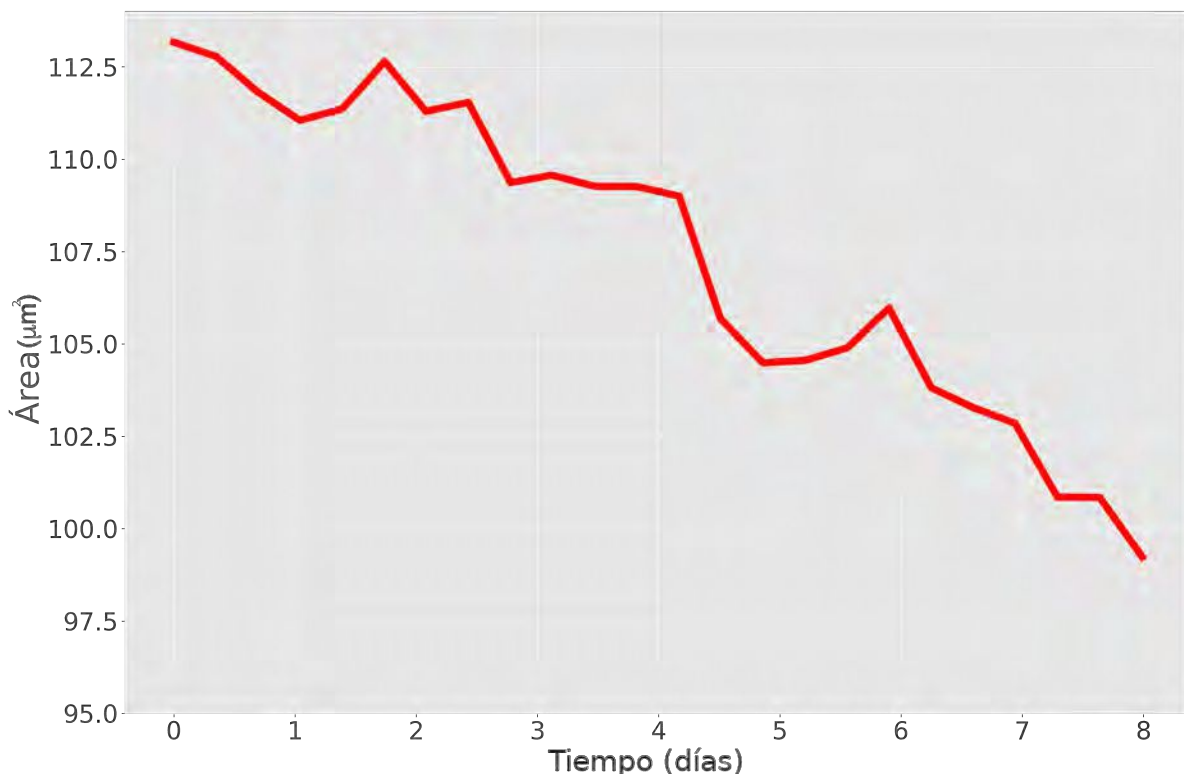


Figura 6.8: Cuando las células se vuelven poco deformables el tejido se comprime por debajo de lo reportado experimentalmente. Según lo reportado por los experimentos el 99% de las células debería tener un área entre 107 y 119 μm^2 . Esto pone una cota en la compresibilidad de las células durante las simulaciones.

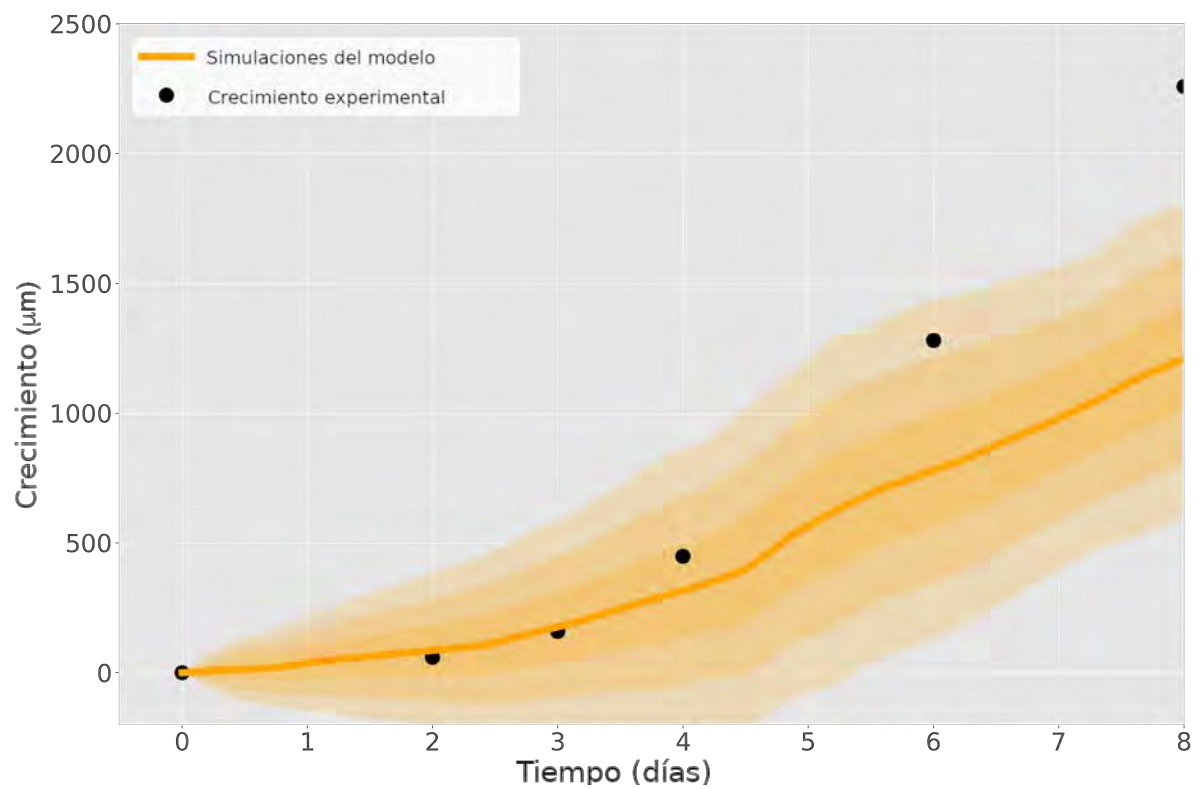


Figura 6.9: Cuando las células se comprimen por debajo del área reportada experimentalmente el modelo no es capaz de reproducir el crecimiento experimental de la médula espinal.

CAPÍTULO 7

Discusión

Fue su quietud lo que me hizo inclinarme fascinado la primera vez que vi a los axolotl. Oscuramente me pareció comprender su voluntad secreta, abolir el espacio y el tiempo con una inmovilidad indiferente.

—Julio Cortázar

La respuesta del tejido a la lesión de la médula espinal difiere mucho entre los vertebrados. En mamíferos, incluidos humanos, las lesiones en la médula espinal a menudo resultan en daños permanentes en los tejidos. En salamandras como el axolotl, sin embargo, la respuesta de las células endodiales está estrechamente orquestada para reconstruir fielmente la médula espinal (Joven y Simon, 2018; Tazaki, Tanaka y Fei, 2017). Después de la amputación de la cola, las células endodiales de la médula espinal del axolotl cambian de divisiones celulares lentas neurogénicas, a más rápidas y proliferativas (Rodrigo Albors *et al.*, 2015). Estos ciclos celulares más rápidos conducen a la expansión de la fuente de células madres endodiales y conduce a un crecimiento regenerativo explosivo. Sin embargo, los mecanismos que regulan la dinámica del ciclo celular durante la regeneración no se comprenden completamente. Aquí, utilizando un enfoque de modelado estrechamente vinculado a datos experimentales, encontramos que el patrón espacio-temporal de proliferación celular en la regeneración de la médula espinal del axolotl es consistente con una señal que se propaga anteriormente $828\mu\text{m}$ desde el sitio de la lesión durante las primeras 85 horas después de la amputación. Aunque, por simplicidad, nos referimos en este manuscrito a una sola señal inducida por la amputación, nuestro modelo podría extenderse naturalmente a la salida combinada de múltiples señales genéticas, químicas y/o biofísicas. Además, mostramos que el acortamiento de la fase *S* es suficiente para explicar el crecimiento explosivo observado durante los primeros días de regeneración, pero que tanto *S* y el acortamiento de *G1* son necesarios para explicar/mantener un mayor crecimiento antes que las primeras neuronas recién nacidas se hagan visibles (Rodrigo Albors *et al.*, 2015).

En comparación con la cantidad de modelos matemáticos diseñados para revelar los fenómenos de formación de patrones durante el desarrollo (Morelli *et al.*, 2012), el modelado en regeneración está todavía en su infancia (Chara *et al.*, 2014). Un ejemplo interesante de modelado aplicado a procesos regenerativos fue el de un sistema de ecuaciones diferenciales ordinarias deterministas que se utilizó magníficamente para desentrañar cómo los factores de señalización secretados podrían usarse para controlar la salida de linajes celulares de múltiples etapas renovando el tejido neural en el epitelio

olfativo de los mamíferos (Lander *et al.*, 2009). Otro modelo matemático basado en ecuaciones diferenciales ordinarias fue concebido para establecer la relación causal entre los procesos celulares cuantificados individualmente para desentrañar la dinámica de las células madre en la médula espinal en desarrollo en pollos y ratones (Kicheva *et al.*, 2014). En un enfoque similar, previamente se modeló la médula espinal del axolotl en regeneración mediante un sistema de ecuaciones diferenciales ordinarias en forma determinista describiendo la cinética del ciclo y los números de células endodiales en reposo que fue mapeado a un modelo de crecimiento de la médula espinal (Rost *et al.*, 2016). Esto nos permitió concluir que mientras la afluencia celular y la reentrada al ciclo celular juegan un papel menor, la aceleración del ciclo celular es el principal impulsor del crecimiento regenerativo de la médula espinal en axolotl (Rost *et al.*, 2016). Un estudio más reciente basado en ecuaciones diferenciales ordinarias y parciales que implican la proliferación celular fue usado para predecir el crecimiento de la médula espinal del pez cuchillo (Ilieş, Sipahi & Zupanc, 2018).

En este estudio, investigamos la distribución espacio-temporal de la proliferación celular durante la regeneración de la médula espinal del axolotl. Para ello, y a diferencia de los artículos mencionados, desarrollamos un modelo basado en células, más general pero preciso, que introduce la dimensión espacial relevante para el problema: el eje *AP*. Para construir un modelo aún más realista, incluimos atributos no deterministas: una distribución exponencial de las coordenadas iniciales a lo largo del ciclo celular y una distribución lognormal de la duración del ciclo celular. En el modelo, una señal acorta el ciclo celular de las células endodiales a lo largo del eje *AP* como consecuencia de acortar sus fases *G1* y *S*, como informamos anteriormente (Rodrigo Albors *et al.*, 2015). La regulación de las fases *G1* y *S* son mecanismos bien conocidos que controlan el destino y la salida de la célula en un número de contextos de desarrollo. En el cerebro, el alargamiento de *G1* da como resultado ciclos celulares más largos en progenitores que experimentan neurogénesis (Lukaszewicz *et al.*, 2005; Calegari *et al.*, 2005; Takahashi, Nowakowski & Caviness, 1995), mientras que el acortamiento experimental de *G1* en progenitores neurales de la corteza cerebral da como resultado divisiones más proliferativas, lo que aumenta la reserva de progenitores y retrasa neurogénesis (Salomoni y Calegari, 2010; Lange, Huttner y Calegari, 2009; Pilaz *et al.*, 2009; Calegari, F. Y Huttner, 2003). Aquí, hemos demostrado que el acortamiento de *G1* durante la regeneración de la médula espinal es necesario para sostener la expansión del conjunto de células endodiales. Juntos, estos hallazgos apuntan a la regulación de la longitud *G1* como mecanismo clave que regula el aumento de divisiones de células madre/neurales progenitoras en el desarrollo y en la regeneración. La longitud de la fase *S* también se regula durante el desarrollo modulando el número de orígenes de replicación de ADN (Nordman & Orr-Weaver, 2012). En mamíferos, el acortamiento de la fase *S* parece desempeñar un papel en la regulación del modo de división celular: los progenitores neuronales en ratón comprometidos con la neurogénesis y los progenitores corticales neurogénicos en el hurón se someten a una fase *S* más corta que sus contrapartes autorrenovables/proliferativas (Turrero García *et al.*, 2016; Arai *et al.*, 2011). En el axolotl, las células endodiales en regeneración acortan la fase *S* durante la fase de expansión/crecimiento. Juntos, estos hallazgos sugieren que la regulación de la fase *S* controla la generación de células en un

contexto de desarrollo y regeneración en lugar de influir en el modo de división celular. El acortamiento combinado de *S* y *G1* en la médula espinal en regeneración sostiene la expansión del conjunto residente de células madre neurales/ependimales a expensas de la neurogénesis. En esta línea, el acortamiento experimental de las fases *G1* y *S* en los progenitores corticales del cerebro de ratón en desarrollo retrasan la aparición de la neurogénesis (Hasenpush-Theil *et al.*, 2018). Nuestros hallazgos se suman a la evidencia de que la regulación del ciclo celular es un mecanismo clave que controla la cantidad y el tipo de células necesarias para generar y regenerar un tejido.

Otra predicción de nuestro modelo es que una señal debe extenderse a unos $800\mu\text{m}$ del lugar de la lesión mientras se reclutan células ependimales 85 horas después de la amputación para explicar el patrón espacio-temporal de proliferación en la médula espinal en regeneración. Para probar esta predicción experimentalmente, adaptamos la tecnología FUCCI a axolotl, lo que nos permitió visualizar la dinámica del ciclo celular *in vivo*. Encontramos un notable acuerdo entre nuestra predicción y el tamaño y momento de aparición de la zona de reclutamiento en la médula espinal AxFUCCI. Nuestra predicción se basó en datos de axolotls de 3cm de hocico-a-cola, mientras que las medidas de AxFUCCI se tomaron de axolotl de $5,5\text{cm}$. Que el tamaño de la zona de reclutamiento sea constante entre estos dos tamaños de animales, podría ser importante para comprender la identidad de la señal inducida por la lesión y cómo se propaga para reclutar células ependimales. Experimentos futuros determinaran si el tamaño de la zona de reclutamiento también permanece constante en axolotls aún más grandes.

Un rasgo característico de nuestro modelo es que el acortamiento de *G1* después de la amputación causa una sincronización parcial de las células ependimales entre sí a medida que pasan por *G1*. En esta tesis exploré dos mecanismos de acortamientos de la fase *G1*, ambos mecanismos dieron lugar a un proceso de sincronización. La sincronización del ciclo celular es difícil de medir en células *in vivo*. Aquí, la propiedad de AxFUCCI de etiquetar puntos de referencia breves y discretos en el ciclo celular (por ejemplo, la transición *G1/S*) nos permitió visualizar una alta sincronía *G1/S* a 1 y 2 días después de amputación *in vivo*. Es de notar que nuestro modelo asume, además, una sincronización parcial en la posición de las células en el ciclo celular en la condición inicial previa a la amputación. Interesantemente, cuando esta sincronización parcial desaparece o cuando es demasiado extrema las predicciones del modelo no reflejan los resultados experimentales, sugiriendo que esta sincronización parcial sería necesaria. Será interesante evaluar si existe sincronización durante la regeneración de otros tejidos en el axolotl (por ejemplo, una extremidad) y en otros organismos regenerativos.

Aunque observamos una excelente coincidencia entre nuestras simulaciones de modelado y los datos de AxFUCCI en términos del tamaño de la zona de reclutamiento en los días 4 y 5 después de la amputación, también encontramos diferencias en puntos de tiempo anteriores. En particular, encontramos que significativamente menos células ependimales en *S/G2* en condiciones basales en el presente estudio en comparación con nuestro estudio anterior (Rost *et al.*, 2016). Los experimentos AxFUCCI informados aquí se llevaron a cabo bajo restricciones operacionales relacionadas al COVID-19, en particular, se redujo la frecuencia de alimentación de los animales. Una reducción de la dieta podría plausiblemente tener impacto en la tasa de proliferación de células epen-

dinales basales y el crecimiento animal. Tampoco podemos excluir un impacto de las condiciones generales de la vivienda, ya que los experimentos anteriores se realizaron en una instalación animal diferente con, por ejemplo, un suministro de agua diferente. Desde el punto de vista del análisis de datos, es importante tener en cuenta que en los experimentos AxFUCCI, las células G0/G1 AxFUCCI se convierten en células AxFUCCI de transición y luego en células S/G2 AxFUCCI. Esto significa que las células de transición podrían ser células en la fase G1 tardía o en la fase S temprana. Las células modeladas, por el contrario, van directamente de la fase G1 a la S. En consecuencia, no podemos equiparar cuantitativamente las células de la “fase de transición” entre experimentos y modelo. Es por eso que los datos AxFUCCI y simulaciones del modelo solo pueden compararse cualitativamente, especialmente en los días 1 y 2, cuando el número de células en transición tiene pico. A pesar de estas consideraciones, encontramos que las proporciones de células ependimales en G0/G1 vs S/G2 AxFUCCI a los 4 y 5 días después de la amputación, es decir, durante el primer ciclo celular regenerativo coincidió de forma precisa y cuantitativa con las simulaciones de nuestro modelo. Además, la tasa de crecimiento de la médula espinal en regeneración fue consistente entre los animales AxFUCCI en este estudio y la medida en los animales de nuestro estudio anterior (Figura 3-figura suplemento 4B). Es decir que los axolotls muestran una respuesta regenerativa notablemente consistente dentro de su primer ciclo celular después de la amputación, posiblemente convergiendo sus respuestas de ciclo celular a pesar de las diferencias en la línea de base. Será fascinante investigar los mecanismos moleculares que permiten esta respuesta regenerativa sea consistente en axolotls de todas las edades/tamaños y disponibilidad nutricional.

Una pregunta importante ahora es si la respuesta espacio-temporal del ciclo celular observada en este estudio concuerda con los eventos de señalización conocidos que operan durante la regeneración de la médula espinal. Una molécula que es candidato fuerte para reclutar células ependimales es la proteína tipo MARCKS de axolotl (AxMLP), un factor secretado involucrado en la respuesta proliferativa durante la regeneración del apéndice del axolotl (Sugiura *et al.*, 2016). AxMLP se expresa normalmente en las células de la médula espinal pero aumenta su expresión después de la amputación de la cola, alcanzando un máximo 12 hasta 24 h después de la amputación y vuelve a los niveles basales un día después (Sugiura *et al.*, 2016). La predicción de nuestro modelo está de acuerdo con el pico de AxMLP seguido de un período descendente de decodificación de señales para instruir cambios celulares intrínsecos que conducen a ciclos celulares más rápidos. Además, la naturaleza de molécula secretada de la proteína AxMLP podría explicar la respuesta proliferativa de largo alcance en la regeneración de la médula espinal. En el futuro, una caracterización del curso temporal más estricto de la localización AxMLP durante la regeneración de la médula espinal del axolotl ayudará a poner a prueba nuestras predicciones.

Los cambios en las propiedades biofísicas de la cola amputada también podrían desencadenar un aumento de la proliferación celular. En los renacuajos de *Xenopus*, la amputación de la cola conduce a la activación del $H^+ V^-$ ATPasa que es necesaria y suficiente para promover la regeneración de la cola (Adams *et al.*, 2007). En el axolotl, la amputación de la cola desencadena cambios en el calcio, el sodio y el potencial de

membrana en el sitio de la lesión (Ozkucur *et al.*, 2010) mientras que la injuria de la médula espinal induce un cambio rápido y dinámico en el potencial de membrana en reposo, potencial que impulsa un programa de expresión génica dependiente de c-Fos promoviendo una respuesta regenerativa (Sabin *et al.*, 2015). La señal que induce la proliferación también podría ser de naturaleza mecánica (Chiou y Collins, 2018). En esta dirección, es interesante que el daño de la médula espinal en el pez cebra induzca una alteración inmediata de las propiedades mecánicas en el sitio de la lesión, lo que vuelve gradualmente a la normalidad (Schlüßler *et al.*, 2018). Interesantemente, las simulaciones preliminares de nuestro modelo de vértices en 2D nos permitió entretener la siguiente hipótesis: independientemente de la naturaleza de la señal que inicia el reclutamiento, el mecanismo de terminación del reclutamiento podría ser de naturaleza mecánica, debido a la compresibilidad de las células en el tejido. En efecto, nuestras simulaciones permitieron determinar cotas para la compresibilidad de las células consistente con la información experimental existente. Nuestras predicciones de la distribución espacio-temporal que podría tener tal señal inductora de proliferación guiará los esfuerzos para deducir los posibles mecanismos responsables de la regeneración exitosa de la médula espinal.

En conjunto, los resultados de esta tesis proporcionan una comprensión mecanística más fina de la distribución espacio-temporal de las células en su ciclo celular que impulsa la regeneración de la médula espinal en el axolotl, y allana el camino en búsqueda de la señal o señales que lanzan la respuesta exitosa de las células ependimales a la amputación. El objetivo por delante será explorar el espacio paramétrico del modelo 2D y realizar nuevas predicciones acerca de las fuerzas que gobiernan la regeneración de la médula espinal del axolotl que puedan ser sopesadas con los resultados experimentales de Fucci y otros experimentos futuros en condiciones de homeostasis, al momento de la amputación del tejido y durante toda la regeneración. Es perspectiva de esta tesis establecer un dialogo entre las predicciones hipotéticas de nuestros modelos con diseños experimentales que han surgido en los últimos años y que permiten medir fuerzas intercelulares e intracelulares que involucran al citoesqueleto. Es de esperar que este camino nos acerque aún más a entender los mecanismos que hacen que el axolotl exhiba las maravillosas propiedades regenerativas que le han dado su fama en el ámbito académico, y que han alimentado, también, la cultura mexicana y particularmente a la mitología azteca, relacionándolo al dios *Xólotl*, hermano de *Quetzalcóatl*, asociado a la idea de vida y movimiento.

Bibliografía

- A.Z. Zaky and M.Z. Moftah. 2014. *Neurogenesis and growth factors expression after complete spinal cord transection in Pleurodeles waltl*. Front. Cell. Neurosci., 8 , p. 458
- Abe T, Sakaue-Sawano A, Kiyonari H, Shioi G, Inoue K, Horiuchi T, Nakao K, Miyawaki A, Aizawa S & Fuji-mori T. 2013. *Visualization of cell cycle in mouse embryos with Fucci2 reporter directed by Rosa26 promoter*. Development. 140:237–246.1024
- Adams DS, Masi A & Levin M. 2007. *H+ Pump-dependent changes in membrane voltage are an early mechanism necessary and sufficient to induce Xenopus tail regeneration*. Development. 34(7): 1323-35.
- Alexander G. Fletcher, James M. Osborne, Philip K. Maini, David J. Gavaghan. 2013. *Implementing vertex dynamics models of cell populations in biology within a consistent computational framework*.
- Arai Y, Pulvers JN, Haffner C, Schilling B, Nüsslein I, Calegari F, Huttner WB. 2011. *Neural stem and progenitor cells shorten S-phase on commitment to neuron production*. Nat Commun. 2:154.
- Banfi A, Muraglia A, Dozin B, Mastrogiacomini M, Cancedda R, Quarto R. 2000. *Proliferation kinetics and differentiation potential of ex vivo expanded human bone marrow stromal cells: Implications for their use in cell therapy*. Exp Hematol 28:707–715.
- Becker CG, Becker T, Hugnot JP. 2018. *The spinal ependymal zone as a source of endogenous repair cells across vertebrates*. Prog Neurobiol. 170: 67-80.
- Bouldin CM, Snelson CD, Farr GH 3rd & Kimelman D. 2014. *Restricted expression of cdc25a in the tailbud is essential for formation of the zebrafish posterior body*. Genes Dev. 28:384–395.
- C.M. O'Hara and E.A. Chernoff. 1994. *Growth factor modulation of injury-reactive ependymal cell proliferation and migration*. Tissue Cell, 26, pp. 599-611
- C.M. O'Hara, M.W. Egar and E.A. Chernoff. 1992. *Reorganization of the ependyma during axolotl spinal cord regeneration: changes in intermediate filament and fibronectin*

expression. Dev. Dyn., 193, pp. 103-115

Calegari F & Huttner WB. 2003. *An inhibition of cyclin-dependent kinases that lengthens, but does not arrest, neuroepithelial cell cycle induces premature neurogenesis*. J Cell Sci. 116: 4947–4955.

Calegari F, Haubensak W, Haffner C, Huttner WB. 2005. *Selective lengthening of the cell cycle in the neurogenic subpopulation of neural progenitor cells during mouse brain development*. J Neurosci. 103625(28): 6533-8.

Chara O, Tanaka EM & Brusch L. 2014. *Mathematical modeling of regenerative processes*. Curr Top Dev Biol. 108: 283-317.1039

Chernoff EA, Stocum DL, Nye HL, Cameron JA. 2003. *Urodele spinal cord regeneration and related processes*. Dev Dyn. 226(2): 295-307.

Chernoff EA. 1996. *Spinal cord regeneration: a phenomenon unique to urodeles*. Int J Dev Biol 40:823– 831.

Chiou K & Collins ES. 2018. *Why we need mechanics to understand animal regeneration*. Dev Biol. 433(2): 155-165.

Clarke JDW, Ferretti P. 1998. *CNS regeneration in lower vertebrates*. In: Ferretti P, Géraudie J, editors. Cellular and molecular basis of regeneration: from invertebrates to humans. Chichester, England: John Wiley and Sons, Ltd. p 255–269.

Csilléry K, Blum MGB, Gaggiotti OE & François O. 2010. *Approximate Bayesian Computation (ABC) in practice*. Trends in Ecology & Evolution. 25 (7): 410-418.1045

Cura Costa E, Otsuki L, Rodrigo Albors A, Tanaka EM & Chara O. 2021. *Spatiotemporal control of cell cycle acceleration during axolotl spinal cord regeneration* -Supplementary notebooks –v2.0. Zenodo.

Duerr T, Jeon EK, Wells KM, Villanueva A, Seifert AW, McCusker CD & Monaghan JR. 2021. *A constitutively expressed fluorescence ubiquitin cell cycle indicator (FUCCI) in axolotls for studying tissue regeneration*. biorxiv. <https://doi.org/10.1101/2021.03.30.437716>.

Fei JF, Knapp D, Schuez M, Murawala P, Zou Y, Singh SP, Drechsel D & Tanaka EM. 2016. *Tissue-and time-directed electroporation of CAS9 protein-gRNA complexes in vivo yields efficient multigene knockout for studying gene function in regeneration*. npj Regen Med.1(1): 2.1057

Fei JF, Schuez M, Tazaki A, Taniguchi Y, Roensch K & Tanaka EM. 2014. *CRISPR-mediated genomic deletion of Sox2 in the axolotl shows a requirement in spinal cord*

neural stem cell amplification during tail regeneration. Stem Cell Reports. 3(3): 444-59.105438

Fletcher A.G., Osborne J.M., Maini P.K., Gavaghan D.J. 2013. *Implementing vertex dynamics models of cell populations in biology within a consistent computational framework*. Progress in Biophysics and Molecular Biology 113 299-326.

FreitasPD, Yandulskaya AS & Monaghan JR. 2019. *Spinal Cord Regeneration in Amphibians: A Historical Perspective*. Dev Neurobiol. 79(5): 437-452.

G.H. Fahmy and M.Z. Moftah Front. 2010. *Fgf-2 in astroglial cells during vertebrate spinal cord recovery*. Cell. Neurosci., 4, p. 129

Gierer A, Meinhardt H. 1972. *A theory of biological pattern formation*. Kybernetik. 12(1):30-9.

Glazier J.A., Graner F. 1992. *Simulation of biological cell sorting using a two-dimensional extended potts model*. Physical Review Letters, Volume 69, N 13.

Glazier J.A., Graner F. 1993. *Simulation of the differential adhesion driven rearrangement of biological cells*. Physical Review E, Volume 47, N 3.

Harris CR, Jarrod Millman K, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ, Kern R, Picus M, Hoyer S, van Kerkwijk MH, Brett M, Haldane A, Fernández del Río J, Wiebe M, Peterson P, Gérard-Marchant P, Sheppard K, Reddy T, Weckesser W, Abbasi H, Gohlke C & Oliphant TE. 2020. *Array programming with NumPy*, Nature, 585, 357–362. DOI:10.1038/s41586-020-10662649-2

Hasenpusch-Theil K, West S, Kelman A, Kozic Z, Horrocks S, McMahon AP, Price DJ, Mason JO, Theil T. 2018. *Gli3 controls the onset of cortical neurogenesis by regulating the radial glial cell cycle through Cdk6 expression*. Development. 145(17).

Hodgkin, A.L.; Huxley, A.F. 1952. *Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo*. J. of Physiology, 116: 449-472.

Huibregtse BA, Johnstone B, Goldberg VM, Caplan AI. 2000. *Effect of age and sampling site on the chondro-osteogenic potential of rabbit marrow-derived mesenchymal progenitor cells*. J Orthop Res 18:18 –24.

Hunter JD. 2007. *Matplotlib: A 2D graphics environment*. Computing in Science & Engineering. 9:90–95.

Ilieş I, Sipahi R & Zupanc GKH. 2018. *Growth of adult spinal cord in knifefish: development and parametrization of a distributed model*. Journal of Theoretical Biology.

437: 101–114.

Joven A & Simon A. 2018. *Homeostatic and regenerative neurogenesis in salamanders*. Prog Neurobiol. 170: 81-98.

K. Hunter, M. Maden, D. Summerbell, U. Eriksson and N. Holder. 1991. *Retinoic acid stimulates neurite outgrowth in the amphibian spinal cord*. Proc. Natl. Acad. Sci. U. S. A., 88, pp. 3666-3670

Kicheva A, Bollenbach T, Ribeiro A, Valle HP, Lovell-Badge R, Episkopou V & Briscoe J. 2014. *Coordination of progenitor specification and growth in mouse and chick spinal cord*. Science 345: 1254927.

Klinger E, Rickert D, Hasenauer J. 2018. *pyABC: distributed, likelihood-free inference*. Bioinformatics. 34(20): 3591-3593.

Lander AD, Gokoffski KK, Wan FY, Nie Q & Calof AL. 2009. *Cell lineages and the logic of proliferative control*. PLoS Biol. 7(1): e15.

Landsberg K.P., Farhadifar R., Ranft J., Umetsu D., Widmann T.J., Bittig T., Said A., Jülicher F, Dahmann C. 2009. *Increased Cell Bond Tension Governs Cell Sorting at the Drosophila Anteroposterior Compartment Boundary*. Current Biology 19, 1–6.

Lange C, Huttner WB & Calegari F. 2009. *Cdk4/cyclinD1 overexpression in neural stem cells shortens G1, delays neurogenesis, and promotes the generation and expansion of basal progenitors*. Cell Stem Cell. 10805(3): 320-31.

Lukaszewicz A, Savatier P, Cortay V, Giroud P, Huissoud C, Berland M, Kennedy H & Dehay C. 2005. *G1 phase regulation, area-specific cell cycle control, and cytoarchitectonics in the primate cortex*. Neuron 108347: 353–364.

M. Moftah, M. Landry, F. Nagy and J.M. Cabelguen. 2008. *Fibroblast growth factor-2 mRNA expression in the brainstem and spinal cord of normal and chronic spinally transected urodeles*. J. Neurosci. Res., 86 (2008), pp. 3348-3358

Magno R., Grieneisen V.G., Marée A.F.M. 2015. *The biophysical nature of cells: potential cell behaviours revealed by analytical and computational studies of cell surface mechanics*. BMC Biophysics DOI 10.1186/s13628-015-0022-x.

Mathews J.H., Kurtis D.F. 2005. *Métodos Numéricos con MATLAB*. Pearson Prentice Hall.

McKinney W. 2010. *Data structures for statistical computing in Python*. Proceedings of the 9th Python in Science Conference: 51-56.

- Meinhardt H. 1982. *Models of Biological Pattern Formation*. Academic Press, London.
- Morelli LG, Uriu K, Ares S & Oates AC. 2012. *Computational approaches to developmental patterning*. Science. 336(6078): 187-91.
- Murray, J. D. 2003. *Mathematical Biology I: An Introduction*. New York: Springer.
- Nagai T. & Honda H.. 2001. *A dynamic cell model for the formation of epithelial tissues*. Philosophical Magazine B, vol. 81, No. 7, 699-719.
- Nordman J & Orr-Weaver TL. 2012. *Regulation of DNA replication during development*. Development 139: 455-464.109039
- Oliphant TE. 2006. A guide to NumPy, USA: Trelgol Publishing.1091Ozkucur N, Epperlein HH & Funk RH. 2010. *Ion imaging during axolotl tail regeneration in vivo*. Dev Dyn. 1092239(7): 2048-57.
- Pende M, Vadiwala K, Schmidbaur H, Stockinger AW, Murawala P, Saghafi S, Dekens MPS, Becker K, Revilla-i-Domingo R, Papadopoulos S-C, Zurl M, Pasierbek P, Simakov O, Tanaka EM, Raible F & Dodt HU. 2020. *A versatile depigmentation, clearing, and labeling method for exploring nervous system diversity*. Sci Adv. 6: eaba0365.
- Pilaz LJ, Patti D, Marcy G, Ollier E, Pfister S, Douglas RJ, Betizeau M, Gautier E, Cortay V, Doerflinger N, Kennedy H & Dehay C. 2009. *Forced G1-phase reduction alters mode of division, neuron number, and laminar phenotype in the cerebral cortex*. Proc Natl Acad Sci U S A. 106(51): 21924-9.
- Progress in Biophysics and Molecular Biology, Volume 113, Issue 2, Pages 299-326, ISSN 0079-6107, doi.org/10.1016/j.pbiomolbio.2013.09.003.
- Rashevsky N. 1938. *Mathematical biophysics: physico-mathematical foundations of biology*. Chicag: University of Chicago Press.
- Rodrigo Albors A, Tazaki A, Rost F, Nowoshilow S, Chara O & Tanaka EM. 2015. *Planar cell polarity-mediated induction of neural stem cell expansion during axolotl spinal cord regeneration*. eLife. 4:e 10230.
- Rost F, Rodrigo Albors A, Mazurov V, Brusch L, Deutsch A, Tanaka EM & Chara O. 2016. *Accelerated cell divisions drive the outgrowth of the regenerating spinal cord in axolotls*. eLife. 5. pii: e20357.
- Roy S, Gardiner DM & Bryant SV. 2000. *Vaccinia as a tool for functional analysis in regenerating limbs: ectopic expression of Shh*. Dev Biol. 218: 199-205.

Sabin K, Santos-Ferreira T, Essig J, Rudasill S & Echeverri K. 2015. *Dynamic membrane depolarization is an early regulator of ependymogial cell response to spinal cord injury in axolotl*. Dev Biol. 408(1): 14-25.

Sakaue-Sawano A, Kurokawa H, Morimura T, Hanyu A, Hama H, Osawa H, Kashiwagi S, Fukami K, Miyata T, Miyoshi H, Imamura T, Ogawa M, Masai H & Miyawaki A. 2008. *Visualizing spatiotemporal dynamics of multicellular cell-cycle progression*. Cell. 132:487–498.

Salomoni P & Calegari F. 2010. *Cell cycle control of mammalian neural stem cells: putting a speed limit on G1*. Trends Cell Biol. 20: 233–243.1114

Schilling S., Willecke M., Aegerter-Wilmsen T., Cirpka O.A., Basler K., Von Mering C. 2011. *Cell-Sorting at the A/P Boundary in the Drosophila Wing Primordium: A Computational Model to Consolidate Observed Non-Local Effects of Hh Signaling*. PLoS Computational Biology, Volume 7, Issue 4, e1002025.

Schindelin J, Arganda-Carreras I, Frise E, Kaynig V, Longair M, Pietzsch T, Preibisch S, Rueden C, Saalfeld S, Schmid B, Tinevez J-Y, White DJ, Hartenstein V, Eliceiri K, Tomancak P & Cardona A. 2012. *Fiji: an open-source platform for biological-image analysis*. Nature Methods. 9: 676–682.

Schloissnig S, Kawaguchi A, Nowoshilow S, Falcon F, Otsuki L, Tardivo P, Timoshevs-kaya N, Keinath MC, Smith JJ, Voss SR & Tanaka EM. 2021. *The giant axolotl genome uncovers the evolution, scaling, and transcriptional control of complex gene loci*. Proc Natl Acad Sci USA. 118(15): e2017176118.

Schlüßler R, Möllmert S, Abuhattum S, Cojoc G, Müller P, Kim K, Möckel C, Zimmermann C, Czarske J & Guck J. 2018. *Mechanical Mapping of Spinal Cord Growth and Repair in Living Zebrafish Larvae by Brillouin Imaging*. Biophys J. 115(5): 911-923.

Sobkow L, Epperlein HH, Herklotz S, Straube WL & Tanaka EM. 2006. *A germline GFP transgenic axolotl and its use to track cell fate: dual origin of the fin mesenchyme during development and the fate of blood cells during regeneration*. Dev. Biol. 290:386-397.

Staple D.B., Farhadifar R., Röper J.C., Aigouy B., Eaton S., Jülicher F. 2010. *Mechanics and remodelling of cell packings in epithelia*. The European Physical Journal E, 33, 117–127.

Stavans J. 1993. *The evolution of cellular structures*. Reports on Progress in Physics, 56 733-789.

Sugiura T, Wang H, Barsacchi R, Simon A & Tanaka EM. 2016. *MARCKS-like protein*

is an initiating molecule in axolotl appendage regeneration. Nature. 531(7593): 237-40.

Sugiyama M, Sakaue-Sawano A, Iimura T, Fukami K, Kitaguchi T, Kawakami K, Okamoto H, Higashijima S & Miyawaki A. 2009. *Illuminating cell-cycle progression in the developing zebrafish embryo.* Proc Natl Acad Sci USA. 106:20812–20817.

Takahashi T, Nowakowski RS & Caviness VS Jr. 1995. *The cell cycle of the pseudos-tratified ventricular epithelium of the embryonic murine cerebral wall.* J Neurosci. 15(9): 6046-57.

Tazaki A, Tanaka EM & Fei JF. 2017. *Salamander spinal cord regeneration: The ultimate positive control in vertebrate spinal cord regeneration.* Dev Biol. 432(1): 63-71.

Thompson D. 1917. *On growth and form*, Cambridge : University Press.

Tinevez JY, PerryN, SchindelinJ, HoopesGM, ReynoldsGD, LaplantineE, BednarekSY, ShorteSL&EliceiriKW. 2017. *TrackMate: An open and extensible platform for single-particle tracking.* Methods. 1137115:80-90.

Toni T, Welch D, Strelkowa N, Ipsen A, Stumpf MP. 2009. *Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems.* J R Soc Interface. 6(31): 187-202.

Turing AM. 1952. *The Chemical Basis of Morphogenesis.* Philosophical Transactions of the Royal Society of London B. 237 (641): 37–72.

Turrero García M, Chang Y, Arai Y & Huttner WB. 2016. *S-phase duration is the main target of cell cycle regulation in neural progenitors of developing ferret neocortex.* J Comp Neurol. 524(3): 456-70.

Virtanen P, GommersR, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, van der Walt SJ, Brett M, Wilson J, Jarrod Millman K, Mayorov N, Nelson ARJ, Jones , Kern R, Larson E, Carey CJ, Polat İ, Feng Y, Moore EW, VanderPlas J, Laxalde D, Perktold J, Cimrman R, Henriksen I, Quintero EA, Harris CR, Archibald AM, Ribeiro AH, Pedregosa F, van Mulbregt P, and SciPy 1.0 Contributors. 2020. *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python.* Nature Methods, 17(3), 261-272.

Von Bertalanffy, Ludwig. 1976. *Teoría general de los sistemas. Fundamentos, desarrollo, aplicaciones.* México: Fondo de Cultura Económica. ISBN 9681606272.

Waskom M, Botvinnik O, O’Kane D, Hobson P, Lukauskas S, Gemperline DC, Augspurger T, Halchenko Y, Cole JB, Warmenhoven J, de Ruiter J, Pye C, Hoyer S, Vanderplas J, Villalba S, Kunter G, Quintero E, Bachant P, Martin M, Meyer K, Miles A, Ram T,

Yarkoni T, Lee Williams M, Evans C, Fitzgerald C, Brian, Fonnesbeck C, Lee A, Qalieh A, Zielke N & Edgar BA. 2015. *FUCCI Sensors: powerful new tools for analysis of cell proliferation*. WIREs 1153 Dev Biol. 4:469-487.

APÉNDICE A

Materiales y métodos

El problema de los sistemas es esencialmente el problema de las limitaciones de los procedimientos analíticos en la ciencia.

—Ludwig von Bertalanffy

A.1 Métodos computacionales

A.1.0.1 Parametrización del modelo

Los parámetros del modelo se resumen en el cuadro 2.1. Brevemente, la longitud de las células endodermiales a lo largo del eje *AP*, las distribuciones de la duración de las fases celulares y la fracción de crecimiento se fijaron de nuestra publicación anterior (Rodrigo Albors *et al.*, 2015). Los únicos parámetros del modelo libres son las células anteriores restantes después amputación N_0 , la longitud máxima λ de la señal putativa a lo largo del eje *AP* y τ , el tiempo máximo de reclutamiento celular.

A.1.0.2 Procedimiento de ajuste del *switchpoint* experimental con el límite de reclutamiento teórico $\zeta(t)$

El *switchpoint* obtenido experimentalmente de la médula espinal del axolotl en regeneración (extraído de Rost *et al.*, 2016) se ajustó con el límite de reclutamiento predicho por el modelo $\zeta(t)$. Usamos el Método Aproximado de Computación Bayesiana (ABC) para estimar la distribución de los parámetros que mejor reproducen los datos experimentales del *switchpoint* mediante nuestro modelo de límite de reclutamiento. Los métodos ABC omiten el requisito de evaluar las funciones de verosimilitud y captura la incertidumbre en las estimaciones de los parámetros del modelo (Csilléry *et al.*, 2010). En particular, utilizamos pyABC (Klinger, Rickert & Hasenauer, 2018), un paquete de alto rendimiento que implementa un esquema de Monte Carlo secuencial (ABC-SMC), que proporciona una técnica particularmente eficiente para la estimación de los parámetros (Toni *et al.*, 2018).

Brevemente, generamos una serie de simulaciones estocásticas del modelo 1D a partir de puntos muestreados del espacio de parámetros. La corrida se inicializó con un

tamaño de población de 1000 muestras. Todas las distribuciones *a priori* de los parámetros se definieron como una distribución uniforme discreta: $N_0 \sim \text{unif}\{100, 300\}$, $\lambda \sim \text{unif}\{500, 1500\}$ y $\tau \sim \text{unif}\{1, 192\}$. Donde los límites de τ están dados básicamente por el tiempo total de observación experimental (8 días) en horas. Los límites de τ (en μm) y N_0 fueron inicialmente estimado por ensayos de simulación previos.

Los valores de los parámetros muestreados se aceptaron sólo cuando la función distancia d entre el límite de reclutamiento simulado y el *switchpoint* experimental fue menor que una tolerancia dada ϵ . La función de distancia se definió de la siguiente manera:

$$d = \sqrt{\frac{\sum_i (x_i - \mu_i)^2}{\sigma_i^2}} \quad (\text{A.1})$$

donde μ_i , σ_i y x_i corresponden a la media, la desviación estándar del *switchpoint* experimental y el límite de reclutamiento simulado, respectivamente, en los puntos de tiempo experimentales i (4, 6 y 8 días). En cada iteración, las distribuciones de parámetros se actualizaron y se volvieron a muestrear. La nueva tolerancia ϵ fue entonces calculada como la mediana de las distancias a partir de las últimas muestras de población aceptadas. El resultado del algoritmo fue una muestra de valores de parámetros que inferen sus distribuciones posteriores (Figura 2.4 A, B). La convergencia del método se evaluó siguiendo el valor de ϵ y la tasa de aceptación, definida como el número aceptado de simulaciones dividido por el número total de simulaciones en cada paso de la iteración (Figura 2.4 C, D).

A.1.1 Sistema de coordenadas

En todas nuestras simulaciones, el tiempo comienza con el evento de amputación. El espacio corresponde al eje anterior-posterior (*AP*), donde 0 representa el plano de amputación y los valores positivos (negativos) son ubicaciones posteriores (anteriores).

A.1.2 Trayectorias y velocidades de los clones

Calculamos las trayectorias de los clones siguiendo las posiciones de cada clon en simulaciones aleatorias. Cuando una célula divide, muestreamos la posición media de las células clones como la posición del clon. En la figura 2.5, se muestran un total de 11 trayectorias, la primera comienza en 0 (el plano de amputación) y la última a $-1100\mu\text{m}$ (con una muestra cada $50\mu\text{m}$, aproximadamente). Para estimar la velocidad media de los clones en diferentes posiciones espaciales en este modelo, el espacio a lo largo del eje *AP* se subdividió en contenedores de $800\mu\text{m}$. Para cada trayectoria del clon, las posiciones se agruparon de acuerdo con estos contenedores. Grupos que contienen menos de dos mediciones fueron excluidos. La velocidad promedio para cada grupo se estimó mediante regresión lineal. Así fue calculada, la media y el desviación estándar de la velocidad de todos los clones en un contenedor.

A.1.3 Implementación del modelo y herramientas computacionales

Los modelos se implementaron en Python 3. Las simulaciones y el análisis de datos se realizaron utilizando Numpy (Oliphant, 2006) y Pandas (McKinney, 2010) mientras que la visualización de datos se llevo a cabo con Matplotlib (Hunter, 2007).

A.2 Materiales y métodos experimentales

A.2.1 Biología Molecular

El plásmido AxFUCCI se construyó mediante clonación de restricción estándar. Las características relevantes de los plásmidos AxFUCCI son: (i) sitio *SceI* para la transgénesis estable; (ii) promotor sintético de CAGG para expresión ubicua; (iii) sonda G0/G1-AxFUCCI (axolotl *Cdt1* [aa1-128] -GSAGSAAGSGEF glicina/enlazador serina-mVenus); (iv) Péptido viral "autoescindible" T2A; (v) Sonda S/G2-AxFUCCI (mCherry-SGGGGGSGGGGS glicina/serina enlazador-axolotl *Gmn* [aa1-93]); (vi) secuencia de poliadenilación de beta-globina de conejo; (vii) Sitio *SceI* para transgénesis estable. Los productos de PCR se amplificaron y se ligaron en dos rondas en un vector que ya alberga sitios *SceI*, un promotor CAGG, una secuencia T2A seguida de mCherry y secuencia de poliadenilación de beta-globina de conejo.

Los cebadores se compraron como soluciones madre $20\mu\text{M}$ (Sigma-Aldrich, des-sal estándar). La secuencia del plásmido AxFUCCI fue verificada por secuenciación de Sanger.

A.2.2 Cultivo de células AL1

La línea celular de axolotl inmortalizada "AL1" se cultivó en una incubadora humidificada a 25°C , con 2% de CO_2 . El medio de cultivo celular contiene: 62,5% de MEM, 10% de suero fetal bovino, 25% de agua, suplementado con 100U de penicilina-estreptomicina, glutamina, insulina. Las células AL1 se pasaron cada semana en una proporción de 1:2 en botellas de cultivo recubiertas de gelatina.

A.2.3 Electroporación de células AL1 e imágenes *in vivo*

La electroporación se realizó utilizando un sistema de transfección de neón (Thermo Fisher Scientific). Las 50.000 células AL1 se sometieron a electroporación con $1\mu\text{g}$ de plásmido AxFUCCI en PBS al 70%/agua utilizando los siguientes ajustes: 750V, ancho de pulso de 35ms, 3 pulsos. Las células AL1 electroporadas se colocaron en placas Ibidi

con fondo de vidrio recubierto con gelatina. Después de dos días, se cambió el medio de cultivo celular y la placa fue colocada en una cámara de microscopia de imágenes de células vivas automatizada Celldiscoverer 7 (Zeiss). La cámara del microscopio se mantuvo a 25°C, con 2% de CO₂. Se obtuvieron imágenes de las células cada hora durante el transcurso de 7 días para fluorescencia de Venus y mCherry y campo claro.

A.2.4 Mediciones de seguimiento de intensidad de fluorescencia

Las intensidades de fluorescencia de AxFUCCI se midieron utilizando el complemento TrackMate para FIJI (Tinevez, Perry & Schindelin *et al.*, 2017). Las intensidades de fluorescencia se normalizaron a la intensidad máxima observada para el respectivo fluoróforo durante el experimento.

A.2.5 Cuantificación de ADN por citometría de flujo

Las células AL1 electroporadas con AxFUCCI se incubaron durante 90 minutos con medio de cultivo celular que contenía colorante Hoechst 10 $\mu\text{g}/\text{ml}$ para la tinción de ADN. Después de la incubación, las células AL1 se lavaron una vez con PBS al 70%/agua, luego disociado en células individuales usando tripsina. La disociación se terminó agregando un volumen 1:1 de suero que contiene medio de cultivo celular. Las células se resuspendieron en 70% de PBS/agua, luego se filtraron a través de un filtro de 50 μm . Las células se analizaron para determinar el contenido de ADN utilizando un citómetro de flujo BD LSRFortessa y el software FlowJo.

A.2.6 Cría y transgénesis de axolotls

Los axolotls d/d y AxFUCCI (*Ambystoma mexicanum*), de longitud del hocico-a-cola 5,5 *cm*, fueron criados en acuarios individuales. Las crías de axolotls fueron realizadas en la instalación de animales IMP. Todos los experimentos se realizaron de acuerdo con las directrices del comité de ética aplicables localmente y dentro de un marco acordado el Magistrado de Viena (Oficina de Organismos Genéticamente Modificados y MA58, Ciudad de Viena, Austria). Los axolotls se anestesiaron con benzocaína (Sigma) diluida en agua del grifo antes de la amputación y/o imágenes.

Los axolotls AxFUCCI se generaron mediante transgénesis mediada por meganucleasa I-SceI utilizando los métodos previamente descritos (Sobkow *et al.*, 2006). Brevemente, se removió la cubierta gelatinosa de huevos fertilizados de axolotl d/d y se los inyectó con 5 *nl* de mezcla para inyección (aproximadamente 0,5 *ng* de plásmido AxFUCCI y 0,005 *U* de meganucleasa I-SceI (NEB) diluido en tampón CutSmart 1X (NEB)). Los huevos de axolotl inyectados se mantuvieron en 0,1 X MMR/agua a temperatura ambiente hasta el tamiz. Los animales fundadores transgénicos (F0) se identificaron por su fluorescencia de Venus utilizando un microscopio de campo amplio AXIOzoom V16

(Zeiss). Los animales AxFUCCI de la F1 transmitida por la línea germinal se utilizaron para todos los experimentos de este estudio. Los tamaños de las muestras se determinaron empíricamente y dentro los límites de la experimentación bajo las restricciones operativas inducidas por la pandemia de COVID19.

A.2.7 Limpieza óptica de tejidos e imágenes de láminas de luz

Las colas de AxFUCCI se fijaron durante la noche a 4°C en paraformaldehído al 4%. Las colas fijadas se lavaron bien con PBS, luego des-lipidada durante 30 minutos a 37°C en la Solución-1 del protocolo de aclarado de tejido DEEP-clear (10% v/v THEED, 5% v/v Triton X-100, 25% p/v de urea en agua) (Pende & Vadiwala *et al.*, 2020). Las colas de-lipidadas se lavaron con PBS, luego se incubaron durante 2 horas en PBS que contenía 10µg/ml de DAPI. Las colas se lavaron bien y luego se incubaron durante la noche en una solución para igualar el índice de refracción Easyindex (LifeCanvas tecnologías). Las muestras se mantuvieron oscuras en todo momento para evitar el blanqueamiento de la fluorescencia de AxFUCCI. Se tomaron imágenes de las colas AxFUCCI limpias en una solución EasyIndex utilizando un microscopio de lámina de luz LightSheet.Z1 (Zeiss) y una cámara personalizada.

A.2.8 Preparación de los cortes de tejido

Las colas de AxFUCCI se fijaron durante la noche a 4°C en paraformaldehído al 4%. Las colas fijadas se lavaron bien con PBS, luego se incubó durante la noche en sacarosa al 30% en PBS. Al día siguiente, las muestras se fijaron en compuesto OCT (temperatura de corte óptima), congelado en hielo seco y almacenado a -80°C hasta el corte. Se prepararon criosecciones de 10µm de espesor a partir de bloques congelados y se almacenaron a -20°C hasta su uso.

A.2.9 Inmunotinción y captura de imágenes de las secciones de tejido

Las criosecciones se calentaron a temperatura ambiente y luego se lavaron extensamente con PBS para eliminar el OCT. Las secciones se bloquearon durante 2 horas a temperatura ambiente con suero normal de cabra (NGS) al 10% diluido en PBS que contiene Triton X-100 (PBTx) al 0,2%. Las muestras bloqueadas se incubaron con anticuerpo primario diluido en NGS al 1% durante la noche a 4°C. Al día siguiente, las secciones se lavaron bien con PBTx y luego se tiñeron con anticuerpos secundarios conjugados con Alexa Fluor diluidos 1:500 en PBTx durante 2 horas a temperatura ambiente. Se incluyó DAPI en la solución de tinción secundaria a una concentración de 10µg/ml. Las secciones fueron bien lavadas con PBTx y se montó en Mowiol que contenía DABCO

(Sigma) al 2,5%. Los siguientes anticuerpos primarios fueron utilizados en este estudio: anti-NeuN (Millipore MAB377, ratón, 1:500), anti-Sox2 (conejo, 1: 1.000, Fei *et al.*, 2016). Las imágenes se adquirieron usando un microscopio confocal invertido LSM980 AxioObserver (Zeiss).

A.2.10 Administración y detección de EdU

Los axolotls anestesiados fueron inyectaron intraperitonealmente con 400 μ M de EdU (diluido en PBS) a una dosis de 20 μ l/g. Se añadió el colorante FastGreen (Sigma-Aldrich) a la mezcla de inyección para ayudar a la visualización. Los axolotls inyectados se mantuvieron fuera del agua durante un período de recuperación de 20 minutos bajo toallas empapadas en benzocaína. Después de la recuperación, los axlotls inyectados se devolvieron al agua. Luego del período deseado del experimento de pulso y caza, los axolotls se sacrificaron, se fijaron los tejidos de la cola y se prepararon las criosecciones.

La detección de EdU se realizó con el kit de detección Click-iT 647 EdU (Thermo Fisher Scientific) de acuerdo con las instrucciones del fabricante.

A.2.11 Análisis de imágenes

Los datos de lámina de luz (colas de axolotl AxFUCCI) se volvieron a seccionar digitalmente o se renderizaron en 3D utilizando Imaris software (Oxford Instruments). Para la cuantificación de datos completos, la médula espinal se volvió a seccionar longitudinalmente, para producir una tira continua de la luz (el lumen) de la médula espinal, y las imágenes se exportaron como TIFF para el recuento de células en FIJI. Las fases del ciclo celular de las células endodiales se cuantificaron a partir de datos digitales de secciones de 25 μ m de espesor. Las células endodiales se definieron como células en contacto directo con la luz de la médula espinal. Los videos de Celldiscoverer 7 (células AL1) se recortaron usando el software ZEN blue (Zeiss), luego se analizaron usando el Complemento TrackMate para FIJI, como se describe anteriormente. Las imágenes de la sección de tejido se analizaron y cuantificaron utilizando Software FIJI (Schindelin *et al.*, 2012).

A.2.12 Determinación del límite AP entre dos zonas espaciales adyacentes dentro de la médulas espinales en regeneración de axolotl

Probamos si las células endodiales en las diferentes fases del ciclo celular estarían heterogéneamente distribuidas a lo largo del eje AP de la médula espinal en regeneración. Con ese objetivo, ajustamos los perfiles experimentales AP espaciales de los porcentajes de células que expresan G0/G1-AxFUCCI y S/G2-AxFUCCI, por animal, con un modelo

matemático que supone dos zonas espaciales homogéneas adyacentes separadas por un borde anterior-posterior, como sigue:

$$g0g1(x) = \begin{cases} g0g1_a & \text{if } x < APborder \\ g0g1_p & \text{if } x \geq APborder \end{cases}$$

$$sg2(x) = \begin{cases} sg2_a & \text{if } x < APborder \\ sg2_p & \text{if } x \geq APborder \end{cases}$$

Donde $g0g1(x)$ y $sg2(x)$ son las variables del modelo que describen la distribución espacial de células G0/G1 y S/G2 a lo largo de x , la posición espacial a lo largo del eje AP . Los parámetros del modelo son $g0g1_a$, el porcentaje anterior de células que expresan G0/G1-AxFUCCI, $g0g1_p$, el porcentaje posterior de células G0/G1-AxFUCCI que expresan, $sg2_a$, el porcentaje anterior de células que expresan S/G2-AxFUCCI, $sg2_p$, el porcentaje de células que expresan S/G2-AxFUCCI y el borde AP , los bordes entre las zonas anteriores y posteriores, asumidas iguales para las células G0/G1-AxFUCCI y S/G2-AxFUCCI.

Ajustamos el modelo simultáneamente al perfil AP del porcentaje de G0/G1-AxFUCCI y S/G2-AxFUCCI que expresan las células de cada animal y a cada tiempo mediante el uso del Método de Cálculo Bayesiano Aproximado (ABC, consulte la sección de métodos computacionales para obtener más detalles de la implementación). Cada ajuste se inició con un tamaño de población constante de 1000 muestras. Las distribuciones *a priori* de los parámetros se definieron como una uniforme discreta entre 0 y 100% para $g0g1_a$, $g0g1_p$, $sg2_a$ y $sg2_p$. Para el borde AP también se definió una uniforme discreta que cubre todas las posiciones medidas a lo largo el eje AP . La función de distancia entre los datos experimentales FUCCI y el modelo de dos zonas fue definida como:

$$d = \sum_x \sqrt{(G0/G1_{exp}(x)g0g1(x))^2 + (S/G2_{exp}(x)sg2(x))^2}$$

Donde $G0/G1_{exp}(x)$ y $S/G2_{exp}(x)$ son los porcentajes de G0/G1-AxFUCCI y S/G2-AxFUCCI que expresan las células, respectivamente, determinadas en el intervalo x del eje AP de la médula espinal (para cada animal y cada tiempo).

Cada procedimiento de ajuste tuvo un total de 30 iteraciones. Los resultados del ajuste se muestran en la Figura 3.2B, Figura 3.3 y 3.4.

Para cada tiempo, comparamos las zonas anterior versus posterior de G0/G1-AxFUCCI y S/G2-AxFUCCI simultáneamente mediante la realización de una prueba de Kolmogorov-Smirnov entre el mejor ajuste de los parámetros $g0g1_a$, $g0g1_p$ versus $sg2_a$, $sg2_p$. Aunque las zonas anterior y posterior eran indistinguibles de cero a 3 días después de la amputación, encontramos una diferencia significativa entre las zonas anterior y posterior en los datos de G0/G1-AxFUCCI y S/G2-AxFUCCI en el día 4 y 5 (Figura 3.3). El borde AP detectado que mejor se ajusta para cada tiempo luego de la amputación se muestra en la Figura 3.2B como áreas grises verticales a día 4 y 5, luego de la amputación.

A.2.13 Análisis estadístico y representación de datos

En la Figura 3.2, se implementó la prueba de Kolmogorov-Smirnov utilizando la biblioteca Scipy (Virtanen *et al.*, 2020). Las funciones matemáticas de alto nivel de Numpy (Harris *et al.*, 2020) se utilizaron a lo largo de las simulaciones y análisis de los datos. En la Figura 2.7, Figura 2.4, figura 2.6, figura 2.8 y la Figura B.4B se realizaron utilizando Matplotlib (Hunter, 2007) mientras que la Figura 3.2B, la Figura 3.3, la Figura 3.4, la Figura 3.6, la Figura 3.7 y Figura 3.8 se realizaron con Seaborn (Waskom *et al.*, 2017). En la Figura 3.5, la Figura B.6 y la Figura B.4, los análisis estadísticos se realizaron utilizando R. Los datos AxFUCCI fueron probados para supuestos de normalidad (prueba de Shapiro-Wilk) e igualdad de varianza (prueba de Levene) para determinar las pruebas estadísticas adecuadas a realizar. No se excluyeron datos. Los detalles de las pruebas estadísticas y sus resultados se pueden encontrar en la leyenda de la figura correspondiente. La significación estadística se definió como $p < 0,05$. Estos gráficos se trazaron utilizando Prism (GraphPad). Todas las figuras fueron compiladas en Adobe Illustrator.

APÉNDICE B

Otros resultados experimentales de Fucci

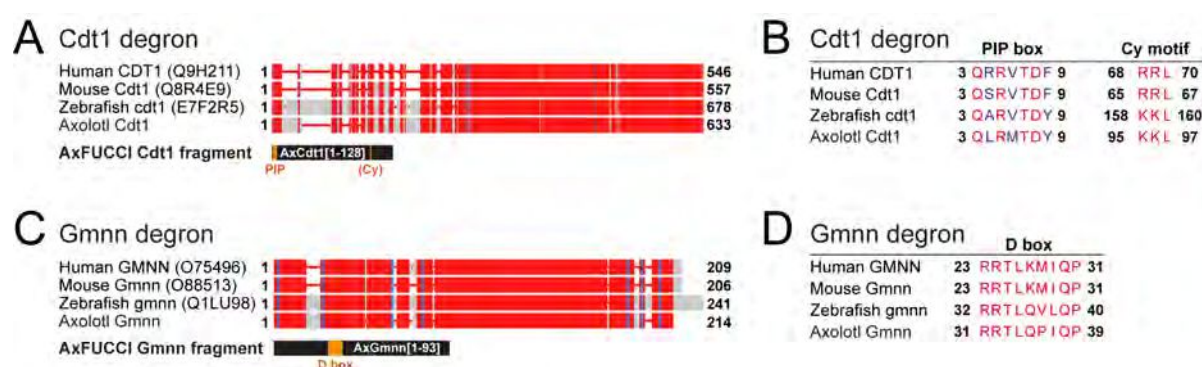
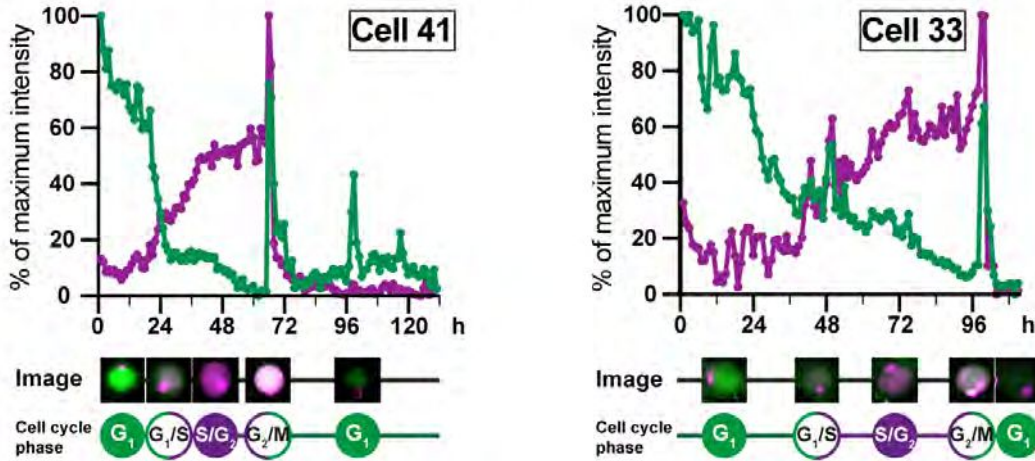


Figura B.1: Determinación de fragmentos de Cdt1 y Gmn para el reportero AxFuCCI. A) Alineamiento de proteínas Cdt1 de longitud completa de ser humano, ratón, pez cebra y axolotl. Los identificadores Uniprot se dan entre paréntesis. Los números indican números de residuos de aminoácidos. Las regiones rojas indican una alta similitud/identidad de aminoácidos; regiones en gris indican baja conservación. Las alineamientos se realizaron utilizando COBALT (NCBI). El fragmento AxFuCCI Cdt1 comprende los aminoácidos 1-128 del axolotl Cdt1, que albergan la caja PIP degron y corresponden a los aminoácidos 1-190 de Cdt1 de pez cebra utilizado para hacer Fucci de pez cebra (Sugiyama et al., 2009, Bouldin et al., 2014). B) Alineamiento de las secuencias de aminoácidos del **degron Cdt1**. La caja PIP está bien conservada evolutivamente. Al igual que el cdt1 de pez cebra, el Cdt1 de axolotl alberga un motivo "KKL" en lugar del motivo "RRL" que se encuentra en humanos y ratones. C) **Alinamiento de longitud completa de proteínas gmn de humanos, ratones, peces cebra y axolotls.** El fragmento AxFuCCI Gmn comprende los aminoácidos 1-93 de axolotl Gmn, que alberga el cuadro de Destrucción (D) y corresponde a los aminoácidos 1-100 del pez cebra gmn1 utilizado para hacer Fucci de pez cebra (Sugiyama et al., 2009, Bouldin et al., 2014). D) Alineaciones de aminoácidos de la caja Gmn D.

A Fluorescence tracking: G_1 to G_1 AxFUCCI - two examples



B Fluorescence tracking: S/G_2 to S/G_2 AxFUCCI - two examples

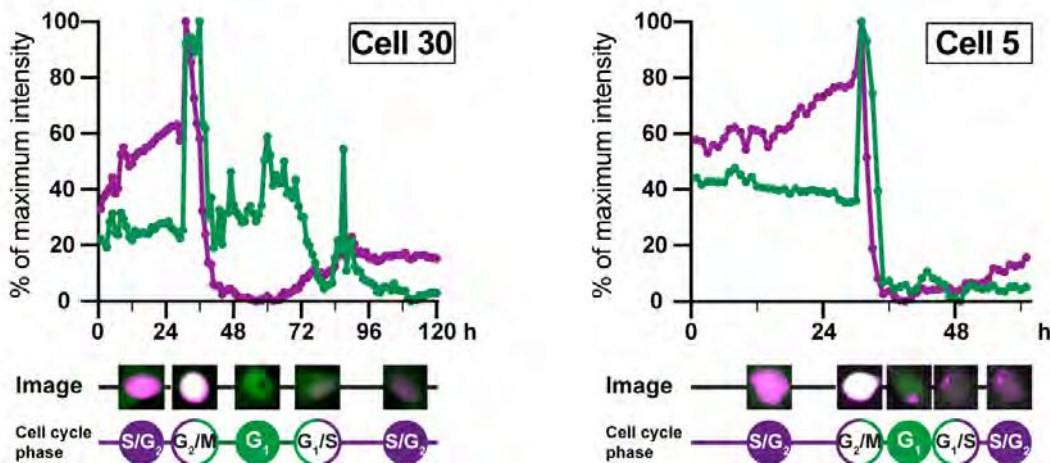


Figura B.2: Determinaciones de intensidad de fluorescencia de AxFUCCI a través del ciclo celular. Mediciones de la intensidad de la fluorescencia de las células AL1 electroporadas con AxFUCCI a medida que transitaban a través de un ciclo celular de G_1 al siguiente G_1 (A) o S/G_2 al siguiente S/G_2 . (B) Se presentan dos ejemplos para cada transición. Las líneas verdes representan la fluorescencia G_0/G_1 -AxFUCCI; las líneas magenta representan la fluorescencia de S/G_2 -AxFUCCI. Para cada punto de tiempo, el núcleo celular se segmentó y se calculó la intensidad media de fluorescencia. Las intensidades de fluorescencia se normalizaron al valor máximo observado durante la sesión de imágenes. h: horas después inicio de la imagen. La máxima fluorescencia se observa en la mitosis, cuando la célula se redondea en preparación para la división celular. La intensidad media de la fluorescencia disminuye drásticamente después de cada mitosis debido a la dilución del fluoróforo y del plásmido entre las dos células hijas. Para ver videos de muestra, vea los Videos 2 y 3.

A DNA content analysis (AxFUCCI-electroporated AL1 cells)

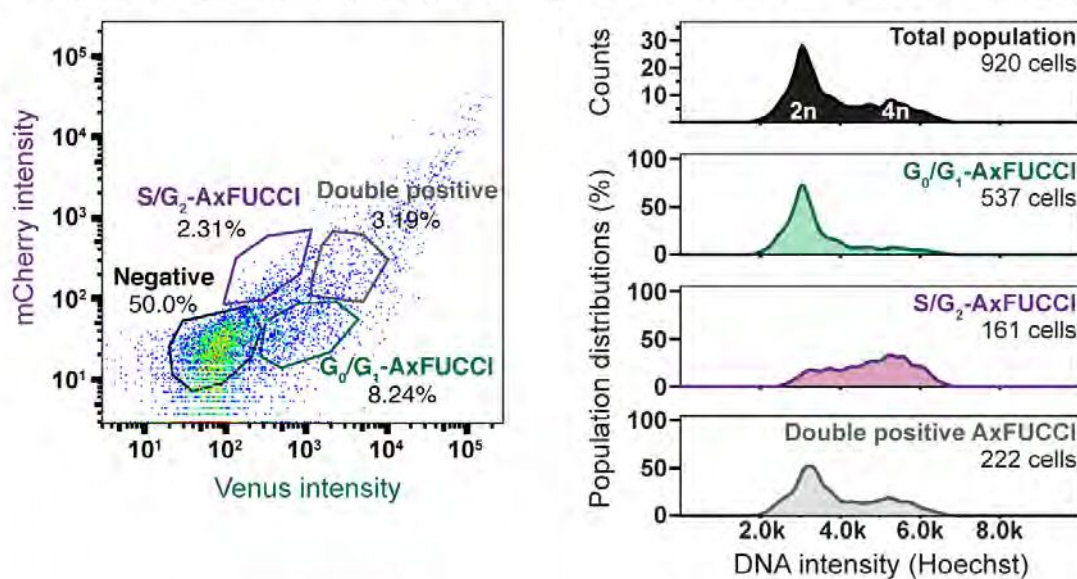


Figura B.3: Cuantificación de ADN por citometría de flujo para validar la funcionalidad AxFUCCI. **A)** Contenido de ADN relativo en células G₀/G₁-AxFUCCI (verde), S/G₂-AxFUCCI (magenta) o Transición-AxFUCCI (gris). Las células AL1 electroporadas con AxFUCCI se incubaron en medio de cultivo celular que contenía tinción de ADN Hoechst durante 90 minutos antes de la disociación y el análisis.

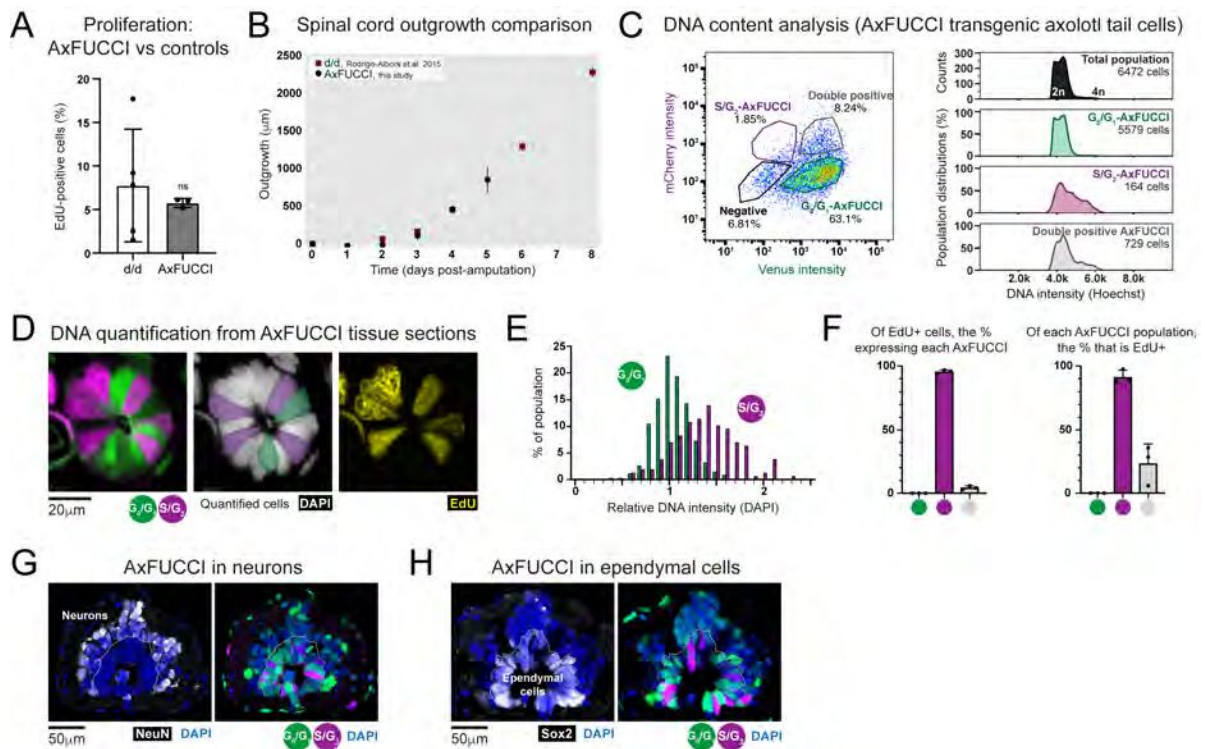


Figura B.4: Validación de la funcionalidad AxFucci mediante análisis de tejidos *in vivo*.

Figura B.4: A) La línea de base de proliferación en la médula espinal AxFucci no es significativamente diferente de los controles d/d. EdU se administró en 2 pulso de una hora a AxFucci en animales no lesionados o de control d/d. Las colas fueron cosechadas y seccionadas para cuantificar células proliferativas marcadas con EdU en la médula espinal. ns: no significativo, $p = 0,62$ (prueba t de dos muestras). Las barras de error indican desviación estándar. $n = 5$ colas (control) o 3 colas (AxFucci), 750 células contadas en total para cada una, de las cuales aproximadamente 50 células fueron marcadas con EdU. **B) Las colas AxFucci se regeneran con cinética normal.** El crecimiento de la médula espinal se midió cada día hasta 5 días después de la amputación de la cola AxFucci, y fue representados junto con las mediciones obtenidas previamente de animales de control d/d (Rodrigo-Albors et al., 2015). **C) Cuantificación de ADN por citometría de flujo.** Las células se incubaron con tinción de ADN de Hoechst durante 60 minutos antes del análisis. **D) Cuantificación de ADN de secciones de tejido AxFucci.** Imagen confocal de una sola sección de una columna vertebral AxFucci de $10\mu m$ de espesor de sección transversal de la médula, teñida conjuntamente con DAPI (gris) para marcar el ADN. El panel central muestra las células que satisfacen los criterios de cuantificación: deben abarcar la médula espinal, entrar en contacto con la luz y no quedar oscurecidas por las células vecinas. Los animales AxFucci también se pulsaron con EdU durante 8 horas antes de la recolección de la cola. **E) Histograma que muestra las intensidades DAPI de las células AxFucci cuantificado en D.** La intensidad media de las células G0/G1-AxFucci se estableció en un valor arbitrario de 1 ($2n$). Las células S/G2-AxFucci tienen un contenido de ADN significativamente más alto que las células G0/G1-AxFucci ($p = 2,20 \times 10^{-16}$, prueba de Welch). $n = 3$ colas (total de 341 células G0/G1-AxFucci frente a 157 células S/G2-AxFucci). **F) Correlación del marcador EdU (ver panel D) con cada población que expresa AxFucci.** $n = 3$ colas, se contaron 819 células que expresan AxFucci en total, de las cuales 171 había incorporado EdU. **G) Las neuronas de la médula espinal expresan G0/G1-AxFucci.** Las neuronas que expresan NeuN se encuentran en la periferia de la médula espinal. $98 \pm 1,5\%$ de las neuronas expresaron G0/G1-AxFucci, y el resto no expresar ningún AxFucci. $n = 5$ colas, se contaron un total de 2170 células NeuN +. **H) Las células ependimales expresan todas las combinaciones de AxFucci.** Las células ependimales que expresan Sox2 están ubicadas en el interior de la médula espinal, que recubren el lumen.

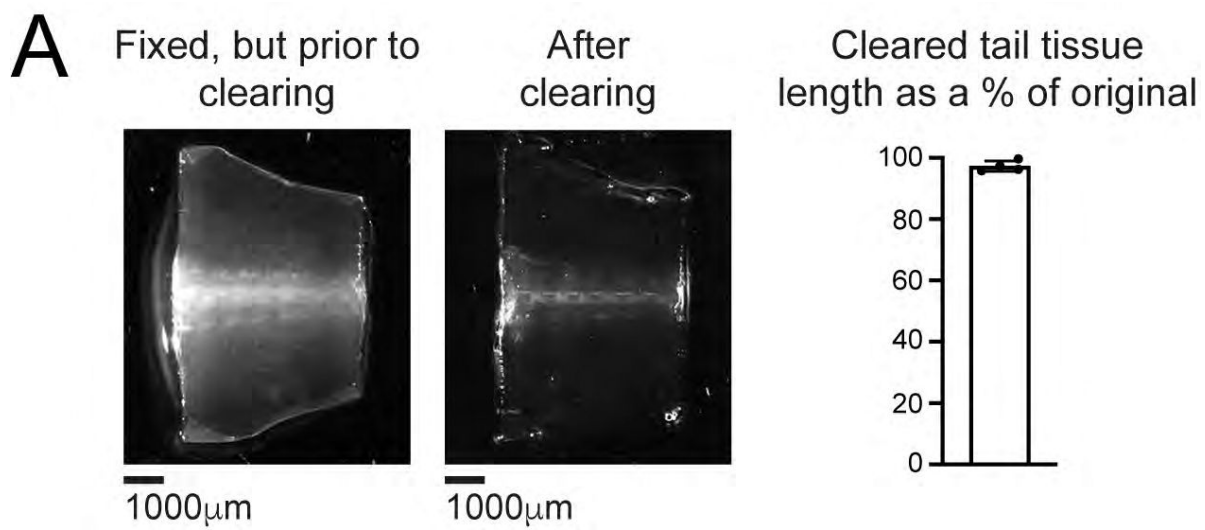


Figura B.5: El protocolo de limpieza de tejidos no altera la longitud de la médula espinal. A) Lo mismo, imagen fija del tejido de la cola antes (izquierda) o después (centro) del aclaramiento del tejido y coincidencia del índice de refracción. Derecha: despejado del tejido de la cola conserva el $97 \pm 1,7\%$ de su longitud en comparación con el tejido de la cola sin despejar. $n = 3$ colas.

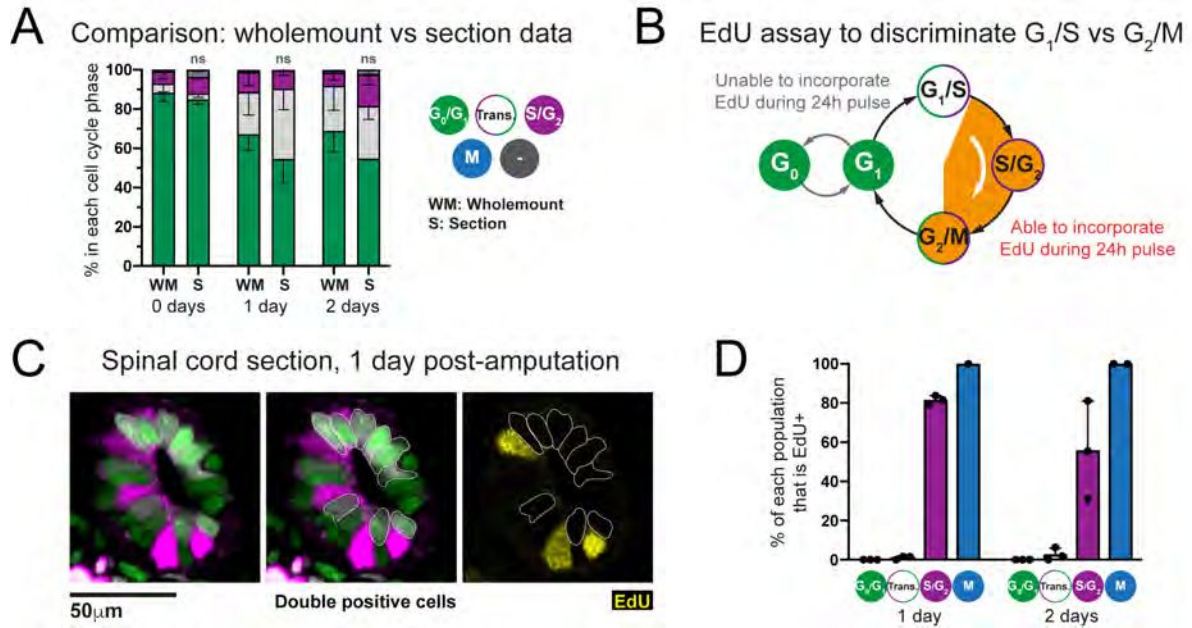


Figura B.6: Las células Transición-AxFucci 1 y 2 días después de la amputación se encuentran en la transición G_1/S . A) Comparación de las cuantificaciones de AxFucci realizadas a partir de muestras completas (completas) o de secciones de tejido (Sección). No se observaron diferencias entre las cuantificaciones basadas en montajes completos y las basadas en secciones (ANOVA de una vía). $n = 3$ colas por punto de tiempo, aproximadamente 400 células contadas en cada una, correspondientes a aproximadamente 750 μm de la médula espinal. Las barras de error indican la desviación estándar. B) Ensayo basado en EdU para distinguir células G_1/S y células G_2/M . C) Imagen confocal de sección única de 10 μm de médula espinal 1 día después de la amputación, pulsada durante 24 horas con EdU antes de la cosecha. Las células AxFucci de transición (doble positivo, delineadas) no han incorporado EdU (amarillo). Las células vecinas S/G_2 -AxFucci (magenta) han incorporado EdU (control interno). D) El porcentaje de cada población de células AxFucci y células mitóticas que incorporaron EdU en el ensayo en B. Como se esperaba, ninguna de las células G_0/G_1 -AxFucci incorporaron EdU y todas las células mitóticas habían incorporado EdU. La mayoría de las células S/G_2 -AxFucci incorporó EdU. Por el contrario, casi ninguna de las células Transición-AxFucci 1 y 2 días después de la amputación incorporó EdU, lo que indica que residen en la transición G_1/S . $n = 3$ colas por punto de tiempo. Total de células contadas por punto de tiempo: 1 día después de la amputación [671 G_0/G_1 -AxFucci, 115 S/G_2 -AxFucci, 442 Transición-AxFucci, 1 célula mitótica], 2 días después de la amputación [643 G_0/G_1 -AxFucci, 191 S/G_2 -AxFucci, 315 Transición-AxFucci, 3 células mitóticas].

