Development of a utility in OpenFOAM to predict
fiber orientation in fiber reinforced thermoplastic materials

Marta Oliveira

UMinho | 2023

Universidade do Minho
Escola de Engenharia

Marta Cláudia Baptista Oliveira

Development of a utility in OpenFOAM to predict fiber orientation in fiber reinforced thermoplastic materials

October 2023

Universidade do Minho
Escola de Engenharia

Marta Cláudia Baptista Oliveira

Development of a utility in OpenFOAM to predict fiber orientation in fiber reinforced thermoplastic materials

Master's Dissertation
Integrated Masters in Polymer Engineering

Work accomplished under the supervision of:

Professor Doctor João Miguel Nóbrega
MSc. Bruno Ramôa

October 2023

Acknowledgements

My deepest gratitude goes to my advisors, Professor Miguel and Master Bruno. Not only for offering me the opportunity to learn from them, but also for their patience with me.

I would also like to thank my friend Castro and those who did not give up on me.

Statement of integrity

Resumo

Os termoplásticos reforçados com fibras utilizam frequentemente fibras de vidro ou carbono para melhorar as propriedades de termoplásticos simples. As propriedades deste tipo de materiais compósitos depende das propriedades dos seus constituintes base, no ambiente termo-mecânico desenvolvido durante o seu processamento e nos parâmetros de distribuição das fibras, nomeadamente a sua concentração e orientação, tendo a última uma relevância maior na propriedades finais da peça.

Atualmente existem vários modelos fenomenológicos desenvolvidos com o objetivo de prever a evolução da orientação das fibras para diversos fluxos. Alguns destes modelos estão presentes em software proprietário, cujo acesso ao código é limitado ou inexistente, o que compromete a verificação e adaptação do mesmo. Como alternativa, software de código-aberto oferecem aos utilizadores acesso irrestrito ao código, permitindo assim análises cientificas detalhadas e contínuo melhoramentos. O OpenFOAM® é uma biblioteca computacional de fonte aberta, que contém diversos solvers capazes de simular diferentes problemas da mecânica computacional, com especialização na dinâmica computacional de fluídos, o que o torna uma ferramenta adequada para o trabalho a realizar.

Neste dissertação é proposta uma estratégia inovadora para a implementação de um modelos estado-de-arte de orientação de fibras no ambiente do OpenFOAM®. Os modelos foram implementados na forma de um utilitário, que pode ser utilizado juntamente com outros solvers, desde que exista acesso ao campo de velocidades. Com a finalidade de realizar verificações numéricas, foi desenvolvido um código em `python` capaz de calcular a evolução da orientação das fibras num domínio sujeito a um campo de velocidades variável e os resultados obtidos foram verificados com base em informação obtida da literatura. Este código foi posteriormente implementado como uma referência para verificar o utilitário introduzido no OpenFOAM®, que foi testado em casos semelhantes aos encontrados em condições realistas para típicas aplicações poliméricas, como fluxo de corte simples e estiramento, tal como o fluxo na injeção de um disco ao centro. Os resultados obtidos nos casos de estudo permitiram concluir que a implementação dos modelos foi adequada, o que transforma esta ferramenta numa alternativa confiável a software proprietário, para a previsão da evolução da orientação de fibras em diversos tipos de fluxo.

Palavras-chave: orientação de fibras, termoplásticos reforçados com fibras, modelação numérica, OpenFOAM®

Abstract

Fiber reinforced thermoplastic materials commonly employ glass or carbon fibers as structural reinforcements to enhance the properties of single thermoplastic materials. The properties of this class of composite materials depend on the properties of the individual constituents, the thermomechanical environment developed during its processing, but also on the fibers distribution parameters, namely the concentration and orientation. The latter has a special relevance in the final part performance.

Currently, there are various phenomenological models devised to predict the evolution of fiber orientation in general flows. Some fiber orientation models are available in proprietary software, which work on a black-box concept, where the users are not allowed to check the underlying code and have limited or no possibilities to adapt the code to their own needs. On the other side, open-source software supplies the end-users with full access to the code, making it available for scientific scrutiny and continuous improvement. OpenFOAM® is an open-source computational library with a wide range of solvers able to simulate different continuum mechanics problems, with special focus on computational fluid dynamics. Such properties make the software a suitable tool to work as base to the proposed work.

In this dissertation a novel strategy for implementing state-of-the-art fiber orientation models is proposed and implemented in the OpenFOAM® framework. The fiber orientation models were implemented as an utility, which can be used with other existing solvers, in which there is access to the velocity field. For numerical verification purposes, a python script able to compute the fiber orientation evolution in a location subjected to variable velocity gradient field, was implemented and verified against data found in literature. This python script was subsequently employed as a benchmark to verify the implementation of the novel OpenFOAM® utility, which was tested considering some flows that represent the ones found in practice, in typical polymer processing applications (simple shear flow and stretch flow), as well as the flow in a standard center-gated disk. The results obtained in the tested case studies allowed to conclude that the fiber orientation models were properly implemented. Consequently, this new OpenFOAM® utility is now a reliable alternative to proprietary software for predicting the evolution of fiber orientation in different flow environments.

Keywords: fiber orientation, fiber reinforced thermoplastic materials, numerical modeling, OpenFOAM®

Table of contents

List of figures

List of tables

Listings

Nomenclature

Latin symbols

$\mathbf{A}$	Second-order orientation tensor
$\mathbb{A}$	Fourth-order orientation tensor
$\mathbf{B}$	Positive definite matrix with determinant one
$\mathbb{C}$	Conversion tensor
$\mathbf{C}$	Rotary diffusion tensor
C_I	Interaction coefficient
C_M	Fiber-matrix interaction coefficient
C_m	Coefficients for the EBOF closures
$\mathbb{D}$	Conversion tensor
$\mathbf{D}$	Rate-of-deformation tensor
$\mathbf{E}$	Matrix of right eigenvectors of $\mathbf{D}$
$\mathbf{F}$	Variable tensor function
$\mathbf{G}$	Variable tensor function
$\mathbf{I}$	Identity tensor
$\mathbb{L}$	Compact definition for $\sum_{i=1}^{3} \lambda_i (\mathbf{e}_i \mathbf{e}_i \mathbf{e}_i \mathbf{e}_i)$
$\mathbb{M}$	Compact definition for $\sum_{i=1}^{3} (\mathbf{e}_i \mathbf{e}_i \mathbf{e}_i \mathbf{e}_i)$
$\mathbf{M}_z$	Manas-Zloczower number
$\mathbf{Q}$	Rotation matrix for ORE closure
Q	Volumetric flux
$\mathbf{R}$	Rotation matrix for the iARD model
$\mathbf{S}$	Symmetric part of a tensor
$\mathbf{W}$	Vorticity tensor

a	Acceleration
b	Half-thickness of a disk
b_i	Parameters for the definition of the rotary diffusion tensor
c	Fiber concentration
d	Fiber diameter
$\mathbf{e}$	Eigenvectors
f	Blending coefficient
l	Fiber length
$\mathbf{p}$	Fiber orientation vector
r	Radius of a disk
t	Time
$\mathbf{u}$	Velocity vector
z	Thickness of a disk

Greek Symbols

α	Strain reduction factor
β	Objective parameter
β_i	Functions of the second and third invariants of $\mathbf{A}$
$\boldsymbol{\delta}$	Unit vector
ϵ	Elongational rate
ε	Error
γ	Shear rate
κ	Slow-down parameter
λ	Eigenvalues
$\mathbf{\Lambda}^{\mathrm{IOK}}$	Diagonal tensor calculated under the intrinsic orientation kinetics

∇_s	Gradient operator on the surface of the unit sphere
ψ	Probability distribution function
ξ	Particle shape function

Superscripts/subscripts

$\bar{\square}$	Average
$\dot{\square}$	Material derivative
$\square^T$	Transpose operation
$\square_0$	Initial value
$\square_f$	Final value
$\tilde{\square}$	Normalized value

CHAPTER 1 Introduction

Fiber reinforced polymers (FRP) are a subclass of composite materials, where thermoplastics or thermoset polymers are used as the continuous phase matrix, and either glass or carbon fibers comprise the reinforcing dispersed phase. The main purpose of the reinforcements is to enhance the ratio mechanical properties per weight of the final parts, however, thermal and electrical behavior can also be tailored to some extent to devise products with additional functionalities [1, 2].

Due to the wide range of properties that FRP can present, they are attractive to several industries, as is the case for the automotive, aerospace and construction sectors [1–3], which benefit from the load bearing capabilities allied with low density of this class of materials. For instance, it is common practice in the automotive industry to replace metallic components by FRP to create lighter components and, therefore, improve fuel consumption [4].

Commercially, most FRP incorporate thermoset resins for the matrix, such as epoxy [3]. This type of polymers can be reshaped until undergoing an irreversible chemical reaction, usually triggered by heat. On the other side, to be reshaped, thermoplastics need to go through a cycle that comprises, at least, heating-forming-cooling phases. Thermoplastic materials can be reheated and reshaped a (limited) number of times, which makes them reusable [5]. Despite the more difficult processing associated with thermoplastics matrices [3], mainly due to their high viscosity at the melt state when compared with uncured thermoset resins, they have been growing in use, not only for their increased recyclability, but also because thermoplastics processing can be much faster [6].

FRP materials can also be grouped by the length of the fiber employed, which can be discontinuous or continuous. The latter results in materials with high strength and stiffness, due to their high orientation along the desired direction. Meanwhile, discontinuous fibers tend to have random orientation and therefore lower strength. However, they can still provide a significant reinforcement, are cheaper and can be used to produce complex parts [1, 2].

Discontinuous fibers can be further divided into two categories, depending on the fibers' length. Long fiber reinforced composites are usually obtained through a pultrusion process, to obtain continuous fiber filaments, which are subsequently pelletized, resulting in pellets comprising fibers with a length of around 12 mm. On the other side, short fiber reinforced composites are produced by compounding extrusion, where chopped fibers are mixed with the matrix, which are subsequently shaped in filaments and then pelletized. These materials comprise fibers having a typical length of around 1 mm [7, 8].

Discontinuous fiber reinforced thermoplastic materials (FRTM) join the advantages of a thermoplastic matrix and discontinuous fibers. Discontinuous FRTM can be processed through conventional thermoplastic processes, like 3D printing, compression molding or injection molding [1]. The latter is widely used by the industry for large scale production, due to its high dimensional precision, variety of complex geometries, high rates of production and ease of automation [3, 6].

1.1 Processing and resulting properties

The injection molding (IM) technique consists of a cyclic process in which the molten plastic, conveyed from the plasticizing unit, is forced into a mold cavity with a predetermined geometry, where it is held under pressure until it cools down and solidifies. The mold can integrate a single or multiple cavities of similar or dissimilar shapes. The processing cycle ends with the opening of the mold and the subsequent extraction of the part(s) [9]. The behavior of FRTM parts manufactured by IM are highly dependent upon processing conditions and fiber-related parameters, such as: orientation, length and concentration, among others. All these interrelated parameters are difficult to control.

Fiber orientation can greatly impact the properties of the final part, such as its mechanical properties, for instance, the part will present a better performance in regions where the fibers are more aligned with the direction of the applied load. The fiber orientation is mostly determined by the flow of the polymeric matrix during injection into the mold cavity, which usually results in a complex anisotropic distribution.

Based on the above, for design purposes it is important to predict fibers distribution and orientation. This knowledge allows for better comprehension of related phenomena such as shrinkage and warpage [10], and also the mechanical behaviour [11], which are mandatory to support part design.

1.2 Computational tools

Nowadays, computational modeling is widely used to predict the filling of mold cavities during the IM process, and some of the available software has the capability of predicting the resulting fiber orientation [12, 13], thanks to phenomenological numerical models based on Jeffery's work [14]. Despite being a powerful ally for design and analysis tasks, the accuracy and generality of the available software present several limitations.

Proprietary software, such as Autodesk Moldflow® [15] and Moldex3D® [16], offer fiber orientation numerical modeling. Despite the availability of these software, the validation work performed with these models is still scarce, which limits significantly the capability of selecting the best option for a specific case. The scarcity in information can be partially attributed to limitations of proprietary software, such as:

- Proprietary software works as a black-box, where the user is not allowed to verify the underlying code, and, therefore, cannot assess how adequate the software is for the envisaged use.
- The source code cannot be accessed, analyzed or modified, limiting the possibility of detecting bugs, performing improvements or customize it.
- Due to the their commercial nature, these software comprise expensive licensing fees, which limits the possibility of their use to only large companies.

An alternative to proprietary software, are the open-source counterparts, such as the OpenFOAM® toolbox [17]. OpenFOAM® is an open-source computational library with a wide range of solvers able to simulate different continuum mechanics problems, with special focus on computational fluid dynamics (CFD) (e.g., incompressible, compressible, multi-phase, conjugate heat transfer). The library, that also includes several tools for pre and post-processing tasks, can be executed in parallel, and is under continuous active development, with two versions being released annually. The open-source character has the advantage of supplying the end-user with full access to the code, making it available for scientific scrutiny and improvement. This framework makes OpenFOAM® a highly suitable vehicle for building numerical tools to model fiber orientation, and to foster the industry to progress in its digital transformation.

1.3 Motivation and Objectives

Due to the limitations of proprietary software, which limit a through analysis and verification of the available models developed to predict the evolution of fiber orientation, the objective of this dissertation is the implementation and verification of state-of-art numerical models for fiber orientation in the open source computational library, OpenFOAM®. To achieve the objective, firstly, the fiber orientation models will be studied, as well as the computational tools that will be used, OpenFOAM® and Python. The later will be utilized to create a benchmark tool to verify the results obtained from the developed OpenFOAM® utility.

1.4 Dissertation organization

The remaining dissertation is organized as follows. Firstly, the state-of-art fiber orientation models will be described and discussed in Chapter 2. Chapter 3 describes the computational tools utilized, Python and OpenFOAM®, as well as the code developed within the respective libraries of both tools. Additionally, the test cases that will be examined in this dissertation are described by expounding the appropriate definition of explored flows. Finally, in Chapter 4, the verification of the developed code will be presented. The `python` script is verified trough comparison with data from available literature, and, by using this tool as a benchmark, the OpenFOAM®'s utility will be verified against it. The main conclusions from this work and the proposals for future activities are reported in Chapter 5.

CHAPTER 2 State-of-the-art

2.1 Descriptors of fiber orientation

In the literature, fibers are usually treated as a straight and rigid cylinder. Under these conditions, its orientation can be described by two angles, θ and ϕ. A unit vector, $\mathbf{p}$, oriented along the fiber's symmetry axis, as shown in Figure 2.1, is an appropriate micro-descriptor for single fiber orientation and it is related to θ and ϕ by:

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{bmatrix} \tag{2.1}$$

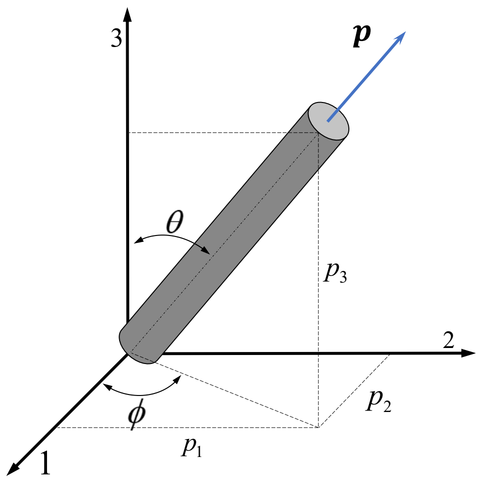

Figure 2.1: Orientation of a fiber.

In FRTM there might be thousands of fibers per cubic millimetre [7]. Consequently, in a general case, the usage of the previous descriptor for each individual fiber is cumbersome and significantly demanding to compute, thus, approaches based in statistical methods were proposed in the literature [7]. In these, the proposed descriptor for large groups of fibers is the a probability distribution function (PDF), $\psi(\mathbf{p},t)$, which reflects a meso-scale descriptor and is defined in such a way that $\psi(\mathbf{p},t)\,\mathrm{d}\mathbf{p}$ defines the probability of a fiber being oriented in the range $\mathbf{p}$ and $\mathbf{p}+\mathrm{d}\mathbf{p}$.

The PDF has the following properties:

- Periodicity: $\psi(\mathbf{p}) = \psi(-\mathbf{p})$ (The fibers ends are not distinguished);
- Normalized: $\oint \psi(\mathbf{p},t)\mathrm{d}\mathbf{p} = 1$ (The integral over all possible orientations in space adds to unity);
- Continuity: $\frac{D\psi}{Dt} = -\nabla_s \cdot (\psi\dot{\mathbf{p}})$ (If the orientation of a fraction of fibers changes, the remaining fibers will suffer a consequent change on their orientation).

The computation of the PDF is possible for planar orientation, but for practical cases in three-dimensions the PDF becomes excessively expensive [18]. For this reason, Advani and Tucker [18] proposed the use of orientation tensors as a macro-scale descriptor. The most commonly used tensors are the 2^{nd} and 4^{th} order orientation tensors, defined as:

$$\mathbf{A} = \oint \psi\mathbf{pp}\,\mathrm{d}\mathbf{p} \tag{2.2}$$

$$\mathbb{A} = \oint \psi\mathbf{pppp}\,\mathrm{d}\mathbf{p} \tag{2.3}$$

The orientation tensors have the following properties:

- Symmetry [18]:
  - $\mathbf{A}_{ij} = \mathbf{A}_{ji}$,
  - $\mathbb{A}_{ijkl} = \mathbb{A}_{jikl} = \mathbb{A}_{kjil} = \mathbb{A}_{ljki} = \mathbb{A}_{ikjl} = \mathbb{A}_{ilkj}$;
- Due to the normalization condition of the PDF, the 2^{nd} order orientation tensor has an unitary trace, $\mathrm{tr}(\mathbf{A}) = 1$ [18];
- Higher order tensors provide complete information about lower order tensors: $\mathbf{A}_{ij} = \mathbb{A}_{ijkk}$ [18];
- The orientation state described by the orientation tensors is not unique [7, 19];

The second-order tensor provides information about the principal directions of the orientation state, as well as its principal values. The principal directions define how the orientation distribution is situated in space, and the principal values quantify the alignment of the fibers with the principal directions. The principal directions are defined by the eigenvectors of the orientation tensor, and the magnitude of said vectors, are defined by its corresponding eigenvalues [7]. Figure 2.2 shows a general representation of the principal directions.

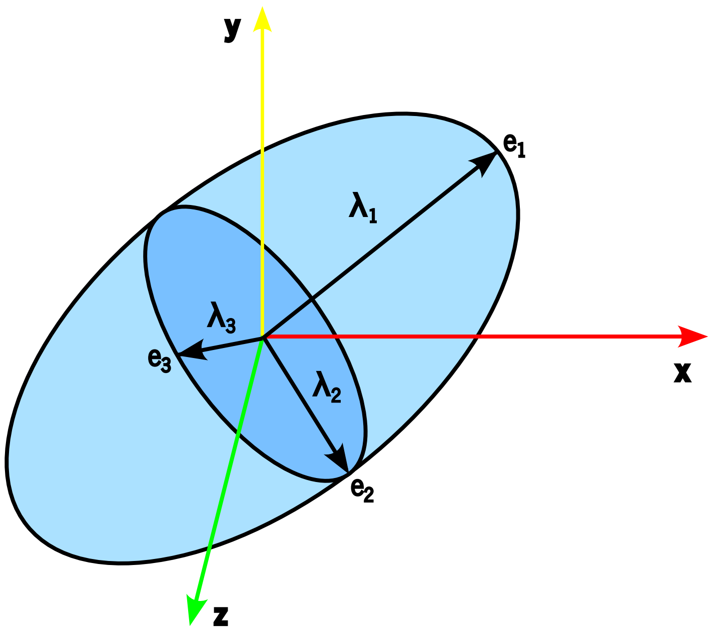

Figure 2.2: Representation of principal directions and values of a second-order orientation tensor.

As it will be further evidenced in this work, the use of tensors to describe the fiber orientation leads to the need of a closure approximation. Despite this, these macro-descriptor are still the state-of-the-art fiber orientation descriptors [7, 19].

2.2 Fiber orientation modeling

Modeling of fiber orientation started with the seminal work of Jeffery in 1922 [14], who modeled a fiber as an inertialess rigid spheroid, suspended in a Newtonian fluid. The proposed equation for the rate of change $\mathbf{p}$ reads:

$$\dot{\mathbf{p}} = \mathbf{W}\cdot\mathbf{p} + \xi(\mathbf{D}\cdot\mathbf{p} - \mathbf{D}:\mathbf{ppp}), \tag{2.4}$$

where:

- $\mathbf{W}$ is the vorticity tensor, defined as $\mathbf{W} = \frac{1}{2}\left(\nabla\mathbf{u} - (\nabla\mathbf{u})^{\mathrm{T}}\right)$;

- $\mathbf{D}$ is the rate-of-deformation tensor, defined as $\mathbf{D} = \frac{1}{2}\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^{\mathrm{T}}\right)$ with $\nabla\mathbf{u}$ as the velocity gradient, defined as $\nabla\mathbf{u} = \partial \mathrm{u_i}/\partial \mathrm{x_j}$;
- ξ is the particle shape function defined as $\xi = \dfrac{(l/d)^2 - 1}{(l/d)^2 + 1}$, where l is the length of the fiber and d its diameter.

The term $\mathbf{W} \cdot \mathbf{p}$ in Equation (2.4) represents the effect of rigid-body rotation on fiber orientation. If the fluid has a rotation motion, this term adds it to the fiber rotation-rate. The second term, $\xi(\mathbf{D} \cdot \mathbf{p} - \mathbf{D} : \mathbf{ppp})$ represents the effect of fluid deformation on the fiber orientation. The unusual term, where the rate-of-deformation tensor is contracted with the triple dyadic product of $\mathbf{p}$, is defined by Tucker [7], and aims at subtracting the portion of $\mathbf{D} \cdot \mathbf{p}$ that is parallel to $\mathbf{p}$, which assures that the time rate of change of $\mathbf{p}$ is always perpendicular to $\mathbf{p}$, and that the length of $\mathbf{p}$ does not change.

This model was proved to be useful in dilute suspensions, where fibers interaction can be neglected [20]. The suspension state is characterized by the concentration of fibers, c, and the fiber aspect ratio, l/d. When $c < (l/d)^2$, the distance in-between fibers is greater than its length, and, therefore, the fibers can rotate freely, which corresponds to the dilute regime [20].

Given the large number of fibers that can exist in FRTM, concentrated suspensions must be considered, where fiber interactions become a relevant, or a dominant effect. To tackle this, Folgar and Tucker (FT) [20], in 1984, formulated a phenomenological model extending the one proposed by Jeffery, which accounts for the effect of fiber interactions through a scalar rotary diffusivity. The model reads:

$$\dot{\psi} = -\nabla_s \cdot (\psi\dot{\mathbf{p}} - C_I\dot{\gamma}\nabla_s\psi), \tag{2.5}$$

where:

- $\mathrm{C_I}$ is a phenomenological constant, the fiber-fiber interaction coefficient;
- $\dot{\gamma}$ is the scalar magnitude of the rate of deformation tensor, defined as $\dot{\gamma} = \sqrt{2\mathbf{D} : \mathbf{D}}$;
- ∇_{s} is the gradient operator on the surface of the unit sphere, defined as $\nabla_{\mathrm{s}} = \hat{\delta}_\theta \frac{\partial}{\partial\theta} + \hat{\delta}_\phi \frac{1}{\sin\delta}\frac{\partial}{\partial\phi}$;
- $\dot{\psi}$ is the material derivative of ψ.

Since the computation of the PDF is too expensive in general 3D cases, orientation tensors are used as macro-descriptors of the fibers' orientation state. For this, Equation (2.5) is rewritten for the evolution of the second order orientation tensor [18] and reads:

$$\dot{\mathbf{A}} = \dot{\mathbf{A}}^{\mathrm{H}} + \dot{\mathbf{A}}^{\mathrm{IRD}} \tag{2.6}$$

$$\dot{\mathbf{A}}^{\mathrm{H}} = \mathbf{W}\cdot\mathbf{A} - \mathbf{A}\cdot\mathbf{W} + \xi(\mathbf{D}\cdot\mathbf{A} + \mathbf{A}\cdot\mathbf{D} - 2\mathbb{A} : \mathbf{D}) \tag{2.7}$$

$$\dot{\mathbf{A}}^{\mathrm{IRD}} = 2C_I\dot{\gamma}(\mathbf{I} - 3\mathbf{A}) \tag{2.8}$$

The time-rate of change of the orientation tensor is decomposed into a hydrodynamic contribution, $\dot{\mathbf{A}}^{\mathrm{H}}$, Equation (2.7) from the Jeffery model, and a diffusive term, from the work of Folgar and Tucker [18], $\dot{\mathbf{A}}^{\mathrm{IRD}}$, Equation (2.8). From the hydrodynamic contribution, it is possible to observe the appearance of a 4th order tensor in the evolution equation of $\mathbf{A}$. To be able to solve this system a closure relationship is required. This topic will be addressed in Section 2.3.

Several practical studies have showed that orientation kinetics predicted by the FT model were much faster than the ones observed experimentally [21, 22], so Huynh [23] introduced the strain reduction factor(SRF), α, rewriting Equation (2.6) to obtain:

$$\dot{\mathbf{A}} = \alpha\left(\dot{\mathbf{A}}^{\mathrm{H}} + \dot{\mathbf{A}}^{\mathrm{IRD}}\right). \tag{2.9}$$

Since the SRF model violates material objectivity requirements, Wang and co-workers [24] proposed the Reduced Strain Closure model (RSC) to slow the orientation kinetics, in order to have a better agreement with experimental observations. This phenomenological model is based on the spectral decomposition of the second order tensor, $\mathbf{A}$, evolution. The model affects the rate-of-growth of the $\mathbf{A}$ eigenvalues, through an empirical slow-down parameter, κ, while keeping the eigenvectors evolution unchanged. The model reads:

$$\dot{\mathbf{A}} = \dot{\mathbf{A}}^{\mathrm{RSC}} + \kappa\dot{\mathbf{A}}^{\mathrm{IRD}}, \tag{2.10}$$

where:

$$\dot{\mathbf{A}}^{\mathrm{RSC}} = \mathbf{W}\cdot\mathbf{A} - \mathbf{A}\cdot\mathbf{W} + \xi(\mathbf{D}\cdot\mathbf{A} + \mathbf{A}\cdot\mathbf{D} - 2[\mathbb{A} + (1-\kappa)(\mathbb{L} - \mathbb{M} : \mathbb{A})] : \mathbf{D}), \tag{2.11}$$

$$\mathbb{L} = \sum_{i=1}^{3}\lambda_i(\mathbf{e}_i\mathbf{e}_i\mathbf{e}_i\mathbf{e}_i), \tag{2.12}$$

$$\mathbb{M} = \sum_{i=1}^{3}(\mathbf{e}_i\mathbf{e}_i\mathbf{e}_i\mathbf{e}_i), \tag{2.13}$$

and

$$\mathbf{A} = \sum_{i=1}^{3}\lambda_i(\mathbf{e}_i\mathbf{e}_i). \tag{2.14}$$

The two models described so far handle the rotary diffusivity as a scalar quantity, however, there is no practical reason to do so. Long fibers exhibit less alignment in the flow-direction [8], and the FT model is not able to adequately reproduce this effect. This motivated the modeling of the rotary diffusivity as a tensor quantity, which allows the rotary diffusion to be anisotropic. This approach was proposed by Phelps et al. [8] in 2009, who built on the work of Phan-Thien et al. [25], to reach the anisotropic rotary diffusion (ARD) model. The model reads:

$$\dot{\mathbf{A}} = \dot{\mathbf{A}}^{\mathrm{H}} + \dot{\mathbf{A}}^{\mathrm{ARD}}, \tag{2.15}$$

$$\dot{\mathbf{A}}^{\mathrm{ARD}} = \dot{\gamma}[2\mathbf{C} - 2\mathrm{tr}(\mathbf{C})\mathbf{A} - 5(\mathbf{C}\cdot\mathbf{A} + \mathbf{A}\cdot\mathbf{C}) + 10\mathbb{A} : \mathbf{C}], \tag{2.16}$$

The rotary diffusion tensor, $\mathbf{C}$, was proposed as a polynomial function of the second-order orientation tensor, $\mathbf{A}$, and the rate-of-deformation, $\mathbf{D}$, tensors:

$$\mathbf{C}(\mathbf{A},\mathbf{D}) = \mathrm{b}_1\mathbf{I} + \mathrm{b}_2\mathbf{A} + \mathrm{b}_3\mathbf{A}^2 + \frac{\mathrm{b}_4}{\dot{\gamma}}\mathbf{D} + \frac{\mathrm{b}_5}{\dot{\gamma}^2}\mathbf{D}^2, \tag{2.17}$$

where the dimensionless parameters, $b_{1,\ldots,5}$, are obtained by fitting experimental data.

Also within the work of Phelps et al. [8] the RSC model was combined with the ARD model to better describe the experimental data, resulting in the ARD-RSC model. This model, which is available in the proprietary software AutoDesk MoldFlow® [13], is equated as:

$$\dot{\mathbf{A}}^{\mathrm{ARD-RSC}} = \dot{\mathbf{A}}^{\mathrm{RSC}} + \dot{\mathbf{A}}^{\mathrm{dARD}} \tag{2.18}$$

$$\begin{aligned}\dot{\mathbf{A}}^{\mathrm{dARD}} = \quad & \dot{\gamma}\{2[\mathbf{C} - (1-\kappa)\mathbb{M} : \mathbf{C}] - 2\kappa\,\mathrm{tr}(\mathbf{C})\mathbf{A} - 5(\mathbf{C}\cdot\mathbf{A} + \mathbf{A}\cdot\mathbf{C}) \\ & +10[\mathbb{A} + (1-\kappa)(\mathbb{L} - \mathbb{M} : \mathbb{A})] : \mathbf{C}\}.\end{aligned} \tag{2.19}$$

Due to inaccurate predictions of the components of the second order orientation tensor, $\mathbf{A}$, for a center-gated disk [26] with the ARD-RSC model, a new model was proposed by Tseng et al. [27, 28], which presented an improved ARD (iARD) calculation combined with a new Retarding Principal Rate (RPR) model, for slowing-down the orientation kinetics. The RPR model considers that the deceleration of the orientation kinetics is due to the fiber-matrix interaction [27], unlike the RSC model, that considers it is due the deformation of the fibers and the fluid [24]. The iARD-RPR model describe the rotary diffusion tensor with a single dependency on the rate-of-deformation tensor, the interaction coefficient proposed by Folgar-Tucker, and a new phenomenological parameter, C_M, describing the fiber-matrix interaction. The model reads:

$$\dot{\mathbf{A}}^{\mathrm{iARD-RPR}} = \dot{\mathbf{A}}^{\mathrm{H}} + \dot{\mathbf{A}}^{\mathrm{ARD}} + \dot{\mathbf{A}}^{RPR}, \tag{2.20}$$

$$\mathbf{C}^{\mathrm{iARD}} = \mathrm{C_I}\left(\mathbf{I} - \mathrm{C_M}\frac{\mathbf{D}^2}{\|\mathbf{D}^2\|}\right), \tag{2.21}$$

$$\mathbf{C} \leftarrow \mathbf{C}^{\mathrm{iARD}}, \tag{2.22}$$

$$\dot{\mathbf{A}}^{\mathrm{RPR}} = -\mathbf{R}\cdot\dot{\mathbf{\Lambda}}^{\mathrm{IOK}}\cdot\mathbf{R}^{\mathrm{T}}, \tag{2.23}$$

$$\dot{\Lambda}_{\mathrm{ii}}^{\mathrm{IOK}} = \alpha\dot{\lambda} \qquad \mathrm{i,j,k} = 1,2,3 \qquad , \tag{2.24}$$

where:

- C_M is a scalar parameter representing the fiber-matrix interaction;
- $\dot{\mathbf{\Lambda}}^{\mathrm{IOK}}$ is the material derivative of a diagonal tensor calculated under the intrinsic orientation kinetics (IOK) assumption [27];
- λ_{i} is the ith eigenvalue of $\mathbf{A}$, such that $(\lambda_1 \geq \lambda_2 \geq \lambda_3)$;
- $\mathbf{R}$ is the rotation matrix built from the eigenvectors columns of $\mathbf{A}$ ($\mathbf{R} = [\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3]$). The superscript T indicates the transpose operator;
- α is a slow-down scalar parameter.

This model is available in the commercial IM simulation software Moldex3D® [27, 28].

Further developments to fiber orientation modelling were made in 2018, when Tseng et al. [29] and Bakharev et al. [30] proposed a similar fiber orientation model, where the anisotropic rotary diffusion tensor was modeled with a single dependency on the second-order fiber orientation tensor.

The models are known as principal Anisotropic rotary diffusion (pARD) [29] and Moldflow Rotational Diffusion (MRD) [30], respectively. In both models the rotary tensor is defined as [31]:

$$\mathbf{C}^{pARD,MRD} = \mathrm{C_I}\mathbf{R}\begin{pmatrix} \mathrm{C}_1 & 0 & 0 \\ 0 & \mathrm{C}_2 & 0 \\ 0 & 0 & \mathrm{C}_3 \end{pmatrix}\mathbf{R}^{\mathrm{T}}. \tag{2.25}$$

However, the choice of parameters and overall equation differs. For the pARD model the RPR model is included and the governing equation is as follows:

$$\dot{\mathbf{A}}^{\mathrm{pARD-RPR}} = \dot{\mathbf{A}}^{\mathrm{H}} + \dot{\mathbf{A}}^{\mathrm{ARD}} + \dot{\mathbf{A}}^{\mathrm{RPR}}, \tag{2.26}$$

$$\mathbf{C} \leftarrow \mathbf{C}^{\mathrm{pARD}}, \tag{2.27}$$

and the authors selected $\mathbf{C}_1 = 1$, $\mathbf{C}_2 = \Omega$ and $\mathbf{C}_3 = 1 - \Omega$ with $0.5 \leq \Omega \leq 1$.

Regarding the MRD model the governing equation reads:

$$\dot{\mathbf{A}}^{\mathrm{MRD}} = \dot{\mathbf{A}}^{\mathrm{H}} + \dot{\mathbf{A}}^{\mathrm{mARD}}, \tag{2.28}$$

$$\dot{\mathbf{A}}^{\mathrm{mARD}} = 2\dot{\gamma}\Big(\mathbf{C}^{\mathrm{MRD}} - \mathrm{tr}(\mathbf{C}^{\mathrm{MRD}})\mathbf{A}\Big). \tag{2.29}$$

As with the pARD model, the coefficients $\mathbf{C}_1$, $\mathbf{C}_2$ and $\mathbf{C}_3$ show a biased rotational diffusion tensor towards the direction of the principal vectors of fiber orientation tensor. In their report [30] the default values identified to provide the best results are $\mathbf{C}_1 = 1$, $\mathbf{C}_2 = 0.5$ and $\mathbf{C}_3 = 0.3$. This model replicates the FT model when all parameters are set to unity.

Wang observed that, for the ARD model, the parameters b_4 and b_5 typically have low values [32], therefore opted by defining b_2, b_4, and b_5 as null, reducing the number of parameters that need fitting. Applying this to Equation 2.17, the rotary diffusion tensor, $\mathbf{C}$, becomes:

$$\mathbf{C}^{\mathrm{Wang}} = \mathrm{b}_1\mathbf{I} + \mathrm{b}_3\mathbf{A}^2. \tag{2.30}$$

Experimental tests, performed by Lambert and Baird [33], for the evolution of fiber orientation under shear and extensional flow showed that, contrary to what was assumed before, κ is dependent on the flow type. While slow orientation kinetics is present for simple shear flow, the same is not observed for extensional flow, and therefore, does not need correction. For this reason, Chen et al. [34] first proposed the idea of a kinetic parameter that changes depending on the flow type, which can unravel the degree of strain imposed on the fibers during shear, extension and mixed flows. The proposed objective parameter reads:

$$\beta = \frac{\mathrm{D}:\mathrm{D}+\overline{\mathrm{W}}:\overline{\mathrm{W}}}{\mathrm{D}:\mathrm{D}-\overline{\mathrm{W}}:\overline{\mathrm{W}}}, \tag{2.31}$$

where:

$$\overline{\mathrm{W}} = -\mathrm{W} - \Omega, \tag{2.32}$$

$$\Omega = \dot{\mathrm{E}} \cdot \mathrm{E}, \tag{2.33}$$

in which Ω is the rate-of-rotation of the principle directions of D, and E is the matrix of right eigenvectors of D. With κ_{e} as the kinetic parameter for extensional flow, and κ_{s} for shear-flow, κ, the scalar parameter in Equation (2.10), is defined as:

$$\kappa = \beta\kappa_{\mathrm{e}} + (1-\beta)\kappa_{\mathrm{s}}. \tag{2.34}$$

More recently, Kugler et al. [35] proposed a different flow type dependent parameter, which resorts to the Manas-Zloczower number, $\mathrm{M_z}$, proposed in Cheng and Manas-Zloczower [36], to characterize the type of flow, which is given by:

$$\mathrm{M_z} = \frac{||\mathrm{D}||}{||\mathrm{D}||+||\mathrm{W}||}, \tag{2.35}$$

where $||\mathrm{D}||$ and $||\mathrm{W}||$ is the magnitude of the respective tensor, defined as:

$$||\mathrm{D}|| = \sqrt{\mathrm{tr}\mathrm{D}^2}, \tag{2.36}$$

$$||\mathrm{W}|| = \sqrt{\frac{1}{2}||\Omega||^2}. \tag{2.37}$$

By assuming that the fiber orientation for mixed shear and elongational flow is a linear combination of the fiber orientation models corresponding to each flow, the proposed model to track the evolution of the second order orientation tensor, $\mathbf{A}$, is given by:

$$\dot{\mathrm{A}} = 2(\mathrm{M_z} - 0.5)\dot{\mathrm{A}}_\mathrm{e} + 2(1 - \mathrm{M_z})\dot{\mathrm{A}}_\mathrm{s}, \tag{2.38}$$

where $\dot{\mathrm{A}}_\mathrm{e}$ and $\dot{\mathrm{A}}_\mathrm{s}$ are the fiber rate of change calculated using the parameters for shear and elongation flow, respectively, with a preferred numerical model for fiber orientation.

Comparison of experimental data with flow dependent models [35, 37] has shown that they yield results with comparable or improved accuracy to other models.

2.3 Closure approximations

As previously mentioned, the use of tensors as a descriptor of fiber orientation requires a closure approximation, to calculate the next even-ordered orientation tensor as a function of the previous, and to be able to calculate the evolution of the previous order tensor (see Equation 2.7)

Hand [38] proposed a fourth-order linear closure, that can be obtained by combining the product of the second order tensor $\mathbf{A}$ and the unit tensor δ. It requires the tensorial expression to be symmetric for any pair of indices, and to meet the normalization condition, $\mathbb{A}_\mathrm{iikl} = \mathbf{A}_\mathrm{kl}$. The proposed relation reads:

$$\begin{aligned}\mathbb{A}^\mathrm{Linear}_{ijkl} &= -\frac{1}{35}\left(\delta_\mathrm{ij}\delta_\mathrm{kl} + \delta_\mathrm{ik}\delta_\mathrm{jl} + \delta_\mathrm{il}\delta_\mathrm{jk}\right)\\ &+\frac{1}{7}\left(\mathbf{A}_\mathrm{ij}\delta_\mathrm{kl} + \mathbf{A}_\mathrm{ik}\delta_\mathrm{jl} + \mathbf{A}_\mathrm{il}\delta_\mathrm{jk} + \mathbf{A}_\mathrm{kl}\delta_{ij} + \mathbf{A}_\mathrm{jl}\delta_\mathrm{ik} + \mathbf{A}_\mathrm{jk}\delta_{il}\right).\end{aligned} \tag{2.39}$$

This relationship represents exactly the isotropic state of orientation [39].

Alternatively, the quadratic closure, proposed by several authors [40–42], is defined by the dyadic the product of the second order tensor with itself:

$$\mathbb{A}^\mathrm{Quadratic} = \mathbf{AA}, \tag{2.40}$$

which is exact for a perfectly aligned fiber orientation state [39].

Since the previously mentioned closure approximations do not provide an accurate solution for all states of fiber orientation, Advani and Tucker [39] proposed subsequently the hybrid closure, which was obtained by blending the linear closure with the quadratic closure, through a blending coefficient,

f.

$$\mathbb{A}^{\text{Hybrid}} = (1-f)\mathbb{A}^{Linear} + f\mathbb{A}^{\text{Quadratic}}$$
$$f = 1 - 27\det(\mathbf{A}). \tag{2.41}$$

The hybrid closure over-predicts the actual fiber alignment, but despite of this, it is still widely used within the industry due to its simplicity and ease of computation [43].

Not satisfied with the solution offered by the hybrid approximation, Verleye and Dupret [44] introduced the natural closure (NAT), that describes $\mathbb{A}$ in terms of $\mathbf{A}$ and the identity second order tensor δ, through a fitting process, which is only valid when the fiber-fiber interactions are neglected (the dilute flow regime), and reads as:

$$\begin{aligned}\mathbb{A}_{\text{ijkl}}^{\text{Natural}} =& \beta_1 S(\delta_{\text{ij}}\delta_{\text{kl}}) + \beta_2 \mathrm{S}(\delta_{\text{ij}}\mathbf{A}_{\text{kl}}) + \beta_3 \mathrm{S}(\mathbf{A}_{\text{ij}}\mathbf{A}_{\text{kl}}) \\ &+ \beta_4 \mathrm{S}(\delta_{\text{ij}}\mathbf{A}_{\text{km}}\mathbf{A}_{\text{ml}}) + \beta_5 \mathrm{S}(\mathbf{A}_{\text{ij}}\mathbf{A}_{\text{km}}\mathbf{A}_{\text{ml}}) + \beta_6 \mathrm{S}(\mathbf{A}_{\text{im}}\mathbf{A}_{\text{mj}}\mathbf{A}_{\text{kn}}\mathbf{A}_{\text{nl}}),\end{aligned} \tag{2.42}$$

where the coefficients $\beta_1 - \beta_6$ are functions of the second and third invariants of $\mathbf{A}$. These coefficients have to satisfy the normalization condition, $\mathbb{A}_{\text{ijkk}} = \mathbf{A}_{\text{ij}}$. S is a function that returns the symmetric part of a 4th order tensor, as:

$$\begin{aligned}\mathrm{S}(\mathbb{T}_{\text{ijkl}}) = \frac{1}{24}\Big(& \mathbb{A}_{\text{ijkl}} + \mathbb{A}_{\text{jikl}} + \mathbb{A}_{\text{ijlk}} + \mathbb{A}_{\text{jilk}} + \mathbb{A}_{\text{klij}} + \mathbb{A}_{\text{lkij}} \\ &+ \mathbb{A}_{\text{klji}} + \mathbb{A}_{\text{lkji}} + \mathbb{A}_{\text{ikjl}} + \mathbb{A}_{\text{kijl}} + \mathbb{A}_{\text{iklj}} + \mathbb{A}_{\text{kilj}} + \mathbb{A}_{\text{jlik}} + \mathbb{A}_{\text{ljik}} + \mathbb{A}_{\text{jlki}} \\ &+ \mathbb{A}_{\text{ljki}} + \mathbb{A}_{\text{iljk}} + \mathbb{A}_{lijk} + \mathbb{A}_{\text{ilkj}} + \mathbb{A}_{\text{likj}} + \mathbb{A}_{\text{jkil}} + \mathbb{A}_{\text{kjil}} + \mathbb{A}_{\text{jkli}} + \mathbb{A}_{\text{kjli}}\Big).\end{aligned} \tag{2.43}$$

It is known that the NAT closure suffers from singularity issues for asymmetric orientation states [44].

Later, Cintra and Tucker [45] developed an orthotropic fitted (ORF) closure approximation and proved the increasing accuracy of this closure approximation over the previous ones. Due to the symmetry restrictions of the second order tensor, the tensor has three orthogonal eigenvectors, that define its principal axes, and the three corresponding eigenvalues, that define its principal values. Since the fourth order tensor can only be obtained trough the information that the second order tensor provides, the fourth order tensor must be orthotropic, and therefore, have the same principal axes as the second order tensor, and the remaining components are a function of the principal values.

$$\hat{\mathbb{A}}_{\mathrm{mm}} = \mathrm{C_m} + \mathrm{C_m^2}\lambda_1 + \mathrm{C_m^3}\lambda_1^2 + \mathrm{C_m^4}\lambda_2 + \mathrm{C_m^5}\lambda_2^2 + \mathrm{C_m^6}\lambda_1\lambda_2 \qquad m = 1,2,3,4,5,6 \qquad , \quad (2.44)$$

where λ_i are the eigenvalues and C_m are coefficients obtained from fitting data obtained from calculations of the distribution function for specific types of flow. $\hat{\mathbb{A}}_{\mathrm{mm}}$ stands for the fourth order tensor in contracted notation, that can be obtained from establishing the following symmetries: $\mathbb{A}_{\mathrm{ijkl}} = \mathbb{A}_{\mathrm{jikl}} = \mathbb{A}_{\mathrm{ijlk}}$, and then the following relationship: $\mathbb{A}_{\mathrm{ijkl}} = \mathbb{A}_{\mathrm{mn}}$, where m and n are related to ij and kl, respectively, according to Table 2.1.

Tensor indices ij	11	22	33	23 or 32	31 or 13	12 or 21
Contracted index m or n	1	2	3	4	5	6

Table 2.1: Relationship between tensor indices and contracted indices.

The ORF approximation performs better than previous closure approximations, but suffers from nonphysical oscillations at low values of C_I.

Chaubal and Leal [46] opted to use a known probability density function to obtain the unknown parameters C_m, the Bingham distribution, and like this created the Bingham closure.

Wetzel [47] and Verweyst [48] introduced another version of an orthotropic fitted closure approximation, ORE, that aimed to improve the ORF closure (Equation(2.44)). The flow data fitted was obtained from analytic solutions corresponding to $C_I = 0$ and $\lambda = 1$, as performed in the NAT closure approximation (Equation (2.42)).

Chung and Kwon [49] also contributed to improve the ORF closure approximation, and were able to remove the nonphysical oscillations. They proposed the ORW and ORW3 closures, in which additional flow data was fitted, to comprise all fiber orientation states.

The previous orthotropic fitted closure approximations (ORF, ORE, ORW and ORW3) are also called eigenvalue based optimal fitting (EBOF) closure approximations. These, when applied to real life flows, require a lot of computational power due the calculation of the eigenvalues [50]. To circumvent this, Chung and Kwon [50] developed the invariant-based optimal fitting (IBOF) closure approximation, that combined the NAT and EBOF closure approximations. They eliminated the singularity problems of the NAT closure, by fitting against data obtained from a more diverse flow types, and the computational burden of EBOF, by using the invariants of the tensor, instead of its eigenvalues. Similar to the NAT approximation, the fourth order orientation tensor is defined in terms of the second order, but the unknown parameters are determined in a similar approach used in EBOF closure approximations.

In the IBOF approximation, the fourth order tensor can be obtained by Equation (2.42), but unlike the natural closure, only the coefficients β_3, β_4 and β_6 are strictly functions of the second and third invariants of $\mathbf{A}$. The remaining coefficients, β_1, β_2 and β_5, are a function of the independent components and the invariants of the second-order orientation tensor, fitted with data obtained from calculations of the PDF. The equations for these coefficients are reported in Appendix A.

The neural network closure, proposed in Jack et al. (NNET) [51], utilizes an artificial neural network to compute the forth-order tensor in function of the corresponding second-order tensor. The artificial neural network is able to map the relationship between input and outputs when a exact mathematical model for this relationship does not exist [51]. For this closure the neural network is trained using known the second-order tensors and the corresponding fourth-order tensors, obtained from experimental data. The neural network closure was shown to have better or equal accuracy to orthotropic closures, and higher efficiency (around 3 times faster than orthotropic closures).

Verleye and Dupret [52] stated that there is an exact closure if the fiber orientation is at one time isotropic. Based on this work, Montgomery-Smith et al. [53] determined the exact closure using Cartesian coordinates on the sphere and the Carlson form of elliptic integrals. Under the assumption that at the initial condition the orientation is isotropic, the second order tensor can be expressed in the following way:

$$\mathbf{A} = \int_S \frac{\mathbf{pp}}{4\pi(\mathbf{B} : \mathbf{pp})^{\frac{3}{2}}} \mathrm{d}\mathbf{p}, \tag{2.45}$$

where

$$\dot{\mathbf{B}} = -\mathbf{B} \cdot (\mathbf{W} + \xi\mathbf{D}) - (-\mathbf{W} + \xi\mathbf{D}) \cdot \mathbf{B}, \tag{2.46}$$

with $\mathbf{B} = \mathbf{I}$ at $t = 0\ s$.

In the same article, Montgomery-Smith et al. [53] also introduced the fast exact closure (FEC), a more computational efficient form of the exact closure. For this, $\mathbf{B}$ is computed via an ordinary differential equation, obtained from solving simultaneously the Equations 2.8 and 2.46. If the initial data is not isotropic, then $\mathbf{B}$ must be calculated for the initial condition by inverting the Equation 2.45.

This format of the FEC is valid only for when the diffusion terms are absent, so later

Montgomery-Smith et al. [54] derived the FEC for other models, which account for diffusion. For this, two fourth-rank tensors, $\mathbb{C}$ and $\mathbb{D}$, defined as conversion tensors, are introduced, such that:

$$\begin{aligned} \frac{\mathrm{D}\mathbf{A}}{\mathrm{Dt}} &= -\mathbb{C} : \frac{\mathrm{D}\mathbf{B}}{\mathrm{Dt}}, \\ \frac{\mathrm{D}\mathbf{B}}{\mathrm{Dt}} &= -\mathbb{D} : \frac{\mathrm{D}\mathbf{A}}{\mathrm{Dt}}. \end{aligned} \tag{2.47}$$

The FEC can be expressed in a general form as:

$$\begin{aligned} \frac{\mathrm{D}\mathbf{A}}{\mathrm{Dt}} &= -\mathbb{C} : \mathrm{F}(\mathbf{B}) + \mathrm{G}(\mathbf{A}) \\ \frac{\mathrm{D}\mathbf{B}}{\mathrm{Dt}} &= \mathrm{F}(\mathbf{B}) - \mathbb{D} : \mathrm{G}(\mathbf{A}) \end{aligned}, \tag{2.48}$$

where the definition of functions F and G depend on the fiber orientation model that is being used [54].

Table 2.2 gives an overview of the presented state-of-the-art for the numerical models and closure approximations.

Name	Acronym	Reference
Numerical models		
Jeffery	-	[14]
Folgar and Tucker	FT	[20]
Advani and Tucker	-	[18]
Strain reduction factor	SRF	[14]
Reduced Strain Closure	RSC	[24]
Anisotropic rotary diffusion	ARD	[8]
improved Anisotropic rotary diffusion	iARD	[27, 28]
Retarding Principal Rate	RPR	[27]
principal Anisotropic rotary diffusion	pARD	[29]
Moldflow Rotational Diffusion	MRD	[30]
Wang	-	[32]
Flow-dependent model	-	[34, 35]
Closure Approximations		
Linear	-	[38]
Quadratic	-	[40–42]
Hybrid	-	[39]
Natural	NAT	[44]
Orthotropic fitted	ORF	[45]
Bingham	-	[46]
Orthotropic fitted Eigenvalues based	ORE	[47, 48]
Orthotropic fitted wide-C_i	ORW	[49]
Invariant-based optimal fitting	IBOF	[50]
Neural network	NNET	[51]
Exact	-	[53]
Fast exact closure	FEC	[53, 54]

Table 2.2: Literature overview of the state-of-art numerical models and closure approximations.

CHAPTER 3 Computational tools

3.1 Python

As a benchmark tool, a script was developed in `python` language to model fiber orientation. This high-level programming language focuses on code readability, is open-source and has a wide range of libraries and tools that can be used for scientific (NumPy [55] and SciPy [56]) and symbolic (Sympy [57]) computations, as well as for data visualization (matplotlib [58]). The results from this script will be validated against data available in the literature and serve as a benchmark for the computed results in OpenFOAM®.

The developed `python` script has the ability of calculating the ordinary derivative of the second-order orientation tensor as a function of time for a given flow field, $\frac{\mathrm{d}\mathbf{A}}{\mathrm{d}t}$. The script can also compute the time-rate-of-change of the second order orientation tensor for a center-gated disk (CGD) as a function of the normalized radius ($\tilde{r}$), given by the ratio between the radius and the disk half-thickness.

Additionally, due the inability of OpenFOAM® to cope with fourth-order tensors, a script was developed to generate code in the C++ programming language for the calculation of the double dot contraction $\mathbb{A} : \mathbf{D}$, an mathematical operation present in all phenomenological fiber orientation models.

The code is available online in GitHUB [59] and will be described in the following subsections.

3.1.1 Calculation of fiber orientation in a given flow field

Listing 3.1 shows the `python` script main function, programmed to calculate the second-order tensor evolution along time, for simple-shear flow, with the Folgar-Tucker model (2.6) and hybrid closure approximation (2.41).

```
 1 import numpy as np
 2 from scipy.integrate import solve_ivp
 3 import fiberOrientationModels as FOM
 4
 5 def gradU(t):
 6     return np.array([[0.0,  1.0, 0.0],
 7                      [0.0,  0.0, 0.0],
 8                      [0.0,  0.0, 0.0]])
 9
10 time = np.arange(0,10.1,0.01)
11
12 A2_0 = np.eye(3)*(1/3)
13
14 closure = 'hybrid'
15
16 xi = 1
17
18 CI = 0.01
19
20 sol = solve_ivp(
21                     FOM.FT,
22                     (time[0], time[-1]),
23                     A2_0.ravel(),
24                     method="RK45",
25                     t_eval=time,
26                     rtol=1e-6,
27                     args=(gradU, closure, xi, CI)
28                )
29
30 A2=np.transpose(sol.y)
```

Listing 3.1: Main function of the python script for a simple flow.

In Lines 1-3, the required libraries and modules are imported. `Numpy` is used for standard numerical computations and data structures creation, the `solve_ivp` routine is imported from the `scipy` library to solve the ordinary differential equation (ODE), and the custom module `fiberOrientationModels` that contains the implemented fiber orientation models.

The `solve_ivp` [60] routine is an ordinary differential equation(ODE) solver for initial value problems (IVP) in the form $\frac{\mathrm{d}\mathbf{A}}{\mathrm{d}t} = RHS$. This routine takes as input:

- A function that allows calculating the right-hand size of the system;
- The time interval to be considered $\left[t_0,\, t_f\right]$;
- The initial condition, $\mathbf{A}_0$

- the numerical technique to solve the initial value problem;
- Solver specific parameters (e.g., absolute/relative tolerances, max time-step, evaluation times, etc.);
- The parameters required by the ODE.

Lines 5-8 show the definition of the velocity gradient tensor function. By design, this function is defined as a function of time t to allow the consideration of a general time-varying velocity gradient. Lines 10-12 define the integration parameters, that are the time interval and initial value of the second order tensor, and in Lines 16 and 18 the fiber orientation model (FOM) dependent parameters. Finally, in lines 20-28 the numerical integration routine is used through a Runga-Kutta method of order 5(4) and the solution is stored for post-processing in Line 30.

The beginning of the custom module `fiberOrientationModels` is shown in Listing 3.2. In line 2, similarly to was done in the `main` function, a costume module is imported, which contains the closure approximations.

Line 85 marks the start of the function that contains the Folgar-Tucker model. Firstly, in line 8, the fourth-order tensor is calculated in function of the selected closure and the current value of the second-order tensor. In line 10, the rate-of-deformation tensor is calculated, in line 12, the vorticity tensor, and, in line 14, the shear rate scalar. The Folgar-Tucker method is then calculated, following the Equation (2.6).

```
 1 import numpy as np
 2 from fiberOrientationClosures import compute_closure
 3
 4 def FT(t, A2, gradU, closure, xi, CI):
 5
 6     A2 = np.reshape(A2, (3, 3))
 7
 8     A4 = compute_closure(A2, closure)
 9
10     D = 0.5*(gradU(t)+np.transpose(gradU(t)))
11
12     W = 0.5*(gradU(t)-np.transpose(gradU(t)))
13
14     shrRate = np.sqrt(2*np.tensordot(D,D,axes=2))
15
16     I = np.eye(3)
17
18     FT = (
19             (np.tensordot(W,A2,axes=1)-np.tensordot(A2,W,axes=1))
```

```
            +xi*(np.tensordot(D,A2,axes=1)+np.tensordot(A2,D,axes=1)
            -2*np.tensordot(A4,D,axes=2))
            +6*(CI*shrRate)*(I/3-A2)
         )

    return FT.ravel()
```

Listing 3.2: Custom module **fiberOrientationModels** which contains the fiber orientation models.

Listing 3.3 shows exerts of the **fiberOrientationClosures** module, in which the function **compute_closure** resides. Inside this function, the closures are defined inside a **if...else** block.

```
def compute_closure(A2, closure):

    if (closure=='linear'):
```

```
    elif (closure=='quadratic'):
```

```
    elif (closure=='hybrid'):
```

```
    elif (closure=='IBOF'):
```

```
    elif (closure=='ORE'):

        lambda_, Q = np.linalg.eig(A2)

        idx = lambda_.argsort()[::-1]
        lambda_ = lambda_[idx]
        Q = Q[:,idx]

        A4 = np.zeros((3,3,3,3))

        Cm = np.array ([[0.636256796880687,   0.636256796880687,   2.74053289560253],
                        [-1.8726629637381   ,  -3.31527229742146 ,  -9.12196509782692],
                        [-4.47970873193738 ,  -3.03709939825406 , -12.2570587036254] ,
                        [11.9589562332320   ,  11.8273285968852   ,  34.3199018916987] ,
                        [3.84459692420086 ,   6.88153952058044 ,  13.8294699121940] ,
                        [11.3420924278159   ,   8.43677746778325 ,  25.8684755253884] ,
                        [-10.9582626069691   , -15.9120667157641   , -37.7029118029384] ,
                        [-20.7277994684132   , -15.1515872606307   , -50.2756431927485] ,
                        [-2.11623214471004 ,  -6.48728933641926 , -10.8801761133174] ,
                        [-12.3875632855619   ,  -8.63891419284016 , -26.9636915239716] ,
                        [9.81598389716748 ,   9.32520343452661 ,  27.3346798054488] ,
                        [3.47901510567439 ,   7.74683751713295 ,  15.2650686148651] ,
                        [11.7492911177026   ,   7.48146870624441 ,  26.1134914005375] ,
                        [0.508041387366637,   2.28476531637958 ,   3.4321384033477] ,
                        [4.88366597771489 ,   3.59772251134254 ,  10.6117418066060] ])
```

```
267
268         for i in range(3):
269             A4[i,i,i,i] =  Cm[0,i] + (Cm[1,i] * lambda_[0]) + (Cm[2,i] * lambda_[1])
270             + (Cm[3,i] * lambda_[0] * lambda_[1]) + (Cm[4,i] * np.power(lambda_[0],2))
271             + (Cm[5,i] * np.power(lambda_[1],2))
272             + (Cm[6,i] * np.power(lambda_[0],2) * lambda_[1])
273             + (Cm[7,i] * lambda_[0] * np.power(lambda_[1],2))
274             + (Cm[8,i] * np.power(lambda_[0],3)) + (Cm[9,i] * np.power(lambda_[1],3))
275             + (Cm[10,i] * np.power(lambda_[0],2) * np.power(lambda_[1],2))
276             + (Cm[11,i] * np.power(lambda_[0],3) * lambda_[1])
277             + (Cm[12,i] * lambda_[0] * np.power(lambda_[1],3))
278             + (Cm[13,i] * np.power(lambda_[0],4) ) + (Cm[14,i] * np.power(lambda_[1],4))
279
280         A4[1,2,1,2] = 0.5*(1.0 - 2.0*lambda_[0] + A4[0,0,0,0] - A4[1,1,1,1] - A4[2,2,2,2])
281         A4[2,0,2,0] = 0.5*(1.0 - 2.0*lambda_[1] - A4[0,0,0,0] + A4[1,1,1,1] - A4[2,2,2,2])
282         A4[0,1,0,1] = 0.5*(-1.0 + 2.0*lambda_[0] + 2.0*lambda_[1] - A4[0,0,0,0]
283         - A4[1,1,1,1] + A4[2,2,2,2])
284
285         A4[2,1,2,1] = A4[1,2,2,1] = A4[2,1,1,2] = A4[1,2,1,2]
286         A4[0,2,0,2] = A4[2,0,0,2] = A4[0,2,2,0] = A4[2,0,2,0]
287         A4[1,0,1,0] = A4[1,0,0,1] = A4[0,1,1,0] = A4[0,1,0,1]
288
289         A4[1,1,0,0] = A4[0,0,1,1] = A4[0,1,0,1]
290         A4[2,2,1,1] = A4[1,1,2,2] = A4[1,2,1,2]
291         A4[0,0,2,2] = A4[2,2,0,0] = A4[2,0,2,0]
292
293         A4_ORE=np.einsum("im,jn,ko,lp,mnop->ijkl", Q, Q, Q, Q, A4)
294
295         return A4_ORE
```

Listing 3.3: Closure approximations defined in a "if" cycle.

After line 242, the definition of the ORE closure (2.44) is defined. First, the eigenvalues, `lambda_`, and eigenvectors, `Q`, are calculated (line 244) and afterwards they are organized in descending order (lines 246-247) of the eigenvalues. In line 250 an empty fourth-order tensor is created, `A4`, in which the components will be stored, after being calculated. In a `Numpy` array (Lines 252-266) the values of the coefficients obtained from fitting data, as explained in Section 2.3, are stored. From lines 268 to 278, the 3 independent components are calculated following Equation 2.44, and in Lines 280-283 the remaining independent components are calculated, following the contraction property of the fourth-order tensor ($\sum_{k=1}^{3} \mathbb{A}_{ijkk} = \mathbf{A}_{ij}$ [7]).

With all the independent components calculated, the remaining tensor components are populated by following its symmetries (Lines 285-291).

As a last step, the fourth order tensor is transformed from the principal coordinates to laboratory coordinates. For this a rotation matrix, $\mathbf{Q}$, is constructed. This operation is done in Line 293, through the `np.einsum` [61] routine. In line 295, the final calculation is `return` to the user.

A similar approach is applied for the calculation of the other closure approximations.

3.1.2 Calculation of fiber orientation in a center-gated disk

The evolution of the second-order tensor, $\mathbf{A}$, along the normalized radius of a center gated disk, $\frac{\mathrm{d}\mathbf{A}}{\mathrm{d}\tilde{r}}$, can be calculated by doing simple modifications to the code provided in Listing 3.1.

The velocity vector and velocity gradient tensor corresponding to this axisymetric flow can be found in Chung and Kwon [50], and are defined in Equations (3.1) and (3.2) respectively.

$$\mathbf{u}_r = \begin{bmatrix} \frac{3Q}{8\pi r b}\left(1-\frac{z^2}{b^2}\right) \\ 0 \\ 0 \end{bmatrix} \tag{3.1}$$

$$\frac{\partial \mathbf{u}_i}{\partial x_j} = \frac{3Q}{8\pi r b}\begin{bmatrix} -\frac{1}{r}\left(1-\frac{z^2}{b^2}\right) & 0 & -\frac{2}{b}\left(\frac{z}{b}\right) \\ 0 & \frac{1}{r}\left(1-\frac{z^2}{b^2}\right) & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.2}$$

where Q is the volumetric flux in, r is the radius of the disk, z, its thickness, and b, the half-thickness. In Figure 3.1, a center-gated disk is shown in reference to the coordinates r and z.

In Lines 5-9 of Listing 3.4 the velocity vector, $\mathbf{u}$, is defined as presented above. The velocity in a disk depends on the radius, r, and on the thickness, z. The velocity gradient is defined as a function of the radius and thickness in lines 11-21. Now the calculation should be performed in the following range $r \in \left[r_0, r_f\right]$, which is defined in Line 24, and the thickness z is defined in Line 26.

```
1 import fiberOrientationModels as FOM
2 import numpy as np
3 from scipy.integrate import solve_ivp
4
```

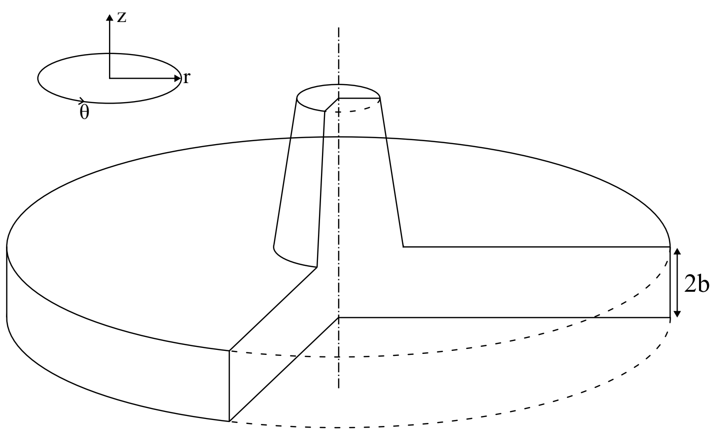

Figure 3.1: Scheme of a center-gated disk.

```python
def U(r,z):
        Q = 1e-3
        b = 0.0015
        return ( (3*Q)/(8*np.pi*r*b) ) *
                 ( 1-(np.power(z,2)/np.power(b,2)) )

def gradU(r, z):
        Q = 1e-3
        b = 0.0015
        Qt= (3.0*Q)/(8.0*np.pi*r*b)

        tmp1=(1/r)*(1-(np.power(z,2)/np.power(b,2)))
        tmp2=-(2/b)*(z/b)

        return Qt*np.array([[-tmp1, 0.0, tmp2],
                            [0.0, tmp1, 0.0],
                            [0.0, 0.0, 0.0]])

r = np.arange(0.01,0.1,0.0001)

z = 0.1

A2_0 = np.eye(3)*(1/3)

closure = 'hybrid'

xi = 1

CI = 0.01

sol = solve_ivp(
                    FOM.FT,
                    (r[0], r[-1]),
                    A2_0.ravel(),
                    method="RK45",
                    t_eval=r,
                    rtol=1e-6,
                    args=(gradU, closure, xi, CI)
               )

A2=np.transpose(sol.y)
```

Listing 3.4: Main function of the python script for the flow in a center-gated disk.

Since $\frac{\mathrm{d}\mathbf{A}}{\mathrm{d}\tilde{r}}$ is the equation to be solved, instead of $\frac{\mathrm{d}\mathbf{A}}{\mathrm{d}t}$ as done in 3.1.1, the relation between these two derivatives has to be devised, which can be obtained by resorting to the chain rule.

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y} \cdot \frac{\mathrm{d}y}{\mathrm{d}x} \tag{3.3}$$

By knowing that the velocity depends either on $\tilde{r}$ or t, whose relation, $t(\tilde{r})$, is known, the following equality is obtained:

$$\begin{aligned} \frac{\mathrm{d}\mathbf{A}}{\mathrm{d}\tilde{r}} &= \frac{\mathrm{d}\mathbf{A}}{\mathrm{d}t} \cdot \frac{\mathrm{d}t}{\mathrm{d}\tilde{r}} \\ &= \frac{\mathrm{d}\mathbf{A}}{\mathrm{d}t} \cdot \frac{1}{u(r,z)} \end{aligned} \tag{3.4}$$

This operation is done in Line 26 of the Listing 3.5.

```
1  import numpy as np
2  from fiberOrientationClosures import compute_closure
3
4  def FT(r, A2, gradU, U, closure, xi, CI, z):
5
6      A2 = np.reshape(A2, (3, 3))
7
8      A4 = compute_closure(A2, closure)
9
10     D = 0.5*(gradU(r,z)+np.transpose(gradU(r,z)))
11
12     W = 0.5*(gradU(r,z)-np.transpose(gradU(r,z)))
13
14     shrRate = np.sqrt(0.5*np.tensordot(D,D,axes=2))
15
16     I = np.eye(3)
17
18     FT = (
19             (np.tensordot(W,A2,axes=1)-np.tensordot(A2,W,axes=1))
20             +xi*
```

```
            (np.tensordot(D,A2,axes=1)+np.tensordot(A2,D,axes=1)
            -2*np.tensordot(A4,D,axes=2))
            +6*(CI*shrRate)*(I/3-A2)
        )

    FT_U = FT * (1/U(r,z))

    return FT_U.ravel()
```

Listing 3.5: Example of a model for fiber orientation in terms of the radius of a disk.

3.1.3 Symbolic computation of the double dot contraction $\mathbb{A} : \mathbf{D}$

Symbolic computation refers to the manipulation and analysis of mathematical expressions using symbols rather than numerical approximations, allowing for exact and symbolic solutions to mathematical problems. Furthermore, symbolic algorithms are capable of simplifying the computed expressions [62].

As mentioned before, OpenFOAM® cannot handle fourth-order tensors, and therefore, the closure approximation cannot be computed in OpenFOAM®. But, since the result of a double dot contraction between a fourth order and seconder tensor is a second order tensor, which is the operation present in all models, $\mathbb{A} : \mathbf{D}$, OpenFOAM® can handle the result of this operation, providing just second order tensors are employed in the code.

Therefore, symbolic computation was used to create a formula that expressed the required double dot contraction, $\mathbb{A} : \mathbf{D}$, as function of the second-order tensor $\mathbf{A}$ and $\mathbf{D}$. In the case of the ORE closure, $\mathbb{A} : \mathbf{D}$ is also dependent on the eigenvectors.

In Listing 3.6, it is shown how the needed variables can be defined using `sympy`.

```
import sympy as sym

def make_sym(T):
    T[1,0]=T[0,1]
    T[2,0]=T[0,2]
    T[2,1]=T[1,2]

    return T

a2 = sym.MatrixSymbol('A',3,3)
A2 = make_sym(sym.Matrix(a2))

d = sym.MatrixSymbol('D',3,3)
```

```
14 D = make_sym(sym.Matrix(d))
15
16 evals = sym.MatrixSymbol('eVals',1,3)
17 eVals = sym.Matrix(evals)
18
19 evecs = sym.MatrixSymbol('eVecs',3,3)
20 eVecs = sym.Matrix(evecs)
21
22 eVals_tmp=eVals.copy()
23 eVals[0]=eVals_tmp[2]
24 eVals[2]=eVals_tmp[0]
25
26 eVecs_tmp=eVecs.copy()
27 eVecs[0,:]=eVecs_tmp[2,:]
28 eVecs[2,:]=eVecs_tmp[0,:]
29 eVecs = sym.transpose(eVecs)
30
31 a = sym.Symbol('alpha')
32 b = sym.Symbol('beta')
33
34 k=sym.Symbol('k')
```

Listing 3.6: Defining variables with sympy.

Since both $\mathbf{A}$ and $\mathbf{D}$ are symmetric tensors, the numbers of tensor components that have to be computed and stored can be reduced. The function used to impose symmetry is defined in Lines 3-6. In Lines 10-14 the variables used to store the tensors $\mathbf{A}$ (tensor `A2`)and $\mathbf{D}$ (tensor `D`)are defined, and in Lines 16-20, the same happens for the variables used to store the eigenvalues (vector `eVals`) and eigenvectors (vector `eVecs`). Additionally, scalar parameters used by the fiber orientation models are defined in Lines 31-34.

The eigenvalues must be organized in descending order($\lambda_1 \geq \lambda_2 \geq \lambda_3$), and the corresponding eigenvectors as column vectors. However, the OpenFOAM® routines that computes the eigenvalues and eigenvectors sort the values in ascending order and store the eigenvectors as row vectors, so they must be reorganized. This operation could be done within OpenFOAM®, but for efficiency reasons, it was chosen to be done within the symbolic computation. In Lines 22-29 the eigenvalues and eigenvectors are reordered, and the eigenvectors tensor is reshaped to be stored in the tensor rows.

Listing 3.7 shows how the computation of a closure approximation was done in symbolic code. The closure approximations are defined as functions.

```
1 import numpy as np
2 import sympy as sym
```

```
def ORE(A2, eVecs, eVals):

    A4 = sym.MutableDenseNDimArray(np.zeros((3,3,3,3)))

    Cm = sym.Matrix ([[0.636256796880687,   0.636256796880687,   2.74053289560253],
                     [-1.8726629637381   ,  -3.31527229742146 ,  -9.12196509782692],
                     [-4.47970873193738 ,  -3.03709939825406 , -12.2570587036254] ,
                     [11.9589562332320   ,  11.8273285968852   ,  34.3199018916987] ,
                     [3.84459692420086 ,   6.88153952058044 ,  13.8294699121940] ,
                     [11.3420924278159   ,   8.43677746778325 ,  25.8684755253884] ,
                     [-10.9582626069691   , -15.9120667157641   , -37.7029118029384] ,
                     [-20.7277994684132   , -15.1515872606307   , -50.2756431927485] ,
                     [-2.11623214471004 ,  -6.48728933641926 , -10.8801761133174] ,
                     [-12.3875632855619   ,  -8.63891419284016 , -26.9636915239716] ,
                     [9.81598389716748 ,   9.32520343452661 ,  27.3346798054488] ,
                     [3.47901510567439 ,   7.74683751713295 ,  15.2650686148651] ,
                     [11.7492911177026   ,   7.48146870624441 ,  26.1134914005375] ,
                     [0.508041387366637,   2.28476531637958 ,   3.4321384033477] ,
                     [4.88366597771489 ,   3.59772251134254 ,  10.6117418066060] ])

    for i in range(3):
        A4[i,i,i,i] =  Cm[0,i] + (Cm[1,i] * eVals[0]) + (Cm[2,i] * eVals[1]) +
        (Cm[3,i] * eVals[0] * eVals[1]) + (Cm[4,i] * np.power(eVals[0],2)) +
        (Cm[5,i] * np.power(eVals[1],2)) + (Cm[6,i] * np.power(eVals[0],2) * eVals[1]) +
        (Cm[7,i] * eVals[0] *
        np.power(eVals[1],2)) + (Cm[8,i] * np.power(eVals[0],3)) + (Cm[9,i] *
        np.power(eVals[1],3)) + (Cm[10,i] * np.power(eVals[0],2) *
        np.power(eVals[1],2)) + (Cm[11,i] * np.power(eVals[0],3) * eVals[1]) +
        (Cm[12,i] * eVals[0] * np.power(eVals[1],3)) + (Cm[13,i] *
        np.power(eVals[0],4) ) + (Cm[14,i] * np.power(eVals[1],4))

    A4[1,2,1,2] = 0.5*( 1.0 - 2.0*eVals[0] + A4[0,0,0,0] - A4[1,1,1,1] - A4[2,2,2,2] )
    A4[2,0,2,0] = 0.5*( 1.0 - 2.0*eVals[1] - A4[0,0,0,0] + A4[1,1,1,1] - A4[2,2,2,2] )
    A4[0,1,0,1] = 0.5*( -1.0 + 2.0*eVals[0] + 2.0*eVals[1] - A4[0,0,0,0] - A4[1,1,1,1]
    + A4[2,2,2,2] )

    A4[2,1,2,1] = A4[1,2,2,1] = A4[2,1,1,2] = A4[1,2,1,2]
    A4[0,2,0,2] = A4[2,0,0,2] = A4[0,2,2,0] = A4[2,0,2,0]
    A4[1,0,1,0] = A4[1,0,0,1] = A4[0,1,1,0] = A4[0,1,0,1]

    A4[0,0,1,1] = A4[0,1,0,1]
    A4[1,1,0,0] = A4[0,0,1,1]

    A4[1,1,2,2] = A4[1,2,1,2]
    A4[2,2,1,1] = A4[1,1,2,2]

    A4[2,2,0,0] = A4[2,0,2,0]
    A4[0,0,2,2] = A4[2,2,0,0]
```

```
304
305     A4_ORE = sym.MutableDenseNDimArray(np.zeros((3,3,3,3)))
306
307     for i in range(3):
308         for j in range(3):
309             for k in range(3):
310                 for l in range(3):
311                     for m in range(3):
312                         for n in range(3):
313                             for o in range(3):
314                                 for p in range(3):
315                                     A4_ORE[i,j,k,l] += eVecs[i,m]*eVecs[j,n]
316                                                     *eVecs[k,o]*eVecs[l,p]*A4[m,n,o,p]
317
318     return A4_ORE
```

Listing 3.7: ORE closure aprroximattion expressed in symbolic code.

Firstly, the necessary libraries are imported. As done for the scientific code, in line 258 a empty tensor is created, but the **sympy** library is used to define it, so that it can be employed in symbolic calculations, as well as the necessary coefficients (lines 260-274), that are stored in a **sympy** matrix. The independent components are calculated and the tensor is populated following its symmetries, as it was previously done. Lastly, the transformation of coordinates as to be done. **sympy** does not have a routine similar to **np.einsum**, therefore a "for" loop was used for this operation.

In Listing 3.8 it is shown how the double contraction is expressed in symbolic code. Firstly, the **sympy** library and custom modules are imported. The file **variables** was shown previously in Listing 3.6, and **fiberOrientationClosures** contains the closures approximations, as Listing 3.7.

```
import sympy as sym
from variables import *
from fiberOrientationClosures import *
from fiberOrientationClosures_RSC import tensorCombination
from generate_code import generateCCode

closure = hybrid(A2)

D_doubleDot_A4 = sym.MutableDenseMatrix
        (sym.tensorcontraction(sym.tensorproduct(hybrid(A2), D), (0,4),(1,5)))

generateCCode(D_doubleDot_A4, nDecimalCases=17, OFSyntax=True)
```

Listing 3.8: Computation of the double contraction $\mathbb{A} : \mathbf{D}$.

The selected closure is indicated in Line 7. In line 9 and 10, the double contraction operation is done. The function generateCCode, from the module generate_code, is responsible for the optimization of the symbolic code, as well as its output to the terminal. This module makes use of sympy's Pow expansion algorithm [63] , with which simple powers up to 6 are expanded, as well as the cse routine, abbreviation for Common Subexpression Detection and Collection, which allows for the identification of common subexpressions and their evaluation only once, which increases computation efficiency.

The obtained code is shown in Listing 3.9.

```
------------D_doubleDot_A4------------
const scalar tmp0 = A.yz()*A.yz();
const scalar tmp1 = A.xy()*A.xy();
const scalar tmp2 = 27*A.zz();
const scalar tmp3 = A.xz()*A.xz();
const scalar tmp4 = 27*tmp0*A.xx() + tmp1*tmp2 - tmp2*A.xx()*A.yy() + 27*tmp3*A.yy()
        - 54*A.xy()*A.xz()*A.yz();
const scalar tmp5 = tmp4 + 1;
const scalar tmp6 = -tmp4;
const scalar tmp7 = (0.4285714285714286)*tmp6;
const scalar tmp8 = tmp7*A.xy();
const scalar tmp9 = tmp5*A.xx();
const scalar tmp10 = tmp8 + tmp9*A.xy();
const scalar tmp11 = 2*D.xy();
const scalar tmp12 = tmp7*A.xz();
const scalar tmp13 = tmp12 + tmp9*A.xz();
const scalar tmp14 = 2*D.xz();
const scalar tmp15 = (0.1428571428571429)*tmp6;
const scalar tmp16 = tmp15*A.yz();
const scalar tmp17 = tmp16 + tmp9*A.yz();
const scalar tmp18 = 2*D.yz();
const scalar tmp19 = (0.1428571428571429)*A.yy();
const scalar tmp20 = (0.1428571428571429)*A.xx() - 0.02857142857142857;
const scalar tmp21 = tmp6*(tmp19 + tmp20);
const scalar tmp22 = tmp21 + tmp9*A.yy();
const scalar tmp23 = (0.1428571428571429)*A.zz();
const scalar tmp24 = tmp6*(tmp20 + tmp23);
const scalar tmp25 = tmp24 + tmp9*A.zz();
const scalar tmp26 = tmp15*A.xy();
const scalar tmp27 = tmp5*A.xy();
const scalar tmp28 = tmp26 + tmp27*A.zz();
const scalar tmp29 = tmp27*A.yy() + tmp8;
const scalar tmp30 = tmp15*A.xz();
const scalar tmp31 = tmp27*A.yz() + tmp30;
const scalar tmp32 = tmp16 + tmp27*A.xz();
const scalar tmp33 = tmp10*D.xx() + tmp11*(tmp1*tmp5 + tmp21) + tmp14*tmp32 + tmp18*tmp31
        + tmp28*D.zz() + tmp29*D.yy();
```

```
const scalar tmp34 = tmp5*A.xz();
const scalar tmp35 = tmp30 + tmp34*A.yy();
const scalar tmp36 = tmp12 + tmp34*A.zz();
const scalar tmp37 = tmp26 + tmp34*A.yz();
const scalar tmp38 = tmp11*tmp32 + tmp13*D.xx() + tmp14*(tmp24 + tmp3*tmp5)
        + tmp18*tmp37 + tmp35*D.yy() + tmp36*D.zz();
const scalar tmp39 = tmp7*A.yz();
const scalar tmp40 = tmp5*A.yy();
const scalar tmp41 = tmp39 + tmp40*A.yz();
const scalar tmp42 = tmp6*(tmp19 + tmp23 - 0.02857142857142857);
const scalar tmp43 = tmp40*A.zz() + tmp42;
const scalar tmp44 = tmp39 + tmp5*A.yz()*A.zz();
const scalar tmp45 = tmp11*tmp31 + tmp14*tmp37 + tmp17*D.xx() + tmp18*(tmp0*tmp5 + tmp42)
        + tmp41*D.yy() + tmp44*D.zz();

result.xx() = tmp10*tmp11 + tmp13*tmp14 + tmp17*tmp18 + tmp22*D.yy() + tmp25*D.zz()
        + D.xx()*(tmp5*(A.xx()*A.xx())
        + tmp6*((0.8571428571428571)*A.xx() - 0.08571428571428572));
result.xy() = tmp33;
result.xz() = tmp38;
result.yy() = tmp11*tmp29 + tmp14*tmp35 + tmp18*tmp41 + tmp22*D.xx() + tmp43*D.zz()
        + D.yy()*(tmp5*(A.yy()*A.yy())
        + tmp6*((0.8571428571428571)*A.yy() - 0.08571428571428572));
result.yz() = tmp45;
result.zz() = tmp11*tmp28 + tmp14*tmp36 + tmp18*tmp44 + tmp25*D.xx() + tmp43*D.yy()
        + D.zz()*(tmp5*(A.zz()*A.zz())
        + tmp6*((0.8571428571428571)*A.zz() - 0.08571428571428572));
```

Listing 3.9: Example of code obtained for computation of the double contraction $\mathbb{A} : \mathbf{D}$ with the Hybrid closure.

3.2 OpenFOAM®

OpenFOAM®, also known as Open-source Field Operation And Manipulation, is an open-source computational library with a wide range of numerical solvers able to simulate different continuum mechanics problems, with special focus on computational fluid dynamics (CFD) problems [64]. OpenFOAM® is programmed in C++, an object oriented programming language. This allows for modularity and easy code re-usability.

The library, that also includes several tools for pre and post-processing tasks, is under continuous active development with two versions being released annually. The open-source character allows the end-user full access to the code, making it available for scientific scrutiny, continuous improvement and customization. For this reason, OpenFOAM® has a growing user base across different areas of industries, academic institutes and research organizations [64].

3.2.1 Finite Volume Method in OpenFOAM®

To numerically solve partial differential equations, OpenFOAM® resorts to the Finite Volume Method (FVM) [65]. This numerical approach discretizes the equations by partitioning the domain of the problem in a set of control volumes (or computational cells). Then, conservation equations, for example, of mass, momentum and energy, are applied to every control volume of the discretized domain, and by calculating the conserved quantity exchanged through the cell faces and source contributions, linear algebraic equations are obtained for each cell. These equations are then assembled and solved to obtain the numerical solution [66].

The set of control volumes from a domain form the computational mesh, being each delimited by planar faces, which can have different polyhedral shapes. The faces can be either internal, which are shared by two control volumes, or external, boundary faces. OpenFOAM® uses a co-located grid, that is, the problem unknowns are stored in a single point of the control volume, which is the centroid [65]. The size of the control volumes, or, the degree of mesh refinement, can be defined by the user, who looks for a balance between accuracy of the results, improved by refining the mesh, and computational power required, which is less for more coarse meshes.

3.2.2 General Case Set Up

OpenFOAM® does not have a graphical user interface, therefore a case is defined by a number of files distributed by a set of folders. In the following scheme, a general set up case is shown.

- Case
  - 0
    - U
    - p

    The 0 folder contains the initial conditions of the different variables, such as velocity and pressure.
  - constant
    - polyMesh
    - transportProperties
    - turbulenceProperties

    In this folder, the files that contain material proprieties, such as density, can be found, and also others properties, such as the presence of turbulence. Additionally, the folder polyMesh can also be found, which contains information about the mesh.
  - system
    - controlDict
    - fvSchemes
    - fvSolution

    The controlDict file controls the start and end point of the simulation, as well as its time step, and how the data is saved. The fvSchemes file sets the discretization schemes for each term of the equation being solved. The tolerances and algorithms for the solution can be selected on the fvSolution dictionary, as well as the residual control for different variables.

The case set up and launch is executed trough command prompts. For the visualization of the data, Paraview [67], an open-source post-processing visualization engine, is usually employed.

3.2.3 Development of a OpenFOAM® utility

python is only capable of solving the fiber orientation in function of time, $\frac{\mathrm{d}\mathbf{A}}{\mathrm{d}t}$, but the fiber orientation depends on space too. OpenFOAM® is capable of solving material derivatives, which describe the rate of change in function of time and space.

The material derivative, $\frac{D\mathbf{A}}{Dt}$, can be expressed as [7]:

$$\frac{D\mathbf{A}}{Dt} = \frac{\partial \mathbf{A}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{A}. \tag{3.5}$$

However, the term $\mathbf{u} \cdot \nabla \mathbf{A}$ cannot be solved implicitly, since OpenFOAM® does not support tensors beyond second-order, as is the case of the third-order tensor $\nabla \mathbf{A}$. For this reason and having in mind the following relation:

$$\nabla \cdot (\mathbf{u}\mathbf{A}) = (\nabla \cdot \mathbf{u})\mathbf{A} + \mathbf{u} \cdot \nabla \mathbf{A}, \tag{3.6}$$

the term that cannot be calculated in Equation (3.5) is replaced by:

$$\mathbf{u} \cdot \nabla \mathbf{A} = \nabla \cdot (\mathbf{u}\mathbf{A}) - (\nabla \cdot \mathbf{u})\mathbf{A}. \tag{3.7}$$

Therefore, the governing equation to be solved in OpenFOAM® will be:

$$\begin{aligned}\frac{\partial \mathbf{A}}{\partial t} + \nabla \cdot (\mathbf{u}\mathbf{A}) - (\nabla \cdot \mathbf{u})\mathbf{A} = \mathbf{W} \cdot \mathbf{A} - \mathbf{A} \cdot \mathbf{W} + \xi(\mathbf{D} \cdot \mathbf{A} + \mathbf{A} \cdot \mathbf{D} - 2\mathbb{A} : \mathbf{D}) \\ + \dot{\gamma}[2\mathbf{C} - 2(\mathrm{tr}\mathbf{C})\mathbf{A} - 5(\mathbf{C} \cdot \mathbf{A} + \mathbf{A} \cdot \mathbf{C}) + 10\mathbb{A} : \mathbf{C}].\end{aligned} \tag{3.8}$$

In Figure 3.2, the methodology that the code follows to solve the $\dot{\mathbf{A}}$ is presented. Firstly, a solver capable of calculating the velocity field of the problem and geometry of interest is required. Generally, such solvers employ the PIMPLE algorithm, that is a segregated algorithm able to calculate the velocity and pressure fields in general flows.

Using the velocity field provided by the solver, the developed OpenFOAM® code is capable of calculating the fiber orientation evolution induced by the calculated flow field. To allow the user the selection of fiber orientation model and closure approximation that they wish to be implemented, the factory method was employed. By calling the factory method instead of a constructor, an object can be

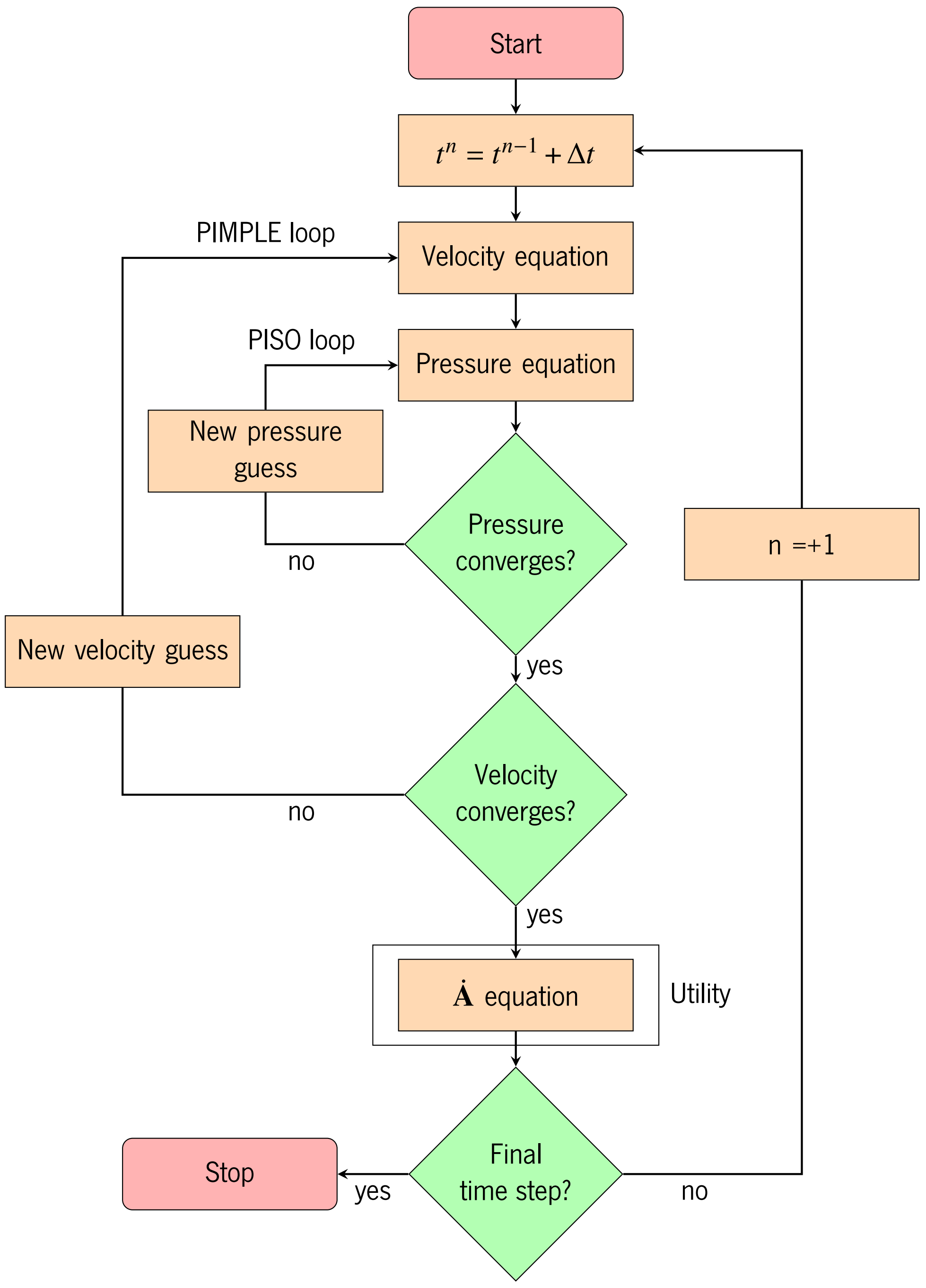

Figure 3.2: Flow chart of the steps to solve $\mathbf{A}$ in OpenFOAM®.

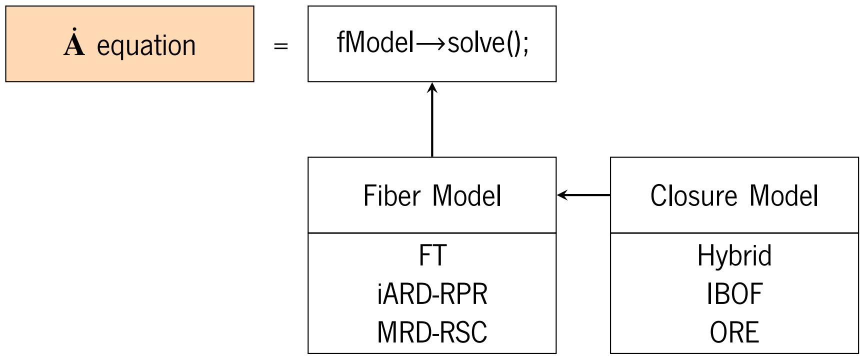

Figure 3.3: Structure of the factory method.

created without instantiating a class, making this the task of the subclasses, which grants it the ability to modify the object's instance [68]. The class diagram is shown in Figure 3.3. There is two main classes, one containing the implementation of the fiber orientation models and the other, the closure approximations. The fiber orientation models classes is capable of using the closure approximation class to create an object containing the selected closure, and implement it on the desired fiber orientation model. The selected model is then solved for the desired time inside the time loop in the selected solver.

Listing 3.10 shows the header file for the base class of fiber orientation models. In Lines 21-31 the variables and parameters that are common to all models are declared. The velocity field (`U_`) and the face flux field(`phi_`) are passed by reference since these variables are calculated in the associated flow solver. In Line 31 the class contained the closure approximations is pointed to.

In Lines 67 and 69, two purely virtual functions are declared. These functions cannot create objects, therefore they do not need to be defined. In derived classes, that can create objects, all purely virtual functions must be defined. The `solve` function will contain the implementation of the fiber orientation models in the derived classes and the `read` function will be used to get data from a dictionary file with the information needed by a specific model.

```
14 namespace Foam
15 {
16     class fiberModel
17     :public IOdictionary
18     {
19 
20     protected:
21         const volVectorField& U_;
22         const surfaceScalarField& phi_;
23         volTensorField gradU_;
```

```
        volSymmTensorField D_;
        volTensorField W_;
        volScalarField shrRate_;
        volSymmTensorField A2_;
        volSymmTensorField D_doubleDot_A4_;
        scalar CI_;
        scalar xi_;
        autoPtr<closureModel> closureModel_;
```

```
        virtual ~fiberModel() = default;

        virtual void solve() = 0;

        virtual bool read() = 0;
    };
}
```

Listing 3.10: Header file for base class of fiber models.

The library created for the closure approximations follows a similar approach. As shown in Listing 3.11, the variables common to all cases are declared in the base class, which are a reference to the dictionary that was declared in the fiber models class and the second-order tensor field `A2_`.

The functions to be implemented in classes derived from the `closureModel` classe are the following: (i) the pure virtual function `compute_D_doubleDot_A4`, where the computation, trough Symbolic code, of the operation $\mathbb{A}:\mathbf{D}$ is implemented (Lines 48-49); (ii) the pure virtual function `compute_D_doubleDot_A4_RSC`, which will also contain the implementation of the double contraction between $\mathbb{A}$ and $\mathbf{D}$, in which case the operation to be computed is: $[\mathbb{A}+(1-\kappa)(\mathbb{L}-\mathbb{M}:\mathbb{A})]:\mathbf{D}$ (Lines 51-53), as shown in Equation (2.11).

```
 8 namespace Foam
 9 {
10 class closureModel
11 {
12 protected:
13         const dictionary& dict_;
14         const volSymmTensorField& A2_;
```

```
48     virtual void compute_D_doubleDot_A4(volSymmTensorField& D_doubleDot_A4,
49     const volSymmTensorField& D) = 0;
50 
51     virtual void compute_D_doubleDot_A4_RSC(volSymmTensorField& D_doubleDot_A4,
52     const volSymmTensorField& D, const volTensorField& eVecs, const volVectorField& eVals,
53     const scalar& k) = 0;
```

```
};
}
```

Listing 3.11: Header file for base class of closure approximations.

Listing 3.12 illustrates an example of a dictionary that must exist in the case folders to use this library.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  3.0.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      fiberProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
CI 0.01;
xi 1;

closureModel IBOF;
fiberModel iARD_RSC;

alpha 0.1;
beta 0;
CM 0.92;
// ************************************************************************* //
```

Listing 3.12: Illustrative Dictionary file of the developed utility (using iARD-RSC as fiber model and IBOF for closure).

Additionally, a modified solver, based on the pimpleFoam solver, that allows using the developed utility is shown in Listings 3.13 and 3.14.

First, a smart pointer to the fiber model library is needed. This is set up in the standard `createFields.H` file of OpenFOAM®, since here are declared all the fields used by the solver, including the ones that this library needs, velocity and face flux. The arrow operator(->), which allows the access to member classes trough a pointer, can then be used to call this utility during the computing step that the user finds more appropriate, per example, after the PIMPLE loop finishes and the velocity field for a given time step is solved. In the developed solver it's simply called after the time is increased by a time step, in line 54.

```
#include "createRDeltaT.H"

Info<< "Reading field U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


Info << "Reading/calculating face flux field phi" << nl << endl;
surfaceScalarField phi
(
    IOobject
    (
        "phi",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    linearInterpolate(U) & mesh.Sf()
);

autoPtr<fiberModel> fModel (fiberModel::New(U, phi));
```

Listing 3.13: File "createFields.H" containing the definition of fields in OpenFOAM.

```
34      Info<< "\nStarting time loop\n" << endl;
35
36      while (runTime.run())
37      {
38          #include "readDyMControls.H"
39
40          if (LTS)
41          {
```

```
            #include "setRDeltaT.H"
        }
        else
        {
            #include "CourantNo.H"
            #include "setDeltaT.H"
        }

        ++runTime;

        Info<< "Time = " << runTime.timeName() << nl << endl;

        fModel->solve();

        runTime.write();

        runTime.printExecutionTime(Info);
    }
```

Listing 3.14: Time loop of the created solver.

CHAPTER 4 Verification of the Developed Computational Tools

4.1 `Python` benchmark tool

Data obtained from the literature [18, 28, 48, 50, 69] was used to validate the developed `python` benchmark tool. Most of the data from the literature is available in graphical format. In these cases, the utility `WebPlotDigitizer` [70] was used to obtain the most possible accurate values. Around 20-30 points were collected from selected images depicting the evolution of fiber orientation under simple homogeneous flows. Due to the uncertainty associated with the literature data collection procedure, including image quality and resolution, the values cannot be treated as exact. However, the obtained points should provide sufficient insight for comparison with the developed `python` benchmark tools.

Throughout this section of the dissertation, figures with the points represent the data collected from literature, while continuous lines represent the data resulting from the developed `python` modules.

4.1.1 Simple Shear flow

When a FRTM is forced to flow under pure shear, the flow field exerts forces and stresses on the fibers, which force their rotation. To minimize the energy associated with the fiber-matrix interaction, the fibers undergo reorientation and align themselves parallel to the streamlines of the flow [7]. The velocity field of simple shear flow is defined as:

$$\mathbf{u} = \begin{bmatrix} \dot{\gamma} y \\ 0 \\ 0 \end{bmatrix}, \tag{4.1}$$

where $\dot{\gamma}$ is a scalar defining the rate of shear forces.

Figure 4.1 illustrates the evolution of fiber orientation resulting from the application of a uniform and constant shear rate, with a velocity field given by $\mathbf{u} = \begin{bmatrix} y & 0 & 0 \end{bmatrix}^{\mathrm{T}}$, calculated with the FT framework with different closure models. In this case, the FT model parameters were set as: $\dot{\gamma} = 1s^{-1}$, $C_I = 0.01$ and $\xi = 1$. The probability of the fibers aligning along the direction of the tensor component A_{11}, which corresponds to the flow direction, increases with time and reaches a steady-state at $\dot{\gamma}t \approx 14$. The orientation ceases to evolve due to the strength of the interaction coefficient. The $\mathbf{A}_{12}$ component starts from zero and initially rises up to $\dot{\gamma}t \approx 2$. Afterwards, the value decreases and reaches a finite steady-state value. This indicates that one of the principal axes of orientation lies between the 1 and 2 directions.

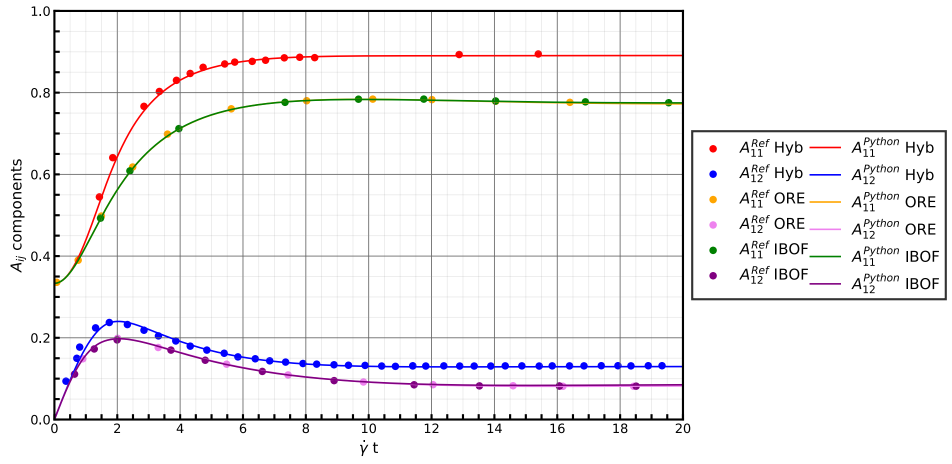

Figure 4.1: Prediction of Folgar-Tucker model under simple shear flow. Data obtained from Hybrid[18], IBOF[50] and ORE[48].

The results obtained with the IBOF and ORE closures are similar, since they are based on similar approaches (both are fitted using data from calculations of the PDF). The most relevant difference between them is their computation time, because the use of invariants of a tensor results in lesser calculations needed to be done, comparatively to the use of eigenvalues [50]. Conversely, the data obtained from the Hybrid closure approximation exhibits a different steady-state value. This discrepancy is a known issue in the literature, where the Hybrid closure tends to overshoot the values of $\mathbf{A}$ [39].

Figure 4.2 displays the evolution of $\mathbf{A}$ for a velocity field defined as $\mathbf{u} = \begin{bmatrix} \dot{\gamma}z & 0 & 0 \end{bmatrix}^{\mathrm{T}}$, using the iARD-RPR model and IBOF closure, with the following parameters: $C_I = 0.025$, $\xi = 1$, $C_M = 1$, $\alpha = 0.967$ and $\beta = 0$.

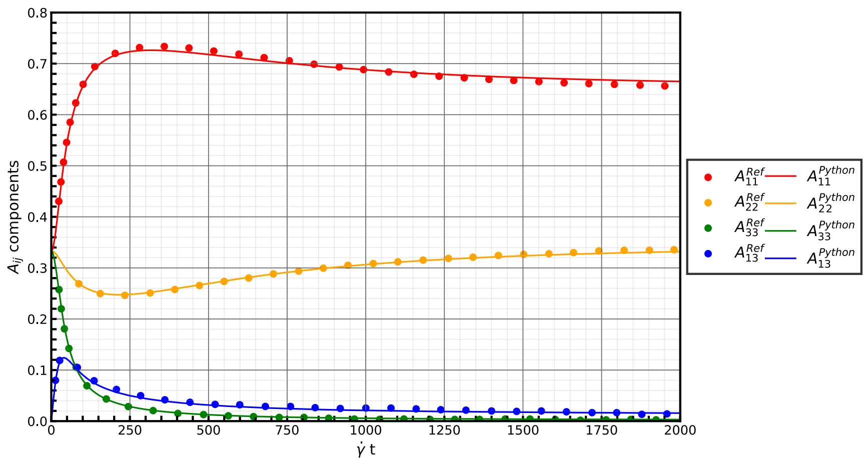

Figure 4.2: iARD-RPR model using the IBOF for simple shear flow. Data obtained from [28].

The fibers exhibit the same trend for A_{11} and A_{22}, as for the iARD-RPR model. Additionally, in this case, the evolution of A_{33} and A_{13} is also represented. A_{33} corresponds to the direction transverse to the flow direction, and therefore, the probability of the fibers being oriented in this directions tends to decrease. The initial rise and subsequent decrease in the evolution of A_{13} can be explained by the principal axis of orientation being A_{11}.

Figure 4.3 displays the results of fiber orientation for a simple-shear flow, as reported for the results in 4.1, and solved with the MRD-RPR model using the IBOF closure. The following parameters were used: $\dot{\gamma} = 1s^{-1}$, $C_I = 0.017$, $\xi = 1$, $C_1 = 1$, $C_2 = 0.66$, $C_3 = 0.34$, and $\kappa = 0.275$.

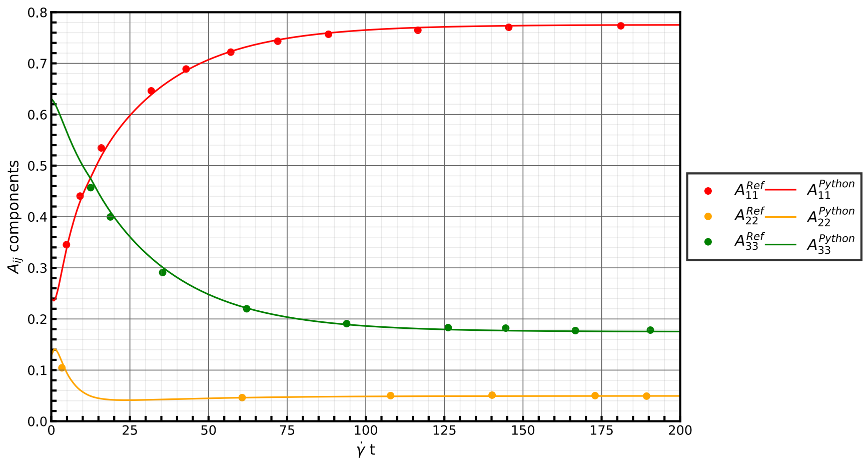

Figure 4.3: MRD-RSC model allied with the IBOF closure approximation under simple shear flow. Data obtained from [69].

For the case in which the MRD-RSC model was tested, the authors made the decision not to start from an isotropic state. Despite the different initial values, the fibers still tend to align along the flow direction and exhibit the same trend as for previous models for all the represented tensor components, therefore, the final state of orientation depends on its initial values. However, the final equilibrium state is affected. The probability of $\mathbf{A}_{11}$ and $\mathbf{A}_{33}$ shows a significant difference between the initial and final states, while, comparatively, $\mathbf{A}_{22}$ undergoes only a slight change.

4.1.2 Shear/Elongational flow

A shearing/elongational flow combines simple shear with elongational flow. This pattern is comparable to the one observed in a center-gated disk, where shearing forces compete with the elongational forces along the thickness of the part. The velocity field of uniaxial elongation flow can be defined as:

$$\mathbf{u} = \begin{bmatrix} -\dot{\epsilon}x \\ -\dot{\epsilon}y \\ 2\dot{\epsilon}y \end{bmatrix}, \tag{4.2}$$

and planar elongational flow can be defined as:

$$\mathbf{u} = \begin{bmatrix} \dot{\epsilon}x \\ -\dot{\epsilon}y \\ 0 \end{bmatrix}. \tag{4.3}$$

The velocity equation corresponding to shear/elongational flow can be obtained by the summation of Equation (4.2) or (4.3) and (4.1).

The shear/elongational ratio $(\dot{\gamma}/\dot{\epsilon})$ determines the steady-state orientation. When the ratio is less than 10, the elongational flow dominates the orientation, resulting in a orientation distribution similar to that one of an elongational flow. In an elongational flow the fibers tend to align themselves along the flow direction. On the other hand, for a ratio greater than 50, the shearing flow dominates. However, when the ratio is equal to 20, the effects of the two flows nearly balance each other, resulting in a steady orientation that is close to random in terms of the orientation direction imposed by the shearing flow and the direction imposed by the elongational flow [18, 45].

For the studied test cases corresponding to Hybrid and IBOF closure approximations, the fibers were subjected to a shear/elongation ratio of $(\dot{\gamma}/\dot{\epsilon}) = 20$, with elongation along axis 3 and shearing in the plane 1-3. This corresponds to the following velocity field $\mathbf{u} = \begin{bmatrix} -\frac{1}{20}x+z & -\frac{1}{20}y & \frac{1}{10}z \end{bmatrix}^{\mathrm{T}}$, with $C_I = 0.01$ and $\xi = 1$.

As observed in Figure 4.4, the fibers are primarily aligned along the direction A_{11}, which is imposed by the shearing flow, as well as the direction A_{33}, which is imposed by the elongational flow.

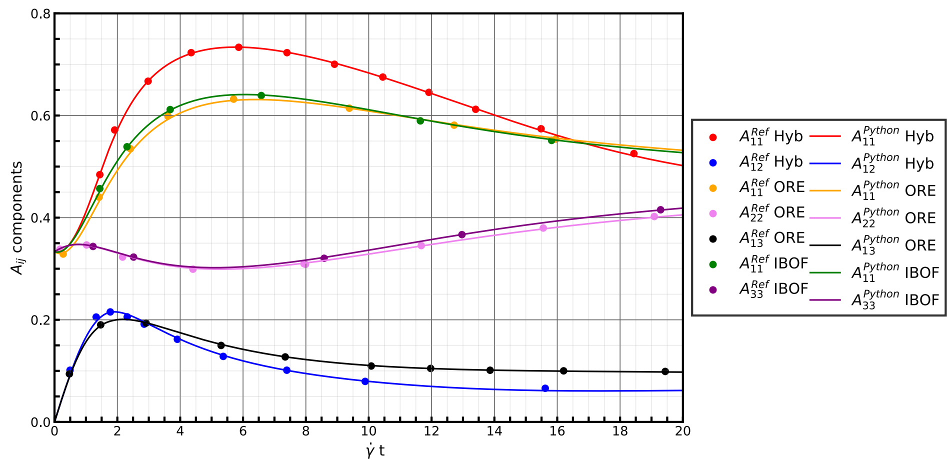

Figure 4.4: Orientation evolution predicted by the FT model using different closure approximation under simple shear/elongational flow. Hybrid and IBOF closures tested under $\dot{\gamma}/\dot{\epsilon} = 20$ and ORE, $\dot{\gamma}/\dot{\epsilon} = 10$. Data obtained from Hybrid[18], IBOF[50] and ORE[48].

For the ORE closure approximation, the orientation distribution was tested under a different flow condition, with a shear/elongation ration $(\dot{\gamma}/\dot{\epsilon}) = 10$. In this case, elongation occurred in the axis plane 2-2, while shearing took place in the plane 1-2. This corresponds to a velocity field of $\mathbf{u} = \begin{bmatrix} -\frac{1}{10}x+z & \frac{1}{10}y & 0 \end{bmatrix}$ with $C_I = 0.01$ and $\xi = 1$. As a result, the fibers preferred aligning along the directions A_{11} and A_{22}.

4.1.3 Center-gated disk

For a center-gated disk, the velocity and velocity gradient can be defined by the Equations (3.1) and (3.2), respectively.

In injection moulded parts, it is common to observe the formation of a core-shell structure [7, 71]. In the shell region of a disk, the fibers are highly orientated in the flow direction. This is attributed to the parabolic velocity profile that forms along the thickness of the disk. The maximum velocity occurs in the middle of the thickness ($\tilde{z} = 0$), and it becomes zero at the wall. This velocity profile generates high shearing forces near the walls, resulting in significant orientation along the flow direction of the disk. Additionally, due to the small injection gate in comparison with the size of the cavity, a diverging flow develops, leading to in in-plane elongation deformation across the flow direction. Consequently, in the core of the disk, the fiber orientation will be perpendicular to the flow direction [7, 71].

In Figure 4.5, the evolution of fiber orientation as a function of the normalized radius, $\tilde{r}$, is presented. At the normalized thickness $\tilde{z} = 0.9$, representing the near-wall regions of the disk, the fibers exhibit a high orientation along the flow direction ($\mathbf{A}_{11}$), at the steady-state. On the other hand, at the middle of the thickness ($\tilde{z} = 0$), the orientation along the flow direction is absent, being, thus, consistent with the previous description of the core-shell morphology.

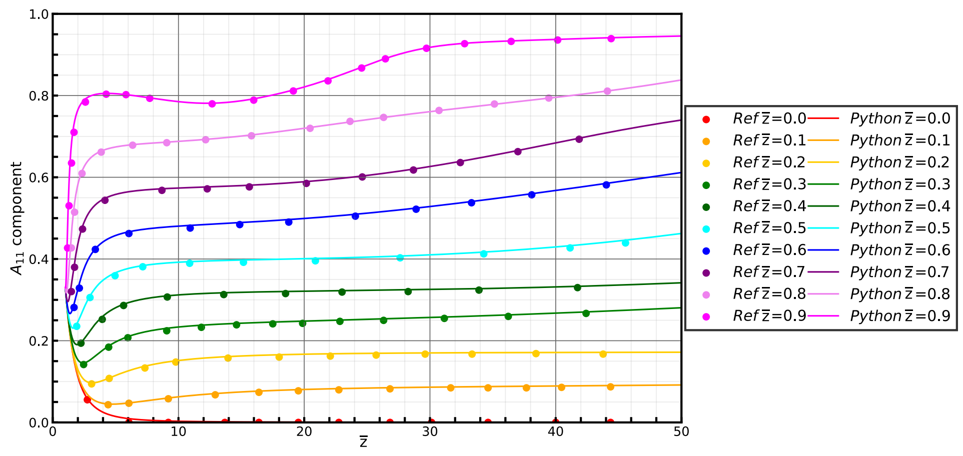

Figure 4.5: FT model associated with IBOF closure approximation, applied to a center-gated disk. Data obtained from [50].

Figure 4.6 depicts the evolution of fiber orientation as a function of the normalized thickness, $\tilde{z}$, for a normalized radius of $\tilde{r} = 20$. This plot clearly demonstrates the formation of a shell-core morphology. Near the walls, at $\tilde{z} = -0.9$ and 0.9, the orientation component A_{11} exhibits the maximum value of orientation. Conversely, at $\tilde{z} = 0$, the fibers show a preferred cross-flow orientation, corresponding to the component A_{22}. The component A_{13} has the same magnitude, but different direction at $\tilde{z} = -0.9$ and 0.9, resulting from the symmetry of the diverging flow.

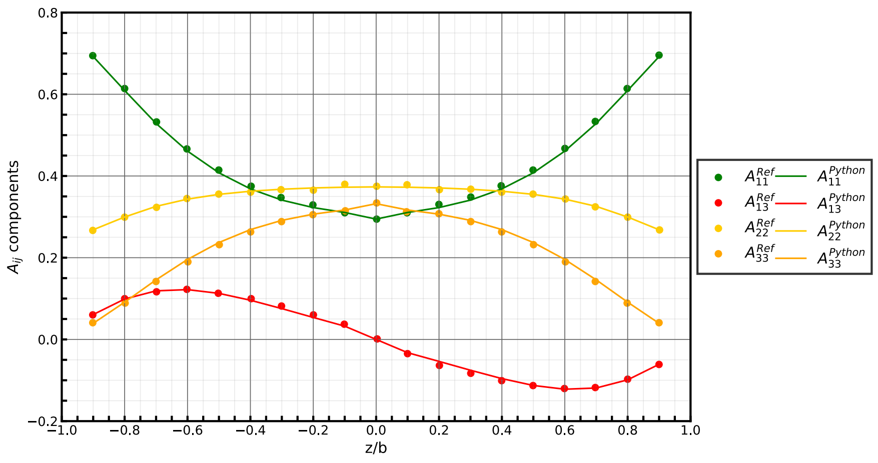

Figure 4.6: iARD-RPR model allied with the IBOF closure approximation applied to a center-gated disk. Data obtained from [28].

4.2 OpenFOAM® utility

For the verification of the OpenFOAM® utility, three flow scenarios were considered, two simple flows (simple-shear and planar elongation) and one more representative of industrial processes (flow in a center-gated disk). For ease of verification, the utility `setExprFields` [72] was used to impose the velocity and velocity gradient fields for all the test cases, and the mesh was created with the `blockMesh` [72] utility.

The codes verification were undertaken by comparing the results predicted by the developed code with data obtained from the `python` code presented in Section ??. Since the results obtained with the `python` script, correspond just to the time evolution in homogeneous flows, and the OpenFOAM® counterpart refer to flows that integrate space and time distributions, a relation between position and time has to be established.

4.2.1 Test cases and Time-Position Relations

4.2.1.1 Simple-shear Flow

Simple-shear flow can be described by only having one non-null velocity component with a constant velocity gradient, as shown in Equation (4.1). For this test case $\dot{\gamma}$ was defined as unitary and it was assumed to take place in an unitary square, which results in the velocity field depicted Figure 4.7, being the employed boundary conditions presented in Table 4.1.

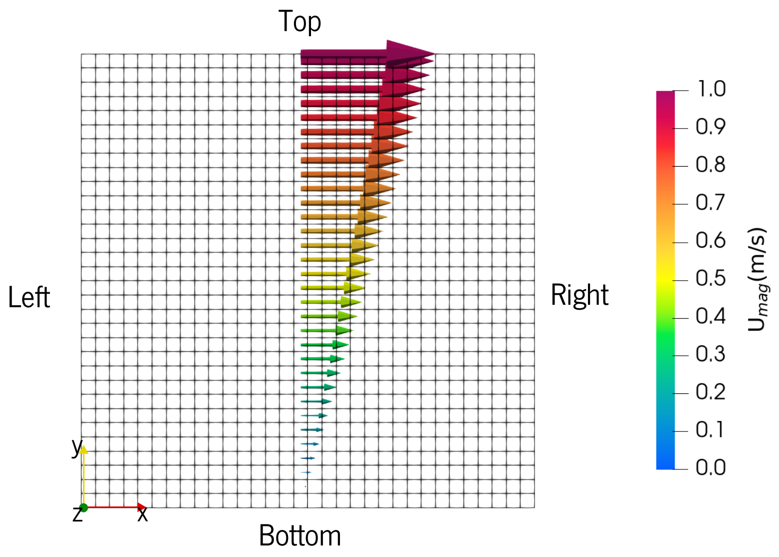

Figure 4.7: Velocity field of the simple shear flow test case.

Boundary	$\mathbf{A}$	$\mathbf{u}$ (m/s)
Left	fixedValue uniform (0.(3) 0 0 0.(3) 0 0.(3))	zeroGradient
Right	zeroGradient	zeroGradient
Top	zeroGradient	fixedValue uniform (1 0 0)
Bottom	zeroGradient	fixedValue uniform (0 0 0)
Front & Back	symmetry	symmetry

Table 4.1: Boundary conditions used for the Simple Shear test case.

Having in mind this flow, the residence time of a virtual particle that enters in the `Left` patch is obtained by dividing its x coordinate by its velocity. Accordingly, for this test case, the relationship between position and time can be obtained by dividing the position of the cell on the x coordinate by its velocity, i.e., $t = x/u_x$.

4.2.1.2 Planar elongational flow

In a planar elongational flow, the material is stretched in one direction (the flow direction), and, due to the incompressibility restriction, and contracts equally in the other direction, as shown in Equation (4.3). For this test study, the velocity profile was defined by setting $\dot{\epsilon} = 2s^{-1}$. The resulting velocity field is shown in Figure 4.8, and the boundary conditions were defined as shown in Table 4.1.

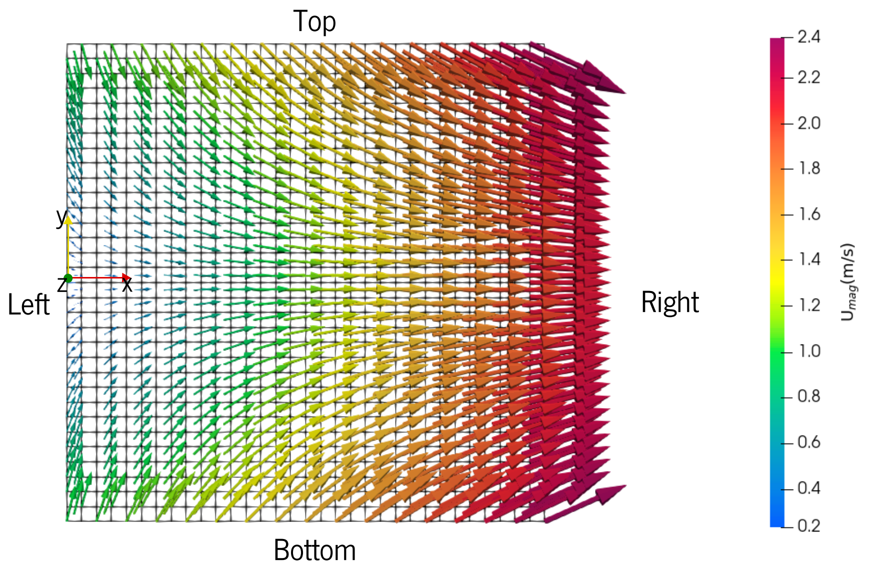

Figure 4.8: Velocity field of the planar elongational flow test case.

Boundary	$\mathbf{A}$	$\mathbf{u}$ (m/s)
Left	fixedValue uniform (0.(3) 0 0 0.(3) 0 0.(3))	fixedGradient uniform (-2 0 0)
Right	zeroGradient	fixedGradient uniform (2 0 0)
Top	zeroGradient	fixedGradient uniform (0 -2 0)
Bottom	zeroGradient	fixedGradient uniform (0 2 0)
Front & Back	symmetry	symmetry

Table 4.2: Boundary conditions used for the planar elongation case.

The relation between position and time can be obtained from the relation between velocity, position and time. The relationship that relates velocity and position is:

$$u = u_0 + \int_x du = u_0 + \int_x \frac{du}{dx} dx = u_0 + \int_x a dx = u_0 + a \int_x dx = u_0 + ax \tag{4.4}$$

and between velocity and time is:

$$\begin{aligned} &\frac{du}{dt} = \frac{du}{dx}\frac{dx}{dt} = au \Leftrightarrow \frac{du}{u} = adt \Leftrightarrow \ln u = at + c \Leftrightarrow u = ce^{at} \Leftrightarrow u = u_0 e^{at} \\ &t = 0 \longrightarrow u = u_0 \Leftrightarrow c = u_0 \end{aligned} \tag{4.5}$$

with u_0 being the initial velocity, and $a = \frac{du}{dx}$, the acceleration.

From these two relationships, we can obtain the relation between the residence time and the position given by the cell x coordinate.

$$u = u_0 + ax = u_0 e^{at} \Leftrightarrow x = \frac{u_0}{a}\left(e^{at} - 1\right) \Leftrightarrow t = \frac{\ln\left(\frac{ax}{u_0} + 1\right)}{a} \tag{4.6}$$

Equation (4.6) only holds when the velocity in the y-axis is null($v_y = 0$), which happens at the middle of the mesh, when $y = 0$.

4.2.1.3 Center-gated disk

The velocity field for the center-gated disk, in local Cartesian coordinates coordinates, is given by Equation (3.1), in which the radial velocity just depends on the radius, which corresponds to the position in the x-axis, and thickness, which corresponds to the position in the z-axis.

The disk geometry was created as a $5°$ slice as depicted in Figure 4.9, and the boundary conditions were defined as shown in Table 4.3.

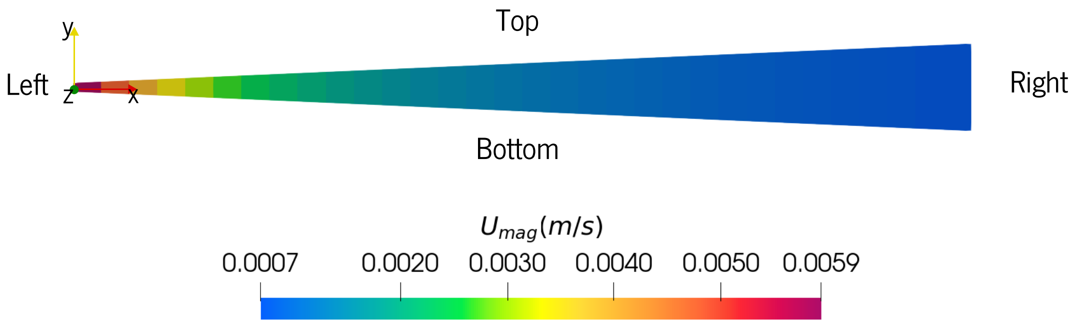

Figure 4.9: Velocity field on the disk segment in which the flow was modelled for the center-gated disk test case.

Boundary	$\mathbf{A}$	$\mathbf{u}$ (m/s)/ grad($\mathbf{u}$)(m/s)
Left	fixedValue uniform (0.(3) 0 0 0.(3) 0 0.(3))	Calculated by setExprBoundaryFields utility
Right	zeroGradient	
Front & Back	zeroGradient	
Top & Bottom	symmetry	

Table 4.3: Boundary conditions used for the center-gated disk test case.

For this case, the boundary conditions of the velocity and gradient velocity were imposed with the setExprBoundaryFields utility, according the Equations 3.1 and 3.2.

For the disk, the data is analyzed in function of the space, so there was no need for a time-position conversion.

4.2.2 Computational framework setup

Table 4.4 shows the discretization schemes used for the test cases, and Table 4.5 presents the linear solver setup and respective tolerances.

Time scheme	Euler
Gradient Scheme	Least Squares
Divergent Scheme	Gauss Upwind
Interpolation Scheme	Linear

Table 4.4: Temporal and spacial discretization schemes for the simple flow cases.

For the center-gated disk case, the method of Crank–Nicolson was used for the time discretization scheme, and for the remaining schemes, the same as for the simple flow cases was employed.

Since, as mentioned before, the velocity field was defined by an mathematical expression, and, therefore, it was not calculated, a linear solver for this parameter was not needed. The

Parameter	$\mathbf{A}$
Linear solver	Stabilized Preconditioned (bi-)conjugate gradient
Smooth solvers	Diagonal incomplete-Cholesky with Gauss-Seidel
Absolute residual tolerance	10^{-11}
Relative tolerance	0

Table 4.5: Solution methods for the test cases.

4.2.3 Error Measure

To evaluate the error between the results from the `python` script and the OpenFOAM® code predictions, a similar approach to the one employed by Cintra and Tucker[45] was used. Firstly, an second order error tensor $(\boldsymbol{\varepsilon}_{ij})$ was calculated as the difference between the `python` results, which is assumed as the most exact result, and the one provided by OpenFOAM®, as follows:

$$\boldsymbol{\varepsilon}_{ij} = \mathbf{A}_{ij}^{\text{python}} - \mathbf{A}_{ij}^{\text{OpenFOAM}}. \tag{4.7}$$

Then the scalar magnitude of the error tensor is used to estimate the overall error, with the following expression:

$$||\boldsymbol{\varepsilon}|| = \sqrt{\frac{1}{2}\boldsymbol{\varepsilon}_{ij}\boldsymbol{\varepsilon}_{ji}}. \tag{4.8}$$

Following this procedure, two different error parameters were calculated. The steady state error that corresponds to the error magnitude obtained for the last calculated time, $||\boldsymbol{\varepsilon}||(t_\mathrm{f})$, and the time average error, given by the integration of $||\boldsymbol{\varepsilon}||$ over time:

$$||\bar{\boldsymbol{\varepsilon}}|| = \frac{1}{t_\mathrm{f}}\int_0^{t_\mathrm{f}} ||\boldsymbol{\varepsilon}||(t)\mathrm{d}t. \tag{4.9}$$

4.2.4 Mesh Sensitivity Studies

Aiming at quantifying the error associated with the refinement of the mesh, calculations were performed with different mesh refinement levels. Since the flow takes place along in the $x - axis$ direction for all test cases, the mesh was just refined along x. In the case of the planar elongational flow, despite existing flow in the y-axis direction, both the velocity and velocity gradient were imposed by exact analytic expressions, and, therefore, mesh refinement along the y-axis direction does not have impact in the results obtained.

For all test cases, the number of cells imposed along the x-axis was increased according the following equation:

$$\text{No.Cells}_\text{x} = 2^{i+4}, i = 1,2,3,4,5,6,7. \tag{4.10}$$

4.2.5 Results and Discussion

Figures 4.10 - 4.14 were obtained with a C_I of 0.01 and ξ equal to 1, for a simple shear flow. The parameters for the iARD-RPR model were set as $C_M = 0.9$ and $\alpha = 0.1$, and for the MRD-RSC model were set as $\kappa = 0.9$ and $C_1 = 1$, $C_2 = 0.5$, $C_3 = 0.3$. The reduction of the error with mesh refinement can be both visually examined and quantified.

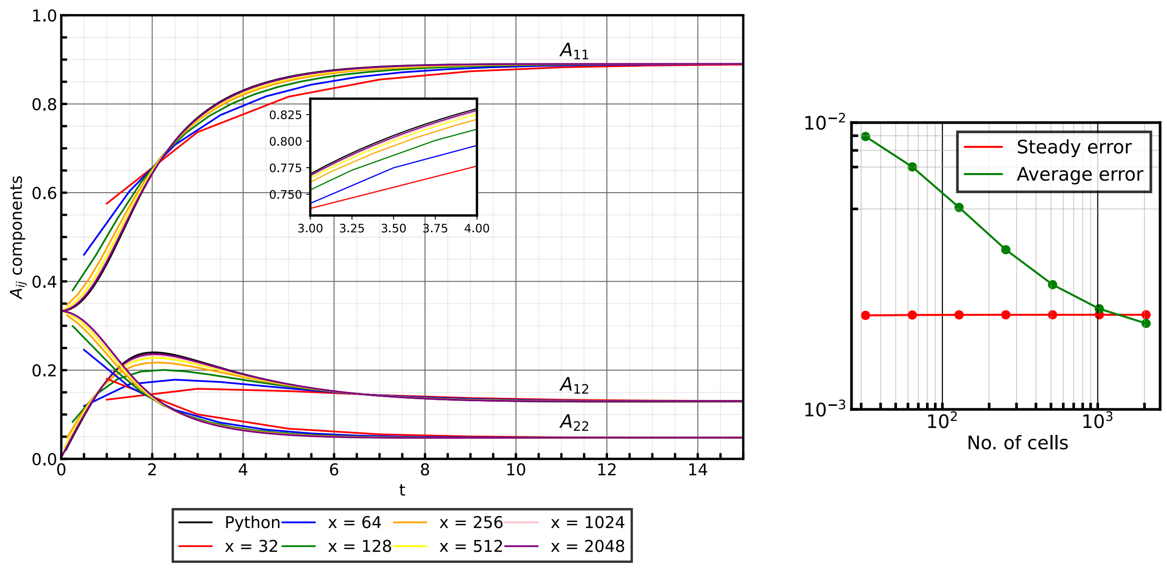

Figure 4.10: Prediction of Folgar-Tucker model with the Hybrid closure for the simple shear flow test case (left) and calculated errors (right), for different mesh refinement levels.

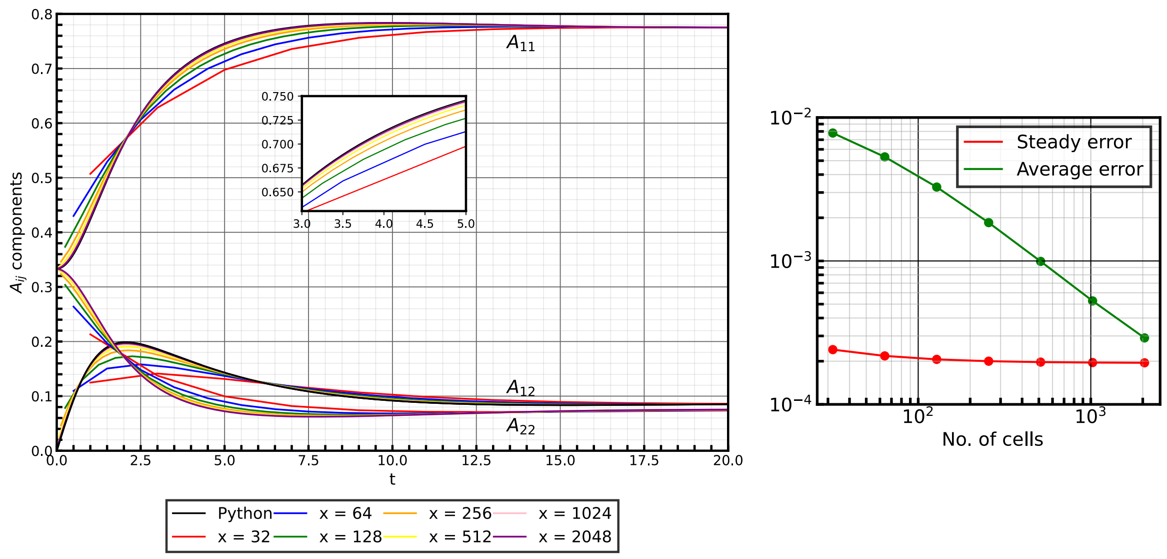

Figure 4.11: Prediction of Folgar-Tucker model with the IBOF closure for the simple shear flow test case (left) and calculated errors (right), for different mesh refinement levels.

By comparing of the Figure 4.10 and 4.11, it is clear the overprediction of the fiber orientation, that results from the Hybrid closure, that was referred to in Section 2.3. For the Hybrid closure, around $t \approx 10s$, the orientation has already reached its steady state. Meanwhile, for the IBOF closure, the steady state is only reached around $t \approx 15s$.

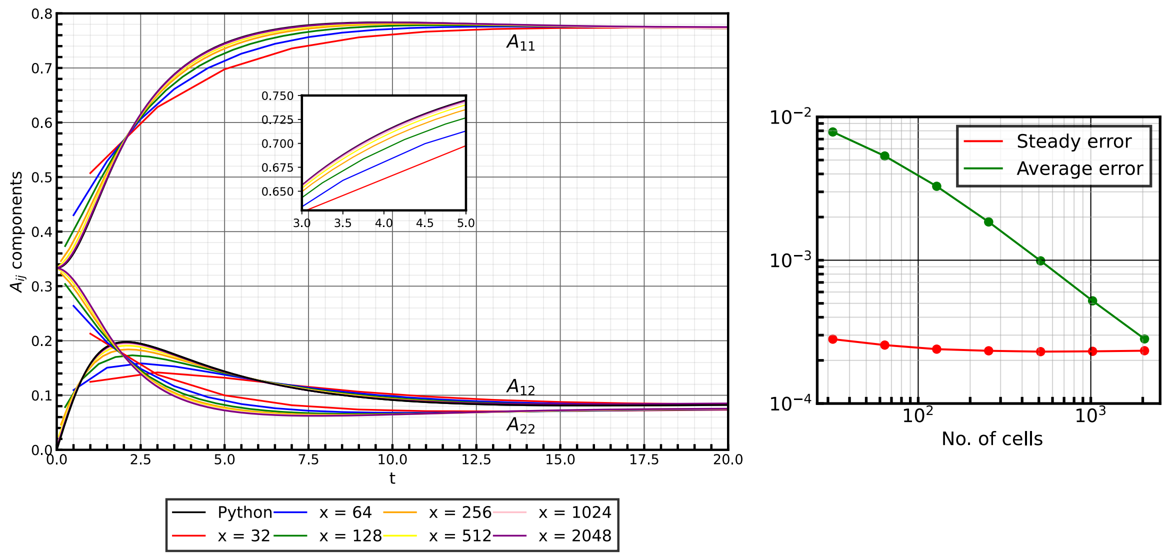

Figure 4.12: Prediction of Folgar-Tucker model with the ORE closure for the simple shear flow test case (left) and calculated errors (right), for different mesh refinement levels.

As previously mentioned in Section 2.3, the IBOF (Figure 4.11) and the ORE (Figure 4.12) closures present similar approaches, and therefore similar results, but their computational efficiency is not equal. By analysing the error, it can be seen that both present similar accuracy for different mesh refinement levels. For instance, a mesh with 1024 cells in the x-axis direction, the test case for the IBOF closure had an execution time of 2:39:27 hours, while the case for the ORE closure, had a execution time of 3:01:18 hours. Therefore, the ORE closure needed more 22 min to solve the test case under the same conditions, then the one used for the IBOF closure. In the developed code, the ORE closure involves the calculation of the tensor's eigenvalues, but the same is not needed for the IBOF closure, so this is the cause for the different times.

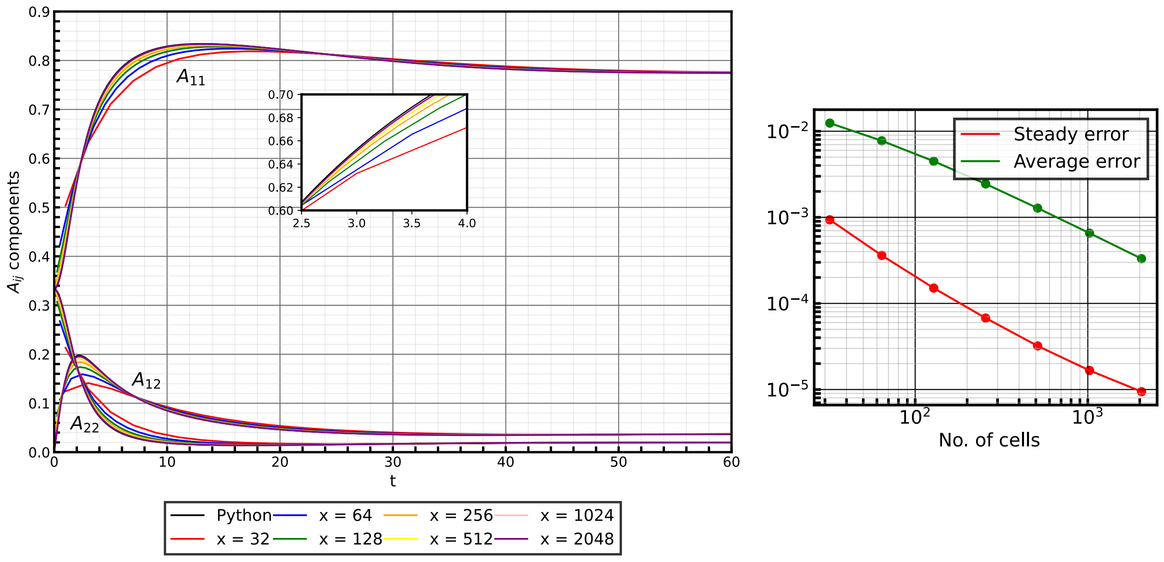

Figure 4.13: Prediction of iARD-RPR model with the IBOF closure under simple shear flow.

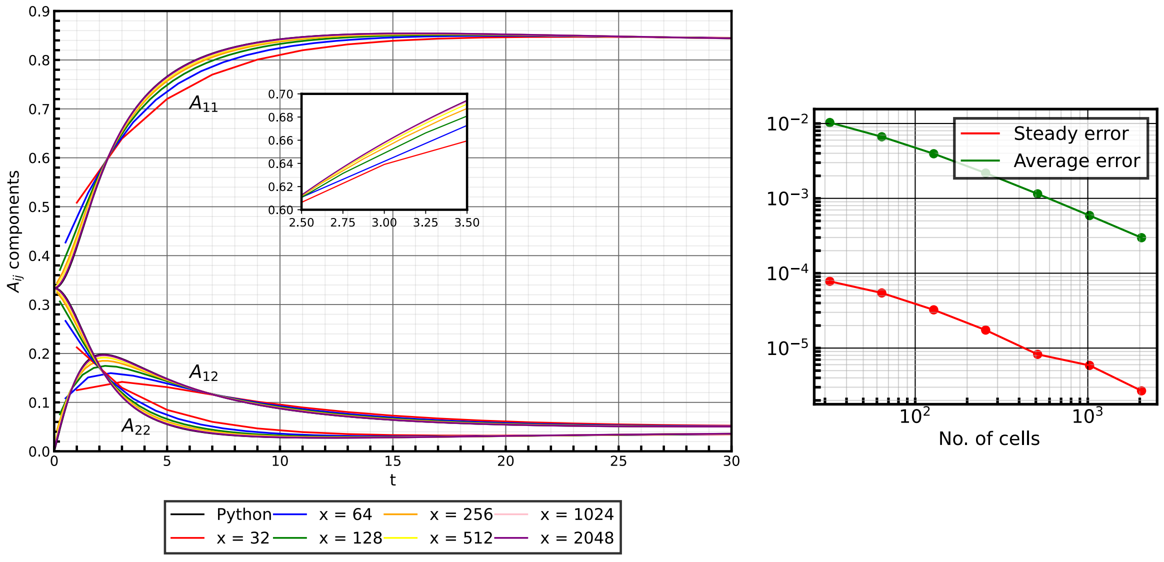

Figure 4.14: Prediction of MRD-RSC model with the IBOF closure under simple shear flow.

Since the remaining results obtained for the remaining test cases present similar trends the additional data is provided in Appendix B.

From the overall analysis of the results obtained, it can be concluded that the accuracy of the prediction of the steady state error is not highly dependent on the mesh refinement for the case of the FT model. This happens because the steady state distribution does not depend on the time step employed. For the iARD-RPR and MRD-RSC model, the steady state error reduces with the refinement because the length of the mesh is not enough for the distribution of the fiber to reach its steady state due the deceleration of the orientation kinetics that this models induce.

The average error decreases with the mesh refinement, as expected, but not with the degree that corresponds to the one of the discretization schemes employed. For first-order discretization schemes, such as Euler and Upwind, if the number of cells of a mesh are doubled, the error should reduce by half.

To identify the source of this behavior, the spacial component of material derivative of $\mathbf{A}$ was removed and the OpenFOAM® results were directly compared to the ones from `python`. To assess the correctness of the implementation, a time-refinement study was carried out and the convergence order was assessed through the following expression [73]:

$$\text{Convergence order} = \frac{\ln \frac{\mathbf{E}_{coarser}}{\mathbf{E}_{finer}}}{\ln \frac{\Delta t_{coarser}}{\Delta t_{finer}}}, \tag{4.11}$$

where $\mathbf{E}$ corresponds to the difference between the results obtained from OpenFOAM® and `python`, and Δt, the time step used. Moreover, the subscript "coarser" stands for the results obtained from the mesh with less number of cells, and finer, with more cells.

For this test case, the FT model with the Hybrid closure was used, with random initial values of the $\mathbf{A}$ and a random velocity gradient, for a final time of $8s$. The results obtained are presented in the Table 4.6.

Δt (s)	Convergence order
5x10^{-3}	1.013
1x10^{-3}	1.005
5x10^{-4}	1.001

Table 4.6: Convergence error in function of the time step.

With the reduction of the time step, the convergence error is approximately first-order, which is consistent with the time discretization scheme used in OpenFOAM®. Therefore, it can be concluded that the time evolution of the fiber orientation is being solved correctly.

The spacial dependency in the equation is given by the term $\nabla \cdot (\mathbf{uA})$ which translates into syntax as `fvm::div(phi, A2)`. Although not formally proven, but given that OpenFOAM® is a well-established software and that with the progressive mesh refinement the results approach its single material point counterpart, the results obtained provide confidence that the code is well implemented and solving the correct equations.

The unexpected behavior can be attributed to the error associated with the time refinement. With the increase of the spacial refinement of the mesh, the time refinement should accompany it. Due to computational efficiency, this was not done, and the time step was regulated according the number of Courant. Therefore, the error associated with the time refinement has impact in the results shown of the error in function of the spatial refinement.

CHAPTER 5 Conclusions and Future Work

5.1 Conclusions

Due the importance of determining the fiber orientation distribution, various proprietary software offer different fiber orientation numerical models. Despite the availability of this software, few validation works have been performed due to the inaccessible source code and costly licences. Open-source software emerges as a solution since it offers full unrestricted access to the code and its modification, as is the case of OpenFOAM®.

The main objective was to develop an utility in OpenFOAM®, to be attached to a existing solver, containing state-of-the-art fiber orientation models and closure approximations, for the prediction of the fiber orientation distribution in different types of flow.

Through a study on the state-of-art numerical models for the prediction of fiber orientation, the models to be implemented were selected. These models were firstly implemented in a `python` script, capable of solving the evolution of the second-order orientation tensor in function of time exclusively. This tool was validated against data found in literature, and it was concluded to be able to replicate the selected data accurately. Therefore, it could be used as a benchmark for the developed OpenFOAM® utility. Moreover, it was also used to create symbolic code to express the closure approximations in terms of second-order tensors exclusively, since OpenFOAM® cannot handle fourth-order tensors.

The OpenFOAM® utility was tested in simple flows, such as simple shear and planar elongational flow, chosen due to their presence in industrial process of FRTM. Additionally, the flow in a center-gated disk was also studied, due to its higher complexity and similarity to real-life manufacturing processes.

The results obtained were in agreement with literature described behaviour, and from the studies on the impact of mesh refinement onto the convergence order, it could be concluded that the developed utility can handle correctly the calculation of the desired models. The developed tool, alongside OpenFOAM®, demonstrated potential to replace proprietary software in industrial application.

5.2 Future work

Although the convergence order calculated is very close to the expected, a formal study on the convergence order in function of the spacial refinement could be performed to locate the cause of this behaviour and further confirm the tool's adequacy. Additionally, more complex flows could be studied, complementing equally complex existing solvers, such as `openInjMoldSim` [74], which would further consolidate the adequacy of the developed utility to industrial use.

Further development of the add-on is also possible and, since the factory method was employed, it should be fairly easy to implement other models, such as flow-dependent ones. Additionally, models that allow for the calculation of viscosity in function of the second-order orientation tensor could be introduced.

References

[1] P. Thori, P. Sharma, and M. Bhargava. An approach of composite materials in industrial machinery: Advantages, disadvantages and applications. International Journal of Research in Engineering Technology, 2(12):350–355, 2013.

[2] Fiber-Reinforced Composites, Available online: https://fog.ccsf.edu/ wkaufmyn/ENGN45/ Course%20Handouts/14_CompositeMaterials/03_Fiber-reinforcedComposites.html (Accessed on 2 May 2023).

[3] Fiber-Reinforced Thermoplastic Composites, Available online: www.mobilityengineeringtech.com/component/content/article/adt/pub/features/articles/33369 (Accessed on 10 April 2023).

[4] Saijod Lau, M Yuhazri, and Mohd Amirhafizan Husin. Performance comparison on using metal and kenaf frp composite hollow structure in oblique crushing. Technology Reports of Kansai University, 62(9):5581–5585, 2020.

[5] Junghoon Kim and Donghwan Cho. Effects of waste expanded polypropylene as recycled matrix on the flexural, impact, and heat deflection temperature properties of kenaf fiber/polypropylene composites. Polymers, 12(11), 2020. doi: 10.3390/polym12112578.

[6] Special Issue "Fiber-Reinforced Thermoplastics", Available online: www.mdpi.com/journal/polymers/special_issues/fiber-reinforced_thermoplastics (Accessed on 10 April 2023).

[7] Charles L. Tucker. Fundamentals of Fiber Orientation: Description, Measurement and Prediction. Hanser, 2022. ISBN 978-1-56990-875-4. doi: 10.3139/9781569908761.fm.

[8] J. H. Phelps and C. L. Tucker. An anisotropic rotary diffusion model for fiber orientation in short- and long-fiber thermoplastics. Journal of Non-Newtonian Fluid Mechanics, 156(3):165–176, 2009. ISSN 0377-0257. doi: 10.1016/j.jnnfm.2008.08.002.

[9] H.N. Gupta, R.C. Gupta, and Arun Mittal. Manufacturing Processes. New Age International, New Delhi, 2 edition, 2009. ISBN 978-0471185758.

[10] Cathelin, Julien. Through-process modelling for accurate prediction of long term anisotropic mechanics in fibre reinforced thermoplastics. MATEC Web Conf., 165:17002, 2018. doi: 10.1051/matecconf/201816517002.

[11] Jonathan Köbler, Matti Schneider, Felix Ospald, Heiko Andrä, and Ralf Müller. Fiber orientation interpolation for the multiscale analysis of short fiber reinforced composite parts. Computational Mechanics, 61(6):729–750, 2018. ISSN 01787675. doi: 10.1007/s00466-017-1478-0.

[12] Moldex3D | Plastic Injection Molding Simulation Software, Available on: https://www.moldex3d.com/products/software/moldex3d/solution-add-on/fiber/ (Accessed on 11 February 2022). .

[13] Moldflow's fiber orientation models (Theory) | Search | Autodesk Knowledge Network, Available on: https://knowledge.autodesk.com/support/moldflow-insight/troubleshooting/caas /sfdcarticles/sfdcarticles/What-is-the-best-fiber-orientation-model-to-use-in-Moldflow-2018.html (Accessed on 11 February 2022). .

[14] George Barker Jeffery and Louis Napoleon George Filon. The motion of ellipsoidal particles immersed in a viscous fluid. Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character, 102(715):161–179, 1922. doi: 10.1098/rspa.1922.0078.

[15] Moldflow | Plastic Injection Compression Molding Software | Autodesk, Available on: https://www.autodesk.com/products/moldflow/overview (Accessed on 11 February 2022). .

[16] Moldex3D | Plastic Injection Molding Simulation Software, Available on: https://www.moldex3d.com/ (Accessed on 11 February 2022). .

[17] OpenFOAM®, Available on: https://www.openfoam.com/ (Accessed on 11 February 2022). .

[18] Suresh G. Advani and Charles L. Tucker. The use of tensors to describe and predict fiber orientation in short fiber composites. Journal of Rheology, 31(8):751–784, 1987. doi: 10.1122/1.549945.

[19] Susanne Katrin Kugler, Armin Kech, Camilo Cruz, and Tim Osswald. Fiber orientation predictions—a review of existing models. Journal of Composites Science, 4(2), 2020. ISSN 2504-477X. doi: 10.3390/jcs4020069.

[20] Fransisco Folgar and Charles L. Tucker. Orientation behavior of fibers in concentrated suspensions. Journal of Reinforced Plastics and Composites, 3(2):98–119, 1984. doi: 10.1177/073168448400300201.

[21] H. M. Huynh. Improved fiber orientation prediction for injection-molded composites, 2001. Master's Thesis, University of Illinois Urbana-Champaign, Champaign County, IL, USA.

[22] M. Sepehr, G. Ausias, and P.J. Carreau. Rheological properties of short fiber filled polypropylene in transient shear flow. Journal of Non-Newtonian Fluid Mechanics, 123(1):19–32, 2004. doi: 10.1016/j.jnnfm.2004.06.005.

[23] H. M. Huynh. Improved fiber orientation prediction for injection-molded composites, 2001. Master's Thesis, University of Illinois Urbana-Champaign, Champaign County, IL, USA.

[24] Jin Wang, John F. O'Gara, and Charles L. Tucker. An objective model for slow orientation kinetics in concentrated fiber suspensions: Theory and rheological evidence. Journal of Rheology, 52(5): 1179–1200, 2008. doi: 10.1122/1.2946437.

[25] N. Phan-Thien, X.-J. Fan, R.I. Tanner, and R. Zheng. Folgar–tucker constant for a fibre suspension in a newtonian fluid. Journal of Non-Newtonian Fluid Mechanics, 103(2):251–260, 2002. doi: https://doi.org/10.1016/S0377-0257(02)00006-X.

[26] S. Kleindel, D. Salaberger, R. Eder, H. Schretter, and C. Hochenauer. Prediction and validation of short fiber orientation in a complex injection molded part with chunky geometry. International Polymer Processing, 30(3):366–380, 2015. doi: 10.3139/217.3047.

[27] H.-C. Tseng, Rong-Yeu Chang, and Chia-Hsiang Hsu. Phenomenological improvements to predictive models of fiber orientation in concentrated suspensions. Journal of Rheology, 57(6): 1597–1631, 2013. doi: 10.1122/1.4821038.

[28] Huan-Chang Tseng, R.-Y. Chang, and Chia-Hsiang Hsu. An objective tensor to predict anisotropic fiber orientation in concentrated suspensions. Journal of Rheology, 60(2):215–224, 2016. doi: 10.1122/1.4939098.

[29] Huan-Chang Tseng, Rong-Yeu Chang, and Chia-Hsiang Hsu. The use of principal spatial tensor to predict anisotropic fiber orientation in concentrated fiber suspensions. Journal of Rheology, 62 (1):313–320, 2018. doi: 10.1122/1.4998520.

[30] Bakharev Alexander, Yu Huagang, Ray Shishir, Speight Russell, and Jin Wang. Using new anisotropic rotational diffusion model to improve prediction of short fibers in thermoplastic injection molding. In SPE ANTEC Conference, 2018.

[31] Anthony J. Favaloro and Charles L. Tucker. Analysis of anisotropic rotary diffusion models for fiber orientation. Composites Part A: Applied Science and Manufacturing, 126:105605, 2019. ISSN 1359-835X. doi: 10.1016/j.compositesa.2019.105605.

[32] Moldflow Research & Development, Available online: damassets.autodesk.net/content/dam/autodesk/www/campaigns/autodesk-moldflow-summit -recording/7_RD-Update%20-%20Jin%20Wang.pdf (Accessed on 25 July 2023).

[33] Gregory M. Lambert and Donald G. Baird. Evaluating Rigid and Semiflexible Fiber Orientation Evolution Models in Simple Flows. Journal of Manufacturing Science and Engineering, 139(3): 031012, 10 2016. doi: 10.1115/1.4034664.

[34] Hongyu Chen, Peter Wapperom, and Donald G. Baird. A model incorporating the effects of flow type on fiber orientation for flows with mixed flow kinematics. Journal of Rheology, 63(3):455–464, 2019. doi: 10.1122/1.5086805.

[35] Susanne Katrin Kugler, Argha Protim Dey, Sandra Saad, Camilo Cruz, Armin Kech, and Tim Osswald. A flow-dependent fiber orientation model. Journal of Composites Science, 4(3), 2020. doi: 10.3390/jcs4030096.

[36] J. J. Cheng and I. Manas-Zloczower. Flow field characterization in a banbury mixer. International Polymer Processing, 5(3):178–183, 1990.

[37] Hongyu Chen, Peter Wapperom, and Donald G. Baird. The use of flow type dependent strain reduction factor to improve fiber orientation predictions for an injection molded center-gated disk. Physics of Fluids, 31(12):123105, 2019. doi: 10.1063/1.5129679.

[38] George L. Hand. A theory of anisotropic fluids. Journal of Fluid Mechanics, 13(1):33–46, 1962. doi: 10.1017/S0022112062000476.

[39] Suresh G. Advani and Charles L. Tucker. Closure approximations for three-dimensional structure tensors. Journal of Rheology, 34(3):367–386, 1990. doi: 10.1122/1.550133.

[40] Masao Doi. Molecular dynamics and rheological properties of concentrated solutions of rodlike polymers in isotropic and liquid crystalline phases. Journal of Polymer Science: Polymer Physics Edition, 19(2):229–243, 1981. doi: 10.1002/pol.1981.180190205.

[41] G.G. Lipscomb, M.M. Denn, D.U. Hur, and D.V. Boger. The flow of fiber suspensions in complex geometries. Journal of Non-Newtonian Fluid Mechanics, 26(3):297–325, 1988. ISSN 0377-0257. doi: 10.1016/0377-0257(88)80023-5.

[42] G. Marrucci and N. Grizzuti. Predicted effect of polydispersity on rodlike polymer behaviour in concentrated solutions. Journal of Non-Newtonian Fluid Mechanics, 14:103–119, 1984. ISSN 0377-0257. doi: 10.1016/0377-0257(84)80039-7.

[43] David A. Jack and Douglas E. Smith. Assessing the use of tensor closure methods with orientation distribution reconstruction functions. Journal of Composite Materials, 38(21):1851–1871, 2004. doi: 10.1177/0021998304048413.

[44] F. Dupret and V. Verleye. Modelling the flow of fiber suspensions in narrow gaps. In D.A. Siginer, D. De Kee, and R.P. Chhabra, editors, Advances in the Flow and Rheology of Non-Newtonian Fluids, volume 8 of Rheology Series, pages 1347–1398. Elsevier, 1999. doi: 10.1016/S0169-3107(99)80020-3.

[45] Joaquim S. Cintra and Charles L. Tucker. Orthotropic closure approximations for flow-induced fiber orientation. Journal of Rheology, 39(6):1095–1122, 1995. doi: 10.1122/1.550630.

[46] Charu V. Chaubal and L. Gary Leal. A closure approximation for liquid-crystalline polymer models based on parametric density estimation. Journal of Rheology, 42(1):177–201, 1998. doi: 10.1122/1.550887.

[47] E. D. Wetzel. Modeling flow-induced microstructure of inhomogeneous liquid-liquid mixtures, 1999. Ph.D. thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA.

[48] B. E. VerWeyst. Numerical predictions of flow-induced fiber orientation in 3-d geometries, 1999. Ph.D. thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA.

[49] Du Hwan Chung and Tai Hun Kwon. Improved model of orthotropic closure approximation for flow induced fiber orientation. Polymer Composites, 22(5):636–649, 2001. doi: 10.1002/pc.10566.

[50] D. H. Chung and Tai Hun Kwon. Invariant-based optimal fitting closure approximation for the numerical prediction of flow-induced fiber orientation. Journal of Rheology, 46(1):169–194, 2002. doi: 10.1122/1.1423312.

[51] David A. Jack, Bryan Schache, and Douglas E. Smith. Neural network-based closure for modeling short-fiber suspensions. Polymer Composites, 31(7):1125–1141, 2010. doi: 10.1002/pc.20912.

[52] F. Dupret and V. Verleye. Prediction of fiber orientation in complex injection molded parts. In Dev. Non-Newtonian Flows, 1993.

[53] S. Montgomery-Smith, D. Jack, and D.E. Smith. Exact tensor closures for the three-dimensional jeffery's equation. Journal of Fluid Mechanics, 680:321–335, 2011. doi: 10.1017/jfm.2011.165.

[54] Stephen Montgomery-Smith, David Jack, and Douglas E. Smith. The Fast Exact Closure for Jeffery's equation with diffusion. Journal of Non-Newtonian Fluid Mechanics, 166(7):343–353, 2011. doi: 10.1016/j.jnnfm.2010.12.010.

[55] NumPy.The fundamental package for scientific computing with Python, Available on: https://numpy.org/ (Accessed on 11 February 2022). .

[56] SciPy. Fundamental algorithms for scientific computing in Python, Available on: https://scipy.org/ (Accessed on 11 February 2022). .

[57] SymPy, Available on: https://www.sympy.org/pt/index.html (Accessed on 11 February 2022). .

[58] Matplotlib: Visualization with Python, Available on: https://matplotlib.org/ (Accessed on 11 February 2022).

[59] Computational-Rheology / FiberOrientation_OFUtility, Available on: https://github.com/Computational-Rheology/FiberOrientation_OFUtility.git (Accessed on 28 July 2023).

[60] scipy.integrate.solve_ivp, Available on: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html (Accessed on 11 February 2022). .

[61] numpy.einsum, Available on: https://numpy.org/doc/stable/reference/generated/numpy.einsum.html (Accessed on 10 May 2023). .

[62] Introduction - SymPy 1.11 documentation, Available on: https://docs.sympy.org/latest/tutorials/intro-tutorial/intro.htmlwhy-sympy (Accessed on 10 May 2023). .

[63] Code Generation, Available on: https://docs.sympy.org/latest/modules/codegen.htmlsympy.codegen.rewriting.create_expand_pow_optimization (Accessed on 28 July 2023).

[64] OpenFOAM: User Guide, Available online: https://www.openfoam.com/documentation/guides /latest/doc/index.html (Accessed on 10 April 2023).

[65] OpenFOAM guide/Finite volume method (OpenFOAM), Available online: https://openfoamwiki.net/index.php/Finite_volume_method_(OpenFOAM) (Accessed on 10 April 2023).

[66] Christopher Greenshields and Henry Weller. Notes on Computational Fluid Dynamics: General Principles. CFD Direct Ltd, Reading, UK, 2022.

[67] ParaView - Open-source, multi-platform data analysis and visualization application, Available on: https://www.paraview.org/ (Accessed on 11 February 2022).

[68] ames William Cooper. Java Design Patterns: A Tutorial. Addison-Wesley Professional, 2000. ISBN 9780201485394.

[69] Susanne K. Kugler, Gregory M. Lambert, Camilo Cruz, Armin Kech, Tim A. Osswald, and Donald G. Baird. Macroscopic fiber orientation model evaluation for concentrated short fiber reinforced polymers in comparison to experimental data. Polymer Composites, 41(7):2542–2556, 2020. doi: 10.1002/pc.25553.

[70] WebPlotDigitizer - Extract data from plots, images, and maps, Available on: https://automeris.io/WebPlotDigitizer/ (Accessed on 16 January 2023).

[71] Randy S. Bay and Charles L. Tucker III. Fiber orientation in simple injection moldings. part i: Theory and numerical methods. Polymer Composites, 13(4):317–331, 1992. doi: 10.1002/pc.750130409.

[72] Standard utilities, Available on: https://www.openfoam.com/documentation/user-guide/a-reference/a.2-standard-utilities (Accessed on 28 July 2023).

[73] Bruno Ramôa, Ricardo Costa, Francisco Chinesta, and J.M. Nobrega. A semi-automatic approach based on the method of manufactured solutions to assess the convergence order in openfoam. 2: 148–165, 11 2022. doi: 10.51560/ofj.v2.75.

[74] GitHub - krebeljk/openInjMoldSim: Open source injection molding simulation. A solver for OpenFOAM , Available on: https://github.com/krebeljk/openInjMoldSim (Accessed on 11 February 2022). .

Appendices

A Calculation of β for the IBOF closure

$$\beta_i = \mathbf{A}_{i,1} + \mathbf{A}_{i,2}\mathrm{II} + \mathbf{A}_{i,3}\mathrm{II}^3 + \mathbf{A}_{i,4}\mathrm{III} + \mathbf{A}_{i,5}\mathrm{III}^2 + \mathbf{A}_{i,6}\mathrm{IIIII} + \mathbf{A}_{i,7}\mathrm{II}^2\mathrm{III} + \mathbf{A}_{i,8}\mathrm{IIIII}^2 + \mathbf{A}_{i,9}\mathrm{II}^3 + \mathbf{A}_{i,10}\mathrm{III}^3 + \mathbf{A}_{i,11}\mathrm{II}^3\mathrm{III} + \mathbf{A}_{i,12}\mathrm{II}^2\mathrm{III}^2 + \mathbf{A}_{i,13}\mathrm{IIIII}^3 + \mathbf{A}_{i,14}\mathrm{II}^4 + \mathbf{A}_{i,15}\mathrm{III}^4 + \mathbf{A}_{i,16}\mathrm{II}^4\mathrm{III} + \mathbf{A}_{i,17}\mathrm{II}^3\mathrm{III}^2 + \mathbf{A}_{i,18}\mathrm{II}^2\mathrm{III}^3 + \mathbf{A}_{i,19}\mathrm{IIIII}^4 + \mathbf{A}_{i,20}\mathrm{II}^5 + \mathbf{A}_{i,21}\mathrm{III}^5, (i = 3, 4, 6)$$

$$\beta_1 = \tfrac{3}{5}[-\tfrac{1}{7} + \tfrac{1}{5}\beta_3(\tfrac{1}{7} + \tfrac{4}{7}\mathrm{II} + \tfrac{8}{3}\mathrm{III}) - \beta_4(\tfrac{1}{5} - \tfrac{8}{15}\mathrm{II} - \tfrac{14}{15}\mathrm{III}) - \beta_6(\tfrac{1}{35} - \tfrac{24}{105}\mathrm{III} - \tfrac{4}{35}\mathrm{II} + \tfrac{16}{15}\mathrm{II}\ \mathrm{III} + \tfrac{8}{35}\mathrm{II}^2)]$$

$$\beta_2 = \tfrac{6}{7}[1 - \tfrac{1}{5}\beta_3(1 + 4\mathrm{II}) + \tfrac{7}{5}\beta_4(\tfrac{1}{6} - \mathrm{II}) - \beta_6(-\tfrac{1}{5} + \tfrac{2}{3}\mathrm{III} + \tfrac{4}{5}\mathrm{II} - \tfrac{8}{5}\mathrm{II}^2)]$$

$$\beta_5 = -\tfrac{4}{5}\beta_3 - \tfrac{7}{5}\beta_4 - \tfrac{6}{5}\beta_6(1 - \tfrac{4}{3}\mathrm{II})$$

B Results and Discussion

The parameters for the following test cases were the same as for the simple shear flow cases, indicated in Section 4.2.5.

B.1 Planar Elongational flow

Due the exponential relationship between time and space, a mesh with a large length along the x-axis would be needed for the analyses of times corresponding to the steady state. The necessary mesh's length is not computationally possible with the available resources, and, therefore, the error for the steady state was not calculated.

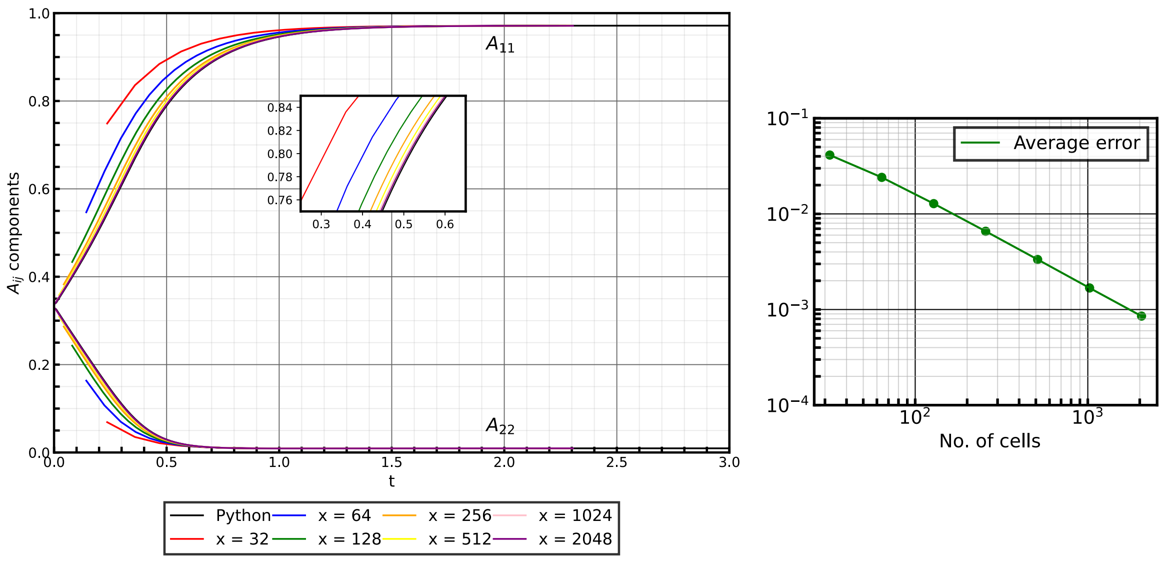

Figure B.1: Prediction of Folgar-Tucker model with the Hybrid closure for the planar elongational flow test case (left) and calculated errors (right), for different mesh refinement levels.

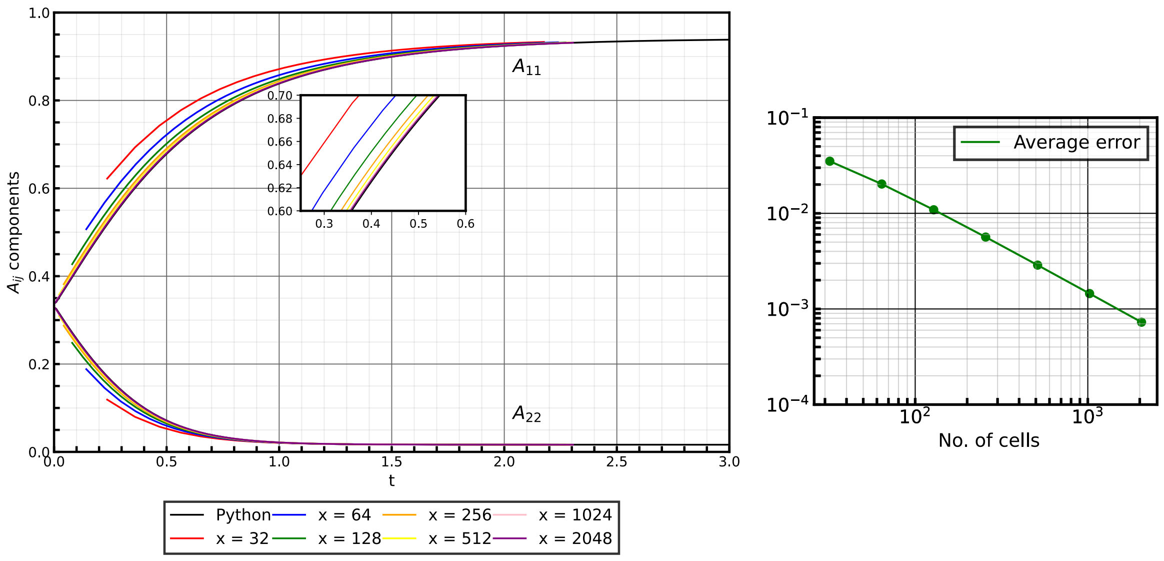

Figure B.2: Prediction of Folgar-Tucker model with the IBOF closure for the planar elongational flow test case (left) and calculated errors (right), for different mesh refinement levels.

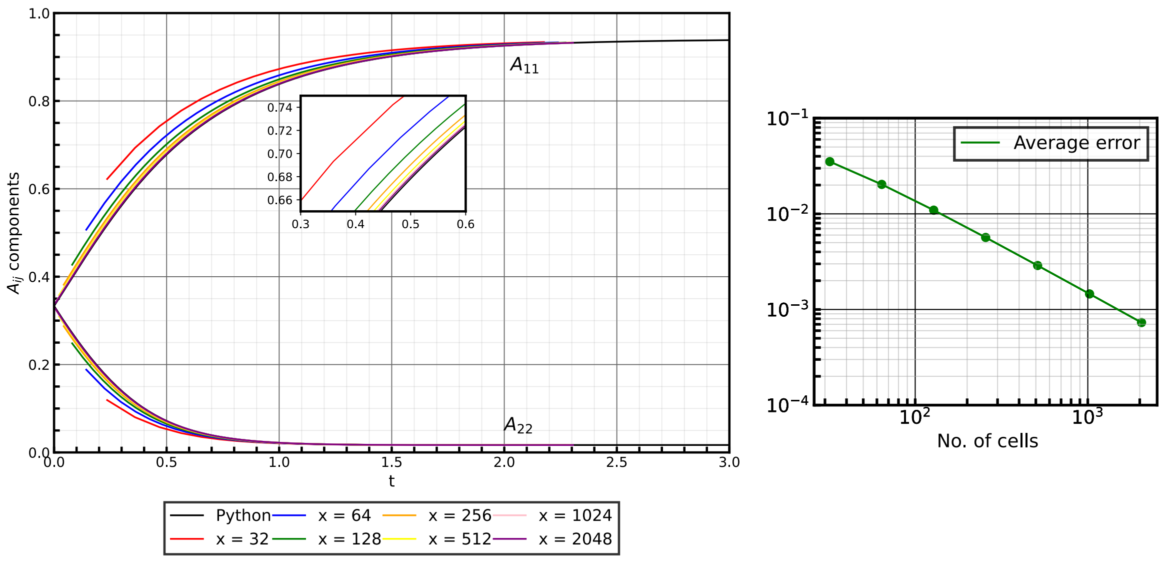

Figure B.3: Prediction of Folgar-Tucker model with the ORE closure for the planar elongational flow test case (left) and calculated errors (right), for different mesh refinement levels.

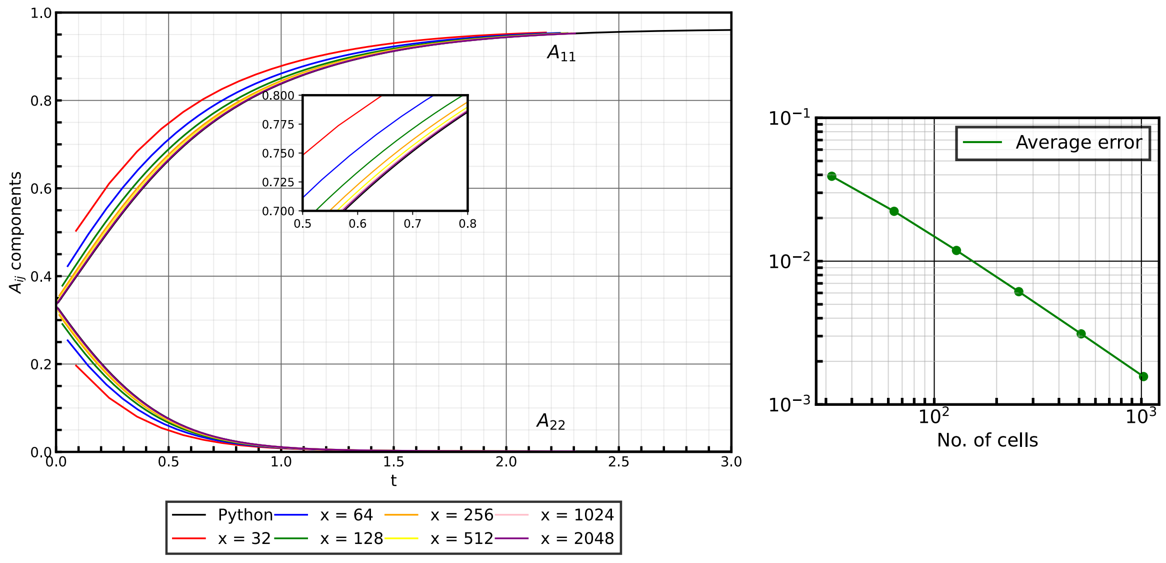

Figure B.4: Prediction of iARD-RPR model with the IBOF closure for the planar elongational flow test case (left) and calculated errors (right), for different mesh refinement levels.

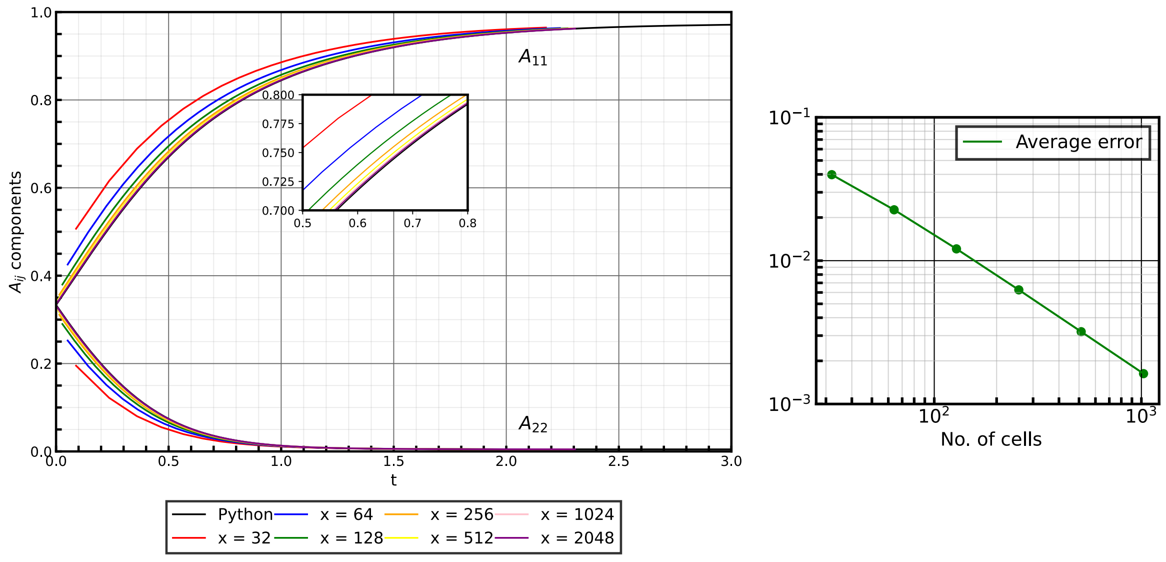

Figure B.5: Prediction of MRD-RSC model with the IBOF closure for the planar elongational flow test case (left) and calculated errors (right), for different mesh refinement levels.

B.2 Flow in a center-gated disk

Since the center-gated disk is analysed in function of space, and not time, the error associated with it was not calculated.

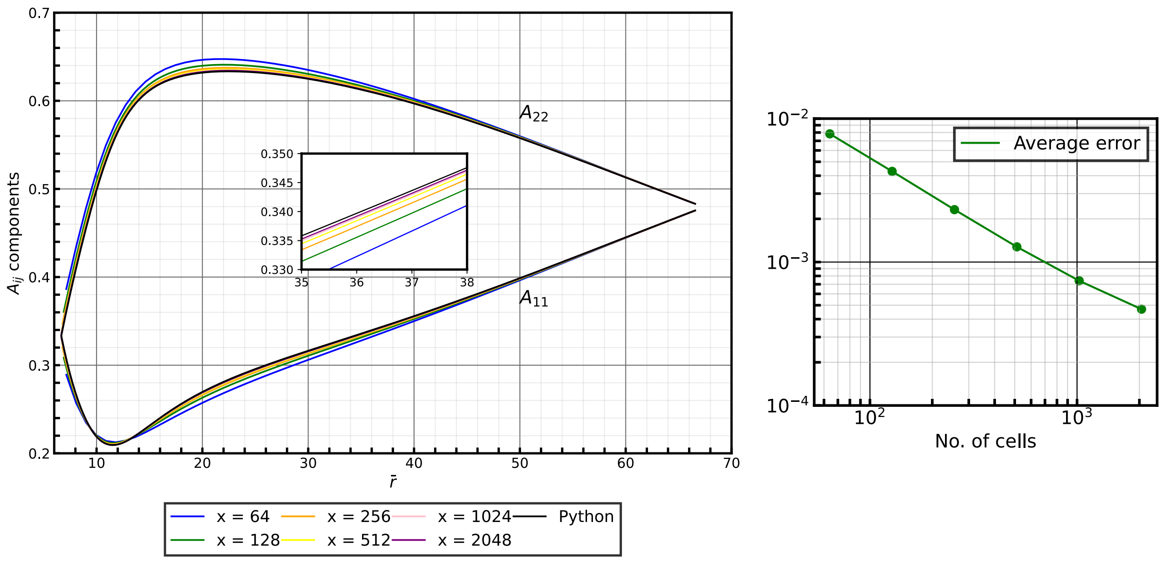

Figure B.6: Prediction of Folgar-Tucker model with the Hybrid closure for the center-gated disk case at $\tilde{z} \approx 0.1$ (left) and calculated errors (right), for different mesh refinement levels.

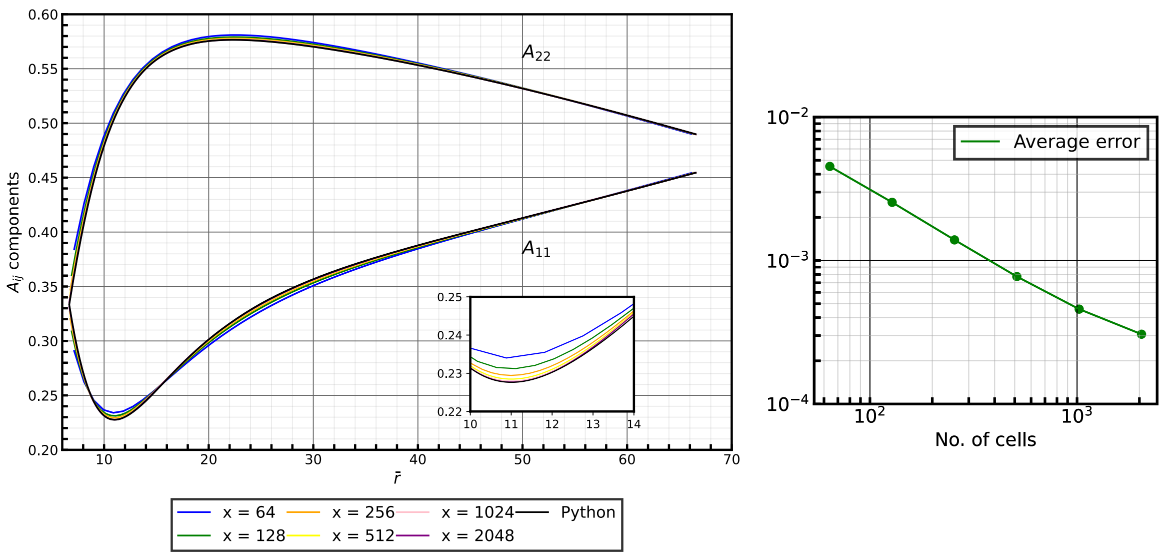

Figure B.7: Prediction of Folgar-Tucker model with the IBOF closure for the center-gated disk case at $\tilde{z} \approx 0.1$ (left) and calculated errors (right), for different mesh refinement levels.

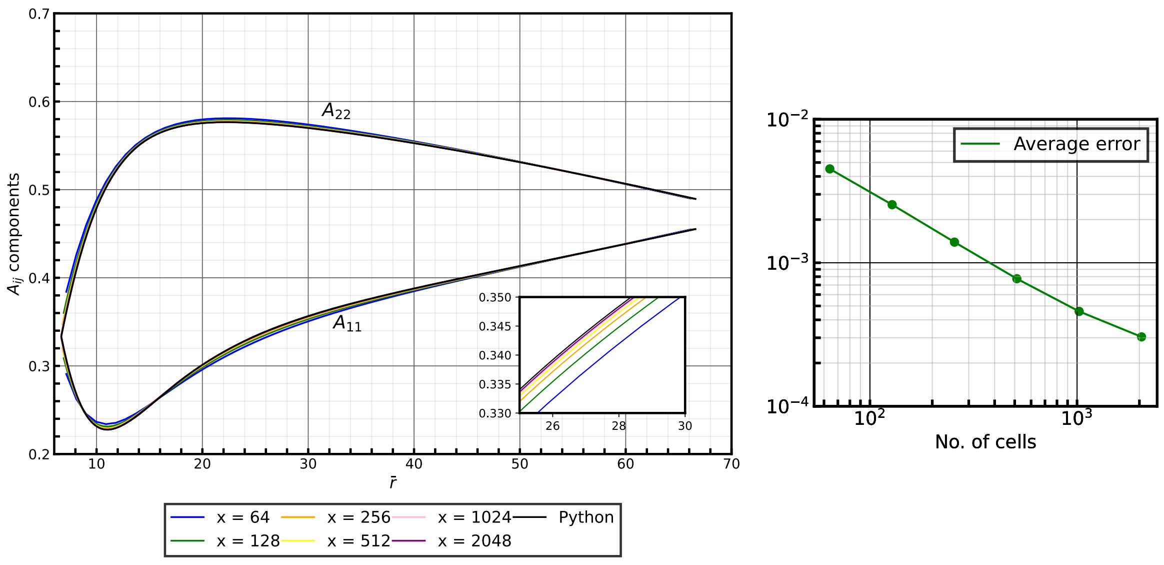

Figure B.8: Prediction of Folgar-Tucker model with the ORE closure for the center-gated disk case at $\tilde{z} \approx 0.1$ (left) and calculated errors (right), for different mesh refinement levels.

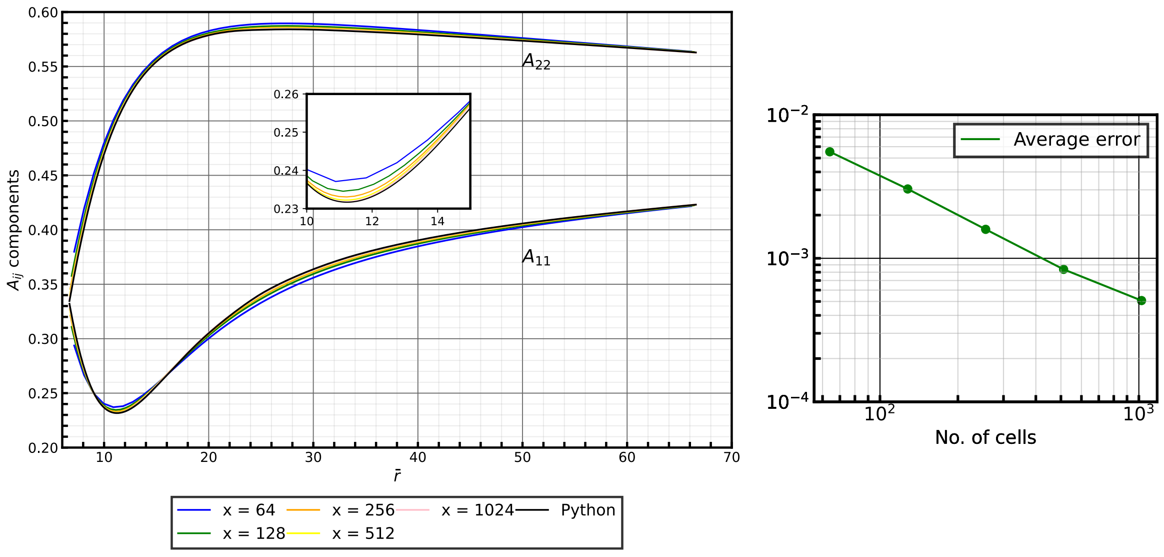

Figure B.9: Prediction of iARD-RPR model with the IBOF closure for the center-gated disk case at $\tilde{z} \approx 0.1$ (left) and calculated errors (right), for different mesh refinement levels.

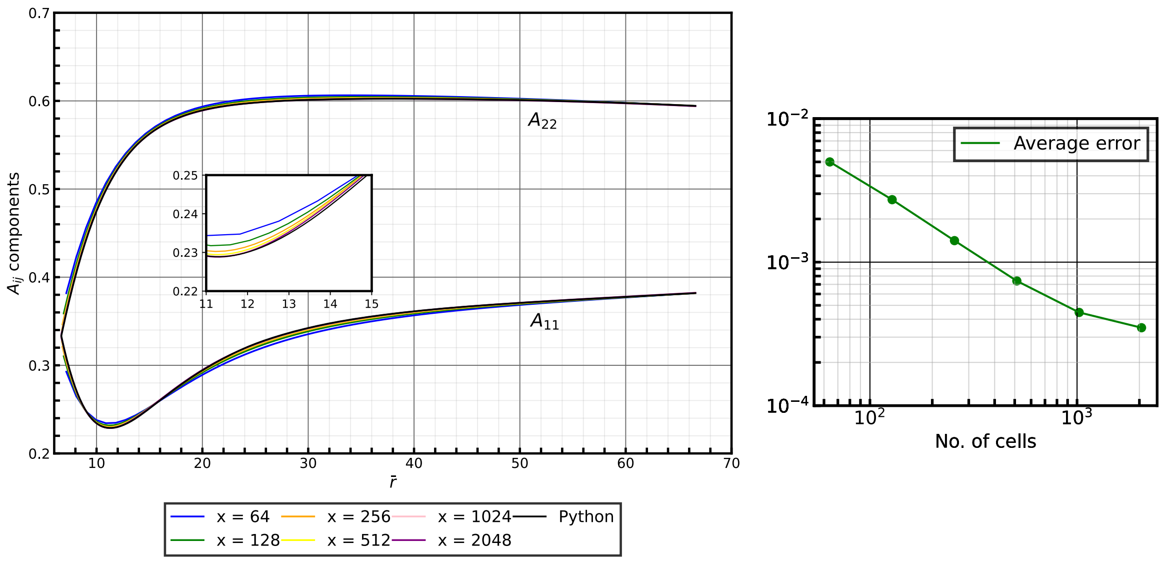

Figure B.10: Prediction of MRD-RSC model with the IBOF closure for the center-gated disk case at $\tilde{z} \approx 0.1$ (left) and calculated errors (right), for different mesh refinement levels.