Facultad de Ingeniería
Universidad Nacional de Mar del Plata

Modelado Constitutivo Multiescala de Matrices Nanofibrosas Electrohiladas

Ing. Daniel Enrique Caballero

Tesis presentada para optar por el título de
Doctor en Ingeniería, Orientación Mecánica

Director: Dr. Santiago A. Urquiza
Codirector: Dr. Guillermo A. Lombera

Mar del Plata, Argentina
Agosto 2020

Facultad de Ingeniería
Universidad Nacional de Mar del Plata

Modelado Constitutivo Multiescala de Matrices Nanofibrosas Electrohiladas

Ing. Daniel Enrique Caballero

Tesis presentada para optar por el título de
Doctor en Ingeniería, Orientación Mecánica

Director: Dr. Santiago A. Urquiza
Codirector: Dr. Guillermo A. Lombera

Mar del Plata, Argentina
Agosto 2020

A mis viejos

Agradecimientos

Esta tesis es el producto de mi esfuerzo y formación durante los últimos años de mi vida. Sin embargo, haber llevado a cabo esta labor no hubiera sido posible sin la participación, sea en el ámbito profesional o personal, de muchas personas que me han guiado y apoyado en el camino.

Agradezco en primer lugar a mi familia, a mis padres, y mis hermanos, por haberme brindado su amor, una infancia feliz, su apoyo incondicional siempre, y la formación más importante como ser humano. Gracias a ellos crecí en quien soy hoy, y todo se los debo a ellos.

Asimismo, gracias a todos mis amigos de cada etapa de mi vida, esa otra familia que fui eligiendo a cada paso. Con ellos compartí vivencias, viajes, aventuras, risas, charlas, reflexiones y más, todas cuestiones esenciales para mi felicidad. También compartí, vale decirlo, momentos difíciles, pero que con su cariño y apoyo pude atravesar con éxito, y hoy forman parte de mi crecimiento personal. Gracias particulares a mis amigos cercanos durante mi doctorado, que me acompañaron en estos años, prestándome un gran apoyo moral y humano, necesarios en los momentos difíciles de este trabajo y esta profesión.

En igual medida, gracias a Luli, mi compañera, por compartir su vida con la mía en los últimos años de mi doctorado. Realmente no debe ser sencillo convivir con un doctorando, especialmente en los últimos años cuando el intenso trabajo de tesis se traduce en largas jornadas de trabajo que incluso se extienden a los días de descanso. Todo su amor y su apoyo me fueron indispensables para recargar energías luego de cada día y semana de trabajo.

En el plano profesional, agradezco a mi director, el Dr. Santiago Urquiza, por haberme ayudado para resolver cada duda y guiado para sortear cada obstáculo que crucé durante mi doctorado. Por su calidad humana y profesional de las que aprendí tanto durante estos años. Asimismo, a mi codirector, el Dr. Guillermo Lombera, por su inspiradora presencia, su confianza en mí y su apoyo desde el inicio de este trabajo. Agradezco también a mis compañeros del Grupo de Ingeniería Asistida por Computadora, por haber creado un gran ambiente para el trabajo. En especial a mis compañeros de box, Nico y Manu, por su buena onda de todos los días, haciendo que ir diariamente a trabajar sea también un encuentro ameno y alegre.

Por su invaluable colaboración desde el punto de vista experimental, agradezco a la Dra. Florencia Montini Ballarin, de la División de Polímeros Biomédicos (INTEMA). Este trabajo de tesis se vio enriquecido en gran medida por su aporte de los materiales

necesarios para realizar los ensayos experimentales, además de compartir su vasto conocimiento acerca de los mismos. También al Grupo de Propiedades Mecánicas de Polímeros, y en especial a su directora, la Dra. Patricia Frontini, por facilitarnos las máquinas y elementos necesarios para llevar a cabo los ensayos mecánicos. Además a la División Ciencia e Ingeniería de Polímeros (INTEMA) por la colaboración prestada en la realización de los estudios mecánicos. En especial al Dr. Nahuel Rull por compartir su tiempo, conocimiento y experiencia para la operación de los equipos.

Resumen

La ingeniería de tejidos busca proveer a las aplicaciones biomédicas de injertos hechos a medida que logren exhibir propiedades mecánicas biomiméticas. En este contexto, el electrohilado ha surgido como una técnica con características prometedoras para la producción de injertos vasculares nanofibrosos. Sin embargo, para mejorar la biomímesis de los injertos electrohilados, con miras a la producción de reemplazos vasculares paciente específicos, es necesario mejorar la comprensión del comportamiento mecánico de las matrices electrohiladas teniendo en cuenta las características morfológicas microestructurales y las propiedades mecánicas de las nanofibras de acuerdo con el material seleccionado. Para este fin, el modelado constitutivo multiescala aparece como una poderosa herramienta, capaz de vincular los mecanismos microscópicos subyacentes bajo deformación con la respuesta mecánica observada a nivel macroscópico. Además, su empleo en ciclos de optimización puede hacer posible identificar las propiedades constitutivas de las nanofibras así como la microtopología de la matriz capaces de alcanzar un comportamiento macroscópico con aceptable grado de aceptación. En base a estas consideraciones, se presenta en este trabajo de tesis doctoral el desarrollo de modelos constitutivos multiescala para matrices nanofibrosas electrohiladas, teniendo en cuenta las características esenciales de la microestructura de los injertos, las propiedades constitutivas de las nanofibras y la interacción entre las mismas.

En primera instancia se presenta el modelo en dos escalas, adoptando una descripción clásica de un sólido continuo para la escala macroscópica, y un Elementos de Volumen Representativos (RVE) discreto conformado por fibras individuales agrupadas en fascículos. El modelo se validó con datos experimentales de ensayos de inflado de tubos electrohilados. Además se estudió el efecto de variar las propiedades microscópicas sobre la respuesta macroscópica y se utilizó esta información para obtener los parámetros necesarios para un injerto capaz de imitar la respuesta en presión-diámetro de una arteria intracranial humana.

A continuación se explora la generación de geometrías realistas que reproduzcan los aspectos más importantes de la microestructura electrohilada. Se desarrolló un algoritmo de deposición virtual de fibras, con inspiración en la deposición de nanofibras durante el proceso de electrohilado, capaz de generar mallas de fibras interconectadas. Para evaluar la aptitud de estas geometrías para servir como RVE de los modelos multiescala, se estudió la variabilidad estadística en función del tamaño de las mallas. Además, se muestra la capacidad de diseño del algoritmo para generar geometrías que reproduzcan microestructuras con diferentes grados de alineamiento y con distintos niveles de enrulamiento de las fibras.

Por último, se desarrolló un modelo micromecánico para ser aplicado a las mallas generadas anteriormente. Este modelo microscópico lleva en consideración la interacción de las fibras en los puntos de intersección, así como la posibilidad de deformación plástica y rotura de las mismas. Por otra parte, se llevaron a cabo ensayos de tracción uniaxial sobre matrices electrohiladas para obtener las curvas de tensión-deformación experimentales necesarias para su validación. Los resultados obtenidos muestran que el modelo logra reproducir exitosamente la respuesta macroscópica elastoplástica de las matrices a partir de parámetros microscópicos realistas.

Abstract

Biomedical applications need tailor-made scaffolds that exhibit biomimetic mechanical properties. In this context, electrospinning has emerged as a technique with promising features for their production, able to yield nanofibrous mats with similar microstructure to the tissues they intend to replace. However, in order to improve the biomimetic capabilities of electrospun scaffolds, their mechanical behavior as a function of the microstructure and nanofiber properties needs to be better understood and controlled. To that end, multiscale constitutive modeling appears as a powerful design tool, not only able to characterize electrospun structures, but also to determine the fiber properties and scaffold microstructure that would achieve the objective response. With focus in this matters, this dissertation presents multiscale constitutive models for nanofibrous structures that takes into account the scaffold microstructure, the material constitutive properties of the nanofibers and their interaction.

As a first step, a two-scale model is presented, adopting a classical description of a continuous three-dimensional body for the macro-scale, and a discrete RVE consisting of individual nanofibers grouped in bundles or fascicles. Experimental data from pressure vs diameter inflation tests of electrospun PLLA tubular scaffolds was used to validate the model. In addition, the influence of the microstructural parameters upon the macroscopic constitutive behavior was studied. Then, the model parameters were adjusted to obtain a vascular graft able to reproduce the mechanical response of a human intracranial artery.

Next, the generation of realistic fiber mesh geometries is explored, aiming to reproduce the most important aspects of the electrospun microstructure. Consequently, a virtual fiber deposition algorithm was developed, that draws inspiration in the actual deposition of nanofibers during the electrospinning process. Then, to assess the aptitude of the obtained geometries to serve as Representative Volume Elements (RVE), the statistical dependence of the mesh with respect to its size was studied. Furthermore, the design capabilities of the algorithm are shown by assembling geometries with different levels of alignment as well as increasing tortuosity of the nanofibers.

Finally, a micromechanic model for the previously generated meshes is established, thus obtaining a complete mechanical representation of the electrospun microstructure. This microscopic model takes into account the interaction between nanofibers at contact points, and the possibility of fiber plastic deformation or even breakage. Uniaxial traction tests of electrospun scaffolds were also performed in order to obtain the necessary experimental data for the validation. The results show that the model successfully reproduces the macroscopic elastic-plastic response of the scaffolds employing realistic values for the geometric and constitutive parameters.

Índice

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación

Los implantes vasculares cumplen un rol cada vez más importante en el tratamiento de un amplio rango de patologías tales como aterosclerosis, aneurismas, malformaciones congénitas, vasculitis y traumas, entre otras [1]. Este rol cobra mayor preponderancia teniendo en cuenta que las enfermedades cardiovasculares representan la principal causa de muerte a nivel mundial [2], incluyendo a los países latinoamericanos y Argentina en particular [3].

Actualmente la primera elección para injertos arteriales es el trasplante autólogo. Sin embargo, debido a condiciones médicas previas, muchas veces no es posible conseguir un tejido adecuado, además que la extracción puede causar nueva morbilidad [1]. A su vez, la obtención de tejidos alogénicos resulta aún más complicada debido a la dificultad agregada de la biocompatibilidad entre donante y huésped. Como consecuencia de esta problemática, actualmente existe una creciente demanda de injertos vasculares para reemplazo de arterias de pequeño porte con buena adaptación a largo plazo. Los injertos sintéticos actualmente disponibles en la práctica clínica carecen de potencial de crecimiento y remodelado, por lo que sus resultados a largo plazo son insatisfactorios por el surgimiento de complicaciones graves como trombosis, hiperplasia, estenosis e infecciones, entre otros [4, 5]. En respuesta a estas limitaciones surge la ingeniería de tejidos, entendida como “un campo multidisciplinario que aplica los principios de la ingeniería y las ciencias biológicas hacia el desarrollo de sustitutos que restauren, mantengan y mejoren la función de los tejidos” [6]. Dentro de la ingeniería de tejidos vasculares, se investigan tres aproximaciones diferentes, la ingeniería de tejidos sin soportes porosos, el empleo de matrices descelularizadas y el uso de matrices porosas [7]. En base a este enfoque se desarrollaron vasos autólogos en ambientes ex-vivo, a través del uso de biorreactores. De esta manera se evitan los problemas de biocompatibilidad, pero aparecen importantes limitaciones desde

el punto de vista del alto costo implicado y del largo tiempo requerido para la obtención del injerto [8].

Un enfoque novedoso de la ingeniería de tejidos consiste en la fabricación de andamios (*scaffolds*) vasculares sintéticos bioabsorbibles capaces de permitir la regeneración de tejido autólogo al mismo tiempo que se produce la degradación del material artificial, concluyendo en la generación de una neoarteria compuesta completamente por tejido endógeno, capaz de alcanzar un estado de homeostasis con capacidad de remodelado, crecimiento y, consecuentemente, de reparación [9]. Para esto, es necesario que el injerto sea capaz de proveer temporalmente un microambiente que permita la infiltración celular y de nutrientes, que sea capaz de soportar las cargas hemodinámicas y que provea de condiciones similares a las fisiológicas que permitan el adecuado desarrollo de los nuevos tejidos [10]. Muchos tipos importantes de tejidos naturales poseen una microestructura de red nanofibrosa. Materiales biológicos como las células citoesqueléticas, redes de fibras de colágeno, elastina, y otras proteínas naturales con forma de fibras son los componentes estructurales principales de tejidos como el cartílago, tendones, vasos sanguíneos, válvulas cardíacas, entre otros. Dentro de los materiales utilizados para la construcción de andamios bioabsorbibles se destacan las matrices nanofibrosas electrohiladas, ampliamente exploradas en esta área de aplicación y que serán objeto de estudio de este trabajo de tesis [4, 5, 9, 11].

La tecnología de electrohilado constituye uno de los métodos de procesamiento de vanguardia que presenta mayores ventajas para la producción de nanofibras. La técnica tiene la habilidad única de producir nanofibras de diferentes materiales y geometrías, bajo costo, relativamente alta velocidad de producción y simplicidad en el diseño del equipamiento. En los últimos años, se han electrohilado numerosos tipos de materiales que incluyen prácticamente todos los polímeros (sintéticos y naturales, que sean solubles o puedan fundirse) y nanocompuestos, para obtener fibras continuas de unos pocos nanometros hasta algunos micrones que generan una matriz hilada no tejida altamente porosa [12]. Esta es una técnica versátil, que permite la producción de fibras de nano y microescala que presentan un gran potencial para imitar el microambiente fibroso de la matriz extracelular natural presente en las arterias [13]. El electrohilado ofrece la posibilidad de ajustar finamente las propiedades mecánicas durante la fabricación, controlando la microarquitectura porosa, tamaño y la orientación de las fibras, logrando comportamientos anisotrópicos como los observados en los vasos sanguíneos. Particularmente, las grandes deformaciones y rotaciones que experimentan las nanofibras elastoméricas consiguen que

los andamios electrohilados puedan soportar deformaciones del mismo nivel que los tejidos blandos que buscan reemplazar [14–16]. Además, la posibilidad de usar un colector rotatorio de pequeño diámetro, resulta en la obtención de una tubuladura sin costura ideal para aplicaciones vasculares.

La respuesta mecánica que manifiesta el injerto vascular una vez implantado es una característica fundamental para su diseño por dos razones principales: en primer lugar, es necesario que el injerto soporte la presión hemodinámica a la que se vea sometido sin sufrir fallas; en segundo lugar, las tensiones son un fuerte estímulo para el control mecanobiológico de la función celular y extracelular [17]. Además, la interacción del andamio con el tejido vivo adyacente al que se lo conecta debe también ser considerada, ya que un desfasaje entre las propiedades mecánicas de los vasos circundantes y el injerto pueden derivar en un remodelado patológico de las arterias vecinas [18, 19].

En vista de estas cuestiones y con el objetivo de contribuir al diseño de injertos vasculares de este tipo, tanto a nivel microscópico como macroscópico, surge la necesidad de modelar el comportamiento mecánico de los materiales involucrados. Con este fin se emplea el modelado constitutivo de materiales, pudiendo realizarse desde varios enfoques. Los distintos modelos pueden clasificarse según la complejidad con que abordan el problema microscópico. Los modelos fenomenológicos se restringen a relacionar la tensión y deformación macroscópicas sin introducir de manera alguna información sobre la microestructura. Son los más usuales en la literatura pero la simplicidad de no requerir información de entrada sobre la microestructura se traduce en la limitación de no poder relacionar propiedades microscópicas con la respuesta macroscópica. Se trata, en última instancia, de un ajuste matemático sobre datos experimentales. Los modelos multiescala, en cambio, proponen predecir la respuesta constitutiva macroscópica a partir de la microestructura subyacente, resultando especialmente útil en materiales heterogéneos donde el comportamiento mecánico depende del tamaño, forma, propiedades, y distribución espacial de los constituyentes microestructurales. Este tipo de modelos surgen dada la dificultad de modelar un componente teniendo en consideración características geométricas de dimensiones muy disímiles entre sí[1]. La solución que se plantea en estos modelos es la separación de escalas, donde el componente macroscópico se trata como un continuo pero a cada punto material se le asigna alguna representación de la microestructura. Si bien lo más usual es la implementación de dos escalas (macro- y micro-), esta separación

[1] En el caso de las matrices electrohiladas, un modelado exhaustivo debe tener en cuenta a escala macroscópica las dimensiones del componente (cm) y a escala microscópica las dimensiones de las nanofibras ($\mu\mathrm{m}$)

puede realizarse recursivamente, obteniendo modelos con tres (macro-, meso- y micro-) o más escalas. Una gran ventaja reside en poder separar el tipo de modelado empleado en diferentes escalas, pudiendo plantear, por ejemplo, un modelo macroscópico continuo discretizado por el método de elementos finitos con un RVE de naturaleza discreta formado por un conjunto de fibras interconectadas [20]. En el caso más completo, se busca calcular una respuesta macroscópica en cada punto de integración de la escala macroscópica a partir de un dominio detallado de la microestructura asociada a ese punto. El dominio microscópico debe ser lo suficientemente representativo de la microestructura real del material, y se lo denomina Elemento de Volumen Representativo (*Representative Volume Element*: RVE). Las técnicas de este tipo poseen las siguientes ventajas: (i) no requieren suposiciones de carácter constitutivo sobre la macroescala, (ii) permiten tratar grandes deformaciones y rotaciones tanto en la escala macro como micro, (iii) son apropiadas para respuestas no lineales de carácter físico, geométrico, y temporal, (iv) habilitan incorporar detalles microestructurales en el análisis macroscópico, y (v) dado que las relaciones obtenidas se derivan de un entendimiento de los mecanismos microscópicos que tienen lugar en el material durante la deformación, permiten una comprensión de las relaciones constitutivas entre las distintas escalas [20]. Esta característica es muy prometedora en ingeniería de tejidos, ya que se pretende diseñar una microestructura en función de un comportamiento macroscópico buscado.

Como herramienta complementaria, la simulación hemodinámica permite evaluar las condiciones de trabajo del segmento original a ser reemplazado y predecir el desempeño mecánico del injerto una vez implantado [21, 22]. Dentro de estos modelos se destacan los modelos heterodimensionales que acoplan submodelos 0D, 1D y 3D y que han emergido como una novedosa herramienta para el modelado del árbol arterial completo [23, 24]. Estos modelos heterodimensionales pueden analizarse como la interacción de dos o tres submodelos. El modelo 1D trata con ecuaciones reducidas de flujo en tubos compliantes, al cual se le acoplan modelos de parámetros concentrados 0D que representan los lechos periféricos y otras singularidades del sistema arterial. Estos modelos son los responsables de considerar las interacciones a nivel sistémico y consecuentemente, de proveer apropiadas condiciones de borde a un modelo detallado y complejo (3D) de un distrito vascular específico. Luego, el modelo 3D aborda la interacción fluido-estructura entre el flujo sanguíneo y la pared arterial con el mayor grado de detalle posible. De esta forma, el modelo 1D da cuenta del comportamiento sistémico integrado del árbol arterial, mientras el modelo local 3D produce gran cantidad de información mecánica en una zona puntual del

mismo. Así, es posible alcanzar un alto grado de realismo en las simulaciones sorteando la imposibilidad actual de manejar modelos completos tridimensionales de todo el sistema vascular [25]. Es necesario aquí resaltar la potencialidad de los modelos heterodimensionales para proveer la infraestructura que hace posible la implementación de los ensayos virtuales (*in silico*) para evaluar el comportamiento de injertos, permitiendo considerar de manera económica la amplia gama de condiciones prevalecientes a lo largo del árbol arterial. Así es posible adaptar la respuesta mecánica del material de acuerdo a la zona específica en la que este será implantado. Estas condiciones marcan, por un lado, la necesidad de adaptar en cada caso la composición y propiedades de los implantes substitutos y, por otro lado, evidencian la imposibilidad de realizar ensayos que puedan abarcar todas las situaciones mencionadas [26]. La simulación computacional resulta, por lo tanto, una herramienta excelente para asistir el diseño de matrices nanofibrosas para ingeniería de tejidos.

1.2. Objetivos

Este trabajo tiene como objetivo general el desarrollo de modelos constitutivos multiescala para matrices nanofibrosas electrohiladas con aplicaciones en ingeniería de tejidos que lleven en consideración las características microscópicas del material y su influencia sobre el comportamiento mecánico macroscópico con el fin de su aplicación en un ciclo de modelado fabricación-experimentación para ingeniería de tejidos. Para esto se plantean los siguientes objetivos específicos:

- Derivar una relación constitutiva multiescala cuyos parámetros característicos sean propiedades microestructurales con sentido físico.
- Obtener los parámetros constitutivos microscópicos a partir de datos experimentales macroscópicos de ensayos mecánicos locales para matrices poliméricas para ingeniería de tejidos.
- Obtener un RVE para matrices nanofibrosas a partir del análisis de la microestructura en base a micrografías y detalles técnicos del método de fabricación.
- Evaluar mediante simulación computacional el comportamiento de las matrices bajo condiciones de carga hemodinámicas mediante un modelo de membrana discretizado mediante el método de elementos finitos.

1.3. Organización

La tesis se organiza en 4 capítulos:

- El capítulo 1 (actual) introduce las bases para el trabajo siguiente, consistiendo en un marco teórico donde se presenta una descripción general de los tejidos arteriales, sus afecciones más comunes y los tratamientos correspondientes, donde surge la importancia de los injertos vasculares. A continuación se realiza una descripción de los distintos tipos de injertos vasculares utilizados, haciendo hincapié en los injertos electrohilados de ingeniería de tejidos, su proceso de fabricación y su caracterización morfológica y mecánica. También se establecen las bases del modelado multiescala y su importancia para el diseño de materiales biomiméticos en la ingeniería de tejidos. Finalmente, a partir de los conceptos revisados, se formulan las hipótesis de trabajo a seguir durante el transcurso del trabajo.

- En el capítulo 2 se presenta un modelo multiescala para matrices electrohiladas isotrópicas donde se utilizó un enfoque estadístico sobre la distribución de enrulamientos de las fibras. El modelo se validó mediante comparación con datos experimentales de ensayos de inflado sobre injertos electrohilados. También se estudian los efectos que variaciones microestructurales provocan sobre la respuesta macroscópica con el objetivo de mejorar la biomímesis desde la perspectiva biomecánica. Finalmente se utilizó el modelo como herramienta de diseño, optimizando los parámetros de un injerto electrohilado capaz de imitar la respuesta en presión-diámetro de una arteria intracranial humana.

- En el capítulo 3 se detalla un método para generar elementos de volumen representativos de geometría fibrosa que reproducen en detalle y con fidelidad los aspectos más importantes de la microestructura de las matrices electrohiladas. Se estudió la variabilidad estadística de las mallas virtuales generadas en función de su tamaño para evaluar su representatividad. También se realizó, mediante el análisis de mallas virtuales, una estimación sobre la densidad superficial de intersecciones, siendo una variable de importancia para el comportamiento mecánico y de difícil medición experimental. Por último, se muestra la versatilidad del método para obtener geometrías que reproducen fielmente las microestructuras típicas de matrices electrohiladas.

- En el capítulo 4 se presenta un modelo micromecánico para mallas fibrosas interco-

nectadas con el objetivo de aplicarse a los RVE generados en el capítulo 3, conformando así un modelo microscópico de alta fidelidad para matrices electrohiladas. La descripción cinemática de la malla admite grandes desplazamientos y rotaciones de las fibras, mientras que para las fibras se admite la posibilidad de deformación plástica y de rotura al ser traccionadas. Luego, el modelo se valida mediante comparación con ensayos experimentales de tracción uniaxial donde se consigue buen acuerdo bajo geometría y parámetros realistas. Finalmente, se estudia la evolución de la microestructura de la malla bajo tracción uniaxial.

1.4. Marco Teórico

1.4.1. Tejidos arteriales

Las enfermedades cardiovasculares son la principal causa de muerte a nivel mundial. Cada año se pierden 17 millones de vidas por causa de estas enfermedades, lo que representa un 29 % de la totalidad de las muertes globales. En particular, las enfermedades coronarias representan un 50 % de las causas de muerte por enfermedades cardiovasculares [8].

Los tejidos arteriales son un subgrupo de los tejidos vasculares, que incluyen también a los capilares y las venas. Dado que las afecciones arteriales representan un mayor riesgo en términos de salud y poseen mayor impacto e incidencia que las enfermedades vasculares venosas y capilares, considerables esfuerzos se han aplicado en el estudio y la búsqueda de estrategias para su reemplazo y regeneración.

Dado que, en gran medida, las afecciones de la pared arterial están en relación a sus propiedades mecánicas, a lo que se suma que las probabilidades de éxito en las intervenciones aumentan cuando los materiales de reemplazo muestran propiedades biomiméticas con los tejidos fisiológicos, es necesario caracterizar el comportamiento mecánico y constitutivo de dichos tejidos. En un sentido descriptivo macroscópico, la estructura de la pared arterial se compone de tres capas distintas: la intima, la media, y la adventicia (figura 1.1). La intima es la capa interna y se compone de una capa de células endoteliales soportada por una fina membrana basal y una capa subendotelial. Cumple la función de controlar la transferencia molecular desde el flujo sanguíneo hacia el interior de la pared y evitar la trombogénesis, además de estar involucrada en el mantenimiento de la homeostasis, la regulación del tono muscular, y la regulación inmunogénica e inflamatoria. La media, como su nombre lo indica, es la capa del medio de la pared arterial, y está compuesta por células de músculo liso orientadas circunferencialmente en conjunto con un arreglo doble helicoidal de fibras de colágeno embebidas en una matriz de elastina. La adventicia es la capa externa de la arteria y consiste principalmente de fibroblastos y fibrocitos (células que sintetizan colágeno y elastina), sustancia fundamental y haces de fibrillas de colágeno conformando un tejido fibroso [27].

Resulta importante resaltar que, desde el punto de vista mecánico, los elementos constituyentes relevantes son las células de músculo liso, la matriz de elastina y las fibras de colágeno. Las células de músculo liso poseen capacidad contráctil y son las responsables, por lo tanto, de regular el tamaño del vaso. La elastina es altamente distensible y le confie-

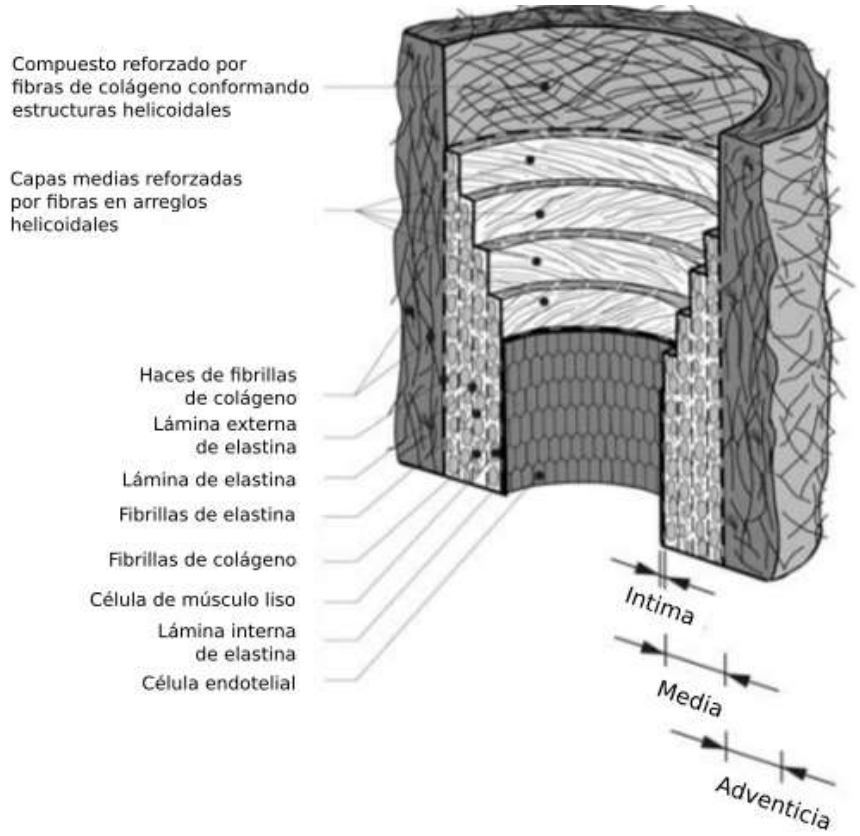


Figura 1.1: Diagrama de la estructura en capas de las arterias, indicando los principales elementos constituyentes de cada una de las tres capas: intima, media y adventicia. Figura adaptada de [28]

re a las arterias una capacidad de deformación elástica elevada a bajas presiones, mientras que las fibras de colágeno, que se encuentran enruladas en reposo, devienen rectas y mecánicamente activas a altas presiones, protegiendo y reforzando el vaso [29, 30].

Hoy en día, se concentra el mayor esfuerzo sobre el estudio de las arterias musculares (que poseen un diámetro menor a $6\,\mathrm{mm}$), haciendo especial foco en las arterias coronarias, dado que los tratamientos actuales frente a enfermedades coronarias presentan serios inconvenientes.

Las principales causas de enfermedad coronaria son la trombosis y aterosclerosis. La trombosis es la obstrucción del flujo sanguíneo en algún sector del sistema circulatorio debido a la formación de un coágulo en el interior de un vaso sanguíneo. La formación de un coágulo toma lugar normalmente como respuesta natural frente a una lesión del endotelio, aunque puede ocurrir también frente a factores hereditarios o debido a enfermedades de la sangre. Cuando un coágulo se libera en el torrente sanguíneo se lo denomina émbolo, por lo cual es también usual la denominación tromboembolismo cuando se produce

la oclusión del flujo. Cuando la trombosis ocurre en arterias (trombosis arterial) se ve afectada la irrigación sanguínea de los tejidos, pudiendo derivar en isquemias y necrosis [31]. La aterosclerosis, en cambio, consiste en el crecimiento de placas de ateroma dentro de la capa intima de la pared arterial, compuestas de grasa, colesterol, calcio y otras sustancias provenientes de la sangre. El material acumulado engrosa la pared arterial que se proyecta hacia el interior del vaso, reduciendo la luz arterial (estenosis) y restringiendo el flujo sanguíneo. En casos severos puede devenir en enfermedad coronaria, infarto, enfermedad vascular periférica o problemas renales, dependiendo de la arteria afectada. Aunque generalmente comienza a edades tempranas, prácticamente todas las personas sufren algún grado de aterosclerosis a partir de los 65 años de edad, tratándose además, de la principal causa de muerte y morbilidad en el mundo desarrollado [32]. Evidentemente, estas afecciones no son mutuamente excluyentes y pueden ocurrir en simultáneo con efectos sinérgicos: la estenosis generada por la acumulación de ateroma resulta un lugar propicio para el alojamiento de un émbolo trombótico, derivando en un tromboembolismo (ver figura 1.2). El infarto de miocardio ocurre cuando disminuye bruscamente el flujo coronario después de una oclusión trombótica de una arteria coronaria, ya estrechada por aterosclerosis (figura 1.2).

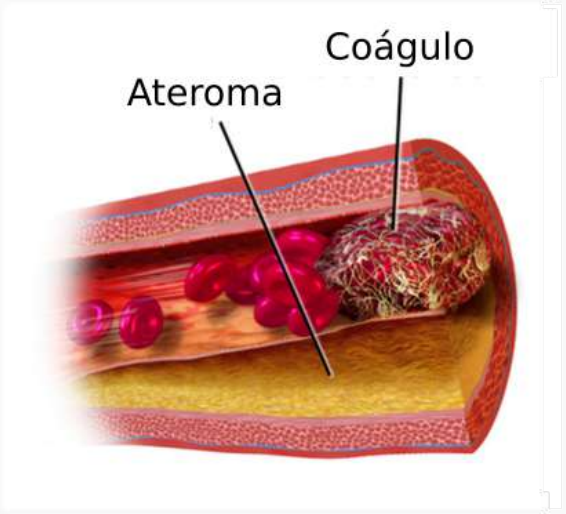


Figura 1.2: El diagrama muestra el bloqueo del flujo sanguíneo provocado por un coágulo suelto en el torrente sanguíneo (émbolo trombótico) alojado en la reducción del lumen (estenosis) provocada por la acumulación de materia grasa (placa de ateroma) en la capa intima de la pared arterial.

Si bien el principal foco de ataque contra estas enfermedades es la prevención, en los casos en que la oclusión o estenosis son severos existen una serie de tratamientos que se pueden realizar. Cuando es posible, se realiza un tratamiento de trombólisis con el fin de

reestablecer de forma inmediata la permeabilidad de la arteria coronaria, mediante la administración intravenosa de agentes trombolíticos de activación tisular del plasminógeno (tPA), estreptoquinasa y complejo anisoilado del activador del plasminógeno y la estreptoquinasa (APSAC). Estos agentes facilitan la conversión del plasminógeno en plasmina, que posteriormente disuelve los trombos de fibrina. En cambio, cuando los medicamentos trombolíticos se encuentran contraindicados, se requiere de intervenciones invasivas de revascularización como la angioplastía o la técnica de *bypass*.

La angioplastía busca recuperar la luz arterial en arterias con estenosis causadas, típicamente, por aterosclerosis. Es un tratamiento invasivo endovascular que consiste en la introducción remota, a través de las arterias femoral o radial, de una guía flexible con un balón inflable dentro de la arteria coronaria afectada. El balón se infla repetidas veces en la zona de la estenosis hasta que la obstrucción desaparece o disminuye. Bajo ciertas condiciones se puede colocar, en el mismo procedimiento, un conducto expansible de metal denominado *stent* que refuerza mecánicamente la zona afectada para evitar estenosis residual y, al mismo tiempo, reducir la probabilidad de futuras reestenosis [33, 34]. Otra posible solución es la cirugía de derivación arterial coronaria (o *bypass* coronario), que consiste en reestablecer el flujo sanguíneo en una arteria obstruida por medio de una conexión artificial por medio de un injerto vascular entre la aorta y la arteria coronaria afectada.

1.4.2. Tipos de injertos vasculares

Como se mencionó, un *bypass* coronario requiere de un injerto vascular. Para este procedimiento se emplean injertos a partir de venas o arterias autólogas, homólogas y heterólogas, e injertos sintéticos de similares dimensiones. De estos injertos, los autólogos siguen siendo hoy en día el *gold standard*, tratándose de vasos sanguíneos del propio paciente que son extraídos de zonas no comprometidas. En particular, la vena safena mayor y la arteria torácica interna (también denominada arteria mamaria interna) son los injertos que han presentado mejores resultados en el tiempo. La vena safena mayor presenta una tasa de permeabilidad (grado de apertura) del 80-90 % luego de un año de implantada. No obstante, a largo tiempo es propensa a sufrir el desarrollo de aterosclerosis y el 50 % de los injertos se cierran luego de 10 años de la operación. La arteria torácica interna presenta una tasa de permeabilidad de 90 % a los 10 años después de la operación. Las causas de este mejor rendimiento no son completamente conocidas. Sin embargo, la vena safena sigue siendo el injerto más elegido por los cirujanos. En los casos que la vena safena y

la arteria torácica interna no se encuentran disponibles para su utilización como injertos, otros vasos sanguíneos tales como la arteria radial, la arteria gastro-omental derecha, la arteria epigástrica inferior, la vena safena corta y venas de las extremidades superiores, son utilizados [35].

Si bien los injertos autólogos presentan un buen desempeño, estos resultan inadecuados o inaccesibles en aproximadamente un tercio de los pacientes, requiriendo la utilización de alternativas como, por ejemplo, el uso de materiales sintéticos [36, 37]. Los injertos sintéticos aprobados para su uso en *bypass* coronario son bioestables y presentan una alta rigidez, siendo los más comunes el Dacron® y el Goretex® (figura 1.3). Las tubuladuras de PET (polietileno tereftalato), patentadas como Dacron®, se forman mediante el tejido de múltiples filamentos de poliéster y poseen alta cristalinidad, alto módulo elástico y alta resistencia a la tracción. Estos injertos se utilizan con éxito para *bypass* de aorta y en la revascularización de injertos de gran diámetro periféricos. Por otra parte, los injertos de ePTFE (politetrafluoroetileno expandido), conocidos comercialmente como Goretex® también presentan alta cristalinidad y alta rigidez, aunque menor que la del Dacron®. El Goretex® se usa con excelentes resultados en injertos para las extremidades inferiores de diámetros internos entre 7-9 mm. Respecto del desempeño de estos injertos como reemplazos para *bypass* coronario, el injerto de ePTFE posee una tasa de permeabilidad media del 14 % a 45 meses de la operación y el de PET resulta en vasos abiertos luego de 17 meses de implantados, no habiendo resultados reportados a mayor tiempo de seguimiento.

Sin embargo, tanto los injertos de Dacron® como de Goretex® fallan en la revascularización de arterias de pequeño diámetro ($<6\,\mathrm{mm}$) [38, 39], debido principalmente a la trombogenicidad de la superficie sintética y al desacuerdo en las propiedades mecánicas entre injerto y tejido nativo en la zona de anastomosis[2] [36, 40, 41]. En el campo de la fisiología cardiovascular, la propiedad mecánica en juego se denomina elastancia o compliancia y cumple un rol fundamental ya que determina la respuesta en presión-diámetro a nivel de componente. Es necesario, entonces, que la compliancia injerto pueda asemejar a la del vaso nativo. Dado que los poliuretanos tienen una naturaleza más distensible y elastomérica que el PET y el ePTFE, son candidatos excelentes para reducir los problemas asociados a la rigidez de los injertos y el desacuerdo mecánico con los tejidos nativos [36, 42]. Si bien se han reportado algunos resultados preliminares con injertos poliuretánicos, no existen datos de seguimiento a tiempo prolongados desde la implantación.

Ninguno de los injertos convencionales (sintéticos, autólogos, provenientes de un do-

[2] Anastomosis: zona de unión o costura entre dos vasos

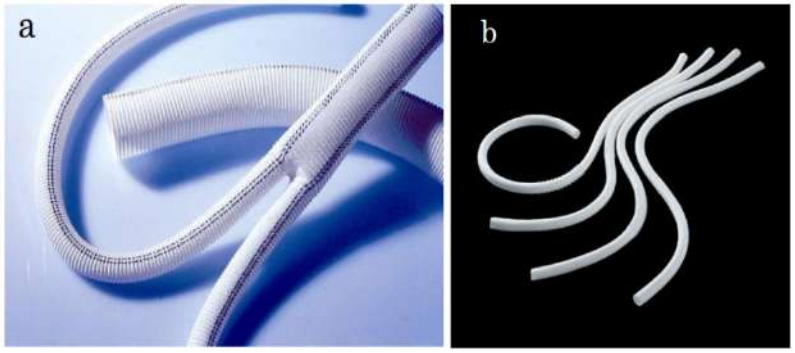


Figura 1.3: Fotografías de injertos vasculares sintéticos comerciales de a) Dacron® y b) Goretex®.

nante o de origen animal) ofrece potencial de regeneración y, más aún, todos están asociados con diferentes niveles de trombogenicidad, reestenosis y susceptibilidad a infecciones. Además, las intervenciones quirúrgicas tratan solamente la manifestación de la aterosclerosis, sin tratar las causas de la condición, por lo que los síntomas tienden a reaparecer y los pacientes con frecuencia requieren una nueva intervención [8, 43].

En este contexto entra la Ingeniería de Tejidos, entendida como el uso combinado de métodos de ingeniería y de las ciencias biológicas para el desarrollo de reemplazos que restauren, mantengan o mejoren la función de los tejidos vivos [44]. Un objetivo de la ingeniería de tejidos es el diseño de injertos vasculares vivos, capaces de responder a estímulos y con propiedades similares a los tejidos nativos [45]. En este sentido, el enfoque de la cirugía vascular emergente ha evolucionado de ‘reemplazar’ a ‘regenerar’ el tejido vascular.

Idealmente un injerto vascular debe carecer de características trombogénicas, tóxicas, cancerígenas e infecciosas, a la vez que debe poseer una adecuada compliancia, facilidad de manejo quirúrgico para su implantación , facilidad de sutura, facilidad de fabricación, disponibilidad en diversos tamaños, capacidad para liberar localmente agentes terapéuticos, capacidad funcional biológica y potencial de regeneración de tejidos. La relación estructura-función de los tejidos vasculares coronarios plantea un criterio de diseño exigente para los sustratos requeridos. Por lo tanto, a la hora de diseñar un injerto vascular hay que tener en cuenta su estructura y funciones tanto a escala macroscópica como microscópica.

La ingeniería de tejidos vasculares plantea dos enfoques diferenciados: los injertos de láminas celulares autoensambladas y los injertos regenerados mediante el uso de andamios (*scaffolds*).

El enfoque de láminas celulares autoensambladas utiliza células humanas exclusiva-

mente para fabricar vasos sanguíneos a partir de células autólogas sin el uso de sustratos externos al paciente. En un biorreactor se cultivan células de músculo liso y fibroblastos *in vitro* propias del paciente, luego se retiran de las placas y se ensamblan sobre un mandril del diámetro deseado, colocando primero células de músculo liso, siguiendo con fibroblastos y culminando con un recubrimiento de células endoteliales. Estos injertos vasculares resisten presiones superiores a los $2000\,\mathrm{mmHg}$[3], resistencia a la sutura adecuada y un endotelio funcional. Incluso estudios *in vivo* demostraron que soportan las condiciones de flujo fisiológico [46, 47]. Si bien los resultados obtenidos son exitosos, las láminas celulares requieren de un tiempo considerable para su fabricación, volviendo a este método poco prometedor para su traslado a la práctica clínica, en especial para los casos en que el injerto se necesita con cierta urgencia. Más aún, existe la dificultad de obtener células funcionales capaces de regenerar el tejido vascular en un biorreactor, lo cual resulta una tarea dificultosa en pacientes de edad avanzada [48].

Como alternativa, los injertos vasculares reconstruidos a partir de matrices soporte, o andamios, surgen como continuación natural en la investigación de injertos vasculares sintéticos, donde se adiciona el procedimiento *in vitro* de infiltración y maduración de células vasculares previamente a la implantación. Un andamio es una estructura tridimensional que provee el soporte mecánico y un ambiente propicio para el desarrollo y crecimiento de un tejido, al tiempo que facilita las funciones celulares como adhesión, diferenciación, migración, proliferación y secreción de los componentes de la matriz extracelular [49, 50]. Para que la infiltración y maduración celular sea exitosa, los andamios deben poseer estructuras altamente porosas e interconectadas, de modo de permitir el transporte de sustancias así como la migración celular [51, 52]. Además, para lograr la proliferación de tejido celular se utilizan biorreactores que simulen condiciones hemodinámicas fisiológicas, incluyendo las señales bioquímicas y biomecánicas que regulen correctamente el desarrollo tisular [53].

Las primeras alternativas para la obtención de injertos mediante esta metodología se basaron en andamios constituidos de materiales naturales propios de los tejidos vasculares como el colágeno y presembrados con células vasculares. Sin embargo presentaron propiedades mecánicas inferiores a las requeridas y, contrariamente al objetivo inicial, necesitaron de refuerzos sintéticos para su uso clínico, generando nuevas complicaciones por rechazo o desacuerdo en compliancia [54–56]. La utilización de biorreactores para favorecer la regeneración del tejido vascular con el soporte mecánico del andamio mejora

[3]Como referencia, la resistencia de la vena safena se estima en $1700\,\mathrm{mmHg}$.

sustancialmente las propiedades mecánicas de estos injertos, logrando pruebas con cierto grado de éxito en animales [57]. Otra opción es la utilización de andamios sintéticos, siendo los más empleados los poliésteres biodegradables compuestos de glicolida y lactida, sus copolímeros (como el ácido poli-L-láctido, el ácido poliglicólico y la policaprolaptona) y los poliuretanos [58]. En comparación con los materiales naturales, los sintéticos presentan grandes ventajas respecto de su amplia disponibilidad y bajo costo, además de permitir mayor control en etapa de producción tanto a nivel de propiedades mecánicas como de porosidad y microestructura, lo que los vuelve candidatos muy atractivos para su fabricación para uso clínico [49, 50, 59]. Como desventaja, los materiales sintéticos sufren de una baja bioactividad respecto de favorecer la implantación y proliferación natural, además de la importancia de tomar en consideración el efecto de la degradación del andamio sobre el tejido celular [60, 61]. Por otro lado, se exploró el uso de andamios a partir de matrices descelularizadas provenientes de donantes o animales (figura 1.4). Se trata de tejidos naturales a los que se los despoja, mediante el uso de diferentes técnicas, de las células pero manteniendo las proteínas de la matriz estructural, como colágeno y elastina, que son predominantemente no inmunogénicas. De esta manera se obtiene un andamio que permitiría ser recelularizado con células del paciente reduciendo drásticamente el riesgo de rechazo inmune, pero aún así existe el riesgo de transmitir patógenos del animal o donante al paciente. Además, las propiedades biomecánicas de la matriz son variables debido a la variabilidad de especies donantes, así como también las distintas edades o sexo de la misma, reduciendo la repetibilidad en la producción con lo cual se ve afectado el resultado clínico [8, 29].

En todo caso, si bien este tipo de injertos ha mostrado resultados interesantes y alentadores, poseen la importante limitación de necesitar un tiempo de cultivo *in vitro* previo a la implantación, y más aún, existe la dificultad de obtener células funcionales capaces de regenerar el tejido vascular en un biorreactor, lo cual resulta una tarea difícil en pacientes de edad avanzada [48]. Hasta la fecha tanto las alternativas sintéticas como las diseñadas por ingeniería de tejidos no consiguen igualar ni mejorar a los injertos autólogos para la cirugía vascular de pequeños vasos. En los últimos 30 años poco ha cambiado en relación a los injertos sintéticos, y la obtención de un injerto vascular de pequeño diámetro ($< 6\,\mathrm{mm}$) con un comportamiento apropiado y permanente aún representa un gran desafío. El desarrollo de un conducto para operaciones de *bypass* coronario que pueda estar inmediatamente disponible, sin tiempos prolongados de cultivos *in vitro*, con las propiedades mecánicas y biológicas necesarias es el centro de las investigaciones actuales

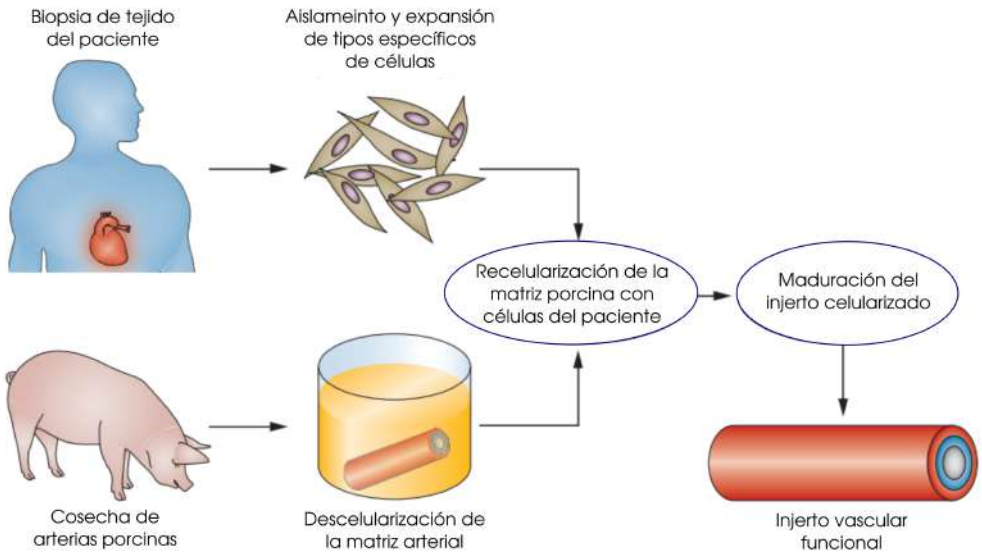


Figura 1.4: Método de obtención de un injerto diseñado por ingeniería de tejidos a partir de un andamio descelularizado. Se obtienen células vasculares del paciente y se cultivan *in vitro*. Al mismo tiempo se extrae una arteria de una fuente alogénica, heterogénica o xenogénica, y se la descelulariza dejando solamente la matriz soporte. Luego se recelulariza el andamio con las células del paciente y se lo madura en un biorreactor, obteniendo un injerto vivo funcional. Imagen adaptada de [8].

en el campo de la ingeniería de tejidos vasculares.

1.4.3. Matrices electrohiladas

Como se mencionó en la sección previa, un requisito fundamental para un *scaffold* de ingeniería de tejidos es que presente una microestructura altamente porosa e interconectada, de modo de permitir el transporte de sustancias así como la migración, adhesión y proliferación celular. Consecuentemente, el número de publicaciones científicas relacionadas con andamios para aplicaciones biomédicas se incrementó notablemente en los últimos años, hecho que revela un alto y sostenido interés en el diseño y preparación de matrices porosas[62]. Teniendo en cuenta los requisitos indispensables que debe reunir una matriz extracelular artificial, la tecnología de procesamiento para su producción debe poseer un control importante de las propiedades macro y microestructurales. El comportamiento viscoso de los polímeros por encima de su temperatura de transición vítrea o temperatura de fusión y su solubilidad en diferentes solventes orgánicos, determina la aplicación de una amplia variedad de técnicas para preparar matrices poliméricas porosas a partir de polímeros sintéticos y naturales, en algunos casos se realizan compuestos con materiales cerámicos [63–65]. No existe una metodología única para crear matrices po-

rosas, la elección de la técnica más apropiada resulta entonces crítica y depende de cada material polimérico y la aplicación específica [66].

La microestructura, porosidad, y topografía del injerto poroso son factores fundamentales para su desempeño exitoso. La producción de injertos vasculares se puede abordar mediante la utilización de diferentes técnicas. P.M. Crapo y colaboradores fabricaron injertos vasculares de poli(ácido láctico-co-glicólico) (PLGA) y poli(glicerol sebacato) (PGS) mediante la técnica de evaporación de solvente y disolución de partículas [67, 68]. El grupo de D.A. Vorp desarrolló injertos de poli(éster uretano)urea (PEUU) obtenidos mediante la técnica de separación de fases inducida por temperatura (TIPS) [69]. Otros grupos utilizaron polímeros naturales, C.E. Ghezzi y colaboradores produjeron injertos densos de colágeno simplemente al envolver circunferencialmente láminas de gel de colágeno densas, comprimidas plásticamente, alrededor de un soporte cilíndrico [70]. S. Liu y colaboradores desarrollaron injertos bicapa reforzados de fibrina de seda con heparina [71].

Si bien se han logrado injertos con propiedades interesantes mediante las técnicas mencionadas, la tecnología de electrohilado resulta una técnica más prometedora para la producción de injertos vasculares. Esta es una técnica versátil, que permite la producción de fibras de nano/microescala que presentan un gran potencial para imitar el microambiente fibroso de la matriz extracelular natural presente en las arterias [13]. El electrohilado ofrece la posibilidad de ajustar finamente las propiedades mecánicas durante la fabricación, controlando la microarquitectura porosa, tamaño y la orientación de las fibras, logrando comportamientos anisotrópicos como los observados en los vasos sanguíneos. Además, la posibilidad de usar un colector rotatorio de pequeño diámetro, resulta en la obtención de una tubuladura sin costura ideal para aplicaciones vasculares. Todas estas ventajas, en conjunto con las ya mencionadas en la sección previa, posicionan a la técnica de electrohilado como una técnica ideal para la producción de injertos vasculares de pequeño diámetro.

Proceso de electrohilado

La tecnología de electrohilado constituye uno de los métodos de procesamiento de vanguardia que presenta mayores ventajas para la producción de nanofibras. La técnica tiene la habilidad única de producir nanofibras de diferentes materiales y geometrías, bajo costo, relativamente alta velocidad de producción y simplicidad en el diseño del equipamiento. En los últimos años, se han electrohilado numerosos tipos de materiales que

incluyen prácticamente todos los polímeros sintéticos y naturales que sean solubles o puedan fundirse, y nanocompuestos, para obtener fibras continuas de unos pocos nanometros hasta algunos micrones que generan una matriz hilada no tejida altamente porosa [12].

Aunque la técnica de electrohilado constituye una vía versátil para la producción de nanofibras, el proceso es sumamente complejo y depende de numerosos parámetros. El diseño experimental básico para electrohilado de soluciones consta de cuatro componentes (figura 1.5): un reservorio de solución o material fundido, una bomba de infusión que permite suministrar un flujo constante y controlado de solución, una fuente de alta tensión y un sistema colector sobre el que se deposita el material electrohilado. Al aplicar una tensión de $5\text{-}30\,\mathrm{kV}$, la solución polimérica se electrifica fuertemente. Se inducen cargas eléctricas que se distribuyen sobre la superficie de la gota de solución polimérica que pende de una boquilla. La gota experimenta un conjunto de fuerzas: fuerza de repulsión eléctrica entre las cargas inducidas, fuerza electrostática producto del campo eléctrico externo generado al aplicar la tensión, fuerza gravitatoria, fuerzas viscoelásticas que dependen del polímero y solvente, y la tensión superficial que se opone al estiramiento y afinamiento de la gota. Bajo la acción de estas interacciones, la gota se distorsiona en forma cónica, fenómeno conocido como cono de Taylor. En estas condiciones el balance de fuerzas llega a un equilibro. Luego, cuando las fuerzas electrostáticas repulsivas superan la tensión superficial del polímero, se produce una situación inestable que provoca la expulsión de un microchorro líquido cargado desde la boquilla del capilar. Este microchorro electrizado sufre estiramiento y movimientos tipo látigo, dando lugar a la formación de hilos largos y delgados. A medida que el chorro líquido se deforma continuamente y se evapora el solvente (o solidifica el fundido), las cargas superficiales aumentan conduciendo a una disminución drástica del diámetro de las fibras. Los entrecruzamientos físicos de las cadenas poliméricas permiten dar continuidad al microchorro, formando fibras que se depositan en el sistema colector que, por otra parte, se encuentra conectado eléctricamente a tierra [72, 73].

Microestructura

En el proceso de electrohilado el solvente se evapora casi completamente en la trayectoria que recorre el microchorro entre los electrodos durante el tiempo de proyección. Las nanofibras así formadas se depositan de manera continua sobre el colector, ya sea plano o rotatorio, debido a la atracción eléctrica por el campo inducido. La estructura resultante

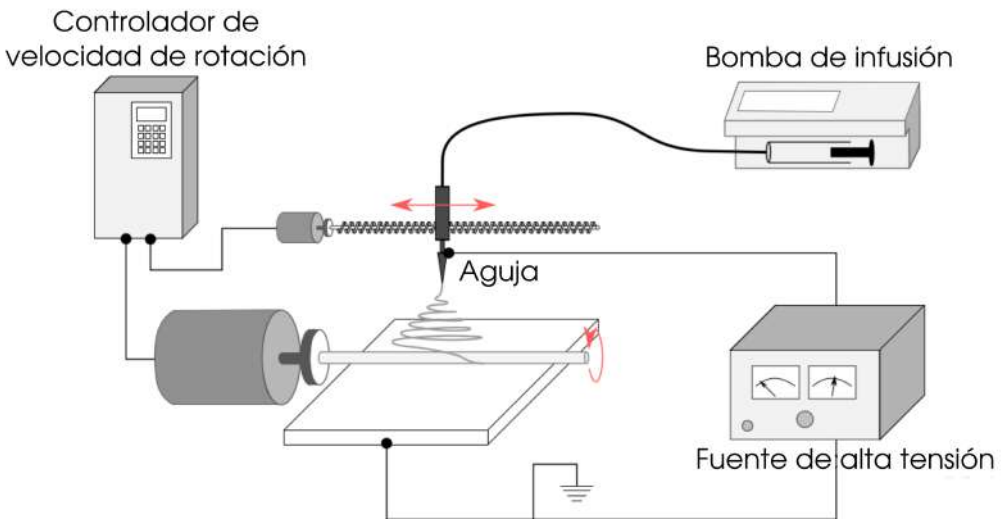


Figura 1.5: Esquema básico de los componentes necesarios para el electrohilado de soluciones sobre un colector cilíndrico.

es la de una malla de nanofibras muy largas, onduladas, no tejidas, con uniones de tipo soldadura en los puntos de contacto donde ha quedado solvente residual. Además, dado que las fibras se van superponiendo unas a otras, se obtiene una estructura de capas en la dirección normal al plano colector (dirección radial en el caso de un colector cilíndrico). Esta morfología microscópica se asemeja a la de los tejidos arteriales (figura 1.6), dando lugar a la idea de que las matrices electrohiladas son buenos candidatos para su utilización como andamios vasculares.

Se puede lograr cierto nivel de control sobre la distribución de orientaciones de las nanofibras variando las velocidades angular y axial del mandril colector (figura 1.7) [78], mientras que otros aspectos de la geometría de la malla pueden controlarse mediante otros parámetros del proceso como el caudal de infusión, la distancia entre la boquilla y el colector, la tensión aplicada, la introducción de terceras fases, etc. De esta manera se obtienen mallas con nanofibras de distinto diámetro y curvatura, así como también se puede variar la densidad de uniones entre fibras y el grado de alineamiento sobre una dirección preferencial.

Es relevante señalar que se ha observado experimentalmente bajo carga, las grandes rotaciones y deformaciones sufridas por las fibras elastoméricas permiten a estos *scaffolds* de microestructura nanofibrosa soportar el mismo nivel de deformaciones que experimentan los tejidos vasculares que pretenden reemplazar [14–16].

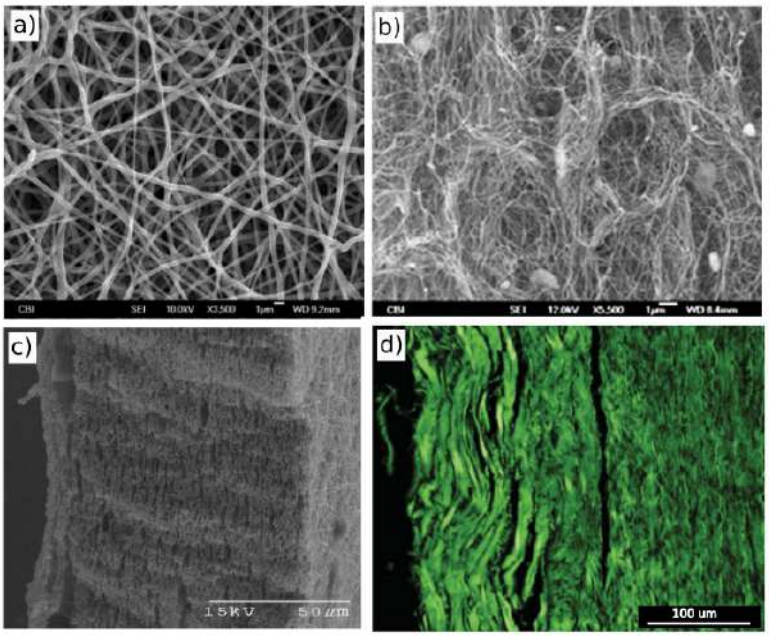


Figura 1.6: Comparación visual entre la microestructura de la matriz extracelular arterial y las matrices producidas mediante la técnica de electrohilado, incluyendo la morfología plana y la estructura de capas: (a) Imagen SEM de la capa superior de un andamio de PEUU electrohilado (adaptada de [74]), (b) Imagen SEM de la superficie de la capa adventicia descelularizada (adaptada de [75]), (c) Imagen SEM transversal de PEUU electrohilado mostrando la estructura de capas (adaptada de [76]) y (d) Imagen SHG (*second-harmonic generation*) transversal de las capas adventicia y media (de izquierda a derecha) mostrando su estructura de capas (adaptada de [77]).

1.4.4. Modelado multiescala

Para alcanzar las demandas mecánicas y funcionales necesarias para el éxito de un andamio para ingeniería de tejidos, se necesitan modelos y simulaciones que otorguen un mayor entendimiento de los procesos microestructurales que ocurren durante la deformación y su relación con la respuesta macroscópica observada. El modelado constitutivo de un material es el proceso de análisis mediante el cual se establece un modelo que representa algunos, sino todos, los aspectos de importancia involucrados, mientras que la simulación es el proceso que utiliza el modelo establecido para determinar la respuesta del material bajo condiciones específicas de carga y/o deformación.

En la realidad, los materiales presentan naturalmente una gran interacción multiescala, desde el comportamiento atómico y molecular, pasando por la microestructura, hasta lo macroscópicamente observado. El modelado convencional restringe su alcance y validez a una sola escala, sin considerar los fenómenos que ocurren por fuera de la misma, excepto sólo por sus efectos observables en el nivel estudiado. Si bien enfocarse en una sola

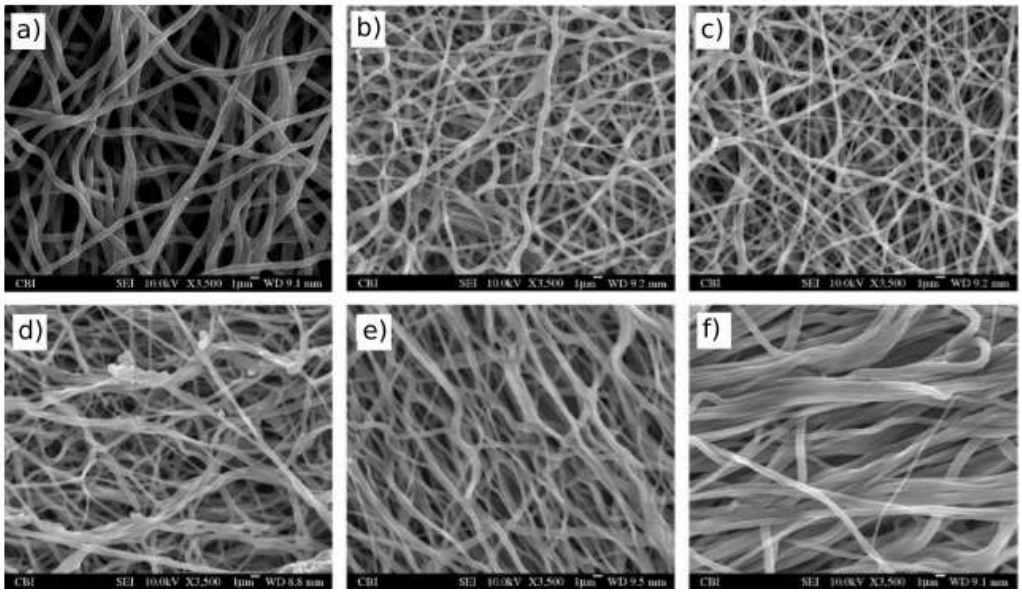


Figura 1.7: Mediante la velocidad de rotación aplicada al mandril colector pueden obtenerse mallas con diferente grado de alineación a lo largo de la dirección circunferencial. (a) Malla isotrópica con $v \approx 0$, (b) $v = 0.3\,\mathrm{m/s}$, (c) $v = 1.5\,\mathrm{m/s}$, (d) $v = 4.5\,\mathrm{m/s}$, (e) $v = 9.0\,\mathrm{m/s}$,(f) $v = 13.8\,\mathrm{m/s}$. Imagen adaptada de [79].

escala simplifica el proceso de modelado, el advenimiento de la nanotecnología permite la fabricación de materiales con la posibilidad de controlar las características microscópicas. Por lo tanto, si se desea optimizar la respuesta macroscópica del material mediante la modificación de su microestructura, se requieren modelos que lleven en consideración el acoplamiento entre las observaciones macroscópicas y los fenómenos microscópicos subyacentes con el fin de lograr un mayor entendimiento de los mecanismos microscópicos subyacentes y su acoplamiento con la escala macroscópica.

Los modelos multiescala surgen dada la necesidad de modelar simultáneamente aspectos estructurales en dos escalas bien diferenciadas[4]. El modelado convencional implicaría el planteo de la escala macroscópica incorporando los aspectos microestructurales únicamente por sus efectos fenomenológicos mediante una ecuación constitutiva, se trata de un modelo eficiente pero se pierde información relevante sobre el comportamiento mecánico de la microestructura. Otra opción es plantear un modelo en escala única cuya resolución permita considerar los aspectos microestructurales al mismo tiempo que la geometría macroscópica, pero se trataría de un problema con un altísimo costo computacional para su implementación. En cambio, el modelado multiescala implica el planteo

[4]En rigor puede tratarse más de dos escalas, en cuyo caso el concepto no presenta mayores diferencias, por lo que se mantuvo el caso de dos escalas para dar mayor claridad a la explicación.

de diferentes modelos para cada escala de manera simultánea con algún método de acoplamiento entre ellos, consiguiendo un equilibrio razonable entre eficiencia y resolución. Los modelos de las distintas escalas pueden originarse de leyes de comportamiento bien diferentes. Mientras que la macroescala suele plantearse como un sólido continuo, la microestructura puede representarse mediante modelos continuos, discretos, estadísticos, de dinámica molecular, entre otros [80].

El modelado multiescala plantea, por lo tanto, que el modelado constitutivo de un material puede plantearse como una jerarquía de modelos de simple escala con diferentes grados de complejidad y acoplados entre sí. Para poder llevar a cabo esta técnica se debe cumplir con la separación de escalas, es decir, que las longitudes características típicas de la microescala sean órdenes de magnitud menores que las longitudes características propias de la macroescala. Bajo esta consideración, siempre será posible encontrar una muestra representativa microscópica del material sobre la cual se puedan realizar y calcular propiedades que resulten estadísticamente representativas del comportamiento del material macroscópico. A una muestra con esas características se la denomina Elemento de Volumen Representativo (*Representative Volume Element*, RVE). Dicho de otra manera, un RVE asociado a un punto material macroscópico de un cuerpo es un volumen material estadísticamente representativo del entorno infinitesimal (desde el punto de vista macroscópico) de ese punto material.

1.5. Antecedentes

El interés en el modelado mecánico de los biomateriales fibrosos ha crecido sostenidamente en las últimas décadas, sin embargo todavía no se ha establecido una base teórica sólida y consensuada al respecto. Como una primera aproximación, el desarrollo de modelos fenomenológicos específicos para biomateriales representó un importante avance al lograr capturar la respuesta no lineal en tensión-deformación típica de estos materiales, compuesta por una región de baja rigidez y una de alta rigidez [79, 81, 82]. Sin embargo, en estos modelos la relación en tensión-deformación consiste en ecuaciones matemáticas macroscópicas cuyos parámetros se ajustan para predecir la respuesta constitutiva del material correspondiente. Por lo tanto, no aportan información acerca de la relación entre los fenómenos microscópicos subyacentes en el material y la respuesta macroscópica observada.

Los modelos microestructuralmente inspirados, en cambio, representan un avance res-

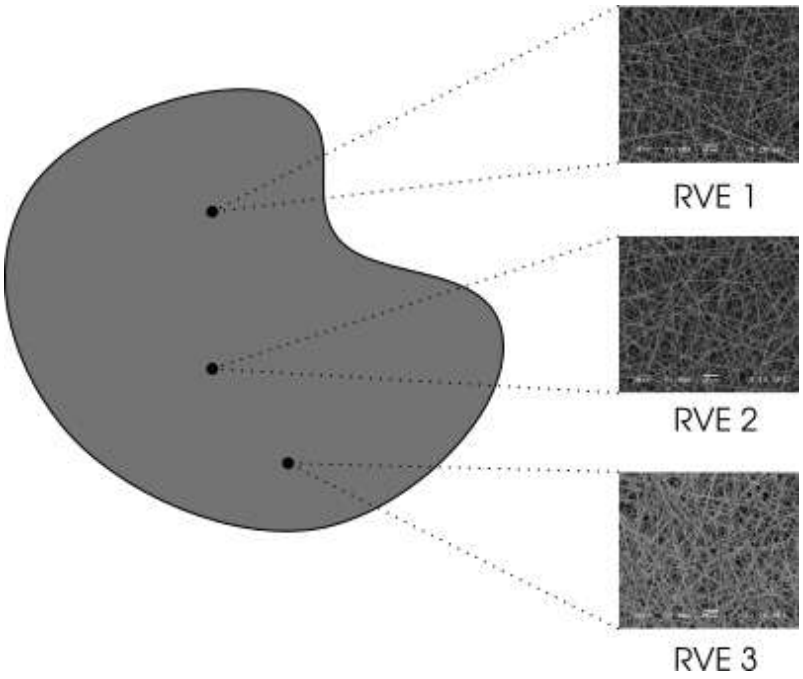


Figura 1.8: Ejemplo de la idea de un tamaño de muestra estadísticamente representativo del material. Las tres muestras microscópicas son diferentes en un sentido estricto, pero no lo son estadísticamente hablando: las distribuciones de probabilidad de diámetro, orientación, enrulamiento son similares, así como la porosidad y otros parámetros que se puedan medir.

pecto de los fenomenológicos, ya que permiten derivar las ecuaciones constitutivas macroscópicas a partir de un análisis de los procesos microscópicos bajo deformación. Estos modelos, en base a suponer deformación afín de la microescala, han permitido vislumbrar que el comportamiento macroscópico anisotrópico y altamente no lineal de los biomateriales se debe en gran medida a la rotación y reclutamiento de las nanofibras, además de a la no linealidad propia del material constituyente [16, 83–87]. En el campo del modelado de materiales fibrosos artificiales, una gran cantidad de trabajos se abocaron a encontrar las propiedades mecánicas macroscópicas efectivas en base a un análisis microestructural. Bajo esta perspectiva, se desarrollaron modelos para diferentes materiales fibrosos como papel, lana, hilados, y otros materiales textiles, con creciente grado de complejidad geométrica [88–92]. Si bien la microestructura se tiene en cuenta en estos modelos para derivar una relación constitutiva, no deja de tratarse de ecuaciones matemáticas que presuponen a priori los modos de deformación microscópicos. Por lo tanto, no logran capturar la heterogeneidad cinemática a escala de cada nanofibra, ya que su deformación estará determinada por las posiciones y deformaciones de las nanofibras circundantes.

La necesidad de una mejor comprensión de los fenómenos microestructurales de las matrices electrohiladas ha llevado el campo de estudio en la dirección del modelado multi-

escala, entendiendo al mismo como una técnica de modelado en la que múltiples modelos en diferentes escalas se plantean de manera simultánea para resolver un mismo sistema [80]. Resulta pertinente aclarar que si bien sería teóricamente posible abordar el problema mediante simulación directa, es decir un modelo del componente macroscópico con un nivel de detalle que alcance hasta la microestructura, tal enfoque resulta prácticamente irrealizable por el alto costo computacional asociado. Un importante primer paso para la realización de simulaciones confiables de este tipo consiste en identificar las características geométricas microscópicas que describen la microestructura nanofibrosa. Algunos de los procesos microestructurales bajo deformación han podido ser observados mediante imágenes SEM de matrices electrohiladas [93, 94]. Aún así, dicha información está restringida a la superficie exterior del material, y poco se puede inferir acerca de cuestiones tridimensionales como la densidad de vínculos cruzados entre las nanofibras, o el largo promedio de los segmentos entre vínculos [95]. Como consecuencia, el modelado multiescala se presenta como una oportunidad no sólo de reproducir la micromecánica de los materiales electrohilados, sino también de elucidar cuáles y cómo son los mecanismos microscópicos que no pueden ser observados experimentalmente. Los modelos basados en el concepto de RVE se encuentran con diversos grados de complejidad, incluyendo composiciones de unas pocas fibras discretas hasta redes fibrosas de geometrías aleatorias tridimensionales. En todo caso, las suposiciones sobre las que se basa la construcción del RVE y el comportamiento mecánico de las fibras resultan esenciales para el comportamiento mecánico obtenido. Los modelos más sencillos requieren mayor número de hipótesis simplificativas, pero permiten focalizar el análisis sobre unas pocas características de interés a la vez que su resolución es muy eficiente. Los modelos más complejos, en cambio, poseen un alto número de variables acopladas que derivan en la respuesta conjunta, conllevando un alto costo computacional asociado. Dependiendo de la aplicación, por lo tanto, se debe hacer un balance entre eficiencia computacional y complejidad del modelo. En las últimas décadas han proliferado los modelos basados en RVEs discretos, encontrando una gran variedad de modelos dependiendo de la combinación de los parámetros que determinan la topología microscópica, así como el comportamiento mecánico individual de las fibras y de sus enlaces.

Stylianopoulos y colaboradores [96] investigaron la influencia de la distribución de orientaciones de las nanofibras sobre la respuesta mecánica macroscópica. Para ello, utilizaron un modelo multiescala computacional para simular matrices electrohiladas de poliuretano bajo tracción uniaxial. Generando geometrías con diferentes grados de alinea-

ción a lo largo de la dirección de carga, encontraron que el módulo de elasticidad a la tracción aumenta considerablemente con la alineación debido a un mayor número de fibras compartiendo la carga. Si bien ese modelo otorgó buena concordancia con resultados experimentales, los parámetros ajustados mostraron discrepancias para la rigidez de las nanofibras y el módulo de rigidez transversal a la dirección de alineación de las fibras. Estas discrepancias se deben, posiblemente, a la falta de consideración por una distribución de enrulamiento inicial así como por el método artificial de determinación de uniones entre fibras.

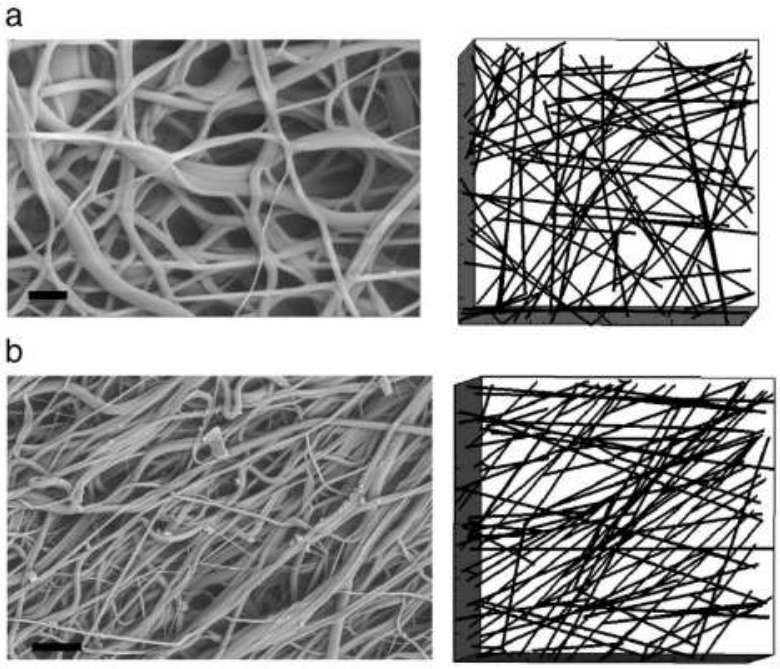


Figura 1.9: RVE de Stylianopoulos y colaboradores [96] para: a) una malla isotrópica, b) una malla alineada.

Para estudiar el efecto de la curvatura de las fibras sobre el módulo de elasticidad de matrices electrohiladas, Pai y colaboradores [97] propusieron un RVE conformado por un arreglo de 4 fibras con curvatura variable. Además de la resistencia de las fibras a la tracción, tuvieron en cuenta también la energía de deformación de las fibras bajo flexión. Encontraron que la curvatura de las fibras es una característica esencial y que su desenrulamiento durante la deformación resulta en un comportamiento más compliante para geometrías con fibras más enruladas. Además, se determinó que otros factores de importancia son la porosidad, el diámetro de las fibras, y la distancia entre uniones.

Wei y colaboradores [98] tomaron un enfoque diferente modelando la mecánica microscópica con métodos de dinámica molecular. Plantearon un RVE compuesto por cien-

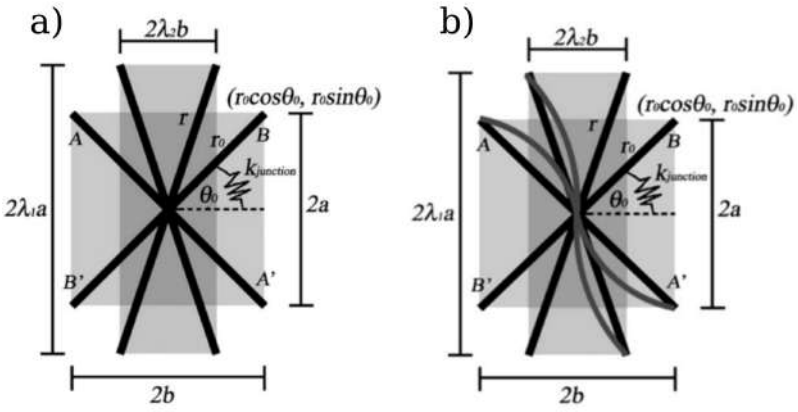


Figura 1.10: RVE de Pai y colaboradores [97]: a) fibras rectas y b) fibras con curvatura.

tos de fibras aleatoriamente distribuidas en un dominio cuadrado, donde cada fibra se representa de forma análoga a una cadena polimérica, con masas esféricas unidas por enlaces covalentes simulados mediante barras elásticas y juntas elásticas. Con este método, se focalizaron en el análisis de las uniones entre las fibras sobre la respuesta macroscópica, pudiendo considerar uniones soldadas e interacciones por proximidad de tipo van der Waals. Realizaron simulaciones con uniones soldadas, mitad soldadas, y no soldadas, encontrando que la densidad de uniones incrementa sensiblemente el módulo de rigidez y la resistencia a la rotura, aunque un exceso de fusiones deriva en la disminución de la energía de fractura. Un resultado interesante es la pequeña diferencia entre uniones soldadas y mitad soldadas, ya que indica que si bien controlar la densidad de puntos de fusión resulta crucial, no es necesario que esa fusión sea total, pudiendo concluir que la resistencia del propio punto de unión se vuelve irrelevante por encima de un determinado valor umbral.

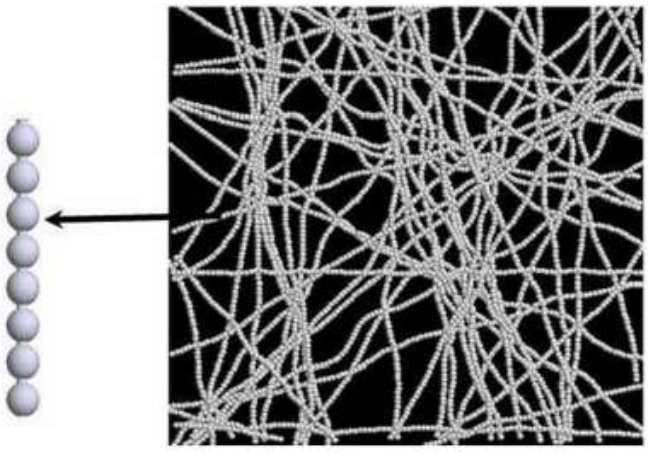


Figura 1.11: RVE de Wei y colaboradores [98].

Rizvi y colaboradores [99, 100] presentaron un modelo matemático en el que se representan las propiedades microestructurales mediante funciones estadísticas, planteando un paralelismo entre un RVE formado por fibras discretas y una descripción probabilística con tres variables de estado: diámetro, longitud de contorno y separación lateral entre los extremos. En esta descripción, la curvatura y la orientación se hallan implícitos como funciones de los tres parámetros de control, pudiendo efectivamente obtener microestructuras con diferentes grados de alineación y tortuosidad. Los resultados de simulaciones bajo tracción uniaxial refuerzan la idea de que las variables microestructurales determinan en gran medida la respuesta macroscópica, destacando que la resistencia a la rotura macroscópica de la matriz depende principalmente de la población de fibras inicialmente curvadas.

Carleton y colaboradores [74] implementaron un algoritmo estocástico de *Random Walk* capaz de obtener RVEs formados por capas bidimensionales con nanofibras de geometrías variables en orientación y tortuosidad. En su modelo, hicieron uso de técnicas de muestreo estadístico para conformar las fibras como una concatenación de segmentos lineales, imponiendo además la condición de periodicidad en los bordes por considerar al RVE como una celda unitaria. Comparando estimaciones matemáticas con simulaciones geométricas obtuvieron una buena concordancia para la fracción de volumen ocupado por las fibras así como para la densidad de intersecciones. En un trabajo subsiguiente [101], modelaron la respuesta mecánica de este RVE donde fibras fueron representadas como vigas de Euler-Bernoulli compuestas de un material hiperelástico de Yeoh, obteniendo buena concordancia con datos experimentales. Adicionalmente, estudiaron el efecto de las variables microscópicas (orientación y tortuosidad de las fibras) sobre la respuesta macroscópica, aunque la utilización de un material no lineal puede haber ocultado los efectos no lineales debidos a estas características.

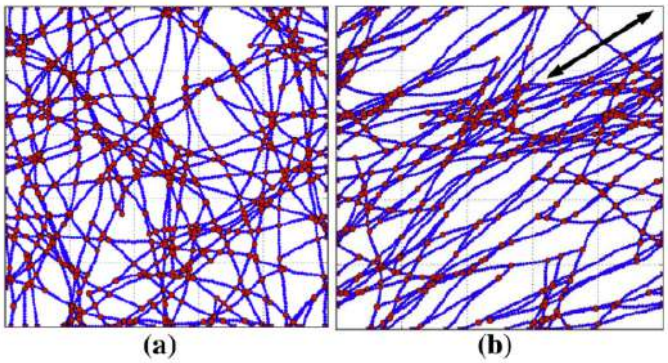


Figura 1.12: RVE de Carleton y colaboradores [74]: a) malla isotrópica, b) malla alineada.

Zundel y colaboradores [95] formularon un modelo basado en un RVE también conformado por capas bidimensionales, introduciendo el concepto de capa de interacción como una distancia normal entre las fibras para la cual se produce una unión. En su trabajo, las fibras se describieron mediante curvas sinusoidales y se les asignó un comportamiento mecánico elástico lineal seguido por plasticidad con endurecimiento lineal, lo cual se asemeja al comportamiento real observado experimentalmente para nanofibras poliméricas. Las simulaciones mostraron muy buena concordancia con datos de ensayos experimentales bajo carga uniaxial sobre matrices elecrohiladas de PA6(3)T previamente reportados por Silberstein y colaboradores [94]. Además llevaron a cabo un análisis sobre la realineación de las fibras y la relación entre la orientación y el estiramiento de las nanofibras bajo deformación. Estos resultados permitieron obtener una diferencia sustancial en el estiramiento de las fibras entre el modelo discreto y modelos afines, algo que en trabajos previos no se había podido elucidar. Posteriormente, Domaschke y colaboradores [102] llevaron a cabo una extensión del modelo en tres dimensiones, encontrando que el modelo bidimensional es una aproximación válida para matrices con porosidades del orden de las encontradas en materiales electrohilados.

1.6. Hipótesis de trabajo

Sobre la base de la revisión bibliográfica presentada acerca de las características experimentales observadas en las mallas electrohiladas, se formulan las siguientes hipótesis de trabajo:

- La microestructura de las mallas electrohiladas es la de un arreglo de nanofibras largas, no tejidas, pero fuertemente unidas en los puntos de contacto por uniones de tipo soldadura debido a solvente residual durante el proceso de deposición.
- Se admite la separación de escalas, permitiendo la implementación de modelos mediante el concepto de RVE.
- En la escala macroscópica la malla puede considerarse como un sólido continuo sujeto a grandes deformaciones donde la tensión depende de la deformación en el caso elástico, y puede incluir dependencia de la tasa de deformación en el caso plástico o viscoelástico.
- La escala microscópica no es factible de ser modelada como un continuo por la presencia predominante de poros entre las fibras. Por lo tanto, es necesario que se modele como un RVE discreto compuesto de un número finito de nanofibras con interacción entre ellas.
- La comunicación entre las escalas puede efectuarse aplicando principios de conservación entre las escalas, resultando en una ley de homogeneización que permite obtener la tensión macroscópica mediante la resolución de un problema microscópico.

Capítulo 2

Modelo constitutivo multiescala para matrices nanofibrosas con reclutamiento progresivo

2.1. Introducción

El modelado convencional, en el marco de la mecánica del continuo, trata con materiales idealizados en los que se asume que la distribución de tensiones y deformaciones puede considerarse homogéneo en el entorno infinitesimal de una dada partícula material (elemento material infinitesimal). En este caso, además, los esfuerzos internos en un punto material cualquiera del continuo queda determinado por la historia de las deformaciones en ese punto. Adicionalmente, en ciertos casos, como el de materiales hiperelásticos, se puede prescindir de las deformaciones pasadas, pudiendo establecer leyes constitutivas que sólo dependen de la deformación actual.

Sin embargo, la homogeneidad aparente a nivel macroscópico suele no ser tal a nivel de microescala, donde el entorno de un punto resulta ser una región que incluye distintos elementos constituyentes con diferentes propiedades y formas. Es decir, el elemento material infinitesimal tiene su propia complejidad bajo la forma de una microestructura no homogénea que, además, evoluciona según la deformación a la que se somete. Por lo tanto, los campos de tensión y deformación dentro del elemento material son igualmente no uniformes a escala microscópica [103]. El planteo de un marco teórico en más de una escala surge debido a la necesidad de modelar con precisión materiales que naturalmente presentan esta separación de escalas, siendo ejemplos típicos: las aleaciones metálicas, combinaciones de polímeros, materiales compuestos, medios porosos, materiales policristalinos y materiales biológicos. Para estos casos, la respuesta macroscópica depende

del tamaño, forma, propiedades y distribución espacial de sus constituyentes microestructurales [104].

La teoría multiescala tiene sus inicios en los trabajos pioneros de Hill [105–108], Hashin y Shtrikman [109], Budiansky [110] y Mandel [111], entre otros. Mayormente se ha enfocado la atención en el desarrollo de modelos basados en el concepto de Elemento de Volumen Representativo (*Representative Volume Element*, RVE): a cada punto material de la macroescala se lo asocia con un dominio de la microescala apropiadamente identificado, que representa la configuración de referencia de la microestructura en ese punto. Es importante que la longitud característica de la microescala sea considerablemente menor a la longitud característica de la macroescala, y que el tamaño del dominio microscópico sea lo suficientemente grande para ser verdaderamente representativo de la microestructura (figura 2.1). Dadas estas condiciones, existe separación de escalas y el dominio asociado a la microescala es considerado estadísticamente representativo del material en el punto material macroscópico asociado, y en consecuencia se lo denomina RVE.

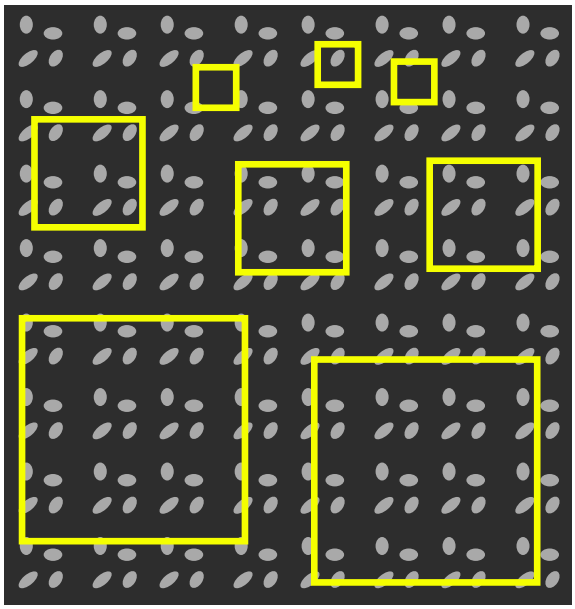


Figura 2.1: Esquema de una microestructura y dominios microscópicos de distinto tamaño. Se ve claramente que a medida que se aumenta el tamaño, el dominio encuadrado contiene en su interior mayor número de constituyentes haciéndolo más representativo de la microestructura.

Esta disociación de un punto material macroscópico de su entorno, detallado en el RVE, requiere de dos operaciones para completar el sistema: la inserción y la homogeneización. La inserción se refiere a la comunicación de la deformación macroscópica sobre el dominio microscópico, derivando generalmente en un problema de valores de contorno

donde se suele optar por condiciones de borde homogéneas o periódicas, aunque otras alternativas son posibles. Inversamente, la homogeneización es la obtención de la tensión macroscópica a partir del campo de tensiones en el RVE, siendo lo más usual considerar el promedio de las tensiones en el volumen microscópico (figura 2.2).

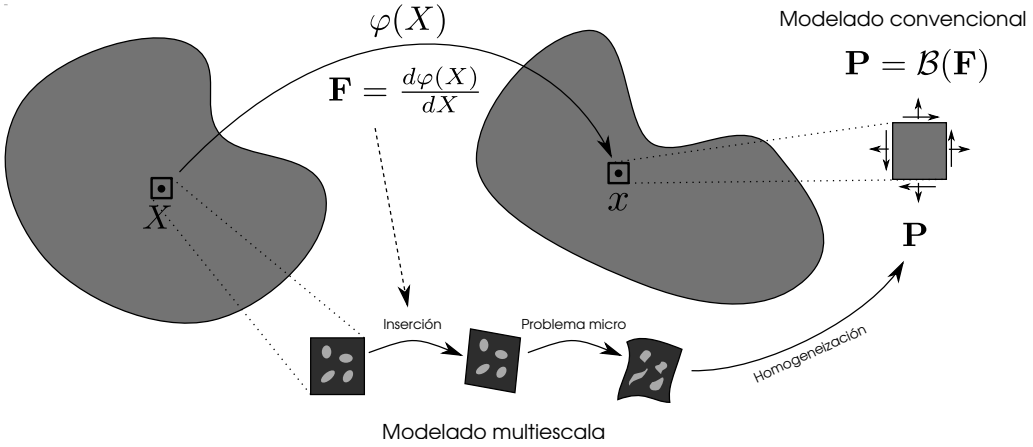


Figura 2.2: Esquema de los dos tipos de modelado constitutivo: convencional y multiescala. El modelado constitutivo convencional plantea un sola escala en la cual se establece una relación entre la tensión y la deformación en la forma de una ecuación matemática ($\mathbf{P}(\mathbf{F})$). El modelado multiescala, en cambio, asocia a cada punto material $\mathbf{X}$ un dominio microscópico que se resuelve simultáneamente a partir de la inserción de información desde la escala macroscópica, y como resultado se obtiene, mediante alguna técnica de homogeneización, la tensión aparente desde la macroescala.

En las últimas décadas, la utilización de modelos multiescala en aplicaciones prácticas se ha basado casi exclusivamente en técnicas de homogeneización computacional [112–123]. Esta técnica no busca la obtención de leyes constitutivas bajo la forma de un sistema cerrado de ecuaciones matemáticas, sino que promueve un análisis local-global en el que la tensión en cada punto de integración del modelo macroscópico se deriva de la resolución del problema microscópico en un RVE suficientemente detallado. Esta metodología presentan una serie de ventajas:

- No requieren de ningún tipo de suposición constitutiva a nivel macroscópico.
- Permiten la incorporación de grandes deformaciones y rotaciones tanto a nivel microscópico como macroscópico.
- Habilitan la posibilidad de incorporar al análisis macroscópico información detallada de la microestructura y su evolución geométrica y física durante la deformación macroscópica.

- Permiten la utilización de diferentes técnicas de modelado para cada escala, pudiendo combinar, por ejemplo, un modelo de continuo a escala macroscópica con un modelo microscópico de componentes discretos.

Consistentemente con las consideraciones previas, se plantea la base general del modelo multiescala para mallas electrohiladas. El punto de partida es considerar el ensamble de nanofibras como una estructura con comportamiento mecánico regular, capaz de ser descripto en términos de los aspectos esenciales de su microestructura. El modelo consiste de dos escalas: la escala microscópica que provee una representación de la microestructura y la escala macroscópica que representa el material homogeneizado con las propiedades emergentes de la malla electrohilada como un todo. Conjuntamente se presentan los operadores de inserción y homogeneización que permiten el acoplamiento entre escalas. A continuación se implementa esta formulación en un RVE prototipo relativamente sencillo que sirve como ejemplo de aplicación de la teoría formulada y se valida mediante comparación con datos experimentales. También se estudia el efecto de las propiedades microestructurales más relevantes sobre la respuesta macroscópica observada. Finalmente se ajustan los parámetros microscópicos para el diseño virtual de una malla electrohilada biomimética de una arteria intracranial humana.

2.2. Descripción macroscópica

La malla electrohilada se modela a nivel macroscópico como un sólido continuo sujeto a grandes deformaciones. En esta sección se presenta la base cinemática de este modelo junto con las ecuaciones de equilibrio mecánico haciendo uso del Principio de Potencias Virtuales [124]. Además, se detalla la linealización de las ecuaciones de equilibrio mediante el método de Newton-Rapshon, necesaria para su implementación en algoritmos computacionales.

2.2.1. Cinemática

Configuración material y espacial

Se tiene un cuerpo β ocupando inicialmente la región Ω_m del espacio. En esta configuración, llamada configuración material o configuración de referencia, el cuerpo se encuentra libre de solicitaciones externas y se asume también como libre de esfuerzos internos.

Luego, bajo una dada deformación, el cuerpo pasa a ocupar en un tiempo t la región Ω, y se dice que se encuentra en su configuración espacial. Esta deformación está caracterizada por el mapeo continuo φ que relaciona la posición $\mathbf{X}$ de las partículas materiales en la configuración de referencia, con la posición $\mathbf{x}$ que ocupan en la configuración espacial (figura 2.3):

$$\begin{aligned} \varphi : \Omega_m &\to \Omega \\ \mathbf{X} &\mapsto \mathbf{x} = \varphi(\mathbf{X}) \end{aligned} \tag{2.1}$$

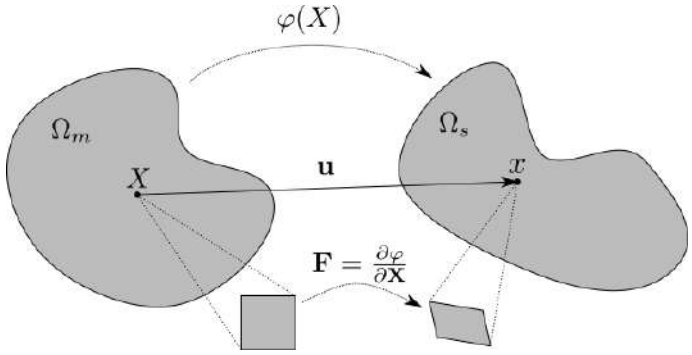


Figura 2.3: Cinemática del continuo aplicada a la escala macroscópica: el mapeo φ define la deformación del cuerpo de la configuración inicial Ω_m a la configuración Ω, y su gradiente $\mathbf{F}$ caracteriza la transformación del elemento diferencial material. Queda definido también el campo de desplazamientos $\mathbf{U}$ como la diferencia entre las posiciones espaciales $\mathbf{x}$ y materiales $\mathbf{X}$ de las partículas del cuerpo.

Luego, se define el campo vectorial de desplazamientos $\mathbf{U}$ como la diferencia entre la posición espacial y material para cada partícula. Por lo tanto, se puede caracterizar la deformación de Ω_m a Ω y su inversa mediante las expresiones:

$$\mathbf{U}(\mathbf{X}) = \varphi(\mathbf{X}) - \mathbf{X} \tag{2.2}$$

$$\mathbf{x} = \varphi(\mathbf{X}) = \mathbf{X} + \mathbf{U}(\mathbf{X}) \tag{2.3}$$

$$\mathbf{X} = \varphi^{-1}(\mathbf{x}) = \mathbf{x} - \mathbf{u}(\mathbf{x}) \tag{2.4}$$

donde φ^{-1} indica el mapeo inverso que relaciona la posición material de las partículas a partir de su posición espacial y $\mathbf{u}(\mathbf{x})$ es el mismo campo de desplazamientos $\mathbf{U}(\mathbf{X})$ pero expresado en función de las posiciones espaciales como variables independientes: $\mathbf{u}(\mathbf{x}) = \mathbf{U}(\mathbf{X} = \varphi^{-1}(\mathbf{x}))$.

Es válido notar que ambos dominios, Ω_m y Ω, coinciden para el instante inicial cuando

el campo de desplazamientos es nulo.

Tensor gradiente de deformaciones

El gradiente del mapeo φ se conoce como tensor gradiente de deformaciones ($\mathbf{F}$), y puede expresarse como:

$$\mathbf{F}(\mathbf{X}) = \nabla\varphi = \frac{\partial \mathbf{x}}{\partial \mathbf{X}} \tag{2.5}$$

$$\mathbf{F}(\mathbf{U}) = \mathbf{I} + \nabla\mathbf{U} \tag{2.6}$$

donde ∇ denota el operador gradiente respecto de las coordenadas materiales $\mathbf{X}$.

Es pertinente notar que el tensor gradiente de deformación $\mathbf{F}$ es el que caracteriza la transformación de un segmento diferencial material $\mathrm{d}\mathbf{X}$ en el segmento espacial $\mathrm{d}\mathbf{x}$, mientras que su determinante $|\mathbf{F}|$ es una medida de la dilatación o contracción del elemento de volumen diferencial material $\mathrm{d}\Omega_m$ al transformarse en su contraparte espacial $\mathrm{d}\Omega$:

$$\mathrm{d}\mathbf{x} = \mathbf{F}\mathrm{d}\mathbf{X} \tag{2.7}$$

$$\mathrm{d}\Omega = |\mathbf{F}|\,\mathrm{d}\Omega_m \tag{2.8}$$

Resulta evidente que el mapeo φ debe cumplir con la condición $|\mathbf{F}| > 0$ para evitar la posibilidad de la desaparición de un volumen material. Además, para materiales incompresibles surge la restricción:

$$|\mathbf{F}| = 1 \tag{2.9}$$

que resulta una condición cinemática razonable en el modelado de tejidos biológicos [125].

2.2.2. Equilibrio Mecánico

Se designa mediante $\partial\Omega$ al contorno del cuerpo en su configuración espacial Ω. A su vez, se divide este contorno en $\partial\Omega^D$ y $\partial\Omega^N$, denotando respectivamente al borde de

Dirichlet y al borde de Neumann, de forma que $\partial\Omega = \partial\Omega^D \cup \partial\Omega^N$, cumpliéndose también que $\partial\Omega^D \cap \partial\Omega^N = \emptyset$[1]. En $\partial\Omega^D$ se prescriben los valores de los desplazamientos $\overline{\mathbf{U}}$, mientras que en $\partial\Omega^N$ se prescribe el valor de la tracción $\mathbf{t}^N$ (figura 2.4).

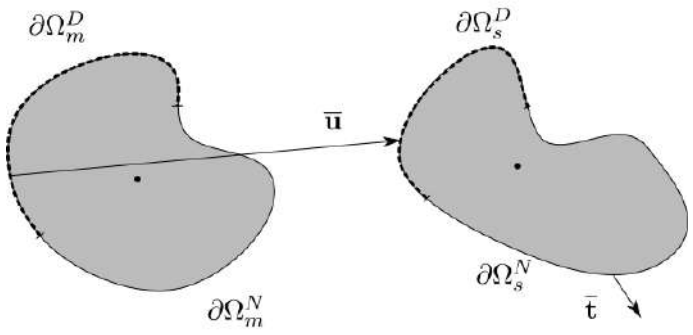


Figura 2.4: Equilibrio del cuerpo continuo de la escala macroscópica: el borde $\partial\Omega$ se subdivide en sus partes de Dirichlet $\partial\Omega^D$ y Neumann $\partial\Omega^N$. En el borde de Dirichlet se prescribe el campo de desplazamientos $\overline{\mathbf{U}}$ y en el borde de Neumann se prescriben las tracciones $\mathbf{t}^N$.

Luego, se plantea el equilibrio mecánico en términos del Principio de Potencias Virtuales (PPV), que a su vez se enmarca en el campo de la mecánica variacional [126–128]. En este contexto resulta fundamental definir los conjuntos de desplazamientos cinemáticamente admisibles: el campo de desplazamientos espacial $\mathbf{u}$ que caracteriza la deformación del cuerpo, pertenece a un espacio funcional $\mathcal{U}_s$, con funciones suficientemente regulares para que las operaciones matemáticas estén correctamente definidas. Generalmente se tiene que $\mathcal{U}_s = \mathbf{H}^1(\Omega)$, es decir, el espacio de funciones con gradientes de cuadrado integrable en Ω.

Se define, además, a $\mathrm{Kin}_s^{\mathcal{U}} \in \mathcal{U}_s$ como el conjunto de desplazamientos cinemáticamente admisibles, siendo los desplazamientos que satisfacen las restricciones cinemáticas sobre la frontera de Dirichlet:

$$\mathrm{Kin}_s^{\mathcal{U}} = \{\mathbf{w} \in \mathcal{U}_s;\, \mathbf{w}|_{\partial\Omega^D} = \overline{\mathbf{w}}\} \tag{2.10}$$

Este espacio puede considerarse como la traslación de otro subespacio $\mathrm{Var}_s^{\mathcal{U}}$, llamado espacio de desplazamientos variacionalmente admisibles, cuyos elementos son nulos en

[1] $\emptyset$ denota a un conjunto de medida nula.

la frontera de Dirichlet:

$$\mathrm{Var}_s^{\mathcal{U}} = \{\mathbf{w} \in \mathcal{U}_s;\, \mathbf{w}|_{\partial\Omega^D} = \mathbf{0}\} \tag{2.11}$$

Una vez correctamente definidos estos conjuntos, según el PPV, el problema de equilibrio mecánico queda planteado en la configuración espacial como:

$$\int_\Omega \boldsymbol{\sigma} \cdot \mathrm{grad}^S \hat{\mathbf{u}}\, \mathrm{d}\Omega = \int_\Omega \mathbf{g} \cdot \hat{\mathbf{u}}\, \mathrm{d}\Omega + \int_{\partial\Omega^N} \mathbf{t}^N \cdot \hat{\mathbf{u}}\, \mathrm{d}\partial\Omega^N \quad \forall \hat{\mathbf{u}} \in \mathrm{Var}_s^{\mathcal{U}} \tag{2.12}$$

donde $\boldsymbol{\sigma}$ es el tensor de tensiones de Cauchy, $\hat{\mathbf{u}}$ una acción de movimiento variacionalmente admisible (también llamada velocidad virtual), $\mathrm{grad}^S \hat{\mathbf{u}}$ el gradiente simétrico de $\hat{\mathbf{u}}$ con respecto a las coordenadas espaciales, $\mathbf{t}^N$ la tracción impuesta en la frontera de Neumann y $\mathbf{g}$ las fuerzas externas de volumen.

Cabe notar que la condición de incompresibilidad ($|\mathbf{F}| = 1$) no ha sido impuesta ya que se va a tratar con materiales altamente porosos en los que la variación a nivel microscópico del tamaño de los poros resultará en una contracción o dilatación volumétrica a nivel macroscópico. Además no se han tenido en cuenta efectos inerciales. Su incorporación al problema podría efectuarse con relativa facilidad en caso de ser necesario, aunque esto iría en perjuicio de la claridad en el desarrollo subsiguiente.

En el apéndice A se da una descripción completa del PPV, incluyendo su linealización para la implementación en esquemas de Newton-Raphson.

2.3. Descripción Microscópica

2.3.1. RVE

A nivel microscópico se considera a la malla como un ensamble de nanofibras individuales, pudiendo encontrar en la literatura diversas representaciones de la microestructura electrohilada en base a unas pocas fibras [94, 97, 129]. Siguiendo a Silberstein et. al. [94], se asume aquí una idealización de la microestructura compuesta por capas superpuestas de nanofibras conformadas en redes triangulares de diferente orientación. Luego, se adopta como RVE una celda compuesta por dos triángulos superpuestos, dando cuenta de la naturaleza de capas propia de las matrices electrohiladas (figura 2.5). El modelo original

de Silberstein et. al. asume que cada miembro del RVE (lado de triángulo) se corresponde con una fibra, teniendo la desventaja de contar con un número reducido de fibras. Para compensar esta limitación, se propone en esta tesis modificar la formulación de Silberstein et a. considerando que el RVE está formado por triángulos cuyos lados representan haces o fascículos de fibras y no fibras individuales. Así, cada lado se considera como un haz de fibras con orientación compartida pero con diferente grado de enrulamiento.

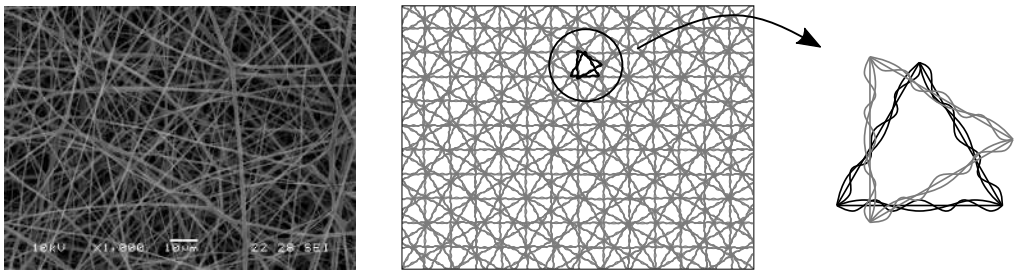


Figura 2.5: Imagen SEM de la microestructura de una malla electrohilada de PLLA (izquierda). La microescala se modela como un arreglo idealizado de dos capas de redes triangulares (centro). El RVE se compone de dos celdas triangulares superpuestas con un desfasaje de orientación de $30°$.

2.3.2. Cinemática

Una de las principales ventajas del RVE propuesto es que se determina la deformación directamente a partir del tensor gradiente de deformaciones macroscópico $\mathbf{F}$. En reposo, la orientación de cada miembro i está dada por un versor $\mathbf{a}^0{}_i$. El vector deformado $\mathbf{a}_i = \mathbf{F}\mathbf{a}^0{}_i$ da cuenta tanto de la orientación del miembro deformado así como de su elongación (figura 2.6), definida como la relación entre la longitud deformada del segmento y su longitud inicial: $\lambda_i = l_i/l_i^0$, o equivalentemente:

$$\lambda_i = \sqrt{\mathbf{a}_i \cdot \mathbf{a}_i} \tag{2.13}$$

2.3.3. Ecuaciones Constitutivas

Fibra individual

Las nanofibras son elementos de gran esbeltez que presentan una alta rigidez a la tracción pero al querer comprimirlas se enrulan con facilidad debido a su baja resistencia a

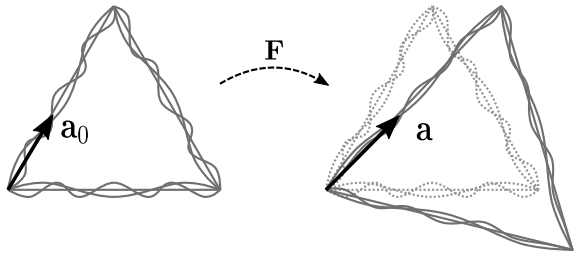


Figura 2.6: Deformación del RVE prototipo compuesto de dos celdas triangulares superpuestas. Aquí se muestra una sola celda triangular bajo deformación afín en la frontera: la deformación de cada haz queda determinada directamente por el tensor macroscópico $\mathbf{F}$ de forma que el versor de orientación inicial $\mathbf{a}^0$ de cada haz transforma en el vector deformado $\mathbf{a} = \mathbf{F}\mathbf{a}^0$.

la flexión [130]. Por lo tanto, su comportamiento elástico puede representarse adecuadamente por medio de una ley bilineal en tensión-deformación caracterizada por un módulo de rigidez fuerte (E_t) y uno débil (E_b) como se muestra en la figura 2.7. E_t representa la resistencia de la fibra recta y tensa a la tracción, mientras que E_b representa la débil resistencia de la fibra enrulada [131].

Además, las fibras en su estado inicial no se hallan necesariamente rectas, sino que pueden presentar diferentes niveles de enrulamiento. Luego, a medida que se la estira (es decir, que sus extremos se alejan entre sí) se va desenrulando hasta que, en un determinado valor de elongación, la fibra deviene recta. Debido a que en ese instante la nanofibra incrementa considerablemente su capacidad de tomar carga, se dirá que es reclutada. Se define como elongación de reclutamiento (λ^r) al valor de elongación para el cual esto ocurre. También, en ocasiones, se referirá a λ^r como el nivel o grado de enrulamiento inicial que presenta una fibra, entendiendo que mayores valores de λ^r se corresponden de forma biunívoca con mayores enrulamientos iniciales.

Luego, se puede expresar la tensión ingenieril (t) de la fibra de la siguiente manera:

$$t(\lambda) = \begin{cases} E_b\left(\lambda - 1\right) & if \quad \lambda < \lambda^r \\ E_b\left(\lambda^r - 1\right) + E_t\left(\frac{\lambda}{\lambda^r} - 1\right) & if \quad \lambda \geq \lambda^r \end{cases} \tag{2.14}$$

Es válido aclarar que la ecuación constitutiva para una nanofibra se define en términos de la tensión ingenieril, es decir, carga soportada por unidad de área de la sección de referencia. De esta forma se tiene la ventaja de poder correlacionar directamente la curva tensión-deformación con datos de mediciones experimentales.

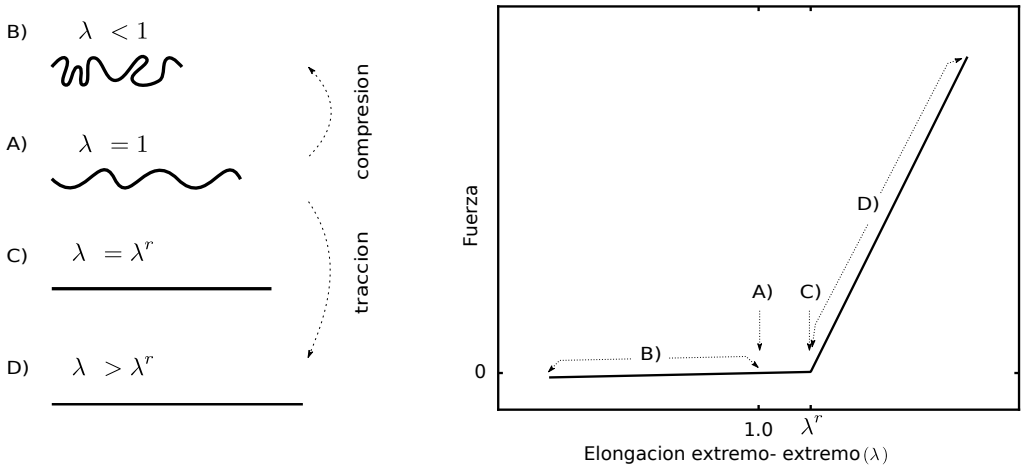


Figura 2.7: Respuesta mecánica bilineal individual de una fibra. La nanofibra se deposita enrulada en su estado de reposo (A). Si se la comprime (acercando sus extremos) o estira (alejando sus extremos) sin llegar a rectificarla, ejerce una leve resistencia flexural. Si se la estira hasta devenir recta (C), su rigidez a la tracción se incrementa considerablemente.

Haz de fibras

La matriz electrohilada se compone de fibras individuales que se depositan con orientaciones, en general, aleatorias y con distintos niveles de enrulamiento. De acuerdo con la cinemática adoptada, todas las fibras que comparten una orientación, están sometidas a igual deformación. Por lo tanto, englobar a todas las fibras que tienen una misma orientación en un solo objeto permite simplificar el análisis y las expresiones que se obtienen. Para ello, se introduce el concepto de haz de fibras como un cierto número de fibras con orientación compartida y que en lugar de encontrarse rectas en su estado inicial, presentan una variabilidad estadística en sus niveles de enrulamiento.

Como se detalló previamente, las fibras enruladas poseen una rigidez a la elongación mucho menor que las que se hayan rectas. Es por eso que a medida que el haz es sometido a tracción, el subconjunto de fibras de la matriz que se encuentran rectas soporta la mayor parte de la carga, en cuanto que el aporte a la resistencia de las fibras enruladas es prácticamente despreciable. Mientras se incrementa la carga, las nanofibras enruladas van progresivamente deviniendo rectas, aumentando en gran medida su resistencia a incrementos de elongación. Este proceso de reclutamiento progresivo de fibras provoca que el módulo de rigidez del haz dependa no solamente de la rigidez de las fibras individuales, sino también de la distribución estadística de enrulamientos de las fibras que componen al haz. Para ejemplificar esto, la figura 2.8 muestra un haz esquemático compuesto por tres fibras con tres elongaciones de reclutamiento diferentes. Bajo una determinada elon-

gación (dada por el alejamiento de los extremos), la fibra f_1 se mantiene enrulada y su resistencia a la tracción es despreciable, la fibra f_2 está deviniendo recta en su elongación de reclutamiento y la fibra f_3 se encuentra recta y tensa, por encima de su elongación de reclutamiento, por lo que ejerce una tensión mucho mayor que f_1 y f_2.

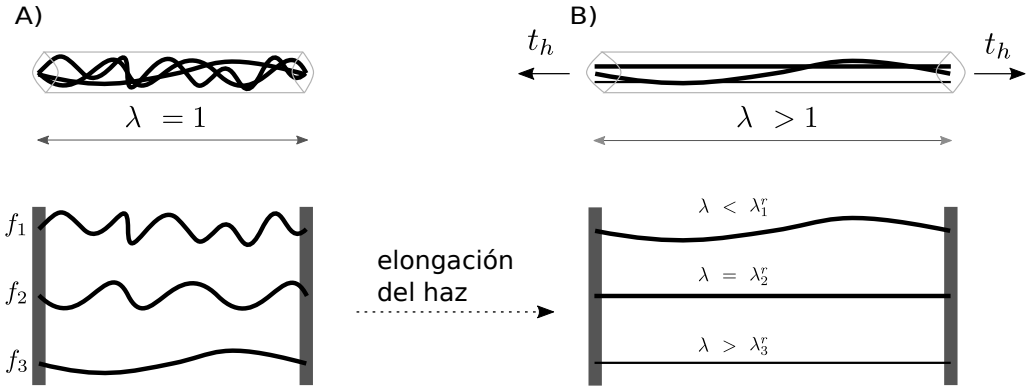


Figura 2.8: Esquema de un haz sencillo compuesto de tres fibras con diferentes valores de reclutamiento. En la parte superior, las nanofibras que componen el haz se encuentran encerradas por un cilindro imaginario. En la configuración de reposo (A) las nanofibras (f_1, f_2 y f_3) se encuentran enruladas según su estado de deposición con distintos valores de reclutamiento (${\lambda^r}_1 > {\lambda^r}_2 > {\lambda^r}_3$). En la configuración deformada (B) la fibra f_1 se mantiene enrulada, la fibra f_2 se encuentra en su elongación de reclutamiento y la fibra f_3 se encuentra recta y estirada. Bajo esta configuración, la resistencia de las fibras f_1 y f_2 es marginal respecto de la fibra f_3.

Si se asume un gran número de fibras por cada haz, es posible realizar un enfoque estadístico para incorporar la dispersión en las elongaciones de reclutamiento de las fibras al análisis bajo la forma de una función densidad de probabilidad de reclutamiento $p^r(\lambda)$. En tal caso, puede considerarse que la cantidad de fibras poseen niveles de enrulamiento entre λ^r y $\lambda^r + \mathrm{d}\lambda^r$ viene dada por $p^r(\lambda)\mathrm{d}\lambda$. Para implementar esta distribución en el RVE propuesto, se asume una distribución de enrulamientos normal truncada, dada por:

$$p^r(\lambda^r) = \begin{cases} 0 & if \quad \lambda^r < {\lambda^r}_{min} \\ \left(\frac{1}{C}\right) e^{\left[-\frac{(\lambda^r-\mu^r)^2}{2(\sigma^r)^2}\right]} & if \quad {\lambda^r}_{min} \leq \lambda^r \leq {\lambda^r}_{max} \\ 0 & if \quad \lambda^r > {\lambda^r}_{max} \end{cases} \tag{2.15}$$

donde μ^r es el valor medio de la distribución, σ^r es el valor de desviación estándar, ${\lambda^r}_{min}$ y ${\lambda^r}_{max}$ son los valores mínimo y máximo de reclutamiento de las fibras en el haz, y C es una constante que toma el valor necesario para asegurar la condición de normalización

2.16:

$$\int_{\lambda^r{}_{min}}^{\lambda^r{}_{max}} p^r(\lambda^r)\mathrm{d}\lambda^r = 1 \tag{2.16}$$

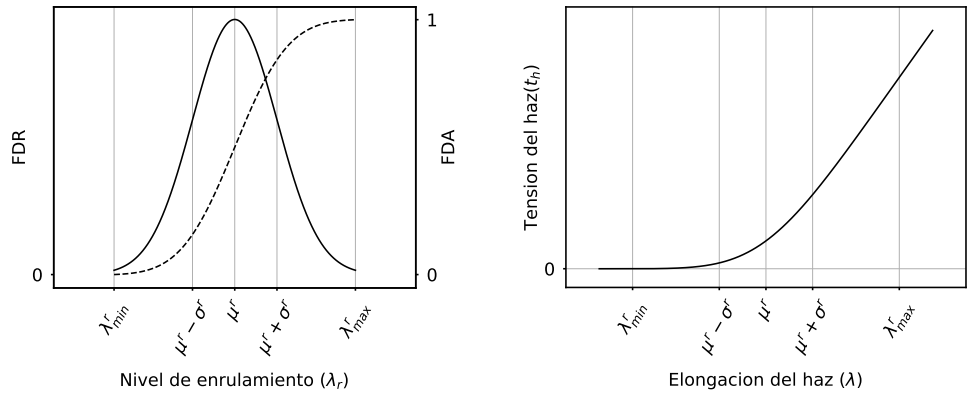
Consecuentemente, la tensión desarrollada por el haz bajo una determinada deformación está dada por el promedio de las tensiones de las fibras que lo componen bajo esa misma deformación:

$$t_h(\lambda) = \int_{\lambda^r{}_{min}}^{\lambda^r{}_{max}} t(\lambda, \lambda^r)p^r(\lambda^r)\mathrm{d}\lambda^r \tag{2.17}$$

donde se ha explicitado la dependencia de la tensión de las fibras de sus correspondientes elongaciones de reclutamiento.

A causa de la dispersión en los enrulamientos de las nanofibras, la cantidad de fibras reclutadas para cada valor de deformación varía de forma gradual como se muestra en la figura 2.9a. En su configuración inicial, el haz posee ninguna o pocas fibras que se encuentran efectivamente rectas, y ninguna está aún soportando carga ya que se trata del estado de reposo. En cuanto el haz se somete a tracción, las fibras comienzan a desenrularse, por lo cual progresivamente van deviniendo rectas e incrementan sustancialmente su aporte a la resistencia frente a la carga externa. A este fenómeno gradual se lo denominará reclutamiento progresivo, y se evidencia en la respuesta en tensión-elongación del haz por una curva en forma de "J", donde el módulo tangente efectivo se incrementa continuamente desde ≈ 0 hasta un valor final constante cuando todas las fibras están reclutadas (2.9b).

Para mayor abundancia, es posible identificar tres regiones bien diferenciadas en la respuesta mecánica del haz: i) una región inicial floja que corresponde a una configuración en la que muy pocas o todas las fibras se hallan enruladas ($\lambda < \mu^r - \sigma^r$), ii) una región de transición donde se produce el reclutamiento progresivo de las fibras a medida que se incrementa la deformación del haz ($\mu^r - \sigma^r < \lambda < \mu^r + \sigma^r$) y iii) una región final de respuesta lineal o cuasilineal donde todas o prácticamente todas las fibras se encuentran reclutadas ($\lambda > \mu^r + \sigma^r$). Queda en evidencia que el rango de elongación en el cual ocurre la transición, es decir la severidad con que se incrementa el módulo tangente del haz, está determinado en mayor medida por la dispersión de la distribución normal truncada (σ^r). En todos los casos, sin embargo, la aplicación de una distribución continua de enrulamientos resulta en curvas de tensión-deformación suaves a pesar de haber adop-



(a) (b)

Figura 2.9: a) Distribución de reclutamiento normal truncada esquemática, incluyendo la función densidad de reclutamiento $p^r(\lambda^r)$ (línea sólida, indicada como FDR) y la función de distribución acumulada (línea punteada, indicada como FDA) del haz. b) Curva de tensión-elongación correspondiente, donde se evidencia el reclutamiento progresivo en la forma "J".

tado, para las fibras individuales, una ley constitutiva bilineal de derivada discontinua en $\lambda = \lambda^r$.

2.3.4. Homogeneización

Dado que el dominio del RVE se corresponde con una partícula material macroscópica, se asume como ley de homogeneización que en cada punto macroscópico el tensor de tensiones de Cauchy ($\boldsymbol{\sigma}$) equivale al promedio en volumen de su contraparte microscópica ($\boldsymbol{\sigma}_\mu$) sobre el volumen deformado del RVE (V_μ) [103].

$$\boldsymbol{\sigma} = \frac{1}{V_\mu} \int_{V_\mu} \boldsymbol{\sigma}_\mu \mathrm{d}V_\mu \tag{2.18}$$

Sin embargo, para un RVE discreto con espacios vacíos carentes de partículas materiales tanto la deformación como el tensor de tensiones no están definidos. En este caso conviene expresar el promedio sobre el volumen microscópico efectivamente ocupado por material. Particularmente para el caso de una microestructura fibrosa, esto resulta en una

sumatoria sobre los volúmenes de las nanofibras, obteniendo:

$$\boldsymbol{\sigma} = \frac{1}{V_\mu} \sum_{i=1}^{N_f} \int_{V_i} \boldsymbol{\sigma}_\mu \mathrm{d}V_i \tag{2.19}$$

donde N_f es el número de fibras en el RVE y V_i es el volumen ocupado por cada nanofibra i.

Luego, resulta necesario establecer una expresión para el tensor de tensiones dentro de cada fibra i, en función de la tensión ingenieril desarrollada por la fibra (t_i), de su elongación extremo-extremo (λ_i) y de su vector orientación ($\mathbf{a}_i$):

$$\boldsymbol{\sigma}_\mu|_{V_i} = \frac{t_i}{\lambda_i} \left(\mathbf{a}_i \otimes \mathbf{a}_i\right) \tag{2.20}$$

Cabe resaltar que esta expresión implica una tensión constante en la fibra, lo cual tiene sentido para fibras reclutadas (rectas). Para fibras enruladas se mantiene esta expresión a modo de aproximación, considerando despreciable el error que se introduce dado que el aporte de estas fibras a la tensión macroscópica es marginal.

Resulta de interés explicitar como variable, por su importancia experimental, a la fracción de volumen material de la malla (η), definida como la fracción del volumen del RVE (V_μ) que se halla ocupado por las nanofibras.

$$\eta = \frac{V_f}{V_\mu} \tag{2.21}$$

donde $V_f = \sum_{i=1}^{N_f} V_i$ es el volumen del RVE ocupado por las nanofibras.

Además, dada la condición de deformación afín en la frontera, es posible expresar el volumen deformado del RVE en función del tensor gradiente de deformaciones macroscópico y del volumen inicial del RVE (V_μ^0):

$$V_\mu = |\mathbf{F}|\, V_\mu^0 \tag{2.22}$$

Incorporando las expresiones 2.20, 2.21 y 2.22 en 2.19, la fórmula de homogeneiza-

ción adopta la forma:

$$\boldsymbol{\sigma} = \frac{\eta}{|\mathbf{F}|\, V_f} \sum_{i=1}^{N_f} \frac{t_i}{\lambda_i} \left(\mathbf{a}_i \otimes \mathbf{a}_i\right) V_i \tag{2.23}$$

La ecuación 2.23 admite la posibilidad de mallas con estructuras aleatorias bajo modos de deformación diferentes para cada fibra. Sin embargo, es posible incorporar la deformación del RVE propuesta donde $\mathbf{a}_i = \mathbf{F}\mathbf{a}^0{}_i$, pudiendo reescribir la fórmula de homogeneización en función de las orientaciones iniciales de las fibras:

$$\boldsymbol{\sigma} = \frac{\eta}{|\mathbf{F}|\, V_f} \mathbf{F} \left[\sum_{i=1}^{N_f} \frac{t_i}{\lambda_i} \left(\mathbf{a}^0{}_i \otimes \mathbf{a}^0{}_i\right) V_i \right] \mathbf{F}^T \tag{2.24}$$

A continuación, para hacer surgir naturalmente el concepto de haz de fibras, se reordena la sumatoria agrupando en una segunda sumatoria a todas las fibras que comparten una misma orientación:

$$\boldsymbol{\sigma} = \frac{\eta}{|\mathbf{F}|\, V_f} \mathbf{F} \left\{ \sum_{j=1}^{N_h} \left[\sum_{k=1}^{N_f^j} \frac{t_{jk}}{\lambda_j} \left(\mathbf{a}^0{}_j \otimes \mathbf{a}^0{}_j\right) V_{jk} \right] \right\} \mathbf{F}^T \tag{2.25}$$

donde N_h es el número de haces que componen al RVE (en el caso propuesto $N_h = 6$), N_f^j es el número de fibras que componen el haz j y el sobreíndice jk indica a la fibra k del haz j.

Al igual que para la distribución de enrulamiento, también es posible, mediante un enfoque estadístico, llevar en consideración una distribución uniforme sobre los volúmenes de las fibras que componen cada haz. De modo que el volumen ocupado por las fibras puede expresarse según:

$$V_f = \sum_{j=1}^{N_h} \sum_{k=1}^{N_f^j} V_{jk} \tag{2.26}$$

Luego, si todas las fibras poseen idéntico diámetro y, por lo tanto, misma sección

transversal A_f:

$$V_f = \sum_{j=1}^{N_h} \int_{\lambda^r{}_{min}}^{\lambda^r{}_{max}} A_f \lambda^r l^0 p^r(\lambda^r) \mathrm{d}\lambda^r \tag{2.27}$$

Además, se asume que todos los haces presentes en el RVE se componen de igual manera, es decir, que todos tienen la misma función distribución de probabilidad $p^r(\lambda^r)$. Luego, es posible llegar a una expresión relativamente sencilla para el volumen ocupado por las fibras cuando éstas se agrupan en haces:

$$V_f = N_h A_f l^0 \mu^r \tag{2.28}$$

Aplicando esta simplificación a la ecuación 2.25, la fórmula de homogeneización puede reexpresarse la tensión macroscópica en función de la deformación macroscópica y de los parámetros geométricos y constitutivos microscópicos:

$$\boldsymbol{\sigma}(\mathbf{F}) = \frac{\eta}{|\mathbf{F}|\, N_h \mu^r} \mathbf{F} \left[\sum_{j=1}^{N_h} \int_{\lambda^r{}_{min}}^{\lambda^r{}_{max}} \frac{t(\lambda, \lambda^r)}{\lambda_j} \left(\mathbf{a}^0{}_j \otimes \mathbf{a}^0{}_j \right) p^r(\lambda^r) \mathrm{d}\lambda^r \right] \mathbf{F}^T \tag{2.29}$$

Finalmente, se condensa la expresión anterior haciendo surgir la tensión del haz (ecuación 2.17):

$$\boldsymbol{\sigma}(\mathbf{F}) = \frac{\eta}{|\mathbf{F}|\, N_h \mu^r} \mathbf{F} \left[\sum_{j=1}^{N_h} \frac{t_h^j(\lambda)}{\lambda_j} \left(\mathbf{a}^0{}_j \otimes \mathbf{a}^0{}_j \right) \right] \mathbf{F}^T \tag{2.30}$$

Cabe resaltar que aunque se aplicó la noción de un haz compuesto por un gran número de fibras, es posible mantener esta última expresión para haces con número reducidos de fibras. En estos casos la tensión de haz debe calcularse como el promedio de las tensiones de las fibras ponderado con las secciones transversales, en lugar de aplicar la ecuación 2.17, utilizada bajo la aproximación de infinitas fibras.

La fórmula de homogeneización 2.30 tiene la ventaja adicional de poder transformarse fácilmente para ser utilizada tanto con el primer tensor de tensiones de Piola-Kirchhoff ($\mathbf{P}$) como con el segundo ($\mathbf{S}$), según las clásicas definiciones $\boldsymbol{\sigma} = \frac{1}{|\mathbf{F}|}\mathbf{P}\mathbf{F}^T$ y $\mathbf{P} = \mathbf{F}\mathbf{S}$:

$$\mathbf{P} = \frac{\eta}{N_h \overline{\lambda^r}} \mathbf{F} \left[\sum_{j=1}^{N_h} \frac{t_h^j}{\lambda^j} \left(\mathbf{a^0}_j \otimes \mathbf{a^0}_j \right) \right] \tag{2.31}$$

$$\mathbf{S} = \frac{\eta}{N_h \overline{\lambda^r}} \left[\sum_{j=1}^{N_h} \frac{t_h^j}{\lambda^j} \left(\mathbf{a^0}_j \otimes \mathbf{a^0}_j \right) \right] \tag{2.32}$$

2.4. Materiales y Métodos

Para validar el modelo propuesto se realizó una comparación con datos experimentales de ensayos de inflado donde un injerto vascular es sometido a presión interior monótonamente creciente a medida que se registra el cambio en diámetro. Los datos experimentales se tomaron de un trabajo externo reportado por Suarez Bagnasco et. al. [132]. A continuación se realiza una breve descripción de los materiales y métodos allí reportados.

2.4.1. Materiales

Los injertos ensayados son tubuladuras de ácido poli-L-láctico (PLLA)[2] de pequeño diámetro producidos mediante la técnica de electrohilado por Montini Ballarin et. al. [133]. Los parámetros del proceso son: concentración de solución $10\ \%\mathrm{wt/V}$, caudal $0.5\,\mathrm{mL/h}$, distancia aguja-colector $15\,\mathrm{cm}$, diferencia de potencial aplicada $13\,\mathrm{kV}$ y velocidad de rotación de colector $1000\,\mathrm{rpm}$. El equipo utilizado consiste de una fuente de alto voltaje[3], una aguja de acero inoxidable de punta roma[4], una bomba de infusión a jeringa[5] y un colector cilíndrico rotativo de acero inoxidable de 5mm de diámetro. Para cada tubuladura, el proceso de electrohilado se llevó a cabo durante $2\,\mathrm{h}$, cambiando la posición de la boquilla cada $15\,\mathrm{min}$ para lograr un espesor uniforme según el largo del injerto. Las tubuladuras obtenidas son de 8 a $11\,\mathrm{cm}$ de largo y espesores alrededor de $0.04\,\mathrm{mm}$. Dado el reducido espesor respecto del diámetro del colector, el largo de los especímenes no es una variable de influencia en el estado tensional de las muestras durante los ensayos de inflado. Adicionalmente, se llevaron a cabo micrografías SEM de las paredes interior y exterior de los injertos (figura 2.10) y se midió el diámetro medio de las fibras en ambas superficies, reportando $0.48\,\mu\mathrm{m}$ ($\pm\ 0.14\,\mu\mathrm{m}$) y $0.36\,\mu\mathrm{m}$ ($\pm\ 0.07\,\mu\mathrm{m}$) respectivamente. Es-

[2] PLA2002D, $Mn = 78.02\,\mathrm{kg/mol}$, Natureworks MN, USA
[3] Gamma High Voltage Research Inc., Ormond Beach, Florida, USA
[4] 18 gauge, Aldrich®
[5] Activa® A22 ADOX, Ituzaingó, Argentina

tos valores, si bien presentan una variación, son cercanos entre sí y se puede observar en las micrografías SEM microestructuras similares.

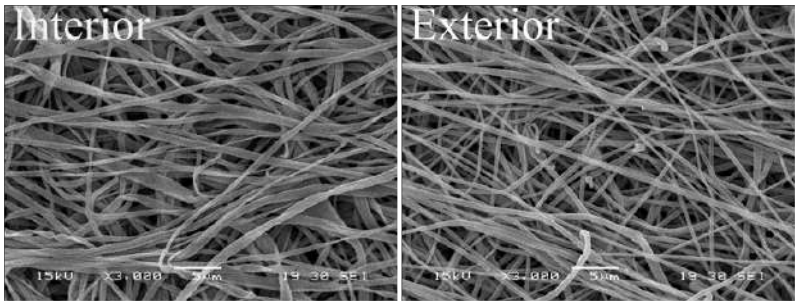


Figura 2.10: Imágenes SEM de las microestructuras nanofibrosas de las paredes interior y exterior de un injerto tubular de PLLA electrohilado.

2.4.2. Ensayos de inflado

Los datos experimentales de los ensayos de inflado se toman de lo reportado por Suares Bagnasco y colaboradores [132] Los ensayos mecánicos se llevaron a cabo en un banco de prueba de simulación hemodinámica diseñado para medir presiones y diámetros instantáneos en vasos sanguíneos e injertos vasculares. El aparato consiste básicamente de una bomba programable especialmente diseñada para suministrar un caudal de líquido en un circuito hidráulico cerrado, y un reservorio de fluido (solución fisiológica) donde las muestras se conectan mientras permanecen inmersas en el líquido. El circuito hidráulico se compone de conducciones de silicona, constricciones variables, la muestra a ensayar y el reservorio de fluido. Se pueden llevar a cabo ensayos estáticos, de presión en aumento monótono o simulaciones con pulsos de frecuencia variable para realizar ensayos dinámicos que imiten el pulso cardíaco de pacientes normales o hipertensos [134]. La medición *in vitro* de presión interna se realiza mediante transductores de estado sólido de alta frecuencia[6]. La variación temporal del diámetro se consigue mediante la técnica de sonomicrometría con transductores (pequeños cristales de ultrasonido)[7] fijados de forma diametralmente opuesta sobre las paredes del injerto. Para estos ensayos la presión intraluminal se incrementó gradualmente desde aproximadamente $50\,\mathrm{mmHg}$ hasta $150\,\mathrm{mmHg}$ (figura 2.11).

[6]Konigsberg Inc., PA, USA
[7]Triton Technology Inc., SD, USA

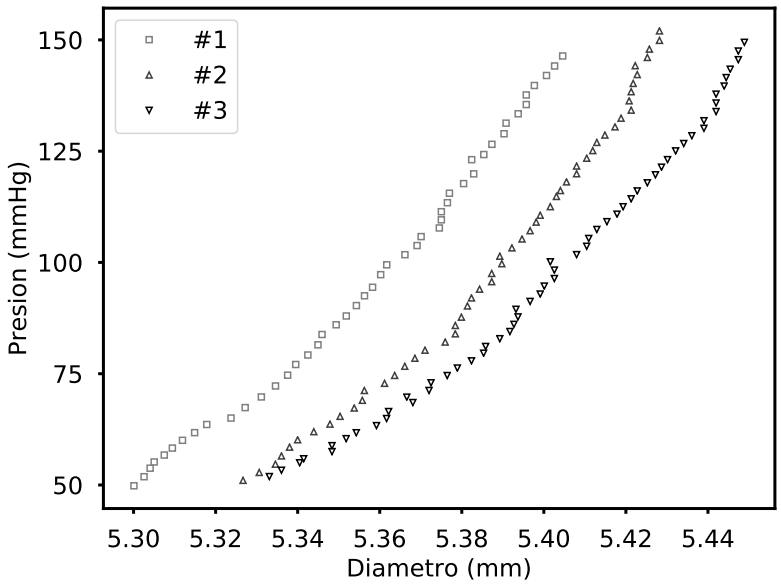


Figura 2.11: Curvas presión-diámetro experimentales para las tres muestras de injertos de PLLA electrohilado.

2.4.3. Simulaciones computacionales

Para simular computacionalmente los ensayos de inflado, se utiliza un modelo macroscópico de membrana tridimensional con geometría cilíndrica discretizado por elementos finitos (figura 2.12) [135]. Sobre los extremos se imponen condiciones de desplazamiento acorde a cómo se sujetan los injertos en el banco de pruebas [132]. A cada punto de integración se le asigna una instancia del RVE propuesto y previamente detallado, componiendo así el modelo multiescala. Si bien se reportó una leve diferencia entre la morfología de las paredes interior y exterior del andamio, se asumirá que la microestructura es uniforme según el espesor de la pared del injerto. La tabla 2.1 lista las dimensiones en reposo de las muestras ensayadas. El largo de los tubos se tomó en $8\,\mathrm{cm}$ para todos los casos, habiendo conducido pruebas a diferentes largos sin encontrar variaciones apreciables en la respuesta constitutiva. Esto tiene sentido dado que la elevada delgadez de la pared respecto del diámetro de los injertos conlleva una muy baja rigidez flexural de la misma.

2.5. Resultados y discusión

En esta sección se presentan los resultados de la aplicación de la teoría desarrollada bajo el RVE prototípico propuesto. En primera instancia se valida el modelo ajustando sus parámetros y realizando una comparación de la respuesta constitutiva del modelo

Tabla 2.1: Dimensiones de los injertos vasculares de PLLA electrohilado utilizados para validar el modelo.

Muestra de PLLA	Espesor de pared (mm)	Diámetro interno (mm)
#1	0.0410	4.9294
#2	0.0385	4.9271
#3	0.0375	4.9324

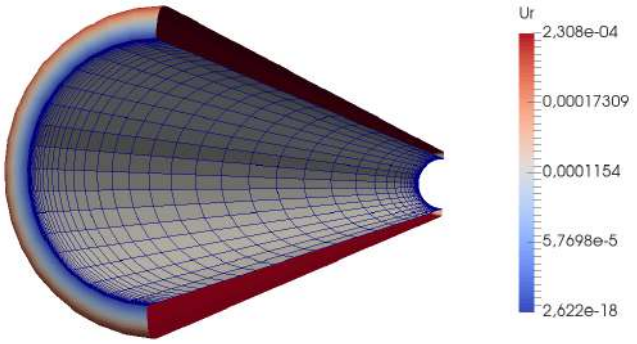


Figura 2.12: Modelo computacional de elementos finitos para simular los ensayos de inflado. La geometría inicial y la discretización se muestra en gris. La geometría deformada se muestra coloreada según la magnitud de los desplazamientos radiales (unidad: m).

con datos experimentales de ensayos de inflado de tubuladuras de PLLA electrohilado. Además, se realiza una serie de simulaciones para evaluar los efectos sobre la respuesta macroscópica que se obtiene mediante variaciones de los parámetros microestructurales, incluyendo tanto el módulo elástico de las fibras como los parámetros de la distribución de reclutamiento. Finalmente, los parámetros se vuelven a ajustar para obtener una respuesta de un posible andamio vascular capaz de imitar la respuesta mecánica de un tejido vascular objetivo, mostrando la capacidad del modelo constitutivo multiescala propuesto en este capítulo como herramienta de diseño de materiales biomiméticos dada su buena capacidad para predecir el comportamiento macroscópico a partir de parámetros microscópicos que podrían manipularse convenientemente durante la fabricación.

2.5.1. Comparación con datos experimentales

Los parámetros microscópicos del modelo constitutivo multiescala se ajustaron para reproducir las curvas experimentales de presión-diámetro reportadas para los injertos vasculares de PLLA electrohilado (figura 2.11). Estas curvas presentan una respuesta en

forma de "J" frente al aumento de la presión interna: la variación en diámetro del injerto frente al aumento de la presión es relativamente menor en el rango de presiones más bajas ($\approx 50\,\mathrm{mmHg}$) respecto del rango de presiones más altas ($\approx 150\,\mathrm{mmHg}$), con una transición gradual en el medio. El modelo constitutivo logra capturar apropiadamente esta característica para los tres conjuntos de datos, tal como se muestra en la figura 2.13. Los valores correspondientes a los parámetros que ajustan la respuesta de cada muestra están listados en la tabla 2.2. A medida que se incrementa la presión interna, las curvas de presión-diámetro capturadas en las simulaciones muestran la misma respuesta con forma de "J", con la pendiente aumentando como resultado de un creciente número de fibras que van siendo reclutadas a medida que avanza la deformación. Este cambio en pendiente se reduce a medida que el número de fibras pendientes de reclutar es menor y finalmente se llega a una región prácticamente lineal al tener a todas las fibras rectas y reclutadas.

Mayores deformaciones podrían provocar fenómenos de rotura de nanofibras o de plasticidad, conllevando adicionales efectos no lineales. Estos escenarios no fueron contemplados en este primer RVE prototipo, donde el objetivo ha sido modelar el comportamiento elástico de un injerto vascular en el rango fisiológico de presiones. Es importante notar que un injerto que busque reemplazar tejido biológico mecánicamente no lineal, debe mostrar un comportamiento semejante no lineal sin incurrir en deformación plástica o rotura de fibras que conlleven a subsecuentes mecanismos de falla. Sin embargo, estos fenómenos son tenidos en cuenta en los capítulos siguientes con RVEs de mayor complejidad geométrica.

La mayoría de los modelos publicados, incluidos los recientes, no llevan en consideración la tortuosidad de las fibras o su dispersión estadística, por lo que no logran capturar el proceso de reclutamiento progresivo de las fibras durante la deformación [94, 96, 101, 136, 137]. Adicionalmente, la respuesta macroscópica no lineal suele reproducirse en los modelos multiescala mediante la adopción de leyes constitutivas hiperelásticas no lineales para las fibras en la microescala. Sin embargo, son numerosos los trabajos experimentales que reportan un comportamiento mecánico lineal para las nanofibras sometidas a deformaciones bajas como las fisiológicas [136, 138–143]. En el modelo aquí presentado, la adopción de una distribución estadística para la tortuosidad de las fibras permite reproducir exitosamente la respuesta constitutiva macroscópica no lineal, aún contemplando un material mecánicamente lineal para las nanofibras individuales.

Tabla 2.2: Parámetros del modelo ajustados para reproducir las curvas de presión-diámetro experimentales de las tres muestras de PLLA. El valor medio (VM) y la desviación estándar (DE) de cada parámetro también se encuentra listado.

	η	E_t(GPa)	E_b(MPa)	μ^r	σ^r	${\lambda^r}_{min}$	${\lambda^r}_{max}$
$\#1$	0.3	2.9	5.186	1.1091	0.0313	1.00	1.40
$\#2$	0.3	2.9	7.961	1.1079	0.0265	1.00	1.40
$\#3$	0.3	2.9	9.625	1.1190	0.0328	1.00	1.40
VM	0.3	2.9	7.591	1.1120	0.0302	1.00	1.40
DE	0.0	0.0	1.831	0.0050	0.0027	0.00	0.00

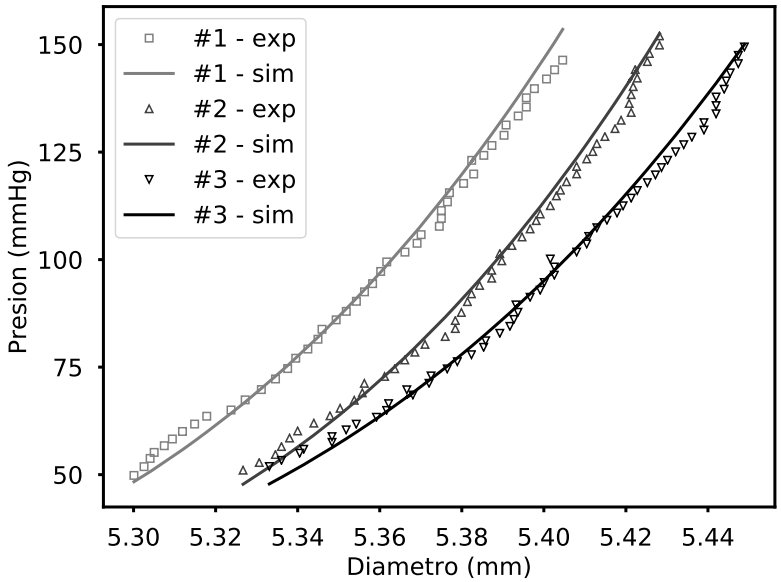


Figura 2.13: Curvas de presión-diámetro experimentales y simuladas para las tres muestras de injertos vasculares de PLLA. Los coeficientes R^2 del ajuste son $0{,}989$ para el ensayo $\#1$ y $0{,}990$ para los ensayos $\#2$ y $\#3$.

2.5.2. Efecto de las propiedades microestructurales sobre la respuesta macroscópica

Las propiedades microscópicas son las que determinan la respuesta macroscópica del material. Dado que es posible acceder a algún grado de control sobre estas propiedades durante el proceso de electrohilado, resulta de interés estudiar el efecto macroscópico producido por variaciones de estos parámetros microestructurales. Para hacerlo, se toma como punto de partida los valores obtenidos para el ajuste con datos experimentales (tabla 2.2). Se seleccionaron los parámetros más significativos: la orientación del RVE para evaluar el nivel de anisotropía, el módulo elástico de las nanofibras como variable más importante relacionada con la rigidez del material y los parámetros de la distribución de reclutamiento para evaluar su función en la respuesta no lineal con forma "J". Luego se llevó a cabo una serie de simulaciones considerando variaciones en estos parámetros y los resultados se compararon entre sí en términos de tensión circunferencial versus elongación circunferencial, cuya relevancia es fundamental en arterias, por ejemplo, si se quieren desarrollar matrices biomiméticas.

Orientación del RVE

Para evaluar la anisotropía del RVE se estudia la sensibilidad de la respuesta mecánica frente a cambios en la dirección de deformación uniaxial. Se realizan dos simulaciones con direcciones de deformación de $90°$ y $75°$ respecto de la horizontal (ver figura 2.5). Dada la geometría de dos celdas triangulares superpuestas (con simetría angular cada $30°$), las dos direcciones de carga elegidas maximizan la variación en la respuesta del RVE. La comparación entre las dos simulaciones se muestra en la figura 2.14a, donde puede verse que ambas curvas están prácticamente solapadas, significando que los efectos de anisotropía son despreciables para este RVE prototipo. En consecuencia, este RVE es un buen candidato para simular mallas de orientación isotrópica de fibras, pero necesitaría modificaciones para poder simular correctamente mallas de fibras alineadas.

Módulo elástico de las nanofibras

El módulo tangente elástico macroscópico, medido sobre el final de la curva cuando todas las fibras se encuentran reclutadas, depende principalmente del módulo elástico a la tracción de las nanofibras. La figura 2.14b muestra cómo varía la curva en tensión-elongación circunferencial para el injerto acorde a variaciones de E_t. Un incremento en la

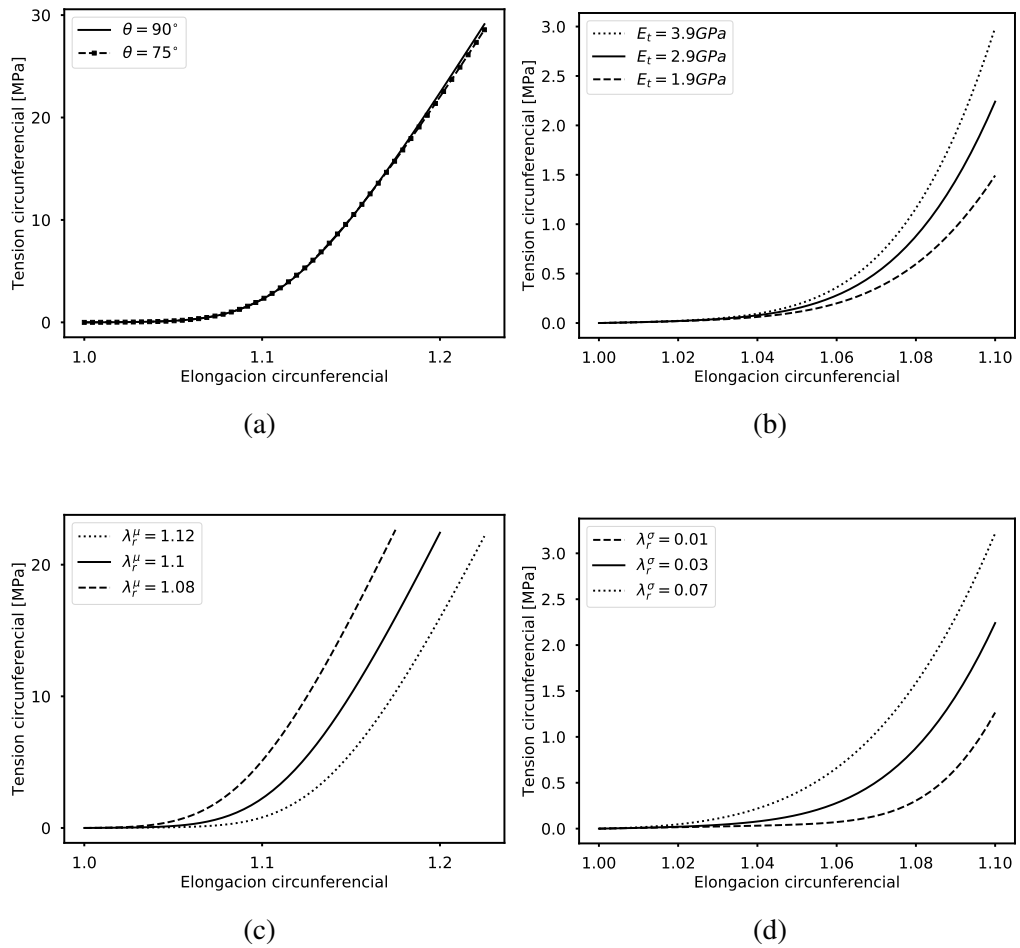


Figura 2.14: Efecto de los parámetros microscópicos sobre la respuesta mecánica macroscópica. Se muestran curvas simuladas de tensión-elongación circunferencial como resultado de variaciones de la dirección de deformación uniaxial (θ) (a), módulo elástico de las fibras a la tracción (b), valor medio de la distribución de reclutamiento (c) y valor de dispersión de la distribución de reclutamiento (d).

rigidez de las nanofibras resulta en un módulo tangente macroscópico mayor, sin modificar la forma "J" de la curva e incluso manteniendo inalterados los valores de elongación para los cuales comienza y termina el reclutamiento de las fibras y, por ende, la transición del módulo tangente desde el pequeño valor inicial hasta su valor final. En consecuencia, controlar la rigidez de las nanofibras (por ejemplo por medio de seleccionar diferentes polímeros base) permite modificar la compliancia del injerto electrohilado en su estado deformado. Este aspecto es esencial para el objetivo de alcanzar la biomímesis, aunque no resulta suficiente por cuenta propia, ya que la respuesta mecánica del material en reposo se mantiene inalterada.

Parámetros de la distribución de tortuosidades

La distribución de enrulamiento cuasi normal adoptada permite controlar dos parámetros: el valor medio μ^r y la dispersión σ^r. En primera instancia se realizó la variación del valor medio de reclutamiento con el objetivo de aumentar la tortuosidad de todas las nanofibras sin alterar su dispersión. Esto resultó en una traslación de la curva de tensión-elongación circunferencial, sin cambios en su forma (figura 2.14c). En otras palabras, valores más altos de μ^r conllevan valores más altos de elongación circunferencial macroscópica para los cuales las nanofibras devienen rectas y se reclutan. En segunda instancia, se varió el valor de dispersión, generando distribuciones de reclutamiento con mayor o menor número de nanofibras con valores de reclutamiento alejados del valor medio. En este caso, se observó que a mayores valores de σ^r se obtienen curvas de tensión-elongación macroscópicas con una transición de módulo elástico más suave y prolongada (figura 2.14d). En consecuencia, modificar la FDR, es decir generar microestructuras con diferentes distribuciones de enrulamiento de las nanofibras, conlleva un cambio en la transición de la respuesta mecánica debido al proceso de reclutamiento progresivo. Por lo tanto, en el diseño de materiales biomiméticos, la posibilidad de controlar la FDR puede ser un factor crucial, ya que su efecto resulta complementario al de seleccionar diferentes polímeros base para controlar la rigidez de las fibras.

2.5.3. Determinación de parámetros de un injerto vascular

Como se discutió previamente, una gran ventaja del modelado multiescala es la posibilidad de vincular propiedades microscópicas con la respuesta mecánica macroscópica, y por ende brindar una herramienta para el diseño de microestructuras de materiales bio-

miméticos. Como ejemplo de esto, el modelo propuesto en este capítulo se utilizó para determinar las propiedades microestructurales de un injerto vascular electrohilado capaz de imitar la respuesta en presión-diámetro para una arteria intracranial humana en el rango de presiones fisiológicas (60-140 mmHg), cuyos datos experimentales se obtienen de la literatura [Hayashi1980].

Entonces, para poner a prueba la capacidad del modelo como herramienta de diseño, se optimizaron los parámetros en el modelo de membrana (figura 2.12), tomando un diámetro interno en reposo de $2.5\,\mathrm{mm}$ y un espesor de $0.04\,\mathrm{mm}$. Los parámetros elegidos para ser optimizados fueron los módulos de elasticidad de las fibras (E_t y E_b) y su distribución de reclutamiento (μ^r y σ^r), mientras que la fracción de volumen se mantuvo en 0.3 y los límites de la FDR quedaron inalterados en 1.0 y 1.4, respectivamente. Cabe notar que los parámetros optimizados corresponden, en la práctica, a la selección de un material de base para las nanofibras que cumpla con la respuesta elástica pedida y la determinación de la morfología de la malla electrohilada para que cumpla con la FDR dada por los valores μ^r y σ^r optimizados.

La figura 2.15 muestra las curvas de presión-diámetro para la arteria intracranial humana y para el injerto optimizado para biomímesis, mientras que los parámetros obtenidos se listan en la tabla 2.3. La curva simulada muestra un buen acuerdo en la región de presión fisiológica, a la vez que se mantiene similar en comportamiento para bajas presiones (0-60 mmHg). Este injerto necesitaría estar constituido por nanofibras menos rígidas, una propiedad que puede controlarse mediante la selección de un polímero diferente para el electrohilado. Entre los valores de E_t hallados en la literatura para nanofibras aisladas, se encuentra que para PCL (poli-ε-caprolactona) un diámetro aproximado de $0.5\,\mathrm{nm}$, el módulo de Young reportado de $0.3\,\mathrm{GPa}$ se corresponde adecuadamente con el obtenido aquí mediante optimización [143]. También podrían probarse otros materiales conocidos que son menos rígidos que el PLLA, como por ejemplo poliuretano segmentado de grado médico, seda, poliglicerol sebacato (PGS), colágeno, entre otros. Otra estrategia podría ser la combinación de diferentes materiales, regulando la relación de cada polímero para obtener la propiedad deseada. Adicionalmente, la distribución de reclutamiento obtenida implica la necesidad de fibras tanto con un mayor valor medio de tortuosidad así como una mayor dispersión de la misma, para poder imitar, mediante el proceso de reclutamiento progresivo, la débil compliancia de los materiales biológicos a bajas presiones junto con una rigidez similar a presiones fisiológicas. Vale mencionar que, de todas maneras, fuera del rango fisiológico no se hace necesaria la biomímesis.

Tabla 2.3: Parámetros del modelo optimizados para un injerto electrohilado capaz de imitar la respuesta mecánica de una arteria intracranial humana.

η	E_t(GPa)	E_b(MPa)	μ^r	σ^r	$\lambda^r{}_{min}$	$\lambda^r{}_{max}$
0.3	0.303	6.824	1.3338	0.0710	1.00	1.40

η	0.3
E_t(GPa)	0.303
E_b(MPa)	6.824
μ^r	1.3338
σ^r	0.0710
$\lambda^r{}_{min}$	1.0
$\lambda^r{}_{max}$	1.4
SE(mmHg)	1.92

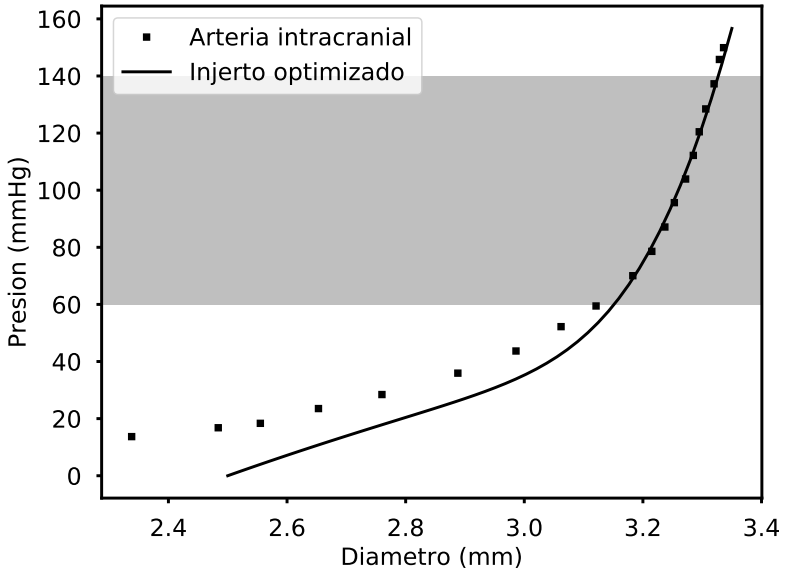


Figura 2.15: Respuesta en presión-diámetro para una arteria intracranial humana y para un injerto simulado biomimético capaz de reproducir la respuesta mecánica en presión-diámetro en el rango fisiológico (indicado por la zona gris). El coeficiente de ajuste R^2 para el rango completo de presiones es de $0{,}89$, mientras que para el rango fisiológico mejora a $0{,}98$.

2.6. Conclusiones

En este capítulo se presentó un modelo constitutivo multiescala para el comportamiento mecánico elástico de injertos vasculares electrohilados de ingeniería de tejidos. A escala macroscópica se adoptó un modelo de membrana tridimensional siguiendo los conceptos clásicos de la mecánica del continuo en grandes deformaciones. La microestructura se idealizó como una red triangulada de haces de fibras, mientras que la morfología de capas superpuestas de las mallas electrohiladas se capturó mediante un elemento de volumen representativo compuesto por capas superpuestas de celdas triangulares.

Se reprodujo exitosamente el comportamiento mecánico macroscópico experimental

de tubos electrohilados de PLLA bajo ensayos mecánicos de inflado, obteniendo además el comportamiento macroscópico a partir de las características microscópicas del material electrohilado. La forma de "J" de la curva constitutiva observada experimentalmente fue reproducida apropiadamente por el modelo como resultado de un enderezamiento de un número creciente de fibras, y su reclutamiento progresivo, acorde al avance de la deformación.

La utilización de un enfoque multiescala implica una contribución al entendimiento y la exploración de las características microscópicas relevantes que resultan en el comportamiento mecánico macroscópico característico de los materiales electrohilados, aportando información sobre el comportamiento mecánico a partir de parámetros naturales que además admiten ser fácilmente reconocidos, medidos y controlados en la fabricación. El modelado computacional también permitió analizar los efectos de estos parámetros de forma individual (por ejemplo el módulo elástico de las nanofibras o el valor medio de enrulamiento) sobre la respuesta en tensión-deformación de los injertos electrohilados. Se encontró que la implementación de una distribución de reclutamiento permite obtener una respuesta macroscópica no lineal incluso haciendo uso de una ley constitutiva bilineal para las nanofibras en la microescala, evitando introducir complejidades no lineales artificiosas como la aplicación de leyes constitutivas hiperelásticas no lineales a nivel de fibras.

Dependiendo en la aplicación objetivo de un biomaterial nanofibroso, los parámetros del proceso de fabricación podrían ajustarse para obtener injertos vasculares realizados a medida, es decir, paciente específicos y de acuerdo al distrito vascular a reemplazar. Esta posibilidad se ejemplificó para el caso de un injerto biomimético respecto de una arteria intracranial humana, obteniendo los parámetros microscópicos necesarios para reproducir la respuesta en presión-diámetro macroscópica de la arteria en el rango de presiones fisiológicas. Además, los resultados indican que una distribución de reclutamiento con alta dispersión y el proceso de reclutamiento progresivo de las fibras son ingredientes clave para reproducir la curva constitutiva con forma de "J" típica de los materiales biológicos. Por lo tanto, poder ejercer control sobre la distribución de reclutamiento de un injerto nanofibroso puede ser crucial en el esfuerzo por conseguir injertos eficazmente biomiméticos.

Dentro de las limitaciones, el RVE prototipo aquí estudiado admite una cinemática sencilla que no permite considerar otros fenómenos que puedan aportar al comportamiento no lineal del material, como por ejemplo el realineamiento de las fibras bajo carga o su

interacción en los puntos de unión. Además, la ley constitutiva de las fibras se mantuvo puramente elástica, evitando incorporar efectos de plasticidad o rotura de las fibras en el RVE. Es importante notar que estos fenómenos no son deseados desde el punto de vista de un injerto biomimético, ya que deben considerarse como mecanismos de falla a ser evitados en el rango de trabajo, prolongando el adecuado desempeño en servicio del reemplazo. Sin embargo, su incorporación se estudia en capítulos subsiguientes resultando en modelos micromecánicos más complejos y completos para las mallas electrohiladas.

Capítulo 3

Algoritmo de generación de geometrías para matrices fibrosas

3.1. Introducción

Las matrices electrohiladas, como se mencionó en el capítulo 1, poseen una morfología de múltiples capas planas superpuestas, siendo cada una de ellas un ensamble bidimensional no tejido de nanofibras [144]. Para poder actuar como sustitutos de materiales biológicos en ingeniería de tejidos, estas membranas deben ser capaces de soportar grandes deformaciones a nivel macroscópico, lo que implica que las fibras sobrelleven grandes desplazamientos y rotaciones, y eventualmente grandes deformaciones, a nivel microscópico. Más aún, es necesario que un injerto pretendidamente biomimético pueda desarrollar múltiples funciones, incluyendo la de permitir la adhesión y proliferación celular, brindar soporte mecánico para el neotejido, ser permeable al transporte de sustancias para promover la síntesis celular, entre otros [144]. Para alcanzar estos requisitos diversos y complementarios, es necesario incrementar el conocimiento actual acerca de los procesos físicos que ocurren a escala de las fibras bajo modos de deformación macroscópicos.

Además, se ha establecido fehacientemente que la geometría microscópica de las matrices nanofibrosas afecta significativamente la respuesta mecánica macroscópica [145, 146]. En concordancia, se ha dedicado un esfuerzo considerable a la caracterización de la geometría fibrosa, haciendo énfasis en la medición de la distribución de orientaciones de las fibras dentro del plano de cada capa, y estableciendo la correspondiente relación con la anisotropía mecánica observada mediante ensayos [78, 79, 145, 147–149].

La distribución de orientaciones es una medida muy usual para las mallas electrohiladas y da cuenta de la anisotropía lograda durante la fabricación, pudiendo obtenerse mallas totalmente isotrópicas, moderadamente alineadas o incluso completamente alinea-

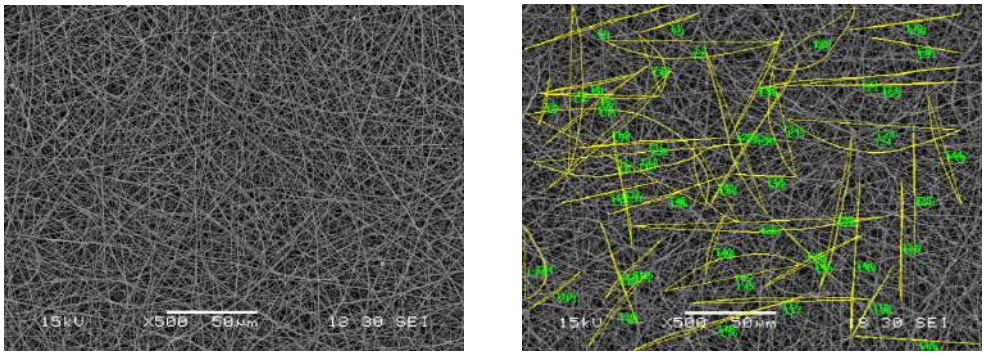
das. Además, se ha reportado que bajo tracción uniaxial se produce un incremento en la alineación de las fibras en la dirección de carga [94], un fenómeno para el cual el modelado multiescala puede resultar de gran ayuda para su comprensión. Por otro lado, no se encuentran reportados datos de distribuciones de reclutamiento, a pesar de que sí se considera al enrulamiento de las fibras como una variable de importancia. Incluso en los artículos en los que se construyen RVEs con fibras curvadas o enruladas, suele adoptarse un valor único de curvatura [94, 97] o se establece de forma cualitativa [74, 95, 101].

Un modelo multiescala que pretenda proporcionar un avance en la comprensión de los mecanismos microscópicos subyacentes reales de las matrices electrohiladas debe, por lo tanto, partir de la base de una geometría lo más cercana posible a la que arrojan las imágenes SEM detalladas en la sección 1.4.3. Las grandes dimensiones planas y estructura en forma de capas de las matrices electrohiladas, en comparación con su bajo espesor (típicamente en el orden de unas pocas décimas de mm), ha motivado la aplicación de modelos de mallas bidimensionales [101, 147, 150, 151]. Algunos métodos empleados para la construcción de geometrías de mallas fibrosas incluyen a la generación de polígonos de Thiessen (también conocidos como diagramas de Voronoi o teselación de Dirichlet) [152], generación de arreglos de fibras con orientación y curvatura prescriptas [94, 97, 153, 154], posprocesamiento de imágenes SEM para segmentar las fibras [147] y aplicación de algoritmos aleatorios de *random walk* bidimensionales [74, 95, 102]. Estos últimos permiten la construcción de geometrías que se asemejan en gran medida a las observadas en mallas electrohiladas mediante SEM.

En este capítulo se presenta un método para la generación de geometrías realistas de mallas de fibras con el objetivo de ser utilizadas para su aplicación como RVE en un modelo multiescala. Para ello se desarrolla un algoritmo de deposición virtual con la posibilidad de controlar la distribución de orientación y la distribución de enrulamiento de las fibras, entre otros parámetros de importancia, conllevando la posibilidad de obtener mallas isotrópicas o alineadas, con fibras rectas o con diferentes grados de tortuosidad. Finalmente, las geometrías obtenidas se comparan con imágenes SEM, comparando las distribuciones de orientación y de enrulamiento, para evaluar la capacidad del método para reproducir la geometría real de la microestructura electrohilada.

3.2. Descripción geométrica

En el capítulo 1 se mostró que una matriz electrohilada puede idealizarse como una superposición de capas bidimensionales (figura 1.6), donde cada capa se conforma de una malla interconectada de fibras largas y posiblemente enruladas, que adicionalmente se conectan a las fibras de capas adyacentes en los puntos de intersección. La caracterización topológica se realiza experimentalmente mediante imágenes SEM de matrices electrohiladas (figura 3.1) de donde se puede estimar el diámetro, curvatura y orientación de las fibras.



(a) (b)

Figura 3.1: Medición de orientaciones y enrulamientos de fibras a partir de una imagen SEM. Para llevarlo a cabo se seleccionan de la imagen fibras de manera aleatoria y se traza su linea de contorno abarcando la mayor longitud posible que la imagen permita sin perder la identificación de la fibra. Luego se traza la linea de extremo-extremo para poder medir su orientación y el valor de elongación de reclutamiento.

Si se realiza la medición de cada una de estas propiedades sobre un número suficientemente elevado de fibras, entonces se puede considerar que se trata de una muestra representativa de toda la población. De esta forma puede calcularse un valor medio y dispersión estándar para la distribución de orientaciones obtenida ($\hat{\mu}^\theta$ y $\hat{\sigma}^\theta$, respectivamente), así como un valor medio y dispersión estándar para la distribución de reclutamiento ($\hat{\mu}^r$ y $\hat{\sigma}^r$, respectivamente)[1]. Otra opción más detallada consiste en graficar el histograma normalizado correspondiente para visualizar la distribución discreta que caracteriza la geometría de la malla.

Sin embargo, es de importancia para la caracterización topológica de una malla fibrosa, disponer de una noción de interconectividad, ya que el comportamiento final va a

[1]En el capítulo 2 se presentó un modelo de haz de fibras donde se prescriben una distribución de enrulamientos p^r en base a parámetros de valor medio μ^r y dispersión σ^r. Aquí, en cambio, se trata con las variables $\hat{\mu}^r$ y $\hat{\sigma}^r$, correspondientes a una distribución de enrulamientos obtenida a partir de mediciones experimentales.

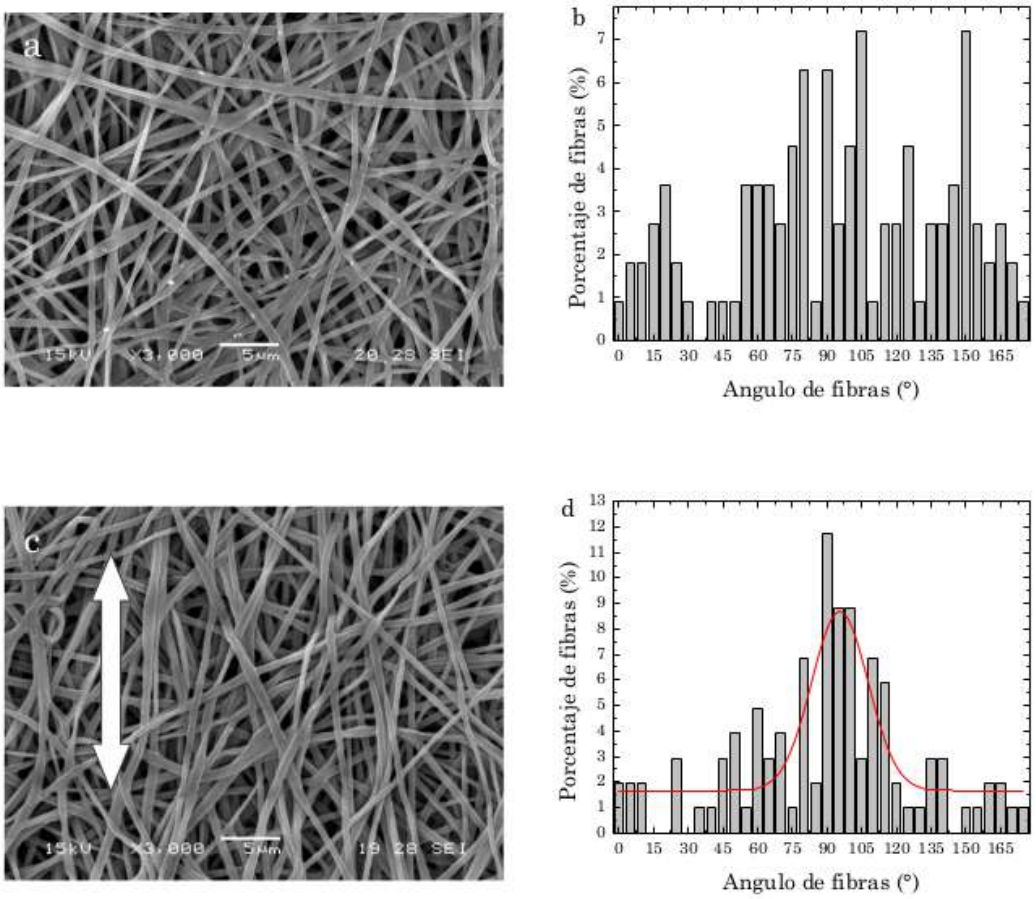


Figura 3.2: Imágenes SEM de matrices electrohiladas y sus respectivos histogramas de orientaciones. a) Imagen SEM de una matriz isotrópica y b) su histograma de orientaciones. c) Imagen SEM de una matriz moderadamente alineada en la dirección vertical (se indica en la figura) y d) su histograma junto con una regresión gaussiana (línea roja). Imagen adaptada de [155]

depender no sólo de la suma de las nanofibras, sino también de la interacción entre las mismas. Se adopta entonces una medida usual: la densidad superficial de intersecciones δ definida como la cantidad de intersecciones por unidad de superficie en una capa de fibras. No obstante, la información que puede extraerse a partir de microscopía es limitada ya que está restringida a la superficie exterior de la matriz y se presenta como una proyección bidimensional donde resulta difícil distinguir entre intersecciones aparentes e intersecciones reales donde las fibras se encuentran unidas entre sí. En este punto resulta de utilidad el modelo geométrico propuesto en esta sección, ya que permite estimar las intersecciones reales entre las fibras, considerando como puntos de unión donde se cruzan fibras que se encuentran en una misma capa o de capas adyacentes entre sí.

3.3. Construcción del RVE

3.3.1. Algoritmo de deposición virtual de fibras

En esta sección se presenta un algoritmo de deposición virtual de fibras para generar geometrías de matrices fibrosas planas simulando la deposición de las fibras en el proceso de electrohilado descripto en la sección 1.4.3, para luego conformar una geometría de matriz fibrosa tridimensional mediante la superposición de muchas capas planas.

El algoritmo busca reproducir la geometría a escala microscópica de la malla de fibras, por lo cual se delimita mediante un recinto cuadrado Ω_μ de tamaño L_μ, dentro del cual se simulará la deposición de las fibras. Cada fibra es inicialmente una curva continua lineal por tramos que se construye mediante la concatenación de segmentos lineales de igual longitud l_s. Además, como se consideran fibras muy largas con respecto al tamaño del recinto, se asume que su deposición, desde la perspectiva microscópica, comienza y termina siempre en un borde Ω_μ. Luego, el algoritmo para la deposición de una sola fibra se detalla de la siguiente manera:

1. Se selecciona de forma aleatoria un punto $\mathcal{P}_1$ sobre el borde de Ω_μ como punto inicial para la construcción de la fibra.

2. Se deposita el primer segmento de la fibra, de largo l_s, con origen en $\mathcal{P}_1$ y orientación dada por el ángulo θ_1 que se obtiene de muestrear una función distribución de orientación (FDO) prescripta (p^θ), consistiendo en una función de densidad de probabilidad de manera que la probabilidad de que el segmento inicial de una fibra cualquiera posea una orientación entre θ y $\theta + \mathrm{d}\theta$ es $p^\theta(\theta)\mathrm{d}\theta$. Por tratarse del ángulo que forma un segmento recto con la horizontal, la FDO es periódica tal que $p^\theta(\theta) = p^\theta(\theta + \pi)$, por lo que conviene restringir al ángulo θ al intervalo $[0, \pi)$.

3. Subsecuentemente, se concatena un segundo segmento con punto de origen en el punto final del segmento anterior, con un ángulo θ_2 igual a θ_1 más una posible desviación θ_{d2} que se obtiene de forma aleatoria a partir del intervalo $I_d = [-{\theta_d}^{max}, {\theta_d}^{max}]$. Por lo tanto el ángulo de orientación de este segmento es $\theta_2 = \theta_1 + \theta_{d2}$.

4. Se continua el proceso de concatenación para cada segmento subsiguiente hasta que un segmento eventualmente sale del recinto Ω_μ. Este segmento es el último de la

fibra y es recortado en el borde del recinto, de forma que la fibra recorre un camino lineal por tramos entre dos puntos del borde.

De esta forma, los parámetros del algoritmo de deposición virtual para una sola fibra son: la FDO prescripta ($p^{\theta}(\theta)$), la longitud de segmento (l_s) y el ángulo de desviación máximo (${\theta_d}^{max}$). Es válido resaltar que la curva así conformada es continua pero de derivada discontinua y que, de ser necesario, se podría realizar una interpolación mediante *splines* para culminar con una curva de mayor grado de continuidad. La figura 3.3 muestra una fibra compuesta de pocos segmentos con el objetivo de esquematizar el algoritmo previo.

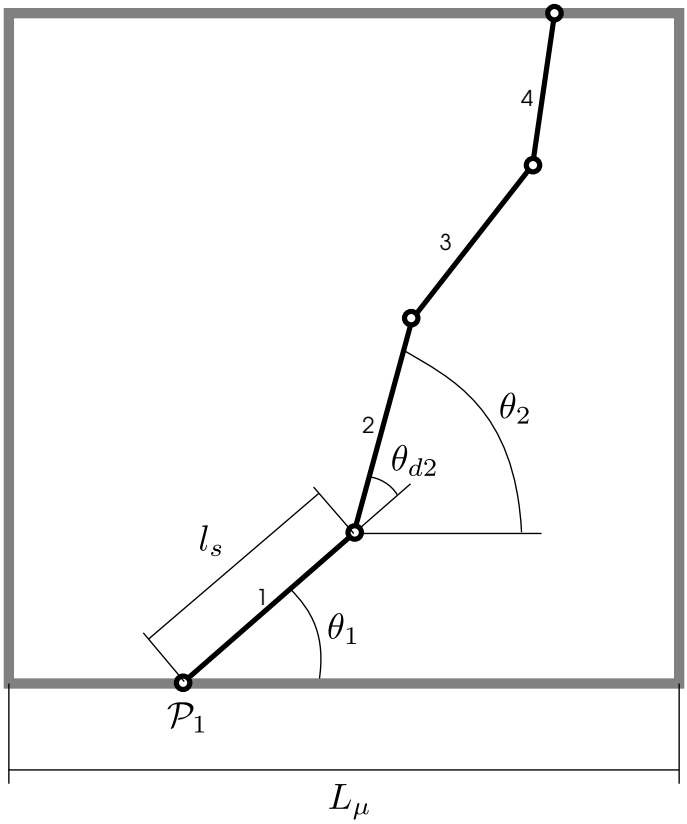


Figura 3.3: Deposición virtual de una fibra según el algoritmo detallado. Se muestra la deposición de 4 segmentos donde se detalla el punto de origen $\mathcal{P}_1$, el ángulo del primer segmento θ_1, el ángulo de desviación del segundo segmento θ_{d2} y el ángulo del segundo segmento $\theta_2 = \theta_1 + \theta_{d2}$. El cuarto y último segmento (que sale del recinto cuadrado de largo L_μ) es recortado de forma que su longitud es menor a la del resto de los segmentos (l_s).

Es importante notar que el valor de ${\theta_d}^{max}$ está relacionado con la tortuosidad de las fibras, cumpliéndose que para un valor nulo se obtendrán fibras rectas. Además, dado que las fibras observadas no poseen cambios bruscos de orientación en puntos cercanos a lo largo de su línea media, es razonable admitir que el valor de ${\theta_d}^{max}$ será pequeño (${\theta_d}^{max} < 90°$).

Resulta útil formular el algoritmo respeto de longitudes adimensionales, utilizando el diámetro promedio $\bar{D}$ de las fibras como medida de normalización. Los parámetros adimensionales se denotan mediante el símbolo $\tilde{(\cdot)}$. Entonces, el largo adimensional de cada lado del recinto es $\tilde{L}_\mu = L_\mu/\bar{D}$, mientras que cada fibra i tiene una longitud de contorno adimensional $\tilde{L}_i = L_i/\bar{D}$, una longitud extremo-extremo adimensional $\tilde{l}_i = l_i/\bar{D}$ y un diámetro adimensional $\tilde{D}_i = D_i/\bar{D}$. Cabe mencionar que en el caso particular que todas las fibras posean un mismo diámetro, todas tendrán diámetro adimensional unitario. Además, durante la construcción de las fibras, cada segmento lineal se construye con una longitud adimensional $\tilde{l}_s = l_s/\bar{D}$.

Ya establecidos los pasos para la generación de cada fibra mediante un camino aleatorio, puede ahora resumirse el algoritmo para la construcción de la geometría total de la malla como sigue:

1. Seleccionar los parámetros del algoritmo: el diámetro promedio de las fibras $\bar{D}$, la FDO prescripta $p^\theta(\theta)$, el ángulo de desviación máximo ${\theta_d}^{max}$, el tamaño de recinto $\tilde{L}_\mu$, la longitud de segmento $\tilde{l}_s$ y la fracción de volumen objetivo η.

2. Generar una capa depositando fibras mediante los pasos descritos previamente hasta alcanzar la fracción de volumen η. Se asume que el espesor de cada capa es equivalente al diámetro medio de las fibras $\bar{D}$, por lo que el volumen de la capa es $L_\mu \times L_\mu \times \bar{D}$. Luego, la fracción de volumen de la capa k es:

$$\eta_k = \frac{\sum_{i=1}^{N_i^k} V_i}{\left(L_\mu\right)^2 \bar{D}} \tag{3.1}$$

   donde $V_i = L_i \frac{\pi D_i^2}{4}$ es el volumen de la fibra i.

3. Repetir el paso 2 hasta tener el número deseado de capas (N_c), considerando cada capa nueva superpuesta sobre la capa anterior. De esta forma se tiene que la capa k está en contacto directo con las capas $k-1$ y $k+1$ (figura 3.5b).

4. Calcular los puntos de intersección entre las fibras dentro de cada capa y entre capas adyacentes. De esta forma se evita la inclusión de intersecciones aparentes en la proyección bidimensional (entre fibras que se encuentran distanciadas entre sí más de un diámetro medio en la dirección normal al plano colector) (figura 3.5c).

5. Con los puntos de intersección identificados, realizar la subdivisión de las fibras originales (fibras largas) en los puntos de unión, obteniendo así las fibras finales (fibras cortas) como elementos estructurales entre dos puntos de unión (figura 3.5c).

6. Calcular la longitud de contorno, longitud extremo-extremo y elongación de reclutamiento para cada una de las fibras cortas (figura 3.5d).

Como resultado se obtiene una geometría de red de fibras interconectadas como se muestra en la figura 3.4, donde puede observarse gran similitud de forma en una primera observación entre una geometría generada mediante el algoritmo (parámetros: $\bar{D} = 1\,\mu\mathrm{m}$, $N_c = 10$, $\tilde{L}_\mu = 100$, $\tilde{l}_s = 10$, $\theta_d{}^{max} = 20°$, $\eta = 0{,}2$, $p^\theta(\theta) = \pi^{-1}$) y la que se ve en imágenes SEM de matrices electrohiladas.

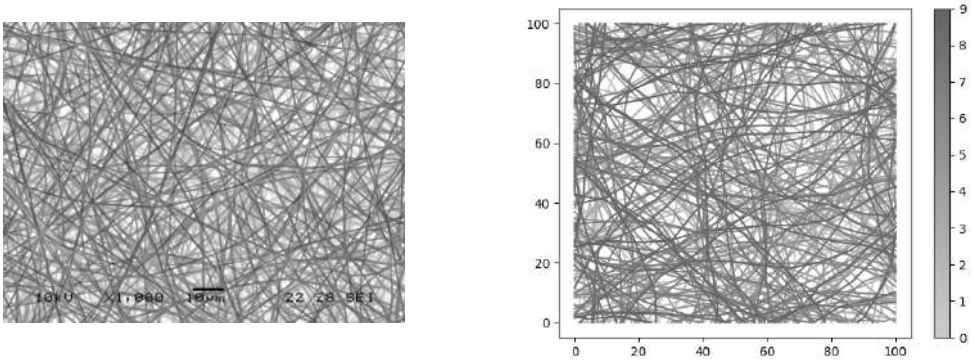


Figura 3.4: Comparación cualitativa entre una imagen SEM de una malla electrohilada de PLLA (izq., colores invertidos para mejor visualización) y una malla obtenida mediante el algoritmo de deposición virtual (der.) donde la escala de grises indica la capa de cada fibra. Puede observarse a primera vista una gran similitud entre la geometría de la malla simulada y la microestructura electrohilada.

3.3.2. Variabilidad estadística de las mallas

Por la naturaleza estocástica de la deposición de las fibras durante el electrohilado, las matrices obtenidas poseen un grado de variabilidad apreciable incluso cuando son procesadas mediante los mismos parámetros de fabricación. Aún así, desde un punto de vista macroscópico el material resulta prácticamente homogéneo y evidencia características que engloban a una población de fibras suficientemente grandes como para poder establecer la separación de escalas. De forma equivalente se desea que las geometrías generadas, para su posterior aplicación como RVE, posean un número de fibras mínimo necesario. Para ello se buscará encontrar el tamaño de las mallas lo suficientemente alto

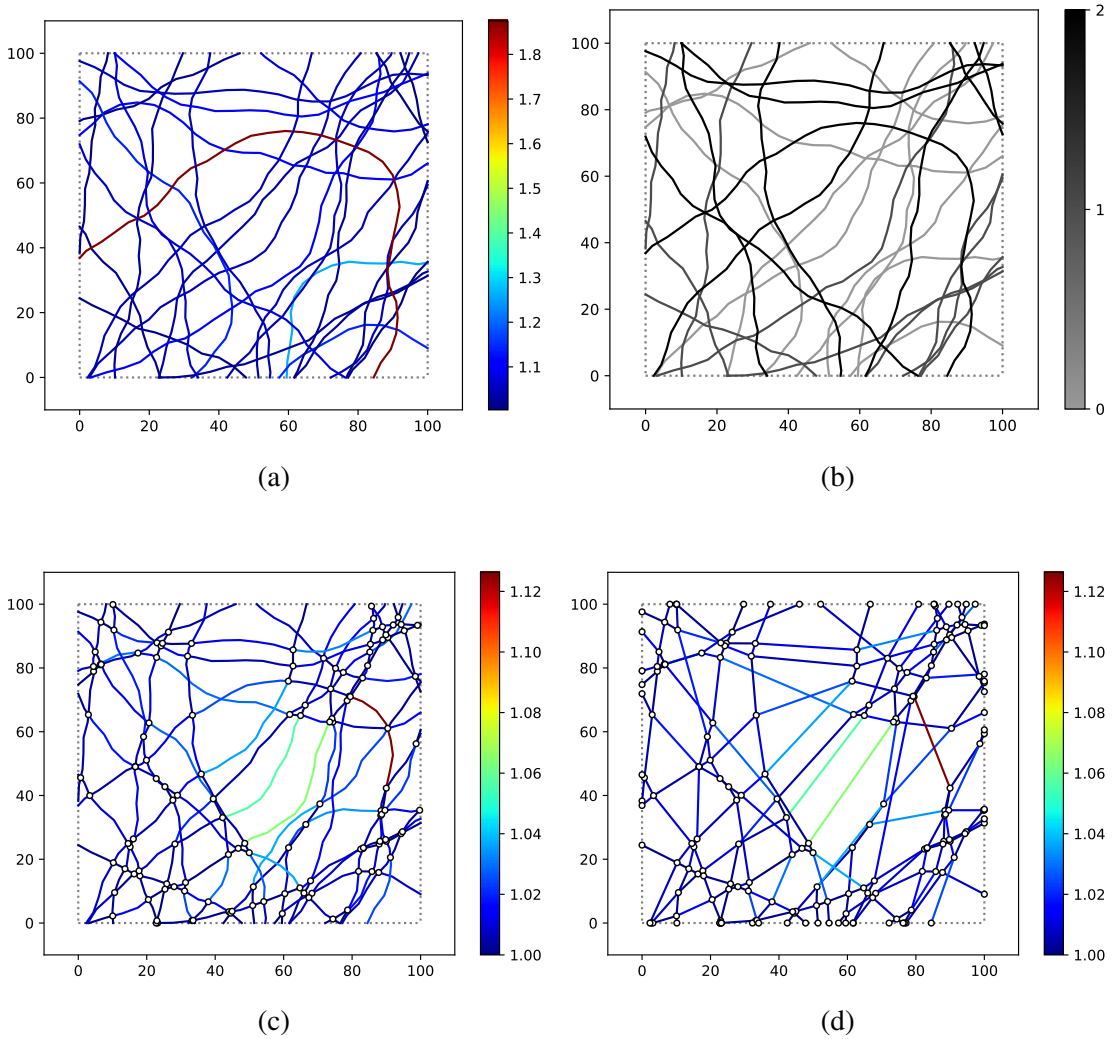


Figura 3.5: Ejemplo de una malla sencilla de tres capas y reducido número de fibras (para su mejor visualización) generada mediante el algoritmo de deposición virtual descripto ($\tilde{L}_\mu = 100$, $\tilde{l}_s = 5$, ${\theta_d}^{max} = 20°$). b) Las fibras en su estado de deposición, sin intersecciones (barra de color indicando λ^r). b) Idem pero con escala de grises indicando a qué capa corresponde cada fibra. b) Las fibras intersectadas (barra de color indicando λ^r para cada fibra comprendida entre dos puntos de unión). c) Líneas extremo-extremo de las fibras entre puntos de unión (barra de color indicando λ^r).

para poder realizar observaciones representativas del material macroscópico y no de una muestra microscópica específica.

Naturalmente, dado un valor para la fracción de volumen, el número de fibras en cada capa va a depender del tamaño de la malla, que está determinado por L_μ, $\bar{D}$ y N_c. Para evaluar si las muestras de determinado tamaño pueden ser consideradas RVEs se puede analizar la variación estadística de alguna determinada magnitud característica de la malla en un conjunto de geometrías construidas bajo los mismos parámetros. Por ejemplo, si se toma como magnitud a evaluar la densidad superficial de intersecciones (δ), entonces todos los RVE deben tener, esencialmente, el mismo valor de δ. Es decir, que si bien habrá variaciones entre distintas muestras, la variabilidad debe ser lo suficientemente pequeña para poder considerar que se trata de dos representaciones de un mismo material. Particularmente, puede requerirse que el coeficiente de variación (la relación entre la dispersión estándar y el valor medio) caiga por debajo de un umbral establecido.

3.4. Resultados

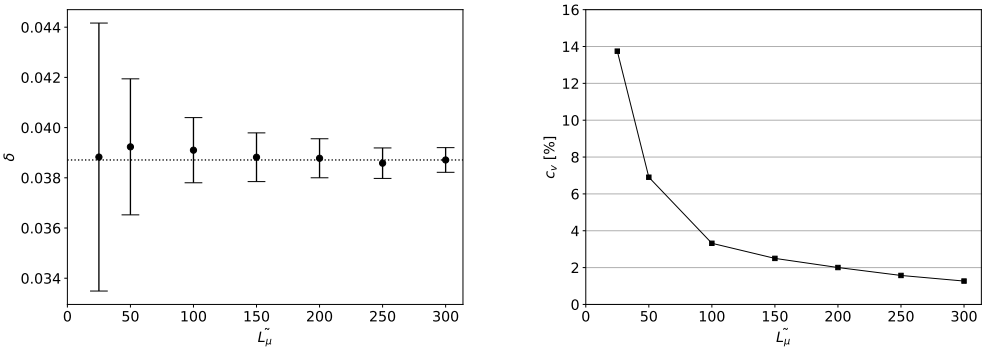
3.4.1. Variabilidad estadística y calibración del RVE

En esta sección se establece el tamaño mínimo de malla para que la misma pueda ser tratada como un RVE. Para cada tamaño de malla se construye un conjunto de 10 mallas generadas mediante el algoritmo de deposición virtual, utilizando los mismos parámetros de entrada. Luego, se estudia la variabilidad estadística de las variables de interés para cada set de parámetros calculando su valor medio y dispersión estándar dentro de cada conjunto, tomando el tamaño de RVE como la variable independiente.

En primera instancia se desea determinar el tamaño mínimo de recinto $\tilde{L}_\mu$ que permite obtener geometrías estadísticamente repetitivas. Para ello, se selecciona como variable a medir la densidad superficial de intersecciones (δ) y se mide el coeficiente de variación estadística como la relación entre la desviación estándar (μ^d) y el valor medio (σ^d) en un conjunto de 10 geometrías generadas bajo un mismo set de parámetros.

Para el primer conjunto se toman mallas isotrópicas de fibras rectas de $1\,\mu\mathrm{m}$ de diámetro: ${\theta_d}^{max} = 0$, $p^\theta(\theta) = \pi^{-1}$, $\bar{D} = 1\,\mu\mathrm{m}$, mientras que el tamaño de recinto $\tilde{L}_\mu$ se adopta como variable independiente. Para este caso la longitud de cada segmento $\tilde{l}_s$ se vuelve irrelevante. La figura 3.6 muestra los resultados obtenidos para μ^d vs. $\tilde{L}_\mu$ y para c_v versus $\tilde{L}_\mu$, donde puede observarse que la dispersión estándar disminuye a medida que se

aumenta el tamaño de recinto. Este resultado es importante ya que sugiere que se puede obtener menor variabilidad estadística en la construcción de las geometrías controlando el tamaño de las mismas. De esta forma es posible alcanzar una repetitividad estadística deseada dependiendo de la aplicación. Por ejemplo, para obtener $c_v < 4\,\%$ es necesario que $\tilde{L}_\mu \geq 100$.

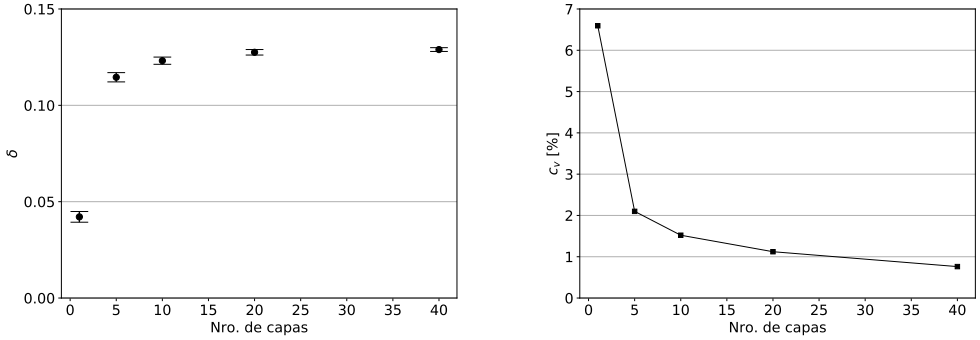


(a) (b)

Figura 3.6: Variabilidad estadística de la densidad superficial de intersecciones en función del tamaño de una malla de fibras de una sola capa. a) Valor medio μ^d (círculos) y dispersión estándar σ^d (barras de error) de la densidad de intersecciones. Se observa que la dispersión se reduce a medida que se aumenta $\tilde{L}_\mu$. b) Coeficiente de variación $c_v = \sigma^d/\mu^d$ versus $\tilde{L}_\mu$. Se observa que para $\tilde{L}_\mu = 100$ la variabilidad cae por debajo de $4\,\%$.

Si se consideran mallas conformadas por más capas, las intersecciones que se producen entre capas adyacentes entre sí, no incluidas en el caso de una sola capa, provoca valores más elevados para δ. La figura 3.7 muestra este aumento en función del número de capas para mallas generadas con $\eta = 0{,}3$, $\bar{D} = 1\,\mu\mathrm{m}$, $L_\mu = 50$, ${\theta_d}^{max} = 0$, $\tilde{l}_s = 1$. Es interesante notar que las capas intermedias incrementan en mayor medida el número de intersecciones que las capas exteriores, puesto que cada capa intermedia es adyacente a dos capas mientras que cada capa exterior es adyacente a solo una. Como resultado se tiene que al aumentar el número de capas, el valor de δ aumenta de forma monótona pero asintóticamente a un valor límite. Este comportamiento conlleva la necesidad de generar mallas con un elevado número de capas (en el caso de la figura, mayor a 20) para mantener una buena representación de una matriz real respecto de la densidad de intersecciones. No obstante, también hay que mantener en consideración el costo computacional asociado, tanto para la generación de la geometría como para su posterior análisis numérico y su implementación en un modelo multiescala mecánico.

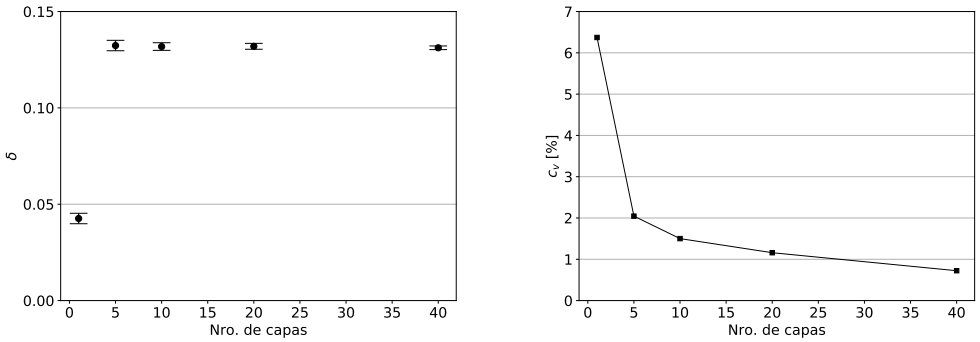
Como solución es posible tomar un número reducido de capas, pero aplicando una condición de periodicidad sobre las intersecciones, de forma que las fibras de la capa



(a) (b)

Figura 3.7: Dependencia de la densidad superficial de intersecciones respecto del número de capas de una malla virtual considerando 1, 5, 10, 20 y 40 capas ($\tilde{L}_\mu = 50$). a) Valor medio (círculos) y dispersión estándar (barras de error) de δ versus N_c. b) Coeficiente de variación de δ versus N_c.

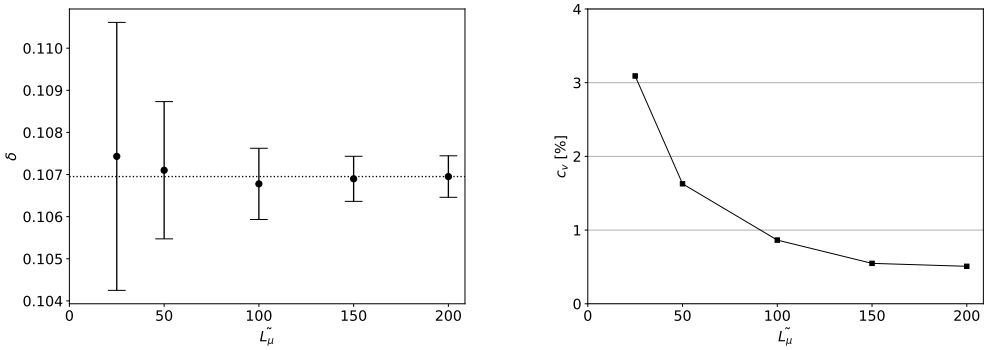
inferior intersecten a las fibras de la capa superior. De esta manera se obtiene un valor de δ independiente del número de capas, con excepción del caso obvio de una sola capa (figura 3.8).



(a) (b)

Figura 3.8: Dependencia de la densidad superficial de intersecciones respecto del número de capas de una malla virtual con la condición periódica de intersecciones, considerando 1, 5, 10, 20 y 40 capas ($\tilde{L}_\mu = 50$). a) Exceptuando el caso de una sola capa, se observa que, debido a la periodicidad en intersecciones, el valor de δ se mantiene constante sin importar el número de capas b) La variabilidad estadística se mantiene respecto del caso sin periodicidad.

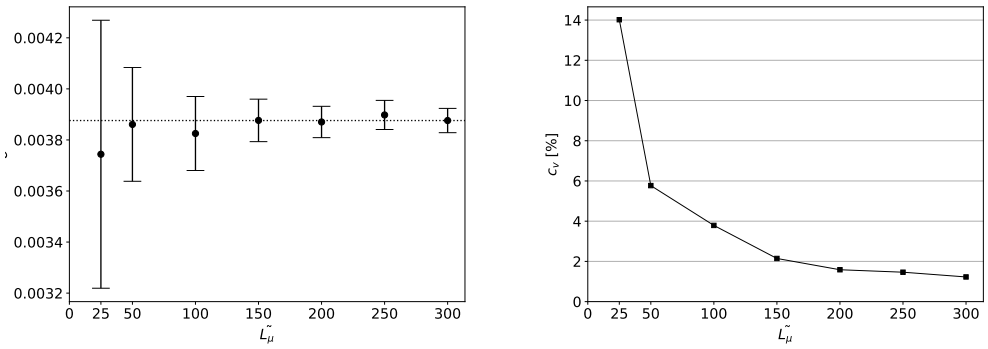
Además, dado que un mayor número de capas implica un mayor número de fibras, se observa una marcada disminución en la variabilidad estadística al aumentar el número de capas. Esto puede apreciarse en la figura 3.9b donde se ve que para $L_\mu = 100$ el valor de c_v cae por debajo del $1\,\%$, mientras que para una sola capa se mantenía por encima del $3\,\%$.



(a) (b)

Figura 3.9: Variabilidad estadística de la densidad superficial de intersecciones en función del tamaño de una malla de fibras de diez capas. a) Valor medio (círculos) y dispersión estándar (barras de error) de δ vs. $\tilde{L}_\mu$. b) Coeficiente de variación vs. $\tilde{L}_\mu$.

Adicionalmente, puede introducirse como variable la tortuosidad de las fibras considerando mallas generadas con ${\theta_d}^{max} > 0$. En este caso, como se mostró en la figura 3.12), el número de fibras es menor para un mismo L_μ dado que cada fibra enrulada ocupa más volumen que una fibra recta. Aún así, el efecto de la disminución de N_f no es apreciable en la variabilidad estadística de las geometrías generadas desde el punto de vista de δ (figura 3.10).



(a) (b)

Figura 3.10: Variabilidad estadística de la densidad superficial de intersecciones en función del tamaño de una malla de fibras enruladas (${\theta_d}^{max} = 10°$). Comparando con 3.6 puede observarse que la tortuosidad de las fibras no afecta la variabilidad estadística de la geometría generada respecto de la densidad superficial de intersecciones.

Distribución de orientaciones

También es importante calibrar el tamaño mínimo del RVE para obtener baja variabilidad en la distribución de orientaciones obtenida. Para ello se generan mallas de fibras rectas tomando una FDO prescripta (p^θ) normal truncada:

$$p^\theta(\theta) = \begin{cases} 0 & if \quad \theta < 0 \\ \left(\frac{1}{C}\right) e^{\left[-\frac{\left(\theta-\mu^\theta\right)^2}{2\left(\sigma^\theta\right)^2}\right]} & if \quad 0 \leq \theta \leq \pi \\ 0 & if \quad \theta > \pi \end{cases} \tag{3.2}$$

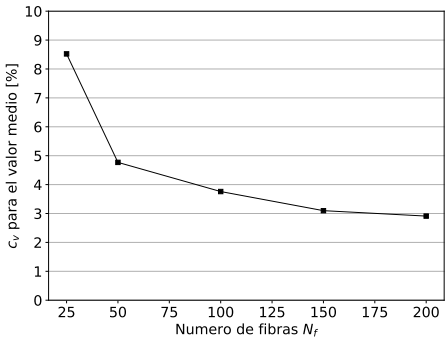
tomando $\mu^\theta = \pi/2$ y $\sigma^\theta = \pi/5$.

Para este análisis se toma como variable independiente el número de fibras y se evalúa el coeficiente de variación para los dos parámetros de la distribución obtenida ($\hat{\mu}^\theta$ y $\hat{\sigma}^\theta$) para 10 geometrías por cada valor de N_f. Los resultados muestran cómo disminuye la variabilidad estadística de las geometrías generadas a medida que se aumenta el número de fibras en la malla, siendo mayor la variabilidad para $\hat{\sigma}^\theta$ que para $\hat{\mu}^\theta$. También se observa que para $N_f = 200$ se obtienen coeficientes de variación por debajo de $5\,\%$ para los dos parámetros (figura 3.11).

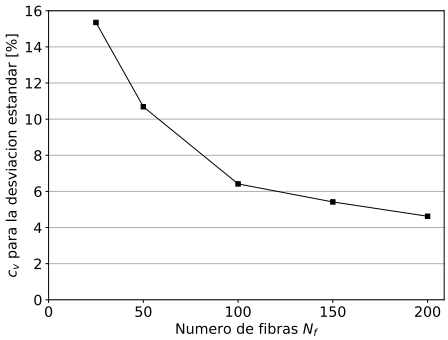
En este caso se tomó como variable independiente al número de fibras N_f que a su vez depende del tamaño de recinto L_μ y del número de capas tal como indica la figura 3.12. Para $\eta = 0{,}3$, El número de fibras puede estimarse de forma aproximada como $N_f = 0{,}4\tilde{L}_\mu N_c$, de forma que para una geometría de 200 fibras se puede realizar generando mallas de 10 capas con $\tilde{L}_\mu = 50$. Este tamaño cumple además con una baja variabilidad para la densidad superficial de intersecciones (3.9b).

Distribución de reclutamiento

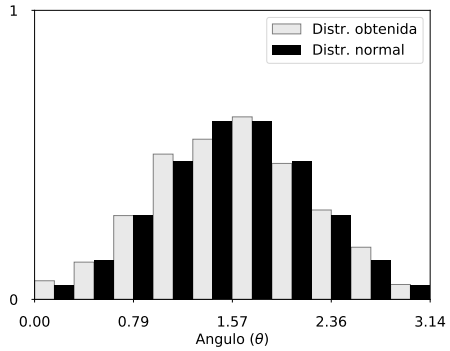
La variabilidad estadística de la distribución de reclutamiento obtenida depende en gran medida del valor de ${\theta_d}^{max}$. Valores pequeños implican mallas con fibras más rectas y baja variabilidad mientras que valores más altos generan mallas con fibras más enruladas, dando lugar también a mayor variabilidad tanto dentro de una malla como entre mallas generadas bajo los mismos parámetros (figura 3.13). Mientras que $\hat{\mu}^r$ posee muy baja variabilidad para todos los valores de ${\theta_d}^{max}$ estudiados, el coeficiente de variación para $\hat{\sigma}^r$ se mantiene por debajo de $5\,\%$ hasta ${\theta_d}^{max} = 30°$. Esto indica que si se quieren



(a)



(b)



(c)

Figura 3.11: Variabilidad estadística de la distribución de orientaciones en función del número de fibras en la malla. a) Coeficiente de variación para $\hat{\mu}^{\theta}$ versus N_f. b) Coeficiente de variación para $\hat{\sigma}^{\theta}$ versus N_f. c) Comparación entre la FDO prescripta (negro) y la obtenida (histograma normalizado, gris) para una malla de 200 fibras.

construir RVEs con distribuciones de tortuosidad más elevadas que las que se obtienen para ${\theta_d}^{max} = 30°$, será necesario incrementar el tamaño de las mismas para mejorar la representatividad estadística.

3.4.2. Diseño de RVEs

El algoritmo de deposición virtual descripto puede generar mallas con un diverso rango de geometrías, permitiendo controlar la fracción de volumen, el grado de alineación de las fibras, su enrulamiento y diámetro. También el proceso de electrohilado permite, a pesar de las limitaciones inherentes a la práctica experimental, ejercer cierto grado de control sobre algunas de estas propiedades y obtener gran diversidad en la topología microscópica. En esta sección se estudia la microestructura obtenida y se exploran las posibilidades del algoritmo de generar distintos tipos de geometrías capaces de ser utilizadas como RVEs para modelos multiescala de matrices electrohiladas.

Alineamiento de la malla

Un ejemplo es la posibilidad de controlar la velocidad de rotación del mandril colector para conseguir microestructuras con diferente nivel de alineamiento de las fibras a lo largo de la dirección circunferencial. A su vez, en el algoritmo de deposición virtual, para generar fibras con diferente grado de alineamiento a lo largo de una dirección se puede controlar la FDO prescripta. La figura 3.14 muestra mallas ($\eta = 0{,}3$, $\bar{D} = 1\,\mu\text{m}$, $\tilde{L}_\mu = 50$ y $N_c = 10$, ${\theta_d}^{max} = 5°$) con diferente grado de alineación graficadas junto con su respectiva distribución de orientaciones, donde puede observarse la capacidad del algoritmo de reproducir geometrías con creciente nivel de alineamiento.

Distribución de enrulamiento

Asimismo, es posible obtener geometrías virtuales con fibras más rectas o más enruladas controlando el ángulo de desviación máxima con la que se concatenan los segmentos lineales ${\theta_d}^{max}$ y la longitud de los segmentos l_s (figura 3.15). En todos los casos, la tendencia de la distribución es decreciente, indicando que la mayoría de las fibras posee valores de λ^r pequeños (cercanos a 1). Aún así, con ángulos ${\theta_d}^{max}$ más grandes se consiguen distribuciones con valores de reclutamiento más elevados en promedio, sin afectar la tendencia decreciente.

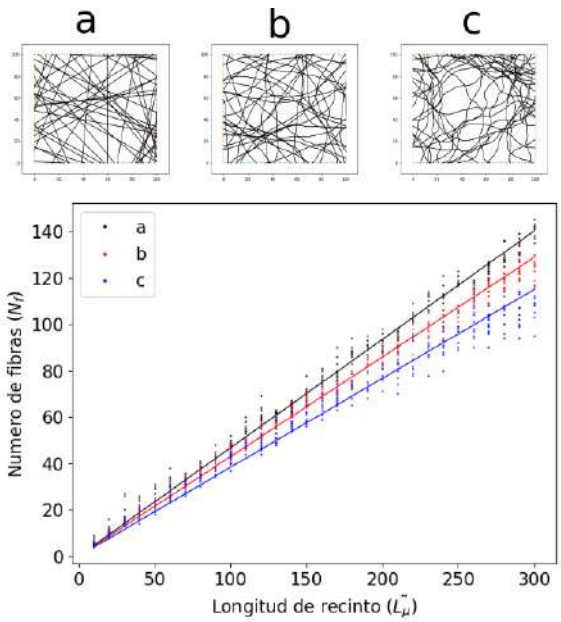
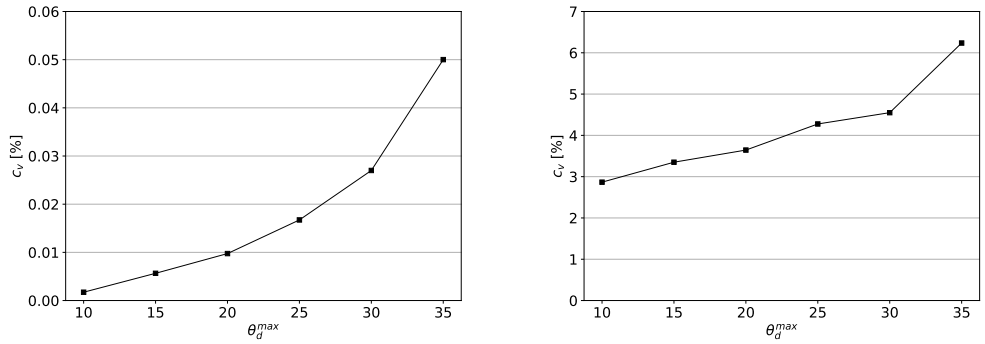


Figura 3.12: Número de fibras en una capa de fibras (N_f) en función del largo del recinto cuadrado (L_μ) y de la tortuosidad de las fibras, tomando constante la fracción de volumen como $\eta = 0{,}3$. Se observa que la cantidad de fibras sigue una relación lineal con cierta variabilidad propia del algoritmo, y que mallas con fibras más tortuosas (c) poseen en general un menor número de fibras que mallas con fibras rectas (a).



(a) (b)

Figura 3.13: Variabilidad estadística de la distribución de reclutamiento en función del ángulo de desviación máximo del algoritmo de deposición virtual. a) Coeficiente de variación para el valor medio de la distribución de reclutamiento. b) Coeficiente de variación para la dispersión estándar de la distribución de reclutamiento.

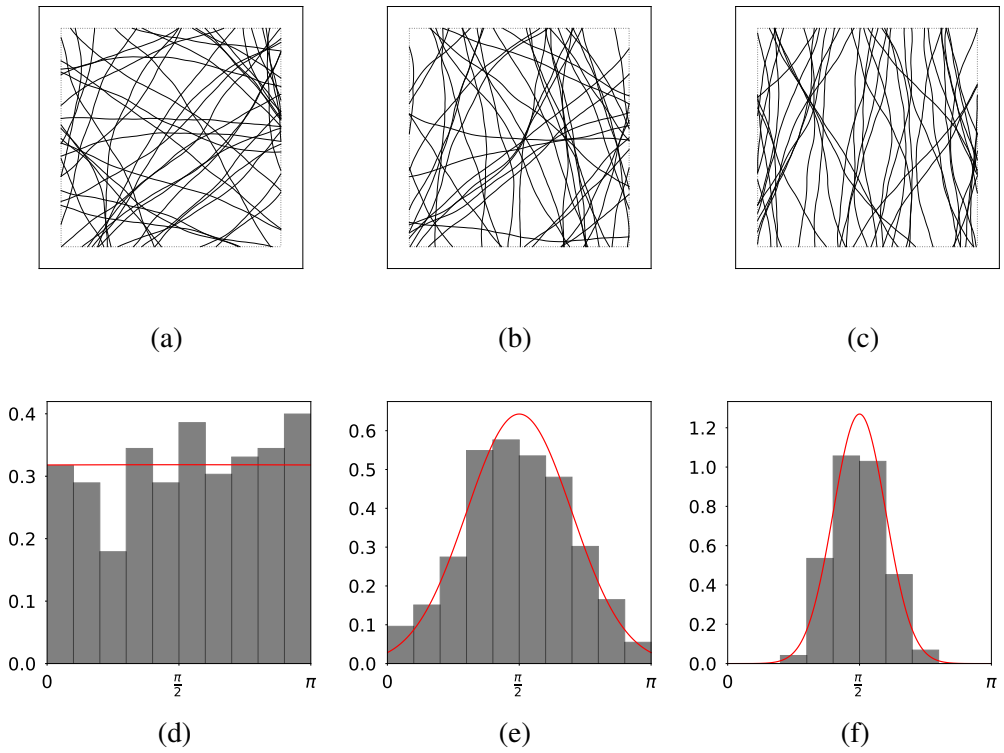


Figura 3.14: Diseño de geometrías con creciente nivel de alineamiento a lo largo de una dirección. a), b) y c) Mallas generadas con $\eta = 0{,}3$, $N_c = 10$, $\bar{D} = 1\,\mu\text{m}$, $\tilde{L}_\mu = 50$, $\tilde{l}_s = 1$, ${\theta_d}^{max} = 5°$. Se muestran solo dos capas para una mejor visualización. a) FDO uniforme. b) Alineamiento moderado: FDO normal truncada con $\sigma^\theta = \pi/5$. c) Alineamiento alto: FDO normal truncada con $\sigma^\theta = \pi/10$. b), d) y e) muestran las correspondientes distribuciones de orientaciones obtenidas (histogramas normalizados, gris) junto con la FDO prescripta (línea sólida, rojo).

Es posible diferenciar entre la población de fibras largas (tal cual son depositadas) y la población de fibras cortas (subdivididas en los puntos de unión). En el segundo caso, aunque se mantiene la misma tendencia en la distribución de λ^r, los valores son marcadamente menores en general. Esto se debe a que las intersecciones provocan la división de una fibra enrulada en varias fibras más rectas. Este resultado es muy interesante si se tiene en cuenta la posible importancia de la distribución de reclutamiento en el comportamiento no lineal de las matrices nanofibrosas como se determinó en el capítulo 2, ya que una mayor densidad de intersecciones afectaría a la distribución de reclutamiento y, consecuentemente, a la no linealidad de la respuesta mecánica.

Para verificar la posibilidad de generar geometrías que asemejen a las reales para matrices electrohiladas, se compara la distribución de reclutamiento extraída a partir de análisis óptico de imágenes SEM (figura 3.1) con la distribución obtenida para una malla virtual generada con $\bar{D} = 1\,\mu\text{m}$, $\tilde{L}_\mu = 250$, $l_s = 5$, ${\theta_d}^{max} = 17°$ y $N_c = 5$, obteniendo una buena concordancia entre los histogramas (figura 3.16).

Densidad superficial de intersecciones

Finalmente, mediante el parámetro de la fracción de volumen (η) también es posible controlar el número de fibras que conforma cada capa, dado que con un menor número de fibras por capa, se generan menos intersecciones entre fibras de una misma capa y de capas adyacentes. La figura 3.17 muestra resultados de δ versus η para mallas generadas con $\bar{D} = 1\,\mu\text{m}$, $\tilde{L}_\mu = 50$, $\tilde{l}_s = 1$, ${\theta_d}^{max} = 0$ y $N_f \approx 200$ ($N_c = 10$ para $\eta = 0{,}3$, $N_c = 15$ para $\eta = 0{,}2$ y $N_c = 30$ para $\eta = 0{,}1$). Se observa un aumento significativo de δ respecto de la fracción de volumen (3.17a) al mismo tiempo que disminuye la variabilidad estadística (3.17b).

Luego, también es posible estudiar la influencia del grado de alineamiento sobre la densidad superficial de intersecciones. A medida que se incrementa el alineamiento de la malla a lo largo de una dirección preferencial, se generan más fibras que son paralelas entre sí y que, por lo tanto, no generarán intersecciones. La figura 3.18 muestra esta disminución para mallas generadas con $\eta = 0{,}3$, $\bar{D} = 1\,\mu\text{m}$, $\tilde{L}_\mu = 50$ y $N_c = 10$, ${\theta_d}^{max} = 5°$ y diferente grado de alineamiento (figura 3.14): isotrópica (FDO uniforme $p^\theta(\theta) = \pi^{-1}$), moderada ($p^\theta(\theta)$ normal truncada con $\sigma^\theta = \pi/5$) y alta ($p^\theta(\theta)$ normal truncada con $\sigma^\theta = \pi/10$).

Tanto la dependencia de la densidad superficial de intersecciones respecto de la fracción de volumen como del grado de alineamiento de la malla resulta de importancia a la

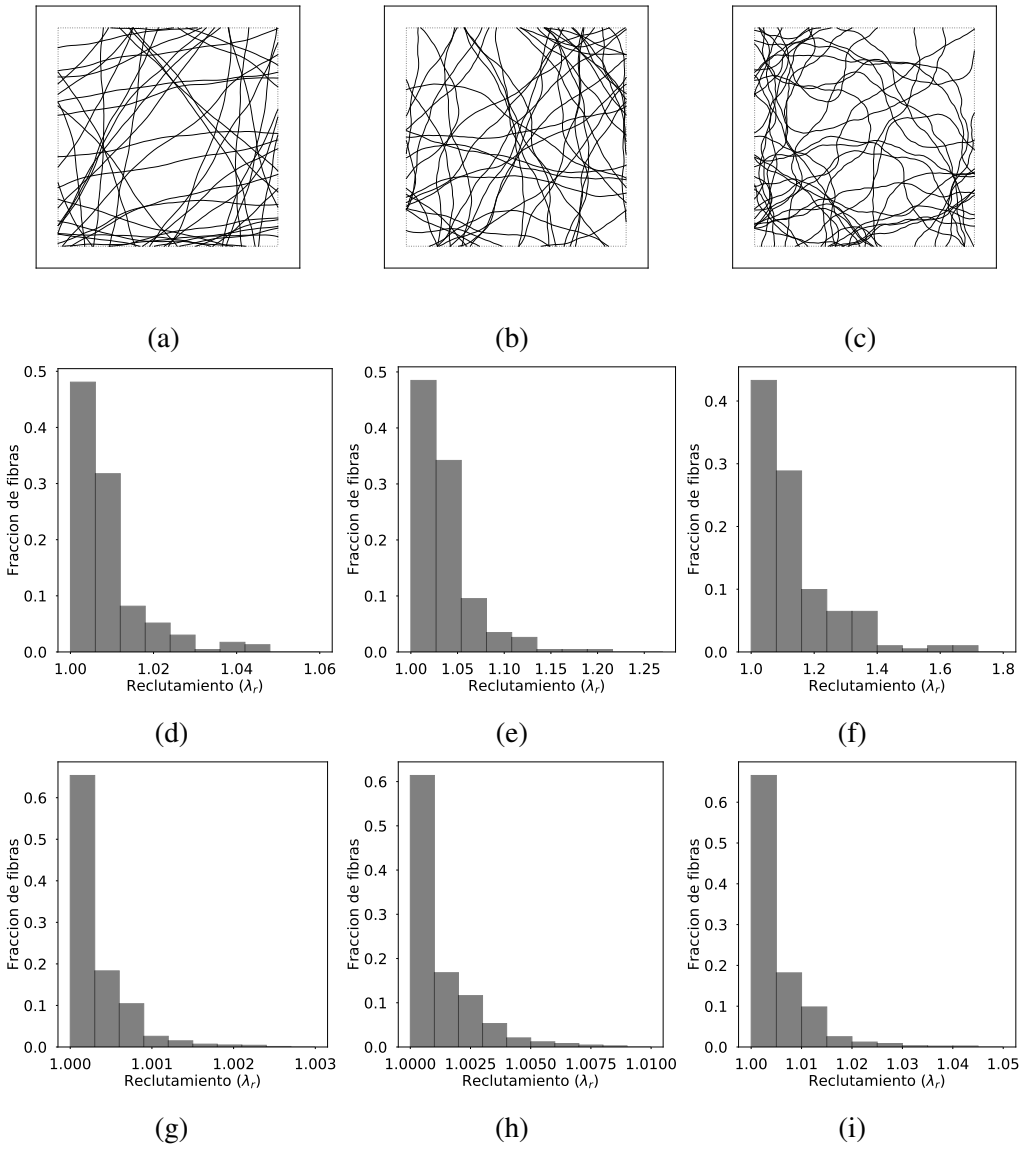
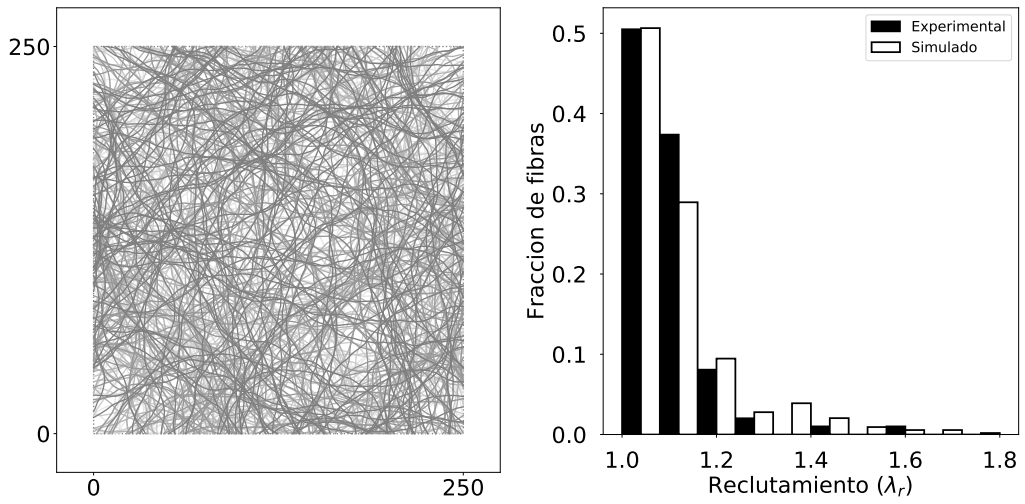
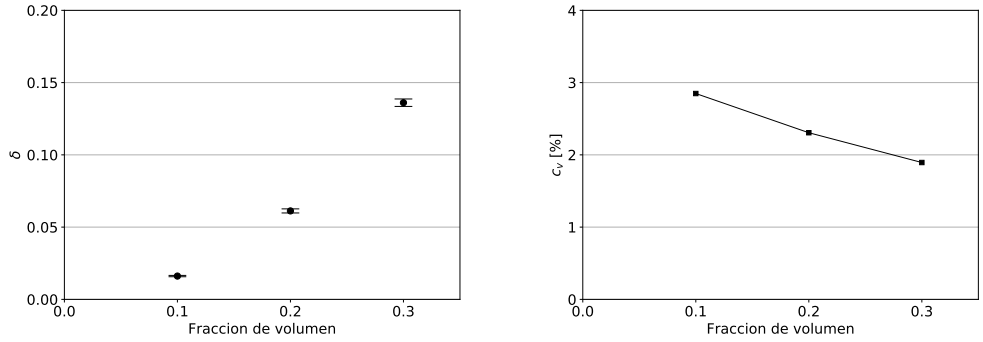


Figura 3.15: Diseño de geometrías con creciente nivel de enrulamiento. a), b) y c) Mallas generadas con $\eta = 0{,}3$, $N_c = 10$, $\bar{D} = 1\,\mu\text{m}$, $\tilde{L}_\mu = 50$, $\tilde{l}_s = 1$ y $p^\theta(\theta) = \pi^{-1}$. Se muestran solo dos capas para mejor visualización. a) ${\theta_d}^{max} = 5°$. b) ${\theta_d}^{max} = 10°$. c) ${\theta_d}^{max} = 20°$. d), e) y f) Distribuciones de reclutamiento correspondientes a cada malla, midiendo los valores de reclutamiento para cada fibra larga tal como fue depositada, sin subdividir las fibras en las intersecciones. g), h) y i) Distribuciones de reclutamiento correspondientes a cada malla, midiendo los valores de reclutamiento para cada fibra comprendida entre dos puntos unión.



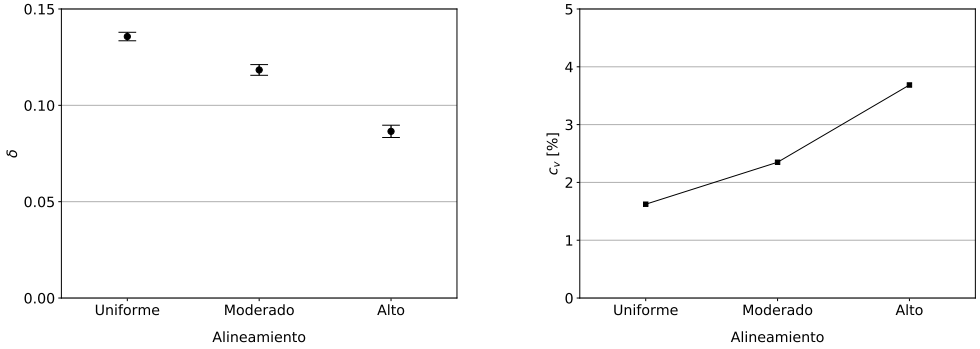
(a) (b)

Figura 3.16: Comparación de la distribución de enrulamiento obtenida para mallas generadas mediante el algoritmo de deposición virtual con datos experimentales obtenidos a partir de análisis óptico de imágenes SEM.



(a) (b)

Figura 3.17: Dependencia de la densidad superficial de intersecciones respecto de la fracción de volumen. a) Valor medio de δ en un conjunto de 10 mallas vs. η. b) Coeficiente de variación para δ versus η.



(a) (b)

Figura 3.18: Dependencia de la densidad superficial de intersecciones respecto del grado de alineamiento de la malla. a) Valor medio de δ en un conjunto de 10 mallas vs. alineamiento. b) Coeficiente de variación para δ vs. alineamiento.

hora del diseño de microestructuras con fines específicos, dado que se trata de un parámetro predominante para la respuesta mecánica de las matrices fibrosas.

3.5. Conclusiones

En este capítulo se presentó un método para generar geometrías capaces de reproducir los aspectos más importantes de la microestructura de las matrices electrohiladas, pudiendo incluso ser extendido a otros materiales con microtopología de matriz fibrosa. El método se basa en la idea fundamental del electrohilado para simular la deposición virtual de fibras en un recinto preestablecido. En consecuencia, las geometrías obtenidas poseen aspectos fundamentales de las matrices electrohiladas como fibras de gran longitud y elevada relación de aspecto, estructura de capas planas superpuestas, alineamiento controlable, distribución de tortuosidad similar y puntos de unión en las intersecciones.

Debido a la naturaleza estocástica del proceso de deposición virtual de fibras, diferentes geometrías generadas bajo los mismos parámetros evidenciaron cierto grado de variabilidad. Se realizó el estudio del coeficiente de variación estadística para los parámetros más importantes que caracterizan la geometría fibrosa, encontrando que en todos los casos la variabilidad disminuye a medida que se aumenta el tamaño de malla. Consecuentemente, se llevó a cabo la calibración del tamaño mínimo necesario para que una malla pueda considerarse un RVE, requiriendo que la variabilidad caiga por debajo de un umbral preestablecido

Por la naturaleza estocástica de la deposición de las fibras durante el electrohilado, las matrices obtenidas poseen un grado de variabilidad apreciable incluso cuando son procesadas mediante los mismos parámetros de fabricación. Aún así, desde un punto de vista macroscópico el material resulta prácticamente homogéneo y evidencia características que engloban a una población de fibras suficientemente grandes como para poder establecer la separación de escalas (es decir, la existencia de propiedades bien definidas a nivel macroscópico). De forma equivalente se desea que las geometrías generadas, para su posterior aplicación como RVEs, sean del tamaño necesario para disponer de la población de fibras suficiente tal que la variabilidad estadística entre distintas mallas caiga por debajo de un umbral preestablecido.

Un aspecto interesante del modelo propuesto es la posibilidad de estimar la densidad superficial de intersecciones, así como su interdependencia con las otras variables. La identificación experimental de las intersecciones a partir de imágenes SEM resulta difícil, ya que las imágenes se limitan a las superficies exteriores de las matrices, además de la posibilidad de incurrir en errores en diferenciar intersecciones reales de intersecciones aparentes. Las pruebas llevadas a cabo mostraron que la densidad de intersecciones

aumenta al incrementar la fracción de volumen y disminuye para mallas alineadas. Esta información puede resultar de gran importancia a la hora de querer fabricar matrices electrohiladas con distinto grado de interconectividad.

La tecnología de biomateriales actual permite ejercer cierto grado de control sobre las microestructuras obtenidas, no siendo el electrohilado una excepción. De la misma manera, se mostró cómo el algoritmo de deposición virtual puede ser utilizado para obtener geometrías de diversas características para ajustarse a diferentes casos de materiales electrohilados, incluyendo fibras de diferente diámetro, mallas de diferente fracción de volumen de fibras, isotrópicas o alineadas y mayor o menor grado de enrulamiento de las fibras. Esta capacidad del algoritmo permite la realización de ensayos *in silico* que guíen a los estudios experimentales, sugieran nuevos desarrollos y optimicen el número de casos a ensayar, señalando a qué parámetros se les debe prestar especial atención y cuáles deberían estar bajo mayor control, a la hora de lograr microestructuras específicas con el objetivo de alcanzar la biomímesis.

La geometría fibrosa obtenida puede resultar una buena idealización para muchos materiales de interés en el campo de los biomateriales, teniendo como limitación más importante la naturaleza bidimensional de las mallas, consiguiendo la tridimensionalidad solo a partir de la superposición de capas planas. Sin embargo, la estructura de capas resulta una buena aproximación para las matrices electrohiladas además de otros biomateriales como las membranas basales presentes en muchos tejidos vivos. En tales materiales, las deflexiones de las fibras por fuera de su capa resultan como máximo de unos pocos diámetros de amplitud. En el modelo propuesto las fibras se mantienen invariablemente dentro de su capa, pero forman intersecciones con fibras de capas adyacentes, es decir, a un diámetro de distancia. Esta aproximación, aunque imperfecta, es una representación razonable de la topología real. De todas maneras, la metodología presentada, puede extenderse fácilmente para considerar la interacción de capaz no adyacentes, si se introduce una probabilidad de contacto entre fibras en función de la distancia entre capas.

Capítulo 4

Modelo micromecánico para matrices fibrosas

4.1. Introducción

Los reemplazos de las arterias naturales, al igual que éstas, se encuentran sometidos a una presión fluctuante impuesta por el corazón que influye en el comportamiento de las células y los tejidos. Además, la implantación de injertos o prótesis altera la hemodinámica local y la distribución de tensiones en la pared arterial en la zona de anastomosis, causando restricciones mecánicas perjudiciales en la deformación de la arteria receptora durante el ciclo cardíaco [156]. Por lo tanto, en el campo de la ingeniería de tejidos, las propiedades mecánicas de los materiales que constituyen los injertos vasculares son un factor de relevancia en su diseño [157].

Actualmente, los injertos sintéticos aprobados clínicamente presentan una alta probabilidad de falla in vivo, debido en la mayoría de los casos a una discrepancia mecánica con la arteria nativa [157]. La discrepancia en la zona de anastomosis causa hiperplasia intimal y una reducción en la tasa de permeabilidad [18]. En particular, la discrepancia entre la compliancia del injerto vascular y la arteria nativa conduce a una falla para períodos prolongados de implantación, especialmente para conductos de pequeño diámetro [158]. En términos del comportamiento biomimético, un injerto vascular necesita una compliancia mecánica que logre un acuerdo geométrico, de tensiones y de deformaciones durante todo el ciclo pulsátil [18]. Teniendo en cuenta esto, la caracterización del desempeño mecánico de los injertos electrohilados juega un papel clave en el diseño y desarrollo de los mismos.

La utilización de simulaciones mecánicas realistas que logren vincular los fenómenos que ocurren a diferentes escalas puede resultar de gran ayuda para el diseño y manufactura de injertos vasculares de ingeniería de tejidos, especialmente si se considera

la capacidad de los métodos de fabricación actuales de ejercer control sobre propiedades microestructurales de relevancia. Más aún, un modelo multiescala que permita especificar una geometría microscópica que resulte en un comportamiento mecánico macroscópico deseado habilitaría la posibilidad de fabricar, en tiempos razonables, injertos vasculares hechos a medida no sólo desde un punto de vista geométrico sino también constitutivo.

Como se mencionó en el capítulo 1, las matrices electrohiladas poseen, al igual que muchos tejidos biológicos, una microestructura conformada por capas fibrosas superpuestas. Además, al igual que en los tejidos nativos, la heterogeneidad y anisotropía a nivel microscópico afecta el comportamiento mecánico macroscópico y la funcionalidad de los correspondientes biomateriales sintéticos. La respuesta constitutiva en tensión-deformación a nivel macroscópico es un resultado del promedio de las tensiones y deformaciones que se producen a nivel microscópico, y a su vez, la deformación de cada fibra a escala microscópica resulta de las posiciones y deformaciones de las fibras circundantes a través de las interconexiones de la malla fibrosa [159]. Claramente, para obtener un entendimiento detallado de estos fenómenos multiescala, es necesario proveer simulaciones mecánicas multiescala con un alto nivel de detalle a nivel microscópico.

En este capítulo se presenta un modelo micromecánico para mallas de fibras con el objetivo de ser implementado en un modelo multiescala para injertos vasculares electrohilados. Inicialmente se da una descripción detallada de la cinemática de una malla de fibras interconectadas y se explicitan las expresiones del equilibrio mecánico. Luego el modelo se valida con datos de ensayos experimentales de tracción uniaxial sobre matrices electrohiladas de PLLA. Finalmente, se efectúa un análisis de la evolución de la microestructura durante la tracción, evidenciando los mecanismos microscópicos que tienen lugar durante la deformación macroscópica.

4.2. Métodos experimentales

4.2.1. Materiales

Se utilizaron matrices nanofibrosas electrohiladas de PLLA (PLA2002D, Mn $78.02\,\mathrm{kg/mol}$, Mw $129.91\,\mathrm{kg/mol}$, IP 1.67) obtenidas empleando un colector plano. Las propiedades de la solución ($10\,\%$ w/v PLLA en diclorometano (DCM) y dimetilformamida (DMF) en proporción 60:40 v/v), los parámetros de procesamiento (Voltaje $10\,\mathrm{kV}$, distancia aguja-colector $10\,\mathrm{cm}$ y el caudal ($0.5\,\mathrm{ml/h}$) fueron optimizados por Montini Ballarin y colabo-

radores para obtener microestructuras nanofibrosas libres de defectos [133].

A partir de las membranas de PLLA, se separaron y prepararon muestras para ser recubiertas con oro y obtener su caracterización morfológica mediante análisis de imágenes SEM Además se prepararon probetas tipo "hueso" para ser sometidas a ensayos de tracción uniaxial (figura 4.1).

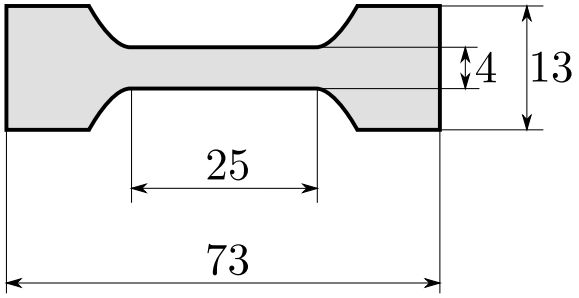


Figura 4.1: Esquema de una probeta de PLLA electrohilado para ensayos de tracción uniaxial. Las medidas se hayan en mm. El espesor se midió para cada probeta, con valores comprendidos entre $0.05\,\mathrm{mm}$ y $0.07\,\mathrm{mm}$.

4.2.2. Caracterización morfológica

La caracterización morfológica de las matrices electrohiladas de PLLA se realizó a partir de micrografías tomadas en un microscopio electrónico de barrido (JSM-6460LV, JEOL, Akishima, Tokio, Japón) luego de recubrir las muestras con oro. Las imágenes obtenidas se analizaron utilizando un software de procesamiento digital de imágenes (Image Pro Plus, Media Cybernetics Inc., USA) para medir el diámetro, la orientación y el enrulamiento de las fibras. A fin de obtener datos estadísticamente significativos se midieron 100 fibras para cada muestra, obteniendo histrogramas correspondientes a cada variable medida.

4.2.3. Caracterización mecánica

Se llevaron a cabo ensayos de tracción uniaxial con probetas tipo "hueso" con dimensiones $25\times4\times0{,}06-0{,}08\mathrm{mm}$ (alto $\times$ ancho $\times$ espesor del área calibrada) y $12{,}5\times12{,}5\mathrm{mm}$ (alto $\times$ ancho de área de sujeción), como indica la figura 4.1. Se utilizó una máquina de ensayos universales Instron EMIC 23-50 de control de desplazamiento, imponiendo una velocidad de mordaza de $6\,\mathrm{mm/min}$.

Debido a la no linealidad esperada en la deformación del material, se optó por determinar la deformación de la probeta a través de la técnica de Correlación Digital de

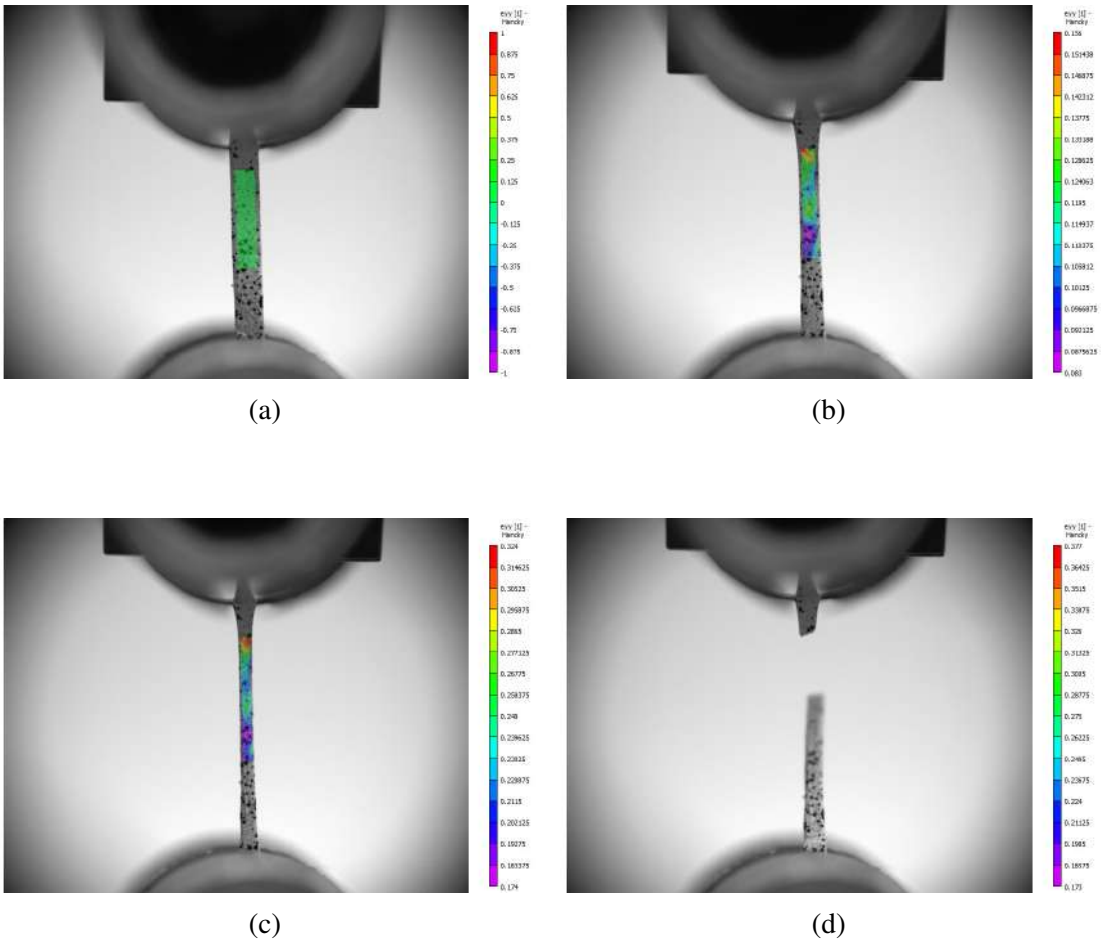
Imágenes (*Digital Image Correlation, DIC*) [160]. La técnica DIC, como se implementa en este trabajo de tesis doctoral, puede entenderse como una técnica de identificación de imágenes aplicada a la medición de la deformación de un objeto. Esta técnica es capaz de correlacionar las imágenes digitales de un objeto antes y después de la deformación y, aún más, determinar el desplazamiento y el campo de deformación de dicho objeto en función de la posición correspondiente en la imagen actual y la de referencia. También tiene ventajas de obtención de campo completo, sin contacto con la muestra y una precisión considerablemente alta para mediciones de desplazamiento y deformación [161, 162].

Figura 4.2: Ensayos de tracción uniaxial con Correlación Digital de Imágenes (DIC). La obtención del campo de deformaciones de la probeta permite establecer la deformación real de la probeta en su punto de encuellamiento, donde consecuentemente se produce la rotura.

Para este fin, previo a los ensayos, se le realiza a cada probeta un tratamiento de *speckle* con tinta para obtener un patrón de puntos distribuidos por todo su área. Se filmó la totalidad de la duración de cada ensayo empleando un sistema con una cámara de fotos de alta calidad[1] (figura 4.2 (Correlated Solutions Inc., Irmo, USA)). Luego, de manera posterior al ensayo, se realizó el procesamiento de las imágenes para obtener los campos de desplazamientos y deformaciones correspondientes utilizando el sistema Vic-2D®(Correlated Solutions Inc., Irmo, SC 29063, USA) (figura 4.3).

Luego, se utilizó esta información para conformar las curvas de tensión-deformación, utilizando la deformación en la sección de falla. Sin embargo, al momento preciso de la rotura, las grandes distorsiones que toman lugar impiden que el software logre el seguimiento de los puntos materiales, por lo que a partir de ese instante se continua la curva mediante una extrapolación empleando la información del desplazamiento de las mordazas.

[1] Resolución 2448×2048, *Frame Rate* $75\,\mathrm{Hz}$



(a) (b)

(c) (d)

Figura 4.3: Ensayos de tracción uniaxial con Correlación Digital de Imágenes (DIC). La obtención del campo de deformaciones de la probeta permite establecer la deformación real de la probeta en su punto de encuellamiento, donde consecuentemente se produce la rotura.

4.3. Resultados Experimentales

La respuesta mecánica de las matrices de PLLA mostró un comportamiento elasto-plástico con cierto grado de endurecimiento por fluencia (figura 4.4). La variación entre las curvas de tensión ingenieril versus deformación es relativamente baja, teniendo valores muy similares en el módulo elástico, tensión de fluencia y endurecimiento por plasticidad (tabla 4.1).

Las muestras presentaron una morfología nanofibrosa uniforme, con una distribución de diámetros unimodal ($463 \pm 131\,\mathrm{nm}$) y una distribución aleatoria de orientaciones ($92° \pm 53°$) (figura 4.5). Por otra parte, el histograma de reclutamiento muestra una distribución asimétrica positiva, típico de variables que presentan una cota inferior. En este caso, la elongación de reclutamiento de una fibra nunca puede ser menor a 1 (correspondiente

Tabla 4.1: Propiedades constitutivas de las matrices electrohiladas de PLLA ensayadas.

	#1	#2	#3	Promedio	Desv.Est.
Módulo elástico (MPa)	76,7	105,6	127,2	103,2	20,7
Tensión de fluencia (MPa)	1,46	1,33	1,47	1,42	0,06
Módulo post-fluencia (MPa)	6,92	5,63	6,08	6,21	0,54

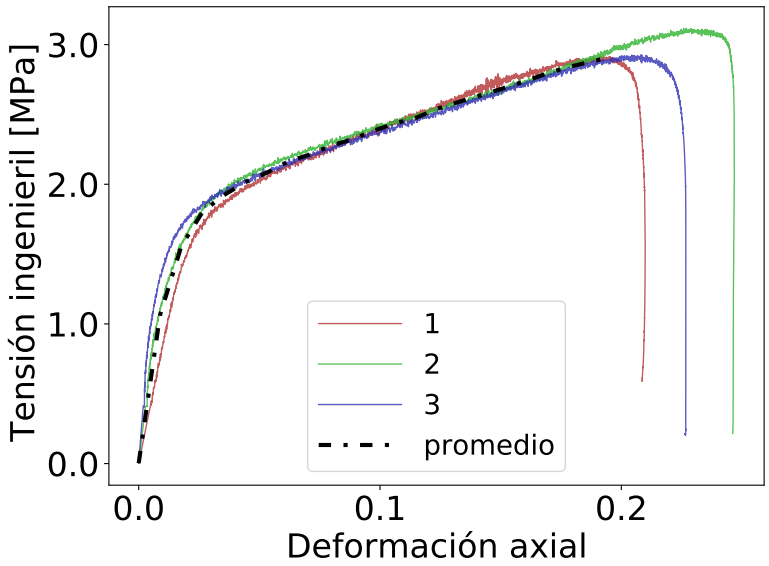


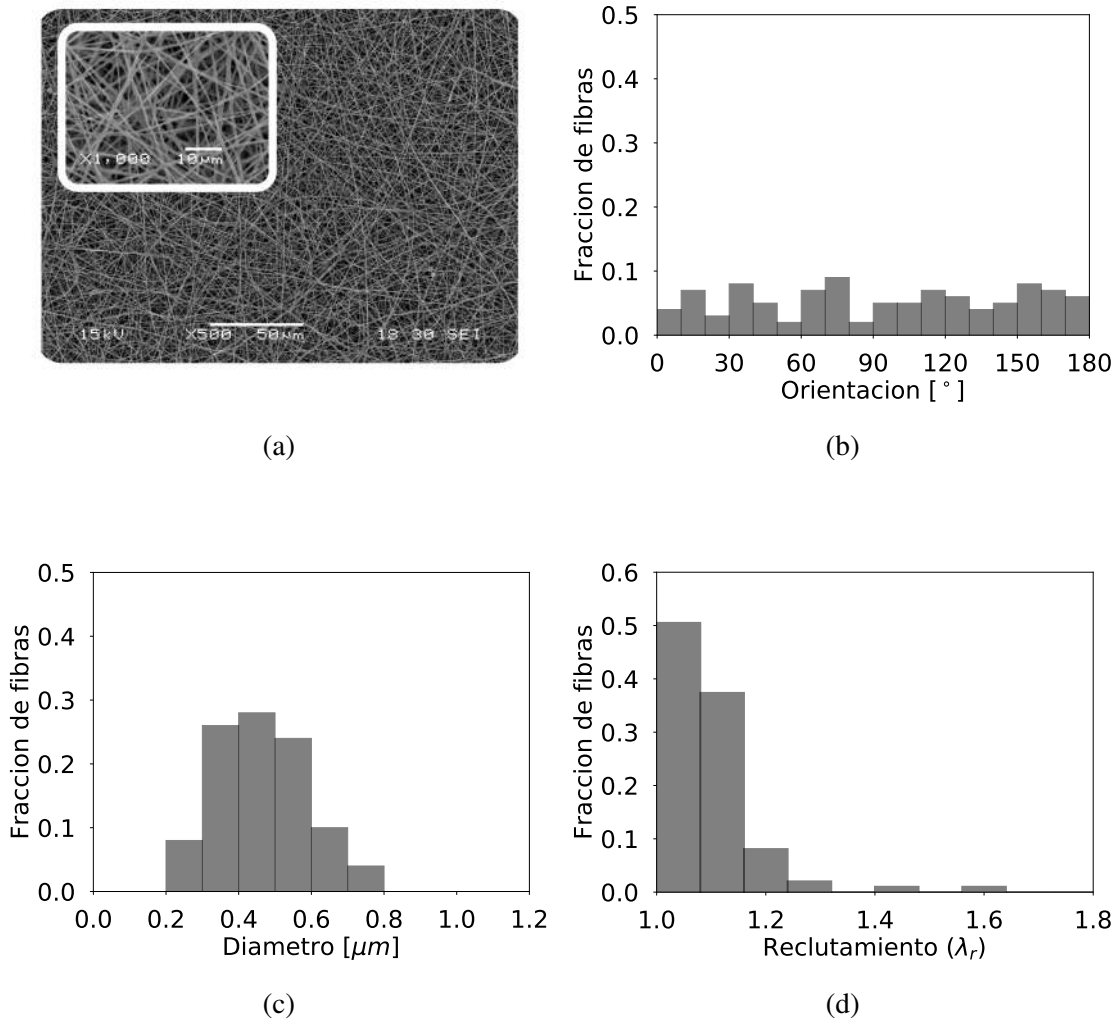
Figura 4.4: Curvas experimentales de tensión versus deformación para las probetas de PLLA. Se ensayaron 3 probetas, obteniendo una baja variación en las curvas. Se muestra también la curva experimental promedio.

a fibras en su estado de reposo) y por eso ocurre la asimetría de la distribución.

4.4. Modelo micromecánico de mallas fibrosas

En el capítulo 3 se estableció una descripción geométrica de una malla fibrosa, así como un algoritmo de deposición virtual capaz de generar geometrías que se asemejan en gran medida a las microestructuras electrohiladas. La base de esta geometría es la deposición, en un recinto previamente delimitado, de fibras largas que abarcan todo el dominio, entrando por un borde y saliendo por otro. Luego, se computan puntos de unión entre las fibras largas en las intersecciones producidas, generando subdivisiones en fibras más cortas comprendidas entre dos puntos de unión, a las que se referirá de aquí en adelante, para mejor claridad, simplemente como fibras.

Esta conceptualización arroja una microestructura virtual que consiste en una red de fibras interconectadas en sus extremos como muestra la figura 4.6, donde los elementos

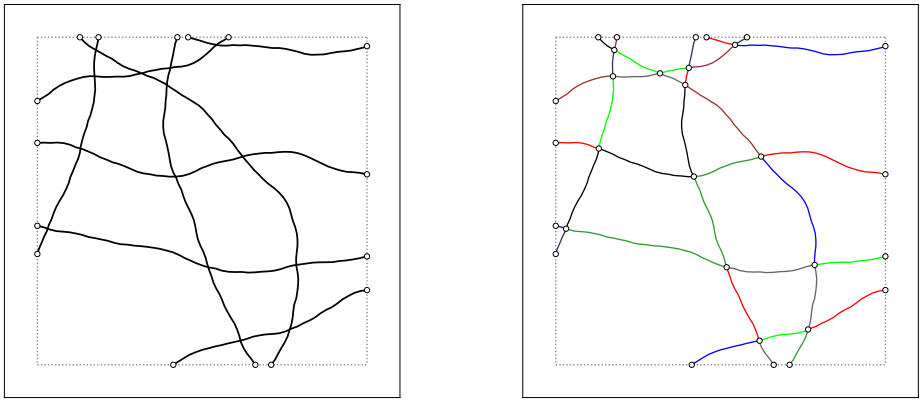


(a) (b)

(c) (d)

Figura 4.5: Caracterización morfológica de las muestras electrohiladas de PLLA. a) Imagen SEM donde se observa la microestructura nanofibrosa uniforme. b) Histograma de orientaciones (ángulo respecto de la horizontal) con una distribución aleatoria ($92° \pm 53°$). c) Histograma de diámetros con una distribución unimodal ($463 \pm 131\,\mathrm{nm}$).d) Histograma de elongaciones de reclutamiento con una distribución asimétrica positiva.

constituyentes son fibras y nodos. Los nodos pueden ser fronterizos o interiores: los fronterizos corresponden a los puntos donde las fibras entran o salen del RVE, mientras que los interiores son aquellos donde las fibras originales se intersectan.



(a) (b)

Figura 4.6: Malla de unas pocas fibras virtualmente depositadas. a) Las fibras tal como se depositan durante el proceso de deposición virtual, sin intersecciones. Los círculos indican los nodos fronterizos. b) Las fibras con intersecciones entre sí y subdivididas en fibras más cortas. Los círculos indican los nodos intersección además de los fronterizos. Los colores se introducen sólo para diferenciar entre fibras, sin responder a ningún valor.

En esta sección se presenta un modelo mecánico para este RVE, incluyendo una descripción cinemática y el planteo de las ecuaciones de equilibrio.

4.4.1. Cinemática

Cada fibra de la malla, comprendida entre dos nodos, está determinada por una curva paramétrica en el dominio bidimensional de la capa a la que pertenece. Además, puede darse una representación reducida de la fibra mediante su linea extremo-extremo y su valor de elongación de reclutamiento. De esta forma, la posición y elongación efectiva de cada fibra queda determinada enteramente por las posiciones de sus nodos extremos. Luego, el desplazamiento de la malla queda determinado únicamente por los desplazamientos de sus nodos.

Resulta conveniente establecer numeraciones para el conjunto de nodos ($\mathcal{N}$) y para el conjunto de fibras ($\mathcal{F}$) de la malla, para lo cual se establece la siguiente notación:

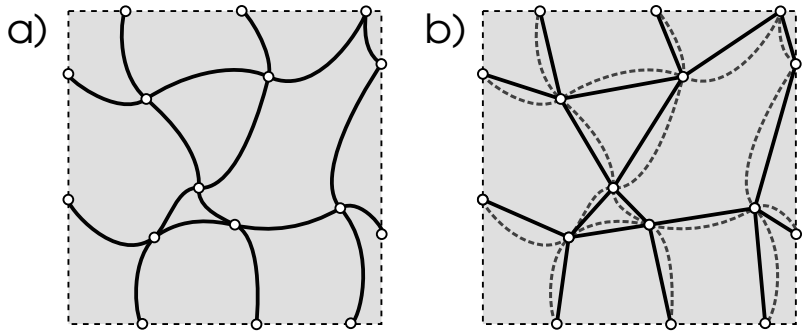


Figura 4.7: Esquema de fibras electrohiladas interconectadas en puntos de unión entre dos fibras: a) Las fibras con su geometría curvada y b) Las fibras descriptas por la linea extremo-extremo (sólida) entre nodos, además de un valor de reclutamiento como la relación entre la longitud extremo-extremo y la longitud de contorno de la fibra curva (linea punteada).

$$\mathcal{N} = \{i; i = 1, ..., N_n\} \tag{4.1}$$

$$\mathcal{F} = \{i; i = 1, ..., N_f\} \tag{4.2}$$

Cada nodo n de la malla se desplaza desde su posición inicial $\mathbf{Y}_n$ hasta su posición actual $\mathbf{y}_n$, pudiendo definir así al vector desplazamiento nodal como:

$$\mathbf{U}^{\mu}{}_n = \mathbf{y}_n - \mathbf{Y}_n \tag{4.3}$$

donde el sobreíndice μ se utiliza para indicar que se trata de desplazamientos a nivel de microescala, para diferenciarlos de los desplazamientos de la macroescala.

Mientras que el desplazamiento de la malla resulta una $N_n - tupla$ de vectores, pudiendo definirse como un elemento del siguiente espacio vectorial:

$$\mathcal{U}^{\mu} = \left\{ \mathbb{U}_{\mu} = \{\mathbf{U}^{\mu}{}_n\} ; \mathbf{U}^{\mu}{}_n \in \mathbb{R}^3, n = 1, ..., N_n \right\} \tag{4.4}$$

Dado que las fibras están definidas siempre entre dos nodos, puede identificarse cada fibra como una dupla ordenada de nodos:

$$\mathcal{F} = \{i \mapsto (i_1, i_2), i_1 \in \mathcal{N}, i_2 \in \mathcal{N}, i = 1, ..., N_f\} \tag{4.5}$$

donde i_1 e i_2 son los nodos inicial y final, respectivamente, de la fibra i.

Además, cada fibra i de la malla posee un diámetro D_i y una longitud de contorno inicial L_i^0 (longitud de arco entre dos extremos). Adicionalmente, determinados por las posiciones de sus nodos extremos puede definirse su longitud extremo-extremo inicial l_i^0 y su correspondiente versor orientación inicial ${\mathbf{a}^0}_i$ (figura 4.8):

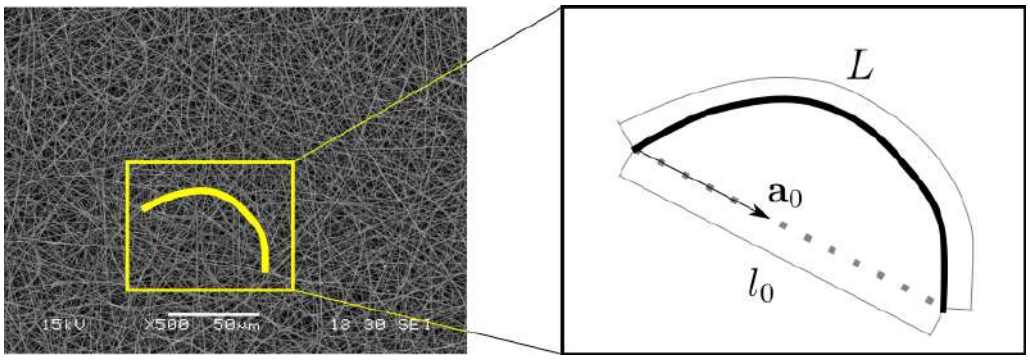


Figura 4.8: En la micrografía se identifica una nanofibra entre dos puntos de unión (izq.) y se muestra un esquema aumentado de la misma con los parámetros que determinan su configuración inicial (der.): longitud de contorno en reposo L^0, longitud extremo a extremo en reposo l^0 y un versor orientación $\mathbf{a}^0$. El diámetro D puede ser medido a partir de una imagen con mayor magnificación.

$$l_i^0 = ||\mathbf{Y}_{i_2} - \mathbf{Y}_{i_1}|| \tag{4.6}$$

$${\mathbf{a}^0}_i = \frac{\mathbf{Y}_{i_2} - \mathbf{Y}_{i_1}}{l_i^0} \tag{4.7}$$

La deformación elástica de la fibra i está determinada por tres parámetros: su longitud de contorno en reposo L_i^0, su longitud extremo-extremo en reposo l_i^0 y su longitud extremo-extremo actual l_i. Se define como medida de esta deformación a la elongación extremo-extremo:

$$\lambda_i = \frac{l_i}{l_i^0} \tag{4.8}$$

Dado que afecta significativamente su comportamiento mecánico, es importante identificar si la fibra se halla enrulada o recta. Para ello se define el valor de elongación de

reclutamiento, como el valor de λ_i para el cual la fibra deviene recta:

$$\lambda^r{}_i = \frac{L_i^0}{l_i^0} \tag{4.9}$$

Entonces, la fibra se encuentra: i) curvada o enrulada si $\lambda_i < \lambda^r{}_i$ ($l_i < L_i^0$) o ii) recta si $\lambda_i \geq \lambda^r{}_i$ ($l_i \geq L_i^0$).

Luego, en caso de existir deformación plástica, ésta resulta en un alargamiento de la longitud natural de la fibra. Como se han identificado dos medidas de longitud de referencia (L_i^0 y l_i^0), pueden establecerse una medida de elongación plástica para cada una: la elongación plástica de contorno λ_p^c y la elongación plástica extremo-extremo λ_p^e. De esta forma, las longitudes naturales o de reposo de la fibra resultan:

$$L_i^{0\prime} = \lambda_p^c L_i^0 \tag{4.10}$$

$$l_i^{0\prime} = \lambda_p^e l_i^0 \tag{4.11}$$

donde $L_i^{0\prime}$ y $l_i^{0\prime}$ representan las longitudes de reposo de la fibra luego de sufrir deformación plástica (L_i^0 y l_i^0 son los valores iniciales, previo a toda deformación plástica).

Adicionalmente, se define el vector orientación deformado $\mathbf{a}_i$ según las posiciones actuales de los nodos extremos:

$$\mathbf{a}_i = \frac{\mathbf{y}_{i_2} - \mathbf{y}_{i_1}}{l_i^0} \tag{4.12}$$

de donde puede deducirse además que $||\mathbf{a}_i|| = \lambda_i$.

Por otro lado, cada nodo n de la malla se encuentra conectado al conjunto de N_f^n fibras $\mathcal{F}_n$:

$$\mathcal{F}_n = \{i = (i_1, i_2)\,, i_1 \in \mathcal{N}, i_2 \in \mathcal{N},\ i = 1, ..., N_f, \text{ tal que } n \in (i_1, i_2)\} \tag{4.13}$$

Y cada fibra de este conjunto posee un vector orientación que puede ser saliente o entrante al nodo según los siguientes casos (figura 4.9):

$$n = i_1 \implies \text{vector orientación saliente} \tag{4.14}$$

$$n = i_2 \implies \text{vector orientación entrante} \tag{4.15}$$

Luego, resulta útil definir un vector orientación siempre saliente para la fibra i conectada con el nodo n, dado por:

$$\mathbf{a}_i^n = \begin{cases} \mathbf{a}_i & \text{si} \quad n = i_1 \\ -\mathbf{a}_i & \text{si} \quad n = i_2 \end{cases} \tag{4.16}$$

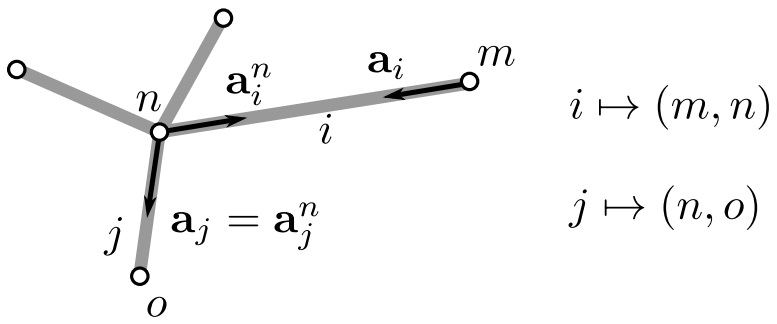


Figura 4.9: Esquema del nodo n de la malla y las fibras que se conectan con el mismo. De las cuatro fibras conectadas se indican dos con numeración i y j respectivamente. De los nodos circundantes se indican dos con la numeración m y o. Para la fibra i, el nodo n es su nodo final ($n = i_2$, su vector orientación $\mathbf{a}_i$ es entrante al nodo), por lo que se da el caso que $\mathbf{a}_i^n = -\mathbf{a}_i$ (ver ecuación 4.16). En cambio, para la fibra j el nodo n es su nodo inicial ($n = j_1$, su vector orientación $\mathbf{a}_j$ es saliente del nodo), por lo que $\mathbf{a}_j^n = \mathbf{a}_j$.

4.4.2. Equilibrio

Bajo una cierta deformación de la malla, cada fibra i desarrolla una tensión ingenieril t_i y, por lo tanto, una fuerza de magnitud $q_i = t_i \pi D_i^2/4$. Luego, el vector fuerza que la fibra i desarrolla sobre cada nodo n está dado por:

$$\mathbf{q}_i^n = q_i \frac{\mathbf{a}_i^n}{\lambda_i} \tag{4.17}$$

A su vez, el nodo n está en equilibrio si la suma de todas las fuerzas que ejercen las fibras conectadas con él se anula:

$$\sum_{i \in \mathcal{F}_n} \mathbf{q}_i^n = \mathbf{0} \tag{4.18}$$

Siguiendo con este razonamiento, el equilibrio de la malla ocurre para el elemento $\mathbb{U}_\mu \in \mathcal{U}^\mu$ tal que la sumatoria de fuerzas ejercidas por las fibras sobre cada uno de los nodos interiores sea cero:

$$\sum_{i \in \mathcal{F}_n} \mathbf{q}_i^n = \mathbf{0} \quad \forall n \in \mathcal{N} \tag{4.19}$$

La expresión 4.19 consiste en un sistema de $2N_n$ ecuaciones con $2N_n$ incógnitas si se considera una malla plana bidimensional. Para su implementación computacional resulta útil reexpresar la sumatoria sobre todas las fibras y no solamente las que se encuentran conectadas al nodo n. Para ello, en primer lugar se redefine al vector orientación saliente $\mathbf{a}_i^n$ para admitir el valor nulo en el caso que la fibra i no se encuentre conectada con el nodo n:

$$\mathbf{a}_i^n = \begin{cases} \mathbf{a}_i & \text{si} \quad n = i_1 \\ -\mathbf{a}_i & \text{si} \quad n = i_2 \\ \mathbf{0} & \text{si} \quad n \notin (i_1, i_2) \end{cases} \tag{4.20}$$

Luego, equilibrio mecánico dado por 4.19 equivale a:

$$\sum_{i \in \mathcal{F}} \mathbf{q}_i^n = \mathbf{0} \quad \forall n \in \mathcal{N} \tag{4.21}$$

Finalmente, para resolver este sistema de ecuaciones mediante un sistema iterativo de tipo Newton-Rapshon se debe plantear una linealización a partir de posiciones nodales conocidas $\mathbf{y}_j^k$, que determinan una deformación dada por el elemento ${\mathbb{U}_\mu}^k$, para obtener

las posiciones nodales de la iteración siguiente $\mathbf{y}_j^{k+1}$:

$$\sum_{i\in\mathcal{F}}\left[\mathbf{q}_i^n({\mathbb{U}_\mu}^k)+\left.\frac{\partial\mathbf{q}_i^n}{\partial\mathbf{y}_j}\right|_{{\mathbb{U}_\mu}^k}\left(\mathbf{y}_j^{k+1}-\mathbf{y}_j^k\right)\right]=\mathbf{0}\quad\forall n\in\mathcal{N} \tag{4.22}$$

4.4.3. Condiciones de contorno y ecuaciones constitutivas

El problema microscópico necesita vincularse mediante restricciones cinemáticas respecto de la deformación macroscópica. Para lograr esto se postula un modelo afín en la frontera, es decir, que todos los nodos fronterizos se tratan como nodos de Dirichlet cuyos desplazamientos van dados a partir del tensor gradiente de deformaciones macroscópico $\mathbf{F}$ según:

$$\mathbf{U}^\mu=(\mathbf{F}-\mathbf{I})\,\mathbf{Y} \tag{4.23}$$

Es necesario, además, proveer de alguna ecuación constitutiva para la tensión t desarrollada por una fibra bajo deformación. Manteniendo la conceptualización realizada en el capítulo 2, y acorde a la cinemática desarrollada en la sección 4.4.1, se asume una ley elasto-plástica según la fibra esté enrulada o recta:

$$t(\lambda)=\begin{cases}E_b\left(\frac{\lambda}{\lambda_p}-1\right) & if\quad \lambda<\lambda^r\lambda_p\\ E_b\left(\frac{\lambda^r}{\lambda_p}-1\right)+E_t\left(\frac{\lambda}{\lambda^r\lambda_p}-1\right) & if\quad \lambda\geq\lambda^r\lambda_p\end{cases} \tag{4.24}$$

donde E_b el módulo de rigidez de la fibra enrulada, E_t el módulo de rigidez de la fibra recta a la tracción y se admite para la deformación plástica un único valor $\lambda_p=\lambda_p^e=\lambda_p^c$ (ver ecuaciones 4.10 y 4.11).

Para calcular λ_p se necesita la tasa de deformación plástica $\dot{\lambda_p}$, para la cual se postula una ley constitutiva dependiente de la tensión ingenieril de la fibra:

$$\dot{\lambda_p}=a_p\sinh\left(\frac{t_i}{b_p\lambda_p^{c_p}}\right) \tag{4.25}$$

donde a_p es un parámetro de proporcionalidad con unidades s^{-1}, b_p una resistencia a la

fluencia con unidades de tensión y c_p un parámetro de endurecimiento por plasticidad.

Adicionalmente, se introduce una tensión límite t^{rot} para el cual las fibras se rompen. De esta forma, cualquier fibra de la malla que bajo deformación alcance este valor de tensión es identificada como rota y su aporte tensional se vuelve nulo para todo instante posterior. A efectos prácticos, resulta equivalente a retirar las fibras que se rompen del set $\mathcal{F}$.

4.5. Resultados

4.5.1. Comparación con datos experimentales

Para validar el modelo micromecánico propuesto se comprueba la capacidad de reproducir correctamente la respuesta mecánica de las mallas electrohiladas. Para ello se compara la respuesta homogeneizada obtenida de simulaciones con los datos experimentales ya presentados de ensayos mecánicos de tracción uniaxial (figura 4.4).

Los parámetros geométricos para la construcción del RVE se toman para reproducir correctamente las distribuciones de orientación y reclutamiento, mientras que se adopta para todas las fibras el diámetro medio (tabla 4.2). El módulo elástico de la fibras rectas a la tracción se fija en $3.0\,\mathrm{GPa}$, valor que ha sido reportado en la literatura para fibras de PLLA con diámetros cercanos al valor medido [138, 163]. El módulo elástico de las fibras enruladas, si bien es crucial para evitar indeterminaciones en los desplazamientos, es mucho menor que el de las fibras rectas y posee una baja incidencia en la respuesta mecánica tal como se vio en el capítulo 2. En este apartado se adopta un valor de $3.0\,\mathrm{MPa}$. El resto de los parámetros (a_p, b_p, c_p, t^{rot}) son optimizados por cuadrados mínimos para obtener un buen ajuste entre la curva tensión-deformación simulada y los datos experimentales (tabla 4.3). La respuesta mecánica simulada reproduce fielmente las curvas de tensión vs. deformación obtenidas experimentalmente, incluyendo el módulo elástico, tensión de fluencia, endurecimiento por plasticidad y tensión de rotura (figura 4.10).

Se halla también una discrepancia en la pendiente de la curva posterior a la rotura de las primeras fibras. Mientras que en las curvas experimentales se tiene un descenso pronunciado de la tensión indicando un rompimiento brusco de un gran número de fibras, en los resultados computacionales se obtiene un rompimiento gradual de fibras, otorgando un descenso igualmente gradual de la tensión. Es factible que esta diferencia se deba al hecho de haber realizado las simulaciones sobre un RVE en lugar de sobre un componen-

te macroscópico que emule macroscópicamente a la probeta ensayada, donde el efecto del encuellamiento genera un aumento localizado de la deformación, resultando en un rompimiento más brusco.

Tabla 4.2: Parámetros del algoritmo de deposición virtual de fibras utilizados para generar los RVE necesarios para validar el modelo micromecánico.

$\bar{D}$[µm]	η	$\tilde{L}_\mu$	$\tilde{l}_s$	${\theta_d}^{max}$	N_c	p^θ
0.45	0.1	250	5	17°	5	π^{-1}

Tabla 4.3: Parámetros optimizados para reproducir la respuesta en tensión-deformación de ensayos experimentales de tracción uniaxial.

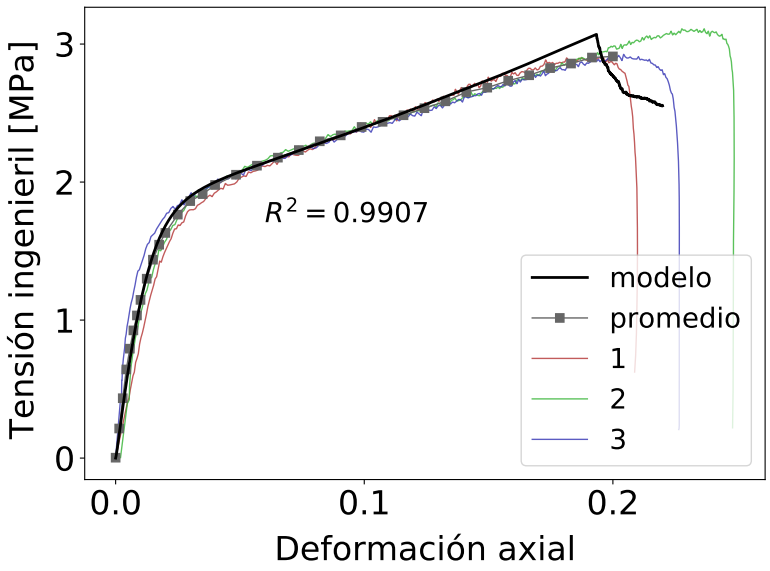
E_t[MPa]	E_b[MPa]	a_p[1/s]	b_p[MPa]	c_p	t^{rot}[MPa]
3.0×10^3	3.0	1×10^{-3}	17.0	0.5	36

Es importante resaltar que esta respuesta se obtiene con una geometría de RVE realista que concuerda con las distribuciones medidas de orientación, diámetro y enrulamiento. Además el valor optimizado para la fracción de volumen resulta razonable ya que se encuentra en el rango reportado por estudios experimentales sobre matrices electrohilada de PLLA ($\eta > 80\,\%$) [164]. Asimismo, la curva constitutiva para cada nanofibra (figura 4.10b) permanece cercana a lo reportado para nanofibras individuales de PLLA de diámetros similares [138]. Esta cuestión refuerza la capacidad del modelo multiescala como herramienta de diseño, ya que logra relacionar de forma acertada las variables microestructurales y micromecánicas con la respuesta mecánica macroscópica.

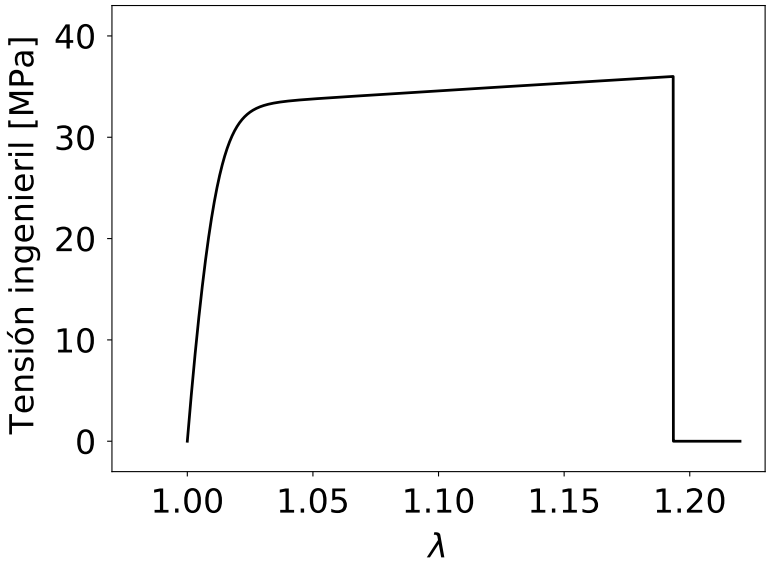
4.5.2. Análisis de la respuesta mecánica del RVE

Análisis de la cinemática microscópica bajo deformación

La deformación de la malla dada por los desplazamientos nodales para cada valor del tensor gradiente de desplazamientos macroscópico resulta muy similar a la deformación afín. En la figura 4.11 puede observarse que las posiciones nodales y las líneas extremo-extremo de las fibras entre puntos de unión mantienen gran coincidencia con sus contrapartes afines. Sólo se advierte una desviación significativa entre la configuración de equilibrio y la afín cuando se producen roturas en las fibras que alteran el balance de fuerzas de sus nodos extremos (4.11).



(a)



(b)

Figura 4.10: Comparación de la respuesta homogeneizada del RVE con curvas de tensión-deformación experimentales. a) La curva tensión-deformación simulada muestra muy buen acuerdo con las curvas obtenidas de ensayos experimentales. El ajuste con la curva promedio arroja un $R^2 = 0{,}9907$ (sin considerar la sección posterior a la rotura). b) Curva constitutiva correspondiente una fibra individual con los parámetros optimizados.

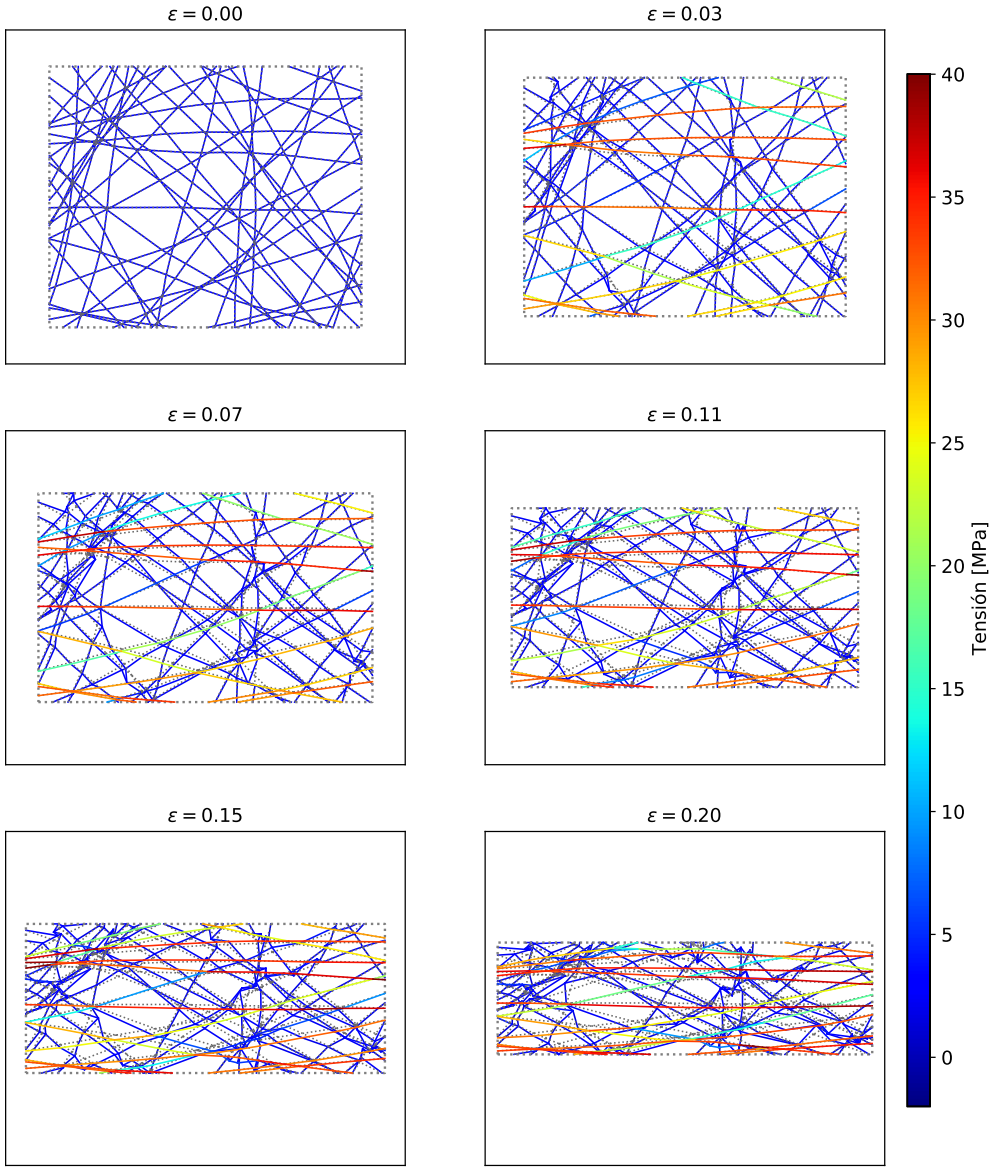


Figura 4.11: Deformación de una malla de pocas fibras (para mejor visualización) bajo tracción uniaxial (en la dirección horizontal). La barra de color indica la tensión de las fibras en MPa. Las líneas punteadas indican la deformación afín para comparación. Puede observarse cómo se forma un subgrupo de fibras alineadas a la dirección de carga que soportan la misma. Además puede verse que las fibras con orientación transversal incrementan su enrulamiento, sin reclutarse.

Adicionalmente, para cuantificar la cercanía entre las configuraciones se presentan resultados de la evolución de diferentes variables propias de la malla a medida que avanza la deformación (figura 4.12).

A medida que aumenta la deformación del RVE bajo tracción uniaxial, las fibras rotan y aumenta la alineación de la malla en la dirección de la carga. Este alineamiento por deformación se da de manera levemente más pronunciada para la malla bajo equilibrio mecánico que para el caso afín.

También puede observarse que sólo las fibras orientadas con la dirección de tracción poseen elongaciones mayores a 1. Más aún, las fibras que se encuentran orientadas transversalmente a la dirección de carga se vuelven cada vez más enruladas. Teniendo en consideración la casi nula resistencia de las fibras enruladas respecto de las que se hallan rectas y estiradas, puede inferirse que existe un subconjunto de fibras que soporta activamente la carga, mientras que el resto no realiza un aporte significativo a la resistencia mecánica.

Es importante destacar que este subconjunto no es estático, sino que por causa del alineamiento por deformación previamente descripto, aumenta en número de fibras acorde avanza la tracción. Esta descripción corresponde a un proceso microscópico de *reclutamiento de fibras por reorientación* que actúa de manera complementaria al reclutamiento de fibras por desenrulamiento ya detallado en el capítulo 2: mientras que el desenrulamiento de las fibras ocurre a bajas deformaciones ($< 0{,}05$), la reorientación se agudiza justamente a deformaciones más altas.

Es pertinente recordar que la configuración de equilibrio se obtiene de plantear los balances de fuerza nodales, mientras que la configuración afín surge de aplicar el tensor gradiente de deformación macroscópico a todo el RVE. Por ello mismo es sorpresivo el alto grado de acuerdo obtenido entre ambas configuraciones bajo deformación. La importancia de este resultado radica en poder considerar como válidos los resultados de modelos afines donde la deformación se determina directamente a partir del $\mathbf{F}$ macroscópico como fue el caso del RVE del capítulo 2.

4.6. Conclusiones

En este capítulo se presentó un modelo micromecánico para mallas de fibras interconectadas susceptible de ser utilizado como herramienta de diseño para reemplazos de ingeniería de tejidos.

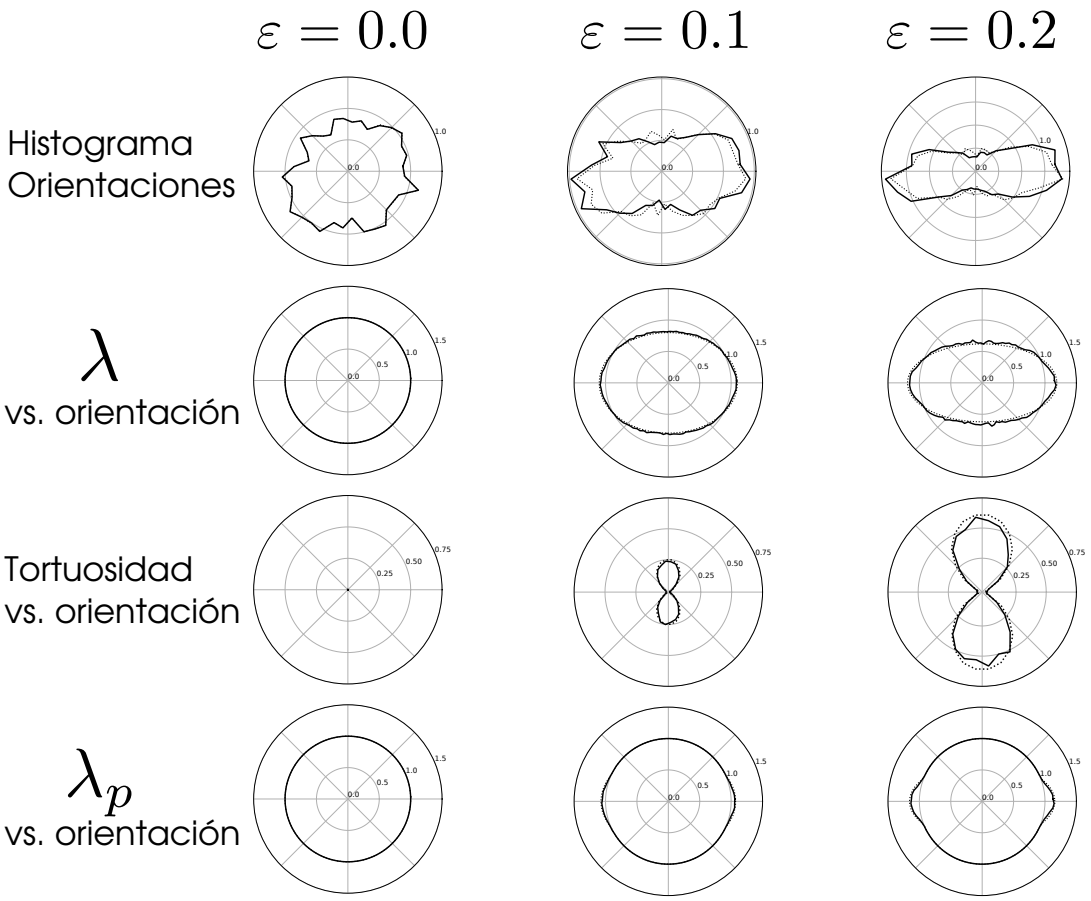


Figura 4.12: Evolución de los parámetros de la malla bajo tracción uniaxial. El histograma de orientaciones presenta, para cada ángulo de orientación, la frecuencia de fibras que ocurre en cada intervalo de $10°$. Para el resto de las variables se grafica el valor medio de la misma para las fibras que caen dentro del intervalo. La tortuosidad de cada fibra está definida como $L/l - 1$.

Para validar el modelo, se comparó la respuesta en tensión homogeneizada frente a la tracción uniaxial con datos experimentales para matrices electrohiladas de PLLA, obteniendo un buen acuerdo entre las curvas de tensión versus deformación. Más aún, la respuesta simulada se obtuvo partiendo de una geometría realista con los parámetros adecuados para simular la microestructura observada en las mismas muestras ensayadas, y tomando también una respuesta mecánica realista para las fibras individuales según información reportada en la literatura especializada.

Además, se mostró cómo el modelo permite estudiar, a medida que se produce la deformación, la evolución de los parámetros y sus distribuciones en el conjunto de fibras que caracterizan la microestructura. Este análisis evidencia los mecanismos microscópicos que tienen lugar bajo deformación macroscópica y permite relacionar los grandes desplazamientos y rotaciones de la microescala con la deformación y la respuesta constitutiva observada en la macroescala.

El enfoque multiescala aplicado permite acceder a las variables microscópicas y poder realizar ajustes sobre la microestructura así como sobre las propiedades de las nanofibras. En consecuencia, es capaz de ser utilizado para predecir *in silico* la respuesta mecánica de matrices electrohiladas con diseños a medida con anterioridad a su fabricación y caracterización morfológica y mecánica. Finalmente, es de gran relevancia práctica la baja complejidad computacional del modelo mecánico adoptado para las fibras, dado que sólo se requiere realizar el seguimiento de sus puntos extremos. Lo antedicho hace de este modelo un buen candidato para su implementación en ciclos de optimización para obtener una respuesta objetivo deseada, obteniendo como resultado las propiedades geométricas y/o mecánicas de la microestructura necesarias para fabricar injertos electrohilados paciente específicos.

Capítulo 5

Conclusiones generales

5.1. Conclusiones y resultados obtenidos

Esta tesis introduce métodos para el modelado mecánico multiescala de materiales con microestructuras nanofibrosas compuestas por capas planas superpuestas. Estos métodos tienen aplicación inmediata en el campo de la ingeniería de tejidos, especialmente en el diseño de injertos nanofibrosos con pretensiones biomiméticas. Se reseñan a continuación los principales resultados y conclusiones del trabajo:

- Se presentó un modelo constitutivo multiescala para el comportamiento mecánico elástico de matrices nanofibrosas electrohiladas con el objetivo de servir como herramienta de diseño para injertos vasculares electrohilados de ingeniería de tejidos. La escala macroscópica se modeló como un sólido continuo mientras que en la escala microscópica se implementó un RVE discreto compuesto por celdas triangulares superpuestas, dando cuenta de una microestructura conformada por una superposición de capas planas nanofibrosas. Cada fibra se modeló en base a una ley elástica bilineal llevando en consideración la diferencia en comportamiento entre fibras enruladas y rectas bajo tracción, evitándose el uso de modelos micromecánicos complejos. Conjuntamente, se tomó en cuenta el fenómeno de reclutamiento de una fibra al incrementar su módulo elástico tangente a la tracción al devenir recta. Adicionalmente se admitió el agrupamiento de nanofibras individuales en haces o fascículos como el elemento mínimo constituyente del RVE, con la ventaja de lograr implementar de forma sencilla y efectiva la distribución estadística de enrulamientos de las fibras. Mediante este enfoque estadístico se logró evidenciar, además, el fenómeno de reclutamiento progresivo bajo tracción, que toma lugar cuando las nanofibras van deviniendo rectas de manera gradual por causa de sus distintos niveles de enrulamiento. Los principales resultados obtenidos son:

  - Se validó el modelo mediante comparación con datos experimentales de ensayos mecánicos de inflado de tubos de PLLA electrohilado, consiguiendo un buen ajuste entre las curvas presión-diámetro experimentales y simuladas,
  - Se reprodujo con éxito la curva presión-diámetro (incluyendo la forma "J") para matrices fibrosas a causa del reclutamiento progresivo de las nanofibras constituyentes.
  - Se estudiaron los efectos de las variaciones microestructurales y constitutivas sobre la respuesta mecánica macroscópica, encontrando que el efecto de controlar la distribución de enrulamientos actúa de manera complementaria al de controlar la rigidez de las fibras mediante la selección de distintos polímeros de base. Esto evidenció que la capacidad de ejercer un mayor control sobre la distribución de enrulamientos de los injertos vasculares electrohilados puede ser un factor crucial en el esfuerzo por conseguir reemplazos verdaderamente biomiméticos.
  - Se optimizaron los parámetros del modelo para un injerto electrohilado capaz de imitar la respuesta mecánica en presión-deformación de una arteria intracranial humana en el rango de presiones fisiológicas. Los resultados indicaron que tal reemplazo debería poseer un mayor valor medio y una mayor dispersión en la distribución de enrulamientos de las fibras.

- Para mejorar el modelo microscópico anterior admitiendo cinemáticas más complejas y realistas, se puso en evidencia la necesidad de contar con geometrías que sean realmente representativas de la microestructura electrohilada. Para ello se desarrolló un algoritmo de deposición virtual capaz de generar geometrías virtuales que reproducen los aspectos más importantes de la microtopología de capas nanofibrosas típica de las matrices electrohiladas. El algoritmo se inspira en la deposición real de las fibras durante el proceso de electrohilado, buscando imitar este fenómeno desde una perspectiva de dominio microscópico, incluyendo la naturaleza estocástica de los caminos que forman las fibras durante su deposición. En consecuencia, se obtienen geometrías que comparten muchos aspectos de la microestructura electrohilada como: fibras de gran longitud y relación de aspecto, estructura de capas planas superpuestas, distribución de enrulamiento similar, uniones entre las fibras en los puntos de contacto y posibilidad de controlar el grado de alineamiento. Los principales resultados alcanzados son los siguientes:

  - Para establecer la validez de las geometrías generadas como elementos de volumen representativos, se llevó a cabo un estudio de la variabilidad estadística en los parámetros de importancia de las mallas virtuales, encontrando el tamaño necesario para que el dominio microscópico pueda considerarse un RVE, es decir, que la variabilidad caiga por debajo de un umbral preestablecido.
  - Se demostró la capacidad del algoritmo para obtener geometrías que reproduzcan la microestructura de mallas con distinta fracción de volumen, grado de alineamiento, fibras de diferente diámetro y de mayor o menor grado de enrulamiento. Esta versatilidad permite reproducir una amplia gama de microestructuras fibrosas, cumpliendo con dos finalidades bien diferenciadas: por un lado generar RVE específicos para evaluar la respuesta de matrices electrohiladas con microestructuras conocidas, y por otro, realizar ensayos *in silico* sobre la microestructura virtual que permitan guiar a los estudios experimentales en la búsqueda de la biomímesis, optimizando el número de casos a ensayar experimentalmente.
  - Se realizó una estimación de la densidad superficial de intersecciones presente en las capas interiores de las matrices electrohiladas. Se analizó también su interdependencia respecto de los demás parámetros geométricos, encontrando que su valor disminuye con el alineamiento de la malla y se incrementa con la fracción de volumen. Este resultado es de particular interés dado que la densidad de intersecciones es una variable de difícil acceso experimental y, a la vez, de relevancia para el comportamiento mecánico global de la matriz.

- En base a los RVE obtenidos mediante el algoritmo de deposición virtual, se desarrolló un modelo micromecánico para mallas de nanofibras interconectadas en puntos de unión o nodos. Para ello se realizó una descripción detallada de la cinemática de la malla y de las relaciones de equilibrio, además se extendió la ley constitutiva previa para contemplar la plasticidad de las fibras y la posibilidad de rotura al superar un valor de tensión límite. Como complemento, se llevaron a cabo ensayos de tracción uniaxial sobre probetas electrohiladas de PLLA para obtener curvas de tensión-deformación experimentales con las cuales validar el modelo. Los principales resultados son:

  - Se comparó la respuesta en tensión homogeneizada frente a la tracción uniaxial con datos experimentales para matrices electrohiladas de PLLA, obte-

niendo un buen acuerdo entre las curvas de tensión versus deformación. El RVE simulado se generó con parámetros geométricos que imitaron la microestructura de las matrices ensayadas. Más aún, los módulos elásticos de las fibras se adoptaron a partir de valores reportados en la bibliografía y el resto de los parámetros, que fueron ajustados para reproducir las curvas experimentales, resultaron en valores adecuados, dentro de los rangos esperados según lo reportado por estudios experimentales.

– Se estudió la evolución de la microestructura y la plasticidad de las fibras a medida que se produce la deformación. Este análisis puso en evidencia la existencia de un subconjunto de fibras, alineadas con la dirección de tracción, que soportan casi en su totalidad a la carga externa. Además, se identificó que este subconjunto crece a medida que se incrementa la deformación, pudiendo explicitar un fenómeno de alineamiento progresivo de la malla.

Como conclusión general, los métodos desarrollados permiten reproducir con alta fidelidad la microtopología nanofibrosa en su estado de deposición y simular adecuadamente la respuesta mecánica de la microestructura, mientras que se emplean parámetros geométricos y constitutivos realistas. Además, la capacidad de controlar tanto la geometría del RVE como la respuesta constitutiva de las fibras, habilita el ensayo *in silico* de matrices electrohiladas con microestructuras diseñadas a medida previamente a su fabricación. Más aún, es posible acoplar el modelo micromecánico desarrollado en ciclos de optimización con el objetivo de producir una respuesta mecánica deseada, obteniendo los parámetros geométricos y/o constitutivos necesarios para procesar injertos electrohilados verdaderamente biomiméticos.

5.2. Trabajos futuros

Durante el desarrollo de este trabajo se han identificado varias posibles direcciones para el trabajo futuro:

- La dirección más relevante para el trabajo futuro es incrementar la capacidad del modelo de reproducir con fidelidad tanto las microestructuras electrohiladas como sus respuestas micromecánicas. Si bien se hizo un avance importante y se alcanzaron resultados prometedores, el RVE propuesto permite su modificación para incorporar aspectos geométricos y constitutivos adicionales. Un punto de mejoría reside

en la determinación de los puntos de unión entre las fibras, que admite diferentes hipótesis para su realización. Por ejemplo, sería posible admitir el contacto entre fibras de capas no adyacentes a partir de una función de probabilidad. Además, la inclusión de fenómenos de fricción o viscoelásticos permitiría simular de manera realista los ciclos de histéresis típicos de estos materiales. Estos fenómenos podrían implementarse tanto a nivel constitutivo de las fibras como a nivel de la interacción, considerando enlaces con resbalamiento con fricción, o incluso falla por rotura en las uniones. Para estos fines resulta necesario proveer al modelado de la información experimental relevante para su validación, lo que requiere un trabajo mancomunado entre simulaciones, caracterización morfológica y ensayos mecánicos.

- Una vez extendida la capacidad del modelo, sería factible avanzar hacia simulaciones que reproduzcan la respuesta mecánica bajo solicitaciones de índole cíclica, con el fin de simular con fidelidad el comportamiento en condiciones hemodinámicas. La incorporación de esta capacidad permitiría la realización de ensayos *in silico* para evaluar la funcionalidad de diferentes materiales y microestructuras como injertos vasculares reales.

- En linea con lo anterior, aparece la posibilidad de un trabajo coordinado entre el modelado y la experimentación, para realizar ciclos de fabricación-ensayo-optimización utilizando el modelo propuesto como herramienta de diseño para determinar las propiedades microestructurales necesarias de los injertos. En el capítulo 2 se mostró esta capacidad del modelo para el caso de un posible injerto de PCL electrohilado capaz de reemplazar una arteria intracranial humana. Esta metodología podría extenderse a diferentes tejidos arteriales, ya sea para distintos pacientes o distritos vasculares, empleando también un *vademecum* de materiales preestablecido para seleccionar los parámetros constitutivos. Desde el punto de vista experimental sería necesario controlar la distribución de enrulamiento de las nanofibras, lo que podría conseguirse mediante la técnica novedosa de *Melt Electrospinning Writing*.

Referencias

[1] Chin Siang Ong, Xun Zhou, Chen Yu Huang, Takuma Fukunishi, Huaitao Zhang, and Narutoshi Hibino. Tissue engineered vascular grafts: current state of the field. *Expert Review of Medical Devices*, 14(5):383–392, 2017. ISSN 17452422.

[2] World Health Organization. *World health statistics 2018: monitoring health for the SDGs, susteinable development goals*. World Health Organization, 2018. ISBN 978-92-4-156558-5. Licence: CC BY-NC-SA 3.0 IGO.

[3] Ministerio de Salud. Mortalidad, 2013. http://www.msal.gob.ar/ent/index.php/vigilancia/areas-de-vigilancia/mortalidad.

[4] Ramak Khosravi, Cameron A Best, Robert A Allen, Chelsea E T Stowell, Ekene Onwuka, Jennifer J Zhuang, Yong-Ung Lee, Tai Yi, Matthew R Bersi, Toshiharu Shinoka, Jay D Humphrey, Yadong Wang, and Christopher K Breuer. Long-Term Functional Efficacy of a Novel Electrospun Poly(Glycerol Sebacate)-Based Arterial Graft in Mice. *Annals of Biomedical Engineering*, 44(8):2402–2416, aug 2016. ISSN 1573-9686.

[5] Takuma Fukunishi, Cameron A. Best, Tadahisa Sugiura, Toshihiro Shoji, Tai Yi, Brooks Udelsman, Devan Ohst, Chin Siang Ong, Huaitao Zhang, Toshiharu Shinoka, Christopher K. Breuer, Jed Johnson, and Narutoshi Hibino. Tissue-engineered small diameter arterial vascular grafts from cell-free nanofiber PCL/chitosan scaffolds in a sheep model. *PLoS ONE*, 11(7):1–15, 2016. ISSN 19326203.

[6] Robert Langer and Joseph P Vacanti. Tissue Engineering. *Science*, 260(5110): 920–926, 1993. ISSN 00368075, 10959203.

[7] Vincenzo Vindigni, Giovanni Abatangelo, and Franco Bassetto. New developments in tissue engineering of microvascular prostheses. In Rosario Pignatello, editor, *Biomaterials Science and Engineering*, chapter 21. IntechOpen, Rijeka, 2011.

[8] Dawit G. Seifu, Agung Purnama, Kibret Mequanint, and Diego Mantovani. Small-diameter vascular tissue engineering. *Nature Reviews Cardiology*, 10(7):410–421, 2013. ISSN 17595002.

[9] Wei Wu, Robert A. Allen, and Yadong Wang. Fast-degrading elastomer enables rapid remodeling of a cell-free synthetic graft into a neoartery. *Nature Medicine*, 18(7):1148–1153, 2012. ISSN 10788956.

[10] Ehsan Benrashid, Christopher C. McCoy, Linda M. Youngwirth, Jina Kim, Roberto J. Manson, James C. Otto, and Jeffrey H. Lawson. Tissue engineered vascular grafts: Origins, development, and current strategies for clinical application. *Methods*, 99:13–19, 2016. ISSN 10959130.

[11] Wojciech Mrówczyński, Damiano Mugnai, Sarra De Valence, Jean Christophe Tille, Ebrahim Khabiri, Mustafa Cikirikcioglu, Michael Möller, and Beat H. Walpoth. Porcine carotid artery replacement with biodegradable electrospun poly-e-caprolactone vascular prosthesis. *Journal of Vascular Surgery*, 59(1):210–219, 2014. ISSN 07415214.

[12] Andreas Greiner and Joachim H. Wendorff. Electrospinning: A fascinating method for the preparation of ultrathin fibers, 2007. ISSN 14337851.

[13] Anwarul Hasan, Adnan Memic, Nasim Annabi, Monowar Hossain, Arghya Paul, Mehmet R. Dokmeci, Fariba Dehghani, and Ali Khademhosseini. Electrospun scaffolds for tissue engineering of vascular grafts, 2014. ISSN 18787568.

[14] David L. Butler, Steven A. Goldstein, and Farshid Guilak. Functional Tissue Engineering: The Role of Biomechanics. *Journal of Biomechanical Engineering*, 122 (6):570–575, dec 2000. ISSN 0148-0731.

[15] Y. C. Fung and S. C. Cowin. Biomechanics: Motion, Flow, Stress, and Growth. *Journal of Applied Mechanics*, 60(2):567–567, jun 1993. ISSN 0021-8936.

[16] Michael S. Sacks. Biaxial mechanical evaluation of planar biological materials, 2000. ISSN 03743535.

[17] J. D. Humphrey. Vascular adaptation and mechanical homeostasis at tissue, cellular, and sub-cellular levels. *Cell Biochemistry and Biophysics*, 50(2):53–78, 2008. ISSN 10859195.

[18] William M. Abbott, Joseph Megerman, Jonathan E. Hasson, Gilbert L'Italien, and David F. Warnock. Effect of compliance mismatch on vascular graft patency. *Journal of Vascular Surgery*, 5(2):376–382, 1987. ISSN 07415214.

[19] Richard L. Binns, David N. Ku, Mark T. Stewart, Joseph P. Ansley, and Kellie A. Coyle. Optimal graft diameter: Effect of wall shear stress on vascular healing. *Journal of Vascular Surgery*, 10(3):326–337, 1989. ISSN 07415214.

[20] V. G. Kouznetsova, M. G. D. Geers, and W. A. M. Brekelmans. Computational homogenisation for non-linear heterogeneous solids. In *Multiscale Modeling in Solid Mechanics*, pages 1–42. Imperial College Press, 2010.

[21] M. Lei, D. P. Giddens, S. A. Jones, F. Loth, and H. Bassiouny. Pulsatile Flow in an End-to-Side Vascular Graft Model: Comparison of Computations With Experimental Data. *Journal of Biomechanical Engineering*, 2001. ISSN 01480731.

[22] Fernando Cacho, Manuel Doblaré, and Gerhard A. Holzapfel. A procedure to simulate coronary artery bypass graft surgery. *Medical & Biological Engineering & Computing*, 2007. ISSN 0140-0118.

[23] S. A. Urquiza, P. J. Blanco, M. J. Vénere, and R. A. Feijóo. Multidimensional modelling for the carotid artery blood flow. *Computer Methods in Applied Mechanics and Engineering*, 2006. ISSN 00457825.

[24] P. J. Blanco and R. A. Feijóo. A dimensionally-heterogeneous closed-loop model for the cardiovascular system and its applications. *Medical Engineering and Physics*, 2013. ISSN 13504533.

[25] Pablo J. Blanco, Gonzalo D. Ares, Santiago A. Urquiza, and Raúl A. Feijóo. On the effect of preload and pre-stretch on hemodynamic simulations: an integrative approach. *Biomechanics and Modeling in Mechanobiology*, 15(3):593–627, 2016. ISSN 16177940.

[26] Tamar B. Wissing, Valentina Bonito, Carlijn V. C. Bouten, and Anthal I. P. M. Smits. Biomaterial-driven in situ cardiovascular tissue engineering—a multidisciplinary perspective. *npj Regenerative Medicine*, 2017. ISSN 2057-3995.

[27] Gerhard A. Holzapfel and Ray W. Ogden. Constitutive modelling of arteries. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 466(2118):1551–1597, 2010. ISSN 14712946.

[28] G a Holzapfel, T C Gasser, and R W Ogden. A new constitutive framework for arterial wall mechanics and a comperative study of material models. *J. Elasticity*, 61:1–48, 2000.

[29] C.V.C. Bouten, P.Y.W. Dankers, A. Driessen-Mol, S. Pedron, A.M.A. Brizard, and F.P.T. Baaijens. Substrates for cardiovascular tissue engineering. *Advanced Drug Delivery Reviews*, 63(4-5):221–241, apr 2011. ISSN 0169409X.

[30] Johannes A. G. Rhodin. Architecture of the Vessel Wall. In *Comprehensive Physiology*, pages 1–31. John Wiley & Sons, Inc., Hoboken, NJ, USA, dec 1980.

[31] Bruce Furie and Barbara C. Furie. Mechanisms of thrombus formation. *New England Journal of Medicine*, 359(9):938–949, 2008. ISSN 15334406.

[32] Eric J. Topol, editor. *The Topol Solution: Textbook of Cardiovascular Medicine, Third Edition*. Lippincot Williams and Wilkins, 2006. ISBN 9780781770125.

[33] P. Libby, E. N. Antman, E. Braunwald, A. P. Selwyn, D. S. Baim, and W. Grossman. Enfermedades vasculares. In A.S. Fauci, E. Braunwald, K.J. Isselbacher, J.D. Wilson, J.B. Martin, D.L. Kasper, S.L. Hauser, and D.L. Longo, editors, *Harrison - Principios de Medicina Interna, Volumen 1, 14a Edicion*, pages 1546–1574. McGraw Hill, 1998.

[34] J. T. Willerson and D. C. Sabiston. Trastornos coronarios. In J.B. Wyndgaarden and L.H. Smith, editors, *Cecil - Tratado de medicina interna, Volumen 1, 18a Edicion*, pages 362–379. Interamerica - McGraw Hill, 1991.

[35] Charles C. Canver. Conduit Options in Coronary Artery Bypass Surgery. *Chest*, 108(4):1150–1155, oct 1995. ISSN 00123692.

[36] Ruben Y. Kannan, Henryk J. Salacinski, Peter E. Butler, George Hamilton, and Alexander M. Seifalian. Current status of prosthetic bypass grafts: A review. *Journal of Biomedical Materials Research Part B: Applied Biomaterials*, 74B(1):570–581, jul 2005. ISSN 1552-4973.

[37] Christopher D. Owens, Nicole Wake, Michael S. Conte, Marie Gerhard-Herman, and Joshua A. Beckman. In vivo human lower extremity saphenous vein bypass grafts manifest flow mediated vasodilation. *Journal of Vascular Surgery*, 50(5): 1063–1070, nov 2009. ISSN 07415214.

[38] William M. Abbott, Allan Callow, Wesley Moore, Robert Rutherford, Frank Veith, and Steven Weinberg. Evaluation and performance standards for arterial prostheses. *Journal of Vascular Surgery*, 17(4):746–756, apr 1993. ISSN 07415214.

[39] R. S. Bennion, R. A. Williams, B. E. Stabile, M. A. Fox, M. L. Owens, and S. E. Wilson. Patency of autogenous saphenous vein versus polytetrafluoroethylene grafts in femoropopliteal bypass for advanced ischemia of the extremity. *Surgery Gynecology and Obstetrics*, 1985. ISSN 00396087.

[40] Joanne E. McBane, Soroor Sharifpoor, Rosalind S. Labow, Marc Ruel, Erik J. Suuronen, and J. Paul Santerre. Tissue Engineering a Small Diameter Vessel Substitute: Engineering Constructs with Select Biomaterials and Cells. *Current Vascular Pharmacology*, 10(3):347–360, mar 2012. ISSN 15701611.

[41] Michel R. Hoenig, Gordon R. Campbell, Barbara E. Rolfe, and Julie H. Campbell. Tissue-Engineered Blood Vessels. *Arteriosclerosis, Thrombosis, and Vascular Biology*, 25(6):1128–1134, jun 2005. ISSN 1079-5642.

[42] Swathi Ravi and Elliot L. Chaikof. Biomaterials for vascular tissue engineering. *Regenerative Medicine*, 5(1):107–120, jan 2010. ISSN 1746-0751.

[43] Hirotsugu Kurobe, Mark W. Maxfield, Christopher K. Breuer, and Toshiharu Shinoka. Concise Review: Tissue-Engineered Vascular Grafts for Cardiac Surgery: Past, Present, and Future. *STEM CELLS Translational Medicine*, 1(7):566–571, jul 2012. ISSN 21576564.

[44] R Langer and JP Vacanti. Tissue engineering. *Science*, 260(5110):920–926, 1993. ISSN 0036-8075.

[45] Toshiharu Shin'oka, Yasuharu Imai, and Yoshito Ikada. Transplantation of a Tissue-Engineered Pulmonary Artery. *New England Journal of Medicine*, 344(7): 532–533, feb 2001. ISSN 0028-4793.

[46] Nicolas L'Heureux, Stéphanie Pâquet, Raymond Labbé, Lucie Germain, and François A. Auger. A completely biological tissue-engineered human blood vessel. *The FASEB Journal*, 12(1):47–56, jan 1998. ISSN 0892-6638.

[47] Gerhardt Konig, Todd N. McAllister, Nathalie Dusserre, Sergio A. Garrido, Corey Iyican, Alicia Marini, Alex Fiorillo, Hernan Avila, Wojciech Wystrychowski, Krzysztof Zagalski, Marcin Maruszewski, Alyce Linthurst Jones, Lech Cierpka, Luis M. de la Fuente, and Nicolas L'Heureux. Mechanical properties of completely autologous human tissue engineered blood vessels compared to human saphenous vein and mammary artery. *Biomaterials*, 30(8):1542–1550, mar 2009. ISSN 01429612.

[48] Gustavo A. Villalona, Brooks Udelsman, Daniel R. Duncan, Edward McGillicuddy, Rajendra F. Sawh-Martinez, Narutoshi Hibino, Christopher Painter, Tamar Mirensky, Benjamin Erickson, Toshiharu Shinoka, and Christopher K. Breuer. Cell-Seeding Techniques in Vascular Tissue Engineering. *Tissue Engineering Part B: Reviews*, 16(3):341–350, jun 2010. ISSN 1937-3368.

[49] Byung-Soo Kim and David J. Mooney. Development of biocompatible synthetic extracellular matrices for tissue engineering. *Trends in Biotechnology*, 16(5):224–230, dec 1998. ISSN 01677799.

[50] Wen Jie Zhang, Wei Liu, Lei Cui, and Yilin Cao. Tissue engineering of blood vessel. *Journal of Cellular and Molecular Medicine*, 11(5):945–957, sep 2007. ISSN 1582-1838.

[51] Joseph D. Berglund and Zorina S. Galis. Designer blood vessels and therapeutic revascularization. *British Journal of Pharmacology*, 140(4):627–636, oct 2003. ISSN 00071188.

[52] A. Vats, N.S. Tolley, J.M. Polak, and J.E. Gough. Scaffolds and biomaterials for tissue engineering: a review of clinical applications. *Clinical Otolaryngology and Allied Sciences*, 28(3):165–172, jun 2003. ISSN 0307-7772.

[53] Ivan Martin, David Wendt, and Michael Heberer. The role of bioreactors in tissue engineering. *Trends in Biotechnology*, 22(2):80–86, feb 2004. ISSN 01677799.

[54] C. Weinberg and Eugene Bell. A blood vessel model constructed from collagen and cultured vascular cells. *Science*, 231(4736):397–400, jan 1986. ISSN 0036-8075.

[55] Lan Yao, Daniel D. Swartz, Sylvia F. Gugino, James A. Russell, and Stelios T. Andreadis. Fibrin-Based Tissue-Engineered Blood Vessels: Differential Effects of Biomaterial and Culture Parameters on Mechanical Strength and Vascular Reactivity. *Tissue Engineering*, 11(7-8):991–1003, jul 2005. ISSN 1076-3279.

[56] Daniel D. Swartz, James A. Russell, and Stelios T. Andreadis. Engineering of fibrin-based functional and implantable small-diameter blood vessels. *American Journal of Physiology-Heart and Circulatory Physiology*, 288(3):H1451–H1460, mar 2005. ISSN 0363-6135.

[57] Hao-Fan Peng, Jin Yu Liu, Stelios T. Andreadis, and Daniel D. Swartz. Hair Follicle-Derived Smooth Muscle Cells and Small Intestinal Submucosa for Engineering Mechanically Robust and Vasoreactive Vascular Media. *Tissue Engineering Part A*, 17(7-8):981–990, apr 2011. ISSN 1937-3341.

[58] Alok Tiwari, Henryk Salacinski, Alexander M. Seifalian, and George Hamilton. New Prostheses for Use in Bypass Grafts with Special Emphasis on Polyurethanes. *Vascular*, 2002. ISSN 17085381.

[59] Brett C. Isenberg, Chrysanthi Williams, and Robert T. Tranquillo. Small-Diameter Artificial Arteries Engineered In Vitro. *Circulation Research*, 98(1):25–35, jan 2006. ISSN 0009-7330.

[60] K.-H. Yow, J. Ingram, S. A. Korossis, E. Ingham, and S. Homer-Vanniasinkam. Tissue engineering of vascular conduits. *British Journal of Surgery*, 93(6):652–661, jun 2006. ISSN 0007-1323.

[61] Steven P. Higgins, Amy K. Solan, and Laura E. Niklason. Effects of polyglycolic acid on porcine smooth muscle cell growth and differentiation. *Journal of Biomedical Materials Research*, 67A(1):295–302, oct 2003. ISSN 0021-9304.

[62] Dietmar Hutmacher. Design and Fabrication of Scaffolds via Solid Free-Form Fabrication. In *Biodegradable Systems in Tissue Engineering and Regenerative Medicine*. CRC Press, nov 2004.

[63] Jeffrey M. Karp, Paul D. Dalton, and Molly S. Shoichet. Scaffolds for Tissue Engineering. *MRS Bulletin*, 28(4):301–306, apr 2003. ISSN 0883-7694.

[64] Peter X. Ma and Jennifer Elisseeff. Scaffolding In Tissue Engineering. In Peter X. Ma and Jennifer Elisseeff, editors, *Scaffolding in Tissue Engineering*. CRC Press, aug 2005. ISBN 9780429121272.

[65] Robert C. Thomson, Albert K. Shung, Michael J. Yaszemski, and Antonios G. Mikos. Polymer Scaffold Processing. In *Principles of Tissue Engineering*, pages 251–262. Elsevier, 2000.

[66] Gustavo Abel Abraham, Pablo Christian Caracciolo, Fabian Alejandro Buffa, and Teresita Raquel Cuadrado. Diseno y preparacion de matrices polimericas porosas para ingenieria de tejidos biologicos. *Anales de la Academia Nacional de Ciencias Exactas, Fisicas y Naturales de Buenos Aires*, 59:1–15, 2007. ISSN 0365-1185.

[67] Peter M. Crapo and Yadong Wang. Physiologic compliance in engineered small-diameter arterial constructs based on an elastomeric substrate. *Biomaterials*, 31(7): 1626–1635, mar 2010. ISSN 01429612.

[68] Jin Gao, Peter Crapo, Robert Nerem, and Yadong Wang. Co-expression of elastin and collagen leads to highly compliant engineered blood vessels. *Journal of Biomedical Materials Research Part A*, 85A(4):1120–1128, jun 2008. ISSN 15493296.

[69] A Nieponice, L Soletti, J Guan, B Deasy, J Huard, W WAGNER, and D VORP. Development of a tissue-engineered vascular graft combining a biodegradable scaffold, muscle-derived stem cells and a rotational vacuum seeding technique. *Biomaterials*, 29(7):825–833, mar 2008. ISSN 01429612.

[70] Chiara E. Ghezzi, Benedetto Marelli, Naser Muja, and Showan N. Nazhat. Immediate production of a tubular dense collagen construct with bioinspired mechanical properties. *Acta Biomaterialia*, 2012. ISSN 17427061.

[71] Shanshan Liu, Chaofei Dong, Guozhong Lu, Qiang Lu, Zhanxiong Li, David L. Kaplan, and Hesun Zhu. Bilayered vascular grafts based on silk proteins. *Acta Biomaterialia*, 2013. ISSN 17427061.

[72] A. L. Andrady. *Science and technology of polymer nanofibers*. John Wiley and Sons Inc., 2008.

[73] Seeram Ramakrishna, Kazutoshi Fujihara, Wee-Eong Teo, Thomas Yong, Zuwei Ma, and Ramakrishna Ramaseshan. Electrospun nanofibers: solving global issues. *Materials Today*, 9(3):40–50, mar 2006. ISSN 13697021.

[74] James B. Carleton, Antonio D'Amore, Kristen R. Feaver, Gregory J. Rodin, and Michael S. Sacks. Geometric characterization and simulation of planar layered elastomeric fibrous biomaterials. *Acta Biomaterialia*, 12(1):93–101, 2015. ISSN 18787568.

[75] C. Williams, J. Liao, E. M. Joyce, B. Wang, J. B. Leach, M. S. Sacks, and J. Y. Wong. Altered structural and mechanical properties in decellularized rabbit carotid arteries. *Acta Biomaterialia*, 5(4):993–1005, 2009. ISSN 17427061.

[76] James Brian Carleton. *Microscale modeling of layered fibrous networks with applications to biomaterials for tissue engineering*. PhD thesis, 2015.

[77] Justyna A. Niestrawska, Christian Viertler, Peter Regitnig, Tina U. Cohnert, Gerhard Sommer, and Gerhard A. Holzapfel. Microstructure and mechanics of healthy and aneurysmatic abdominal aortas: Experimental analysis and modelling. *Journal of the Royal Society Interface*, 13(124), 2016. ISSN 17425662.

[78] Nicholas J. Amoroso, Antonio D'Amore, Yi Hong, William R. Wagner, and Michael S. Sacks. Elastomeric electrospun polyurethane scaffolds: The interrelationship between fabrication conditions, fiber topology, and mechanical properties. *Advanced Materials*, 23(1):106–111, 2011. ISSN 09359648.

[79] Todd Courtney, Michael S. Sacks, John Stankus, Jianjun Guan, and William R. Wagner. Design and analysis of tissue engineering scaffolds that mimic soft tissue mechanical anisotropy. *Biomaterials*, 27(19):3631–3638, 2006. ISSN 01429612.

[80] E. Weinan. *Principles of Multiscale Modeling*. Cambridge University Press, first edition, 2011. ISBN 9781107096547.

[81] Hwa Soon Choi and R. P. Vito. Two-Dimensional Stress-Strain Relationship for Canine Pericardium. *Journal of Biomechanical Engineering*, 112(2):153–159, may 1990. ISSN 0148-0731.

[82] Pin Tong and Yuang-Cheng Fung. The stress-strain relationship for the skin. *Journal of Biomechanics*, 9(10):649–657, jan 1976. ISSN 00219290.

[83] John C. Criscione, Michael S. Sacks, and William C. Hunter. Experimentally Tractable, Pseudo-elastic Constitutive Law for Biomembranes: I. Theory. *Journal of Biomechanical Engineering*, 125(1):94–99, feb 2003. ISSN 0148-0731.

[84] Rong Fan and Michael S. Sacks. Simulation of planar soft tissues using a structural constitutive model: Finite element implementation and validation. *Journal of Biomechanics*, 47(9):2043–2054, jun 2014. ISSN 00219290.

[85] Gerhard A Holzapfel and Ray W Ogden. Constitutive modelling of passive myocardium: a structurally based framework for material characterization. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1902):3445–3475, 2009.

[86] Y. Lanir. A structural theory for the homogeneous biaxial stress-strain relationships in flat collagenous tissues. *Journal of Biomechanics*, 12(6):423–436, jan 1979. ISSN 00219290.

[87] Michael S Sacks. Incorporation of experimentally-derived fiber orientation into a structural constitutive model for planar collagenous tissues. *J. Biomech. Eng.*, 125 (2):280–287, 2003.

[88] C. M. van Wyk. 20—NOTE ON THE COMPRESSIBILITY OF WOOL. *Journal of the Textile Institute Transactions*, 37(12):T285–T292, dec 1946. ISSN 1944-7027.

[89] H. L. Cox. The elasticity and strength of paper and other fibrous materials. *British Journal of Applied Physics*, 3(3):72–79, 1952. ISSN 05083443.

[90] Meltem A. Narter, Subhash K. Batra, and David R. Buchanan. Micromechanics of three-dimensional fibrewebs: Constitutive equations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 455(1989):3543–3563, 1999. ISSN 13645021.

[91] Garth A. Carnaby and N. Pan. Theory of the Compression Hysteresis of Fibrous Assemblies. *Textile Research Journal*, 59(5):275–284, 1989. ISSN 00405175.

[92] N Pan and Garth A Carnaby. Theory of the Shear Deformation of Fibrous Assemblies. *Textile Research Journal*, 59(5):285–292, may 1989. ISSN 0040-5175.

[93] John A Stella, Jun Liao, Yi Hong, W David Merryman, William R Wagner, and Michael S Sacks. Tissue-to-cellular deformation coupling in cell-microintegrated elastomeric scaffolds. In *IUTAM Symposium on Cellular, Molecular and Tissue Mechanics*, pages 81–89. Springer, 2010.

[94] Meredith N. Silberstein, Chia Ling Pai, Gregory C. Rutledge, and Mary C. Boyce. Elasticplastic behavior of non-woven fibrous mats. *Journal of the Mechanics and Physics of Solids*, 60(2):295–318, 2012. ISSN 00225096.

[95] Manuel Zündel, Edoardo Mazza, and Alexander E. Ehret. A 2.5D approach to the mechanics of electrospun fibre mats. *Soft Matter*, 13(37):6407–6421, 2017. ISSN 17446848.

[96] Triantafyllos Stylianopoulos, Chris A. Bashur, Aaron S. Goldstein, Scott A. Guelcher, and Victor H. Barocas. Computational predictions of the tensile properties of electrospun fibre meshes: Effect of fibre diameter and fibre orientation. *Journal of the Mechanical Behavior of Biomedical Materials*, 1(4):326–335, 2008. ISSN 17516161.

[97] Chia Ling Pai, Mary C. Boyce, and Gregory C. Rutledge. On the importance of fiber curvature to the elastic moduli of electrospun nonwoven fiber meshes. *Polymer*, 52(26):6126–6133, 2011. ISSN 00323861.

[98] Xiaofan Wei, Zhenhai Xia, Shing Chung Wong, and Avinash Baji. Modelling of mechanical properties of electrospun nanofibre network. *International Journal of Experimental and Computational Biomechanics*, 1(1):45, 2009. ISSN 1755-8735.

[99] M. S. Rizvi, P. Kumar, D. S. Katti, and A. Pal. Mathematical model of mechanical behavior of micro/nanofibrous materials designed for extracellular matrix substitutes. *Acta Biomaterialia*, 8(11):4111–4122, 2012. ISSN 18787568.

[100] Mohd Suhail Rizvi and Anupam Pal. Statistical model for the mechanical behavior of the tissue engineering non-woven fibrous matrices under large deformation. *Journal of the Mechanical Behavior of Biomedical Materials*, 37(June 2013):235–250, 2014. ISSN 18780180.

[101] James B. Carleton, Gregory J. Rodin, and Michael S. Sacks. Layered Elastomeric Fibrous Scaffolds: An In-Silico Study of the Achievable Range of Mechanical

Behaviors. *ACS Biomaterials Science and Engineering*, 3(11):2907–2921, 2017. ISSN 23739878.

[102] Sebastian Domaschke, Manuel Zündel, Edoardo Mazza, and Alexander E. Ehret. A 3D computational model of electrospun networks and its application to inform a reduced modelling approach. *International Journal of Solids and Structures*, 158: 76–89, 2018. ISSN 00207683.

[103] S. Nemat-Nasse and M. Hori. *Micromechanics: Overall Properties of Heterogeneous Materials*. North Holland, second edition, 1998. ISBN 9780444500847.

[104] V. Kouznetsova, M. G.D. Geers, and W. A.M. Brekelmans. Multi-scale constitutive modelling of heterogeneous materials with a gradient-enhanced computational homogenization scheme. *International Journal for Numerical Methods in Engineering*, 54(8):1235–1260, 2002. ISSN 00295981.

[105] R. Hill. Elastic properties of reinforced solids: Some theoretical principles. *Journal of the Mechanics and Physics of Solids*, 1963. ISSN 00225096.

[106] R. Hill. Continuum micro-mechanics of elastoplastic polycrystals. *Journal of the Mechanics and Physics of Solids*, 1965. ISSN 00225096.

[107] R. Hill. A self-consistent mechanics of composite materials. *Journal of the Mechanics and Physics of Solids*, 1965. ISSN 00225096.

[108] R. Hill. On Constitutive Macro-Variables for Heterogeneous Solids at Finite Strain. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 1972. ISSN 1364-5021.

[109] Z. Hashin and S. Shtrikman. A variational approach to the theory of the elastic behaviour of multiphase materials. *Journal of the Mechanics and Physics of Solids*, 1963. ISSN 00225096.

[110] B. Budiansky. On the elastic moduli of some heterogeneous materials. *Journal of the Mechanics and Physics of Solids*, 1965. ISSN 00225096.

[111] Jean Mandel. Plasticité classique et viscoplasticité: course held... udine, september-october 1971, 1972.

[112] V. G. Kouznetsova, M. G.D. Geers, and W. A.M. Brekelmans. Multi-scale second-order computational homogenization of multi-phase materials: A nested finite element solution strategy. *Computer Methods in Applied Mechanics and Engineering*, 2004. ISSN 00457825.

[113] J.C. Michel, H. Moulinec, and P. Suquet. Effective properties of composite materials with periodic microstructure: a computational approach. *Computer Methods in Applied Mechanics and Engineering*, 172(1-4):109–143, apr 1999. ISSN 00457825.

[114] Christian Miehe, Jan Schotte, and Jörg Schröder. Computational micro–macro transitions and overall moduli in the analysis of polycrystals at large strains. *Computational Materials Science*, 16(1-4):372–382, dec 1999. ISSN 09270256.

[115] K. Terada and N. Kikuchi. A class of general algorithms for multi-scale analyses of heterogeneous media. *Computer Methods in Applied Mechanics and Engineering*, 190(40-41):5427–5464, jul 2001. ISSN 00457825.

[116] P.M. Suquet. Local and global aspects in the mathematical theory of plasticity. In A. Sawczuk and G. Bianchi, editors, *Plasticity Today: Modelling, Methods and Applications*, pages 279–310. Elsevier Applied Science Publishers, 1985.

[117] JoséMiranda Guedes and Noboru Kikuchi. Preprocessing and postprocessing for materials based on the homogenization method with adaptive finite element methods. *Computer Methods in Applied Mechanics and Engineering*, 83(2):143–198, oct 1990. ISSN 00457825.

[118] K. Terada and N. Kikuchi. Nonlinear homogenization method for practical applications. In *American Society of Mechanical Engineers, Applied Mechanics Division, AMD*, 1995.

[119] Somnath Ghosh, Kyunghoon Lee, and Suresh Moorthy. Multiple scale analysis of heterogeneous elastic structures using homogenization theory and voronoi cell finite element method. *International Journal of Solids and Structures*, 32(1):27–62, jan 1995. ISSN 00207683.

[120] Somnath Ghosh, Kyunghoon Lee, and Suresh Moorthy. Two scale analysis of heterogeneous elastic-plastic materials with asymptotic homogenization and Voronoi

cell finite element model. *Computer Methods in Applied Mechanics and Engineering*, 132(1-2):63–116, may 1996. ISSN 00457825.

[121] R.j.m. Smit, W.a.m. Brekelmans, and H.e.h. Meijer. Prediction of the large-strain mechanical response of heterogeneous polymer systems: local and global deformation behaviour of a representative volume element of voided polycarbonate. *Journal of the Mechanics and Physics of Solids*, 47(2):201–221, feb 1999. ISSN 00225096.

[122] Frédéric Feyel and Jean-Louis Chaboche. FE2 multiscale approach for modelling the elastoviscoplastic behaviour of long fibre SiC/Ti composite materials. *Computer Methods in Applied Mechanics and Engineering*, 183(3-4):309–330, mar 2000. ISSN 00457825.

[123] V. Kouznetsova, W. A. M. Brekelmans, and F. P. T. Baaijens. An approach to micro-macro modeling of heterogeneous materials. *Computational Mechanics*, 27 (1):37–48, jan 2001. ISSN 0178-7675.

[124] P. Germain. The Method of Virtual Power in Continuum Mechanics. Part 2: Microstructure. *SIAM Journal on Applied Mathematics*, 25(3):556–575, nov 1973. ISSN 0036-1399.

[125] Y. C. Fung. *Biomechanics: Mechanical Properties of Living Tissues*. Springer-Verlag, second edition, 1998. ISBN 978-1-4757-2257-4.

[126] Pablo J Blanco, Pablo J Sánchez, Eduardo A de Souza Neto, and Raúl A Feijóo. Variational foundations and generalized unified theory of rve-based multiscale models. *Archives of Computational Methods in Engineering*, 23(2):191–253, 2016.

[127] EA de Souza Neto and RA Feijóo. On the equivalence between spatial and material volume averaging of stress in large strain multi-scale solid constitutive models. *Mechanics of materials*, 40(10):803–811, 2008.

[128] Eduardo A de Souza Neto, Raúl A Feijóo, and AA Novotny. Variational foundations of large strain multiscale solid constitutive models: kinematical formulation. *Advanced computational materials modeling: from classical to multi-scale techniques-scale techniques*, 2011.

[129] M Arslan and M C Boyce. Constitutive modeling of the finite deformation behavior of membranes possessing a triangulated network microstructure. *Journal of Applied Mechanics-Transactions of the Asme*, 73(4):536–543, 2006. ISSN 00218936.

[130] K. L. Billiar and M. S. Sacks. Biaxial Mechanical Properties of the Natural and Glutaraldehyde Treated Aortic Valve Cusp—Part I: Experimental Results. *Journal of Biomechanical Engineering*, 122(1):23–30, feb 2000. ISSN 0148-0731.

[131] M.H. Schwartz, P.H. Leo, and J.L. Lewis. A microstructural model for the elastic response of articular cartilage. *Journal of Biomechanics*, 27(7):865–873, 1994. ISSN 00219290.

[132] D. Suarez-Bagnasco, F. Montini-Ballarin, L.J. Cymberknop, G.Balay, C. Negreira, G.A. Abraham, and R.L.Armentano. Elasticity assessment of electrospun nanofibrous vascular grafts: A comparison with femoral ovine arteries. *Materials Science and Engineering C*, 45:1–12, 2014.

[133] F. Montini Ballarin, P. C. Caracciolo, E. Blotta, V. L. Ballarin, and G. A. Abraham. Optimization of poly(l-lactic acid)/segmented polyurethane electrospinning process for the production of bilayered small-diameter nanofibrous tubular structures. *Materials Science and Engineering C*, 42:489–499, 2014. ISSN 09284931.

[134] G Balay, J Brum, D Bia, R L Armentano, and C A Negreira. Improvement of artery radii determination with single ultra sound channel hardware amp;amp; in vitro artificial heart system. In *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*, pages 2521–2524, aug 2010.

[135] G.A. Wempner and D. Talaslidis. *Mechanics of Solids and Shells: Theories and Approximations*. CRC Press, first edition, 1995.

[136] Haitao Niu, Hongxia Wang, Hua Zhou, and Tong Lin. Ultrafine PDMS fibers: Preparation from in situ curing-electrospinning and mechanical characterization. *RSC Advances*, 4(23):11782–11787, 2014. ISSN 20462069.

[137] Jed Johnson, Anirban Ghosh, and John Lannutti. Microstructure-property relationships in a tissue-engineering scaffold. *Journal of Applied Polymer Science*, 104(5):2919–2927, 2007.

[138] Ryuji Inai, Masaya Kotaki, and Seeram Ramakrishna. Structure and properties of electrospun PLLA single nanofibres. *Nanotechnology*, 16(2):208–213, 2005. ISSN 09574484.

[139] E. P.S. Tan, C. N. Goh, C. H. Sow, and C. T. Lim. Tensile test of a single nanofiber using an atomic force microscope tip. *Applied Physics Letters*, 86(7):1–3, 2005. ISSN 00036951.

[140] E. P.S. Tan, S. Y. Ng, and C. T. Lim. Tensile testing of a single ultrafine polymeric fiber. *Biomaterials*, 26(13):1453–1456, 2005. ISSN 01429612.

[141] Eunice P S Tan and C. T. Lim. Effects of annealing on the structural and mechanical properties of electrospun polymeric nanofibres. *Nanotechnology*, 17(10):2649–2654, 2006. ISSN 09574484.

[142] Fei Chen, Xinwen Peng, Tingting Li, Shuiliang Chen, Xiang Fa Wu, Darrell H. Reneker, and Haoqing Hou. Mechanical characterization of single high-strength electrospun polyimide nanofibres. *Journal of Physics D: Applied Physics*, 41(2), 2008. ISSN 00223727.

[143] Shing-Chung Chung Wong, Avinash Baji, and Siwei Leng. Effect of fiber diameter on tensile properties of electrospun poly(ϵ-caprolactone). *Polymer*, 49(21):4713–4722, oct 2008. ISSN 00323861.

[144] John A. Stella, Antonio D'Amore, William R. Wagner, and Michael S. Sacks. On the biomechanical function of scaffolds for engineering load-bearing soft tissues. *Acta Biomaterialia*, 6(7):2365–2381, 2010. ISSN 17427061.

[145] John A. Stella, William R. Wagner, and Michael S. Sacks. Scale-dependent fiber kinematics of elastomeric electrospun scaffolds for soft tissue engineering. *Journal of Biomedical Materials Research Part A*, 93(3):1032–1042, 2010. ISSN 15493296.

[146] Mohammad F. Hadi and Victor H. Barocas. Microscale fiber network alignment affects macroscale failure behavior in simulated collagen tissue analogs. *Journal of Biomechanical Engineering*, 135(2), feb 2013. ISSN 0148-0731.

[147] Antonio D'Amore, John A. Stella, William R. Wagner, and Michael S. Sacks. Characterization of the complete fiber network topology of planar fibrous tissues and scaffolds. *Biomaterials*, 31(20):5345–5354, 2010. ISSN 01429612.

[148] E. A. Sander and V. H. Barocas. Comparison of 2D fiber network orientation measurement methods. *Journal of Biomedical Materials Research Part A*, 88(2): 322–331, feb 2009. ISSN 15493296.

[149] Antonio D'Amore, Nicholas Amoroso, Riccardo Gottardi, Christopher Hobson, Christopher Carruthers, Simon Watkins, William R. Wagner, and Michael S. Sacks. From single fiber to macro-level mechanics: A structural finite-element model for elastomeric fibrous biomaterials. *Journal of the Mechanical Behavior of Biomedical Materials*, 39:146–161, 2014. ISSN 18780180.

[150] G. Argento, M. Simonet, C. W J Oomens, and F. P T Baaijens. Multi-scale mechanical characterization of scaffolds for heart valve tissue engineering. *Journal of Biomechanics*, 45(16):2893–2898, 2012. ISSN 00219290.

[151] Alvaro Ridruejo, Carlos González, and Javier Llorca. A constitutive model for the in-plane mechanical behavior of nonwoven fabrics. *International Journal of Solids and Structures*, 49(17):2215–2229, 2012. ISSN 00207683.

[152] Susan Nachtrab, Sebastian C. Kapfer, Christoph H. Arns, Mahyar Madadi, Klaus Mecke, and Gerd E. Schröder-Turk. Morphology and Linear-Elastic Moduli of Random Network Solids. *Advanced Materials*, 23(22-23):2633–2637, jun 2011. ISSN 09359648.

[153] Triantafyllos Stylianopoulos and Victor H. Barocas. Volume-averaging theory for the study of the mechanics of collagen networks. *Computer Methods in Applied Mechanics and Engineering*, 196(31-32):2981–2990, 2007. ISSN 00457825.

[154] F F Rocha, P J Blanco, R A Feijóo, and P J Sanchez. Multi-scale modelling of arterial tissue: Linking networks of fibers to continua. *Computer Methods in Applied Mechanics and Engineering (in Submission Process)*, 341:740–787, 2018. ISSN 0997-7538.

[155] Florencia Montini Ballarin. *Estructuras poliméricas nanofibrosas biorreabsorbibles para ingeniería de tejidos vasculares*. PhD thesis, Universidad Nacional de Mar del Plata. Facultad de Ingeniería. Argentina, 2015.

[156] Alexander Rachev, Luc Felden, and David N. Ku. Design and Fabrication of a Mechanically Matched Vascular Graft. *Journal of Biomechanical Engineering*, 133(9), sep 2011. ISSN 0148-0731.

[157] Sang Jin Lee, Jie Liu, Se Heang Oh, Shay Soker, Anthony Atala, and James J Yoo. Development of a composite vascular scaffolding system that withstands physiological vascular conditions. *Biomaterials*, 29(19):2891–2898, 2008.

[158] Hiromichi Sonoda, Keiichi Takamizawa, Yasuhide Nakayama, Hisataka Yasui, and Takehisa Matsuda. Small-diameter compliant arterial graft prosthesis: Design concept of coaxial double tubular graft and its fabrication. *Journal of Biomedical Materials Research*, 55(3):266–276, jun 2001. ISSN 0021-9304.

[159] Preethi L. Chandran and Victor H. Barocas. Deterministic Material-Based Averaging Theory Model of Collagen Gel Micromechanics. *Journal of Biomechanical Engineering*, 129(2):137, 2007. ISSN 01480731.

[160] Satoru Yoneyama and Go Murasawa. Digital Image Correlation, in Experimental Mechanics. *Encyclopedia of Life Support Systems (EOLSS)*, pages 1–10, 2009.

[161] F. Hild and S. Roux. Digital image correlation: From displacement measurement to identification of elastic properties - A review. *Strain*, 42(2):69–80, 2006. ISSN 00392103.

[162] X. F. Yao, L. B. Meng, J. C. Jin, and H. Y. Yeh. Full-field deformation measurement of fiber composite pressure vessel using digital speckle correlation method. *Polymer Testing*, 24(2):245–251, 2005. ISSN 01429418.

[163] Alexandre Morel, Sebastian Domaschke, V Urundolil Kumaran, Dmitry Alexeev, Amin Sadeghpour, Shivaprakash N Ramakrishna, Stephen John Ferguson, René Michel Rossi, Edoardo Mazza, Alexander E Ehret, et al. Correlating diameter, mechanical and structural properties of poly (l-lactide) fibres from needleless electrospinning. *Acta Biomaterialia*, 81:169–183, 2018.

[164] F. Yang, R. Murugan, S. Ramakrishna, X. Wang, Y. X. Ma, and S. Wang. Fabrication of nano-structured porous PLLA scaffold intended for nerve tissue engineering. *Biomaterials*, 25(10):1891–1900, 2004. ISSN 01429612.

[165] Morton E Gurtin. *An introduction to continuum mechanics*. Academic press, 1982.

Apéndices

Apéndice A

Principio de Potencia Virtual

A.1. Introducción

En este apéndice se presenta con mayor detalle la formulación variacional de equilibrio utilizado en el problema macroscópico del modelo multiescala del capítulo 2.

En la formulación clásica de la mecánica se admite *a priori* la existencia de los esfuerzos externos [165], definidos a través de campos vectoriales asociados a una medida. De esta forma, se introducen las fuerzas de volumen y fuerzas de superficie, entre otras. En cambio, en la formulación variacional de la mecánica, se admite *a priori* el concepto de potencia o trabajo virtual, mientras que los esfuerzos externos quedarán definidos a través de la dualidad entre los mismos y el movimiento que se realiza sobre el cuerpo, donde dicha dualidad caracteriza la potencia consumida para realizarlos. Este segundo enfoque resulta natural y deviene de una experiencia física muy común: si alguien quiere conocer el peso de un objeto cualquiera, lo que hace es levantarla ligeramente y evaluar el peso por la potencia (o trabajo) que se necesitó para realizar el movimiento.

De manera similar, bajo esta conceptualización los esfuerzos internos surgen *a posteriori* a partir del concepto de potencia consumida para realizar una deformación. Para ejemplificar, si alguien quiere conocer la tensión a la que se encuentra sometida una correa, debe perturbar con los dedos su configuración actual y, a través de la potencia consumida para realizar esa deformación, puede evaluar el valor de la tensión.

A continuación, entonces, se presenta la cinemática de medios continuos desde el punto de vista de la mecánica variacional, así como la dualidad entre esfuerzos y acciones de movimiento como elementos que generan potencia y, finalmente, el equilibrio a partir del Principio de Potencia Virtual como principio fundamental de la mecánica [124].

A.2. Cinemática

A.2.1. Configuración material y espacial

Un cuerpo $\mathcal{C}$ posee una configuración de referencia Ω_m, llamada *configuración material*, dada por la región del espacio determinada por las posiciones $\mathbf{X}$ de todas las partículas materiales que lo componen. Luego, toda otra configuración Ω del cuerpo en un tiempo $t \in [t_0, t_f]$, llamada *configuración espacial*, viene dada por la aplicación:

$$\varphi : \Omega_m \to \Omega \tag{A.1}$$

$$\mathbf{X} \mapsto \mathbf{x} \tag{A.2}$$

donde $\mathbf{x}$ indica la posición de la partícula que inicialmente se ubicaba en $\mathbf{X}$.

Siendo $\mathbf{x} = \varphi(\mathbf{X}, t)$, es posible introducir el campo de desplazamientos relativo a la configuración material:

$$\mathbf{U}(\mathbf{X}, t) = \mathbf{x}(\mathbf{X}, t) - \mathbf{X} = \varphi_t(\mathbf{X}, t) - \mathbf{X} \tag{A.3}$$

Por otra parte, la aplicación φ debe satisfacer algunas propiedades para ser considerada una *deformación*: no debe ocurrir interpenetración del material (para lo que φ debe ser biunívoca) y no puede ocurrir la supresión de un volumen material:

$$\det \nabla\varphi(\mathbf{X}, t) > 0 \quad \forall t \in [t_0, t_f] \tag{A.4}$$

donde $\nabla\varphi$ indica el gradiente del mapeo φ, también conocido como *tensor gradiente de deformaciones* $\mathbf{F}$.

Asimismo, el campo $\mathbf{U}$ debe satisfacer ciertas restricciones para garantizar que se cumpla lo anterior:

$$\mathbf{F} = \nabla\mathbf{X} + \nabla\mathbf{U} = \mathbf{I} + \nabla\mathbf{U} \tag{A.5}$$

donde $\mathbf{I}$ es el tensor identidad.

Es válido aclarar que, dada la configuración de referencia Ω_m, la configuración espacial $\Omega(t)$ puede obtenerse a partir del campo de desplazamientos $\mathbf{U}(\mathbf{X}, t)$ definido en Ω_m. Por lo tanto es equivalente hablar de la configuración $\Omega(t)$ o del campo asociado $\mathbf{U}(\mathbf{X}, t)$. Además, en base a los requerimientos previamente impuestos a la deformación, se define el espacio vectorial $\mathcal{U}_m$ como el conjunto de todas las *configuraciones posibles* que el cuerpo puede tomar.

También es posible definir el campo de desplazamientos $\mathbf{u}(\mathbf{x})$ relativo a la configuración espacial:

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{x} - \varphi_t^{-1}(\mathbf{x}, t) \tag{A.6}$$

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{U}(\varphi(\mathbf{X}), t), \quad \mathbf{u} \in \mathcal{U}_s, \quad \mathbf{U} \in \mathcal{U}_m \tag{A.7}$$

donde $\mathcal{U}_s$ es la contraparte de $\mathcal{U}_m$ en la configuración espacial.

A.2.2. Deformación de un elemento infinitesimal

Dado un entorno suficientemente pequeño de un punto $\mathbf{X}_0$, las posiciones de las partículas pertenecientes a ese entorno pueden escribirse como:

$$\mathbf{X} = \mathbf{X}_0 + \mathbf{F}(\mathbf{X}_0, t)(\mathbf{X} - \mathbf{X}_0) + o(\mathbf{X} - \mathbf{X}_0) \tag{A.8}$$

donde $o(\mathbf{X} - \mathbf{X}_0)$ indica términos de orden superior.

Luego, la deformación de un segmento material infinitesimal dado por $\mathrm{d}\mathbf{X} = \mathbf{X} - \mathbf{X}_0$ queda definida en este entorno por:

$$\mathrm{d}\mathbf{x} = \mathbf{F}\mathrm{d}\mathbf{X} \tag{A.9}$$

$$\mathrm{d}\mathbf{x} \cdot \mathrm{d}\mathbf{x} = \mathbf{F}^T\mathbf{F}\mathrm{d}\mathbf{X} \cdot \mathrm{d}\mathbf{X} \tag{A.10}$$

Siguiendo, una medida de deformación para el segmento $\mathrm{d}\mathbf{X}$ al pasar a la configura-

ción deformada $\mathrm{d}\mathbf{x}$ está dado por:

$$\mathrm{d}\mathbf{x}\cdot\mathrm{d}\mathbf{x}-\mathrm{d}\mathbf{X}\cdot\mathrm{d}\mathbf{X}=\left(\mathbf{F}^T\mathbf{F}-\mathbf{I}\right)\mathrm{d}\mathbf{X}\cdot\mathrm{d}\mathbf{X}=2\mathbf{E}\mathrm{d}\mathbf{X}\cdot\mathrm{d}\mathbf{X} \tag{A.11}$$

donde

$$\mathbf{E}=\frac{1}{2}\left(\mathbf{F}^T\mathbf{F}-\mathbf{I}\right)=\frac{1}{2}\left(\nabla\mathbf{U}+\nabla^T\mathbf{U}+\nabla^T\mathbf{U}\nabla\mathbf{U}\right) \tag{A.12}$$

es conocido como *tensor de deformación de Green* y el sobreíndice T indica el tensor transpuesto ($\nabla^T\mathbf{U}=(\nabla\mathbf{U})^T$).

Hasta aquí se ha dado la descripción *Lagrangiana* de la deformación, donde se sigue a la partícula material $\mathbf{X}$ durante la deformación. Además, si bien es posible establecer otras medidas de deformación, para el objetivo de este apéndice resulta suficiente contar con la medida dada por el tensor $\mathbf{E}$, que a su vez depende del tensor $\mathbf{F}$.

A.2.3. Movimiento y tasa de deformación

El *movimiento* del cuerpo $\mathcal{C}$ está dado por la familia uniparamétrica de configuraciones posibles $\Omega(t)$, $t\in[t_0,t_f]$. A este movimiento, le corresponde en cada instante t una deformación dada por $\varphi(\mathbf{X},t)$ y un campo de velocidad $\mathbf{v}(\mathbf{x},t)$ que se denomina *acción de movimiento* [1]:

$$\mathbf{v}(\mathbf{x},t)=\left.\frac{\partial\mathbf{U}(\mathbf{X},t)}{\partial t}\right|_{\mathbf{X}=\varphi^{-1}(\mathbf{x},t)} \tag{A.13}$$

Luego, se define el espacio vectorial $\mathcal{V}$ como el conjunto de todas las acciones de movimiento posibles a partir de $\mathbf{U}(\mathbf{X},t)$. Notar que, bajo esta definición, el campo de velocidad real del cuerpo es en el instante t, un elemento de $\mathcal{V}$.

Es usual que en un problema de equilibrio mecánico, el cuerpo deba satisfacer ciertas *restricciones cinemáticas*. Las configuraciones posibles del cuerpo que además cumplen con estas restricciones se conocen como *configuraciones admisibles*, y el conjunto de

[1] Es válido notar que se trata de un campo en función de las posiciones espaciales $\mathbf{x}$, por lo que se dice que es la *descripción espacial* de la velocidad

estas define al subespacio vectorial:

$$\mathrm{Kin}_{\mathbf{u}} = \{\mathbf{u} \in \mathcal{U}_s \ , \ \mathbf{u} \text{ es una configuración cinemáticamente admisible}\} \tag{A.14}$$

En función de esta última definición, se dice que todo movimiento a partir de $\mathbf{u}(\mathbf{X}, t) \in \mathrm{Kin}_{\mathbf{u}}$ se considera *movimiento admisible* si todas las configuraciones que lo componen son configuraciones admisibles. Además, a cada movimiento admisible a partir de $\mathbf{u}(\mathbf{X}, t)$ le corresponde una *acción de movimiento admisible* y el conjunto de todas ellas constituye el subconjunto:

$$\mathrm{Kin}_{\mathbf{v}} = \{\mathbf{v} \in \mathcal{V} \ , \ \mathbf{v} \text{ es una acción de movimiento cinemáticamente admisible}\} \tag{A.15}$$

También es posible definir el subconjunto de acciones de movimiento *variacionalmente admisible* $\mathrm{Var}_{\mathbf{v}}$ dado por el conjunto de los campos de velocidad tales que $\mathbf{v} = \mathbf{0}$ en aquellos puntos donde están prescriptas las acciones de movimiento. Se verifica, entonces, que $\mathrm{Kin}_{\mathbf{v}} = \bar{v} + \mathrm{Var}_{\mathbf{v}}$, donde $\bar{v} \in \mathrm{Kin}_{\mathbf{v}}$ es una acción de movimiento arbitraria compatible con las restricciones cinemáticas.

Finalmente, se define un operador tasa de deformación ($\mathscr{D}$) que aplicado sobre el campo de velocidades ($\mathbf{v}$) otorga el tensor tasa de deformación ($\mathbf{D}$) definido como:

$$\mathscr{D}(\mathbf{v}) = \mathbf{D} = \mathrm{grad}^S \mathbf{v} \tag{A.16}$$

donde $\mathrm{grad}^S \mathbf{v}$ indica el gradiente simétrico del campo $\mathbf{v}$.

Esta definición introduce, además, el espacio vectorial $\mathcal{W}$, cuyos elementos son todos los campos tensoriales simétricos definidos en la configuración $\mathbf{u}(\mathbf{X}, t)$. Es posible notar que no todo $\mathbf{D} \in \mathcal{W}$ está asociado a un campo $\mathbf{v} \in \mathcal{V}$, pues no todo tensor simétrico proviene de un gradiente simétrico sobre $\mathbf{v}$. En cambio, se indica que, si dado $\mathbf{D} \in \mathcal{W}$, se puede hallar un campo $\mathbf{v} \in \mathcal{V}$ asociado de forma que se verifique la expresión anterior, entonces $\mathbf{D}$ es una tasa de deformación *compatible* cinemáticamente admisible.

Finalmente, se identifica el conjunto de todas las acciones de movimiento posibles *rígidas*, siendo aquellos $\mathbf{v}$ con tasas de deformación nula. Este conjunto se denomina

espacio nulo del operador $\mathscr{D}$ y está dado por:

$$\mathcal{N}(\mathscr{D}) = \{\mathbf{v} \in \mathcal{V},\ \mathscr{D}(\mathbf{v}) = \mathbf{0} \quad \forall \mathbf{x} \in \Omega\} \tag{A.17}$$

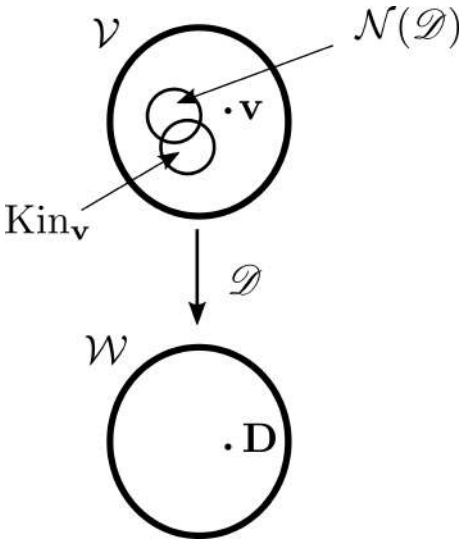


Figura A.1: Espacios y subespacios vectoriales introducidos por la cinemática adoptada.

A.3. Dualidad

A.3.1. Entre fuerzas y acciones de movimiento

La forma "clásica" de caracterizar la acción que un cuerpo realiza sobre otro es a través del concepto *fuerza* como un ente conocido *a priori* y de manera independiente de la cinemática adoptada. En el PPV, en cambio, la fuerza surge sólo como un elemento en *dualidad* a una determinada acción de movimiento, donde la dualidad se define a través del concepto, ahora sí *a priori*, de potencia o trabajo virtual.

Entonces, se define a la *potencia externa* (P_e) como el valor de un funcional lineal y continuo en $\mathcal{V}$ ($\mathcal{P}_e(\mathbf{v})$). Luego, en base al Teorema de la Representación de Riesz, es posible identificar un elemento $\mathbf{f}$, de la misma naturaleza que $\mathbf{v}$[2].

$$P_e = \langle \mathbf{f}, \mathbf{v} \rangle \tag{A.18}$$

[2]En este caso, como $\mathbf{v}$ es un campo vectorial, $\mathbf{f}$ también lo es.

El conjunto de todos los sistemas de fuerzas $\mathbf{f}$ (es decir, el conjunto de todos los funcionales lineales y continuos en $\mathcal{V}$) define al espacio vectorial $\mathcal{V}'$, denominado *espacio de fuerzas externas*. Este espacio es *dual* al espacio de acciones de movimiento $\mathcal{V}$: una vez definido el modelo cinemático, el sistema de fuerzas compatible queda totalmente definido por la dualidad a través de la siguiente forma bilineal:

$$\langle \cdot, \cdot \rangle : \mathcal{V}' \times \mathcal{V} \to \mathbb{R} \tag{A.19}$$

$$(\mathbf{f}, \mathbf{v}) \mapsto P_e = \langle \mathbf{f}, \mathbf{v} \rangle \quad \mathbf{f} \in \mathcal{V}', \quad \mathbf{v} \in \mathcal{V} \tag{A.20}$$

A.3.2. Entre esfuerzos internos y tasas de deformación

En la mecánica del continuo *clásica*, el concepto de tensión se obtiene como una extensión del concepto de fuerza. En el PPV, similarmente a lo expuesto en la sección anterior, los esfuerzos internos se obtienen a partir de un funcional lineal y continuo sobre las tasas de deformación ($\mathcal{P}_i(\mathbf{D})$). Este funcional resulta en un campo escalar p_i definido en Ω. Luego, *potencia interna total* viene dada por:

$$P_i = \int_\Omega p_i \mathrm{d}V = -\int_\Omega \mathbf{T} \cdot \mathbf{D} \mathrm{d}V \tag{A.21}$$

En la expresión anterior, el esfuerzo interno $\mathbf{T}$ es el elemento que caracteriza el funcional lineal y continuo sobre las acciones de deformación ($\mathbf{D} \in \mathcal{W}$) , es decir, $\mathbf{T}$ es el elemento dual de $\mathbf{D}$. Por lo tanto, podemos definir el espacio $\mathcal{W}'$ como aquel constituido por todos los funcionales lineales y continuos sobre $\mathbf{D}$, siendo los mismos representados por campos tensoriales simétricos $\mathbf{T}$[3].

A.3.3. Operador de equilibrio

Como se vio, los espacios $\mathcal{V}'$ y $\mathcal{W}'$ son los espacios duales de $\mathcal{V}$ y $\mathcal{W}$, mientras que las formas bilineales $\langle \cdot, \cdot \rangle$ y $(\cdot, \cdot)$ representan los pares en dualidad de $\mathcal{V}' \times \mathcal{V}$ y $\mathcal{W}' \times \mathcal{W}$, respectivamente. También se definió el operador tasa de deformación $\mathscr{D} : \mathcal{V} \to \mathcal{W}$.

Ahora, es posible introducir el *operador adjunto* u *operador de equilibrio* $\mathscr{D}^* : \mathcal{W}' \to$

[3]Resulta evidente a esta altura que $\mathbf{T}$ se corresponde con el tensor de tensiones de Cauchy.

$\mathcal{V}'$, que se define de la siguiente manera:

$$(\mathbf{T}, \mathscr{D}(\mathbf{v})) = \langle \mathscr{D}^*(\mathbf{T}), \mathbf{v} \rangle \tag{A.22}$$

Este operador permite, conjuntamente con el Principio de Potencias Virtuales, caracterizar los esfuerzos internos *compatibles* con el modelo cinemático elaborado.

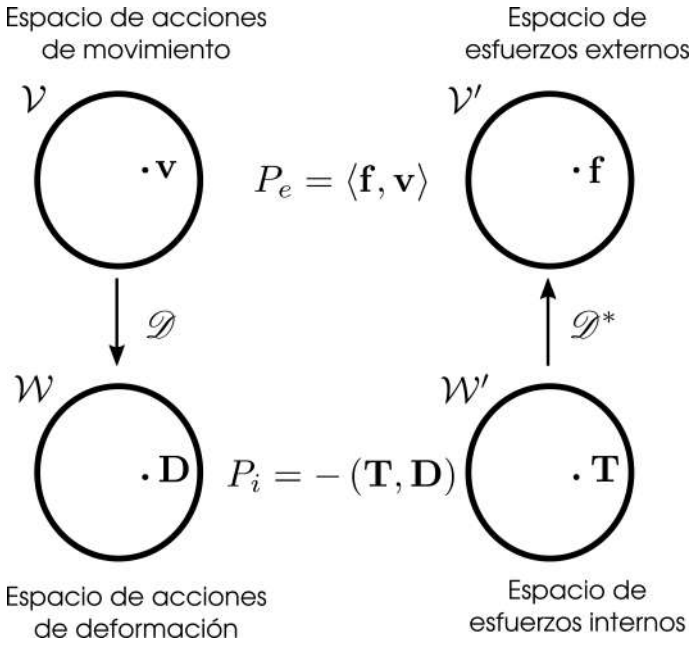


Figura A.2: Espacios y subespacios vectoriales del PPV.

A.4. Principio de Potencia Virtual

Para todo sistema de referencia inercial, y para cada instante de tiempo t, el cuerpo $\mathcal{C}$ se encuentra en equilibrio (estático[4]) en la configuración Ω, con restricciones cinemáticas bilaterales y bajo la acción del sistema de fuerzas externas $\mathbf{f} \in \mathcal{V}$ si la *potencia virtual externa* de dichas fuerzas es nula para toda acción de movimiento virtual rígida a partir de esta configuración:

$$P_e = \langle \mathbf{f}, \hat{\mathbf{v}} \rangle = 0 \quad \forall \hat{\mathbf{v}} \in \mathrm{Var}_{\mathbf{v}} \cap \mathcal{N}(\mathscr{D}) \tag{A.23}$$

[4]La formulación puede extenderse fácilmente al caso de equilibrio dinámico si se incluye en el análisis a las fuerzas de inercia como fuerzas externas adicionales.

Además, se satisface que la suma de las potencias externa e interna es nula para toda acción de movimiento virtual:

$$P_e + P_i = \langle \mathbf{f}, \hat{\mathbf{v}} \rangle - (\mathbf{T}, \mathscr{D}(\hat{\mathbf{v}})) = 0 \quad \forall \hat{\mathbf{v}} \in \mathrm{Var}_{\mathbf{v}} \tag{A.24}$$

La primera parte (A.23) permite caracterizar el sistema de fuerzas compatible con la cinemática adoptada:

$$\langle \mathbf{f}, \hat{\mathbf{v}} \rangle = \langle \mathscr{D}^*(\mathbf{T}), \hat{\mathbf{v}} \rangle = (\mathbf{T}, \mathscr{D}(\hat{\mathbf{v}})) \quad \forall \hat{\mathbf{v}} \in \mathcal{V} \tag{A.25}$$

Específicamente para el caso de la cinemática planteada de cuerpos continuos, se tiene que:

$$\begin{aligned}
\langle \mathbf{f}, \hat{\mathbf{v}} \rangle = (\mathbf{T}, \mathscr{D}(\hat{\mathbf{v}})) &= \int_\Omega \mathbf{T} \cdot \mathrm{grad}^S \hat{\mathbf{v}} \mathrm{d}V && \text{(A.26)} \\
&= \int_\Omega \mathbf{T} \cdot \mathrm{grad} \hat{\mathbf{v}} \mathrm{d}V && \text{(A.27)} \\
&= \int_\Omega \left[\mathrm{div}\left(\mathbf{T}\hat{\mathbf{v}}\right) - \mathrm{div}\mathbf{T} \cdot \hat{\mathbf{v}} \right] \mathrm{d}V && \text{(A.28)} \\
&= \int_{\partial\Omega^N} \mathbf{T}\mathbf{n} \cdot \hat{\mathbf{v}} \mathrm{d}S + \int_{\partial\Omega} \mathrm{div}\mathbf{T} \cdot \hat{\mathbf{v}} \mathrm{d}V && \text{(A.29)} \\
&= \int_{\partial\Omega^N} \mathbf{a} \cdot \hat{\mathbf{v}} \mathrm{d}S + \int_{\partial\Omega} \mathbf{b} \cdot \hat{\mathbf{v}} \mathrm{d}V && \text{(A.30)} \\
&= \langle \mathscr{D}^*(\mathbf{T}), \hat{\mathbf{v}} \rangle && \text{(A.31)}
\end{aligned}$$

donde $\partial\Omega^N$ es la frontera de Neumann de Ω, donde no se encuentran prescriptas las acciones de movimiento.

Es decir, se ha encontrado la forma del operador de equilibrio para este caso:

$$\mathscr{D}^*(\cdot) = \begin{cases} -\mathrm{div}(\cdot) & en\ \Omega \\ (\cdot)\mathbf{n} & en \partial\Omega^N \end{cases} \tag{A.32}$$

Además, se ha encontrado que el sistema de fuerzas externas que es compatible con

acciones de movimiento $\hat{\mathbf{v}} \in \mathrm{Var}_{\mathbf{v}}$, está caracterizado por una fuerza por unidad de volumen ($\mathbf{b}$) en Ω y por una fuerza por unidad de superficie ($\mathbf{a}$) en $\partial\Omega^N$ (figura A.3).

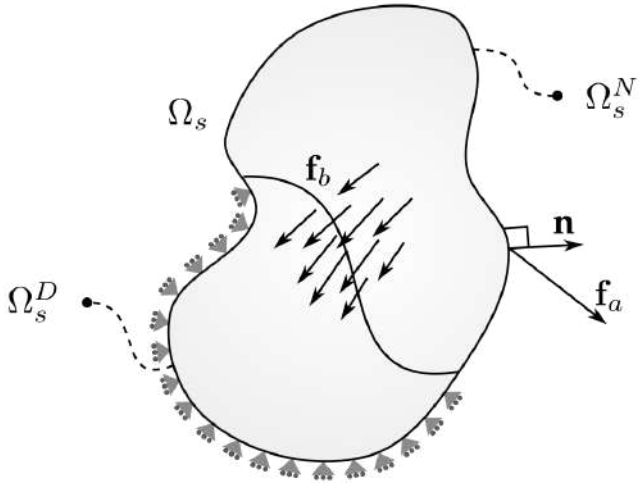


Figura A.3: Esquema del cuerpo en la configuración Ω, con restricciones cinemáticas bilaterales en la frontera de Dirichlet ($\partial\Omega^D$) y bajo dos tipos de esfuerzos externos: una fuerza de volumen ($\mathbf{b}$) en la región Ω y una fuerza de superficie ($\mathbf{a}$) en la frontera de Neumann ($\partial\Omega^N$).

Finalmente, el principio de potencia virtual, para el caso particular bajo consideración, adopta la siguiente forma:

$$\int_\Omega \mathbf{T} \cdot \mathrm{grad}^S \hat{\mathbf{v}} \mathrm{d}V - \int_\Omega \mathbf{b} \cdot \hat{\mathbf{v}} \mathrm{d}V - \int_{\partial\Omega^N} \mathbf{a} \cdot \hat{\mathbf{v}} \mathrm{d}S = 0 \quad \forall \hat{\mathbf{v}} \in \mathrm{Var}_{\mathbf{v}} \tag{A.33}$$

A.5. Linealización

A.5.1. Expresión del PPV en la configuración material

Hasta aquí se presentó el PPV en la configuración en que se encuentra realmente definido, es decir, en la configuración actual o espacial. Sin embargo, como es usual en la mecánica, se desea resolver el campo de desplazamientos a partir de conocer la configuración material.

Para ello, se hace uso de las siguientes relaciones:

$$(\mathrm{d}V)_m = \det\mathbf{F}\mathrm{d}V_m \tag{A.34}$$

$$(\mathrm{d}S)_m = \det\mathbf{F}\left|\left|\mathbf{F}^{-T}\mathbf{N}\right|\right|\mathrm{d}S_m \tag{A.35}$$

$$\mathbf{n}_m = \frac{\mathbf{F}^{-T}\mathbf{N}}{||\mathbf{F}^{-T}\mathbf{N}||} \tag{A.36}$$

$$\mathbf{S} = \det\mathbf{F}\ \mathbf{F}^{-1}\mathbf{T}\mathbf{F}^{-T} \tag{A.37}$$

donde $\mathbf{N}$ es el versor normal a la superficie $\partial\Omega_m^N$, $\mathbf{S}$ el tensor de tensiones de Piola-Kirchhoff de segunda especie y el subíndice m indica que se expresan en términos de las coordenadas materiales ($\mathbf{X}$), variables que naturalmente están dadas en términos de las coordenadas espaciales ($\mathbf{x}$).

Luego, el problema mecánico puede expresarse equivalentemente de la siguiente manera: encontrar $\mathbf{U} \in \mathrm{Kin}_{\mathbf{U}}$ tal que

$$\begin{aligned}\int_{\Omega_m} \mathbf{S}(\mathbf{E}(\mathbf{U})) \cdot \left(\mathbf{F}^T\nabla\hat{\mathbf{V}}\right)^S \mathrm{d}V_m - \int_{\partial\Omega_m^N} \mathbf{b}_m \cdot \hat{\mathbf{V}}\ \det\mathbf{F}\ \mathrm{d}S_m \\ - \int_{\partial\Omega_m^N} \mathbf{a}_m \cdot \hat{\mathbf{V}}\ \det\mathbf{F}\ \left|\left|\mathbf{F}^{-T}\mathbf{N}\right|\right|\ \mathrm{d}S_m = 0 \quad \forall\hat{\mathbf{V}} \in \mathrm{Var}_{\mathbf{V}}\end{aligned} \tag{A.38}$$

donde $\hat{\mathbf{V}} = \hat{\mathbf{v}}_m$ y $\mathrm{Var}_{\mathbf{V}} = (\mathrm{Var}_{\mathbf{v}})_m$.

Por practicidad, en adelante se expresará (A.38) en forma compacta como sigue:

$$\left\langle \mathscr{R}_m\left(\mathbf{U}\right), \hat{\mathbf{V}}\right\rangle_{\Omega_m} = 0 \quad \forall\hat{\mathbf{V}} \in \mathrm{Var}_{\mathbf{V}} \tag{A.39}$$

A.5.2. Linealización

Una vez definida la expresión del PPV en la configuración material (conocida), se plantea el método de Newton-Raphson como la siguiente aproximación: dada una configuración $\mathbf{U}^k \in \mathrm{Kin}_{\mathbf{U}}$, hallar el incremento $\delta\mathbf{U} \in \mathrm{Var}_{\mathbf{V}}$ tal que

$$\begin{aligned}&\left\langle \mathscr{R}_m\left(\mathbf{U}^k\right), \hat{\mathbf{V}}\right\rangle_{\Omega_m} \\ &\quad + \frac{\mathrm{d}}{\mathrm{d}\tau}\left\langle \mathscr{R}_m\left(\mathbf{U}^k + \tau\delta\mathbf{U}\right), \hat{\mathbf{V}}\right\rangle_{\Omega_m}\Bigg|_{\tau=0} = 0 \quad \forall\hat{\mathbf{V}} \in \mathrm{Var}_{\mathbf{V}}\end{aligned} \tag{A.40}$$

donde el operador $\frac{\mathrm{d}}{\mathrm{d}\tau}\left(\cdot\right)\big|_{\tau=0}$ es la conocida derivada de Gateaux.

En consecuencia PPV linealizado en la configuración material queda:

$$\begin{aligned}
&\int_{\Omega_m} \mathbf{S}\cdot\left(\mathbf{F}^T\nabla\hat{\mathbf{V}}\right)^S \,\mathrm{d}V_m - \int_{\partial\Omega_m^N} \mathbf{b}_m\cdot\hat{\mathbf{V}}\,\mathrm{det}\mathbf{F}\,\mathrm{d}S_m \\
&\quad - \int_{\partial\Omega_m^N} \mathbf{a}_m\cdot\hat{\mathbf{V}}\,\mathrm{det}\mathbf{F}\,\left|\left|\mathbf{F}^{-T}\mathbf{N}\right|\right|\,\mathrm{d}S_m \\
&\quad + \int_{\Omega_m}\left(\frac{\partial\mathbf{S}}{\partial\mathbf{E}}\right)\left(\mathbf{F}^T\nabla\delta\mathbf{U}\right)^S\cdot\left(\mathbf{F}^{-T}\nabla\hat{\mathbf{V}}\right)\,\mathrm{d}V_m \\
&\quad - \int_{\Omega_m}\mathbf{b}_m\cdot\hat{\mathbf{V}}\left(\mathbf{F}^{-T}\cdot\nabla\delta\mathbf{U}\right)\,\mathrm{det}\mathbf{F}\,\mathrm{d}V_m \\
&\quad - \int_{\partial\Omega_m^N}\mathbf{a}_m\cdot\hat{\mathbf{V}}\left[\left(\nabla\delta\mathbf{U}\mathbf{F}^{-1}\right)\cdot\mathbf{P}_m^t\right]\,\mathrm{det}\mathbf{F}\,\left|\left|\mathbf{F}^{-T}\mathbf{N}\right|\right|\,\mathrm{d}S_m \\
&\qquad = 0 \quad \forall\hat{\mathbf{V}}\in\mathrm{Var}_{\mathbf{V}}
\end{aligned} \tag{A.41}$$

donde $\mathbf{P}_m^t = \left(\mathbf{I} - \frac{\mathbf{F}^{-T}\mathbf{N}}{||\mathbf{F}^{-T}\mathbf{N}||}\otimes\frac{\mathbf{F}^{-T}\mathbf{N}}{||\mathbf{F}^{-T}\mathbf{N}||}\right)$ es el tensor proyección tangencial en términos de variables materiales.

Es importante tener en cuenta que si bien en la expresión anterior se ha dejado implícita la dependencia de $\mathbf{U}^k$ para facilitar la lectura, se debe tener en cuenta que donde se lee $\mathbf{S}$ y $\mathbf{F}$, se trata de $\mathbf{S}(\mathbf{E}(\mathbf{U}^k))$ y $\mathbf{F}(\mathbf{U}^k)$, respectivamente.

Finalmente, es posible reescribir el PPV linealizado nuevamente en la configuración espacial. Esto presenta la ventaja de mejorar la precisión con la que se evalúan los términos en la frontera en esquemas discretos, por no intervenir el tensor $\mathbf{F}$ en su cálculo:

$$\begin{aligned}
&\int_\Omega\left[\frac{1}{\mathrm{det}\mathbf{F}}\mathbf{F}\mathbf{S}\mathbf{F}\right]_e\cdot\left(\mathrm{grad}\hat{\mathbf{v}}\right)^S\,\mathrm{d}V - \int_\Omega\mathbf{b}\cdot\hat{\mathbf{v}}\,\mathrm{d}V - \int_{\partial\Omega^N}\mathbf{a}\cdot\hat{\mathbf{v}}\,\mathrm{d}S \\
&\quad + \int_\Omega\mathbf{D}_e\left(\mathrm{grad}\delta\mathbf{u}\right)^S\cdot\mathrm{grad}\hat{\mathbf{v}}\,\mathrm{d}V - \int_\Omega\mathbf{b}\cdot\hat{\mathbf{v}}\left(\mathrm{div}\delta\mathbf{u}\right)\,\mathrm{d}V \\
&\quad - \int_{\partial\Omega^N}\mathbf{a}\cdot\hat{\mathbf{v}}\left(\mathrm{grad}\delta\mathbf{u}\cdot\mathbf{P}^t\right)\,\mathrm{d}S = 0 \quad \forall\hat{\mathbf{v}}\in\mathrm{Var}_{\mathbf{v}}
\end{aligned} \tag{A.42}$$

donde $\mathbf{P}^t = (\mathbf{I} - \mathbf{n}\otimes\mathbf{n})$ es el tensor proyección tangencial.

Apéndice B

Código desarrollado

B.1. RVE de celdas triangulares

```

PROGRAM TrianglesRVE
   USE MOD_Programs, ONLY : Haz_Curva, RVE, RVE_Optimizar
   USE MOD_RVE, ONLY : RVE_Curva_Simple, RVE_Curva_Biaxial
   USE MOD_Auxiliar, ONLY : FindStringInFile
   ! = = = = = = = = = =
   IMPLICIT NONE
   ! = = = = = = = = = =
   CHARACTER(LEN=120) :: configFile
   CHARACTER(LEN=120) :: outFile
   ! = = = = = = = = = =
   ! Current working directory
   CHARACTER(LEN=255) :: cwd
   ! = = = = = = = = = =
   INTEGER :: fid
   INTEGER :: iStatus
   CHARACTER(LEN=10) :: instruccion
   ! = = = = = = = = = =


   CALL GETCWD(cwd)

   configFile = '0d.config'
   fid = 11
   OPEN(UNIT=fid, FILE=TRIM(configFile), STATUS="OLD")
   iStatus = FindStringInFile("*CONTROL", fid, .TRUE.)
   READ(fid,*) instruccion
   READ(fid,*) outFile
   CLOSE(UNIT=fid)


   IF (TRIM(instruccion) == "computar") THEN
      CALL RVE(RVE_Curva_Simple, configFile, outFile)
   ELSEIF (TRIM(instruccion) == "optimizar") THEN
      CALL RVE_Optimizar(RVE_Curva_Simple, configFile, outFile)
   ELSEIF (TRIM(instruccion) == "curva_haz") THEN
      CALL Haz_Curva(configFile, outFile)
   ELSE
      WRITE(*,*) "instruccion␣erronea,␣no␣se␣hace␣nada"
   END IF

   STOP

END PROGRAM TrianglesRVE
```

```
! = = = = = = = = = =
MODULE MOD_Programs

```

```
4     ! - - - - - - - - - -
5     IMPLICIT NONE
6     PRIVATE ! NOTE_avoid_public_variables_if_possible
7     ! - - - - - - - - - -
8     PUBLIC :: RVE
9     PUBLIC :: Haz_Curva
10    PUBLIC :: RVE_Optimizar
11    ! - - - - - - - - - -
12
13  ! = = = = = = = = = =
14  CONTAINS
15  ! = = = = = = = = = =
16
17
18    ! = = = = = = = = = =
19    SUBROUTINE Haz_Curva(configurationFile, outputFile)
20       ! - - - - - - - - - -
21       USE MOD_Bundle, ONLY : Get_CurvaConstitutivaHaz
22       USE MOD_Auxiliar, ONLY : FindStringInFile
23       IMPLICIT NONE
24       ! - - - - - - - - - -
25       INTEGER, PARAMETER :: nDim = 2
26       ! - - - - - - - - - -
27       CHARACTER(LEN=*), INTENT(IN) :: configurationFile
28       CHARACTER(LEN=*), INTENT(IN) :: outputFile
29       ! - - - - - - - - - -
30       ! Elastic parameters
31       REAL(8) :: Et
32       REAL(8) :: Eb
33       ! Bundle parameters
34       REAL(8) :: lamr0, lamrS, lamr_min, lamr_max
35       ! Parametros haz - discretizacion curva constitutiva
36       INTEGER :: CC_nPuntos, CC_precision
37       REAL(8) :: CC_lam_min, CC_lam_max
38       ! Curva constitutiva
39       REAL(8), ALLOCATABLE :: CC_lam(:), CC_ten(:)
40       ! RVE parameters
41       INTEGER :: nBundlesRVE
42       REAL(8), ALLOCATABLE :: bundles_v0(:,:)
43       REAL(8) :: angleRVE
44       ! Mat parameters
45       REAL(8) :: volFraction
46       ! - - - - - - - - - -
47       ! Counter
48       INTEGER :: last, i
49       ! Reading status
50       INTEGER :: iStatus
51       ! File ID
52       INTEGER, PARAMETER :: fid1 = 11
53       ! Current working directory
54       CHARACTER(LEN=255) :: cwd
55       ! - - - - - - - - - -
56       ! Arrays for optimization
57       INTEGER :: numParFix
58       REAL(8), ALLOCATABLE :: ParFix(:)
59       INTEGER :: numParFit
60       REAL(8), ALLOCATABLE :: ParFit(:)
61       ! Fitting
62       REAL(8) :: MEA, TSS, SSR, R2
63       ! - - - - - - - - - -
64
65       ! - - - - - - - - -
66       WRITE(*,*) "Configuracion␣en:␣", TRIM(configurationFile)
67       WRITE(*,*) "Salida␣en:␣", TRIM(outputFile)
68       ! - - - - - - - - -
69       ! Leer parametros
70       OPEN(UNIT=fid1, FILE=TRIM(configurationFile), STATUS="OLD")
71       iStatus = FindStringInFile("*FIXPAR", fid1, .TRUE.)
72       READ(fid1,*) numParFix
73       ALLOCATE( ParFix(numParFix) )
```

```
74        READ(fid1,*) ParFix
75        iStatus = FindStringInFile("*FITPAR", fid1, .TRUE.)
76        READ(fid1,*) numParFit
77        ALLOCATE( ParFit(numParFit) )
78        READ(fid1,*) ParFit
79        CLOSE(UNIT=fid1)
80        ! - - - - - - - - - -
81        ! Parametros
82        nBundlesRVE = NINT(ParFix(1))
83        ALLOCATE( bundles_v0(nDim, nBundlesRVE) )
84        bundles_v0 = RESHAPE( ParFix(2:1+nDim*nBundlesRVE), SHAPE(bundles_v0) )
85        last = 1 + nDim*nBundlesRVE
86        !
87        CC_nPuntos = ParFix(last+1)
88        CC_lam_min = ParFix(last+2)
89        CC_lam_max = ParFix(last+3)
90        CC_precision = ParFix(last+4)
91        !
92        Et = ParFit(1)
93        Eb = ParFit(2)
94        lamr0 = ParFit(4)
95        lamrS = ParFit(5)
96        lamr_min = ParFit(6)
97        lamr_max = ParFit(7)
98        ! - - - - - - - - - -
99        ! Calcular curva constitutiva de un haz
100       CALL Get_CurvaConstitutivaHaz &
101       (CC_nPuntos, CC_lam_min, CC_lam_max, &
102       Et, Eb, lamr0, lamrS, lamr_min, lamr_max, &
103       CC_precision, &
104       CC_lam, CC_ten)
105       ! - - - - - - - - - -
106       OPEN(UNIT=fid1,FILE=outputFile,STATUS="replace")
107       WRITE(fid1, *) CC_nPuntos
108       DO i=1,CC_nPuntos
109          WRITE(fid1,'(2E20.8E3)') CC_lam(i), CC_ten(i)
110       END DO
111       CLOSE(fid1)
112       ! - - - - - - - - - -
113
114
115       ! - - - - - - - - - -
116   END SUBROUTINE Haz_Curva
117   ! = = = = = = = = = =
118
119
120   ! = = = = = = = = = =
121   SUBROUTINE RVE(ExtFun_Curva, configurationFile, outputFile)
122      ! - - - - - - - - - -
123      ! Esta subrutina computa y escribe en archivo la curva constitutiva de
124      ! un RVE, el tipo de deformacion aplicado viene dado por la subrutina
125      ! externa ExtFun_Curva. Puede ser uniaxial, biaxial, simple. (Ver ejem-
126      ! plos en el modulo MOD_RVE)
127      ! - - - - - - - - - -
128      USE MOD_Auxiliar, ONLY : FindStringInFile, Compute_SSR
129      IMPLICIT NONE
130      ! - - - - - - - - - -
131      INTEGER, PARAMETER :: nDim = 2
132      ! - - - - - - - - - -
133      EXTERNAL :: ExtFun_Curva
134      INTERFACE
135         SUBROUTINE ExtFun_Curva(nPuntos, lam_array, nPar1, Par1, nPar2, Par2, P)
136            INTEGER, PARAMETER :: nDim = 2
137            INTEGER, INTENT(IN) :: nPuntos
138            REAL(8), INTENT(IN) :: lam_array(nPuntos)
139            INTEGER, INTENT(IN) :: nPar1
140            REAL(8), INTENT(IN) :: Par1(nPar1)
141            INTEGER, INTENT(IN) :: nPar2
142            REAL(8), INTENT(IN) :: Par2(nPar2)
143            REAL(8), INTENT(OUT) :: P(nDim,nDim,nPuntos)
```

```
144         END SUBROUTINE
145       END INTERFACE
146       CHARACTER(LEN=*), INTENT(IN) :: configurationFile
147       CHARACTER(LEN=*), INTENT(IN) :: outputFile
148       ! - - - - - - - - - -
149       ! Counter
150       INTEGER :: last, i
151       ! Reading status
152       INTEGER :: iStatus
153       ! Data arrays
154       INTEGER :: numExpDataPoints, lambdas_n
155       REAL(8) :: lambdas_ini, lambdas_fin, lambdas_delta
156       REAL(8), ALLOCATABLE :: exp_lam(:), exp_ten(:)
157       !
158       REAL(8), ALLOCATABLE :: sim_stress(:,:,:)
159       ! File ID
160       INTEGER, PARAMETER :: fid1 = 11, fid2=12
161       ! Current working directory
162       CHARACTER(LEN=255) :: cwd
163       ! - - - - - - - - - -
164       ! Arrays for optimization
165       INTEGER :: numParFix
166       REAL(8), ALLOCATABLE :: ParFix(:)
167       INTEGER :: numParFit
168       REAL(8), ALLOCATABLE :: ParFit(:)
169       ! Fitting
170       REAL(8) :: MEA, TSS, SSR, Residuo
171       ! - - - - - - - - - -
172
173       ! - - - - - - - - -
174       WRITE(*,*) "Configuracion␣en:␣", TRIM(configurationFile)
175       WRITE(*,*) "Salida␣en:␣", TRIM(outputFile)
176       ! - - - - - - - - -
177       ! LEER PARAMETROS
178       OPEN(UNIT=fid1, FILE=TRIM(configurationFile), STATUS="OLD")
179       ! parametros fijos
180       iStatus = FindStringInFile("*FIXPAR", fid1, .TRUE.)
181       READ(fid1,*) numParFix
182       ALLOCATE( ParFix(numParFix) )
183       READ(fid1,*) ParFix
184       ! parametros a estimar
185       iStatus = FindStringInFile("*FITPAR", fid1, .TRUE.)
186       READ(fid1,*) numParFit
187       ALLOCATE( ParFit(numParFit) )
188       READ(fid1,*) ParFit
189       ! - - - - - - - - -
190       ! LEER DEFORMACIONES Y TENSIONES
191       ! chequear si los lambdas se dan en 0d.config
192       iStatus = FindStringInFile("*LAMBDAS", fid1, .FALSE.)
193       IF (iStatus == 0) THEN
194          ! si se dan ahi se ensambla el array de lambdas a partir de unos pocos parametros
195          ! y a la tension se asigna un array de ceros
196          READ(fid1,*) lambdas_ini, lambdas_fin, lambdas_n
197          ! Ensamblar array de deformaciones para computar tensiones
198          ALLOCATE( exp_lam(lambdas_n) )
199          ALLOCATE( exp_ten(lambdas_n) )
200          exp_ten = 0.0d0
201          exp_lam(1) = lambdas_ini
202          lambdas_delta = (lambdas_fin-lambdas_ini)/DFLOAT(lambdas_n-1)
203          DO i=2,lambdas_n
204             exp_lam(i) = exp_lam(i-1)+lambdas_delta
205          END DO
206       ELSE IF (iStatus < 0) THEN
207          ! si no se dan en 0d.config entonces se lee de una curva experimental
208          OPEN( UNIT=fid2, FILE="curva_experimental.txt", STATUS="old")
209          READ(fid2,*) numExpDataPoints
210          lambdas_n = numExpDataPoints
211          ALLOCATE( exp_lam(lambdas_n) )
212          ALLOCATE( exp_ten(lambdas_n) )
213          DO i=1,lambdas_n
```

```
214             READ(fid2, *) exp_lam(i), exp_ten(i)
215          END DO
216          CLOSE(UNIT=fid2)
217          ! Mean average and total sum of squares
218          MEA = SUM(exp_ten) / lambdas_n
219          TSS = SUM( (exp_lam-MEA)*(exp_lam-MEA) )
220       ELSE
221          WRITE(*,*) "Error␣leyendo␣archivo,␣iStatus=␣", iStatus
222          STOP
223       END IF
224
225       CLOSE(UNIT=fid1)
226       ! - - - - - - - - - -
227
228       ! - - - - - - - - - -
229       ! COMPUTAR CURVA CONSTITUTIVA TENSION-DEFORMACION
230       ALLOCATE( sim_stress(nDim, nDim, lambdas_n), STAT=iStatus)
231       CALL ExtFun_Curva &
232       (lambdas_n, exp_lam, &
233       numParFix, ParFix, &
234       numParFit, ParFit, &
235       sim_stress)
236       ! - - - - - - - - - -
237       ! ESCRIBIR CURVA CONSTITUTIVA
238       OPEN(UNIT=fid1,FILE=outputFile,STATUS="replace")
239       IF (iStatus<0) THEN
240          ! Residuo
241          SSR = Compute_SSR(exp_ten, sim_stress(1,1,:), lambdas_n)
242          Residuo = SSR/TSS
243          WRITE(*,*) "Residuo:␣", Residuo
244       END IF
245       WRITE(fid1,"(6A20)") "exp_lambda", "exp_stress", "P11", "P21", "P12", "P22"
246       DO i=1,lambdas_n
247          WRITE(fid1,'(6E20.8E3)') exp_lam(i), exp_ten(i), sim_stress(:,:,i)
248       END DO
249       CLOSE(fid1)
250       ! - - - - - - - - - -
251
252
253       ! - - - - - - - - - - -
254    END SUBROUTINE RVE
255    ! = = = = = = = = = =
256
257
258    ! = = = = = = = = = =
259    SUBROUTINE RVE_Optimizar(ExtFun_Curva, configurationFile, outputFile)
260       ! - - - - - - - - - -
261       USE MOD_Optimization, ONLY : BruteForceOptimization
262       USE MOD_Auxiliar, ONLY : FindStringInFile
263       IMPLICIT NONE
264       ! - - - - - - - - - -
265       INTEGER, PARAMETER :: nDim = 2
266       ! - - - - - - - - - -
267       EXTERNAL :: ExtFun_Curva
268       INTERFACE
269          SUBROUTINE ExtFun_Curva(nPuntos, lam_array, nPar1, Par1, nPar2, Par2, P)
270             INTEGER, PARAMETER :: nDim = 2
271             INTEGER, INTENT(IN) :: nPuntos
272             REAL(8), INTENT(IN) :: lam_array(nPuntos)
273             INTEGER, INTENT(IN) :: nPar1
274             REAL(8), INTENT(IN) :: Par1(nPar1)
275             INTEGER, INTENT(IN) :: nPar2
276             REAL(8), INTENT(IN) :: Par2(nPar2)
277             REAL(8), INTENT(OUT) :: P(nDim,nDim,nPuntos)
278          END SUBROUTINE
279       END INTERFACE
280       CHARACTER(LEN=*), INTENT(IN) :: configurationFile
281       CHARACTER(LEN=*), INTENT(IN) :: outputFile
282       ! - - - - - - - - - -
283       ! Counter
```

```
284       INTEGER :: i
285       ! Reading Status
286       INTEGER :: iStatus
287       ! Data arrays
288       INTEGER :: numExpDataPoints
289       REAL(8), ALLOCATABLE :: exp_lambdas(:)
290       REAL(8), ALLOCATABLE :: exp_stress(:)
291       !real(8) :: sim_stress(2,2,60)
292       REAL(8), ALLOCATABLE :: sim_P(:,:,:)
293       REAL(8), ALLOCATABLE :: sim_P_old(:,:,:)
294       REAL(8), ALLOCATABLE :: sim_F22(:)
295       ! File ID
296       INTEGER, PARAMETER :: fid1 = 11
297       ! Current working directory
298       CHARACTER(LEN=255) :: cwd
299       ! - - - - - - - - - -
300       ! Arrays for optimization
301       INTEGER :: numParFix
302       REAL(8), ALLOCATABLE :: ParFix(:)
303       INTEGER :: numParFit
304       REAL(8), ALLOCATABLE :: ParFit(:)
305       REAL(8), ALLOCATABLE :: ParFitUpd(:)
306       REAL(8), ALLOCATABLE :: FitDeltas(:)
307       LOGICAL, ALLOCATABLE :: FitMask(:)
308       INTEGER :: maxiter_opt
309       REAL(8) :: tolerance_opt
310       REAL(8) :: Residuo
311       ! - - - - - - - - - -
312
313
314       ! - - - - - - - - - -
315       ! Leer parametros
316       OPEN(UNIT=fid1, FILE=TRIM(configurationFile), STATUS="OLD")
317
318       iStatus = FindStringInFile("*FIXPAR", fid1, .TRUE.)
319       READ(fid1,*) numParFix
320       ALLOCATE( ParFix(numParFix) )
321       READ(fid1,*) ParFix
322
323       iStatus = FindStringInFile("*FITPAR", fid1, .TRUE.)
324       READ(fid1,*) numParFit
325       ALLOCATE( ParFit(numParFit) )
326       READ(fid1,*) ParFit
327
328       iStatus = FindStringInFile('*OPTPAR', fid1, .TRUE.)
329       ALLOCATE( FitDeltas(numParFit) )
330       ALLOCATE( FitMask(numParFit) )
331       READ(fid1,*) maxiter_opt, tolerance_opt
332       READ(fid1,*) FitDeltas
333       READ(fid1,*) FitMask
334       CLOSE(UNIT=fid1)
335       ! - - - - - - - - - -
336       OPEN(UNIT=fid1, FILE='FitParUpd.txt', STATUS='old')
337       READ(fid1,*) ParFit
338       CLOSE(fid1)
339       ! - - - - - - - - - -
340
341       ! - - - - - - - - - -
342       ! Read experimental data
343       OPEN( UNIT=fid1, FILE='curva_experimental.txt', STATUS='old')
344       READ(fid1,*) numExpDataPoints
345       ALLOCATE( exp_lambdas(numExpDataPoints) )
346       ALLOCATE( exp_stress(numExpDataPoints) )
347       DO i=1,numExpDataPoints
348          READ(fid1, *) exp_lambdas(i), exp_stress(i)
349       END DO
350       exp_lambdas = exp_lambdas
351       exp_stress = exp_stress
352       CLOSE(fid1)
353       ! - - - - - - - - - -
```

```
354
355         ! - - - - - - - - - -
356         ALLOCATE( ParFitUpd(numParFit) )
357         ! - - - - - - - - - -
358         CALL BruteForceOptimization(ExtFun_Curva, &
359         numExpDataPoints, exp_lambdas, exp_stress, numParFix, ParFix, numParFit, ParFit, &
360         FitDeltas, FitMask, maxiter_opt, tolerance_opt, .TRUE., 99, &
361         ParFitUpd, Residuo)
362         ! - - - - - - - - - -
363         OPEN(UNIT=fid1, FILE='FitParUpd.txt', STATUS='replace')
364         WRITE(fid1,'(F20.8,␣E20.8,␣4F20.8)') ParFitUpd
365         WRITE(fid1,*) Residuo
366         CLOSE(fid1)
367         ! - - - - - - - - - -
368         ALLOCATE( sim_P_old(nDim, nDim, numExpDataPoints) )
369         CALL ExtFun_Curva(numExpDataPoints, exp_lambdas, numParFix, ParFix, numParFit, ParFit, sim_P_old)
370         ALLOCATE( sim_P(nDim, nDim, numExpDataPoints) )
371         CALL ExtFun_Curva(numExpDataPoints, exp_lambdas, numParFix, ParFix, numParFit, ParFitUpd, sim_P)
372         ! - - - - - - - - - -
373         OPEN(UNIT=fid1,FILE=outputFile,STATUS='replace')
374         WRITE(fid1,'(10A20)') 'exp_lambda', 'exp_stress', 'P11', 'P21', 'P12', 'P22', 'Pold11', 'Pold21', 'Pold12', '
                ↪ Pold22'
375         DO i=1,numExpDataPoints
376            WRITE(fid1,'(10E20.8E3)') exp_lambdas(i), exp_stress(i), sim_P(:,:,i), sim_P_old(:,:,i)
377         END DO
378         CLOSE(UNIT=fid1)
379         ! - - - - - - - - - -
380
381
382         ! - - - - - - - - - - -
383      END SUBROUTINE RVE_Optimizar
384      ! = = = = = = = = = =
385
386
387
388   END MODULE MOD_Programs
```

```
1   ! = = = = = = = = = =
2   MODULE MOD_RVE
3
4      ! - - - - - - - - - -
5      IMPLICIT NONE
6      PRIVATE ! NOTE_avoid_public_variables_if_possible
7      ! - - - - - - - - - -
8      PUBLIC :: RVE_Curva_Biaxial
9      PUBLIC :: RVE_Curva_Simple
10     ! - - - - - - - - - -
11
12  ! = = = = = = = = = =
13  CONTAINS
14  ! = = = = = = = = = =
15
16
17
18     ! = = = = = = = = = =
19     ! ROUTINE:
20     ! RVE_Fibers_Initialization
21     !
22     ! DESCRIPTION:
23     ! Given the number of fibers that compose the RVE
24     ! returns the fibers initial unity vectors as the columns of an array of size=(2, nFibers)
25     ! Also takes in an optional argument to rotate the entire RVE counter-clock-wise (in degrees)
26     ! = = = = = = = = = =
27     FUNCTION RVE_Deformation(FF, nMembers, members0) RESULT(lambdas)
28        ! - - - - - - - - - -
29        IMPLICIT NONE
30        ! - - - - - - - - - -
31        REAL(8), INTENT(IN) :: FF(2,2) ! Deformation gradient
32        INTEGER, INTENT(IN) :: nMembers ! Number of fibers that compose the RVE
33        REAL(8), INTENT(IN) :: members0(2,nMembers) ! Array with the fibers initial unity vectors as columns
```

```
34        ! - - - - - - - - - -
35        REAL(8) :: lambdas(nMembers) ! array with the stretch of each fiber of the RVE
36        ! - - - - - - - - - -
37        INTEGER :: i ! counter
38        REAL(8) :: member(2) ! member deformed vector
39        ! - - - - - - - - - -
40
41        ! - - - - - - - - - -
42        DO i=1,nMembers
43           member(1) = FF(1,1)*members0(i,1) + FF(1,2)*members0(i,2)
44           member(2) = FF(2,1)*members0(i,1) + FF(2,2)*members0(i,2)
45           lambdas(i) = DSQRT(SUM(member*member))
46        END DO
47        ! - - - - - - - - - -
48
49        ! - - - - - - - - - -
50     END FUNCTION RVE_Deformation
51     ! = = = = = = = = = =
52
53
54     ! = = = = = = = = = =
55     ! ROUTINE:
56     ! TODO_function_or_subroutine_name
57     !
58     ! DESCRIPTION:
59     ! TODO_describe_routine_purpose.
60     ! = = = = = = = = = =
61     FUNCTION RVE_Respuesta(FF, nBundlesRVE, bundles0, CC_n, CC_l, CC_t, volumeFraction, lr0) RESULT(fpkStress)
62        ! - - - - - - - - - -
63        USE CLASS_Curve, ONLY : Curva
64        USE MOD_Auxiliar, ONLY : OuterProduct_2x2, ComputeFromCurve
65        IMPLICIT NONE
66        ! - - - - - - - - - -
67        INTEGER, PARAMETER :: nDim = 2
68        ! - - - - - - - - - -
69        REAL(8), INTENT(IN) :: FF(nDim,nDim) ! Deformation gradient
70        INTEGER, INTENT(IN) :: nBundlesRVE ! Number of members that compose the RVE
71        REAL(8), INTENT(IN) :: bundles0(nDim,nBundlesRVE) ! Array with the initial unity vectors of the fibers
72        INTEGER, INTENT(IN) :: CC_n ! numero de puntos de la curva constitutiva del haz
73        REAL(8), INTENT(IN) :: CC_l(CC_n), CC_t(CC_n) ! arrays de la curva constitutiva del haz (lambdas y tensiones)
74        REAL(8), INTENT(IN) :: volumeFraction ! Volume fraction of the macroscopic material
75        REAL(8), INTENT(IN) :: lr0 ! valor medio de la distribucion de reclutamiento
76        ! - - - - - - - - - -
77        REAL(8) :: fpkStress(nDim,nDim) ! First Piola Kirchhoff Stress Tensor (average of the fibers)
78        ! - - - - - - - - - -
79        REAL(8) :: bundle(nDim) ! member deformed vector
80        REAL(8) :: b0xb0(nDim,nDim) ! outer product of a bundle direction with itself
81        REAL(8) :: lambda
82        INTEGER :: i ! counter
83        REAL(8) :: bundleStress
84        ! - - - - - - - - - -
85
86        ! - - - - - - - - - -
87        ! RVE Stress as a sum over the RVE members
88        fpkStress = 0.0d0
89        DO i=1,nBundlesRVE
90           ! Affine deformation of each member
91           bundle(1) = FF(1,1)*bundles0(1,i) + FF(1,2)*bundles0(2,i)
92           bundle(2) = FF(2,1)*bundles0(1,i) + FF(2,2)*bundles0(2,i)
93           lambda = DSQRT(SUM(bundle*bundle))
94           ! Stress excerted by the member
95           bundleStress = ComputeFromCurve(lambda, CC_n, CC_l, CC_t)
96           ! Sum bundle stress to the RVE stress
97           b0xb0 = OuterProduct_2x2(bundles0(1,i), bundles0(1,i))
98           fpkStress = fpkStress + bundleStress/lambda * b0xb0
99        END DO
100       ! Now complete the computation of RVE stress (products common to all members)
101 ! WRITE(*,*) fpkStress
102       fpkStress = ( (volumeFraction/lr0) / nBundlesRVE ) * MATMUL(FF,fpkStress)
103 ! WRITE(*,*) fpkStress
```

```
104       ! - - - - - - - - - - -
105
106       ! - - - - - - - - - - -
107     END FUNCTION RVE_Respuesta
108     ! = = = = = = = = = =
109
110
111     ! = = = = = = = = = =
112     SUBROUTINE RVE_Curva_Simple &
113        (nPoints, lambdaArray, &
114        nParam1, Param1, &
115        nParam2, Param2, &
116        fpkStress_array)
117        ! - - - - - - - - - - -
118        USE CLASS_Curve, ONLY : Curva
119        USE MOD_Bundle, ONLY : Get_BundlesVectors, Get_CurvaConstitutivaHaz
120        ! - - - - - - - - - - -
121        IMPLICIT NONE
122        ! - - - - - - - - - - -
123        INTEGER, PARAMETER :: nDim = 2
124        ! - - - - - - - - - - -
125        INTEGER, INTENT(IN) :: nPoints ! Number of points in the curve
126        REAL(8), INTENT(IN) :: lambdaArray(nPoints) ! Array with the lambda values for the nPoints
127        INTEGER, INTENT(IN) :: nParam1
128        REAL(8), INTENT(IN) :: Param1(nParam1)
129        INTEGER, INTENT(IN) :: nParam2
130        REAL(8), INTENT(IN) :: Param2(nParam2)
131        ! - - - - - - - - - - -
132        REAL(8), INTENT(OUT) :: fpkStress_array(nDim,nDim,nPoints)
133        ! - - - - - - - - - - -
134        ! Parametros RVE
135        INTEGER :: nBundlesRVE
136        REAL(8) :: thetaRVE
137        REAL(8), ALLOCATABLE :: bundles_v0(:,:)
138        REAL(8) :: volFraction
139        ! Parametros haz - elasticos
140        REAL(8) :: Et, Eb
141        ! Parametros haz - distribucion
142        REAL(8) :: lamr0, lamrS, lamr_min, lamr_max
143        ! Parametros haz - discretizacion curva constitutiva
144        INTEGER :: CC_nPuntos ! numero de puntos con los que se discretiza la curva constitutiva
145        INTEGER :: CC_precision ! numero de puntos de la curva de distribucion con los que se computa la curva
               ↪ constitutiva
146        REAL(8) :: CC_lam_min, CC_lam_max
147        ! Curva constitutiva
148        REAL(8), ALLOCATABLE :: CC_lam(:), CC_ten(:)
149        ! Auxiliares
150        INTEGER :: last, i
151        REAL(8) :: F11
152        REAL(8) :: FF(nDim,nDim)
153        ! - - - - - - - - - - -
154
155        ! - - - - - - - - - - -
156        ! Parametros
157        nBundlesRVE = NINT(Param1(1))
158        ALLOCATE( bundles_v0(nDim, nBundlesRVE) )
159        !bundles_v0 = RESHAPE( Param1(2:1+nDim*nBundlesRVE), SHAPE(bundles_v0) )
160        !last = 1 + nDim*nBundlesRVE
161        thetaRVE = Param1(2)
162        CALL Get_BundlesVectors(nBundlesRVE, thetaRVE, bundles_v0)
163        last = 2
164        !
165        CC_nPuntos = Param1(last+1)
166        CC_lam_min = Param1(last+2)
167        CC_lam_max = Param1(last+3)
168        CC_precision = Param1(last+4)
169        !
170        volFraction = Param2(1)
171        Et = Param2(2)
172        Eb = Param2(3)
```

```
173       lamr0 = Param2(4)
174       lamrS = Param2(5)
175       lamr_min = Param2(6)
176       lamr_max = Param2(7)
177       ! Hago una correccion: la fraccion de volumen la divido por lamr0 para no modificar la funcion RVE_Respuesta
178       volFraction = volFraction
179
180       ! - - - - - - - - - - -
181       ! Calcular curva constitutiva de un haz
182       CALL Get_CurvaConstitutivaHaz(CC_nPuntos, CC_lam_min, CC_lam_max, &
183       Et, Eb, lamr0, lamrS, lamr_min, lamr_max, &
184       CC_precision, &
185       CC_lam, CC_ten)
186       ! - - - - - - - - - - -
187       ! Computing
188 ! WRITE(*,*) 'Computando curva en RVE_Curva_Simple()'
189 ! WRITE(*,*) 'Numero de puntos: ', nPoints
190       DO i=1,nPoints
191 ! WRITE(*,'(I4)',ADVANCE='NO') i
192          FF = RESHAPE([lambdaArray(i), 0.0d0, 0.0d0, 1.0d0], SHAPE(FF))
193          fpkStress_array(:,:,i) = RVE_Respuesta(FF, nBundlesRVE, bundles_v0, CC_nPuntos, CC_lam, CC_ten, volFraction,
                 ↪ lamr0)
194       END DO
195 ! WRITE(*,*)
196       ! - - - - - - - - - - -
197
198       ! - - - - - - - - - - -
199    END SUBROUTINE RVE_Curva_Simple
200    ! = = = = = = = = = = =
201
202    ! = = = = = = = = = = =
203    SUBROUTINE RVE_Curva_Biaxial &
204       (nPoints, lambdaArray, &
205       nParam1, Param1, &
206       nParam2, Param2, &
207       fpkStress_array)
208       ! - - - - - - - - - - -
209       USE CLASS_Curve, ONLY : Curva
210       USE MOD_Bundle, ONLY : Get_CurvaConstitutivaHaz
211       ! - - - - - - - - - - -
212       IMPLICIT NONE
213       ! - - - - - - - - - - -
214       INTEGER, PARAMETER :: nDim = 2
215       ! - - - - - - - - - - -
216       INTEGER, INTENT(IN) :: nPoints ! Number of points in the curve
217       REAL(8), INTENT(IN) :: lambdaArray(nPoints) ! Array with the lambda values for the nPoints
218       INTEGER, INTENT(IN) :: nParam1
219       REAL(8), INTENT(IN) :: Param1(nParam1)
220       INTEGER, INTENT(IN) :: nParam2
221       REAL(8), INTENT(IN) :: Param2(nParam2)
222       ! - - - - - - - - - - -
223       REAL(8), INTENT(OUT) :: fpkStress_array(nDim,nDim,nPoints)
224       ! - - - - - - - - - - -
225       ! Parametros RVE
226       INTEGER :: nBundlesRVE
227       REAL(8), ALLOCATABLE :: bundles_v0(:,:)
228       REAL(8) :: volFraction
229       ! Parametros haz - elasticos
230       REAL(8) :: Et, Eb
231       ! Parametros haz - distribucion
232       REAL(8) :: lamr0, lamrS, lamr_min, lamr_max
233       ! Parametros haz - discretizacion curva constitutiva
234       INTEGER :: CC_nPuntos, CC_precision
235       REAL(8) :: CC_lam_min, CC_lam_max
236       ! Curva constitutiva
237       REAL(8), ALLOCATABLE :: CC_lam(:), CC_ten(:)
238       ! Auxiliares
239       INTEGER :: last, i
240       REAL(8) :: F11
241       REAL(8) :: FF(nDim,nDim)
```

```
242       ! - - - - - - - - - - -
243
244       ! - - - - - - - - - - -
245       ! Parametros
246       nBundlesRVE = NINT(Param1(1))
247       ALLOCATE( bundles_v0(nDim, nBundlesRVE) )
248       bundles_v0 = RESHAPE( Param1(2:1+nDim*nBundlesRVE), SHAPE(bundles_v0) )
249       last = 1 + nDim*nBundlesRVE
250       !
251       CC_nPuntos = Param1(last+1)
252       CC_lam_min = Param1(last+2)
253       CC_lam_max = Param1(last+3)
254       CC_precision = Param1(last+4)
255       !
256       volFraction = Param2(1)
257       Et = Param2(2)
258       Eb = Param2(3)
259       lamr0 = Param2(4)
260       lamrS = Param2(5)
261       lamr_min = Param2(6)
262       lamr_max = Param2(7)
263       ! - - - - - - - - - - -
264       ! Calcular curva constitutiva de un haz
265       CALL Get_CurvaConstitutivaHaz(CC_nPuntos, CC_lam_min, CC_lam_max, &
266       Et, Eb, lamr0, lamrS, lamr_min, lamr_max, &
267       CC_precision, &
268       CC_lam, CC_ten)
269       ! - - - - - - - - - - -
270       ! Computing
271 ! WRITE(*,*) 'Computando curva en RVE_Curva_Biaxial()'
272 ! WRITE(*,*) 'Numero de puntos: ', nPoints
273       DO i=1,nPoints
274 ! WRITE(*,'(I4)',ADVANCE='NO') i
275          FF = RESHAPE([lambdaArray(i), 0.0d0, 0.0d0, lambdaArray(i)], SHAPE(FF))
276          fpkStress_array(:,:,i) = RVE_Respuesta(FF, nBundlesRVE, bundles_v0, CC_nPuntos, CC_lam, CC_ten, volFraction,
                 ↪ lamr0)
277       END DO
278 ! WRITE(*,*)
279       ! - - - - - - - - - - -
280
281       ! - - - - - - - - - - -
282    END SUBROUTINE RVE_Curva_Biaxial
283    ! = = = = = = = = = = =
284
285 END MODULE MOD_RVE
286 ! = = = = = = = = = = =
```

```
1  ! = = = = = = = = = = =
2  MODULE MOD_Optimization
3
4     ! - - - - - - - - - - -
5     IMPLICIT NONE
6     PRIVATE ! NOTE_avoid_public_variables_if_possible
7     ! - - - - - - - - - - -
8     PUBLIC :: BruteForceOptimization
9     ! - - - - - - - - - - -
10
11 ! = = = = = = = = = = =
12 CONTAINS
13 ! = = = = = = = = = = =
14
15    ! = = = = = = = = = = =
16    SUBROUTINE BruteForceOptimization &
17       (FITFUN, nExpData, expData_x, expData_y, &
18       nFixPar, FIXPAR, nFitPar, FITPAR, FITDEL, FITMASK, &
19       maxiter, Tolerancia, &
20       verbose, logfileID, &
21       FitParUpd, Residuo)
22       ! - - - - - - - - - - -
23       ! DESCRIPTION:
```

```
24      ! Optimization of parameters to minimize sum of squared deviations
25      ! between FITFUN and experimentalData
26      ! - - - - - - - - - - -
27      USE MOD_Auxiliar, ONLY : DeltaArray, Compute_SSR
28      ! - - - - - - - - - - -
29      IMPLICIT NONE
30      ! - - - - - - - - - - -
31      EXTERNAL :: FITFUN
32      INTERFACE
33         SUBROUTINE FITFUN(nPoints, xValues, nFixParam, FixParam, nFitParam, FitParam, yValues)
34            INTEGER, INTENT(IN) :: nPoints
35            REAL(8), INTENT(IN) :: xValues(nPoints)
36            INTEGER, INTENT(IN) :: nFixParam
37            REAL(8), INTENT(IN) :: FixParam(nFixParam)
38            INTEGER, INTENT(IN) :: nFitParam
39            REAL(8), INTENT(IN) :: FitParam(nFitParam)
40            REAL(8), INTENT(OUT) :: yValues(2,2,nPoints)
41         END SUBROUTINE
42      END INTERFACE
43      INTEGER, INTENT(IN) :: nExpData
44      REAL(8), INTENT(IN) :: expData_x(nExpData)
45      REAL(8), INTENT(IN) :: expData_y(nExpData)
46      INTEGER, INTENT(IN) :: nFixPar
47      REAL(8), INTENT(IN) :: FIXPAR(nFixPar)
48      INTEGER, INTENT(IN) :: nFitPar
49      REAL(8), INTENT(IN) :: FITPAR(nFitPar)
50      REAL(8), INTENT(IN) :: FITDEL(nFitPar)
51      LOGICAL, INTENT(IN) :: FITMASK(nFitPar)
52      INTEGER, INTENT(IN) :: maxiter
53      REAL(8), INTENT(IN) :: Tolerancia
54      LOGICAL, INTENT(IN) :: verbose
55      INTEGER, INTENT(IN) :: logfileID
56      ! - - - - - - - - - - -
57      REAL(8), INTENT(OUT) :: FitParUpd(nFitPar) ! Fitting parameters updated
58      REAL(8), INTENT(OUT) :: Residuo
59      ! - - - - - - - - - - -
60      REAL(8) :: MEA ! Mean of experimental array
61      REAL(8) :: TSS ! Total sum of squares
62      REAL(8) :: SSR ! Sum of squared deviations
63      REAL(8) :: SSR_plus
64      REAL(8) :: SSR_minus
65      INTEGER :: i, j
66      REAL(8) :: simData_y_full(2,2,nExpData)
67      REAL(8) :: simData_y(nExpData)
68      REAL(8) :: FitParVar(nFitPar) ! Fitting parameters variated
69      INTEGER :: min_index
70      INTEGER :: FitParChange(nFitPar) ! Change in fitting parameters per iteration
71      LOGICAL :: FitSuccess ! Dictates if optimization was succesfull
72      ! - - - - - - - - - - -
73
74      ! - - - - - - - - - - -
75      FitSuccess = .FALSE.
76      FitParUpd = FITPAR
77      ! - - - - - - - - - - -
78      MEA = SUM(expData_y) / nExpData
79      TSS = SUM((expData_y - MEA)*(expData_y - MEA))
80      ! Centered value of SSR
81      CALL FITFUN(nExpData, expData_x, nFixPar, FIXPAR, nFitPar, FitParUpd, simData_y_full)
82      simData_y = simData_y_full(1,1,:)
83      SSR = Compute_SSR(expData_y, simData_y, nExpData)
84      Residuo = SSR/TSS
85      WRITE(*,'(A30)') 'Starting␣with␣parameters:␣'
86      WRITE(*,*) FitParUpd
87      WRITE(*,*) 'Residuo:␣', Residuo
88      ! - - - - - - - - - - -
89      DO i=1,maxiter
90         ! Check if tolerance is met
91         IF ( Residuo < Tolerancia ) THEN
92            WRITE(*,*) 'Achieved␣Residuo␣tolerance'
93            FitSuccess = .TRUE.
```

```
94            EXIT
95         END IF
96         DO j=1,nFitPar
97            WRITE(*,'(A14,␣I4,␣A3)', ADVANCE='NO') 'Parameter:␣', j, '->␣'
98            ! Check if the parameter is to be optimized
99            IF ( .NOT. FITMASK(j) ) THEN
100              WRITE(*,'(A15)') '-----------.'
101              FitParChange(j) = 0
102              CYCLE
103           END IF
104           ! Plus value of SSR
105           FitParVar = FitParUpd + FITDEL(j)*DeltaArray(nFitPar,j)
106           CALL FITFUN(nExpData, expData_x, nFixPar, FIXPAR, nFitPar, FitParVar, simData_y_full)
107           simData_y = simData_y_full(1,1,:)
108           SSR_plus = Compute_SSR(expData_y, simData_y, nExpData)
109           ! Minus value of SSR
110           FitParVar = FitParUpd - FITDEL(j)*DeltaArray(nFitPar,j)
111           CALL FITFUN(nExpData, expData_x, nFixPar, FIXPAR, nFitPar, FitParVar, simData_y_full)
112           simData_y = simData_y_full(1,1,:)
113           SSR_minus = Compute_SSR(expData_y, simData_y, nExpData)
114           ! Move along the direction which SSR is minimum
115           min_index = MINLOC( [SSR, SSR_plus, SSR_minus], DIM=1 )
116           SELECT CASE (min_index)
117           CASE (1)
118              WRITE(*,'(A15)') 'Unchanged.'
119              FitParChange(j) = 0
120           CASE (2)
121              WRITE(*,'(A15)') 'Incremented.'
122              FitParChange(j) = 1
123           CASE (3)
124              WRITE(*,'(A15)') 'Decremented.'
125              FitParChange(j) = -1
126           CASE DEFAULT
127              WRITE(*,'(A50)') 'Error␣in␣searching␣of␣a␣minimum␣for␣parameter.'
128              STOP
129           END SELECT
130        END DO
131        ! Updated fitting parameters
132        FitParUpd = FitParUpd + FitParChange*FITDEL
133        ! Centered value of SSR
134        CALL FITFUN(nExpData, expData_x, nFixPar, FIXPAR, nFitPar, FitParUpd, simData_y_full)
135        simData_y = simData_y_full(1,1,:)
136        SSR = Compute_SSR(expData_y, simData_y, nExpData)
137        Residuo = SSR/TSS
138        ! Check if there is some change
139        WRITE(*,'(A12,␣I7,␣A30)') 'Iteration␣', i, 'complete␣with␣parameters:'
140        WRITE(*,*) FitParUpd
141        WRITE(*,*) 'Residuo:␣', Residuo
142        IF ( ALL(FitParChange == 0) ) THEN
143           WRITE(*,*) 'No␣change␣in␣parameters,␣local␣minimum␣found.'
144           FitSuccess = .TRUE.
145           EXIT
146        END IF
147     END DO
148     ! - - - - - - - - - -
149     WRITE(*,*) 'Optimization␣ended␣with␣succes␣status:␣', FitSuccess
150     ! - - - - - - - - - -
151
152     ! - - - - - - - - - -
153  END SUBROUTINE BruteForceOptimization
154  ! = = = = = = = = = =
155
156
157  ! = = = = = = = = = =
158  SUBROUTINE ConjugateGradientOptimization &
159     (FITFUN, nExpData, expData_x, expData_y, &
160     nFixPar, FIXPAR, nFitPar, FITPAR, FITDEL, FITMASK, &
161     maxiter, R2tolerance, &
162     verbose, logfileID, &
163     FitParUpd, R2)
```

```
164       ! - - - - - - - - - -
165       ! DESCRIPTION:
166       ! Optimization of parameters to minimize sum of squared deviations
167       ! between FITFUN and experimentalData
168       ! - - - - - - - - - -
169       USE MOD_Auxiliar, ONLY : DeltaArray, VxV_rR, TxV_rV, VxTxV_rR, Compute_SSR
170       ! - - - - - - - - - -
171       IMPLICIT NONE
172       ! - - - - - - - - - -
173       EXTERNAL :: FITFUN
174       INTERFACE
175          FUNCTION FITFUN(nPoints, xValues, nFixParam, FixParam, nFitParam, FitParam) RESULT(yValues)
176             INTEGER, INTENT(IN) :: nPoints
177             REAL(8), INTENT(IN) :: xValues(nPoints)
178             INTEGER, INTENT(IN) :: nFixParam
179             REAL(8), INTENT(IN) :: FixParam(nFixParam)
180             INTEGER, INTENT(IN) :: nFitParam
181             REAL(8), INTENT(IN) :: FitParam(nFitParam)
182             REAL(8) :: yValues(nPoints)
183          END FUNCTION
184       END INTERFACE
185       INTEGER, INTENT(IN) :: nExpData
186       REAL(8), INTENT(IN) :: expData_x(nExpData)
187       REAL(8), INTENT(IN) :: expData_y(nExpData)
188       INTEGER, INTENT(IN) :: nFixPar
189       REAL(8), INTENT(IN) :: FIXPAR(nFixPar)
190       INTEGER, INTENT(IN) :: nFitPar
191       REAL(8), INTENT(IN) :: FITPAR(nFitPar)
192       REAL(8), INTENT(IN) :: FITDEL(nFitPar)
193       LOGICAL, INTENT(IN) :: FITMASK(nFitPar)
194       INTEGER, INTENT(IN) :: maxiter
195       REAL(8), INTENT(IN) :: R2tolerance
196       LOGICAL, INTENT(IN) :: verbose
197       INTEGER, INTENT(IN) :: logfileID
198       ! - - - - - - - - - -
199       REAL(8), INTENT(OUT) :: FitParUpd(nFitPar) ! Fitting parameters updated
200       REAL(8), INTENT(OUT) :: R2
201       ! - - - - - - - - - -
202       REAL(8) :: MEA ! Mean of experimental array
203       REAL(8) :: TSS ! Total sum of squares
204       REAL(8) :: SSR ! Sum of squared deviations
205       REAL(8) :: SSR_p(nFitPar)
206       REAL(8) :: SSR_m(nFitPar)
207       REAL(8) :: SSR_pp
208       REAL(8) :: SSR_mm
209       INTEGER :: i, j, k
210       REAL(8) :: simData_y(nExpData)
211       REAL(8) :: FitParVar(nFitPar) ! Fitting parameters variated
212       INTEGER :: min_index
213       INTEGER :: FitParChange(nFitPar) ! Change in fitting parameters per iteration
214       LOGICAL :: FitSuccess ! Dictates if optimization was succesfull
215       ! - - - - - - - - - -
216       REAL(8) :: Jacobian(nFitPar)
217       REAL(8) :: Hessian(nFitPar, nFitPar)
218       REAL(8) :: alpha
219       REAL(8) :: aux1, aux2
220       ! - - - - - - - - - -
221
222       ! - - - - - - - - - -
223       FitSuccess = .FALSE.
224       FitParUpd = FITPAR
225       ! - - - - - - - - - -
226       MEA = SUM(expData_y) / nExpData
227       TSS = SUM((expData_y - MEA)*(expData_y - MEA))
228       ! Centered value of SSR
229       simData_y = FITFUN(nExpData, expData_x, nFixPar, FIXPAR, nFitPar, FitParUpd)
230       SSR = Compute_SSR(expData_y, simData_y, nExpData)
231       R2 = 1 - SSR/TSS
232       WRITE(*,'(A30)') 'Starting␣with␣parameters:␣'
233       WRITE(*,*) FitParUpd
```

```
234       WRITE(*,*) 'R2:␣', R2
235       ! - - - - - - - - - -
236       DO i=1,maxiter
237          ! Check if tolerance is met
238          IF ( R2 > R2tolerance ) THEN
239             WRITE(*,*) 'Achieved␣R2␣tolerance'
240             FitSuccess = .TRUE.
241             EXIT
242          END IF
243
244          DO j=1,nFitPar
245             ! Plus value of SSR
246             FitParVar = FitParUpd + FITDEL(j)*DeltaArray(nFitPar,j)
247             simData_y = FITFUN(nExpData, expData_x, nFixPar, FIXPAR, nFitPar, FitParVar)
248             SSR_p(j) = Compute_SSR(expData_y, simData_y, nExpData)
249             ! Minus value of SSR
250             FitParVar = FitParUpd - FITDEL(j)*DeltaArray(nFitPar,j)
251             simData_y = FITFUN(nExpData, expData_x, nFixPar, FIXPAR, nFitPar, FitParVar)
252             SSR_m(j) = Compute_SSR(expData_y, simData_y, nExpData)
253          END DO
254
255          DO j=1,nFitPar
256             ! Jacobian values
257             Jacobian(j) = (SSR_p(j) - SSR_m(j)) / ( 2.0d0 * FITDEL(j) )
258             ! Hessian diagonal values
259             Hessian(j,j) = ( SSR_p(j) - 2*SSR + SSR_m(j) ) / ( FITDEL(j)**2 )
260             DO k=j+1,nFitPar
261                ! Plus value of SSR
262                FitParVar = FitParUpd + FITDEL(j)*DeltaArray(nFitPar,j) + FITDEL(k)*DeltaArray(nFitPar,k)
263                simData_y = FITFUN(nExpData, expData_x, nFixPar, FIXPAR, nFitPar, FitParVar)
264                SSR_pp = Compute_SSR(expData_y, simData_y, nExpData)
265                ! Minus value of SSR
266                FitParVar = FitParUpd - FITDEL(j)*DeltaArray(nFitPar,j) - FITDEL(k)*DeltaArray(nFitPar,k)
267                simData_y = FITFUN(nExpData, expData_x, nFixPar, FIXPAR, nFitPar, FitParVar)
268                SSR_mm = Compute_SSR(expData_y, simData_y, nExpData)
269                ! Hessian non-diagonal values
270                Hessian(j,k) = ( SSR_pp - SSR_p(j) - SSR_p(k) + 2*SSR - SSR_m(j) - SSR_m(k) + SSR_mm ) / ( 2.0d0*FITDEL
                       ↪ (j)*FITDEL(k) )
271                Hessian(k,j) = Hessian(j,k)
272             END DO
273          END DO
274
275
276          aux1 = VxV_rR(Jacobian,Jacobian,nFitPar)
277          aux2 = VxTxV_rR(Jacobian, Hessian, Jacobian, nFitPar)
278          alpha = - aux1 / aux2
279
280          FitParUpd = FitParUpd - alpha * Jacobian
281          ! Centered value of SSR
282          simData_y = FITFUN(nExpData, expData_x, nFixPar, FIXPAR, nFitPar, FitParUpd)
283          SSR = Compute_SSR(expData_y, simData_y, nExpData)
284          R2 = 1 - SSR/TSS
285          ! Check if there is some change
286          WRITE(*,'(A12,␣I4,␣A30)') 'Iteration␣', i, 'complete␣with␣parameters:'
287          WRITE(*,*) FitParUpd
288          WRITE(*,*) 'R2:␣', R2
289
290       END DO
291       ! - - - - - - - - - -
292       WRITE(*,*) 'Optimization␣ended␣with␣succes␣status:␣', FitSuccess
293       ! - - - - - - - - - -
294
295       ! - - - - - - - - - -
296    END SUBROUTINE ConjugateGradientOptimization
297    ! = = = = = = = = = =
298
299 END MODULE MOD_Optimization
300 ! = = = = = = = = = =
```

```
1 ! = = = = = = = = = =
```

```
2  MODULE MOD_Bundle
3
4     ! - - - - - - - - -----
5     IMPLICIT NONE
6     PRIVATE ! NOTE_avoid_public_variables_if_possible
7     ! - - - - - - - - -----
8     PUBLIC :: Get_CurvaConstitutivaHaz
9     PUBLIC :: Get_BundlesVectors
10    ! - - - - - - - - -----
11
12 ! = = = = = = = = = =
13 CONTAINS
14 ! = = = = = = = = = =
15
16    ! = = = = = = = = = =
17    SUBROUTINE Get_CurvaConstitutivaHaz &
18       (CCh_nPuntos, lambda_min, lambda_max, &
19       Et, Eb, lamr0, lamrS, lamr_min, lamr_max, &
20       nPuntos_cdnt, &
21       CCh_lambdas, CCh_tensiones)
22       ! - - - - - - - - -
23       ! Calcula la curva constitutiva de un haz de fibras en funcion de de los
24       ! parametros constitutivos de una fibra y de los parametros de distribu-
25       ! cion normal.
26       ! Primero calcula la curva de distribucion normal truncada y normalizada
27       ! discreta, luego calcula el valor de tension realizando la integral dis-
28       ! creta para cada valor de deformacion del haz segun:
29       !
30       ! s_haz = int_-lam_min^+lam_max tf(lam, lamr)*n(lamr)*dlamr
31       !
32       ! donde: s_haz es la tension desarrollada por el haz
33       ! tf es la tension de la fibra
34       ! lam es la elongacion del haz
35       ! lamr es el valor de enrulamiento de cada fibra
36       ! n(lamr) es la densidad de probabilidad normal
37       !
38       !
39       !
40       ! notar: 1) que se suponen infinitas fibras en el haz
41       ! 2) que E es distinto a la traccion que a la flexion, por lo
42       ! que hay que desdoblar la integral en dos.
43       !
44       ! Ver apunte "haz_curvaconstitutiva_corregido"
45       ! Daniel Enrique Caballero - danielcaballero88@gmail.com
46       ! - - - - - - - - -
47       USE MOD_Auxiliar, ONLY : Get_CurvaDistribucionNormalTruncada, IntegCumTrapz, ComputeFromCurve
48       ! - - - - - - - - -
49       IMPLICIT NONE
50       ! - - - - - - - - -
51       INTEGER, INTENT(IN) :: CCh_nPuntos! cantidad de puntos de la curva constitutiva
52       REAL(8), INTENT(IN) :: lambda_min, lambda_max
53       REAL(8), INTENT(IN) :: Et, Eb, lamr0, lamrS, lamr_min, lamr_max
54       INTEGER, INTENT(IN) :: nPuntos_cdnt ! cantidad de puntos de la curva distribucion normal truncada
55       ! - - - - - - - - -
56       REAL(8), ALLOCATABLE, INTENT(OUT) :: CCh_lambdas(:)
57       REAL(8), ALLOCATABLE, INTENT(OUT) :: CCh_tensiones(:)
58       ! - - - - - - - - -
59       ! curva distribucion normal truncada y normalizada
60       REAL(8) :: lamr(nPuntos_cdnt), pdf1(nPuntos_cdnt), cdf1(nPuntos_cdnt)
61       ! integrandos e integrados de la curva de distribucion:
62       REAL(8) :: pdf2(nPuntos_cdnt), cdf2(nPuntos_cdnt), pdf3(nPuntos_cdnt), cdf3(nPuntos_cdnt)
63       ! auxiliares
64       REAL(8) :: dlambda
65       INTEGER :: ii
66       REAL(8) :: lambda_i, C1, C2, C3, tension_i
67       ! - - - - - - - - -
68
69       ! - - - - - - - - -
70       ALLOCATE( CCh_lambdas(CCh_nPuntos) )
71       ALLOCATE( CCh_tensiones(CCh_nPuntos) )
```

```
      ! - - - - - - - - - -
      dlambda = (lambda_max-lambda_min)/DFLOAT(CCh_nPuntos-1)
      ! - - - - - - - - - -
      CALL Get_CurvaDistribucionNormalTruncada(lamr0, lamrS, nPuntos_cdnt, lamr_min, lamr_max, lamr, pdf1, cdf1)
      ! - - - - - - - - - -
      pdf2 = pdf1 * lamr
      pdf3 = pdf1 / lamr
      cdf2 = IntegCumTrapz(nPuntos_cdnt, lamr, pdf2)
      cdf3 = IntegCumTrapz(nPuntos_cdnt, lamr, pdf3)
      DO ii=1,CCh_nPuntos
         lambda_i = lambda_min + DFLOAT(ii-1)*dlambda
         C1 = ComputeFromCurve(lambda_i, nPuntos_cdnt, lamr, cdf1, .TRUE.)
         C2 = ComputeFromCurve(lambda_i, nPuntos_cdnt, lamr, cdf2, .TRUE.)
         C3 = ComputeFromCurve(lambda_i, nPuntos_cdnt, lamr, cdf3, .TRUE.)
         tension_i = Eb*(lambda_i-1.0d0)*(1.0d0 - C1) + Eb*(C2-C1) + Et*(lambda_i*C3-C1)
         CCh_lambdas(ii) = lambda_i
         CCh_tensiones(ii) = tension_i
      END DO
      ! - - - - - - - - - -

      ! - - - - - - - - - -
   END SUBROUTINE Get_CurvaConstitutivaHaz
   ! = = = = = = = = = =

   ! = = = = = = = = = =
   SUBROUTINE Get_BundlesVectors(nBundles, angulo_gra, b0)
      ! - - - - - - - - - -
      ! Given: the number of bundles in the RVE
      ! and the orientation angle in degrees
      ! Returns: the unit vectors of the bundles of the RVE
      ! - - - - - - - - - -
      IMPLICIT NONE
      ! - - - - - - - - - -
      REAL(8), PARAMETER :: PI = 4.0d0*ATAN(1.0d0)
      ! - - - - - - - - - -
      INTEGER, INTENT(IN) :: nBundles
      REAL(8), INTENT(IN) :: angulo_gra
      REAL(8), INTENT(OUT) :: b0(2,nBundles)
      ! - - - - - - - - - -
      REAL(8) :: angulo
      REAL(8) :: dang
      INTEGER :: ii
      REAL(8) :: angulo_ii
      ! - - - - - - - - - -

      ! - - - - - - - - - -
      angulo = angulo_gra * PI / 180.0d0
      dang = PI/DFLOAT(nBundles)
      ! - - - - - - - - - -
      DO ii=1,nBundles
         angulo_ii = angulo + DFLOAT(ii-1)*dang
         b0(1,ii) = DCOS(angulo_ii)
         b0(2,ii) = DSIN(angulo_ii)
      END DO
      ! - - - - - - - - - -

      ! - - - - - - - - - -
   END SUBROUTINE
   ! = = = = = = = = = =


END MODULE MOD_Bundle
! = = = = = = = = = =
```

```
! = = = = = = = = = =
MODULE MOD_Auxiliar

   ! - - - - - - - - - -
   IMPLICIT NONE
   PRIVATE ! NOTE_avoid_public_variables_if_possible
```

```
7     ! - - - - - - - - -
8     PUBLIC :: ComputeFromCurve
9     PUBLIC :: OuterProduct_2x2
10    PUBLIC :: FindStringInFile
11    PUBLIC :: DeltaArray
12    PUBLIC :: TxV_rV
13    PUBLIC :: VxTxV_rR
14    PUBLIC :: VxV_rR
15    PUBLIC :: Compute_SSR
16    PUBLIC :: Get_CurvaDistribucionNormalTruncada
17    PUBLIC :: IntegCumTrapz
18    ! - - - - - - - - -
19
20  ! = = = = = = = = = =
21  CONTAINS
22  ! = = = = = = = = = =
23
24
25    ! = = = = = = = = = =
26    FUNCTION ComputeFromCurve(x, nPoints, xArray, yArray, extrapolar_IN) RESULT(y)
27       ! - - - - - - - - -
28       ! Esta funcion calcula el valor y=f(x) de una curva f(x) discreta
29       ! INPUT:
30       ! x= valor de variable independiente
31       ! nPoints= numero de puntos en la curva discreta
32       ! xArray= valores de x de la curva discreta
33       ! yArray= valores de y de la curva discreta
34       ! extrapolar_IN= opcional, indica si se extrapola en caso de que
35       ! x caiga fuera de xArray. Se interpola con los
36       ! valores de los extremos
37       ! - - - - - - - - -
38       IMPLICIT NONE
39       ! - - - - - - - - -
40       REAL(8), INTENT(IN) :: x
41       INTEGER, INTENT(IN) :: nPoints
42       REAL(8), INTENT(IN) :: xArray(nPoints)
43       REAL(8), INTENT(IN) :: yArray(nPoints)
44       LOGICAL, OPTIONAL, INTENT(IN) :: extrapolar_IN
45       ! - - - - - - - - -
46       REAL(8) :: y
47       ! - - - - - - - - -
48       LOGICAL :: extrapolar
49       LOGICAL :: fdi1, fdi2 ! logicals "fuera de intervalo"
50       INTEGER :: i_pre ! indice del valor previo a x en xArray
51       REAL(8) :: slope
52       ! - - - - - - - - -
53
54       ! - - - - - - - - -
55       extrapolar = .FALSE.
56       IF ( PRESENT(extrapolar_IN) ) THEN
57          extrapolar = extrapolar_IN
58       END IF
59       ! - - - - - - - - -
60       fdi1 = ( x < xArray(1) )
61       fdi2 = ( x > xArray(nPoints) )
62       IF ( (fdi1) .OR. (fdi2) ) THEN
63          IF ( extrapolar) THEN
64             IF (fdi1) THEN
65                y = yArray(1)
66             ELSE
67                y = yArray(nPoints)
68             END IF
69          ELSE
70             WRITE(*,*) 'Error, trying to comput out of bounds - ComputeFromCurve()'
71             WRITE(*,*) 'xmin, xmax, x:', xArray(1), xArray(nPoints), x
72             STOP
73          END IF
74       ELSE
75          DO i_pre=2,nPoints
76             IF (x < xArray(i_pre)) THEN
```

```
77              EXIT
78            END IF
79          END DO
80          i_pre = i_pre - 1
81          ! linear interpolation within the sub-interval
82          slope = ( yArray(i_pre+1) - yArray(i_pre) ) / ( xArray(i_pre+1) - xArray(i_pre) )
83          y = yArray(i_pre) + slope * (x - xArray(i_pre))
84       END IF
85       ! - - - - - - - - - -
86
87       ! - - - - - - - - - -
88    END FUNCTION ComputeFromCurve
89    ! = = = = = = = = = =
90
91
92    ! = = = = = = = = = =
93    FUNCTION OuterProduct_2x2(x, y) RESULT(outer) ! Producto tensorial entre 2 vectores
94       ! - - - - - - - - - -
95       IMPLICIT NONE
96       ! - - - - - - - - - -
97       ! Parameters
98       INTEGER, PARAMETER :: nDim = 2
99       ! - - - - - - - - - -
100      ! Arguments
101      REAL(8), INTENT(IN) :: x(nDim), y(nDim) !vectors of size nDim
102      ! - - - - - - - - - -
103      ! Result
104      REAL(8) :: outer(nDim,nDim)
105      ! - - - - - - - - - -
106      ! Locals
107      INTEGER :: i,j
108      ! - - - - - - - - - -
109
110      ! - - - - - - - - - -
111      DO i=1,nDim
112         DO j=1,nDim
113            outer(i,j) = x(i)*y(j)
114         END DO
115      END DO
116      ! - - - - - - - - - -
117
118      ! - - - - - - - - - -
119   END FUNCTION
120   ! ============================================================================
121
122
123   ! = = = = = = = = = ===
124   ! = = = = = = = = = ===
125   FUNCTION FindStringInFile(Str, ioUnit, Mandatory) RESULT (iError)
126      ! = = = = = = = = = ===
127      ! Busca un String en un archivo (STR), sino lo encuentra rebovina el archivo
128      ! y pone iError < 0 como indicador de no hallazgo
129      ! Str: String to find, ioUnit: Unit assigned to Input File; iError: Error Status variable
130      ! = = = = = = = = = ===
131      IMPLICIT NONE
132      ! = = = = = = = = = ===
133      ! Parameters
134      LOGICAL,PARAMETER :: Segui=.True.
135      ! = = = = = = = = = ===
136      ! Arguments
137      CHARACTER(*), INTENT(IN) :: Str
138      INTEGER, INTENT (IN) :: ioUnit
139      LOGICAL, OPTIONAL, INTENT(IN) :: Mandatory
140      ! = = = = = = = = = ===
141      ! Locals
142      LOGICAL :: MandatoryL
143      CHARACTER(LEN=120) :: DummyString
144      CHARACTER(LEN=120) :: upStr
145      INTEGER :: iError
146      INTEGER :: Leng
```

```
147       ! = = = = = = = = = ===
148
149       ! = = = = = = = = = ===
150       IF ( PRESENT(Mandatory) ) THEN
151          MandatoryL = Mandatory
152       ELSE
153          MandatoryL = .FALSE.
154       END IF
155       ! = = = = = = = = = ===
156       iError=0
157       Leng = LEN_TRIM(Str)
158       upStr = Upper_Case(Str) ! Line added by NB. Converts Str to Upper Case string
159       ! = = = = = = = = = ===
160       REWIND(ioUnit)
161       Search_Loop: DO WHILE (segui)
162          READ(ioUnit,'(1A120)',IOSTAT=iError) DummyString
163          DummyString = Upper_Case(DummyString) ! line added by NB
164          ! if (iError==59) backspace(ioUnit)
165          IF (iError.lt.0) THEN
166             REWIND (ioUnit)
167             EXIT Search_Loop
168          END IF
169          IF ( DummyString(1:1) /= '*' ) CYCLE Search_Loop
170          IF ( DummyString(1:Leng) == upStr(1:Leng) ) EXIT Search_Loop
171       END DO Search_Loop
172       ! = = = = = = = = = ===
173       IF (MandatoryL) THEN
174          IF ( .NOT. ( iError == 0 ) ) THEN
175             WRITE(*,*) upStr, 'NOT␣FOUND'
176             STOP
177          END IF
178       END IF
179       ! = = = = = = = = = ===
180
181    END FUNCTION FindStringInFile
182    ! = = = = = = = = = ===
183    ! = = = = = = = = = ===
184
185
186    ! = = = = = = = = = ===
187    ! = = = = = = = = = ===
188    FUNCTION Upper_Case(string)
189
190       !-----------------------------------------------------------------------------------------
191       !! The Upper_Case function converts the lower case characters of a string to upper case one.
192       !! Use this function in order to achieve case-insensitive: all character variables used
193       !! are pre-processed by Uppper_Case function before these variables are used. So the users
194       !! can call functions without pey attention of case of the keywords passed to the functions.
195       !-----------------------------------------------------------------------------------------
196
197       IMPLICIT NONE
198
199       !-----------------------------------------------------------------------------------------
200       CHARACTER(LEN=*), INTENT(IN):: string ! string to be converted
201       CHARACTER(LEN=LEN(string)):: Upper_Case ! converted string
202       INTEGER:: n1 ! characters counter
203       !-----------------------------------------------------------------------------------------
204
205       Upper_Case = string
206       DO n1=1,LEN(string)
207          SELECT CASE(ICHAR(string(n1:n1)))
208             CASE(97:122)
209                Upper_Case(n1:n1)=CHAR(ICHAR(string(n1:n1))-32) ! Upper case conversion
210          END SELECT
211       ENDDO
212       RETURN
213
214       !-----------------------------------------------------------------------------------------
215    END FUNCTION Upper_Case
216    ! = = = = = = = = = ===
```

```
217   ! = = = = = = = = = ===
218
219
220   ! = = = = = = = = = =
221   FUNCTION DeltaArray(n,i) RESULT(d)
222      ! - - - - - - - - - -
223      IMPLICIT NONE
224      ! - - - - - - - - - -
225      INTEGER, INTENT(IN) :: n,i
226      REAL(8) :: d(n)
227      ! - - - - - - - - - -
228
229      ! - - - - - - - - - -
230      d=0.0d0
231      d(i) = 1.0d0
232      ! - - - - - - - - - -
233
234      ! - - - - - - - - - -
235   END FUNCTION
236   ! = = = = = = = = = =
237
238
239   ! = = = = = = = = = =
240   FUNCTION VxV_rR(v1, v2, n) RESULT(r)
241      IMPLICIT NONE
242      REAL(8), INTENT(IN) :: v1(n), v2(n)
243      INTEGER, INTENT(IN) :: n
244      REAL(8) :: r
245      INTEGER :: i
246
247      r = 0.0d0
248      DO i=1,n
249         r = r + v1(i)*v2(i)
250      END DO
251
252   END FUNCTION
253   ! = = = = = = = = = =
254
255
256   ! = = = = = = = = = =
257   FUNCTION TxV_rV(T, v, n) RESULT(w)
258      IMPLICIT NONE
259      REAL(8), INTENT(IN) :: T(n,n), v(n)
260      INTEGER, INTENT(IN) :: n
261      REAL(8) :: w(n)
262      INTEGER :: i, j
263
264      w = 0.0d0
265      DO i=1,n
266         DO j=1,n
267            w(i) = w(i) + T(i,j)*v(j)
268         END DO
269      END DO
270
271   END FUNCTION
272   ! = = = = = = = = = =
273
274
275   ! = = = = = = = = = =
276   FUNCTION VxTxV_rR(v1, T, v2, n) RESULT(r)
277      IMPLICIT NONE
278      REAL(8), INTENT(IN) :: v1(n), T(n,n), v2(n)
279      INTEGER, INTENT(IN) :: n
280      REAL(8) :: r
281      REAL(8) :: w(n)
282      INTEGER :: i,j
283
284      w = TxV_rV(T,v2,n)
285      r = VxV_rR(v1,w,n)
286
```

```
287    END FUNCTION
288    ! = = = = = = = = = = =
289
290
291    ! = = = = = = = = = = =
292    FUNCTION Compute_SSR(expArray, simArray, n) RESULT(SSR)
293       ! - - - - - - - - - -
294       ! Calculo de residuo global como suma de residuos cuadrados
295       ! (sum of squared residuals: SSR)
296       ! - - - - - - - - - -
297       IMPLICIT NONE
298       REAL(8), INTENT(IN) :: expArray(n), simArray(n)
299       INTEGER, INTENT(IN) :: n
300       REAL(8) :: SSR
301
302       SSR = SUM((simArray- expArray)*(simArray- expArray))
303       !SSR = SSR + 100.d0*simArray(1)*simArray(1)
304       !SSR = SSR + 1000.d0*SUM(simArray(1:5)*simArray(1:5))
305
306    END FUNCTION
307    ! = = = = = = = = = = =
308
309
310    ! = = = = = = = = = = =
311    SUBROUTINE Get_CurvaDistribucionNormal(mu, sigma, nPuntos, xx, pdf, cdf)
312       ! - - - - - - - - - -
313       ! Devuelve una curva de distribucion normal
314       ! mu = valor medio
315       ! sigma = desviacion estandar
316       ! npuntos = numero de puntos de los arrays de la curva
317       ! xx = array de valores de abscisas
318       ! pdf = array de distribucion normal
319       ! cdf = array de distribucion normal acumulada
320       !
321       ! NOTA: la curva esta truncada en "mu-10*sigma" y "mu+10*sigma"
322       ! - - - - - - - - - -
323       IMPLICIT NONE
324       ! - - - - - - - - - -
325       REAL(8), PARAMETER :: PI = 4.0d0*DATAN(1.0d0)
326       ! - - - - - - - - - -
327       REAL(8), INTENT(IN) :: mu, sigma
328       INTEGER, INTENT(IN) :: nPuntos
329       REAL(8), INTENT(OUT) :: xx(nPuntos), pdf(nPuntos), cdf(nPuntos)
330       ! - - - - - - - - - -
331       REAL(8) :: x_ini, x_fin, dx, x
332       INTEGER :: ii
333       ! - - - - - - - - - -
334
335       ! - - - - - - - - - -
336       x_ini = mu - 10.0d0*sigma
337       x_fin = mu + 10.0d0*sigma
338       dx = (x_fin - x_ini ) / DFLOAT(nPuntos-1)
339       ! - - - - - - - - - -
340       DO ii=1,nPuntos
341          x = x_ini + DFLOAT(ii-1)*dx
342          xx(ii) = x
343          pdf(ii) = (1.0d0 / DSQRT(2.0d0*PI*sigma**2)) * DEXP( - (x-mu)**2 / (2.0d0*sigma**2) )
344          cdf(ii) = 0.5d0 * ( 1.0d0 + ERF( (x-mu) / (DSQRT(2.0d0)*sigma) ) )
345       END DO
346       ! - - - - - - - - - -
347
348       ! - - - - - - - - - -
349    END SUBROUTINE Get_CurvaDistribucionNormal
350    ! = = = = = = = = = = =
351
352
353    ! = = = = = = = = = = =
354    SUBROUTINE Get_CurvaDistribucionNormalTruncada(mu, sigma, nPuntos, x_min, x_max, xx, pdf, cdf)
355       ! - - - - - - - - - -
356       ! Devuelve una curva de distribucion normal
```

```
357       ! la curva esta truncada entre "x_min" y "x_max"
358       ! y renormalizada para que el area bajo la curva sea 1
359       !
360       ! mu = valor medio
361       ! sigma = desviacion estandar
362       ! npuntos = numero de puntos de los arrays de la curva
363       ! xx = array de valores de abscisas
364       ! pdf = array de distribucion normal
365       ! cdf = array de distribucion normal acumulada
366       ! - - - - - - - - - -
367       IMPLICIT NONE
368       ! - - - - - - - - - -
369       REAL(8), PARAMETER :: PI = 4.0d0*DATAN(1.0d0)
370       ! - - - - - - - - - -
371       REAL(8), INTENT(IN) :: mu, sigma
372       INTEGER, INTENT(IN) :: nPuntos
373       REAL(8), INTENT(IN) :: x_min, x_max ! limites donde se trunca la distribucion normal
374       REAL(8), INTENT(OUT) :: xx(nPuntos), pdf(nPuntos), cdf(nPuntos)
375       ! - - - - - - - - - -
376       REAL(8) :: dx, x
377       INTEGER :: ii
378       REAL(8) :: aux
379       ! - - - - - - - - - -
380
381       ! - - - - - - - - - -
382       ! calculo el valor de espaciado entre puntos
383       dx = (x_max - x_min ) / DFLOAT(nPuntos-1)
384       ! - - - - - - - - - -
385       ! ensamblo el array de abscisas (xx) y el de ordenadas (pdf)
386       DO ii=1,nPuntos
387          x = x_min + DFLOAT(ii-1)*dx
388          xx(ii) = x
389          pdf(ii) = (1.0d0 / DSQRT(2.0d0*PI*sigma**2)) * DEXP( - (x-mu)**2 / (2.0d0*sigma**2) )
390       END DO
391       ! - - - - - - - - - -
392       ! computo el array de distribucion acumulada sin normalizar (cdf)
393       cdf = IntegCumTrapz(nPuntos, xx, pdf)
394       ! normalizo las curvas
395       aux = cdf(nPuntos)
396       DO ii=1,nPuntos
397          pdf(ii) = pdf(ii) / aux
398          cdf(ii) = cdf(ii) / aux
399       END DO
400       ! - - - - - - - - - -
401
402       ! - - - - - - - - - -
403    END SUBROUTINE Get_CurvaDistribucionNormalTruncada
404    ! = = = = = = = = = =
405
406
407    ! = = = = = = = = = =
408    FUNCTION IntegCumTrapz(nn, xx, yy) RESULT(yy_acum)
409       ! - - - - - - - - - -
410       ! Integral Cumulative Trapezoidal
411       ! Regla de integracion trapezoidal para una curva discreta
412       !
413       ! nn: numero de puntos de la curva de entrada
414       ! xx: valores de abscisas
415       ! yy: valores de ordenadas
416       ! yy_acum: valores de la integral a medida que se acumula
417       ! - - - - - - - - - -
418       IMPLICIT NONE
419       ! - - - - - - - - - -
420       INTEGER, INTENT(IN) :: nn
421       REAL(8), INTENT(IN) :: xx(nn), yy(nn)
422       ! - - - - - - - - - -
423       REAL(8) :: yy_acum(nn)
424       ! - - - - - - - - - -
425       INTEGER :: ii
426       REAL(8) :: y_medio, delta_x
```

```
427       ! - - - - - - - - - -
428
429       ! - - - - - - - - - -
430       yy_acum(1) = 0.0d0
431       DO ii=2,nn
432          y_medio = 0.5d0 * ( yy(ii-1) + yy(ii) )
433          delta_x = xx(ii) - xx(ii-1)
434          yy_acum(ii) = yy_acum(ii-1) + y_medio*delta_x
435       END DO
436       ! - - - - - - - - - -
437
438       ! - - - - - - - - - -
439    END FUNCTION IntegCumTrapz
440    ! = = = = = = = = = =
441
442
443 END MODULE MOD_Auxiliar
444 ! = = = = = = = = = =
```

```
1  ! = = = = = = = = = =
2  MODULE CLASS_Curve
3  ! = = = = = = = = = =
4
5     ! Modulo para tener variables globales accesibles
6     ! Por ej: curva de deformacion-tension del material
7
8     IMPLICIT NONE
9     PRIVATE
10
11    PUBLIC :: Curva
12
13    TYPE Curva
14       INTEGER :: n
15       REAL(8), ALLOCATABLE :: x(:)
16       REAL(8), ALLOCATABLE :: y(:)
17    CONTAINS
18       PROCEDURE :: iniciar => IniciarCurva
19       PROCEDURE :: asignar => AsignarPunto
20       PROCEDURE :: leer => LeerCurva
21       PROCEDURE :: calcular => CalcularValor
22    END TYPE
23
24 ! = = = = = = = = = =
25 CONTAINS
26 ! = = = = = = = = = =
27
28    ! = = = = = = = = = =
29    SUBROUTINE IniciarCurva(self, nPoints)
30       ! - - - - - - - - - -
31       IMPLICIT NONE
32       CLASS(Curva), INTENT(INOUT) :: self
33       INTEGER, INTENT(IN) :: nPoints
34       ! - - - - - - - - - -
35       self%n = nPoints
36       ALLOCATE( self%x(nPoints) )
37       ALLOCATE( self%y(nPoints) )
38       ! - - - - - - - - - -
39    END SUBROUTINE
40    ! = = = = = = = = = =
41
42    ! = = = = = = = = = =
43    SUBROUTINE AsignarCurva(self, xdata, ydata)
44       ! - - - - - - - - - -
45       IMPLICIT NONE
46       CLASS(Curva), INTENT(INOUT) :: self
47       REAL(8), INTENT(IN) :: xdata(self%n)
48       REAL(8), INTENT(IN) :: ydata(self%n)
49       ! - - - - - - - - - -
50       self%x = xdata
51       self%y = ydata
```

```
52      ! - - - - - - - - - -
53    END SUBROUTINE
54    ! = = = = = = = = = =
55
56    ! = = = = = = = = = =
57    SUBROUTINE AsignarPunto(self, i, xval, yval)
58       ! - - - - - - - - - -
59       IMPLICIT NONE
60       CLASS(Curva), INTENT(INOUT) :: self
61       INTEGER, INTENT(IN) :: i
62       REAL(8), INTENT(IN) :: xval
63       REAL(8), INTENT(IN) :: yval
64       ! - - - - - - - - - -
65       self%x(i) = xval
66       self%y(i) = yval
67       ! - - - - - - - - - -
68    END SUBROUTINE
69    ! = = = = = = = = = =
70
71    ! = = = = = = = = = =
72    SUBROUTINE LeerCurva(self, archivo)
73       ! - - - - - - - - - -
74       IMPLICIT NONE
75       CLASS(Curva), INTENT(INOUT) :: self
76       CHARACTER(LEN=260), INTENT(IN) :: archivo
77       INTEGER :: fid, nn, ii
78       ! - - - - - - - - - -
79       fid = 99
80       OPEN(UNIT=fid, FILE=TRIM(archivo), STATUS='OLD')
81       READ(fid,*) nn
82       CALL IniciarCurva(self,nn)
83       DO ii=1,nn
84          READ(fid,*) self%x(ii), self%y(ii)
85       END DO
86       CLOSE(fid)
87       ! - - - - - - - - - -
88    END SUBROUTINE LeerCurva
89    ! = = = = = = = = = =
90
91
92    ! = = = = = = = = = =
93    FUNCTION CalcularValor(self, xval, extrapolar_IN) RESULT(yval)
94       ! - - - - - - - - - -
95       IMPLICIT NONE
96       ! - - - - - - - - - -
97       CLASS(Curva), INTENT(IN) :: self
98       REAL(8), INTENT(IN) :: xval
99       LOGICAL, OPTIONAL, INTENT(IN) :: extrapolar_IN
100      ! - - - - - - - - - -
101      REAL(8) :: yval
102      ! - - - - - - - - - -
103      LOGICAL :: extrapolar
104      LOGICAL :: fdi1, fdi2 ! logicals "fuera de intervalo"
105      INTEGER :: iSI ! index of sub-interval where x lands in self%x (array of x values of discrete curve)
106      REAL(8) :: slope
107      ! - - - - - - - - - -
108
109      ! - - - - - - - - - -
110      extrapolar = .FALSE.
111      IF ( PRESENT(extrapolar_IN) ) THEN
112         extrapolar = extrapolar_IN
113      END IF
114      ! - - - - - - - - - -
115      fdi1 = ( xval < self%x(1) )
116      fdi2 = ( xval > self%x(self%n) )
117      IF ( (fdi1) .OR. (fdi2) ) THEN
118         IF ( extrapolar) THEN
119            IF (fdi1) THEN
120               yval = self%y(1)
121            ELSE
```

```
122                yval = self%y(self%n)
123             END IF
124          ELSE
125             WRITE(*,*) 'Error, fuera de rango al CalcularValor() de curva'
126             STOP
127          END IF
128       END IF
129       iSI = MINLOC(self%x, DIM=1, MASK=(self%x>=xval)) - 1
130       ! linear interpolation within the sub-interval
131       slope = ( self%y(iSI+1) - self%y(iSI) ) / ( self%x(iSI+1) - self%x(iSI) )
132       yval = self%y(iSI) + slope * (xval - self%x(iSI))
133       ! - - - - - - - - - -
134
135       ! - - - - - - - - - -
136    END FUNCTION CalcularValor
137    ! = = = = = = = = = =
138
139 ! = = = = = = = = = =
140 END MODULE CLASS_Curve
141 ! = = = = = = = = = =
```

B.2. Deposición virtual de fibras

```
1  """
2  Modulo para ensamblar una malla de fibras con intersecciones
3  La malla tiene tres especies: fibras, segmentos y nodos.
4  Los segmentos se componen de dos nodos
5  Las fibras se componen de muchos segmentos (random walk)
6  Cada especie tiene una numeracion global
7  Las fibras tienen una conectividad que son los indices de los segmentos que la componen
8  Los segmentos tienen una conectividad dada por los indices de los dos nodos que lo componen
9  Los nodos tienen coordenadas y tipo (0=continuacion, 1=frontera, 2=interseccion)
10 """
11
12 import numpy as np
13 from matplotlib import pyplot as plt
14 import matplotlib.colors as colors
15 from Aux import iguales, calcular_interseccion_entre_segmentos as calcular_interseccion, find_string_in_file
16 from Aux import calcular_longitud_de_segmento
17 from Aux import calcular_angulo_de_segmento
18
19 class Nodos(object):
20    def __init__(self):
21       self.r = list() # coordenadas de los nodos
22       self.tipos = list() # lista de tipos (0=cont, 1=fron, 2=inter)
23
24    def add_nodo(self, r_nodo, tipo):
25       self.r.append(r_nodo)
26       self.tipos.append(tipo)
27
28    def get_r(self, i_nodo):
29       return self.r[i_nodo]
30
31    def __len__(self):
32       return len(self.r)
33
34
35 class Segmentos(object):
36    def __init__(self):
37       self.con = list() # lista de listas de dos nodos (indices)
38       self.thetas = list()
39       self.longs = list()
40
41    def __len__(self):
42       return len(self.con)
```

```
43
44    def add_segmento(self, seg_con, coors):
45       """
46       aca las coordenadas las necesito para calcularle a cada segmento su longitud y angulo
47       seg_con es la conectividad (2 nodos) del segmento
48       coors son las coordenadas (lista de listas de a dos floats) de todos los nodos
49       (con todos los nodos hasta el momento de crear este segmento esta bien,
50       alcanza con que esten presentes en la lista los dos nodos de seg_con)
51       intersec indica si el segmento ha sido intersectado aun o no"""
52       self.con.append(seg_con)
53       try:
54          longitud, angulo = self.calcular_long_y_theta(seg_con, coors)
55       except ValueError:
56          raise ValueError("Error, segmento de longitud nula!!")
57       self.thetas.append(angulo)
58       self.longs.append(longitud)
59
60    def actualizar_segmento(self, j, coors):
61       """ en caso de que se mueva un nodo y haya que actualizar theta y longitud """
62       long, ang = self.calcular_long_y_theta(self.con[j], coors)
63       self.thetas[j] = ang
64       self.longs[j] = long
65
66    def mover_nodo(self, j, n, coors, new_r):
67       """ mueve un nodo del segmento
68       coors es una lista, es un objeto mutable
69       por lo que al salir de este metodo se va ver modificada
70       es decir, es un puntero
71       j es el indice del segmento a moverle un nodo
72       n es el indice del nodo para el segmento: 0 es inicial, 1 es final """
73       assert n in (0,1)
74       nglobal = self.con[j][n]
75       coors[nglobal] = new_r # se lo modifica resida donde resida (normalmente en un objeto nodos)
76       self.actualizar_segmento(j, coors)
77
78    def cambiar_conectividad(self, j, new_con, coors):
79       """ se modifica la conectividad de un segmento (j) de la lista
80       se le da la nueva conectividad new_con
81       y por lo tanto se vuelve a calcular su angulo y longitud
82       (util para dividir segmentos en 2) """
83       self.con[j] = new_con
84       longitud, angulo = self.calcular_long_y_theta(new_con, coors)
85       self.thetas[j] = angulo
86       self.longs[j] = longitud
87
88    @staticmethod
89    def calcular_long_y_theta(seg, coors):
90       n0 = seg[0]
91       n1 = seg[1]
92       dx = coors[n1][0] - coors[n0][0]
93       dy = coors[n1][1] - coors[n0][1]
94       long = np.sqrt( dx*dx + dy*dy )
95       # ahora theta
96       if iguales(dx,0.0):
97          # segmento vertical
98          if iguales(dy,0.0,1.0e-12):
99             raise ValueError("Error, segmento de longitud nula!!")
100         elif dy>0:
101            theta = np.pi*.5
102         else:
103            theta = 1.5*np.pi
104      elif iguales(dy,0):
105         # segmento horizontal
106         if dx>0:
107            theta = 0.0
108         else:
109            theta = np.pi
110      else:
111         # segmento oblicuo
112         if dx<0:
```

```
113             # segundo o tercer cuadrante
114             theta = np.pi + np.arctan(dy/dx)
115          elif dy>0:
116             # primer cuadrante (dx>0)
117             theta = np.arctan(dy/dx)
118          else:
119             # dx>0 and dy<0
120             # cuarto cuadrante
121             theta = 2.0*np.pi + np.arctan(dy/dx)
122       return long, theta
123
124    def get_right(self, j, coors):
125       n0 = self.con[j][0]
126       n1 = self.con[j][1]
127       x0 = coors[n0][0]
128       x1 = coors[n1][0]
129       return np.maximum(x0,x1)
130
131    def get_left(self, j, coors):
132       n0 = self.con[j][0]
133       n1 = self.con[j][1]
134       x0 = coors[n0][0]
135       x1 = coors[n1][0]
136       return np.minimum(x0,x1)
137
138    def get_top(self, j, coors):
139       n0 = self.con[j][0]
140       n1 = self.con[j][1]
141       y0 = coors[n0][1]
142       y1 = coors[n1][1]
143       return np.maximum(y0,y1)
144
145    def get_bottom(self, j, coors):
146       n0 = self.con[j][0]
147       n1 = self.con[j][1]
148       y0 = coors[n0][1]
149       y1 = coors[n1][1]
150       return np.minimum(y0,y1)
151
152    def get_dx(self, j, coors):
153       n0 = self.con[j][0]
154       n1 = self.con[j][1]
155       x0 = coors[n0][0]
156       x1 = coors[n1][0]
157       return x1-x0
158
159    def get_dy(self, j, coors):
160       n0 = self.con[j][0]
161       n1 = self.con[j][1]
162       y0 = coors[n0][1]
163       y1 = coors[n1][1]
164       return y1-y0
165
166    def get_dx_dy_brtl(self, j, coors):
167       n0 = self.con[j][0]
168       n1 = self.con[j][1]
169       x0 = coors[n0][0]
170       y0 = coors[n0][1]
171       x1 = coors[n1][0]
172       y1 = coors[n1][1]
173       return x1-x0, y1-y0, np.minimum(y0,y1), np.maximum(x0,x1), np.maximum(y0,y1), np.minimum(x0,x1)
174
175
176 class Fibras(object):
177    """ es algo como una lista con algunas funciones particulares
178    tiene un atributo con (conectividad) que es una lista
179    pero la propia instancia se comporta como la lista """
180    def __init__(self):
181       self.con = list() # conectividad: va a ser una lista de listas de segmentos (sus indices nada mas), cada
              ↪ segmento debe ser una lista de 2 nodos
```

```
182         self.dls = list()
183         self.ds = list()
184         self.dthetas = list()
185 
186     def add_fibra(self, fib_con, dl, d, dtheta):
187         self.con.append(fib_con)
188         self.dls.append(dl)
189         self.ds.append(d)
190         self.dthetas.append(dtheta)
191 
192     def insertar_segmento(self, j, k, s):
193         """ inserta un segmento en la conectividad de una fibra
194         j: indice de la fibra
195         k: indice donde se inserta el nuevo segmento
196         s: indice del nuevo segmento para agregar a la conectividad """
197         self.con[j].insert(k,s)
198 
199 class Capas(object):
200     """ es como una lista de fibras que compone cada capa """
201     def __init__(self):
202         self.con = list()
203 
204     def set_capas_listoflists(self, capas_con):
205         self.__init__()
206         for capa_con in capas_con:
207             self.add_capa(capa_con)
208 
209     def add_capa(self, cap_con):
210         self.con.append(cap_con)
211 
212     def add_fibra_a_capa(self, capa, fibra):
213         """ capa y fibra son los indices """
214         self.con[capa].append[fibra]
215 
216 
217 class Malla(object):
218     def __init__(self, L, Dm, volfrac, ls, devangmax, fundisor=None):
219         self.L = L
220         self.Dm = Dm # diametro medio de fibras y espesor de las capas
221         self.volfrac = volfrac # si es float < 1 es volfrac, si es > 1 es num de fibras por capa
222         self.ls = ls
223         self.devangmax = devangmax
224         self.fundisor = fundisor # tiene que ser una funcion (or callable object) que devuelve un valor de orientacion
225         self.caps = Capas() # lista vacia
226         self.fibs = Fibras() # lista vacia
227         self.segs = Segmentos() # lista vacia
228         self.nods = Nodos() # tiene dos listas vacias
229         self.bordes_n = Nodos() # lista de coordenadas con los 4 nodos del borde
230         self.bordes_s = Segmentos() # lista con los segmentos con los 4 nodos del borde
231         self.calcular_marco()
232         self.pregraficado = False
233         self.fig = None
234         self.ax = None
235 
236 
237     def calcular_marco(self):
238         # agrego los 4 nodos
239         self.bordes_n.add_nodo([0., 0.], 1)
240         self.bordes_n.add_nodo([self.L, 0.], 1)
241         self.bordes_n.add_nodo([self.L, self.L], 1)
242         self.bordes_n.add_nodo([0., self.L], 1)
243         # agrego los 4 segmentos
244         self.bordes_s.add_segmento([0,1], self.bordes_n.r)
245         self.bordes_s.add_segmento([1,2], self.bordes_n.r)
246         self.bordes_s.add_segmento([2,3], self.bordes_n.r)
247         self.bordes_s.add_segmento([3,0], self.bordes_n.r)
248 
249     def make_capa(self, dl=None, d=None, dtheta=None, volfraction=None, orient_distr=None):
250         """
251         armo una capa con fibras, todas van a armarse con los
```

```
252         mismos parmetros dl y dtheta (se debe modificar para usar distribuciones)
253         se depositan fibras hasta que se supera la fraccion de volumen dictada
254         """
255         # si volfraction es int estoy dando el numero de fibras!
256         if isinstance(volfraction,int):
257            cond_fin_n = True
258            n_final = volfraction
259         else:
260            cond_fin_n = False
261            volc = self.L*self.L*self.Dm # volumen de la capa
262            vols_final = volfraction*volc # volumen de solido (ocupado por fibras) a alcanzar
263         # chequeo si uso los parametros globales de la malla o si di parametros diferentes para esta capa
264         if dl is None:
265            dl = self.ls
266         if d is None:
267            d = self.Dm
268         if dtheta is None:
269            dtheta = self.devangmax
270         if volfraction is None:
271            volfraction = self.volfrac
272         if orient_distr is None:
273            orient_distr = self.fundisor
274         # --
275         ncapas = len(self.caps.con)
276         capa_con = list()
277         i = 0
278         vols = 0. # volumen de solido actual
279         while True:
280            i += 1
281            j = self.make_fibra(dl, d, dtheta, orient_distr)
282            if j == -1:
283               i -= 1
284            else:
285               volf = self.calcular_volumen_de_una_fibra(j)
286               vols += volf
287               capa_con.append(j)
288            # me fijo si complete la capa
289            if cond_fin_n:
290               if i == n_final:
291                  break
292            else:
293               if vols >= vols_final:
294                  break
295         self.caps.add_capa(capa_con)
296
297      def calcular_loco_de_una_fibra(self, f):
298         """ calcula la longitud de contorno de una fibra """
299         nsegs = len(self.fibs.con[f])
300         loco = 0.
301         for seg in self.fibs.con[f]:
302            n0, n1 = self.segs.con[seg]
303            r0 = self.nods.r[n0]
304            r1 = self.nods.r[n1]
305            lseg = calcular_longitud_de_segmento(r0,r1)
306            loco += lseg
307         return loco
308
309      def calcular_volumen_de_una_fibra(self, f):
310         """ calcula el volumen ocupado por una fibra """
311         loco = self.calcular_loco_de_una_fibra(f)
312         dl = self.fibs.dls[f]
313         d = self.fibs.ds[f]
314         return loco*np.pi*d*d/4.
315
316
317      def calcular_fraccion_de_volumen_de_una_capa(self, capcon):
318         """ calcula la fraccion de volumen de una capa como
319         el volumen ocupado por fibras
320         dividido el volumen total de la capa """
321         volfs = 0
```

```
322         for f in capcon: # recorro las fibras de la capa
323             volf = self.calcular_volumen_de_una_fibra(f)
324             volfs += volf
325         # el volumen total de la capa es:
326         volc = self.L*self.L*self.Dm
327         # luego la fraccion de volumen
328         fracvol = volfs / volc
329         return fracvol
330 
331     def calcular_orientacion_de_una_fibra(self, f):
332         """ calcula la orientacion de una fibra como el promedio
333         de las orientacions de sus segmentos
334         cada orientacion es un angulo en [0,pi) """
335         fcon = self.fibs.con[f]
336         nsegs = len(fcon)
337         theta_f = 0.
338         for s in fcon:
339             # s es un indice de segmento
340             theta_s = self.segs.thetas[s]
341             # theta_s esta en [0,pi)
342             if theta_s>=np.pi:
343                 theta_s = theta_s-np.pi
344             if theta_s < 0:
345                 pass
346             # ahora voy haciendo el promedio
347             theta_f += theta_s
348         theta_f = theta_f / float(nsegs)
349         return theta_f
350 
351     def calcular_orientacion_extremo_extremo_de_una_fibra(self, f):
352         fcon = self.fibs.con[f]
353         s0 = fcon[0]
354         s1 = fcon[1]
355         n0 = self.segs.con[s0][0]
356         n1 = self.segs.con[s1][1]
357         r0 = self.nods.r[n0]
358         r1 = self.nods.r[n1]
359         theta_2pi = calcular_angulo_de_segmento(r0,r1)
360         if theta_2pi>=np.pi:
361             theta = theta_2pi - np.pi
362         else:
363             theta = theta_2pi
364         if theta_2pi < 0.:
365             raise ValueError
366         return theta
367 
368     def make_fibra(self, dl, d, dtheta, orient_distr=None):
369         """ tengo que armar una lista de segmentos
370         nota: todos los indices (de nodos, segmentos y fibras)
371         son globales en la malla, cada nodo nuevo tiene un indice +1 del anterior
372         idem para segmentos y fibras
373         los indices de los nodos, de los segmentos y de las fibras van por separado
374         es decir que hay un nodo 1, un segmento 1 y una fibra 1
375         pero no hay dos de misma especie que compartan indice """
376         # ---
377         # primero hago un segmento solo
378         # para eso pongo un punto sobre la frontera del rve y el otro lo armo con un desplazamiento recto
379         # tomo un angulo random entre 0 y pi, saliente del borde hacia adentro del rve
380         # eso me da un nuevo segmento
381         # agrego todas las conectividades
382         # ---
383         # Voy a ir guardando en una lista las coordenadas de los nodos
384         coors = list()
385         # Armo el primer segmento
386         # primero busco un nodo en el contorno
387         x0, y0, b0 = self.get_punto_sobre_frontera()
388         if orient_distr is None:
389             theta_abs = np.random.rand() * np.pi
390         else:
391             theta_abs = orient_distr()
```

```
392         # theta_abs = 179. * np.pi/180.
393         if theta_abs == np.pi:
394            theta_abs = 0.
395         elif theta_abs > np.pi:
396            raise ValueError("theta_abs de una fibra no comprendido en [0,pi)")
397         # veo el cuadrante
398         if theta_abs < np.pi*1.0e-8:
399            cuad = -1 # direccion horizontal
400         elif np.abs(theta_abs-np.pi*0.5) < 1.0e-8:
401            cuad = -2 # direccion vertical
402         elif theta_abs < np.pi*0.5:
403            cuad = 1 # primer cuadrante
404         else:
405            cuad = 2 # segundo cuadrante
406         # ahora me fijo la relacion entre cuadrante y borde
407         if cuad == -1: # fibra horizontal
408            if b0 in (0,2):
409               return -1 # esta fibra no vale, es horizontal sobre un borde horizontal
410            elif b0 == 1:
411               theta = np.pi
412            else: #b0 == 3
413               theta = 0.
414         elif cuad == -2: # fibra vertical
415            if b0 in (1,3):
416               return -1
417            elif b0 == 0:
418               theta = 0.5*np.pi
419            else: # b0 == 2
420               theta = 1.5*np.pi
421         elif cuad == 1: # primer cuadrante
422            if b0 in (0,3):
423               theta = theta_abs
424            else: #b0 in(1,2)
425               theta = theta_abs + np.pi
426         else: # cuad == 2 segundo cuadrante
427            if b0 in (0,1):
428               theta = theta_abs
429            else: # b0 in (2,3)
430               theta = theta_abs + np.pi
431         # ya tengo el angulo del segmento
432         dx = dl * np.cos(theta)
433         dy = dl * np.sin(theta)
434         coors.append( [x0,y0] )
435         coors.append( [x0+dx, y0+dy] )
436         # ahora agrego nuevos nodos en un bucle
437         # cada iteracion corresponde a depositar un nuevo segmento
438         n = 1
439         while True:
440            # si el nodo anterior ha caido fuera del rve ya esta la fibra
441            if self.check_fuera_del_RVE(coors[-1]):
442               break
443            n += 1
444            # de lo contrario armo un nuevo segmento a partir del ultimo nodo
445            # el angulo puede sufrir variacion
446            theta = theta + dtheta * (2.0*np.random.rand() - 1.0)
447            # desplazamiento:
448            dx = dl * np.cos(theta)
449            dy = dl * np.sin(theta)
450            # nuevo nodo
451            x = coors[-1][0] + dx
452            y = coors[-1][1] + dy
453            coors.append( [x,y] )
454         # -
455         # Aqui termine de obtener las coordenadas de los nodos que componen la fibra
456         # si la fibra es muy corta la voy a descartar
457         # para eso calculo su longitud de contorno
458         loco = dl*float(len(coors)-1) # esto es aproximado porque el ultimo segmento se recorta
459         if loco < 0.3*self.L:
460            return -1
461         # Voy a ensamblar la fibra como concatenacion de segmentos, que a su vez son concatenacion de dos nodos
```

```
462         f_con = list()
463         # agrego el primer nodo a la conectividad de nodos
464         self.nods.add_nodo(coors[0], 1)
465         for coor in coors[1:]: # reocrro los nodos desde el nodo 1 (segundo nodo)
466            self.nods.add_nodo(coor, 0)
467            nnods = len(self.nods)
468            s0 = [nnods-2, nnods-1]
469            self.segs.add_segmento(s0, self.nods.r)
470            nsegs = len(self.segs)
471            f_con.append(nsegs-1)
472         # al final recorto la fibra y la almaceno
473         self.nods.tipos[-1] = 1
474         self.trim_fibra_at_frontera(f_con) # lo comento porque a veces quedan segmentos super pequenos
475         self.fibs.add_fibra(f_con, dl, d, dtheta)
476         return len(self.fibs.con) - 1 # devuelvo el indice de la fibra
477
478
479      def get_punto_sobre_frontera(self):
480         boundary = np.random.randint(4)
481         d = np.random.rand() * self.L
482         if boundary==0:
483            x = d
484            y = 0.0
485         elif boundary==1:
486            x = self.L
487            y = d
488         elif boundary==2:
489            x = self.L - d
490            y = self.L
491         elif boundary==3:
492            x = 0.0
493            y = self.L - d
494         return x, y, float(boundary)
495
496      def check_fuera_del_RVE(self, r):
497         x = r[0]
498         y = r[1]
499         if x<=0 or x>=self.L or y<=0 or y>=self.L:
500            return True
501         else:
502            return False
503
504      def trim_fibra_at_frontera(self, fib_con):
505         """ subrutina para cortar la fibra que ha salido del rve """
506         # debo cortar la ultima fibra en su interseccion por el rve
507         # para eso calculo las intersecciones de los nodos con los bordes
508         # coordenadas del ultimo segmento de la fibra de conectividad fib_con
509         s = fib_con[-1]
510         rs0 = self.nods.r[ self.segs.con[s][0] ] # coordenadas xy del nodo 0 del segmento s
511         rs1 = self.nods.r[ self.segs.con[s][1] ] # coordenadas xy del nodo 1 del segmento s
512         # pruebo con cada borde
513         for b in range( len(self.bordes_s.con) ): # recorro los 4 bordes
514            # puntos del borde en cuestion
515            rb0 = self.bordes_n.r[ self.bordes_s.con[b][0] ] # coordenadas xy del nodo 0 del borde b
516            rb1 = self.bordes_n.r[ self.bordes_s.con[b][1] ] # coordenadas xy del nodo 1 del borde b
517            interseccion = calcular_interseccion(rs0, rs1, rb0, rb1)
518            if interseccion is None: # no hubo interseccion
519               continue # con este borde no hay interseccion, paso al que sigue
520            else: # hubo interseccion
521               in_r, in_tipo, in_seg0, in_seg1 = interseccion
522               if in_tipo==2: # interseccion en el medio
523                  try: # tengo que mover el ultimo nodo y por lo tanto cambia el segmento
524                     self.segs.mover_nodo(s, 1, self.nods.r, in_r)
525                     rs1 = in_r
526                  except ValueError as e:
527                     print "error"
528                     print fib_con, b, interseccion
529                     quit()
530               else: # interseccion coincide con uno o dos extremos
531                  # en este caso solo me importa el segundo nodo del primer segmento (seg 0)
```

```
532                 # porque el segmento 1 es el borde, y el primer nodo del seg 0 siempre deberia estar dentro del rve
533                 # (o en el borde a lo sumo si se trata de una fibra de un solo segmento)
534                 # y en ese caso no hay nada que hacer! puesto que el nodo ya esta en el borde
535                 pass
536
537     def cambiar_capas(self, new_ncapas):
538         """ un mapeo de las fibras en un numero de capas diferente """
539         # me fijo cuantas fibras van a entrar en cada capa
540         nfibras = len(self.fibs.con)
541         nf_x_capa = int(nfibras / new_ncapas)
542         # armo una nueva conectividad de capas
543         capas_con = list()
544         for c in range(new_ncapas-1): # -1 porque la ultima capa la hare aparte
545             print c
546             capa_con = range(c*nf_x_capa, (c+1)*nf_x_capa)
547             capas_con.append(capa_con)
548         # la ultima capa puede tener alguna fibra de mas
549         c = new_ncapas - 1
550         capa_con = range(c*nf_x_capa, (c+1)*nf_x_capa)
551         capas_con.append(capa_con)
552         self.caps.set_capas_listoflists(capas_con)
553
554     def calcular_conectividad_de_interfibras(self):
555         """ ojo son diferentes a las subfibras de una malla simplificada
556         aqui las interfibras son concatenaciones de segmentos entre nodos interseccion
557         en una ms las subfibras son una simplificacion de una fibra dando solamente los nodos extremos y enrulamiento
                ↪ """
558         infbs_con = list() # conectividad: lista de listas de segmentos
559         for f, fcon in enumerate(self.fibs.con): # recorro las fibras
560             # cada fibra que empieza implica una nueva interfibra
561             infb = list() # conectividad de la interfibra: lista de segmentos
562             # tengo que ir agregando segmentos hasta toparme con un nodo interseccion o frontera
563             for s in fcon: # recorro los segmentos de la fibra f
564                 scon = self.segs.con[s]
565                 n0, n1 = scon
566                 # agrego el segmento s a la interfibra
567                 infb.append(s)
568                 # si el ultimo nodo de s es interseccion o frontera aqui termina la interfibra
569                 if self.nods.tipos[n1] in (1,2):
570                     infbs_con.append(infb) # agrego la interfibra a la conectividad
571                     infb = list() # preparo una nueva interfibra vacia para continuar agregando segmentos
572         # aqui ya deberia tener una conectividad terminada
573         return infbs_con
574
575     def calcular_orientaciones(self):
576         """ calcular las orientaciones de las fibras de toda la malla """
577         thetas_f = list()
578         for f, fcon in enumerate(self.fibs.con):
579             # theta_f = self.calcular_orientacion_de_una_fibra(f)
580             theta_f = self.calcular_orientacion_extremo_extremo_de_una_fibra(f)
581             thetas_f.append(theta_f)
582         return thetas_f
583
584     def calcular_distribucion_de_orientaciones(self, rec_orientaciones=None, n=10):
585         """ calcula la distribucion de orientaciones en la malla
586         contando las frecuencias en los bins """
587         # obtengo las orientaciones de todas las fibras
588         if rec_orientaciones is None:
589             phis = self.calcular_orientaciones()
590         else:
591             phis = rec_orientaciones
592         phis = np.array(phis, dtype=float)
593         #
594         conteo, x_edges = np.histogram(phis, bins=n, range=(0., np.pi))
595         delta = (np.pi - 0.) / float(n)
596         # pdf = conteo / float(np.sum(conteo)) / delta
597         x = x_edges[:-1] + 0.5*delta
598         return x, delta, conteo
599
600     def calcular_enrulamientos(self):
```

```
601         """ calcular para todas las fibras sus longitudes de contorno y
602         sus longitudes extremo a extremos (loco y lete)
603         y calcula el enrulamiento como lamr=loco/lete """
604         lamsr = []
605         for fcon in self.fibs.con: # recorro las fibras del rve
606             loco = 0.
607             for s in fcon: # recorro los segmentos de cada fibra
608                 scon = self.segs.con[s]
609                 n0, n1 = scon
610                 r0 = self.nods.r[n0]
611                 r1 = self.nods.r[n1]
612                 try:
613                     loco += calcular_longitud_de_segmento(r0, r1)
614                 except ValueError:
615                     raise ValueError("Error, segmento de longitud nula!!")
616             n_ini = self.segs.con[fcon[0]][0]
617             n_fin = self.segs.con[fcon[-1]][1]
618             r_ini = self.nods.r[n_ini]
619             r_fin = self.nods.r[n_fin]
620             try:
621                 lete = calcular_longitud_de_segmento(r_ini, r_fin)
622             except ValueError:
623                 raise ValueError("Error, lete de longitud nula!!")
624             lamsr.append( loco/lete )
625         return lamsr
626 
627     def calcular_distribucion_de_enrulamiento(self, rec_lamsr=None, lamr_min=None, lamr_max=None, n=10, binwidth=None):
628         """ calcular la distribucion de enrulamientos (pdf)
629         para eso calculo el histograma y luego normalizo con el area
630         parametros de entrada
631         rec_lamsr: valores de lamsr para todas las fibras (por si ya lo tengo asi no lo calculo al dope de nuevo)
632         lamr_min: valor minimo de lamr para hacer el histograma (ojo pueden quedar fibras afuera)
633         lamr_max: valor maximo de lamr para hacer el histograma (ojo pueden quedar fibras afuera)
634         n: numero de bins
635         parametros de salida: x, delta, pdf
636         x: valores medios de cada bin
637         delta: ancho de cada bin (son todos iguales)
638         pdf: valor de pdf
639         """
640         if rec_lamsr is None:
641             lamsr = self.calcular_enrulamientos()
642         else:
643             lamsr = rec_lamsr
644         lamsr = np.array(lamsr, dtype=float)
645         if lamr_min is None:
646             lamr_min = np.min(lamsr)
647         if lamr_max is None:
648             lamr_max = np.max(lamsr)
649         # me fijo si hay imposicion de ancho de bin, entonces calculo con eso el numero de bins
650         if binwidth is not None:
651             n = int((lamr_max - lamr_min) / binwidth + 0.5) # es el entero mas cercano
652         conteo, x_edges = np.histogram(lamsr, bins=n, range=(lamr_min, lamr_max))
653         delta = (lamr_max - lamr_min) / n
654         # pdf = conteo / float(np.sum(conteo)) / delta
655         x = x_edges[:-1] + 0.5*delta
656         return x, delta, conteo
657 
658     def get_histograma_lamr(self, lamr_min=None, lamr_max=None, nbins=5, binwidth=None, opcion="fibras"):
659         if opcion=="fibras":
660             lamsr = self.calcular_enrulamientos()
661         elif opcion=="interfibras":
662             lamsr = self.calcular_enrulamientos_de_interfibras()
663         else:
664             raise ValueError
665         #
666         lrs, dlr, conteo = self.calcular_distribucion_de_enrulamiento(rec_lamsr=lamsr, lamr_min=lamr_min, lamr_max=
                ↪ lamr_max, n=nbins, binwidth=binwidth)
667         frecs = np.array(conteo, dtype=float) / float(np.sum(conteo))
668         pdf = frecs / dlr
669         return lrs, dlr, conteo, frecs, pdf
```

```
670
671     def get_histograma_orientaciones(self, nbins=5, opcion="fibras"):
672         if opcion=="fibras":
673             rec_thetas = self.calcular_orientaciones() # todos los angulos de las fibras
674             # thetas a continuacion son los angulos medio de cada bin
675         elif opcion=="interfibras":
676             raise NotImplementedError
677         else:
678             raise ValueError
679         #
680         thetas, dth, conteo = self.calcular_distribucion_de_orientaciones(rec_orientaciones=rec_thetas, n=nbins)
681         frecs = np.array(conteo, dtype=float) / float(np.sum(conteo))
682         pdf = frecs / dth
683         return thetas, dth, conteo, frecs, pdf
684
685     def calcular_enrulamientos_de_interfibras(self):
686         """ calcular para todas las interfibras sus longitudes de contorno y
687         sus longitudes extremo a extremos (loco y lete)
688         y calcula el enrulamiento como lamr=loco/lete """
689         lamsr = []
690         infbs_con = self.calcular_conectividad_de_interfibras()
691         for infb_con in infbs_con: # recorro las interfibras (fibras interectadas) del rve
692             loco = 0.
693             for s in infb_con: # recorro los segmentos de cada interfibra
694                 scon = self.segs.con[s]
695                 n0, n1 = scon
696                 r0 = self.nods.r[n0]
697                 r1 = self.nods.r[n1]
698                 loco += calcular_longitud_de_segmento(r0, r1)
699             n_ini = self.segs.con[infb_con[0]][0]
700             n_fin = self.segs.con[infb_con[-1]][1]
701             r_ini = self.nods.r[n_ini]
702             r_fin = self.nods.r[n_fin]
703             lete = calcular_longitud_de_segmento(r_ini, r_fin)
704             lamsr.append( loco/lete )
705         return lamsr
706
707     def calcular_distribucion_de_enrulamiento_de_interfibras(self, rec_lamsr=None, lamr_min=None, lamr_max=None, n=10):
708         """ calcular la distribucion de enrulamientos (pdf)
709         para eso calculo el histograma y luego normalizo con el area
710         parametros de entrada
711         rec_lamsr: valores de lamsr para todas las fibras (por si ya lo tengo asi no lo calculo al dope de nuevo)
712         lamr_min: valor minimo de lamr para hacer el histograma (ojo pueden quedar fibras afuera)
713         lamr_max: valor maximo de lamr para hacer el histograma (ojo pueden quedar fibras afuera)
714         n: numero de bins
715         parametros de salida: x, delta, pdf
716         x: valores medios de cada bin
717         delta: ancho de cada bin (son todos iguales)
718         pdf: valor de pdf
719         """
720         if rec_lamsr is None:
721             lamsr = self.calcular_enrulamientos_de_interfibras()
722         else:
723             lamsr = rec_lamsr
724         lamsr = np.array(lamsr, dtype=float)
725         if lamr_min is None:
726             lamr_min = np.min(lamsr)
727         if lamr_max is None:
728             lamr_max = np.max(lamsr)
729         delta = (lamr_max - lamr_min) / n
730         conteo, x_edges = np.histogram(lamsr, bins=n, range=(lamr_min, lamr_max))
731         delta = (lamr_max - lamr_min) / n
732         pdf = conteo / float(np.sum(conteo)) / delta
733         x = x_edges[:-1] + 0.5*delta
734         return x, delta, pdf
735
736     def guardar_en_archivo(self, archivo="Malla.txt"):
737         fid = open(archivo, "w")
738         # ---
739         # primero escribo L, dl y dtheta
```

```
740         fid.write("*Parametros␣(L,␣Dm,␣volfrac,␣ls,␣devangmax)␣\n")
741         fid.write("{:20.8f}\n".format(self.L))
742         fid.write("{:20.8f}\n".format(self.Dm))
743         fid.write("{:20.8f}\n".format(self.volfrac))
744         fid.write("{:20.8f}\n".format(self.ls))
745         fid.write("{:20.8f}\n".format(self.devangmax*180./np.pi)) # lo escribo en grados
746         # ---
747         # escribo los nodos: indice, tipo, y coordenadas
748         dString = "*Coordenadas␣\n" + str(len(self.nods.r)) + "\n"
749         fid.write(dString)
750         for n in range( len(self.nods.r) ):
751            dString = "{:12d}".format(n)
752            dString += "{:2d}".format(self.nods.tipos[n])
753            dString += "".join( "{:+17.8e}".format(val) for val in self.nods.r[n] ) + "\n"
754            fid.write(dString)
755         # ---
756         # sigo con los segmentos: indice, nodo inicial y nodo final
757         dString = "*Segmentos␣\n" + str( len(self.segs.con) ) + "\n"
758         fid.write(dString)
759         for s in range( len(self.segs.con) ):
760            n0, n1 = self.segs.con[s] # conectividad del segmento (nodo inicial n0 y nodo final n1)
761            fmt = "{:12d}"*3
762            dString = fmt.format(s, n0, n1) +"\n"
763            fid.write(dString)
764         # ---
765         # sigo con las fibras: indice, dl, d, dtheta, nsegs_f, y segmentos (conectividad)
766         dString = "*Fibras␣\n" + str( len(self.fibs.con) ) + "\n"
767         fid.write(dString)
768         for f, fcon in enumerate(self.fibs.con):
769            dString = "{:12d}".format(f) # indice
770            dString += "{:17.8e}{:17.8e}{:17.8e}".format(self.fibs.dls[f], self.fibs.ds[f], self.fibs.dthetas[f]) # dl, d
                  ↪ y dtheta
771            dString += "{:12d}".format(len(fcon)) # indice
772            dString += "".join( "{:12d}".format(val) for val in fcon ) + "\n" # conectividad
773            fid.write(dString)
774         # termino con las capas: indice y fibras (conectividad):
775         dString = "*Capas␣\n" + str( len(self.caps.con) ) + "\n"
776         fid.write(dString)
777         for c, ccon in enumerate(self.caps.con):
778            dString = "{:12d}".format(c) # indice
779            dString += "{:12d}".format(len(ccon)) # indice
780            dString += "".join( "{:12d}".format(val) for val in ccon ) + "\n" # conectividad
781            fid.write(dString)
782         # ---
783         # termine
784         fid.close()
785
786      @classmethod
787      def leer_de_archivo(cls, archivo="Malla.txt"):
788         fid = open(archivo, "r")
789         # primero leo los parametros
790         target = "*parametros"
791         ierr = find_string_in_file(fid, target, True)
792         L = float( fid.next() )
793         Dm = float( fid.next() )
794         volfrac = float( fid.next() )
795         if volfrac > 1.:
796            volfrac = int(volfrac +.5)
797         ls = float( fid.next() )
798         devangmax = float( fid.next() ) # en grados
799         devangmax = devangmax * np.pi / 180.
800         # luego busco coordenadas
801         target = "*coordenadas"
802         ierr = find_string_in_file(fid, target, True)
803         num_r = int(fid.next())
804         coors = list()
805         tipos = list()
806         for i in range(num_r):
807            j, t, x, y = (float(val) for val in fid.next().split())
808            tipos.append(int(t))
```

```
809             coors.append([x,y])
810         # luego los segmentos
811         target = "*segmentos"
812         ierr = find_string_in_file(fid, target, True)
813         num_s = int(fid.next())
814         segs = list()
815         for i in range(num_s):
816             j, n0, n1 = (int(val) for val in fid.next().split())
817             segs.append([n0,n1])
818         # luego las fibras
819         target = "*fibras"
820         ierr = find_string_in_file(fid, target, True)
821         num_f = int(fid.next())
822         fibs = list()
823         dls = list()
824         ds = list()
825         dthetas = list()
826         for i in range(num_f):
827             svals = fid.next().split()
828             j = int(svals[0])
829             dl = float(svals[1])
830             d = float(svals[2])
831             dtheta = float(svals[3])
832             nsegsf = int(svals[4])
833             fcon = [int(val) for val in svals[5:]]
834             fibs.append(fcon)
835             dls.append(dl)
836             ds.append(d)
837             dthetas.append(dtheta)
838         # luego la capas
839         target = "*capas"
840         ierr = find_string_in_file(fid, target, True)
841         num_c = int(fid.next())
842         caps = list()
843         for c in range(num_c):
844             svals = fid.next().split()
845             j = int(svals[0])
846             nfibsc = int(svals[1])
847             ccon = [int(val) for val in svals[2:]]
848             caps.append(ccon)
849         # ahora que tengo todo armo el objeto
850         malla = cls(L, Dm, volfrac, ls, devangmax)
851         # le asigno los nodos
852         for i in range(num_r):
853             malla.nods.add_nodo(coors[i], tipos[i])
854         # le asigno los segmentos
855         for i in range(num_s):
856             s_con = segs[i]
857             try:
858                 malla.segs.add_segmento(s_con, coors)
859             except ValueError:
860                 print "Segmento de long nula"
861                 # raise ValueError("Error, segmento de longitud nula!!")
862         # le asigno las fibras
863         for i in range(num_f):
864             f_con = fibs[i]
865             dl = dls[i]
866             d = ds[i]
867             dtheta = dthetas[i]
868             malla.fibs.add_fibra(f_con, dl, d, dtheta)
869         # le asigno las capas
870         for c in range(num_c):
871             c_con = caps[c]
872             malla.caps.add_capa(c_con)
873         # listo
874         return malla
875 
876     def pre_graficar_bordes(self, fig, ax, byn=False):
877         # seteo
878         margen = 0.1*self.L
```

```
879         ax.set_xlim(left=0-margen, right=self.L+margen)
880         ax.set_ylim(bottom=0-margen, top=self.L+margen)
881         # dibujo los bordes del rve
882         fron = []
883         fron.append( [[0,self.L], [0,0]] )
884         fron.append( [[0,0], [self.L,0]] )
885         fron.append( [[0,self.L], [self.L,self.L]] )
886         fron.append( [[self.L,self.L], [self.L,0]] )
887         plt_fron0 = ax.plot(fron[0][0], fron[0][1], linestyle=":", c="gray")
888         plt_fron1 = ax.plot(fron[1][0], fron[1][1], linestyle=":", c="gray")
889         plt_fron2 = ax.plot(fron[2][0], fron[2][1], linestyle=":", c="gray")
890         plt_fron3 = ax.plot(fron[3][0], fron[3][1], linestyle=":", c="gray")
891
892
893     def pre_graficar_capas(self, fig, ax, byn=True):
894         nc = len(self.caps.con)
895         if byn:
896             mi_colormap = plt.cm.gray
897         else:
898             mi_colormap = plt.cm.rainbow
899         sm = plt.cm.ScalarMappable(cmap=mi_colormap, norm=plt.Normalize(vmin=0, vmax=nc-1))
900         # dibujo las fibras (los segmentos)
901         # preparo las listas, una lista para cada fibra
902         xx = [ list() for f in self.fibs.con ]
903         yy = [ list() for f in self.fibs.con ]
904         grafs = list()
905         for c, c_con in enumerate(self.caps.con): # recorro las capas
906             for f in c_con: # recorro las fibra de la capa
907                 f_con = self.fibs.con[f]
908                 # antes de recorrer los segmentos de cada fibra
909                 # el primer nodo del primer segmento lo agrego antes del bucle
910                 s = f_con[0] # primer segmento de la fibra f
911                 n = self.segs.con[s][0] # primer nodo del segmento s
912                 r = self.nods.r[n] # coordenadas de ese nodo
913                 xx[f].append(r[0])
914                 yy[f].append(r[1])
915                 for s in f_con: # recorro los segmentos de la fibra f
916                     s_con = self.segs.con[s]
917                     n = s_con[1] # ultimo nodo del segmento s
918                     r = self.nods.r[n] # coordenadas de ese nodo
919                     xx[f].append(r[0])
920                     yy[f].append(r[1])
921                 grafs.append( ax.plot(xx[f], yy[f], linestyle="-", marker="", label=str(f), color=sm.to_rgba(nc-1-c) ) )
922         sm._A = []
923         fig.colorbar(sm)
924
925     @staticmethod
926     def truncate_colormap(cmap, minval=0.0, maxval=1.0, n=100):
927         new_cmap = colors.LinearSegmentedColormap.from_list(
928             'trunc({n},{a:.2f},{b:.2f})'.format(n=cmap.name, a=minval, b=maxval),
929             cmap(np.linspace(minval, maxval, n)))
930         return new_cmap
931
932     def pre_graficar_fibras(self, fig, ax, ncapas=None, lamr_min=None, lamr_max=None, byn=False, barracolor=True,
        ↪ color_por="nada", linewidth=2, colores_cm=None, ncolores_cm=20):
933         # preparo un mapa de colores mapeable por escalar
934         lamsr = self.calcular_enrulamientos()
935         if byn:
936             mi_colormap = plt.cm.gray_r
937             # lo trunco para que no incluya el cero (blanco puro que no hace contraste con el fondo)
938             mi_colormap = self.truncate_colormap(mi_colormap, 0.1, 0.5)
939         else:
940             mi_colormap = plt.cm.jet
941             if colores_cm is not None:
942                 mi_colormap = colors.LinearSegmentedColormap.from_list("mi_colormap", colores_cm, N=ncolores_cm)
943         if color_por == "lamr":
944             if lamr_min is None:
945                 lamr_min = np.min(lamsr)
946             if lamr_max is None:
947                 lamr_max = np.max(lamsr)
```

```
948             sm = plt.cm.ScalarMappable(cmap=mi_colormap, norm=plt.Normalize(vmin=lamr_min, vmax=lamr_max))
949         elif color_por == "fibra":
950             sm = plt.cm.ScalarMappable(cmap=mi_colormap, norm=plt.Normalize(vmin=0, vmax=len(self.fibs.con)-1))
951         elif color_por == "capa":
952             sm = plt.cm.ScalarMappable(cmap=mi_colormap, norm=plt.Normalize(vmin=0, vmax=len(self.caps.con)-1))
953         elif color_por == "angulo":
954             sm = plt.cm.ScalarMappable(cmap=mi_colormap, norm=plt.Normalize(vmin=0, vmax=np.pi))
955         # dibujo las fibras (los segmentos)
956         # preparo las listas, una lista para cada fibra
957         xx = [ list() for f in self.fibs.con ]
958         yy = [ list() for f in self.fibs.con ]
959         grafs = list()
960         if ncapas is None:
961             caps_con = self.caps.con
962         else:
963             caps_con = self.caps.con[:ncapas]
964         for c, c_con in enumerate(caps_con): # recorro las capas
965             for f in c_con: # recorro las fibra de la capa
966                 f_con = self.fibs.con[f]
967                 # antes de recorrer los segmentos de cada fibra
968                 # el primer nodo del primer segmento lo agrego antes del bucle
969                 s = f_con[0] # primer segmento de la fibra f
970                 n = self.segs.con[s][0] # primer nodo del segmento s
971                 r = self.nods.r[n] # coordenadas de ese nodo
972                 xx[f].append(r[0])
973                 yy[f].append(r[1])
974                 for s in f_con: # recorro los segmentos de la fibra f
975                     s_con = self.segs.con[s]
976                     n = s_con[1] # ultimo nodo del segmento s
977                     r = self.nods.r[n] # coordenadas de ese nodo
978                     xx[f].append(r[0])
979                     yy[f].append(r[1])
980                 if color_por == "lamr":
981                     col = sm.to_rgba(lamsr[f])
982                 elif color_por =="fibra":
983                     col = sm.to_rgba(f)
984                 elif color_por == "capa":
985                     col = sm.to_rgba(c)
986                 elif color_por == "angulo":
987                     theta = self.calcular_orientacion_extremo_extremo_de_una_fibra(f)
988                     col = sm.to_rgba(theta)
989                 elif color_por == "nada":
990                     col = "k"
991                 grafs.append( ax.plot(xx[f], yy[f], linestyle="-", marker="", label=str(f), color=col, linewidth=linewidth
                        ↪ ) )
992         if barracolor and color_por not in ("nada", "fibra"):
993             sm._A = []
994             cbar = fig.colorbar(sm)
995             if color_por == "capa":
996                 cbar.set_ticks(range(len(self.caps.con)))
997
998
999     def pre_graficar_interfibras(self, fig, ax, lamr_min=None, lamr_max=None, byn=False, barracolor=True, color_por="
            ↪ nada", colormap="jet", colores_cm=None, ncolores_cm=20):
1000         # preparo un mapa de colores mapeable por escalar
1001         infbs_con = self.calcular_conectividad_de_interfibras()
1002         lamsr = self.calcular_enrulamientos_de_interfibras()
1003         if byn:
1004             mi_colormap = plt.cm.gray_r
1005             # lo trunco para que no incluya el cero (blanco puro que no hace contraste con el fondo)
1006             mi_colormap = self.truncate_colormap(mi_colormap, 0.4, 1.0)
1007         else:
1008             if colores_cm is not None:
1009                 mi_colormap = colors.LinearSegmentedColormap.from_list("mi_colormap", colores_cm, N=ncolores_cm)
1010             elif colormap == "jet":
1011                 mi_colormap = plt.cm.jet
1012             elif colormap == "prism":
1013                 mi_colormap = plt.cm.prism
1014             elif colormap == "Dark2":
1015                 mi_colormap = plt.cm.Dark2
```

```
1016        if color_por == "lamr":
1017           if lamr_min is None:
1018              lamr_min = np.min(lamsr)
1019           if lamr_max is None:
1020              lamr_max = np.max(lamsr)
1021           sm = plt.cm.ScalarMappable(cmap=mi_colormap, norm=plt.Normalize(vmin=lamr_min, vmax=lamr_max))
1022        elif color_por == "interfibra":
1023           sm = plt.cm.ScalarMappable(cmap=mi_colormap, norm=plt.Normalize(vmin=0, vmax=len(infbs_con)-1))
1024        elif color_por == "nada":
1025           sm = plt.cm.ScalarMappable(cmap=mi_colormap, norm=plt.Normalize(vmin=0, vmax=len(infbs_con)-1))
1026        # dibujo las fibras (los segmentos)
1027        # preparo las listas, una lista para cada fibra
1028        xx = [ list() for infb_con in infbs_con ]
1029        yy = [ list() for infb_con in infbs_con ]
1030        grafs = list()
1031        print "-"
1032        print "graficando␣interfibras"
1033        pcc = 0.
1034        nif = float(len(infbs_con))
1035        pctp = np.arange(0., 101., 10.).tolist()
1036        pcpd = np.zeros(len(pctp), dtype=bool).tolist()
1037        for i, infb_con in enumerate(infbs_con): # recorro las interfibras
1038           pc = round(float(i)/nif * 100.,0)
1039           if pc in pctp:
1040              ipc = pctp.index(pc)
1041              if not pcpd[ipc]:
1042                 print "{:4.0f}%␣".format(pc),
1043                 pcpd[ipc] = True
1044           # antes de recorrer los segmentos de cada interfibra
1045           # el primer nodo del primer segmento lo agrego antes del bucle
1046           s = infb_con[0] # primer segmento de la interfibra i
1047           n = self.segs.con[s][0] # primer nodo del segmento s
1048           r = self.nods.r[n] # coordenadas de ese nodo
1049           xx[i].append(r[0])
1050           yy[i].append(r[1])
1051           for s in infb_con: # recorro los segmentos de la interfibra i
1052              s_con = self.segs.con[s]
1053              n = s_con[1] # ultimo nodo del segmento s
1054              r = self.nods.r[n] # coordenadas de ese nodo
1055              xx[i].append(r[0])
1056              yy[i].append(r[1])
1057           if color_por == "lamr":
1058              col = sm.to_rgba(lamsr[i])
1059           elif color_por =="interfibra":
1060              col = sm.to_rgba(i)
1061           elif color_por == "nada":
1062              col = "k"
1063           grafs.append( ax.plot(xx[i], yy[i], linestyle="-", marker="", label=str(i), color=col) )
1064        print "-"
1065        if not byn and barracolor:
1066           sm._A = []
1067           fig.colorbar(sm)
1068
1069     def pre_graficar_nodos_frontera(self, fig, ax, markersize=8):
1070        # dibujo las fibras (los segmentos)
1071        # preparo las listas, una lista para cada fibra
1072        xx = [ list() for f in self.fibs.con ]
1073        yy = [ list() for f in self.fibs.con ]
1074        # grafs = list() # un plot para cada fibra
1075        for f in range(len(self.fibs.con)): # f es un indice
1076           # el primer nodo y el ultimo de cada fibra son fronteras
1077           s = self.fibs.con[f][0] # obtengo el indice del primer segmento de la fibra numero f
1078           n = self.segs.con[s][0] # obtengo el indice del primer nodo del segmento numero s
1079           r = self.nods.r[n] # obtengo las coordenadas del nodo numero n
1080           xx[f].append(r[0])
1081           yy[f].append(r[1])
1082           s = self.fibs.con[f][-1] # obtengo el indice del ultimo segmento de la fibra numero f
1083           n = self.segs.con[s][1] # obtengo el indice del segundo nodo del ultimo numero s
1084           r = self.nods.r[n] # obtengo las coordenadas del nodo numero n
1085           xx[f].append(r[0])
```

```
1086            yy[f].append(r[1])
1087            # grafs.append( ax.plot(xx[f], yy[f], linewidth=0, marker="x", mec="k", markersize=markersize) )
1088         ax.plot(xx, yy, linewidth=0, marker="o", mec="k", mfc="w", markersize=markersize)
1089
1090      def pre_graficar_nodos_interseccion(self, fig, ax, markersize=8):
1091         # dibujo las fibras (los segmentos)
1092         # preparo las listas, una lista para cada fibra
1093         xx = list()
1094         yy = list()
1095         # grafs = list() # un plot para cada fibra
1096         for n in range(len(self.nods.r)):
1097            if self.nods.tipos[n] == 2:
1098               xx.append(self.nods.r[n][0])
1099               yy.append(self.nods.r[n][1])
1100         ax.plot(xx, yy, linewidth=0, marker="o", mec="k", mfc="w", markersize=markersize)
1101
1102      def pre_graficar_nodos_internos(self, fig, ax):
1103         # dibujo las fibras (los segmentos)
1104         # preparo las listas, una lista para cada fibra
1105         xx = list()
1106         yy = list()
1107         grafs = list() # un plot para cada fibra
1108         for n in range(len(self.nods.r)):
1109            if self.nods.tipos[n] == 0:
1110               xx.append(self.nods.r[n][0])
1111               yy.append(self.nods.r[n][1])
1112         ax.plot(xx, yy, linewidth=0, marker=".", markersize=1)
1113
1114      def pre_graficar(self, fig, ax, lamr_min = None, lamr_max = None, byn = False):
1115         self.pre_graficar_bordes(fig, ax, byn)
1116         self.pre_graficar_nodos_frontera(fig, ax)
1117         self.pre_graficar_nodos_interseccion(fig, ax)
1118         self.pre_graficar_nodos_internos(fig, ax)
1119         self.pre_graficar_fibras(fig, ax, lamr_min=lamr_min, lamr_max=lamr_max, byn=byn)
1120         #ax.legend(loc="upper left", numpoints=1, prop={"size":6})
1121
1122      def graficar(self, fig=None, ax=None, lamr_min=None, lamr_max=None, byn=False):
1123         if ax is None:
1124            fig, ax = plt.subplots()
1125         self.pre_graficar(fig, ax, lamr_min, lamr_max, byn)
1126         plt.show()
```

```
1  """ modulo de funciones auxiliares """
2
3  import numpy as np
4  from scipy import stats
5
6  def find_string_in_file(fid, target, mandatory=False):
7     fid.seek(0) # rewind
8     target = target.lower()
9     len_target = len(target)
10    for linea in fid:
11       if target == linea[:len_target].lower():
12          ierr = 0
13          break
14    else:
15       if mandatory:
16          raise EOFError("final␣de␣archivo␣sin␣encontrar␣target␣string")
17       ierr = 1
18    return ierr
19
20 def iguales(x, y, tol=1.0e-8):
21    """ returns True if x is equal to y, both floats """
22    return np.abs(x-y)<tol
23
24 def calcular_longitud_de_segmento(r0, r1):
25    """ dados dos nodos de un segmento
26    r0: coordenadas "xy" del nodo 0
27    r1: coordenadas "xy" del nodo 1
28    calcula el largo del segmento """
```

```
29     dx = r1[0] - r0[0]
30     dy = r1[1] - r0[1]
31     return np.sqrt(dx*dx + dy*dy)
32 
33 def calcular_angulo_de_segmento(r0, r1):
34     """ dados dos nodos de un segmento
35     r0: coordenadas "xy" del nodo 0
36     r1: coordenadas "xy" del nodo 1
37     calcula el angulo que forman con el eje horizontal """
38     dx = r1[0] - r0[0]
39     dy = r1[1] - r0[1]
40     if iguales(dx,0.): # segmento vertical
41         if iguales(dy,0.): # segmento nulo
42             raise ValueError("Error, segmento de longitud nula!!")
43         elif dy > 0.: # segmento hacia arriba
44             return 0.5*np.pi
45         else: # segmento hacia abajo
46             return -0.5*np.pi
47     elif iguales(dy,0.): # segmento horizontal
48         if dx>0.: # hacia derecha
49             return 0.
50         else: # hacia izquierda
51             return np.pi
52     else: # segmento oblicuo
53         if dx<0.: # segundo o tercer cuadrante
54             return np.pi + np.arctan(dy/dx)
55         elif dy>0.: # primer cuadrante
56             return np.arctan(dy/dx)
57         else: # cuarto cuadrante
58             return 2.*np.pi + np.arctan(dy/dx)
59 
60 def compute_from_curve(x, xx, yy, extrapolar=False):
61     # primero chequeo si x cae fuera de intervalo
62     if x<xx[0] or x>xx[-1]:
63         if extrapolar:
64             if x<xx[0]:
65                 return yy[0]
66             else:
67                 return yy[-1]
68     # de lo contrario tengo que calcularlo dentro de intervalo interpolando linealmente
69     nx = len(xx)
70     for i in range(1,nx):
71         if x<xx[i]:
72             slope = ( yy[i] - yy[i-1] ) / ( xx[i] - xx[i-1] )
73             return yy[i-1] + slope * (x - xx[i-1])
74 
75 def compute_discrete_normal_distribution(mu=1.0, sigma=1.0, n=1001):
76     from scipy.stats import norm
77     x = np.linspace(mu-10.*sigma, mu+10.*sigma, num=n)
78     y = norm.pdf(x)
79     Y = norm.cdf(x)
80     return x, y, Y
81 
82 class Discrete_normal_distribution(object):
83     def __init__(self, mu=0., sigma=1., n=1001):
84         # armo curva discreta
85         self.x = np.linspace(mu - 10.*sigma, mu+10.*sigma, n)
86         self.y = stats.norm.pdf(self.x, mu, sigma)
87         self.Y = stats.norm.cdf(self.x, mu, sigma)
88 
89     def get_sample(self):
90         r = np.random.random()
91         x = compute_from_curve(r, self.Y, self.x, True)
92         return x
```

B.3. Cálculo de intersecciones, simplificación y equilibrio

```
1
2  program Malla_Nanofibras_Fortran
3     USE class_malla_completa
4     use class_mallita
5     USE Aux
6     use programs
7     implicit none
8     ! ========================================================================
9     ! ========================================================================
10    ! Current working directory
11    CHARACTER(LEN=255) :: cwd
12    ! ========================================================================
13    TYPE(MallaCom) :: MC, MC2
14    TYPE(MallaSim) :: ms, ms2
15    CHARACTER(LEN=120) :: configfile
16    integer :: fid_cf
17    integer :: fid_lista_mallas
18    integer :: nmallas
19    integer :: num_instruc
20    integer, allocatable :: lista_instruc(:)
21    integer :: i_etiqueta
22    character(len=3) :: str_etiqueta
23    character(len=120) :: str_instruccion
24    integer :: j_instr
25    ! ----------
26    ! variables para instruccion "Intersectar"
27    integer :: opcion_archivo
28    character(len=120) :: nombre_archivo, nombre_malla
29    integer :: num_pasadas
30    logical :: period_intersec
31    integer :: j_malla
32    ! ----------
33    ! variables para instruccion "Simplificar"
34    integer :: numparam
35    real(8), allocatable :: param(:)
36    ! ----------
37    ! variables para instruccion "Equilibrar"
38    integer :: opcion_Fmacro
39    character(len=120) :: archivo_Fmacro
40    integer :: fid_Fmacro
41    real(8) :: Fmacro(2,2)
42    integer :: num_pasos_vibracion
43    integer, allocatable :: vec_veces(:)
44    real(8), allocatable :: vec_drmags(:)
45    real(8) :: fuerza_ref, fuerza_tol
46    integer :: num_Fmacro
47    real(8), allocatable :: vec_Fmacro(:,:,:)
48    integer :: j_F
49    character(len=8) :: str_j_F
50    ! variables para instruccion "Traccion"
51    real(8) :: delta_t, dot_F11, dot_F22, F11_fin
52    CHARACTER(LEN=120) :: filename_curvacon
53    integer :: opcion_guardar
54    real(8) :: dF_guardar
55    ! ----------
56    CHARACTER(LEN=120) :: filename, mallaname
57    integer :: i,j,k,n,m
58    INTEGER :: iStat, iStat1, iStat2
59    real(8), allocatable :: r1(:,:), Ag(:,:), bg(:), dr(:,:)
60    character(len=120) :: formato
61    real(8), allocatable :: fuerzas_f(:,:), fuerzas_n(:,:)
62    integer :: opcion ! 1=intersectar, 2=simplificar, 3=equilibrio
63    ! ========================================================================
64
65    ! ------------------------------------------------------------------------
66    ! Imprimo carpeta de trabajo actual
67    print *, "Multiscale␣Nanofibers␣Mesh␣RVE␣2.0"
```

```
68  CALL GETCWD(cwd)
69  PRINT *, "Current␣Working␣Directory:", cwd
70  ! ------------------------------------------------------------------------
71  ! Leer ConfigurationFile.txt para obtener instrucciones
72  configfile = "ConfigurationFile.txt"
73  fid_cf = get_file_unit()
74  OPEN(unit=fid_cf, file=trim(configfile), status="old")
75  iStat = FindStringInFile("*␣Numero␣de␣acciones", fid_cf, .true.)
76  read(fid_cf,*) num_instruc
77  allocate( lista_instruc(num_instruc) )
78  read(fid_cf,*) lista_instruc
79  ! Ya que estamos tambien busco y leo los parametros constitutivos
80  iStat = FindStringInFile("*␣Parametros␣Constitutivos", fid_cf, .false.)
81  if (iStat==0) then
82     read(fid_cf,*) numparam
83     allocate( param(numparam) )
84     read(fid_cf,*) param
85  end if
86  CLOSE(unit=fid_cf)
87  ! ------------------------------------------------------------------------
88  ! Recorro las etiquetas de las instrucciones a realizar
89  do j_instr=1,num_instruc
90     ! ---------------------------------------------------------------------
91     ! Busco cada etiqueta en configfile y leo la string de instruccion (es el identificador: Intersectar,
          ↪ Simplificar o Equilibrar)
92     i_etiqueta = lista_instruc(j_instr)
93     WRITE(str_etiqueta,'(A2,I1)') "*␣", i_etiqueta
94     OPEN(unit=fid_cf, file=trim(configfile), status="old")
95     iStat = FindStringInFile(str_etiqueta, fid_cf, .true.)
96     read(fid_cf,*) str_instruccion
97     ! ---------------------------------------------------------------------
98     ! Luego, segun la instruccion me fijo que hay que hacer
99     select case (trim(str_instruccion))
100    ! ---------------------------------------------------------------------
101    ! INTERSECTAR
102    case ("Intersectar")
103       ! Leo los parametros de configuracion
104       read(fid_cf,*) opcion_archivo, nombre_archivo
105       read(fid_cf,*) num_pasadas
106       read(fid_cf,*) period_intersec
107       ! Comienzo a intersectar
108       if (opcion_archivo==1) then
109          ! Caso de una sola malla a intersectar
110          call main_intersectar(nombre_archivo, num_pasadas, period_intersec)
111       elseif (opcion_archivo==2) then
112          ! Caso de una lista de mallas a intersectar, la lista se da en un archivo
113          fid_lista_mallas = get_file_unit()
114          open(unit=fid_lista_mallas, file=trim(nombre_archivo), status="old")
115          read(fid_lista_mallas,*) nmallas
116          ! Recorro las mallas de la lista, calculando las intersecciones para cada una
117          do j_malla=1,nmallas
118             read(fid_lista_mallas,*) nombre_malla
119             write(*,*) "Intersectando␣malla:"
120             write(*,*) nombre_malla
121             call main_intersectar(nombre_malla, num_pasadas, period_intersec)
122          end do
123          ! Cierro el archivo de la lista de mallas
124          close(unit=fid_lista_mallas)
125       else
126          ! Si no encontre opcion 1 o 2, entonces hay algun error!!!
127          write(*,*) "Error,␣para␣Intersectar,␣opcion_archivo␣debe␣ser␣1␣o␣2,␣y␣es:␣", opcion_archivo
128          write(*,*) "En␣etiqueta:␣", str_etiqueta
129          write(*,*) "En␣Instruccion:␣", str_instruccion
130          stop
131       end if
132    ! FIN INTERSECTAR
133    ! ---------------------------------------------------------------------
134    ! SIMPLIFICAR
135    case ("Simplificar")
136       ! Leo los parametros de configuracion
```

```
137         read(fid_cf,*) opcion_archivo, nombre_archivo
138         ! Comienzo a simplificar
139         if (opcion_archivo==1) then
140            ! Caso de una sola malla
141            call main_simplificar(nombre_archivo, numparam, param)
142         elseif (opcion_archivo==2) then
143            ! Caso de una lista de mallas en un archivo
144            fid_lista_mallas = get_file_unit()
145            open(unit=fid_lista_mallas, file=trim(nombre_archivo), status="old")
146            read(fid_lista_mallas,*) nmallas
147            ! Recorro las mallas de la lista
148            do j_malla=1,nmallas
149               read(fid_lista_mallas,*) nombre_malla
150               write(*,*) "Simplificando␣malla:"
151               write(*,*) nombre_malla
152               call main_simplificar(nombre_malla, numparam, param)
153            end do
154            ! Cierro el archivo de la lista de mallas
155            close(unit=fid_lista_mallas)
156         else
157            ! Si no encontre opcion 1 o 2, entonces hay algun error!!!
158            write(*,*) "Error,␣para␣Simplificar,␣opcion_archivo␣debe␣ser␣1␣o␣2,␣y␣es:␣", opcion_archivo
159            write(*,*) "En␣etiqueta:␣", str_etiqueta
160            write(*,*) "En␣Instruccion:␣", str_instruccion
161            stop
162         end if
163      ! FIN SIMPLIFICAR
164      ! ------------------------------------------------------------------------
165      ! EQUILIBRAR
166      case ("Equilibrar")
167         ! Leo parametros
168         read(fid_cf,*) opcion_archivo, nombre_archivo
169         read(fid_cf,*) num_pasos_vibracion
170         if (allocated(vec_veces)) deallocate(vec_veces)
171         if (allocated(vec_drmags)) deallocate(vec_drmags)
172         if (num_pasos_vibracion>0) then
173            allocate( vec_veces(num_pasos_vibracion) )
174            allocate( vec_drmags(num_pasos_vibracion) )
175         end if
176         read(fid_cf,*) vec_veces
177         read(fid_cf,*) vec_drmags
178         read(fid_cf,*) fuerza_ref, fuerza_tol
179         read(fid_cf,*) opcion_Fmacro
180         ! Armo array de deformaciones
181         if (opcion_Fmacro==1) then
182            ! Caso de un solo Fmacro
183            read(fid_cf,*) Fmacro
184            allocate( vec_Fmacro(2,2,1) )
185            vec_Fmacro(:,:,1) = Fmacro
186         elseif (opcion_Fmacro==2) then
187            ! Caso de un archivo con una lista de deformaciones
188            read(fid_cf,*) archivo_Fmacro
189            fid_Fmacro = get_file_unit()
190            open(unit=fid_Fmacro, file=trim(archivo_Fmacro), status="old")
191            read(fid_Fmacro,*) num_Fmacro
192            allocate( vec_Fmacro(2,2,num_Fmacro) )
193            do j_F=1,num_Fmacro
194               read(fid_Fmacro,*) vec_Fmacro(:,:,j_F)
195            end do
196         else
197            write(*,*) "Error␣en␣opcion_Fmacro,␣deberia␣ser␣1␣o␣2,␣y␣es:␣", opcion_Fmacro
198            write(*,*) "En␣etiqueta:␣", str_etiqueta
199            write(*,*) "En␣instruccion:␣", str_instruccion
200            stop
201         end if
202         ! Ahora resuelvo el equilibrio para las deformaciones que tengo
203         if (opcion_archivo==1) then
204            ! Caso de una sola malla
205            ! Recorro los Fmacro de la lista para hacer todos los equilibrios
206            do j_F=1,num_Fmacro
```

```
207                Fmacro = vec_Fmacro(:,:,j_F)
208                write(str_j_F, "(A1,␣I7.7)") "_", j_F ! 7.7 indica que el campo es de 7 y se usan como minimo 7,
                       ↪ entonces imprime los ceros
209                call main_equilibrar(nombre_archivo, numparam, param, Fmacro, num_pasos_vibracion, vec_veces,
                       ↪ vec_drmags, fuerza_ref, fuerza_tol, str_j_F)
210             end do
211          elseif (opcion_archivo==2) then
212             ! Caso de una lista de mallas en un archivo
213             fid_lista_mallas = get_file_unit()
214             open(unit=fid_lista_mallas, file=trim(nombre_archivo), status="old")
215             read(fid_lista_mallas,*) nmallas
216             ! Recorro las mallas de la lista
217             do j_malla=1,nmallas
218                read(fid_lista_mallas,*) nombre_malla
219                write(*,*) "Equilibrando␣malla:"
220                write(*,*) nombre_malla
221                ! Recorro los Fmacro de la lista para hacer todos los equilibrios
222                do j_F=1,num_Fmacro
223                   Fmacro = vec_Fmacro(:,:,j_F)
224                   write(str_j_F, "(A1,␣I7.7)") "_", j_F ! 7.7 indica que el campo es de 7 y se usan como minimo 7,
                          ↪ entonces imprime los ceros
225                   call main_equilibrar(nombre_archivo, numparam, param, Fmacro, num_pasos_vibracion, vec_veces,
                          ↪ vec_drmags, fuerza_ref, fuerza_tol, str_j_F)
226                end do
227             end do
228             ! Cierro el archivo de la lista de mallas
229             close(unit=fid_lista_mallas)
230          else
231             ! Si no encontre opcion 1 o 2, entonces hay algun error!!!
232             write(*,*) "Error,␣opcion_archivo␣debe␣ser␣1␣o␣2,␣y␣es:␣", opcion_archivo
233             write(*,*) "En␣etiqueta:␣", str_etiqueta
234             write(*,*) "En␣Instruccion:␣", str_instruccion
235             stop
236          end if
237          ! Cierro archivo de deformaciones
238          close(unit=fid_Fmacro)
239       ! FIN EQUILIBRAR
240       ! ------------------------------------------------------------------------
241       ! TRACCION
242       case ("Traccion")
243          ! Leo parametros
244          read(fid_cf,*) opcion_archivo, nombre_archivo
245          read(fid_cf,*) num_pasos_vibracion
246          if (allocated(vec_veces)) deallocate(vec_veces)
247          if (allocated(vec_drmags)) deallocate(vec_drmags)
248          if (num_pasos_vibracion>0) then
249             allocate( vec_veces(num_pasos_vibracion) )
250             allocate( vec_drmags(num_pasos_vibracion) )
251          end if
252          read(fid_cf,*) vec_veces
253          read(fid_cf,*) vec_drmags
254          read(fid_cf,*) fuerza_ref, fuerza_tol
255          read(fid_cf,*) delta_t, dot_F11, dot_F22, F11_fin
256          read(fid_cf,*) filename_curvacon
257          read(fid_cf,*) opcion_guardar, dF_guardar
258          !
259          if (opcion_archivo==1) then
260             ! Caso de una sola malla
261             ! Recorro los Fmacro de la lista para hacer todos los equilibrios
262             call main_traccion(nombre_archivo, numparam, param, num_pasos_vibracion, vec_veces, vec_drmags, fuerza_ref
                    ↪ , fuerza_tol, delta_t, dot_F11, dot_F22, F11_fin, filename_curvacon, opcion_guardar, dF_guardar)
263    ! elseif (opcion_archivo==2) then
264    ! stop
265          else
266             ! Si no encontre opcion 1 o 2, entonces hay algun error!!!
267             write(*,*) "Error,␣opcion_archivo␣debe␣ser␣1,␣y␣es:␣", opcion_archivo
268             write(*,*) "En␣etiqueta:␣", str_etiqueta
269             write(*,*) "En␣Instruccion:␣", str_instruccion
270             stop
271          end if
```

```
272       ! FIN TRACCION
273       ! ------------------------------------------------------------------------
274       ! UNIAXIAL
275       case ("Uniaxial")
276          ! Leo parametros
277          read(fid_cf,*) opcion_archivo, nombre_archivo
278          read(fid_cf,*) num_pasos_vibracion
279          if (allocated(vec_veces)) deallocate(vec_veces)
280          if (allocated(vec_drmags)) deallocate(vec_drmags)
281          if (num_pasos_vibracion>0) then
282             allocate( vec_veces(num_pasos_vibracion) )
283             allocate( vec_drmags(num_pasos_vibracion) )
284          end if
285          read(fid_cf,*) vec_veces
286          read(fid_cf,*) vec_drmags
287          read(fid_cf,*) fuerza_ref, fuerza_tol
288          read(fid_cf,*) delta_t, dot_F11, dot_F22, F11_fin
289          read(fid_cf,*) filename_curvacon
290          read(fid_cf,*) opcion_guardar, dF_guardar
291          !
292          if (opcion_archivo==1) then
293             ! Caso de una sola malla
294             ! Recorro los Fmacro de la lista para hacer todos los equilibrios
295             call main_uniaxial(nombre_archivo, numparam, param, num_pasos_vibracion, vec_veces, vec_drmags, fuerza_ref
                    ↪ , fuerza_tol, delta_t, dot_F11, dot_F22, F11_fin, filename_curvacon, opcion_guardar, dF_guardar)
296 ! elseif (opcion_archivo==2) then
297 ! stop
298          else
299             ! Si no encontre opcion 1 o 2, entonces hay algun error!!!
300             write(*,*) "Error, opcion_archivo debe ser 1, y es: ", opcion_archivo
301             write(*,*) "En etiqueta: ", str_etiqueta
302             write(*,*) "En Instruccion: ", str_instruccion
303             stop
304          end if
305       ! FIN UNIAXIAL
306       ! ------------------------------------------------------------------------
307       case default
308          write(*,*) "Instruccion desconocida: ", str_instruccion
309          write(*,*) "En etiqueta: ", str_etiqueta
310          stop
311       ! ------------------------------------------------------------------------
312       end select
313       ! ------------------------------------------------------------------------
314
315       close(unit=fid_cf)
316    end do
317    ! ------------------------------------------------------------------------
318
319
320 end program Malla_Nanofibras_Fortran
321 ! ============================================================================
322 ! ============================================================================
323 ! ============================================================================
```

```
1  module programs
2
3  use Aux
4  USE class_malla_completa, ONLY : MallaCom
5  use class_mallita, only : MallaSim
6
7  contains
8
9  ! ============================================================================
10 subroutine main_intersectar(filename_malla_in, npasadas, periodicidad)
11    use class_malla_completa
12    implicit none
13    CHARACTER(LEN=120), intent(in) :: filename_malla_in
14    integer, intent(in) :: npasadas
15    logical, intent(in) :: periodicidad
16    character(len=120) :: filename_malla_in2, filename_malla_out
```

```
17    TYPE(MallaCom) :: MC, MC2
18    integer :: i
19    integer :: iStat1, iStat2
20
21    if (trim(filename_malla_in) == "default") then
22       filename_malla_in2 = "Malla.txt"
23    else
24       filename_malla_in2 = filename_malla_in
25    end if
26
27    write(*,*) "Leer malla, intersectar fibras y reescribir:"
28    CALL leer_malla(MC, filename_malla_in2)
29 ! if (MC%sidelen > 99.d0) then
30 ! write(*,*) "malla con problema"
31 ! end if
32    ! Hago la interseccion muchas veces porque cada vez tengo la limitacion de no cortar al mismo segmento dos veces
33    i = 0
34    write(*,*) "Intersectando fibras"
35    iStat1 = 0
36    iStat2 = 0
37    write(*,*) mc%nsegs
38    DO WHILE (.true.)
39       i = i+1
40       WRITE(*,'(I4)', ADVANCE='no') i
41       CALL intersectar_fibras(MC, MC2, .FALSE., periodicidad, iStat1) ! dentro de la misma capa
42       MC = MC2
43       CALL intersectar_fibras(MC, MC2, .TRUE., periodicidad, iStat2) ! con capas adyacentes
44       MC = MC2
45       write(*,*) mc%nsegs
46       IF ( (iStat1 == 1).AND.(iStat2 == 1) ) EXIT
47       if (i==npasadas) exit
48    END DO
49    write(*,*)
50
51    write(*,*) "Escribiendo malla intersectada"
52    filename_malla_out = "_i"
53    call modify_txt_filename(filename_malla_in2, filename_malla_out)
54    CALL escribir_malla(mc, filename_malla_out)
55    write(*,*) "Malla intersectada OK"
56
57 end subroutine main_intersectar
58 ! ========================================================================
59
60 ! ========================================================================
61 subroutine main_simplificar(filename_malla_in, nparamcon, paramcon)
62    use class_malla_completa
63    use class_mallita
64    implicit none
65    CHARACTER(LEN=120), intent(in) :: filename_malla_in
66    integer, intent(in) :: nparamcon
67    real(8), intent(in) :: paramcon(nparamcon)
68    character(len=120) :: filename_malla_in2, filename_malla_out
69    type(MallaCom) :: mc
70    type(MallaSim) :: ms
71
72    if (trim(filename_malla_in) == "default") then
73       filename_malla_in2 = "Malla_i.txt"
74    else
75       filename_malla_in2 = "_i"
76       call modify_txt_filename(filename_malla_in, filename_malla_in2)
77    end if
78
79
80    write(*,*) "Leer malla intersectada y generar malla simplificada:"
81    call leer_malla(mc, filename_malla_in2)
82    call Desde_MallaCom(mc, ms, nparamcon, paramcon)
83
84    write(*,*) "Escribiendo mallita"
85    filename_malla_out = "_s"
86    call modify_txt_filename(filename_malla_in2, filename_malla_out)
```

```
87    CALL escribir_mallita(ms, filename_malla_out)
88    write(*,*) "Malla simplificada OK"
89
90 end subroutine main_simplificar
91 ! ========================================================================
92
93 ! ========================================================================
94 subroutine main_equilibrar(filename_malla_in, nparcon, parcon, Fmacro, num_pasos, lista_veces, lista_drmags, fzaref,
       ↪ fzatol, str_num_output_opt)
95    ! Calcula el equilibrio elastico de una malla dado un tensor F macroscopico (Fmacro)
96    ! ----------
97    use class_mallita
98    implicit none
99    ! ----------
100   CHARACTER(LEN=120), intent(in) :: filename_malla_in
101   integer, intent(in) :: nparcon
102   real(8), intent(in) :: parcon(nparcon)
103   real(8), intent(in) :: Fmacro(2,2)
104   integer, intent(in) :: num_pasos
105   integer, intent(in) :: lista_veces(num_pasos)
106   real(8), intent(in) :: lista_drmags(num_pasos)
107   real(8), intent(in) :: fzaref
108   real(8), intent(in) :: fzatol
109   character(len=8), intent(in), optional :: str_num_output_opt
110   ! ----------
111   character(len=120) :: filename_malla_in2, filename_malla_out, aux_string
112   type(MallaSim) :: ms
113   integer :: n, iStat1
114   ! ----------
115
116   if (trim(filename_malla_in) == "default") then
117      filename_malla_in2 = "Malla_i_s.txt"
118   else
119      filename_malla_in2 = "_i_s"
120      call modify_txt_filename(filename_malla_in, filename_malla_in2)
121   end if
122
123   write(*,*) "Calculando equilibrio"
124   call leer_mallita(ms, filename_malla_in2, nparcon, parcon)
125   n = ms%nnods
126
127   call calcular_equilibrio(ms, num_pasos, lista_veces, lista_drmags, fzaref, fzatol, Fmacro)
128
129   filename_malla_out = "_e"
130   call modify_txt_filename(filename_malla_in2, filename_malla_out)
131   if (present(str_num_output_opt)) then
132      aux_string = str_num_output_opt
133      call modify_txt_filename(filename_malla_out, aux_string)
134      filename_malla_out = aux_string
135   end if
136   call escribir_mallita(ms, filename_malla_out)
137   write(*,*) "Equilibrio calculado OK"
138
139   ! ----------
140 end subroutine main_equilibrar
141 ! ========================================================================
142
143
144 ! ========================================================================
145 subroutine main_traccion(filename_malla_in, nparcon, parcon, num_pasos, lista_veces, lista_drmags, fzaref, fzatol,
       ↪ dtime, dotF11, dotF22, F11fin, filename_curva, opcion_save, dF_save)
146   ! Simula un ensayo de traccion con un esquema explicito
147   ! imponiendo tasas de deformacion axial y transversal
148   ! ----------
149   use class_mallita
150   implicit none
151   ! ----------
152   CHARACTER(LEN=120), intent(in) :: filename_malla_in
153   integer, intent(in) :: nparcon
154   real(8), intent(in) :: parcon(nparcon)
```

```
155     integer, intent(in) :: num_pasos
156     integer, intent(in) :: lista_veces(num_pasos)
157     real(8), intent(in) :: lista_drmags(num_pasos)
158     real(8), intent(in) :: fzaref
159     real(8), intent(in) :: fzatol
160     real(8), intent(in) :: dtime
161     real(8), intent(in) :: dotF11
162     real(8), intent(in) :: dotF22
163     real(8), intent(in) :: F11fin
164     CHARACTER(LEN=120), intent(in) :: filename_curva
165     integer, intent(in) :: opcion_save
166     real(8), intent(in) :: dF_save
167     ! ----------
168     character(len=120) :: filename_malla_in2, filename_malla_out
169     real(8) :: F11ini
170     integer :: fid_curva
171     real(8) :: time
172     real(8) :: Fmacro(2,2)
173     real(8) :: lamps
174     type(MallaSim) :: ms
175     integer :: nsaves
176     real(8), allocatable :: lista_saves_F(:)
177     logical, allocatable :: lista_saves_if(:)
178     logical :: listo_saves = .false.
179     integer :: isave
180     ! ----------
181     integer :: f
182     ! ----------
183
184     write(*,*) "Empezando␣Traccion"
185     ! Preparo la lista de saves si es que hay
186     if (opcion_save==1) then
187        nsaves = int((F11fin-1.0d0) / dF_save) + 1
188        allocate( lista_saves_F(nsaves) )
189        allocate( lista_saves_if(nsaves) )
190        do isave=1,nsaves
191           lista_saves_F(isave) = 1. + dfloat(isave - 1)*dF_save
192        end do
193        lista_saves_if = .false.
194     end if
195     ! Leo la malla
196     write(*,*) "Leyendo␣Malla:"
197     if (trim(filename_malla_in) == "default") then
198        filename_malla_in2 = "Malla_i_s.txt"
199     else
200        filename_malla_in2 = trim(filename_malla_in)
201     end if
202     write(*,*) "archivo:␣", filename_malla_in2
203     call leer_mallita(ms, filename_malla_in2, nparcon, parcon)
204     if (ms%status_deformed) then
205        ! Si la malla esta previamente deformada, empiezo a trabajar desde alli
206        Fmacro = ms%Fmacro
207        F11ini = Fmacro(1,1)
208        lista_saves_if = (F11ini > lista_saves_F)
209        isave = count(lista_saves_if) + 1
210        ! Abro un archivo viejo para continuar la curva constitutiva
211        fid_curva = get_file_unit()
212        open(unit=fid_curva, file=trim(filename_curva), status="old", position="append", action="write")
213     else
214        ! Si la malla esta virge, empiezo desde deformacion nula
215        Fmacro = reshape(source=[1.d0, 0.d0, 0.d0, 1.d0], shape=shape(Fmacro))
216        isave = 1
217        ! Abro un archivo nuevo para escribir la curva constitutiva
218        fid_curva = get_file_unit()
219        open(unit=fid_curva, file=trim(filename_curva), status="replace")
220     end if
221
222     ! Comienzo esquema temporal
223     time = 0.d0
224     do while ( Fmacro(1,1) .le. F11fin )
```

```
225       ! Calculo el equilibrio de la malla para el Fmacro dado en este paso de tiempo
226       call calcular_equilibrio(ms, num_pasos, lista_veces, lista_drmags, fzaref, fzatol, Fmacro)
227       ! Guardo en archivo la informacion constitutiva para este step
228       write(*,"(2E20.8E4)") Fmacro(1,1), ms%Tmacro(1,1)
229       write(fid_curva,"(8E20.8E4)") Fmacro, ms%Tmacro
230       ! Calculo plasticidad y/o rotura de fibras
231       call calcular_plasticidad_rotura(ms, dtime)
232       ! Me fijo si guardo la malla o no
233       if (.not. listo_saves) then
234          if ( dabs( Fmacro(1,1) - lista_saves_F(isave) ) < dotF11*dtime ) then
235             write(filename_malla_out,"(A6,␣I4.4)") "_save_", isave
236             call modify_txt_filename(filename_malla_in2, filename_malla_out)
237             call escribir_mallita(ms, filename_malla_out)
238             isave = isave + 1
239             if (isave > nsaves) listo_saves = .true.
240          end if
241       end if
242       ! Incremento tiempo y deformacion para siguiente paso
243       Fmacro(1,1) = Fmacro(1,1) + dotF11*dtime
244       Fmacro(2,2) = Fmacro(2,2) + dotF22*dtime
245       time = time + dtime
246    end do
247    ! Cierro el archivo donde guarda la curva constitutiva
248    close(unit=fid_curva)
249
250    ! ----------
251 end subroutine main_traccion
252 ! ========================================================================
253
254
255 ! ========================================================================
256 subroutine main_uniaxial(filename_malla_in, nparcon, parcon, num_pasos, lista_veces, lista_drmags, fzaref, fzatol,
        ↪ dtime, dotF11, T22, F11fin, filename_curva, opcion_save, dF_save)
257    ! Simula un ensayo de traccion con un esquema explicito
258    ! imponiendo tasas de deformacion axial y transversal
259    ! ----------
260    use class_mallita
261    implicit none
262    ! ----------
263    CHARACTER(LEN=120), intent(in) :: filename_malla_in
264    integer, intent(in) :: nparcon
265    real(8), intent(in) :: parcon(nparcon)
266    integer, intent(in) :: num_pasos
267    integer, intent(in) :: lista_veces(num_pasos)
268    real(8), intent(in) :: lista_drmags(num_pasos)
269    real(8), intent(in) :: fzaref
270    real(8), intent(in) :: fzatol
271    real(8), intent(in) :: dtime
272    real(8), intent(in) :: dotF11
273    real(8), intent(in) :: T22 ! tension contra la cual contraer (en uniaxial verdadero seria cero)
274    real(8), intent(in) :: F11fin
275    CHARACTER(LEN=120), intent(in) :: filename_curva
276    integer, intent(in) :: opcion_save
277    real(8), intent(in) :: dF_save
278    ! ----------
279    character(len=120) :: filename_malla_in2, filename_malla_out
280    real(8) :: F11ini
281    integer :: fid_curva
282    real(8) :: time
283    real(8) :: Fmacro(2,2)
284    real(8) :: lamps
285    type(MallaSim) :: ms
286    integer :: nsaves
287    real(8), allocatable :: lista_saves_F(:)
288    logical, allocatable :: lista_saves_if(:)
289    logical :: listo_saves = .false.
290    integer :: isave
291    ! ----------
292    integer :: f, k, maxk=100
293    real(8) :: dF22 = 0.001d0
```

```
294    ! ----------
295
296    write(*,*) "Empezando␣Uniaxial"
297    ! Preparo la lista de saves si es que hay
298    if (opcion_save==1) then
299       nsaves = int((F11fin-1.0d0) / dF_save) + 1
300       allocate( lista_saves_F(nsaves) )
301       allocate( lista_saves_if(nsaves) )
302       do isave=1,nsaves
303          lista_saves_F(isave) = 1. + dfloat(isave - 1)*dF_save
304       end do
305       lista_saves_if = .false.
306    end if
307    ! Leo la malla
308    write(*,*) "Leyendo␣Malla:"
309    if (trim(filename_malla_in) == "default") then
310       filename_malla_in2 = "Malla_i_s.txt"
311    else
312 ! filename_malla_in2 = "_i_s"
313 ! call modify_txt_filename(filename_malla_in, filename_malla_in2)
314       filename_malla_in2 = trim(filename_malla_in)
315    end if
316    write(*,*) "archivo:␣", filename_malla_in2
317    call leer_mallita(ms, filename_malla_in2, nparcon, parcon)
318    if (ms%status_deformed) then
319       ! Si la malla esta previamente deformada, empiezo a trabajar desde alli
320       Fmacro = ms%Fmacro
321       F11ini = Fmacro(1,1)
322       lista_saves_if = (F11ini > lista_saves_F)
323       isave = count(lista_saves_if) + 1
324       ! Abro un archivo viejo para continuar la curva constitutiva
325       fid_curva = get_file_unit()
326       open(unit=fid_curva, file=trim(filename_curva), status="old", position="append", action="write")
327    else
328       ! Si la malla esta virge, empiezo desde deformacion nula
329       Fmacro = reshape(source=[1.d0, 0.d0, 0.d0, 1.d0], shape=shape(Fmacro))
330       isave = 1
331       ! Abro un archivo nuevo para escribir la curva constitutiva
332       fid_curva = get_file_unit()
333       open(unit=fid_curva, file=trim(filename_curva), status="replace")
334    end if
335
336    ! Comienzo esquema temporal
337    time = 0.d0
338    do while ( Fmacro(1,1) .le. F11fin )
339       ! Calculo el equilibrio de la malla para el Fmacro dado en este paso de tiempo y F22 de la iteracion anterior
340       do k=1,maxk
341          call calcular_equilibrio(ms, num_pasos, lista_veces, lista_drmags, fzaref, fzatol, Fmacro)
342          if (ms%Tmacro(2,2)>T22) then
343             Fmacro(2,2) = Fmacro(2,2) - dF22
344          else if (ms%Tmacro(2,2) < T22) then
345             Fmacro(2,2) = Fmacro(2,2) + dF22
346             exit
347          else
348             exit
349          end if
350       end do
351       ! Guardo en archivo la informacion constitutiva para este step
352       write(*,"(4E20.8E4)") Fmacro(1,1), ms%Tmacro(1,1), Fmacro(2,2), ms%Tmacro(2,2)
353       write(fid_curva,"(8E20.8E4)") Fmacro, ms%Tmacro
354       ! Calculo plasticidad y/o rotura de fibras
355       call calcular_plasticidad_rotura(ms, dtime)
356       ! Me fijo si guardo la malla o no
357       if (.not. listo_saves) then
358          if ( dabs( Fmacro(1,1) - lista_saves_F(isave) ) < dotF11*dtime ) then
359             write(filename_malla_out,"(A7,␣I4.4)") "_uniax_", isave
360             call modify_txt_filename(filename_malla_in2, filename_malla_out)
361             call escribir_mallita(ms, filename_malla_out)
362             isave = isave + 1
363             if (isave > nsaves) listo_saves = .true.
```

```
364         end if
365       end if
366       ! Incremento tiempo y deformacion para siguiente paso
367       Fmacro(1,1) = Fmacro(1,1) + dotF11*dtime
368       time = time + dtime
369    end do
370    ! Cierro el archivo donde guarda la curva constitutiva
371    close(unit=fid_curva)
372
373    ! ----------
374 end subroutine main_uniaxial
375 ! ========================================================================
376
377 end module
```

```
1  ! ================================================================================
2  MODULE class_malla_completa
3  ! ================================================================================
4     USE Aux
5     IMPLICIT NONE
6
7     !PRIVATE
8     !PUBLIC :: MallaCom
9     !PUBLIC :: leer_malla
10
11    TYPE MallaCom
12       ! diemensiones generales
13       REAL(8) :: sidelen
14       REAL(8) :: diamed
15       real(8) :: volfrac
16       real(8) :: lenseg
17       real(8) :: themax
18       ! nodos
19       INTEGER :: nnods
20       INTEGER, ALLOCATABLE :: tipos(:)
21       REAL(8), ALLOCATABLE :: rnods(:,:)
22       ! segmentos
23       INTEGER :: nsegs
24       INTEGER, ALLOCATABLE :: segs(:,:)
25       ! fibras
26       INTEGER :: nfibs
27       INTEGER :: lenfibsje
28       INTEGER, ALLOCATABLE :: fibsne(:), fibsie(:), fibsje(:)
29       REAL(8), ALLOCATABLE :: fibsds(:), fibsdls(:), fibsdths(:)
30       ! capas
31       INTEGER :: ncaps
32       INTEGER :: lencapsje
33       INTEGER, ALLOCATABLE :: capsne(:), capsie(:), capsje(:)
34    CONTAINS
35       !
36    END TYPE MallaCom
37
38    TYPE Interseccion
39       INTEGER :: tipo
40       INTEGER :: fibs(2)
41       INTEGER :: segs(2)
42       INTEGER :: segs_f(2)
43       INTEGER :: seg1(2)
44       INTEGER :: seg2(2)
45       INTEGER :: inewseg1
46       INTEGER :: inewseg2
47       INTEGER :: inewnod
48       REAL(8) :: newnodr(2)
49    CONTAINS
50       !
51    END TYPE Interseccion
52
53 ! ================================================================================
54 CONTAINS
55 ! ================================================================================
```

```
56
57
58  ! ================================================================================
59  ! ================================================================================
60  SUBROUTINE leer_malla(malla, nomarch)
61     IMPLICIT NONE
62     CHARACTER(LEN=120), INTENT(IN) :: nomarch
63     TYPE(MallaCom), INTENT(INOUT) :: malla
64     !
65     INTEGER :: fid, iStatus
66     INTEGER :: i, j, k
67     INTEGER :: tipo
68     REAL(8) :: r(2)
69     INTEGER :: seg(2)
70     REAL(8) :: dl, d, dth
71     INTEGER :: nfibsegs, lenfibsje
72     INTEGER :: ncapfibs, lencapsje
73     INTEGER :: ie0, ie1
74
75     ! ----------
76     fid = get_file_unit()
77     OPEN(UNIT=fid, FILE=TRIM(nomarch), STATUS="OLD")
78     ! ----------
79     iStatus = FindStringInFile("*parametros", fid, .TRUE.)
80     READ(fid,*) malla%sidelen ! viene em microns
81     READ(fid,*) malla%diamed ! viene em microns
82     READ(fid,*) malla%volfrac
83     READ(fid,*) malla%lenseg ! viene en microns
84     READ(fid,*) malla%themax ! lo leo en grados
85     malla%themax = malla%themax * PI / 180.d0 ! lo paso a radianes
86     ! ----------
87     iStatus = FindStringInFile("*coordenadas", fid, .TRUE.)
88     READ(fid,*) malla%nnods
89     ALLOCATE( malla%rnods(2,malla%nnods) )
90     ALLOCATE( malla%tipos(malla%nnods) )
91     DO i=1,malla%nnods
92        READ(fid,*) j, tipo, r ! los leo en microns
93        malla%rnods(:,i) = r
94        malla%tipos(i) = tipo
95     END DO
96     ! ----------
97     iStatus = FindStringInFile("*segmentos", fid, .TRUE.)
98     READ(fid,*) malla%nsegs
99     ALLOCATE( malla%segs(2,malla%nsegs) )
100    DO i=1,malla%nsegs
101       READ(fid,*) j, seg
102       malla%segs(:,i) = seg + 1 ! sumo uno porque vengo de python con base 0
103    END DO
104    ! ----------
105    iStatus = FindStringInFile("*fibras", fid, .TRUE.)
106    READ(fid,*) malla%nfibs
107    ALLOCATE( malla%fibsdls(malla%nfibs) )
108    ALLOCATE( malla%fibsds(malla%nfibs) )
109    ALLOCATE( malla%fibsdths(malla%nfibs) )
110    ALLOCATE( malla%fibsne(malla%nfibs) )
111    ALLOCATE( malla%fibsie(malla%nfibs + 1) )
112    ! Recorro dos veces las fibras
113    ! la primera guardo el ne y calculo lenfibsje (cant total de segs)
114    ! la segunda calculo el ie y guardo el je
115    ! notar que el lenje de las fibras deberia ser el numero total de segmentos
116    ! porque cada segmento solo pertenece a una fibra, y el total de fibras contienen a todos los segmentos
117    lenfibsje = 0
118    DO i=1,malla%nfibs
119       READ(fid,*) j, dl, d, dth, nfibsegs
120       malla%fibsne(i) = nfibsegs
121       lenfibsje = lenfibsje + nfibsegs
122    END DO
123    malla%lenfibsje = lenfibsje
124    iStatus = FindStringInFile("*fibras", fid, .TRUE.)
125    READ(fid,*) malla%nfibs ! redundante pero necesario leer algo
```

```
126     ALLOCATE( malla%fibsje(lenfibsje) )
127     malla%fibsie(1) = 1
128     DO i=1,malla%nfibs
129        malla%fibsie(i+1) = malla%fibsie(i) + malla%fibsne(i)
130        ie0 = malla%fibsie(i)
131        ie1 = malla%fibsie(i+1)
132        READ(fid,*) j, dl, d, dth, nfibsegs, malla%fibsje(ie0:ie1-1)
133        malla%fibsje(ie0:ie1-1) = malla%fibsje(ie0:ie1-1) + 1 ! necesario porque vengo de python con base 0
134        malla%fibsdls(i) = dl
135        malla%fibsds(i) = d
136        malla%fibsdths(i) = dth
137     END DO
138     ! ----------
139     iStatus = FindStringInFile("*capas", fid, .TRUE.)
140     READ(fid,*) malla%ncaps
141     ALLOCATE( malla%capsne(malla%ncaps) )
142     ALLOCATE( malla%capsie(malla%ncaps + 1) )
143     ! Recorro dos veces las capas
144     ! la primera guardo el ne y calculo lenje (cant total de fibras)
145     ! la segunda calculo el ie y guardo el je
146     ! notar que el lenje de las capas deberia ser el numero total de fibras
147     ! porque cada fibra solo pertenece a una capa, y el total de capas contienen a todas las fibras
148     lencapsje = 0
149     DO i=1,malla%ncaps
150        READ(fid,*) j, ncapfibs
151        malla%capsne(i) = ncapfibs
152        lencapsje = lencapsje + ncapfibs
153     END DO
154     malla%lencapsje = lencapsje
155     iStatus = FindStringInFile("*capas", fid, .TRUE.)
156     READ(fid,*) malla%ncaps ! redundante pero necesario leer algo
157     ALLOCATE( malla%capsje(lencapsje) )
158     malla%capsie(1) = 1
159     DO i=1,malla%ncaps
160        malla%capsie(i+1) = malla%capsie(i) + malla%capsne(i)
161        ie0 = malla%capsie(i)
162        ie1 = malla%capsie(i+1)
163        READ(fid,*) j, ncapfibs, malla%capsje(ie0:ie1-1)
164        malla%capsje(ie0:ie1-1) = malla%capsje(ie0:ie1-1) + 1 ! vengo de python con base 0
165     END DO
166     ! ----------
167     CLOSE(fid)
168     ! ----------
169
170
171 END SUBROUTINE leer_malla
172 ! ================================================================================
173 ! ================================================================================
174
175
176 ! ================================================================================
177 ! ================================================================================
178 SUBROUTINE escribir_malla(malla, nomarch)
179     IMPLICIT NONE
180     CHARACTER(LEN=120), INTENT(IN) :: nomarch
181     TYPE(MallaCom), INTENT(INOUT) :: malla
182     !
183     INTEGER :: fid
184     INTEGER :: i
185     CHARACTER(LEN=120) :: formato
186
187     ! ----------
188     fid = get_file_unit()
189     OPEN(UNIT=fid, FILE=TRIM(nomarch), STATUS="REPLACE")
190     ! ----------
191     ! Parametros
192     WRITE(fid,'(A11)') "*Parametros"
193     WRITE(fid,'(E20.8E4)') malla%sidelen
194     WRITE(fid,'(E20.8E4)') malla%diamed
195     WRITE(fid,'(E20.8E4)') malla%volfrac
```

```
196     WRITE(fid,'(E20.8E4)') malla%lenseg
197     WRITE(fid,'(E20.8E4)') malla%themax * 180.d0 / PI ! lo guardo en grados
198     ! ----------
199     ! Nodos
200     WRITE(fid,'(A12)') "*Coordenadas"
201     WRITE(fid,'(I12)') malla%nnods
202     DO i=1,malla%nnods
203        WRITE(fid, '(I12,I2,2E20.8E4)') i-1, malla%tipos(i), malla%rnods(:,i)
204     END DO
205     ! ----------
206     ! Segmentos
207     WRITE(fid,'(A10)') "*Segmentos"
208     WRITE(fid, '(I12)') malla%nsegs
209     DO i=1,malla%nsegs
210        WRITE(fid,'(I12,␣2I12)') i-1, malla%segs(:,i) - 1
211     END DO
212     ! ----------
213     ! Fibras
214     WRITE(fid,'(A7)') "*Fibras"
215     WRITE(fid,'(I12)') malla%nfibs
216     DO i=1,malla%nfibs
217        WRITE(formato, '(A18,I0,A4)') "(I12,3E20.8E4,I12,",malla%fibsne(i),"I12)"
218        WRITE(fid,formato) i-1, malla%fibsdls(i), malla%fibsds(i), malla%fibsdths(i), malla%fibsne(i), malla%fibsje(
               ↪ malla%fibsie(i):malla%fibsie(i+1)-1) - 1
219     END DO
220     ! ----------
221     ! Capas
222     WRITE(fid,'(A6)') "*Capas"
223     WRITE(fid,'(I12)') malla%ncaps
224     DO i=1,malla%ncaps
225        WRITE(formato, '(A6,I0,A4)') "(2I12,",malla%capsne(i),"I12)"
226        WRITE(fid,formato) i-1, malla%capsne(i), malla%capsje(malla%capsie(i):malla%capsie(i+1)-1) - 1
227     END DO
228     ! ----------
229     CLOSE(fid)
230     ! ----------
231
232
233 END SUBROUTINE escribir_malla
234 ! ================================================================================
235 ! ================================================================================
236
237
238 ! ================================================================================
239 ! ================================================================================
240 SUBROUTINE intersectar_fibras(m_in, m_out, label_intercapas, label_periodicidad, iStatus)
241     ! voy a recorrer m_in, sus capas y fibras, encontrando intersecciones entre fibras
242     ! como resultado la malla de salida es una malla aumentada porque tiene mas nodos y mas segmentos
243     ! en primera instancia la voy a sobredimensionar en memoria
244     ! luego readjudico la memoria correcta
245     IMPLICIT NONE
246     TYPE(MallaCom), INTENT(IN) :: m_in
247     TYPE(MallaCom), INTENT(OUT) :: m_out
248     LOGICAL, INTENT(IN) :: label_intercapas
249     logical, intent(in) :: label_periodicidad
250     INTEGER, INTENT(OUT) :: iStatus
251     !
252     INTEGER :: c1, c2 ! numeracion de capas
253     INTEGER :: f1_c1, f1 ! numeracion de fibra 1: local dentro de la capa c1 y global
254     INTEGER :: s1_f1, s1 ! numeracion de segmento 1: local dentro de la fibra f1 y global
255     INTEGER :: f2_c1, f2 ! numeracion de fibra 2: local y global
256     INTEGER :: s2_f2, s2 ! numeracion de segmento 2: local y global
257     INTEGER :: n11, n12, n21, n22
258     REAL(8) :: r11(2), r12(2), r21(2), r22(2)
259     INTEGER :: iStat
260     REAL(8) :: rin(2)
261     !
262     INTEGER :: i1fibs1, nfibs1
263     INTEGER, ALLOCATABLE :: fibs1(:)
264     !
```

```
265    ! preadjudico las variables con el doble de tamanio para que sobre
266    INTEGER :: nins ! numero de intersecciones
267    TYPE(Interseccion) :: ins(100000) ! preadjudicado grande para que sobre
268    INTEGER :: nfibins(m_in%nfibs) ! aqui guardo la cantidad de intersecciones sobre cada fibra
269    INTEGER :: ifibins(10000, m_in%nfibs) ! aqui guardo en que indice local de cada fibra colocar los segmentos nuevos
           ↪ (i_ins, j_fib)
270    INTEGER :: jfibins(10000, m_in%nfibs) ! aqui guardo el indice global de cada segmento nuevo de cada fibra (i_ins,
           ↪ j_fib)
271    INTEGER :: nsegins(m_in%nsegs) ! cantidad de intersecciones por cada segmento
272    INTEGER :: isegins(100, m_in%nsegs) ! nodo viejo para cada interseccion para cada segmento (i_ins, j_seg)
273    INTEGER :: jsegins(100, m_in%nsegs) ! nodo nuevo para cada interseccion para cada segmento (i_ins, j_seg)
274    INTEGER :: newisegins(m_in%nsegs) ! indice nuevo del ultimo segmento creado debido a una interseccion del segmento
           ↪ i_seg
275    !
276    INTEGER :: i, j
277    INTEGER :: m_in_fie0, m_in_fie1, m_out_fie0, m_out_fie1
278    INTEGER :: nins2
279    INTEGER :: aux1, aux2, aux3, aux4
280    INTEGER :: inews1, inews2, inewn
281    !
282
283
284    ! ----------
285    ! inicializo algunos arrays
286    nins = 0
287    nfibins = 0
288    nsegins = 0
289    DO i=1,m_in%nsegs
290       newisegins(i) = i
291    END DO
292    ! ----------
293    ! recorro las capas e intersecto fibras dentro de cada capa y con capas adyacentes
294    DO c1 = 1,m_in%ncaps
295       ! recorro las fibras de la capa c1
296       DO f1_c1 = 1,m_in%capsne(c1)
297          f1 = m_in%capsje( m_in%capsie(c1)-1 + f1_c1 )
298          ! recorro los segmentos de la fibra f1
299          DO s1_f1 = 1,m_in%fibsne(f1)
300             s1 = m_in%fibsje( m_in%fibsie(f1)-1 + s1_f1 )
301 ! if (s1==1) then
302 ! write(*,*) s1
303 ! end if
304             ! si el segmento s1 ya sufrio una interseccion paso al siguiente
305             n11 = m_in%segs(1,s1)
306             n12 = m_in%segs(2,s1)
307             r11 = m_in%rnods(:,n11)
308             r12 = m_in%rnods(:,n12)
309             ! puedo estar buscando intersecciones dentro de la misma capa
310             ! o entre capas
311             IF (ALLOCATED(fibs1)) DEALLOCATE( fibs1 )
312             IF (label_intercapas) THEN
313                ! recorro las fibras de la capa siguiente
314                ! o sea que si estoy en la ultima capa, no lo hago
315                IF (c1==m_in%ncaps) THEN
316                   if (.not. label_periodicidad) CYCLE ! si activo esto la ultima capa no intersecta con la primera
317                   i1fibs1 = 1
318                   nfibs1 = m_in%capsne(1) ! voy a chequear la ultima capa con la primera
319                   allocate( fibs1(nfibs1) )
320                   fibs1 = m_in%capsje( m_in%capsie(1) : m_in%capsie(1)-1 + nfibs1 )
321                else
322                   i1fibs1 = 1
323                   nfibs1 = m_in%capsne(c1+1)
324                   ALLOCATE( fibs1(nfibs1) )
325                   fibs1 = m_in%capsje( m_in%capsie(c1+1) : m_in%capsie(c1+1)-1 + nfibs1 )
326                END IF
327             ELSE
328                ! recorro las fibras restantes de la capa c1 para intersectar
329                i1fibs1 = f1_c1 + 1
330                nfibs1 = m_in%capsne(c1)
331                ALLOCATE( fibs1(nfibs1) )
```

```
332             fibs1 = m_in%capsje( m_in%capsie(c1) : m_in%capsie(c1)-1 + nfibs1 )
333          END IF
334          DO f2_c1 = i1fibs1,nfibs1
335             f2 = fibs1(f2_c1)
336             ! recorro los segmentos de la fibra f2
337             DO s2_f2 = 1,m_in%fibsne(f2)
338                s2 = m_in%fibsje( m_in%fibsie(f2)-1 + s2_f2 )
339                ! si el segmento s2 ya sufrio una interseccion paso al siguiente
340                IF ( (nsegins(s2)>0) .OR. (nsegins(s1)>0) ) THEN
341                   CYCLE
342                END IF
343                n21 = m_in%segs(1,s2)
344                n22 = m_in%segs(2,s2)
345                r21 = m_in%rnods(:,n21)
346                r22 = m_in%rnods(:,n22)
347                CALL intersectar_segmentos(r11, r12, r21, r22, rin, iStat)
348                IF (iStat == 2) THEN ! 2 es interseccion tipo medio-medio
349                   nins = nins + 1
350                   inewn = m_in%nnods + nins
351                   inews1 = m_in%nsegs + 2*nins - 1
352                   inews2 = inews1 + 1
353                   nsegins(s1) = nsegins(s1) + 1
354                   nsegins(s2) = nsegins(s2) + 1
355                   isegins(nsegins(s1),s1) = inews1
356                   jsegins(nsegins(s1),s1) = inewn
357                   isegins(nsegins(s2),s2) = inews2
358                   jsegins(nsegins(s2),s2) = inewn
359                   nfibins(f1) = nfibins(f1) + 1
360                   nfibins(f2) = nfibins(f2) + 1
361                   ifibins(nfibins(f1),f1) = s1_f1 + 1
362                   ifibins(nfibins(f2),f2) = s2_f2 + 1
363                   jfibins(nfibins(f1),f1) = inews1
364                   jfibins(nfibins(f2),f2) = inews2
365                   ins(nins)%tipo = iStat ! ojo que no es el tipo de nodo, sino el tipo de interseccion (aunque
                          ↪ coincida que es un 2 de tuje)
366                   ins(nins)%fibs = [f1,f2] ! fibras que participan en la interseccion
367                   ins(nins)%segs = [s1,s2] ! segmentos que participan en la interseccion
368                   ins(nins)%segs_f = [s1_f1, s2_f2] ! indices locales de los segmentos dentro de sus fibras
369                   ins(nins)%seg1 = [n11, n12]
370                   ins(nins)%seg2 = [n21, n22]
371                   ins(nins)%inewseg1 = inews1
372                   ins(nins)%inewseg2 = inews2
373                   ins(nins)%inewnod = inewn
374                   ins(nins)%newnodr = rin
375                   newisegins(s1) = inews1 ! en caso de que vuelva a tener alguna interseccion
376                   newisegins(s2) = inews2 ! en caso de que vuelva a tener alguna interseccion
377                END IF
378             END DO
379          END DO
380       END DO
381    END DO
382 END DO
383 ! ----------
384
385 IF (nins==0) THEN
386    iStatus = 1
387    m_out = m_in
388    RETURN
389 END IF
390
391 ! ----------
392 ! aqui ya tengo todas las intersecciones guardadas, ahora tengo que crear la nueva malla
393 ! dimensiones generales
394 m_out%sidelen = m_in%sidelen
395 m_out%diamed = m_in%diamed
396 m_out%volfrac = m_in%volfrac
397 m_out%lenseg = m_in%lenseg
398 m_out%themax = m_in%themax
399 ! ----------
400 ! nodos
```

```
401   m_out%nnods = m_in%nnods + nins ! cada interseccion implica un nuevo nodo
402   ALLOCATE( m_out%tipos(m_out%nnods) ) ! adjudico
403   ALLOCATE( m_out%rnods(2,m_out%nnods) ) ! adjudico
404   m_out%tipos(1:m_in%nnods) = m_in%tipos ! copio los tipos viejos en los tipos nuevos
405   m_out%tipos(m_in%nnods+1:m_out%nnods) = 2 ! los tipos nuevos extras son todos tipo interseccion
406   m_out%rnods(:,1:m_in%nnods) = m_in%rnods ! copio coordenadas de los nodos viejos en los nodos nuevos
407   DO i=1,nins ! copio coordenadas de los nodos interseccion en los nodos nuevos
408      m_out%rnods(:,m_in%nnods+i) = ins(i)%newnodr(:)
409   END DO
410   ! ----------
411   ! segmentos
412   m_out%nsegs = m_in%nsegs + 2*nins ! cada interseccion implica dos nuevos segmentos
413   ALLOCATE( m_out%segs(2,m_out%nsegs) ) ! adjudico
414   m_out%segs(:,1:m_in%nsegs) = m_in%segs ! copio los segmentos viejos en los segmentos nuevos
415   ! (los segmentos quedan incompletos, hay que modificarlos luego en un bucle sobre las intersecciones)
416   DO i=1,nins
417      s1 = ins(i)%segs(1)
418      s2 = ins(i)%segs(2)
419      n11 = m_in%segs(1,s1)
420      n12 = m_in%segs(2,s1)
421      n21 = m_in%segs(1,s2)
422      n22 = m_in%segs(2,s2)
423      inews1 = ins(i)%inewseg1
424      inews2 = ins(i)%inewseg2
425      inewn = ins(i)%inewnod
426      ! cambio la conectividad de los dos segmentos viejos
427      ! y agrego los dos segmentos nuevos
428      m_out%segs(2,s1) = inewn
429      m_out%segs(2,s2) = inewn
430      m_out%segs(:,inews1) = [inewn, n12]
431      m_out%segs(:,inews2) = [inewn, n22]
432   END DO
433   ! ----------
434   ! fibras
435   m_out%nfibs = m_in%nfibs
436   m_out%lenfibsje = m_in%lenfibsje + 2*nins ! tengo dos nuevos elemento en la conectividad de las fibras por cada
          ↪ interseccion
437   ALLOCATE( m_out%fibsds(m_out%nfibs) )
438   ALLOCATE( m_out%fibsdls(m_out%nfibs) )
439   ALLOCATE( m_out%fibsdths(m_out%nfibs) )
440   ALLOCATE( m_out%fibsne(m_out%nfibs) )
441   ALLOCATE( m_out%fibsie(m_out%nfibs + 1) )
442   ALLOCATE( m_out%fibsje(m_out%lenfibsje) )
443   m_out%fibsds = m_in%fibsds
444   m_out%fibsdls = m_in%fibsdls
445   m_out%fibsdths = m_in%fibsdths
446   m_out%fibsie(1) = 1
447   ! primero copio la conectividad vieja en la conectividad nueva
448   ! teniendo cuidado de donde va cada fibra
449   m_out%fibsje = -1 ! inicializo en -1 indicando lugares libres (sin segmento asignado)
450   DO i=1,m_in%nfibs
451      m_out%fibsne(i) = m_in%fibsne(i) + nfibins(i)
452      m_out%fibsie(i+1) = m_out%fibsie(i) + m_out%fibsne(i)
453   END DO
454   !
455   m_out%fibsje(1:m_in%lenfibsje) = m_in%fibsje
456   ! ahora divido cada segmento en dos recorriendo el je de las fibras
457   nins2 = 0
458   DO i=1,m_in%lenfibsje
459      s1 = m_in%fibsje(i)
460      IF (nsegins(s1)>0) THEN
461         ! este segmento fue intersectado, hay que partirlo en dos
462         nins2 = nins2 + 1
463         inews1 = isegins(1,s1) ! nodo nuevo generado al partir al segmento s1 en dos
464         ! le hago lugar al nuevo segmento desplazando la segunda parte de la conectividad hacia adelante
465         m_out%fibsje(i+nins2+1:m_in%lenfibsje+nins2) = m_out%fibsje(i+nins2:m_in%lenfibsje+nins2-1)
466         ! pongo el segmento nuevo
467         m_out%fibsje(i+nins2) = inews1
468      END IF
469   END DO
```

```
    ! ----------
    ! capas
    m_out%ncaps = m_in%ncaps
    m_out%lencapsje = m_in%lencapsje
    ALLOCATE( m_out%capsne(m_out%ncaps) )
    ALLOCATE( m_out%capsie(m_out%ncaps + 1) )
    ALLOCATE( m_out%capsje(m_out%lencapsje) )
    m_out%capsne = m_in%capsne
    m_out%capsie = m_in%capsie
    m_out%capsje = m_in%capsje
    ! ----------

    iStatus = 0


END SUBROUTINE intersectar_fibras
! ================================================================================
! ================================================================================


! ================================================================================
END MODULE class_malla_completa
! ================================================================================
```

```
! ================================================================================
MODULE class_mallita
! ================================================================================
   USE Aux
   USE class_malla_completa, ONLY : MallaCom
   IMPLICIT NONE

   real(8), parameter :: delta = 1.d-4
   real(8), parameter :: delta21 = 1.d0 / (2.d0 * delta)
   real(8), parameter :: deltax(2) = delta * [1.d0, 0.d0]
   real(8), parameter :: deltay(2) = delta * [0.d0, 1.d0]

   !PRIVATE
   !PUBLIC :: MallaCom
   !PUBLIC :: leer_malla

   TYPE MallaSim
      ! diemensiones generales
      REAL(8) :: sidelen
      real(8) :: diamed
      integer :: ncapas
      ! nodos
      INTEGER :: nnods
      INTEGER, ALLOCATABLE :: tipos(:)
      REAL(8), ALLOCATABLE :: rnods0(:,:), rnods(:,:)
      LOGICAL, ALLOCATABLE :: mf(:), mi(:)
      ! fibras
      INTEGER :: nfibs
      INTEGER, ALLOCATABLE :: fibs(:,:)
      REAL(8), ALLOCATABLE :: letes0(:)
      REAL(8), ALLOCATABLE :: lamsr(:)
      real(8), allocatable :: lamps(:)
      logical, allocatable :: brokens(:)
      REAL(8), ALLOCATABLE :: diams(:)
      ! parametros constitutivos
      INTEGER :: nparam
      REAL(8), ALLOCATABLE :: param(:) ! por ahora son los mismos para todas las fibras
      ! informacion de deformacion y tension
      logical :: status_deformed = .false.
      real(8) :: Fmacro(2,2)
      real(8), allocatable :: lams(:)
      real(8), allocatable :: tens(:)
      real(8) :: Tmacro(2,2)
   CONTAINS
      !
   END TYPE MallaSim
```

```
47
48  ! ========================================================================
49  CONTAINS
50  ! ========================================================================
51
52
53  ! ========================================================================
54  ! ========================================================================
55  SUBROUTINE Desde_MallaCom(macom, masim, nparcon, parcon)
56     ! ----------
57     IMPLICIT NONE
58     TYPE(MallaCom), INTENT(IN) :: macom
59     TYPE(MallaSim), INTENT(OUT) :: masim
60     integer, intent(in) :: nparcon
61     real(8), intent(in) :: parcon(nparcon)
62     ! ----------
63     INTEGER :: i,j,k ! contadors
64     INTEGER :: f,s,n,n1,n2 ! indices de fibras, segmentos, nodos
65     INTEGER :: nins_f ! para contar el numero de intersecciones por fibra
66     integer :: nfibs ! numero de fibras en la malla simplificada
67     integer :: nnods ! numero de nodos en la malla simplificada
68     integer :: ifib, inod
69     integer, allocatable :: oldnods(:) ! array para saber como se conectan los nodos de macom y de masim
70     real(8) :: r1(2), r2(2), dr(2), loco0_s, loco0, lete0
71     logical :: cond
72     real(8) :: L_2
73     ! ----------
74
75     ! ----------
76     ! parametros
77     masim%sidelen = macom%sidelen
78     masim%diamed = macom%diamed
79     masim%ncapas = macom%ncaps
80     ! ----------
81     write(*,*) "Contando␣el␣numero␣de␣nodos␣y␣fibras␣en␣la␣malla␣simplificada"
82     ! primero voy a contar el numero de intersecciones en cada fibra de la malla completa
83     ! para saber el numero de fibras y de nodos en la malla simplificada
84     nfibs = 0
85     nnods = 0
86     allocate( oldnods(macom%nnods) ) ! aca voy chequeando si a cada nodo ya lo considere o si es una nueva interseccion
87     oldnods = 0
88     do f=1,macom%nfibs
89        nins_f = 0
90        ! el primer nodo de cada fibra siempre es un nodo nuevo de masim
91        nnods = nnods + 1
92        do j=1,macom%fibsne(f)-1 ! el ultimo no lo tengo en cuenta porque el ultimo nodo siempre es frontera, nunca
              ↪ interseccion
93           s = macom%fibsje(macom%fibsie(f)-1+j)
94           n1 = macom%segs(1,s)
95           n2 = macom%segs(2,s)
96           if (macom%tipos(n2) == 2) THEN
97              nins_f = nins_f + 1
98              ! chequeo si ya esta considerado o aun no
99              if (oldnods(n2)==0) then
100                ! este nodo no lo sume aun a los nodos de masim
101                oldnods(n2) = 1
102                nnods = nnods + 1
103 ! else
104 ! ! este nodo ya lo tuve en cuenta, no hace falta hacer nada
105             end if
106          end if
107       end do
108       ! sumo las fibras de masim que salen de esta fibra de macom segun las intersecciones que tuvo
109       nfibs = nfibs + 1 + nins_f
110       ! y sumo el ultimo nodo de la fibra que siempre es un nodo nuevo de masim
111       nnods = nnods + 1
112    end do
113    write(*,*) "nnods:␣", nnods, "␣nfibs:␣", nfibs
114    ! ----------
115    ! ahora que se cuantas fibras y nodos voy a tener, adjudico memoria
```

```
116     masim%nnods = nnods
117     masim%nfibs = nfibs
118     allocate( masim%tipos(nnods) )
119     allocate( masim%rnods0(2,nnods) )
120     allocate( masim%mf(nnods) )
121     allocate( masim%mi(nnods) )
122     allocate( masim%fibs(2,nfibs) )
123     allocate( masim%letes0(nfibs) )
124     allocate( masim%lamsr(nfibs) )
125     allocate( masim%lamps(nfibs) )
126     allocate( masim%brokens(nfibs) )
127     allocate( masim%diams(nfibs) )
128  ! allocate( masim%param(10,nfibs) )
129     ! ----------
130     write(*,*) "Construyendo malla simplificada"
131     ! ahora voy llenando los valores
132     ifib = 0
133     inod = 0
134     oldnods = 0
135     do f=1,macom%nfibs
136        ! siempre que empieza una fibra de macom, empieza una nueva fibra de masim
137        ifib = ifib + 1
138        loco0 = 0.d0
139        ! identifico el primer nodo de la fibra antes de hacer un loop por los demas segmentos
140        s = macom%fibsje(macom%fibsie(f))
141        n1 = macom%segs(1,s)
142        ! este nodo siempre es un nodo nuevo de masim, entonces
143        ! sumo este nodo a los nodos de masim y a la conectividad de la nueva fibra de masim
144        inod = inod + 1
145        masim%tipos(inod) = 1 ! frontera
146        masim%rnods0(:,inod) = macom%rnods(:,n1)
147        oldnods(n1) = inod ! guardo el indice que este nodo tendra en masim
148        masim%fibs(1,ifib) = inod
149        masim%diams(ifib) = macom%fibsds(f)
150        ! ahora recorro todos los segmentos para ir chequeando todos los n1, asi comprendo todos los nodos de la fibra
151        do j=1,macom%fibsne(f)
152           ! identifico el segmento y sus nodos en macom
153           s = macom%fibsje(macom%fibsie(f)-1+j)
154           n1 = macom%segs(1,s)
155           n2 = macom%segs(2,s)
156           ! calculo long de seg para sumar a la long de contorno (loco0)
157           r1 = macom%rnods(:,n1)
158           r2 = macom%rnods(:,n2)
159           dr = r2-r1
160           loco0_s = dsqrt(sum(dr*dr))
161           cond = iguales(loco0_s,0.d0)
162           if (cond) then
163              write(*,*) "longitud de segmento nula"
164              stop
165           end if
166           loco0 = loco0 + loco0_s
167           ! me fijo si llegue al final de la fibra de masim (si encuentro una interseccion o una frontera)
168           if ((macom%tipos(n2) == 2).or.(macom%tipos(n2)==1)) THEN
169              ! me tengo que fijar si este nodo que encontre ya lo tengo presente en oldnods
170              if (oldnods(n2)==0) then
171                 ! si no esta presente tengo que agregarlo a los nodos de masim
172                 inod = inod + 1 !nuevo nodo de masim
173                 masim%tipos(inod) = macom%tipos(n2) ! interseccion
174                 masim%rnods0(:,inod) = macom%rnods(:,n2)
175                 oldnods(n2) = inod
176              end if
177              ! estuviera ya presente o no, lo agrego a la conectividad de la fibra
178              masim%fibs(2,ifib) = oldnods(n2)
179              ! calculo la longitud extremo-extremo de la fibra sim (lete0)
180              r1 = masim%rnods0(:,masim%fibs(1,ifib))
181              r2 = masim%rnods0(:,masim%fibs(2,ifib))
182              dr = r2-r1
183              lete0 = dsqrt(sum(dr*dr))
184              masim%letes0(ifib) = lete0
185              masim%lamsr(ifib) = loco0 / lete0
```

```
            ! si no llegue a un nodo frontera, comienzo una nueva fibra
            if (macom%tipos(n2)==2) then
               ifib = ifib + 1
               loco0 = 0.d0
               masim%fibs(1,ifib) = oldnods(n2)
               masim%diams(ifib) = macom%fibsds(f)
            end if
         end if
      end do
      nfibs = nfibs + 1 + nins_f
      nnods = nnods + 2 + nins_f
   end do
   ! tambien agrego informacion sobre deformacion plastica y rotura de fibras
   masim%lamps = 1.d0 ! sin plasticidad
   masim%brokens = .false. ! todas sanas
   ! ----------
   ! ya tengo los nodos (tipos y coordenadas), ahora hago las masks
   masim%mf = masim%tipos == 1
   masim%mi = masim%tipos == 2
   ! Desplazo las coordenadas para dejar el cero en el medio del rve
   L_2 = 0.5d0 * masim%sidelen
   masim%rnods0 = masim%rnods0 - L_2
   ! y copio rnods0 a rnods
   allocate( masim%rnods(2,nnods) )
   masim%rnods = masim%rnods0
   ! ----------
   ! parametros constitutivos
   allocate( masim%param(nparcon) )
   masim%nparam = nparcon
   masim%param = parcon
   ! ----------
   ! ----------
   write(*,*) "Malla simplificada lista"

   ! ----------
END SUBROUTINE Desde_MallaCom
! ================================================================================
! ================================================================================


! ================================================================================
! ================================================================================
SUBROUTINE leer_mallita(masim, nomarch, numparamcon, paramcon)
   IMPLICIT NONE
   CHARACTER(LEN=120), INTENT(IN) :: nomarch
   TYPE(MallaSim), INTENT(OUT) :: masim
   integer, intent(in) :: numparamcon
   real(8), intent(in) :: paramcon(numparamcon)
   !
   INTEGER :: fid
   integer :: iStatus
   INTEGER :: i,j,k
   integer :: tipo
   real(8) :: r0(2), r(2)
   real(8) :: diam, lete0, lamr, lamp
   logical :: broken
   integer :: n0, n1
   real(8) :: L_2
   ! ----------
   fid = get_file_unit()
   OPEN(UNIT=fid, FILE=TRIM(nomarch), STATUS="OLD")
   ! ----------
   ! Parametros
   iStatus = FindStringInFile("*parametros", fid, .TRUE.)
   READ(fid,*) masim%sidelen
   READ(fid,*) masim%diamed
   read(fid,*) masim%ncapas
   if (.false.) then ! Codigo viejo que estoy reemplazando
      READ(fid,*) masim%nparam
      allocate( masim%param(masim%nparam) )
```

```
256       read(fid,*) masim%param
257    end if
258    masim%nparam = numparamcon
259    allocate( masim%param(masim%nparam) )
260    masim%param = paramcon
261    ! ----------
262    ! Deformacion (puede no estar)
263    iStatus = FindStringInFile("*deformacion", fid, .false.)
264    if (iStatus == 0) then
265       masim%status_deformed = .true.
266       read(fid,*) masim%Fmacro
267       read(fid,*) masim%Tmacro
268    end if
269    ! ----------
270    ! Nodos
271    iStatus = FindStringInFile("*coordenadas", fid, .TRUE.)
272    READ(fid,*) masim%nnods
273    ALLOCATE( masim%rnods0(2,masim%nnods) )
274    allocate( masim%rnods(2,masim%nnods) )
275    ALLOCATE( masim%tipos(masim%nnods) )
276    DO i=1,masim%nnods
277       READ(fid,*) j, tipo, r0, r
278       masim%rnods0(:,i) = r0
279       masim%rnods(:,i) = r
280       masim%tipos(i) = tipo
281    END DO
282    ! Cambio coordenadas para tener el cero en el centro del RVE
283    ! OJO por ahora si hay distorsion (F12 o F21) esto no anda tan facil
284    L_2 = 0.5d0 * masim%sidelen
285    masim%rnods0 = masim%rnods0 - L_2
286    if (masim%status_deformed) then
287       L_2 = 0.5d0 * masim%sidelen * masim%Fmacro(1,1)
288       masim%rnods(1,:) = masim%rnods(1,:) - L_2
289       L_2 = 0.5d0 * masim%sidelen * masim%Fmacro(2,2)
290       masim%rnods(2,:) = masim%rnods(2,:) - L_2
291    else
292       masim%rnods = masim%rnods0
293    end if
294    ! ----------
295    ! Fibras
296    iStatus = FindStringInFile("*fibras", fid, .TRUE.)
297    READ(fid,*) masim%nfibs
298    allocate( masim%fibs(2,masim%nfibs) )
299    allocate( masim%diams(masim%nfibs) )
300    allocate( masim%letes0(masim%nfibs) )
301    allocate( masim%lamsr(masim%nfibs) )
302    allocate( masim%lamps(masim%nfibs) )
303    allocate( masim%brokens(masim%nfibs) )
304    DO i=1,masim%nfibs
305       READ(fid,*) j, diam, lete0, lamr, lamp, broken, n0, n1
306       masim%diams(i) = diam
307       masim%letes0(i) = lete0
308       masim%lamsr(i) = lamr
309       masim%lamps(i) = lamp
310       masim%brokens(i) = broken
311       masim%fibs(1,i) = n0+1 !+1 porque vengo de python que tiene base 0
312       masim%fibs(2,i) = n1+1
313    END DO
314    ! ----------
315    CLOSE(fid)
316    ! ----------
317
318
319 END SUBROUTINE leer_mallita
320 ! ================================================================================
321 ! ================================================================================
322
323
324
325 ! ================================================================================
```

```
326 ! ========================================================================
327 SUBROUTINE escribir_mallita(masim, nomarch)
328    IMPLICIT NONE
329    CHARACTER(LEN=120), INTENT(IN) :: nomarch
330    TYPE(MallaSim), INTENT(IN) :: masim
331    !
332    INTEGER :: fid
333    INTEGER :: i
334    CHARACTER(LEN=120) :: formato
335    real(8) :: L_2, r0(2,masim%nnods), r1(2,masim%nnods)
336    ! ----------
337    fid = get_file_unit()
338    OPEN(UNIT=fid, FILE=TRIM(nomarch), STATUS="REPLACE")
339    ! ----------
340    ! Parametros
341    WRITE(fid,'(A11)') "*Parametros"
342    WRITE(fid,'(E20.8E4)') masim%sidelen
343    WRITE(fid,'(E20.8E4)') masim%diamed
344    WRITE(fid,'(I10)') masim%ncapas
345    WRITE(fid,'(I10)') masim%nparam
346    WRITE(formato,'(A1,I0,A8)') "(", masim%nparam, "E20.8E4)"
347    WRITE(fid,formato) masim%param
348    ! ----------
349    ! Deformacion
350    if (masim%status_deformed) then
351       write(fid,'(A12)') "*Deformacion"
352       WRITE(fid,'(4E20.8E4)') masim%Fmacro
353       WRITE(fid,'(4E20.8E4)') masim%Tmacro
354    end if
355    ! ----------
356    ! Nodos
357    WRITE(fid,'(A12)') "*Coordenadas"
358    WRITE(fid,'(I12)') masim%nnods
359    DO i=1,masim%nnods
360       ! Tengo que escribir las coordenadas con el cero en el vertice inferior izquierdo
361       L_2 = 0.5d0 * masim%sidelen
362       r0 = masim%rnods0 + L_2
363       if (masim%status_deformed) then
364          L_2 = 0.5d0 * masim%sidelen * masim%Fmacro(1,1)
365          r1(1,:) = masim%rnods(1,:) + L_2
366          L_2 = 0.5d0 * masim%sidelen * masim%Fmacro(2,2)
367          r1(2,:) = masim%rnods(2,:) + L_2
368       else
369          r1 = r0
370       end if
371       WRITE(fid, '(I12,I2,4E20.8E4)') i-1, masim%tipos(i), r0(:,i), r1(:,i)
372    END DO
373    ! ----------
374    ! Fibras
375    WRITE(fid,'(A7)') "*Fibras"
376    WRITE(fid, '(I12)') masim%nfibs
377    DO i=1,masim%nfibs
378       WRITE(fid,'(I12,␣4E20.8E4,␣L20␣,2I12)') i-1, masim%diams(i), masim%letes0(i), masim%lamsr(i), masim%lamps(i),
             ↪ masim%brokens(i), masim%fibs(:,i) - 1
379    END DO
380    ! ----------
381    CLOSE(fid)
382    ! ----------
383
384
385 END SUBROUTINE escribir_mallita
386 ! ========================================================================
387 ! ========================================================================
388
389
390
391 ! ========================================================================
392 ! ========================================================================
393 subroutine deformar_afin(masim, FF, r1, axis)
394    ! Deforma la mallita de manera Afin: r = F r0
```

```
395     ! input: FF (tensor gradiente de deformaciones 2x2)
396     !
397     implicit none
398     ! ----------
399     type(MallaSim), intent(inout) :: masim
400     real(8), intent(in) :: FF(2,2) ! tensor de deformaciones
401     real(8), intent(inout) :: r1(2,masim%nnods)
402     integer, optional, intent(in) :: axis ! eje que se deforma afin: 0= los dos, 1= x, 2=y
403     ! ----------
404     integer :: i,j,k
405     integer :: axisL
406     ! ----------
407
408     ! Chequeo si se da eje
409     axisL = 0
410     if (present(axis)) axisL=axis
411
412
413     ! ----------
414     if (axisL==0) then
415        ! Deformo de manera afin ambos ejes
416        do k=1,masim%nnods
417           r1(:,k) = 0.d0 ! lo hago cero antes de empezar la sumatoria
418           do i=1,2
419              do j=1,2
420                 r1(i,k) = r1(i,k) + FF(i,j)*masim%rnods0(j,k)
421              end do
422           end do
423        end do
424     else if (axisL==1) then
425        ! Solamente deformo de manera afin en x (y queda igual)
426        do k=1,masim%nnods
427           i=1
428           r1(i,k) = 0.d0
429           do j=1,2
430              r1(i,k) = r1(i,k) + FF(i,j)*masim%rnods0(j,k)
431           end do
432        end do
433     else if (axisL==2) then
434        ! Solamente deformo de manera afin en y (x queda igual)
435        do k=1,masim%nnods
436           i=2
437           r1(i,k) = 0.d0
438           do j=1,2
439              r1(i,k) = r1(i,k) + FF(i,j)*masim%rnods0(j,k)
440           end do
441        end do
442     else
443        print *, "Mal␣valor␣de␣axisL"
444        stop
445     end if
446     ! ----------
447  ! masim%rnods = r1
448     ! ----------
449
450     ! ----------
451  end subroutine deformar_afin
452  ! ================================================================================
453  ! ================================================================================
454
455
456  ! ================================================================================
457  ! ================================================================================
458  subroutine deformar_afin_frontera(masim, FF, r1)
459     ! Toma un array de coordenadas r1
460     ! Y a los nodos de la frontera le aplica la deformacion afin
461     ! Al resto los deja como estaban
462     ! input: FF (tensor gradiente de deformaciones 2x2)
463     !
464     implicit none
```

```
465     ! ----------
466     type(MallaSim), intent(inout) :: masim
467     real(8), intent(in) :: FF(2,2) ! tensor de deformaciones
468     real(8), intent(inout) :: r1(2,masim%nnods)
469     ! ----------
470     integer :: i,j,k
471     ! ----------
472
473     ! ----------
474     do k=1,masim%nnods
475        if (masim%tipos(k)==1) then
476           r1(:,k) = 0.d0
477           do i=1,2
478              do j=1,2
479                 r1(i,k) = r1(i,k) + FF(i,j)*masim%rnods0(j,k)
480              end do
481           end do
482        end if
483     end do
484     ! ----------
485  ! masim%rnods = r1
486     ! ----------
487
488     ! ----------
489  end subroutine deformar_afin_frontera
490  ! ================================================================================
491  ! ================================================================================
492
493
494  ! ================================================================================
495  ! ================================================================================
496  subroutine calcular_tension_fibra(masim, f, dr_f, tension, fuerzav)
497     ! Calcula la tension ingenieril de una fibra (identificada mediante numeracion f)
498     ! y la fuerza como vector (yendo del nodo inicial de la fibra al nodo final)
499     ! dr_f es el vector de la fibra desde r_n0 hasta r_n1 (desde nodo inicial hasta nodo final)
500     ! ----------
501     implicit none
502     type(MallaSim), intent(in) :: masim
503     integer, intent(in) :: f
504     real(8), intent(in) :: dr_f(2)
505     real(8), intent(out) :: tension ! tension ingenieril de la fibra (escalar, >0 elongacion, <0 compresion)
506     real(8), intent(out) :: fuerzav(2) ! fuerza de la fibra, vector desde nodo inicial hacia nodo final
507     ! ----------
508     integer :: selector
509     integer :: last
510     ! ---
511     real(8) :: k1 ! Constante elastica (en fuerza) de fibra recta a la traccion
512     real(8) :: k2 ! Constante elastica (en fuerza) de fibra enrulada
513     real(8) :: Et ! Modulo elastico de fibra recta a la traccion
514     real(8) :: Eb_Et ! Eb/Et
515     real(8) :: Eb ! Modulo elastico de fibra enrulada
516     real(8) :: Ep_Et ! Ep/Et
517     real(8) :: Ep ! Modulo elastico para simular plasticidad
518     real(8) :: lamp0_lamr ! lamp0/lamr
519     real(8) :: lamp0 ! lamp0 = valor de lam al cual empezaria la plasticidad
520     real(8) :: diam ! diametro inicial de la fibra
521     real(8) :: lete0 ! longitud extremo-extremo inicial de la fibra
522     real(8) :: lamr ! valor de reclutamiento de la fibra
523     real(8) :: lamp
524     real(8) :: lamrp ! valor de reclutamiento aumentado por la plasticidad
525     logical :: broken
526     real(8) :: area ! area inicial de la fibra (seccion transversal)
527     real(8) :: lete ! longitud extremo-extremo actual
528     real(8) :: lam ! elongacion extremo-extremo = lete/lete0
529     real(8) :: fuerza ! fuerza actual (modulo) de la fibra
530     real(8) :: fuerzar ! fuerza de la fibra para la cual se recluta
531     real(8) :: tensionr ! tension de la fibra para la cual se recluta
532     real(8) :: tensiony ! tension de la fibra para la cual plastifica
533     ! ----------
534
```

```
535    ! ----------
536    ! Por ahora tengo un param con 3 valores: selector, k1 o Et, k2 o Eb
537    selector = nint( masim%param(1) )
538    last = 1
539    select case (selector)
540
541    case (1) ! se dan k1 y k2 para calcular la fuerza directamente
542       k1 = masim%param(last+1)
543       k2 = masim%param(last+2)
544       diam = masim%diams(f)
545       area = pi * diam * diam / 4.d0
546       lete0 = masim%letes0(f)
547       lamr = masim%lamsr(f)
548       lete = dsqrt(sum(dr_f*dr_f))
549       lam = lete / lete0
550       if ( lam <= lamr ) then
551          fuerza = k2*(lam - 1.0d0)
552       else
553          fuerzar = k2 * (lamr - 1.0d0)
554          fuerza = fuerzar + k1 * (lam/lamr - 1.0d0)
555       end if
556       tension = fuerza / area
557       fuerzav = fuerza * dr_f / lete
558    case (2) ! se dan Et y Eb para calcular la tension ingenieril
559       Et = masim%param(last+1)
560       Eb_Et = masim%param(last+2)
561       Eb = Eb_Et*Et
562       lamr = masim%lamsr(f)
563       diam = masim%diams(f)
564       area = pi * diam * diam / 4.d0
565       lete0 = masim%letes0(f)
566       lete = dsqrt(sum(dr_f*dr_f))
567       lam = lete / lete0
568       if ( lam <= lamr ) then
569          tension = Eb*(lam - 1.0d0) * diam
570       else
571          tensionr = Eb * (lamr - 1.0d0)
572          tension = tensionr + Et * (lam/lamr - 1.0d0)
573       end if
574       fuerza = tension * area
575       fuerzav = fuerza * dr_f / lete
576    case (3) ! doy Et, Eb y Ep, es una ley trilineal para simular a groso modo una respuesta con deformacion plastica
577       Et = masim%param(last+1)
578       Eb_Et = masim%param(last+2)
579       Eb = Eb_Et*Et
580       Ep_Et = masim%param(last+3)
581       Ep = Ep_Et*Et
582       lamp0_lamr = masim%param(last+4)
583       lamr = masim%lamsr(f)
584       lamp0 = lamp0_lamr*lamr
585       diam = masim%diams(f)
586       area = pi * diam * diam / 4.d0
587       lete0 = masim%letes0(f)
588       lete = dsqrt(sum(dr_f*dr_f))
589       lam = lete / lete0
590       if ( lam <= lamr ) then
591          tension = Eb*(lam - 1.0d0) * diam
592       else if (lam <= lamp0 ) then
593          tensionr = Eb * (lamr - 1.0d0)
594          tension = tensionr + Et * (lam/lamr - 1.0d0)
595       else ! simulando plasticidad en la respuesta
596          tensionr = Eb * (lamr - 1.0d0)
597          tensiony = tensionr + Et * (lamp0/lamr - 1.0d0)
598          tension = tensionr + tensiony + Ep*(lam/lamp0 - 1.0d0)
599       end if
600       fuerza = tension * area
601       fuerzav = fuerza * dr_f / lete
602    case (4) ! Ecuacion con plasticidad
603       Et = masim%param(last+1)
604       Eb_Et = masim%param(last+2)
```

```
605         Eb = Eb_Et*Et
606         lamr = masim%lamsr(f)
607         lamp = masim%lamps(f)
608         lamrp = lamr*lamp
609         broken = masim%brokens(f)
610         diam = masim%diams(f)
611         area = pi * diam * diam / 4.d0
612         lete0 = masim%letes0(f)
613         lete = dsqrt(sum(dr_f*dr_f))
614         lam = lete / lete0
615         ! Ahora si calculo la tension
616         if (broken) then
617            tension = 0.d0
618         elseif ( lam <= lamrp ) then
619            tension = Eb*(lam/lamp - 1.0d0) * diam
620         else
621            tensionr = Eb * (lamrp - 1.0d0)
622            tension = tensionr + Et * (lam/lamrp - 1.0d0)
623         end if
624         fuerza = tension * area
625         fuerzav = fuerza * dr_f / lete
626      case default
627         write(*,*) "Ley␣constitutiva␣desconocida,␣selector:␣", selector
628      end select
629
630      ! ----------
631
632      ! ----------
633   end subroutine calcular_tension_fibra
634   ! ================================================================================
635   ! ================================================================================
636
637
638   ! ================================================================================
639   ! ================================================================================
640   subroutine calcular_tensiones_fibras(masim, r1, lams_fibs, tens_fibs)
641      ! calcula las fuerzas de las fibras y las fuerzas nodales (resultantes)
642      ! se calculan las fuerzas como vectores, entonces las dimensiones son (2,nfibs) y (2,nnods)
643      implicit none
644      type(MallaSim), intent(in) :: masim
645      real(8), intent(in) :: r1(2,masim%nnods)
646      real(8), intent(out) :: lams_fibs(masim%nfibs)
647      real(8), intent(out) :: tens_fibs(masim%nfibs)
648      integer :: f
649      integer :: n1f, n2f
650      real(8) :: rn1f(2), rn2f(2), drf(2)
651      real(8) :: lamf, tenf, fzafv(2)
652
653      do f=1,masim%nfibs
654         ! nodos de la fibra f
655         n1f = masim%fibs(1,f)
656         n2f = masim%fibs(2,f)
657         ! posiciones de esos nodos
658         rn1f = r1(:,n1f)
659         rn2f = r1(:,n2f)
660         ! vector fibra
661         drf = rn2f - rn1f
662         ! fuerza
663         call calcular_tension_fibra(masim, f, drf, tenf, fzafv)
664         tens_fibs(f) = tenf
665         ! elongacion
666         lamf = dsqrt(sum(drf*drf)) / masim%letes0(f)
667         lams_fibs(f) = lamf
668      end do
669
670   end subroutine calcular_tensiones_fibras
671   ! ================================================================================
672   ! ================================================================================
673
674
```

```
675 ! ================================================================================
676 ! ================================================================================
677 subroutine calcular_fuerzas(masim, r1, fzas_fibs, fzas_nods)
678    ! calcula las fuerzas de las fibras y las fuerzas nodales (resultantes)
679    ! se calculan las fuerzas como vectores, entonces las dimensiones son (2,nfibs) y (2,nnods)
680    implicit none
681    type(MallaSim), intent(in) :: masim
682    real(8), intent(inout) :: r1(2,masim%nnods)
683    real(8), intent(out) :: fzas_fibs(2,masim%nfibs)
684    real(8), intent(out) :: fzas_nods(2,masim%nnods)
685    integer :: f
686    integer :: n1f, n2f
687    real(8) :: rn1f(2), rn2f(2), drf(2)
688    real(8) :: tenf, fzafv(2)
689 
690    fzas_nods = 0.d0
691    do f=1,masim%nfibs
692       ! nodos de la fibra f
693       n1f = masim%fibs(1,f)
694       n2f = masim%fibs(2,f)
695       ! posiciones de esos nodos
696       rn1f = r1(:,n1f)
697       rn2f = r1(:,n2f)
698       ! vector fibra
699       drf = rn2f - rn1f
700       ! fuerza
701       call calcular_tension_fibra(masim, f, drf, tenf, fzafv)
702       fzas_fibs(:,f) = fzafv
703       fzas_nods(:,n1f) = fzas_nods(:,n1f) + fzafv
704       fzas_nods(:,n2f) = fzas_nods(:,n2f) - fzafv
705    end do
706 
707 end subroutine calcular_fuerzas
708 ! ================================================================================
709 ! ================================================================================
710 
711 
712 ! ================================================================================
713 ! ================================================================================
714 subroutine desplazar_nodos_malla(masim, nveces, drmag, fza_ref, fza_tol, r1, balanced)
715    ! Mueve los nodos nveces segun la direccion de la resultante
716    ! Todos los nodos se mueven segun el valor drmag
717    ! Cada vez se calculan las fuerzas de las fibras y la resultante
718    ! Entonces la malla va "vibrando" hasta su posicion de equilibrio
719    ! ----------
720    implicit none
721    ! ----------
722    type(MallaSim), intent(inout) :: masim
723    integer, intent(in) :: nveces
724    real(8), intent(in) :: drmag
725    real(8), intent(in) :: fza_ref
726    real(8), intent(in) :: fza_tol
727    real(8), intent(inout) :: r1(2,masim%nnods)
728    logical, intent(out) :: balanced(masim%nnods)
729    ! ----------
730    real(8) :: fzas_fibs(2,masim%nfibs)
731    real(8) :: fzas_nods(2,masim%nnods)
732    real(8) :: fzas_nods_mags(masim%nnods)
733    integer :: vez, n
734    real(8) :: fza_n(2), fza_n_mag, dr_n(2)
735    ! ----------
736 
737    balanced = .false.
738    do vez=1,nveces
739       call calcular_fuerzas(masim, r1, fzas_fibs, fzas_nods)
740       do n=1,masim%nnods
741          if (masim%tipos(n) == 1) then
742             ! nodo de frontera = nodo de Dirichlet
743             r1(:,n) = r1(:,n) ! sentencia innecesaria pero por ahora la dejo para que quede claro
744             fzas_nods_mags(n) = 0.d0
```

```
745             balanced(n) = .true.
746          else
747             fza_n = fzas_nods(:,n)
748             fza_n_mag = dsqrt(sum(fza_n*fza_n))
749             fzas_nods_mags(n) = fza_n_mag
750             if (fza_n_mag < fza_tol) then
751                dr_n = 0.d0
752                balanced(n) = .true.
753             else
754                dr_n = drmag * fza_n / fza_ref
755                balanced(n) = .false.
756             end if
757             r1(:,n) = r1(:,n) + dr_n(:)
758          end if
759       end do
760       if ( all(balanced) ) then
761          write(*,*) "balanced!", vez
762          exit
763       end if
764    end do
765    if ( .not. all(balanced) ) then
766       write(*,*) "maxres:␣", maxval( fzas_nods_mags )
767    end if
768
769    ! ----------
770 end subroutine desplazar_nodos_malla
771 ! ================================================================================
772 ! ================================================================================
773
774
775 ! ================================================================================
776 ! ================================================================================
777 subroutine calcular_equilibrio(masim, npasos, vec_veces, vec_drmags, f_ref, f_tol, Fmacro)
778    ! Forma de calcular el equilibrio moviendo los nodos de forma iterativa segun
779    ! la direccion de la resultante en un valor prescripto de desplazamiento (drmag)
780    ! Se hace de forma progresiva empezando con drmag grande y terminando con drmag chico
781    ! ----------
782    implicit none
783    ! ----------
784    type(MallaSim), intent(inout) :: masim
785    integer, intent(in) :: npasos
786    integer, intent(in) :: vec_veces(npasos)
787    real(8), intent(in) :: vec_drmags(npasos)
788    real(8), intent(in) :: f_ref
789    real(8), intent(in) :: f_tol
790    real(8), intent(in) :: Fmacro(2,2)
791    ! ----------
792    real(8) :: r1(2,masim%nnods)
793    integer :: paso
794    integer :: nveces
795    real(8) :: drmag
796    ! ----------
797    real(8) :: lams_fibras(masim%nfibs)
798    real(8) :: tens_fibras(masim%nfibs)
799    real(8) :: Tmacro(2,2)
800    logical :: equilibrio(masim%nnods)
801    ! ----------
802
803    ! deformo afin
804    if (npasos==0) then
805       ! Deformo toda la malla afin antes de comenzar la vibracion
806       call deformar_afin(masim, Fmacro, r1)
807    else
808       ! Deformo solamente la frontera de forma afin
809       r1 = masim%rnods
810       call deformar_afin_frontera(masim, Fmacro, r1)
811 ! call deformar_afin(masim, Fmacro, r1, axis=2)
812    end if
813    ! Comienzo a mover los nodos segun las resultantes nodales
814    do paso=1,npasos
```

```
815       nveces = vec_veces(paso)
816       drmag = vec_drmags(paso)
817       call desplazar_nodos_malla(masim, nveces, drmag, f_ref, f_tol, r1, equilibrio)
818       ! chequeo equilibrio
819       if ( all(equilibrio) ) then
820          exit
821       end if
822    end do
823    ! recalculo el equilibrio para tener las tensiones
824    call calcular_tensiones_fibras(masim, r1, lams_fibras, tens_fibras)
825    call homogeneizacion(masim, r1, Tmacro)
826    ! guardo la deformacion en la malla
827    masim%rnods = r1
828    masim%status_deformed = .true.
829    masim%Fmacro = Fmacro
830    masim%lams = lams_fibras
831    masim%tens = tens_fibras
832    masim%Tmacro = Tmacro
833    ! y voila
834
835    ! ----------
836 end subroutine calcular_equilibrio
837 ! ================================================================================
838 ! ================================================================================
839
840
841 ! ================================================================================
842 ! ================================================================================
843 subroutine calcular_plasticidad_rotura(masim, dt)
844    ! Ante una deformacion dada (masim%rnods) calcula la deformacion plastica
845    ! que sufren las fibras. Son dos valores:
846    ! dotlamp = tasa de deformacion plastica
847    ! broken = indicador de si la fibra se rompe o no
848    ! ----------
849    implicit none
850    ! ----------
851    type(MallaSim), intent(inout) :: masim
852    real(8), intent(in) :: dt
853    ! ----------
854    logical :: brokens(masim%nfibs)
855    real(8) :: dotlamps(masim%nfibs)
856    ! parametros constitutivos de plasticidad y rotura
857    real(8) :: doteps0 ! parametro dimensional proporcional de plasticidad
858    real(8) :: s0 ! resistencia a la fluencia inicial
859    real(8) :: nhard ! parametro de endurecimiento por deformacion plastica
860    real(8) :: elonlimit ! limite de rotura por elongacion
861    ! resto
862    integer :: f
863    real(8) :: d_f
864    real(8) :: Area_f
865    real(8) :: lete0_f
866    real(8) :: Vol_f
867    integer :: n1_f, n2_f
868    real(8) :: r_n1_f(2), r_n2_f(2)
869    real(8) :: dr_f(2)
870    real(8) :: lete_f
871    real(8) :: lam_f
872    real(8) :: a_f(2)
873    real(8) :: ten_f, vfza_f(2)
874    real(8) :: lamp_f
875    real(8) :: lamr_f
876    real(8) :: lamef_f ! es un valor de elongacion efectiva, o elongacion elastica (la que va con la tension) = lam/
           ↪ lamr/lamp
877    ! ----------
878    real(8) :: s_f
879    real(8) :: dotlamp_f
880    logical :: broken_f
881    ! ----------
882
883    ! Parametros constitutivos importantes para plasticidad y rotura
```

```
884    doteps0 = masim%param(4)
885    s0 = masim%param(5)
886    nhard = masim%param(6)
887    elonlimit = masim%param(7)
888    ! Bucle por las fibras para ir calculando tensiones actuales
889    do f=1,masim%nfibs
890       ! Guardo informacion en variables mas sencillas
891       d_f = masim%diams(f) ! Diametro inicial de la fibra f
892       Area_f = pi * d_f * d_f / 4.d0 ! Seccion transversal inicial de la fibra f
893       lete0_f = masim%letes0(f) ! Longitud extremo-extremo inicial de la fibra f
894       Vol_f = Area_f * lete0_f ! Volumen de la fibra f
895       n1_f = masim%fibs(1,f) ! Nodo inicial de la fibra f
896       n2_f = masim%fibs(2,f) ! Nodo final de la fibra f
897       r_n1_f = masim%rnods(:,n1_f) ! Posicion actual de n1_f
898       r_n2_f = masim%rnods(:,n2_f) ! Posicion actual de n2_f
899       dr_f = r_n2_f - r_n1_f ! Vector fibra: apunta de nodo inicial a nodo final y su magnitud es la longitud actual
900       lete_f = dsqrt(sum(dr_f*dr_f)) ! Longitud extremo-extremo actual
901       lam_f = lete_f / lete0_f ! Elongacion extremo-extremo
902       lamr_f = masim%lamsr(f)
903       a_f = dr_f/lete_f * lam_f ! Vector orientacion de la fibra, su magnitud es la elongacion lam_f
904       ! Calculo las tensiones
905       call calcular_tension_fibra(masim, f, dr_f, ten_f, vfza_f)
906       ! Calculo la plasticidad
907       broken_f = masim%brokens(f)
908       lamp_f = masim%lamps(f)
909       lamef_f = lam_f / lamp_f / lamr_f
910       if (broken_f) then
911          ! esta fibra esta rota y no deforma mas, su plasticidad queda constante
912          dotlamp_f = 0.d0
913       else
914          ! esta fibra puede romperse o plastificar
915          if (ten_f>elonlimit) then
916             ! se rompe
917             broken_f = .true.
918             dotlamp_f = 0.d0
919          else
920             ! plastifica poco o mucho
921             s_f = s0 * lamp_f**nhard
922             dotlamp_f = doteps0 * dsinh(ten_f/s_f)
923          end if
924       end if
925       brokens(f) = broken_f
926       dotlamps(f) = dotlamp_f
927       lamp_f = lamp_f + dotlamp_f*dt
928       if ( (lam_f>1.d0) .and. (lamp_f > lam_f) ) then
929          !print *, "lamp_f>lam_f en fibra: ", f, lam_f, lamp_f
930          if (lamp_f > elonlimit) then
931             print *, "Tac!", lam_f, lamp_f
932             !call escribir_mallita(masim, "malla_con_error.txt"//repeat(" ", 120))
933             brokens(f) = .true.
934             dotlamps(f) = 0.d0 !
935             lamp_f = masim%lamps(f) ! no plastifico, porque rompi
936          end if
937       end if
938       ! Actualizo informacion en la propia malla
939       masim%brokens(f) = broken_f
940       masim%lamps(f) = lamp_f
941    end do
942    ! Actualizo la propia malla
943
944    ! ----------
945 end subroutine calcular_plasticidad_rotura
946 ! ================================================================================
947 ! ================================================================================
948
949
950 ! ================================================================================
951 ! ================================================================================
952 subroutine homogeneizacion(masim, r1, Tmacro)
953    ! Calcula el tensor de tensiones como producto de la homogeneizacion de las
```

```
954     ! tensiones de las fibras.
955     ! ----------
956     implicit none
957     ! ----------
958     type(MallaSim), intent(in) :: masim
959     real(8), intent(in) :: r1(2,masim%nnods)
960     real(8), intent(out) :: Tmacro(2,2) ! tension macroscopica de Cauchy
961     ! ----------
962     integer :: f
963     real(8) :: d_f
964     real(8) :: Area_f
965     real(8) :: lete0_f
966     real(8) :: Vol_f
967     integer :: n1_f, n2_f
968     real(8) :: r_n1_f(2), r_n2_f(2)
969     real(8) :: dr_f(2)
970     real(8) :: lete_f
971     real(8) :: lam_f
972     real(8) :: a_f(2)
973     real(8) :: ten_f, vfza_f(2)
974     real(8) :: Vol_RVE
975     ! ----------
976
977     Tmacro = 0.d0
978     do f=1,masim%nfibs
979        d_f = masim%diams(f)
980        Area_f = pi * d_f * d_f / 4.d0
981        lete0_f = masim%letes0(f)
982        Vol_f = Area_f * lete0_f
983        ! nodos de la fibra f
984        n1_f = masim%fibs(1,f)
985        n2_f = masim%fibs(2,f)
986        ! posiciones de esos nodos
987        r_n1_f = r1(:,n1_f)
988        r_n2_f = r1(:,n2_f)
989        ! vector fibra
990        dr_f = r_n2_f - r_n1_f
991        ! elongacion fibra (extremo-extremo)
992        lete_f = dsqrt(sum(dr_f*dr_f))
993        lam_f = lete_f / lete0_f
994        ! vector orientacion de la fibra
995        a_f = dr_f/lete_f * lam_f
996        ! tension fibra
997        call calcular_tension_fibra(masim, f, dr_f, ten_f, vfza_f)
998        ! homogeneizacion (queda poco eficiente, a mejorar luego de que funcione si hace falta)
999        Tmacro(1,1) = Tmacro(1,1) + ten_f/lam_f * a_f(1) * a_f(1) * Vol_f
1000       Tmacro(1,2) = Tmacro(1,2) + ten_f/lam_f * a_f(1) * a_f(2) * Vol_f
1001       Tmacro(2,1) = Tmacro(2,1) + ten_f/lam_f * a_f(2) * a_f(1) * Vol_f
1002       Tmacro(2,2) = Tmacro(2,2) + ten_f/lam_f * a_f(2) * a_f(2) * Vol_f
1003    end do
1004    ! termino de acomodar la tension
1005    Vol_RVE = masim%ncapas * masim%diamed * masim%sidelen * masim%sidelen
1006    Tmacro = Tmacro / Vol_RVE
1007
1008    ! ----------
1009 end subroutine homogeneizacion
1010 ! ================================================================================
1011 ! ================================================================================
1012
1013
1014 ! ================================================================================
1015 END MODULE class_mallita
1016 ! ================================================================================
```

```
1 MODULE Aux
2
3
4 real(8), parameter :: pi = 3.1415926536
5 REAL(8), PARAMETER :: pi2 = pi*2.0d0, pi15 = pi*1.5d0, pi05 = pi*0.5d0
6 real(8), parameter :: e = 2.7182818285
```

```
7
8
9  CONTAINS
10
11 ! ================================================================================
12 ! ================================================================================
13 FUNCTION FindStringInFile(Str, ioUnit, Mandatory) RESULT (iError)
14    ! ============================================================================
15    ! Busca un String en un archivo (STR), sino lo encuentra rebovina el archivo
16    ! y pone iError < 0 como indicador de no hallazgo
17    ! Str: String to find, ioUnit: Unit assigned to Input File; iError: Error Status variable
18    ! ============================================================================
19    IMPLICIT NONE
20    ! ============================================================================
21    ! Parameters
22    LOGICAL,PARAMETER :: Segui=.True.
23    ! ============================================================================
24    ! Arguments
25    CHARACTER(*), INTENT(IN) :: Str
26    INTEGER, INTENT (IN) :: ioUnit
27    LOGICAL, OPTIONAL, INTENT(IN) :: Mandatory
28    ! ============================================================================
29    ! Locals
30    LOGICAL :: MandatoryL
31    CHARACTER(LEN=120) :: DummyString
32    CHARACTER(LEN=120) :: upStr
33    INTEGER :: iError
34    INTEGER :: Leng
35    ! ============================================================================
36
37    ! ============================================================================
38    IF ( PRESENT(Mandatory) ) THEN
39       MandatoryL = Mandatory
40    ELSE
41       MandatoryL = .FALSE.
42    END IF
43    ! ============================================================================
44    iError=0
45    Leng = LEN_TRIM(Str)
46    upStr = Upper_Case(Str) ! Line added by NB. Converts Str to Upper Case string
47    ! ============================================================================
48    REWIND(ioUnit)
49    Search_Loop: DO WHILE (segui)
50       READ(ioUnit,'(1A120)',IOSTAT=iError) DummyString ! iError es 0 si lee con exito, >0 si hay error y <0 si es end
             ↪ of file.
51       DummyString = Upper_Case(DummyString) ! line added by NB
52       ! if (iError==59) backspace(ioUnit)
53       IF (iError.lt.0) THEN
54          REWIND (ioUnit)
55          EXIT Search_Loop
56       END IF
57       IF ( DummyString(1:1) /= '*' ) CYCLE Search_Loop
58       IF ( DummyString(1:Leng) == upStr(1:Leng) ) EXIT Search_Loop
59    END DO Search_Loop
60    ! ============================================================================
61    IF (MandatoryL) THEN
62       IF ( .NOT. ( iError == 0 ) ) THEN
63          WRITE(*,*) upStr, 'NOT␣FOUND'
64          STOP
65       END IF
66    END IF
67    ! ============================================================================
68
69 END FUNCTION FindStringInFile
70 ! ================================================================================
71 ! ================================================================================
72
73
74 !! ================================================================================
75 !! ================================================================================
```

```
!subroutine read_instruction(configfilename)
!
!end subroutine
!! ================================================================================
!! ================================================================================


! ================================================================================
! ================================================================================
FUNCTION Upper_Case(string)

   !-----------------------------------------------------------------------------------------------
   !! The Upper_Case function converts the lower case characters of a string to upper case one.
   !! Use this function in order to achieve case-insensitive: all character variables used
   !! are pre-processed by Uppper_Case function before these variables are used. So the users
   !! can call functions without pey attention of case of the keywords passed to the functions.
   !-----------------------------------------------------------------------------------------------

   IMPLICIT NONE

   !-----------------------------------------------------------------------------------------------
   CHARACTER(LEN=*), INTENT(IN):: string ! string to be converted
   CHARACTER(LEN=LEN(string)):: Upper_Case ! converted string
   INTEGER:: n1 ! characters counter
   !-----------------------------------------------------------------------------------------------

   Upper_Case = string
   DO n1=1,LEN(string)
      SELECT CASE(ICHAR(string(n1:n1)))
         CASE(97:122)
            Upper_Case(n1:n1)=CHAR(ICHAR(string(n1:n1))-32) ! Upper case conversion
      END SELECT
   ENDDO
   RETURN

   !---------------------------------------------------------------------------------------------------
END FUNCTION Upper_Case
! ================================================================================
! ================================================================================



! ================================================================================
! ================================================================================
FUNCTION get_file_unit() RESULT(lu)
   ! ============================================================================
   ! get_file_unit returns a unit number that is not in use
   ! ============================================================================
   INTEGER :: lu_max, lu, checkIOSTAT
   LOGICAL checkOPENED
   INTEGER, PARAMETER :: m = 99
   !
   DO lu = m,1,-1
      INQUIRE (UNIT=lu, OPENED=checkOPENED, IOSTAT=checkIOSTAT)
      IF (checkIOSTAT.ne.0) CYCLE
      IF (.NOT.checkOPENED) EXIT
   END DO
   !

   RETURN
END FUNCTION get_file_unit
! ================================================================================
! ================================================================================



! ================================================================================
! ================================================================================
subroutine modify_txt_filename(txt_filename_in, txt_filename_out)
   ! Agrego caracteres a un archivo txt antes de la extension ".txt"7
   ! El nombre original del archivo viene en txt_filename_in, que se mantiene sin modificar
```

```
146     ! Los caracteres a agregar van en la variable txt_filename_out
147     ! que es donde tambien se guarda el nombre modificado del archivo
148     implicit none
149     character(len=120), intent(in) :: txt_filename_in
150     character(len=120), intent(inout) :: txt_filename_out
151     integer :: nchars
152 
153     nchars = len(trim(txt_filename_in))
154     txt_filename_out = trim(txt_filename_in(1:nchars-4)) // trim(txt_filename_out) // ".txt"
155 
156 end subroutine modify_txt_filename
157 ! ================================================================================
158 ! ================================================================================
159 
160 
161 ! ================================================================================
162 ! ================================================================================
163 SUBROUTINE angulo_de_segmento(r0, r1, th, iError)
164     IMPLICIT NONE
165     REAL(8), INTENT(IN) :: r0(2), r1(2)
166     REAL(8), INTENT(OUT) :: th ! angulo
167     INTEGER, INTENT(OUT) :: iError ! -1: No hay interseccion, 0: hay interseccion en el medio, 1: hay interseccion en
          ↪ un punto extremo de uno de los segmentos
168     !
169     REAL(8) :: dr(2), dx, dy
170 
171 
172     iError = 0
173     ! ----------
174     dr = r1 - r0
175     dx = dr(1)
176     dy = dr(2)
177     IF (iguales(dx,0.d0)) THEN ! segmento vertical
178        IF (iguales(dy,0.d0)) THEN ! segmento nulo
179           WRITE(*,*) "Error, segmento nulo!"
180           STOP
181        ELSE IF (dy>0.d0) THEN ! vertical hacia arriba
182           th = pi05
183           RETURN
184        ELSE ! vertical hacia abajo
185           th = pi15
186           RETURN
187        END IF
188     ELSE IF (iguales(dy,0.d0)) THEN ! segmento horizontal
189        IF (dx>0.d0) THEN ! segmento hacia derecha
190           th = 0.d0
191           RETURN
192        ELSE ! segmento hacia izquierda
193           th = pi
194           RETURN
195        END IF
196     ELSE ! segmento oblicuo
197        IF (dx<0) THEN ! segundo o tercer cuadrante
198           th = pi + DATAN(dy/dx)
199           RETURN
200        ELSE IF (dy>0) THEN ! primer cuadrante
201           th = DATAN(dy/dx)
202           RETURN
203        ELSE ! cuarto cuadrante
204           th = pi2 + DATAN(dy/dx)
205           RETURN
206        END IF
207     END IF
208     ! ----------
209     th = 0.d0
210     iError = 1 ! si llegamos hasta aca estamos mal
211     WRITE(*,*) "Error calculando el angulo de un segmento!"
212     STOP
213 
214 
```

```
215 END SUBROUTINE angulo_de_segmento
216 ! ================================================================================
217 ! ================================================================================
218
219
220 ! ================================================================================
221 ! ================================================================================
222 SUBROUTINE intersectar_segmentos(rs1n1, rs1n2, rs2n1, rs2n2, r_in, iStatus)
223    ! Busca interseccion entre dos segmentos
224    ! devuelve iStatus >= 0 si hay interseccion, -1 si no hay interseccion
225    ! r_in son las coordenadas de la interseccion
226    IMPLICIT NONE
227    REAL(8), INTENT(IN) :: rs1n1(2), rs1n2(2), rs2n1(2), rs2n2(2)
228    REAL(8), INTENT(OUT) :: r_in(2)
229    INTEGER, INTENT(OUT) :: iStatus ! -1: No hay interseccion, 0: hay interseccion en el medio, 1: hay interseccion en
           ↪ un punto extremo de uno de los segmentos
230    !
231    INTEGER :: ierr
232    REAL(8) :: th1, th2
233    LOGICAL :: paralelos
234    REAL(8) :: bot1, rig1, top1, lef1, bot2, rig2, top2, lef2
235    REAL(8) :: maxbot, minrig, mintop, maxlef
236    LOGICAL :: lejos
237    REAL(8) :: th2rel
238    LOGICAL :: m2relinf
239    REAL(8) :: cos1, sin1
240    REAL(8) :: A1(2,2) ! matriz de cambio de base con th1
241    REAL(8) :: s12(2), s21(2), s22(2), s_in ! posiciones relativas en el sistema intrinseco al segmento 1 (s11 = [0,0])
242    REAL(8) :: m2rel
243    LOGICAL :: inex11, inex12, inex21, inex22
244    INTEGER :: inex1, inex2 ! interseccion en extremo
245
246
247    ! ----------
248    ! chequeo paralelismo (no habria interseccion)
249    CALL angulo_de_segmento(rs1n1, rs1n2, th1, ierr)
250    CALL angulo_de_segmento(rs2n1, rs2n2, th2, ierr)
251    paralelos = iguales(th1, th2, pi*1.d-5)
252    IF (paralelos) THEN
253       iStatus = -1
254       RETURN
255    END IF
256    ! ----------
257    ! chequeo lejania (no habria interseccion)
258    bot1 = MIN(rs1n1(2),rs1n2(2))
259    rig1 = MAX(rs1n1(1),rs1n2(1))
260    top1 = MAX(rs1n1(2),rs1n2(2))
261    lef1 = MIN(rs1n1(1),rs1n2(1))
262    bot2 = MIN(rs2n1(2),rs2n2(2))
263    rig2 = MAX(rs2n1(1),rs2n2(1))
264    top2 = MAX(rs2n1(2),rs2n2(2))
265    lef2 = MIN(rs2n1(1),rs2n2(1))
266    ! ----------
267    maxbot = MAX(bot1,bot2)
268    minrig = MIN(rig1,rig2)
269    mintop = MIN(top1,top2)
270    maxlef = MAX(lef1,lef2)
271    ! ----------
272    lejos = ( ((maxlef-minrig)>1.0d-8) .OR. ((maxbot-mintop)>1.0d-8) )
273    IF (lejos) THEN
274       iStatus = -1
275       RETURN
276    END IF
277    ! ----------
278    ! sistema de referencia relativo, intrinseco al segmento 1
279    th2rel = th2 - th1
280    DO WHILE ( (th2rel<0.d0) .OR. (th2rel>=pi2) )
281       IF (th2rel<0.d0) THEN
282          th2rel = th2rel + pi2
283       ELSE IF (th2rel>=pi2) THEN
```

```
284         th2rel = th2rel - pi2
285       END IF
286     END DO
287     ! me fijo si en el sistema intrinsieco al segmento 1, el segmento 2 resulta vertical
288     m2relinf = ( (iguales(th2rel,pi05,pi*1.d-5)) .OR. (iguales(th2rel,pi15,pi*1.d-5)) ) ! guardo esa informacion
289     ! coseno y seno de th1
290     cos1 = DCOS(th1)
291     sin1 = DSIN(th1)
292     A1 = RESHAPE( [cos1, -sin1, sin1, cos1], [2,2] )
293     s12 = MATMUL(A1,rs1n2-rs1n1)
294     s21 = MATMUL(A1,rs2n1-rs1n1)
295     s22 = MATMUL(A1,rs2n2-rs1n1)
296     ! ----------
297     ! chequeo que el segmento 2 corte al eje del segmento 1
298     IF ( NINT(SIGN(1.0d0,s21(2))) == NINT(SIGN(1.0d0,s22(2))) ) THEN
299       iStatus = -1
300       RETURN
301     END IF
302     ! ----------
303     ! calculo a que distancia sobre el eje del segmento 1 se produce la posible interseccion
304     IF (m2relinf) THEN ! si el segmento 2 es perpendicular al 1
305       s_in = s21(1)
306     ELSE ! si es oblicuo
307       m2rel = DTAN(th2rel)
308       s_in = s21(1) - s21(2)/m2rel
309     END IF
310     ! ----------
311     ! ahora chequeo si la interseccion se produjo por fuera del segmento 1
312     IF ((s_in<0.d0) .OR. (s_in>s12(1))) THEN
313       iStatus = -1
314       RETURN
315     END IF
316     ! ----------
317     ! HAY INTERSECCION
318     ! me fijo si ocurre en el medio de los segmentos, o si coincide con los nodos ya existentes (extremos de segmentos)
319     inex11 = iguales(s_in, 0.d0, tol_in=1.d-2)
320     inex12 = iguales(s_in,s12(1), tol_in=1.d-2)
321     inex21 = iguales(s21(2),0.d0, tol_in=1.d-2)
322     inex22 = iguales(s22(2),0.d0, tol_in=1.d-2)
323     ! HAY VARIAS POSIBILIDADES
324     ! 4: extremo-extremo
325     IF ((inex11 .OR. inex12) .AND. (inex21 .OR. inex22)) THEN
326       ! Interseccion extremo-extremo, ocurre rara vez
327 ! WRITE(*,*) " Interseccion tipo extremo-extremo. Por ahora no esta implementada."
328       iStatus = -1
329       RETURN
330     END IF
331     ! 3: extremo-medio
332     IF ((inex11 .OR. inex12) .OR. (inex21 .OR. inex22)) THEN
333       ! Interseccion extremo-medio, ocurre rara vez
334 ! WRITE(*,*) " Interseccion tipo extremo-medio. Por ahora no esta implementada."
335       iStatus = -1
336       RETURN
337     END IF
338     ! 2: medio-medio
339     ! la inmensa mayoria de las intersecciones
340     ! chequeo redundante
341     IF ( (s_in>0.d0) .AND. (s_in<s12(1)) ) THEN
342       iStatus = 2
343       r_in = rs1n1 + s_in * [cos1, sin1]
344       RETURN
345     END IF
346     ! ----------
347     ! si llegue hasta aca hay algo mal
348     WRITE(*,*) "Error en subrutina calcular interseccion entre segmentos!"
349     STOP
350     ! ----------
351
352
353
```

```
354 END SUBROUTINE intersectar_segmentos
355 ! ================================================================================
356 ! ================================================================================
357
358
359 ! ================================================================================
360 ! ================================================================================
361 FUNCTION iguales(x,y,tol_in) RESULT(igual)
362    IMPLICIT NONE
363    REAL(8), INTENT(IN) :: x,y
364    REAL(8), OPTIONAL, INTENT(IN) :: tol_in
365    LOGICAL :: igual
366    REAL(8) :: tol
367
368    IF (PRESENT(tol_in)) THEN
369       tol = tol_in
370    ELSE
371       tol = 1.d-10
372    END IF
373
374    igual = DABS(x-y)<tol
375
376 END FUNCTION
377 ! ================================================================================
378 ! ================================================================================
379
380
381 ! ================================================================================
382 ! ================================================================================
383 subroutine diagonal_iterativo(n, A, b, x, iStatus)
384    ! ----------
385    implicit none
386    integer, intent(in) :: n
387    real(8), intent(in) :: A(n,n), b(n)
388    real(8), intent(out) :: x(n)
389    integer, intent(out) :: iStatus
390    ! ----------
391    integer, parameter :: maxiter = 100
392    real(8), parameter :: tol = 1.d-6
393    ! ----------
394    real(8) :: x1(n)
395    real(8) :: tmp
396    integer i,j,k
397    real(8) :: error_dx
398    ! ----------
399
400    ! ----------
401    ! valor semilla
402    x(:) = 0.d0
403    x1(:) = x(:)
404    ! ----------
405    ! iteraciones
406    do k=1,maxiter
407       ! hago la sumatoria de A(i,j)*x(j) exceptuando la diagonal
408       do i=1,n
409          ! calculo cada valor de x
410          x(i) = b(i) / A(i,i)
411       end do
412       ! chequeo error y convergencia
413       error_dx = maxval(dabs(x-x1))
414       if (error_dx<tol) then
415          iStatus = 0
416          return
417       end if
418       ! actualizo el valor de la iteracion anterior
419       x1(:) = x(:)
420    end do
421    ! ----------
422    ! si llegue hasta aca es porque llegue a maxiter
423    write(*,*) "WARNING: maxiter alcanzado en metodo diagonal_iterativo"
```

```
424    iStatus = 1
425    ! ----------
426
427    ! ----------
428 end subroutine diagonal_iterativo
429 ! ================================================================================
430 ! ================================================================================
431
432
433 ! ================================================================================
434 ! ================================================================================
435 subroutine jacobi(n, A, b, x, iStatus)
436    ! ----------
437    implicit none
438    integer, intent(in) :: n
439    real(8), intent(in) :: A(n,n), b(n)
440    real(8), intent(out) :: x(n)
441    integer, intent(out) :: iStatus
442    ! ----------
443    integer, parameter :: maxiter = 100
444    real(8), parameter :: tol = 1.d-6
445    ! ----------
446    real(8) :: x1(n)
447    real(8) :: tmp
448    integer i,j,k
449    real(8) :: error_dx
450    ! ----------
451
452    ! ----------
453    ! valor semilla
454    x(:) = 0.d0
455    x1(:) = x(:)
456    ! ----------
457    ! iteraciones de gauss-seidel
458    do k=1,maxiter
459       ! hago la sumatoria de A(i,j)*x(j) exceptuando la diagonal
460       do i=1,n
461          tmp = 0
462          do j=1,n
463             if (j<i) then
464                tmp = tmp + A(i,j)*x1(j)
465             else if (j>i) then
466                tmp = tmp + A(i,j)*x1(j)
467             end if
468          end do
469          ! calculo cada valor de x
470          x(i) = (b(i)-tmp) / A(i,i)
471       end do
472       ! chequeo error y convergencia
473       error_dx = maxval(dabs(x-x1))
474       if (error_dx<tol) then
475          iStatus = 0
476          return
477       end if
478       ! actualizo el valor de la iteracion anterior
479       x1(:) = x(:)
480    end do
481    ! ----------
482    ! si llegue hasta aca es porque llegue a maxiter
483    write(*,*) "WARNING: maxiter alcanzado en metodo jacobi"
484    iStatus = 1
485    ! ----------
486
487    ! ----------
488 end subroutine jacobi
489 ! ================================================================================
490 ! ================================================================================
491
492
493 ! ================================================================================
```

```
494 ! ================================================================================
495 subroutine gauss_seidel(n, A, b, x, maxiter, tol, iStatus)
496    ! ----------
497    implicit none
498    integer, intent(in) :: n
499    real(8), intent(inout) :: A(n,n), b(n)
500    real(8), intent(out) :: x(n)
501    integer, intent(out) :: iStatus
502    integer, intent(in) :: maxiter
503    real(8), intent(in) :: tol
504    ! ----------
505    real(8) :: x1(n)
506    real(8) :: tmp, mintmp, maxtmp
507    integer i,j,k
508    real(8) :: error_dx
509    ! ----------
510
511    ! ----------
512    ! valor semilla
513    x(:) = 0.d0
514    x1(:) = x(:)
515    ! ----------
516    ! divido cada fila de matriz y vector por el valor de la diagonal
517    ! ojo que no deberia ser un valor muy pequeno porque error
518 ! mintmp = dabs( A(1,1) )
519 ! maxtmp = dabs( A(1,1) )
520 ! do i=1,n
521 ! tmp = 1.d0 / dabs(A(i,i))
522 ! A(i,:) = A(i,:) * tmp
523 ! b(i) = b(i) * tmp
524 ! if (tmp<mintmp) mintmp = tmp
525 ! if (tmp>maxtmp) maxtmp = tmp
526 ! end do
527    ! ----------
528    ! iteraciones de gauss-seidel
529    do k=1,maxiter
530       ! hago la sumatoria de A(i,j)*x(j) exceptuando la diagonal
531       do i=1,n
532          tmp = 0
533          do j=1,n
534             if (j<i) then
535                tmp = tmp + A(i,j)*x(j)
536             else if (j>i) then
537                tmp = tmp + A(i,j)*x1(j)
538             end if
539          end do
540          ! calculo cada valor de x
541          x(i) = (b(i)-tmp) / A(i,i)
542       end do
543       ! chequeo error y convergencia
544       error_dx = maxval(dabs(x-x1))
545       write(*,*) "k=", k, "error_dx=", error_dx, "/ tol=", tol
546       if (error_dx<tol) then
547          write(*,*) "Gauss-Seidel converge"
548          iStatus = 0
549          return
550       end if
551       ! actualizo el valor de la iteracion anterior
552       x1(:) = x(:)
553    end do
554    ! ----------
555    ! si llegue hasta aca es porque llegue a maxiter
556    write(*,*) "WARNING: maxiter alcanzado en metodo gauss_seidel: ", maxiter
557    iStatus = 1
558    ! ----------
559
560    ! ----------
561 end subroutine gauss_seidel
562 ! ================================================================================
563 ! ================================================================================
```

```
564
565 ! ================================================================================
566 ! ================================================================================
567 subroutine directo(n, A, b, x, iStatus)
568    ! ----------
569    implicit none
570    integer, intent(in) :: n
571    real(8), intent(in) :: A(n,n), b(n)
572    real(8), intent(out) :: x(n)
573    integer, intent(out) :: iStatus
574    ! ----------
575    integer, parameter :: maxiter = 100
576    real(8), parameter :: tol = 1.d-6
577    ! ----------
578    real(8) :: A1(n,n)
579    real(8) :: residuo(n)
580    real(8) :: tmp
581    integer i,j,k
582    real(8) :: error_dx
583    ! ----------
584
585
586    ! invierto la matriz A
587    A1 = matinv(A)
588    ! calculo x haciendo x=A1 b
589    do i=1,n
590       x(i) = 0.d0
591       do j=1,n
592          x(i) = x(i) + A1(i,j)*b(j)
593       end do
594    end do
595    iStatus = 0
596
597    ! ----------
598 end subroutine directo
599 ! ================================================================================
600 ! ================================================================================
601
602 function matinv(A) result(Ainv)
603   real(8), dimension(:,:), intent(in) :: A
604   real(8), dimension(size(A,1),size(A,2)) :: Ainv
605
606   real(8), dimension(size(A,1)) :: work ! work array for LAPACK
607   integer, dimension(size(A,1)) :: ipiv ! pivot indices
608   integer :: n, info
609
610   ! External procedures defined in LAPACK
611   external DGETRF
612   external DGETRI
613
614   ! Store A in Ainv to prevent it from being overwritten by LAPACK
615   Ainv = A
616   n = size(A,1)
617
618   ! DGETRF computes an LU factorization of a general M-by-N matrix A
619   ! using partial pivoting with row interchanges.
620   call DGETRF(n, n, Ainv, n, ipiv, info)
621
622   if (info /= 0) then
623      stop 'Matrix␣is␣numerically␣singular!'
624   end if
625
626   ! DGETRI computes the inverse of a matrix using the LU factorization
627   ! computed by DGETRF.
628   call DGETRI(n, Ainv, n, ipiv, work, n, info)
629
630   if (info /= 0) then
631      stop 'Matrix␣inversion␣failed!'
632   end if
633 end function matinv
```

```
634
635
636
637 END MODULE Aux
```