Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Energy and Process Engineering

Agsagan Ragunathan

Numerical Investigation of Natural Convection of Oil Flow in Transformer

Master's thesis in MTPROD

Supervisor: Reidar Kristoffersen and Karl Yngve Lervaag

June 2020

NTNU
Norwegian University of Science and Technology

SINTEF

Agsagan Ragunathan

Numerical Investigation of Natural Convection of Oil Flow in Transformer

Master's thesis in MTPROD
Supervisor: Reidar Kristoffersen and Karl Yngve Lervaag
June 2020

Norwegian University of Science and Technology
Faculty of Engineering
Department of Energy and Process Engineering

Abstract

The power loss and subsequent cold start of an electrical transformer may cause extensive temperatures that results in premature aging. Oil is therefore used to advect the heat from the transformer. However, the viscosity of oil is highly temperature-dependent such that, as temperatures get decreases, oils may become very viscous. This can be a problem in cold environments during a cold start of the transformer. A cold start is when the transformer is powered up after the oil has reached the ambient temperatures.

In this thesis, a CFD model is used to simulate the cold-start problem in a section of a transformer that consists of 4 passes stacked on top of each other. The top pass is fully resolved, while a porous-medium approximation is employed on the 3 bottom passes to significantly decrease the computation time. The CFD model is solved using OpenFOAM, where a mesh is constructed to adapt to the stacked transformer passes. Both the implementation and the mesh are verified and shown to accurately solve the governing equations. The simulations reveal that the maximum HST during a cold start in a cold environment is found to be $17.2\,\mathrm{K}$ higher than one conducted in milder climate. This indicates that cold starting a transformer in cold environments may cause premature aging.

Sammendrag

Effekttapet, samt gjentatte kaldstart av en elektrisk transformator kan føre til at temperaturen i transformatoren blir så høy at levetiden reduseres. For å minke denne temperaturøkningen brukes tranformatorolje som kjøler ned transformatoren ved hjelp av varmeadveksjon. Oljens viskositet er svært temperaturavhengig og kan bli svært viskøs når den blir kald. Dette kan skape problomer dersom man kaldstarter en transformator i kalde omgivelser. En kaldstart er når transformatoren blir startet opp etter at oljen har nådd temperaturen til omgivelsene.

I denne master oppgaven er CFD tatt i bruk for å simulere et slikt kaldstartscenario. Dette er gjort ved å betrakte en del av transformatoren som består av 4 pass lagt oppå hverandre. Det øverste passet er løst fullt ut, porøst materiale approksimasjon er brukt på de 3 nederste passene for å redusere kjøretiden. Dette CFD problemet er simulert ved bruk av OpenFOAM, hvor et grid av celler er laget til å tilpasse passene i transformatoren. Både implementasjonen og grid er verifisert og vist at de løser ligningene som beskriver problemet, nøyaktig. Simuleringen viser at den maksimale HST under en kaldstart ved lave omgivelsestemperaturer er $17.2\,\mathrm{K}$ høyere enn ved milde omgivelsestemperaturer. Dette indikerer at kaldstarting av transformatorer ved lave omgivelsestemperaturer kan føre til at levetiden til transformatorene reduseres.

Table of Contents

Chapter 1

Introduction

Electrical transformers, hereon called transformers, are a vital component of today's power system. Today there exist more than a thousand transformers in the power system in Norway, some of these have a capacity of hundreds of MW. Theses transformers are typically $2\,\mathrm{m}$ to $3\,\mathrm{m}$ high and cost up to 7.5 million USD [15]. It is therefore of great interest to keep the lifespan of the transformer as high as possible.

The purpose of a transformer is to change the voltage of an alternating current (AC). This is achieved by using different number of windings in the low-voltage (LV) and high-voltage (HV) winding. A simple transformer model is shown in Figure 1.1. In this figure, the primary and secondary winding are HV and LV winding, respectively. The primary winding is where the current is given as an input, while the secondary is the output. When the current flows through the windings, it is subjected to heat loss [49]. The major contributor to the heat loss is the electrical resistance in the windings given by ohm's law, $R = U/I$. Here, U is the voltage and I is the current. Moreover, the power $P = UI$ is ideally the same in both windings. Thus, the current will be higher at the LV winding compared to the HV winding, and therefore also the heat loss, as Joule's law of heating is given as $\dot{E}_{\mathrm{g}} = I^2R$ [10, page 132]. For a transformer, the total losses are around $1\,\%$ [3]. Consequently, a high power capacity transformer may cause a significant amount of heat loss to be significant, especially in the LV windings.

To counteract this heat buildup, it is common to use an oil to convect the heat out of the windings. This oil flows inside a closed loop starting from the windings and flowing to a radiator through a pipe and then back, as seen in Figure 1.2. There are mainly two mechanisms used for driving the oil convection through the windings. The first is called *oil natural* (ON) and relies on natural convection, where the buoyancy force is the driving factor. The second one is called *oil forced* (OF). Here, a pump is used to move the oil between the radiator and the windings. Convection is the transfer of heat from a solid to a fluid in the presence of advection. Advection is the transfer of heat by the flow of a fluid. Similarly, air cooling of the radiator is driven by *air natural* (AN) and *air forced* (AF). For the ONAN transformer, the velocity will be low, due to the low compressibility of the oil [5]. The oil flow in these transformers can therefore be assumed as purely laminar. Fur-

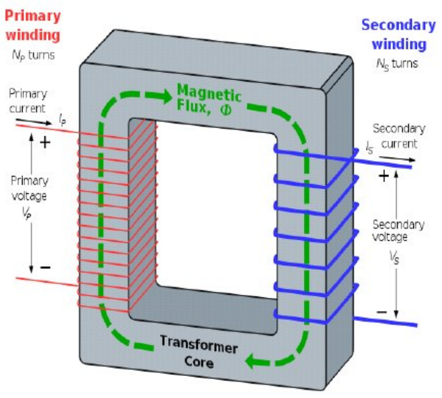


Figure 1.1: A simple transformer circuit [42].

thermore, there exist different kinds of geometric arrangements to steer the flow through the transformer windings. One such arrangement is zig-zag. This arrangement is when the windings are divided into blocks, called passes, where a washer is used for each pass to divert the flow from one corner to the opposite corner. In this type of transformer, several passes are stacked on top of each other as shown in Figure 1.3 [28].

The main cause of a transformer breakdown is the degradation of the insulating layer of paper between the oil and the windings. This degradation is caused by high temperature working as a catalyst for different chemical processes on the oil, which then reacts with the insulating paper. For a typical transformer, the degradation process becomes significant at oil temperatures above $413\,\mathrm{K}$. In addition, a sudden transformer failure may occur if the oil temperature exceeds $453\,\mathrm{K}$ [37]. Therefore, adequate heat convection is important to provide sufficient cooling to keep the temperature below this level.

Most transformers that are built today are equipped with several temperature sensors. This enables better prediction of the hot-spot temperature (HST) and its locations. However, older transformers lack these sensors, which makes it harder to accurately predict their lifespan and characteristics. These transformers are therefore often significantly overdimensioned. By being able to more accurately determine the HST and its locations from the load history of the transformers, the expensive reinvestments may be delayed. Additionally, by increasing the knowledge about the load characteristics of the transformers, the operators will be able to better optimize their transformer operation. This may further increase the lifespan of their transformers.

The viscosity of the transformer oil is highly temperature-dependent, especially at lower temperatures. The lowest temperature where oil is not capable of flowing under gravity is called pour point [7]. For the commonly used transformer oil `MIDEL7131` [5], the pour point is reached at $253.15\,\mathrm{K}$. This temperature level is quite common during a Norwegian winter. This increase in viscosity will restrict the oil flow, potentially making the oil unable to convect the heat from the windings. Consequently, a cold start of a transformer in a cold environment may cause significant degradation of the insulation

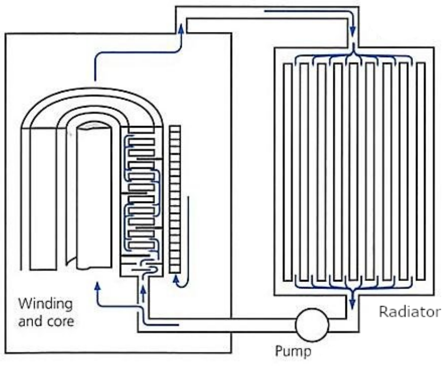


Figure 1.2: Transformer schematics [8].

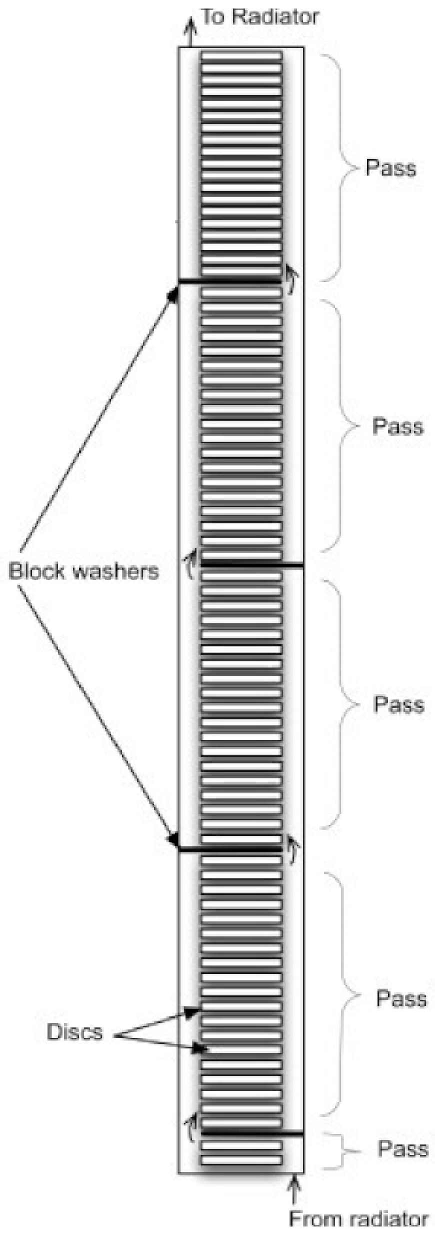


Figure 1.3: Winding geometry for a zig-zag arragment [38].

paper, compromising the lifespan of the transformer. Thus, to increase the knowledge of the transformer's load characteristics, it is important to model the viscosity accurately.

A typical scenario of a cold start at sub-zero temperatures is when the transformer is powered up after a prolonged power outage. In this case, the oil may have reached the ambient sub-zero temperature. Moreover, houses with temperature controlled devices such as electrical heaters will simultaneously be on full load to increase temperature to the set level before the power outage [16]. This will increase the load of the transformers, further increasing the risk of too high HST.

This thesis presents a computational fluid dynamics (CFD) model to study the cold-start problem of a transformer. The model is based on an adapted transformer from Torriano et al. [43], which is a 2D ONAN type of transformer that consists of 4 passes configured in a zig-zag arrangement. A porous-medium approximation is used for the 3 bottom passes to reduce the computation cost. Only the passes enclosing the LV windings are considered. The inlet and the outlet of the transformer are connected with a channel, which includes an artificial radiator to simulate the external cooling of the oil. The model is hereafter referred to as the *LV loop*. The model is implemented within the open-source CFD toolbox, OpenFOAM.

The CFD model is used to study a cold-start case where the initial temperature is $253.15\,\mathrm{K}$. The model is shown to have comparable results with results found in Ref.[43]. The simulation results reveal a maximum HST of $447.2\,\mathrm{K}$. Moreover, this simulation reveals an increased in the maximum HST compared to its steady-state value of $21.7\,\mathrm{K}$. Additionally, the maximum HST is observed to be $17.2\,\mathrm{K}$ higher for a cold-start initiated from cold temperature compared to mild temperature. These results, all indicate that this situation may indeed cause significant degradation of the transformer. However, further investigation is required, as some of the model uncertainties are shown to have significant influence on these results.

The thesis is outlined as follows. Chapter 2 provides a literature study of the several existing methods to obtain the HSTs and some relevant transformer cold-start studies. The governing equations and the model geometry are presented in Chapter 3. In Chapter 4, the implementation details are described. A couple of verification and validation cases, the cold-start case and a sensitivity analysis are specified in Chapter 5. The results from these case studies are presented and discussed in Chapter 6. Finally, concluding remarks are given in Chapter 7.

Chapter 2

Literature Review

Power transformers have existed for more than 100 years, and the problem with high hot-spot temperatures (HSTs) has been considered for almost as long [30]. As such, there has been a lot of research both on how to determine the HST at normal steady-state co-operation, and in the transient phase of a cold-start situation. The following chapter will provide a literature review of the most relevant studies. In particular, the different methods that have been used to find the HST and studies that have investigated the cold-start phenomena at sub-zero temperatures.

Today, there exist several methods to determine the HST. However, the industry standard is to employ the models by the International Electrotechnical Commission (IEC) [22] and the Institute of Electrical and Electronics Engineers (IEEE) [25]. These models predict the HST from the temperature of the oil at the top of the transformer, called the top-oil temperature (TOT). The TOT can be determined either by a so-called heat-run test or by using some simple empirical equations that are made for simple geometries. The initial models presented by the IEEE and IEC assumed that the TOT and the HST were the same [34, 24, 23]. However, with the introduction of fiber-optic temperature sensors in transformers in recent years, many authors have addressed that TOT and HST are far from equal during a change in the load [41, 32, 34]. In fact, the data from the acquired sensors have shown that the rise in HST was up to 2 times higher than the rise in TOT during a change in the load. IEC and IEEE have therefore later changed their loading guides to consider this transient behavior, as well as different geometries and cooling methods. Nonetheless, they still rely on primitive thermal models with only a few coefficients [22, 25]. Thus, compromising the accuracy of the obtained HST value.

Numerous authors have tried to introduce more advanced methods in order to more accurately predict the HST [35, 11]. The thermal-hydraulic network model (THNM) and computational fluid dynamics (CFD) are the two main methods used by these authors to determine the HST. THNM is a method where a given problem is modeled using an electrical circuit analogy. The model assigns constant properties to each part of the domain, called lumped elements. Radakovic and Sorgic [35] were the first to apply this method on a transformer. However, due to the complexity of the physics and the geometry in addition of being time constrained, they were not able to obtain a solution. Furthermore, a problem with the THNM that is addressed by Campelo et al. [11], is that the hydraulic and thermal resistance terms have to be determined for all the lumped elements in the network. Additionally, these are different for different transformers. The resistance terms can be determined by the estimated friction factors and heat transfer coefficients or by performing a heat-run test. The latter have been done by Radakovic and Sorgic [35] to obtained those coefficients on the investigated transformer. However, this requires the transformer to be taken out of service. Consequently, Campelo et al. [11] did instead run several CFD simulations in order to obtain some general expressions for these coefficients. THNM have later been successfully adapted to include the transient behavior of a transformer by Cotas et al. [14].

CFD is a numerical method to simulate a fluid flow based on the conservation of mass, momentum and energy. This allows detailed simulations of the heat and mass flow through the transformer winding. However, large computational resources are needed from a transformer design point of view [35]. That said, as the computational power is rapidly increasing, the use of CFD to find the HST is gaining popularity. Nonetheless, most CFD simulations of the oil flow in the transformer are only applied on some simple cases. Here, the most common approach is to simulate the steady-state case of a single transformer using a 2D axisymmetrical assumption, as done by Kranenborg et al. [27] and Torriano et al. [43]. However, simplifying the simulation to be 2D axisymmetric neglects the influence of instabilities in 3D due the duct spacers [44]. A 2D axisymmetric approximation is still justifiable because the distance between the spacers are relative large compared to the distance between the windings [38]. That said, some researchers have choose to include more of the transformer geometry in their domain. As an example, Skillen et al. [38] considered five transformer passes stacked upon each other to create a full windings model. They found that by restricting the simulated domain to a single pass, the effects of hot streaks will be neglected. Consequently, the commonly assumed uniform inlet oil temperature and velocity when considering a single pass will be inaccurate. However, the influence of neglecting the temperature and velocity inlet profiles have not been considered in their study. Although, Torriano et al. [43] only considered a single transformer pass, they do provide a detailed description of both the geometry of the used ONAN transformer pass, and its boundary conditions. Additionally, their results have been validated. This case will therefore be interesting as a reference for comparisons.

As stated, CFD becomes very expensive, in particular when one considers fully 3D geometries under high resolutions. Torriano et al. [44] reports that a full 3D steady-state simulation of only a single pass required four days of compute time on a memory-cluster compromised of 60 CPU nodes. To counteract this problem with high computational cost, Gastelurrutia et al. [17] run a 3D simulation of the oil flow in the transformer by approximating the vertical cooling channels as porous medium. It is interesting to note that Gastelurrutia et al. [17] considered the flow to have some turbulent regions. They employed the standard $k - \epsilon$ turbulence CFD model to find the HST and its location in the transformer. They endorsed the presence of turbulence by referring to some previous analytical estimates of the Rayleigh number. This number can in fact be used to determine if a buoyancy driven flow is turbulent. Furthermore, they confirmed the presence of turbulence by referring to Oh and Ha [33]. Nevertheless, Oh and Ha [33] investigated a much simpler geometry where the whole transformer consisted of a single horizontal heated core that was surrounded by oil. As a result, their setup experienced very little drag forces due to less wall area compared to the oil volume. Hence, the flow velocity did get abnormally high. Whereas Gastelurrutia et al. [17] examined a more realistic ONAN transformer with zig-zag flow pattern making the drag forces more prominent.

Recently, Meyer et al. [29] proposes a similar porous-medium approximation as Gastelurrutia et al. [17]. However, their approximation have been conducted on the transformer pass instead of the cooler. The approximation was shown to significantly reduce the number of cells needed to achieve reasonable macroscopic results compared to a fully resolved model.

There exist a lot of literature regarding whether the THNM or CFD should be the most preferable method to find the HST and its location in a transformer [38, 47, 45]. Weinlader et al. [47] and Torriano et al. [45] are stating that the THNM provides fast-to-use approximations, as the number of lumped elements in THNM are far less than the number of cells in CFD simulations. However, the THNM are based on some underlying assumptions and empirical relations that compromises the accuracy. Moreover, the results from the THNM do not provide a complete depiction of the oil flow and temperature distribution throughout the windings [38, 47]. Similarly, Torriano et al. [45] concluded that although the THNM is capable of predicting the global thermal behavior of the windings, some local discrepancies can be observed. As an example, the hot streaks mentioned by Skillen et al. [38] should be observed inside the transformer. Nonetheless, THNM is not able to resolve this effect due to the need for fine discretization [26]. In conclusion, the use of THNM neglects important flow characteristics. Hence, detailed knowledge about the transformer's temperature distribution, such as the HST location cannot be obtained.

The cold start of a transformer, and especially from sub-zero temperatures, has not received a lot of attention in the literature. The IEEE loading guide [25] states that cold starts may yield localized hot spots, but it does not consider this particular case in more detail. One of the few papers that investigate this effect is Ref. [16], which investigates a cold start in a cold environment after a prolonged power outage. The authors claim that the described situation will induce the thermally controlled devices, such as all the buildings with electrical heaters, to work on full power simultaneously. This may cause a possible overloading of the transformer. However, they neither perform calculations nor experiments on a sub-zero ambient case.

Conversely, Rapp et al. [36] performed experimental cold-start simulations at $-30\,^{\circ}\mathrm{C}$ on a small $167\,\mathrm{kV\,A}$ transformer. For these experiments, different liquids, all being in a solid state at $-30\,^{\circ}\mathrm{C}$ where considered. Temperatures where measured for the top oil, the core structure, the primary winding duct exit oil and the secondary winding. Here, the primary is the input and the secondary is the output. These experiments, revealed the maximum temperature at full load to be about $80\,^{\circ}\mathrm{C}$.

Similarly, Cloet et al. [13] considered the cold-start phenomena at sub-zero ambient temperature by performing experiments. However, these experiments was conducted on an offshore wind turbine transformer, which is a bit larger than the transformer that Rapp et al. [36] examined. Their experiments were done at $-30\,^{\circ}\mathrm{C}$, using temperature sensors at the top oil in addition to the top and bottom windings. From these experiments, they reached a conclusion that the temperatures did not rise above $60\,^{\circ}\mathrm{C}$. However, as previously mentioned, Susa et al. [41], Nordman et al. [32], and Pierce [34] all concluded that the HST increase may be up to twice the increase in TOT during a step change in the load. Thus, the actual HST may be much higher then the predicted values by Cloet et al. [13] and Rapp et al. [36]. Therefore, further investigation is needed.

CFD simulation of the cold-start scenario has been performed by Moore et al. [31] on both a large ONAN and ONAF type of transformer pass. The investigated transformer pass was assumed to be 2D axissymmetric. Their CFD model was successfully validated through experimental results by performing a test run with an initial temperature of $-13.8\,^{\circ}\mathrm{C}$. The validated CFD model was then used to run another simulation with $-25\,^{\circ}\mathrm{C}$ as the initial temperature, revealing that the HST was below $105\,^{\circ}\mathrm{C}$. However, they found that the transformer needed a significant amount of heating, more precisely 6 hours of fully load, before the fluid reached a mass flow rate of $20\,\%$ of its steady-state value. This indicates that the oil temperature in some regions may reach dangerously high levels, although this was not captured in the paper. In addition, they did not consider the zig-zag type of transformer.

Rapp et al. [36], Cloet et al. [13], Moore et al. [31] all consider the cold-start problem, but not the likely case of overloading of the transformer after a prolonged power breakdown as Edstrom et al. [16] mentioned. However, both effects are considered by Adibi [9]. They conducted experiments by overloading different transformers at an ambient temperature of as low as $-40\,^{\circ}\mathrm{C}$. The HSTs were found using thermocouples in the accessible portion of the windings. The maximum measured temperatures were between $102\,^{\circ}\mathrm{C}$ and $210\,^{\circ}\mathrm{C}$ when overloading at three times the rated capacity for 2 hours, depending on the transformer. This reveals that a cold-start overload may in fact cause serious degradation.

To summarize, this literature review exposes a knowledge gap in the literature regarding the HST in the cold-start problem. The THNM does not accurately resolve the HST as the resolution in the domain is low, and the average quantities are used for the resistance terms [45]. Therefore, CFD models seem to be the only viable option to study the HST accurately [26].However, using CFD on the whole geometry of the transformer LV loop will bring the computational cost unrealistically high with the limited resources at hand. Therefore, a porous-medium approximation as proposed by Meyer et al. [29] should be applied on all the passes in the LV loop except for the top pass. Furthermore, there are some uncertainties whether the flow contain some turbulent regions that need to be considered to correctly estimate the HST Gastelurrutia et al. [17]. Moreover, a 2D axisymmetric model is demonstrated by Skillen et al. [38] to be reasonably accurate. Thus, a 2D transformer pass given by Torriano et al. [43] will be used in this study. This has an inner radius of curvature of $316.2\,\mathrm{mm}$, which is much higher than its width of $67.2\,\mathrm{mm}$ for the pass. Thus, the effects in the radial direction can be neglected, thereby considering the LV loop to be purely 2D in this thesis.

Chapter 3

Model and governing equations

As described in the introduction, the main objective of this thesis is to investigate the cold-start phenomena in a cold environment. In such an environment, one must account for highly temperature-dependent fluid properties, in particular with regard to viscosity. In this chapter, the governing equations for fluid flow in a transformer will be presented and described. First, the simplified case of incompressible, steady-state Navier-Stokes equations are presented for verification purposes. Next, the compressible Navier-Stokes equations are presented, as these are required to considered the temperature dependent properties of the cold-start problem. Then, a porous-medium approximation is considered for the less interesting part of the domain, that will significantly decrease the computational cost. Finally, a brief discussion of the geometry and the boundary condition will be given in this chapter.

3.1 Compressible flow

To model the flow of oil through the transformer in cold-start cases with strongly temperature-dependent fluid properties, the well-known compressible Navier-Stokes equations with energy conservation are used [48, page 73]:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0, \tag{3.1a}$$

$$\frac{\partial \rho \boldsymbol{u}}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}\boldsymbol{u}) = -\nabla p + \rho \boldsymbol{g} + \nabla \cdot \left[\mu \left(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^{\mathrm{tr}}\right)\right] + \boldsymbol{f}, \tag{3.1b}$$

$$\frac{\partial \rho h}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} h) + \frac{\partial \rho K}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} K) = \frac{\partial p}{\partial t} + \nabla \cdot \left(\frac{\kappa}{c_p} \nabla h\right) + \rho \boldsymbol{u} \cdot \boldsymbol{g} + \Phi + S. \tag{3.1c}$$

Here, h is the enthalpy, κ is the thermal conductivity coefficient, $\boldsymbol{f}$ is the volumetric body forces except for the gravity, which is given as $\boldsymbol{g}$, K is the specific energy field given as

$K = \frac{1}{2}\boldsymbol{u}\cdot\boldsymbol{u}$ and Θ is the (viscous) dissipation function [48, page 72]:

$$\Phi = \mu\left[2\left(\frac{\partial u_{\mathrm{x}}}{\partial x}\right)^2 + 2\left(\frac{\partial u_{\mathrm{y}}}{\partial y}\right)^2 + 2\left(\frac{\partial u_{\mathrm{z}}}{\partial z}\right)\right.$$
$$\left. + \left(\frac{\partial u_{\mathrm{y}}}{\partial x} + \frac{\partial u_{\mathrm{x}}}{\partial y}\right)^2 + \left(\frac{\partial u_{\mathrm{z}}}{\partial y} + \frac{\partial u_{\mathrm{y}}}{\partial z}\right)^2 + \left(\frac{\partial u_{\mathrm{x}}}{\partial z} + \frac{\partial u_{\mathrm{z}}}{\partial x}\right)^2\right], \quad (3.2)$$

where x, y and z are the coordinates of the three dimensions in space, these have also been used as a subscript for the velocity u to denote directions.

In the case of an ONAN transformer, the buoyancy force is the driving force for the oil flow. Transformer oil is typically not very compressible and the temperature change is expected to be within $413\,\mathrm{K}$ [37]. Therefore, the velocities are expected to be low. This implies that the rate of change of the kinetic and potential energy, respectively the $\frac{\partial \rho K}{\partial t} + \nabla\cdot(\rho\boldsymbol{u}K)$ and the $\rho\boldsymbol{u}\cdot\boldsymbol{g}$, can be neglected [18]. This also applies for the (viscous) dissipation function Φ. In addition, the pressure work dp/dt have also been neglected. Applying these assumption, the compressible energy equation Eq. (3.1c) can be rewritten as:

$$\frac{\partial \rho h}{\partial t} + \nabla\cdot(\rho\boldsymbol{u}h) = \nabla\cdot\left(\frac{\kappa}{c_p}\nabla h\right) + S. \quad (3.3)$$

Thus, this simplified compressible energy equations will be considered for the compressible cases instead of Eq. (3.1c). For the steady-state compressible cases, the time derivative terms will vanish. Furthermore, these governing equations are solved from an initially specified state in some domain with boundaries, that will be discussed in the last section.

3.2 Incompressible flow

For verification purposes, some temperature-independent cases for which there exist analytical solutions are considered. For these cases is resolved by the incompressible Navier-Stokes equations with energy conservation [48, page 97],

$$\nabla\cdot\boldsymbol{u} = 0, \quad (3.4a)$$
$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla\cdot(\boldsymbol{u}\boldsymbol{u}) = -\frac{1}{\rho_0}\nabla p + \nabla\cdot\left[\nu\left(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^{\mathrm{tr}}\right)\right], \quad (3.4b)$$
$$\frac{\partial T}{\partial t} + \nabla\cdot(\boldsymbol{u}T) = \nabla\cdot(\alpha\nabla T) + \frac{S}{c_p\rho_0}. \quad (3.4c)$$

Here, $\boldsymbol{u}$ is the velocity vector, t is the time, g is the gravitational constant, T is the temperature, p is the pressure, S is the volumetric heat source, ρ_0 is the constant reference density, ν is the kinematic viscosity, and c_p is the specific heat capacity at constant pressure. Similarly to the compressible equations, the time derivative will be zero for the steady-state cases. In addition, these steady-state incompressible equations will be solved from an initially specified state in some domain with boundaries, that will be discussed in the following section.

3.3 Geometry

As discussed in Chapter 1, the heat production is higher for the LV- than the HV-winding, because the power loss is electrical resistance times current squared [10, page 132]. Consequently, only the passes enclosing the low voltage windings will be considered in this thesis. Furthermore, the 2D geometry of the low voltage pass that is used by Torriano et al. [43] will be used to resolve the cold-start situation of a transformer at low-ambient temperatures. This geometry is presented in section 3.3.1.

Instead of only considering a single transformer pass as done by Torriano et al. [43], the whole stack of 4 LV passes configured in a zig-zag arrangement is considered. Each pass have the same geometry, with a height of $367\,\mathrm{mm}$ and a length of $66.1\,\mathrm{mm}$. Inside each pass there are 19 windings, each with a height and width of $15\,\mathrm{mm}$ and $50.8\,\mathrm{mm}$, respectively. In between each winding and at the top and bottom, there is a space of $4.1\,\mathrm{mm}$ creating 20 horizontal channels. At each side of the windings, continuous vertical channels are made. Here, the right channel have a width of $8.9\,\mathrm{mm}$ and the left $6.4\,\mathrm{mm}$. The whole geometry has a depth of $10\,\mathrm{mm}$[1].

Furthermore, Torriano et al. [43] did only considered the top pass with predefined inflow mass rate and temperature. The same cannot be applied for the cold-start situation as both the inflow mass rate and its temperature are transient parameters. Accordingly, the inlet at the bottom pass and the outlet at the top pass are connected through an artificial canal. The canal is made to have a constant width of $8.9\,\mathrm{mm}$. This canal is extended by $17.8\,\mathrm{mm}$ out of the inlet and outlet. These are then extended to the right with a length of $26.7\,\mathrm{mm}$, and are then connected through a straight vertical canal. The parameters of the canal are only assumed values. A sensitivity analysis will therefore be performed using different canal geometries. Moreover, a radiator has to be added to simulate the external cooling of the oil, which in turn is necessary for the process to reach steady state. That said, the geometry of the radiator will not be resolved as it is beyond the scope of this thesis. In this thesis the radiator is assumed to reset the temperature with the purpose of reaching steady state. The influence of this assumption will also be investigated in a sensitivity analysis.

To fully specify the cases, boundary conditions along the domain boundaries have to be determined. For the temperature, the same uniform heat flux from the windings to the oil as Torriano et al. [43] is applied, as the same geometry is used. That is $2336.4\,\mathrm{W\,m^{-2}}$. This is not an perfect assumption because the heat production at the windings and the temperature profile of the fluid along the windings are not uniform [45, 43]. However, only a overestimation of $8\,\mathrm{K}$ for the HST is found as investigated by Torriano et al. [43]. Thus, this assumption is considered to be sufficiently accurate for this case.

The remaining of the walls are assumed to be adiabatic, as they are made of insulating material. For the velocity a no-slip condition is applied for all the walls, and the pressure is set to correspond with that. Rest of the boundary conditions are case specific and will be presented with the cases. This also applies for the initial conditions.

[1]Torriano et al. [43] have used another depth for this case, as they consider the transformer pass to be 2D axisymmetric

3.3.1 Porous-medium approximation

The horizontal channels, are fairly narrow compared to its length. Additionally, most of the heat from the windings are transferred to the fluid at this region. These will result in large gradients. Accordingly, a relatively large amount of cells have to be used compared to its width. However, the HST is assumed to be located in the top pass [45, 27]. It is therefore unnecessary to fully resolve the three bottom passes, as long as the macroscopic properties are conserved at the inlet for of the top pass. An approach to significantly reduce the number of cells, while conserving the macroscopic properties, are introduced and validated by Meyer et al. [29]. This approximation is therefore implemented for the three bottom passes. The resulting geometry of the stack of LV passes including the canal with the radiator is shown in section 3.3.1.

The porous-medium approximation can be implemented in the momentum equation Eq. (3.1b) as a pressure drop in the volumetric body force term $\boldsymbol{f}$. This relation is given by the 2D Darcy-Forchheimer equation [20]:

$$f_{\mathrm{i}} = -\mu D_{\mathrm{i}} u_{\mathrm{i}} - \frac{1}{2}\rho F_{\mathrm{i}} |u_{\mathrm{kk}}| u_{\mathrm{i}}. \tag{3.5}$$

Here, the D_{i} and F_{i} are the i-component of the Darcy and Forchheimer coefficients. The Forchheimer coefficients denote the inertial effects. These can be neglected for this problem as only low velocities are considered [39]. The Darcy coefficients are given as the inverse of the effective permeabilities k_{p} in their respective directions. At the flow direction of the porous-medium approximated region, the effective permeability for multiple parallel channels is given as [29]:

$$\kappa_{\mathrm{p}} = \frac{N h_{\mathrm{c}}}{H} \frac{h_{\mathrm{c}}^2}{12} = \phi \kappa_{\mathrm{p}'}, \tag{3.6}$$

where, N is the number of channels, h_{c} is the channel height, ϕ is the porosity specified as $\phi = N h_{\mathrm{c}}/H$ and $\kappa_{\mathrm{p}'}$ is the single-channel permeability prescribed as $h_{\mathrm{c}}^2/12$. This equation consider the drag in all the channels. Hence, the slip boundary condition should be used for the top and the bottom of the pass that is included in the porous region. Moreover, the Darcy coefficients in the transverse directions have to be set to a very high number, to restrict the flow to only travel in the horizontal direction.

The simplified compressible energy equation Eq. (3.3) equivalent for a porous block is specified as [29]:

$$\frac{\partial \rho h}{\partial t} + \nabla \cdot (\frac{\rho}{\phi} \boldsymbol{u} h) = \nabla \cdot \left(\frac{\boldsymbol{\kappa}}{c_p} \nabla h \right) + S, \tag{3.7}$$

where $\boldsymbol{\kappa}$ is the heat conductivity tensor. The volumetric heat source term S is given as:

$$S = \frac{q'' A_{\text{win}}}{\phi V_{\text{por}}}. \tag{3.8}$$

Here, q'' is the heat flux, A_{win} is the surface area where the heat flux is active, and V_{por} is the volume of the porous region.

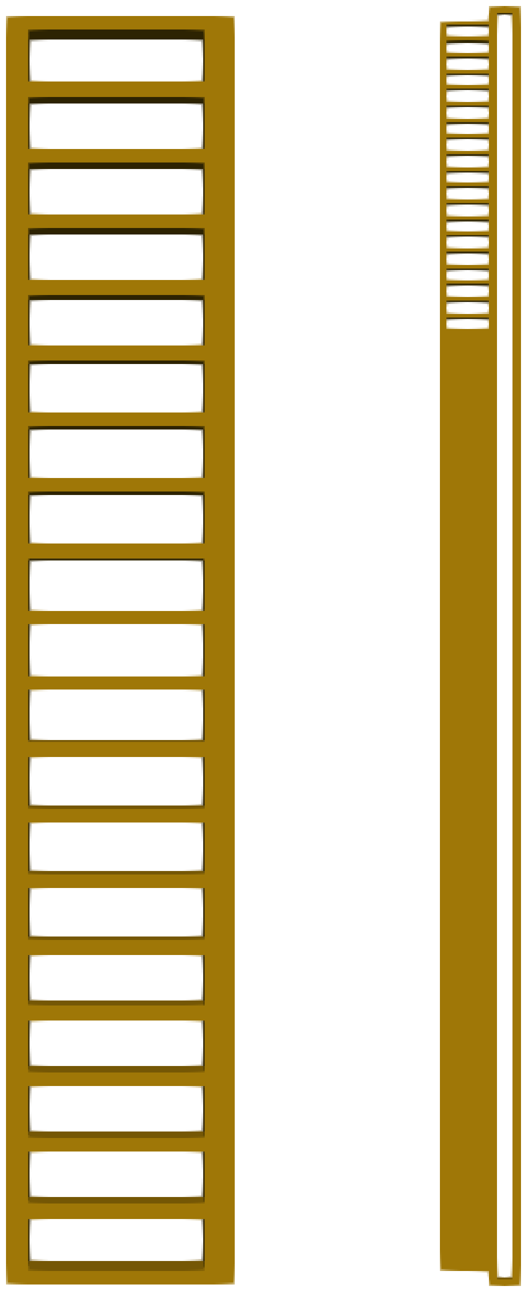


Figure 3.1: The geometry of a single pass on the left and the whole LV winding on the right.

Chapter 4

Numerical methods

The open-source framework OpenFOAM version 18.12 from OpenCFD Ltd. is used to solve the governing equations. For each case, one must provide a set of input parameters to specify the solvers, numerical schemes, mesh, and physical parameters such as the geometry and fluid properties. These parameters and their implementation will be described in detail in the following sections. However, the boundary and initial conditions are excluded from this chapter, since these are case specific and will be described in their respective cases.

4.1 Solver

The builtin compressible transient solver `buoyantPimpleFoam` solves the compressible Navier-Stokes equations presented in Eq. (3.1) by default. However, the energy equation in this solver is modified to correspond to the simplified compressible energy equation given in Eq. (3.3). This solver is used to run both the transient incompressible and compressible cases. For the steady-state cases, the builtin solver `buoyantSimpleFoam` is used. This also solve the same compressible Navier-Stokes equations specified in Eq. (3.1) excluding the time derivative terms. Additionally, the simplified energy equation, Eq. (3.3), is implemented for the energy equation.

In both solvers, temperature-dependent fluid properties can be specified using the `thermophysicalProperties` parameter file. Nevertheless, the cold-start phenomena of the transformer in a cold environment is the main focus of this thesis. It is therefore important to model the highly temperature-dependent viscosity accurately for the whole relevant temperature range. This is not fully supported by the builtin solvers. Consequently, advanced temperature-dependent viscosity function is implemented by changing the source code. Both the `buoyantPimpleFoam` and `buoyantSimpleFoam` solvers have been duplicated to the custom solvers `compressibleSteady` and `compressibleTransient` with the improved temperature-dependent properties.

The developed porous-medium approximation in section 3.3.1 is also implemented in the transient solver via a custom solver, `porousTransient`. It was constructed by duplicating the `compressibleTransient` solver. The continuity equation Eq. (3.1a) is the same in both cases. The pressure drop found in Eq. (3.5) can be implemented in the already provided volumetric body force term in Eq. (3.1b). However, the same procedure cannot be applied for the implementation of the porous energy equation Eq. (3.7), since this equation and the fully resolved compressible energy equation Eq. (3.3) have different enthalphy advection terms. A parameter ϵ is therefore implemented in order to make the solver differentiate between the porous and the fully resolved regions. This parameter is set to 1 in the porous region and 0 in the fully resolved region. Using this parameter, the coupled compressible energy equation can be expressed as:

$$\frac{\partial \rho h}{\partial t} + (1 - \epsilon + \frac{\epsilon}{\phi})\nabla \cdot (\rho \boldsymbol{u} h) = (1-\epsilon)\,\nabla \cdot \left(\frac{\kappa}{c_p}\nabla h\right) + \epsilon\left[\nabla \cdot \left(\frac{\boldsymbol{\kappa}}{c_p}\nabla h\right) + S\right]. \quad (4.1)$$

Accordingly, the compressible energy equation is modified to match this equation in the `porousTransient` solver. In addition, the field parameter ϵ and the thermal conductivity tensor divided by the specific heat capacity $\boldsymbol{\kappa}/c_\mathrm{p}$ is implemented in `createFields.H`. The value of the tensor $\boldsymbol{\kappa}/c_\mathrm{p}$ is implemented inside the time loop of the source file `porousTransient.C` according to the specifications given in section 3.3.1. By including this tensor in the time loop, it is updated at each time step. This will make it consistent with the temperature-dependent fluid properties.

4.1.1 Numerical schemes

The bounded second-order numerical scheme `Gauss vanLeer` is used for the divergence schemes to avoid instabilities. For the velocity divergence scheme, the `Gauss vanLeerV` version is used to make the limiter consider the direction of the velocity field [4]. Moreover, the second-order central difference method `Gauss linear` is used for the gradient schemes. The Laplacian and surface-normal gradient schemes are set to orthogonal, because the considered geometries only have right angles. To match the second-order spatial discretization, the second-order Crank-Nicolson time discretization method is used [4].

4.1.2 Solver parameters

The discretized governing equations have to be solved using matrix solvers. These solvers need to be defined for all the equations. Additionally, a set tolerance of the residual for the matrix solvers should be specified. In this thesis, the symmetrical matrices are solved with the `PCG` method using a `DIC` preconditioner. While the asymmetrical matrices are solved using the `PBiCGStab` solver with the `DILU` preconditioner. The residuals for the matrix solvers are normalized with respect to the absolute residual using the average field value added to the difference between the field values that is compared to its average value [2]. The tolerances for these residuals are set to 10^6 for all the solvers.

For the steady-state solvers, either the residual controls or the end time is the terminating condition. The solver should terminate when the obtained solution are sufficiently close to the steady-state solution. The residuals give an indication of how close to steady state the solution is. Therefore, to ensure that the solver has reached a solution close to steady state, the residual controls are set to 10^{-4}.

The pressure-velocity coupling algorithms SIMPLE and PIMPLE are used for the steady-state and transient cases, respectively. The PIMPLE algorithm is effectively a combination of the SIMPLE and the PISO algorithms [46]. These algorithms need a set of parameters to work, namely the `momentumpredictor` and `nNonOrthogonalCorrectors` for both, in addition to `nOuterCorrectors` and `nCorrectors` for the PIMPLE algorithm. The `momentumpredictor` is turned off, as it is expected that velocities are small [19]. The `nNonOrthogonalCorrectors` is set to 0, because only right-angled meshes are considered. The `nOuterCorrectors` gives the number of times that the whole set of equations are solved or until the optional residual controls are reached. This value is set to 1 as recommended by the OpenFOAM user guide [19]. However, by doing this, the PIMPLE algorithm will replicate the PISO algorithm. The PISO algorithm alone is only stable for Courant numbers lower than 1 [21]. The Courant number, also called the Courant-Friedrichs-Levy (CFL) number is defined as:

$$C = U\frac{\Delta t}{\Delta x}, \tag{4.2}$$

where Δt is the time step, Δx is the cell length and U is the velocity in the given cell at the given time step. A value of the Courant number under 1 effectively imply that some particles have moved through more than one cell during one time step. Lastly, the value for `nCorrectors` determines the number of times the PISO algorithm is run. This is set to 2 as recommended by the OpenFOAM user guide [19].

Moreover, relaxation factors can be applied to suppress the solver in order to avoid oscillations and instabilities. This is done by limiting the amount which a variable changes from one iteration to the next [19]. These do only apply for the SIMPLE algorithm, and is therefore not considered in the transient solver. For the steady-state cases the energy equation and the velocity field have been assigned a relaxation factor of 0.3. Whereas for the pressure field it is set to 0.7.

4.2 Mesh structure

The mesh structure for the geometry has to be determined for the simulations. This should be carefully considered to obtain a good balance between accuracy and run time. The accuracy of the mesh should be verified by replicating cases where the solutions are known. As the implementation of the cases may be wrong, a couple of mesh refinements should be performed to check if the solution converges. This is called a grid convergence study. A grid convergence study will also provide valuable information about the size of cells that is needed to get within an acceptable error. This analysis will therefore be performed in the upcoming sections. For the same reasons, the time step is investigated in a similar way.

For the convergence analysis, a way of determining the error have to be found for the different meshes and time steps. This will be done by calculating the difference between the simulated and the exact solution at different sample points in cases where the solution is available. The errors will then be given as the average of the errors normalized with the exact solution. This error is hereafter called the relative error and can be formulated as

$$E = \frac{1}{n_{\text{total}}} \sum_{n=1}^{n_{\text{total}}} \frac{|f_{\text{e}}^{n} - f_{\text{s}}^{n}|}{f_{\text{e}}^{n}}. \tag{4.3}$$

Here, n_{total} is the total number of sample points, f_{e}^{n} and f_{s}^{n} are respectively the exact and the simulated solution for a given quantity at a given point n.

Chapter 5

Cases

The previous chapters have presented the governing equations for oil flow through a transformer and the numerical methods used to solve the equations for a particular case. As discussed in the introduction, the goal of the thesis is to investigate the challenge with excessive oil temperatures during the cold start of a transformer. This chapter will specify test cases for verifying that the equations are correctly implemented and solved, as well as cases used to study the cold-start phenomenon. The test cases for verification are selected to have known analytical solutions or already validated solutions. At the end, the case of the LV loop will be presented along with its input parameters and fluid properties. The results and corresponding discussions for the various cases are presented in the next chapter.

Most of the following cases consider the transformer oil `MIDEL7131`. The properties of this oil are specified in its data sheet [5] for various temperatures at an increment of $10\,\mathrm{K}$. Meyer et al. [29] presents regression functions that determine the properties as continuous functions of the temperature with almost perfect accuracy,

$$\ln \nu(T) = 20.81369191 \ln^2 T - 252.81869067 \ln T + 755.03026555, \tag{5.1a}$$

$$\kappa(T) = -7.2 \times 10^{-7} T^2 + 3.71 \times 10^{-4} T + 9.75 \times 10^{-2}, \tag{5.1b}$$

$$c_p(T) = 2.17T + 1249.29. \tag{5.1c}$$

Here, ν is the kinematic viscosity, c_p is the specific heat capacity at constant pressure and κ is the thermal heat conductivity. Moreover, a thermal expansion coefficient is given in the data sheet. This coefficient is used to determine the density as a function of temperature through the Boussinesq approximation,

$$\rho = \rho_0 \left[1 - \beta(T - T_0)\right], \tag{5.2}$$

where T_0 is the reference temperature at the temperature where $\rho(T) = \rho_0$ and β is the thermal expansion coefficient. These values are measured at $293.15\,\mathrm{K}$ to be $\rho_0 = 968\,\mathrm{kg\,m^{-3}}$, $\beta = 7\,5 \cdot 10^{-4}$ and $T_0 = 293.15\,\mathrm{K}$ [5].

Some of the following cases assume constant fluid properties. In these cases, `MIDEL7131` is used at a constant temperature of $293.15\,\mathrm{K}$, which corresponds to the values $\kappa = 0.147\,\mathrm{W\,m^{-1}\,K^{-1}}$, $c_p = 1902\,\mathrm{J\,kg^{-1}\,K^{-1}}$, $\rho_0 = 968\,\mathrm{kg\,m^{-3}}$, and $\nu = 7.47 \times 10^{-5}\,\mathrm{m^2\,s^{-1}}$.

Finally, this thesis considers a transformer pass similar to the one used by Torriano et al. [43], who specified a constant uniform heat flux from the windings into the oil of $2336.4\,\mathrm{W\,m^{-2}}$. The same heat flux is used in this work. Unless otherwise specified, the initial velocity in the whole domain is set to 0.

5.1 Horizontal channel flow

From the literature review, it is clear that the HSTs are going to be located at the horizontal channels [38]. It is therefore important to model this flow region with sufficient accuracy. This will be considered in the following subsections by performing verification studies.

One of the horizontal channels is presented in Figure 5.1. As seen from this figure, the length, height and the depth are $50.8\,\mathrm{mm}$, $4.1\,\mathrm{mm}$ and $10\,\mathrm{mm}$, respectively. Thus, the channels are very narrow. This combined with the heat flux from the windings will cause high temperature gradients throughout this region. To capture these gradients with sufficient accuracy, the traverse direction of the horizontal channels must be sufficiently resolved. This implies very small grid cells. Further, uniform square cells would lead to very large requirements of computational resources. Different rectangular shapes of the cells should therefore be considered.

By assuming constant Newtonian fluid properties, there exist analytical steady-state and transient solutions for some specific channel flow problems. These can be used for verification, and in the following subsections, an extension of the well-known Graetz problem and the transient Poiseuille equation will be presented. The former is used to determine the grid resolution, as its solution consider both the hydrodynamic and thermal solution. The latter is used to verify that the correct transient evolution is captured.

For the channel flow test case, the inlet boundary condition is specified with a constant volumetric flow rate of $1.8944 \times 10^{-7}\,\mathrm{m^3\,s^{-1}}$ and a constant temperature of $300\,\mathrm{K}$. The flow rate corresponds to the flow rate used in Torriano et al. [43], where they specify a volumetric flow rate of $3.7888 \times 10^{-6}\,\mathrm{m^3\,s^{-1}}$ for a pass with 20 horizontal channels [1]. The outlet boundary condition is set to be adiabatic with a pressure of $1 \times 10^5\,\mathrm{Pa}$.

5.1.1 Graetz extended problem

The Graetz problem considers both the thermal and the hydraulic part of the steady-state incompressible governing equations for an enclosed laminar flow Eq. (3.4). Its analytical solution is therefore ideal for investigating the horizontal channels presented in the previous section. However, the original Graetz problem does only apply for a thermally and hydrodynamically fully developed flow. Nonetheless, there exist different extensions of this problem depending on whether the flow is hydrodynamically or thermally developing or developed. Consequently, the flow regime needs to be determined. This is done by finding the hydrodynamic and thermal entry lengths using empirical correlations.

[1]Since another depth is used compared to Torriano et al. [43], their inlet velocity is used to find the corresponding volumetric flow rate for this case

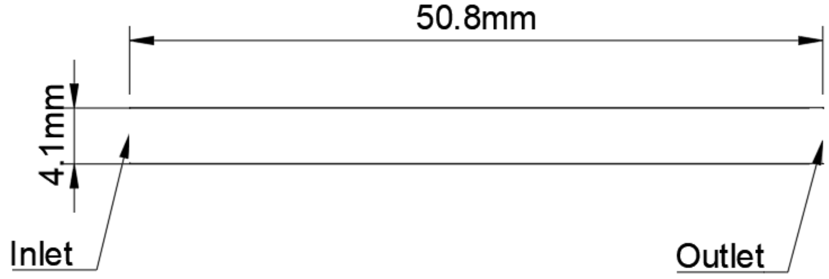


Figure 5.1: The geometry of a horizontal channel.

The empirical correlations for the entry lengths depend on the dimensionless Reynolds and Prandtl numbers. The former is the ratio between the inertial and viscous forces, while the latter is the ratio of the viscous and heat conduction effects. These dimensionless numbers are given as:

$$\mathrm{Re}_{D_\mathrm{h}} = \frac{u_\mathrm{avg} D_\mathrm{h}}{\nu}, \tag{5.3a}$$

$$\mathrm{Pr} = \frac{\alpha}{\nu}. \tag{5.3b}$$

Here, α is the thermal diffusivity coefficient, that is given as $\alpha = \kappa/(c_p\rho)$. The rest of the notations are u_avg for the average inlet velocity and D_h for the hydraulic diameter at the inlet defined as $D_\mathrm{h} = 4A_\mathrm{c}/P$, where P is the perimeter and A_c is the cross-sectional area.

The dimensionless numbers are calculated using the determined constant fluid properties of `MIDEL7131` and the geometry of the investigated horizontal channel. For this geometry the hydraulic diameter is $5.82\,\mathrm{mm}$, resulting in $\mathrm{Re}_{D_\mathrm{h}} = 0.360$, while $\mathrm{Pr} = 1002$. The estimated Reynolds number of 0.360 for the horizontal channels is well below the turbulent transition region at about 2×10^3. Thus, the channel flow is verified to be purely laminar.

The entry length of a laminar uniform inlet velocity can be estimated by the following equation: $L_\mathrm{e,h}/D_\mathrm{h} = 0.5 + 0.05\mathrm{Re}_{D_\mathrm{h}}$ [48, page 107] ,where $L_\mathrm{e,h}$ is the hydrodynamic entry length. This revealed a hydrodynamic entry length of $3.01\,\mathrm{mm}$. Moreover, the thermal entry length is estimated using the following empirical correlation: $L_\mathrm{e,t}/D_\mathrm{h} = 0.05\mathrm{Re}_{D_\mathrm{h}}\mathrm{Pr}$ [48, page 124]. From this, it is estimated that the thermal entry length is $105\,\mathrm{mm}$. Therefore, except for a first $3.01\,\mathrm{mm}$ of the $50.8\,\mathrm{mm}$ long horizontal channel, the flow is estimated to be hydrodynamically developed while still thermally developing. Consequently, an extension of the Graetz problem that considered this flow regime will be employed in the flowing investigation of the channel mesh.

As mentioned in the previous section, the HST is estimated to be in the horizontal channels. A detailed investigation will therefore be conducted on its mesh. This will consist of testing different cell sizes and aspect ratios. The aspect ratio is the ratio between the longest and the shortest length of a cell. The solution, using different mesh structures will then be compared with an analytical solution. That is the solution to the Graetz problem, extended to hydrodynamically developed, thermally developing flow. Such an extension that also includes the uniform heat flux between parallel plates have been implemented by Cess and Shaffer [12]. They provided the following analytical solution for this problem:

$$T_{\text{w}} = T_{\text{in}} + \frac{q_{\text{w}} a}{\kappa} \left[\frac{4}{\text{Pe}} \frac{x}{a} + \frac{17}{35} + \sum_{n=1}^{\infty} c_n Y_n(1) \exp\left(-\frac{8}{3} \frac{\beta_n^2}{\text{Pe}} \frac{x}{a} \right) \right]. \tag{5.4}$$

Here, T_{w} is fluid temperature at the wall, T_{in} is the inlet temperature, q_{w} is the uniform heat flux through the walls, a is half the channel width, Pe is the dimensionless Peclet number, that is for this particular problem defined as $\text{Pe} = 4u_{\text{avg}}a/\alpha$. Furthermore, β_n and $Y_n(1)$ are respectively the eigenvalues and eigenfunctions of an equation presented in Cess and Shaffer [12]. While the c_n are some coefficients given by another equation in Cess and Shaffer [12]. The three first values for β_n, $Y_{n(1)}$ and c_n can be found in Cess and Shaffer [12], whereas the 10 first values are given in Sparrow et al. [40] and presented in Table 5.1. This solution does not include the gravitational force, and it assumes that the fluid properties are not temperature dependent. The gravitational force is therefore excluded from the corresponding simulation, and the fluid properties are regarded as constants as specified in the chapter introduction.

Table 5.1: Values for the given extension of the Graetz problem.

n	β_n	$Y_n(1)$	c_n
1	4.2872	-1.2697	0.17503
2	8.3037	1.4022	-0.051727
3	12.3106	-1.4916	0.025053
4	16.3145	1.5601	-0.014924
5	20.3171	-1.6161	0.0099692
6	24.3189	1.6638	-0.0071637
7	28.3203	-1.7054	0.0054147
8	32.3214	1.7425	-0.0042475
9	36.3223	-1.7760	0.0034280
10	40.3231	1.8066	-0.0028294

5.1.2 Transient Poiseuille flow

The transient solver needs to be verified to correctly capture the transient behavior of a case. This will be done by replicating a transient Poiseuille flow between parallel plates for which there exist an analytical solution [6]. This is performed by inducing a sudden pressure gradient on a standing still fluid in the horizontal channel presented in section 5.1. However, its solution only considers the hydrodynamic part of the problem. Therefore, an additional verification of the solver is conducted using the extended Graetz problem with the determined mesh from the Graetz extended problem analysis. Its steady-state solution is then compared to the solution of the Graetz extended problem.

The previous verification only identifies if the transient solver is able to reach the correct steady-state solution in the steady-state case. To also verify the transient behavior of the solver, the analytical solution of the transient Poiseuille flow is used. Its solution is given as [6]:

$$u(y,t) = -\frac{1}{2\mu}\frac{\mathrm{d}p}{\mathrm{d}x}H^2\left\{1 - \left(\frac{y}{H}\right)^2 - \frac{32}{\pi^3}\sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{(2k-1)^3}\cos\left[\frac{(2k-1)y\pi}{2H}\right]\exp\left[-\frac{(2k-1)^2\pi^2\nu t}{4H^2}\right]\right\}, \quad (5.5)$$

where H is half the channel height and y is the vertical position determined from the middle of the channel. This equation neglects the gravity and assumes constant fluid properties. Additionally, the equation requires pressure gradient as input parameter instead of the previously implemented constant volumetric inflow rate. To find the pressure gradient corresponding to the volumetric flow rate of $1.8944 \times 10^{-7}\,\mathrm{m^3\,s^{-1}}$, one may use the steady-state solution of this equation [1]:

$$\Delta p = -\frac{12LQ\mu}{H^3 w}, \quad (5.6)$$

where w is the depth and L is the length. The pressure drop is calculated to be $12.11\,\mathrm{Pa}$. Accordingly, the inlet pressure is set to $100\,012.11\,\mathrm{Pa}$ while the outlet is left as $1 \times 10^5\,\mathrm{Pa}$ for this case. Moreover, as mentioned in the introduction of this subsection, the presented analytical solution of the transient Poiseuille flow does not consider the thermal solution. Consequently, the heat flux from the walls are excluded in this analysis.

5.2 Transformer pass

The previous section considered the horizontal channels between the windings. Zooming out to the full pass shown in Figure 5.2, one must consider the vertical channels. These are less complicated than the horizontal channels as they are wider and with one side at the outer adiabatic wall. Consequently, sharp temperature gradients will only be present at one side of their walls. Additionally, as mentioned in the Chapter 2 the HSTs are expected to be in the horizontal channels [38]. Therefore, using the same mesh structure at the vertical channels as the horizontal will result in an unnecessarily huge computational cost. Moreover, the transition from the outlet of the vertical channels to the inlet of the horizontal channels as well as the opposite, should also be considered. In order to consider the accuracy of a mesh for the transformer pass, a test case is needed. Therefore, a case is created by allocating the inlet and the outlet of this pass in the same way as Torriano et al. [43]. That is, inlet at the bottom of the left vertical channel, and the outlet at the top of the right vertical channel. For this geometry, a constant volumetric inflow rate of $3.7888 \times 10^{-6}\,\mathrm{m^3\,s^{-1}}$ is used, which is the same as in Ref. [43] [2]. Nonetheless, the rest of the boundary and initial conditions are kept the same as the channel case section 5.1. However, for simplicity only the steady-state solution with constant fluid properties are considered, and the gravity is neglected.

[2]Due to the different depth that is been employed, the inlet velocity is used with the cross-section area of the inlet to determine the constant volumetric inflow rate.

Unfortunately, there are no analytical solution for this case with this complex geometry. Therefore, a reference result is simulated using the steady-state solver with constant fluid properties and a very fine mesh. This mesh consists of square, uniform cells with cell lengths of $0.1\,\mathrm{mm}$.

5.2.1 Mesh verification

The reference case will be used to verify the new mesh that will be specified later in this section, as it consist of a very fine mesh. For this verification analysis, the relative error of the pressure drop from the inlet to the outlet between the reference and the new mesh case will be used.

The new mesh structure of the transformer pass will be based on the obtained mesh structure from the grid convergence analysis of the horizontal channel. However, a symmetrical expansion factor of 1.2 is applied on the horizontal channels toward the middle, by keeping the same number of cells. This is done to better capture the inlet and outlet effect. Moreover, the mesh at the vertical channels are assigned an expansion factor of 1.2 towards the outer walls for the first 2/3 of their horizontal lengths. Then, a decrease by the inverse of 1.2 is introduced for the rest of the lengths. With this, the inlet and outlet effects, in addition to the temperature gradient from the winding heat flux and the velocity gradient from the walls are expected to be better resolved. For the regions of the vertical channels that are not connected by the (structured) mesh of the horizontal channels, a symmetric vertical expansion factor of 1.1 is used towards the middle.

5.2.2 Single pass validation

Torriano et al. [43] investigated the HST and its location in a single transformer pass at steady-state conditions. Their results have been successfully validated through experiments. These results can therefore be used to validate the implementation of the case with the single transformer pass. The boundary conditions and geometry is specified similar to Torriano et al. [43] and have been described earlier. The initial and inlet temperatures are set to $319.9\,\mathrm{K}$ and the gravity is included. Additionally, the fluid properties in this case are

$$\mu(T) = 0.08467 - 4.0 \times 10^{-4}T + 5.0 \times 10^{-7}T^2, \tag{5.7a}$$

$$\kappa(T) = 0.1509 - 7.101 \times 10^{-5}T, \tag{5.7b}$$

$$c_p(T) = 807.163 + 3.58T, \tag{5.7c}$$

$$\rho(T) = 1098.72 - 0.712T. \tag{5.7d}$$

The solution obtained for this case from the steady-state solver is then validated by comparing the resulting HST and its location to the results of Torriano et al. [43].

5.3 Porous-medium approximation

The LV loop consist of 4 passes stacked upon each other. However, only the HST and its location are of interest for the cold-start problem, which is the focus of this thesis. This

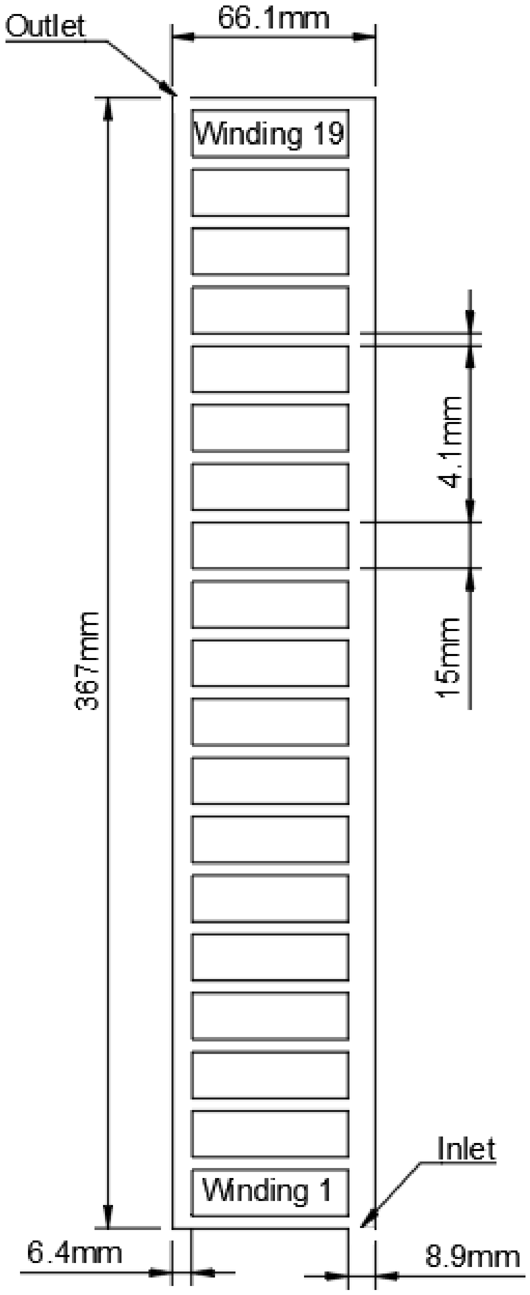


Figure 5.2: The geometry of the single pass.

HST is assumed to be located in the top pass [45]. It is therefore unnecessary to fully resolved the 3 bottom passes. A porous-medium approximation presented by Meyer et al. [29] and described in Chapter 3 is used to approximate these bottom passes.

The implementation of this approximation need to be verified. For this a single porous-medium approximated pass will be considered, as the geometry for each pass is the same. Furthermore, only the macroscopic accuracy of the approximated passes are of interest, particularly the top-oil temperature, defined as,

$$TOT = \frac{\int \rho c_p T \boldsymbol{u} \cdot \mathrm{d}\boldsymbol{A}}{\int \rho c_p \boldsymbol{u} \cdot \mathrm{d}\boldsymbol{A}}. \tag{5.8}$$

Here, $\boldsymbol{A}$ is the surface-normal vector of the outlet of the pass. The TOT is a critical parameter for the lower passes as it determines the average temperature that is transported to the upstream passes. Hence, the TOT will be used to verify the implemented porous-medium approximation by comparing to a fully-resolved simulation. Both cases will be run using the same parameters as defined in section 5.2. However, the case is modified to capture the transient evolution of the TOT by using the transient solvers, as that is of importance for the cold-start problem. Additionally, a uniform, square mesh with cell lengths of $1\,\mathrm{mm}$ is used for the porous-medium approximated case. This reduces the number of cells by about $53\,\%$ compared to the fully-resolved case.

5.4 Cold start of LV loop

In this section, the cold-start problem of the transformer in a cold environment will be described. For this problem the HST and its location are of interest, as they provide valuable insight about its cold-load capabilities.

As discussed in Chapter 1, most of the heat loss in a transformer will be in the LV windings. Accordingly, only the passes enclosing the LV windings are considered in this simulation. Moreover, some assumptions have to be made to fully specify this model. Thus, a square canal is implemented to create a closed loop between the outlet and the inlet of the stacked LV passes. In addition, an artificial radiator is placed near the inlet of the canal to cool down the oil. Its only function is to reset the temperature of the oil to a predefined value. This geometry as a whole is denoted as *LV loop* and is presented in Figure 1.2.

The different components of the LV loop are implemented according to the parameters determined in their respective verification studies. Nevertheless, a grading factor of approximately 1.1 is applied in order to ensure a smooth transition between the separately considered regions. The canal is implemented using almost uniform cells where 8 cells is used in its width of $8.9\,\mathrm{mm}$.

The oil inside this geometry will initially be set to $253.15\,\mathrm{K}$ to reproduce an extreme cold-start situation in a cold environment. The radiator at the inlet of the canal is assigned to reset the oil temperature to $293.15\,\mathrm{K}$. Both the canal and the radiator are model uncertainties. The influence of them will therefore be assessed in a sensitivity analysis in the final subsection.

5.4.1 Verification of LV loop

The implementation of this case is verified by running the simulation until steady-state conditions appear. The residuals are inspected to identify if they are below the set tolerances. Additionally, a mass and energy balance analysis will be conducted in a similar way as the compressible channel case presented in **??**.

5.4.2 Validation of LV loop

Validating the solution is of paramount importance as the input and model uncertainties have to be resolved. Thus, the LV loop will have to be validated. The solution of the case considered by Torriano et al. [43] is used for this study to examine if similar steady-state solution is obtained. However, they only consider a single transformer pass with predefined inlet and outlet conditions as presented in section 5.2.2. That said, these are set to recreate normal working conditions at steady state. Hence, similar conditions should be observed when resolved the LV loop.

To get comparable results, the same oil is used here as in section 5.2.2. Its properties are specified in Eq. (5.7). The steady-state results of both the mass flow rate and the TOT are used as a comparison in this validation[3].

5.4.3 Cold-start problem

Here, the results from the cold-start simulation will be presented. As mentioned in the introduction of this section, the time evolution of the HST and its location are of interest as they provide valuable insight about the load capabilities of the transformer. Thus, the time trace of the HST will be graphed, and a time series of the temperature distribution will be visualized.

5.4.4 Sensitivity analysis

The LV loop model contains a few model uncertainties, in particular the length and width of the canal and the assigned reset temperature and the drag of the radiator. This sensitivity analysis will only consider the radiator reset temperature. However, by using the steady-state Poiseuille flow solution given in Eq. (5.6) a sensitivity parameter can be constructed as follows:

$$\beta = \mu_\mathrm{p} \frac{L_\mathrm{p}}{h_\mathrm{p}^2}. \tag{5.9}$$

Here, μ_p is the dynamic viscosity at the radiator reset temperature, whereas L_p is the total length of the canal and h_p is the width of the canal. The sensitivity parameter β determines the pressure drop divided by the velocity for a straight canal section, according to the steady-state Poiseuille flow solution. Therefore, if this parameter is constant for two different cases, the pressure drop in the canal will be the comparable assuming the velocity is the same.

[3]Another depth is used for this geometry compared to Ref.[43]. This is therefore compensated for by multiplying their mass flow rate to the ratio between the inlet area for this geometry and the inlet area for their geometry, when obtaining their mass flow rate in this comparison.

The radiator reset temperatures used in this analysis will be 313.15 K and 283.15 K. These will be compared to the reset temperature of 293.15 K, considered in the previous section.

Chapter 6

Results and Discussion

The cases that have been described in the previous chapter will be employed in this chapter. That is, using the Graetz extended problem to verify the steady-state solver and the mesh in the horizontal channel in addition to the temperature-dependent fluid properties. Then, the transient Poiseuille solution is used to verify the transient solver by considering the same channel. This will be followed by verification and validation of the single transformer pass using its reference case and the case considered in Ref.[43], respectively. This case will then be used to verify the porous-medium approximation. Lastly, the cold-start case is considered with a verification and validation study, continued by its relevant results and ending with a sensitivity analysis.

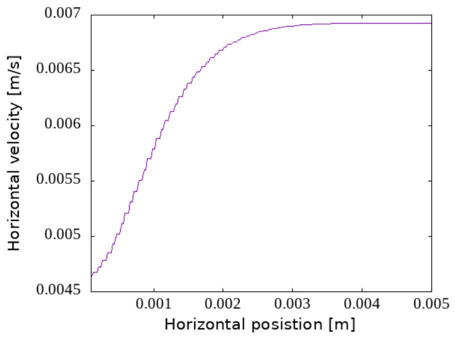
6.1 Horizontal channel flow

Section 5.1 presents and describes two flow problems within simple horizontal channel. The extended Graetz problem involves both thermal and hydrodynamical elements, while the transient Poiseuille flow only involves flow patterns. Both problems have analytical solutions and are useful to verify the implementation of the governing equations in the OpenFOAM framework.

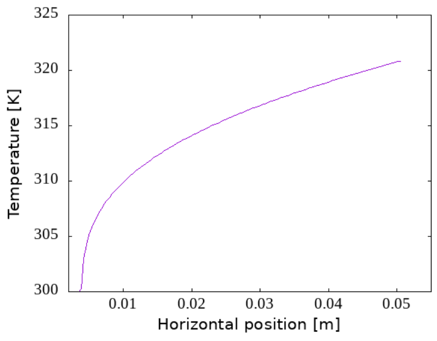
In the following, results are presented first for the Graetz extended problem, then for the transient Poiseuille flow.

6.1.1 Graetz extended problem

The solution of the presented extension of the Graetz problem is used to verify the implementation of the solver by conducting a grid convergence study. Then, different mesh structures are assessed to investigate the possibility to reduce the number of cells and therefore the computational cost. However, the solution of this extension of the Graetz problem is as discussed only valid for the hydrodynamically fully developed and thermally developing region. Fortunately, this region is estimated from the empirical correlations of the entry lengths. Nonetheless, the hydrodynamic and thermal entry length correlations



(a) Horizontal velocity along the middle of the channel.



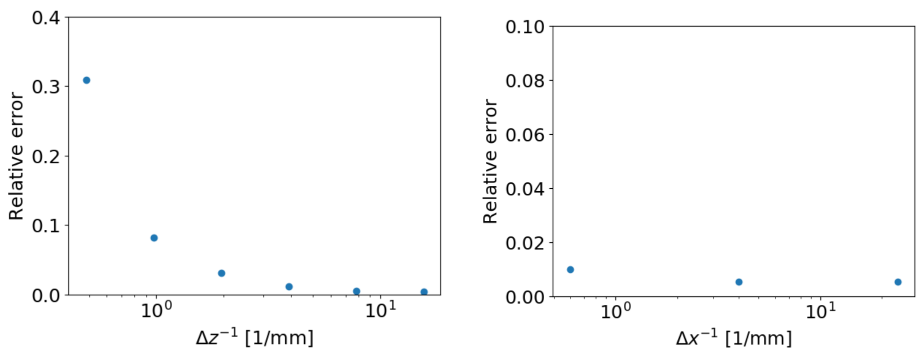
(b) Temperature along the wall of the channel.

Figure 6.1: Results for the uniform mesh with 64 cells in the transverse direction.

are empirical and assumes uniform inlet profiles. Hence, the determined entry lengths are only estimates. Therefore, a test case with a fine mesh consisting of square, uniform cells with 64 cells in the vertical direction is initially run to verify the determined entry lengths. The hydrodynamically entry length is determined by investigating where the velocity along the middle of the channel, visualized in Figure 6.1a, become uniform. From this figure, it is seen that the velocity is independent of the horizontal position at about $3.5\,\mathrm{mm}$ of the total $50.8\,\mathrm{mm}$, which is a little above the estimated value in section 5.1.1 of $3.01\,\mathrm{mm}$. Such a small deviation is however reasonable, because the previously estimated value is both empirical and assumes an uniform inlet flow velocity. Moreover, Figure 6.1b presents the temperature profile along the wall. This profile shows that the temperature at the wall does not become linear, confirming the thermally developing assumption done in section 5.1.1. Therefore, the first $4\,\mathrm{mm}$ of the inlet will be excluded from the following analysis, to replicate the extended Graetz problem.

The grid convergence study is conducted with square, uniform meshes. Simulations are first run on a coarse square mesh which is incrementally refined by a factor of 2. This mesh refinement process is performed until the solution is observed to have converged to solution of the extended Graetz problem. The resulting relative errors for the different grid resolutions are given in Figure 6.2a. These error are calculated with respect to the temperature difference between the inlet and wall. In addition, linear interpolation is used for the simulated cases to estimate the wall temperature from the temperature of the cell closest to the wall. Looking at Figure 6.2a the convergence is obvious. Thus, the implementation is verified. Additionally, this figure reveal that 32 cells in the vertical direction is sufficient to capture the flow characteristics, as the relative error is only $0.53\,\%$. Hence, 32 cells in the vertical direction will be used for all subsequent simulations.

The uniform mesh with 32 cells in the vertical direction is then relaxed in the flow direction, because the aspect ratio is allowed to be high in the flow direction. The relative errors for different relaxations that are determined in a similar way as for the uniform meshes, are visualized in Figure 6.2b. Here, negligible differences are seen by relaxing the number of cells in the flow direction of the channel. This is a reasonable, since there are only small gradients in this direction. Therefore, only 30 cells will be used in this



(a) Uniform meshes. **(b)** Horizontal relaxation of the number of cells.

Figure 6.2: Grid convergence tests with the extended Graetz problem, using interpolated values at the cells.

direction for the following simulations.

Furthermore, it should be remarked that fine cells near the walls are preferred due to the large gradients. However, this will cause problems when including the vertical channels, since the structured mesh will cause the flow to be in the direction with the low aspect ratio of the cell at the vertical channels. Hence, the time steps would have been unnecessarily small due to the stability criteria of the Courant number for the employed PISO algorithm. Grading in the transverse direction of the horizontal channel is therefore not considered.

The determined mesh resolution is only verified for this specific case. That said, the input parameters are determined from a real ONAN transformer in Ref.[43]. As the same geometry will be employed for the cold-start problem, the flow characteristic will be comparable.

In conclusion, this study reveal that the steady-state solver has been successfully implemented. In addition, a good balance between accuracy and cell count can be achieved by using 30 cells in the flow direction and 32 cells in the vertical direction for this case. This mesh is shown in Figure 6.3. The flow characteristic will be similar for the cold-start problem. This mesh will therefore be used as a basis for all subsequent simulations that include the horizontal channels.

The implemented temperature-dependent fluid properties are verified using the case with the Graetz extended problem. This is initially performed by verifying that the mass and energy are conserved in the channel. The former is investigated by subtracting the mass flow at the inlet to the outlet. Whereas the latter is determined by subtracting inlet to the outlet enthalpy, kinetic and potential energy flow rate, and adding the heat transfer subtracted with the pressure work due to the friction at the wall. The results from these analyses reveal that the mass is perfectly conserved, whereas a relative error of $0.014\,\%$ is found for the energy conservation with respect to the total heat transfer from the walls. Hence, the initial verification of the implemented temperature-dependent properties is proven successful, as either the mass or the energy can be created from nowhere. The conserved mass and energy also verify that the convergence criteria of the residuals and that the assump-

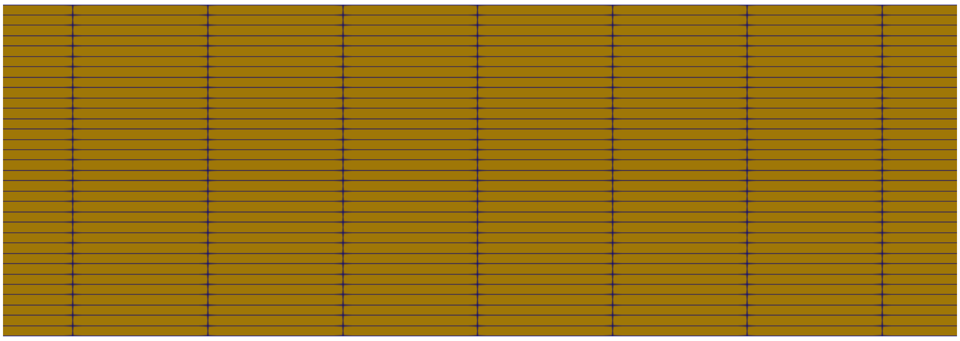


Figure 6.3: A close-up of the resulting mesh from the grid convergence study of the horizontal channel.

tions made in the simplified energy equation, Eq. (3.3), are valid [48, page 90-92].

Furthermore, the values of the temperature-dependent fluid properties should be verified to correspond to the temperature at their cells according to Eq. (5.1). For this, the bottom cell at the end of the channel is used. Its temperature is found to be $316.5\,\mathrm{K}$. Which from Eq. (5.1) should correspond to $951.0\,\mathrm{kg\,m^{-3}}$, $2.314 \times 10^{-2}\,\mathrm{Pa\,s}$, $0.1428\,\mathrm{W\,K^{-1}\,m^{-1}}$ and $1936\,\mathrm{J\,kg^{-1}\,K^{-1}}$ for respectively the density, dynamic viscosity, thermal conductivity coefficient and the specific heat capacity. The resulting values from the simulation for these properties at that cell are found to be exactly the same as the expected values. Consequently, the implementation of the temperature dependent fluid properties are successfully verified.

6.1.2 Transient Poiseuille flow

The Graetz problem was used to verify the steady-state solver and to analyse the lengths of the thermal and hydrodynamic developing regions. A sufficiently accurate mesh for that particular case was determined. This is visualised in Figure 6.3. This mesh will be used in the horizontal channel presented in Figure 5.1 to verify the transient solver. The verification will be conducted in two steps. First, the steady-state solution of the Graetz extended problem using the transient solver will be verified. Then, the transient behavior of the solver will be verified using the transient Poiseuille flow solution. This is done by applying a sudden pressure gradient on the oil from a standing still position.

The implementation of the transient solver is initially investigated by replicating the Graetz extended problem of the previous case in section 6.1.1, using the transient solver. As mentioned in section 5.1.2, this investigation is conducted since the transient Poiseuille flow that will later be employed does not consider the thermal solution. The steady-state solution of the linearly interpolated temperature profile along the wall is then compared to the analytical solution. This result is presented in Figure 6.4. The figure shows that the steady-state result from the simulation matches the analytical solution, verifying that the implemented transient solver is able to reach the correct steady-state solution in the steady-state case.

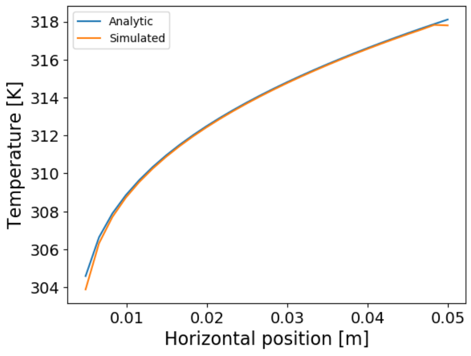
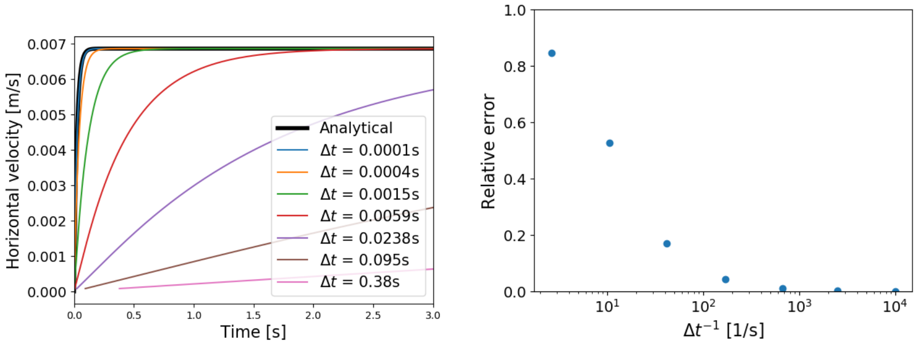


Figure 6.4: Temperature profile along the wall of the channel, using the transient solver.

The transient behavior of the solver is then verified by comparing the results of decreasing time steps to the analytical transient Poiseuille solution of parallel plates. This will also be used to estimate the time step for the cold-start simulation. For this study a time step for the first convergence iteration has to be determined. As the PISO algorithm is used in the transient solver, the Courant number needs to be below 1 to ensure stability. It should be noted that the implemented solver `buoyantPimpleFoam` has the ability to change the time step based on an maximum Courant number and time step. Nevertheless, this feature is not used in this convergence study since the Courant number is dependent on both the local cell size and velocity as shown inEq. (4.2). A Courant number, that is found from this study to provide accurate results, will therefore not be characteristic for the cold-start phenomena using the whole LV pass. For this purpose, a time step found in a similar way, will be more representative. That said, the Courant number must be considered to make sure that it never exceeds unity. This number depends on both the local cell length and velocity. Fortunately, the maximum velocity for this Poiseuille flow is known and given as 1.5 times the average velocity [1], resulting in $4.44 \times 10^{-3}\,\mathrm{m\,s^{-1}}$. In addition, as the resulting uniformly distributed mesh from the previous mesh convergence study is employed, the maximum cell length is $1.69\,\mathrm{mm}$ in the flow direction. Thus, Eq. (4.2) implies a time step of $0.38\,\mathrm{s}$ for the first iteration. This value is reduced by a factor of 2 until reasonable convergence to the analytical solution is observed. The results of this investigation is shown in Figure 6.5. The figure shows that the transient behavior of the solver reaches the analytical solution as the time step goes towards zero. This indicates that the transient solver is correctly implemented. Furthermore, the plots reveals that the time step has to be in an order of $1 \times 10^{-3}\,\mathrm{s}$ to capture the transient behavior with sufficient accuracy. Nevertheless, the considered transient Poiseuille flow Eq. (5.5) does not use the energy conservation equation Eq. (3.4c) and is only valid for this specific case. That adds an extra layer of error. However, a time step in the order of $1 \times 10^{-3}\,\mathrm{s}$ still provides a reasonable indication of what the time steps should be in the final cold-start simulation.



(a) Velocity evolution for different time steps.

(b) Relative errors for different time steps.

Figure 6.5: Velocity at the middle of the outlet.

6.2 Transformer pass

Now the vertical channels are merged with the previously considered horizontal channels to create a single transformer pass as described in section 5.2. A new mesh is generated for this geometry. However, to verify that the mesh is sufficiently resolved for the problem at hand, a reference case is made for comparison. This, in addition to a validation case will be the main focus in the following subsection.

The case that has been described in section 5.2 is initially investigated for the presence of turbulence, since Gastelurrutia et al. [17] stated that their ON transformer had some turbulent regions. This investigation is conducted by calculating the Reynolds number using Eq. (5.3a) for both the inlet and the outlet of the transformer pass. These are estimated using the determined constant properties in Chapter 5 with the volumetric inflow rate for this case, which is $3.7888 \times 10^{-6}\,\mathrm{m^3\,s^{-1}}$. By using this, in addition to the width of respectively 6.4 and 8.9 for the inlet and outlet, result in Reynolds numbers of 6.18 and 5.37 for respectively the inlet and outlet. Additionally, the previously calculated Reynolds number for the horizontal channel was 0.360. These values are far below the transition region from laminar to turbulence flow at about 2×10^3. It is therefore safe to assume that the fluid flow in this transformer pass is purely laminar. This is also verified, running the reference case, as no instabilities are observed for the residuals. Because the turbulent effects will cause instabilities, and lead to major convergence issues for the residuals.

Furthermore, the results from the mass and the energy balance analysis conservation show that the mass is perfectly conserved, while the relative error for the energy conservation is $0.020\,\%$ with respect to the total heat transfer from the windings at $58.42\,\mathrm{W}$. Thus, indicating that the mesh and the set tolerances for the convergence of the residuals are legitimate.

Table 6.1: Comparison of some parameters for different meshes in a single transformer pass.

	Pressure drop [Pa]	TOT [K]	HST [K]
Reference	3658.4	308.4	361.4
New mesh	3653.3	308.4	360.4

6.2.1 Mesh verification

A mesh is created for the single pass geometry, using the specifications presented in section 5.2.1. Which is by beginning with the resulting mesh for the horizontal channels from the analysis of the extended Graetz problem. That is the That is 32 and 30 uniformly distributed cells in respectively the height of $4.1\,\mathrm{mm}$ and length of $50.8\,\mathrm{mm}$ of the channel. The different grading factors are then applied. The resulting mesh is presented in Figure 6.6. As mentioned in the introduction of this section, a reference case is created to verify that this mesh is sufficiently resolved for the problem in hand. This case is made using a very fine mesh, consisting of square, uniform cells with lengths of $0.1\,\mathrm{mm}$.

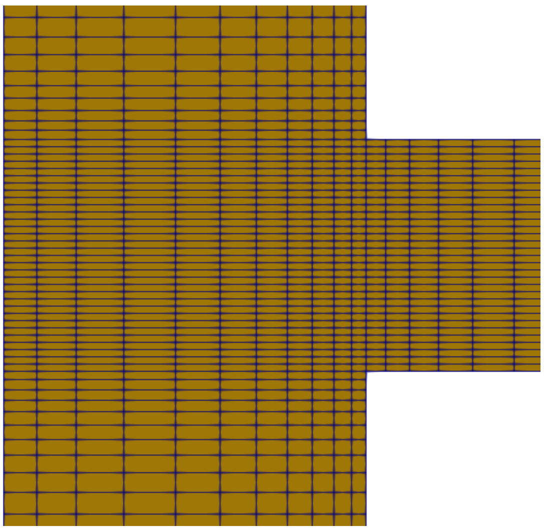


Figure 6.6: Details of the mesh near intersection between vertical and horizontal channels.

The results for the HST, TOT and pressure drop between the inlet and the outlet by using the new mesh and the reference mesh are presented in Table 6.1. The table shows that the TOT is the same in both cases, whereas the relative errors for both the pressure drop and the HST are small. This shows that the new mesh is sufficiently resolved for problem at hand.

6.2.2 Single pass validation

As discussed in section 5.2.2, Torriano et al. [43] performed CFD simulations of a steady-state single transformer pass. They found the HST to be $376.6\,\mathrm{K}$. The HST was located at the 16^{th} winding counting from the bottom. They have successfully compared this HST with fiber-optic measurements.

The fluid properties used in the case is specified in Chapter 5. The same simulation is performed here, and the results are presented in Figure 6.7. The figure shows the resulting maximum temperature at each winding. It is observed that the HST is located at the 17^{th} winding, where the temperature is $379.9\,\mathrm{K}$. This gives a relative error of only $0.64\,\%$ with respect to the HST found by Torriano et al. [43]. Whereas, the difference in the location is only of one winding. In addition, the maximum temperature at the 16^{th} winding is just marginally below the HST. The result therefore indicates that the current model provides physically adequate solutions.

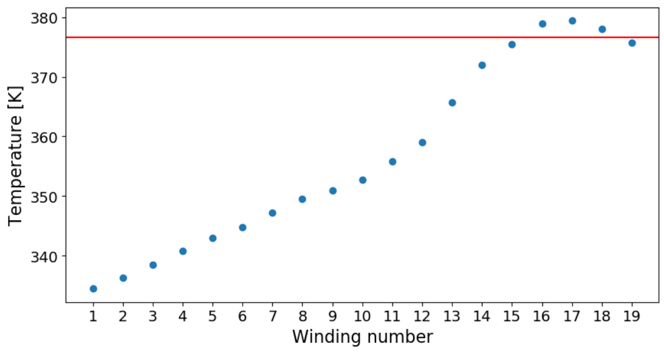


Figure 6.7: Maximum temperature at the different windings for the case presented by Torriano et al. [43] marked by blue dots. While the HST from Torriano et al. [43] is visualized by the red line.

6.3 Porous-medium approximation

The porous-medium approximation is implemented to significantly decrease the computational resources needed to simulate the cold-start problem. This implementation will be verified by comparing a porous with a fully resolved single transformer pass.

The porous simulation is run until steady-state conditions are achieved. The resulting time evolution of the TOT for this simulation is then compared to the TOT of the equivalent fully resolved simulation. The results are presented in Figure 6.8 and show somewhat faster dynamics for the porous case as the maximum TOT appears about $10\,\mathrm{s}$ earlier. The maximum deviation for TOT is only $1\,\mathrm{K}$, which gives a relative error with respect to the fully resolved TOT of about $0.3\,\%$. Additionally, the steady-state TOT is almost the same, as only a temperature difference of $0.1\,\mathrm{K}$ is observed. The porous-medium approximated pass is therefore verified, since these differences are small.

The motivation for using a porous medium approximation is to reduce the computational time. In this case, the fully-resolved simulation uses $8092\,\mathrm{s}$ to reach $300\,\mathrm{s}$ of simu-

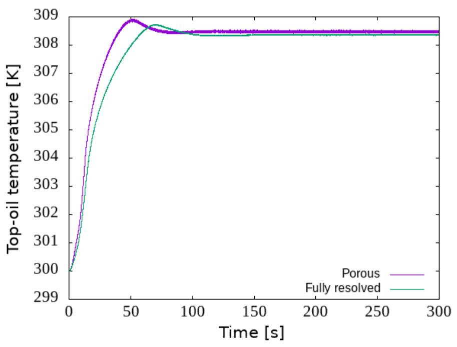


Figure 6.8: Comparison between porous and full resolved pass with respect to the TOT.

lation time, while the porous-medium approximated simulation only uses $735\,\mathrm{s}$. Hence, a reduction in compute time of remarkable $91.2\,\%$ is gained. That said, the given reduction only applies for this specific case, as part of this reduction is due to the increased time step in the porous case. Which is as a consequence of implemented larger cells combined with the use of maximum Courant number to determine the time steps. Therefore, when later considering the whole LV winding which has both fully-resolved and porous passes, the gain with increased time step for the porous case will mostly vanish. Nonetheless, as the number of cells is reduced by $53\,\%$ in the porous case compared to the fully-resolved, the reduction in compute time will still be significant.

6.4 Cold-start of LV loop

The previous cases has shown that the current model and its implementation should be able to simulate the passes enclosing the LV windings as described in section 5.4.3. In this section, the implementation of the cold-start problem will be further verified and validated. Then, the simulation results for the cold-start case of the LV loop will be presented and discussed. This is followed by a sensitivity analysis that investigates the radiator reset temperature.

In addition to the mentioned case description of the cold-start problem given in section 5.4.3, the maximum time step is set to $1 \times 10^{-2}\,\mathrm{s}$ based on to the results in section 6.1.2. However, the maximum Courant number remains at 0.9 to ensure stability.

6.4.1 Verification of LV loop

The cold-start simulation is verified by conducting mass and energy balance analyses for the steady-state solution. These analyses show that the mass is perfectly conserved, while the relative error for the energy conservation is $0.81\,\%$ with respect to the total heat transfer from the windings at $233.68\,\mathrm{W}$.

Table 6.2: A comparison between the TOT and the mass flow rate in Torriano et al. [43] and the steady-state results for the cold-start validation simulation.

	Mass flow rate [$\mathrm{kg\,s^{-1}}$]	TOT [K]
Simulation of Torriano et al. [43]	3.30×10^{-3}	328.3
The validation simulation	3.27×10^{-3}	331.6

6.4.2 Validation of LV loop

It is important to ensure that the implemented cold-start case is physically accurate. The results of Torriano et al. [43] will again be used as a reference. Since Torriano et al. [43] considers the normal working condition at steady-state for this transformer, comparable results should be obtained by adapting to the same fluid properties as used here.

Table 6.2 shows the resulting TOTs and mass flow rates for the present simulation and the simulation performed by Torriano et al. [43][1]. The differences in the TOT and mass flow rates are $3.3\,\mathrm{K}$ and $3 \times 10^{-5}\,\mathrm{kg\,s^{-1}}$, respectively. These correspond to relative errors of $1.0\,\%$ and $0.9\,\%$, respectively. The low relative errors indicate that the physics is well captured.

6.4.3 Cold-start problem

In this subsection, the results from the cold-start simulation is presented. The parameters and configurations of this simulation are specified in section 5.4.3.

In this simulation, the magnitude of the HST and its time duration are the critical output parameters as they provide an indication of the degradation of the transformer [37]. The time evolution of the HST is plotted in Figure 6.9. In addition, the locations of the HSTs are of interest as they provide valuable information on where the transformer may fail. A time series of the temperature distribution in the LV loop is visualized in Figure 6.10. The figure shows that high temperatures are found in the top two passes. The highest temperatures, the HSTs, are observed at around $500\,\mathrm{s}$. This is confirmed by Figure 6.9, which displays a peak in maximum HST of $447.2\,\mathrm{K}$ at time $535\,\mathrm{s}$. This temperature is detected at the 11^{th} winding of the top pass.

As the observed maximum HST is above $413\,\mathrm{K}$, the transformer can be expected to experience significant degradation [37]. In addition, this temperature level is dangerously close to the temperature where a sudden transformer failure may occur. However, this failure temperature is very dependent on the type of transformer insulation being used, and some are rated to withstand temperatures up to $493\,\mathrm{K}$ [37]. The steady-state HST is $425.5\,\mathrm{K}$ as shown in Figure 6.9 and is located at the same 11^{th} winding on the top pass. The HST of the oil will therefore at steady-state operation always be above the temperature level where significant degradation is expected, given this geometry and heat flux. The steady-state result of the HST is not coherent with the conducted literature review, as none

[1]Another depth is used in the geometry of this simulation compared to Ref.[43]. This is compensated by multiplying their mass flow rates to the ratio between the inlet area of this geometry and the inlet area of their geometry, when obtaining their mass flow rates in this comparison.

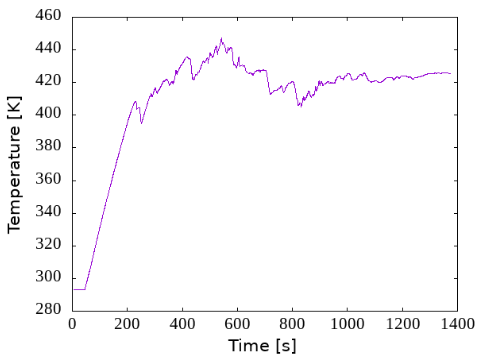
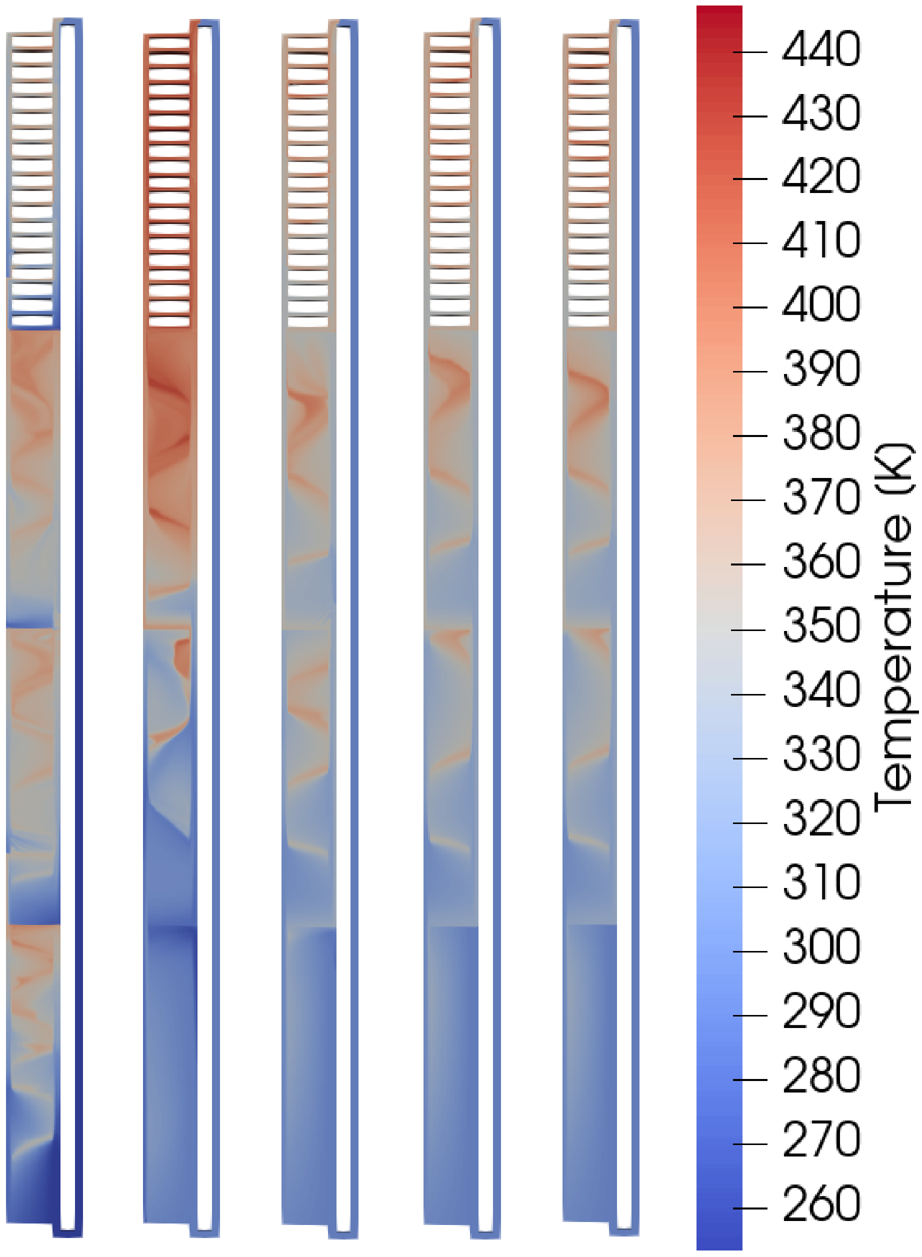


Figure 6.9: Temperature evolution of a cold-start case with radiator reset temperature set to $293.15\,\mathrm{K}$ and initial temperature assigned to be $253.15\,\mathrm{K}$.

of the considered studies have reached HSTs above $383.15\,\mathrm{K}$ during normal, steady-state operations. A further investigation of this case is therefore conducted.

This investigation shows that the dynamic viscosity at $293.15\,\mathrm{K}$ of the considered `MIDEL7131` is $7.3 \times 10^{-2}\,\mathrm{Pa\,s^{-1}}$, while the oil considered by Torriano et al. [43] had a dynamic viscosity of $1.0 \times 10^{-2}\,\mathrm{Pa\,s^{-1}}$ at the same temperature. A much more significant amount of drag is therefore present in the simulated case. In addition, the dynamic viscosities at $333.15\,\mathrm{K}$ are $1.3 \times 10^{-2}\,\mathrm{Pa\,s^{-1}}$ and $6.9 \times 10^{-3}\,\mathrm{Pa\,s^{-1}}$ for the `MIDEL7131` and the oil considered by Torriano et al. [43], respectively. The viscosity of `MIDEL7131` is therefore much more temperature dependent. Thus, also more sensitive to the assumed canal geometry and radiator reset temperature. A probable explanation for these unrealistically high temperatures are therefore that the assigned radiator reset temperature of $293.15\,\mathrm{K}$ and the canal geometry are not completely realistic. To determine the influence of these model uncertainties a sensitivity analysis will be conducted in the final subsection.

To further determine if the cold-start situation in a cold environment may be a problem in terms of transformer degradation, a new test case is developed where an initial temperature of $293.15\,\mathrm{K}$ is used instead of $253.15\,\mathrm{K}$. The remaining parameters are kept similar. This case is created to analyse the maximum HST as well as the increase in the HST compared to its steady-state value between the two cases with the different initial temperatures, hereon called *HST rise*. The resulting plot of the time evolution of the HST for this test case is presented in Figure 6.11. The figure shows that the steady-state HST is $426.5\,\mathrm{K}$ and its maximum value only rises with $3.5\,\mathrm{K}$ reaching $430\,\mathrm{K}$. Whereas the case with an initial temperature of $253.15\,\mathrm{K}$, this value is determined to be $21.7\,\mathrm{K}$. Thus, the rise in HST is significantly higher during a cold-start from $253.15\,\mathrm{K}$ compared to $293.15\,\mathrm{K}$. Additionally, the maximum HST is higher in the case initiated from a colder state. A cold-start in a cold environment may therefore be problematic in terms of high HST. Therefore, further investigation is needed.



(a) 250 s. **(b)** 500 s. **(c)** 750 s. **(d)** 1000 s. **(e)** 1250 s.

Figure 6.10: Time series of the temperature profile for the case with radiator at the top of the canal and the reset temperature set to 293.15 K.

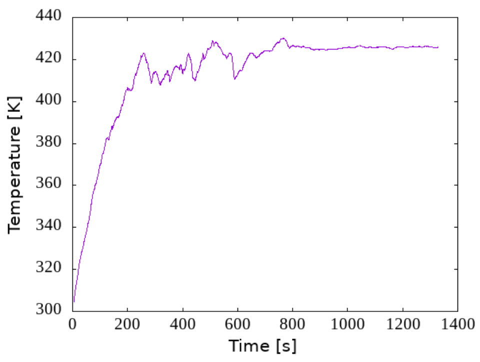


Figure 6.11: The hot-spot temperatures for a cold-start case from an initial and radiator reset temperature of $293.15\,\mathrm{K}$.

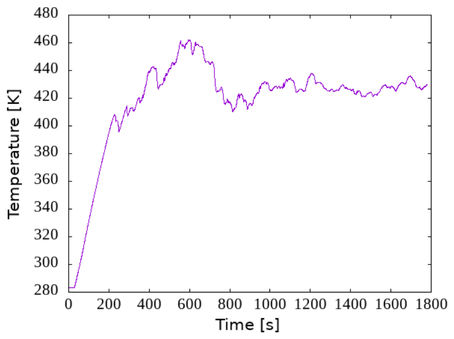
6.4.4 Sensitivity analysis

The influence of the radiator reset temperature and thus also an indication of the influence from the canal width and length through the sensitivity parameter β, will be used in this sensitivity analysis. The resulting time trace of the HSTs using a reset temperature of $283.15\,\mathrm{K}$ and $313.15\,\mathrm{K}$ are visualized in Figure 6.12a and Figure 6.12b, respectively. The maximum and steady-state HSTs, and the difference between them for the different radiator reset temperatures, are given in Table 6.3.

This table shows that the HST rise are doubled for the cases with temperature reset of $283.15\,\mathrm{K}$ and $313.15\,\mathrm{K}$ compared to $293.15\,\mathrm{K}$. Revealing that the HST rise is very sensitive to changes in radiator reset temperature.

Moreover, the table shows that the HST increases significantly when the reset temperature is decreased. The reason for this is that the viscosity increases significantly when decreasing the temperature. This very viscous oil will then have to flow through the whole channel, leading to a significant drag force. The oil velocity will therefore be decreased, which results in insufficient cooling.

The β values are given in the same table. This value gives an indicator of the pressure drop divided by the velocity. Thus, if this parameter is constant for two different cases, the pressure drop in the canal should be the comparable assuming the velocity is the same. Accordingly, the case $\beta = 2646\,\mathrm{Pa\,s\,m^{-1}}$ gives an indication of what HSTs to expect if the reset temperature is kept to $293\,\mathrm{K}$ and increased the $L_\mathrm{p}h_\mathrm{p}^2$ by $82\,\%$. From Table 6.3, it is shown that the max HST follows β as it is increased when increasing the β and similar for a decrease. Revealing promising results for the assumptions with the introduction of β.


(a) Radiator reset temperature set to $283.15\,\mathrm{K}$.

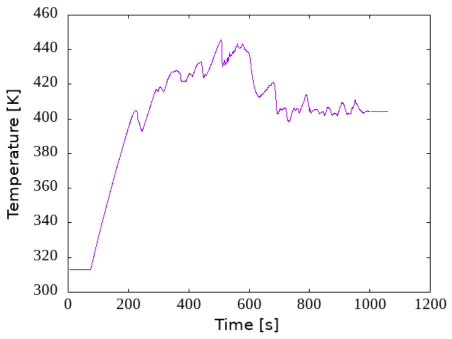

(b) Radiator reset temperature set to $313.15\,\mathrm{K}$.

Figure 6.12: HST for different radiator reset temperatures.

Table 6.3: Some critical parameters for several radiator reset temperature simulations.

Reset [K]	Max HST [K]	Steady-state HST [K]	HST rise [K]	β $[\mathrm{Pa\,s\,m^{-1}}]$
283.15	463	425.5	43.0	2646
293.15	447.2	425.5	21.7	1453
313.15	445.7	404.3	41.4	532.8

Chapter 7

Conclusion

The CFD model is used to study a transformer's cold-start problem in a cold environment. The cold environment is assumed to be at $253.15\,\mathrm{K}$. A section of the transformer where the hot-spot temperature (HST) is likely to be, is considered in this model. A simple canal with a simplified radiator is made to create a closed loop and to cool down the oil by resetting the temperature to a predefined value. The geometry and some input parameters are obtained from Ref.[43]. Due to the limited computational resources available and the complexity of the problem, both the computational time and accuracy of the solution is considered. Thus, a porous-medium approximation is used on the part of the domain where only the macroscopic properties are of interest, revealing significant simulation speed-up. This model is verified though the analytical solution of the Graetz extended problem and the transient Poiseuille flow, in addition to some reference simulations using very fine uniform meshes. The model shows comparable results with results found in Ref.[43].

The results from this simulation show a maximum HST of $447.2\,\mathrm{K}$, exceeding the temperature of $413\,\mathrm{K}$ where a typical transformer will experience significant degradation. Moreover, the maximum HST is observed to be $17.2\,\mathrm{K}$ higher during a cold-start initiated in a cold environment compared to an environment with mild temperature. This further indicates that a cold-start in a cold environment require careful consideration. However, the performed sensitivity analysis reveals that the radiator reset temperature may have a significant influence on the results. The pipe geometry is suggested to also have a similar significant impact. Further investigation is therefore needed to model the radiator and the pipe more accurately.

Bibliography

[1] 2.25 advancedfluidmechanics, 2008. URL https://ocw.mit.edu/courses/mechanical-engineering/2-25-advanced-fluid-mechanics-fall-2013/equations-of-viscous-flow/MIT2_25F13_Couet_and_Pois.pdf.

[2] Residuals, 2016. URL https://www.openfoam.com/documentation/guides/latest/doc/guide-solvers-residuals.html.

[3] Efficiency of transformer, 2018. URL https://www.electrical4u.com/efficiency-of-transformer/.

[4] 6.2 numerical schemes, 2018. URL https://www.openfoam.com/documentation/user-guide/fvSchemes.php.

[5] Midel7131 thermal properties, 2018. URL https://www.midel.com/app/uploads/2018/09/MIDEL_7131_Thermal_Properties.pdf.

[6] Fundamentals of non-newtonian fluid mechanics, 2018. URL http://euclid.mas.ucy.ac.cy/~georgios/courses/PMT17/Chapter6.pdf.

[7] Pour point analysis for the viscosity measurements of oils, 2018. URL https://www.azom.com/article.aspx?ArticleID=16493.

[8] Sharin Ab Ghani, Zulkarnain Ahmad Noorden, N.A. Muhamad, Hidayat Zainuddin, and Mohd Talib. A review on the reclamation technologies for service-aged transformer insulating oils. *Indonesian Journal of Electrical Engineering and Computer Science*, 10:426–435, 2018. doi: 10.11591/ijeecs.v10.i2.pp426-435.

[9] M. M. Adibi. *Distribution Transformer Overloading Capability under ColdLoad Pickup Conditions*. Wiley-IEEE Press, 2000. ISBN 9780470545607. doi: 10.1109/9780470545607.ch80.

[10] Theodore L. Bergman and Adrienne S. Lavine. *Incropera's Principles of Heat and Mass Transfer*. Wiley, 8 edition, 2017.

[11] H. M. R. Campelo, L. F. Braña, and X. M. Lopez-Fernandez. Thermal hydraulic network modelling performance in real core type transformers. *2014 International Conference on Electrical Machines (ICEM)*, pages 2275–2281, 2014. doi: 10.13140/2.1.2783.9369.

[12] R. D. Cess and E. C. Shaffer. Heat transfer to laminar flow between parallel plates with a prescribed wall heat flux. *Applied Scientific Research, Section A*, 8(1):339–344, 1959. ISSN 1573-1987. doi: 10.1007/BF00411758. URL `https://doi.org/10.1007/BF00411758`.

[13] Bram Cloet, Pieter Jan Joraens, Jama Nuri, and Raymond Van Schevensteen. Cold start of a 5.5mva offshore transformer, 2014. URL `http://www.owi-lab.be/sites/default/files/Final_CG_paper%20Cold%20start%20and%20storage%20test%20on%205%205MVA%20WTG%20transformer.pdf`.

[14] C. Cotas, G. Nelson, and S. Ricardo et al. Development of a dynamic thermal hydraulic network model for core-type power transformers windings. In *SmarTHER CORE Transformers, Conference: CIGRE Session 2018, Paris, France*, 2018. URL `https://www.researchgate.net/publication/327745480_Development_of_a_dynamic_thermal_hydraulic_network_model_for_core-type_power_transformers_windings`.

[15] Edvard Csanyi. Large power transformer tailored to customers' specifications, 2013. URL `https://electrical-engineering-portal.com/an-overview-of-large-power-transformer-lpt`.

[16] F. Edstrom, J. Rosenlind, P. Hilber, and L. Soder. Modeling impact of cold load pickup on transformer aging using ornstein-uhlenbeck process. *IEEE Transactions on Power Delivery*, 27:590–595, 2012. ISSN 1937-4208. doi: 10.1109/TPWRD.2012.2185521. x.

[17] Jon Gastelurrutia, Juan Carlos Ramos, and Gorka S. Larraona et al. Numerical modelling of natural convection of oil insidedistribution transformers. *Applied ThermalEngineering, Elsevier*, 31:493–505, 2011. ISSN 1359-4311. doi: 0.1016/j.applthermaleng.2010.10.004.

[18] Chris Greenshields. Energy equation in openfoam, 2016. URL `https://cfd.direct/openfoam/energy-equation/`.

[19] Chris Greenshields. Openfoam v6 user guide: 4.5 solution and algorithm control, 2018. URL `https://cfd.direct/openfoam/user-guide/v6-fvsolution/`.

[20] Bernhard Gschaider. Darcyforchheimer, 2019. URL `https://openfoamwiki.net/index.php/DarcyForchheimer`.

[21] Tobias Holzmann. *Mathematics, Numerics, Derivations and OpenFOAM®*. 11 2019. doi: 10.13140/RG.2.2.27193.36960.

[22] IEC 60076-7:2018. Power transformers - part 7: Loading guide for mineral-oil-immersed power transformers. Standard, International Electrotechnical Commision, Geneva, CH, 2018.

[23] IEC 60354:1991. Loading guide for oil-immersed power transformers. Standard, International Electrotechnical Commision, Geneva, CH, 1991.

[24] IEEE C57.91-1981. Ieee c57.91-1981 - ieee guide for loading mineral-oil-immersed overhead and pad-mounted distribution transformers rated 500kva and less with 65 c or 55 c average winding rise. Standard, Institute of Electrical and Electronics Engineers, New Tork City, US, 1981.

[25] IEEE C57.91-2011. Ieee c57.91-2011 - ieee guide for loading mineral-oil-immersed transformers and step-voltage regulators. Standard, Institute of Electrical and Electronics Engineers, New Tork City, US, 2012.

[26] E. J. Kranenborg, C. O. Olsson, B. R. Samuelsson, and L. Lundin. Numerical study on mixed convection and thermal streaking in power transformer windings. In *Fifth European Thermal-Sciences Conf., The Netherlands*, pages 1–8, 2008. URL `http://www.kranenborg.org/jurjenCV/EuroTherm2008_CFD_TFO_diskwinding_study_ABB.pdf`.

[27] J. Kranenborg, A. Gustafsson, and T. Laneryd et al. Hot spot determination in transformer windings through cfd analysis. In *VII WORKSPOT-International workshop on power transformers, equipment,substationsand materialsRIO DE JANEIRO*, pages 1–8, 2014. URL `http://www.kranenborg.org/jurjenCV/Accurate_Hot_Spot_Calc-VII_CIGRE_Workspot_2014.pdf`.

[28] M. A. Laughton and D. F. Warne. *Electrical engineer's reference book*. Elsevier Science, 16 edition, 2003.

[29] Ole H. H. Meyer, Karl Yngve Lervåg, and Åsmund Ervik. A multiscale porous–resolved methodology for efficient simulation of heat and fluid transport in complex geometries, with application to electric power transformers. Submitted to Applied Thermal Engineering, 2020.

[30] V. M. Montsinger. Loading transformers by temperature. *Transactions of the American Institute of Electrical Engineers*, 49(2):776–790, 1930. ISSN 2330-9431. doi: 10.1109/T-AIEE.1930.5055572.

[31] S. P. Moore, W. Wangard, and K. J. Rapp et al. Cold start of a 240-mva generator step-up transformer filled with natural ester fluid. *IEEE Transactions on Power Delivery*, 30:256 – 263, 2014. ISSN 1937-4208. doi: 10.1109/TPWRD.2014.2330514.

[32] H. Nordman, N. Rafsback, and D. Susa. Temperature responses to step changes in the load current of power transformers. *IEEE Transactions on Power Delivery*, 18: 1110–1117, 2003. ISSN 1937-4208. doi: 10.1109/TPWRD.2003.817516.

[33] Keon-Je Oh and Soo-Seok Ha. Numerical calculation of turbulent natural convection in acylindrical transformer enclosure. *Heat Transfer—Asian Research*, 28:429ñ441,, 1999. doi: 10.1002/(SICI)1523-1496.

[34] L. W. Pierce. Predicting liquid filled transformer loading capability. *IEEE Transactions on Industry Applications*, 30:170–178, 1994. ISSN 1939-9367. doi: 10.1109/28.273636.

[35] Z. R. Radakovic and M. S. Sorgic. Basics of detailed thermal-hydraulic model for thermal design of oil power transformers. *IEEE Transactions on Power Delivery*, 25: 790 – 802, 2010. ISSN 1937-4208. doi: 10.1109/TPWRD.2009.2033076.

[36] K. J. Rapp, G. A. Gauger, and J. Luksich. Behavior of ester dielectric fluids near the pour point. In *Conference on Electrical Insulation and Dielectric Phenomena*, pages 469–472, 1999. URL `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=807834`.

[37] P. K. Sen and Sarunpong Pansuwan. Overloading and loss-of-life assessment guidelines of oil-cooled transformers. *2001 Rural Electric Power Conference. Papers Presented at the 45th Annual Conference*, pages B4/1–B4/8, 2001. doi: 10.1109/REPCON.2001.949516.

[38] Alex Skillen, Alistair Revell, Hector Iacovides, and Wei Wub. Numerical prediction of local hot-spot phenomena in transformer windings. *Applied Thermal Engineering*, 36:96–105, 2012. ISSN 1359-4311. doi: https://doi.org/10.1016/j.applthermaleng.2011.11.054.

[39] Wojciech Sobieski and Anna Trykozko. Darcy's and forchheimer'slaws in practice. part 1. the experiment. *Technical Sciences*, 17:321–335, 2014. URL `http://uwm.edu.pl/wnt/technicalsc/tech_17_4/b02.pdf`.

[40] E.M. Sparrow, J.L. Novotny, and S.H. Lin. Laminar flow of a heat-generating fluid in a paralle-plate channel. *A.I.Ch.E. (Am. Inst. Chem. Engrs.) J.*, 9, 1963. doi: 10.1002/aic.690090618.

[41] D. Susa, M. Lehtonen, and H. Nordman. Dynamic thermal modelling of power transformers. *IEEE Transactions on Power Delivery*, 20:197–204, 2005. ISSN 1937-4208. doi: 10.1109/TPWRD.2004.835255.

[42] Omokhafe James Tola, James Garba Ambafi, Asizehi Yahaya Enesi, and Abiola Adefemi Shadrack. The application of computer programming in transformer design. *AU JOURNAL OF TECHNOLOGY*, 16:231–240, 05 2013.

[43] F. Torriano, M. Chaaban, and P. Picher. Numerical study of parameters affecting the temperature distribution in a disc-type transformer winding. *Applied Thermal Engineering*, 30(14):2034 – 2044, 2010. ISSN 1359-4311. doi: https://doi.org/10.1016/j.applthermaleng.2010.05.004. URL `http://www.sciencedirect.com/science/article/pii/S1359431110002000`.

[44] F. Torriano, P. Picher, and M. Chaaban. Numerical investigation of 3d flow and thermal effects in a disc-type transformer winding. *Applied Thermal Engineering, Elsevier*, 40:121–131, 2012. ISSN 1359-4311. doi: 10.1016/j.applthermaleng.2012.02.011.

[45] F. Torriano, H. Campelo, P. Labbé, and et al. Experimental and numerical thermofluid study of a disc-type transformer winding scale model. *2016 XXII International Conference on Electrical Machines (ICEM), Lausanne*, pages 2833–2839, 2016. ISSN er inproceedings. doi: 10.1109/ICELMACH.2016.7732924.

[46] H. K. Versteeg and W. Malalasekera. *An introduction to computational fluid dynamics - The finite volume method*. Pearson Education Limited, 1 edition, 1995. ISBN 0.582.21884.5.

[47] A. Weinlader, W. Wu, S. Tenbohlen, and Z. Wang. Prediction of the oil flow distribution in oil-immersed transformer windings by network modelling and computational fluid dynamics. *IET Electric Power Applications*, 6:82–90, 2011. doi: 10.1049/iet-epa.2011.0122.

[48] Frank M. White. *Viscous Fluid Flow*. Mc Graw Hill Education, 3 edition, 2006.

[49] J. Wijaya, Wenyu Guo, and T. Czaszejko et al. Thermal energy storagetemperature distribution in a disc-type transformer winding. *2012 7th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, x:838–843, 2012. ISSN 2158-2297. doi: 10.1109/ICIEA.2012.6360841.

Appendix

The following are the main files used to run the cold-start simulation in OpenFOAM v18.12.

Appendix A

Case files

The files included in this section are the case files for the cold-start simulation.

System

These files should be in the system-directory.

controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1812                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

//application       buoyantPimpleFoam;
//application       compressibleTransient;
//application       compressibleSteady;
application     porousTransient;

startFrom       latestTime;

//startTime       0;

stopAt          endTime;

endTime         1800;

deltaT          0.1;

writeControl    adjustableRunTime;//runTime;
```

```
writeInterval    10;

purgeWrite       0;

writeFormat      binary;

writePrecision   6;

writeCompression  off;

timeFormat       general;

timePrecision    6;

runTimeModifiable  true;

adjustTimeStep   yes;

//maxCo            5;
maxCo            0.9;
maxDeltaT        0.1;

functions
{
#includeEtc "caseDicts/postProcessing/probes/probes.cfg"
/*    probes1
    {
    type             probes;
    libs             ("libsampling.so");
    writeControl     adjustableRunTime;
    writeInterval    0.1;
    probeLocations   ((0.075  0.005  0.001));
    fields           (U);
    }
*/

    T_in
    {
        type             surfaces;
        libs             ("libsampling.so");
        writeControl     writeTime;
        surfaceFormat    raw;
        fields
        (
            T phi
        );
        interpolationScheme  cellPoint;

        surfaces
        (
            in
            {
                type          patch;
                patches       (inlet);
                triangulate   false;
            }
        );
    }
    T_out
    {
        type             surfaces;
        libs             ("libsampling.so");
        writeControl     writeTime;
        surfaceFormat    raw;
        fields
        (
            T phi
        );
        interpolationScheme cellPoint;

        surfaces
        (
            out
            {
                type        patch;
                patches     (outlet);
                triangulate false;
            }
        );
    }

    pressureDelta
{
    type            fieldValueDelta;
    libs            ("libfieldFunctionObjects.so");

    // Difference operation
    operation       subtract;

    // Volume averaged pressure
    region1
    {
        type            surfaceFieldValue;
        operation       areaAverage;
        fields          (p);
        writeFields     no;
        regionType      patch;
        name            inlet;
    }

    // Minimum pressure at 'outlet' patch
    region2
    {
        type            surfaceFieldValue;
        operation       areaAverage;
        fields          (p);
        writeFields     no;
        regionType      patch;
        name            outlet;
    }
}

    TOTdelta
{
    type            fieldValueDelta;
    libs            ("libfieldFunctionObjects.so");

    // Difference operation
    operation       subtract;

    // Volume averaged pressure
    region1
    {
        type            surfaceFieldValue;
        operation       weightedAverage;//weightedSum;
        fields          (T);
        writeFields     no;
        regionType      patch;
        name            inlet;

        //scaleFactor     1902;  //cp

         weightField     phi;
    }
    // Minimum pressure at 'outlet' patch
    region2
    {
        type            surfaceFieldValue;
        operation       weightedAverage;
        fields          (T);
        writeFields     no;
        regionType      patch;
        name            outlet;

        //scaleFactor      1902;

        weightField     phi;
    }
}

 //    #includeFunc singleGraphECell
    #includeFunc residuals
    #includeFunc temp
    minMax
    {
        type      fieldMinMax;
        libs      ("libfieldFunctionObjects.so");
        writeControl adjustableRunTime;
        writeInterval 0.5;
        fields        (T);
    }


}

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

blockMeshDict.m4

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    'format'      ascii;
    class       dictionary;
    object      blockMeshDict;
}
dnl> -----------------------------------------------------------------
dnl> <STANDARD DEFINTIONS>
dnl>
changecom(//)changequote([,]) dnl>
define(calc, [esyscmd(perl -e 'print ($1)')]) dnl>
define(VCOUNT, 0)  dnl>
define(vlabel, [[// ]pt VCOUNT ($1) define($1, VCOUNT)define([VCOUNT], incr(VCOUNT))])
dnl>
dnl> </STANDARD DEFINTIONS>
dnl> -----------------------------------------------------------------
dnl> Coordinates used in definition of verticies
dnl> ----------------------------------------------------------------- x-values
define(x0,0) dnl> origin position of x
define(xin,6.4) dnl> position at inlet (x)
define(xout, 57.2) dnl> position at outlet (x)
define(xmax, 66.1) dnl> maximum of x, length of the entire sustem
dnl> ----------------------------------------------------------------- y-values
define(y0, 0) dnl> origin position of y
define(ymax,10) dnl> maximum value of y, width of the entire system
dnl> ----------------------------------------------------------------- z-values
define(z0,0) dnl> origin position of z
define(channelHeight, 4.1)
define(channelGap, 15)
dnl define(firstChannel, 9)
define(firstChannel, 0)
define(firstChannelB, 0)
define(zpass,367) dnl> Heigth of a single pass

dnl>edited
dnl> -----------------------------------------------------------------
define(zp, calc(367*2)) dnl> Where the detailed pass start
define(zp1, 8.9)
define(zp2, calc(2*8.9))
define(xp1, calc(xmax+2*8.9))
define(xp2, calc(xp1+8.9))
define(calc_round, [esyscmd(perl -e "printf ('%d', $1)")]) dnl> same as floor()
dnl> -----------------------------------------------------------------
dnl> -----------------------------------------------------------------
dnl> Number of cells in Porous region
dnl> -----------------------------------------------------------------
dnl>define(nxPoroLeft, 25)
define(nxPoroCenterAdj, 55)
define(nxPoroCenter, 50)
dnl>define(nxPoroRight, 25)
define(nyPoro, 1)
define(nzPoro, 367)
define(nxPoroE,calc_round(xmax-xout))
define(nxPoroW,calc_round(xin))
dnl> -----------------------------------------------------------------
dnl>edited
dnl> ——————————————————————————————————————
dnl> Number of cells
dnl> ——————————————————————————————————————
define(nxchannel, 30) dnl> Number of cells in x−direction in channel (50.8 mm)
define(nychannel, 1) dnl> Number of cells in y−direction in channel (10 mm)
define(nzchannel, 32) dnl> Number of cells in z−direction in channel (4.1 mm)
define(nzchannelTB, 32) dnl> Number of cells in z−direction in channel (4.1 mm)
define(nxfineW, 11) dnl> Number of cells in x−direction in the high resolutuion legs (6.4 mm)
define(nxfineE, 13)dnl> Number of cells in x−direction in the high resolutuion legs (8.9 mm)
define(nyfine, 1) dnl> Number of cells in y−direction in the high resolutuion legs (10 mm)
define(nzfine, 1324) dnl> Number of cells in z−direction in the high resolutuion legs (367 mm)
dnl> ——————————————————————————————————————
dnl> Definition of blocks
dnl> ——————————————————————————————————————
define(createChannelBlock, hex ($1a $1b $1c $1d $1e $1f $1g $1h)
  channel$1 (nxchannel nychannel nzchannel) simpleGrading (
  ((0.5 15 12.84 )(0.5 15 calc(1/12.84))) 1 1)) dnl> se senere om dette er 1.2 i grading
define(createChannelBlockTB, hex ($1a $1b $1c $1d $1e $1f $1g $1h)
  channel$1 (nxchannel nychannel nzchannelTB) simpleGrading (
  ((0.5 15 12.84)(0.5 15 calc(1/12.84))) 1 1))


define(createSideBlockWest, hex ($1a $1b $1c $1d $1e $1f $1g $1h)
  West (nxfineW nyfine nzfine) simpleGrading (((2.112 3 1.44)(4.288 8 0.279)) 1
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1))
  )) dnl> Do not use for central block

define(createSideBlockEast, hex ($1a $1b $1c $1d $1e $1f $1g $1h)
  East (nxfineE nyfine nzfine) simpleGrading (((5.963 9 4.3)(2.937 4 0.578)) 1
  (
    (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)
  (4.1 32 1)(7.5 18 5.6)(7.5 18 0.1786)(4.1 32 1))

  )) dnl> Do not use for central block


dnl> edited
dnl> ——————————————————————————————————————
dnl> Definition of porous blocks
dnl> ——————————————————————————————————————
define(createCoarseBlockWestAdj, hex ($1a $1b $1c $1d $1e $1f $1g $1h)
coarseL$1 (nxfineW nyPoro nzPoro) simpleGrading (((2.112 3 1.44)(4.288 8 0.279)) 1
((0.9 300 1)(0.1 67 calc(1./10.))))) dnl> Used to define porous block on the right side
define(createPorousBlockCenterAdj, hex ($1b $1i $1l $1c $1f $1m $1p $1g)
porosity (nxPoroCenterAdj nyPoro nzPoro) simpleGrading (((0.1 10 3)(0.9 45 1)) 1
((0.9 300 1)(0.1 67 calc(1./10.))))) dnl> Used to define porous block in the middle
define(createCoarseBlockEastAdj, hex ($1i $1j $1k $1l $1m $1n $1o $1p)
coarseR$1 (nxPoroE nyPoro nzPoro) simpleGrading (1 1 ((0.9 300 1)(0.1 67 calc(1./10.)))))
dnl> Used to define porous block on the right side

define(createCoarseBlockWest, hex ($1a $1b $1c $1d $1e $1f $1g $1h)
coarseL$1 (nxPoroW nyPoro nzPoro) simpleGrading (1 1 1))
dnl> Used to define porous block on the right side
define(createPorousBlockCenter, hex ($1b $1i $1l $1c $1f $1m $1p $1g)
porosity (nxPoroCenter nyPoro nzPoro) simpleGrading (1 1 1))
dnl> Used to define porous block in the middle
define(createCoarseBlockEast, hex ($1i $1j $1k $1l $1m $1n $1o $1p)
coarseR$1 (nxPoroE nyPoro nzPoro) simpleGrading (1 1 1))
dnl> Used to define porous block on the right side
define(createPipeBlock, hex ($1a $1b $1c $1d $1e $1f $1g $1h))
dnl> edited


dnl> ——————————————————————————————————————————————————————————————
dnl> Definition of patches
dnl> ——————————————————————————————————————————————————————————————
define(upPatch, ($1e $1f $1g $1h)) dnl> up patch, not for central block or ribchannels
define(downPatch, ($1a $1d $1c $1b)) dnl> down patch, not for central block or ribchannels
define(leftPatch, ($1e $1h $1d $1a)) dnl> left patch, not for central block or ribchannels
define(rightPatch, ($1g $1f $1b $1c)) dnl> right patch, not for central block or ribchannels
define(frontPatch, ($1e $1a $1b $1f)) dnl> front patch, not for central block or ribchannels
define(backPatch, ($1g $1c $1d $1h)) dnl> back patch, not for central block or ribchannels

define(upPatchPoro, ($1f $1m $1p $1g)) dnl> up patch, not for central block or ribchannels
define(downPatchPoro, ($1b $1i $1l $1c)) dnl> down patch, not for central block or ribchannels
define(leftPatchPoro, ($1b $1f $1g $1c)) dnl> left patch, not for central block or ribchannels
define(rightPatchPoro, ($1m $1i $1l $1p)) dnl> right patch, not for central block or ribchannels
define(frontPatchPoro, ($1b $1i $1m $1f)) dnl> front patch, not for central block or ribchannels
define(backPatchPoro, ($1l $1c $1g $1p)) dnl> back patch, not for central block or ribchannels

define(upPatchCR, ($1m $1n $1o $1p)) dnl> up patch, not for central block or ribchannels
define(downPatchCR, ($1i $1l $1k $1j)) dnl> down patch, not for central block or ribchannels
define(leftPatchCR, ($1i $1m $1p $1l)) dnl> left patch, not for central block or ribchannels
define(rightPatchCR, ($1k $1o $1n $1j)) dnl> right patch, not for central block or ribchannels
define(frontPatchCR, ($1j $1n $1m $1i)) dnl> front patch, not for central block or ribchannels
define(backPatchCR, ($1p $1o $1k $1l)) dnl> back patch, not for central block or ribchannels

dnl> ——————————————————————————————————————————————————————————————
define(channelInternalPatch, ($1h $1d $1a $1e)
      ($1f $1b $1c $1g)) dnl> Defines all the internal patches for the channels
define(channelExternalFrontBack, ($1e $1a $1b $1f)
      ($1g $1c $1d $1h))
define(channelExternalTopBottom, ($1h $1e $1f $1g)
      ($1a $1d $1c $1b)) dnl> Defines all the external patches of the channels

dnl> ——————————————————————————————————————————————————————————————

convertToMeters 0.001;

vertices
(

  //West
  ( x0 y0 calc(z0+zp) ) vlabel(wa)
  ( xin y0 calc(z0+zp) ) vlabel(wb)
  ( xin ymax calc(z0+zp) ) vlabel(wc)
  ( x0 ymax calc(z0+zp) ) vlabel(wd)
  ( x0 y0 calc(zpass+zp) ) vlabel(we)
  ( xin y0 calc(zpass+zp) ) vlabel(wf)
  ( xin ymax calc(zpass+zp) ) vlabel(wg)
  ( x0 ymax calc(zpass+zp) ) vlabel(wh)

  //East
  ( xout y0 calc(z0+zp) ) vlabel(ea)
  ( xmax y0 calc(z0+zp) ) vlabel(eb)
  ( xmax ymax calc(z0+zp) ) vlabel(ec)
  ( xout ymax calc(z0+zp) ) vlabel(ed)
  ( xout y0 calc(zpass+zp) ) vlabel(ee)
  ( xmax y0 calc(zpass+zp) ) vlabel(ef)
  ( xmax ymax calc(zpass+zp) ) vlabel(eg)
  ( xout ymax calc(zpass+zp) ) vlabel(eh)

  // A−channel
  ( xin y0 calc(firstChannelB+zp) ) vlabel(Aa)
  ( xout y0 calc(firstChannelB+zp) ) vlabel(Ab)
  ( xout ymax calc(firstChannelB+zp) ) vlabel(Ac)
  ( xin ymax calc(firstChannelB+zp) ) vlabel(Ad)
  ( xin y0 calc(zp+firstChannelB+channelHeight) ) vlabel(Ae)
  ( xout y0 calc(zp+firstChannelB+channelHeight) ) vlabel(Af)
  ( xout ymax calc(zp+firstChannelB+channelHeight) ) vlabel(Ag)
  ( xin ymax calc(zp+firstChannelB+channelHeight) ) vlabel(Ah)

  // B−channel
  ( xin y0 calc(zp+firstChannel + 1*(channelHeight+channelGap)) ) vlabel(Ba)
  ( xout y0 calc(zp+firstChannel + 1*(channelHeight+channelGap)) ) vlabel(Bb)
  ( xout ymax calc(zp+firstChannel + 1*(channelHeight+channelGap)) ) vlabel(Bc)
  ( xin ymax calc(zp+firstChannel + 1*(channelHeight+channelGap)) ) vlabel(Bd)
  ( xin y0 calc(zp+firstChannel+2*channelHeight+1*channelGap) ) vlabel(Be)
  ( xout y0 calc(zp+firstChannel+2*channelHeight+1*channelGap) ) vlabel(Bf)
  ( xout ymax calc(zp+firstChannel+2*channelHeight+1*channelGap) ) vlabel(Bg)
  ( xin ymax calc(zp+firstChannel+2*channelHeight+1*channelGap) ) vlabel(Bh)

  // C−channel
  ( xin y0 calc(zp+firstChannel + 2*(channelHeight+channelGap)) ) vlabel(Ca)
  ( xout y0 calc(zp+firstChannel + 2*(channelHeight+channelGap)) ) vlabel(Cb)
  ( xout ymax calc(zp+firstChannel + 2*(channelHeight+channelGap)) ) vlabel(Cc)
  ( xin ymax calc(zp+firstChannel + 2*(channelHeight+channelGap)) ) vlabel(Cd)
  ( xin y0 calc(zp+firstChannel+3*channelHeight+2*channelGap) ) vlabel(Ce)
  ( xout y0 calc(zp+firstChannel+3*channelHeight+2*channelGap) ) vlabel(Cf)
  ( xout ymax calc(zp+firstChannel+3*channelHeight+2*channelGap) ) vlabel(Cg)
  ( xin ymax calc(zp+firstChannel+3*channelHeight+2*channelGap) ) vlabel(Ch)

  // D−channel
  ( xin y0 calc(zp+firstChannel + 3*(channelHeight+channelGap)) ) vlabel(Da)
  ( xout y0 calc(zp+firstChannel + 3*(channelHeight+channelGap)) ) vlabel(Db)
  ( xout ymax calc(zp+firstChannel + 3*(channelHeight+channelGap)) ) vlabel(Dc)
  ( xin ymax calc(zp+firstChannel + 3*(channelHeight+channelGap)) ) vlabel(Dd)
  ( xin y0 calc(zp+firstChannel+4*channelHeight+3*channelGap) ) vlabel(De)
  ( xout y0 calc(zp+firstChannel+4*channelHeight+3*channelGap) ) vlabel(Df)
  ( xout ymax calc(zp+firstChannel+4*channelHeight+3*channelGap) ) vlabel(Dg)
  ( xin ymax calc(zp+firstChannel+4*channelHeight+3*channelGap) ) vlabel(Dh)

  // E−channel
  ( xin y0 calc(zp+firstChannel + 4*(channelHeight+channelGap)) ) vlabel(Ea)
  ( xout y0 calc(zp+firstChannel + 4*(channelHeight+channelGap)) ) vlabel(Eb)
  ( xout ymax calc(zp+firstChannel + 4*(channelHeight+channelGap)) ) vlabel(Ec)
  ( xin ymax calc(zp+firstChannel + 4*(channelHeight+channelGap)) ) vlabel(Ed)
  ( xin y0 calc(zp+firstChannel+5*channelHeight+4*channelGap) ) vlabel(Ee)
  ( xout y0 calc(zp+firstChannel+5*channelHeight+4*channelGap) ) vlabel(Ef)
  ( xout ymax calc(zp+firstChannel+5*channelHeight+4*channelGap) ) vlabel(Eg)
  ( xin ymax calc(zp+firstChannel+5*channelHeight+4*channelGap) ) vlabel(Eh)

  // F−channel
  ( xin y0 calc(zp+firstChannel + 5*(channelHeight+channelGap)) ) vlabel(Fa)
  ( xout y0 calc(zp+firstChannel + 5*(channelHeight+channelGap)) ) vlabel(Fb)
  ( xout ymax calc(zp+firstChannel + 5*(channelHeight+channelGap)) ) vlabel(Fc)
  ( xin ymax calc(zp+firstChannel + 5*(channelHeight+channelGap)) ) vlabel(Fd)
  ( xin y0 calc(zp+firstChannel+6*channelHeight+5*channelGap) ) vlabel(Fe)
  ( xout y0 calc(zp+firstChannel+6*channelHeight+5*channelGap) ) vlabel(Ff)
  ( xout ymax calc(zp+firstChannel+6*channelHeight+5*channelGap) ) vlabel(Fg)
  ( xin ymax calc(zp+firstChannel+6*channelHeight+5*channelGap) ) vlabel(Fh)

  // G−channel
  ( xin y0 calc(zp+firstChannel + 6*(channelHeight+channelGap)) ) vlabel(Ga)
  ( xout y0 calc(zp+firstChannel + 6*(channelHeight+channelGap)) ) vlabel(Gb)
  ( xout ymax calc(zp+firstChannel + 6*(channelHeight+channelGap)) ) vlabel(Gc)
  ( xin ymax calc(zp+firstChannel + 6*(channelHeight+channelGap)) ) vlabel(Gd)
  ( xin y0 calc(zp+firstChannel+7*channelHeight+6*channelGap) ) vlabel(Ge)
  ( xout y0 calc(zp+firstChannel+7*channelHeight+6*channelGap) ) vlabel(Gf)
  ( xout ymax calc(zp+firstChannel+7*channelHeight+6*channelGap) ) vlabel(Gg)
  ( xin ymax calc(zp+firstChannel+7*channelHeight+6*channelGap) ) vlabel(Gh)

  // H-channel
  ( xin y0 calc(zp+firstChannel + 7*(channelHeight+channelGap)) ) vlabel(Ha)
  ( xout y0 calc(zp+firstChannel + 7*(channelHeight+channelGap)) ) vlabel(Hb)
  ( xout ymax calc(zp+firstChannel + 7*(channelHeight+channelGap)) ) vlabel(Hc)
  ( xin ymax calc(zp+firstChannel + 7*(channelHeight+channelGap)) ) vlabel(Hd)
  ( xin y0 calc(zp+firstChannel+8*channelHeight+7*channelGap) ) vlabel(He)
  ( xout y0 calc(zp+firstChannel+8*channelHeight+7*channelGap) ) vlabel(Hf)
  ( xout ymax calc(zp+firstChannel+8*channelHeight+7*channelGap) ) vlabel(Hg)
  ( xin ymax calc(zp+firstChannel+8*channelHeight+7*channelGap) ) vlabel(Hh)

  // I-channel
  ( xin y0 calc(zp+firstChannel + 8*(channelHeight+channelGap)) ) vlabel(Ia)
  ( xout y0 calc(zp+firstChannel + 8*(channelHeight+channelGap)) ) vlabel(Ib)
  ( xout ymax calc(zp+firstChannel + 8*(channelHeight+channelGap)) ) vlabel(Ic)
  ( xin ymax calc(zp+firstChannel + 8*(channelHeight+channelGap)) ) vlabel(Id)
  ( xin y0 calc(zp+firstChannel+9*channelHeight+8*channelGap) ) vlabel(Ie)
  ( xout y0 calc(zp+firstChannel+9*channelHeight+8*channelGap) ) vlabel(If)
  ( xout ymax calc(zp+firstChannel+9*channelHeight+8*channelGap) ) vlabel(Ig)
  ( xin ymax calc(zp+firstChannel+9*channelHeight+8*channelGap) ) vlabel(Ih)

  // J-channel
  ( xin y0 calc(zp+firstChannel + 9*(channelHeight+channelGap)) ) vlabel(Ja)
  ( xout y0 calc(zp+firstChannel + 9*(channelHeight+channelGap)) ) vlabel(Jb)
  ( xout ymax calc(zp+firstChannel + 9*(channelHeight+channelGap)) ) vlabel(Jc)
  ( xin ymax calc(zp+firstChannel + 9*(channelHeight+channelGap)) ) vlabel(Jd)
  ( xin y0 calc(zp+firstChannel+10*channelHeight+9*channelGap) ) vlabel(Je)
  ( xout y0 calc(zp+firstChannel+10*channelHeight+9*channelGap) ) vlabel(Jf)
  ( xout ymax calc(zp+firstChannel+10*channelHeight+9*channelGap) ) vlabel(Jg)
  ( xin ymax calc(zp+firstChannel+10*channelHeight+9*channelGap) ) vlabel(Jh)

  // K-channel
  ( xin y0 calc(zp+firstChannel + 10*(channelHeight+channelGap)) ) vlabel(Ka)
  ( xout y0 calc(zp+firstChannel + 10*(channelHeight+channelGap)) ) vlabel(Kb)
  ( xout ymax calc(zp+firstChannel + 10*(channelHeight+channelGap)) ) vlabel(Kc)
  ( xin ymax calc(zp+firstChannel + 10*(channelHeight+channelGap)) ) vlabel(Kd)
  ( xin y0 calc(zp+firstChannel+11*channelHeight+10*channelGap) ) vlabel(Ke)
  ( xout y0 calc(zp+firstChannel+11*channelHeight+10*channelGap) ) vlabel(Kf)
  ( xout ymax calc(zp+firstChannel+11*channelHeight+10*channelGap) ) vlabel(Kg)
  ( xin ymax calc(zp+firstChannel+11*channelHeight+10*channelGap) ) vlabel(Kh)

  // L-channel
  ( xin y0 calc(zp+firstChannel + 11*(channelHeight+channelGap)) ) vlabel(La)
  ( xout y0 calc(zp+firstChannel + 11*(channelHeight+channelGap)) ) vlabel(Lb)
  ( xout ymax calc(zp+firstChannel + 11*(channelHeight+channelGap)) ) vlabel(Lc)
  ( xin ymax calc(zp+firstChannel + 11*(channelHeight+channelGap)) ) vlabel(Ld)
  ( xin y0 calc(zp+firstChannel+12*channelHeight+11*channelGap) ) vlabel(Le)
  ( xout y0 calc(zp+firstChannel+12*channelHeight+11*channelGap) ) vlabel(Lf)
  ( xout ymax calc(zp+firstChannel+12*channelHeight+11*channelGap) ) vlabel(Lg)
  ( xin ymax calc(zp+firstChannel+12*channelHeight+11*channelGap) ) vlabel(Lh)

  // M-channel
  ( xin y0 calc(zp+firstChannel + 12*(channelHeight+channelGap)) ) vlabel(Ma)
  ( xout y0 calc(zp+firstChannel + 12*(channelHeight+channelGap)) ) vlabel(Mb)
  ( xout ymax calc(zp+firstChannel + 12*(channelHeight+channelGap)) ) vlabel(Mc)
  ( xin ymax calc(zp+firstChannel + 12*(channelHeight+channelGap)) ) vlabel(Md)
  ( xin y0 calc(zp+firstChannel+13*channelHeight+12*channelGap) ) vlabel(Me)
  ( xout y0 calc(zp+firstChannel+13*channelHeight+12*channelGap) ) vlabel(Mf)
  ( xout ymax calc(zp+firstChannel+13*channelHeight+12*channelGap) ) vlabel(Mg)
  ( xin ymax calc(zp+firstChannel+13*channelHeight+12*channelGap) ) vlabel(Mh)

  // N-channel
  ( xin y0 calc(zp+firstChannel + 13*(channelHeight+channelGap)) ) vlabel(Na)
  ( xout y0 calc(zp+firstChannel + 13*(channelHeight+channelGap)) ) vlabel(Nb)
```

```
  ( xout ymax calc(zp+firstChannel + 13*(channelHeight+channelGap)) ) vlabel(Nc)
  ( xin ymax calc(zp+firstChannel + 13*(channelHeight+channelGap)) ) vlabel(Nd)
  ( xin y0 calc(zp+firstChannel+14*channelHeight+13*channelGap) ) vlabel(Ne)
  ( xout y0 calc(zp+firstChannel+14*channelHeight+13*channelGap) ) vlabel(Nf)
  ( xout ymax calc(zp+firstChannel+14*channelHeight+13*channelGap) ) vlabel(Ng)
  ( xin ymax calc(zp+firstChannel+14*channelHeight+13*channelGap) ) vlabel(Nh)

  // O-channel
  ( xin y0 calc(zp+firstChannel + 14*(channelHeight+channelGap)) ) vlabel(Oa)
  ( xout y0 calc(zp+firstChannel + 14*(channelHeight+channelGap)) ) vlabel(Ob)
  ( xout ymax calc(zp+firstChannel + 14*(channelHeight+channelGap)) ) vlabel(Oc)
  ( xin ymax calc(zp+firstChannel + 14*(channelHeight+channelGap)) ) vlabel(Od)
  ( xin y0 calc(zp+firstChannel+15*channelHeight+14*channelGap) ) vlabel(Oe)
  ( xout y0 calc(zp+firstChannel+15*channelHeight+14*channelGap) ) vlabel(Of)
  ( xout ymax calc(zp+firstChannel+15*channelHeight+14*channelGap) ) vlabel(Og)
  ( xin ymax calc(zp+firstChannel+15*channelHeight+14*channelGap) ) vlabel(Oh)

  // P-channel
  ( xin y0 calc(zp+firstChannel + 15*(channelHeight+channelGap)) ) vlabel(Pa)
  ( xout y0 calc(zp+firstChannel + 15*(channelHeight+channelGap)) ) vlabel(Pb)
  ( xout ymax calc(zp+firstChannel + 15*(channelHeight+channelGap)) ) vlabel(Pc)
  ( xin ymax calc(zp+firstChannel + 15*(channelHeight+channelGap)) ) vlabel(Pd)
  ( xin y0 calc(zp+firstChannel+16*channelHeight+15*channelGap) ) vlabel(Pe)
  ( xout y0 calc(zp+firstChannel+16*channelHeight+15*channelGap) ) vlabel(Pf)
  ( xout ymax calc(zp+firstChannel+16*channelHeight+15*channelGap) ) vlabel(Pg)
  ( xin ymax calc(zp+firstChannel+16*channelHeight+15*channelGap) ) vlabel(Ph)

  // Q-channel
  ( xin y0 calc(zp+firstChannel + 16*(channelHeight+channelGap)) ) vlabel(Qa)
  ( xout y0 calc(zp+firstChannel + 16*(channelHeight+channelGap)) ) vlabel(Qb)
  ( xout ymax calc(zp+firstChannel + 16*(channelHeight+channelGap)) ) vlabel(Qc)
  ( xin ymax calc(zp+firstChannel + 16*(channelHeight+channelGap)) ) vlabel(Qd)
  ( xin y0 calc(zp+firstChannel+17*channelHeight+16*channelGap) ) vlabel(Qe)
  ( xout y0 calc(zp+firstChannel+17*channelHeight+16*channelGap) ) vlabel(Qf)
  ( xout ymax calc(zp+firstChannel+17*channelHeight+16*channelGap) ) vlabel(Qg)
  ( xin ymax calc(zp+firstChannel+17*channelHeight+16*channelGap) ) vlabel(Qh)

  // R-channel
  ( xin y0 calc(zp+firstChannel + 17*(channelHeight+channelGap)) ) vlabel(Ra)
  ( xout y0 calc(zp+firstChannel + 17*(channelHeight+channelGap)) ) vlabel(Rb)
  ( xout ymax calc(zp+firstChannel + 17*(channelHeight+channelGap)) ) vlabel(Rc)
  ( xin ymax calc(zp+firstChannel + 17*(channelHeight+channelGap)) ) vlabel(Rd)
  ( xin y0 calc(zp+firstChannel+18*channelHeight+17*channelGap) ) vlabel(Re)
  ( xout y0 calc(zp+firstChannel+18*channelHeight+17*channelGap) ) vlabel(Rf)
  ( xout ymax calc(zp+firstChannel+18*channelHeight+17*channelGap) ) vlabel(Rg)
  ( xin ymax calc(zp+firstChannel+18*channelHeight+17*channelGap) ) vlabel(Rh)

  // S-channel
  ( xin y0 calc(zp+firstChannel + 18*(channelHeight+channelGap)) ) vlabel(Sa)
  ( xout y0 calc(zp+firstChannel + 18*(channelHeight+channelGap)) ) vlabel(Sb)
  ( xout ymax calc(zp+firstChannel + 18*(channelHeight+channelGap)) ) vlabel(Sc)
  ( xin ymax calc(zp+firstChannel + 18*(channelHeight+channelGap)) ) vlabel(Sd)
  ( xin y0 calc(zp+firstChannel+19*channelHeight+18*channelGap) ) vlabel(Se)
  ( xout y0 calc(zp+firstChannel+19*channelHeight+18*channelGap) ) vlabel(Sf)
  ( xout ymax calc(zp+firstChannel+19*channelHeight+18*channelGap) ) vlabel(Sg)
  ( xin ymax calc(zp+firstChannel+19*channelHeight+18*channelGap) ) vlabel(Sh)

  // T-channel
  ( xin y0 calc(zp+firstChannel + 19*(channelHeight+channelGap)) ) vlabel(Ta)
  ( xout y0 calc(zp+firstChannel + 19*(channelHeight+channelGap)) ) vlabel(Tb)
  ( xout ymax calc(zp+firstChannel + 19*(channelHeight+channelGap)) ) vlabel(Tc)
  ( xin ymax calc(zp+firstChannel + 19*(channelHeight+channelGap)) ) vlabel(Td)
  ( xin y0 calc(zp+firstChannel+20*channelHeight+19*channelGap) ) vlabel(Te)
  ( xout y0 calc(zp+firstChannel+20*channelHeight+19*channelGap) ) vlabel(Tf)
  ( xout ymax calc(zp+firstChannel+20*channelHeight+19*channelGap) ) vlabel(Tg)
  ( xin ymax calc(zp+firstChannel+20*channelHeight+19*channelGap) ) vlabel(Th)

//CoarseWest1
```

```
  ( x0 y0 calc(0*zpass) ) vlabel(p1a)
  ( xin y0 calc(0*zpass) ) vlabel(p1b)
  ( xin ymax calc(0*zpass) ) vlabel(p1c)
  ( x0 ymax calc(0*zpass) ) vlabel(p1d)
  ( x0 y0 calc(1*zpass) ) vlabel(p1e)
  ( xin y0 calc(1*zpass) ) vlabel(p1f)
  ( xin ymax calc(1*zpass) ) vlabel(p1g)
  ( x0 ymax calc(1*zpass) ) vlabel(p1h)

  //CoarseEast1
  ( xout y0 calc(0*zpass) ) vlabel(p1i)
  ( xmax y0 calc(0*zpass) ) vlabel(p1j)
  ( xmax ymax calc(0*zpass) ) vlabel(p1k)
  ( xout ymax calc(0*zpass) ) vlabel(p1l)
  ( xout y0 calc(1*zpass) ) vlabel(p1m)
  ( xmax y0 calc(1*zpass) ) vlabel(p1n)
  ( xmax ymax calc(1*zpass) ) vlabel(p1o)
  ( xout ymax calc(1*zpass) ) vlabel(p1p)

 //CoarseWest2
  ( x0 y0 calc(1*zpass) ) vlabel(p2a)
  ( xin y0 calc(1*zpass) ) vlabel(p2b)
  ( xin ymax calc(1*zpass) ) vlabel(p2c)
  ( x0 ymax calc(1*zpass) ) vlabel(p2d)
  ( x0 y0 calc(2*zpass) ) vlabel(p2e)
  ( xin y0 calc(2*zpass) ) vlabel(p2f)
  ( xin ymax calc(2*zpass) ) vlabel(p2g)
  ( x0 ymax calc(2*zpass) ) vlabel(p2h)

  //CoarseEast2
  ( xout y0 calc(1*zpass) ) vlabel(p2i)
  ( xmax y0 calc(1*zpass) ) vlabel(p2j)
  ( xmax ymax calc(1*zpass) ) vlabel(p2k)
  ( xout ymax calc(1*zpass) ) vlabel(p2l)
  ( xout y0 calc(2*zpass) ) vlabel(p2m)
  ( xmax y0 calc(2*zpass) ) vlabel(p2n)
  ( xmax ymax calc(2*zpass) ) vlabel(p2o)
  ( xout ymax calc(2*zpass) ) vlabel(p2p)

 //CoarseWest0
  ( x0 y0 calc(-1*zpass) ) vlabel(p0a)
  ( xin y0 calc(-1*zpass) ) vlabel(p0b)
  ( xin ymax calc(-1*zpass) ) vlabel(p0c)
  ( x0 ymax calc(-1*zpass) ) vlabel(p0d)
  ( x0 y0 calc(0*zpass) ) vlabel(p0e)
  ( xin y0 calc(0*zpass) ) vlabel(p0f)
  ( xin ymax calc(0*zpass) ) vlabel(p0g)
  ( x0 ymax calc(0*zpass) ) vlabel(p0h)

  //CoarseEast0
  ( xout y0 calc(-1*zpass) ) vlabel(p0i)
  ( xmax y0 calc(-1*zpass) ) vlabel(p0j)
  ( xmax ymax calc(-1*zpass) ) vlabel(p0k)
  ( xout ymax calc(-1*zpass) ) vlabel(p0l)
  ( xout y0 calc(0*zpass) ) vlabel(p0m)
  ( xmax y0 calc(0*zpass) ) vlabel(p0n)
  ( xmax ymax calc(0*zpass) ) vlabel(p0o)
  ( xout ymax calc(0*zpass) ) vlabel(p0p)

//pipe1
  ( xout y0 calc(1*zpass+zp) ) vlabel(r1a)
  ( xmax y0 calc(1*zpass+zp) ) vlabel(r1b)
  ( xmax ymax calc(1*zpass+zp) ) vlabel(r1c)
  ( xout ymax calc(1*zpass+zp) ) vlabel(r1d)
  ( xout y0 calc(1*zpass+zp+zp2) ) vlabel(r1e)
  ( xmax y0 calc(1*zpass+zp+zp2) ) vlabel(r1f)
  ( xmax ymax calc(1*zpass+zp+zp2) ) vlabel(r1g)
  ( xout ymax calc(1*zpass+zp+zp2) ) vlabel(r1h)
```

```
//pipe2
  ( xmax y0 calc(1*zpass+zp+zp1) ) vlabel(r2a)
  ( xp1 y0 calc(1*zpass+zp+zp1) ) vlabel(r2b)
  ( xp1 ymax calc(1*zpass+zp+zp1) ) vlabel(r2c)
  ( xmax ymax calc(1*zpass+zp+zp1) ) vlabel(r2d)
  ( xmax y0 calc(1*zpass+zp+zp2) ) vlabel(r2e)
  ( xp1 y0 calc(1*zpass+zp+zp2) ) vlabel(r2f)
  ( xp1 ymax calc(1*zpass+zp+zp2) ) vlabel(r2g)
  ( xmax ymax calc(1*zpass+zp+zp2) ) vlabel(r2h)

//pipe3
  ( xp1 y0 calc(-zp2-zpass) ) vlabel(r3a)
  ( xp2 y0 calc(-zp2-zpass) ) vlabel(r3b)
  ( xp2 ymax calc(-zp2-zpass) ) vlabel(r3c)
  ( xp1 ymax calc(-zp2-zpass) ) vlabel(r3d)
  ( xp1 y0 calc(1*zpass+zp+zp2) ) vlabel(r3e)
  ( xp2 y0 calc(1*zpass+zp+zp2) ) vlabel(r3f)
  ( xp2 ymax calc(1*zpass+zp+zp2) ) vlabel(r3g)
  ( xp1 ymax calc(1*zpass+zp+zp2) ) vlabel(r3h)

//pipe4
  ( xout y0 calc(-zp2-zpass) ) vlabel(r4a)
  ( xp1 y0 calc(-zp2-zpass) ) vlabel(r4b)
  ( xp1 ymax calc(-zp2-zpass) ) vlabel(r4c)
  ( xout ymax calc(-zp2-zpass) ) vlabel(r4d)
  ( xout y0 calc(-zp1-zpass) ) vlabel(r4e)
  ( xp1 y0 calc(-zp1-zpass) ) vlabel(r4f)
  ( xp1 ymax calc(-zp1-zpass) ) vlabel(r4g)
  ( xout ymax calc(-zp1-zpass) ) vlabel(r4h)


//pipe5
  ( xout y0 calc(-zp1-zpass) ) vlabel(r5a)
  ( xmax y0 calc(-zp1-zpass) ) vlabel(r5b)
  ( xmax ymax calc(-zp1-zpass) ) vlabel(r5c)
  ( xout ymax calc(-zp1-zpass) ) vlabel(r5d)
  ( xout y0 -zpass ) vlabel(r5e)
  ( xmax y0 -zpass ) vlabel(r5f)
  ( xmax ymax -zpass ) vlabel(r5g)
  ( xout ymax -zpass ) vlabel(r5h)

);

blocks
(
  createSideBlockWest(w)

  createSideBlockEast(e)

  createChannelBlockTB(A)
  createChannelBlock(B)
  createChannelBlock(C)
  createChannelBlock(D)
  createChannelBlock(E)
  createChannelBlock(F)
  createChannelBlock(G)
  createChannelBlock(H)
  createChannelBlock(I)
  createChannelBlock(J)
   createChannelBlock(K)
  createChannelBlock(L)
  createChannelBlock(M)
  createChannelBlock(N)
  createChannelBlock(O)
  createChannelBlock(P)
  createChannelBlock(Q)
  createChannelBlock(R)
  createChannelBlock(S)
  createChannelBlockTB(T)

  createCoarseBlockWest(p1)
  createPorousBlockCenter(p1)
  createCoarseBlockEast(p1)

  createCoarseBlockWestAdj(p2)
  createPorousBlockCenterAdj(p2)
  createCoarseBlockEastAdj(p2)

  createCoarseBlockWest(p0)
  createPorousBlockCenter(p0)
  createCoarseBlockEast(p0)

  createPipeBlock(r1)
  pipe (nxfineE nyfine calc_round(zp2+10)) simpleGrading (((5.963 9 4.3)(2.937 4 0.578)) 1
  ((0.5 calc_round(zp2*0.5+10) 8)(0.5 calc_round(0.5*zp2) 1)))
  createPipeBlock(r2)
  pipe (calc_round(xp1-xmax+2) nyfine calc_round(zp2-zp1)) simpleGrading
  (((0.15 calc_round(2+0.25*(zp1-xmax)) 1.6)(0.85 calc_round(0.75*(zp1-xmax)) 1)) 1 1)
  createPipeBlock(r3)
  pipe (calc_round(xp2-xp1) nyfine calc_round(2*zp2+zpass*2+zp)) simpleGrading (1 1
  ((8.9 calc_round(8.9) 1)(calc(zpass*2+zp+zp1*2) calc_round(zpass*2+zp+zp1*2) 1)
  (8.9 calc_round(8.9) 1)))
  createPipeBlock(r4)
  pipe (calc_round(xp1-xout) nyfine calc_round(zp2-zp1)) simpleGrading
  (((8.9 calc_round(8.9) 1)(calc(xp1-xout-8.9) calc_round(xp1-xout-8.9) 1)) 1 1)
  dnl>(calc_round((2*(xp1-xout)*1.4)+30) nyfine calc_round(2*(zp2-zp1))) simpleGrading
  (((calc(0.25*xin/xp1) 3 1.44)(calc(0.75*xin/xp1) 17 0.2325)(calc(0.4*xin/xp1) 17 4)
  (calc(1-1.4*xin/xp1) calc_round(2*(xp1-xin*1.4)) 1)) 1 1)
  createPipeBlock(r5)
  pipeInletBlock (nxPoroE nyfine calc_round(zp1)) simpleGrading (1 1 1)


);

edges
(
);

//patches
boundary
(
/*
    wall legsFrontBack
    (
      frontPatch(w)
      frontPatch(e)
      backPatch(w)
      backPatch(e)
    )

    empty porousFrontBack
    (
    frontPatch(pc1)
    backPatch(pc1)

    frontPatch(pc2)
    backPatch(pc2)

    )


    empty coarseFrontBack
    (
    frontPatch(ce1)
    backPatch(ce1)
    frontPatch(cw1)
    backPatch(cw1)

    frontPatch(ce2)
    backPatch(ce2)
    frontPatch(cw2)
    backPatch(cw2)

    )

    empty pipeFrontBack
    (
    frontPatch(r1)
    frontPatch(r2)
    frontPatch(r3)
    frontPatch(r4)
    frontPatch(r5)
    backPatch(r1)
    backPatch(r2)
    backPatch(r3)
    backPatch(r4)
    backPatch(r5)
    )


    wall coilFrontBack
    (
      channelExternalFrontBack(A)
      channelExternalFrontBack(B)
      channelExternalFrontBack(C)
      channelExternalFrontBack(D)
      channelExternalFrontBack(E)
      channelExternalFrontBack(F)
      channelExternalFrontBack(G)
      channelExternalFrontBack(H)
      channelExternalFrontBack(I)
      channelExternalFrontBack(J)
      channelExternalFrontBack(K)
    )
*/

    pipeWall
    {
    type wall;
    faces
    (
      upPatch(r1)
      leftPatch(r1)
//      rightPatch(r1)
      upPatch(r2)
      downPatch(r2)
      upPatch(r3)
      downPatch(r3)
//      leftPatch(r3)
      rightPatch(r3)
      leftPatch(r4)
      downPatch(r4)
//      upPatch(r4)
      leftPatch(r5)
      rightPatch(r5)

    );
    }


    legsTopBottom
    {
```

```
    type  wall;
    faces
    (
      upPatch(w)
      downPatch(e)
      downPatch(A)
      upPatch(T)
      //upPatch(cw1)
      //downPatch(ce1)
        upPatch(p1)
        //downPatch(p1)
        upPatchPoro(p1)
        downPatchPoro(p1)
 //       upPatchCR(p1)
        downPatchCR(p1)
        downPatch(p2)
        upPatchCR(p2)
        downPatchPoro(p2)
        upPatchPoro(p2)
        upPatchCR(p0)
        downPatch(p0)
        upPatchPoro(p0)
        downPatchPoro(p0)
//      upPatch(ce2)
//      downPatch(cw2)

    );
    }

    legsLeftRight
    {
    type  wall;
    faces
    (
      leftPatch(w)
      rightPatch(e)
    );
    }

    coilTopBottom
    {
    type  wall;
    faces
    (
      //channelExternalTopBottom(A)

      upPatch(A)
      channelExternalTopBottom(B)
      channelExternalTopBottom(C)
      channelExternalTopBottom(D)
      channelExternalTopBottom(E)
      channelExternalTopBottom(F)
      channelExternalTopBottom(G)
      channelExternalTopBottom(H)
      channelExternalTopBottom(I)
      channelExternalTopBottom(J)
      channelExternalTopBottom(K)
      channelExternalTopBottom(L)
      channelExternalTopBottom(M)
      channelExternalTopBottom(N)
      channelExternalTopBottom(O)
      channelExternalTopBottom(P)
      channelExternalTopBottom(Q)
      channelExternalTopBottom(R)
      channelExternalTopBottom(S)
      downPatch(T)
    );
    }
    inlet
    {
    type cyclic;
    neighbourPatch outletPipe;
    faces
    (
      downPatchCR(p0)
      //downPatch(w)
      //leftPatch(w)
    );
    }

    outlet
    {
    type cyclic;
    neighbourPatch inletPipe;
    faces
    (
      upPatch(e)
      //rightPatch(e)
    );
    }

    master
    {
    type wall;
    faces
    (
      leftPatch(e)
      rightPatch(w)
    );
    }

    coarseLeftRight
    {
    type wall;
    faces
    (
      leftPatch(p1)
      rightPatchCR(p1)
      leftPatch(p2)
      rightPatchCR(p2)
      leftPatch(p0)
      rightPatchCR(p0)

//        leftPatch(cw2)
//        rightPatch(ce2)

    );
    }

    pipeSlave3 //outlets
    {
    type patch;
    faces
    (
      rightPatch(r4)
      rightPatch(r2)
    );
    }

    pipeMaster3 //inlets
    {
      type patch;
      faces
    (
      leftPatch(r3)
```

```
);
}

pipeSlave1 // outlets
{
type patch;
faces
(

  leftPatch(r2)
);
}

pipeMaster1 // inlets
{
type patch;
faces
(
  rightPatch(r1)
);
}

pipeSlave5 // outlets
{
type patch;
faces
(
  upPatch(r4)
);
}

pipeMaster5 // inlets
{
type patch;
faces
(
  downPatch(r5)
);
}

inletPipe
{
type cyclic;
neighbourPatch outlet;
faces
(
   downPatch(r1)
);
}

outletPipe
{
type cyclic;
neighbourPatch inlet;
faces
(
    upPatch(r5)
);
}

internalInlet
{
type patch;
faces
(
  //downPatch(ce2)
  downPatch(w)
  downPatchCR(p2)
```

```
        //downPatch(p1)
      //downPatchCR(p0)
    );
    }

    internalOutlet
    {
    type patch;
    faces
    (
      upPatchCR(p1)
      upPatch(p2)
     //  upPatch(p0)
      //upPatch(cw2)

    );
    }

    porousInlet1  //missvisende navn
    {
    type patch;
    faces
    (
    downPatch(p1)
    );
    }

    porousOutlet1
    {
    type patch;
    faces
    (
    upPatch(p0)
    );
    }

    slave  //Used to merge with the outer blocks, no boundary condition necessary
    {
    type wall;
    faces
    (
      channelInternalPatch(A)
      channelInternalPatch(B)
      channelInternalPatch(C)
      channelInternalPatch(D)
      channelInternalPatch(E)
      channelInternalPatch(F)
      channelInternalPatch(G)
      channelInternalPatch(H)
      channelInternalPatch(I)
      channelInternalPatch(J)
      channelInternalPatch(K)
      channelInternalPatch(L)
      channelInternalPatch(M)
      channelInternalPatch(N)
      channelInternalPatch(O)
      channelInternalPatch(P)
      channelInternalPatch(Q)
      channelInternalPatch(R)
      channelInternalPatch(S)
      channelInternalPatch(T)

    );
    }

);

mergePatchPairs
```

```
(
  (master slave) //merging between the channels and the outer blocks
  //(coarseSlave porousMaster)
  (internalInlet internalOutlet)
  (pipeMaster1 pipeSlave1)
  (pipeSlave3 pipeMaster3)
  (pipeMaster5 pipeSlave5)
  //(pipeOutlet inlet)
  //(outlet pipeInlet)
  (porousInlet1 porousOutlet1)
);

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

fvSchemes

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1812                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{

   default          CrankNicolson 1;//0.9;//Euler;
//    default          Euler;// -> works
}

gradSchemes
{
    default         Gauss linear;
//    grad(U)         cellMDLimited Gauss linear 1.0;
//        default         Gauss linear vanLeer 1;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss vanLeerV;//Gauss linearUpwind grad(U);
//    div(phi,K)      Gauss vanLeer;//linear; //what is K?
    div(phi,h)      Gauss vanLeer; //use a better method

    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss vanLeer phi 1;//Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear orthogonal;//corrected;
//    default Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         orthogonal;//corrected;
//    default         corrected;
}


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

fvSolution

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1812                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
   "rho.*"
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-6;
        relTol          0.05;
    }

    p_rgh
    {
        //solver          GAMG;
        //smoother          DIC;
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-6;//1e-6;
        relTol          0.001;
        //maxIter          1000;
    }

    p_rghFinal
    {
        $p_rgh;
        relTol        0;//1e-6;//1e-6
    }

    "(U|T|h|k|epsilon)"
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance       1e-6;//1e-7
        relTol          0.01;
      //   maxIter          1000;
    }

    "(U|T|h|k|epsilon)Final"
    {
        $U;
        relTol          0;
    //     maxIter          1000;
    }

}

PIMPLE
{
    momentumPredictor  no;//no;
    nOuterCorrectors     1;//50;
    nNonOrthogonalCorrectors  0;
    nCorrectors          2;//2          //4;
    //pRefCell               10000;
    pRefPoint            (0.06165  0.005  0.5505);
    pRefValue            1e5;
/*
    //outerCorrectorResidualControl
        residualControl
    {
        "(U|p|p_rgh|rho|h)"
        {
            tolerance 1e-3;
            relTol        0;
        }
    }
*/
}
/*
relaxationFactors
{
    fields
    {
        rho                 1.0;
        p_rgh               0.7;
    }
    equations
    {
        U                   0.3;
        h                   0.2;
        "(k|epsilon|omega)"  0.7;
    }
}
*/

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

topoSetDict

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  7
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      topoSetDict;
}

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

actions
(
    {
        name radiator;
        type cellSet;
        action new;

        source boxToCell;
        sourceInfo
        {
            box         (0.07 0 1.102) (0.08 0.01 1.2);
        }
    }
);


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

setFieldsDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  plus                                  |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue perm 0
);

regions
(
    zoneToCell
    {
        zone    "porosity" ;
        fieldValues
        (
            volScalarFieldValue perm 1
        );
    }
);


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

Constant

These files should be in the constant-directory.

thermophysicalProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  7
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      thermophysicalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

thermoType
{

    type            heRhoThermo;
    mixture         pureMixture;
    transport       custom;
    //transport       const;
    //thermo            hConst;
    //thermo            hCustom;
    thermo          hPolynomial;
    //equationOfState rhoConst;
    equationOfState Boussinesq;
    specie          specie;
    energy          sensibleEnthalpy;
/*
   type             heRhoThermo;
   mixture          pureMixture;
   transport        polynomial;
   //transport        custom;
   thermo           hPolynomial;
   equationOfState  icoPolynomial;
   specie           specie;
   energy           sensibleEnthalpy;
*/
}

mixture
{

    specie
    {
        molWeight   17.0;
    }
    equationOfState
    {
        T0          293.15;
        beta        7.5e-4;
        rho0        968;
//          rho           1000;
    }
    thermodynamics
    {
        Hf          0;
        //kappa         1.19;
        //Cp            1902;
        Sf           0;
        CpCoeffs<8> (1249.29  2.17  0 0 0 0 0 0);
    }
    transport
    {
        mu             7.23e-2;//1e-5;
        Pr             935.6;   //0.150;
    }

/*
    specie
    {
        molWeight  17.0;
    }
    equationOfState
    {
    //      rhoCoeffs<8>     (968  0 0 0 0 0 0 0);
     //     rhoCoeffs<8>    (1184.46  -0.726 0 0 0 0 0 0); // rearranged from Boussinesq.
    rhoCoeffs<8>    (1098.72  -0.712 0 0 0 0 0 0);
    }
    thermodynamics
    {
        Hf            0;
        Sf            0;
       // CpCoeffs<8> (0  0  0.021133333  0 0 0 0 0);
 //     CpCoeffs<8>      (1902 0 0 0 0 0 0 0);
       // CpCoeffs<8>       (1249.29  2.17  0 0 0 0 0 0);
        CpCoeffs<8>      (807.163  3.58  0 0 0 0 0 0);


    }
    transport
    {
        //mu        7.23e-2;
        //Pr        935.6; // These values should have nothing to say! Test that!
//          muCoeffs<8>      ( 7.23e-2 0 0 0 0 0 0 0);
//          kappaCoeffs<8>   ( 0.147  0 0 0 0 0 0 0);
//          muCoeffs<8>      ( 7.23e-2 1e-3 0 0 0 0 0 0);
//          kappaCoeffs<8>   ( 0.147  0.1  0 0 0 0 0 0);
        muCoeffs<8>      ( 0.08467 -4e-4 5e-7 0 0 0 0 0);
        kappaCoeffs<8>   ( 0.1509  -7.101e-5 0 0 0 0 0 0);

    }
*/
}


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

turbulenceProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1812                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType laminar;
/*
RAS
{
    RASModel        kEpsilon;

    turbulence      on;

    printCoeffs     on;
}
*/

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

fvOptions

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      fvOptions;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //


options
{
    energySource
    {
        type scalarSemiImplicitSource;
        selectionMode cellZone;
        //cellZone centralWest;
        cellZone porosity;
        volumeMode specific;

        injectionRateSuSp
        {
            h   (1.4024e6 0); //for specific case;
        }
    }

    porositySource
    {

        type explicitPorositySource;

        explicitPorositySourceCoeffs
        {
            type                DarcyForchheimer;
            selectionMode       cellZone;
            cellZone            porosity;
            //volumeMode            specific;

            d       d [0 -2 0 0 0 0 0] (3.2e6 -1e2 -1e2);
            f       f [0 -1 0 0 0 0 0] (0 0 0);

            coordinateSystem
            {
                //type      cartesian;
                origin  (0 0 0);
                e1      (1 0 0);
                e2      (0 1 0);

                /*coordinateRotation
                {
                    type        axesRotation;
                    e1          (1 0 0);
                    e2          (0 1 0);
                }
                */
            }
        }
    }
    fixedInletTemp
    {
        type                fixedTemperatureConstraint;

        //selectionMode     cellSet;
        //cellSet           pipeSet;
        selectionMode       cellSet;
        cellSet             radiator;//pipeInletBlock;

        //mode              lookup;
        //T                 maxT;
        mode                uniform;
        temperature         293.15;
    }

}

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

g

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1812                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -2 0 0 0 0];
value           (0 0 -9.81);  // (0 -9.81 0);
//value           (0 0 0);

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

0

These files should be in the 0-directory.

p_rgh

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 1e5;

boundaryField
{
    "(coarseFrontBack|coilFrontBack|porousFrontBack|legsFrontBack)"
    {
        type empty;
    }

    "(legsLeftRight|coarseLeftRight|legsTopBottom|porousTopBottom|coilTopBottom|master|porousMaster|co
    {
//          type            zeroGradient;
        type            fixedFluxPressure;
        //gradient      uniform 0;
        value           $internalField;

    }


    "(inlet|inletPipe|outlet|outletPipe)"
    {
        type            cyclic;
    }
/*
    outlet
    {
        //type          fixedFluxPressure;
        //type          fixedValue;
        //value         $internalField;


       // --- for g neq 0 ---
        type            fixedMean;
        meanValue           $internalField;
        value           uniform 0;

    }
*/
}

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

p

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1812                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 1e5;

boundaryField
{
    "(coarseLeftRight|porousTopBottom|legsTopBottom|legsLeftRight|porousLeftRight|coilTopBottom|master
    {
        type            calculated;
        value           $internalField;
    }

    "(inlet|inletPipe|outlet|outletPipe)"
    {
        type cyclic;
    }


    "(legsFrontBack|porousFrontBack|coilFrontBack|coarseFrontBack)"
    {
        type empty;
    }
}

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

U

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  plus                                  |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    "(coarseFrontBack|legsFrontBack|coilFrontBack|porousFrontBack)"
    {
        //type            slip;
        type            empty;
    }
    legsTopBottom
    {
        type            noSlip;
    }

    "(pipe.*)"
    {
        type    noSlip;
    }

    /*
    legsBottom
    {
        type            noSlip;
    }
    */
    "(legsLeftRight|coarseLeftRight|porousMaster|coarseSlave|internalInlet|internalOutlet|slave)"
    {
        type            noSlip;
    }

    porousTopBottom
    {
        type            noSlip;
    }

    coilTopBottom
    {
        type            noSlip;
    }


    master
    {
        //type            slip;
        //value           $internalField;
        type            noSlip;
    }
    "(inlet|inletPipe|outlet|outletPipe)"
    {
        type      cyclic;
    }

/*
    inlet
    {
        type              flowRateInletVelocity;
        //type              fixedValue;
        volumetricFlowRate   constant 7.4e-6;
        //value             uniform (1e-2 0 0);


    }
    outlet
    {
        type              zeroGradient;
        //type              inletOutlet;
        //inletValue        uniform (0 0 0);
        //inletValue        uniform 0;

        //type              flowRateOutletVelocity;
        //type              zeroGradient;
        //volumetricFlowRate   constant 1e-05;
        //type              inletOutlet;
        //value             uniform (0 0 0);
        //value             uniform 0;
        //inletValue        uniform (0 0 0);

    }
*/
}

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

T

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

dimensions      [0 0 0 1 0 0 0];

internalField   uniform 253.15;//300;

boundaryField
{
    "(porousMaster|coarseSlave|internalInlet|internalOutlet|slave|pipe.*)"
    {
        type zeroGradient;
    }

    "(coarseFrontBack|coilFrontBack|legsFrontBack|porousFrontBack)"
    {
        type            empty;
    }
    legsTopBottom
    {
        type            zeroGradient;
    }

    "(coarseLeftRight|legsLeftRight)"
    {
        type            zeroGradient;
    }

    "(master|coilTopBottom)"
    {
        type externalWallHeatFluxTemperature;
        mode flux;
        q        2336.4;
        kappaMethod     fluidThermo;
        value       $internalField;


    }
    "(inlet|inletPipe|outlet|outletPipe)"
    {
        type            cyclic;
    }
/*    outlet
    {
        type            zeroGradient;
    }
*/
    porousTopBottom
    {
        type            zeroGradient;
    }
}
```

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

Appendix B

Transport property source code

The following is the advanced fluid transport property implemented directly in the source code.

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2017 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/>.

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

// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

template<class Thermo>
inline Foam::customTransport<Thermo>::customTransport
(
    const Thermo& t,
    const scalar mu,
    const scalar Pr
)
:
    Thermo(t),
    mu_(mu),
    rPr_(1.0/Pr)
{}
```

```
template<class Thermo>
inline Foam::customTransport<Thermo>::customTransport
(
    const word& name,
    const customTransport& ct
)
:
    Thermo(name, ct),
    mu_(ct.mu_),
    rPr_(ct.rPr_)
{}


template<class Thermo>
inline Foam::autoPtr<Foam::customTransport<Thermo>>
Foam::customTransport<Thermo>::clone() const
{
    return autoPtr<customTransport<Thermo>>::New(*this);
}


template<class Thermo>
inline Foam::autoPtr<Foam::customTransport<Thermo>>
Foam::customTransport<Thermo>::New
(
    const dictionary& dict
)
{
    return autoPtr<customTransport<Thermo>>::New(dict);
}


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

template<class Thermo>
inline Foam::scalar Foam::customTransport<Thermo>::mu
(
    const scalar p,
    const scalar T
) const
{
    //return mu_;
    return 968.*(1.-7.5e-4*(T-293.15))*exp(20.81369191*log(T)*log(T) - 252.81869067*log(T) + 755.03026
}


template<class Thermo>
inline Foam::scalar Foam::customTransport<Thermo>::kappa
(
    const scalar p,
    const scalar T
) const
{
    //return this->Cp(p, T)*mu(p, T)*rPr_;
    return -7.2e-7*T*T + 3.71e-4*T + 9.75e-2;
}


template<class Thermo>
inline Foam::scalar Foam::customTransport<Thermo>::alphah
(
    const scalar p,
    const scalar T
) const
{
    //return mu(p, T)*rPr_;
    return kappa(p,T)/this->Cp(p,T);
}
```

```
// * * * * * * * * * * * * * * * Member Operators  * * * * * * * * * * * * * //

template<class Thermo>
inline void Foam::customTransport<Thermo>::operator=
(
    const customTransport<Thermo>& ct
)
{
    Thermo::operator=(ct);

    mu_ = ct.mu_;
    rPr_ = ct.rPr_;
}


template<class Thermo>
inline void Foam::customTransport<Thermo>::operator+=
(
    const customTransport<Thermo>& st
)
{
    scalar Y1 = this->Y();

    Thermo::operator+=(st);

    if (mag(this->Y()) > SMALL)
    {
        Y1 /= this->Y();
        scalar Y2 = st.Y()/this->Y();

        mu_ = Y1*mu_ + Y2*st.mu_;
        rPr_ = 1.0/(Y1/rPr_ + Y2/st.rPr_);
    }
}


template<class Thermo>
inline void Foam::customTransport<Thermo>::operator*=
(
    const scalar s
)
{
    Thermo::operator*=(s);
}


// * * * * * * * * * * * * * * * Friend Operators  * * * * * * * * * * * * * //

template<class Thermo>
inline Foam::customTransport<Thermo> Foam::operator+
(
    const customTransport<Thermo>& ct1,
    const customTransport<Thermo>& ct2
)
{
    Thermo t
    (
        static_cast<const Thermo&>(ct1) + static_cast<const Thermo&>(ct2)
    );

    if (mag(t.Y()) < SMALL)
    {
        return customTransport<Thermo>
        (
            t,
            0,
            ct1.rPr_
        );
    }
    else
    {
        scalar Y1 = ct1.Y()/t.Y();
        scalar Y2 = ct2.Y()/t.Y();

        return customTransport<Thermo>
        (
            t,
            Y1*ct1.mu_ + Y2*ct2.mu_,
            1.0/(Y1/ct1.rPr_ + Y2/ct2.rPr_)
        );
    }
}


template<class Thermo>
inline Foam::customTransport<Thermo> Foam::operator*
(
    const scalar s,
    const customTransport<Thermo>& ct
)
{
    return customTransport<Thermo>
    (
        s*static_cast<const Thermo&>(ct),
        ct.mu_,
        1.0/ct.rPr_
    );
}


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

Appendix C

porousTransient solver

The following are the main files that are implemented in the custom solver `porousTransient`.

porousTransient.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2017 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
    buoyantPimpleFoam

Group
    grpHeatTransferSolvers

Description
    Transient solver for buoyant, turbulent flow of compressible fluids for
    ventilation and heat-transfer.

    Turbulence is modelled using a run-time selectable compressible RAS or
    LES model.
```

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

#include "fvCFD.H"
#include "rhoThermo.H"
#include "turbulentFluidThermoModel.H"
#include "radiationModel.H"
#include "fvOptions.H"
#include "pimpleControl.H"

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

int main(int argc, char *argv[])
{
    argList::addNote
    (
        "Transient solver for buoyant, turbulent fluid flow"
        " of compressible fluids, including radiation."
    );

    #include "postProcess.H"

    #include "addCheckCaseOptions.H"
    #include "setRootCaseLists.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createControl.H"
    #include "createFields.H"
    #include "createFieldRefs.H"
    #include "initContinuityErrs.H"
    #include "createTimeControls.H"
    #include "compressibleCourantNo.H"
    #include "setInitialDeltaT.H"

    turbulence->validate();

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

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

        rho = thermo.rho();
        Cp = thermo.Cp();
        mu = thermo.mu();
        kappa = thermo.kappa();
/*
   //tensorField& alphaTensorin_ = alphaTensor.ref();
    forAll(alphaTensor, i)
        {
            alphaTensor[i].xx() = kappa[i]*poro.value()/Cp[i];
            alphaTensor[i].yy() = kappa[i]*poro.value()/Cp[i];
            alphaTensor[i].zz() = kappa[i]/(Cp[i]*poro.value());
        //if(i<10)
        //{
      Info<< "alphaTensor_value_zz: "<< alphaTensor[i].zz()
<< "alphaTensor_value_xx: "<< alphaTensor[i].xx() << nl << endl;
        //}
      }
     // Info<< "alphaTensor_value_zz: "<< alphaTensor[i].xx()  << nl << endl;
*/
    while (runTime.run())
    {
        #include "readTimeControls.H"
        #include "compressibleCourantNo.H"
        #include "setDeltaT.H"

        ++runTime;

           forAll(alphaTensor, i)
           {
                alphaTensor[i].xx() = kappa[i]*poro.value()/Cp[i];
                alphaTensor[i].yy() = kappa[i]*poro.value()/Cp[i];
                alphaTensor[i].zz() = kappa[i]/(Cp[i]*poro.value());
              /*
                int a = 1;
                if(a==1)
                {
                  a = 0;
                  Info<< "alphaTensor_value_zz: "<< alphaTensor[i].zz()
<< "alphaTensor_value_xx: "<< alphaTensor[i].xx() << nl << endl;
                }
                */
        }


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

        #include "rhoEqn.H"

        // --- Pressure-velocity PIMPLE corrector loop
        while (pimple.loop())
        {
            #include "UEqn.H"
            #include "EEqn.H"

            // --- Pressure corrector loop
            while (pimple.correct())
            {
                #include "pEqn.H"
            }

            if (pimple.turbCorr())
            {
                turbulence->correct();
            }
        }

        rho = thermo.rho();
        Cp = thermo.Cp();
        mu = thermo.mu();
        kappa = thermo.kappa();

        runTime.write();

        runTime.printExecutionTime(Info);
    }

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

    return 0;
}


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

createFields.H

```
Info<< "Reading thermophysical properties\n" << endl;

autoPtr<rhoThermo> pThermo(rhoThermo::New(mesh));
rhoThermo& thermo = pThermo();
thermo.validate(args.executable(), "h", "e");

volScalarField rho
(
    IOobject
    (
        "rho",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    thermo.rho()
);


volScalarField Cp
    (
    IOobject
        (
            "Cp",
            runTime.timeName(),
            mesh,
            IOobject::READ_IF_PRESENT,
            IOobject::AUTO_WRITE
        ),
         thermo.Cp()
     );

volScalarField kappa
    (
    IOobject
        (
            "kappa",
            runTime.timeName(),
            mesh,
            IOobject::READ_IF_PRESENT,
            IOobject::AUTO_WRITE
        ),
         thermo.kappa()
     );


volScalarField mu
   (
   IOobject
       (
           "mu",
           runTime.timeName(),
          mesh,
          IOobject::READ_IF_PRESENT,
          IOobject::AUTO_WRITE
      ),
       thermo.mu()
   );

volScalarField& p = thermo.p();

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

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



#include "compressibleCreatePhi.H"


//#include "readTransportProperties.H"

dimensionedScalar poro
(
    "poro",
    dimless,
    runTime.controlDict().lookupOrDefault<scalar>("poro", 0.223)
);


volTensorField alphaTensor
(
     IOobject
     (
        "alphaTensor",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    mesh,
    dimensionedTensor("alphaTensor", dimensionSet(1,-1,-1,0,0,0,0),Zero) //(poro*thermo.kappa()/thermo
);


  /*
  volTensorField a
  (
  IOobject
  (
  "fileName",
  runTime.timeName(),
  mesh,
  IOobject::NO_READ,
  IOobject::AUTO_WRITE
  ),
  mesh,
  dimentionedTensor("name", dimensionSet(0,0,0,0,0,0,0), //correct dimensions here
  tensor::zero)
  );
*/


Info<< "Creating turbulence model\n" << endl;
autoPtr<compressible::turbulenceModel> turbulence
(
    compressible::turbulenceModel::New
    (
        rho,
        U,
        phi,
        thermo
    )
);


#include "readGravitationalAcceleration.H"
#include "readhRef.H"
#include "gh.H"


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

// Force p_rgh to be consistent with p
p_rgh = p - rho*gh;

mesh.setFluxRequired(p_rgh.name());

label pRefCell = 0;
scalar pRefValue = 0.0;

if (p_rgh.needReference())
{
    setRefCell
    (
        p,
        p_rgh,
        pimple.dict(),
        pRefCell,
        pRefValue
    );

    p += dimensionedScalar
    (
        "p",
        p.dimensions(),
        pRefValue - getRefCellValue(p, pRefCell)
    );
}


dimensionedScalar initialMass("initialMass", fvc::domainIntegrate(rho));

#include "createDpdt.H"
```

```
#include "createK.H"

#include "createMRF.H"
#include "createRadiationModel.H"
#include "createFvOptions.H"
```

createFieldRefs.H

```
const volScalarField& psi = thermo.psi();
```

readTransportProperties.H

```
singlePhaseTransportModel laminarTransport(U, phi);
/*
// Thermal expansion coefficient [1/K]
dimensionedScalar beta
(
    "beta",
    dimless/dimTemperature,
    laminarTransport
);

// Reference temperature [K]
dimensionedScalar TRef("TRef", dimTemperature, laminarTransport);

// Laminar Prandtl number
dimensionedScalar Pr("Pr", dimless, laminarTransport);

// Turbulent Prandtl number
dimensionedScalar Prt("Prt", dimless, laminarTransport);
*/

// Porosity [-]
dimensionedScalar poro("poro", dimless, laminarTransport);

dimensionedTensor alphaTensor("alphaTensor", dimensionSet(0,2,-1,0,0,0,0), laminarTransport);
```

EEqn.H

```
{
    volScalarField& he = thermo.he();

    fvScalarMatrix EEqn
    (
        fvm::ddt(rho, he) + (scalar(1) - perm + perm/poro)*fvm::div(phi, he)
     // + fvc::ddt(rho, K) + fvc::div(phi, K)
      /* + (
            he.name() == "e"
          ? fvc::div
            (
                fvc::absolute(phi/fvc::interpolate(rho), U),
                p,
                "div(phiv,p)"
            )
          : -dpdt*(scalar(1) - perm)
        ) */
      - (scalar(1) - perm)*fvm::laplacian(turbulence->alphaEff(), he)
     ==
       // rho*(U&g)*(scalar(1) - perm)
      //+ radiation->Sh(thermo, he)
        perm*fvm::laplacian(alphaTensor, he)
      + perm*fvOptions(rho, he)
    );

    EEqn.relax();

    fvOptions.constrain(EEqn);

    EEqn.solve();

    fvOptions.correct(he);

    thermo.correct();
    radiation->correct();
}
```

UEqn.H

```
// Solve the Momentum equation

MRF.correctBoundaryVelocity(U);

fvVectorMatrix UEqn
(
    fvm::ddt(rho, U) + fvm::div(phi, U)
  + MRF.DDt(rho, U)
  + turbulence->divDevRhoReff(U)
 ==
    fvOptions(rho, U)
);

UEqn.relax();

fvOptions.constrain(UEqn);

if (pimple.momentumPredictor())
{
    solve
    (
        UEqn
     ==
        fvc::reconstruct
        (
            (
              - ghf*fvc::snGrad(rho)
              - fvc::snGrad(p_rgh)
            )*mesh.magSf()
        )
    );

    fvOptions.correct(U);
    K = 0.5*magSqr(U);
}
```

pEqn.H

```
dimensionedScalar compressibility = fvc::domainIntegrate(psi);
bool compressible = (compressibility.value() > SMALL);

rho = thermo.rho();

// Thermodynamic density needs to be updated by psi*d(p) after the
// pressure solution
const volScalarField psip0(psi*p);

volScalarField rAU(1.0/UEqn.A());
surfaceScalarField rhorAUf("rhorAUf", fvc::interpolate(rho*rAU));
volVectorField HbyA(constrainHbyA(rAU*UEqn.H(), U, p_rgh));

surfaceScalarField phig(-rhorAUf*ghf*fvc::snGrad(rho)*mesh.magSf());

surfaceScalarField phiHbyA
(
    "phiHbyA",
    (
        fvc::flux(rho*HbyA)
      + MRF.zeroFilter(rhorAUf*fvc::ddtCorr(rho, U, phi))
    )
  + phig
);

MRF.makeRelative(fvc::interpolate(rho), phiHbyA);

// Update the pressure BCs to ensure flux consistency
constrainPressure(p_rgh, rho, U, phiHbyA, rhorAUf, MRF);

fvScalarMatrix p_rghDDtEqn
(
    fvc::ddt(rho) + psi*correction(fvm::ddt(p_rgh))
  + fvc::div(phiHbyA)
  ==
    fvOptions(psi, p_rgh, rho.name())
);

while (pimple.correctNonOrthogonal())
{
    fvScalarMatrix p_rghEqn
    (
        p_rghDDtEqn
      - fvm::laplacian(rhorAUf, p_rgh)
    );

    p_rghEqn.setReference
    (
        pRefCell,
        compressible ? getRefCellValue(p_rgh, pRefCell) : pRefValue
    );

    p_rghEqn.solve(mesh.solver(p_rgh.select(pimple.finalInnerIter())));

    if (pimple.finalNonOrthogonalIter())
    {
        // Calculate the conservative fluxes
        phi = phiHbyA + p_rghEqn.flux();

        // Explicitly relax pressure for momentum corrector
        p_rgh.relax();

        // Correct the momentum source with the pressure gradient flux
        // calculated from the relaxed pressure
        U = HbyA + rAU*fvc::reconstruct((phig + p_rghEqn.flux())/rhorAUf);
        U.correctBoundaryConditions();
        fvOptions.correct(U);
        K = 0.5*magSqr(U);
    }
}

p = p_rgh + rho*gh;

#include "rhoEqn.H"
#include "compressibleContinuityErrs.H"

if (p_rgh.needReference())
{
    if (!compressible)
    {
        p += dimensionedScalar
        (
            "p",
            p.dimensions(),
            pRefValue - getRefCellValue(p, pRefCell)
        );
    }
    else
    {
        p += (initialMass - fvc::domainIntegrate(psi*p))
            /compressibility;
        thermo.correctRho(psi*p - psip0);
        rho = thermo.rho();
        p_rgh = p - rho*gh;
    }
}
else
{
    thermo.correctRho(psi*p - psip0);
}

rho = thermo.rho();

if (thermo.dpdt())
{
    dpdt = fvc::ddt(p);
}
```

Agsagan Ragunathan

Numerical Investigation of Natural Convectionof Oil Flow in Transformer

NTNU
Norwegian University of
Science and Technology

SINTEF