An axisymmetric model for the meat analog production process in a Couette Cell based device

by Wouter van den Hoed

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday March 21, 2022.

Student number:	4204964	
Project duration:	September 1, 2020 – February 1, 2022	
Thesis committee:	Prof. dr. ir. B.P. Tighe,	TU Delft, supervisor
	Prof. dr. ir. M.J.B.M. Pourquie,	TU Delft, co-supervisor
	Prof. dr. ir. D.S.W. Tam,	TU Delft
	Ir. D. van der Stoel,	Rival Foods

TU Delft

Preface

I would like to thank my supervisor Dr. B.P. Tighe for his guidance throughout this thesis and co-supervisor Dr. M.J.B.M. Pourquie for the great effort he put into helping me understand how to handle the wild beast called "OpenFOAM".

Wouter van den Hoed
Delft, January 2022

Abstract

During the last years the demand for vegetarian products has increased. A subgroup of these vegetarian products consists of meat analogs, which are products that resemble meat in its functionality and are prepared in a similar fashion. One of the companies producing these meat analogs is Rival Foods. Rival Foods has been working on a production process based on a Couette cell, in which the dough is sheared in an annular region between two concentric cylinders. This allows them to create highly fibrous products with a thickness of roughly 3cm. Other production processes such as extrusion cooking are unable to achieve this combination of structure and thickness.

In upscaling the production process, preferably a larger gap width between the cylindrical surfaces of the cell is preferred, because the thickness of the product is a unique selling point. Larger gap widths lead to greater temperature inhomogeneity and gradients, which negatively impacts product quality. It is currently not possible to accurately measure the temperature profile throughout the cell. Therefore, in this thesis a model has been developed in OpenFOAM which calculates the temperature profile with a small number of material parameters and process conditions as input. The model assumes the ingredient mixture behaves as a temperature dependent power law fluid. Rheological measurements have been performed to quantify these temperature dependent power law parameters.

To study the influence of the viscous dissipation, preheat temperature, mixture density, and product thickness on the temperature field during processing, multiple simulations have been performed. The simulations used a time step of 0.0001s, for which the temperature, velocity, viscosity, and viscous dissipation were not yet fully converged. The principal flow in the Couette cell geometry was in the direction of rotation of the inner cylinder as expected and had a velocity with order of magnitude e-01 m/s. Besides the principal flow a secondary flow pattern has been found as well, consisting of vortices which had a velocity with order of magnitude e-03 m/s. These secondary velocity components were responsible for additional advection of heat in the simulation which resulted in a different temperature profile than expected. Since the Taylor number was well below the critical Taylor number, the existence of Taylor vortices could be excluded. After a further refinement of the time step the vortices disappeared. Running the full simulations with this time step would take months per simulation and was therefore not an option. It was concluded OpenFOAM had trouble simulating power law fluids with high viscosities.

Contents

1 Introduction

1.1. Background

In today's world quite some people are reducing their meat consumption, which leads to an increasing demand for vegetarian products. The main factors contributing to this shift towards vegetarian food consumption are health awareness, natural resource depletion, animal suffering and disease, and reduction of greenhouse gas emissions [1].

A subgroup of these vegetarian products consists of meat analogs. The definition of meat analog refers to the replacement of the main ingredient with something other than meat [1]. Different names are also used, such as meat substitute, meat alternative, fake or mock meat, and imitation meat.

Quoting Birgit Dekkers [2], one of the founders of Rival Foods:

"Meat analogs are products that resemble meat in its functionality, and are prepared by the consumer in a similar fashion as meat. Products that approach the original meat product best are considered to be the most promising to reduce meat consumption for the largest group of consumers [3]. Therefore, meat analogs should resemble meat in terms of their sensory properties, unique texture and taste, since these are key properties appreciated in meat by consumers [4]."

Rival Foods produces meat analogs that are highly structured. Structured meat analogs can already be produced using extrusion processes, but the disadvantage of these processes is that the final product can only reach a thickness of roughly 10 to 15mm. The unique selling point of Rival Foods' products is that their structured meat analogs can already reach a thickness of up to 32mm. This means that thick meat products like steaks can be recreated with their production process, whereas extrusion processes are only able to simulate the structure of smaller and thinner meat analogs. In order to obtain this structure across the entire product a specific combination of heat, shear force, and process time has to be applied.

During the last years Rival Foods has created multiple prototypes that are able to create this structure and extensive research has been done on the mechanics behind the process. Currently Rival Foods is working on increasing the product thickness while maintaining the same structure quality across the entire product. Due to the fact that heat does not distribute uniformly when applied from the outside this can easily cause a non-uniform structure, which is not desirable.

The goal of this thesis is to produce a CFD model that helps to gain more insight into the thermal and viscous behaviour of the meat analog during the production process. This model should be able to create more insight on how the temperature profile across the product changes under the influence of varying process conditions.

1.2. Introduction to meat analog structuring techniques

In order to mimic the structure of meat, it is important to know which components the structure consists of. Most of the meat that humans consume is skeletal muscle meat. The texture of these skeletal muscles is the result of the hierarchical structural organization of the muscle, which can be seen in Figure 1.1. Inside muscles, muscle fibers are organized into bundles, called fascicles. Inside these muscle fibers, myofibrils are organized into bundles as well, and these myofibril bundles are all surrounded by a sarcolemma. Together these fibrous bundles create the fibrous texture of the muscle.

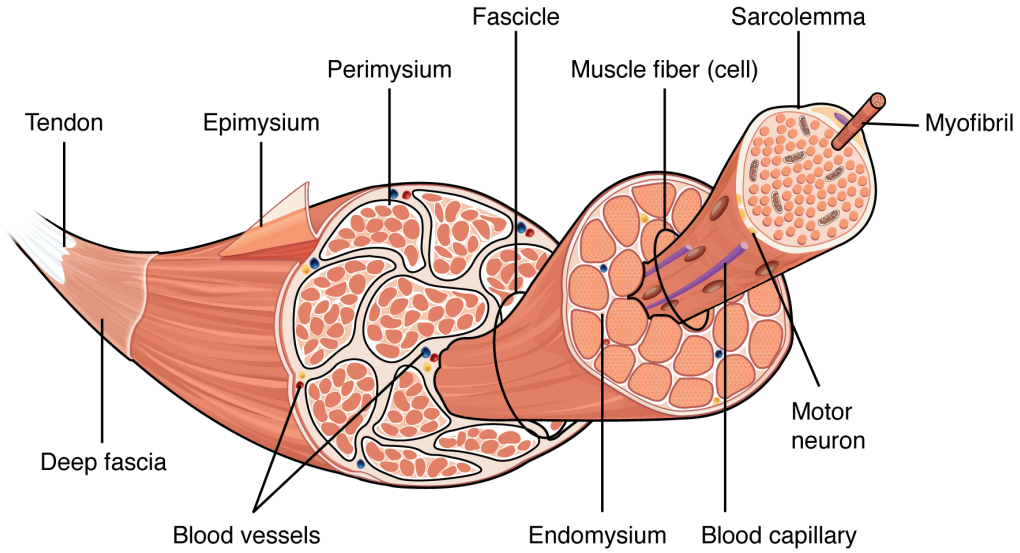


Figure 1.1: Hierarchical structure of skeletal muscle tissue [5]

The sensory attributes of meat are often related to these structural elements [6] [7]. For example, the way that the myofibrils and connective tissue are distributed have a big influence on the toughness of the meat, while the water inside the meat has a great influence on the juiciness [2]. When one wants to reproduce the sensory and textural properties of real meat in a meat analog, different structuring techniques can be used.

Globally there are two different fundamental approaches to mimic the structure of muscle-meat. One of these approaches is a top-down strategy and the other one a bottom-up strategy. The top-down strategy aims to create fibrous structure by structuring biopolymer blends using techniques like extrusion [8]. The bottom-up strategy is based on mimicking the small structural elements of real muscle fibers which can then be assembled into larger products. In general, top-down strategies are easier to upscale, but the bottom-up strategies approach closer to the structure of meat. An overview of the different strategies and techniques is depicted in Figure 1.2.

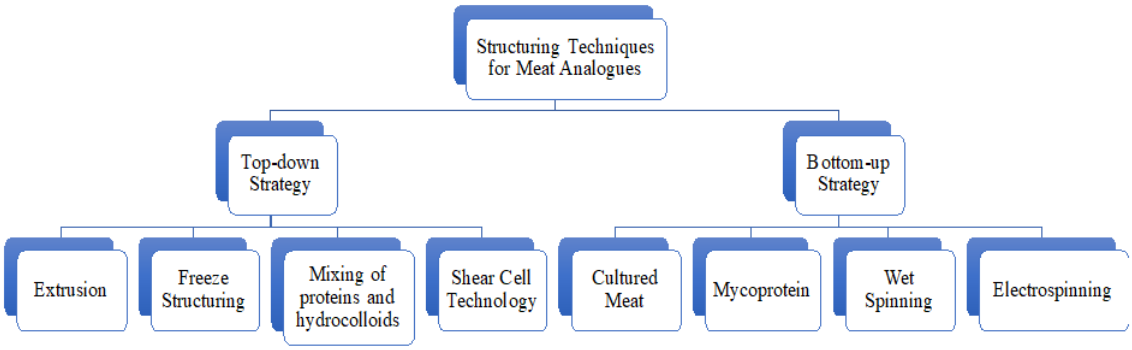


Figure 1.2: Structuring techniques for meat analogs [8]

Currently, one of the most commercially applied technologies for the production of meat analogs is extrusion cooking, see Figure 1.3. A downside of the extrusion process is that the process conditions (time, temperature, shear) are coupled, which does not allow for structure optimization by changing these process conditions independently from each other.

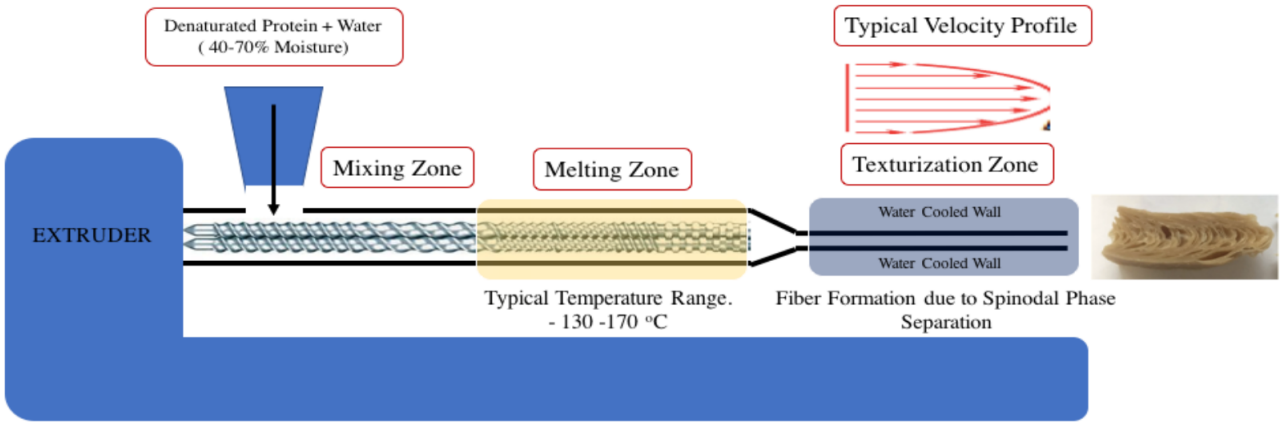


Figure 1.3: Basic setup of an extrusion process to develop meat analogs [8]

New methods of protein structuring have been introduced during the last 15 years. Shearing devices in which intensive shear can be applied, based on the design of rheometers, called shear cells were first developed using a cone-in-cone geometry at Wageningen University [9] [10]. Later the geometry of these devices was changed by Krintiras [11] to improve the uniformity of the shear force distribution which led to the development of the Couette cell geometry [11].

An important advantage of using the novel shear and Couette cell is that the process conditions (shear force, temperature, and process time) can be decoupled. Also, the behavior of (bio)polymers under shearing deformation is less complex as compared to the very complex and sometimes even chaotic flow patterns that can be encountered during extrusion. This makes it easier to model the process.

The device geometries for these two new methods can be seen in Figure 1.4. In the cone-in-cone device, the bottom cone is rotating while the top cone is stationary, and the material placed in between the cones is sheared. In the concentric cylinder device the inner cylinder is rotating while the outer cylinder is stationary [2].

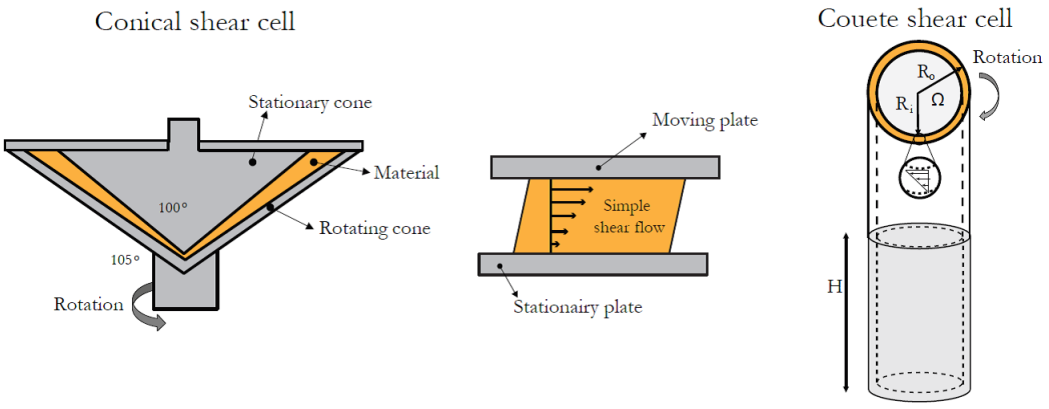


Figure 1.4: Illustration of cone-in-cone and concentric cylinder device in which materials can be deformed with simple shear flow [2]

These two new technologies were developed based on the principle of applying simple shear and heat to the protein mixture. First, the shear cell was developed, which is a device which has a cone-in-cone design that is able to create structure in soy-based mixtures, to create a final product that is similar to

meat (structure wise). However, the shear cell design does not allow for much upscaling due to the fact that the thickness of the slab increases with the increasing radius of the cell. This leads to significant differences in temperature and stress gradients over the shear cell, resulting in an inhomogeneous final product. Therefore, the Couette cell concept was developed and presented by TU Delft student George A. Krintiras in 2016 [12], see Figure 1.5. This Couette cell has a height of 332 mm, an inner radius of 95 mm, an outer radius of 125 mm, and can produce a slab with a total volume of 6.88L per batch. The Couette Cell concept, which looks like two concentric cylinders with a gap in between for the ingredient mixture has been studied previously by TU Delft students because it seems to be a promising design when it comes to upscaling to industrially relevant production capacities.

Figure 1.5: "The big Couette Cell" design by Krintiras (Courtesy of Rival Foods)

Krintiras's research on the Couette Cell has been done in collaboration with Wageningen University and Research (WUR). As a result of the extensive research in this area on WUR's part, Rival Foods has been created by Birgit Dekkers and Ernst Breel. This spin-off company from the Food Process Engineering lab of Wageningen University and Research revolutionizes the plant-based meat analogs market with a unique process and products. Their mission is stated as follows.

"Develop, produce and sell whole-cut plant-based meats that provide unparalleled structure, juiciness and taste for anyone who loves the art of cooking."

At the core of their company is the Couette cell-based production process, for which they have developed several (prototype) machines so far. Currently they are developing the next generation of this machine to allow for the production of larger quantities of plant-based meat products. This requires obtaining a deeper understanding of the dynamics of the process, scalability of the process and to develop and realize new, tailor-made technical solutions that improve efficiency and reduce complexity.

1.3. Meat analog structuring mechanics using Couette Cells

To get more insight into the mechanics behind the structure formation using Couette/shear cells, the research by Dekkers [2] has been used. This research has shown that a concentrated two-phase biopolymer blend is needed for fibrous structure formation. These two phases consist of mostly protein(s) and/or polysaccharides in water.

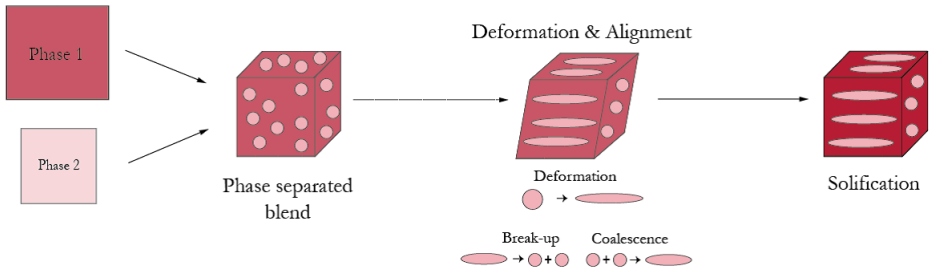


Figure 1.6: Graphical illustration of a phase separated biopolymer blend, which forms a water-in-water emulsion, which shows droplet coalescence, break-up and deformation [2]

The amount that each of the phases deforms is highly influenced by the interaction between the two phases, which depends on the following process conditions:

- shear rate
- temperature
- time

In order to achieve the most fibrous structure, the shear should be able to deform the dispersed phase, but should cause only limited break-up of this phase. The onset of break-up depends on several forces, namely the external forces, the viscous forces, and the interfacial forces. The viscosity of both the continuous and dispersed phase can be greatly influenced by the temperature during the process. It is also important to note that the temperature can cause (chemical) changes in the material over time, which influence the viscosity and deformability of the product as well.

1.4. Research Motivation

Currently Rival Foods is working on up-scaling their production capacity. The production process can be up-scaled using several approaches. The length of the Couette Cell can be increased, the width of the gap inside the Couette Cell can be increased by changing the radii, thus resulting in a thicker slab, or the amount of Couette Cells being used for production can simply be increased.

For Rival Foods up-scaling by increasing the thickness of the slab is the most interesting option. This is due to the fact that the thickness of their product together with the structure are their unique selling points which make it stand out from their competition. "The Big Couette Cell" from Krintiras's research had a gap width of 30mm. Currently Rival Foods is optimizing a Couette Cell design which has a gap width of 50mm. The problem with this larger gap width is that the heat does not distribute as uniformly as desired and thus it is hard to obtain a product that has the same quality throughout the entire slab.

Larger gap widths lead to greater temperature inhomogeneity and gradients, which negatively impacts product quality. It is currently not possible to accurately measure the temperature profile throughout the cell. Therefore, the goal of this research is to develop a model that can predict the profile with a small number of material parameters and process conditions as input. Different variations on the geometry of the Couette Cell based device can then be tested.

1.5. Goal and scope

The goal of this thesis is to model part of the production process that is currently used at Rival Foods for producing structured meat analogs. In this process shear and heat are applied to a non-Newtonian water-in-water emulsion in a complex geometry.

The original production unit was a conical shear cell, later the design was changed to a Couette Cell, and currently a modified device, still based on the Couette Cell, is being used. This device has a more complex geometry than the Couette cell, which makes it more challenging to model and therefore the geometry will be assumed to be similar to the Couette cell.

Apart from the geometry, the behavior of the ingredient mixture will also be complex. Based on research by van Dijk [13] a power law model is suitable to model the mixture. The varying temperatures that are applied to the mixture during the process will likely have a significant effect on the viscosity of this water-in-water emulsion, consisting of Soy Protein Isolate (SPI) and Wheat Gluten (WG).

It is important to make a well-considered choice on which software should be used to model the process. One option would be to create a model using Matlab, and the other option would be the use of a CFD program such as OpenFOAM, Ansys Fluent, or Ansys Polyflow.

The advantage of using Matlab is that there is complete control over the governing equations and calculation methods used. The disadvantage is that the 3D geometry is likely too complex to model and therefore it will be necessary to assume a more simple geometry so that the geometry can be modelled in 2D.

The advantage of using a CFD program is that more complex geometries are easier to use due to the availability of specific meshing software that works well together with CFD programs. The disadvantage is that the user can be more limited in choice regarding the possible models and governing equations, depending on which CFD program is used of course. From the available CFD options, Ansys Fluent seems to be the most robust and easiest to learn, but more exotic Newtonian laws (visco-elasticity) are not available in this software package. Some versions of OpenFOAM do offer visco-elastic modelling, but there are different versions with limited compatibility. Considering the requirements on the model, Ansys Polyflow seems to be the best option, due to the fact that Polyflow is ideally suited for modelling viscous/viscoelastic flows and rheologically complex fluids. This program is often used for applications with polymers, glass, plastics, rubber, and paints. However, this program requires significant start-up costs in the form of user training, which were considered prohibitive for this project. Therefore OpenFOAM seems to be the best option due to its capabilities when it comes to visco-elastic modelling and due to the availability of extensive documentation and an existing user base in the Process & Energy department at TU Delft. Another important reason for choosing OpenFOAM is that it is free to use. As an engineer being able to work with free open source software like OpenFOAM can be a useful tool to have, due to its availability in nearly any work environment.

Apart from the modelling part there will also be a small experimental part where rheological measurements will be made to obtain data that is needed for the model. Once the model is functional, validation tests should be done to check whether the model behaves as expected and if the real process is in agreement with the predictions made by the model. If this all works out the CFD model can be used to figure out what the influence of varying the geometry of the device or certain process variables will be without having to perform experiments or build new prototypes for every variation made on the production process.

1.6. Timeline

The project started with a literature study to gather information regarding the shear and Couette cell production process, the different prototypes used, and the used measurement equipment. The main sources used for this purpose were the master and PhD theses by Göbel [14], van Dijk [13] , Diaz [15], Krintiras [12], and Dekkers [2]. The first four of these theses were mainly focused on the technical aspects of the prototypes and modelling. The PhD thesis by Birgit Dekkers focused more on understanding the underlying mechanisms for structure formation and finding ways to observe and measure the structure properties.

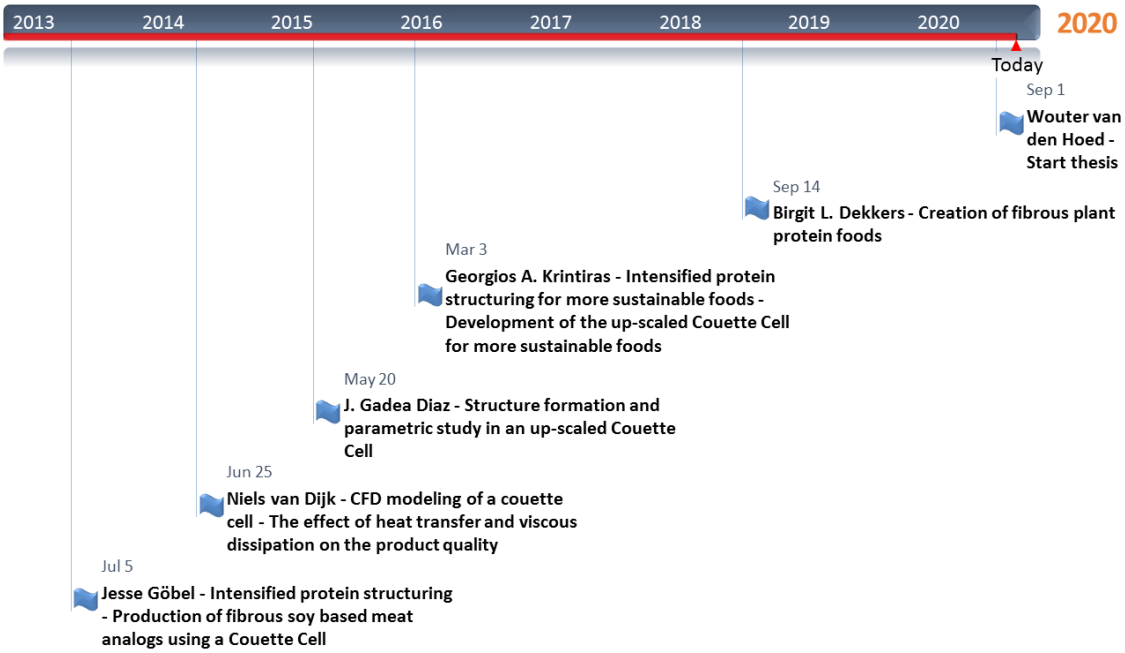


Figure 1.7: Timeline relevant research from previous TU Delft students

With the available information from the previous theses written on Couette cells, a plan was made on how to model the process. Then a timeline for the thesis with a global overview of expected activities was created roughly 1.5 months after starting the thesis. At the end of the thesis this timeline can be re-evaluated to see what did and did not go according to planning. The timeline has been included in Figure 1.8

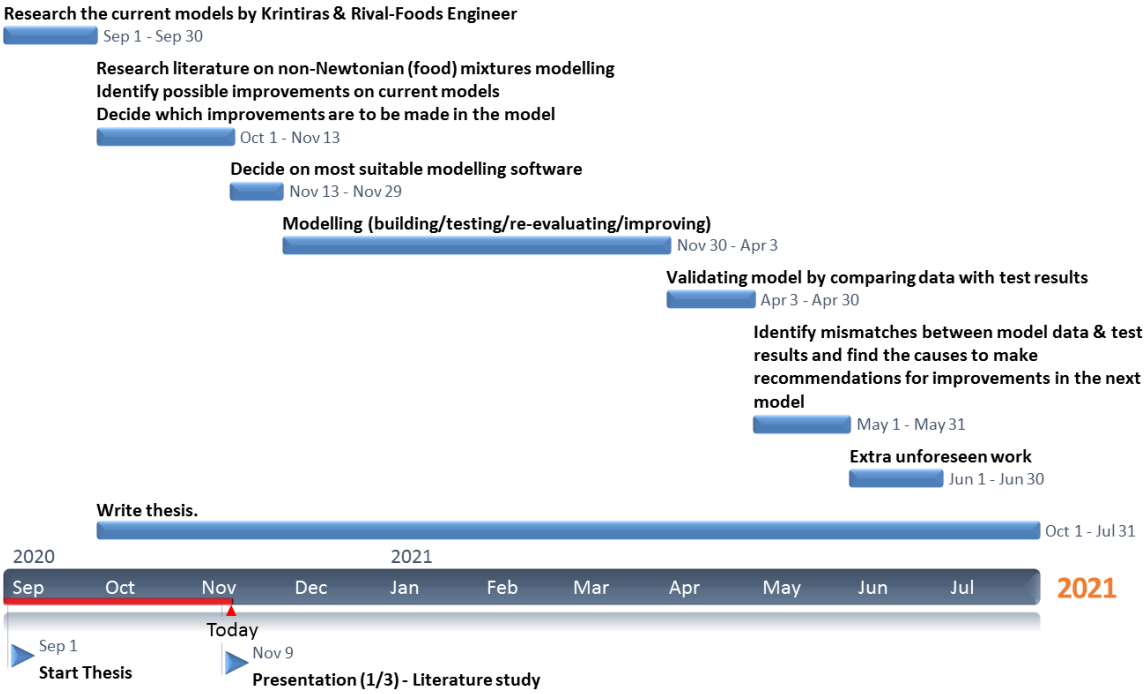


Figure 1.8: Timeline thesis

2

Literature Review

Couette Cells as a production unit for structured meat analogues have been researched during the past 15 years. The literature reviewed for this project focused on finding relevant information such as the requirements for structure formation and the underlying processes, the optimal process conditions to maximise the structure formation in the meat analogue, and the boundary conditions and assumptions that lead to the governing equations used in the CFD model.

2.1. Meat analogue structuring mechanics and process conditions

More detailed information based on the conclusions from the research by Dekkers [2] is provided below to better understand the mechanics behind structure formation using Couette Cells.

In order to understand what is meant when talking about a fibrous product it is important to have a clear definition of it. The definition of a fibrous product that is used by Dekkers [2], and thus is used by Rival Foods, is as follows:

"We describe a fibrous product as a protein matrix, which is disrupted by a dispersed, deformed, weak phase: filaments that act as matrix breakers. When this dispersed phase is not or insufficiently deformed, an isotropic product is obtained, whereas sufficient deformation of the dispersed phase results in anisotropic properties. This dispersed, deformed phase may be weak, or the adhesion between the continuous and the dispersed phase may be weak. Upon stretching the material, it will break upon the deformed, dispersed phase, resulting in a fibrous appearance as shown in Figure 2.1. The dispersed phase is aligned in the shear flow direction, while there is no orientation in the other directions."

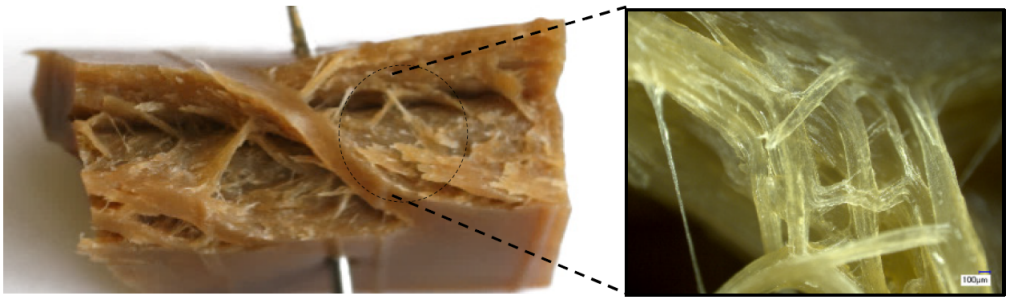


Figure 2.1: Fibrous appearance of a product prepared from a soy protein isolate (SPI) - pectin blend at 140 °C in a conical shear cell [2]

To create a fibrous structure, a mixture containing solids and liquids is normally required. The solids consists of mostly protein and/or polysaccharides and the liquid is water. The mixture of solids can be created from separate ingredients, such as soy protein isolate (SPI) and wheat gluten (WG) but there are also naturally occurring ingredients which contain both of these solids, such as soy protein concentrate

(SPC). In the products created up till now, almost no fat has been used. If sustainability is considered most important in the production process then it is better to choose the SPC option. This is due to the fact that quite a lot of energy is required in order to create SPI, because soy beans have to go through extensive fractionation processes.

In order to create the most fibrous structure, many different ingredient mixtures have been tested [2], of which the mixtures with SPI and SPC seemed most promising. There are 3 ingredient compositions which have been the most successful for creating structure. These mixtures are as follows:

1. 23 wt% SPI, 7 wt% WG, 69 wt% demineralized water, and 1 wt% sodium chlorine
2. 35 wt% SPC and 65 wt% demi water
3. 45 wt% SPC and 55 wt% demi water

Since this research the ideal ingredient composition has been further optimized and currently a mixture of 15 wt% SPI, 15 wt% WG, 69.5 wt% demineralized water, and 0.5 wt% sodium chlorine is being used.

In previous research the water holding capacity (WHC) of the ingredient mixtures was measured and experiments showed that the best structure was achieved when the dough was not completely hydrated. Thus the mixtures were only hydrated to a certain level which was below the WHC, so that a crumbly dough was created.

During the shearing process a certain amount of heat and shear are applied to the ingredient mixture over a certain period of time. Diaz [15] has performed a parametric study for Krintiras' up-scaled Couette cell design to find the optimal process temperature, time, and shear. This study showed that temperatures below 100 °C did not show the required structure. Decent structure could be created when the inner cylinder had a rotation rate of 25-35 RPM, the process time had to be between 25-35 minutes and the temperature at 120 °C. Using measurements of the Anisotropy Index (AI_σ) and visual inspection it was determined that the best structure was obtained using 30 RPM, 30 minutes and 120 °C as process conditions.

2.2. Rheology

Rheology is the branch of science that studies the way that a fluid flows and deforms in order to find relations between force, deformation and time. The word rheology is a combination of the Greek words 'rheo' and 'logia' which translate to 'flow' and 'study of', thus rheology means the study of flow. It is important to know that rheology not only deals with the flow of fluids but also with the deformation of solid-like materials. A part of rheology that is of special interest for this thesis is the behaviour of complex viscoelastic materials which can exhibit behaviour of liquids as well as solids, depending on the forces that are applied to it and the speed at which these forces are applied [16].

2.2.1. Solids vs fluids

In rheology a fundamental difference between solids and fluids can be found when looking at the mechanism behind stress build up. The stress inside a solid material is determined by the amount of deformation the material undergoes. As long as this deformation takes place in the linear elastic regime, this stress can be calculated using equation 2.1.

$$\tau = G\gamma \tag{2.1}$$

Where G is the shear modulus and γ is the shear strain.

In rheology, fluids can be classified as either Newtonian or non-Newtonian. An overview with common types of fluids is shown in Figure 2.2

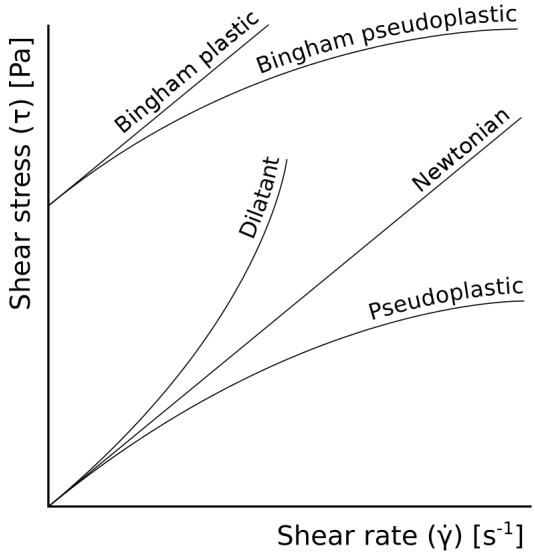


Figure 2.2: Overview of the stress as a consequence of an applied steady state shear flow in common types of fluids in rheology [17]

In Newtonian fluids the shear stress and shear rate are linearly related, as shown by Equation 2.2. Therefore the viscosity is only variable with temperature and pressure, where the viscosity generally decreases at higher temperatures and increases at higher pressures.

$$\tau = \mu \frac{du}{dy} \tag{2.2}$$

Non-Newtonian fluids are those where the viscosity varies not only with temperature but also as a function of the applied shear rate or shear stress. The most common type of non-Newtonian behaviour is pseudoplastic flow, or shear thinning flow, in which the fluid viscosity decreases with increasing shear.

2.2.2. Viscoelasticity

In materials science there is also a group of materials called viscoelastic materials. These materials exhibit both the elastic behaviour found in solid materials and the viscous behaviour found in liquids while

being deformed. Depending on whether these materials behave more like a solid or fluid they are called viscoelastic solids or viscoelastic fluids. A viscoelastic solid tends to go back to its original shape after deformation, while a viscoelastic fluid does not resist changes in shape if it is given enough time to relax all shear stresses.

Whether a material behaves more like a viscoelastic solid or liquid can be determined using dynamic mechanical analysis (DMA). In such an analysis an oscillatory stress can be applied to the material and the resulting strain can be measured. DMA can be performed in Small amplitude oscillatory shear (SAOS) form or in Large amplitude oscillatory shear (LAOS) form. The two most used test modes used in DMA are temperature sweeps and frequency sweeps.

2.2.3. Small amplitude oscillatory testing

SAOS tests are the most common method to measure the viscoelastic properties when using a rotational rheometer [16]. In this test a sinusoidal oscillation is applied to the sample in a continuous manner. The amplitude of the oscillation is the maximum strain or stress applied to the sample and the number of oscillations per second is equal to the frequency of the test.

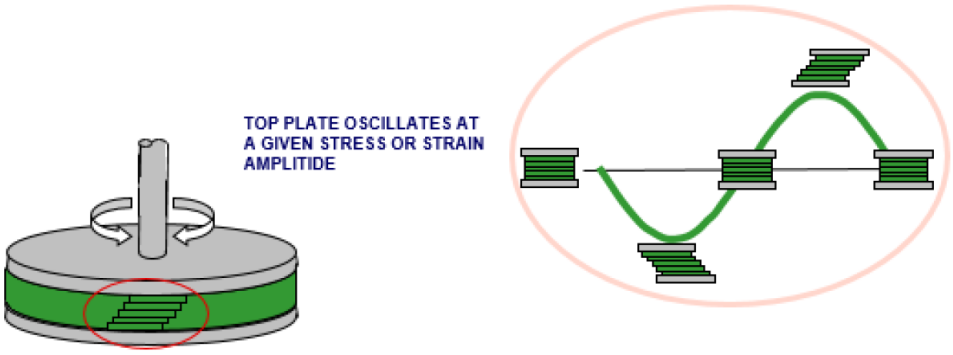


Figure 2.3: Illustration showing a sample loaded between parallel plates with an oscillatory (sinusoidal) shear profile applied [16]

A parallel plate setup is common for oscillatory testing. In this setup a sample is placed in the gap (h) that exists between two plates. The upper plate can be oscillated at adjustable stress and strain amplitudes and frequencies (Figure 2.3). The motion created by the oscillating plate can be represented as a sinusoidal wave, where the strain or stress amplitude can be plotted on the y-axis and the time on the x-axis. In a controlled strain test the angular displacement is the controlled variable and the measured torque required for that displacement is used to calculate the shear stress. In a controlled stress test the an oscillating torque is applied to the upper plate and the amount of angular displacement is measured, which is used to calculate the strain.

The ratio of the applied stress (or strain) to the measured strain (or stress) gives the complex modulus (G^*), which is a quantitative measure of material stiffness or resistance to deformation.

$$G^* = G' + iG'' \tag{2.3}$$

$$G' = \frac{\sigma_0}{\epsilon_0} * cos(\delta) \tag{2.4}$$

$$G'' = \frac{\sigma_0}{\epsilon_0} * sin(\delta) \tag{2.5}$$

Where σ_0 and ϵ_0 are the amplitudes of the stress and strain respectively, and δ is the phase shift between them as shown in Figure 2.4.

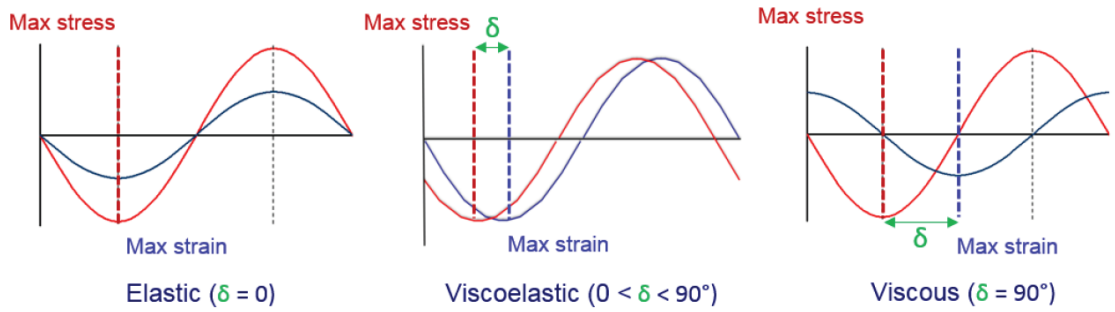


Figure 2.4: Stress and strain wave relationships for a purely elastic (ideal solid), purely viscous (ideal liquid) and a viscoelastic material [16]

For a material that is purely elastic, where the stress is proportionate with the strain, the maximum stress will be at the maximum strain and therefore the stress and strain are in phase. For a material that is purely viscous, where the stress is proportionate with strain rate, the maximum stress will be at the maximum strain rate and therefore the stress and strain are out of phase by a quarter of a cycle (90° or $\pi/2$ radians). Since viscoelastic materials show a combination of elastic and viscous behaviour, the phase difference between stress and strain will be somewhere between these two extremes.

If the elastic modulus is bigger the material is called a viscoelastic solid and if the viscous modulus is bigger the material is called a viscoelastic liquid. The complex viscosity of a viscoelastic material can be calculated using Equation 2.6.

$$|\eta^*| = \sqrt{\left(\frac{G'}{\omega}\right)^2 + \left(\frac{G''}{\omega}\right)^2} \tag{2.6}$$

The complex viscosity of viscoelastic materials is a useful property to know because of the Cox-Merz relation. The Cox-Merz relation can be used in case it is not possible to perform steady state viscosity measurements under the required process conditions, due to equipment restrictions or sample breakdown. It was observed by Cox and Merz [18] that for many polymeric systems an empirical relation exists between the steady state shear viscosity, η, plotted against shear rate, $\dot{\gamma}$, and the magnitude of the complex viscosity, $|\eta^*|$, plotted against angular frequency, ω, see Equation 2.7

$$|\eta^*(\omega)| = \eta(\dot{\gamma}) \tag{2.7}$$

2.3. Mixture properties

A short description of the rheological measurements performed on Rival Foods' ingredient mixture in previous research by Krintiras [12] and van Dijk [13] will be given below, where the measurement methods and data processing will be explained. As described in section 2.2.1, the viscosity of non-Newtonian fluids can be dependent on shear rate (apart from the temperature dependency).

This type of behaviour has also been found in Rival Foods' unprocessed ingredient mixture. Three different mixtures have been tested, one based on Soy Protein Isolate (SPI) and two based on Soy Protein Concentrate (SPC). The composition of each of the mixtures was as follows. The first mixture contained 23% SPI, 69% demineralized water, 1% sodium chlorine, and 7% gluten. The second mixture contained 35% SPC and 65% demi water. The third mixture contained 45% SPC and 55% demi water.

During these measurements a frequency sweep was performed at constant temperature to find the elastic modulus G', also called the storage modulus, and the viscous modulus G'', also called the loss modulus, at different frequencies. The results from these measurements on the SPI sample are shown in Figure 2.5. The measurements were performed using a TA Instruments AR-G2 Rheometer at the Chemical Engineering Faculty of the TU Delft.

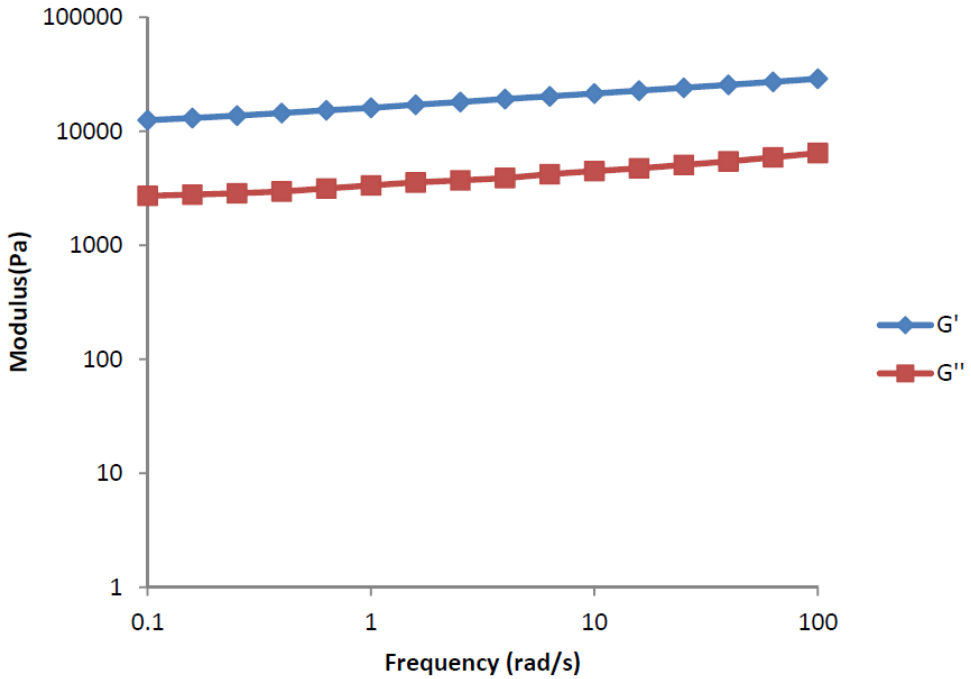


Figure 2.5: Results oscillatory test with SPI and wheat gluten mixture [13]

The moduli were then used to calculate the complex viscosity using equation 2.6. The complex viscosity is plotted against the angular frequency and the power-law model fitted to these data points. In Figure 2.6 it can be seen that the trend line has a perfect fit with $R^2 = 1$ using the Equation $y = 16603x^{-0.878}$.

Using the Cox-Merz rule, the complex viscosity is taken equal to the dynamic viscosity. The relations between the shear stress and shear rate, and between the viscosity and shear rate for a power-law fluid are shown in Equation 2.8 and Equation 2.9. Equation 2.9 shows, that contrary to the Newtonian fluid model which uses a viscosity μ, the power-law model makes use of two different parameters, namely K and n. K is the flow consistency index with SI units $[Pa * s^n]$, and n is the flow behaviour index which is dimensionless. By equating the trend line formula and Equation 2.9, the power-law indices were calculated and the following values were found, $K = 16603$ and $n = 0.13$.

$$\tau = \kappa \left(\frac{du}{dy} \right)^n \tag{2.8}$$

$$\mu = \kappa \left(\frac{du}{dy} \right)^{n-1} \tag{2.9}$$

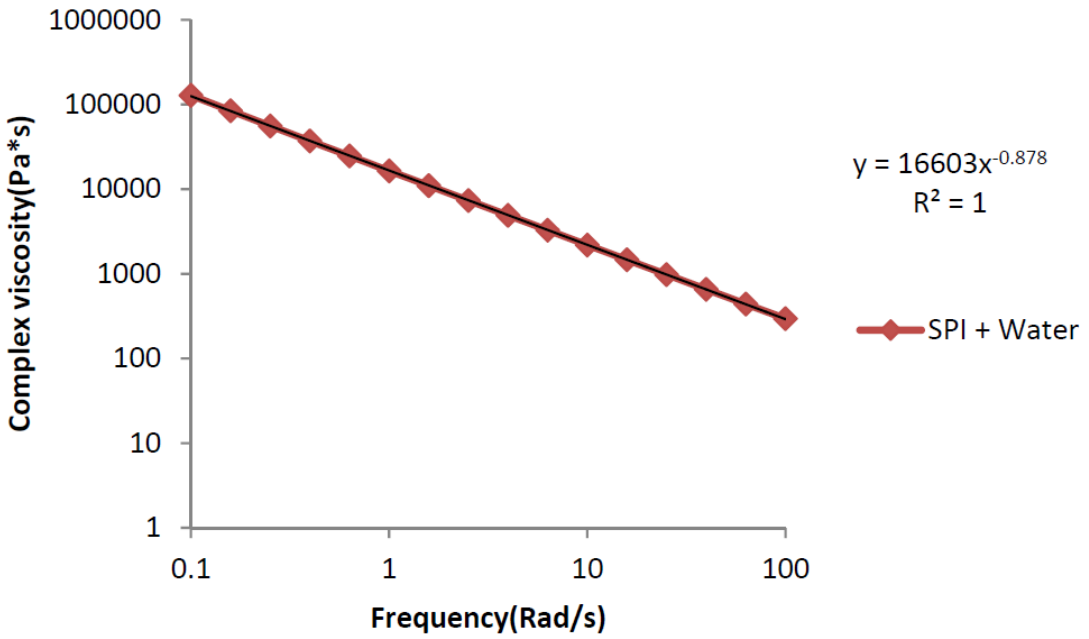


Figure 2.6: Complex viscosity of the SPI mixture with a regression line [13]

The same tests were also performed for both of the SPC mixtures. For the 35% and 45% SPC mixture this resulted in power-law indices of $K = 93097$ and $n = 0.102$, and $K = 171015$ and $n = 0.123$ respectively.

In van Dijk's research several features were mentioned that may have impacted the accuracy of the measurements. First, the measurements were not performed under optimal conditions, due to the sample not being in a closed chamber. This means that water could evaporate out of the sample during testing, which may have influenced the composition and therefore the results. Secondly, it was reported that the measurements have all been performed at room temperature, even though the viscous properties of the SPC mixtures were highly influenced by the temperature. This temperature dependence was therefore not taken into account in the measurements.

2.4. Couette flow and Taylor-Couette flow

In fluid dynamics Couette flow is the flow between two parallel plates, where one of the plates has a tangential velocity relative to the other. The velocity profile of the Newtonian fluid in between the plates can be visualised as in Figure 2.7.

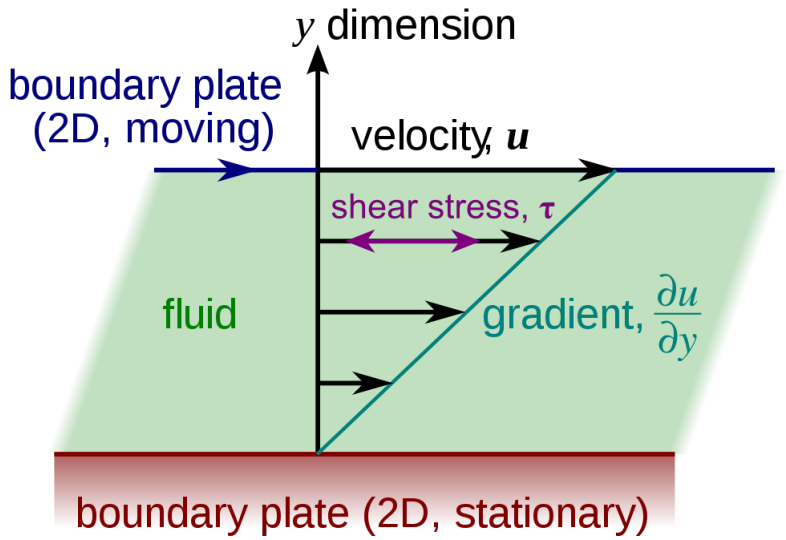


Figure 2.7: Simple Couette configuration using two infinite flat plates [19].

Couette flow between infinite parallel plates is often imitated by using a Couette cell, in which Taylor-Couette flow can be found. A Couette cell consists of two concentric cylinders, both of which can rotate, with a fluid in between.

A Couette cell can be characterized by the following parameters:

- the radius ratio, $\eta = \frac{R_i}{R_o}$
- the aspect ratio, $\Gamma = \frac{L}{R_o - R_i}$
- the Reynolds number at the inner cylinder, $Re_i = \frac{R_i(R_o - R_i)\Omega_i}{\nu}$
- the Reynolds number at the outer cylinder, $Re_o = \frac{R_o(R_o - R_i)\Omega_o}{\nu}$

Where $R_i[m]$ and $R_o[m]$ are the inner and outer radius, L is the height of the Couette Cell, $\Omega_i[rad/s]$ and $\Omega_o[rad/s]$ are the angular velocities at the inner and outer cylinder, and $\nu[m^2/s]$ is the kinematic viscosity.

Couette cells with a radius ratio $\eta > 0.97$ are described as "narrow-gap" cells [20]. The Couette cell used in this study has a radius ratio $\eta = 0.83$ and is called a "wide-gap" concentric-cell cylinder. If the radius ratio is not close enough to the value 1, the flow in the Couette cell can no longer be used as an approximation of the flow between two infinite plates because the velocity gradient will deviate too much from the linear relationship shown in Figure 2.7.

This study, apart from using a wide-gap Couette cell, also uses a non-Newtonian fluid, for which the velocity gradient across the cylinder gap by definition does not follow a linear relationship. The velocity profile and the shear rate can be calculated for given boundary conditions and fluid properties. The following boundary conditions are used: $v_{\theta_i} = \Omega R_i$ at $r = R_i$, and $v_{\theta_o} = 0$ at $r = R_o$, where v_θ is the azimuthal velocity of the rotating cylinder.

With only the inner cylinder rotating the shear stress can be defined as [21]

$$\tau = \frac{T}{2\pi R_i^2 H} \tag{2.10}$$

where τ [Pa] is the shear stress, T [Nm] is the torque applied at the inner cylinder, and H [m] is the height of the Couette cell.

The shear stress for a power law fluid can be written as

$$\tau = K\dot{\gamma}^n = K\left(r\frac{d(v_\theta/r)}{dr}\right)^n \tag{2.11}$$

where $\dot{\gamma}$ [s^{-1}] is the shear rate, K [Pa·s^n] is the consistency index and n is the flow behaviour index. Using Equations 2.10 and 2.11 the shear rate can be calculated [22]

$$\dot{\gamma} = \left(\frac{T}{2\pi KH}\right)^{(1/n)} \cdot \frac{1}{r^{(2/n)}} \tag{2.12}$$

After integrating Equation 2.12 and applying the boundary condition $v_{\theta_o} = 0$ at $r = R_o$ Equation 2.13 is found

$$v_\theta = \frac{rn}{2}\left(\frac{T_i}{2\pi KH}\right)^{(1/n)}\left(\frac{1}{R_o^{(2/n)}} - \frac{1}{r^{(2/n)}}\right) \tag{2.13}$$

Applying the boundary condition $v_{\theta_i} = \Omega R_i$ at $r = R_i$ yields Equation 2.14

$$\frac{2\Omega}{n\left[\frac{1}{R_o^{(2/n)}} - \frac{1}{R_i^{(2/n)}}\right]} = \left(\frac{T}{2\pi KH}\right)^{(1/n)} \tag{2.14}$$

Substituting the left hand side of Equation 2.14 into Equation 2.12 results in Equation 2.15, which results in the relation between the rotational speed of the inner cylinder and the shear rate profile across the Couette cell gap

$$\dot{\gamma} = \frac{2\Omega}{n\left[\frac{1}{R_o^{(2/n)}} - \frac{1}{R_i^{(2/n)}}\right]} \cdot \frac{1}{r^{(2/n)}} \tag{2.15}$$

Substituting the left hand side of Equation 2.14 into Equation 2.13 results in the relation between rotational speed and the azimuthal velocity

$$v_\theta = \frac{\Omega r}{\left[\frac{1}{R_o^{(2/n)}} - \frac{1}{R_i^{(2/n)}}\right]}\left(\frac{1}{R_o^{(2/n)}} - \frac{1}{r^{(2/n)}}\right) \tag{2.16}$$

2.5. OpenFOAM solver

In order to model the process in OpenFOAM it is important to use the right solver. OpenFOAM offers a wide range of different solvers for many different purposes. Each of the solvers offers a unique combination of problems that it is able to handle. For OpenFOAM's 2012 version, which contained 23 standard solvers, a capability matrix has been produced which gives a clear overview of the capabilities for each of the standard solvers that came with OpenFOAM, as can be seen in Figure 2.8. Alas such a capability matrix is not available for the newest OpenFOAM version from June 2020, which already comes with 48 standard solvers. Apart from the standard solvers there are also many more custom made variants. It is possible to use the already existing customized solvers, but it is important to remember that OpenFOAM is open source software and every 1 or 2 years a newer version is published. Over the years many customized solvers have been created which very often are no longer fully functional with newer OpenFOAM versions. So trying to use them can cause a lot of unnecessary complications. Therefore the choice has been made to stick with the standard solvers which are supported in all OpenFOAM versions. To find out which solver meets the requirements it is important to determine the assumptions for the model. For each of the capabilities a short explanation will be given on its importance:

- **Transient** - The process to be modelled is a transient one, so this is a requirement.
- **Compressible** - The density of the ingredient mixture is roughly the same as that of water. In fluid calculations the assumption is often made that water is incompressible and thus it is deemed reasonable to make the same assumption for this system.
- **Viscosity model** - The SPI and WG mixture shows non-Newtonian behaviour and therefore the solver should be able to handle non-Newtonian viscosity models. In this case the non-Newtonian behaviour seems to follow a temperature dependent power-Law model, and therefore it would be ideal if the solver is able to handle this.
- **Turbulence** - The maximum velocity inside the dough mixture occurs near the rotating inner cylinder. If the inner cylinder rotates at 30 RPM and has a radius of 0.159m then the velocity at the surface where the dough and cylinder contact each other will be 0.5 m/s. In a study by van Dijk [13] it has been shown that the Reynolds number stays below 0.02 for the case where $R_i = 0.095$m and the rotational speed is 30 RPM. In this thesis similar dimensions and process conditions apply and thus the laminar flow can be safely assumed.
- **Heat-transfer** - The simulation of the temperature distribution inside the ingredient mixture during processing is a requirement and thus this is necessary. It is important to note that it is possible to add the temperature equations to solvers which do not take into account heat transfer. This would require customizing the solver, for which tutorials are available.
- **Buoyancy** - Constant density will be assumed in this model and thus buoyancy effects can be neglected.
- **Combustion** - No combustion will take place in this process.
- **Multiphase** - The mixture is assumed to be a homogeneous fluid and thus multiphase calculations are not required.
- **Particles** - No particle calculations will be required in this process, because the dough mixture will be treated as a homogeneous fluid, instead of looking at every single particle alone, which is not possible yet due to several reasons. First of all because the mechanics behind the interaction between the SPI and WG mixture are not yet fully understood and secondly because this would require an enormous amount of computational power.
- **Dynamic mesh** - This is not required, since for this model the velocity can be implemented via a "rotatingWallVelocity" boundary condition, which will be explained later on.
- **Multi-region** - Can be useful if the cylinder walls and possibly the heat transfer medium outside of the cylinder walls are to be simulated in the model as well.
- **fvOptions** - This offers a collection of run-time selectable finite volume options to manipulate systems of equations by adding sources/sinks, imposing constraints and applying corrections. This could be useful, but is not required.

Capability matrix

Solver	transient	compressible	turbulence	heat-transfer	buoyancy	combustion	multiphase	particles	dynamic mesh	multi-region	fv Options
boundaryFoam											
buoyantPimpleFoam	✓	✓	✓	✓	✓						✓
buoyantSimpleFoam		✓	✓	✓	✓						✓
chemFoam	✓			✓		✓					
chtMultiRegionFoam	✓	✓	✓	✓	✓					✓	✓
coldEngineFoam	✓	✓	✓	✓		✓			✓		✓
engineFoam	✓	✓	✓	✓		✓			✓		✓
fireFoam	✓	✓	✓	✓	✓	✓				✓	✓
icoFoam	✓										
interFoam	✓		✓				✓		✓		✓
laplacianFoam	✓										✓
pimpleFoam	✓		✓						✓		✓
pisoFoam	✓		✓								✓
potentialFoam											
reactingFoam	✓	✓	✓	✓		✓					✓
reactingParcelFoam	✓	✓	✓	✓	✓	✓		✓			✓
rhoCentralFoam	✓	✓	✓	✓							
rhoPimpleFoam	✓	✓	✓	✓					✓		✓
rhoSimpleFoam		✓	✓	✓							✓
scalarTransportFoam	✓										
simpleFoam			✓								✓
sprayFoam	✓	✓	✓	✓	✓	✓		✓			✓
XiFoam	✓	✓	✓	✓		✓					✓

Figure 2.8: Capability matrix from OpenFOAMs userguide v2012 [23]

Using the aforementioned required capabilities, a suitable solver can be selected. For this project nonNewtonianIcoFoam was chosen to start with. This is a transient solver for incompressible, laminar flow of non-Newtonian fluids. As implied by the name, this solver is the same as the regular icoFoam solver apart from the fact that it can handle non-Newtonian fluids as well.

IcoFoam (or nonNewtonianIcoFoam to be precise) is a solver for which quite a lot of customization tutorials exist, allowing for example the addition of the temperature equations. For solvers with more capabilities it is often more complicated to customize the solver, because much more has to be altered, making it a more tedious process.

3

Temperature dependency of the Power-Law parameters

In previous research the power-law parameters have been determined for several of the mixture compositions used by Rival-Foods. All the measurements on these mixtures were performed at room temperature, while the real process takes place in the temperature range from roughly 20 degrees Celsius to 130 degrees Celsius. In the study by Van Dijk it was stated that experience with the Couette cell suggested the viscous properties of the mixture were highly influenced by the temperature, which was not taken into account in that study. Therefore the model can be improved if the temperature dependency of the power-law model can be added to the model. It is important to note that the research by Van Dijk has been carried out in 2014. Since then the ingredient mixture and production process has been optimised, which results in different product properties.

3.1. Plan of approach

The method to determine the values of K and n has been shown in Section 2.3. Using the same method a new plan of approach can be made on how to determine the temperature dependency of the power-law coefficients.

The temperature range of interest is determined by the temperatures that the mixture will go through during processing, which is 20-130°C. In order to find the temperature dependency the measurements need to be performed at several temperatures. In this case the choice was made to take measurements at the following 5 discrete intervals, 25/50/75/100/125°C.

The frequency range and the amount of measurement points that have been taken during testing need to be determined as well. The frequency range used is 0.1-10 Hz instead of 0.1-100 Hz, because the RPA Elite rheometer has a maximum frequency of 50 Hz. A total of 10 measurement points have been used in the range from 0.1-10 Hz. During Van Dijk's research it was necessary to have more detailed data in order to determine how the fluid behaved. In the current case the material is already expected to behave as a power-law fluid and the tests are performed to find out how the graph shifts (change in K) and how it's angle changes (change in n) under the influence of temperature. Due to corona restrictions no measurements could be performed at the TU Delft. Therefore the measurements were performed by a member of the Rival Foods team, Matthijs van Alfen, who had to go to Wageningen University in order to perform the measurements.

The strain percentage used in Van Dijk's experiments has not been reported and therefore cannot be reproduced. In the new tests a small strain percentage of 0.977% has been used, which falls in the linear viscoelastic region (LVR) as shown by Dekkers [2]. This means that the tests are Small Amplitude Oscillatory Shear (SAOS).

In previous research by Dekkers [2] and Schreuders [24] time sweeps have been performed on the mixture at different temperatures to find how the apparent (G') or complex modulus (G*) changed over time. Their

measurements have been performed on a mixture consisting of 20 wt% SPI and 20 wt% WG instead of the 15 wt% SPI and 15 wt% WG that is used in the mixture nowadays, but it should still give a decent indication of the time dependency of similar mixtures. The measurements in Dekkers' research showed that at 95°C after the first 5 minutes the apparent modulus stayed roughly constant and similar results have been obtained by Schreuders. In Figure 3.1 the solid green line shows the behaviour of a 40 wt.% SPI-WG mixture (50-50). It can be seen that after 5 minutes this line stays roughly horizontal, indicating that the apparent G' has stabilized.

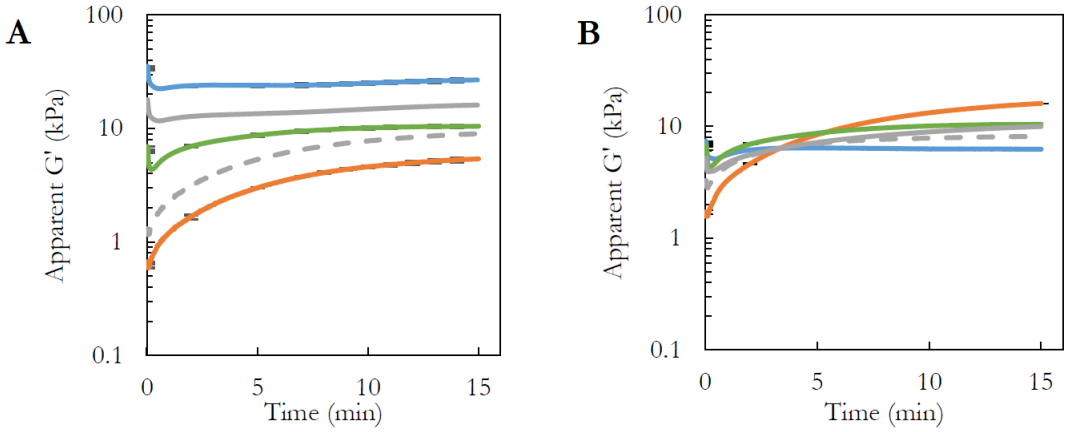


Figure 3.1: From Dekkers' thesis [2] *"Apparent storage modulus (G') measured during a time sweep at high strain (80%) and high frequency (10 Hz) and the apparent storage modulus (G') calculated based on the isostrain (gray solid line) and isostress (gray dashed line) models: A) 40 wt.% SPI (blue line), 40 wt.% WG (orange line), and 40 wt.% SPI-WG (50-50) (green line), calculated 40 wt.% SPI-WG (50-50) based on mass fraction (50-50) with power law. B) 33 wt.% SPI (blue line), 50 wt.% WG (orange line), and 40 wt.% SPI-WG (50-50) (green line), calculated 40 wt.% SPI-WG (50-50) based on volume fraction (0.62-0.38) with power laws (gray solid line is isostrain and gray dashed line is isostress), standard deviation is plotted every min."*

This means the behaviour of the mixture is not only shear and temperature dependent, but also time dependent. This makes the measurements on the mixture more complex, due to the fact that there is intricate coupling of the process variables. With the machinery available it is impossible to completely decouple all these process variables in order to measure them independently. Therefore it seems best to first let the mixture sample settle for 5 minutes at the temperature at which the measurements are going to be performed. The 5 minute waiting time at elevated temperature can be performed statically or under oscillatory shear, where the latter option is expected to give the most similar results when compared to the real steady shear process, where deformation of the dough also occurs during heating. Therefore the choice was made to use this approach.

The measurements have been done using the RPA Elite rubber process analyzer. It is important to note that these tests have been performed on a mixture which is not exactly the same as before. The ratio of the components has changed from 23% to 15% SPI, from 7% WG to 15% and from 1% to 0.5% salt. The SPC mixture mentioned in Van Dijk's report is no longer in use by Rival Foods due to the fact that the SPI mixture results in a more structured final product.

3.2. Data processing

The data from the SAOS measurements consisted of five data sets, one set for each temperature interval. At each temperature two measurements were performed. The data was then processed as described in Section 2.3. During the processing some issues were encountered at 100°C and 125°C which will be discussed later on. In the following 5 subsections the produced data will be shown per temperature interval.

3.2.1. Data at 25°C

All the raw data from the measurements with the RPA Elite Rheometer can be found in Appendix A. This data has been used to calculate the average, the standard deviation, and the standard error of the mean for the elastic modulus G', the viscous modulus $G"$, and the complex viscosity η^*. The average G' and $G"$ together with their standard error of the mean (SEM) error bars have been plotted against the frequency in Figure 3.2. It can be seen that the elastic modulus is roughly 2 to 3 times higher than the viscous modulus over the entire frequency range. It is important to note that the error bars might give a distorted view of the actual errors because the vertical axis has a logarithmic scale. Therefore it is useful to use the raw data in the appendices as well.

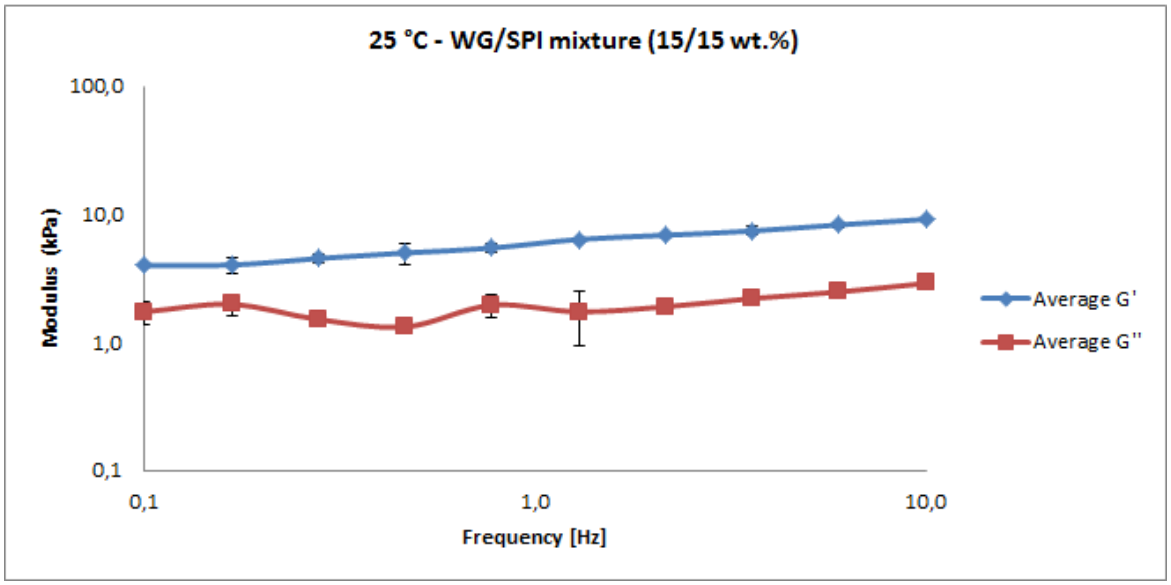


Figure 3.2: Graph of the elastic and viscous modulus vs frequency at 25°C

Using these values of G' and $G"$ the complex viscosity has been calculated according to Equation 2.6. In Figure 3.3 the complex viscosity data is shown in green. The black line is a Power-Law fit to the data. The Power-Law parameters can be found by setting the equation for the Power-Law fit in Figure 3.3 equal to Equation 2.9, which results in $K = 999.50[Pa \cdot s^n]$ and $n = 0.180$.

In Figure 3.3 the value R^2 is also shown. This is the squared Pearson correlation coefficient. The Pearson coefficient is calculated according to Equation 3.1. A value of 1 indicates perfect alignment between the two sets of data while a value of -1 would mean the opposite. On a log-log scale this coefficient can be used to suggest that the data follows the Power-Law relationship, but only if a high value of R^2 is found. A low value for R^2 would suggest the data follows a different relationship or contains random noise. In this case the problem is that the log-log correlation is a necessary but not sufficient condition to actually prove a Power-Law relationship. Based on the values found for R^2 in this section, the assumption that the Power-Law relationship is valid does not have to be rejected.

$$|r| = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 (y_i - \overline{y})^2}} \tag{3.1}$$

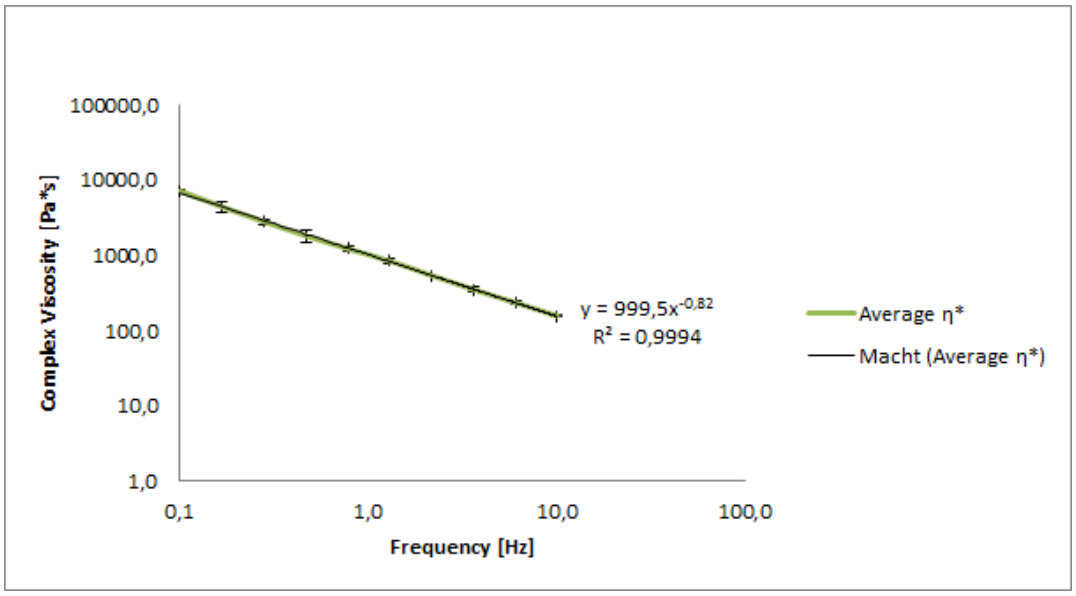


Figure 3.3: Graph of the complex viscosity η^* vs frequency at 25°C

3.2.2. Data at 50°C

The Power-Law parameters calculated at 50°C are $K = 599.95[Pa \cdot s^n]$ and $n = 0.198$.

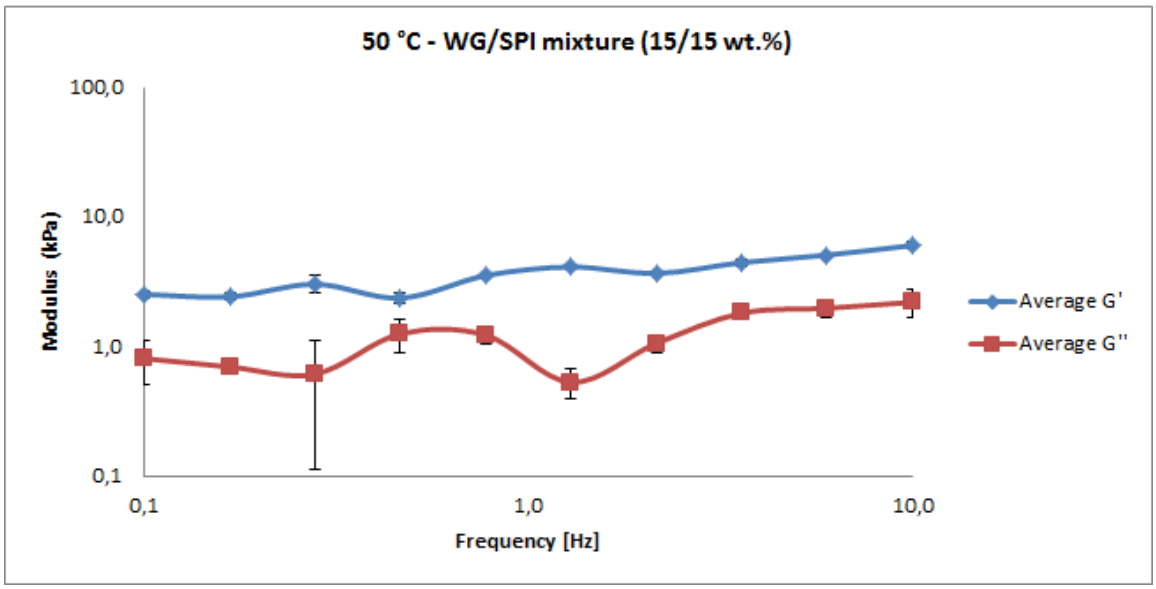


Figure 3.4: Graph of the elastic and viscous modulus vs frequency at 50°C

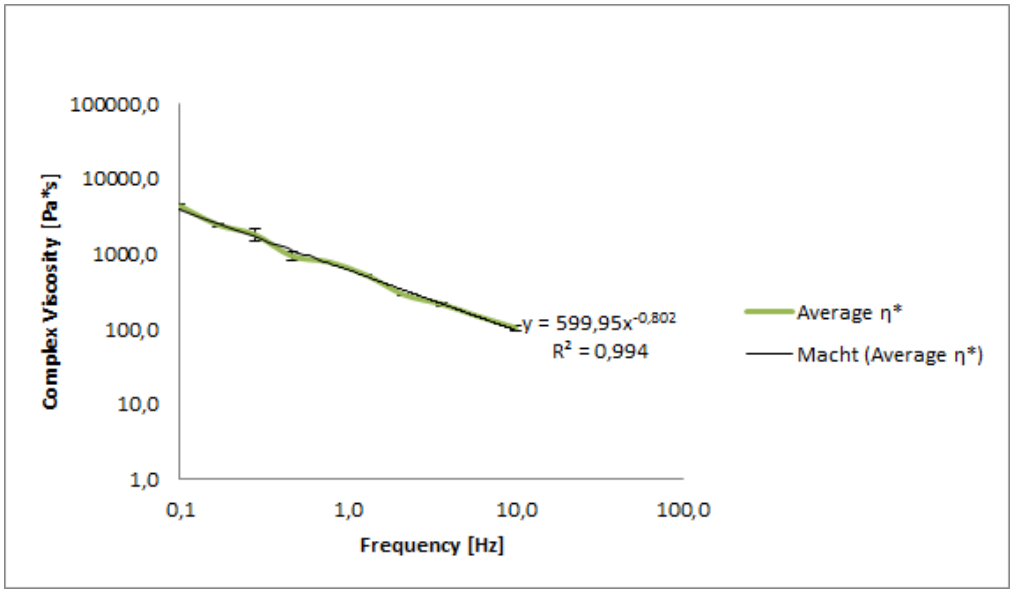


Figure 3.5: Graph of the complex viscosity η^* vs frequency at 50°C

3.2.3. Data at 75°C

The Power-Law parameters calculated at 75°C are $K = 653.20[Pa \cdot s^n]$ and $n = 0.089$.

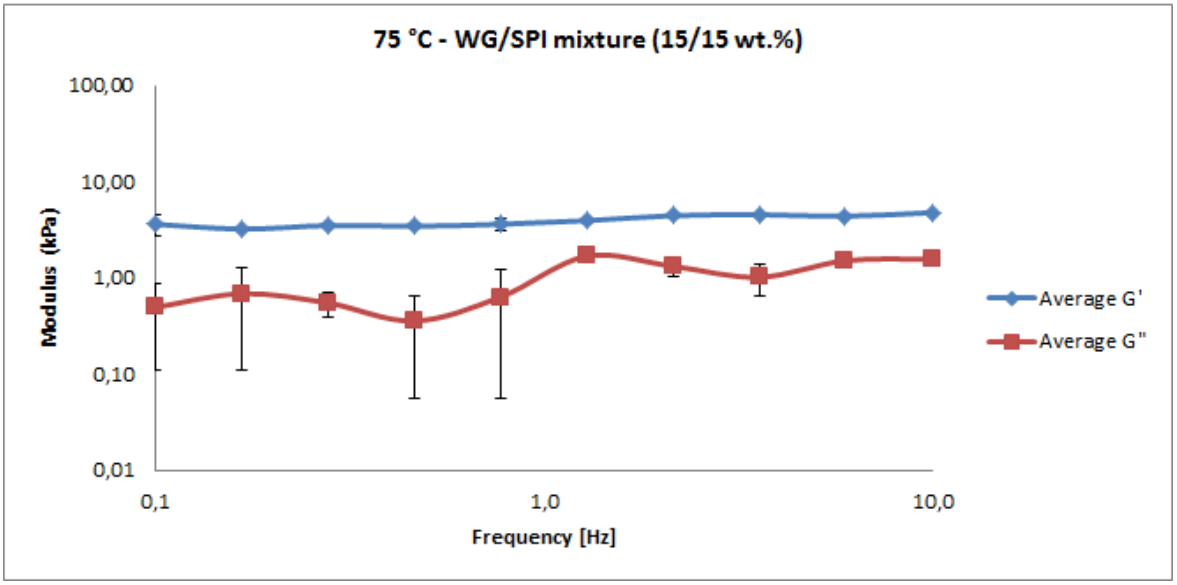


Figure 3.6: Graph of the elastic and viscous modulus vs frequency at 75°C

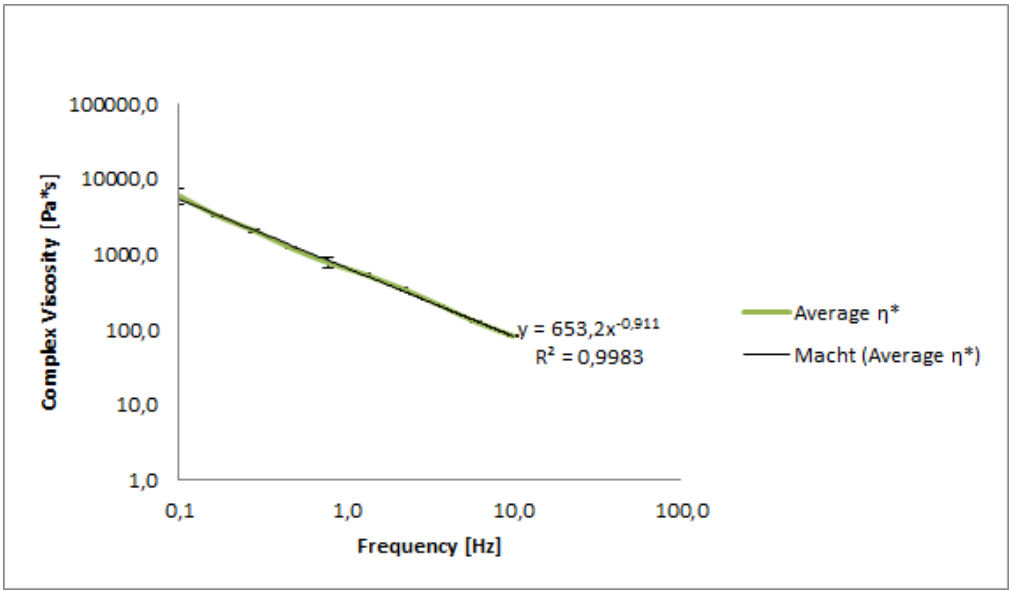


Figure 3.7: Graph of the complex viscosity η^* vs frequency at 75°C

3.2.4. Data at 100°C

The results at 100°C were anomalous and therefore require a more detailed description. Originally, when only 2 measurements were performed, the Power-Law parameters calculated at 100°C were $K = 1280.60[Pa \cdot s^n]$ and $n = -0.01$. A value of $n = 0$ would mean the flow is infinitely shear thinning and therefore a negative value for n is physically impossible, meaning there has to be something wrong with this result. As a first check an additional measurement was performed to find out if a similar result would be produced. Combining the data from the 3 measurements the new Power-Law parameters were $K = 1495.60[Pa \cdot s^n]$ and $n = -0.005$, thus the value of n was still non-physical. The Power-Law parameters have also been calculated based on each measurement on itself. The first measurement would result in $K = 1233.00[Pa \cdot s^n]$ and $n = 0.005$, the second in $K = 1325.30[Pa \cdot s^n]$ and $n = -0.024$, and the third in $K = 1924.20[Pa \cdot s^n]$ and $n = 0.003$. The values for 2 out of 3 of these measurements could be physically possible, but it is important to realise that such low shear thinning indices are still unlikely to be true. A more detailed discussion on why these values could have been found will be provided in Section 3.3. For the OpenFOAM model the values $K = 1495.60$ and $n = 0.005$ were used. Here a small positive value of n was used in order to keep the OpenFOAM simulation from crashing due to infinitely shear thinning behaviour.

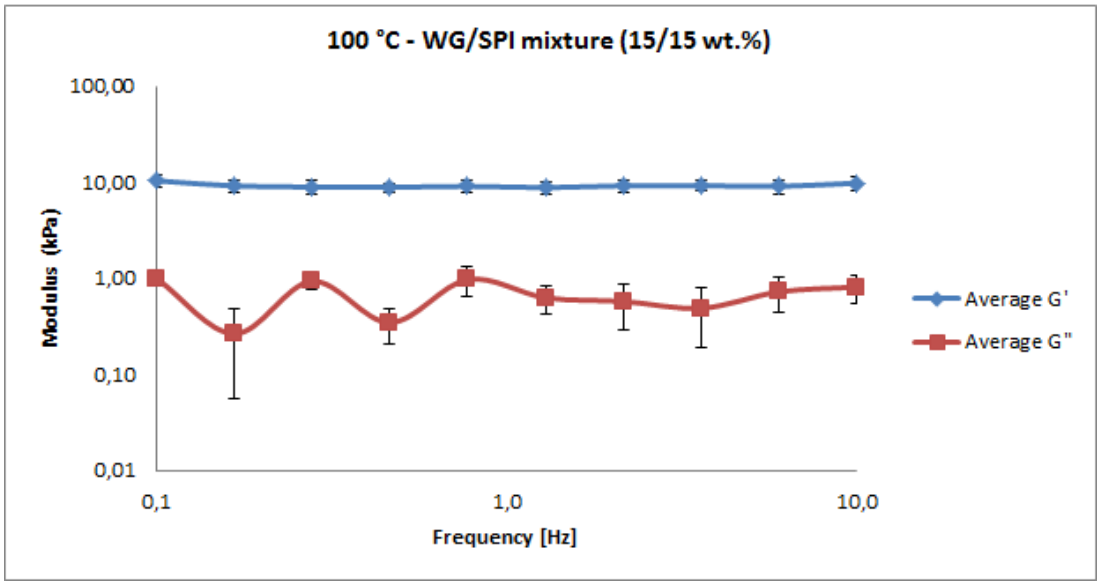


Figure 3.8: Graph of the elastic and viscous modulus vs frequency at 100°C

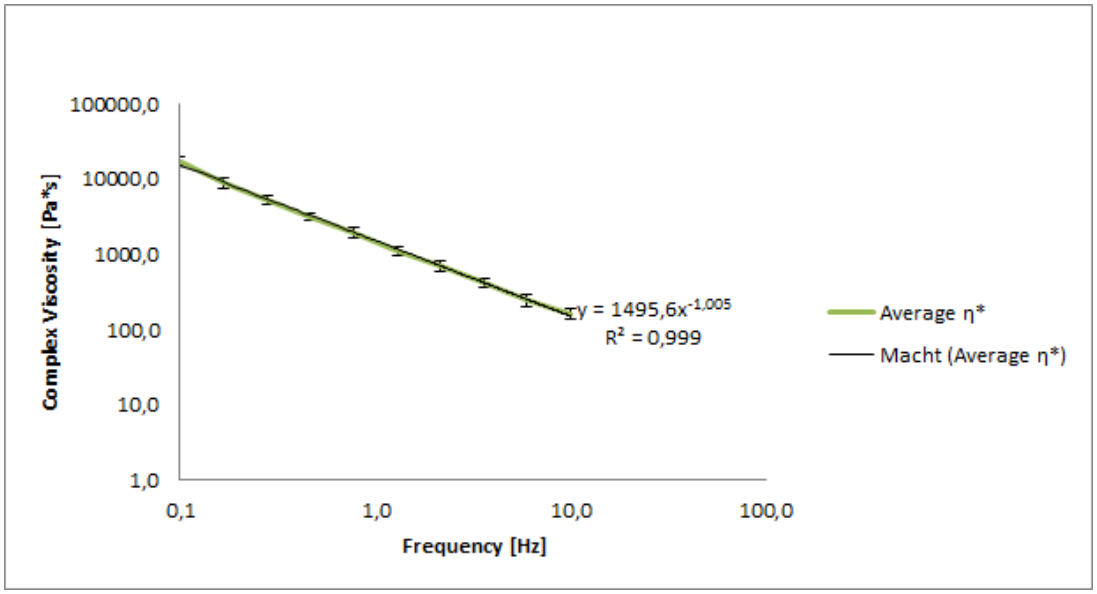


Figure 3.9: Graph of the complex viscosity η^* vs frequency at 100°C

3.2.5. Data at 125°C

At 125°C the standard error of the mean (SEM) calculated for the complex viscosity was often as big as 50% to 60% of the average value. Because the variation between the data sets was this big, an additional measurement was performed. The complex viscosity calculated from the data at 125°C for the three measurements is shown in Figure 3.10 to give the reader some idea of the variation. Apart from the complex viscosity, the raw data for the elastic and viscous moduli, G' and G", can also be studied to gain more insight in these large variations. In Figure 3.11 the raw data is shown that is used to calculate the average G' and G" which are shown in Figure 3.12. In theory more measurements could be performed to reduce the SEM, but this was no longer achievable in the current project. The average Power-Law parameters calculated at 125°C are $K = 691.09[Pa \cdot s^n]$ and $n = 0.084$.

Frequency [Hz]	η* [Pa*s] - Test #1	η* [Pa*s] - Test #2	η* [Pa*s] - Test #3
10,000	80,841	27,742	111,627
5,988	153,271	86,656	210,119
3,597	203,826	74,022	345,688
2,155	371,465	134,900	508,611
1,292	659,667	226,803	756,130
0,774	1077,978	324,747	1396,076
0,464	1797,186	564,869	1850,814
0,278	2759,006	999,096	3902,787
0,167	4393,197	978,233	5536,570
0,100	7267,653	3285,149	3904,066

Figure 3.10: Comparison of the complex viscosities calculated for the data from measurement #1, #2, and #3

Frequency [Hz]	G' [kPa] - Test #1	G' [kPa] - Test #2	G' [kPa] - Test #3
10,000	5,019	1,729	6,831
5,988	5,688	3,179	7,786
3,597	4,573	1,004	7,778
2,155	5,019	1,785	6,867
1,292	5,298	1,840	6,132
0,774	5,242	1,171	6,607
0,464	5,242	1,506	5,391
0,278	4,740	1,673	6,662
0,167	4,406	0,948	5,411
0,100	4,015	2,063	1,608
Frequency [Hz]	**G'' [kPa] - Test #1**	**G'' [kPa] - Test #2**	**G'' [kPa] - Test #3**
10,000	0,781	0,223	1,591
5,988	0,948	0,725	1,367
3,597	0,558	1,338	0,739
2,155	0,335	0,390	0,514
1,292	0,781	0,056	0,277
0,774	0,056	1,060	1,563
0,464	0,056	0,669	0,231
0,278	0,892	0,502	1,444
0,167	1,338	0,390	2,114
0,100	2,175	0,056	1,853

Figure 3.11: Comparison of the elastic and viscous moduli for the data from measurement #1, #2, and #3

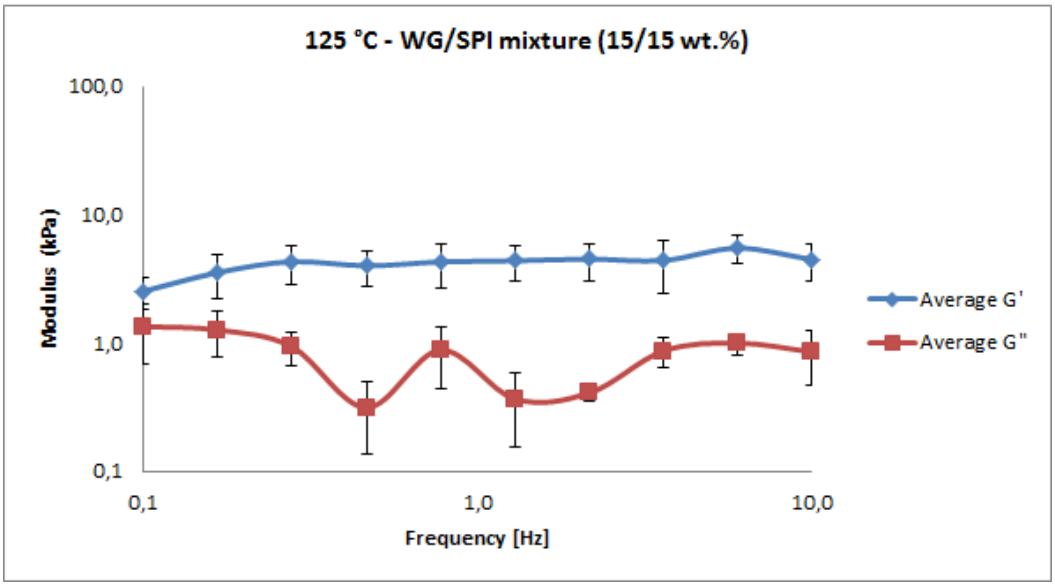


Figure 3.12: Graph of the elastic and viscous modulus vs frequency at 125°C

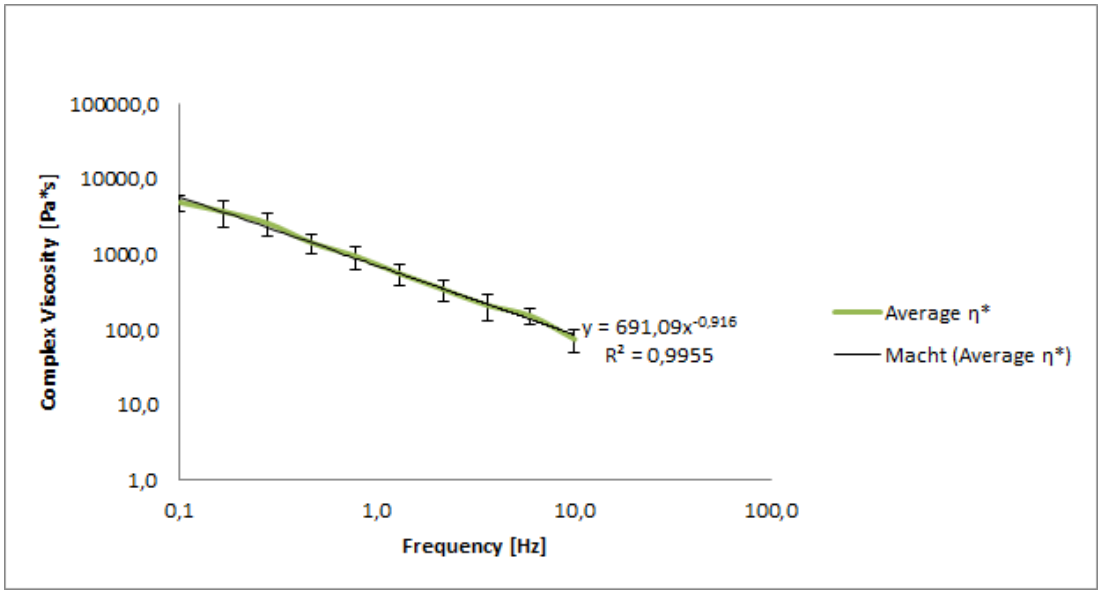


Figure 3.13: Graph of the complex viscosity η^* vs frequency at 125°C

3.3. Results for the temperature dependency of the Power-Law parameters

The values for K and n that have been calculated are plotted in Figure 3.14 and Figure 3.15 to give a clear overview of how they vary with temperature. The error bars included show the SEM.

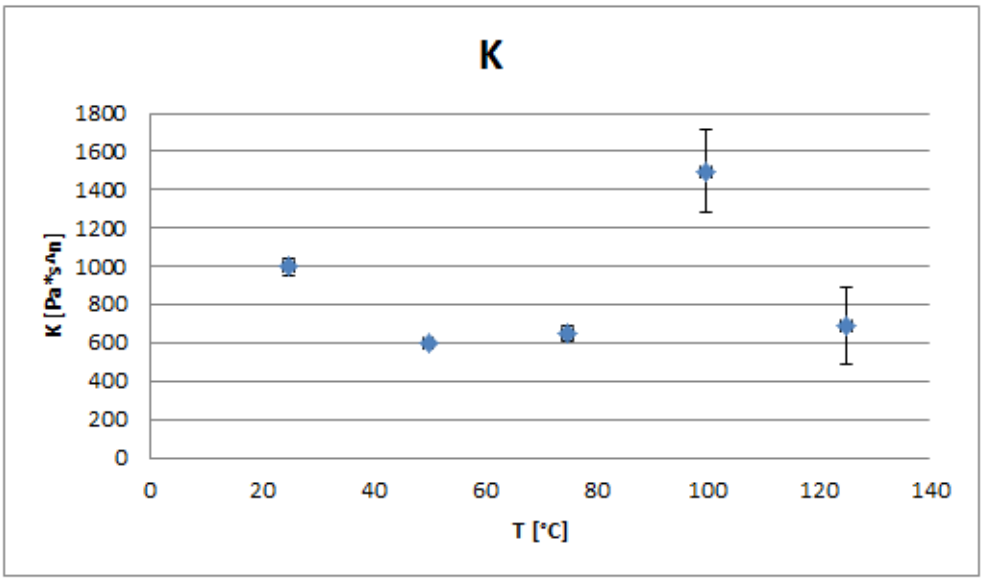


Figure 3.14: Graph of the temperature dependency of K, where the error bars show the SEM.

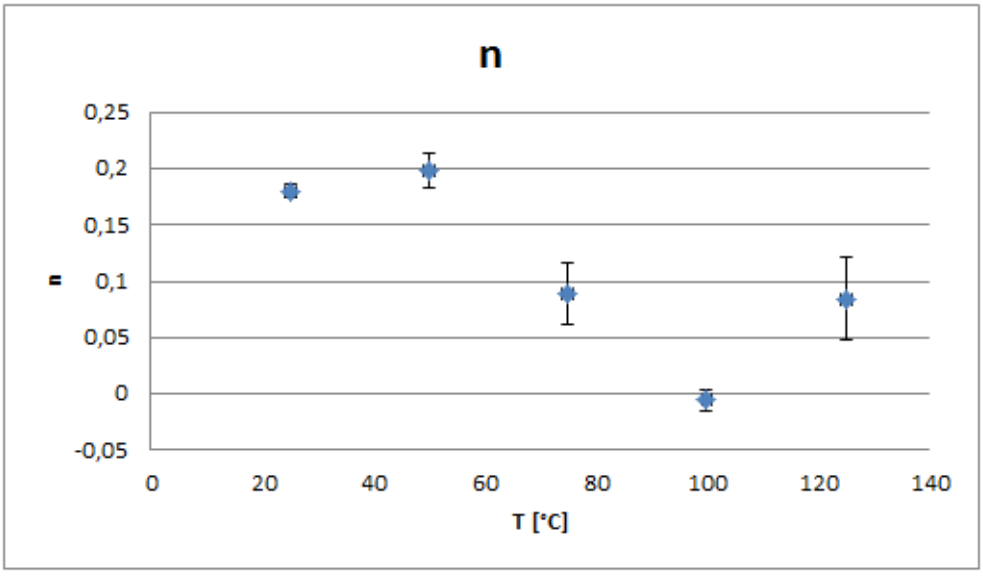


Figure 3.15: Graph of the temperature dependency of n, where the error bars show the SEM.

The large variations found in the data for the elastic and viscous moduli could be caused by a low precision of the measurement equipment in the specific measurement range. The torque range for the RPA Elite Rheometer is 0.0001 to 25 Nm. In the raw measurement data, the measured torque values sometimes reach values of 0.0001 or 0.0000 Nm, which is at this limit. The RPA Elite uses the measured torque in combination with shape factors to calculate the G' and G", viscosities, and shear stresses etc. Therefore

if the torque is being measured at the lowest end of its range it is possible that the resulting G' and G" values change in a choppy manner due to the resolution of the equipment.

In the process of calculating the Power-Law parameters K and n for the SPI and WG mixture, the assumption is made that the Cox-Merz rule holds. This means the complex viscosity calculated from the oscillatory measurements can be set equal to the dynamic viscosity. This assumption has also been used in the studies by Van Dijk and Krintiras. However, this assumption could be faulty, which can be tested with the right equipment. This equipment should be able to perform steady shear torque measurements on a sample at the specific constant temperatures. It is also important to note that at lower frequencies the influence of the resolution for the measured torque values gets bigger. Revisiting the equation for the complex viscosity, see Equation 3.2, it is clear that the smaller the value of ω becomes, the larger the influence of the resolution of the measured torque (used to calculate G'and G") becomes. Since the frequency varies from 0.1 to 10 Hz this means that the resolution errors in measuring the torque at the lower frequencies get magnified up to a 100 times in comparison to the errors at the highest frequencies.

$$|\eta^*| = \sqrt{\left(\frac{G'}{\omega}\right)^2 + \left(\frac{G''}{\omega}\right)^2} \tag{3.2}$$

The curve fitting of the Power-Law equation to the complex viscosity plotted on a log-log scale in Excel uses the least squares method. This method minimises the sum of the squares of the offsets of the points from the curve. It is important to note that the highest values of the complex viscosities on the log-log scale are attained at the lowest frequencies. This means that the Power-Law curve fit in Excel prioritises perfectly fitting the lower frequencies over the higher frequencies. Remember that the lowest frequencies contained the biggest influence of resolution errors. Therefore the reliabability of this curve fit could be compromised. This is expected to be partly responsible for the nonphysical negative value of the Power-Law parameter n that was encountered after fitting the data at 100°C, and of course this also introduces errors in the other curve fits.

Apart from the nonphysical value of n and the large variations between the measurement data, the way that K and n vary with the temperature might seem strange. It is important to realise that multiple processes and interactions take place at the microscopic scale under the influence of temperature and time which have been researched by Dekkers and are still being researched at Wageningen University and Research.

In the SPI-WG mixture, the soy proteins are denatured before it enters the mixture, but the vital wheat gluten go through a polymerisation and de-polymerisation process as shown by Emin [25]

In Figure 3.16 the change in complex modulus as a function of increasing temperature can be seen. All three water contents show the same behaviour where the complex modulus initially shows a decrease, then an increase, and then a final decrease. Emin's study stated that the initial decrease can be directly related to the higher molecular mobility due to increased temperature. The increase that follows is caused by the polymerisation/aggregation reactions of glutenin and gliadin molecules, which results in a cross-linked network structure. The final decrease is caused by the de-polymerisation.

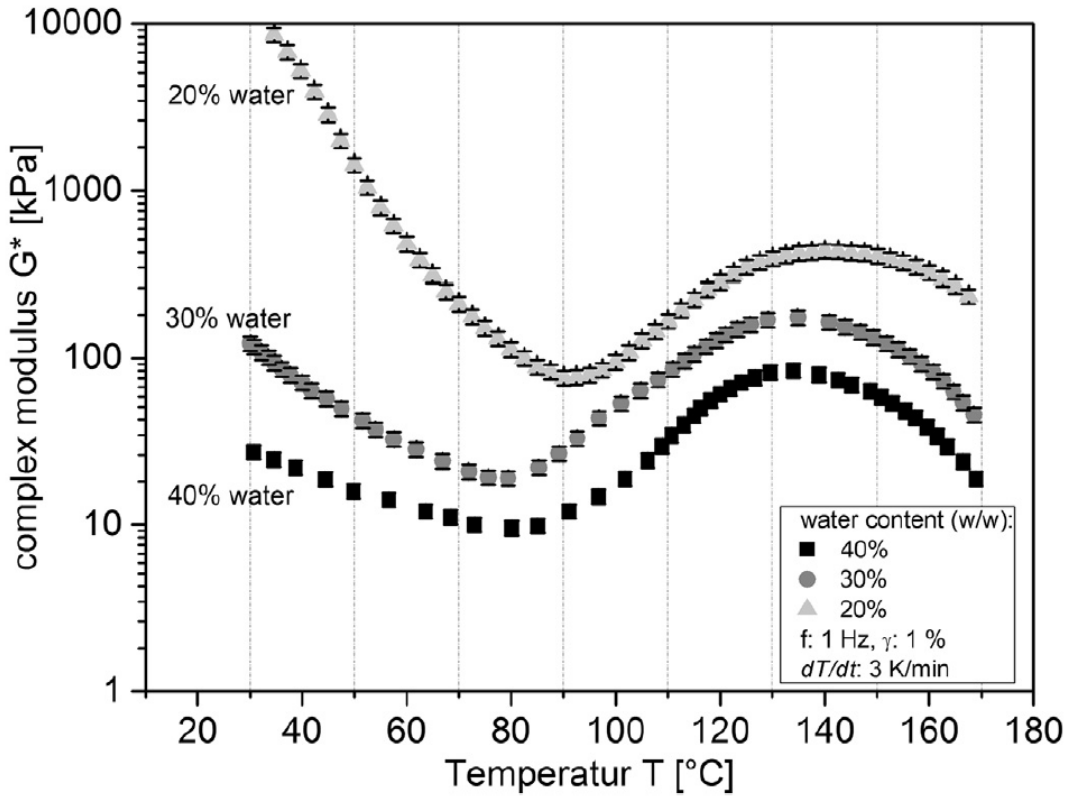


Figure 3.16: Change in the complex modulus (G*) as a function of increasing temperature at various water contents of 20%, 30%, 40%. The temperature was raised with the rate of 3 K/min. The frequency and strain were kept constant at 1Hz and 1% respectively [25].

4

OpenFOAM Simulation Setup

The process of setting up the OpenFOAM case will be described in this chapter. Section 4.1 will describe the equations used in the OpenFOAM model. Section 4.2 explains how the solver has been customized. Section 4.3 describes the process of customizing the power-law viscosity model. Section 4.4 describes how the directory structure is created and organized. Section 4.5 will describe the geometry used. Section 4.6 contains the boundary conditions. Section 4.7 will describe the numerical schemes used in the simulations, and finally Section 4.8 will explain which different cases have been simulated.

This chapter will include many short bits of code accompanied by explanations, which together should create a clear overview of how the OpenFOAM code has been customized. The actual code might not be of interest to everyone, but the inclusion of the code with the explanations alongside it might be of help for future students who consider working with OpenFOAM. Simply adding the code as an Appendix with a small description of the functionalities will not be as useful.

4.1. nonNewtonianIcoFoam equations

As previously mentioned nonNewtonianIcoFoam is a solver that is able to handle transient, incompressible, laminar problems for non-Newtonian fluids. It is not able to handle problems that involve heat transfer, buoyancy, combustion, multi-phase flow, particles, and multi-region geometries [23]. Since heat transfer is an integral part of this simulation the nonNewtonianIcoFoam solver has been customized, as will be shown in Section 4.2.

IcoFoam solves the incompressible continuity equation (Eq 4.1) and incompressible laminar Navier-Stokes equations (Eq 4.2) for the case of constant viscosity.
NonNewtonianIcoFoam solves the same continuity equation, but an extra term has to be included in the momentum equations which results in Eq 4.3. In equation 4.2 and 4.3 u is the velocity $[\frac{m}{s}]$, ν is the kinematic viscosity $[\frac{m^2}{s}]$, and p is the kinematic pressure $\frac{m^2}{s^2}$, which is the pressure divided by the density. The symbol $\otimes$ denotes the tensorial product.

$$\vec{\nabla} \cdot (\vec{u}) = 0 \tag{4.1}$$

$$\frac{\partial \vec{u}}{\partial t} + \vec{\nabla} \cdot (\vec{u} \otimes \vec{u}) - \vec{\nabla} \cdot (\nu \vec{\nabla} \vec{u}) = -\vec{\nabla} p \tag{4.2}$$

$$\frac{\partial \vec{u}}{\partial t} + \vec{\nabla} \cdot (\vec{u} \otimes \vec{u}) - \vec{\nabla} \cdot (\nu \vec{\nabla} \vec{u}) - \vec{\nabla} \vec{u} \cdot \vec{\nabla} \nu = -\vec{\nabla} p \tag{4.3}$$

The equations above are given in vector form but all the components can also be fully written out, resulting in Equation 4.4, 4.5, 4.6, and 4.7.

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{4.4}$$

$$\frac{\partial u}{\partial t} + \frac{\partial uu}{\partial x} + \frac{\partial vu}{\partial y} + \frac{\partial wu}{\partial z} - \frac{\partial}{\partial x}\nu\frac{\partial u}{\partial x} - \frac{\partial}{\partial y}\nu\frac{\partial u}{\partial y} - \frac{\partial}{\partial z}\nu\frac{\partial u}{\partial z} - \frac{\partial v}{\partial x}\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\frac{\partial v}{\partial x} - \frac{\partial v}{\partial z}\frac{\partial w}{\partial x} = -\frac{\partial p}{\partial x} \tag{4.5}$$

$$\frac{\partial v}{\partial t} + \frac{\partial uv}{\partial x} + \frac{\partial vv}{\partial y} + \frac{\partial wv}{\partial z} - \frac{\partial}{\partial x}\nu\frac{\partial v}{\partial x} - \frac{\partial}{\partial y}\nu\frac{\partial v}{\partial y} - \frac{\partial}{\partial z}\nu\frac{\partial v}{\partial z} - \frac{\partial v}{\partial x}\frac{\partial u}{\partial y} - \frac{\partial v}{\partial y}\frac{\partial v}{\partial y} - \frac{\partial v}{\partial z}\frac{\partial w}{\partial y} = -\frac{\partial p}{\partial y} \tag{4.6}$$

$$\frac{\partial w}{\partial t} + \frac{\partial uw}{\partial x} + \frac{\partial vw}{\partial y} + \frac{\partial ww}{\partial z} - \frac{\partial}{\partial x}\nu\frac{\partial w}{\partial x} - \frac{\partial}{\partial y}\nu\frac{\partial w}{\partial y} - \frac{\partial}{\partial z}\nu\frac{\partial w}{\partial z} - \frac{\partial v}{\partial x}\frac{\partial u}{\partial z} - \frac{\partial v}{\partial y}\frac{\partial v}{\partial z} - \frac{\partial v}{\partial z}\frac{\partial w}{\partial z} = -\frac{\partial p}{\partial z} \tag{4.7}$$

When using nonNewtonianIcoFoam, the continuity and momentum equations are solved using the PISO algorithm, which stands for Pressure-Implicit with Splitting of Operators. This algorithm was first proposed by Issa in 1986. Since the original publishment of the PISO algorithm, improvements have been made and the notation and form of the PISO algorithm that is currently used in nonNewtonianIcoFoam is most similar to the algorithm described in Jasak's [26] and Rusche's [27] PhD theses.

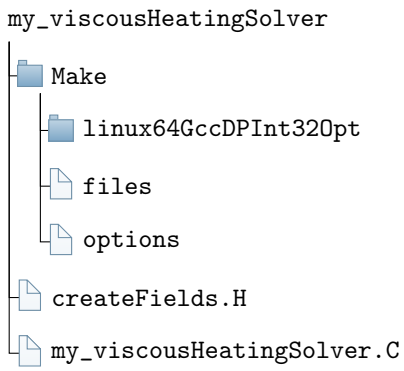
In short the algorithm consists of the following steps [23]:

1. Set the boundary conditions.
2. Solve the discretized momentum equation to compute an intermediate velocity field.
3. Compute the mass fluxes at the cells faces.
4. Solve the pressure equation.
5. Correct the mass fluxes at the cell faces.
6. Correct the velocities on the basis of the new pressure field.
7. Update the boundary conditions.

8. Repeat from 3 for the prescribed number of times.
9. Increase the time step and repeat from 1.

4.2. nonNewtonianIcoFoam customization

This section describes how the nonNewtonianIcoFoam solver has been customized. A new solver, called "my_viscousHeatingSolver" has been created, which is based on a copy of the old nonNewtonianIcoFoam solver. The folder structure of the my_viscousHeatingSolver looks as follows:



The contents of the folder linux64GccDPInt320pt are automatically generated and therefore not shown, since they will not have to be created or modified.

4.2.1. Adding the temperature equation

The first improvement to the nonNewtonianIcoFoam solver is the addition of the temperature equation to the my_viscousHeatingSolver.C file. Assuming the material properties ρ, c_p, and k are constant, Equation 4.8 is found.

$$\rho c_p \frac{\partial(T)}{\partial t} + \rho c_p \nabla \cdot (uT) = \kappa \nabla^2 T \tag{4.8}$$

Dividing both sides of Equation 4.8 by ρc_p and taking the diffusion term to the left side results in Equation 4.9, where $\alpha = \frac{\kappa}{\rho c_p}$.

$$\frac{\partial(T)}{\partial t} + \nabla \cdot (uT) - \alpha \nabla^2 T = 0 \tag{4.9}$$

The my_viscousHeatingSolver folder contains two files:

- `createFields.H`
- `my_viscousHeatingSolver.C`

The temperature equation is added to the solver via the `nonNewtonianIcoFoam.C` file. The customised version of `nonNewtonianIcoFoam.C` has been renamed to `my_viscousHeatingSolver.C` and can be found in Appendix B.1. Because the transport of the temperature depends on the velocity, the temperature equation is inserted after the calculations for the momentum equations, so after the PISO loop, but before the time step is written in the code. The temperature equation in C++ code looks as shown in Listing 4.1

```
1    fvScalarMatrix TEqn
2        (
3            fvm::ddt(T)
4            + fvm::div(phi, T)
5            - fvm::laplacian(DT, T)
6        );
7
8    TEqn.solve();
```

Listing 4.1: The temperature equation code

Here the command "fvm::" is used, which will discretize the term into the matrix equation. The command "ddt(T)" indicates the use of the partial differential of the temperature to time. The phi used in the temperature equation is the velocity. The transport property DT used is called the thermal diffusivity. In heat transfer literature the thermal diffusivity is often denoted as α, see Equation 4.9. For the thermal diffusivity a dimensioned scalar is created in the `createFields.H` file, for which the value and dimensions will be read in from the transportProperties file as follows, see Listing 4.2.

```
dimensionedScalar DT
(
     transportProperties.lookup("DT")
);
```

Listing 4.2: Creation of the thermal diffusivity in the createFields.H file

The value of DT can be calculated as follows.

$$DT = \frac{\kappa}{\rho c_p} = \frac{0.44}{1020 * 3500} = 1.23 * 10^{-7} \frac{m^2}{s} \tag{4.10}$$

In the `transportProperties` file which can be found in the `constant` folder, the value of DT has been defined together with its dimensions, see Listing 4.3.

```
DT              DT [0 2 -1 0 0 0 0] 1.23e-7;
```

Listing 4.3: The thermal diffusivity value

On itself the icoFoam/nonNewtonianIcoFoam solvers only solve for the pressure and velocity fields. If the solver is required to calculate the temperature equation this means a temperature field has to be created. This temperature field has been declared in the `createFields.H` file, see Listing 4.4

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

Listing 4.4: Declaration of the temperature field

With the temperature field in place the next step is to create a new initial and boundary condition file in the 0-directory. Subsection 4.6.1 will provide a detailed description of this file.

After adding the temperature equation to the solver it is also necessary to tell OpenFOAM which numerical schemes should be applied to these equations. This is done in the `fvSchemes` file, which can be found under the `system`-directory. A divergence and Laplacian term have been added to the thermal transport equation and therefore a numerical scheme has to be defined for both of these. Listing 4.5 shows what the code looked like at first.

```
divSchemes
{
    default          none;
    div(phi,U)       Gauss linear;
```

```
}

laplacianSchemes
{
    default          Gauss linear orthogonal;
}
```

Listing 4.5: The original fvSchemes code

The code was then modified to the code shown in Listing 4.6. This code includes a term for the temperature under the divergence header in line 5. The lines under laplacianSchemes have also been modified to include the thermal diffusivity in line 13. The entire fvSchemes file can be found in Appendix B.4. Section 4.7 describes the reasoning behind selecting specific schemes.

```
divSchemes
{
    default          none;
    div(phi,U)       Gauss linear;
    div(phi,T)       Gauss vanLeer;
}

laplacianSchemes
{
    default          none;
    laplacian(nu,U) Gauss linear corrected;
    laplacian((1|A(U)),p) Gauss linear corrected;
    laplacian(DT,T) Gauss linear corrected;
}
```

Listing 4.6: The new fvSchemes code

Last but not least some information has to be specified in the `fvSolution` file, which controls the equations solvers, tolerances, and algorithms. For the temperature the following section is added, see Listing 4.7.

```
    T
    {
        solver            BICCG;
         preconditioner   DILU;
         tolerance        1e-14;
         relTol           0;
    };
```

Listing 4.7: The fvSolution temperature section

Initially a temperature tolerance value of 1e-7 was used in the `fvSolution` file. However, multiple simulations seemed to suffer from a problem; after running for some time the calculations for the temperature profile, which should be performed for every cell at every time step, would simply stop being executed. After many attempts of figuring out why this happened the cause was found. The temperature tolerance is the convergence criterion used by the solver to check if more iterations for the temperature equation are necessary. If the change in temperature is so small that the tolerance criterion is met already with 0 iterations, no iterations will take place. During some simulations the change in temperature was so small that no iterations on the temperature equation were performed and therefore the temperature profile simply stopped developing in these cells. After discovering this the tolerance was set to 1e-14 which solved the problem. The entire fvSolution file can be found in Appendix B.5.

4.2.2. Adding viscous dissipation

Viscous dissipation is the irreversible process where the shear forces in a fluid create heat. This process can be taken into account in this model by adding a viscous dissipation term to the temperature equation.

The temperature equation with viscous dissipation included looks as shown in Equation 4.11, where the last term on the RHS is the viscous dissipation term.

$$\rho c_p \frac{\partial(T)}{\partial t} + \rho c_p \nabla \cdot (uT) = \kappa \nabla^2 T + \tau : \nabla v \tag{4.11}$$

Dividing both sides of Equation 4.11 by ρc_p and taking the diffusion term to the left side results in Equation 4.12.

$$\frac{\partial(T)}{\partial t} + \nabla \cdot (uT) - \alpha \nabla^2 T = \frac{1}{\rho c_p} \tau : \nabla v \tag{4.12}$$

This term can be added to the temperature equation by adding line 6 to the code shown in Listing 4.8.

```
1    fvScalarMatrix TEqn
2        (
3            fvm::ddt(T)
4            + fvm::div(phi, T)
5            - fvm::laplacian(DT, T)
6            == (1/c)*(tau && gradU) // viscous heat dissipation term
7        );
```

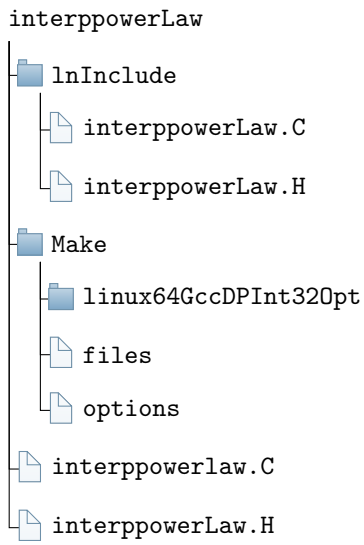
Listing 4.8: The temperature equation with viscous dissipation

Here it is important to note that the ρ is missing in the denominator on the right half side of the equation because it has been incorporated in the tau term during the customisation of the power-law, as will be explained in Subsection 4.3.

4.3. Power-Law customization

The standard power-law model that comes with OpenFOAMs viscosity models is not temperature dependent and therefore the temperature dependency of the power-law parameters that is expected to exist and has been quantified in Subsection 3.3 cannot be taken into account using this viscosity model. To fix this, the existing power-law code has been customized and renamed to "interppowerLaw". The old and new code will be compared in this subsection and the changes made will be described. The full code of the customised power-law model files can be found in Appendices B.2 and B.3.

The folder structure of the interppowerLaw looks as follows:



The `interppowerLaw.C` file contains all the calculations that are performed by the viscosity model and therefore the modifications to make the power-law model into a temperature dependent power-law will have to be applied here.

The `interppowerLaw.C` code is divided into 4 sections:

- static data members
- private member functions
- constructors
- member functions

The static data members section remained the same, apart from renaming "powerLaw" to "interppowerLaw". The private member functions section is where the important changes to the calculations were made. The code for the standard power-law model, which is included when a user downloads OpenFOAM, contains the following, see Listing 4.9.

The way in which this piece of code implements the powerlaw is as follows: The value given to the viscosity Nu is the value that is returned by the function "max". This function compares the value of nuMin and min(...) and gives the highest value as output. Here nuMin is the minimum viscosity, which has been declared in the transportProperties file which will be discussed in Section 4.4. The function min(...) is similar to the function max, except that it returns the lowest value as output. The three dots in min(...) replace the piece of code given in lines 11 to 20 in Listing 4.9. Min(...) compares the value of nuMax and k_*pow(...) and returns the lowest of the two. Where nuMax is the maximum viscosity, declared in the transportProperties. Here "k_*pow(...)" represents the flow consistency index k_ times the function pow. The function pow, which raises a base input to a power, has two inputs which are separated by the comma in line 18. The first input is the base value. The second input is the exponent value. In this case the first input is another max(...) function, which selects the highest value between "dimensionedScalar(dimTime, 1.0)*strainRate()", which represents the calculated strainrate, and "dimensionedScalar(dimless, small)", which represents a very small strainrate value (1.0e-06) which will be used if the calculated strainrate is almost 0, to avoid

the possibility of an almost infinite viscosity. The second input is "n_.value() - 1", where n_.value() is the shear thinning index. This structure of min and max functions calculates the kinematic viscosity for a power law fluid while making sure that the viscosity cannot be lower than the nuMin or higher than the nuMax, which are the minimum and maximum allowable viscosity as declared in the transportProperties file. The equation used to calculate the kinematic viscosity therefore boils down to the one shown in Equation 4.13, just with some extra constrictions to make sure the model functions properly.

$$\nu = k\left(\frac{\partial u}{\partial y}\right)^{n-1} \tag{4.13}$$

Normally the flow consistency index k has the dimensions $Pa \cdot s$, but if the kinematic instead of dynamic viscosity has to be calculated in OpenFOAM then the flow consistency index must be declared with units m^2/s, which is the result of dividing the flow consistency index k by the density ρ.

```
// * * * * * * * * * * Private Member Functions  * * * * * * * * * *//

Foam::tmp<Foam::volScalarField>
Foam::viscosityModels::powerLaw::calcNu() const
{
    return max
    (
        nuMin_,
        min
        (
            nuMax_,
            k_*pow
            (
                max
                (
                    dimensionedScalar(dimTime, 1.0)*strainRate(),
                    dimensionedScalar(dimless, small)
                ),
                n_.value() - scalar(1)
            )
        )
    );
}
```

Listing 4.9: The code for the standard power-law model

The customisation of the power-law begins with the creation of a scalar field for the new temperature dependent viscosity via the `createFields.H` file, as shown in Listing 4.10.

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

Listing 4.10: The code required to create the mynu field

Then the temperature and viscosity scalar fields have to be included and a new mystrainrate scalar field is created based on the pre-existing strainRate field, see Listing 4.11.

```
const volScalarField& T= U_.mesh().lookupObject<volScalarField>("T");
volScalarField mynu= U_.mesh().lookupObject<volScalarField>("mynu");
volScalarField mystrainrate = strainRate();
```

Listing 4.11: Declaration of the temperature, velocity, and strainrate fields

Then the dimensioned scalars are created for the power-law parameters k and n , and for the viscosity nu_tmp as shown in Listing 4.12.

```
dimensionedScalar k_local=k_;
dimensionedScalar n_local=n_;
dimensionedScalar nu_tmp = nuMin_;
```

Listing 4.12: Declaration of K, n, and nu_tmp

A scalar is created for myeps, which is a very small value that will be used to avoid errors due to rounding of small numbers in the interpolation steps, see Listing 4.13.

```
scalar myeps=1e-8;
```

Listing 4.13: Declaration of myeps

The final step before the code for the interpolated power law can be build consists of making a declaration of the 5 values for the temperature and the power law parameters k and n, which will be used to interpolate between. These values are declared in the transportProperties file under the section interppowerLawCoeffs as shown in Listing 4.14.

```
interppowerLawCoeffs
{
    T1   T1 [0 0 0 0 0 0 0] 298;
    T2   T2 [0 0 0 0 0 0 0] 323;
    T3   T3 [0 0 0 0 0 0 0] 348;
    T4   T4 [0 0 0 0 0 0 0] 373;
    T5   T5 [0 0 0 0 0 0 0] 398;

    k1   k1 [0 2 -1 0 0 0 0] 0.9799;
    k2   k2 [0 2 -1 0 0 0 0] 0.58819;
```

```
    k3   k3 [0 2 -1 0 0 0 0] 0.64039;
    k4   k4 [0 2 -1 0 0 0 0] 1.46627;
    k5   k5 [0 2 -1 0 0 0 0] 0.67754;

    n1   n1 [0 0 0 0 0 0 0] 0.18;
    n2   n2 [0 0 0 0 0 0 0] 0.198;
    n3   n3 [0 0 0 0 0 0 0] 0.089;
    n4   n4 [0 0 0 0 0 0 0] 0.005;
    n5   n5 [0 0 0 0 0 0 0] 0.084;
}
```

Listing 4.14: The declaration of the temperatures and power law parameters in the transportProperties files

The parameters which have just been declared in the transportProperties file are read in the interppowerLaw.C file using the following piece of code which can be found in the "Member Functions" section, see Listing 4.15

```
// * * * * * * * * * * * * * Member Functions  * * * * * * * * * * *//

bool Foam::viscosityModels::interppowerLaw::read
(
    const dictionary& viscosityProperties
)
{
    viscosityModel::read(viscosityProperties);

    interppowerLawCoeffs_ = viscosityProperties.optionalSubDict(
       typeName + "Coeffs");

    interppowerLawCoeffs_.lookup("T1") >> T1_;
    interppowerLawCoeffs_.lookup("T2") >> T2_;
    interppowerLawCoeffs_.lookup("T3") >> T3_;
    interppowerLawCoeffs_.lookup("T4") >> T4_;
    interppowerLawCoeffs_.lookup("T5") >> T5_;
    interppowerLawCoeffs_.lookup("k1") >> k1_;
    interppowerLawCoeffs_.lookup("k2") >> k2_;
    interppowerLawCoeffs_.lookup("k3") >> k3_;
    interppowerLawCoeffs_.lookup("k4") >> k4_;
    interppowerLawCoeffs_.lookup("k5") >> k5_;
    interppowerLawCoeffs_.lookup("n1") >> n1_;
    interppowerLawCoeffs_.lookup("n2") >> n2_;
    interppowerLawCoeffs_.lookup("n3") >> n3_;
    interppowerLawCoeffs_.lookup("n4") >> n4_;
    interppowerLawCoeffs_.lookup("n5") >> n5_;

    return true;
}


// ************************************************************************* //
```

Listing 4.15: Reading the parameters from the transportProperties file into the interppoweLaw.C file

Now that the preparatory work is done, the equations for the interpolation can be included. Linear interpolation will be applied via an if / if else / else statement to calculate the values of k and n over the sections between the five temperatures at which the values are known. The viscosity will be calculated using these values. The value of the viscosity will be written to the mynu scalar field. The boundary

conditions will be corrected, and finally the scalar field mynu will be returned as final output for the calculated nu. This part forms the core of the viscosity model. In code it looks as shown in Listing 4.16.

```
forAll(T.internalField(), cellI)
    {
    if (T[cellI]<298+myeps)
        {
        k_local.value() = 0.97990;
        n_local.value() = 0.18;
        }
    else if (T[cellI]>=298-myeps && T[cellI]<323+myeps)
        {
        k_local = k1_ + ((k2_-k1_)/(T2_-T1_))*(T[cellI]-T1_);
        n_local = n1_ + ((n2_-n1_)/(T2_-T1_))*(T[cellI]-T1_);
        }

    else if (T[cellI]>=323-myeps && T[cellI]<348+myeps)
        {
        k_local = k2_ + ((k3_-k2_)/(T3_-T2_))*(T[cellI]-T2_);
        n_local = n2_ + ((n3_-n2_)/(T3_-T2_))*(T[cellI]-T2_);
        }

    else if (T[cellI]>=348-myeps && T[cellI]<373+myeps)
        {
        k_local = k3_ + ((k4_-k3_)/(T4_-T3_))*(T[cellI]-T3_);
        n_local = n3_ + ((n4_-n3_)/(T4_-T3_))*(T[cellI]-T3_);
        }

    else if (T[cellI]>=373-myeps && T[cellI]<=398+myeps)
        {
        k_local = k4_ + ((k5_-k4_)/(T5_-T4_))*(T[cellI]-T4_);
        n_local = n4_ + ((n5_-n4_)/(T5_-T4_))*(T[cellI]-T4_);
        }

    else
        {
        k_local.value() = 0.67754;
        n_local.value() = 0.084;
        }

        nu_tmp = max
            (
        nuMin_,
        min
        (
            nuMax_,
            (k_local)*pow
            (
                max
                (
                    mystrainrate[cellI],
                    small
                ),
                n_local.value() - scalar(1)
            )
        )
            );
```

```
55
56              mynu[cellI] = nu_tmp.value()/1020;
57      }
58                  mynu.correctBoundaryConditions();
59          return mynu;
60  }
```

Listing 4.16: The code from the interppowerLaw.C file which calculates the temperature dependent viscosity

It is important to note that the flow consistency index k, in the code labeled k_local, is read in from the transportProperties file as a dimensionedScalar with the dimensions $\frac{m^2}{s}$. Since the dimensions of k are normally given in $\frac{kg}{m*s}$ this means the value of k in the transportProperties file has already been divided by the density, which results in $\frac{k}{\rho}$ which has units $\frac{m^2}{s}$. The value of nu that is calculated by the interppowerlaw is then used by the `my_viscousHeatingSolver.C` file in line 120 as fluid.nu(), see Listing 4.17.

```
1       volTensorField tau = fluid.nu() * (gradU + gradU.T());
```

Listing 4.17: Line 120 from the my_viscousHeatingSolver

This is why in Subsection 4.2.2 it was mentioned that ρ was missing on the right half side of the equation because it had been incorporated via the tau term already.

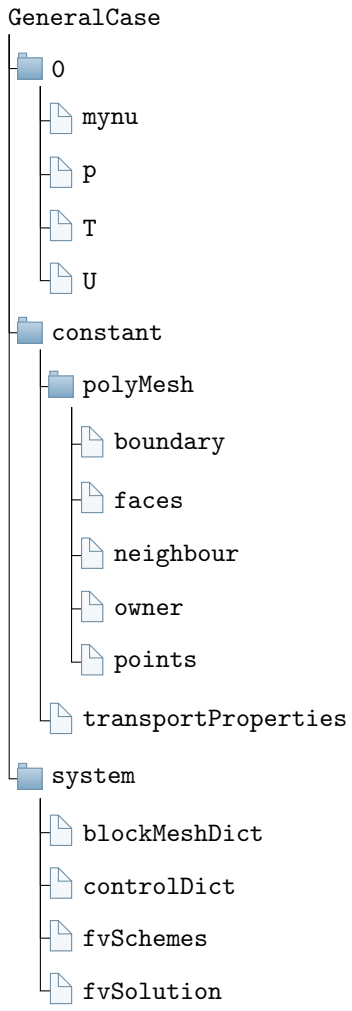
Now that the nonNewtonianIcoFoam solver and the interppowerlaw viscosity model are fully functioning, the next step is to start building a folder structure that is required to run a simulation in OpenFOAM. This process will be described in the next subsection.

4.4. Building the general case structure

Now that the my_viscousHeatingSolver and the interppowerlaw viscosity model have been finalized, the general case structure that is required to run a simulation in OpenFOAM has to be built.

A basic case in OpenFOAM should include 3 directories: 0, constant, and system. In the 0 directory, or some other time directory, the initial values or boundary conditions are defined for all the relevant parameters. The constant folder contains files specifying the physical properties, such as the file transportProperties. All information regarding the case mesh is stored in the subdirectory polyMesh, under the constant folder. Meshes can either be generated in OpenFOAM using for instance a blockMeshDict file, or can be imported after being generated by other software. The system directory contains at least the following 3 files: controlDict, fvSchemes, and fvSolution. In the controlDict file the settings are defined which specify how the simulation should be run. This includes for example the start/end time, time step, and the parameters for data output. The fvSchemes file contains information regarding which discretization schemes should be used. The fvSolution file contains information that specifies the equation solvers, tolerances, and other algorithm controls.

For a case using the my_viscousHeatingSolver in combination with the interppowerLaw viscosity model, a general case directory contains the following files:



4.5. Geometry

The geometry used for this model is a simplified version of the true geometry of Rival Foods' apparatus. The reason for simplifying the geometry is that it is easier to recreate the geometry in OpenFOAM when the geometry does not contain many small details such as valves and corrugations and secondly this also makes sure the true geometry of Rival Foods' apparatus stays private. The geometry can be seen in Figure 4.1. Here R1 is the outer radius of the outer cylinder, R2 is the inner radius of the outer cylinder, R3 is the outer radius of the inner cylinder, R4 is the inner radius of the inner cylinder, and H is the height of the cylinder.

For the OpenFOAM simulation only one cell will be used in the angular direction, which will form a wedge of 5°. The reason for this is that the flow inside the geometry is axi-symmetric and therefore simulating only part of the 360° allows for a greatly reduced computation time and load on the TU Delft cluster, while producing the same results.

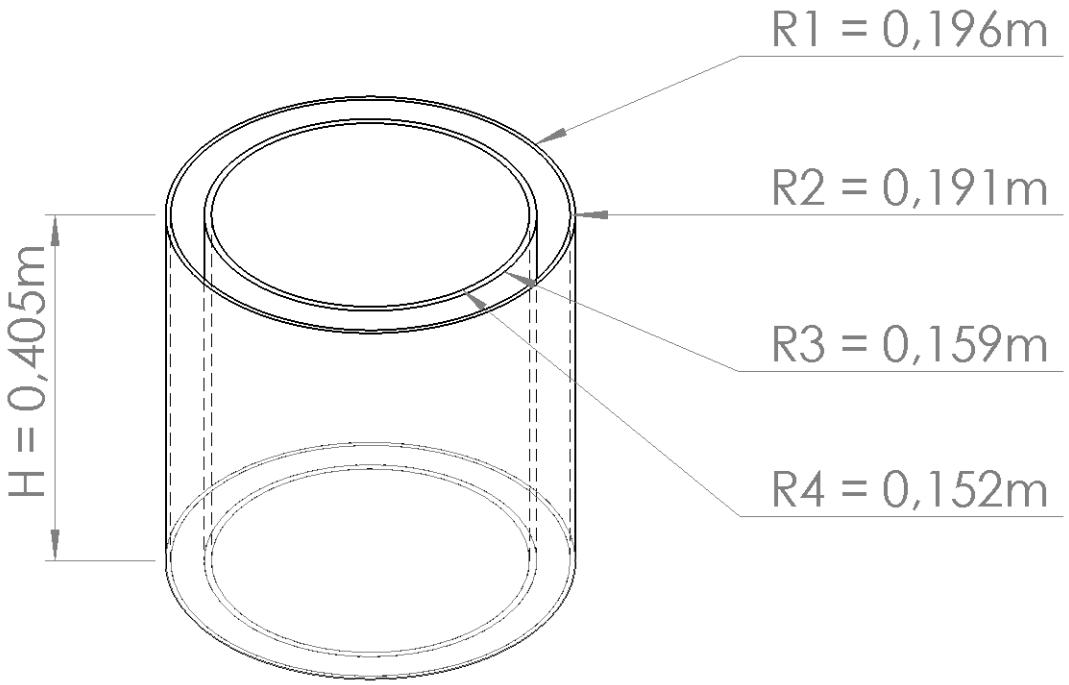


Figure 4.1: Geometry of the Couette-Cell based case modelled in OpenFOAM, where the radius and height are given in meters

In OpenFOAM the geometry of the case is defined in the blockMeshDict file, which can be found under the system directory. The blockMeshDict file contains the code that is shown in Listing 4.18.

```
convertToMeters 0.001;

vertices
(
(   158.849 -6.935      0.000)      //1
(   190.818 -8.331      0.000)      //2
(   190.818 8.331       0.000)      //3
(   158.849 6.935       0.000)      //4
(   158.849 -6.935      405.000)    //5
(   190.818 -8.331      405.000)    //6
(   190.818 8.331       405.000)    //7
(   158.849 6.935       405.000)    //8
);

blocks
(
    hex (0 1 2 3 4 5 6 7) ( 20 1 80 ) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
     frontwedge
     {
        type    wedge;
        faces   ((0 1 5 4));
     }

     backwedge
     {
        type    wedge;
        faces   ((3 7 6 2));
     }

     bot
     {
        type    wall;
        faces   ((0 3 2 1));
     }

     top
     {
        type    wall;
        faces   ((4 5 6 7));
     }

     outerwall
     {
        type    wall;
        faces   ((1 2 6 5));
     }
```

```
55
56      innerwall
57      {
58          type    wall;
59          faces   ((0 4 7 3));
60      }
61  )
62
63  mergePatchPairs
64  (
65  );
```

Listing 4.18: The blockMeshDict code

The code starts with a conversion `converToMeters` which indicates that the numbers will be given in mm.

In line 3 to 13 the 8 vertices of the wedge-shaped hexahedron are defined. Figure 4.2 shows these vertices together with the coordinate system, to make the blockMeshDict file more clear. These vertices have to be defined in Cartesian coordinates and therefore the radius $R_2 = 0.191m$ and $R_3 = 0.159m$ are used in combination with Equation 4.14 and 4.15 to calculate the coordinates of the vertices. Since the wedge has an angle of 5° the vertices are calculated at $\theta = 2.5°$ and $\theta = -2.5°$.

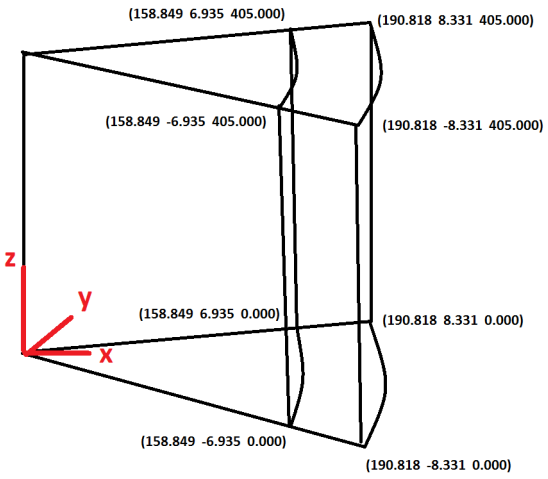


Figure 4.2: Overview of vertices of the wedge, where at each vertex 3 numbers are given, indicating the x/y/z-coordinates

$$x = r * cos(\theta) \tag{4.14}$$

$$y = r * sin(\theta) \tag{4.15}$$

In line 15 to 18 the hexahedron block is defined by giving the 8 vertices "hex (0 1 2 3 4 5 6 7)" together with the amount of cells in each Cartesian direction "(20 1 80)" and the cell expansion ratios "simpleGrading (1 1 1)", where the three numbers indicate the expansion ratios in each of the directions of the block. The thickness of the slab is $191 - 159 = 32mm$ and will be divided over 20 cells, which will therefore each have a length of $32/20 = 1.6mm$ in x-direction. The height of the slab is $405mm$ and will be divided over 80 cells, which will each have a length of $405/80 = 5.0625mm$ in z-direction. The cell expansion ratio allows the mesh to be refined or graded in specified directions. The ratio is that of the width of the end cell δ_e along one edge of a block to the width of the start cell δ_s along that edge, as shown in Figure 4.3. For this case the effects at the inner and outer cylinder are equally important and therefore a constant expansion ratio with a value of 1 has been chosen, so that the cells are of constant size throughout the entire mesh.

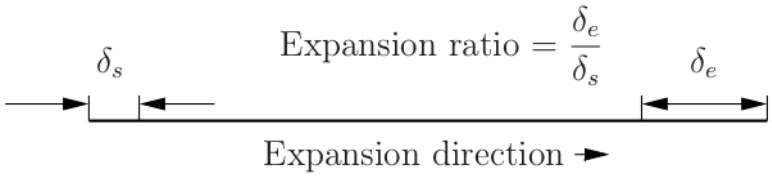


Figure 4.3: Mesh grading along a block edge [23]

In line 24 to 61 the boundaries are defined. The boundary of the wedge consists of 6 patches and each patch has its own name. For each patch the type and faces are defined. The patch type defines how the patch should be treated, for this case only wedge and wall types are used. The wedge type can be applied in pairs to 2-D rotationally periodic cases to represent planes in the swirl direction. The wall type, as the name suggests, indicates that the patch should be treated as a solid wall. The faces are defined by giving the four vertices of the face.

4.6. Boundary conditions

When working with OpenFOAM the boundary conditions need to be specified in the 0 folder. This folder contains the boundary conditions that will be applied at the start of the simulation. For the wedge geometry the 6 boundaries (boundary patches) that have been defined in the previous section need to be specified. A graphical overview of these patches can be seen in Figure 4.4:

- frontwedge, the nearest plane where the wedge has been cut out from the cylinder
- backwedge, the furthest plane where the wedge has been cut out from the cylinder
- bot, part of the bottom side of the cylinder wedge
- top, part of the top side of the cylinder
- outerwall, part of outer cylinder
- innerwall, part of the inner cylinder

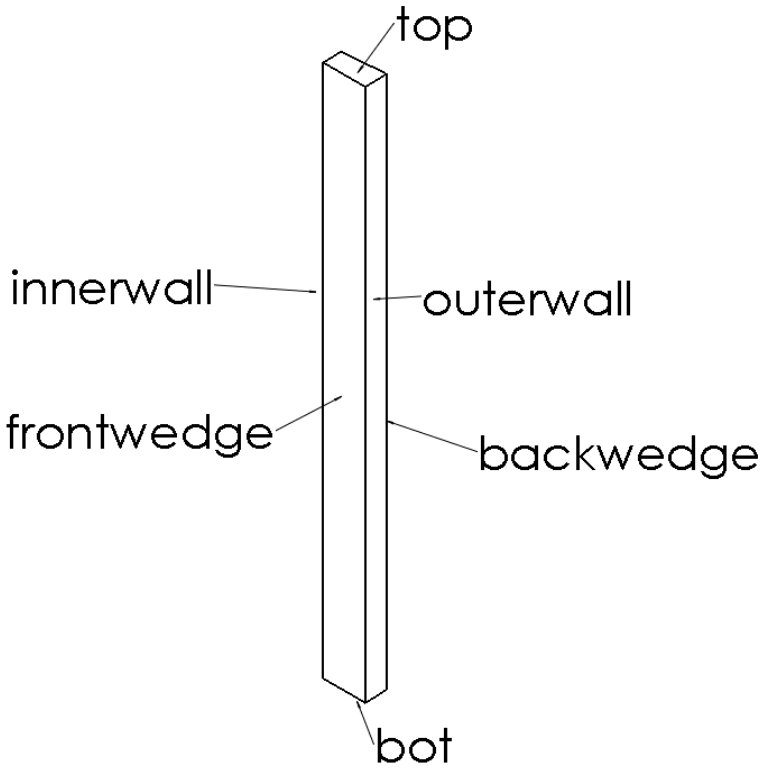


Figure 4.4: Overview of the wedge boundaries

4.6.1. Temperature boundary conditions

On the frontwedge and backwedge a wedge boundary condition has been applied, which indicates that these boundaries should be treated as if more exact copies of the wedge were adjacent to it and therefore the geometry continues in the direction perpendicular to these faces. In other CFD packages this wedge boundary condition is often named rotationally periodic.

The bot and top have a zeroGradient boundary condition, which indicates that the gradient of temperature perpendicular to the face is equal to zero and therefore there is no heat flux through this face. Listing 4.19 shows how the temperature boundary conditions are applied in the OpenFOAM code of the T-file in the 0-directory

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      T;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions        [0 0 0 1 0 0 0];

internalField    uniform 298;

boundaryField
{
    frontwedge
    {
    type    wedge;
    }

    backwedge
    {
    type    wedge;
    }

    bot
    {
        type    zeroGradient;
    }

    top
    {
        type    zeroGradient;
    }

    outerwall
    {
    type                    groovyBC;
        variables               "htot=1340;Tinf=403;rho=1020.0;cp
```

```
            =3506.8;k=DT*rho*cp;";
47          valueExpression         "Tinf";
48          fractionExpression      "1.0/(1.0 + k/(mag(delta())*htot))";
49      }
50
51      innerwall
52      {
53      type                    groovyBC;
54          variables               "htot=800;Tinf=403;rho=1020.0;cp=3506.8;
               k=DT*rho*cp;";
55          valueExpression         "Tinf";
56          fractionExpression      "1.0/(1.0 + k/(mag(delta())*htot))";
57      }
58
59  }
60
61
62  // ************************************************************************* //
```

Listing 4.19: The temperature boundary conditions as defined in the T-file in the 0 directory

The most difficult boundary condition is the one applied on the outerwall and innerwall of the cylinder. In this case steam will be used to heat the cylinders of the machine. This steam will condensate on the surface of the cylinders and the heat will be transferred through the stainless steel walls into the mixture. The situation here is that we are dealing with a solid region that is adjacent to a fluid/gas and we wish to describe the heat transfer between the two using Newton's law of cooling. The most suitable boundary condition to apply here is the third or mixed kind as described in Chapter 3.2.3 of Basic Heat and Mass Transfer by Mills [28], see Eq. 4.16. Here the position x=R1 indicates the boundary surface at the radius R1 in Figure 4.1 if the heat transfer at the outer cylinder is considered.

$$-k\frac{\partial T}{\partial x}\Big|_{x=R1} = h_c(T|_{x=R1} - T_e) \tag{4.16}$$

In OpenFOAM this boundary condition is not part of the standard boundary conditions that can be used. In order to use this boundary condition a software package called "swak4Foam" has been installed. This software package contains "groovyBC". An overall heat transfer coefficient can be calculated for the convective heat transfer by steam and the conduction through the stainless steel wall. This overall heat transfer coefficient, together with several other values are used as input for the groovyBC as shown in line 46 and 54 in Listing 4.19, where the overall heat transfer coefficient is defined as "htot".

In order to determine the average heat transfer coefficient for condensation, a number of calculations have been made based on Mills' chapter 7.2 Film condensation. This average heat transfer coefficient is then used to find the overall heat transfer coefficient. The calculations will be described below.

Calculation of the overall heat transfer coefficient

The temperature of the saturated steam used to heat the cylinders is 130°C. Of course the temperature of the wall varies during the process. Initially the wall will be at 20°C. The dough mixture in between the cylinders has a low thermal conductivity of $0.44\frac{W}{m*K}$ while the stainless steel has a thermal conductivity of $16\frac{W}{m*K}$ and thus the thermal resistance of the metal wall is much lower than the thermal resistance of the dough. Therefore the cylinder wall temperature will be very close to 130°C during the entire process. A simulation made by one of Rival Foods' engineers confirmed this assumption, showing that after several seconds the wall reached a temperature of 129.4°C. Therefore the assumption is made that the wall temperature is 129.5°C for these calculations.

The average heat transfer coefficient for condensation on the outside of a vertical wall or tube is given by Equation 4.17. Note that this equation ignores the liquid subcooling term, $c_p(T_{sat} - T)$. This term accounts for the sensible heat given up by the condensate as it cools below the saturation temperature T_{sat}. Usually this term is small: for example, when steam condenses at a pressure of 1 atm and there is a

temperature drop of 10 degrees across the film, the subcooling term is at most 2% of the enthalpy of phase change h_{fg}. Since the pressure of the saturated steam at 130°C is roughly 1.7 bar and the temperature drop across the film is similar the subcooling term is assumed to be negligible. Additionally the effect of tube curvature has been ignored since the surfaces on which the steam condenses have relatively large radii of R1=0,196m and R2=0,152m.

$$\bar{h} = 0.943 \left[\frac{h_{fg} g (\rho_l - \rho_v) k_l^3}{L (T_{sat} - T_w) \nu_l} \right]^{1/4} \tag{4.17}$$

Where h_{fg} is the difference in enthalpy between the gas and fluid states, g is the gravitational constant, ρ_v is the density of the steam,, ρ_l is the density of the liquid, k_l is the thermal conductivity of the liquid, L is the height of the wall or tube, T_w is the temperature of the wall, and ν_l is the kinematic viscosity of the liquid.

The values of h_{fg} and ρ_v are taken at 403°K (130°C) from Mills' Table A.12a.

- $h_{fg} = 2.1743 * 10^6 J/kg$
- $\rho_v = 1.4997 kg/m^3$

The liquid phase properties have to be evaluated at the reference temperature, where $\alpha = 0.5$ has been used.

$$T_r = T_w + \alpha (T_{sat} - T_w) = 402.5 + 0.5(403 - 402.5) = 402.75^\circ K \tag{4.18}$$

The thermal conductivity, density, and kinematic viscosity have been taken from Mills' Table A.8 at T_r = 402.75°K.

- $k_l = 0.685 W/m * K$
- $\rho_l = 937.75 kg/m^3$
- $\nu_l = 0.24 * 10^{-6} m^2/s$

Inserting all of these values in Equation 4.19 results in

$$\bar{h} = 0.943 \left[\frac{(2.1743 * 10^6)(9.81)(937.75 - 1.4997)(0.685)^3}{(0.405)(403 - 402.5)(0.24 * 10^{-6})} \right]^{1/4} = 17963 \frac{W}{m^2 K} \tag{4.19}$$

The total condensation rate, $\dot{m}$, can now be calculated, which will be done for both the outer and inner cylinders.

The surface area of the outer cylinder is $A_{out} = \pi * D_{out} * h = \pi * 0.392 * 0.405 = 0.4988 m^2$.
The surface area of the inner cylinder is $A_{in} = \pi * D_{in} * h = \pi * 0.304 * 0.405 = 0.3868 m^2$.

$$\dot{m}_{out} = \frac{\dot{Q}}{h_{fg}} = \frac{\bar{h} \Delta T A}{h_{fg}} = \frac{(17963)(0.5)(0.4988)}{(2.154 * 10^6)} = 0.00208 \frac{kg}{s} \tag{4.20}$$

$$\dot{m}_{in} = \frac{\dot{Q}}{h_{fg}} = \frac{\bar{h} \Delta T A}{h_{fg}} = \frac{(17963)(0.5)(0.3868)}{(2.154 * 10^6)} = 0.00161 \frac{kg}{s} \tag{4.21}$$

The mass flow rate per unit width of film Γ is

$$\Gamma_{out} = \frac{\dot{m}_{out}}{\pi * D} = \frac{0.00208}{\pi * 0.392} = 0.00169 \frac{kg}{m * s} \tag{4.22}$$

$$\Gamma_{in} = \frac{\dot{m}_{in}}{\pi * D} = \frac{0.00161}{\pi * 0.304} = 0.00169 \frac{kg}{m * s} \tag{4.23}$$

Assuming $\rho_v \ll \rho_l$ the film thickness can be calculated at the bottom of the cylinder, where the thickness is at its maximum

$$\delta_{out} = \delta_{in} = \left(\frac{3 * \nu_l * \Gamma_{out}}{\rho_l * g}\right)^{1/3} = \left(\frac{(3)(0.24 * 10^{-6})(0.00169)}{(934.75)(9.81)}\right)^{1/3} = 5.0994 * 10^{-5} m \tag{4.24}$$

The Reynolds number of a falling film is defined in terms of the bulk velocity u_b and hydraulic diameter D_h of the film, where the value of $\mu_l = 2.21 * 10^{-4}$ is taken from Mills' Table A.8.

$$u_{b_{out}} = u_{b_{in}} = \frac{\Gamma_{out}}{\rho_l * \delta_{out}} = \frac{0.00169}{(934.75)(5.0994 * 10^{-5})} = 0.03543 \frac{m}{s} \tag{4.25}$$

$$D_{h_{out}} = D_{h_{in}} = 4 * \delta_{out} = 4 * (5.0994 * 10^{-5}) = 2.0398 * 10^{-4} m \tag{4.26}$$

Using these to calculate the Reynolds number

$$Re_{out} = Re_{in} = \frac{\rho_l * u_{b_{out}} * D_{h_{out}}}{\mu_l} = \frac{(934.75)(0.05627)(2.5705 * 10^{-4})}{0.000221} = 31 \tag{4.27}$$

This film Reynolds number can be used to define which kind of flow is encountered in this problem.
Three different types of flow are possible [28]: *laminar, wavy laminar, and turbulent.* For water at roughly 300°K the flow starts to show wavy laminar behaviour at $Re \simeq 30$. The transition to turbulent flow starts to happen at the outer region of the film at $Re \simeq 1000$ and the transition to turbulent flow becomes complete at the inner region when $Re \simeq 1800$. The value of $Re = 31$ is right at the transition region to wavy laminar flow and therefore the average heat transfer coefficient will also be calculated for the wavy laminar case to check if the results differ much. To do so, it is necessary to calculate the Jakob number Ja_l, for which Mills' Table A.8 has been used to find $c_{p_l} = 4255 J/kgK$.

$$Ja_l = \frac{c_{p_l}(T_{sat} - T_w)}{h_{fg}} = \frac{4255(403 - 402.5)}{2.1743 * 10^6} = 0.00978 \tag{4.28}$$

The average Nusselt number for wavy laminar film condensation can then be calculated, using Mills' Table A.8 to find $Pr = 1.376$.

$$\overline{Nu} = \left[\frac{Pr_l}{4Ja_l} * \frac{\left(\frac{\nu_l^2}{g}\right)^{1/3}}{L}\right]^{0.18} = \left[\frac{1.376}{4 * 0.00978} * \frac{\left(\frac{(0.24 * 10^{-6})^2}{9.81}\right)^{1/3}}{0.405}\right]^{0.18} = 0.473 \tag{4.29}$$

With this Nusselt number the average heat transfer coefficient can be calculated

$$\bar{h} = \frac{\overline{Nu} * k_l}{\left(\frac{\nu_l^2}{g}\right)^{1/3}} = \frac{0.473 * 0.685}{\left(\frac{(0.24 * 10^{-6})^2}{9.81}\right)^{1/3}} = 17968 \frac{W}{m^2 K} \tag{4.30}$$

This results in an overall heat transfer coefficient on the outer cylinder of

$$U_{out} A_{out} = \frac{1}{\frac{1}{2\pi * L * R_1 * \bar{h}} + \frac{ln(\frac{R_1}{R_2})}{2\pi * L * k_{rvs}}} = \frac{1}{\frac{1}{2\pi * 0.405 * 0.196 * 17968} + \frac{ln(\frac{0.196}{0.191})}{2\pi * 0.405 * 16}} = 1340 \frac{W}{K} \tag{4.31}$$

Using an outer area of $A_{out} = 2\pi * R_1 * L = 2\pi * 0.196 * 0.405 = 0.4988 m^2$ results in an overall heat transfer coefficient for the outer cylinder of

$$U_{out} = \frac{1340}{0.4988} = 2686 \frac{W}{m^2 K} \tag{4.32}$$

The same can be done for the inner cylinder resulting in an overall heat transfer coefficient of

$$U_{in} A_{in} = \frac{1}{\frac{1}{2\pi * L * R_4 * \bar{h}} + \frac{ln(\frac{R_3}{R_4})}{2\pi * L * k_{rvs}}} = \frac{1}{\frac{1}{2\pi * 0.405 * 0.152 * 17968} + \frac{ln(\frac{0.159}{0.152})}{2\pi * 0.405 * 16}} = 800 \frac{W}{K} \tag{4.33}$$

Using an inner area of $A_{in} = 2\pi * R_4 * L = 2\pi * 0.152 * 0.405 = 0.3868^2$ results in an overall heat transfer coefficient for the outer cylinder of

$$U_{in} = \frac{800}{0.3868} = 2068 \frac{W}{m^2 K} \tag{4.34}$$

These values for the overall heat transfer coefficients can now be used in the groovyBC applied on the cylinder walls, as shown in the code at the beginning of this subsection. In the simulations the values $U_{out} = 1340 W/m^2 K$ and $U_{in} = 800 W/m^2 K$ were accidentally used instead of the values $U_{out} = 2686 W/m^2 K$ and $U_{in} = 2068 W/m^2 K$. This means the thermal resistance consisting of the heat transfer by condensation of steam and conduction through the metal, will be even lower and therefore the wall will reach the assumed wall temperature of 129.5°C even faster. However, this effect is minimal and therefore this small error has been accepted without rerunning all the simulations.

4.6.2. Velocity boundary conditions

On the front- and backwedge the wedge boundary condition is applied once again to make sure these boundaries are treated as if the geometry would continu on these planes.

The bot, top, and outerwall boundaries have a no-slip boundary condition applied to them to make sure they are treated as stationary planes where the velocity is equal to zero where the fluid contacts the plane.

The innerwall is the only moving part in this geometry and therefore a rotatingWallVelocity boundary condition is applied. For this boundary condition the wall has a rotational velocity around the z-axis of 30 RPM which converts to 3.14 rad/s.

The velocity boundary conditions as applied in the OpenFOAM code are shown in Listing 4.20.

```
1
2  /*--------------------------------*- C++ -*----------------------------------*\
3    =========                 |
4    \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
5     \\    /   O peration     | Website:  https://openfoam.org
6      \\  /    A nd           | Version:  8
7       \\/     M anipulation  |
8  \*---------------------------------------------------------------------------*/
9  FoamFile
10 {
11     version     2.0;
12     format      ascii;
13     class       volVectorField;
14     object      U;
15 }
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 dimensions      [0 1 -1 0 0 0 0];
19
20 internalField   uniform (0 0 0);
21
22 boundaryField
23 {
24     frontwedge
25     {
26         type    wedge;
27     }
28
29     backwedge
30     {
31      type   wedge;
32     }
33
34     bot
35     {
36      type   noSlip;
37     }
38
39     top
40     {
41         type    noSlip;
42     }
43
44     outerwall
45     {
```

```
46          type    noSlip;
47      }
48
49      innerwall
50      {
51          type    rotatingWallVelocity;
52          origin  (0 0 0);
53          axis    (0 0 1);
54          omega   constant    3.1415927;      // rad/s
55      }
56
57
58  }
59
60  // ************************************************************************* //
```

Listing 4.20: The velocity boundary conditions as defined in the T-file in the 0 directory

4.6.3. Pressure boundary conditions

On the front- and backwedge the wedge boundary condition is applied again. All the other boundaries have a zeroGradient boundary condition applied to them because there is no pressure-gradient on these boundaries. The pressure boundary conditions as defined in the OpenFOAM code are shown in Listing 4.21.

It is important to realise that the values of pressure will not be calculated in this simulation and only the derivatives of the pressure, with respect to the spatial coordinates, are used to solve the equations in the PISO-algorithm.

```
1
2  /*--------------------------------*- C++ -*----------------------------------*\
3    =========                 |
4    \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
5     \\    /   O peration     | Website:  https://openfoam.org
6      \\  /    A nd           | Version:  8
7       \\/     M anipulation  |
8  \*---------------------------------------------------------------------------*/
9  FoamFile
10 {
11     version     2.0;
12     format      ascii;
13     class       volScalarField;
14     object      p;
15 }
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 dimensions      [0 2 -2 0 0 0 0];
19
20 internalField   uniform 0;
21
22 boundaryField
23 {
24     frontwedge
25     {
26     type    wedge;
27     }
28
29     backwedge
30     {
31     type    wedge;
32     }
33
34     bot
35     {
36     type    zeroGradient;
37     }
38
39     top
40     {
41     type    zeroGradient;
42     }
43
44     outerwall
45     {
46         type    zeroGradient;
47     }
48
```

```
49      innerwall
50      {
51          type    zeroGradient;
52      }
53
54  }
55
56  // ************************************************************************* //
```

Listing 4.21: The pressure boundary conditions as defined in the p-file in the 0 directory

4.6.4. Viscosity boundary conditions

Originally the nonNewtonianIcoFoam solver did not contain a viscosity boundary condition file, but while making the power-law model temperature dependent it turned out to be necessary to add this file.

On the front- and backwedge the wedge boundary condition is applied again. All the other boundaries have a zeroGradient boundary condition applied to them because there is no viscosity-gradient on these boundaries.

The viscosity boundary conditions as defined in the OpenFOAM code are shown in Listing 4.22

```
1
2  /*--------------------------------*- C++ -*----------------------------------*\
3    =========                 |
4    \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
5     \\    /   O peration     | Website:  https://openfoam.org
6      \\  /    A nd           | Version:  8
7       \\/     M anipulation  |
8  \*---------------------------------------------------------------------------*/
9  FoamFile
10 {
11     version     2.0;
12     format      ascii;
13     class       volScalarField;
14     object      mynu;
15 }
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 dimensions        [0 2 -1 0 0 0 0];
19
20 internalField     uniform 1e-10;
21
22
23 boundaryField
24 {
25     frontwedge
26     {
27     type     wedge;
28     }
29
30     backwedge
31     {
32     type     wedge;
33     }
34
35     bot
36     {
37     type     zeroGradient;
38     }
39
40     top
41     {
42     type     zeroGradient;
43     }
44
45     outerwall
46     {
47         type     zeroGradient;
48         value    uniform 1e-10;
```

```
49      }
50
51      innerwall
52      {
53          type    zeroGradient;
54          value   uniform 1e-10;
55      }
56
57  }
58
59
60  // ************************************************************************* //
```

Listing 4.22: The viscosity boundary conditions as defined in the mynu-file in the 0 directory

4.7. Numerical schemes

In the file `fvSchemes`, which can be found in the system directory, the numerical schemes are specified. Normally there are 6 sets of terms for which the numerical schemes have to be specified.

- timeSchemes: first and second order time derivatives, *e.g.* $\partial/\partial t, \partial^2/\partial^2 t$
- gradSchemes: gradient ∇
- divSchemes: divergence $\nabla\cdot$
- laplacianSchemes: Laplacian ∇^2
- interpolationSchemes: cell to face interpolations of values.
- snGradSchemes: component of gradient normal to a cell face.

The schemes chosen will be discussed per term.

timeSchemes The time scheme was initially set to Euler (Euler implicit), which is a basic first order scheme. After encountering some strange phenomena during simulations (faster temperature distribution near the top and bottom ends of the cylinder, possibly due to numerical diffusion) this scheme was changed to the backward scheme, which uses second order backward-differencing.

gradSchemes The gradient scheme is set to Gauss linear, since this is the default discretization scheme that is primarily used for gradient terms, according to the OpenFOAM userguide. The Gauss entry specifies the standard finite volume discretization of Gaussian integration, which requires the interpolation of values from cell centres to face centres. The interpolation scheme is then given by the 'linear' entry, meaning linear interpolation or central differencing.

divSchemes The divergence schemes contain both advection terms, where the velocity U provides the advective flux, and other terms, which are often diffusive, such as $\vec{\nabla}\cdot(\nu\vec{\nabla}\vec{u})$ and $\vec{\nabla}\vec{u}\cdot\vec{\nabla}\nu$. In this case the Gauss linear scheme is used for the velocity term, since this is generally recommended in the userguide. For the temperature initially the first order Gauss upwind scheme was used, but after testing the simulation it was concluded that this resulted in large deviations from the expected temperature profile near the top and bottom regions of the model. Therefore the divergence scheme was changed to the second order vanLeer scheme, after which the large deviations were resolved.

laplacianSchemes For the Laplacian terms a Gauss linear corrected scheme has been used, which is second order. The Gauss schemes are the only choice of discretization. 'Linear' refers to the interpolation scheme that has been used and 'corrected' refers to the surface normal gradient scheme that has been used.

interpolationSchemes For the interpolation of values, typically from cell centres to cell faces, OpenFOAM uses linear interpolation for practically all cases. Therefore linear interpolation has been chosen for this simulation as well. This interpolation is mostly used to calculate the velocity in order to calculate phi, which is the mass flux through the cell face.

snGradSchemes For the surface normal gradient the orthogonal scheme is used. The surface normal gradient is calculated at the cell face. The calculation is second order accurate for the gradient normal to the face if the vector that connects the cell centres is orthogonal to the face. Normally the orthogonal scheme is only recommended for meshes with very low non-orthogonality, *e.g.* maximum 5°. Since this wedge only spans an angle of 5° this scheme suffices.

4.8. Simulations

This section will describe the differences between the simulated cases.

4.8.1. Base case

The cases that have been simulated are all built from the same base case. The geometry, boundary conditions, discretization schemes, solver, and viscosity model of this base case have already been discussed in this chapter. Only the material properties that will be used in the simulations still need to be determined.

For the SPI & WG mixture the relevant properties are:

- thermal conductivity $k = 0.44 \frac{W}{m*K}$
- specific heat $c_p = 3507 \frac{J}{kg*K}$
- density $\rho = 1020 \frac{kg}{m^3}$

For the housing made of stainless steel 316 the relevant properties are:

- thermal conductivity $k = 16 \frac{W}{m*K}$
- specific heat $c_p = 490 \frac{J}{kg*K}$
- density $\rho = 8070 \frac{kg}{m^3}$

4.8.2. Variations on the base case

Multiple variations on the base case simulation have been made. Apart from the standard product thickness of 32mm, a product with a thickness of 23mm and 41mm have been simulated as well. To allow for a fair comparison between the 23mm, 32mm, and 41mm case, the number of cells in the direction of the slab's thickness has been adjusted accordingly. Otherwise the 23mm and 41mm case would have the same number of cells, which would result in a much coarser grid for the 41mm case, making the results less accurate. The 32mm case has 20 cells in the x-direction, which means that each cell is $32/20 = 1.6mm$. Using this same value for the 23mm case results in $23/1.6 = 14.375$. Rounding upwards this means 15 cells will be required. For the 41mm case this results in $41/1.6 = 25.625$, so 26 cells will be required.

Quantifying the influence that the starting temperature of the product mixture has on the final temperature distribution is also of interest for Rival Foods and therefore two different preheat temperatures have been simulated. The product mixture currently enters the production unit at roughly 25°C. Preheating the mixture to 50°C is also an option and therefore a case with a starting temperature of 50°C has been simulated as well. The product mixture is normally kept in a cooling unit at roughly 10°C. Therefore another case has been created where the assumption is made that the product is moved directly from the cooling unit into the production unit, which would give a preheat temperature of 10°C instead of 25°C.

Two different variations on the density of the mixture have been simulated for $\rho = 880 \frac{kg}{m^3}$ and $\rho = 1280 \frac{kg}{m^3}$, since composition and density might vary in future ingredient mixtures.

To quantify the influence of viscous dissipation on the heating process a case with and without viscous dissipation have been been simulated.

The viscous dissipation depends on the shear rate and therefore the viscous dissipation will be higher in regions where the shear rate changes rapidly. Looking at the geometry in this model it should be clear that there are two borders where the shear rate changes very rapidly. These borders are where the rotating inner cylinder and the stationary top and bottom plates meet. A refined grid has also been simulated, where the number of points in the x-direction has been increased from 20 to 40 and in the z-direction from 80 to 160 to find out what the influence on the viscous dissipation in these vertices will be.

So to summarize the following simulations have been done, where all cases were copies of the base case with just 1 modification:

- base case
- slab thickness of 23mm

- slab thickness of 41mm
- preheat temperature of 10°C
- preheat temperature of 50°C
- mixture density of $\rho = 880 \frac{kg}{m^3}$
- mixture density of $\rho = 1280 \frac{kg}{m^3}$
- no viscous dissipation

These simulations have been performed for two scenarios; using the constant power law parameters $K = 1600 Pa * s^n$ and $n = 0.13$, and using the interppowerLaw model which calculates the viscosity based on the temperature.

5

Model verification

A model is only useful if the results give a trustworthy depiction of reality. In order to determine whether or not the model is trustworthy, the model needs to be verified. This section will provide a verification of the temperature dependent model by first studying the spatial convergence, then the temporal convergence, and finally looking at a comparison between the analytical and numerical solution of the velocity profile.

5.1. Spatial convergence

Once the OpenFOAM model was made it could be tested. For multiple successive grid refinements the temperature profile has been plotted as can be seen in Figure 5.1. Just by eyeballing it can be seen that convergence seems to be reached for the 40 cells case, since almost no change takes place anymore.

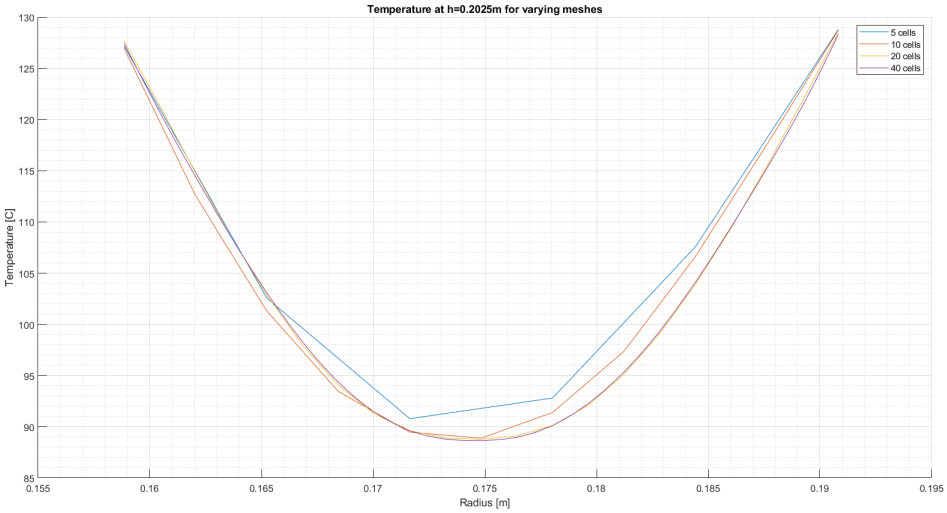


Figure 5.1: Grid qualities of 5, 10, 20, and 40 cells in the radial direction of the cylinder

To quantify the grid convergence, the method advocated by Roache [29] is used. This method makes use of a grid-convergence index (GCI), which is based on the generalized theory of Richardson Extrapolation [30].

Richardson Extrapolation states that the discrete solutions f are assumed to have a series representation based on the grid spacing h.

$$f = f[exact] + g_1 h + g_2 h^2 + g_3 h^3 + ... \tag{5.1}$$

Where the functions g_1, g_2, g_3, etc. are defined in the continuum and do not depend on any discretisation.

Roaches method may be used in one of two ways; either a fine-grid Richardson error estimator can be used to approximate the error in a fine-grid solution by comparing the solution of the fine grid to the coarse grid, where the fine-grid Richardson error estimator is defined as

$$E_1^{fine} = \frac{\epsilon}{1 - r^p} \tag{5.2}$$

Or a coarse-grid Richardson error estimator can be used to approximate the error in a coarse-grid solution by comparing the solution of the coarse grid to the fine grid, where the coarse-grid Richardson error estimator is defined as

$$E_2^{coarse} = \frac{r^p \epsilon}{1 - r^p} \tag{5.3}$$

where

- $\epsilon = f_2$ - f_1,
- $f_2 =$ the coarse-grid numerical solution for the grid spacing h_2,
- $f_1 =$ the fine-grid numerical solution for the grid spacing h_1,
- $r =$ the refinement factor between the coarse and fine grid ($r = \frac{h_2}{h_1} > 1$)
- $p =$ formal order of accuracy of the algorithm (which can be calculated according to Equation 5.4)

For this case the fine-grid and coarse-grid Richardson error estimators will both be used to estimate the numerical errors introduced in the temperature profile by the chosen grid. The estimator will be applied on the temperature data. Apart from the standard grid used for the simulations which has 20 cells in x-direction and 80 cells in y-direction as described in Section 4.5, two other grids will be created, for which a refinement factor, r, of 2 will be used. The coarser grid, which has 10 cells in x-direction and 40 cells in y-direction, will be used to calculate E_1^{fine}. The finer grid, which has 40 cells in x-direction and 160 cells in y-direction, will be used to calculate E_2^{coarse}. The temperature data for these three cases has been produced using the same time step of 0.00001s. This was necessary for the finest grid simulation, since larger time steps resulted in an erroneous velocity profile.

In Roache's study it is stated that if the grid refinement is performed with constant r, then the order of accuracy of the algorithm, p, can be calculated directly by using the three grid solutions in combination with Equation 5.4.

$$p = ln\left(\frac{f_3 - f_2}{f_2 - f_1}\right) / ln(r) \tag{5.4}$$

where subscript 1 indicates the finest grid.

Knowing the Error Estimator, E_1^{fine}, the GCI can now be calculated according to either Equation 5.5 or Equation 5.6.

$$GCI_1^{fine} = F_s |E_1^{fine}| \tag{5.5}$$

$$GCI_2^{coarse} = F_s |E_2^{coarse}| \tag{5.6}$$

where F_s is a safety factor.

A safety factor of 1.25 is recommended by Roache [31]. According to the study by Schwer [32] this safety factor "should be thought of as representing a 95% confidence bound on the estimated relative error".

Using the calculated GCI the relative GCI (RGCI) can be calculated, for both fine and coarse grid, using Equation 5.7.

$$RGCI = \frac{GCI}{f_1} * 100\% \tag{5.7}$$

where f_1 is the result of the grid that needs to be verified. The GCI gives a value for the error band, while the RGCI gives an indication of how big this band is relatively.

The three different grids have 10 coinciding points. At 5 of these points the calculations for the RGCI will be made. The x-coordinates of these 5 points are shown in Figure 5.2.

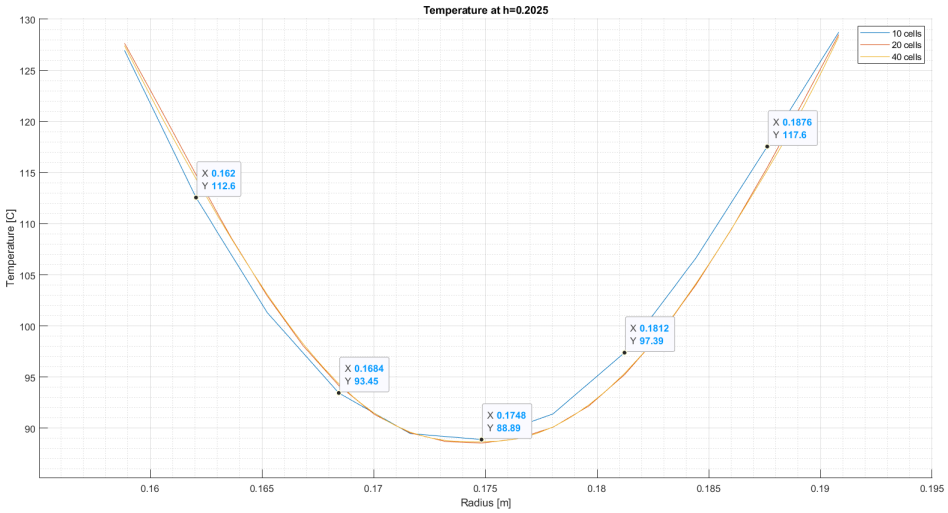


Figure 5.2: Overview of the 5 x-coordinates at which the calculations for the RGCI were made

At these points the values of p, E_1^{fine}, E_2^{coarse}, GCI_1^{fine}, GCI_2^{coarse}, $RGCI_1^{fine}$, and $RGCI_2^{coarse}$ have all been calculated and the final results together with the average values can be found in Figure 5.3. The RGCI values show that the calculated temperature differs on average only 0.45% from the result that would have been found for an infinite amount of cells, where a 95% confidence bound is used. Note that the RGCI fine and RGCI coarse in Figure 5.3 result in exactly the same value, so it does not matter which one of the two is used.

x-coordinate [m]	P (order or accuracy)	RGCI fine [%]	RGCI coarse [%]
0,1620	2,45943	0,532327	0,532327
0,1684	1,14684	0,678468	0,678468
0,1748	-0,48543	0,492736	0,492736
0,1812	3,36075	0,320253	0,320253
0,1876	3,45943	0,238302	0,238302
Average	1,98820	0,452417	0,452417

Figure 5.3: Results grid verification

Looking at Figure 5.3 the negative -p-value of x-coordinate 0,1748 might seem strange. At this coordinate the 3 graphs are very close to each other with the distance between the 10 and 20 cell graphs being just a fraction smaller than the distance between the 20 cell and 40 cell graphs. This causes Equation 5.4 to result in a negative value for p, because the graphs seem to diverge instead of converge when the grid is refined. Therefore it should be noted that this equation can be useful when the distances between the graphs are relatively large and become smaller when the grid quality is increased, however, if the graphs are already very close to each other and little change takes place between successive grid refinements, the results from a grid convergence study like this become less useful. If the graphs stay the same while the grid is further refined it can be concluded that convergence has already been reached.

5.2. Temporal convergence

Just as the grid size can be refined to check for convergence, one can also refine the time step to check for convergence. The temperature profile has been plotted for 4 different time steps: 0.001s, 0.0001s, 0.00001s, and 0.000001s.

The temperature data produced at t=900s using these different time steps has been plotted in Figure 5.4, where the grid with 20 cells in the x-direction has been used. The figure seems to show only 2 graphs instead of 4, but if one looks closer it can be seen that the graphs for 0.0001s, 0.00001s, and 0.000001s are almost plotted on top of each other. This seems to indicate that convergence has been reached at time step 0.0001s, since further time step refinements do not seem to result in further convergence. Performing a grid convergence study will be of little added value since the differences between the 3 finest time steps are practically nonexistent.

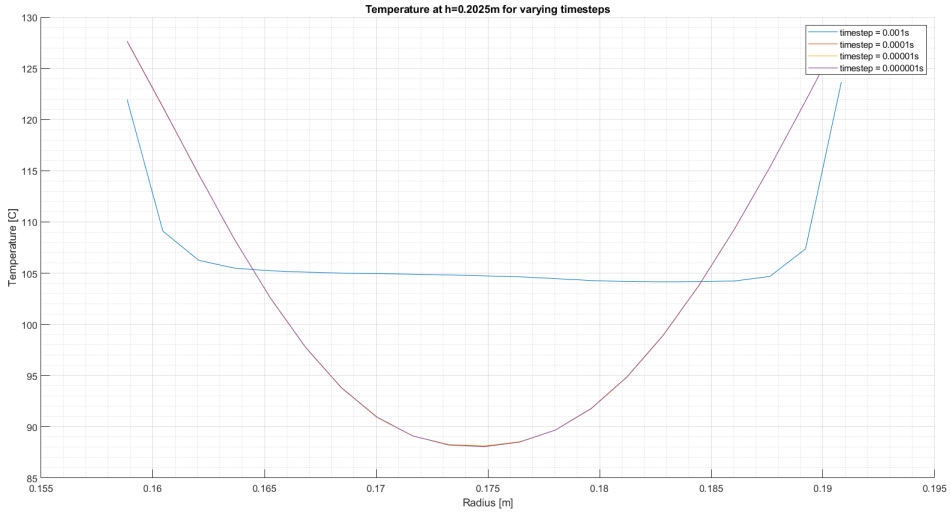


Figure 5.4: The temperature profiles for different time steps at h=0.2025

Apart from the temperature, other parameters can also be used to study the convergence. For this case there are 3 other parameters which can be used: velocity, viscous dissipation, and viscosity. Figure 5.5 compares the velocity profiles for each time step. The graph for the coarsest time step of 0.001s, the blue line, shows an almost linear velocity profile. This graph is in conflict with the expected velocity profile, as described by Equation 2.16. The velocity profile is expected to show a large initial drop off in velocity near the rotating inner cylinder, which should become steeper for smaller values of the shear thinning index n. After the initial drop off in velocity, the drop off should become more and more gradual as the x-coordinate approaches the outer cylinder. It can be seen that as the time steps are refined, the velocity profile keeps developing towards the more pronounced power law form for which is expected for a low value of n.

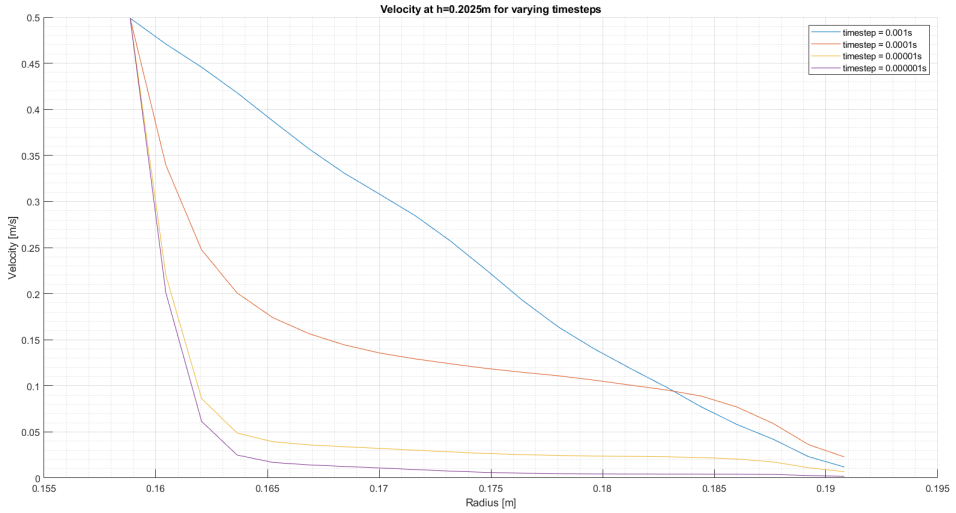


Figure 5.5: The velocity profiles for different time steps at h=0.2025

A convergence study has been performed on the velocity using the same approach as described before. For this study the graphs of the 3 finest time steps have been used. Calculating the RGCI for the time step of 0.0001s is the most valuable, since this time step has been used in most of the simulations and therefore an indication of the errors is very useful.

Figure 5.6 shows the velocity values that have been used in the RGCI calculation, which have been taken at the same x-coordinates as in the previous convergence study in Section 5.1.

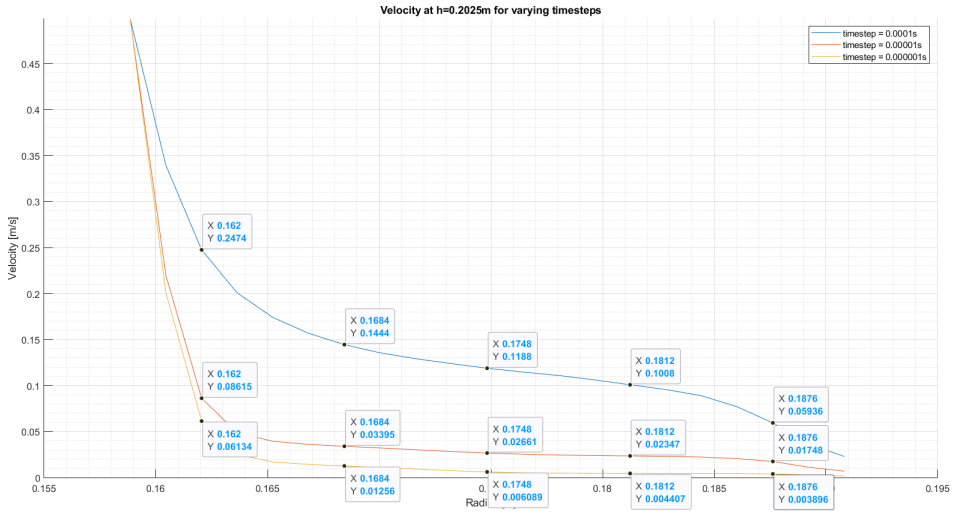


Figure 5.6: Overview of the x-coordinates and the velocities used in the calculations for the RGCI

The results for the temporal convergence can be found in Figure 5.7. Note that a refinement factor of $r = 10$ has been used. The p-values show an average order of accuracy of 0,65509. These low p-values in combination with the large refinement factor result in a large RGCI of 119,49% on average. Based on the RGCI results it should be clear that preferably a more refined time step is used in the simulations, but due to the characteristics of the simulation this was impossible. The reason for not using a smaller time step is that the simulation has a duration of 30 minutes, which equates to 1800s. This means a total of $1800/0,0001 = 18.000.000$ steps have to be calculated during a single run. Apart from the large

amount of calculations there was another factor which influenced the calculation time for the simulations. The customized solver performs iterations until the convergence criteria for the pressure, temperature, and velocity are met. The convergence of the pressure term often required up to 100 or 200 iterations per time step. Even with access to the computer cluster at the TU Delft the combination of this large amount of time steps and iterations resulted in extremely long simulation times. Multiple test runs with a time step of 0.00001s have been performed. Running the simulation for 24 hours on the cluster produced roughly 10 to 20 seconds of simulation time. Based on the fact that the entire simulation has a duration of 1800 seconds this would require roughly $1800/15 = 120$ days to complete a single simulation. In contrast, the simulations using a time step of 0.0001s would require "only" 7 days.

The velocity profiles produced in the simulations using a time step of 0.0001s are less accurate than one would prefer, but it is currently the only way to proceed with the simulations.

x-coordinate [m]	P (order or accuracy)	RGCI coarse [%]
0,1620	0,81287	96,287
0,1684	0,71295	118,57
0,1748	0,65249	124,78
0,1812	0,60816	127,27
0,1876	0,48898	130,53
Average	0,65509	119,49

Figure 5.7: Results time step verification

A visual comparison of the viscous dissipation graphs calculated using the different time steps is shown in Figure 5.8. It can be seen that the graph for the biggest time step shows dissipation peaks throughout the entire cross section, which are caused either by a high shear rate, a high viscosity, or a combination of the two. As the time step is refined these peaks disappear, except for the one near the inner cylinder. The graphs of the two most refined time steps are almost converged and only deviate slightly from each other near the inner and outer cylinder. The graph clearly shows that in general the viscous dissipation is of no importance, and that it only plays a minor role near the inner cylinder. It can be seen that the viscous dissipation field converges slower than the temperature field as the time steps are refined. This is due to the fact that the viscous dissipation term contains a derivative. If the grid is refined by a factor 2, the error of the temperature field reduces by a factor 4. For a derivative of the temperature the error only reduces by a factor 2 instead of 4, thus requiring a more refined time step to reach convergence.

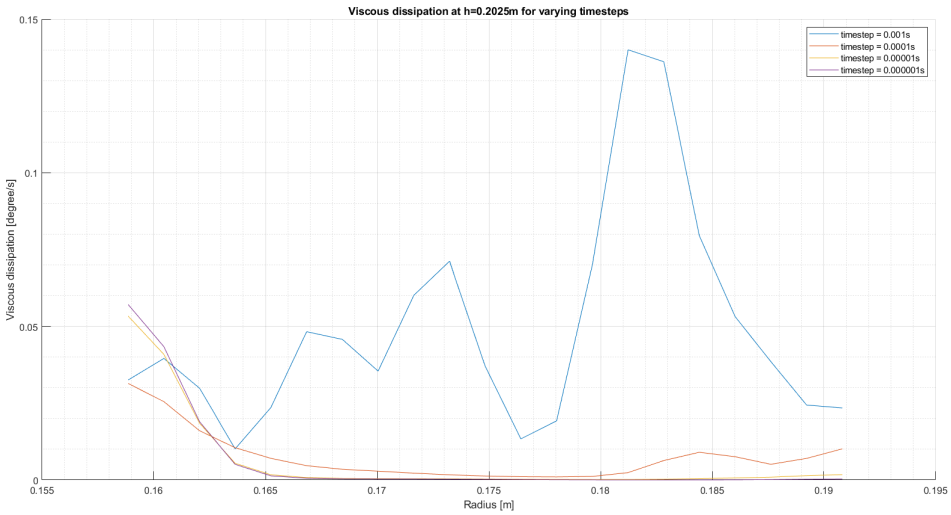


Figure 5.8: The viscous dissipation profiles for different time steps at h=0.2025

Finally, comparing the viscosity profiles for the different time steps it can be seen that as the time steps get more refined, the viscosity profiles seem to increase more and more, which results in divergence instead of convergence. The large increase in viscosity with the time step refinements is caused by the fact that the velocity profile shown in Figure 5.5 becomes almost horizontal for the small time steps. This means that a shear rate of 0 is approached, which according to the power law would result in an infinite viscosity. For these simulations the maximum allowable viscosity is given a value of 17 m^2/s, to keep the model from crashing due to divergence. This specific value was used because it was the maximum value that came out of the measurements in Chapter 3.

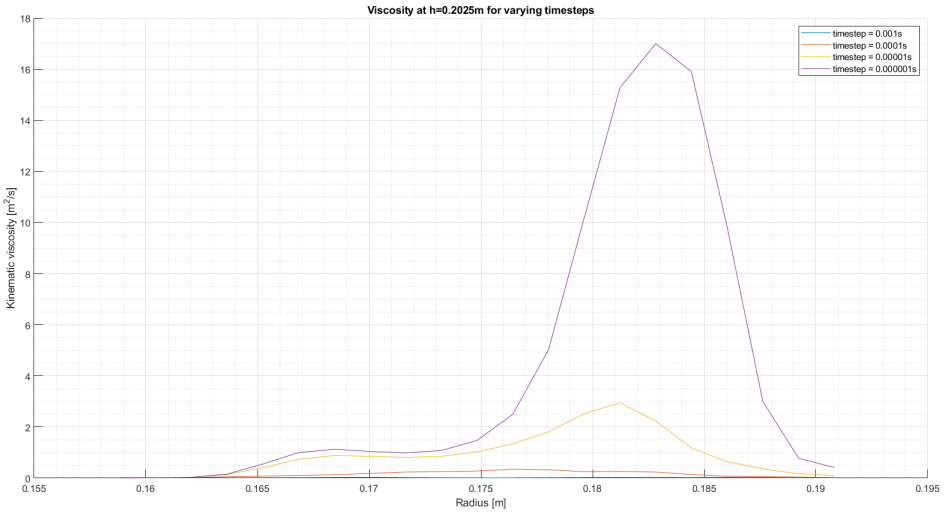


Figure 5.9: The viscosity profiles for different time steps at h=0.2025

5.3. Comparison between the analytical and numerical solution

Using Matlab the numerical solution of the azimuthal velocity for the case with constant power law parameters, $K = 1600Pa * s^n$ and $n = 0.13$, and a refined grid with 40 cells in x-direction combined with a time step of 0.00001s has been plotted versus the analytical solution. The result can be found in Figure 5.10. The analytical solution for the velocity has been calculated using Equation 2.16 from Section 2.4. The two solutions are close to each other, but it can be seen that there is some deviation. This deviation is most likely caused by the the fact that the velocity profile is not yet fully converged for the time step 0.00001s, as shown in Section 5.2.

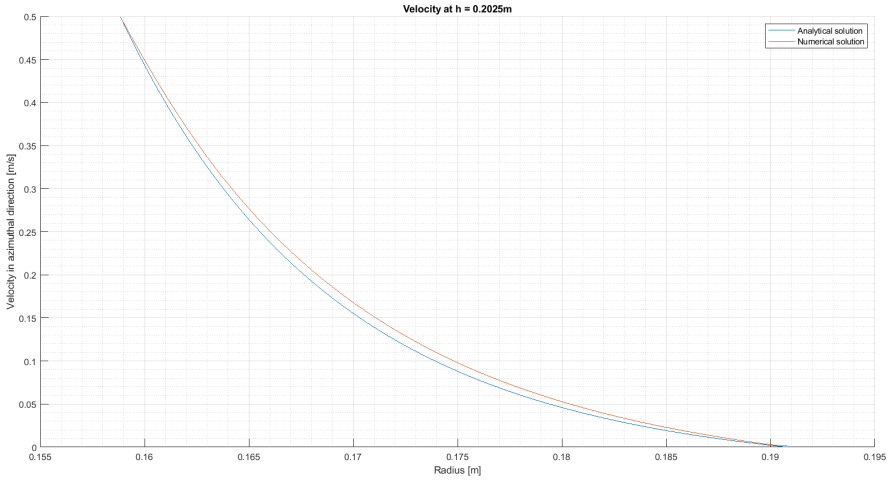


Figure 5.10: Analytical vs numerical solution of the azimuthal velocity between the inner and outer cylinder

6

Results

The results will be discussed in this chapter. For each simulation data for the temperature, velocity, viscosity, and viscous dissipation is available at every second of the 1800 seconds simulated per case. The most relevant data will be presented in the following sections. The results for the base case will first be discussed, for both constant and interpolated power law parameters. After this, the results for the variations on the base case will be discussed.

6.1. The base case

Figure 6.1 shows the temperature profile after 1800s of simulation for the case with constant power law parameters on the left, and interpolated parameters on the right.

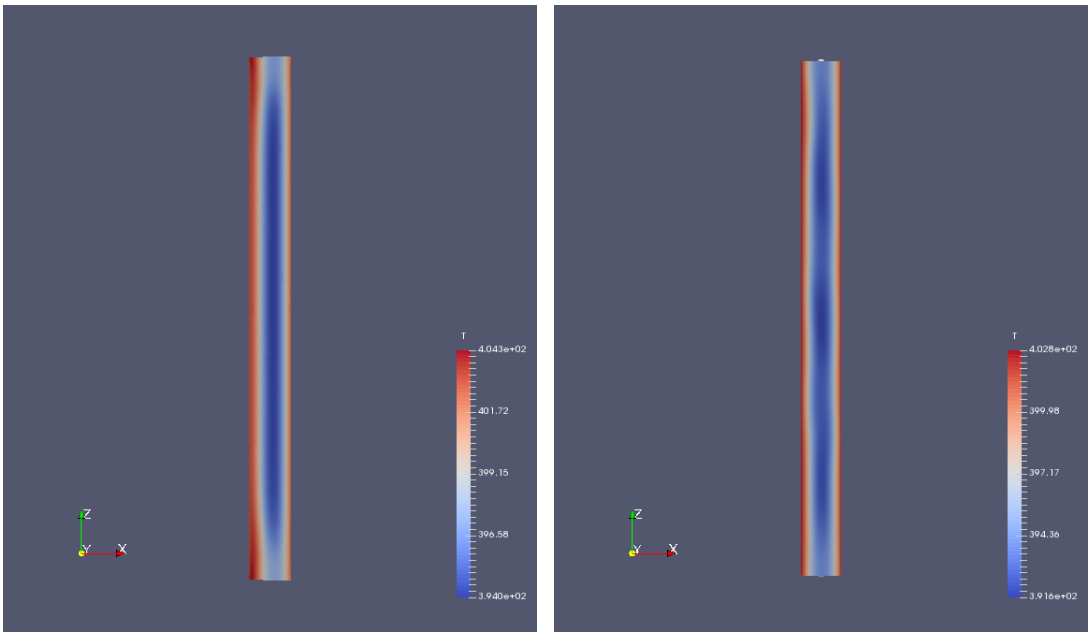


Figure 6.1: Temperature profile after 1800s. The left image shows the case with constant K and n values and the right image shows the case with interpolated values. The legends show temperature in degrees Kelvin.

When comparing the temperature profiles it is important to notice that at this moment all the temperatures fall in the range 390-405°K, or 117-132°C, as shown in the legend. This relatively small range for the legend makes sure that differences in temperature are easy to see. The main difference between the two figures is that the one on the left shows a large vertically elongated cold spot halfway in the middle of the slab in x-direction, while the one on the right shows 3 vertically separated cold spots. It might still

be somewhat hard to see the 3 separated cold spots solely based on the color differences, therefore a graph has been included in Figure 6.2 which shows the temperature along the z-axis in the middle of the slab. Here a temperature fluctuation of roughly 1°K can be seen in between the cold spots. Even though this temperature fluctuation seems minor, the mechanics behind it are of interest. This fluctuation in the temperature profile could be caused by any of the terms in the temperature equation, so either the viscous dissipation, diffusion, or advection term is responsible.

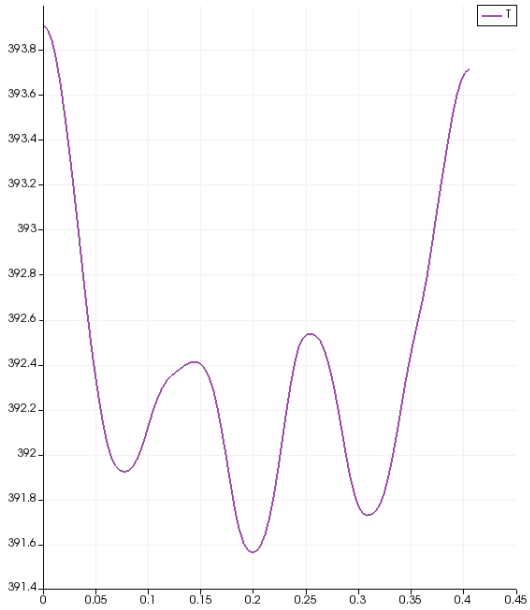


Figure 6.2: The temperature along the z-axis halfway through the slab at time t=1800s in Kelvin.

Viscous dissipation

It can easily be shown that the viscous dissipation is not responsible for this effect by using the plot of the viscous dissipation over the entire geometry, see Figure 6.3. The left wall represents the moving cylinder. The viscous dissipation is highest on this side of the geometry, as expected. In the temperature profiles from Figure 6.1 the areas in between the cold spots show slightly elevated temperatures. Looking at the same areas in the viscous dissipation profiles no elevated dissipation is found and therefore no extra heat is added to the system in these areas. Note that the maximum viscous dissipation values are found in the high shear regions in the left upper and lower vertex. This is where the stationary upper and lower plates are adjacent to the rotating inner cylinder, which causes a high shear rate.

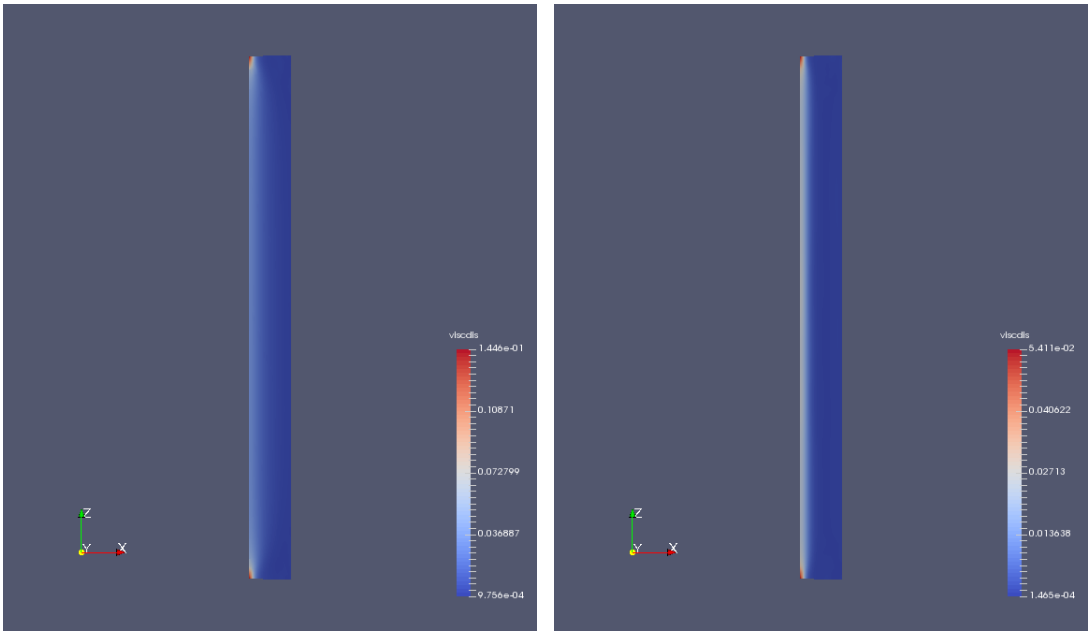


Figure 6.3: Viscous dissipation profile after 1800s. The left image shows the case with constant K and n and the right image shows the case with interpolated values. The values in the legends are given in degrees/s.

Diffusion

The driving force for most of the heat diffusion in the system is the heat that is supplied by the steam at the inner and outer cylinders. This heat diffuses into the system at roughly the same rate along the entire height of the cylinder. Only near left upper and lower vertex, where the viscous dissipation is highest, the diffusion is somewhat higher. Based on the fact that the heat diffuses in an evenly distributed manner from the inner and outer cylinder towards the middle of the slab, the diffusion cannot be the thermal transport mechanism which is responsible for the slightly elevated temperatures in between the 3 cold spots.

Advection

The transport of heat by movement of bulk fluid is contained in the advection term of the temperature equation. Since the diffusion and viscous dissipation terms do not seem to cause the elevated temperatures between the cold spots, the advection term must be responsible. To verify this, the velocity profiles can be used. The velocity profile in the y-direction (azimuthal direction) has already been reviewed during the comparisons between the analytical and numerical solution for the case with constant K and n. For this case with temperature dependent power law parameters the velocity profile has a steeper initial decline in velocity than before due to the low value of the shear thinning index n, but other than that the system behaves as expected, which can be seen in Figure 6.4 where the velocity over the entire geometry has been plotted.

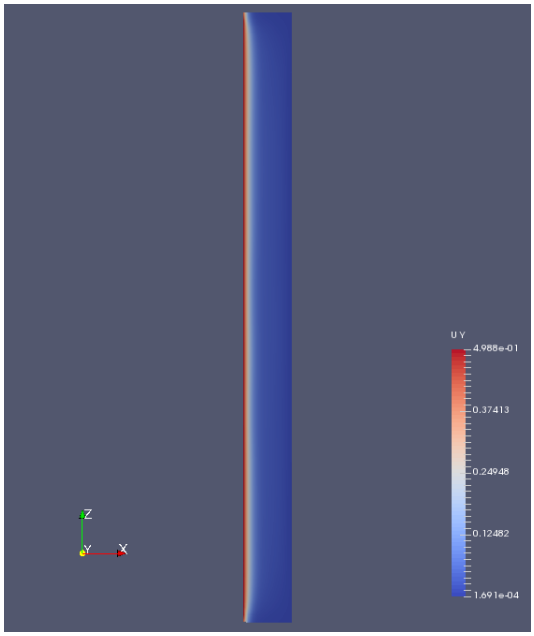


Figure 6.4: The velocity in m/s in y-direction at time step t=1800s.

The velocity profiles of interest are the ones in the x- and z-direction, which are shown in Figure 6.5. In the left image the red areas indicate a flow in positive x-axis direction and the blue areas indicate a flow in negative x-axis direction. In the right image idem dito, but for the z-axis direction. If one would combine the velocity components from both the left and the right picture this would result in circular flows, vortices, around the locations where the 3 cold spots were found. It is important to pay attention to the magnitude of the velocity components. Where the velocity in y-direction has a maximum value of 0.5 m/s, the velocity components in the x- and z-direction have a maximum value of 6.6e-03 m/s and 9.8e-03 m/s respectively, as can be seen in the legends.

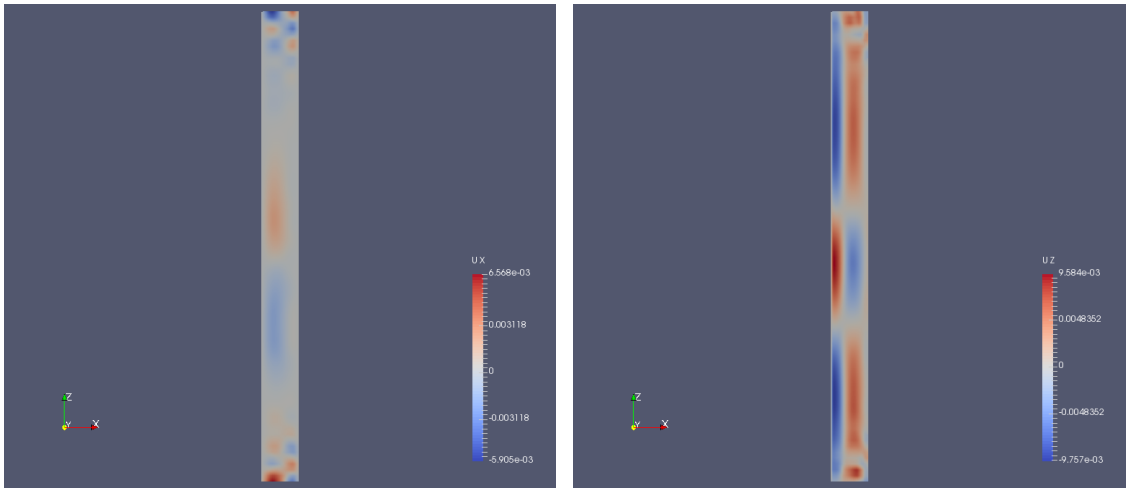


Figure 6.5: The velocity in m/s in x-direction (left) and z-direction (right) at time step t=1800s for the case with interpolated K and n.

The low velocity vortex-like flow encountered in the velocity profiles for the x- and z-direction was not expected and efforts have been made to explain where these vortices come from. When vortices are encountered in a Taylor-Couette cell, it seems logical to assume that these are Taylor vortices, see Figure 6.6.

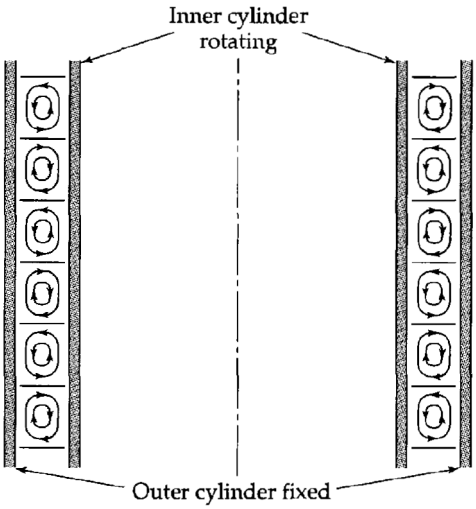


Figure 6.6: Counter-rotating toroidal vortices, called Taylor vortices, observed in the annular space between two cylinders [33].

Taylor vortices are secondary flow patterns consisting of toroidal vortices which can develop in a Taylor-Couette cell once a critical Taylor number, Ta_c, is reached. The Taylor number is the dimensionless number which gives the relation between the inertial forces and the viscous forces. As long as the Taylor number is below the critical value, the viscous forces are able to dampen out any instabilities and therefore the flow remains steady. The Taylor number for Taylor-Couette flow with a rotating inner cylinder and a stationary outer cylinder can be calculated using Equation 6.1 [34].

$$Ta = \frac{\Omega^2 R_1 (R_2 - R_1)^3}{\nu^2} \tag{6.1}$$

where Ω is the angular velocity in rad/s, R_1 is the inner radius, R_2 is the outer radius, and ν is the kinematic viscosity in m^2/s.

For Newtonian fluids the critical Taylor number is roughly 1700 [35]. For non-Newtonian fluids the critical Taylor number is not as well known. In a study by Sinevic et al. [36] it was found that the critical Taylor number of non-Newtonian fluids depends strongly on the gap width and the shear thinning index. For most of the cases the critical Taylor number has been found to be greater than in the Newtonian cases, but it approaches the Newtonian value as n tends to 1.

To verify if the existence of Taylor vortices in this model is theoretically possible, the Taylor number can be calculated, for which the kinematic viscosity is required. The lowest value of the kinematic viscosity will be used to check if the Taylor number has a value higher than the critical Taylor number of 1700, because this should result in the highest possible Taylor number. Figure 6.7 shows that the lowest kinematic viscosity at t=1800s is equal to 5,973e-03 m^2/s.

The Taylor number can then be calculated as shown in Equation 6.2.

$$Ta = \frac{\pi^2 * 0,159(0,191 - 0,159)^3}{(5,973 * 10^{-3})^2} = 1,44 \tag{6.2}$$

The Taylor number 1,44 is nowhere near the critical value of 1700. This means the existence of the Taylor vortices in the model seems unlikely and thus there should be a different explanation for the vortex-like flow.

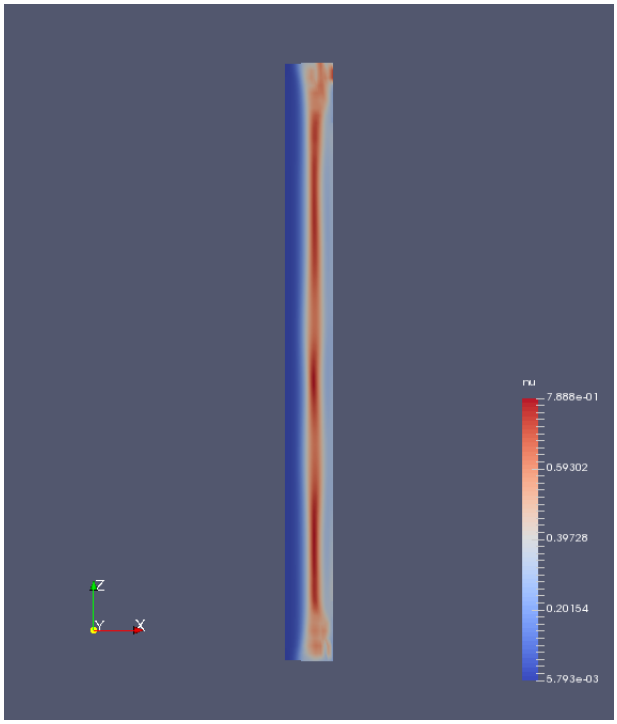


Figure 6.7: The kinematic viscosity in m^2/s at t=1800s.

To find out what causes the vortex like flow, different test runs have been performed. This included experimenting with an iterative start-up phase of the simulation for the values of K and n and changing the no-slip boundary conditions at the top and bottom wall to slip boundary conditions. None of these tests resulted in a velocity profile without the vortices.

Since no physical cause for the vortices could be found, numerical artifacts were investigated. Additional tests were performed, which included a 1 second long simulation using a time step of 0.000001s, which did not show any of the vortex-like flow.

The set-up for this simulation which showed no vortex-like flow was as follows: First a basic simulation was run for 1800 seconds using a time step of 0.0001s. The final time folder produced by this simulation, "1800", which contains all information regarding the fields (velocity, temperature, etc.) was then used as initial condition folder for a new simulation which used a time step of 0.000001s. After running this new simulation for several time steps the vortices started decaying rapidly. After 0.005s the maximum velocity in x- and z-direction was reduced by 3 to 4 orders of magnitude and therefore negligible as can be seen in Figure 6.8. Thus a further refinement in the time step seems to resolve the unexplained vortex-like flow. Ideally further research would be performed to find out why the simulation shows this vortex-like flow when larger time-steps are used, but this does not fit in the time span of the current study.

The problem remains that it is currently practically impossible to perform the full duration simulations due to this time step. Simulating a single second already requires 1.000.000 calculations, which took several hours on the cluster. Therefore a choice had to be made between the accuracy and the time required to run the simulation. For the simulations performed in this study a time step of 0.0001s was used because this allowed a single simulation to be run in roughly a week time. If a finer time step of 0.00001s is used the simulation would take roughly 120 days. In order to run the simulation without the vortex-like flow, the time step would have to be 0.000001s, which would take much longer than 120 days to complete.

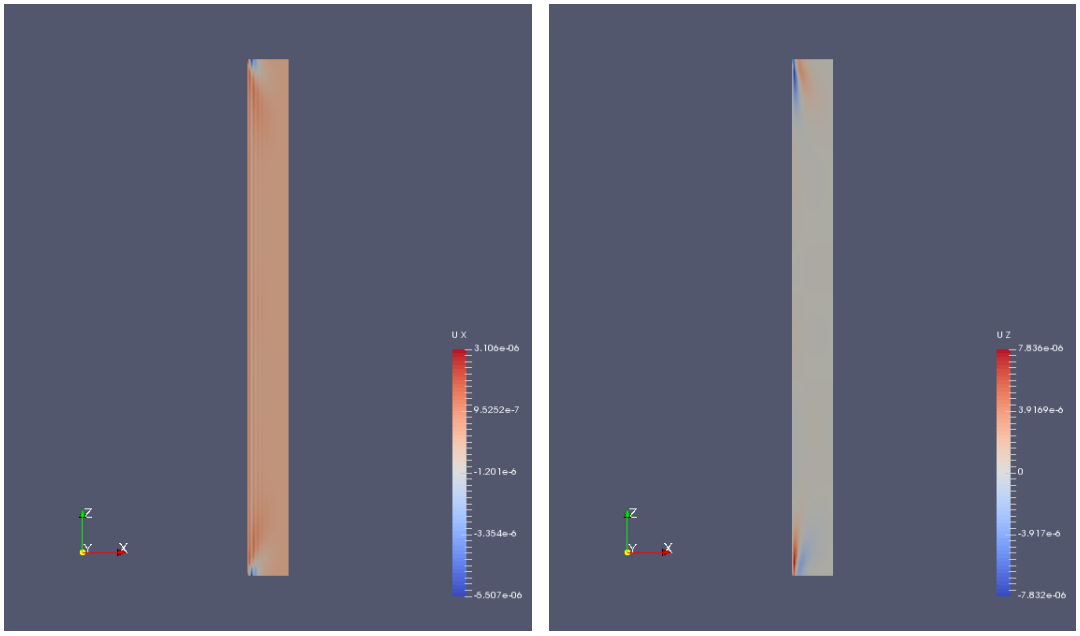


Figure 6.8: The velocity in m/s in x-direction (left) and z-direction (right).

6.2. Case comparisons

In this section the comparisons between the base case and the variations will be discussed. As explained in subsection 4.8.2 multiple variations have been made for the slab thickness, preheat temperature, and density. Additionally a simulation without viscous dissipation has also been performed, which will be discussed first.

6.2.1. Viscous dissipation

The effect of viscous dissipation on the temperature field can be quantified. To do so, the results of a simulation where the viscous dissipation term is included in the temperature equation need to be compared to the results of a simulation where the viscous dissipation term is not included in the temperature equation. This comparison has been made for 2 scenarios. The first scenario uses the constant power law parameters $K = 1600Pa * s^n$ and $n = 0.13$, while the second scenario uses the interpolated power law parameters.

The results for the comparison with constant power law parameters are shown in Figure 6.9. The dashed and solid lines show the temperature profiles with and without viscous dissipation respectively. By subtracting the temperature graphs for the case without viscous dissipation from the case with viscous dissipation, the temperature difference caused by viscous dissipation can be better visualized, see Figure 6.10. At the start of the process the difference between the two is of course very small, but as time goes on the influence of the viscous dissipation increases. At the end of the simulation the maximum temperature difference is 9.8°C at the radius 0.17m. The results for the comparison with interpolated power law parameters are shown in Figure 6.11 and the temperature difference caused by viscous dissipation is shown in Figure 6.12. The maximum temperature difference at the end of this simulation is located around the radius 0.165m and is only 2.6°C.

By comparing the two scenarios it becomes clear that the viscous dissipation plays a much smaller role in the scenario with interpolated values. It is important to keep in mind that these temperature profiles were calculated using the time step 0.0001s. If the time step is further refined the viscous dissipation field changes, as previously shown in Figure 5.8. Revisiting this figure it can be seen that for the time step 0.0001s the viscous dissipation has a relatively high value near the inner cylinder and a lower value throughout the rest of the geometry. This results in only a small amount of heat generation throughout the entire geometry, resulting in the temperature difference of 2.6°C for the interpolated power law value scenario. Looking at the time step 0.00001 it can be seen that the viscous dissipation has a higher value near the inner cylinder, but throughout the rest of the geometry it is practically equal to zero and therefore almost no heat is generated by viscous dissipation in those areas.

Since the simulations in this study have been performed with a time step of 0.0001s this means that the temperature shown in the data will be slightly higher than it would be for the more refined time steps, where the viscous dissipation profile is fully converged. However, this deviation has now been quantified by showing that the temperature fields for simulations with interpolated power law values and a 0.0001s time step show a temperature field that is at most 2.6°C higher than it would be for the more refined time steps, which is a relatively small difference.

The viscous dissipation for the scenario with constant power law values is expected to diminish in a similar manner as the viscous dissipation for the interpolated scenario did as the time steps got more refined. Only in the area near the inner cylinder a slight elevation in viscous dissipation is expected. This means the relatively large temperature difference between the graphs with and without viscous dissipation in Figure 6.9 should disappear if a time step of 0.00001 would be used, leaving only a small temperature difference near the inner cylinder. However, since no convergence study has been performed for the constant power law parameter scenario, this can only be speculated based on the experience with the interpolated scenario.

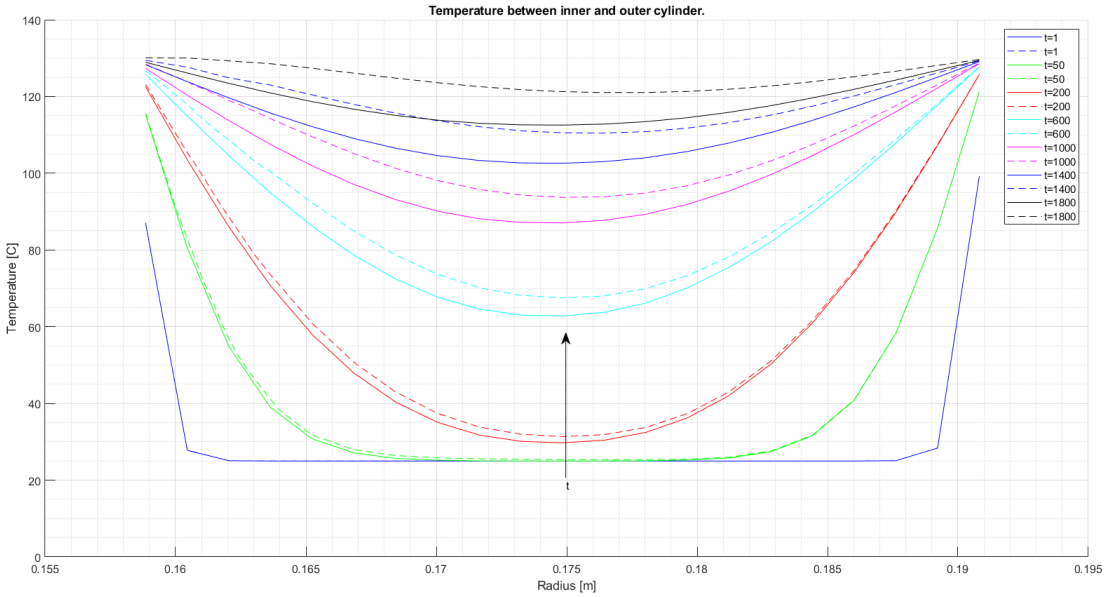


Figure 6.9: A comparison between the temperature profiles with and without viscous dissipation for the case with constant power law parameters. Solid lines are used for the case where viscous dissipation is included in the calculations and dashed lines for the case where it is excluded. The temperature profiles are given at multiple moments in time. The graph for t=1s starts at the bottom of the figure and the graphs shift upwards as the time increases.

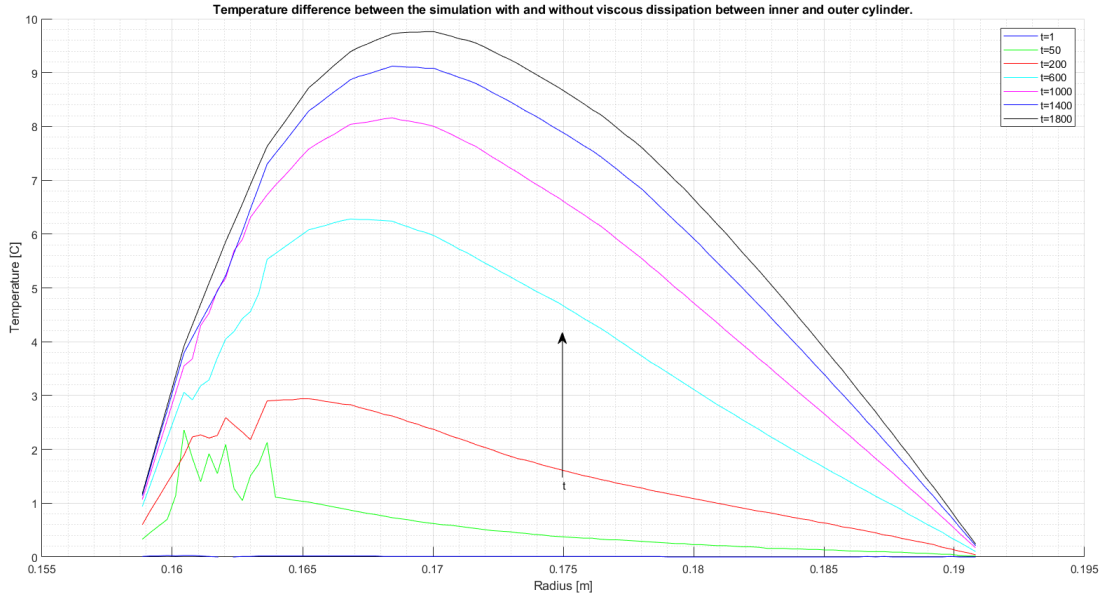


Figure 6.10: The temperature difference between the case with and without viscous dissipation. The temperature profiles are given at multiple moments in time. The graph for t=1s starts at the bottom of the figure and the graphs shift upwards as the time increases.

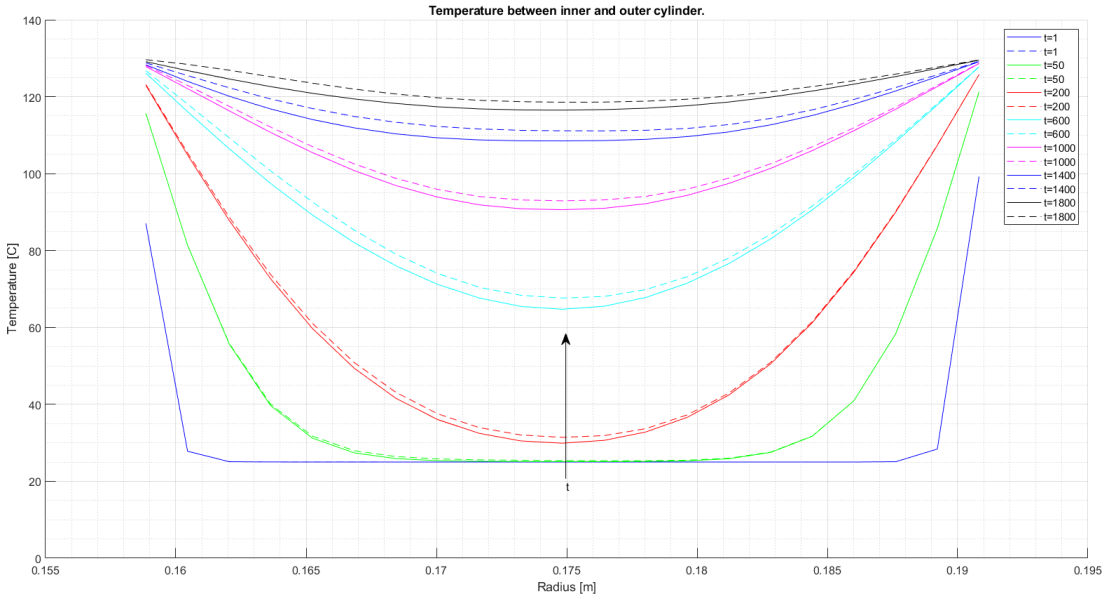


Figure 6.11: A comparison between the temperature profiles with and without viscous dissipation for the case with interpolated power law parameters. Solid lines are used for the case where viscous dissipation is included in the calculations and dashed lines for the case where it is excluded. The temperature profiles are given at multiple moments in time. The graph for t=1s starts at the bottom of the figure and the graphs shift upwards as the time increases.

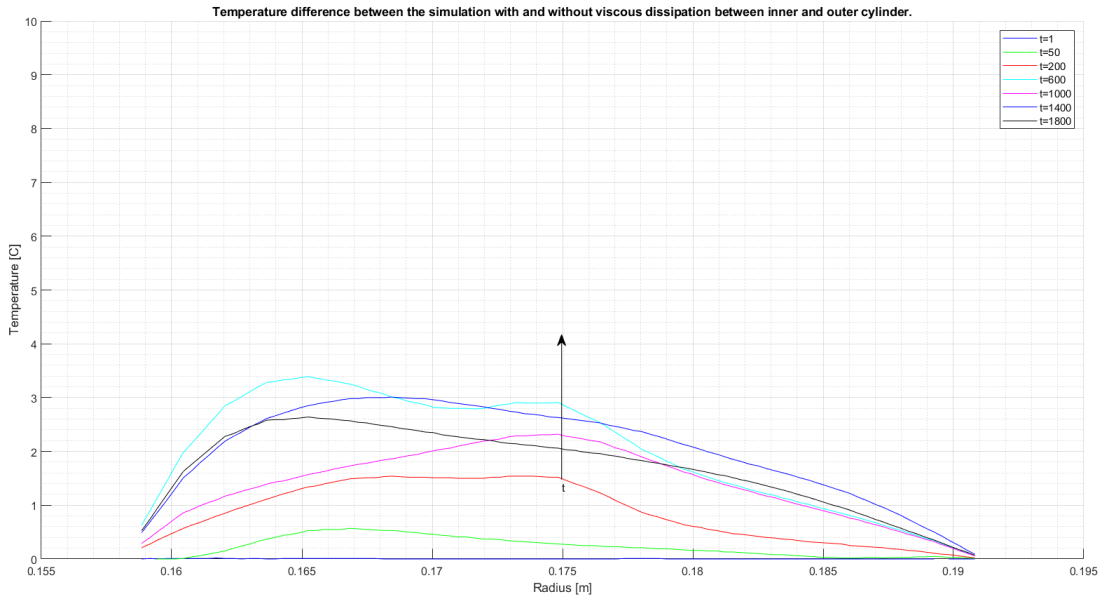


Figure 6.12: The temperature difference between the case with and without viscous dissipation. The temperature profiles are given at multiple moments in time. The graph for t=1s starts at the bottom of the figure and the graphs shift upwards as the time increases.

6.2.2. Geometry

The effect of the geometry on the temperature field has been quantified by performing simulations for 3 different cases. The results for the simulations with constant and interpolated power law values are shown in Figure 6.13 and Figure 6.14 respectively. These figures contain graphs for the 23mm, 32mm, and 41mm geometry. Solid lines are used for the 23mm case, dashed lines for the 32mm case, and dotted lines for the 41mm case. For each case the temperature profiles are given at multiple moments in time as indicated in the legend. The results for the constant power law value simulations will be discussed first. Figure 6.13 shows the highest temperatures are attained in the 23mm case. A strange effect that occurs in this case is the fact that the temperature inside the dough reaches a value of 136°C after 1800s, which is higher than the maximum value of 130°C imposed on the inner and outer cylinders as the temperature boundary condition. This rise in temperature is caused by the viscous dissipation, which as shown in Figure 6.9 adds additional heat to the system. For the 32mm case the minimum temperature is 121°C at the end of the simulation. For the 41mm case the minimum temperature is 100.5°C.

In Figure 6.14, which shows the results for the interpolated power law parameters, it can be seen that for the 23mm case a maximum temperature of 131°C is reached , which again is higher than the temperature at the boundaries. The 32mm case has a minimum of roughly 118.5°C and the 41mm case has a minimum of roughly 99.5°C.

The viscous dissipation which is included in these simulations is expected to diminish once a smaller time step is used in the simulations. For the constant and interpolated power law cases the final temperatures are expected to give an overestimation of roughly 9.8°C and 2.6°C respectively.

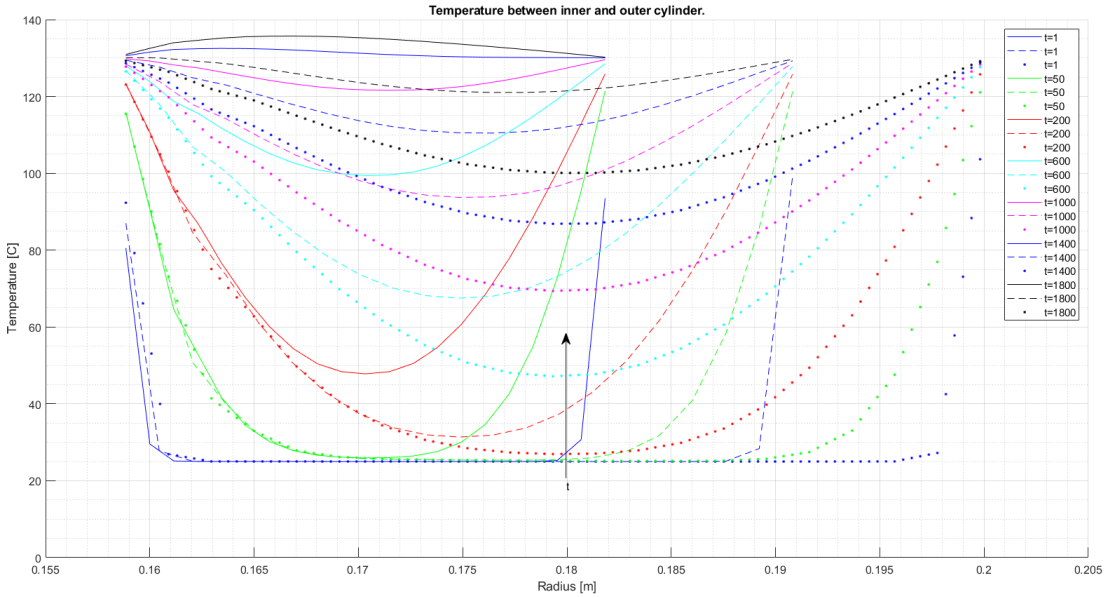


Figure 6.13: A comparison between the temperature profiles for 3 different geometries for the case with constant power law parameters. Solid lines are used for the 23mm case, dashed lines for the 32mm case, and dotted lines for the 41mm case. The temperature profiles are given at multiple moments in time. The graph for t=1s starts at the bottom of the figure and the graphs shift upwards as the time increases.

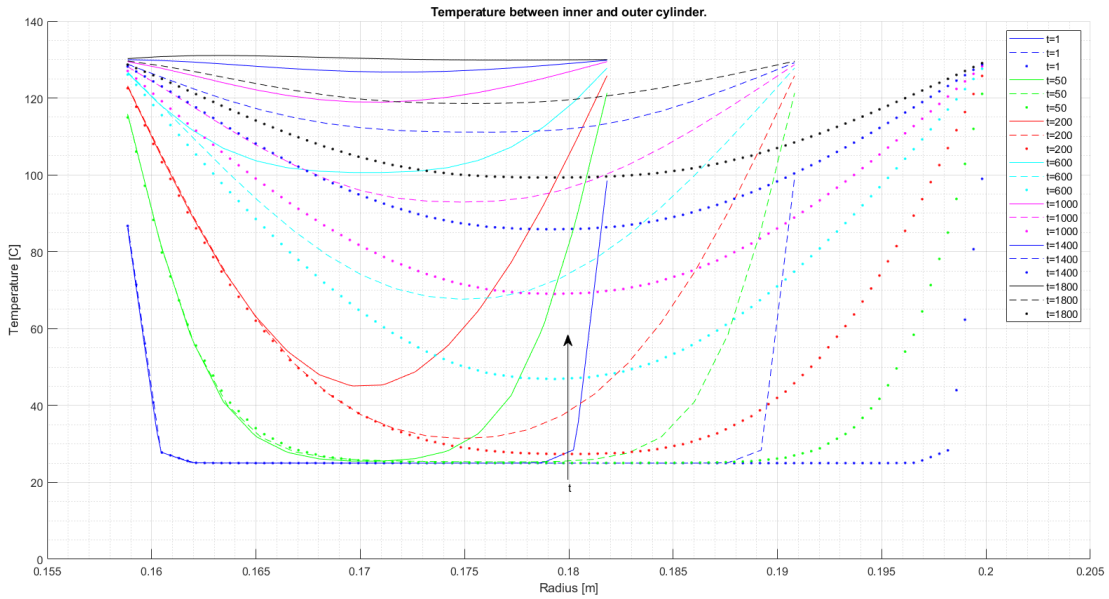


Figure 6.14: A comparison between the temperature profiles for 3 different geometries for the case with interpolated power law parameters. Solid lines are used for the 23mm case, dashed lines for the 32mm case, and dotted lines for the 41mm case. The temperature profiles are given at multiple moments in time. The graph for t=1s starts at the bottom of the figure and the graphs shift upwards as the time increases.

6.2.3. Preheat

The effect of the preheat temperature has been studied as well. Three different starting temperatures have been used; 10°C, 25°C, and 50°C. The results for the case with constant and interpolated power law parameters are shown in Figure 6.15 and Figure 6.16 respectively, where solid lines have been used for the preheat temperature of 10°C, dashed lines for 25°C, and dotted lines for 50°C. At the start of the process the maximum temperature difference between these graphs is 40°C, but as the process progresses this difference becomes smaller and smaller. At the end of the process the maximum temperature difference is roughly 7°C for the constant power law case and roughly 6°C for the interpolated power law case. Once again take into account the fact that the viscous dissipation is included in these models for the 0.0001s time step, which introduces the temperature deviation as discussed previously. For the constant power law case this results in a slight elevation above the boundary condition temperature as can be seen in Figure 6.15. Here the case with a preheat temperature of 50°C reaches a maximum temperature of roughly 131°C in the area close to the inner cylinder. Although the temperature difference at the end of the process is only 6°C or 7°C, the preheating of the mixture could still have a beneficial effect, namely in reducing the total processing time. In order to achieve the fibrous structure in the final product, shear has to be applied to the ingredient mixture for a certain amount of time at a certain temperature. This temperature lies in the range of 100°C to 130°C. Figure 6.16 shows that the 10°C preheat case reaches a temperature of roughly 105°C after 1400 seconds, whereas the 50°C preheat case reaches this temperature after only 1000s. This means that preheating to 50°C allows the ingredient mixture to enter the temperature zone where fibrous structure can be created roughly 400 seconds (or 6,7 minutes) earlier than the 10°C case. This could possibly reduce the processing time for the 50°C preheat case by 6,7 minutes compared to the 10°C preheat case. So if processing time is a limiting factor for the production capacity, then preheating the ingredient mixture might be interesting to consider.

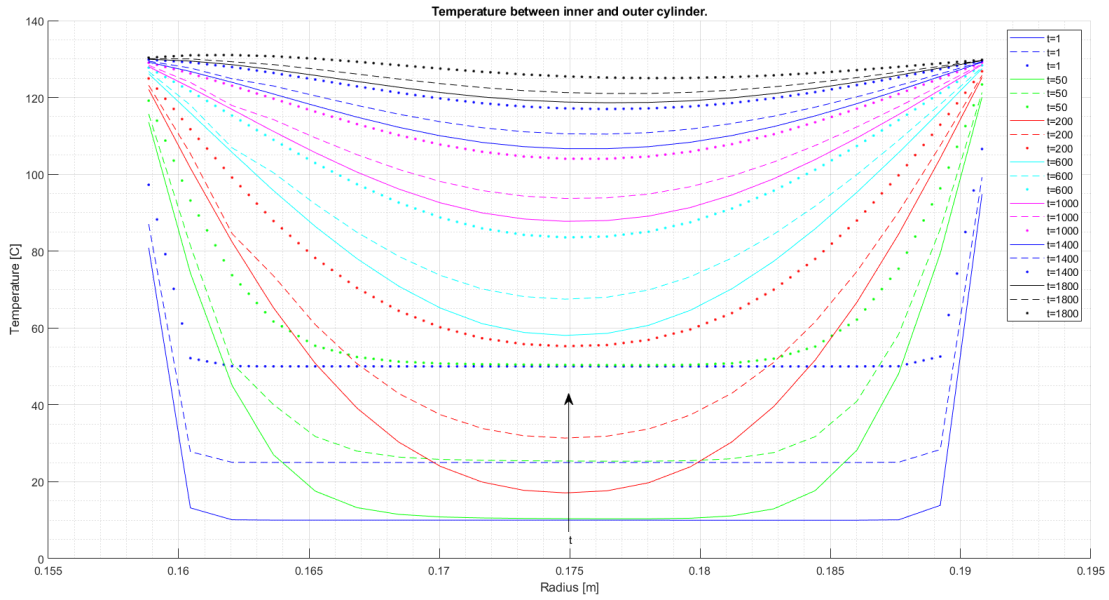


Figure 6.15: A comparison between the temperature profiles using 3 different preheat temperatures for the case with constant power law parameters. Solid lines are used for the preheat temperature 10°C, dashed lines for 25°C, and dotted lines for 50°C. The temperature profiles are given at multiple moments in time. The graph for t=1s starts at the bottom of the figure and the graphs shift upwards as the time increases.

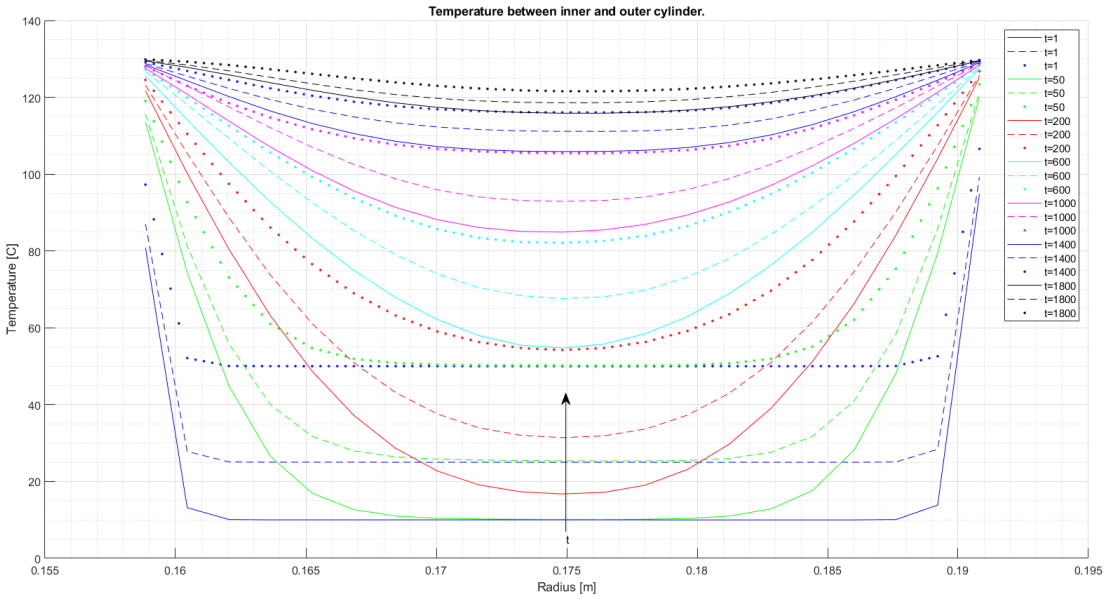


Figure 6.16: A comparison between the temperature profiles using 3 different preheat temperatures for the case with interpolated power law parameters. Solid lines are used for the preheat temperature 10°C, dashed lines for 25°C, and dotted lines for 50°C. The temperature profiles are given at multiple moments in time. The graph for t=1s starts at the bottom of the figure and the graphs shift upwards as the time increases.

6.2.4. Density

The influence of the density on the temperature has been studied as well. Three densities have been used; 820 kg/m^3, 1020 kg/m^3, and 1220 kg/m^3. The results for the case with constant and interpolated power law parameters can be found in Figure 6.17 and Figure 6.18 respectively, where solid lines have been used for the density 820 kg/m^3, dashed lines for 1020 kg/m^3, and dotted lines for 1220 kg/m^3. Equation 4.12 shows that the diffusion term scales with the thermal diffusivity, $\alpha = \kappa/\rho c_p$. As the density becomes lower, this term becomes larger. The same is valid for the viscous dissipation term, which contains the factor $1/\rho c_p$. Therefore the case with the lowest density is expected to reach the highest final temperature. The results confirm this expectation.

At time step t=1800s the constant power law parameter comparison shows a maximum temperature difference of roughly 12°C between the 820 kg/m^3 and 1220 kg/m^3 case, which is quite substantial. Note that the viscous dissipation is once again responsible for raising the maximum temperature to 132°C, which is a value slightly above the maximum temperature at the inner and outer cylinders. For the interpolated power law parameter comparison the maximum temperature difference between the 820 kg/m^3 and 1220 kg/m^3 case at t=1800s is roughly 10°C and the temperature does not rise above the 130°C because the viscous dissipation has a smaller influence in this case.

As previously discussed, the viscous dissipation adds a maximum of 9.8°C and 2.6°C to the constant and interpolated power law parameters cases respectively. However, for these density comparisons the density has been increased and decreased by 200 kg/m^3 from the regular value of 1000 kg/m^3, which is roughly 20% and therefore the viscous dissipation is expected to increase by roughly 20% for the 820 kg/m^3 case and decrease by roughly 20% for the 1220 kg/m^3 case. For the constant power law parameter case the final temperature graphs for all 3 densities are expected to overestimate the final temperature by at most 9.8°C due to the viscous dissipation error for the time step t=0.0001s, as discussed in subsection 6.2.1. For the lowest density of 820 kg/m^3 this error should be corrected with an increase of 20%, which results in a total of 11.76°C overestimation. For the highest density of 1220 kg/m^3 this error should be corrected with a decrease of 20%, which results in a total of 7.84°C overestimation. For the interpolated power law parameter case the same corrections can be applied, which results in an overestimation of 3.12°C and 2.08°C for the lowest and highest density respectively.

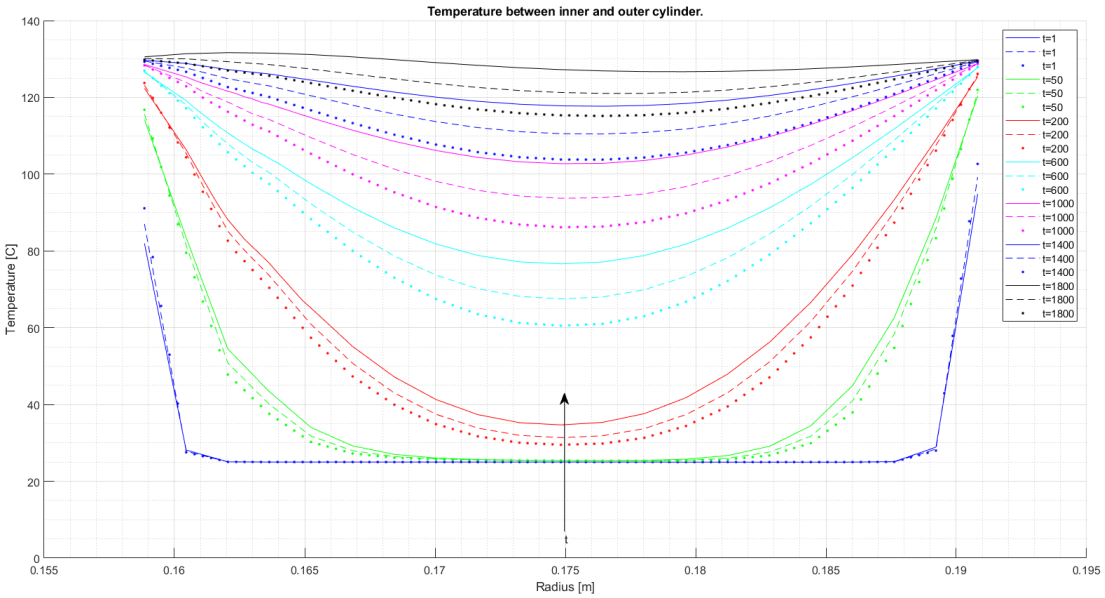


Figure 6.17: A comparison between the temperature profiles for 3 different densities for the case with constant power law parameters. Solid lines are used for the density 820 kg/m^3, dashed lines for the density 1020 kg/m^3, and dotted lines for the density 1220 kg/m^3. The temperature profiles are given at multiple moments in time. The graph for t=1s starts at the bottom of the figure and the graphs shift upwards as the time increases.

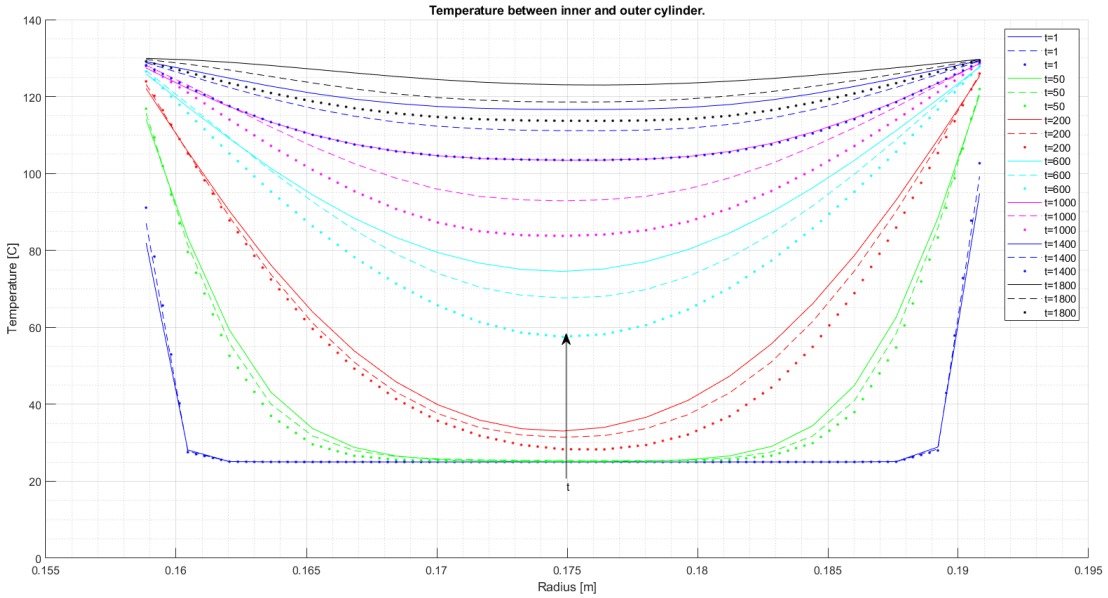


Figure 6.18: A comparison between the temperature profiles for 3 different densities for the case with interpolated power law parameters. Solid lines are used for the density 820 kg/m^3, dashed lines for the density 1020 kg/m^3, and dotted lines for the density 1220 kg/m^3. The temperature profiles are given at multiple moments in time. The graph for t=1s starts at the bottom of the figure and the graphs shift upwards as the time increases.

7

Conclusion

The aim of this research paper was to create a model which can simulate the temperature profile in a geometry similar to the Couette cell. First, experiments and data analysis were performed to identify the appropriate constitutive relations which could then be implemented in the model. The model calculates the temperature, velocity, viscosity, and viscous dissipation fields based on the given material parameters and process conditions. Simulations have been run in OpenFOAM using constant power law parameters and temperature dependent power law parameters. Overall the results from the simulations are in good agreement with the expectations. Some improvements can of course still be made, for which recommendations will be given in Chapter 8.

The temporal convergence study showed that for the time step 0.0001s the velocity, viscosity, and viscous dissipation fields were not yet fully converged, which resulted in errors in the final temperature fields. The temperature field for the constant power law parameter case showed one vertically elongated cold spot which agreed with the expectations. However, the temperature field for the temperature dependent case unexpectedly showed 3 vertically separated cold spots. This separation between the cold spots is caused by small vortex-like velocity components in the x- and z- direction, which had maximum values of 6.6e-03 m/s and 9.8e-03 m/s respectively. In comparison, the velocity in y-direction had a maximum value of 0.5 m/s, which is 2 orders of magnitude higher. It has been found that by further refining the time steps from 0.0001s to 0.000001s the velocity components in x- and z-direction disappeared. The problem with using a time step of 0.000001s for the simulations is that each simulation would take months to run, which was not achievable within the time frame of this study and therefore the simulations have been performed using a time step of 0.0001s. Ansys Fluent is able to run simulations for the same high viscosity power law fluid using much larger time steps. It remains unclear why OpenFOAM requires such small time steps to correctly calculate the velocity fields when simulating high viscosity power law fluids. Some users on CFD fora have reported similar issues, but no reason for this has been found yet.

For the simulations with time step 0.0001s the error in the final temperature field caused by the viscous dissipation has been quantified. For the case with constant and interpolated power law parameters the viscous dissipation can increase the final temperature by up to 9.8°C and 2.6°C respectively when compared to the same case without viscous dissipation. The convergence study has shown that as the time step is refined to 0.000001s the viscous dissipation throughout the entire geometry diminishes, except in the area near the inner cylinder. Therefore refined simulations using the time step 0.000001s are expected to shift the temperature profiles down by an amount on the order of 10°C at most.

When upscaling to larger thicknesses such as the 41mm case it can be seen that the temperature near the middle of the geometry barely reaches 100°C. Keeping in mind that the temperature profile likely shifts down (by at most 9.8°C) when a refined time step is used, this means the temperature is probably too low to create the required structure in the product. Combining the larger thickness option with a preheat process could help reduce temperature inhomogeneity and gradients and allow the coldest part of the product to reach the required temperature zone earlier on.

8

Recommendations and Future Work

The model which has been created for this thesis takes into account the temperature dependency of the power law indices k and n. The number of measurements performed to determine these indices is rather small and therefore it is recommended to perform more measurements if this model is to be used in the future. These new indices can then simply be inserted into the model via the transportProperties file and the interppowerLaw.C file.

Furthermore, to calculate the power law indices the assumption is made that the Cox-Merz relation holds, as has been done in research by Krintiras [12] and van Dijk [13]. It is recommended to verify this assumption. This can be done by performing additional rheological measurements, where the results for the steady state shear viscosity and the complex viscosity (oscillatory shear) should be compared. In addition to testing the validity of Cox-Merz, this would provide an additional check on the presumed form of the constitutive relation. While the current measurements are compatible with a power law relationship between stress and strain rate, it remains possible that there is a qualitatively different relationship at sufficiently low strain rates, e.g. a yield stress plateau or a crossover to a high-viscosity Newtonian regime. Since the highest viscosities occur at the lowest strain rates, this behavior has a strong influence on the convergence properties of the model.

In this study only one parameter was varied per simulation. If more research will be performed using this model it would be interesting to combine multiple parameter variations in a single case to find out if the right conditions for structure formation are still achieved. Results from this thesis have shown that the case with 41mm thickness currently barely reaches 100°C in the center and therefore this case would not be interesting since the required structure would normally not be achieved at this temperature. However, using the 41mm geometry in combination with a preheated ingredient mixture of 50°C could change this and therefore similar combined parameter cases should be studied, since they could broaden the range of scenarios where a properly structured product is created.

Lastly, to make this model more useful for long simulation times it is crucial to find out more about why OpenFOAM has trouble simulating high viscosity fluids using time steps that can easily be handled by similar CFD programs such as Ansys Fluent. Instead of fixing this issue one can also choose to work with Ansys Fluent instead of OpenFOAM, avoiding the convergence issues, but this would mean abandoning the current model. Going forward, working with this OpenFOAM model does not seem like a practical solution for Rival Foods if simulation times of 1800s are used, unless an experienced OpenFOAM user is available who is able to solve the time step issue.

Bibliography

[1] Ishamri Ismail, Young-Hwa Hwang, and Seon-Tea Joo. Meat analog as future food: a review. *Journal of animal science and technology*, 62(2):111, 2020.

[2] Birgit L Dekkers. *Creation of fibrous plant protein foods*. PhD thesis, Wageningen University, 2018.

[3] EE Johanna, CH Annet, and AJS Martinus. B, pietemel al. 2011. consumer acceptance and appropriateness of meat substitutes in a meal context. *Food Qual. Pref*, 22(3):233–240, 2011.

[4] Lone Bredahl, Karen Brunso, and KG Grunert. Consumer perception of meat quality and implications for product development in the meat sector-a reviews. *Meat Science*, 66(2):259–272, 2004.

[5] Lindsay M. Biga, Sierra Dawson, Amy Harwell, Robin Hopkins, Joel Kaufmann, Mike LeMaster, Philip Matern, Katie Morrison-Graham, Devon Quick, Jon Runyeon, and et al. 10.2 skeletal muscle, 2021.

[6] Jean-Louis Damez and Sylvie Clerjon. Meat quality assessment using biophysical methods related to meat structure. *Meat science*, 80(1):132–149, 2008.

[7] Jean-Louis Damez and Sylvie Clerjon. Quantifying and predicting meat and meat products quality attributes using electromagnetic waves: An overview. *Meat science*, 95(4):879–896, 2013.

[8] Navneet Singh Deora and Madhuresh Dwivedi. Structuring meat analogues using extrusion: An insight. *EC Gastroenterology and Digestive System 6.1*, pages 29–31, 2018.

[9] Julita M Manski, Atze J van der Goot, and Remko M Boom. Advances in structure formation of anisotropic protein-rich foods through novel processing concepts. *Trends in Food Science & Technology*, 18(11):546–557, 2007.

[10] Julita M Manski, Atze J van der Goot, and Remko M Boom. Formation of fibrous materials from dense calcium caseinate dispersions. *Biomacromolecules*, 8(4):1271–1279, 2007.

[11] Georgios A Krintiras, Jesse Göbel, Wim G Bouwman, Atze Jan Van Der Goot, and Georgios D Stefanidis. On characterization of anisotropic plant protein structures. *Food & function*, 5(12):3233–3240, 2014.

[12] Georgios A. Krintiras. *Intensified protein structuring for more sustainable foods*. PhD thesis, Delft University of Technology, 2016.

[13] Niels van Dijk. Cfd modeling of a couette cell. the effect of heat transfer and viscous dissipation on the product quality, 2014.

[14] Jesse Göbel. Intensified protein structuring. production of fibrous soy based meat analogs using a couette cell, 2013.

[15] J. Gadea Diaz. Structure formation and parametric study in an up-scaled couette cell, 2015.

[16] Malvern Instruments Limited. A basic introduction to rheology. `https://cdn.technologynetworks.com/TN/Resources/PDF/WP160620BasicIntroRheology.pdf`.

[17] By g-sec - Own work, CC BY-SA 3.0 Wikipedia, the free encyclopedia. Non-newtonian fluid, 2021. [Online; accessed May 11, 2021].

[18] WP Cox and EH Merz. Correlation of dynamic and steady flow viscosities. *Journal of Polymer Science*, 28(118):619–622, 1958.

[19] Wikimedia Commons. Laminar shear in a fluid., 2005-04-04, 2008-06-05.

[20] Raj P Chhabra and John Francis Richardson. *Non-Newtonian flow in the process industries: fundamentals and engineering applications*. Butterworth-Heinemann, 1999.

[21] CW Macosko. Rheology principles, measurements, and applications, vch publ. *Inc, New York*, 1994.

[22] MA Andy Rao. *Rheology of fluid and semisolid foods: principles and applications*. Springer Science & Business Media, 2010.

[23] . Openfoam: User guide v2012, 2012. [Online; accessed June 7, 2021].

[24] Floor KG Schreuders, Igor Bodnár, Philipp Erni, Remko M Boom, and Atze Jan van der Goot. Water redistribution determined by time domain nmr explains rheological properties of dense fibrous protein blends at high temperature. *Food Hydrocolloids*, 101:105562, 2020.

[25] MA Emin, M Quevedo, M Wilhelm, and HP Karbstein. Analysis of the reaction behavior of highly concentrated plant proteins in extrusion-like conditions. *Innovative Food Science & Emerging Technologies*, 44:15–20, 2017.

[26] Hrvoje Jasak. Error analysis and estimation for the finite volume method with applications to fluid flows. 1996.

[27] Henrik Rusche. *Computational fluid dynamics of dispersed two-phase flows at high phase fractions*. PhD thesis, Imperial College London (University of London), 2003.

[28] Anthony F Mills. *Basic heat and mass transfer*. Prentice hall, 1999.

[29] Patrick J Roache. Quantification of uncertainty in computational fluid dynamics. *Annual review of fluid Mechanics*, 29(1):123–160, 1997.

[30] Lewis Fry Richardson and J Arthur Gaunt. Viii. the deferred approach to the limit. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, 226(636-646):299–361, 1927.

[31] Patrick J Roache. Perspective: a method for uniform reporting of grid refinement studies. 1994.

[32] Leonard E Schwer. Is your mesh refined enough? estimating discretization error using gci. *7th LS-DYNA Anwenderforum*, 1(1):45–54, 2008.

[33] R Byron Bird, Warren E Stewart, and Edwin N Lightfoot. Transport phenomena revised 2nd edition, 2006.

[34] Frank M White. Fluid mechanics, mcgraw-hill. *New York*, 1994.

[35] A Davey, RC Di Prima, and JT Stuart. On the instability of taylor vortices. *Journal of Fluid Mechanics*, 31(1):17–52, 1968.

[36] V Sinevic, R Kuboi, and AW Nienow. Power numbers, taylor numbers and taylor vortices in viscous newtonian and non-newtonian fluids. *Chemical engineering science*, 41(11):2915–2923, 1986.

A

Appendix A: RPA Elite Measurement Data

25 C					
Frequency [Hz]	G' [kPa] - Test #1	G' [kPa] - Test #2	GEMIDDELDE(G')	STDEV.S(G')	SEM
10,000	9,146	9,369	9,257	0,157755523	0,11155
5,988	8,700	8,030	8,365	0,473195858	0,3346
3,597	8,086	6,915	7,501	0,828092751	0,58555
2,155	6,971	6,915	6,943	0,039385848	0,02785
1,292	6,636	6,246	6,441	0,275983777	0,19515
0,774	5,911	5,131	5,521	0,552108975	0,3904
0,464	6,023	4,071	5,047	1,380131016	0,9759
0,278	4,183	4,963	4,573	0,552088264	0,39085
0,167	4,629	3,513	4,071	0,788636198	0,55765
0,100	4,015	4,071	4,043	0,039456558	0,0279
Frequency [Hz]	G'' [kPa] - Test #1	G'' [kPa] - Test #2	GEMIDDELDE(G'')	STDEV.S(G'')	SEM
10,000	2,900	2,956	2,928	0,039385848	0,02785
5,988	2,398	2,621	2,510	0,157684812	0,1115
3,597	2,119	2,342	2,231	0,157755523	0,11155
2,155	1,785	2,063	1,924	0,197212081	0,13945
1,292	2,565	0,948	1,757	1,143603797	0,80865
0,774	2,398	1,562	1,980	0,591494822	0,41825
0,464	1,394	1,283	1,338	0,078913117	0,0558
0,278	1,506	1,562	1,534	0,039456558	0,0279
0,167	2,398	1,617	2,008	0,552108975	0,3904
0,100	1,394	2,119	1,757	0,512581706	0,36245
Frequency [Hz]	η* [Pa*s] - Test #1	η* [Pa*s] - Test #2	GEMIDDELDE(η*)	STDEV.S(η*)	SEM
10,000	152,701	156,353	154,527	2,582353965	1,826
5,988	239,849	224,520	232,185	10,83923985	7,6645
3,597	369,855	323,032	346,444	33,10886082	23,4115
2,155	531,382	532,912	532,147	1,081873375	0,765
1,292	876,442	778,215	827,329	69,4569778	49,1135
0,774	1311,729	1102,761	1207,245	147,7626899	104,484
0,464	2119,331	1463,239	1791,285	463,9271023	328,046
0,278	2542,708	2976,167	2759,438	306,5017983	216,7295
0,167	4973,819	3690,256	4332,038	907,6161014	641,7815
0,100	6764,852	7304,413	7034,533	381,6686633	269,8805

Figure A.1: Data from the SAOS measurements at 25°C

50C					
Frequency [Hz]	G' [kPa] - Test #1	G' [kPa] - Test #2	GEMIDDELDE(G')	STDEV.S(G')	SEM
10,000	5,632	6,469	6,051	0,591494822	0,41825
5,988	5,186	5,019	5,103	0,118298964	0,08365
3,597	4,127	4,796	4,461	0,473195858	0,3346
2,155	3,681	3,681	3,681	0	0
1,292	4,127	4,127	4,127	0	0
0,774	3,569	3,513	3,541	0,039456558	0,0279
0,464	2,119	2,621	2,370	0,354896893	0,25095
0,278	3,513	2,565	3,039	0,670837229	0,474
0,167	2,565	2,286	2,426	0,197212081	0,13945
0,100	2,398	2,677	2,537	0,197141371	0,1394
Frequency [Hz]	G'' [kPa] - Test #1	G'' [kPa] - Test #2	GEMIDDELDE(G'')	STDEV.S(G'')	SEM
10,000	1,673	2,733	2,203	0,749250845	0,5298
5,988	1,673	2,286	1,980	0,4337393	0,3067
3,597	1,952	1,673	1,812	0,197141371	0,1394
2,155	0,892	1,227	1,060	0,236597929	0,1673
1,292	0,390	0,669	0,530	0,197141371	0,1394
0,774	1,060	1,394	1,227	0,236597929	0,1673
0,464	0,892	1,617	1,255	0,512581706	0,36245
0,278	1,115	0,112	0,613	0,709793787	0,5019
0,167	0,725	0,669	0,697	0,039456558	0,0279
0,100	0,502	1,115	0,809	0,4337393	0,3067
Frequency [Hz]	η* [Pa*s] - Test #1	η* [Pa*s] - Test #2	GEMIDDELDE(η*)	STDEV.S(η*)	SEM
10,000	93,514	111,765	102,640	12,90540586	9,1255
5,988	144,841	146,590	145,716	1,23672976	0,8745
3,597	201,980	224,737	213,359	16,09162902	11,3785
2,155	279,677	286,507	283,092	4,829539816	3,415
1,292	510,625	514,996	512,811	3,090763741	2,1855
0,774	765,559	777,236	771,398	8,256885884	5,8385
0,464	788,252	1055,821	922,037	189,1998543	133,7845
0,278	2108,453	1468,726	1788,590	452,3552998	319,8635
0,167	2543,473	2273,084	2408,279	191,1938955	135,1945
0,100	3899,181	4615,281	4257,231	506,359166	358,05

Figure A.2: Data from the SAOS measurements at 50°C

75C					
Frequency [Hz]	G' [kPa] - Test #1	G' [kPa] - Test #2	GEMIDDELDE(G')	STDEV.S(G')	SEM
10,000	4,908	4,740	4,824	0,118298964	0,08365
5,988	4,461	4,406	4,433	0,039385848	0,02785
3,597	4,573	4,629	4,601	0,039385848	0,02785
2,155	4,517	4,461	4,489	0,039456558	0,0279
1,292	4,071	3,904	3,987	0,118298964	0,08365
0,774	4,238	3,067	3,653	0,828092751	0,58555
0,464	3,625	3,346	3,485	0,197141371	0,1394
0,278	3,736	3,346	3,541	0,276054487	0,1952
0,167	3,067	3,458	3,262	0,275983777	0,19515
0,100	4,517	2,788	3,653	1,222446208	0,8644
Frequency [Hz]	G'' [kPa] - Test #1	G'' [kPa] - Test #2	GEMIDDELDE(G'')	STDEV.S(G'')	SEM
10,000	1,785	1,394	1,589	0,275983777	0,19515
5,988	1,562	1,506	1,534	0,039456558	0,0279
3,597	1,394	0,669	1,032	0,512652416	0,3625
2,155	1,617	1,060	1,338	0,394282741	0,2788
1,292	1,729	1,673	1,701	0,039456558	0,0279
0,774	1,227	0,056	0,641	0,828092751	0,58555
0,464	0,669	0,056	0,363	0,4337395	0,3067
0,278	0,725	0,390	0,558	0,236597929	0,1673
0,167	1,283	0,112	0,697	0,828092751	0,58555
0,100	0,892	0,112	0,502	0,552108975	0,3904
Frequency [Hz]	η* [Pa*s] - Test #1	η* [Pa*s] - Test #2	GEMIDDELDE(η*)	STDEV.S(η*)	SEM
10,000	83,108	78,637	80,873	3,161474419	2,2355
5,988	125,630	123,745	124,688	1,332896283	0,9425
3,597	211,521	206,924	209,223	3,250569873	2,2985
2,155	354,313	338,624	346,469	11,09579829	7,8445
1,292	544,830	523,178	534,004	15,31027605	10,826
0,774	907,287	630,800	769,044	195,5058326	138,2435
0,464	1263,664	1147,234	1205,449	82,32844253	58,215
0,278	2177,071	1926,903	2051,987	176,8954892	125,084
0,167	3172,069	3300,666	3236,368	90,98181074	64,2985
0,100	7328,101	4441,319	5884,710	2041,263128	1443,391

Figure A.3: Data from the SAOS measurements at 75°C

100 C						
Frequency [Hz]	G' [kPa] - Test #1	G' [kPa] - Test #2	G' [kPa] - Test #3	GEMIDDELDE(G')	STDEV.S(G')	SEM
10,000	8,755	8,086	12,924	9,922	2,621618182	1,513591963
5,988	7,417	7,696	12,407	9,173	2,803946021	1,61885899
3,597	8,365	7,863	11,846	9,358	2,169280639	1,252434761
2,155	7,361	8,700	11,912	9,324	2,338591282	1,350186306
1,292	7,082	8,142	11,553	8,926	2,336153294	1,348778733
0,774	7,194	8,588	11,856	9,213	2,392907481	1,381545779
0,464	7,752	8,309	10,894	8,985	1,676787558	0,968093748
0,278	7,194	8,254	11,739	9,062	2,378170057	1,373037123
0,167	7,919	7,584	12,348	9,284	2,659179663	1,535278094
0,100	8,421	10,038	13,102	10,520	2,377489234	1,372632502
Frequency [Hz]	G'' [kPa] - Test #1	G'' [kPa] - Test #2	G'' [kPa] - Test #3	GEMIDDELDE(G'')	STDEV.S(G'')	SEM
10,000	0,390	0,781	1,311	0,827	0,461857431	0,266653512
5,988	0,502	0,390	1,315	0,736	0,504922342	0,29151705
3,597	0,112	0,279	1,101	0,497	0,529720845	0,305834472
2,155	0,446	0,167	1,145	0,586	0,503768424	0,290850835
1,292	0,223	0,892	0,799	0,638	0,362386192	0,209223766
0,774	1,004	0,390	1,573	0,989	0,591308684	0,341392228
0,464	0,613	0,112	0,334	0,353	0,251499083	0,145203063
0,278	0,781	1,283	0,780	0,948	0,290105132	0,167492276
0,167	0,056	0,056	0,699	0,270	0,371607211	0,214547523
0,100	0,725	1,060	1,252	1,012	0,266549584	0,153892474
Frequency [Hz]	η* [Pa*s] - Test #1	η* [Pa*s] - Test #2	η* [Pa*s] - Test #3	GEMIDDELDE(η*)	STDEV.S(η*)	SEM
10,000	139,484	129,294	206,754	158,511	42,08931409	24,30027682
5,988	197,585	204,809	331,613	244,669	75,3821323	43,52189438
3,597	370,143	348,122	526,410	414,892	97,20343619	56,12043005
2,155	544,607	642,566	883,772	690,315	174,5511668	100,7771631
1,292	872,880	1008,976	1426,563	1102,806	288,5207456	166,5775301
0,774	1493,600	1767,772	2459,238	1906,870	497,61946	287,3007292
0,464	2865,698	2848,826	3738,551	3084,358	573,8987872	331,3406193
0,278	4139,093	4777,674	6735,546	5217,438	1352,93637	781,1181776
0,167	7555,846	7236,601	11787,094	8859,847	2540,090875	1466,522151
0,100	13451,642	16064,728	20947,129	16821,166	3804,567109	2196,567845

Figure A.4: Data from the SAOS measurements at 100°C

125 C						
Frequency [Hz]	G' [kPa] - Test #1	G' [kPa] - Test #2	G' [kPa] - Test #3	GEMIDDELDE(G')	STDEV.S(G')	SEM
10,000	5,019	1,729	6,831	4,526	2,586455516	1,4932908
5,988	5,688	3,179	7,786	5,551	2,306937181	1,3319108
3,597	4,573	1,004	7,778	4,451	3,388587164	1,9564017
2,155	5,019	1,785	6,867	4,557	2,572791809	1,485402
1,292	5,298	1,840	6,132	4,423	2,275513158	1,3137681
0,774	5,242	1,171	6,607	4,340	2,827976452	1,632733
0,464	5,242	1,506	5,391	4,046	2,201421097	1,2709911
0,278	4,740	1,673	6,662	4,359	2,516525055	1,4529164
0,167	4,406	0,948	5,411	3,588	2,341234293	1,3517122
0,100	4,015	2,063	1,608	2,562	1,278868464	0,7383551
Frequency [Hz]	G'' [kPa] - Test #1	G'' [kPa] - Test #2	G'' [kPa] - Test #3	GEMIDDELDE(G'')	STDEV.S(G'')	SEM
10,000	0,781	0,223	1,591	0,865	0,688068796	0,3972567
5,988	0,948	0,725	1,367	1,013	0,325709925	0,1880487
3,597	0,558	1,338	0,739	0,878	0,408544933	0,2358735
2,155	0,335	0,390	0,514	0,413	0,092085047	0,0531653
1,292	0,781	0,056	0,277	0,371	0,371449709	0,2144566
0,774	0,056	1,060	1,563	0,893	0,767391263	0,4430536
0,464	0,056	0,669	0,231	0,319	0,315888599	0,1823784
0,278	0,892	0,502	1,444	0,946	0,473240422	0,2732255
0,167	1,338	0,390	2,114	1,281	0,863017449	0,4982634
0,100	2,175	0,056	1,853	1,361	1,141866087	0,6592567
Frequency [Hz]	η* [Pa*s] - Test #1	η* [Pa*s] - Test #2	η* [Pa*s] - Test #3	GEMIDDELDE(η*)	STDEV.S(η*)	SEM
10,000	80,841	27,742	111,627	73,403	42,43431593	24,499464
5,988	153,271	86,656	210,119	150,015	61,79603051	35,677955
3,597	203,826	74,022	345,688	207,845	135,8775533	78,448942
2,155	371,485	134,900	508,611	338,325	189,046628	109,14612
1,292	659,667	226,803	756,130	547,533	281,9169169	162,76481
0,774	1077,978	324,747	1396,076	932,934	550,1953384	317,65543
0,464	1797,186	564,889	1850,814	1404,290	727,4538974	419,9957
0,278	2759,006	999,096	3902,787	2553,630	1462,699431	844,48991
0,167	4393,197	978,233	5536,570	3636,000	2371,627864	1369,26
0,100	7267,653	3285,149	3904,066	4818,956	2143,094146	1237,316

Figure A.5: Data from the SAOS measurements at 125°C

B

Appendix B: OpenFOAM code

B.1. mu_viscousHeatingSolver.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Copyright (C) 2011-2018 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    nonNewtonianIcoFoam

Description
    Transient solver for incompressible, laminar flow of non-Newtonian
    fluids.

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
```

```
#include "singlePhaseTransportModel.H"
#include "pisoControl.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//

int main(int argc, char *argv[])
{
    #include "postProcess.H"

    #include "setRootCaseLists.H"
    #include "createTime.H"
    #include "createMeshNoClear.H"
    #include "createControl.H"
    #include "createFields.H"
    #include "initContinuityErrs.H"

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//

    Info<< "\nStarting time loop\n" << endl;

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

        #include "CourantNo.H"

        fluid.correct();

        // Momentum predictor

        fvVectorMatrix UEqn
        (
            fvm::ddt(U)
          + fvm::div(phi, U)
          - fvm::laplacian(fluid.nu(), U)
          - (fvc::grad(U) & fvc::grad(fluid.nu()))
        );

        if (piso.momentumPredictor())
        {
            solve(UEqn == -fvc::grad(p));
        }

        // --- PISO loop
        while (piso.correct())
        {
            volScalarField rAU(1.0/UEqn.A());
            volVectorField HbyA(constrainHbyA(rAU*UEqn.H(), U, p));
            surfaceScalarField phiHbyA
            (
                "phiHbyA",
                fvc::flux(HbyA)
              + fvc::interpolate(rAU)*fvc::ddtCorr(U, phi)
            );

            adjustPhi(phiHbyA, U, p);
```

```
89
90          // Update the pressure BCs to ensure flux consistency
91          constrainPressure(p, U, phiHbyA, rAU);
92
93          // Non-orthogonal pressure corrector loop
94          while (piso.correctNonOrthogonal())
95          {
96              // Pressure corrector
97
98              fvScalarMatrix pEqn
99              (
100                 fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
101             );
102
103             pEqn.setReference(pRefCell, pRefValue);
104
105             pEqn.solve();
106
107             if (piso.finalNonOrthogonalIter())
108             {
109                 phi = phiHbyA - pEqn.flux();
110             }
111         }
112
113         #include "continuityErrs.H"
114
115         U = HbyA - rAU*fvc::grad(p);
116         U.correctBoundaryConditions();
117     }
118
119 volTensorField gradU = fvc::grad(U);
120 volTensorField tau = fluid.nu() * (gradU + gradU.T());
121
122 viscdis = (1/c)*(tau && gradU);     //viscous dissipation term
123 scalar viscdismax = max(viscdis).value() ;          //maximum
    viscous dissipation wegschrijven
124  scalar viscdismin = max(viscdis).value() ;          //minimum
       viscous dissipation wegschrijven
125 Info << "viscdismax, viscdismin "    << viscdismax  << " " <<
    viscdismin <<  endl;
126
127 fvScalarMatrix TEqn
128     (
129         fvm::ddt(T)
130         + fvm::div(phi, T)
131         - fvm::laplacian(DT, T)
132         == (1/c)*(tau && gradU) // viscous heat dissipation term
133     );
134
135     TEqn.solve();
136
137     runTime.write();
138
139     Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
140         << "  ClockTime = " << runTime.elapsedClockTime() << " s"
141         << nl << endl;
```

```
    }

    Info<< "End\n" << endl;

    return 0;
}


// ************************************************************************* //
```

B.2. interppowerLaw.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Copyright (C) 2011-2020 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

\*---------------------------------------------------------------------------*/

#include "interppowerLaw.H"
#include "addToRunTimeSelectionTable.H"
#include "surfaceFields.H"

// * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
namespace viscosityModels
{
    defineTypeNameAndDebug(interppowerLaw, 0);

    addToRunTimeSelectionTable
    (
        viscosityModel,
```

```
41          interppowerLaw,
42          dictionary
43      );
44  }
45  }
46
47
48  // * * * * * * * * * * Private Member Functions  * * * * * * * *//
49
50  Foam::volScalarField Foam::viscosityModels::interppowerLaw::calcNu()
        const
51  {
52  const volScalarField& T= U_.mesh().lookupObject<volScalarField>("T");
53  volScalarField mynu= U_.mesh().lookupObject<volScalarField>("mynu");
54  volScalarField nu0 = mynu;
55
56  dimensionedScalar k_local=k_;
57  dimensionedScalar n_local=n_;
58  volScalarField mystrainrate = strainRate();
59  dimensionedScalar nu_tmp = nuMin_;
60
61  scalar myeps=1e-8;
62
63  forAll(T.internalField(), cellI)
64      {
65
66      if (T[cellI]<298+myeps)
67          {
68          k_local.value() = 0.97990;
69          n_local.value() = 0.18;
70          }
71      else if (T[cellI]>=298-myeps && T[cellI]<323+myeps)
72          {
73          k_local = k1_ + ((k2_-k1_)/(T2_-T1_))*(T[cellI]-T1_);   //
               interpolate to find k value
74          n_local = n1_ + ((n2_-n1_)/(T2_-T1_))*(T[cellI]-T1_);   //
               interpolate to find n value
75          }
76
77      else if (T[cellI]>=323-myeps && T[cellI]<348+myeps)
78          {
79          k_local = k2_ + ((k3_-k2_)/(T3_-T2_))*(T[cellI]-T2_);   //
               interpolate to find k value
80          n_local = n2_ + ((n3_-n2_)/(T3_-T2_))*(T[cellI]-T2_);   //
               interpolate to find n value
81          }
82
83      else if (T[cellI]>=348-myeps && T[cellI]<373+myeps)
84          {
85          k_local = k3_ + ((k4_-k3_)/(T4_-T3_))*(T[cellI]-T3_);   //
               interpolate to find k value
86          n_local = n3_ + ((n4_-n3_)/(T4_-T3_))*(T[cellI]-T3_);   //
               interpolate to find n value
87          }
88
89      else if (T[cellI]>=373-myeps && T[cellI]<=398+myeps)
```

```
90          {
91          k_local = k4_ + ((k5_-k4_)/(T5_-T4_))*(T[cellI]-T4_);   //
               interpolate to find k value
92          n_local = n4_ + ((n5_-n4_)/(T5_-T4_))*(T[cellI]-T4_);   //
               interpolate to find n value
93          }
94
95      else
96          {
97          k_local.value() = 0.67754;
98          n_local.value() = 0.084;
99          }
100
101         nu_tmp = max
102             (
103         nuMin_,
104         min
105         (
106             nuMax_,
107             (k_)*pow
108             (
109                 max
110                 (
111                     mystrainrate[cellI],
112                     small
113                 ),
114                 n_.value() - scalar(1)
115             )
116         )
117             );
118
119
120             mynu[cellI] = nu_tmp.value();
121
122
123
124     }
125
126                 mynu.correctBoundaryConditions();
127
128
129         return mynu;
130
131 }
132
133
134 // * * * * * * * * * * * * * * Constructors  * * * * * * * * * *//
135
136
137 // * * * * * * * * * * * * * * Constructors  * * * * * * * * * *//
138
139 Foam::viscosityModels::interppowerLaw::interppowerLaw
140 (
141     const word& name,
142     const dictionary& viscosityProperties,
143     const volVectorField& U,
```

```
144     const surfaceScalarField& phi
145 )
146 :
147     viscosityModel(name, viscosityProperties, U, phi),
148     interppowerLawCoeffs_(viscosityProperties.optionalSubDict(typeName
            + "Coeffs")),
149     k_("k", dimViscosity, interppowerLawCoeffs_),
150     n_("n", dimless, interppowerLawCoeffs_),
151     nuMin_("nuMin", dimViscosity, interppowerLawCoeffs_),
152     nuMax_("nuMax", dimViscosity, interppowerLawCoeffs_),
153     T1_("T1", dimless, interppowerLawCoeffs_),
154     T2_("T2", dimless, interppowerLawCoeffs_),
155     T3_("T3", dimless, interppowerLawCoeffs_),
156     T4_("T4", dimless, interppowerLawCoeffs_),
157     T5_("T5", dimless, interppowerLawCoeffs_),
158     k1_("k1", dimViscosity, interppowerLawCoeffs_),
159     k2_("k2", dimViscosity, interppowerLawCoeffs_),
160     k3_("k3", dimViscosity, interppowerLawCoeffs_),
161     k4_("k4", dimViscosity, interppowerLawCoeffs_),
162     k5_("k5", dimViscosity, interppowerLawCoeffs_),
163     n1_("n1", dimless, interppowerLawCoeffs_),
164     n2_("n2", dimless, interppowerLawCoeffs_),
165     n3_("n3", dimless, interppowerLawCoeffs_),
166     n4_("n4", dimless, interppowerLawCoeffs_),
167     n5_("n5", dimless, interppowerLawCoeffs_),
168 
169     nu_
170     (
171         IOobject
172         (
173             name,
174             U_.time().timeName(),
175             U_.db(),
176             IOobject::NO_READ,
177             IOobject::AUTO_WRITE
178         ),
179         calcNu()
180     )
181 {}
182 
183 
184 // * * * * * * * * * * * * * Member Functions  * * * * * * * * * * *//
185 
186 bool Foam::viscosityModels::interppowerLaw::read
187 (
188     const dictionary& viscosityProperties
189 )
190 {
191     viscosityModel::read(viscosityProperties);
192 
193     interppowerLawCoeffs_ = viscosityProperties.optionalSubDict(
            typeName + "Coeffs");
194 
195     interppowerLawCoeffs_.lookup("k") >> k_;
196     interppowerLawCoeffs_.lookup("n") >> n_;
197     interppowerLawCoeffs_.lookup("nuMin") >> nuMin_;
```

```
198     interppowerLawCoeffs_.lookup("nuMax") >> nuMax_;
199
200     interppowerLawCoeffs_.lookup("T1") >> T1_;
201     interppowerLawCoeffs_.lookup("T2") >> T2_;
202     interppowerLawCoeffs_.lookup("T3") >> T3_;
203     interppowerLawCoeffs_.lookup("T4") >> T4_;
204     interppowerLawCoeffs_.lookup("T5") >> T5_;
205     interppowerLawCoeffs_.lookup("k1") >> k1_;
206     interppowerLawCoeffs_.lookup("k2") >> k2_;
207     interppowerLawCoeffs_.lookup("k3") >> k3_;
208     interppowerLawCoeffs_.lookup("k4") >> k4_;
209     interppowerLawCoeffs_.lookup("k5") >> k5_;
210     interppowerLawCoeffs_.lookup("n1") >> n1_;
211     interppowerLawCoeffs_.lookup("n2") >> n2_;
212     interppowerLawCoeffs_.lookup("n3") >> n3_;
213     interppowerLawCoeffs_.lookup("n4") >> n4_;
214     interppowerLawCoeffs_.lookup("n5") >> n5_;
215
216     return true;
217 }
218
219
220 // ************************************************************************* //
```

B.3. interppowerLaw.H

```
1  /*---------------------------------------------------------------------------*\
2    =========                 |
3    \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
4     \\    /   O peration     | Website:  https://openfoam.org
5      \\  /    A nd           | Copyright (C) 2011-2020 OpenFOAM
         Foundation
6       \\/     M anipulation  |
7  -------------------------------------------------------------------------------
8  License
9      This file is part of OpenFOAM.
10
11     OpenFOAM is free software: you can redistribute it and/or modify it
12     under the terms of the GNU General Public License as published by
13     the Free Software Foundation, either version 3 of the License, or
14     (at your option) any later version.
15
16     OpenFOAM is distributed in the hope that it will be useful, but
          WITHOUT
17     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
          or
18     FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public
          License
19     for more details.
20
21     You should have received a copy of the GNU General Public License
22     along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.
23
24 Class
25     Foam::viscosityModels::interppowerLaw
26
```

```
27 Description
28      Customized temperature dependent power-law non-Newtonian viscosity
            model.
29
30 SourceFiles
31     interppowerLaw.C
32
33 \*---------------------------------------------------------------------*/
34
35 #ifndef interppowerLaw_H
36 #define interppowerLaw_H
37
38 #include "viscosityModel.H"
39 #include "dimensionedScalar.H"
40 #include "volFields.H"
41
42 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
43
44 namespace Foam
45 {
46 namespace viscosityModels
47 {
48
49 /*---------------------------------------------------------------------*\
50                            Class interppowerLaw Declaration
51 \*---------------------------------------------------------------------*/
52
53 class interppowerLaw
54 :
55     public viscosityModel
56 {
57     // Private Data
58
59         dictionary interppowerLawCoeffs_;
60
61         dimensionedScalar k_;
62         dimensionedScalar n_;
63         dimensionedScalar nuMin_;
64         dimensionedScalar nuMax_;
65
66         dimensionedScalar T1_;
67         dimensionedScalar T2_;
68         dimensionedScalar T3_;
69         dimensionedScalar T4_;
70         dimensionedScalar T5_;
71         dimensionedScalar k1_;
72         dimensionedScalar k2_;
73         dimensionedScalar k3_;
74         dimensionedScalar k4_;
75         dimensionedScalar k5_;
76         dimensionedScalar n1_;
77         dimensionedScalar n2_;
78         dimensionedScalar n3_;
79         dimensionedScalar n4_;
80         dimensionedScalar n5_;
81
```

```
82          volScalarField nu_;
83
84
85      // Private Member Functions
86
87          //- Calculate and return the laminar viscosity
88           Foam::volScalarField  calcNu() const ;
89
90
91  public:
92
93      //- Runtime type information
94      TypeName("interppowerLaw");
95
96
97      // Constructors
98
99          //- Construct from components
100         interppowerLaw
101         (
102             const word& name,
103             const dictionary& viscosityProperties,
104             const volVectorField& U,
105             const surfaceScalarField& phi
106         );
107
108
109     //- Destructor
110     virtual ~interppowerLaw()
111     {}
112
113
114     // Member Functions
115
116         //- Return the laminar viscosity
117         virtual tmp<volScalarField> nu() const
118         {
119             return nu_;
120         }
121
122         //- Return the laminar viscosity for patch
123         virtual tmp<scalarField> nu(const label patchi) const
124         {
125             return nu_.boundaryField()[patchi];
126         }
127
128         //- Correct the laminar viscosity
129         virtual void correct()
130         {
131             nu_ = calcNu();
132         }
133
134         //- Read transportProperties dictionary
135         virtual bool read(const dictionary& viscosityProperties);
136 };
137
```

```

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//

} // End namespace viscosityModels
} // End namespace Foam

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//

#endif

// ****************************************************************** //
```

B.4. fvSchemes

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default          backward;
}

gradSchemes
{
    default          Gauss linear;
    grad(p)          Gauss linear;
}

divSchemes
{
    default          none;
    div(phi,U)       Gauss linear;
    div(phi,T)       Gauss vanLeer;
}


laplacianSchemes
{
    default          none;
    laplacian(nu,U) Gauss linear corrected;
    laplacian((1|A(U)),p) Gauss linear corrected;
    laplacian(DT,T) Gauss linear corrected;
```

```
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         orthogonal;
}


// ************************************************************************* //
```

B.5. fvSolution

```

/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

    T
    {
       solver           PBiCG;
        preconditioner  DILU;
        tolerance       1e-14;
```

```
40          relTol           0;
41      };
42
43      U
44      {
45          solver          smoothSolver;
46          smoother        symGaussSeidel;
47          tolerance       1e-8;
48          relTol          0;
49      }
50  }
51
52  PISO
53  {
54      nCorrectors     2;
55      nNonOrthogonalCorrectors 0;
56      pRefCell        0;
57      pRefValue       0;
58  }
59
60
61  // ************************************************************************* //
```