



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

A comparison between single and multi-region continuous adjoint optimization for couple thermal fluid problems

TESI DI LAUREA MAGISTRALE IN
AERONAUTICAL ENGINEERING - INGEGNERIA AERONAUTICA

Author: **Fabio Bianchi**

Student ID: 942526

Advisor: Prof. Federico Piscaglia

Co-advisors: Dott. Emanuele Gallorini

Academic Year: 2021-2022

Abstract

The aim of this thesis is the comparison between two solvers, whose purpose is the optimization of the geometry of heat exchangers: the first solver can perform single region optimizations, whereas the second one performs multi-region optimizations. These solvers employ the continuous adjoint method and optimize the geometry with respect to the total pressure losses and the temperature extracted minimizing a properly defined objective function. The solvers are implemented in OpenFOAM-dev, the single region solver, called *thermalAdjointShapeOptimizationFoam* is an expanded version of the already present *adjointShapeOptimizationFoam*, whereas the multi-region solver *adjointMultiregionFoam* is the expansion of the single region solver for multiple regions problems, following the same structure of the already present *chtMultiRegionFoam*. In this document, after the presentation of the mathematical formalism that is behind the continuous adjoint optimization applied to a problem comprising fluid and solid regions, the structure of the multiple region solver itself is presented. After that, the geometry chosen for the comparison is presented, this latter is a heat exchanger devoted to cooling down the electric motor's inverter of the Formula SAE car of team Dynamis. The comparison of the results given by the two solvers highlights the best convergence speed of the single region solver, whereas the second solver produces geometry changes in the same zone as the first solver, but in general different, with a higher computational cost; given the differences in the results, an experimental verification would be beneficial to assess which geometry is the best.

Keywords: OpenFOAM, continuous adjoint method, topology optimization, multi-region simulation, formula SAE, conjugate heat transfer

Abstract in lingua italiana

L'obiettivo di questa tesi è comparare due solver il cui scopo è l'ottimizzazione della forma degli scambiatori di calore: il primo solver preso in esame ha la capacità di gestire solo problemi fluidi a singola regione, mentre il secondo riesce a risolvere casi con multiple regioni, siano esse solide o fluide. Questi solver impiegano il metodo dell'aggiunto continuo e ottimizzano la geometria sia da un punto di vista delle perdite di pressione totali che da un punto di vista della temperatura estratta minimizzando un'appositamente definita funzione obiettivo. I solver sono implementati in OpenFOAM-dev; il solver monoregione, chiamato *thermalAdjointShapeOptimizationFoam* è implementato sulla base del già presente *adjointShapeOptimizationFoam*, mentre il solver multiregione *adjointMultiregionFoam* è una espansione del solver monoregione, al quale viene aggiunta la possibilità di risolvere geometrie a più regioni seguendo la stessa struttura del già presente *chtMultiRegionFoam*. In questo documento, dopo la presentazione del formalismo matematico alla base del metodo dell'aggiunto continuo, viene presentata la struttura del codice di *adjointMultiRegionFoam* stesso. Dopodiché viene mostrata la geometria scelta per effettuare la comparazione, la quale rappresenta uno scambiatore di calore il cui scopo è quello di raffreddare l'inverter dei motori elettrici della vettura di Formula SAE realizzata dal team Dynamis. La comparazione dei risultati evidenzia la più rapida velocità di convergenza del solver monoregione, mentre il secondo solver genera un cambiamento di geometria nella stessa sfera d'influenza del primo solver ma in generale diversi, con un più alto costo computazionale; date le differenze nei risultati, una verifica sperimentale sarebbe necessaria per valutare quale tra le due geometrie sia la migliore.

Parole chiave: OpenFOAM, metodo dell'aggiunto continuo, ottimizzazione topologica, simulazione multiregione, formula SAE, scambio termico coniugato

Contents

Abstract	i
Abstract in lingua italiana	iii
Contents	v
1 Introduction	1
2 The continuous adjoint method applied to a multi-region problem	5
2.1 Equations of the primal problem	6
2.2 Adjoint problem equations	8
2.3 Adjoint problem boundary conditions	19
2.3.1 Inlet patch boundary conditions	19
2.3.2 Fixed heat walls boundary conditions	20
2.3.3 Fixed temperature walls boundary conditions	21
2.3.4 Outlet patch boundary conditions	22
3 Description of the solver	27
4 Results comparison	33
4.1 Description and meshing of the geometry for the baseline adjoint simulation	33
4.2 baseline adjoint case set up	37
4.3 Description and meshing of the geometry for the multiregion adjoint simulation	42
4.4 Simulations results	45
4.4.1 Non optimized flow	45
4.4.2 Single region optimization	46
4.4.3 Multiregion optimization	50
5 Conclusion	55

Bibliography	57
A Appendix A	59
B Appendix B	67
List of Figures	75
List of Tables	77
List of Symbols	79
Acknowledgements	81

1 | Introduction

Heat transfer is nowadays one of the most important and challenging engineering topics; this is partially due to the fact that electrification is becoming more and more pressing due to climate change, and with batteries and electric motors comes the need for proper cooling of batteries packs and inverters. But the electrification of the traditional combustion engines is not the only reason for the rising importance of heat transfer, also other fields are facing a rising in the request for efficient heat exchangers, starting with the ones employed by the rocket engines used in the space industry, passing through the ones employed to extract heat generated by nuclear reactions in nuclear power plants, and ending with the cooling requested by high-performance computers.

In this framework, it is clear that better designed heat exchangers can drastically improve the performance and, most importantly, the efficiency of the products of the industries cited above. The design of a heat exchanger is generally a trade-off choice between the pressure drop generated and the thermal power recovered or, in other words, the temperature of the refrigerating fluid at the outlet, and this is due to the fact that, in general, a radiator that creates more pressure losses through fins, restrictions and higher surface area is capable of extract more thermal power from the refrigerated fluid or solid. However, it is not possible to generate an arbitrary (huge) pressure drop, since it is prescribed by the pump adopted in the system, which must have limited power and dimensions dictated by the plant characteristics, the cost and the overall dimensions. Moreover, it can be noted that in general, a surface with more complex features performs better in terms of heat exchange; if the manufacturing of these types of surfaces once with the traditional manufacturing techniques was a problem, now with the introduction of new technologies such as 3D printing it can be easily overcome, at the price of slower production rates and more expensive finished pieces.

In this thesis is presented an already developed OpenFOAM solver, *thermalAdjointShapeOptimizationFoam*, which is capable of optimizing the geometry of a given fluid domain using the topology optimization technique, a theory firstly introduced by Bendsoe and Kikuchi [5] and applied for a long time in the field of structural shape optimization; this theory,

from the early 2000 years have been expanded also to fluid dynamics of ducted flows [10] [3] [8], due to the exponential increment in computational power. This solver is based on the already built-in *adjointShapeOptimizationFoam* whose (almost) complete description can be found in [9], which performs an optimization of geometry with the objective of minimizing the total pressure losses; whereas *thermalAdjointShapeOptimizationFoam* also includes the possibility to maximize the recovered heat power or, more interesting, a weighted combination of the two chosen by the user. Both solvers employ the same optimization method to perform the optimization, where an objective function is defined to describe the pressure losses and another objective function is defined to describe the thermal power recovered (obviously this latter is just for the second cited solver), then a defined by the user weighted sum of the two is performed to define the overall objective function, and this will be the function to be optimized (minimized), subject to a set of constraints, namely the equations describing the physics of the problem, i.e. the Navier Stokes equations combined with an equation which describes the temperature evolution. To apply the changes in geometry to a computational domain related to a finite volume method, a value of porosity is associated with each cell, so that when the porosity is at its maximum the cells will be treated as solid; the purpose of the solver is to find the optimum porosity distribution through the domain. The topology optimization theory relies on the continuous adjoint method, a mathematical theory that applies a set of Lagrange multipliers to the constrained equations of the problem and uses them to compute the sensitivity, i.e. the derivative of the objective function with respect to the design variable. Once this is computed, the objective function can be optimized by using an optimization algorithm, for example, the steepest gradient or the method of moving asymptotes. To the already developed code *thermalAdjointShapeOptimizationFoam* was added the functionality to simulate multi-region problems, i.e. problems with a geometry containing multiple fluids and solid regions inside of itself. It is interesting to note that rather than a new solver this is a combination of the solver cited above and the already built-in solver *chtMultiRegionFoam* [2], which is capable of handle multi-region problems with heat exchange, but not to optimize their geometry. To solve this type of problem, the equations describing the solid region have to be taken into account to derive an expression for the adjoint variables also in the solid domain; a detailed discussion of the mathematics adopted is the subject of chapter 2. The name of the combined solver that can perform the multi-region optimization is *adjointMultiRegionFoam*, and a discussion of its structure is the subject of chapter 3. The purpose of this thesis is to perform a comparison between *thermalAdjointShapeOptimizationFoam* and *adjointMultiRegionFoam*, to highlight the main performance differences and possibilities opportunities; to do this the geometry chosen was the one of a plate used to refrigerate the inverter of the Formula SAE vehicle

made by Dynamis Team; starting from the step file it underwent to a process of geometry cleaning to prepare an STL file for the meshing process. Given the nature of the solver, two different geometries were prepared, one describing the fluid region and one describing the solid region. After that, the STL files were used to generate the computational domain adopting the OpenFOAM utilities `blockMesh` and `snappyHexMesh`; once this step was done, the simulations were run and the results compared, as will be shown in chapter 4.

2 | The continuous adjoint method applied to a multi-region problem

In the following chapter the mathematical formalism describing the continuous adjoint method applied to a heated duct is presented. The method, based on [7] takes into account the presence of a solid phase, so the derivation of the equation for the adjoint temperature in the solid region is also presented. The continuous adjoint approach consists in minimizing an objective function J with respect to a variable η , applying the method of moving asymptotes for which the computation of the sensitivity fields associated with the objective function are needed. The variable η is proportional to the porosity associated with the computational cell: when η is equal to 1 the cell is considered fluid and so the porosity is null and the cell doesn't give any penalty in the momentum equation; on the other side, when η tends to 0, the value of the porosity of the cell tends to its maximum, and it follows that the cell gives a penalty to the fluid's momentum. Since just the values of the thermophysical properties related to the fluid and the solid considered in the simulation are provided to the solver, an interpolation for these latter, discussed in chapter 3 will be needed to associate them to intermediate values of η .

Since the objective of the problem is to find an optimal compromise solution between heat transfer and pressure losses in a multi-region heated duct, the overall objective function is composed of two sub-functions J_p and J_t , adopting a bi-objective optimization strategy where each function is weighted by a quantity chosen by the user (ω_1, ω_2):

$$J = \omega_1 J_p + \omega_2 J_t \quad (2.1)$$

Where the function J_p represents the difference between the outlet and the inlet of the total pressure flux:

$$J_p = \int_{inlet} \left(p + \frac{v^2}{2} \right) \mathbf{v} \cdot \hat{\mathbf{n}} dS - \int_{inlet} \left(p + \frac{v^2}{2} \right) \mathbf{v} \cdot \hat{\mathbf{n}} dS \quad (2.2)$$

and the function J_T represents the thermal power difference between the inlet and the outlet:

$$J_T = \int_{outlet} (\rho c_p T) \mathbf{v} \cdot \hat{\mathbf{n}} dS - \int_{inlet} (\rho c_p T) \mathbf{v} \cdot \hat{\mathbf{n}} dS \quad (2.3)$$

Note that both functions are positive; if in the first case this perfectly matches with the minimization of J by the solver, it is not the same case for J_T : since the solver has to maximize the heat exchange, it is needed to multiply the thermal power function by a negative number, i.e. by a negative ω_2 .

In section 2.1 the equations of the primal problem will be presented: they consist of the continuity and momentum equation, integrated with an equation for the transport of the temperature in the fluid and an equation for the transport of the temperature in the solid. In section 2.2 the equations of the adjoint problem will be derived by applying Lagrangian multipliers to the equations describing the primal problem. Finally, in section 2.3 the boundary conditions for a multi-region problem with a fixed temperature or a fixed thermal flux will be obtained.

2.1. Equations of the primal problem

The equations represented in this section will describe a steady, incompressible turbulent flow with the addition of a diffusion equation for the temperature in the solid. The flow is represented by the continuity and the momentum equation, this latter including a penalty term given by the cells with porosity greater than zero; furthermore, note that since the flow is incompressible, in the momentum equation the pressure term is given by the pressure divided by the density of the fluid, as commonly used in incompressible problems. The energy equation, given the nature of the problem, is a transport equation that considers the convection of the heat in the fluid and the thermal diffusivity of the heat in the porous cells; finally, the equation describing the temperature field in the solid region is a Laplace equation. Every equation is written in the residual form R_i :

$$R_1 = \nabla \cdot \mathbf{v} \quad (2.4)$$

$$(R_2, R_3, R_4)^T = (\mathbf{v} \cdot \nabla) \mathbf{v} + \nabla p - \nabla \cdot (\nu(\nabla \mathbf{v} + (\nabla \mathbf{v})^T)) + \alpha(\eta) \mathbf{v} \quad (2.5)$$

$$R_5 = (\mathbf{v} \cdot \nabla)T - \nabla \cdot (D_T(\eta)\nabla T) \quad (2.6)$$

$$R_6 = \nabla \cdot (D_{T_{solid}} \nabla T) \quad (2.7)$$

\mathbf{v} represents the vectorial velocity of the flow and p represents, as stated above, the ratio between the pressure and the density of the flow.

The porosity of the cells is considered in the momentum equation through the term $\alpha(\eta)\mathbf{v}$. This term, referred to as the Darcy term, represents the momentum penalty given by the cells with porosity different from zero and can be treated as a sink term. When the porosity of the cell is sufficiently large, the Darcy term dominates the momentum equation and the convective and diffusive terms can be neglected, obtaining the banal equation for a solid cell:

$$\alpha_{max}\mathbf{v} = 0 \quad (2.8)$$

that implies a velocity of the flow equal to zero, and so a non-penetrating boundary condition for the flow in the considered cell.

D_T is the interpolated thermal diffusivity, whose dependence with the design variable η (and so the porosity of the cells) is exploited; when the porosity is zero the conductivity assumes the value of $D_{T_{fluid}}$, when the porosity is α_{max} the conductivity assumes the value of $D_{T_{solid}}$. As stated above, if a cell is completely solid, the velocity in that cell is zero: this is reflected also in the temperature equation of the fluid, which becomes a pure Laplace diffusion equation:

$$\nabla \cdot (D_{T_{solid}}\nabla T) = 0 \quad (2.9)$$

Note that, for a Conjugated Heat Transfer problem, the conditions that need to hold at the interface between the fluid and solid cells are the following:

$$T_{fluid} = T_{solid} \quad (2.10)$$

and

$$\int_{\Gamma_{interface}} D_{T_{fluid}} \hat{\mathbf{n}} \cdot \nabla T_{fluid} = \int_{\Gamma_{interface}} D_{T_{solid}} \hat{\mathbf{n}} \cdot \nabla T_{solid} \quad (2.11)$$

which represent respectively the temperature and the heat flux continuity between the fluid and the solid interface. Given the method employed to solve the equations (the finite volume method), the first condition will always hold, because values of temperature at the shared face center are interpolated using the values of the neighbouring cells centers, and so the temperatures are unique. The conservation of the heat fluxes is granted instead from the equation of temperature of the flow.

Considering the above equation, the optimization problem can be formulated as follows, where \mathbf{R} is the vector containing the R_i equations:

$$\begin{aligned} & \text{minimize } J = J(\eta, \mathbf{v}, p, T) \\ & \text{subject to } \mathbf{R}(\eta, \mathbf{v}, p, T) = 0 \end{aligned}$$

2.2. Adjoint problem equations

To derive the adjoint problem equations it is needed to define a Lagrange function L augmenting the objective function J with the Lagrange multipliers, multiplied by the primal problem equations \mathbf{R} :

$$L = J + \int_{\Omega} (p_a, \mathbf{u}_a, T_a, T_{a_s}) \mathbf{R} d\Omega \quad (2.12)$$

where Ω is the problem's domain and p_a , \mathbf{u}_a , and T_a are the Lagrange multipliers introduced to obtain the adjoint equations, and they respectively take the name of adjoint pressure, adjoint velocity and adjoint temperature; it is useful to note that they are purely mathematical objects without any physical meanings. Since it must be $\mathbf{R}=0$, the integral added to the objective function is null, and so the optimized solution for L is the same optimized solution for J .

The method employed for the optimization of L is the method of moving asymptotes (MMA), which transforms the problem into an equivalent convex problem that has better convergence properties; references for the math on which is based the method can be found in [13] [15], whereas a more practical application in structural optimization problems can be found in [14]; furthermore, an implementation in Matlab of the MMA algorithm can be found in [12] To perform the optimization, MMA requires the values of the sensitivity of the objective function $\frac{dL}{d\eta}$, and the continuous adjoint method offers a very efficient way to compute it. Considering the total variation of L :

$$dL = \delta L = \delta_\eta L + \delta_v L + \delta_p L + \delta_T L + \delta_{T_s} L \quad (2.13)$$

where, for example, $\delta_\eta L = \frac{\partial L}{\partial \eta} d\eta$. The Lagrange multipliers are selected such that:

$$\delta_v L + \delta_p L + \delta_T L + \delta_{T_s} L = 0 \quad (2.14)$$

From this condition, the equations for the adjoint problem are produced, and they will need to be solved for the adjoint variables p_a , \mathbf{u}_a and T_a . The satisfaction of (2.14) implies

$$\frac{dL}{d\eta} = \frac{\partial J}{\partial \eta} + \int_{\Omega} (p_a, \mathbf{u}_a, T_a, T_{a_s}) \delta_\eta \mathbf{R} d\Omega \quad (2.15)$$

Since the design variable is used to represent a continuous transition between solid and liquid, as a result, there is no explicit dependence of the objective function on η , and so the term $\frac{\partial J}{\partial \eta}$ is null. From equation 2.15 it is clear that, to compute the sensitivity, the values of the adjoint variables are needed, as well as the values of the derivative of \mathbf{R} with respect to η , which are easily obtainable from the primal problem solution. Performing the derivations of the constraints \mathbf{R} with respect to η , the following set of equations is obtained:

$$\frac{\partial \mathbf{R}}{\partial \eta} = \begin{pmatrix} 0 \\ (\frac{\partial \alpha}{\partial \eta})v \\ \nabla \cdot (\frac{\partial D_T}{\partial \eta} \nabla T) \\ 0 \end{pmatrix} \quad (2.16)$$

Note that the porosity α and the thermal diffusivity D_T are dependent on the design variable η , since they are proportional to the quantity of solid material distribution. These two derivatives will be computed adopting the RAMP interpolation model, which will be described in 3; however, this model will not be introduced in the following derivation of the adjoint equations, for the sake of simplicity. Substituting 2.16 in 2.15 and performing the multiplications with the Lagrange multipliers:

$$\frac{dL}{d\eta} = \int_{\Omega} \left(\mathbf{u}_a \cdot \mathbf{v} \frac{\partial \alpha}{\partial \eta} \right) d\Omega - \int_{\Omega} T_a \nabla \cdot \left(\frac{\partial D_T}{\partial \eta} \nabla T \right) d\Omega \quad (2.17)$$

Considering the second integral of 2.17, integrating by parts and applying the Gauss theorem:

$$\begin{aligned}
& - \int_{\Omega} T_a \nabla \cdot \left(\frac{\partial D_T}{\partial \eta} \nabla T \right) d\Omega = - \int_{\Omega} \nabla \cdot \left(T_a \frac{\partial D_T}{\partial \eta} \nabla T \right) d\Omega + \\
& + \int_{\Omega} \frac{\partial D_T}{\partial \eta} \nabla T_a \cdot \nabla T d\Omega = \\
& = - \int_{\Gamma} \left(T_a \frac{\partial D_T}{\partial \eta} \nabla T \right) \cdot \hat{n} d\Gamma + \int_{\Omega} \frac{\partial D_T}{\partial \eta} \nabla T_a \cdot \nabla T d\Omega = \\
& = \int_{\Omega} \frac{\partial D_T}{\partial \eta} \nabla T_a \cdot \nabla T d\Omega
\end{aligned} \tag{2.18}$$

Where the surface integral vanishes since, as will be seen in the following section, the boundary conditions for an adiabatic patch are set to *zeroGradient*, whereas for fixed heat flux patches the value of T_a will be set to zero. Once the integration by parts and the Gauss theorem have been applied, the equation for the computation of the sensitivity reads:

$$\frac{dL}{d\eta} = \int_{\Omega} \left(\mathbf{u}_a \cdot \mathbf{v} \frac{\partial \alpha}{\partial \eta} \right) d\Omega + \int_{\Omega} \frac{\partial D_T}{\partial \eta} \nabla T_a \cdot \nabla T d\Omega \tag{2.19}$$

which in the finite volume framework, if the general cell i has a volume V_i :

$$\frac{dL}{d\eta_i} = \left(\mathbf{u}_{a_i} \cdot \mathbf{v}_i \frac{\partial \alpha}{\partial \eta_i} \right) V_i + \frac{\partial D_T}{\partial \eta_i} \nabla T_{a_i} \cdot \nabla T_i V_i \tag{2.20}$$

Equation 2.20 shows that to compute the sensitivity the quantities involved are the adjoint and primal velocity and temperature, and also the derivative of thermal diffusivity with respect to the design variable. The primal velocity and temperature are obtained through the solution of the primal problem equations, whereas the adjoint quantities are obtained from the solution of equation 2.13; in the last part of the section, a solution for this equation will be presented.

Considering equation 2.13, and substituting the value of the augmented objective function L from equation 2.12:

$$\begin{aligned}
\delta_v L + \delta_p L + \delta_T L &= \delta_v J + \delta_p J + \delta_T J + \delta_{T_s} J + \\
& + \int_{\Omega} (p_a, \mathbf{u}_a, T_a, T_{a_s}) \delta_v \mathbf{R} d\Omega + \int_{\Omega} (p_a, \mathbf{u}_a, T_a, T_{a_s}) \delta_p \mathbf{R} d\Omega + \\
& + \int_{\Omega} (p_a, \mathbf{u}_a, T_a, T_{a_s}) \delta_T \mathbf{R} d\Omega + \int_{\Omega} (p_a, \mathbf{u}_a, T_a, T_{a_s}) \delta_{T_s} \mathbf{R} d\Omega = 0
\end{aligned} \tag{2.21}$$

Here are listed the products between the derivatives δ and the constraints R_i ; note that

this passage constitutes a linearization of the governing equations:

$$\delta_{\mathbf{v}}R_1 = -\nabla \cdot \delta\mathbf{v} \quad (2.22)$$

$$\delta_{\mathbf{v}}(R_2, R_3, R_4)^T = (\delta\mathbf{v} \cdot \nabla)\mathbf{v} + (\mathbf{v} \cdot \nabla)\delta\mathbf{v} - \nabla \cdot (\nu(\nabla\delta\mathbf{v} + (\nabla\delta\mathbf{v})^T)) + \alpha\delta\mathbf{v} \quad (2.23)$$

$$\delta_{\mathbf{v}}R_5 = (\delta\mathbf{v} \cdot \nabla)T \quad (2.24)$$

$$\delta_{\mathbf{v}}R_6 = 0 \quad (2.25)$$

$$\delta_p R_1 = 0 \quad (2.26)$$

$$\delta_p(R_2, R_3, R_4)^T = \nabla\delta p \quad (2.27)$$

$$\delta_p R_5 = 0 \quad (2.28)$$

$$\delta_p R_6 = 0 \quad (2.29)$$

$$\delta_T R_1 = 0 \quad (2.30)$$

$$\delta_T(R_2, R_3, R_4)^T = 0 \quad (2.31)$$

$$\delta_T R_5 = (\mathbf{v} \cdot \nabla)\delta T - \nabla \cdot (D_T \nabla \delta T) \quad (2.32)$$

$$\delta_T R_6 = 0 \quad (2.33)$$

$$\delta_{T_S}(R_1, R_2, R_3, R_4, R_5)^T = 0 \quad (2.34)$$

$$\delta_{T_S}R_6 = \nabla \cdot (D_{T_{solid}}\nabla\delta T_S) \quad (2.35)$$

In the operations performed so far, the cinematic viscosity ν hasn't been treated as a function of the design variable η : this is an approximation called "frozen turbulence" that has to be done in order to simplify the problem, that undermines the accuracy of the wall shear models computation. The approach without this type of approximation would require adding to \mathbf{R} another set of equations describing the turbulence model, and so also a set of adjoint variables related to the turbulence quantities; these equations are dependent on the turbulence model that is chosen, in [4] can be found a derivation for the Spalart-Allmaras approach, whereas in [11] a practical application of this method can be appreciated. This would require also considering the variation of L with respect to the turbulence quantities. However, this approximation isn't in general considered as a source of consistent errors except for the computation of the wall shear models, and so it would surely be misleading to use this method to extract the wall shear stresses. Nevertheless, in the case of laminar flows, it is correct to treat the viscosity as constant with η . Substituting the linearized equations in 2.21, the equation becomes:

$$\begin{aligned} & \delta_v J + \delta_p J + \delta_T J + \delta_{T_S} J + \int_{\Omega} p_a (-\nabla \cdot \delta \mathbf{v}) d\Omega + \\ & + \int_{\Omega} \mathbf{u}_a \cdot ((\delta v \cdot \nabla) \mathbf{v} + (\mathbf{v} \cdot \nabla) \delta \mathbf{v} - \nabla \cdot (\nu (\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v})^T)) + \alpha \delta \mathbf{v}) d\Omega + \\ & + \int_{\Omega} \mathbf{u}_a \cdot \nabla \delta p d\Omega + \int_{\Omega} T_a (\delta \mathbf{v} \cdot \nabla) T d\Omega + \\ & + \int_{\Omega} T_a (\mathbf{v} \cdot \nabla) \delta T - \nabla \cdot (D_T \nabla T \delta T) d\Omega + \\ & + \int_{\Omega} \nabla \cdot (D_T \nabla \delta T_S) T_{a_S} = 0 \end{aligned} \quad (2.36)$$

Given the presence of the variations δp , $\delta \mathbf{v}$ and δT inside the gradient operators, an integration by parts followed by the application of the Gauss theorem of these terms is needed to obtain the variations multiplying the factors in the volume integral. Considering the first integral:

$$\begin{aligned}
\int_{\Omega} p_a(-\nabla \cdot \delta \mathbf{v}) d\Omega &= \int_{\Omega} -\nabla \cdot (p_a \delta \mathbf{v}) d\Omega + \int_{\Omega} -\nabla p_a \cdot \delta \mathbf{v} d\Omega = \\
&= \int_{\Gamma} -p_a \hat{\mathbf{n}} \cdot \delta \mathbf{v} d\Gamma + \int_{\Omega} p_a \cdot \delta \mathbf{v} d\Omega
\end{aligned} \tag{2.37}$$

Now, considering one half of the second integral, the integration by parts is performed two times and, again, the Gauss theorem is applied:

$$\begin{aligned}
&\int_{\Omega} \mathbf{u}_a \cdot ((\delta \cdot \nabla) \mathbf{v} + (\mathbf{v} \cdot \nabla) \delta \mathbf{v}) d\Omega = \\
&= \int_{\Omega} \nabla \cdot ((\mathbf{u}_a \cdot \mathbf{v} \delta \mathbf{v}) d\Omega + \int_{\Omega} \nabla \cdot ((\mathbf{u}_a \cdot \delta \mathbf{v}) \mathbf{v}) d\Omega + \\
&- \int_{\Omega} \mathbf{v} \cdot (\nabla \cdot (\mathbf{u}_a \times \delta \mathbf{v})) d\Omega - \int_{\Omega} \delta \mathbf{v} \cdot (\nabla \cdot (\mathbf{u}_a \times \mathbf{v})) d\Omega = \\
&= \int_{\Gamma} (\hat{\mathbf{n}} (\mathbf{u}_a \cdot \mathbf{v}) + \mathbf{u}_a (\mathbf{v} \cdot \hat{\mathbf{n}})) \cdot \delta \mathbf{v} d\Gamma - \int_{\Omega} ((\nabla \mathbf{u}_a)^T \mathbf{v} + (\mathbf{v} \cdot \nabla) \mathbf{u}_a) \cdot \delta \mathbf{v} d\Omega
\end{aligned} \tag{2.38}$$

Where for the terms $\nabla \cdot (\mathbf{u}_a \times \delta \mathbf{v})$ and $\nabla \cdot (\mathbf{u}_a \times \mathbf{v})$ have been exploited the flow incompressibility, and so:

$$\nabla \cdot (\mathbf{u}_a \times \delta \mathbf{v}) = \underline{\mathbf{u}_a \nabla \cdot \delta \mathbf{v}} + (\delta \mathbf{v} \cdot \nabla) \mathbf{u}_a = (\delta \mathbf{v} \cdot \nabla) \mathbf{u}_a \tag{2.39}$$

$$\nabla \cdot (\mathbf{u}_a \times \mathbf{v}) = \underline{\mathbf{u}_a \nabla \cdot \mathbf{v}} + (\mathbf{v} \cdot \nabla) \mathbf{u}_a = (\mathbf{v} \cdot \nabla) \mathbf{u}_a \tag{2.40}$$

The operations performed in the previous step can be better appreciated by exploiting Einstein notation; in this way can be noted that the terms $\frac{\partial \delta v_j}{\partial x_j}$ and $\frac{\partial \delta v_j}{\partial x_j}$ are null since the flow is stationary and incompressible, and so divergence-free.

$$\begin{aligned}
&\int_{\Omega} u_{a_i} \left(\left(\delta v_j \frac{\partial}{\partial x_j} v_i \right) + \left(v_j \frac{\partial}{\partial x_j} \right) \delta v_i \right) d\Omega = \\
&= \int_{\Omega} \frac{\partial}{\partial x_j} (u_{a_i} \delta v_j v_i) d\Omega + \int_{\Omega} \frac{\partial}{\partial x_j} (u_{a_i} v_j \delta v_i) d\Omega - \int_{\Omega} v_i \frac{\partial (u_{a_i} \delta v_j)}{\partial x_j} d\Omega + \\
&- \int_{\Omega} \delta v_i \frac{\partial (u_{a_i} v_i)}{\partial x_j} d\Omega = \\
&= \int_{\Gamma} n_j (u_{a_i} v_i) \delta v_j d\Gamma + \int_{\Gamma} u_{a_i} (v_j n_j) \delta v_i d\Gamma - \int_{\Omega} v_i \left(\frac{\partial u_{a_i}}{\partial x_j} \right) \delta v_j d\Omega + \\
&- \int_{\Omega} v_j \left(\frac{\partial u_{a_i}}{\partial x_j} \right) \delta v_i d\Omega
\end{aligned} \tag{2.41}$$

Considering now the second half of the second integral of 2.36 and applying integration by parts and the Gauss theorem:

$$\begin{aligned}
& \int_{\Omega} \mathbf{u}_a \cdot (-\nabla \cdot (\nu(\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v})^T))) d\Omega = \\
& = \int_{\Omega} \nabla \cdot (-\nu(\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v})^T) \mathbf{u}_a) d\Omega + \int_{\Omega} (\nu(\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v})^T)) : \nabla \mathbf{u}_a d\Omega = \\
& = \int_{\Gamma} -\nu \hat{\mathbf{n}}^T ((\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v})^T) \mathbf{u}_a) d\Gamma + \int_{\Omega} (\nu(\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v})^T)) : \nabla \mathbf{u}_a d\Omega = \\
& = \int_{\Gamma} -\mathbf{u}_a^T (\nu(\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v})^T) \hat{\mathbf{n}}) d\Gamma + \\
& + \int_{\Omega} \nabla \cdot (\nu(\nabla \mathbf{u}_a) \delta \mathbf{v}) + \nabla \cdot (\nu(\nabla \mathbf{u}_a)^T \delta \mathbf{v}) d\Omega + \\
& + \int_{\Omega} (-\nabla \cdot (\nu \nabla \mathbf{u}_a) \cdot \nabla \mathbf{v} - \nabla \cdot (\nu(\nabla(\mathbf{u}_a)^T) \cdot \delta \mathbf{v})) d\Omega = \tag{2.42} \\
& = \int_{\Gamma} (-\nu(\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v})^T) \hat{\mathbf{n}}) \cdot \mathbf{u}_a d\Gamma + \int_{\Omega} \nabla \cdot (\nu(\nabla \mathbf{u}_a + (\nabla \mathbf{u}_a)^T) \delta \mathbf{v}) d\Omega + \\
& + \int_{\Omega} (-\nabla \cdot (\nu(\nabla \mathbf{u}_a + (\nabla \mathbf{u}_a)^T))) \cdot \delta \mathbf{v} d\Omega + \\
& + \int_{\Omega} (-\nabla \cdot (\nu(\nabla \mathbf{u}_a + (\nabla \mathbf{u}_a)^T))) \cdot \delta \mathbf{v} d\Omega = \\
& = \int_{\Gamma} (-\nu(\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v})^T) \hat{\mathbf{n}}) \cdot \mathbf{u}_a d\Gamma + \int_{\Gamma} (\nu(\nabla \mathbf{u}_a + (\nabla \mathbf{u}_a)^T) \hat{\mathbf{n}}) \cdot \delta \mathbf{v} d\Gamma + \\
& + \int_{\Omega} (-\nabla \cdot (\nu(\nabla \mathbf{u}_a + (\nabla \mathbf{u}_a)^T))) \cdot \delta \mathbf{v} d\Omega
\end{aligned}$$

The term $\alpha \delta \mathbf{v}$ hasn't been included in the operations since $\delta \mathbf{v}$ is already operator-free. Again, the manipulations can be better appreciated by exploiting Einstein's notation:

$$\begin{aligned}
& \int_{\Omega} u_{a_i} \left(-\frac{\partial}{\partial x_j} \left(\nu \left(\frac{\partial \delta v_i}{\partial x_j} + \frac{\partial \delta v_j}{\partial x_i} \right) \right) \right) d\Omega = \\
& = \int_{\Omega} \frac{\partial}{\partial x_j} \left(-\nu \left(\frac{\partial \delta v_i}{\partial x_j} + \frac{\partial \delta v_j}{\partial x_i} \right) u_{a_i} \right) d\Omega + \int_{\Omega} \nu \left(\frac{\partial \delta v_i}{\partial x_j} + \frac{\partial \delta v_j}{\partial x_i} \right) \frac{\partial u_{a_i}}{\partial x_j} d\Omega = \\
& = \int_{\Gamma} -\nu n_j \left(\frac{\partial \delta v_i}{\partial x_j} + \frac{\partial \delta v_j}{\partial x_i} \right) u_{a_i} d\Gamma + \\
& + \int_{\Omega} \left(\frac{\partial}{\partial x_j} \left(\nu \delta v_i \frac{\partial u_{a_i}}{\partial x_j} \right) + \frac{\partial}{\partial x_i} \left(\nu \delta v_j \frac{\partial u_{a_i}}{\partial x_j} \right) \right) d\Omega + \\
& - \int_{\Omega} \left(\delta v_i \frac{\partial}{\partial x_j} \left(\nu \frac{\partial u_{a_i}}{\partial x_j} \right) + \delta v_i \frac{\partial}{\partial x_i} \left(\nu \frac{\partial u_{a_i}}{\partial x_j} \right) \right) d\Omega = \\
& = \int_{\Gamma} -\nu n_j \left(\frac{\partial \delta v_i}{\partial x_j} + \frac{\partial \delta v_j}{\partial x_i} \right) u_{a_i} d\Gamma + \\
& + \int_{\Omega} \left(\frac{\partial}{\partial x_j} \left(\nu \delta v_i \frac{\partial u_{a_i}}{\partial x_j} \right) + \frac{\partial}{\partial x_j} \left(\nu \delta v_i \frac{\partial u_{a_j}}{\partial x_i} \right) \right) d\Omega + \\
& - \int_{\Omega} \left(\delta v_i \frac{\partial}{\partial x_j} \left(\nu \frac{\partial u_{a_i}}{\partial x_j} \right) + \delta v_i \frac{\partial}{\partial x_j} \left(\nu \frac{\partial u_{a_j}}{\partial x_i} \right) \right) d\Omega \\
& = \int_{\Gamma} -\nu n_j \left(\frac{\partial \delta v_i}{\partial x_j} + \frac{\partial \delta v_j}{\partial x_i} \right) u_{a_i} d\Gamma + \int_{\Omega} \frac{\partial}{\partial x_j} \left(\nu \left(\frac{\partial u_{a_i}}{\partial x_j} + \frac{\partial u_{a_j}}{\partial x_i} \right) \delta v_i \right) d\Omega + \\
& - \int_{\Omega} \frac{\partial}{\partial x_j} \left(\nu \left(\frac{\partial u_{a_i}}{\partial x_j} + \frac{\partial u_{a_j}}{\partial x_i} \right) \right) \delta v_i d\Omega = \\
& = \int_{\Gamma} -\nu n_j \left(\frac{\partial \delta v_i}{\partial x_j} + \frac{\partial \delta v_j}{\partial x_i} \right) u_{a_i} d\Gamma + \int_{\Gamma} \nu n_j \left(\frac{\partial u_{a_i}}{\partial x_j} + \frac{\partial u_{a_j}}{\partial x_i} \right) \delta v_i d\Gamma + \\
& - \int_{\Omega} \frac{\partial}{\partial x_j} \left(\nu \left(\frac{\partial u_{a_i}}{\partial x_j} + \frac{\partial u_{a_j}}{\partial x_i} \right) \right) \delta v_i d\Omega
\end{aligned} \tag{2.43}$$

Now, consider the third integral of 2.36:

$$\begin{aligned}
& \int_{\Omega} \mathbf{u}_a \cdot \nabla \delta p d\Omega = \int_{\Omega} \nabla \cdot (\mathbf{u}_a \delta p) \delta p d\Omega - \int_{\Omega} (\nabla \cdot \mathbf{u}_a) \delta p d\Omega = \\
& = \int_{\Gamma} (\mathbf{u}_a \cdot \hat{\mathbf{n}}) \delta p d\Gamma - \int_{\Omega} (\nabla \cdot \mathbf{u}_a) \delta p d\Omega
\end{aligned} \tag{2.44}$$

Then, the fourth integral of 2.36:

$$\begin{aligned}
& \int_{\Omega} T_a ((\delta \mathbf{v} \cdot \nabla) T) d\Omega = \int_{\Omega} \nabla \cdot (T_a \delta \mathbf{v}) T d\Omega - \int_{\Omega} (T \nabla T_a) \cdot \delta \mathbf{v} d\Omega = \\
& = \int_{\Gamma} (T_a \hat{\mathbf{n}} T) \cdot \delta \mathbf{v} d\Gamma - \int_{\Omega} (T \nabla T_a) \cdot \delta \mathbf{v} d\Omega
\end{aligned} \tag{2.45}$$

Apply the same manipulations on the fifth integral of 2.36:

$$\begin{aligned}
\int_{\Omega} T_a(\mathbf{v} \cdot \nabla) \delta T d\Omega &= \int_{\Omega} \nabla \cdot (T_a \mathbf{v} \delta T) d\Omega - \int_{\Omega} \nabla \cdot (\mathbf{v} T_a) \delta T d\Omega = \\
&= \int_{\Gamma} T_a(\mathbf{v} \cdot \hat{\mathbf{n}}) \delta T d\Gamma - \int_{\Omega} (\mathbf{v} \cdot \nabla T_a) \delta T d\Omega
\end{aligned} \tag{2.46}$$

Then, the sixth integral of 2.36:

$$\begin{aligned}
\int_{\Omega} T_a \nabla \cdot (D_T \nabla \delta T) d\Omega &= \int_{\Omega} \nabla \cdot (T_a D_T \nabla \delta T) d\Omega - \int_{\Omega} \nabla T_a \cdot (D_T \nabla \delta T) d\Omega = \\
&= \int_{\Gamma} T_a D_T \hat{\mathbf{n}} \cdot \nabla \delta T d\Gamma - \int_{\Omega} \nabla T_a \cdot (D_T \nabla \delta T) d\Omega = \\
&= \int_{\Gamma} T_a D_T \hat{\mathbf{n}} \cdot \nabla \delta T d\Gamma - \int_{\Omega} \nabla \cdot (\nabla T_a D_T \delta T) d\Omega + \int_{\Omega} \nabla \cdot \nabla T_a D_T \delta T d\Omega = \\
&= \int_{\Gamma} T_a D_T \hat{\mathbf{n}} \cdot \nabla \delta T d\Gamma - \int_{\Gamma} \nabla T_a \cdot \hat{\mathbf{n}} D_T \delta T d\Gamma + \int_{\Omega} \nabla^2 T_a D_T \delta T d\Omega
\end{aligned} \tag{2.47}$$

The same applies to the last integral regarding the solid temperature:

$$\begin{aligned}
\int_{\Omega} T_{a_s} \nabla \cdot (D_T \nabla \delta T_s) d\Omega &= \int_{\Omega} \nabla \cdot (T_{a_s} D_T \nabla \delta T_s) d\Omega - \int_{\Omega} \nabla T_{a_s} \cdot (D_T \nabla \delta T_s) d\Omega = \\
&= \int_{\Gamma} T_{a_s} D_T \hat{\mathbf{n}} \cdot \nabla \delta T_s d\Gamma - \int_{\Omega} \nabla T_{a_s} \cdot (D_T \nabla \delta T_s) d\Omega = \\
&= \int_{\Gamma} T_{a_s} D_T \hat{\mathbf{n}} \cdot \nabla \delta T_s d\Gamma - \int_{\Omega} \nabla \cdot (\nabla T_{a_s} D_T \delta T_s) d\Omega + \int_{\Omega} \nabla \cdot \nabla T_{a_s} D_T \delta T_s d\Omega = \\
&= \int_{\Gamma} T_{a_s} D_T \hat{\mathbf{n}} \cdot \nabla \delta T_s d\Gamma - \int_{\Gamma} \nabla T_{a_s} \cdot \hat{\mathbf{n}} D_T \delta T_s d\Gamma + \int_{\Omega} \nabla^2 T_{a_s} D_T \delta T_s d\Omega
\end{aligned} \tag{2.48}$$

The variations of the objective function J can be expressed as the sum of the one contribution from the inside of the domain and a contribution from the boundary of the domain. Given the fact that the objective functions considered for the case at hand are defined on the inlet and outlet, the contribution of the variation of the objective function inside the domain will be null.

$$\begin{aligned}
& \delta_p J + \delta_{\mathbf{v}} J + \delta_T J + \delta_{T_S} J = \\
& = \int_{\Gamma} \frac{\partial J_{\Gamma}}{\partial p} \delta p d\Gamma + \int_{\Omega} \frac{\partial J_{\Omega}}{\partial p} \delta p d\Omega + \int_{\Gamma} \frac{\partial J_{\Gamma}}{\partial \mathbf{v}} \delta \mathbf{v} d\Gamma + \int_{\Omega} \frac{\partial J_{\Omega}}{\partial \mathbf{v}} \delta \mathbf{v} d\Omega + \\
& + \int_{\Gamma} \frac{\partial J_{\Gamma}}{\partial T} \delta T d\Gamma + \int_{\Omega} \frac{\partial J_{\Omega}}{\partial T} \delta T d\Omega \\
& + \frac{\partial J_{\Gamma}}{\partial T_S} \delta T_S d\Gamma + \int_{\Omega} \frac{\partial J_{\Omega}}{\partial T_S} \delta T_S d\Omega
\end{aligned} \tag{2.49}$$

Now, the equations that result from the manipulation performed above can be substituted for inside equation 2.36 and, grouping the terms multiplied by δp , $\delta \mathbf{u}_a$ and δT the following expression is obtained:

$$\begin{aligned}
& \int_{\Omega} \left(-\nabla \cdot \mathbf{u}_a + \frac{\partial J_{\Omega}}{\partial p} \right) \delta p d\Omega + \\
& + \int_{\Omega} \left(-(\mathbf{v} \cdot \nabla) \mathbf{u}_a - (\nabla \mathbf{u}_a)^T \mathbf{v} - \nabla \cdot (\nu (\nabla \mathbf{u}_a + (\nabla \mathbf{u}_a)^T)) \right) \cdot \delta \mathbf{v} d\Omega + \\
& + \int_{\Omega} \left(\nabla p_a + \alpha \mathbf{u}_a - T \nabla T_a + \frac{\partial J_{\Omega}}{\partial \mathbf{v}} \right) \cdot \delta \mathbf{v} d\Omega + \\
& + \int_{\Omega} \left(-\mathbf{v} \cdot \nabla T_a - D_T \nabla^2 T_a + \frac{\partial J_{\Omega}}{\partial T} \right) \delta T d\Omega + \int_{\Omega} \left(\nabla^2 T_{a_S} D_T + \frac{\partial J_{\Omega}}{\partial T_S} \right) \delta T_S d\Omega \\
& + \int_{\Gamma} \left(\mathbf{u}_a \cdot \hat{n} + \frac{\partial J_{\Gamma}}{\partial p} \right) \cdot \delta p d\Gamma + \\
& + \int_{\Gamma} \left(\hat{n} (\mathbf{u}_a \cdot \mathbf{v}) + \mathbf{u}_a (\mathbf{v} \cdot \hat{n}) - p_a \hat{n} + \nu (\nabla \mathbf{u}_a + (\nabla \mathbf{u}_a)^T) \hat{n} + \right. \\
& + T_a \hat{n} T + \left. \frac{\partial J_{\Gamma}}{\partial \mathbf{v}} \right) \cdot \delta \mathbf{v} d\Gamma + \\
& - \int_{\Gamma} \left(\nu (\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v})^T) \hat{n} \right) \cdot \mathbf{u}_a d\Gamma + \\
& + \int_{\Gamma} \left(T_a (\mathbf{v} \cdot \hat{n}) + D_T (\nabla T_a \cdot \hat{n}) + \frac{\partial J_{\Gamma}}{\partial T} \right) \delta T d\Gamma + \\
& - \int_{\Gamma} D_T (\hat{n} \cdot \nabla \delta T) T_a d\Gamma \\
& + \int_{\Gamma} (T_{a_S} D_T \hat{n} \cdot \nabla \delta T_S) d\Gamma - \int_{\Gamma} \left(\nabla T_{a_S} \cdot \hat{n} D_T + \frac{\partial J_{\Gamma}}{\partial T_S} \right) \delta T_S d\Gamma = 0
\end{aligned} \tag{2.50}$$

Equation 2.50 implies that every integral has to be null to satisfy the condition. This equation provides both boundary conditions in the form of surface integrals and the governing equations that have to be solved by the solver to obtain the adjoint variables \mathbf{u}_a , p_a , T_a needed to compute the sensitivity for the objective function J with respect to

η . Considering the term multiplying δp , the adjoint mass conservation is obtained:

$$-\nabla \cdot \mathbf{u}_a + \frac{\partial J_\Omega}{\partial p} = 0 \quad (2.51)$$

Then, considering the term multiplying $\delta \mathbf{v}$, the momentum equation is obtained:

$$\begin{aligned} & -(\mathbf{v} \cdot \nabla) \mathbf{u}_a - (\nabla \mathbf{u}_a)^T \mathbf{v} - \nabla \cdot (\nu(\nabla \mathbf{u}_a + (\nabla \mathbf{u}_a)^T)) + \\ & + \nabla p_a + \alpha \mathbf{u}_a - T \nabla T_a + \frac{\partial J_\Omega}{\partial \mathbf{v}} = 0 \end{aligned} \quad (2.52)$$

Note the term $-T \nabla T_a$: this indicates a one way coupling between the adjoint momentum equation and the adjoint temperature equation. In the primal problem the coupling is present but in a converse manner: it is the velocity that affects the temperature equation through the convection term, whereas the momentum equation is not affected by the temperature, since the density is assumed constant and no models for density change in function of temperature are introduced. Considering the terms in 2.50 multiplied by δT , the adjoint temperature equation is obtained:

$$-(\mathbf{v} \cdot \nabla) T_a - D_T \nabla^2 T_a + \frac{\partial J_\Omega}{\partial T} = 0 \quad (2.53)$$

As mentioned before, note the absence of the adjoint velocity in the temperature equation: this latter depends just on the convection from the primary velocity field, the thermal diffusivity and the adjoint temperature. The equation for the adjoint temperature in the solid region is represented by the terms in 2.50 multiplied by δT_s :

$$\nabla^2 T_{a_s} D_T + \frac{\partial J_\Omega}{\partial T_s} = 0 \quad (2.54)$$

Finally, note that for the case at hand the objective functions are referred to the inlet and the outlet, and this means that there is no contribution to them from the interior domain and so the derivatives of J in equation 2.51, 2.52, 2.53 and 2.54 can be neglected.

2.3. Adjoint problem boundary conditions

The adjoint boundary conditions are a direct consequence of the integrals over the boundary of the domain in equation 2.50. These integrals, here repeated for clarity, have to be null to satisfy the condition imposed by the equation.

$$\begin{aligned}
& \int_{\Gamma} \left(\mathbf{u}_a \cdot \hat{n} + \frac{\partial J_{\Gamma}}{\partial p} \right) \delta p d\Gamma + \\
& + \int_{\Gamma} (\hat{n}(\mathbf{u}_a \cdot \mathbf{v}) + \mathbf{u}_a(\mathbf{v} \cdot \hat{n} - p_a \hat{n} + \nu (\nabla \mathbf{u}_a + (\nabla \mathbf{u}_a)^T) \hat{n} + \\
& + T_a \hat{n} T + \frac{\partial J_{\Gamma}}{\partial \mathbf{v}}) \cdot \delta \mathbf{v} d\Gamma + \\
& - \int_{\Gamma} (\nu (\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v}^T) \hat{n}) \cdot \mathbf{u}_a d\Gamma + \\
& - \int_{\Gamma} (\nu (\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v}^T) \hat{n}) \cdot \mathbf{u}_a d\Gamma + \\
& + \int_{\Gamma} \left(T_a(\mathbf{v} \cdot \hat{n} + D_T(\nabla T_a \cdot \hat{n}) + \frac{\partial J_{\Gamma}}{\partial T}) \right) \delta T d\Gamma - \int_{\Gamma} D_T(\hat{n} \cdot \nabla \delta T) T_a d\Gamma \\
& + \int_{\Gamma} (T_{a_s} D_T \hat{n} \cdot \nabla \delta T_s) d\Gamma - \int_{\Gamma} \left(\nabla T_{a_s} \cdot \hat{n} D_T + \frac{\partial J_{\Gamma}}{\partial T_s} \right) \delta T_s d\Gamma
\end{aligned} \tag{2.55}$$

Given the nature of the case at hand, it is convenient to decompose the boundary of the domain into three zones, the inlet, the outlet and the walls: $\Gamma = \Gamma_{inlet} + \Gamma_{outlet} + \Gamma_{wall}$

2.3.1. Inlet patch boundary conditions

Considering the inlet patch, in general, in the primal problem a fixed velocity and a fixed temperature are imposed as boundary conditions: this implies that $\delta \mathbf{v}$ and δT are null on this boundary. So, in equation 2.55 just remains the integral:

$$\int_{\Gamma_{inlet}} \left(\mathbf{u}_a \cdot \hat{n} + \frac{\partial J_{\Gamma}}{\partial p} \right) \delta p d\Gamma \tag{2.56}$$

To make the integral vanish, the condition that has to be imposed on the normal component of the velocity is:

$$u_{a_n} = -\frac{\partial J_{\Gamma}}{\partial p} \tag{2.57}$$

That, considering the objective function in the case at hand, can be rewritten as:

$$u_{a_n} = \omega_1 v_n \quad (2.58)$$

The equations don't provide any boundary condition for \mathbf{u}_{a_t} , p_a and T_a at the inlet, however, since the flow of the primal problem is normal to the inlet, a null condition for \mathbf{u}_{a_t} can be imposed. Considering p_a , since as will be discussed in the outlet patch section, a fixed value will be imposed at the outlet, it is reasonable to impose a *zeroGradient* boundary condition at the inlet, to not over constrain the problem. Regarding T_a , no arbitrary condition can be imposed for this quantity at the inlet, however, considering the formulation for the sensitivity computation that derives from 2.18, the adjoint temperature is required to be zero on the inlet patch so that this formulation can be used.

2.3.2. Fixed heat walls boundary conditions

Considering the boundary conditions regarding the solid walls of the geometry, no specific value for the adjoint velocity and the adjoint pressure can be derived from the main equation, however, the same reasoning done with the inlet boundary conditions can be applied: given the nature of the primal flow, a no-slip boundary condition can be imposed for the adjoint velocity, i.e. $\mathbf{u}_a = 0$, and for the adjoint pressure, a zero gradient boundary condition is set. Regarding the boundary condition for the adjoint temperature, the last two integrals of equation 2.55 can be considered to extract the value to be set. In particular, considering the last integral, it can be manipulated in the following way:

$$\int_{\Gamma_{wall}} D_T(\hat{n} \cdot \nabla \delta T) T_a d\Gamma = \int_{\Gamma_{wall}} D_T T_a \delta(\hat{n} \cdot \nabla T) d\Gamma \quad (2.59)$$

This comes from the following equivalence:

$$\hat{n} \cdot \nabla \delta T = n_i \frac{\partial \delta T}{\partial x_i} = n_i \delta \frac{\partial T}{\partial x_i} = \delta \left(n_i \frac{\partial T}{\partial x_i} \right) = \delta(\hat{n} \cdot \nabla T) \quad (2.60)$$

The integral of equation 2.59 is equal to zero when a fixed heat flux is present on the wall since the term $\delta(\hat{n} \cdot \nabla T)$ is null. Considering now the first integral of equation 2.55:

$$\int_{\Gamma_{wall}} (T_a(\mathbf{v} \cdot \hat{n}) + D_T(\nabla T_a \cdot \hat{n}) + \frac{\partial J_\Gamma}{\partial T}) \delta T d\Gamma \quad (2.61)$$

the first term is null since on a wall the flux of the primal velocity is zero ($\mathbf{v} \cdot \hat{n} = 0$), so the remaining boundary condition is:

$$D_T(\nabla T_a \cdot \hat{n}) + \frac{\partial J_\Gamma}{\partial T} = 0 \quad (2.62)$$

that, if $\frac{\partial J_\Gamma}{\partial T} = 0$, reduces to a zero gradient boundary condition for the adjoint temperature T_a .

Applying the same reasoning to the boundary condition equation regarding the adjoint temperature of the solid phase to the first part of the integral, it reads:

$$D_T T_{a_S} \delta(\hat{n} \cdot T_S) = 0 \quad (2.63)$$

and the second part of the integral gives:

$$-\nabla T_{a_S} \cdot \hat{n} D_T + \frac{\partial J_\Gamma}{\partial T_S} = 0 \quad (2.64)$$

that reduces to a zeroGradient boundary condition for T_{a_S}

2.3.3. Fixed temperature walls boundary conditions

For a fixed temperature boundary condition, the expressions for p_a and \mathbf{u}_a remain unchanged, however, the zeroGradient boundary condition on T_a doesn't hold anymore. Given the fixed value of T, the expression:

$$D_T(\nabla T_a \cdot \hat{n}) \quad (2.65)$$

becomes null, whereas the expression:

$$D_T(\hat{n} \cdot \nabla \delta T) T_a d\Gamma = 0 \quad (2.66)$$

requires to fix T_a to zero. The same reasoning can be applied to T_{a_S}

2.3.4. Outlet patch boundary conditions

Considering now the outlet patch, in general p is set as a fixed value, commonly zero for incompressible flows, so δp is null, and the first integral of 2.55 is cancelled out. Furthermore, common values for the primal velocity and primal temperature boundary conditions are zero gradients. To extract the boundary condition for the adjoint fields, it is needed to decompose the terms regarding the viscosity in equation 2.55:

$$\begin{aligned}
& \int_{\Gamma_{outlet}} (\nu ((\nabla \mathbf{u}_a + (\nabla \mathbf{u}_a)^T) \hat{n}) \cdot \delta \mathbf{v} - (\nu (\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v})^T) \hat{n}) \cdot \mathbf{u}_a) d\Gamma = \\
& = \int_{\Gamma_{outlet}} \nu ((\hat{n} \cdot \nabla) \mathbf{u}_a \cdot \delta \mathbf{v} - (\hat{n} \cdot \nabla) \delta \mathbf{v} \cdot \mathbf{u}_a) d\Gamma + \\
& - \int_{\Gamma_{outlet}} \nabla \nu \cdot ((\mathbf{u}_a \cdot \hat{n}) \delta \mathbf{v} - (\delta \mathbf{v} \cdot \hat{n}) \mathbf{u}_a) d\Gamma
\end{aligned} \tag{2.67}$$

Here the last integral is simplified since arises the problem of flow blockage at the outlet during the simulation. Actually, this is an approximation just for the turbulent cases: in laminar problems, it is true that $\nabla \nu = 0$. As in the case treated before, adopting Einstein notation allows a better explanation for the decomposition of the previous equation:

$$\begin{aligned}
& \int_{\Gamma_{outlet}} (\nu ((\nabla \mathbf{u}_a + (\nabla \mathbf{u}_a)^T) \hat{n}) \cdot \delta \mathbf{v} - (\nu (\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v})^T) \hat{n}) \cdot \mathbf{u}_a) d\Gamma = \\
& = \int_{\Gamma} (\nu ((\nabla \mathbf{u}_a + (\nabla \mathbf{u}_a)^T) \hat{n}) \cdot \delta \mathbf{v} - (\nu (\nabla \delta \mathbf{v} + (\nabla \delta \mathbf{v})^T) \hat{n}) \cdot \mathbf{u}_a) d\Gamma = \\
& = \int_{\Gamma} \left(\left(\nu \left(\frac{\partial u_{a_i}}{\partial x_j} + \frac{\partial u_{a_j}}{\partial x_i} \right) n_j \right) \delta v_i - \left(\nu \left(\frac{\partial \delta v_i}{\partial x_j} + \frac{\partial \delta v_j}{\partial x_i} \right) n_j \right) u_{a_i} \right) d\Gamma = \\
& = \int_{\Gamma} \nu n_j \left(\frac{\partial u_{a_i}}{\partial x_j} \delta v_i + \frac{\partial u_{a_j}}{\partial x_i} \delta v_i - \frac{\partial \delta v_i}{\partial x_j} u_{a_i} - \frac{\partial \delta v_j}{\partial x_i} u_{a_i} \right) = \\
& = \int_{\Gamma} \nu n_j \left(\frac{\partial u_{a_i}}{\partial x_j} \delta v_i - \frac{\partial \delta v_i}{\partial x_j} u_{a_i} \right) d\Gamma + \\
& + \int_{\Gamma} \nu n_j \left(\frac{\partial u_{a_j}}{\partial x_i} \delta v_i - \frac{\partial u_{a_j}}{\partial x_i} \delta v_i - \frac{\partial \delta v_j}{\partial x_i} u_{a_i} \right) d\Gamma = \\
& = \int_{\Gamma} \nu n_j \left(\frac{\partial u_{a_i}}{\partial x_j} \delta v_i - \frac{\partial \delta v_i}{\partial x_j} u_{a_i} \right) d\Gamma + \int_{\Gamma} \frac{\partial}{\partial x_i} (\nu n_j u_{a_j} \delta v_i - \nu \delta v_j u_{a_i}) d\Gamma + \\
& - \int_{\Gamma} \left(\frac{\partial \nu}{\partial x_i} n_j u_{a_j} \delta v_i - \frac{\partial \nu}{\partial x_i} n_j \delta v_j u_{a_i} \right) d\Gamma = \tag{2.68} \\
& = \int_{\Gamma} \nu n_j \left(\frac{\partial u_{a_i}}{\partial x_j} \delta v_i - \frac{\partial \delta v_i}{\partial x_j} u_{a_i} \right) d\Gamma + \int_{\Omega} \frac{\partial^2}{\partial x_i \partial x_j} (\nu u_{a_j} \delta v_i - \nu \delta v_j u_{a_i}) d\Omega + \\
& - \int_{\Gamma} \frac{\partial \nu}{\partial x_i} (n_j u_{a_j} \delta v_i - n_j \delta v_j u_{a_i}) d\Gamma = \\
& = \int_{\Gamma} \nu ((\hat{n} \cdot \nabla) \mathbf{u}_a \cdot \delta \mathbf{v} - (\hat{n} \cdot \nabla) \delta \mathbf{v} \cdot \mathbf{u}_a) d\Gamma + \int_{\Omega} \frac{\partial^2 \nu}{\partial x_i \partial x_j} (u_{a_j} \delta v_i - \delta v_j u_{a_i}) d\Omega + \\
& - \int_{\Gamma} \frac{\partial \nu}{\partial x_i} (n_j u_{a_j} \delta v_i - n_j \delta v_j u_{a_i}) d\Gamma = \\
& = \int_{\Gamma} \nu ((\hat{n} \cdot \nabla) \mathbf{u}_a \cdot \delta \mathbf{v} - (\hat{n} \cdot \nabla) \delta \mathbf{v} \cdot \mathbf{u}_a) d\Gamma + \\
& - \int_{\Gamma} \nabla \nu \cdot ((\mathbf{u}_a \cdot \hat{n}) \delta \mathbf{v} - (\delta \mathbf{v} \cdot \hat{n}) \mathbf{u}_a) d\Gamma = \\
& = \int_{\Gamma_{outlet}} \nu ((\hat{n} \cdot \nabla) \mathbf{u}_a \cdot \delta \mathbf{v} - (\hat{n} \cdot \nabla) \delta \mathbf{v} \cdot \mathbf{u}_a) d\Gamma + \\
& - \int_{\Gamma_{outlet}} \nabla \nu \cdot ((\mathbf{u}_a \cdot \hat{n}) \delta \mathbf{v} - (\delta \mathbf{v} \cdot \hat{n}) \mathbf{u}_a) d\Gamma
\end{aligned}$$

Substituting this decomposition for the terms in 2.55 multiplied by \mathbf{u}_a and $\delta \mathbf{v}$ and neglecting the terms that are null due to the boundary conditions imposed on the primal flow, it yields:

$$\begin{aligned} & \int_{\Gamma_{outlet}} \left(\hat{n}(\mathbf{u}_a \cdot \mathbf{v}) + \mathbf{u}_a(\mathbf{v} \cdot \hat{n}) - p_a \hat{n} + \nu(\hat{n} \cdot \nabla) \mathbf{u}_a + T_a \hat{n} T + \frac{\partial J_\Gamma}{\partial \mathbf{v}} \right) \cdot \delta \mathbf{v} d\Gamma + \\ & + \int_{\Gamma_{outlet}} \left(T_a(\mathbf{v} \cdot \hat{n}) + D_T(\hat{n} \cdot \nabla T_a) + \frac{\partial J_T}{\partial T} \right) \delta T d\Gamma = 0 \end{aligned} \quad (2.69)$$

that brings the following equations:

$$\hat{n}(\mathbf{u}_a \cdot \mathbf{v}) + \mathbf{u}_a(\mathbf{v} \cdot \hat{n}) - p_a \hat{n} + \nu(\hat{n} \cdot \nabla) \mathbf{u}_a + T_a \hat{n} T + \frac{\partial J_\Gamma}{\partial \mathbf{v}} = 0 \quad (2.70)$$

$$T_a(\mathbf{v} \cdot \hat{n}) + D_T(\hat{n} \cdot \nabla T_a) + \frac{\partial J_T}{\partial T} = 0 \quad (2.71)$$

Equation 2.71 gives the boundary condition that has to be imposed at the outlet for the temperature that, after having applied the definition of the objective function, becomes:

$$T_a(\mathbf{v} \cdot \hat{n}) + D_T(\hat{n} \cdot \nabla T_a) + \omega_2 \rho c_p v_n = 0 \quad (2.72)$$

Equation 2.70, instead, can be projected in the normal and tangential component with respect to the outlet:

$$\mathbf{u}_a \cdot \mathbf{v} + u_{a_n} v_n - p_a + \nu(\hat{n} \cdot \nabla) u_{a_n} + T_a T + \frac{\partial J_\Gamma}{\partial \mathbf{v}} \cdot \hat{n} = 0 \quad (2.73)$$

$$v_n \mathbf{u}_{a_t} + \nu(\hat{n} \cdot \nabla) \mathbf{u}_{a_t} + \frac{\partial J_\Gamma}{\partial \mathbf{v}} \cdot \hat{t} = 0 \quad (2.74)$$

Equation 2.73 provides a boundary condition for the adjoint pressure:

$$p_a = \mathbf{u} \cdot \mathbf{v} + u_{a_n} v_n + \nu(\hat{n} \cdot \nabla) u_{a_n} + T_a T + \frac{\partial J_\Gamma}{\partial \mathbf{v}} \cdot \hat{n} \quad (2.75)$$

that, applying the definition of objective function becomes:

$$\mathbf{u}_a \cdot \mathbf{v} + u_{a_n} v_n - p_a + \nu(\hat{n} \cdot \nabla) u_{a_n} + T_a T - \omega_1 v_n^2 - \omega_1 \left(p + \frac{v^2}{2} \right) + \omega_2 (\rho c_p T) = 0 \quad (2.76)$$

Equation 2.74, instead, provides a boundary condition for the tangential component of the adjoint velocity. Again, the definition of the objective function can be applied, resulting in:

$$v_n \mathbf{u}_{at} + \nu(\hat{n} \cdot \nabla) \mathbf{u}_{at} - \omega_1 \mathbf{v}_t v_n = 0 \quad (2.77)$$

The condition on the normal component is given by imposing the compliance with the adjoint continuity equation, considering that no sensitivity contributions are given from the interior domain ($\frac{\partial J_\Omega}{\partial p} = 0$):

$$\nabla \cdot \mathbf{u}_a = (\hat{n} \cdot \nabla) u_{an} + \nabla_{//} \cdot \mathbf{u}_{at} = 0 \quad (2.78)$$

that gives:

$$(\hat{n} \cdot \nabla) u_{an} = -\nabla_{//} \cdot \mathbf{u}_{at} \quad (2.79)$$

with the operator $\nabla_{//}$ that indicates the partial derivative in the tangential direction, i.e. $\nabla_{//} = \frac{\partial}{\partial t}$.

3 | Description of the solver

adjointMultiRegionFoam is divided into three main parts: the declaration of the common Headers needed to start the simulation, the resolution of the fluid regions and the solution of the solid regions; at the end of this chapter, in figure 3.1, a diagram shows the structure of the solver. The solver takes into account the fluid regions by means of a for cycle: during the *i*-th cycle, the *i*-th fluid region is solved and, eventually, the update for the design variable η is performed. At the beginning of this cycle each variable field such as velocity, temperature or pressure is stored in a vector or scalar field common to all regions so that the *i*-th field value corresponds to the *i*-th region. By means of "setRegionFluidFields.H", the value in the *i*-th position of the multi-region vector or scalar field is stored inside a single region vector or scalar field variable. For instance, if the field UFluid is the multi-region field containing the velocities of each fluid region, in the *i*-th cycle a variable U is defined such that it contains the values UFluid[i], so the entry in setRegionFluidFields reads:

```
volVectorField& U = UFluid[i];
```

This declaration is done for each and every variable that the solver needs to perform its operations, so from this part of the solver each variable belongs to region *i*, even if the index [i] is not present. The solver proceeds with "solveFluid.H". Here the solution of the equations related to the primal and adjoint problem is executed, and eventually, the design variable update is performed. The solver here has the same structure of a SIMPLE algorithm for the part that regards the fluid dynamic, i.e. it is composed of a predictor equation described in "UEqn.H" and by a corrector equation "pEqn.H". The predictor equation is written as follows:

$$\begin{aligned} & \text{fvm::div}(\text{phi}, U) \\ & + \text{turb.divDevSigma}(U) \\ & + \text{fvm::Sp}(\text{alpha}, U) \\ & == \\ & \text{fvModels.source}(U) \end{aligned}$$

Note that the third term is generally not present in a classical SIMPLE algorithm equa-

tion: it represents the penalization contribution of the velocity given by the porous (and eventually solid) cells. The "pEqn.H" file, instead, perfectly resembles the one employed by the SIMPLE algorithm, with the Laplacian equation:

$$\text{fvm::laplacian}(\text{rAtU}(), p) == \text{fvc::div}(\text{phiHbyA})$$

And the momentum corrector equation:

$$U = \text{HbyA} - \text{rAtU}() * \text{fvc::grad}(p)$$

Once the predictor-corrector steps are completed, the primal temperature equation is solved. As stated in chapter 2, it has the structure of a scalar transport equation:

$$\begin{aligned} & \text{fvm::div}(\text{phi}, T) \\ & - \text{fvm::laplacian}(\text{materialInterpolationMethod.DT}(), T) \\ & == \\ & \text{fvModels.source}(T) \end{aligned}$$

Note that inside the Laplacian operator it is present the entry `materialInterpolationMethod.DT()`, which represents the interpolated thermal diffusivity; later in the chapter it will be more clear why an interpolation is needed. Once the primal flow problem is solved, *adjointMultiRegionFoam* proceeds with the resolution of the adjoint problem, starting with the solution of the adjoint temperature equation:

$$\begin{aligned} & \text{fvm::div}(-\text{phi}, \text{Ta}) \\ & - \text{fvm::laplacian}(\text{materialInterpolationMethod.DT}(), \text{Ta}) \\ & == \\ & \text{fvModels.source}(\text{Ta}) \end{aligned}$$

This equation needs to be solved first since, as can be seen in equation 2.53, it does not contain any adjoint variable (i.e. p_a and \mathbf{u}_a) except for the adjoint temperature itself; on the other side, to solve the adjoint velocity and adjoint pressure equation, the adjoint temperature is needed, as can be seen in equation 2.51 and equation 2.52. After that, the adjoint predictor step is performed in "UaEqn.H":

$$\begin{aligned} & \text{fvm::div}(-\text{phi}, \text{Ua}) \\ & - \text{adjointTransposeConvection} \\ & - \text{adjointThermalGradient} \\ & + \text{turb.divDevSigma}(\text{Ua}) \\ & + \text{fvm::Sp}(\text{alpha}, \text{Ua}) \\ & == \\ & \text{fvModels.source}(\text{Ua}) \end{aligned}$$

where `adjointTransposeConvection` and `adjointThermalGradient` are declared respectively as:

```
volVectorField adjointTransposeConvection = fvc::grad(Ua) & U;
volVectorField adjointThermalGradient = T*fvc::grad(Ta);
```

After that, the momentum corrector step is solved in "pEqn.H":

```
fvm::laplacian(rAUa, pa) == fvc::div(phiHbyAa)
```

and the adjoint velocity is updated:

```
Ua = HbyAa - rAUa*fvc::grad(pa);
```

Now both the primal flow and the adjoint flow are solved and, if a number of iterations `updateEvery` is reached, it is possible to update the design variable η , to filter it and interpolate the properties of the cells through "variablesUpdate.H". An interpolation function of the material properties is needed since just the physical properties of the solid and the fluid are provided to the solver, however during the solver execution the design variable η can assume intermediate values. Given that, an interpolating scheme of the porosity and the thermal diffusivity is mandatory to perform the optimization: the chosen model is the pseudo density RAMP (rational approximation of material properties) one, introduced again by Svanberg and Stolpe [6] in the field of structural optimization. According to this model, a generic properties P can be interpolated between two extreme values P_{max} and P_{min} as a function of the design variable η as:

$$P(\eta) = P_{solid} + (P_{fluid} - P_{solid})\eta \frac{1+q}{\eta+q} \quad (3.1)$$

That, applied to the properties of interest porosity α and thermal diffusivity D_T , it reads:

$$\alpha(\eta) = \alpha_{solid} + (\alpha_{fluid} - \alpha_{solid})\eta \frac{1+q}{\eta+q} = \alpha_{solid} - \alpha_{solid}\eta \frac{1+q}{\eta+q} \quad (3.2)$$

$$D_T(\eta) = D_{T_{solid}} + (D_{T_{fluid}} - D_{T_{solid}})\eta \frac{1+q}{\eta+q} \quad (3.3)$$

since the porosity of a fluid is null. The main feature of this interpolation method is that, once η is known, the function depends on q , and it is always a concave function; furthermore can be noted that for q that tends to infinite, a linear interpolation is recovered. It follows that, since the interpolating function is convex, the outcome of the interpolation P will more likely be close to the P_{fluid} or P_{solid} , rather than an intermediate value, and this

avoids unphysical results such as cells with intermediate porosity or intermediate thermal diffusivity. Note that the grade of concavity of the function is controlled by q , so the lower the value of q , the more concave is the function and it will be less probable to conclude the simulation with unphysical cells. This interpolation scheme is applied also to ρ and c_p . As mentioned in chapter 2, also the derivatives of α and D_T with respect to η are needed to compute the sensitivity field, and using this interpolation method they can be easily carried out:

$$\frac{\partial \alpha}{\partial \eta} = -\alpha_{max} \frac{q(1+q)}{(\eta+q)^2} \quad (3.4)$$

$$\frac{\partial D_T}{\partial \eta} = (D_{T_{fluid}} - D_{T_{solid}}) \frac{q(1+q)}{(\eta+q)^2} \quad (3.5)$$

Note that the derivatives of D_T and η become steeper as the parameter q is reduced: this behaviour is reflected in the sensitivity and causes instabilities whenever η tends to zero in a cell. Moreover, has been noted [16] that lower values of q produce stiffer optimization problems, and so this prevents the RAMP method to exploit its power, and the choice of q becomes a trade-off choice; the best practice is to start with high values of q and then lower it until the porosity field is considered satisfying by the user.

Regarding the thermal diffusivity, it is interesting to note that before the interpolation, the value is corrected with the Prandtl number and the turbulent viscosity using the expression:

$$D_{T_{fluid_{corr}}} = D_{T_{fluid}} + \frac{\nu_t}{Pr} \quad (3.6)$$

This is done because the fluid thermal diffusivity depends on turbulence, and to take this into account the above correction is implemented.

Once the interpolation of D_T and α is done and their derivatives are computed, it is possible to compute the sensitivity $\frac{dL}{d\eta}$. However, literature shows that a straightforward computation of the sensitivity field leads to checkerboard (and so purely mathematical and non-manufacturable) structures of porosity in the domain. To avoid this problem, an Helmholtz type differential equation can be used to filter the sensitivity values:

$$-r^2 \nabla^2 \left(\frac{dL}{d\eta} \right)_f + \left(\frac{dL}{d\eta} \right)_f = \left(\frac{dL}{d\eta} \right) \quad (3.7)$$

with on each boundary the Neumann boundary condition:

$$\frac{\partial \frac{dL}{d\eta}}{\partial n} = 0 \quad (3.8)$$

note that $\left(\frac{dL}{d\eta}\right)$ represent the sensitivity field before the filtering and $\left(\frac{dL}{d\eta}\right)_f$ represents the filtered sensitivity field; finally, r represents a regulating parameter of the filter.

At this point, both in the case the number of iteration *updateEvery* is reached or not, the solid regions can be solved, and the method is similar to the one seen for the fluid regions, so a for cycle is performed and for each cycle is solved the i -th solid region. As for the fluid case, "setRegionSolidFields.H" is responsible to assign the variables of the i -th region to the variables that will be used in the code. After that, the equation related to the primal solid problem is solved:

$$- \text{fvM}::\text{laplacian}(\text{DTF}, \text{T}) == \text{fvModels.source}(\text{T})$$

and the adjoint solid temperature equation is solved as well:

$$- \text{fvM}::\text{laplacian}(\text{DTF}, \text{Ta}) == \text{fvModels.source}(\text{Ta})$$

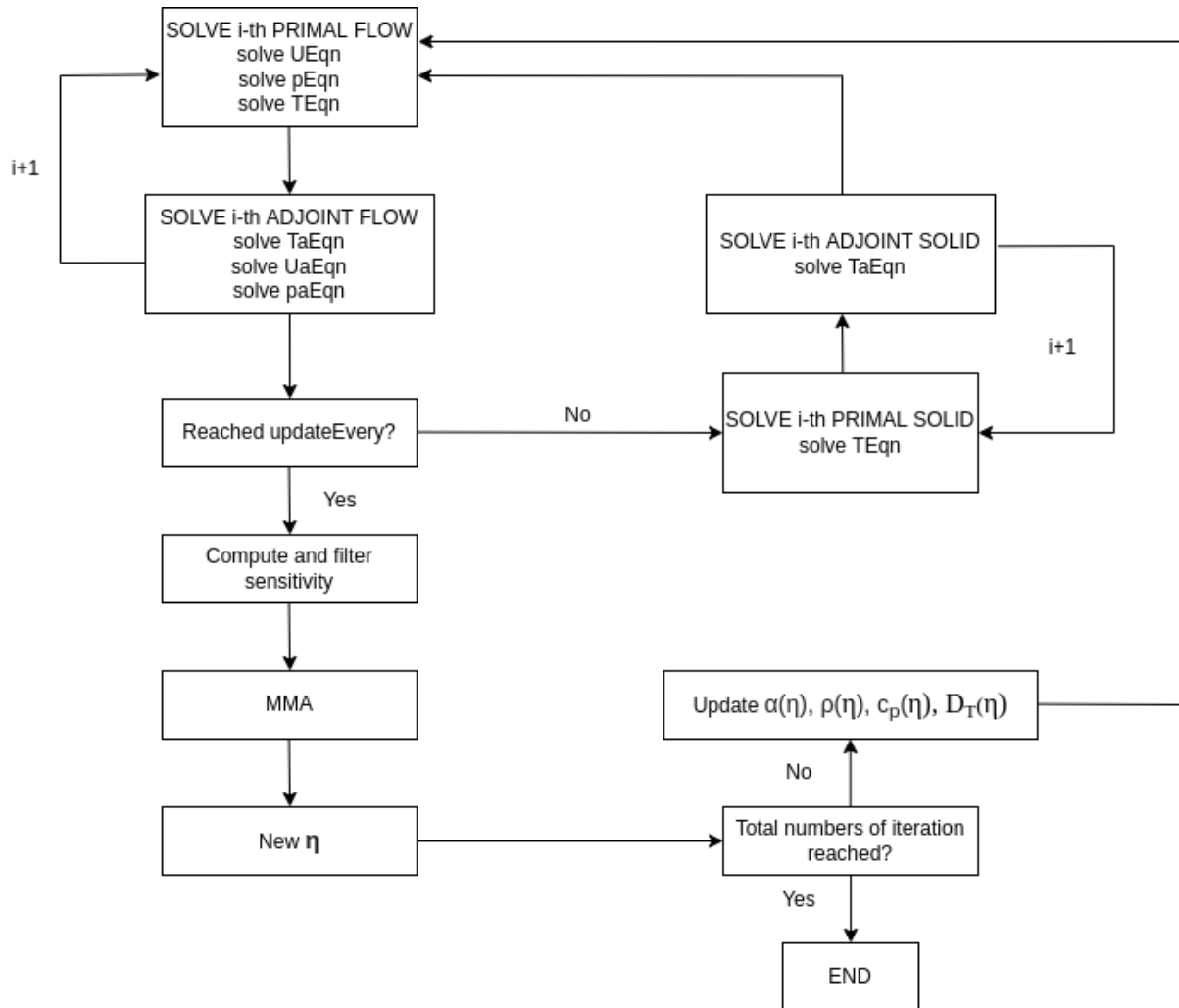


Figure 3.1: Diagram of adjointMultiRegionFoam algorithm

4 | Results comparison

4.1. Description and meshing of the geometry for the baseline adjoint simulation

The geometry that has been studied in the case at hand was gently provided by the Dynamis Team. It is a step file of a heat exchanger adopted for the cooling of the inverter of the 2021/2022 racing car. The geometry isn't complex, as required by the solver: it is composed of a circular inlet of 7 mm in diameter, that through a divergent channel becomes a rectangular duct with a variable cross-sectional area. The duct at the end of the plate makes a turn of 180° and comes back with a path parallel to the first part, and through a convergent channel, it becomes circular at the outlet, with again a diameter of 7 mm.

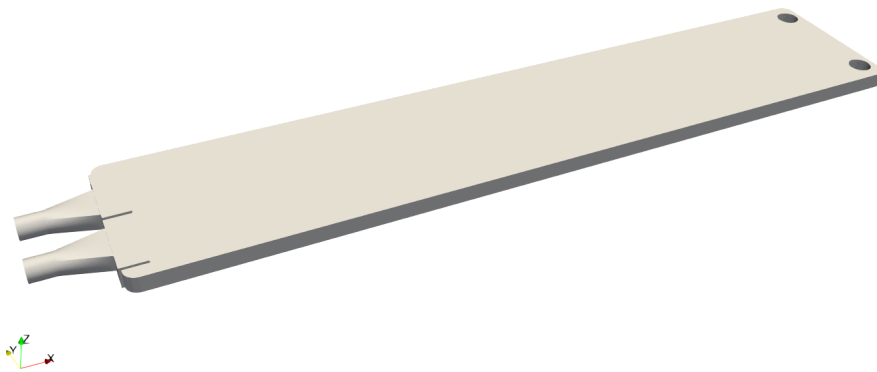


Figure 4.1: Geometry of the cooling plate provided by team Dynamis

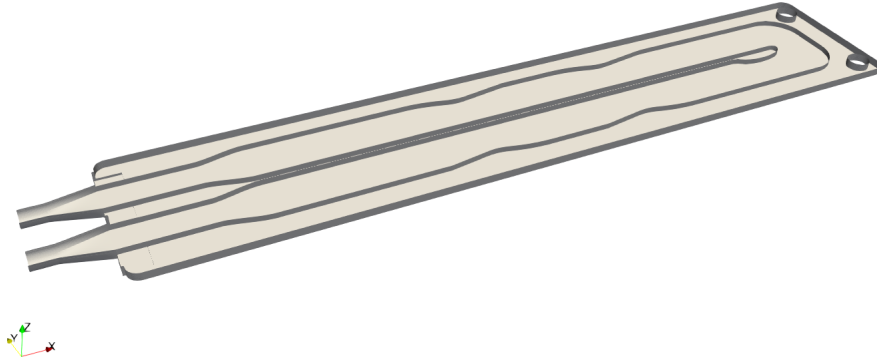


Figure 4.2: Section of the cooling plate

The step file, before meshing, has been manipulated with the software Salome to clean the geometry from irregularities inevitably generated by the CAD, convert it into an STL file, extract the surfaces defining the fluid volume, define proper patches and make the inlet and outlet longer. As it has been shown by simulations not cited in this thesis for the sake of brevity, longer inlet and outlet ducts make the adjoint solver less prone to instabilities, and so their presence is in general beneficial; this benefit is paid with longer computational time, but in general it is considered a good trade-off solution. In the case at hand, the ducts have been extended by 10 times the diameter of the duct, so by 70 mm. After that, as it is shown in figure 4.3 seven patches have been defined to make it possible to refine the computational domain in the next steps of the mesh generation. The patches are *inlet*, *outlet*, that defines the entrance and the exit of the flow; *inletTube* and *outletTube*, which defines the extended duct and the convergent and divergent ducts that shift from a circular to a rectangular section; *walls*, that defines the inside and outside sidewalls, *top* which defines the upper part of the duct, and finally *bottom*, that defines the lower part of the duct.

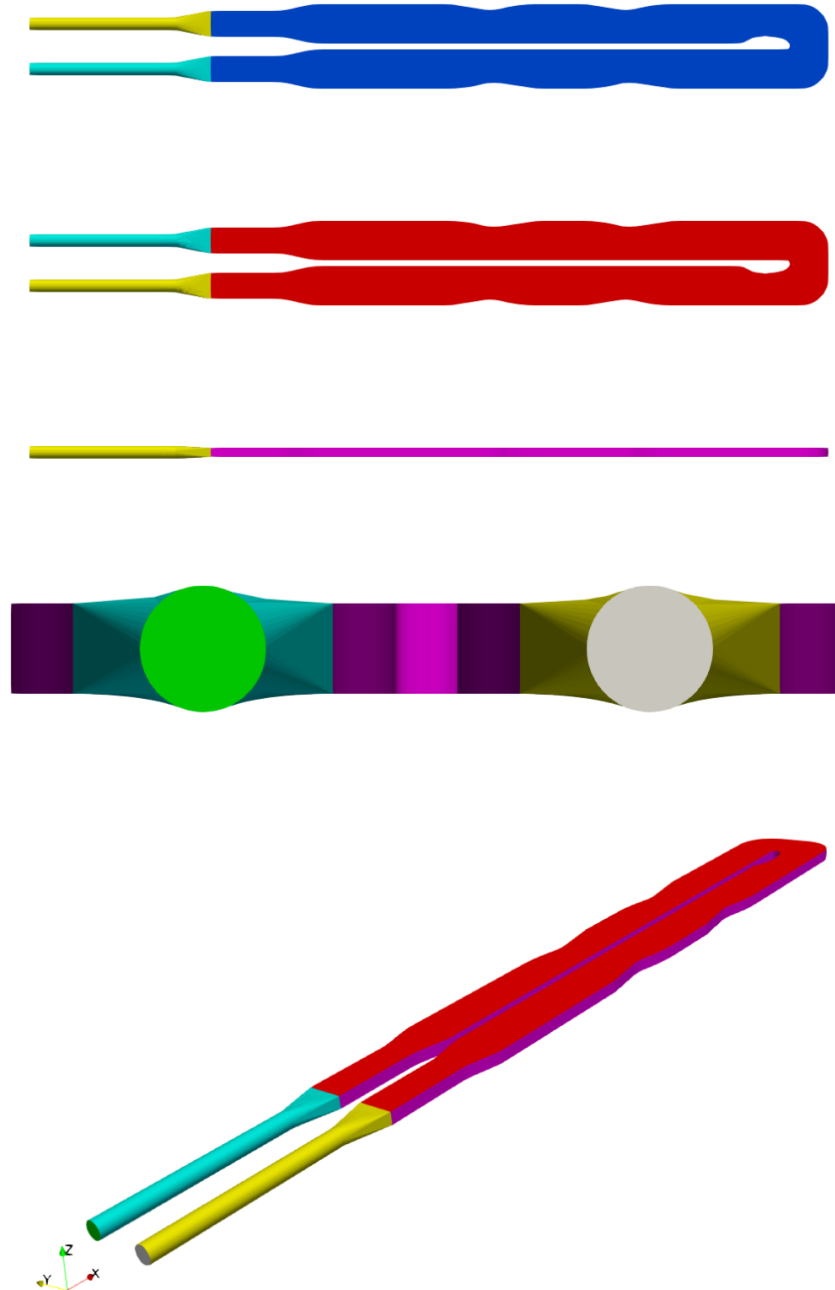


Figure 4.3: Different views of the stl file, each patch is coloured differently: green for inlet, light blue for inletTube, white for outlet, yellow for outletTube, red for top, blue for bottom and pink for walls

Once the patches were defined, a background mesh with the OpenFOAM utility `blockMesh` was generated, and then `snappyHexMesh` was used to discretize the geometry. In appendix A can be found the particular settings that have been used in `snappyHexMeshDict`, and in table 4.1 the main quality parameters of the mesh are listed, whereas in table 4.2 the percentage of layers extrusion and the thickness of the overall layers for each patch is shown.

number of cells	248969
max skewness	2.995
max non orthogonality	54.98

Table 4.1: Mesh quality parameters

	% of layers	overall thickness [m]
inletPipe	94.7	$4.22 \cdot 10^{-4}$
outletPipe	93	$4.16 \cdot 10^{-4}$
top	58.1	$3.25 \cdot 10^{-4}$
bottom	49.4	$3.06 \cdot 10^{-4}$
walls	48.1	$2.13 \cdot 10^{-4}$

Table 4.2: Layers parameters

Due to the fact that every simulation will be run on a laptop computer, the mesh has to be a compromise between the accuracy of the discretization and the total number of cells; however, it is important to note that the optimization, as well as the simulation, will be more and more precise as the number of cells grows.

After the generation of the computational domain, the patches `pins` and `pins2` were extracted generating a `faceSet` employing `topoSet` and using `createPatch` to generate the patches. These two latter are needed since they are the hot walls where the boundary condition for the temperature will be set as `fixedValue`; these two patches are shown in Figure 4.4.

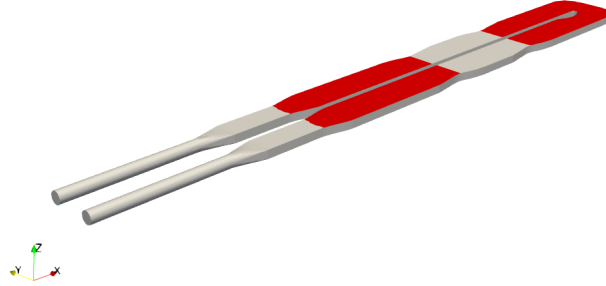


Figure 4.4: The patches pins and pins2 are highlighted in red

4.2. baseline adjoint case set up

As can be deduced from the equations presented in chapter 2, the simulation doesn't take into account any time-varying term or density variation, and so the flow is under a steady-state incompressible regime. The temperature was set at 0 at the inlet and 1 on *pins* and *pins2*; this normalization was done because it helps to stabilize the solver; if T_{min} indicates the temperature at the inlet and T_{max} indicates the temperature set on the non-adiabatic walls, the real temperature can be recovered by the simple formula:

$$T = (T_{max} - T_{min}) T_{norm} + T_{min} \quad (4.1)$$

The temperature was set as *zeroGradient* on the adiabatic walls as well as on the outlet. The pressure was set to zero at the inlet and as *zeroGradient* on the remaining patches. The velocity was set at 0.6 m/s, that with a hydraulic diameter d_h of 7 mm and a kinematic viscosity ν of water at 50° of $4.98 \cdot 10^{-7}$ gives a number of Reynolds Re equal to 8434, which is enough to consider the flow turbulent. This introduces an error in the simulation since the equations described in chapter 2 are based on the assumption of frozen turbulence, which holds for laminar flow but not for transition and turbulent flow. However, a velocity of 0.6 m/s is a very low velocity for a water heat exchanger, and performing a simulation with a velocity that would make the flow laminar would probably lead to an even worst result. In addition to that, in previous simulations, it was noticed that the solver needs a certain amount of velocity to be able to perform the porosity update of the cells, and values of velocities lower than 0.6 m/s lead to almost null changes

in the porosity distribution. This required setting a turbulence model: the chosen one was RAS k-epsilon since previous simulations showed the best reliability. Using the previously mentioned quantities and adopting the equations described in [1], first and foremost it is needed to compute the intensity:

$$I = 0.16Re^{(-\frac{1}{8})} = 5.17\% \quad (4.2)$$

and the turbulent length scale:

$$l = 0.038d_h = 2.66 \cdot 10^{-4}m \quad (4.3)$$

then, knowing that the empirical constant C_μ is equal to 0.09, it is possible to compute the turbulent kinetic energy:

$$k = \frac{3}{2}(vI)^2 = 0.00144 \frac{m^2}{s^2} \quad (4.4)$$

and the dissipation rate:

$$\epsilon = C_\mu \frac{k^{(\frac{3}{2})}}{l} = 0.0185 \frac{m^2}{s^2} \quad (4.5)$$

A summary of the primal flow boundary conditions can be seen in the following table:

	inlet	outlet	adiabatic walls	hot walls
v	(0.6,0,0)	$\frac{\partial \mathbf{v}}{\partial \hat{n}} = 0$	(0,0,0)	(0,0,0)
p	$\frac{\partial p}{\partial \hat{n}} = 0$	0	$\frac{\partial p}{\partial \hat{n}} = 0$	$\frac{\partial p}{\partial \hat{n}} = 0$
T	0	$\frac{\partial T}{\partial \hat{n}} = 0$	$\frac{\partial T}{\partial \hat{n}} = 0$	1
k	0.00144	$\frac{\partial k}{\partial \hat{n}} = 0$	kqRWallFunction	kqRWallFunction
ε	0.0185	$\frac{\partial \epsilon}{\partial \hat{n}} = 0$	epsilonWallFunction	epsilonWallFunction

Table 4.3: Summary of the primal flow boundary conditions

Regarding the adjoint problem boundary conditions, they were set according to the equations discussed in 2. In tables 4.4 and 4.5 can be found a summary of that boundary conditions. Of course, thanks to the assumption of frozen turbulence, there is no need to define adjoint variables for the turbulence quantities k and ϵ . However, in literature are

present several theoretical discussions about the implementation of turbulence models in the adjoint equations, such as Spalart-Allmaras [4].

	inlet	outlet
\mathbf{v}_a	$(\omega_1 v_x, 0, 0)$	$v_n u_{a_t} - \omega_1 v_n v_t = 0; u_{a_{new}} = u_{a_{old}}$
p_a	$\frac{\partial p_a}{\partial \hat{n}} = 0$	$\mathbf{u}_a \cdot \mathbf{v} + u_{a_n} v_n + \rho_f c_{p_f} T T_a - \omega_1 \left(p + \frac{v^2}{2} \right) - \omega_1 v_n^2 + \omega_2 \rho c_p T$
T_a	$\frac{\partial T_a}{\partial \hat{n}} = 0$	$\rho_f c_{p_f} T_a v_n + k \frac{\partial T_a}{\partial \hat{n}} + \omega_2 \rho c_p v_n = 0$

Table 4.4: Summary of the adjoint flow boundary conditions for inlet and outlet

	adiabatic walls	hot walls
\mathbf{v}_a	$(0,0,0)$	$(0,0,0)$
p_a	$\frac{\partial p_a}{\partial \hat{n}} = 0$	$\frac{\partial p_a}{\partial \hat{n}} = 0$
T_a	0	$\frac{\partial T_a}{\partial \hat{n}} = 0$

Table 4.5: Summary of the adjoint flow boundary conditions for adiabatic and hot walls

As mentioned in the previous sections, the method employed to search for the optimum η is the method of moving asymptotes. In a previous version of the solver, the method employed was the steepest gradient method, however, the MMA showed better performance in terms of computational time. Both methods, however, require setting the number of iterations between the update of the variables. This number has to be set as the number of iterations that the steady solution of the primal and adjoint equations requires to reach convergence. The method to assess if the problem is converged or not is through the analysis of the objective function J , instead of the common residuals analysis. The method that has been used to determine the number of iterations *updateEvery* between each update is the following: a simulation was run with an arbitrary *updateEvery*, and the functions J_p , J_T and J were saved every 10 iterations. If the objective functions are converged in *updateEvery* iterations, it means that that number of iterations is the correct one to use during the simulation, otherwise, it needs to be increased. Eventually, if convergence is reached before *updateEvery*, the value can be reduced to speed up the simulations or, in other words, perform more updates in the same amount of iterations. The simulations run showed that a reasonable value of *updateEvery* for this simulation is 2000 iterations. Another aspect that has to be discussed is the setting of the weight of the objective functions. Recalling their formulation:

$$J = \omega_1 J_p + \omega_2 J_T \quad (4.6)$$

with

$$J_p = \int_{inlet} \left(p + \frac{v^2}{2} \right) \mathbf{v} \cdot \hat{\mathbf{n}} dS - \int_{outlet} \left(p + \frac{v^2}{2} \right) \mathbf{v} \cdot \hat{\mathbf{n}} dS \quad (4.7)$$

$$J_T = \int_{outlet} (\rho c_p T) \mathbf{v} \cdot \hat{\mathbf{n}} dS - \int_{inlet} (\rho c_p T) \mathbf{v} \cdot \hat{\mathbf{n}} dS \quad (4.8)$$

First and foremost can be noted that the two objective functions don't have the same unit of measurement, since $J_p = [\frac{m^3}{s^2}]$ and $J_T = [\frac{Kg}{s^3}]$: this mismatching of units of measurement can be easily coped using dimensional weights ω_1 and ω_2 to recover adimensional products $\omega_1 J_p$ and $\omega_2 J_T$. Secondly, can be noted that in general J_p and J_T differ consistently in terms of order of magnitude; as a matter of comparison, the simulation run with $\omega_1 = 1$ and $\omega_2 = 0$ gives, after 50'000 iterations, a J_p equal to $1.1986 \cdot 10^{-5}$, whereas the simulation run setting $\omega_1 = 0$ and $\omega_2 = -1$ gave a final J_T equal to $1.8982 \cdot 10^0$, so it is clear that the two functions have to be normalized, as it is also stated and explained in [16]. The normalization values in the code are set in constant/optimization, and they are represented by limitP and limitT; this two values divide respectively the products $\omega_1 J_p$ and $\omega_2 J_T$ when the two are summed to obtain the main objective function. A wrong setting of these two values can lead to no change in the porosity field at all with respect to the case with $\omega_1 = 1$ and $\omega_2 = 0$ or to an exaggerated change in porosity very close to the case with $\omega_1 = 0$ and $\omega_2 = -1$, resulting in an impossible geometry with flow blockages at the inlet or the outlet.

The chosen fluid for the simulation was water at a temperature of 50°C, whereas the material that constitutes the porosity field is aluminium. Table **SRPproperties** shows a summary of the thermophysical properties and other parameters employed for the simulations. These properties can be set in constant/transportProperties, except for *updateEvery*, limitT and limitP that are set in constant/optimization.

Physical quantity	Symbol	Variable name in the code	Value
Density of the fluid	ρ_{fluid}	rhoFluid	985 [kg/m ³]
Density of the solid	ρ_{solid}	rhoSolid	2710 [kg/m ³]
Fluid kinematic viscosity	ν	nu	$4.9759 \cdot 10^{(-7)} [\frac{m^2}{s}]$
Fluid specific heat	c_{pfluid}	cpFluid	$4.061 \cdot 10^3 [\frac{J}{Kg \cdot K}]$
Solid specific heat	c_{psolid}	cpSolid	$0.897 \cdot 10^3 [\frac{J}{Kg \cdot K}]$
Fluid thermal diffusivity	D_{Tfluid}	DTfluidLam	$1.62 \cdot 10^{-7} [\frac{Kg^2}{s}]$
Solid thermal diffusivity	D_{Tsolid}	DTSolid	$9.75 \cdot 10^{-5} [\frac{Kg^2}{s}]$
Iterations between MMA updates	-	updateEvery	2000
sensitivity filter regulating parameter	r	radius	0.001 [Kg]
RAMP interpolation regulating parameter	q	k	0.1
Prandtl number	Pr	Prt	3.074
maximum allowed value for porosity	α_{max}	alphaMax	100 [$\frac{1}{s}$]
normalization factor of J_P	-	limitP	1
normalization factor of J_P	-	limitT	$10^4 [\frac{m^3}{Kg}]$

Table 4.6: Summary of the single region solver parameters

4.3. Description and meshing of the geometry for the multiregion adjoint simulation

To manipulate the multi-region geometry the same workflow of the single region geometry was adopted, however, to generate a multi-region mesh certain steps require special care. Again, the CAD file was cleaned using Salome and the inlet and outlet tubes were made longer for the reason of better stability of the solver; however in this case were defined three main regions: fluid, which comprises just the inlet and the outlet, fluid_to_solid, which defines the interface between the fluid region and the solid region, and solid, which defines the outside surfaces that belong to the solid region. This latter, as it is highlighted in figure 4.6 was formed by 5 subgroups of patches to have more freedom in the mesh refinement with snappyHexMesh, namely inletTube_solid, outletTube_solid, divergent_solid, convergent_solid and solidNoDucts. After that, the Mesh utility of Salome was adopted: since the thickness of the solid region is very small in certain spots, it is needed to have a high-quality and high-resolution STL file to avoid regions overlapping. Salome Mesh allows to generate an STL composed of triangles of chosen dimension; in this case, every patch was set to have meshed with triangles with a reference dimension of 0.0005 m.

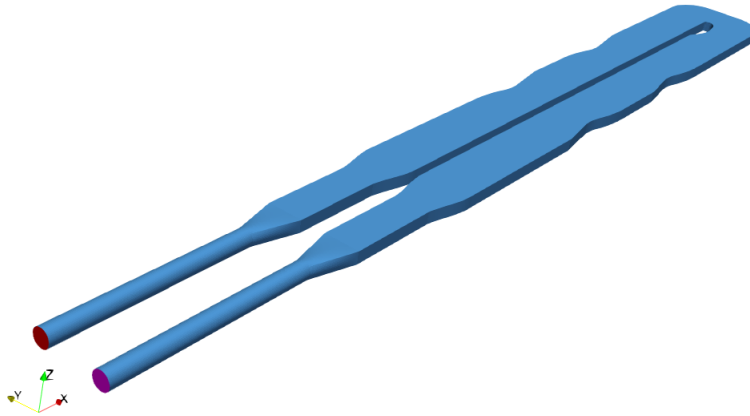


Figure 4.5: Inlet, outlet and interface patches that defines the fluid domain

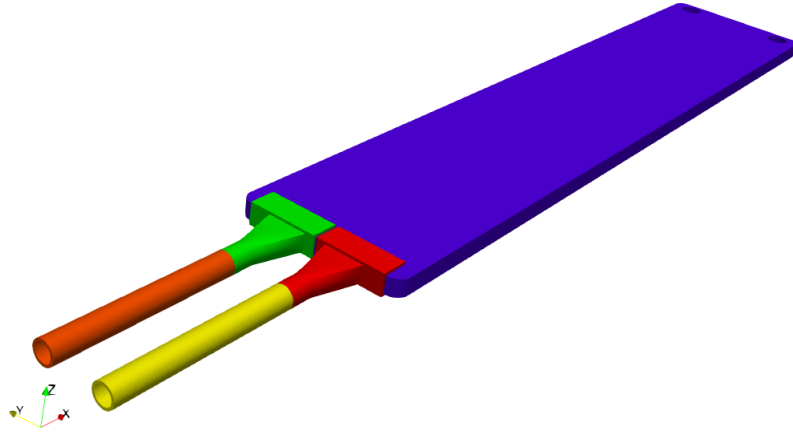


Figure 4.6: Patches defining the solid domain

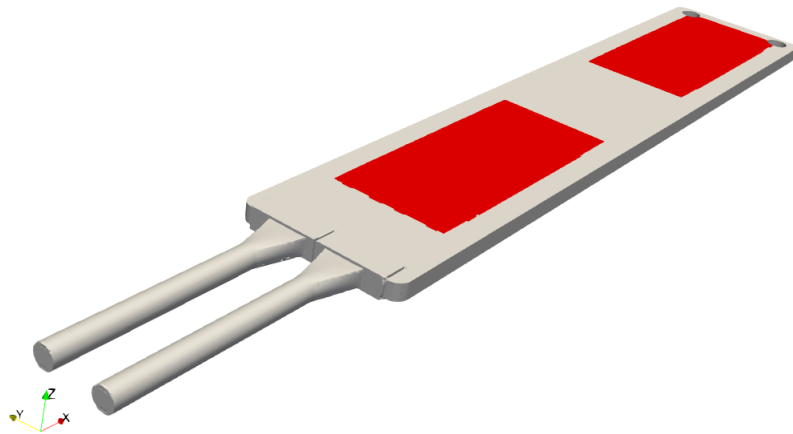


Figure 4.7: Pins and pins2 are highlighted in red

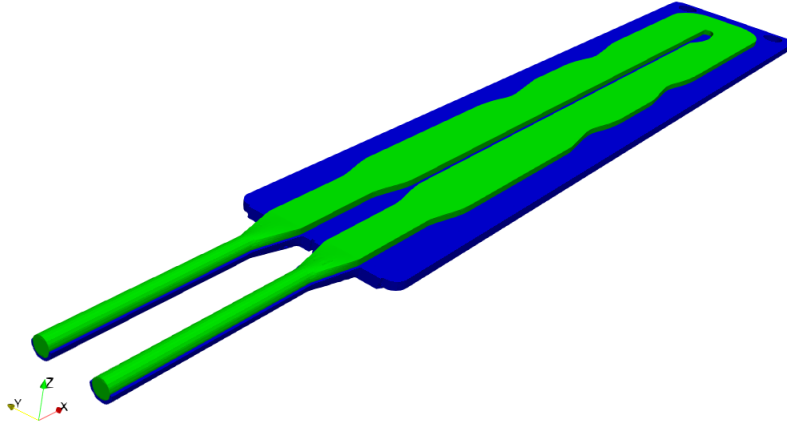


Figure 4.8: Solid and fluid domain: the solid domain has been clipped by half to better visualize the fluid domain

After that, as for the previous geometry, a background mesh was generated using `blockMesh`; this time the background mesh was assigned to the solid region. Then, once the refinement options were set in `snappyHexMeshDict` which can be consulted in Appendix B, the homonymous utility was run until the snap phase. Once this was done, the mesh is generated but it is as if it was composed just by one single monolithic block, so it is needed to run `splitMeshRegion` to divide the geometry into the regions named before, so fluid and solid. Once this is done, to generate the layers it is needed to move the content inside `constant/fluid/poliMesh` in `constant/polyMesh`, and re-run `snappyHexMesh` with just the `addLayers` sub-dictionary turned on; the result of this step is then copied inside `constant/fluid/polyMesh`. Finally, as in the previous case, the patches `pins` and `pins2`, shown in figure 4.7 are extracted to apply the temperature boundary conditions for the simulation, this time with the command `createPatch -region solid`. The quality parameters both for the fluid and for the solid regions are depicted in table 4.7, whereas the percentage of layers extruded on the `fluid_to_solid` patch is ? note that the number of cells has to be a compromise between discretization accuracy and computational time.

	fluid	solid
number of cells	254480	218761
max skewness	2.992	3.417
max non orthogonality	63.54	57.72

Table 4.7: Multiregion mesh quality parameters

4.4. Simulations results

4.4.1. Non optimized flow

As a matter of comparison, a simulation was run with the solver *thermalAdjointShapeOptimizationFoam* with the optimization turned out, i.e. setting the weight ω_1 and ω_2 to zero; performing a slice with Paraview normal to the z-axis and positioned exactly in the middle of the geometry, the velocity field and the temperature field can be visualized:

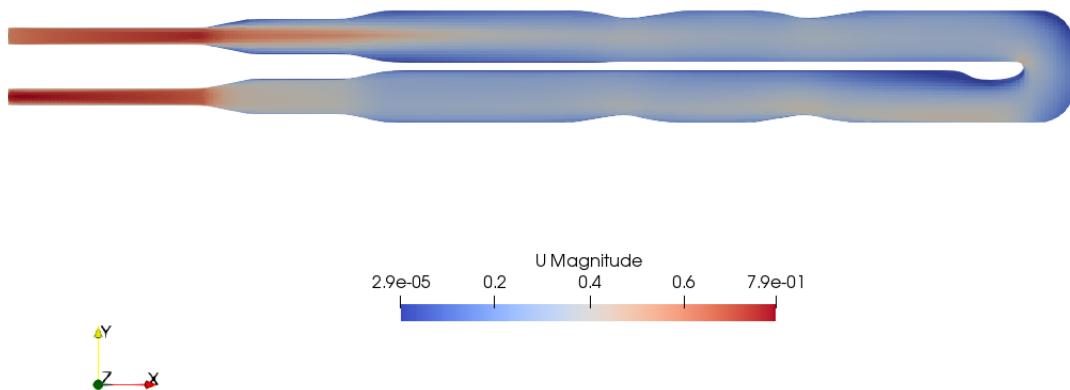


Figure 4.9: Velocity field of the non optimized simulation

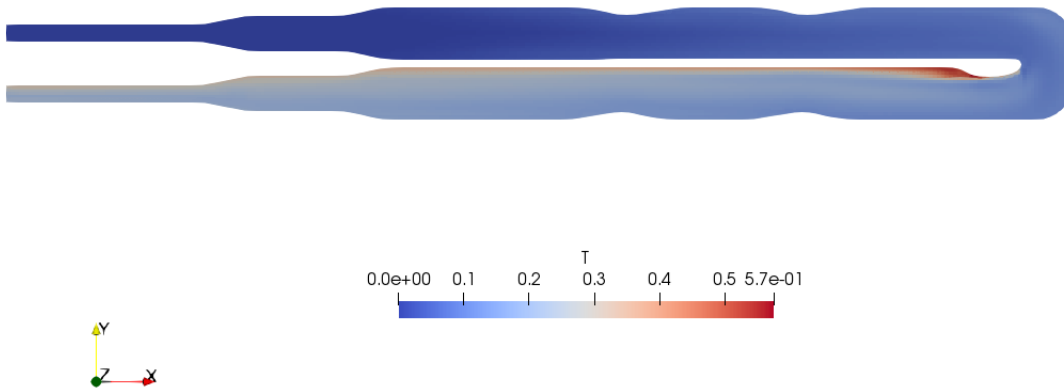


Figure 4.10: Temperature field of the non optimized simulation

4.4.2. Single region optimization

A simulation was run using the solver *thermalAdjointShapeOptimization*, with the weight set respectively as $\omega_1 = 0.5$ and $\omega_2 = -0.5$. This simulation, to perform correctly, required a value of `limitP` set to 1 and a value of `limitT` set to $1 \cdot 10^6$. The design variable is updated every 2000 iterations, and the simulation itself is run for 50000 iterations, so performing a total of 25 variables update. In figure 4.11 the velocity field can be visualized; the technique adopted is the same as the previous simulations; here can be noted that the regions close to the walls have less speed compared to the previous case, especially in the 180° turn region: this is because, in general, the solver tends to generate cells with maximum porosity in the regions close to the hot walls. In figure 4.12 can be noted a more uniform temperature field with a higher temperature at the outlet. Note also that the high-temperature region at the end of the 180° turn has almost disappeared; in figure 4.13 the design variable field can be appreciated; again this is a z-axis normal slice positioned in the middle of the domain. Remember that the design variable field is inversely proportional to the porosity field, i.e the zones where η is equal to one the domain is completely fluid, whereas the zones where η is equal to zero are completely

solid. From this picture can also be appreciated the fact that thanks to the RAMP porosity interpolation, the regions with medium values of η are almost absent. In figure 4.14 the η field is visualized applying in Paraview the filter clip to this latter variable, setting the clip value to 0.1; here can be seen that the mainly modified region is the 180° turn. Figure 4.15 shows a zoom in the 180° turn, and can be noted two main structures of solid cells; from this view can be also noted the tendency of the solver to add solid cells in the corners of the turn. Lastly, in figure 4.16 is shown the history of the objective function during the iteration: this latter is used to check for the convergence of the simulation.

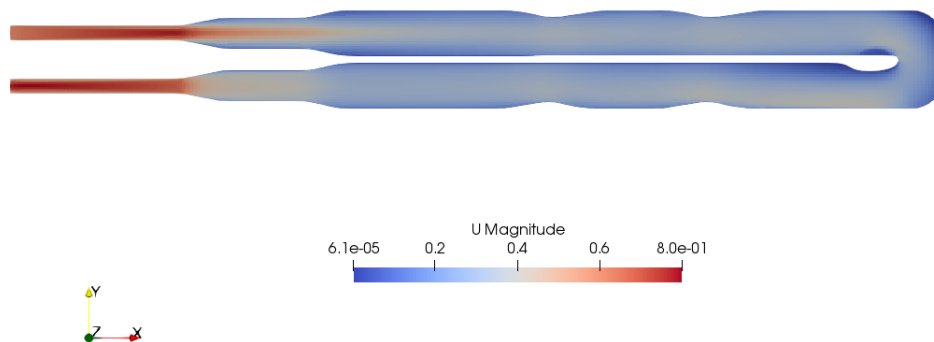


Figure 4.11: Velocity field of the single region optimized simulation

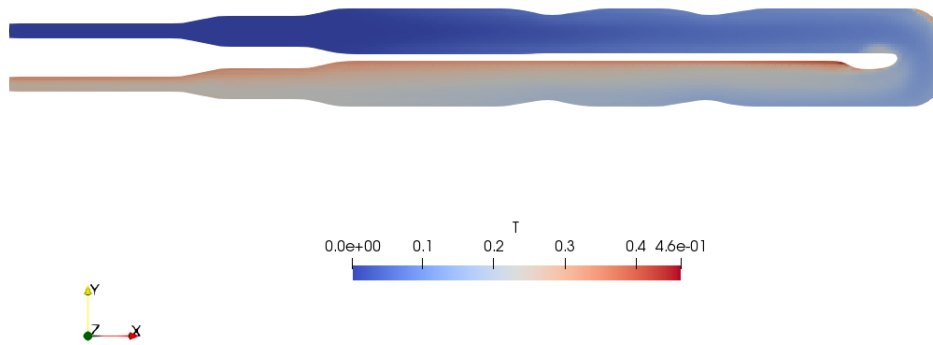


Figure 4.12: Temperature field of the single region optimized simulation

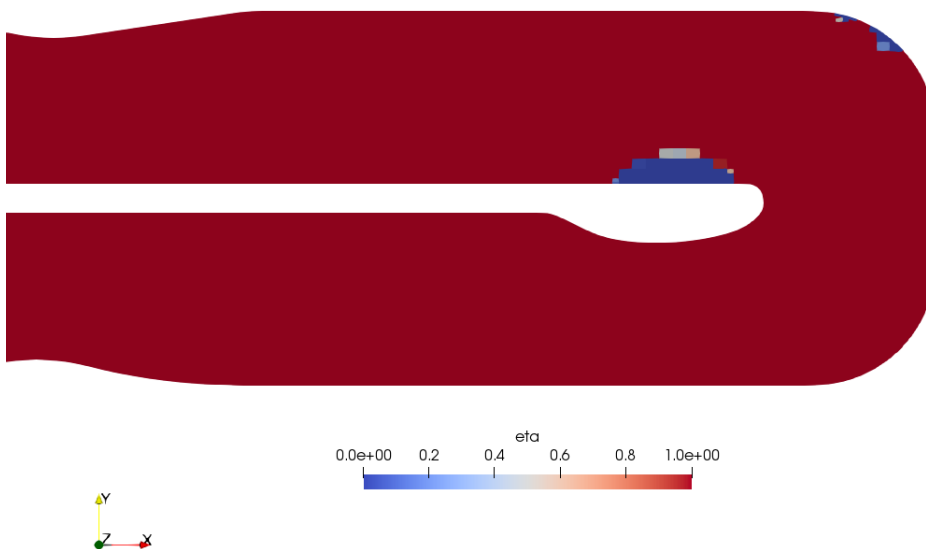


Figure 4.13: Slice of η field of the single region optimized simulation

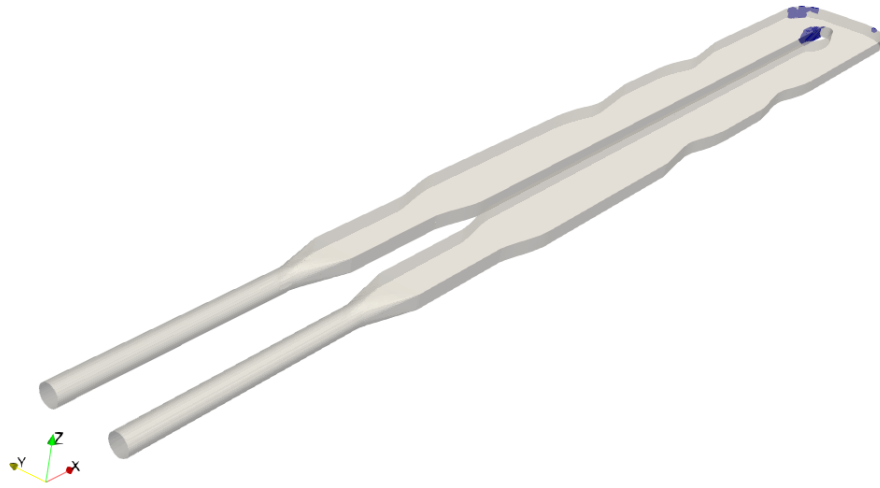


Figure 4.14: Eta field of the single region optimized simulation

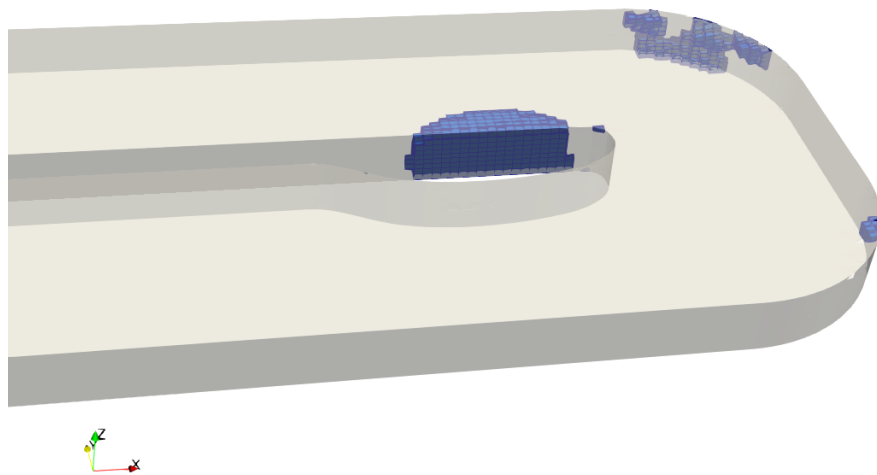


Figure 4.15: Eta field of the single region optimized simulation, particular in the pins2 patch region

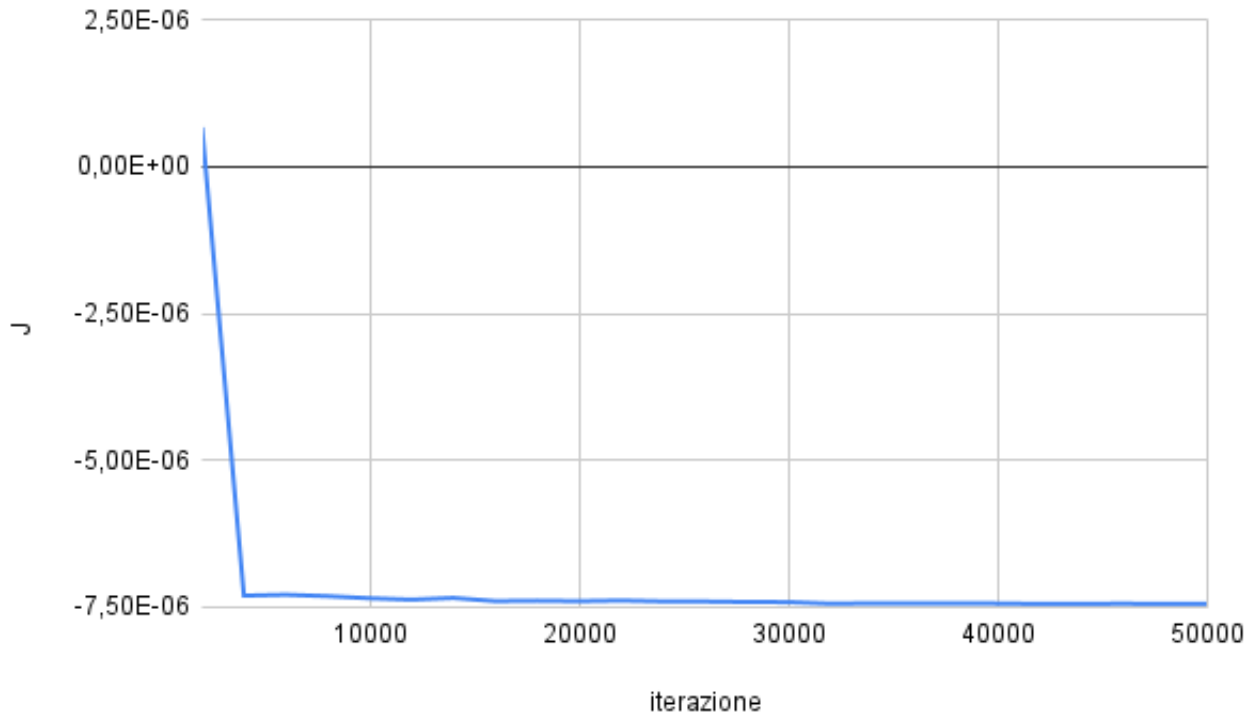


Figure 4.16: Objective function of the single region optimization

4.4.3. Multiregion optimization

For the multi-region simulation, the same objective functions weights and physical quantities of the previous case were adopted; however, the value of `limitT` was changed to $1 \cdot 10^4$. This value has been chosen since it was the best value for the convergence of the objective function; it follows that the same setting can't be used both for the single region and multi-region case, and each simulation has to be studied as a stand-alone case. Also, the value of `updateEvery` has been changed from 2000 to 4000: as test simulations have shown, the objective function takes more time to converge from one update to the next, this can be due to the fact that an additional Laplacian equation for adjoint temperature in the solid region is added in this approach. So also the total number of iterations was increased to 160000, leading to a total of 40 design variable updates. In figure 4.17 can be seen the velocity field in a slice of the fluid and solid fields, generated with the same positioning of the previous case, whereas in figure 4.18 the temperature field is depicted. In figure 4.19 a slice of η field is shown, and the first difference with the precedent solver can be appreciated: in this case, the solver doesn't generate high porosity zones before the turn, it rather prefer to generate them after the duct. In addition to that, the solver generates in the upper corner a solid cells agglomerate to split the duct in two. Finally,

in figure 4.22, the convergence of the objective function is shown.

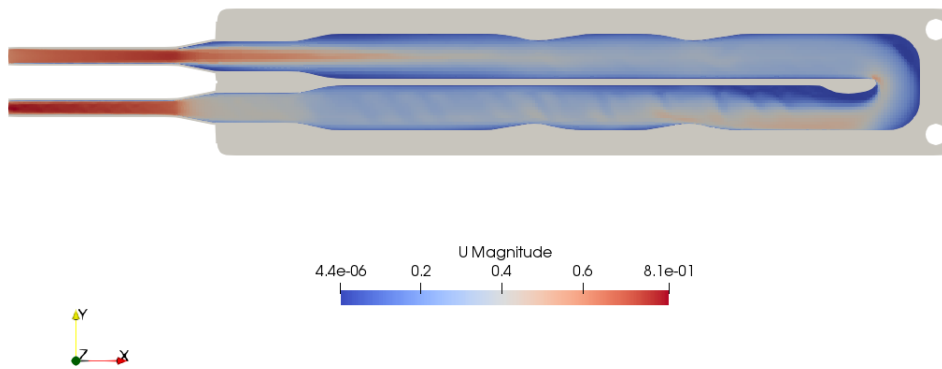


Figure 4.17: Velocity field of the multiregion simulation

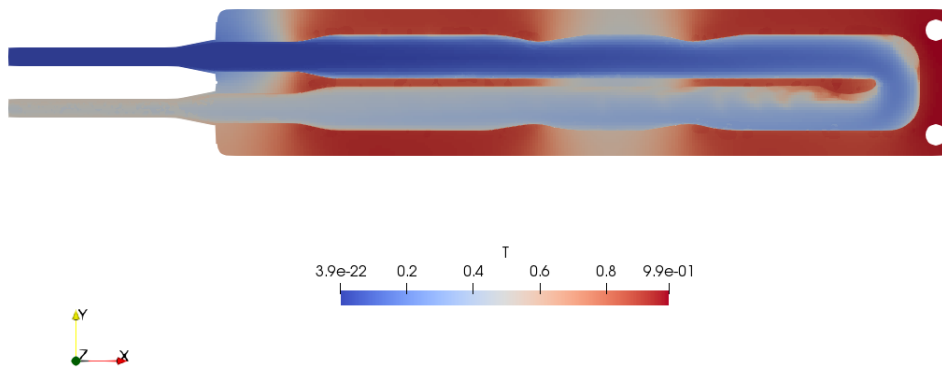


Figure 4.18: Temperature field of the multiregion simulation

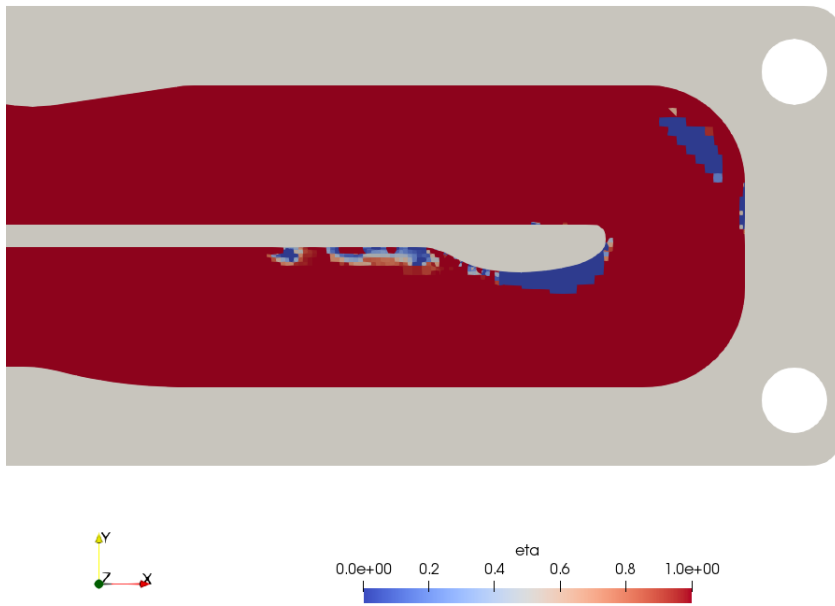


Figure 4.19: Slice of eta field of the multiregion simulation

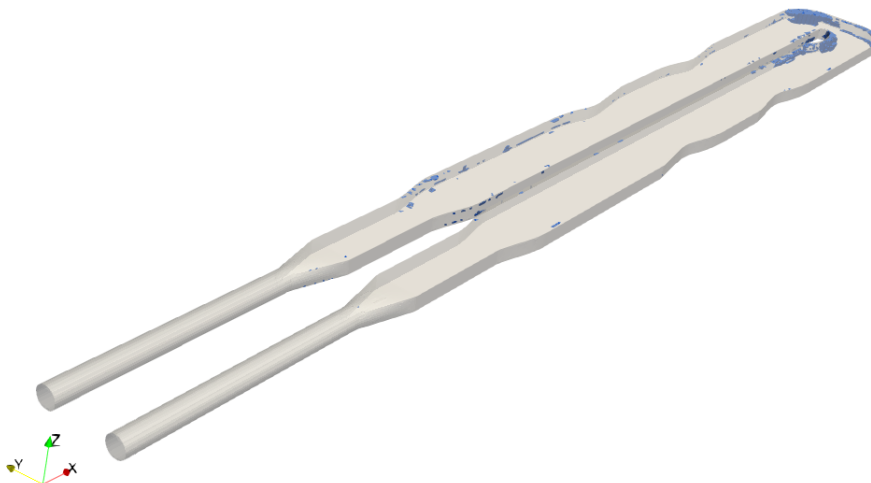


Figure 4.20: Eta field of the multiregion simulation

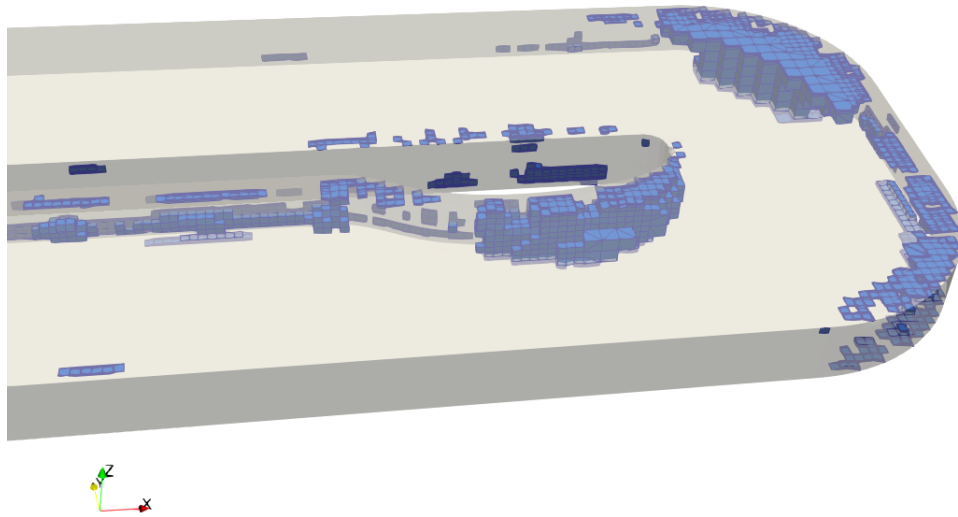


Figure 4.21: Eta field of the multiregion simulation, particular in the 180° turn region

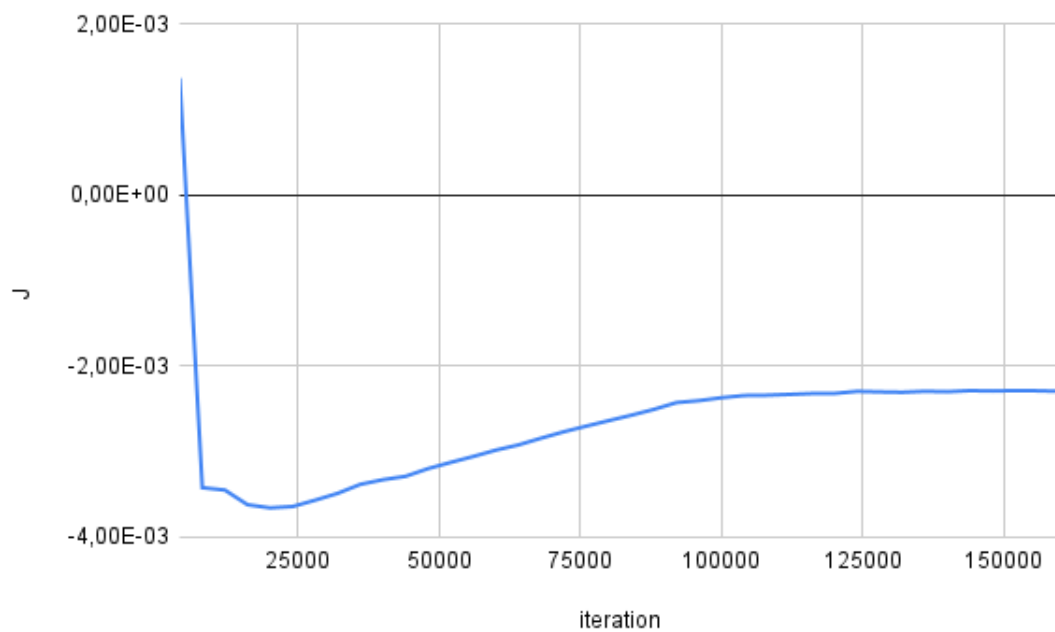


Figure 4.22: Convergence of the objective function

5 | Conclusion

This work aimed to point out the main differences between a single region and a multi-region adjoint-based optimization solver applied to a real case geometry. In general, the two solvers seem to focus on the optimization in the same region, however, the resulting porosity fields are quite different; this is probably due to the different nature of the problem: in the single region case, the temperature boundary conditions are set directly on the boundary of the fluid domain, whereas in the multi-region optimization the temperature boundary conditions in the fluid domain are a result of the solution of the Laplace equation inside the solid region. Another stunning difference is the speed of convergence of the solver: the single region solver takes 50'000 iterations to reach convergence, which corresponds to 13 hours on a 6-cores laptop, whereas the multi-region solver takes 160'000 iterations, which are translated into 48 hours of computational time. It follows that the first solver is for sure the better option in a preliminary analysis, and the second one is the better choice if a more in-depth analysis is looked for. Given the fact that no experimental activities were conducted, it is not possible to assess for sure which porosity field is the best one; furthermore, an experimental activity would require manufacturing the heat exchanger, which is not a banal task, given the fact that the output of the simulation is not an STL file that could be sliced by a 3D printing slicer software and 3D printed; design the optimized heat exchanger with traditional method by trying to manually copy the results visualized in Paraview in a CAD software would probably lead to a coarse approximation of the optimized geometry. For this reason, one of the main improvements that can be added to the multi-region simulation would be a software that, given the optimized geometry, translates it into an STL file ready to be 3D printed and tested.

Bibliography

- [1] Turbulence free stream boundary conditions. URL https://www.cfd-online.com/Wiki/Turbulence_free-stream_boundary_conditions.
- [2] Openfoam wiki, chtmultiregionfoam, 2019. URL <https://openfoamwiki.net/index.php/ChtMultiRegionFoam>.
- [3] E. d. V. Henry G. Weller, Carsten Othmer. Implementation of a continuous adjoint for topology optimization of ducted flows. 2007. URL <https://doi.org/10.2514/6.2007-3947>.
- [4] D. P. K.C. Giannakoglou et al., A.S. Zymaris. Continuous adjoint approach to the spalart–allmaras turbulence model for incompressible flows. *Computers Fluids*, pages 1528–1538, 2009. URL <https://doi.org/10.1016/j.compfluid.2008.12.006>.
- [5] B. M. P. Kikuchi Noboru. Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering*, pages 197–224, 1988. URL [https://doi.org/10.1016/0045-7825\(88\)90086-2](https://doi.org/10.1016/0045-7825(88)90086-2).
- [6] K. S. Mathias Stolpe. An alternative interpolation scheme for minimum compliance topology optimization. *Struct Multidisc Optim* 22, pages 116–124, 2001. URL <https://doi.org/10.1007/s001580100129>.
- [7] L. Montoli. Multi-objective topology optimization for conjugate heat transfer with the adjoint method. Master’s thesis, Politecnico di Milano, 2021.
- [8] M. B. G. Niles A. Pierce. An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion* 65, pages 393–415, 2000. URL [doi:https://doi.org/10.1023/A:1011430410075](https://doi.org/10.1023/A:1011430410075).
- [9] U. Nilson. Description of adjointshapeoptimizationfoam and how to implement new objectiv functions, 2014. URL http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2013/UlfNilsson/reportAdjoint.pdf.
- [10] C. Othmer. A continuous adjoint formulation for the computation of topological and

- surface sensitivities of ducted flows. *International journal for Numerical Methods in Fluids*, pages 861–877, 2008. URL <https://doi.org/10.1002/flid.1770>.
- [11] K. C. G. E. M. Papoutsis-Kiachagias. Continuous adjoint methods for turbulent flows, applied to shape and topology optimization: Industrial applications. *Archives of Computational Methods in Engineering*, pages 255–299, 2016. URL <https://doi.org/10.1007/s11831-014-9141-9>.
- [12] K. Svanberg. Mma matlab algorithm. URL <http://www.smoptit.se/>.
- [13] K. Svanberg. The method of moving asymptotes — a new method for structural optimization. *International Journal for Numerical Methods in Engineering*, pages 359–373, 1987. URL <https://doi.org/10.1002/nme.1620240207>.
- [14] K. Svanberg. Lecture notes for the dcamm course, advanced topics in structural optimization at the royal institute of technology. *SIAM Journal on Optimization*, 1998. URL <https://doi.org/10.1137/S1052623499362822>.
- [15] K. Svanberg. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM Journal on Optimization*, pages 555–573, 2002. URL <https://doi.org/10.1137/S1052623499362822>.
- [16] J.-L. H. V. Subramaniam, T. Dbouk. Topology optimization of conjugate heat transfer systems: A competition between heat transfer enhancement and pressure drop reduction. *International Journal of Heat and Fluid Flow* 75, pages 165–184, 2019. URL <https://doi.org/10.1016/j.ijheatfluidflow.2019.01.002>.

A | Appendix A

```

/*-----* C++ *-----*/
=====
\\      / Field      | OpenFOAM: The Open Source CFD Toolbox
\\      / Operation  | Website:  https://openfoam.org
\\      / And        | Version:  dev
\\      / Manipulation |
/*-----*-----*/
FoamFile
{
    version            2;
    format            ascii;
    class            dictionary;
    object           snappyHexMeshDict;
}
// * * * * *

castellatedMesh true;

snap                true;

addLayers           true;

geometry
{
    geom
    {
        type          triSurfaceMesh;
        file          "fluid.stl";
    }
}

```

```
regions
{
  walls
  {
    name          walls;
  }
  outletPipe
  {
    name          outletPipe;
  }
  inletPipe
  {
    name          inletPipe;
  }
  outlet
  {
    name          outlet;
  }
  inlet
  {
    name          inlet;
  }
  top
  {
    name          top;
  }
  bottom
  {
    name          bottom;
  }
}
refinementBoxInlet
{
  type            searchableBox;
  min             ( 0.34 0.032 -0.0035 );
  max             ( 0.39 0.042 0.0065 );
}
```

```
    }
    refinementBox2
    {
        type            searchableBox;
        min              ( 0.512148 0.03449 0 );
        max              ( 0.536147 0.040252 0.005 );
    }
}

castellatedMeshControls
{
    maxLocalCells    1000000;
    maxGlobalCells   1500000;
    minRefinementCells 10;
    nCellsBetweenLevels 4;
    features          ( { file "fluid.eMesh" ; level 1 ; } );
    refinementSurfaces
    {
        geom
        {
            level          ( 0 0 );
            regions
            {
                inletPipe
                {
                    level          ( 1 2 );
                    patchInfo
                    {
                        type          wall;
                    }
                }
                outletPipe
                {
                    level          ( 1 2 );
                    patchInfo
                    {
```

```

        type          wall;
    }
}
inlet
{
    level            ( 1 1 );
    patchInfo
    {
        type          patch;
    }
}
outlet
{
    level            ( 1 1 );
    patchInfo
    {
        type          patch;
    }
}
walls
{
    level            ( 1 2 );
    patchInfo
    {
        type          wall;
    }
}
top
{
    level            ( 1 2 );
    patchInfo
    {
        type          wall;
    }
}
bottom
{

```



```
relativeSizes    true;
expansionRatio   1.2;
firstLayerThickness 0.26;
minThickness     0.5e-6;
layers
{
    "walls"
    {
        nSurfaceLayers 3;
    }
    "top"
    {
        nSurfaceLayers 3;
    }
    "bottom"
    {
        nSurfaceLayers 3;
    }
    "inletPipe"
    {
        nSurfaceLayers 3;
    }
    "outletPipe"
    {
        nSurfaceLayers 3;
    }
}
nGrow            0;
featureAngle     330;
nRelaxIter       5;
nSmoothSurfaceNormals 1;
nSmoothNormals   3;
nSmoothThickness 10;
maxFaceThicknessRatio 0.5;
maxThicknessToMedialRatio 0.3;
minMedianAxisAngle 90;
nBufferCellsNoExtrude 0;
```

```
        nLayerIter      50;
        nRelaxedIter    20;
    }

    meshQualityControls
    {
        maxNonOrtho      55;
        maxBoundarySkewness 20;
        maxInternalSkewness 3;
        maxConcave      80;
        minVol           1e-13;
        minTetQuality    -1;
        minArea          -1;
        minTwist         0.01;
        minDeterminant   0.001;
        minFaceWeight    0.05;
        minVolRatio      0.01;
        minTriangleTwist -1;
        nSmoothScale     4;
        errorReduction   0.75;
        relaxed
        {
            maxNonOrtho      55;
        }
    }

    writeFlags      ( scalarLevels layerSets layerFields );

    mergeTolerance  1e-06;

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


B | Appendix B

```

/*-----* C++ -*-----*/
===== |
\\ / Field | OpenFOAM: The Open Source CFD Toolbox
\\ / Operation | Website: https://openfoam.org
\\ / And | Version: dev
\\ \ Manipulation |
/*-----*/

FoamFile
{
    version          2;
    format          ascii;
    class          dictionary;
    object         snappyHexMeshDict;
}
// * * * * *

castellatedMesh false;

snap                false;

addLayers           true;

geometry
{
    fluid
    {
        type          triSurfaceMesh;
        file          "fluid.stl";
        regions
    }
}

```

```
{
    outlet
    {
        name            outlet;
    }
    inlet
    {
        name            inlet;
    }
}

fluid_to_solid
{
    type triSurfaceMesh;
    file "fluid_to_solid.stl";
}

solid
{
    type triSurfaceMesh;
    file "solid.stl";
    regions
    {
        solidNoDucts
        {
            name            solidNoDucts;
        }
        inletTube_solid
        {
            name            inletTube_solid;
        }
        outletTube_solid
        {
            name            outletTube_solid;
        }
        convergent_solid
    }
}
```

```

        {
            name                convergent_solid;
        }
    divergent_solid
    {
        name                divergent_solid;
    }
}
};

castellatedMeshControls
{
    maxLocalCells    1000000;
    maxGlobalCells   1500000;
    minRefinementCells 10;
    nCellsBetweenLevels 4;
    features        ( { file "fluid.eMesh" ; level 1 ; }
                    { file "solid.eMesh" ; level 2; }
                    { file "fluid_to_solid.eMesh" ; level 2 ; }
                    );
    refinementSurfaces
    {
        fluid
        {
            level            ( 1 2 );
            regions
            {

                inlet
                {
                    level            ( 1 2 );
                    patchInfo
                    {
                        type            patch;
                    }
                }
            }
        }
    }
}

```

```

        outlet
        {
            level          ( 1 2 );
            patchInfo
            {
                type          patch;
            }
        }
    }
fluid_to_solid
{
    level (1 2); // 11 22
    faceZone fluid_to_solid;
    cellZone fluid;
    mode          insidePoint;
    insidePoint (0.7 0.037371 0.0045);

}
solid
{
    level (1 1);
    regions
    {
        inletTube_solid
        {
            level          ( 1 1 );
            patchInfo
            {
                type          wall;
            }
        }
        outletTube_solid
        {
            level          ( 1 2 );
            patchInfo
            {

```



```
snapControls
{
    nSmoothPatch      5;
    tolerance         3;
    nSolveIter       30;
    nRelaxIter       10;
    nFeatureSnapIter 15;
    implicitFeatureSnap false;
    explicitFeatureSnap true;
    multiRegionFeatureSnap true;
}

addLayersControls
{
    relativeSizes      false;
    expansionRatio     1.2;
    firstLayerThickness 1e-4;
    minThickness       0.5e-6;
    layers
    layers
    {
        fluid_to_solid
        {
            nSurfaceLayers 3;
        }
    }
    nGrow              0;
    featureAngle       330;
    nRelaxIter         5;
    nSmoothSurfaceNormals 1;
    nSmoothNormals     3;
    nSmoothThickness   10;
    maxFaceThicknessRatio 0.5;
    maxThicknessToMedialRatio 0.3;
    minMedianAxisAngle 90;
    nBufferCellsNoExtrude 0;
```

```
        nLayerIter      50;
        nRelaxedIter    20;
    }

    meshQualityControls
    {
        maxNonOrtho      65;
        maxBoundarySkewness 20;
        maxInternalSkewness 3;
        maxConcave      80;
        minVol           1e-13;
        minTetQuality    -1;
        minArea          -1;
        minTwist         0.01;
        minDeterminant   0.001;
        minFaceWeight    0.05;
        minVolRatio      0.01;
        minTriangleTwist -1;
        nSmoothScale     4;
        errorReduction   0.75;
        relaxed
        {
            maxNonOrtho      65;
        }
    }

    writeFlags      ( scalarLevels layerSets layerFields );

    mergeTolerance  1e-06;

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


List of Figures

3.1	Diagram of adjointMultiRegionFoam algorithm	32
4.1	Geometry of the cooling plate provided by team Dynamis	33
4.2	Section of the cooling plate	34
4.3	Different views of the stl file, each patch is coloured differently: green for inlet, light blue for inletTube, white for outlet, yellow for outletTube, red for top, blue for bottom and pink for walls	35
4.4	The patches pins and pins2 are highlighted in red	37
4.5	Inlet, outlet and interface patches that defines the fluid domain	42
4.6	Patches defining the solid domain	43
4.7	Pins and pins2 are highlighted in red	43
4.8	Solid and fluid domain: the solid domain has been clipped by half to better visualize the fluid domain	44
4.9	Velocity field of the non optimized simulation	45
4.10	Temperature field of the non optimized simulation	46
4.11	Velocity field of the single region optimized simulation	47
4.12	Temperature field of the single region optimized simulation	48
4.13	Slice of eta field of the single region optimized simulation	48
4.14	Eta field of the single region optimized simulation	49
4.15	Eta field of the single region optimized simulation, particular in the pins2 patch region	49
4.16	Objective function of the single region optimization	50
4.17	Velocity field of the multiregion simulation	51
4.18	Temperature field of the multiregion simulation	51
4.19	Slice of eta field of the multiregion simulation	52
4.20	Eta field of the multiregion simulation	52
4.21	Eta field of the multiregion simulation, particular in the 180° turn region	53
4.22	Convergence of the objective function	53

List of Tables

4.1	Mesh quality parameters	36
4.2	Layers parameters	36
4.3	Summary of the primal flow boundary conditions	38
4.4	Summary of the adjoint flow boundary conditions for inlet and outlet . . .	39
4.5	Summary of the adjoint flow boundary conditions for adiabatic and hot walls	39
4.6	Summary of the single region solver parameters	41
4.7	Multiregion mesh quality parameters	45

List of Symbols

Variable	Description	SI unit
\mathbf{u}	primal flow velocity	m/s
\mathbf{u}_a	adjoint flow velocity	m/s
p	primal flow pressure	m^3/s^2
p_a	adjoint flow pressure	m^3/s^2
T	fluid temperature	K
T_S	solid temperature	K
T_a	adjoint fluid temperature	K
T_{a_S}	adjoint solid temperature	K
η	design variable	-
α	porosity	-
D_T	thermal diffusivity	m^2/s
ρ	density	kg/m^3
k	conductivity	W/mK
ν	kinematic viscosity	m^2/s
J	objective function	-
J_p	pressure objective function	m^3/s
J_T	temperature objective function	Kg/s^3
ω_1	J_p weght	-
ω_2	J_T weght	m^3/Kg

Acknowledgements

Here you might want to acknowledge someone.

