The Dissertation Committee for Ajay Vadakkepatt
certifies that this is the approved version of the following dissertation:

Topology Optimization for Thermal-Fluid Applications using an Unstructured Finite Volume Scheme

Committee:

David G. Bogard, Supervisor

Jayathi Y. Murthy, Co-Supervisor

Carolyn C. Seepersad

Nedialko B. Dimitrov

Sanjay R. Mathur

Topology Optimization for Thermal-Fluid Applications using an Unstructured Finite Volume Scheme

by

Ajay Vadakkepatt, B.E.; M.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2016

Dedicated to my Guru

Her holiness ‘Sri Sri Mata Amritanandamayi Devi’

The embodiment of infinite compassion!

Acknowledgments

I would like to first express my infinite gratitude to my spiritual master Amma (Her holiness Sri Sri Mata Amritanandamayi Devi), known as the 'Hugging Saint', for everything she has bestowed upon me. Her guidance throughout my life and PhD is inexplicable. I also thank all the Swamis (monks) of Amma's Ashram for their positive influence and guidance in my life.

My mother, father, sister, wife and family have been my constant source of support and inspiration during all stages of my life. Tears trickle down my cheeks whenever I think about the sacrifices my parents have endured throughout their life for others. I sincerely thank each and every person of my family for all their help and support.

I consider myself extremely fortunate to have been advised by my supervisors Prof. Jayathi Y. Murthy and Dr. Sanjay R. Mathur. What more can I ask than being able to stand on the shoulders of these two world-class giants of Computational Fluid Dynamics? I thank both my advisers and all other professors who have guided throughout my PhD in various ways.

Finally, I would like to sincerely express my gratitude to all my friends, lab-mates, room-mates and colleagues who have helped me throughout the course of my PhD.

Topology Optimization for Thermal-Fluid Applications using an Unstructured Finite Volume Scheme

Publication No. _____

Ajay Vadakkepatt, Ph.D.
The University of Texas at Austin, 2016

Supervisors: David G. Bogard
Jayathi Y. Murthy

Topology optimization is a method for developing optimized geometric designs that maximize a quantity of interest (QoI) subject to constraints. Unlike shape optimization, which optimizes the dimensions of a template shape, topology optimization does not start with a pre-conceived shape. Instead, the algorithm builds the geometry iteratively by placing material pixels in a specified background domain, aiming to maximize the QoI subject to a constraint on the volume of material or other constraints. The power of topology optimization lies in its ability to realize design solutions that are not initially apparent to the engineer. Topology optimization, though well established in structural applications, has not percolated to the thermal-fluids community to any great degree, and most published papers have not addressed sufficiently realistic engineering problems. However, the methodology has immense application

potential in the area of fluid flow, heat and mass transfer and other transport phenomena at all length scales. In the literature, the solution methodology used for topology optimization is based mostly on finite element methods. However, unstructured finite volume methods are frequently the numerical method of choice in the industry for those addressing thermal-fluid or other transport problems. It is essential that methods for topology optimization work well in the finite volume framework if they are to find traction in industry. Regardless of the numerical method employed for forward solution, the most popular methodology employed for topology optimization is the solid isotropic material with penalization (SIMP) approach in conjunction with a gradient-based optimization algorithm. This optimization approach requires the calculation of sensitivity derivatives of the QoI with respect to design variables through a discrete adjoint method. The Method of Moving Asymptotes (MMA) is a widely-used algorithm for topology optimization. Thus the objective of this dissertation is to build a robust framework for topology optimization for thermal-fluid problems, employing SIMP and MMA, within the framework of industry-standard finite volume schemes.

Towards realizing this goal, we first develop and demonstrate topology optimization for multidimensional steady heat conduction problems in a cell-centered unstructured finite volume framework. The fundamental methodologies for SIMP/RAMP interpolation of thermal conductivity and the basic optimization infrastructure using MMA are developed and tested in this chapter. The effect of including secondary gradients in sensitivity computations is

evaluated for typical heat conduction problems. Topologies that maximize or minimize relevant quantities of interest in heat conduction applications with and without volumetric heat generation are presented.

Industry standard finite volume codes for fluid flow are built on unstructured cell-centered formulations employing co-located pressure-velocity storage, and a sequential solution algorithm. This type of algorithm is very widely used, but poses a number of difficulties when used as the solution kernel for performing efficient gradient-based topology optimization. The complete Jacobian required for discrete adjoint sensitivity computation is never available in a sequential technique. Also, the complexities of co-located algorithms must be correctly reflected in the Jacobian and sensitivity computations if correct optimal structures are to evolve.

We build an Automatic Differentiation library, christened '$\mathcal{R}$apid', to compute accurate Jacobians and other necessary derivatives for the discrete adjoint method in the context of an unstructured co-located sequential pressure based algorithm. The library is designed to provide a problem-agnostic pathway to automatically computing all required derivatives to machine accuracy. With sensitivities obtained from the $\mathcal{R}$apid library, we next develop and demonstrate topology optimization for multidimensional laminar flow applications. We present a variety of test cases involving internal channel flows as well as external flows, for a range of Reynolds numbers.

An essential feature of $\mathcal{R}$apid is that it is not necessary to write new code to find sensitivities when new physics, such as turbulence models, are

added, or when new cost functions are considered. The next step is therefore to extend the topology optimization for flow problems to the turbulent regime. Based on the Spalart-Allmaras RANS turbulence model, the topology optimization methodology for steady state turbulent flow problems is developed and demonstrated for channel flow problems.

Finally we develop topology optimization methodology for forced convection applications which requires the coupling of the Navier-Stokes and energy equations and which are typically solved sequentially in finite volume schemes. The coupled nature of the problem introduces the concept of multi-objective opposing cost functions from the two physical models, for example, minimizing pressure drop and simultaneously maximizing heat transfer. Techniques to obtain sensitivities for forced convection with laminar and turbulent flow with $\mathcal{R}$apid are presented. Challenges for topology optimization resulting from multi-objective cost functions are discussed.

We believe this is the first time that a complete topology optimization framework using an unstructured finite volume method and the discrete adjoint method, fully generalizable to practical use in commercial solvers and for industrial applications, has been demonstrated in the open literature. The methodologies developed here provide a basis for performing topology optimization involving other transport phenomena, more complex cost functions and more realistic constraints.

Table of Contents

List of Tables

List of Figures

Chapter 1

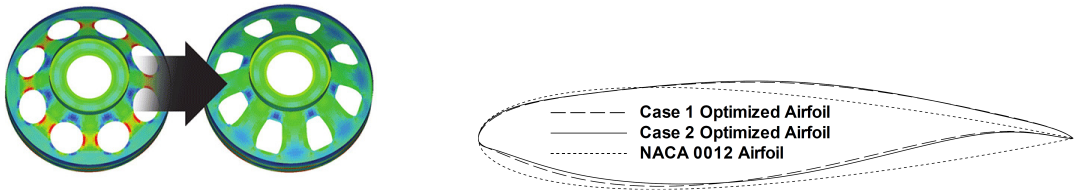
Introduction

1.1 Introduction to topology optimization

In Computer Aided Engineering (CAE), a design engineer creates a geometry, sets relevant design constraints, and boundary and initial conditions, and performs relevant engineering analyses to ensure that the certain design objectives are met. For instance, the goal of a structural engineer may be to keep stress concentration levels below yield limits. In fluid mechanics or heat transfer applications, the design may seek to reduce the pressure drop or to increase heat transfer. The design may be optimized for best performance. A variety of simulation methods, such as finite element or finite volume methods, are typically used to perform the analysis that forms the core of the optimization process.

Many papers have been published on shape optimization by both the structural and the fluids-thermal community [6, 7, 8]. Shape optimization starts with a notional or proposed shape; the optimization procedure determines the parameters of the shape (say, airfoil chord length or camber) to achieve the optimal quantity of interest (QoI) (illustration in Figure 1.1).

Topology optimization differs from shape optimization in a fundamental



(a) (b)

Figure 1.1: Illustration of shape optimization. (a) The shape of the holes of the structure is optimized to reduce stress concentration levels (Courtesy-Altair [1]). (b)Shape optimization of airfoil shapes for different criteria (Adapted from [2]).

way. The objective here is to determine the template shape itself, as illustrated in Figures 1.2 and 1.3. Here, there is no template which is the design input - the design is the output of topology optimization process. The process is intended to bring out the optimal geometrical design (or topology) for the set of given loads, boundary conditions and other requirements. Topology optimization is a method used to determine the material distribution in a given design space that maximizes a quantity of interest (QoI) [9, 10]. The power of topology optimization lies in its ability to realize design solutions that are not initially apparent to the engineer.

1.2 Current status of topology optimization

In the area of structural mechanics, topology optimization has become a well-established procedure, and has been used to obtain initial conceptual designs. Here, the QoI to be maximized or minimized is typically the stiffness or compliance of the structure [11, 3]. In the last decade, commercial CAE

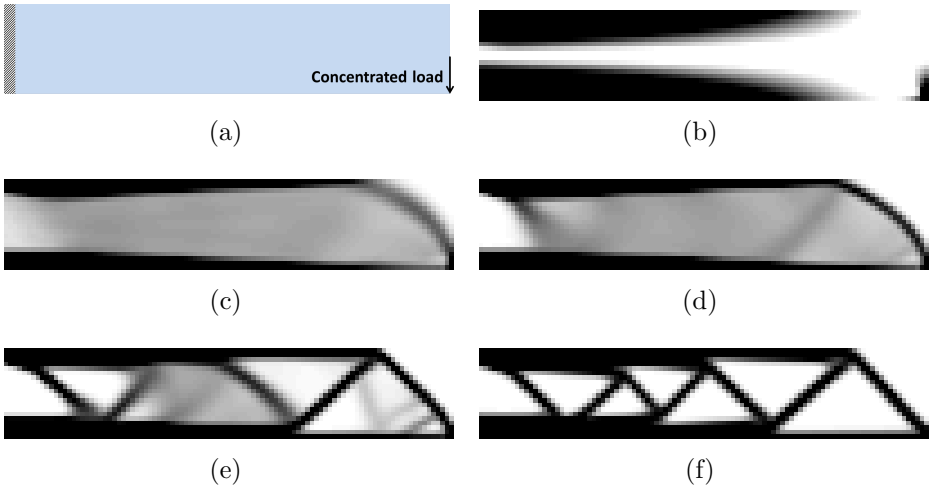


Figure 1.2: Illustration of topology optimization to determine the structure with maximum stiffness subjected to a concentrated load at its free end [3]. (a) Design space; 50% of this space is constrained to be filled with material. (b) - (f) Evolution of the design leading to the optimized geometry. (Based on open source MATLAB code in [3]).

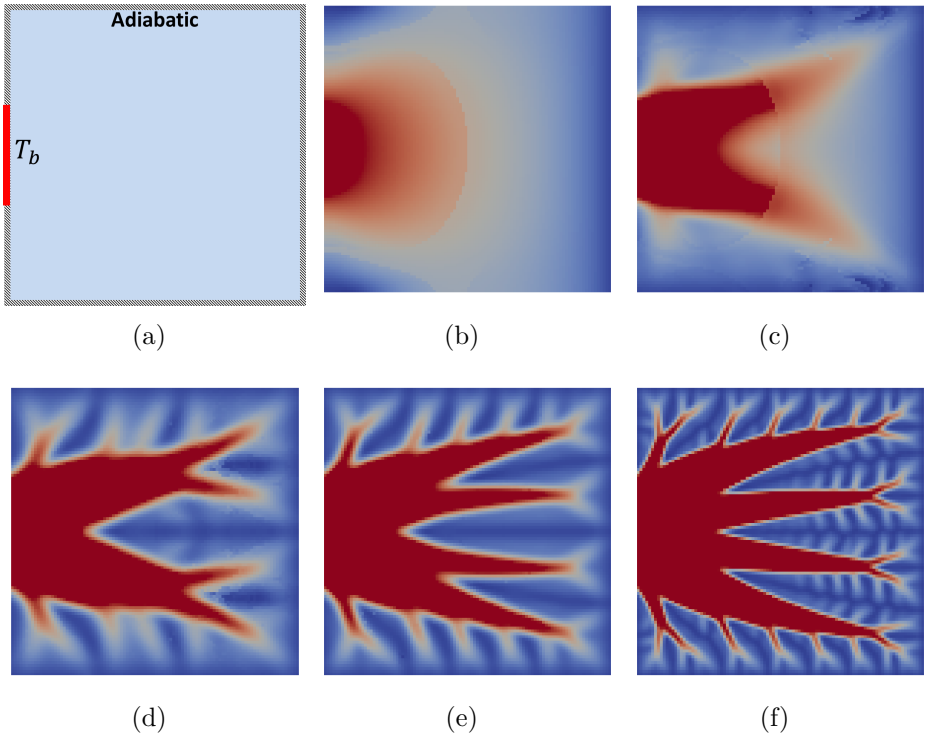


Figure 1.3: Topology optimization of a heat dissipating structure. The design space is filled with a fixed amount of conducting material to remove the heat generated in the volume. (a) Design space with part of the left boundary maintained at a specified temperature T_b while keeping the remaining boundaries adiabatic. (b) - (f) Representative steps depicting the evolution of the optimal geometry.

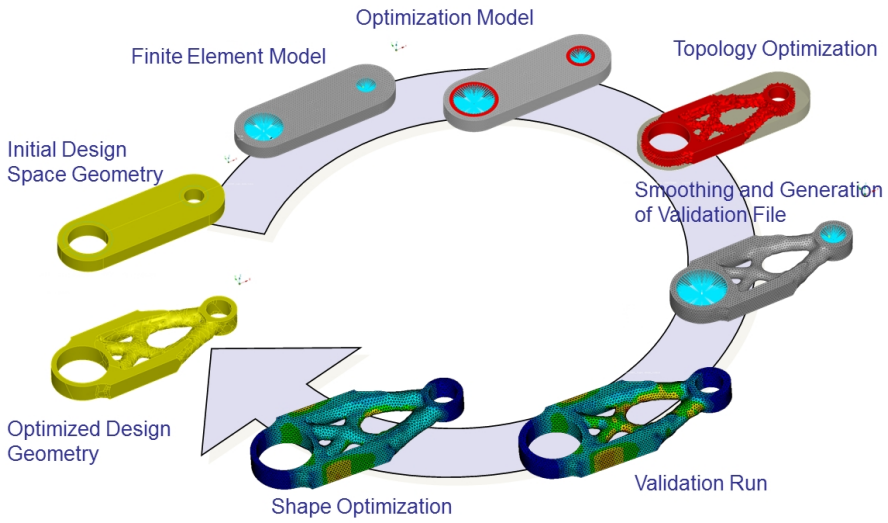


Figure 1.4: Illustration of topology optimization being used in industry in a design work flow (Courtesy - TOSCA [4]).

packages (e.g., OptiStruct, Genesis, MSC/Nastran, Ansys, Tosca, etc.) have added modules on topology optimization [12], and thus the technique is now beginning to be used in industry. Figure 1.4 illustrates a case study of topologically optimized structure obtained using the commercial package TOSCA [4]. Starting from an approximate design domain, the workflow in the figure demonstrates the evolution of the design of the structure trying to achieve maximum stiffness using minimum material.

Topology optimization has important practical applications in the automotive, aerospace, biomedical, electronics and various other industries and its applicability is likely to increase significantly with the advent of 3D printing [13, 14].

Topology optimization in industrial applications involving flow, heat or

mass transfer phenomena is far less common. The last decade has witnessed a moderate number of publications in flow and heat transfer applications; however the industry has not yet embraced the field on any large scale. The primary motivation of this dissertation is to make strides in this direction.

1.3 Application potential in transport phenomena

Topology optimization has a variety of applications in fluid flow, heat and mass transfer and other transport phenomena in all length scales. In this section, we present few of the possible applications out of the many that would serve as motivation.

For heat conduction applications, topology optimization can be used to design optimal heat dissipating structures (or fins) with maximum efficiency under given design constraints [15, 16, 17]. With the advancement of computationally feasible methods for nano-scale heat transfer applications (like the Boltzman transport equations) one can envisage the design of nano-composites to maximize a relevant QoI (say, the heat transfer rate or the thermo-electric effect) [18]. Furthermore, we may seek to determine battery electrode geometries to maximize energy density to power density ratios to significantly improve conventional battery architectures [19, 20].

Topology optimization has been performed for flow applications [21, 22] and has very recently witnessed advancements targeting real life applications [23]. However it is still far from use in practical industrial flows. At the smaller length scales, design of microfluidic devices, especially those manu-

facturable with 3D printing, have immense application potential [24]. At the larger length scales, spanning both laminar and turbulent flow regimes, efficient design of diffusors, nozzles, venturis, aerofoils, turbine blades, minimum drag bodies, and a variety of flow devices can be performed with topology optimization. Though many efficient designs have evolved over decades for such conventional applications, topology optimization can still aid in achieving better designs under various design constraints. Efficient design of flow switches [25, 26] and various biomedical devices like artificial heart valves [27] can also be investigated using topology optimization.

The application potential of topology optimization for coupled flow and thermal (or mass transfer) problems is even greater. This is because designs maximizing a flow QoI may negatively impact the thermal QoI and vice versa. Designs with an optimal balance of various QoI's can be obtained using topology optimization. For example, one can think of the design of porous structures in heat pipes which maximize the heat transfer rate while minimizing pressure drop [28]. Microfluidic mixers to achieve maximum mixing at low Reynolds number can be developed as well [29]. Similarly heat sinks, micro-pumps, heat exchanges can also be designed.

Very often topology optimization may result in complex structures beyond the reach of conventional manufacturing techniques. However, recent advances in 3D printing technologies are making it possible to manufacture very complicated geometries made of polymers or metals of different length scales [20, 30]. This will add impetus to the development of topology optimization

methodologies suitable for the fluid flow and heat transfer community.

1.4 Motivation of current work

The overwhelming majority of papers published on topology optimization use finite element methods because of the origins of the work in the structures community. The underlying numerical method enters the picture in two ways. First, the process of optimization requires the numerical solution of the physical problem at chosen points in the parameter space. Second, gradient-based optimization techniques require the evaluation of first derivatives of the QoI with respect to the design variables; these gradients are typically evaluated from the discrete algebraic equations resulting from the underlying numerical scheme. Since much of the published work has been on evolving optimal geometries for structural mechanics, topology optimization methods using finite element techniques have thus far been the norm. For fluid mechanics and heat transfer applications finite volume formulations have generally been the norm. Here, the conservation of mass, momentum and energy is enforced discretely on each computational cell [31, 32], with specialized handling of incompressible and compressible flows. Most available commercial computational fluid dynamics (CFD) packages employ the finite volume framework.

As will be discussed in later chapters, most of the literature on topology optimization applications for thermal flow problems use structured Cartesian meshes. In real life applications, the design space need not be always Cartesian. As shown in Figure 1.4, the initial design space can be non-Cartesian.

The final topology can also be quite sensitive to the geometry of the design space, especially for flow applications. It is necessary for the design space to be versatile enough to admit arbitrary geometries. Therefore topology optimization in an unstructured mesh framework is indispensable for it to be integrated into industrial applications.

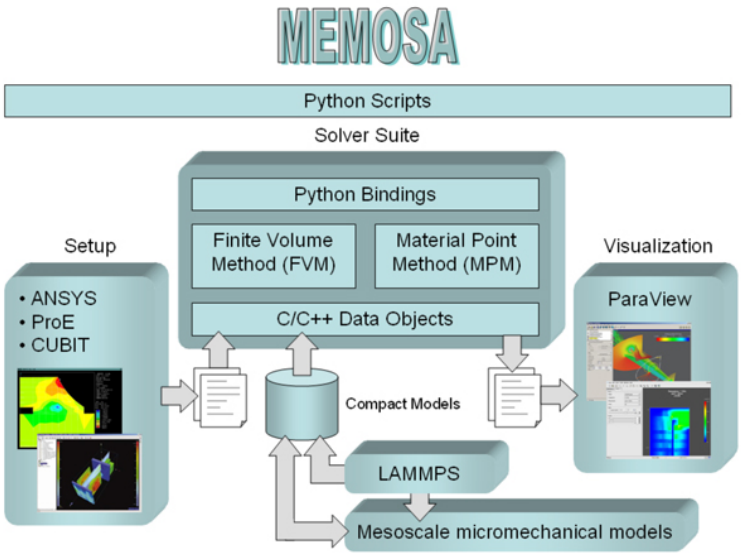
Over the years finite volume solvers have established very robust solution methodologies. The topology optimization algorithm built on such a finite volume framework should be able to work with these established procedures. For finite volume codes, the central issue that arises in the computation of sensitivity derivatives is that typical finite volume schemes do not assemble the complete Jacobian during the solution process. Secondary gradients in unstructured mesh solvers [31] are typically computed in deferred fashion, complicating the computation of sensitivity derivatives. Sequential flow solver algorithms like SIMPLE for pressure-velocity coupling [32] and colocated pressure-velocity [31] schemes also never assemble complete Jacobians. It is unclear from published finite volume solution schemes whether these issues have been completely addressed for topology optimization [33]. Modern commercial incompressible finite volume solvers use colocated pressure velocity schemes for unstructured meshes. To our knowledge, there has not been a comprehensive work on topology optimization for flow problems in an unstructured colocated finite volume schemes. This step is also necessary for elevating the field to address real-life engineering applications.

To the knowledge of authors, there is only one paper published very

recently on topology optimization for turbulent flow in SIMP methodology [34], again in an finite element framework. Forced convection with flow and thermal coupling is also an active area of research.

1.5 MEMOSA

The implementations in this dissertation will be performed in the MEMOSA framework. MEMOSA is a state-of-the-art, industry standard, parallel, C++ open source multiscale multiphysics software suite developed in-house at the PRISM center [35] funded by National Nuclear Security Administration(NNSA). MEMOSA is built on a unstructured finite volume framework and can solve a wide gamut of problems in physics including fluid flow, heat transfer, electrostatics, rarefied gas dynamics, charge and species transport, phonon transport and structural mechanics. It is designed to use modular C/C++ building blocks combined with Python, which enables flexible orchestration of the solver suite, and ease of maintenance and flexibility (Figure 1.5). Meshes generated from commercial or open source software in certain formats (.cas or .msh) can be imported into MEMOSA. Various linear solvers and preconditioners have been incorporated in MEMOSA as well. In addition to a variety of post-processing tools available in MEMOSA, computational results can also be exported to various formats readable from Paraview [36] or Tecplot [37] for visualization or for further post-processing. The inclusion of topology optimization within the MEMOSA framework will make available to the open source community a serious research tool to help develop this methodology



MEMOSA Software Environment

Figure 1.5: Momosa software environment

further.

1.6 Objectives of dissertation

Our overall objective in this dissertation is to develop a robust multi-dimensional topology optimization framework based on an unstructured finite volume formulation to address laminar and turbulent incompressible flow and heat transfer problems. The intent is also to solve physically-realistic problems involving realistic boundary conditions and cost functions compared to the ones published in the literature thus far.

The specific objectives of the dissertation are to:

1. Develop and demonstrate topology optimization for multidimensional steady heat conduction problems in an unstructured finite volume framework.

2. Develop an Automatic Differentiation library to obtain all required derivatives needed for adjoint method based sensitivity computation in a problem-agnostic way.

3. Develop and demonstrate topology optimization for multidimensional laminar flow applications. An unstructured co-located finite volume scheme using a sequential pressure velocity coupling algorithm will be used as the basis.

4. Develop and demonstrate topology optimization for multidimensional turbulent flow applications within the same finite volume framework.

5. Develop and demonstrate a topology optimization methodology for coupled thermal-flow applications.

1.7 Dissertation road-map

The intent of this chapter was to motivate the field of topology optimization for applications in which flow, heat and mass transfer phenomena are dominant. In the process, a general survey of the published work in the field has been presented in this chapter. However, a comprehensive literature survey pertinent to the specific content is presented in each of the remaining chapters. We briefly outline the contents of each chapter below.

Chapter 2 presents the topology optimization procedure for heat conduction applications in an unstructured finite volume framework. The chapter also presents the building blocks and work-flow for topology optimization that is common for all applications, be it flow or coupled thermal fluid problems. The Solid Isotropic Material with Penalization (SIMP) approach which we follow for all applications throughout the dissertation is described here. The most widely used optimization algorithm for topology optimization, the Method of Moving Asymptotes (MMA), is briefly presented for completeness. The sensitivities needed for gradient-based optimization are obtained using the adjoint method, which is described in detail. The adjoint method needs partial derivatives of the discrete residuals and cost functions with respect to the state and design variables. These are obtained using manual differentiation in this chapter. Various test cases are presented at the end.

By the end of chapter 2, it will be clear that a problem-agnostic way to obtain sensitivities is needed in order to generalize and scale topology optimization to more complex applications. Towards achieving this goal, an Automatic Differentiation library christened $\mathcal{R}$apid is developed to compute all the required derivatives for the adjoint method. The development of the library is described in detail in Chapter 3. Readers interested in using or implementing the library for their applications can read the chapter as an independent one.

Having set up the necessary foundation, in Chapter 4 we consider topology optimization to flow applications. Laminar flow in an unstructured framework is considered in this chapter. The SIMPLE family of algorithms is widely

used in the finite volume community to solve the Navier-Stokes equations. We present the use of the $\mathcal{R}$apid library to obtain the sensitivities of the QoI with respect to velocities, pressure and the design variables. The great potential of the $\mathcal{R}$apid library is revealed in this chapter. Various test cases for flow applications are presented at the end of the chapter.

In Chapter 5, we turn to topology optimization for turbulent flows. We choose the Spalart-Allmaras turbulent. The Reynolds-Averaged Navier-Stokes (RANS) equations are augmented with an additional governing equation for turbulent viscosity, and yet another for computing wall distance. Again, the $\mathcal{R}$apid library is exploited to obtain sensitivity derivatives. We then go on to present topology optimization results for turbulent flows.

Topology optimization for forced convection problems is presented in Chapter 6. Here flow equations and energy transport equations are coupled. This also necessitates the use of multi-objective cost functions, considering both the flow and energy equations. Test cases are presented for forced convection problems in both laminar and turbulent flows. Some of the complexities associated with performing topology optimization involving multi-objective cost functions are also presented.

Finally, in Chapter 7, we conclude the dissertation by discussing general guidelines, pitfalls and thoughts regarding topology optimization garnered from our experience. We also discuss in detail the various avenues that needs further work to advance topology optimization to real life applications in flow, heat and other transport applications.

Chapter 2

Heat conduction on un-structured meshes

The objective of this chapter is to develop and demonstrate the topology optimization procedure for steady state heat conduction problems using an unstructured cell-centered finite volume framework. Topologies that maximize or minimize relevant quantities of interest in heat conduction applications are presented. Addressing topology optimization for steady state heat conduction also serves to introduce all the necessary framework, building blocks and work-flows. The chapter also sets the stage for later chapters where more complicated transport processes are solved. Nomenclature and typesetting conventions presented in the chapter are followed throughout the thesis.

2.1 Introduction

Heat conduction at the continuum scale is governed by diffusion. Diffusion phenomena are well explored and relatively easy to solve numerically. Topology optimization first percolated from structural applications to transport phenomena through heat conduction applications. The majority of the topology optimization research in the literature for structural applications employs the well-established 'solid isotropic material with penalization' (SIMP)

approach (Section 2.5.1) in conjunction with a gradient-based optimization algorithm (Section 2.5.5). Similarly, topology optimization for heat conduction has also used SIMP and gradient-based optimization in the last decade and half.

Bendsoe and Sigmund presented a simple application of topology optimization for steady state heat conduction in [9, 3]. Since then many papers have been published along similar lines. Gersborg-Hansen *et al.* presented the same problem using finite elements in [9] and with a finite volume methods in [38]. Gao *et al.* extended the methodology for heat conduction by considering both design-independent and design-dependent heat loads [39]. Similar work with slight variations was presented in [40, 41]. A multi-objective TO conduction problem with two opposing cost functions was presented by Marck *et al.* using a finite volume method [16].

Bruns extended topology optimization for heat transfer applications incorporating convection effects, but did not explicitly solve the underlying fluid flow and convective heat transfer problem. Convection effects were absorbed into the heat conduction equation through source/sink terms. Similar work was done by Iga *et al.* [42]. Very recently, Zhou *et al.* published similar problems using CAE industry software tools[43].

Only a few papers have been published on topology optimization using the finite volume framework. In their implementation of TO for heat conduction, Hansen *et al.* [38] discuss an elementary implementation using a Cartesian node-based finite volume method. The work by Marck *et al.* [16]

employed a Cartesian cell-based finite volume method.

Most of the literature on TO for any applications uses structured Cartesian meshes. A few authors [44, 45] have presented topology-optimized designs using unstructured meshes, all in the finite element framework, but few details have been given. Ref. [46] discusses the integration of topology optimization into the CAD process using unstructured meshes. Talischi *et al.* present a general 2D topology optimization framework using unstructured polygonal finite element meshes [47] and also discuss some ways to reduce the computational expenses of unstructured meshes compared to their structured counterpart. It would be difficult to integrate topology optimization into industrial applications unless the design space is versatile enough to admit arbitrary geometries. Furthermore, design constraints and boundary conditions may be more accurately represented in such geometries, and the ability to adapt the mesh to the solution presents opportunities to refine preliminary designs resulting from topology optimization.

Many additions are incorporated for solving governing equations on unstructured meshes as compared to structured meshes [31, 48]. For instance, the non-orthogonal nature of the grids necessitates the computation of a 'secondary gradient flux' term for proper discretization of the diffusion operator [31, 49]. The gradients of the transported variable at the face centroids (which in turn requires gradients at cell centroids) are required to compute the secondary gradient flux term. Gradients of the transported variable, be it temperature or velocity components, are also required in many other cases. For

example, velocity gradients are required to compute the production term in turbulence models or to compute the strain rate for non-Newtonian viscosity models [48].

The central issue that arises in the computation of sensitivity derivatives is that typical finite volume schemes do not assemble the complete Jacobian during the solution process. Secondary gradients in unstructured mesh solvers [31] are typically computed in deferred fashion, complicating the computation of sensitivity derivatives. Similarly, production terms in turbulence models are not included in the Jacobian, but are treated explicitly. This again complicates the computation of sensitivity derivatives needed for topology optimization. It is unclear from the published finite volume solution schemes whether these issues have been completely addressed for topology optimization [33]. In this dissertation, an automatic differentiation framework is developed which allows us to address the lack of complete Jacobians in finite volume schemes.

The objective of this chapter is to develop and demonstrate the topology optimization procedure for steady state heat conduction problems using an unstructured cell-centered finite volume framework. We perform topology optimization in 2D and 3D heat conduction problems, including conjugate heat transfer, heat transfer in the presence of heat sources, and for a variety of boundary conditions. A variety of cost functions, including that of maximizing heat transfer or minimizing material temperature (i.e. minimizing enthalpy), are explored, subject to volume constraints on the amount of available volume

of a particular material. The implementation in this chapter is performed using MEMOSA [50] based on the methodology described in [31].

We now briefly discuss the contents of the chapter. The governing equation for heat conduction, boundary conditions and the corresponding finite volume formulation are briefly outlined in Sections 2.3 and 2.4 with specific emphasis on residual formulation. In Section 2.5, we describe the building blocks of topology optimization. We first describe the SIMP (Solid Isotropic Material with Penalization) method for topology optimization. The mathematical formulation for topology optimization using this method and the corresponding numerical solution procedures are also discussed in the following sub-sections. The gradient-based topology optimization algorithm requires calculation of the sensitivity field. In Section 2.5.4 we develop the procedures employed to compute the sensitivity field based on the finite volume framework. The method of moving asymptotes (MMA) is discussed briefly in the subsequent sub-section. In results section, we demonstrate the process of topology optimization for heat conduction using three test cases motivated by applications in heat transfer.

2.2 Notation and terminology

The following notations and terminology is used throughout the dissertation.

Fields are generic quantities defined throughout the domain that are functions of space and time. The un-bold variables (e.g T or k) are continuous

scalar fields. Variables with an overhead arrow (e.g. $\vec{V}, \vec{J}$ or $\vec{A}$) are vector fields in the domain. Variables with subscripts (e.g. T_i, k_i or β_i) are discrete values of the corresponding variables defined at cell centroids or face centroids (discussed in Section 2.4). Variables with subscripts (e.g. the components of velocity vector V_i) may also be used to identify the scalar components of the vector depending on the context. The bold variables (e.g. $\boldsymbol{T}$, $\boldsymbol{k}$, $\vec{\boldsymbol{V}}$ or $\boldsymbol{\beta}$) represent the collection of discrete variables in all the n cells in the domain. For instance $\boldsymbol{T} = \{T_1, T_2, \cdots, T_n\}$ or $\vec{\boldsymbol{V}} = \left\{\vec{V}_1, \vec{V}_2, \cdots, \vec{V}_n\right\}$.

We classify variables as state variables and design variables. The variables solved in the differential equations are termed state variables ($T, \vec{V}, p$ for example) while those determined by the optimization process are termed design variables (β, the cell volume fraction of solid, for example).

2.3 Governing equations and boundary conditions

The governing equation for steady state heat conduction is given by

$$\nabla \cdot (k \nabla T) = S^T \tag{2.1}$$

where k represents the thermal conductivity, $T(\vec{r})$ the temperature field, $\vec{r}$ the position vector, S^T the heat source in the domain.

We consider Dirichlet and Neumann boundary conditions in this chapter. A Dirichlet condition corresponds to a boundary at a given temperature, $T_b = T_{(b,given)}$, while a Neumann condition corresponds to a specified heat flux

at the boundary expressed as $-(k\nabla T)_b \cdot \vec{n} = q_{b,given}$, where $\vec{n}$ represents the outward-pointing unit normal vector at the boundary.

2.4 Numerical method

2.4.1 Finite volume method (FVM)

We employ the unstructured cell-centered finite volume scheme described in [31] for solving Eq. 2.1. The design domain is discretized using arbitrary convex polyhedral cells. All the variables of interest including temperature, thermal conductivity and design variables are stored at cell centroids. The governing equations are discretized by enforcing conservation on each cell as described in [32, 31] which yields a system of linear equations for temperature at cell centroids.

The linear system is set up in a 'residual formulation' which is described shortly. Accurate computations of sensitivity derivatives are very important in the process of topology optimization. Residual formulations are also employed for the computations of sensitivity derivatives based on the adjoint method described in Section 2.5.4. Though the governing equations increase in complexity in the subsequent chapters, the residual formulation and the adjoint method based sensitivity computation used throughout the dissertation for all physical models retain the structure and form presented in this chapter.

2.4.2 Discretization

Let the total number of cells in the discretized domain be n. Consider two representative cells $C0$ and $C1$ as shown in Figure 2.1(a), separated by a shared face f. A non-orthogonal local coordinate system (ξ, η) as shown in Figure 2.1 is employed. The centroid-to-centroid direction is denoted by ξ and a direction tangential to face f is given by η; the corresponding unit vectors are $\vec{e}_\xi$ and $\vec{e}_\eta$. The area vector normal to the face is given by $\vec{A}_f$. Details of the discretization are given in [31]. For later discussions, two representative cells of a Cartesian mesh are depicted in Figure 2.1(b).

The heat conduction equation (Eq. 2.1) is integrated over each control volume and the divergence theorem applied to yield

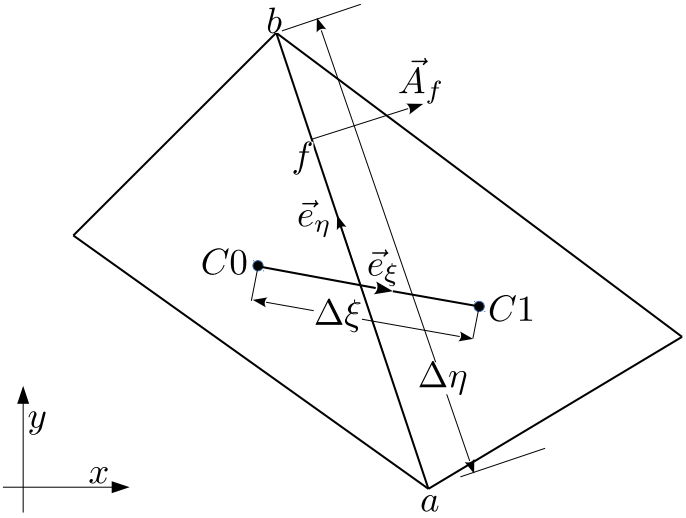
$$\int_A \vec{J}_f \cdot d\vec{A} = \int_{\Delta\mathscr{V}} S^T d\mathscr{V} \tag{2.2}$$

where $\Delta\mathscr{V}$ is cell volume and $d\vec{A}$ is the elemental area vector pointing outward from the control volume. $\vec{J}_f$ is the heat flux vector on faces of control volume, defined as

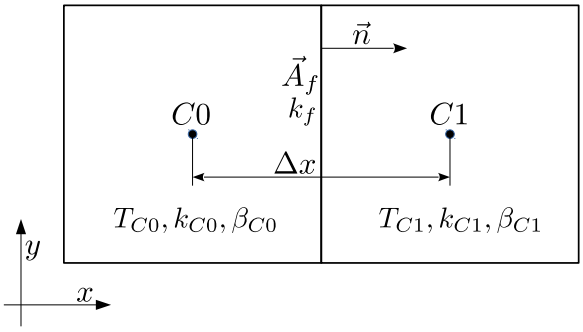
$$\vec{J}_f = -k_f \nabla T \tag{2.3}$$

The thermal conductivity at the face k_f, is calculated using harmonic average, which is given by

$$k_f = \frac{2k_{C0}k_{C1}}{k_{C0} + k_{C1}} \tag{2.4}$$



(a) Two representative neighboring cells and sharing a common face f are shown.



(b) Two neighboring finite volume cells, $C0$ and $C1$, in a cartesian mesh.

Figure 2.1: Domain discretization and variable storage in a cell-based finite volume method.

The heat transfer rate leaving the cell $C0$ through face f and entering $C1$ is approximated by [48],

$$\vec{J}_f \cdot \vec{A}_f = -\frac{k_f}{\Delta\xi} \frac{\vec{A}_f \cdot \vec{A}_f}{\vec{A}_f \cdot \vec{e}_\xi} \left(T_{C1} - T_{C0}\right) + \mathcal{S}_f \tag{2.5}$$

where $\Delta\xi$ is the centroid-to-centroid distance and T_{C1} and T_{C0} are the cell centroid temperatures. $\mathcal{S}_f$ is termed the secondary gradient flux and results from mesh non-orthogonality. The term drops out for the Cartesian mesh shown in Figure 2.1(b). The secondary gradient is given by [48],

$$\mathcal{S}_f = -k_f \left(\nabla T\right)_f \cdot \vec{A}_f + \frac{k_f}{\Delta\xi} \frac{\vec{A}_f \cdot \vec{A}_f}{\vec{A}_f \cdot \vec{e}_\xi} \left(\nabla T\right)_f \cdot \vec{e}_f \Delta\xi \tag{2.6}$$

This term requires the temperature gradient $\left(\nabla T\right)_f$ on the face between the cells. It is generally calculated as the average of the cell gradients $\left(\nabla T\right)_{C0}$ and $\left(\nabla T\right)_{C1}$ calculated at cell centroids of the cells $C0$ and $C1$. The cell gradient $\left(\nabla T\right)_{C0}$ is computed using a linear least squares approximation[48] given by,

$$\left(\nabla T\right)_{C0} = \left(\boldsymbol{M}^T \boldsymbol{M}\right)^{-1} \boldsymbol{M}^T \left(\Delta T\right)_{C0} \tag{2.7}$$

where the matrix $\boldsymbol{M}$ is formed with the differences in co-ordinates of the cell centroids of cell $C0$ with each of its j neighbors. In 2D it is given by

$$\boldsymbol{M} = \begin{bmatrix} \Delta x_1 & \Delta y_1 \\ \Delta x_2 & \Delta y_2 \\ \vdots & \vdots \\ \Delta x_j & \Delta y_j \end{bmatrix} \tag{2.8}$$

$(\Delta T)_{C0}$ is the difference in temperature of the the cell $C0$ with each of its neighbors given by,

$$(\Delta T)_{C0} = \begin{bmatrix} T_{C1} - T_{C0} \\ T_{C2} - T_{C0} \\ \vdots \\ T_{Cj} - T_{C0} \end{bmatrix} \tag{2.9}$$

where T_{C0} and T_{C1} are temperatures at the cell centroids and $(\Delta x, \Delta y)$ denote the distance between the cell centroids.

2.4.3 Residual formulation and linear system

The discrete residual $\mathcal{R}_i^T$ for a cell i is given by

$$\mathcal{R}_i^T = \left(\sum_{faces} -\vec{J}_f \cdot \vec{A}_f + S^T \Delta \mathscr{V} \right)_i \tag{2.10}$$

where the quantity $\vec{J}_f \cdot \vec{A}_f$ is given by Eq. (2.5) is summed over all the faces of the corresponding control volume i. At convergence, $\boldsymbol{\mathcal{R}}^T(\boldsymbol{T}) = \mathbf{0}$ for the conservation principle to hold true i.e.,

$$\boldsymbol{\mathcal{R}}^T(\boldsymbol{T}) = \begin{bmatrix} \mathcal{R}_1^T(T_1, T_2 \dots T_n) \\ \mathcal{R}_2^T(T_1, T_2 \dots T_n) \\ \vdots \\ \mathcal{R}_n^T(T_1, T_2 \dots T_n) \end{bmatrix} = \mathbf{0}. \tag{2.11}$$

To solve for $\boldsymbol{T}$, we may recast the nominally linear system in delta form as:

$$\frac{\partial \boldsymbol{\mathcal{R}}^T}{\partial \boldsymbol{T}} \boldsymbol{\delta} + \boldsymbol{\mathcal{R}}^T = \mathbf{0} \tag{2.12}$$

where $\frac{\partial \boldsymbol{\mathcal{R}}^T}{\partial \boldsymbol{T}}$ is the Jacobian matrix given by,

$$\frac{\partial \boldsymbol{\mathcal{R}}^T}{\partial \boldsymbol{T}} = \begin{bmatrix} \frac{\partial \mathcal{R}_1^T}{\partial T_1} & \frac{\partial \mathcal{R}_1^T}{\partial T_2} & \cdots & \frac{\partial \mathcal{R}_1^T}{\partial T_n} \\ \frac{\partial \mathcal{R}_2^T}{\partial T_1} & \frac{\partial \mathcal{R}_2^T}{\partial T_2} & \cdots & \frac{\partial \mathcal{R}_2^T}{\partial T_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{R}_n^T}{\partial T_1} & \frac{\partial \mathcal{R}_n^T}{\partial T_2} & \cdots & \frac{\partial \mathcal{R}_n^T}{\partial T_n} \end{bmatrix} \tag{2.13}$$

Here $\boldsymbol{\delta} = \boldsymbol{T} - \boldsymbol{T}^{prev}$ is the difference between the current solution and the previous iterate.

A variety of linear solvers, including an algebraic multigrid solver [51], may be used for solving the resulting linear equations with Picard iteration to address non-linearities. The temperature field thus obtained can be processed to obtain a QoI such as the heat rate on the boundary.

The fact that finite volume schemes do not assemble the complete Jacobian during the solution process, mentioned earlier, can be clarified now. Though the residual $\mathcal{R}_i$ retains the the secondary gradient term in Eq. 2.10, it is neglected while forming (linearizing) the Jacobian (Eq. 2.13) for the linear system. Secondary gradients are typically computed in deferred fashion. The linear system (Eq. 2.12) is solved multiple times (Picard iterations), each time with corrected residuals. The secondary flux term in Jacobian is neglected not

only because it is difficult to linearize, but also for *stability* purposes[48]. The reader is reminded that the Jacobians need not be exact and are sometimes modified in the path to solution of a linear system. However, for computing sensitivities using the adjoint method, one needs the exact Jacobian. We will get back to this discussion later.

2.5 Topology optimization

In this section we describe the main elements of topology optimization.

2.5.1 Solid isotropic microstructure with penalization (SIMP)

Topology optimization is generally formulated as a material distribution problem [52, 53]. In a typical problem, a design space is specified and discretized using elements or cells. The optimal placement of a material in the cells is determined in order to maximize or minimize an objective function, subject to constraints. Typical objective functions for conduction heat transfer may maximize the heat transfer rate on a boundary, or minimize the maximum temperature. Constraints typically require the total solid fraction to be held below a specified value. Single and multi-material problems may be posed for heat conduction, and when fluid flow is considered, multiple phases as well. In this chapter, the problem that we are considering is to fill the design space with two materials with different thermal conductivities.

As discussed in Section 2.4 on FVM, the design space in which material must be distributed is discretized into convex polyhedral control volumes on

which energy balances are enforced. The goal of topology optimization is to determine which of these control volumes are filled with one material and which with other. The material distribution process is parameterized by defining a cell-wise constant binary design variable, $\beta^i = \{0, 1\}$ which indicates whether cell i consists of the chosen material ($\beta^i = 1$) or not ($\beta^i = 0$). This design variable is discrete and may represent material or void, two heterogeneous materials, or multiple phases, depending on the problem being solved.

Direct search (discrete) methods like evolutionary algorithms or simulated annealing may be used to determine such a material distribution process [9]. However these methods can be prohibitively expensive or intractable for large scale problems. The goal is to be able to use gradient-based continuous optimization methods to tackle such problems. The most commonly used approach is to replace the integer variables with continuous variables i.e. $\beta \in [0, 1]$, and then introduce some form of penalty that steers the solution to discrete binary values.

This method is the most popular numerical method for topology optimization, now known as the '*solid isotropic microstructure with penalization*' (SIMP) method [13]. Each cell or control volume is associated with a value β which can be interpreted as the microscopic volume fraction of the material in the cell.

This design variable is introduced into the governing transport equations by the interpolation of material properties as functions of β. In addition, terms like the source term or forcing terms in the governing equations are also

interpolated in terms of the design variable. Such interpolation functions are generally chosen based on physical arguments, though this is not always necessary. The continuous nature of β provides a continuous transition between the two phases or materials. To push the continuous variable to binary values, penalization is introduced in the chosen interpolation function. The interpolation scheme must also be closely linked to the optimization problem being solved. Properly chosen interpolation schemes may bestow other theoretical or computationally advantageous features for specific problems.

Various interpolation schemes have been used in the literature for performing topology optimization. Properties like conductivity, diffusivity, macroscopic density, specific heat capacity etc. have been interpolated with the design variable as [41],

$$\Gamma_{eff}\left(\beta\right) = \Gamma_2 + \left(\Gamma_1 - \Gamma_2\right)\beta^p \tag{2.14}$$

Here Γ_1 and Γ_2 are the properties of the two materials respectively, Γ_{eff} is the effective material property of a cell and p is the penalization factor. The intent of the optimization is to converge Γ_{eff} to either to Γ_1 or Γ_2, thus giving one of the identities to the corresponding discretized cell.

At this juncture, we discuss the commonly used interpolation functions for effective properties or fields found in the literature. For topology optimization of Stokes flow problems,α, the impermeability of the porous medium, must be interpolated in terms of β. Whenβ varies between 0 and 1, it corresponds

to the volume fraction of artificial solid and fluidic phases respectively. The interpolation used in [21] is,

$$\alpha_{eff}\left(\beta\right) = \alpha_f + \left(\alpha_s - \alpha_f\right)\beta\frac{1+p}{\beta+p} \tag{2.15}$$

where α_s (a very high number) and α_f $(= 0)$ represent impermeability of solid and fluid phases respectively. p is the penalization factor.

For coupled natural convection-flow problems, the effective Peclet number is interpolated in [54] as ,

$$Pe_{eff} = \frac{C_k\left(1+p\beta\right)}{\beta\left(C_k\left(1+p\right)-1\right)+1}Pe_f \tag{2.16}$$

where, Pe_f is the Peclet number of the fluid, $C_k = \frac{k_f}{k_s}$ is the ratio of the fluid to solid conductivity, $Pe_s = C_k Pe_f$ is the Peclet number of the solid and p is the penalization factor. Eqs. 2.15 and 2.16 are known as '*Rational Approximation of Material Properties*' (RAMP) functions in the literature.

Evgrafov *et al.* [55] performs topology optimization for nano-scale heat transfer applications, where the material property that enters the governing equations is the mean free path of phonons, Λ. The authors use the following interpolation scheme for the effective mean free path,

$$\Lambda_{eff}^{-1} = \left(1-\beta\right)\Lambda_1^{-1} + \beta\Lambda_2^{-1} \tag{2.17}$$

where Λ_1 and Λ_2 are the mean free path of phonons for two different materials. There is no explicit penalization factor here.

The interpolation function chosen for the topology optimization for steady heat conduction in this chapter is Eq. 2.14. The only material property that must be considered here is the thermal conductivity, and therefore

$$k_{eff}(\beta) = k_2 + (k_1 - k_2)\beta^p \tag{2.18}$$

where k_1 corresponds to the high conducting material and k_2 corresponds to the low conducting material. β represents the elemental volume fraction of high conducting material in each finite volume element. The heat generation source term may also be interpolated in terms of β, as presented in some of the results (Section 2.6).

$$S = \alpha\beta^p \tag{2.19}$$

2.5.2 Mathematical formulation

Topology optimization is essentially a partial differential equation (PDE) constrained optimization. Here, a functional or cost function c is minimized. The cost function is, generally a function of the field (called the state variable) that is solved for in the PDE, as well as the design variable β. For heat conduction the cost function is denoted as $c(T, \beta)$, where T is the temperature field.

The goal of topology optimization is to distribute specified volumes of two materials in the design space. Let ε and $1-\varepsilon$ be the specified volume fractions of materials 1 and 2 with which we wish to fill the design space. In topology optimization, we seek appropriate values for the design variable that minimizes the functional c by satisfying all the constraints of the problem. If $\mathscr{V}_0$ is the volume of the initial design space, then we can define a function $\mathscr{V}_1(\beta)$ which is equal to the specified volume of material 1. Thus mathematically, the topology optimization problem in a continuous formulation is given by

$$\begin{array}{rc} \min: & c = c(T,\beta) \\ \text{subject to}: & \frac{\mathscr{V}_1(\beta)}{\mathscr{V}_0} \leq \varepsilon \\ & \nabla \cdot (-k_{eff}(\beta)\nabla T) = S(\beta) \\ & 0 \leq \beta \leq 1 \end{array} \tag{2.20}$$

In discretized form, the optimization problem is given by

$$\begin{array}{rc} \min: & c = c(\boldsymbol{T},\boldsymbol{\beta}) \\ \text{subject to}: & g := \frac{\sum_i^n \beta_i}{n} - \varepsilon \leq 0 \\ & \boldsymbol{\mathcal{R}}^T(\boldsymbol{T},\boldsymbol{\beta}) = \mathbf{0} \\ & 0 \leq \boldsymbol{\beta} \leq 1 \end{array} \tag{2.21}$$

It must be noted that the formulation may have more than just the volume constraint or replaced with another constraint. One can also pose other types of constraints such as in [29], where the total pressure drop is kept below a given value. However specifying volume fraction constraints is the most common type.

The optimization problem 2.21 is generally solved using a nested formulation, wherein the discretized system of equations for the state field is

solved separately from the design problem. An algorithm to perform such an optimization is discussed in the next section.

2.5.3 Algorithm for topology optimization

Nearly all published work on topology optimization follows the methodology depicted in the Figure 2.2. Further details may be found in [9]. First, the design variable field $\boldsymbol{\beta}$ is initialized and effective material properties are calculated using the material interpolation schemes (Eq. 2.18) discussed in Section 2.5.1. The residual equations are set up using the numerical scheme used for discretizing the governing equations (Section 2.4.2). These discrete equations are solved for the state variable field $\boldsymbol{T}$. The cost function is then calculated based on the solved state and the design variable field. Using a non-linear optimization program, the design variable field is updated so as to minimize the cost function. The new effective material properties are then calculated. The iteration continues as shown in the flowchart until the difference in the values of $\boldsymbol{\beta}$ satisfies a prescribed cost function no longer falls.

2.5.4 Sensitivity calculations using adjoint method

In gradient-based optimization, the process of finding the optimal value of the $\boldsymbol{\beta}$ field relies on the calculation of sensitivity derivatives $\frac{dc}{d\boldsymbol{\beta}}$. They represent how the cost function c responds to changes in the discrete design variable in each cell in the computational domain. There are several methods for calculating sensitivity derivatives including finite differencing, direct differentiation,

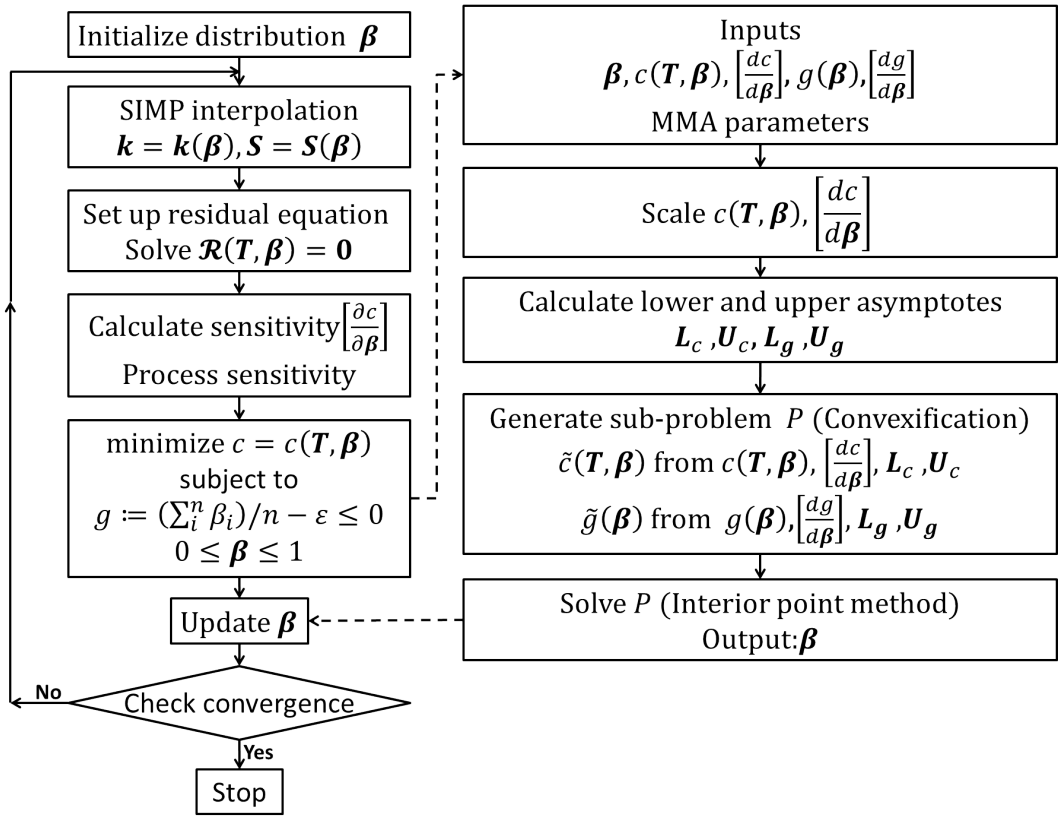
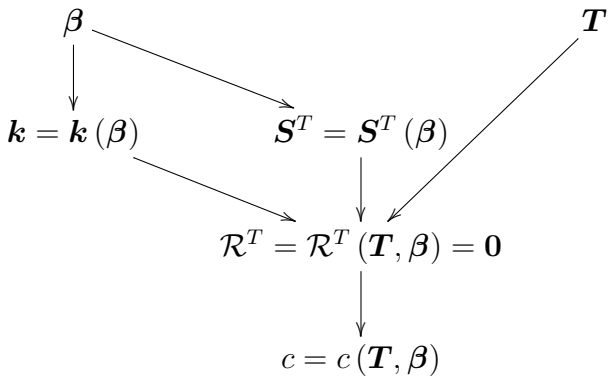


Figure 2.2: Flowchart for topology optimization (left). Flowchart for solving non-linear optimization problem iteratively using convexified optimization sub-problems (right)

and the adjoint method. We employ the adjoint method in this study. Adjoint methods are very efficient in calculating sensitivities when the numbers of optimization variables is large. Since the numbers of optimization variables is equal to the mesh size which can be quite large, adjoint methods are well suited for topology optimization [56, 15].

The adjoint method that we mention throughout the dissertation refers to discrete adjoint method. Here the adjoint system is formed with the discretized or linearized PDE. In continuous adjoint method, the adjoint of the PDE is first formed and then discretized for solution purposes.

We drop the super-script T for $\mathcal{R}^T$ in the following discussion, since the method is general and will be used for all the physical models described in the later chapters. Given a distribution of the cell-wise values of $\boldsymbol{\beta}$, the finite volume solution for the discrete temperature field (or any state variable) is obtained as discussed in Section 2.4. At this point, we have the residual vector $\mathcal{R} = \mathbf{0}$ (Eq. 2.11). The adjoint method is applied following a converged finite volume solve. At this point, the variables of interest and their dependencies are given by



(2.22)

A perturbation in the design variables $\boldsymbol{\beta}$ changes $\boldsymbol{k}$ and $\boldsymbol{S}$, which is in turn propagated to the residual $\boldsymbol{\mathcal{R}}$ and cost function c. Note that the residual $\boldsymbol{\mathcal{R}}$ has moved away from $\mathbf{0}$ at this point. A new finite volume solve to drive back $\boldsymbol{\mathcal{R}}$ to $\mathbf{0}$ changes the temperature $\boldsymbol{T}$, also contributing an additional change to the cost function c.

For simplicity, let us consider just one design variable β_γ. The total variation in the cost function c due to the perturbation of this single design variableβ_γ is given by,

$$\delta c = \left[\frac{\partial c}{\partial T_i}\delta T_i\right]_I + \left[\frac{\partial c}{\partial \beta_\gamma}\delta\beta_\gamma\right]_{II} \tag{2.23}$$

Indices with Roman subscripts imply Einstein summations, while indices with Greek subscripts do not imply summation. The total variation in residual vector is given by,

$$\delta\mathcal{R}_j = \left[\frac{\partial \mathcal{R}_j}{\partial T_i}\delta T_i\right]_I + \left[\frac{\partial \mathcal{R}_j}{\partial \beta_\gamma}\delta\beta_\gamma\right]_{II} = \mathbf{0} \tag{2.24}$$

The power of adjoint method lies here. Eqs. 2.23 and 2.24 are also true for simultaneous variations of all design variables$\boldsymbol{\beta}$ independently.

$$\delta c = \left[\frac{\partial c}{\partial T_i}\delta T_i\right]_I + \left[\frac{\partial c}{\partial \beta_k}\delta \beta_k\right]_{II} \tag{2.25}$$

The total variation in residual vector is given by,

$$\delta \mathcal{R}_j = \left[\frac{\partial \mathcal{R}_j}{\partial T_i}\delta T_i\right]_I + \left[\frac{\partial \mathcal{R}_j}{\partial \beta_k}\delta \beta_k\right]_{II} = \mathbf{0} \tag{2.26}$$

Note that the term I in Eq. 2.23 is different from term I of Eq. 2.25.

Since $\delta\boldsymbol{\mathcal{R}} = 0$, one can see that the scalar product of the vector $\delta\boldsymbol{\mathcal{R}}$ with any vector $\boldsymbol{\psi}$ can be added to δc without any change to it, as is shown below,

$$\delta c = \left[\frac{\partial c}{\partial T_i}\delta T_i\right]_I + \left[\frac{\partial c}{\partial \beta_k}\delta \beta_k\right]_{II} - \psi_j \left(\left[\frac{\partial \mathcal{R}_j}{\partial T_i}\delta T_i\right]_I + \left[\frac{\partial \mathcal{R}_j}{\partial \beta_k}\delta \beta_k\right]_{II} \right) \tag{2.27}$$

Eq. 2.27 is slightly regrouped as follows,

$$\delta c = \left\{ \left[\frac{\partial c}{\partial T_i}\delta T_i\right]_I - \psi_j \left[\frac{\partial \mathcal{R}_j}{\partial T_i}\delta T_i\right]_I \right\} + \left\{ \left[\frac{\partial c}{\partial \beta_k}\delta \beta_k\right]_{II} - \psi_j \left[\frac{\partial \mathcal{R}_j}{\partial \beta_k}\delta \beta_k\right]_{II} \right\} \tag{2.28}$$

$$\delta c = \left\{ \frac{\partial c}{\partial T_i} - \psi_j \frac{\partial \mathcal{R}_j}{\partial T_i} \right\} \delta T_i + \left\{ \frac{\partial c}{\partial \beta_k} - \psi_j \frac{\partial \mathcal{R}_j}{\partial \beta_k} \right\} \delta \beta_k \tag{2.29}$$

Since $\boldsymbol{\psi}$ can be any vector, one can choose a particular vector that satisfies the following equation,

$$\frac{\partial \mathcal{R}_j}{\partial T_i}\psi_j = \frac{\partial c}{\partial T_i} \tag{2.30}$$

where $\frac{\partial \mathcal{R}_j}{\partial T_i}$ is the Jacobian matrix $\boldsymbol{J}$ and the vector $\frac{\partial c}{\partial T_i}$ may be analytically computed using the discrete equations. Eq. 2.30 is known as the adjoint equation and the corresponding $\boldsymbol{\psi}$ the adjoint field. Such a choice of $\boldsymbol{\psi}$ results in

$$\delta c = \left\{ \frac{\partial c}{\partial \beta_k} - \psi_j \frac{\partial \mathcal{R}_j}{\partial \beta_k} \right\} \delta \beta_k \tag{2.31}$$

$$\implies \frac{dc}{d\beta_k} = \frac{\partial c}{\partial \beta_k} - \psi_j \frac{\partial \mathcal{R}_j}{\partial \beta_k} \tag{2.32}$$

The terms in Eq. 2.32 are computed using the chain rule as follows,

$$\frac{\partial c}{\partial \beta_k} = \frac{\partial c}{\partial k_i}\frac{\partial k_i}{\partial \beta_k} + \frac{\partial c}{\partial S_i}\frac{\partial S_i}{\partial \beta_k} \tag{2.33}$$

$$\frac{\partial \mathcal{R}_j}{\partial \beta_k} = \frac{\partial \mathcal{R}_j}{\partial k_i}\frac{\partial k_i}{\partial \beta_k} + \frac{\partial \mathcal{R}_j}{\partial S_i}\frac{\partial S_i}{\partial \beta_k} \tag{2.34}$$

Each individual term in Eqs. 2.33 and 2.34 may be analytically computed from the discrete algebraic finite volume equations.

Summarizing, the calculation of sensitivity derivatives is broken down into two steps. The first is the solution of the linear equation set to obtain

the adjoint field $\boldsymbol{\psi}$ defined by Eq. 2.30. The second is the computation of the necessary sensitivity derivatives. For the first step, an iterative linear solver is usually used. This step can be computationally intensive, since the adjoint linear system is generally stiff. For very stiff systems, it may be necessary to resort to direct solvers.

The second step (computing Eqs. 2.33 and 2.34), though not computationally intensive, requires the computation of the analytical derivatives of the algebraic equations resulting from the finite volume discretization. For the finite volume scheme used here for unstructured meshes, calculation of the derivative term $\frac{\partial \mathcal{R}_j}{\partial \beta_k}$ involves finding derivatives of the secondary gradient flux term (Eq. 2.6). It is extremely difficult to differentiate the secondary flux term (Eq. 2.6) with respect to state and design variables. We will develop the infrastructure for differentiating complex functions in the Chapter 3. However, the secondary gradient flux term for orthogonal meshes is zero. In such a case, all the derivative terms required for sensitivity computation for a pure heat conduction problem can be obtained through manual differentiation.

2.5.5 Non-linear optimization algorithm - Method of Moving Asymptotes (MMA)

Many optimization algorithms have been explored for topology optimization [9], including gradient-based methods such as optimality criteria (OC) methods [3], sequential linear programming (SLP) methods [57], the '*method of moving asymptotes*' (MMA)[9] and evolutionary methods such as

genetic algorithms, bi-directional evolutionary structural optimization (BESO) and others [58, 59].

We follow the MMA algorithm developed by Svanberg [60, 61]. MMA is an optimization algorithm for general nonlinear constrained problems, which is popular in the topology optimization community. The MMA algorithm handles problems with a large number of design variables subject to few constraint functions very well. Since the number of design variables is equal to the the number of discrete cells in the domain, which can be very large, MMA is well-suited to the problem considered here. The methodology underlying MMA is also well suited for parallelization (refer [62]).

MMA is a one of the class of optimization algorithms called Conservative Convex Separable Approximation (CCSA) methods . Algorithms like Sequential Linear Programming (SLP) or Sequential Quadratic Programming (SQP) iteratively solve a sequence of convex sub-problems (linearized functions or quadratic functions respectively) to reach the minimum. In a similar way MMA also solves the general non-linear optimization problem by iteratively solving a sequence of convex sub-problems constructed with some separable approximations of original functions (both objective functions and constraint functions). It is recalled that a separable function is one that can be expressed as a sum (or linear combination) of functions of each individual optimization variable. Unlike SLP or SQP, a CCSA algorithm guarantees that iterations points of the sub-problems are feasible with original constraints when solved with its generated convex separable approximations . Use of these separable

approximations makes MMA a good method capable of handling extremely large number of design variables.

Figure 2.2 also depicts the general algorithm of MMA used for topology optimization. For a given finite volume solve, the objective function $c = c(\boldsymbol{T}, \boldsymbol{\beta})$ and the sensitivities $\frac{\partial c}{\partial \beta_k}$ are calculated as mentioned in previous sections. Calculation of volume constraint function g (in Eq. 2.21) and its derivative $\frac{\partial g}{\partial \beta_k}$ are straightforward. The function values, the derivatives and the current values of $\beta_i^{(0)}$ are fed into the MMA optimizer, as shown in Figure 4. With the function values and its derivatives, approximate convex optimization sub-problems are created (described in the next paragraph), which is solved using a standard optimization algorithm such as interior point method [60, 61, 62]. The new design variable are updated with which a new subproblem is created. The procedure repeated until the difference between the successive iterative values of β_i fall below a prescribed tolerance value.

The convex optimization sub-problem is generated by replacing all the original functions (cost function and the constraint function) with conservative separable convex functions. This process is outlined here briefly from a user perspective (based on Refs. [60]). Let the original function (either cost function or constraint function) value be $f(\boldsymbol{\beta})$ and its gradient vector be $\frac{\partial f}{\partial \beta_k}(\boldsymbol{\beta})$, evaluated at a given $\boldsymbol{\beta}$. The the approximated function is constructed with a linear combination of basis functions of the form $\frac{1}{U_j - \beta_j}$ or $\frac{1}{\beta_j - L_j}$ (Eq. 2 of Ref. [60]) depending on the sign of the particular sensitivity value $\frac{\partial f}{\partial \beta_k}$, given in general form by

$$f(\boldsymbol{\beta}) \approx \tilde{f}_i(\boldsymbol{\beta}) = \sum_{j=1}^{n} \left(\frac{p_{ij}}{U_j - x_j} + \frac{q_{ij}}{x_j - L_j} \right) + r_j, \;\; i = 0, 1, \ldots m, \qquad (2.35)$$

r_j is the difference between original and approximated function calculated at $\boldsymbol{\beta}$. U_j and L_j are called upper and lower asymptotes. They are calculated using their previous iterate counterparts U_j^{prev} or L_j^{prev}, the current and previous iterates β and β^{prev} and a parameter (γ in Ref. [61]) which controls the level of conservativeness. If the asymptotes (U_j or L_j) are chosen closer to β_j the approximation functions are said to be more conservative and vice versa. For two different approximating functions $\tilde{f}_1$ and $\tilde{f}_2$ of a given original function, the former is more conservative than latter if $\tilde{f}_1 < \tilde{f}_2$. Sometimes, for instance, the constraint functions need to be chosen very conservatively depending on the problem to aid the path to optimal solution in the initial stages.

The constant p_{ij} is next determined with β , original derivatives $\frac{\partial f}{\partial \beta_k}(\boldsymbol{\beta})$ and the current upper asymptotes U_j. Similarly q_{ij} is determined but with L_j. The bounds of the original problem are also modified in each iteration, with variables called move limits (α and β in Ref. [61]; note that this β should not be confused with design variable β_i used in this dissertation as design variables). They are obtained from the original bounds of the problem, current asymptote values U_j and L_j, current iterate values $\boldsymbol{\beta}$ and two parameters (asymincr and asymdecr in Ref. [61]). Changing these parameters also change the conservativeness of the function.

Tuning the above discussed parameters by trial and error can help in obtaining convergence or increasing the rate of convergence. In addition, in practice, the MMA sub-problem also have additional linear and quadratic functions of artificial optimization variables (variables y_i, z of Eq. 3.1 in Ref. [61]) to ensure that the sub-problem always has a feasible solution. The coefficients accompanying these functions (a_i, c_i, d_i in Eq. 3.1 in Ref. [61]) also need to be chosen judiciously for practical reasons (further details in [60, 63]]).

Svanberg [61] made some practical adjustments for MMA to work effectively. We have found that it is important to scale the objective function c and its senstitiviites $\frac{dc}{d\beta_j}$, as depicted in Figure 2.2. Svanberg recommends that the objective function $c = c(\boldsymbol{T}, \boldsymbol{\beta})$ be scaled such that $1 \leq c(\boldsymbol{T}, \boldsymbol{\beta}) \leq 100$, for reasonable values of the design variables β_j. The actual order of magnitude of the objective function is obviously dependent on the problem and its parameters. The sensitivity derivatives of the objective function must also be scaled with the same scale factor. Generally, a scale factor may be chosen based on the objective function of the initial configuration.

2.5.6 Filtering sensitivities

Topology optimization is generally ill-posed unless appropriate geometric restrictions are applied [59]. For instance, the final topology can be highly mesh-dependent if not properly handled. It is not uncommon for the optimization process to get trapped in a local minimum. A filtering process is generally employed to circumvent such issues. In this chapter we use a simple

filter whereby the sensitivity derivatives in each cell are modified by adding the weighted average of the product of the sensitivity and design variable values of the neighboring cells (Eq. 7 in Ref. [64]]). The sensitivity filter modifies the sensitivities $\frac{dc}{d\beta_i}$ as follows,

$$\widehat{\frac{dc}{d\beta_i}} = \frac{1}{\max(\gamma, \beta_i) \sum_{j \in n_e} H_{ij}} \sum_{j \in n_e} H_{ij} \beta_j \frac{dc}{d\beta_j} \tag{2.36}$$

$\widehat{\frac{dc}{d\beta_i}}$ is the modified sensitivity for cell i, n_e is the set of elements i for which the center-to-center distance $\Delta(i, j)$ to element j is smaller than the filter radius r_{min}, $\gamma = 0.001$ is a small positive number to avoid division by 0, and H_{ij} is a weight factor defined as:

$$H_{ij} = \max\left(0, r_{\min} - \Delta(i, j)\right) \tag{2.37}$$

The choice of how many neighboring cells are used is heuristic. During the initial phases of optimization, a large number of neighbor cells is typically used. The neighborhood is gradually reduced as the optimization proceeds [9]. Though no sound theoretical basis exists for filtering sensitivities, numerous papers have been published based on such sensitivity modification, both in 2D and 3D problems and with many constraints. Bendsoe [9] claims that computational experience has shown that filtering is a highly efficient way to ensure mesh-independency. Sensitivity filters are also called mesh independence filters in the literature. Filters can also be used for controlling the minimum and

maximum length scales in the evolving geometry [65, 66]. This is important when taking manufacturability into consideration. Other methods, such as perimeter control and gradient restriction, have been published and may be incorporated as constraints [9, 61].

2.6 Results

We consider three test cases to demonstrate the application of the topology optimization algorithm to heat conduction problems. In the examples below, we consider both 2D and 3D problems, as well as initial design spaces that are both rectangular and non-rectangular. In all cases, a volume fraction constraint for the high-conducting material is stated as an inequality, as in Eq. 2.20. In every case presented below, the optimal solution is found to have a volume fraction equal to the specified maximum, ε.

2.6.1 Test case 1: Conduction in plane slab

In this test case, we perform a verification test for the optimal material distribution for heat conduction in a plane slab. Figure 2.3 shows the 2D rectangular design space, with two materials of given volume fractions which must be distributed to maximize a cost function. A chosen maximum volume fraction of the design space, $\varepsilon = 0.4$ in this case, is to be filled with a highly conductive material of conductivity k_1, with the remaining volume occupied by a material with low thermal conductivity k_2. Constant temperature boundary conditions are applied on the two vertical boundaries. The top and bottom

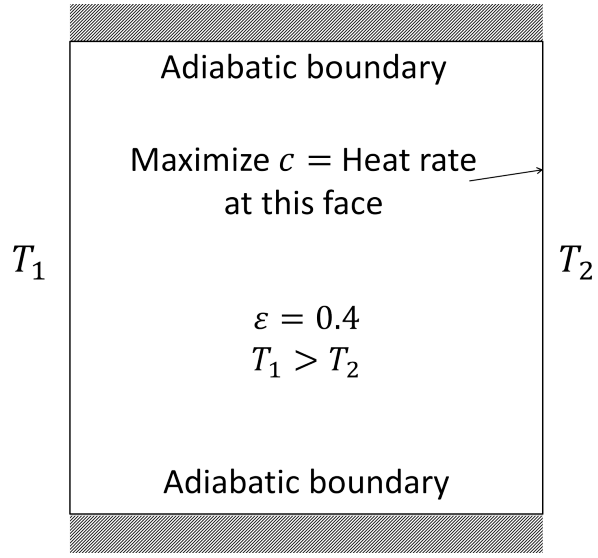


Figure 2.3: Schematic for problem solved in test case 1.

faces are adiabatic boundaries. The objective is to maximize the heat rate on the right boundary,

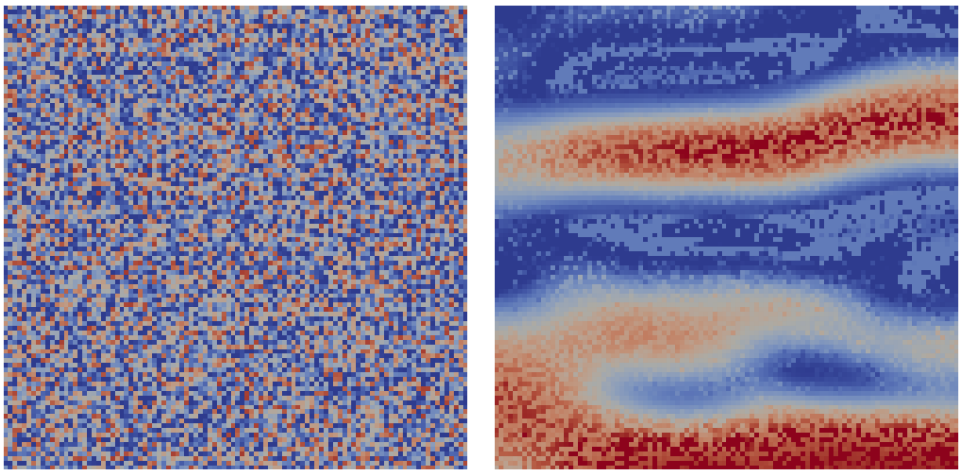
$$c = \int_{\Gamma} (-k\nabla T) \cdot d\vec{A} \qquad \Gamma = \text{right boundary} \tag{2.38}$$
$$\frac{\mathscr{V}_1(\beta)}{\mathscr{V}_0} \leq 0.4$$

The maximum heat flow occurs in a scenario where the thermal resistance is the least. Therefore a geometry with minimum path length and maximum cross-sectional area is the optimal solution. A rectangular strip filled by the highly conductive material and stretching the entire length of the domain is one optimal solution. A collection of discretely spaced strips are equally valid solutions.

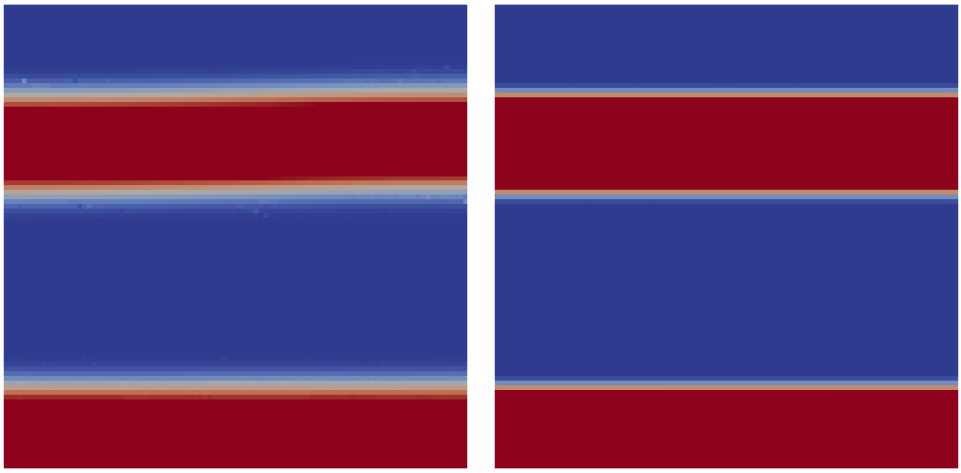
Figure 2.4 shows the process of topology optimization being performed on the described problem. The design domain in Figure 2.4 is made of a mesh of 100 X 100 quadrilateral cells. To demonstrate the robustness of the algorithm and its insensitivity to the initial distribution of the material, we choose a random distribution of the initial $\boldsymbol{\beta}$ field. Figure 2.4(a)-(d) shows the process of topology optimization and steps in the evolution of the random distribution of $\boldsymbol{\beta}$ to the optimal rectangular strip topology. The red region represents the high conducting material while the blue region represents the low conducting material. Results are shown for $k_1/k_2 = 10$.

As mentioned before, various other geometries are possible as solution to this problem. One such solution is shown in Figure 2.5(a) which was arrived with a different initial random distribution of $\boldsymbol{\beta}$. Both geometries Figure 2.4 (d) and Figure 2.5(a) have same value of the final objective function as expected. Figure 2.5(b) shows the temperature distribution of all the final geometries obtained.

Figure 2.6 shows plots of normalized objective function and volume fraction of the high-conductivity material plotted at each iteration of the optimization process. We also plot the filter radius normalized with the domain length for each iteration. As was mentioned before, the neighborhood area on which the filtering is performed is gradually reduced during the optimization process. We observe that the volume fraction constraint is active (satisfied in its equality) as the optimization process reaches its conclusion. The maximum cost function can be calculated analytically for this problem and is used to

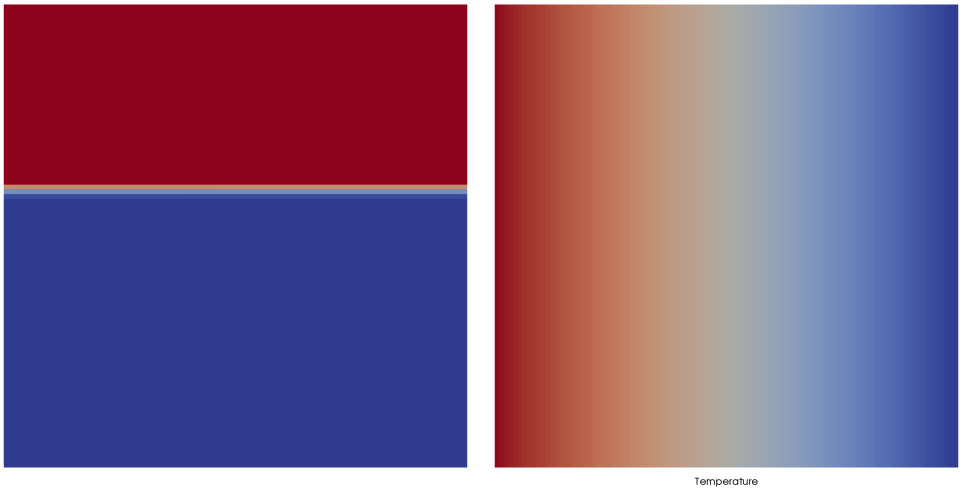


(a) (b)



(c) (d)

Figure 2.4: Topology optimization in test case 1. (a) Initial random distribution of $\boldsymbol{\beta}$ (b) & (c) Representative steps leading to optimized geometry. (d) One possible optimal topology.



(a) (b)

Figure 2.5: (a) Another possible optimal geometry obtained with a different initial random distribution of $\boldsymbol{\beta}$. (b) Linear temperature distribution in the final geometry.

normalize the cost function during each iteration. The normalized cost function can be seen approaching the maximum possible value. The evolution of geometry in Figure 2.4(a)-(d) is marked on the cost function plot in Figure 2.6. It is seen that an approximate geometry is evolved quickly in the first twenty iterations, and a slower evolution to the final optimal solution then occurs.

A 3D version of the same problem is now considered, as shown in Figure 2.7. The cuboidal design space is made up of a mesh of 50X50X50 hexahedral cells. Here again, two chosen opposite faces are given-temperature boundaries while all the remaining faces are subject to symmetry boundary conditions. The volume fraction of the highly conducting material is set at 40% of the design space. Figure 2.7 shows the final geometry obtained with a random

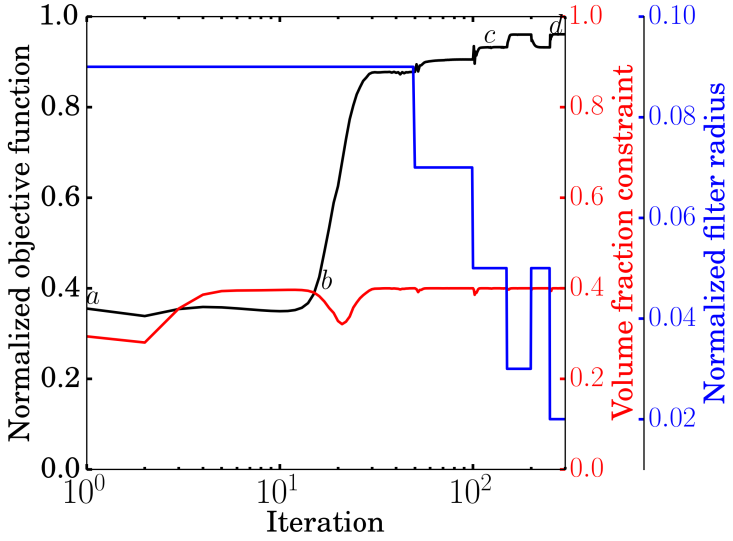
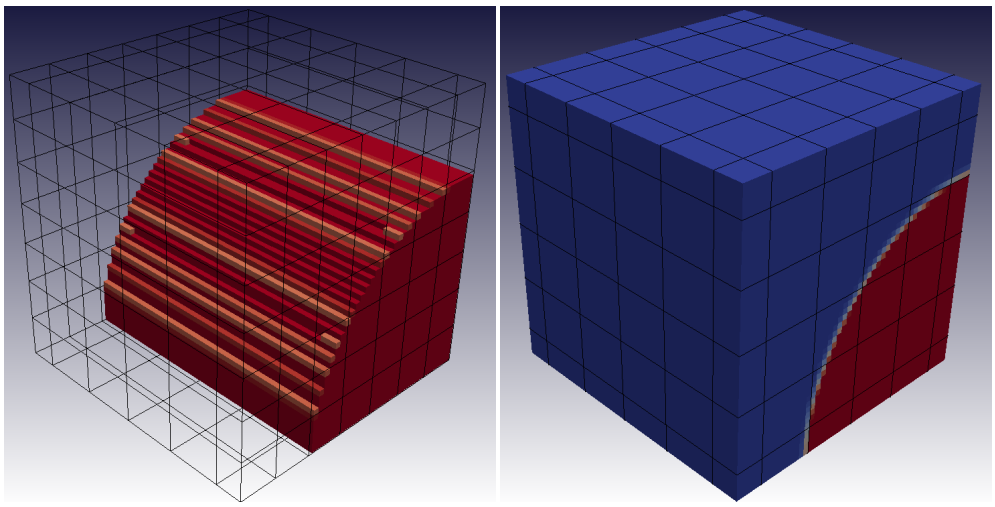


Figure 2.6: Normalized objective function and volume fraction of high-conductivity material versus iteration number. The plot also shows the normalized filter radius applied at various phases of the topology optimization.

initial distribution. Though many optimal solutions are possible, the final geometry must have a constant cross-section spanning the two given-temperature boundaries shown. This is depicted in in Figure 2.7(a) where only the high conducting material is shown. Figure 2.7(b) shows both materials for the same final geometry.

We contrast the computational cost for 2D and 3D topology optimization for same mesh resolution of the design space in Table 2.1. Keeping all parameters the same, on a Xeon E5-2680 processor, this particular test case on a 50X50X50 3D mesh took around 1337 seconds while that on a 50X50 2D mesh took around 35.02 seconds, i.e., roughly 38 times more. The 3D mesh size is larger than the 2D mesh by a factor of 50. The cost of the corresponding



(a) (b)

Figure 2.7: (a) An optimal topology for the 3D version of the problem in test case1 with a volume fraction of 0.4. Note that the cross-section is constant in the heat transfer direction. Only one material is shown to demonstrate that the cross section remains the same across the width (b) The same geometry with both materials.

Table 2.1: Comparison of 2D and 3D topology optimization.

Operation	2D (secs)	3D (secs)	Scaling (3D/2D)
Finite Vol. Soln.	6.27	392.89	62.62
Senstivity	18.66	830.93	44.54
Filtering	0.23	53.28	231.25
Optimizer	9.86	59.51	6.04
Total	35.02	1336.61	38.17

finite volume solution scaled by a factor of 63, sensitivity calculations using the adjoint method by a factor of 45, and the filtering operation by a factor of 231. On the other hand, the optimizer scaled by a factor of 6, very much lower than the mesh size scaling. The vast majority of the time spent is on the computation of sensitivity derivatives, though the finite volume solution also takes significant time in 3D.

2.6.2 Test case 2: Conduction with heat generation

In this problem we consider a 2D design space containing two materials k_1 and $k_2$with $k_1 > k_2$. In addition, a volumetric heat source term is included in Eq. 2.1. Three sub-problems are considered, and are outlined in Table 2.2.

2.6.2.1 Test case 2a

The schematic for test case 2a is shown in Figure 2.8(a). A rectangular domain with two materials is considered, with k_1 greater than k_2. Part of the left boundary near the center is maintained at a temperature T_1 while keeping the remaining boundaries adiabatic. A constant thermal energy source term

Table 2.2: Summary of three sub-problems in test case 3.

Test Case 3	Description	Modeling Source term	Conductivity ratio	Objective function; Minimize
(a)	Constant source throughout the domain; independent of design variable	$S = \alpha$	$\frac{k_1}{k_2} = 10^5, 25, 5$	Average temperature of the domain
(b)	Source term only present in the high-conducting region; dependent on the design variable	$S = \alpha \cdot \beta$	$\frac{k_1}{k_2} = 10^5$	Center point temperature
(c)	Source term only present in low-conducting material; dependent on the design variable	$S = \alpha \cdot (1 - \beta)$	$\frac{k_1}{k_2} = 10$	Center point temperature

$S = \alpha$ is applied throughout the domain. The objective is to minimize the average temperature of the domain, with the volume fraction of material 1 held at $\varepsilon = 0.4$. The objective function in this case is given by

$$c = \frac{1}{\mathscr{V}} \int_{\Omega} T d\mathscr{V} = \sum_{i=1}^{n} \frac{T_i}{n} \tag{2.39}$$

$$\frac{\mathscr{V}_1(\beta)}{\mathscr{V}_0} \leq 0.4$$

Three different conductivity ratios are investigated viz., $k_1/k_2 = 10^5, 25$ and 5. A Cartesian mesh of 100X100 quadrilateral cells is used.

The optimal topology for the case $k_1/k_2 = 10^5$ is shown in Figure 2.8, and has been published previously in [23, 25]. The fractal-like structure maximizes the surface area of heat-conducting material exposed to the heat-generating material, and the thick 'stem' of the fractal structure minimizes the heat transfer resistance to cold sink.

The optimal topology for the finite conductivity ratio $k_1/k_2 = 25$ is shown in Figure 2.8(c). We note that the branches of the geometry have reduced in number compared with Figure 2.8(b). Figure 2.8(d) shows an even smaller number of branches for $k_1/k_2 = 5$. Though more interpenetrating paths increase the reach of the conducting material into a larger volume of the heat generating region, they also increases the resistance to heat flow due to increase in the conduction path length and decrease in the cross-sectional areas. For the optimal structure in Figure 2.8(b), a fractal interpenetrating structure is essential since the blue regions are virtually non-conducting. However,

as observed in Figure 2.8(c) and (d) for finite conductivities, the algorithm balances the surface area exposed to the heat source with the increase in conduction path length. The effect is most pronounced in Figure 2.8(b) where material 1 displays a much less 'fingered' structure; since material 2 is sufficiently conducting it also offers a low-resistance path to the sink and deeply interpenetrating structures are no longer required.

2.6.2.2 Test case 2b

The schematic for this problem is shown in Figure 2.9(a). A source exists only in regions filled with material 1. Since β represents the element-wise volume fraction of high conducting material of each finite volume cell, the source term in each cell can be modeled as $S = \alpha \cdot \beta$. The goal of the design is to minimize the maximum temperature, which occurs at the domain center.

$$\begin{aligned} &\min\ c = T(\mathbf{0}) \\ &\frac{\mathscr{V}_1(\beta)}{\mathscr{V}_0} \leq 0.5 \end{aligned} \tag{2.40}$$

Both 2D and 3D cases are considered. For the 2D case, a quadrilateral mesh of 99 X 99 cells is used. For the 3D case, a hexahedral mesh of 49 X 49 X 49 cells is used.

The optimal geometry computed for this problem for a specified maximum volume fraction $\varepsilon = 0.5$ is shown in Figure 2.9(b). A uniform initial distribution β is used. Hourglass patterns are obtained as shown, with the

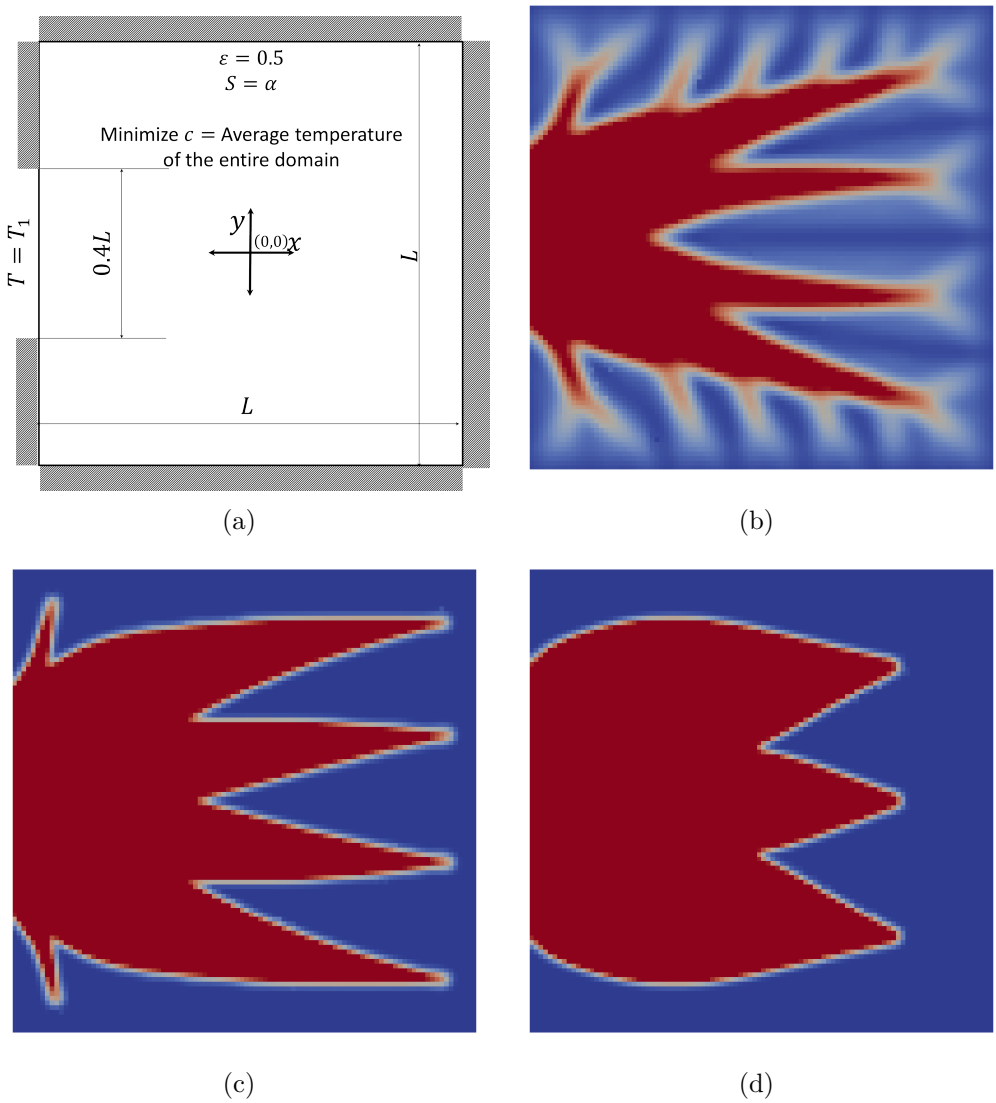


Figure 2.8: Optimal heat dissipating structure for test case 2a for three conductivity ratios (a) $k_1 \gg k_2$ (b) $k_1/k_2 = 25$ (c) $k_1/k_2 = 5$.

regions near the boundaries having maximum contact with the cold thermal boundary condition. Similar hourglass patterns are obtained for other volume fractions as well.

The 3D counterpart of the test case 2b where the initial design space is a cube has its final topology shown in Figure 2.9(c). The topology has symmetric hour-glass form and is very similar to the 2D topologies in Figure 2.9(b), except that it has six legs in 3D.

2.6.2.3 Test case 2c

The schematic for this test case is the same as shown in Figure 2.9 (a). Here, the source term is modeled as $S = \alpha \cdot (1 - \beta)$ so that heat generation is confined to the low-conducting phase. Similar to the previous case, the objective is to minimize the center point temperature (Eq. 2.40). Figure 2.9(d) shows the optimal topology for $\varepsilon = 0.5$. We note the difference in the final geometries in Figure 2.9(b) and (d). In the former case, the algorithm maximizes the contact area of the high conducting material (which contains the source) with the cold boundary. In the latter case, the algorithm not only increases the contact area between the low-conducting material (which contains the source) and the cold boundary, but also provides pathways to the boundary through the high-conducting material. This is because heat generated in the low-conducting material can find a path to the cold boundary though the high-conducting path as well.

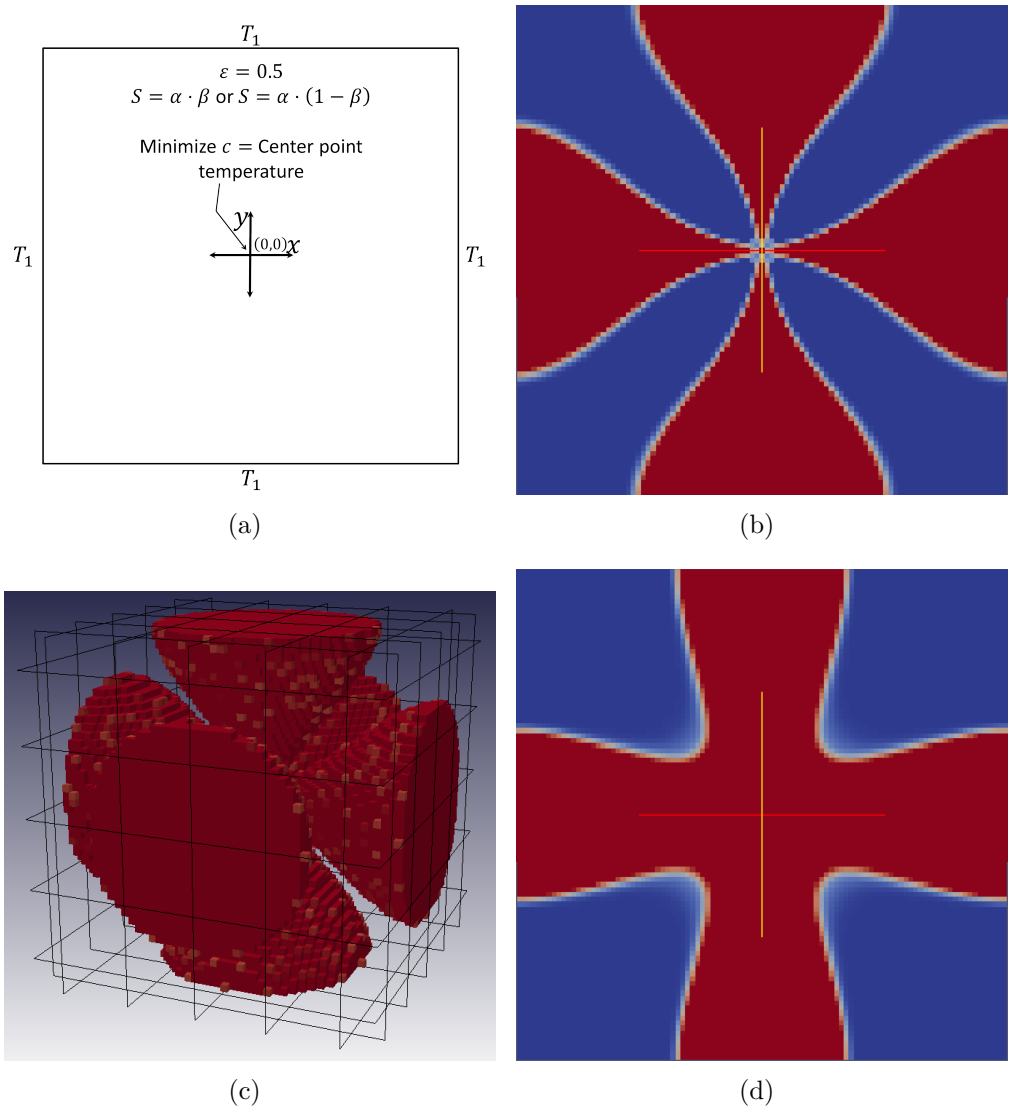


Figure 2.9: Topology optimization in test case 2b and 2c (a) Schematic diagram of the both the test cases (b) Optimal topology for $\varepsilon = 0.5$ for 2D for test case 2b (c) Optimal topology for the same volume fraction in 3D for test case 2b, where the design space is a cube (d) Optimal topology for $\varepsilon = 0.5$ for 2D for test case 2c.

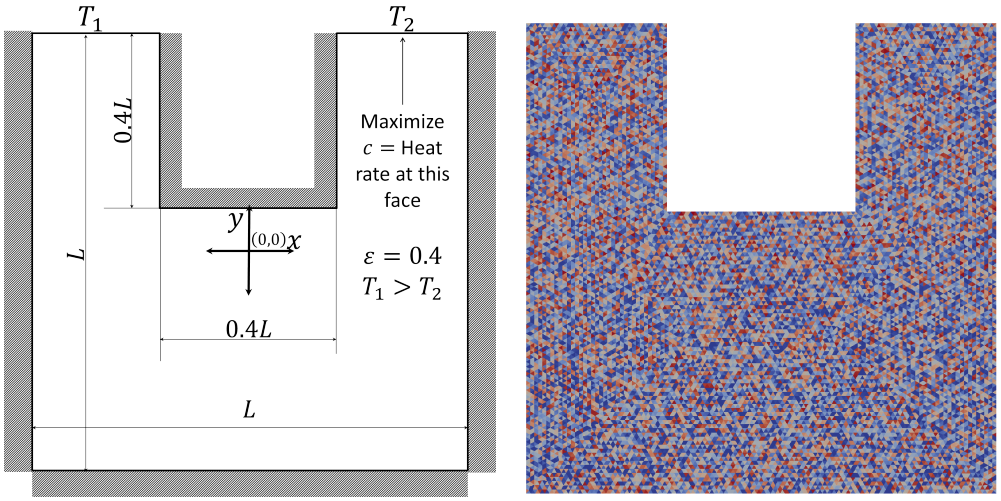
2.6.3 Test case 3: Comparison of structured and unstructured meshes

In this test case, we consider a problem in 2D, with a mix of temperature and symmetry boundary conditions in a non-rectangular initial design space. We consider both unstructured triangular meshes and structured Cartesian meshes, and compare the final optimal geometry obtained. As was mentioned before, there is no contribution of the secondary gradient flux (Eq. 2.6) to the residual of a cell for Cartesian meshes. However there is a contribution from the secondary gradient to the residual of a cell for non-orthogonal meshes. Therefore we intend to investigate the effect of the secondary gradient term in the sensitivity calculation on the final optimal topologies in this test case. As mentioned in Section 2.5.4, we develop the infrastructure ($\mathcal{R}$apid Automatic Differentiation Library) for computing derivatives of secondary gradient flux terms in Chapter 3. However we use $\mathcal{R}$apid in this test case to investigate the influence of secondary gradient flux term on final topologies.

Figure 2.10(a) describes the problem. The objective is to maximize the heat rate on the top right edge.

$$c = \int_{\Gamma} (-k\nabla T) \cdot d\vec{A} \qquad \Gamma = \text{top right boundary} \tag{2.41}$$
$$\frac{\mathscr{V}_1(\beta)}{\mathscr{V}_0} \leq 0.4$$

Thermal conductance is directly proportional to cross sectional area while inversely proportional to length of the path and the final solution should



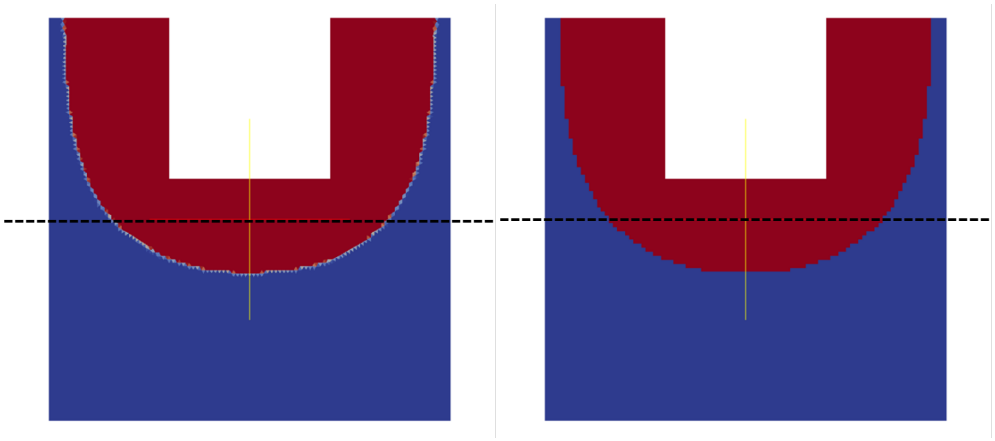
(a) (b)

Figure 2.10: Schematic for problem solved in test case 3.

reflect this balance. The design space is constrained to be filled with a maximum volume fraction $\varepsilon = 0.4$ of the high conducting material. The conductivity ratio considered is $k_1/k_2 = 10$. The non-rectangular region is discretized with high-quality triangular elements as is shown in Figure 2.10(b). Here in the figure the triangular elements are populated with random values for as its initial distribution.

Figure 2.11 (a) shows the final optimized topology for the problem on such an unstructured mesh of 19346 triangular cells. The high-conducting material is seen to be deployed in a continuous band connecting the hot and cold boundaries, as expected.

The same problem is now solved on the Cartesian mesh with 8400 quadrilateral cells. For purposes of comparison, both the meshes have been



(a) (b)

Figure 2.11: Optimal topology for volume fractions $\varepsilon = 0.4$ for test case 3 obtained using (a) an unstructured mesh, and (b) a Cartesian mesh.

matched to each other in terms of cell length scale. The final topology for Cartesian meshed domain is shown in Figure 2.11 (b). Notice that the topologies obtained using unstructured and Cartesian meshes are visually nearly identical.

We now make more quantitative comparisons. Figure 2.12(a) depicts the evolution of the objective function and volume fraction of the high conducting material with iteration. The objective function is normalized with the finally-achieved value. We observe that the curves for the two meshes almost fall on top of each other. In Figure 2.12(b), we also plot the temperature along the horizontal axis for the final topologies in Figures 2.11(a) and (b). The temperature is normalized with the difference in the boundary temperatures and the axial distance with the size of the design domain. The

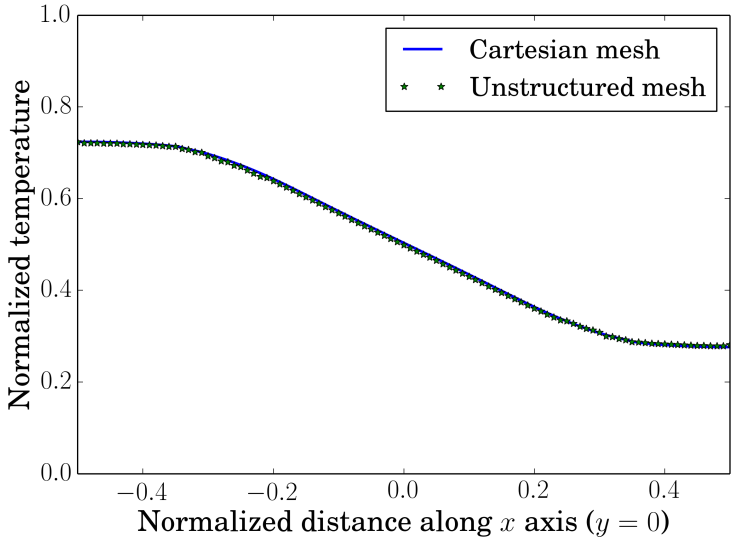


Figure 2.12: Schematic for problem solved in test case in Section 2.6.3.

temperature distribution is also found to be near identical.

These quantitative measures establish that secondary gradients do not significantly influence the sensitivity computations thereby not affecting the final optimal solution if the unstructured mesh is of high-enough quality. Or in other words, we can neglect the secondary gradient flux for pure heat conduction problems for good-quality meshes.

2.7 Closure

In this chapter, topology optimization for heat conduction problems has been explored within a unstructured finite volume framework based on a residual formulation. Sensitivities are calculated employing the adjoint method and MMA is used as the optimization algorithm. Secondary gradients, which

appear in the discretization for unstructured meshes, are accounted for in the computation of the temperature field, but not in the calculation of sensitivity gradients. Topology optimization is demonstrated on a number of heat conduction problems and the optimal geometries so obtained are rationalized based on physical arguments. Our computations demonstrate that secondary gradients for unstructured meshes may be neglected for high quality meshes for pure conduction problems. The various test problems presented in this chapter demonstrate the ability of topology optimization in obtaining non-intuitive geometry designs.

A number of challenges remain in generalizing the unstructured mesh topology optimization method presented here for general fluid flow and convective heat transfer. As mentioned earlier, among the most popular methods for the solution of incompressible flows are sequential pressure-based algorithms like SIMPLE[32] which do not assemble complete Jacobians, making the computation of sensitivity derivatives more complex. Furthermore, nearly all published topology optimization work thus far has focused on staggered mesh implementations of SIMPLE, whereas the most widely-used codes employ non-staggered or co-located pressure-velocity formulations[31]. Indeed, for unstructured meshes, staggered formulations are difficult to formulate and have rarely been used. Sensitivity derivative calculations for momentum-interpolation schemes such as that described in are cumbersome, and alternative techniques which yield optimal or near-optimal results are necessary. Finally, nearly all published papers on topology optimization for fluid flow have

addressed laminar flows, while most flows of industrial importance are turbulent. But if these challenges can be addressed, finite-volume based topology optimization can emerge as a powerful tool for the generation of preliminary designs for industrial problems.

The biggest impediment to such an extension is the process of computing sensitivities using the adjoint method. We build an infrastructure named $\mathcal{R}$apid, an automatic differentiation library, as the first step to obtaining these sensitivities. We describe the implementation of this library in the next chapter.

Chapter 3

$\mathcal{R}$esidual Automatic PartIal Differentiator ($\mathcal{R}$APID)

"Everything should be made as simple as possible, but no simpler." - Einstein

The goal of this chapter is to present the development of the automatic differentiation tool, christened $\mathcal{R}$apid. In the process of presenting the methodology, we touch upon the basics of automatic differentiation and present a brief survey of the current state of automatic differentiation that has motivated us in building our tool. Throughout the chapter, we use simple examples designed to elucidate the necessary concepts.

3.1 Introduction to Automatic Differentiation (AD)

Automatic differentiation (AD), also referred to in the literature as algorithmic differentiation, code differentiation and computational differentiation, has the potential to become a very powerful component of computational science and engineering if utilized to its full potential. Function derivatives (or sensitivities) are necessary for many applications such as design optimization, sensitivity analysis, inverse problems, uncertainty quantification, ma-

chine learning as well as elementary applications such as solution of algebraic equations and curve fitting [67]. However one can encounter very complex mathematical functions, for instance, the source terms in the Spalart-Allmaras turbulence model which is of interest in this dissertation. Manual differentiation and coding of the derivatives of complex terms is prone to human error. AD tools can automatically and efficiently compute derivatives of functions accurately to machine precision. If used judiciously, AD can eliminate hard labor and human errors in differentiation, and can help to obtain derivatives of already programmed codes non-intrusively. Many excellent resources (http://www.autodiff.org) and books are available in the area of AD [67, 68].

AD facilitates the computation of derivatives of any mathematical function expressed using a computer program. The programmer only needs to write the program for function evaluation at specified parameter values, AD tools can evaluate the derivative of the same function at the same specified values automatically. AD does not perform symbolic computation of derivatives (as computer algebra packages like Mathematica or Maple do). It does not determine derivative functions but only evaluates the derivative values. At the same time, AD does not evaluate derivatives using finite differencing of derivatives and therefore does not carry the burden of truncation errors. AD operates by systematic repeated application of the chain rule of differentiation to obtain actual numerical values of algebraic expressions and their derivatives [67]. AD exploits the semantics of a programming language to evaluate a mathematical expression and employs well-defined unique rules of differential calculus to

evaluate derivatives.

Methods to perform AD have evolved over the years. When procedural programming languages like Fortran were popular, 'source transformation' AD tools were developed. Here, the AD tool parses a given source code written to compute a mathematical function, and generates another source code to compute the derivative of the function with respect to specified independent variables. ADIFOR [69] and TAF [70] are some of the source transformation AD tools available.

A shift from procedural to object oriented programming languages like C++ led to new approaches [71, 72]. Features of C++ like operator overloading and template programming naturally made it easier to implement AD in a more direct fashion. Many AD tools have been developed in C++ in recent years. FADBAD [73], Sacado[74, 75], ADEPT [72], CppAD [76] are some of the AD tools using operator overloading. Since $\mathcal{R}$apid is based on operator overloading, we discuss the concept more deeply about it in a little detail in the next section.

3.2 AD based on templating and operator overloading

'*Templating*' is a feature available in C++ that allows the user to decide the *type* of a variable at compile time. Consider the following code snippet, where the programmer has defined variables *x,y* and *z* of a generic *type T*.

```
1 T x,y,z;
2 .
```

```
3 .
4 z=x+y;
```

If T is of type *double*, the usual addition operation is performed. If T is some user defined *type*, the compiler can be forced to execute some other user defined operations, called overloaded functions, for example, for the '+' operator. Thus by specifying a generic *type 'T'*, the user can avoid specifying the actual *type* of the variables while programming. The user can later decide at compile time what *type 'T'* should be. Templating can be very useful for code re-usability.

Operator overloading feature incorporated in C++[77] allows the user to customize the functionality of operators based on the user-defined *type* of the operands. For instance, in the computation of the expression $x + y$, the operator '+' computes the usual sum of its operands x and y if these variables are of the basic data type, say '*double*'. However, if the variables are of some other user-defined *type*, then the same operator '+' can be made to perform another user defined functionality acting on the operands. Thinking from an AD perspective, if the new *type* is defined by the user to have two fields, one for storing function value and another for storing derivative value, the operator '+' can be re-implemented not only to compute the sum $x + y$ and store it in the first field, but also to compute the derivative $dx + dy$ and store it in the second field. Similarly, the multiplication operator '*' can be forced to simultaneously compute $x * y$ that is stored in the first field and to compute $xdy + ydx$ and store it in the second field. AD tools based on operator overloading are built

based on this concept.

This combination of operator overloading and templating facilitates building of powerful unintrusive AD tools for obtaining derivatives and thus greatly enhances code re-usability.

There are two modes of AD - forward mode and reverse mode [67, 78]. Both have their own advantages and disadvantages. Both forward and reverse mode AD tools have been implemented using source transformation and operator overloading/template programming techniques. We primarily discuss the two modes with the latter.

3.2.1 Forward mode

In forward mode AD, derivatives propagate along the flow of computation from the beginning to the end until the final output variable is computed. We illustrate this statement with simple examples after defining some notation.

Suppose $X = \{x_1, x_2 \dots x_n\}$ is the set of n independent variables and $F = F(X) = \{f_1(X), f_2(X) \dots f_m(X)\}$ be a set of m dependent functions each defined over X. Further, let function f_i be decomposed into k (varying) elementary steps as depicted in 3.1(a). As an example if we define two functions f_1 and f_2 as,

$$\begin{aligned} f_1(x_1, x_2) &= \exp(x_1 x_2 + \sin(x_1)) \\ f_2(x_1, x_2) &= x_1^2 + x_2^2 \end{aligned} \tag{3.1}$$

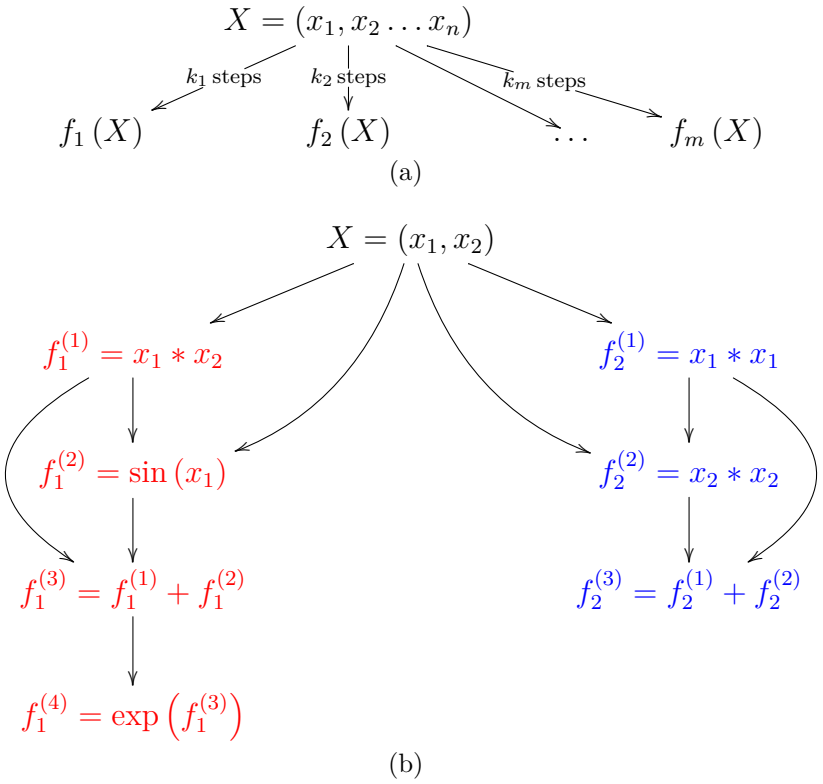


Figure 3.1: (a) Breaking down the evaluation of function $f_i(X)$ into k_i elementary function evaluations. (b) Illustration of elementary functions $f_i^{(j)}$ for two example functions.

then Figure 3.1(b) shows the series of 4 elementary steps needed to compute f_1 and 3 elementary steps to compute f_2 from the independent variables $X = (x_1, x_2)$.

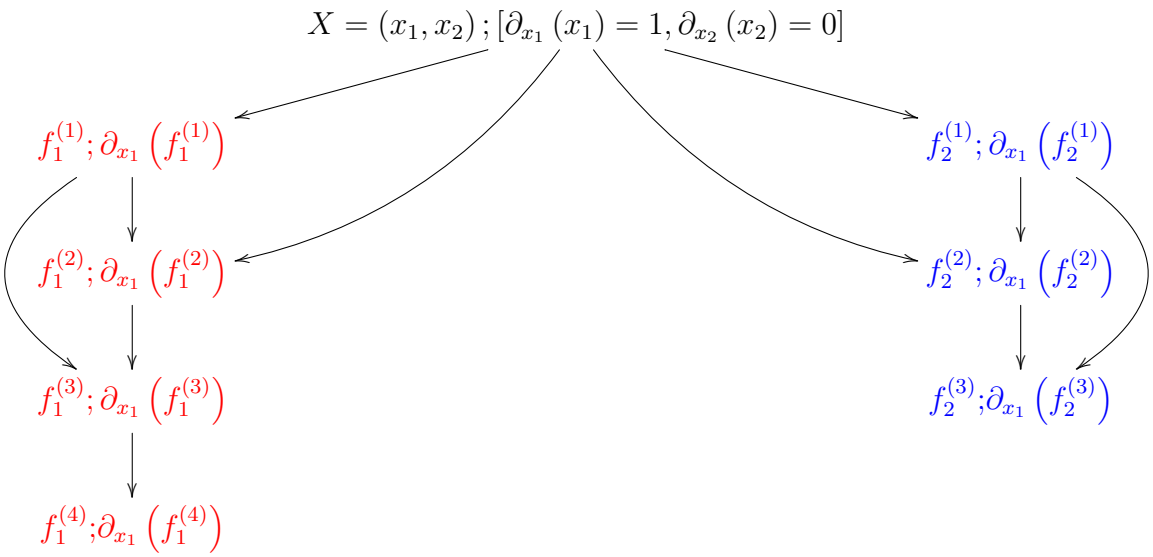
In a forward mode AD, the user first chooses one of the independent variables in the set X with which the derivatives of the each dependent functions f_i need to be computed. Henceforth, for each output functions f_i, derivatives with respect to the chosen independent variable are computed along with

each elementary function along the tree. For the two examples in Eq. 3.1, suppose one chooses x_1 to be the independent variable, then the derivatives $w.r.t$ x_1 propagate in the flow of computation at each elementary function evaluation of f_1 and f_2 as shown in the Figure 3.2(a). A second sweep all the way from the beginning by setting x_2 as the second independent variable will be required to compute the derivative of functions f_1 and f_2 $w.r.t$ x_2 shown in Figure 3.2(b). The independent variable x_1 is signalled by denoting $\partial_{x_1}(x_1) = 1; \partial_{x_1}(x_2) = 0$ and x_2 by $\partial_{x_2}(x_1) = 0; \partial_{x_2}(x_2) = 1$ as seen in Figure 3.2. The rationale behind such a construct will become clear in the forthcoming sections.

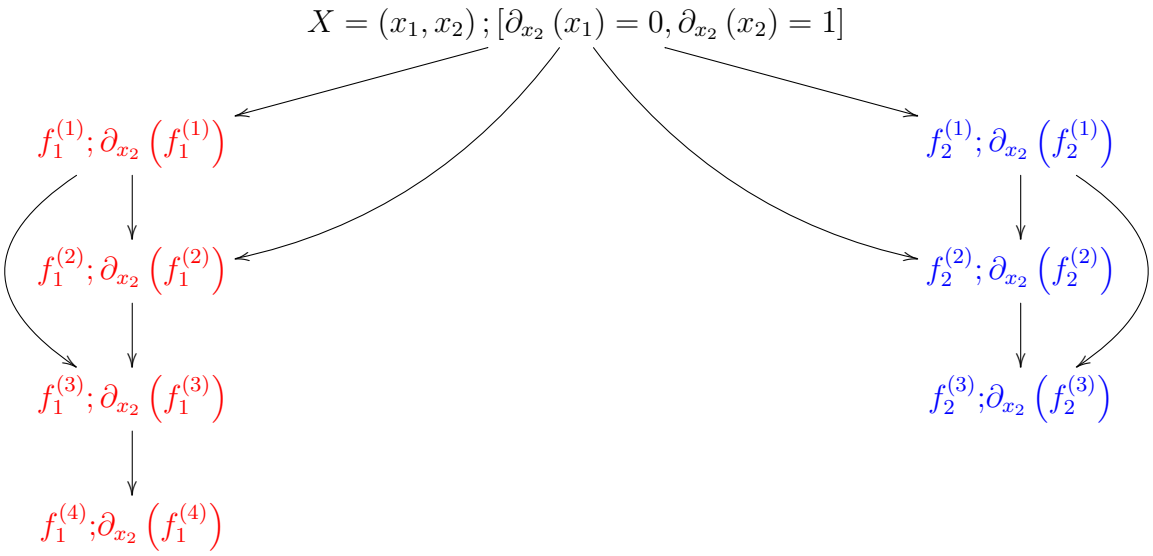
3.2.2 Reverse mode

Reverse mode AD propagates adjoints[67], in contrast to the forward mode, which propagates derivatives. Our goal is to understand the reverse mode just enough to distinguish the features of the $\mathcal{R}$apid AD infrastructure that we have developed. In contrast to the forward mode, the reverse mode derivative computation starts from the output variable and propagates backward in the reverse flow of computation.

We choose the same example as in Figure 3.1 to illustrate reverse mode AD. In Figure 3.3, the elementary functions are computed along the forward direction marked with letter 'f' like the forward mode computation. Thus all dependent functions F are computed. Unlike the forward mode, we do not choose the independent variable in the beginning and no derivatives propogate along forward sweep. However the history of the operations and associ-



(a) Function evaluation tree of f_1 and f_2 and their derivatives when x_1 is set as independent variable.

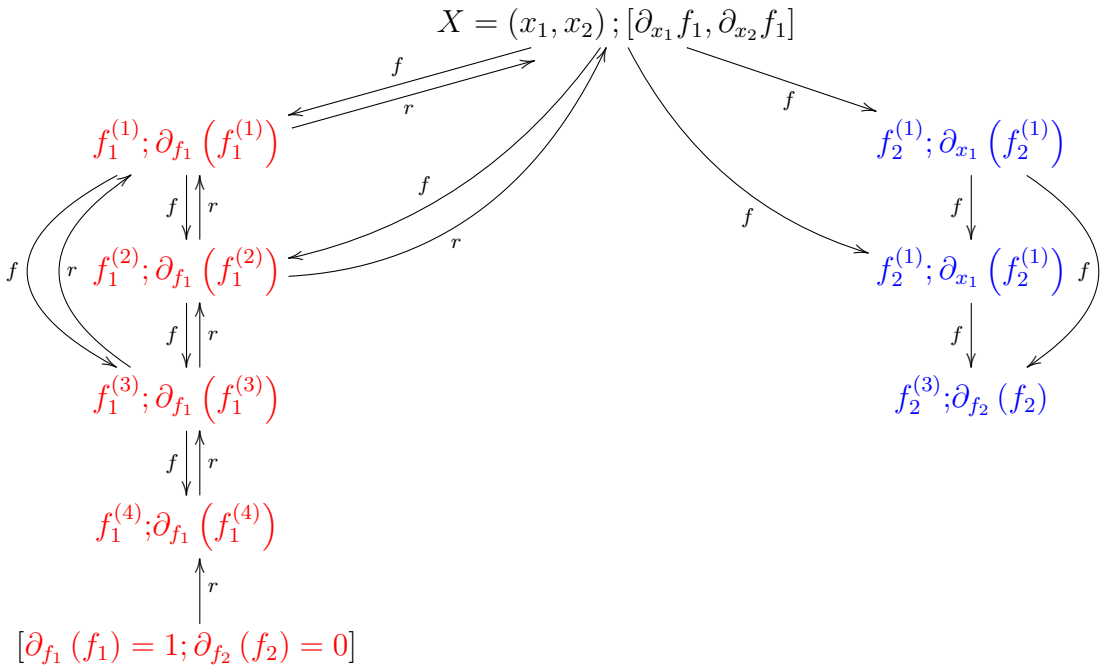


(b) Evaluation tree of f_1 and f_2 and their derivatives when x_2 is set as independent variable.

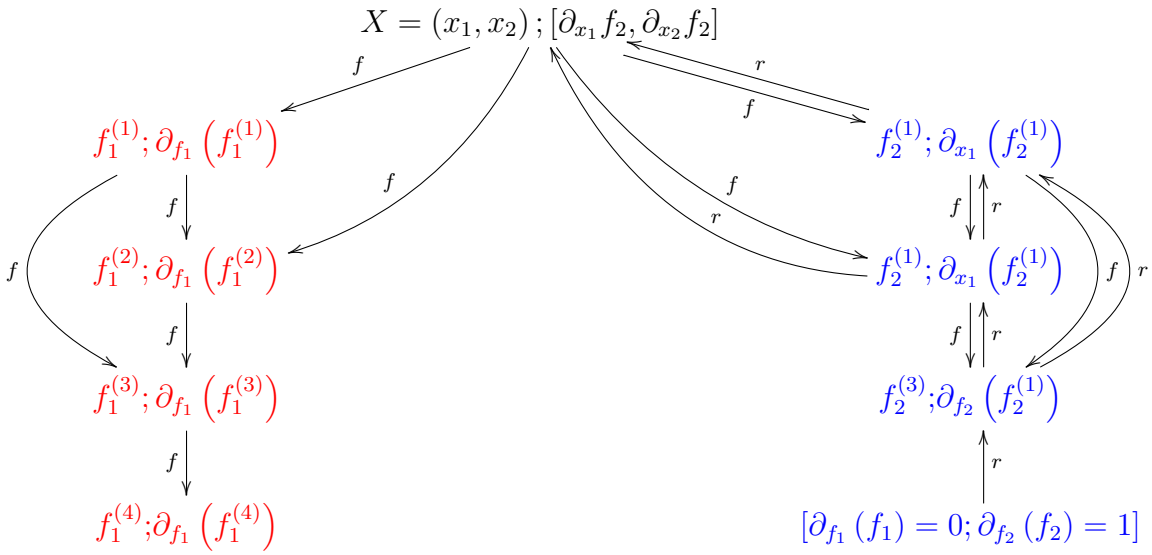
Figure 3.2: Derivative propagation of a forward mode AD during each step of the elementary function evaluation.

ated operands are recorded in the memory. Once the dependent functions are computed and a trace recorded, the user chooses one dependent function (or output variable) with respect to which all its derivatives are required. Once the output variable is selected, the reverse mode AD starts sweeping back up the evaluation tree that is depicted with the arrow marked 'r' in the Figure 3.3. In Figure 3.3(a) f_1 is chosen as the output variable while f_2 is chosen in Figure 3.3(b). At each re-tracing step of the stack, the sensitivity of a corresponding elementary function with respect to the chosen output variable is computed. Finally when one reaches the beginning of the stack, the sensitivity of all the input variables with respect to the chosen output variable is computed. This also implies that the sensitivity of the chosen output variable with respect to each input independent variable is in place at the end of retrace.

We now contrast the differences between forward mode and reverse mode AD. Forward and reverse modes are two extreme cases of doing AD to compute derivatives. Generally, for m output functions $F = \{f_1, f_2 \dots f_m\}$, a forward mode AD computes derivatives of all the functions f_i with respect to the chosen independent variable x_α in one single sweep. For each function computation f_i, derivatives of a chosen variable x_α propagates along each of its k_i elementary steps. Thus a single sweep produces the derivatives of all functions f_i with respect to x_α. It is noted that n such sweeps would be required to compute derivatives of the output functions with respect to each input independent variable x_i. Forward mode AD is therefore preferable when $m \gg n$.



(a) Propogation of adjoints when f_1 is set as independent output variable.



(b) Propagation of adjoints when f_2 is set as independent output variable.

Figure 3.3: Derivative propagation of a reverse mode AD during each step of the elementary function evaluation.

In a reverse mode AD, all the derivatives of a chosen dependent function f_i with respect to all the independent variables are obtained in a single reverse sweep. However m such reverse sweeps need to be performed if the derivatives of all the output variables with respect to all the independent variables are required to be computed. There are many practical situations when the number of input variables is much greater than the number of output variables of interest. Reverse mode AD is attractive in such cases. Generally if $n \gg m$, reverse mode AD is the choice. However the implementation of reverse mode AD is more complicated than forward mode AD because of the necessity of keeping the forward trace in memory.

With regarding to the speed of AD for either mode, when the the number of elementary function evaluations of a function (k_i in Figure 3.1(a)) becomes large, AD generally becomes slow due to the overhead of creating and destroying many temporary objects during each step of the computation. Expression templates[79], another advanced feature supported by C++, have been used to speed up AD. The forward mode AD library FAD/SACADO was accelerated with expression templates and illustrated with 20 input design variables in[80]. Further speed up by improving the use of expression templates was reported in[81]. Similarly reverse mode AD has been accelerated using expression templates in the library Adept[72].

3.3 Number of variables in topology optimization

We discuss the order of the number of independent and dependent variables in topology optimization problems. This will help us to choose the right mode of AD to use. In topology optimization, the number of design variables $\boldsymbol{\beta}$ is the number of elements N in the mesh, which is quite large (for discussions here we ignore ghost cells). We only compute one or two cost functions in topology optimization and therefore the number of output variables $m = \mathcal{O}(1)$. The number of independent input variables currently is $n = N$. The flow of the function evaluation c is shown in Figure 3.5.

Consider a thought experiment in such a setting. Solely looking at the number of sweeps required to compute all sensitivities $\frac{dc}{d\beta_i}$, using forward mode AD is very unattractive. At a first glance reverse mode AD looks promising since only few reverse swipes (m), are theoretically enough to obtain all sensitivities. However the process of solving the simultaneous linearized equations (or linear system) $\boldsymbol{\mathcal{R}}(\boldsymbol{\phi}, \boldsymbol{\beta}) = \mathbf{0}$, be it using iterative or direct methods, involves a tremendous number of elementary evaluations that would make both the modes infeasible for topology optimization.

One of the feasible ways to obtain sensitivities is by using the discrete adjoint method [56, 82]. The reader is warned that reverse mode AD is also known as adjoint mode AD in the literature [67]. The discrete adjoint method should not be confused with adjoint mode AD. As discussed in detail in Section 2.5.4, Chapter 2, the derivatives of all the residuals and cost functions with respect to both state variables and design variables are required to compute

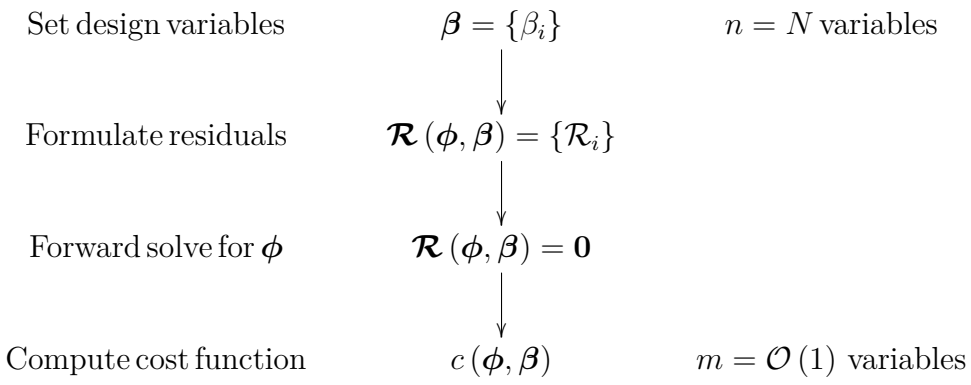


Figure 3.4: Order of the number of variables in topology optimization

the sensitivities $\frac{dc}{d\beta_i}$. If AD is used to compute the residuals and cost functions from the converged state variables and current design variables, the required derivatives can be automatically computed. As shown in Figure 3.5, the converged state variables and design variables are set as independent variables. If p is the total number of unknowns in the PDEs ($p = 1$ for pure conduction equations, $p = 4$ for momentum and continuity equations, $p = 5$ for a coupled laminar flow-thermal equations etc.), then there are $n = N \cdot (p + 1)$ input variables[1]. There are $p \cdot N$ residual equations and $\mathcal{O}(1)$ cost functions making the total output variables to $m \sim p \cdot N$. We see that n and m are almost of the same order. Neither forward mode nor reverse mode AD to compute the necessary derivatives required for discrete adjoint method would deliver superior advantage in terms of computational cost of AD.

It is appropriate here to briefly survey of the use of AD specific to the

[1]If we account for ghost cells, we have N design variables, and a little more than $N \cdot p$ state variables. This is because we associate state variables of ghost cells, but not for design variables. See Section 3.4.

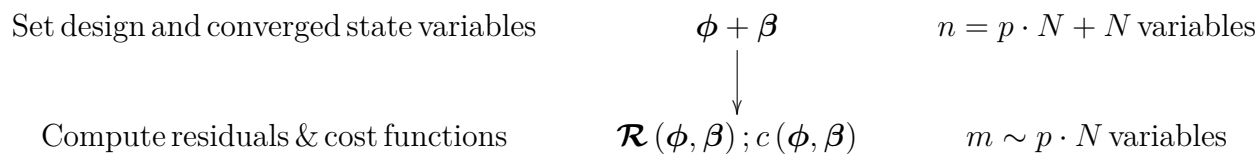


Figure 3.5: Order of the number of variables in topology optimization with adjoint method

applications of interest involving many input variables. One of them is the large scale sparse Jacobian computation that arises in many scientific applications. Averick *et al.* used AD to compute sparse Jacobians matrices using ADIFOR. Using graph coloring technique and AD, Coleman *et al.* [83] computed sparse Jacobians. AD has been introduced into the finite element PDE solver FEniCS to be used for a variety of purposes [84].

Sensitivity computation using adjoint methods is very popular in control theory and optimization where the number of optimization variables is large. The computation of sensitivities using the discrete adjoint method using AD for various applications in transport problems has picked up momentum in the last decade and in an active area of research. A flow control problem to minimize the drag on a cylinder by controling the parameters (twenty) of the boundary condition was implemented by Pierre *et al.* [80]. Aerodynamic shape optimization of aerofoils with an adjoint method was performed using AD by Gauger *et al.* [85]. An extension of similar work was implemented in the framework of the open-source finite volume suite SU2 [86]. Towara *et al.* [82, 87] implemented a discrete adjoint version of the Navier-Stokes equations in the open-source finite volume framework OpenFOAM[88] using the AD tool

dco/c++[89].

Regarding the choice of AD, a forward mode AD library is much easier to implement than a reverse mode one, as mentioned before. It is also our understanding that it is much easier to use the application (specifically a numerical PDE solver) with a forward mode AD as it requires minimal code intrusion. However, even with the existence of a wide variety of forward mode AD libraries like FAD, Sacado, CppAD, we felt the need to develop a new library keeping our end application and solution strategies in mind. Next we discuss the rationale for developing such a new AD library.

3.4 Need for a new AD library

By now it is clear that AD does not simplify expressions like the computer algebra packages. The efficiency of an AD tool cannot get better than the efficiency of the program written for the function evaluation. Though we would like to avoid code intrusion as discussed before, blind application of AD to existing code may not be advisable. An AD tool that takes advantage of the user's insight into the underlying structure of the program achieves best performance [67].

In our case, we use the finite volume framework to solve the governing PDEs in transport phenomena, following well-developed methodologies evolved over decades. On the other hand, topology optimization has certain characteristics in its algorithm. The $\mathcal{R}$apid library was developed keeping both methodologies in consideration to perform the intended tasks effectively. We

motivate more quantitatively the need for the development of $\mathcal{R}$apid with a simple example described below.

Consider the model Cartesian mesh shown in Figure 3.6 of dimensions 5X3 units. This model mesh will be used throughout this dissertation wherever necessary. The mesh has a total of 31 cells which includes 15 regular cells (red color coded) and 16 ghost cells (green color coded). The mesh contains a total of 38 faces with 22 internal faces that have cells on both sides and 16 boundary faces that have cells only on one side. We again consider the discretization of the pure diffusion equation (Eq. 2.1) modeling heat conduction. Since there are 31 cells in the mesh, there are 31 state variables T_i (temperature) and 15 design variables β_i (no design variables for ghost cells) summing to a total of 46 independent input variables.

First we consider the computation of the residual of the PDE for each cell $\mathcal{R}_i$. Out of the total set of variables, $\mathcal{R}_i$ of each cell depends only on few of the total variables. For instance, the residual $\mathcal{R}_7$ (excluding secondary gradient flux) is computed as follows,

$$\begin{aligned}
\mathcal{R}_7 &= \sum_{(i,f)} \frac{k^f}{\Delta\xi} \frac{\vec{A}^f \cdot \vec{A}^f}{\vec{A}^f \cdot \vec{e}^f} \left(T_i - T_7\right);\ (i,f) = \{(2,3),(8,7),(12,15),(6,4)\} \\
k^f &= \frac{k_7 k_i}{k_7 + k_i}\ i = 2,8,12,6 \\
k_i &= f\left(\beta_i\right)\ \ i = 7,2,8,12,6
\end{aligned} \tag{3.2}$$

Out of the 46 state and design variables, the residual $\mathcal{R}_7$ depends only

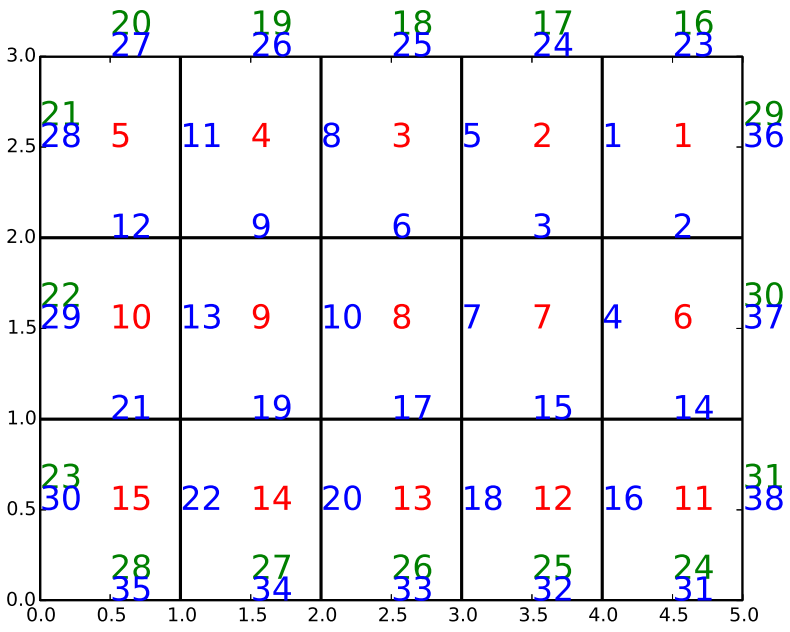


Figure 3.6: Model mesh

on 10 of the total variables given by

$$\mathcal{R}_7 = \mathcal{R}_7 \left(T_7, T_2, T_8, T_{12}, T_6, \beta_7, \beta_2, \beta_8, \beta_{12}, \beta_6 \right) \tag{3.3}$$

Only the partial derivatives of $\mathcal{R}_7$ with the above 10 variables need be computed and stored. Similarly the residual of a ghost cell $\mathcal{R}_{22}$ depends on three other variables from the total list given by,

$$\begin{aligned} \mathcal{R}_{22} &= \frac{k^{29}}{\Delta\xi} \frac{\vec{A}^{29} \cdot \vec{A}^{29}}{\vec{A}^{29} \cdot \vec{e}^{29}} \left(T_{10} - T_{22} \right) && (3.4) \\ &= \mathcal{R}_{22} \left(T_{22}, T_{10}, \beta_{10} \right) && (3.5) \end{aligned}$$

Each of the 31 residual values $\mathcal{R}_i$ depends on only few of the total state and design variables. This number can vary depending on connectivity.

Similarly, a cost function computation defined as the total heat rate on the right boundary becomes,

$$c = \sum_{(i,b)} -k \frac{(T_i - T_b)}{\Delta\xi}; \; (i,b) = \{(1,29), (6,30), (11,31)\} \tag{3.6}$$

For a heat conduction problem, the number of independent variables on which $\mathcal{R}_i$ depends would be of the similar order as the above examples however large the mesh size may be. A secondary gradient computation for an unstructured mesh (discussed in Section 2.4.2, Chapter 2) would bring dependencies of $\mathcal{R}_i$ on a few more neighbor variables.

In summary, one one hand, the total number of both independent (T_i, β_i) and depedent $(\mathcal{R}_i)$ variables are very large since they are propotional to the mesh size. However each of the dependent variable depends only on a few out of the total set of independent variables. As mentioned in Section 2.5.4, Chapter 2, Jacobians $\frac{\partial \mathcal{R}_i}{\partial T_j}, \frac{\partial \mathcal{R}_i}{\partial \beta_j}$ and cost function derivatives $\frac{\partial c}{\partial T_j}, \frac{\partial c}{\partial \beta_j}$ are required to compute sensitivities needed for topology optimization. The data structure for AD needs to be defined to compute these derivatives not only exploiting their sparsity, but also ease of population and post-manipulation. Motivated by an understanding the underlying structure of the problem being solved, we develop the $\mathcal{R}$apid library described in the next section. Efficiency, minimal code intrusion, generality, ease of implementation, and most importantly, capturing dependencies in the best possible way given a current function implementation, have guided the development. The last aspect will be understood better after the chapters on fluid flow (Chapter 4) is covered.

3.5 Implementation of $\mathcal{R}$apid library

The acronym $\mathcal{R}$apid stands for $\boldsymbol{\mathcal{R}}$esidual **A**utomatic **P**art**I**al **D**ifferentiator. The purpose of the library is to obtain the partial derivatives of PDE residuals with respect to both state and design variables through automatic differentiation, viz. in a rapid way. This section is devoted to the main aspects of the implementation of the $\mathcal{R}$apid library in C++.

A forward automatic differentiation library Tangent was built by the founding authors of MEMOSA and has been used extensively to perform un-

certainty quantification for various physical problems [90]. Subsequently, other researchers extended the Tangent library to obtain higher order derivatives of functions upto 4th order for applications in nano-scale heat transfer [91]. The *'Tangent* library' is very similar in implementation to various existing open-source forward mode AD packages, especially FAD [92]. $\mathcal{R}$apid was built based on Tangent but with significant differences in its implementation.

Memosa is a software suite that is being developed continuously. A forward automatic differentiation library '*Tangent'* was built by founding authors of Memosa and has been used extensively to perform Uncertainty Quantification of various physical models [90]. Subsequently, other researchers extended *'Tangent'* to obtain higher order derivatives of functions upto 4^{th} order for applications in nano-scale heat transfer [91]. Tangent is very similar in implementation to various open-source existing forward mode AD packages, especially FAD[92]. $\mathcal{R}$apid was built based on *Tangent* but with significant differences in its implementation.

3.5.1 Formal notations for building $\mathcal{R}$apid

We recall that $X = \{x_1, x_2, \ldots x_n\}$ represents the set of n independent variables and $F = \{f_1, f_2, \ldots f_m\}$ the set of functions (or dependent variables) with each function defined on the set X. Let the set X be partitioned into two subsets X_a and X_{na}, the former named as independent active set containing q variables and latter named as independent inactive set containing the remaining $n - q$ variables. X_a is named active, because we wish to compute the

partial derivatives of functions f_i with respect to variables in X_a using AD. We do not wish to compute derivatives of the function f_i with any variables in X_{na}. Constants can be classified into this category. Therefore, we intend to obtain

$$\mathcal{R}_{ij} = \frac{\partial f_i(X)}{\partial x_j}; \;\; x_j \in X_a \tag{3.7}$$

An important characteristic of the functions defined on F, as discussed in Section 3.4, is that any particular function f_i depends on only few independent variables of the set X_a. If the number of variables from set X on which f_i depend be r_i, then $\mathcal{O}(r_i) \ll p$. $\mathcal{R}_{ij}$ is thus a sparse second order tensor. We will exploit this nature when building the $\mathcal{R}$apid class for efficient computation of partial derivatives through AD.

A general mathematical function is a combination of various mathematical operators operating on operands. The most common operators implemented in any programming language[80] is the set $\mathcal{F} = \mathcal{F}_A \cup \mathcal{F}_M \cup \mathcal{F}_L$, the union of arithmetic operators $\mathcal{F}_A$, logical operators $\mathcal{F}_L$ and various individual mathematical functions $\mathcal{F}_M$, listed in Eq. 3.8.

$$\begin{aligned} \mathcal{F}_A &= \{+, -, *, /, +(unary), -(unary), ++, --, \dots\} \\ \mathcal{F}_M &= \{\text{pow}, \text{sqrt}, \exp, \log, \sin, \cos, \tan, \dots\} \\ \mathcal{F}_L &= \{!=, ==, <, >, <=, >=, \dots\} \end{aligned} \tag{3.8}$$

The operators in $\mathcal{F}_A$ have uniquely defined rules in differential calculus while that in $\mathcal{F}_M$ have uniquely defined expressions for their derivatives. The user can define rules for derivatives for operators in $\mathcal{F}_L$ based on the context.

A roadmap for the implementation is given ahead. A minimal set of operators is selected from the set $\mathcal{F}$ and a simplified implementation of the AD library for these operators are presented. We select addition, multiplication, power function, sin and exponential function as model operators for the minimal library. All other operations can be implemented following same methodology. For an easier understanding of $\mathcal{R}$apid, we first present briefly the implementation of the minimal *Tangent* library. Examples have been chosen carefully to demonstrate the functioning of the library. This is followed by the implementation details of $\mathcal{R}$apid corresponding to the same minimal *Tangent* library. Throughout we list modified snippets of the C++ *Tangent* and $\mathcal{R}$apid implementations. The listings have been highly simplified for didactic purposes. Interested readers willing to use or implement $\mathcal{R}$apid for their purposes will benefit from going through these listings before looking through the actual code.

3.5.2 *Tangent* library

The *Tangent* library consists of a *Tangent* class and other associated functions. The flow of derivative with the following example will help us to design the *Tangent* library.

3.5.2.1 Flow of derivatives

If one numerically sets $x_1^{'} = 1$ (implying $\partial_{x_1}(x_1) = 1$) and $x_2^{'} = 0$ (implying $\partial_{x_2}(x_2) = 0$), then the derivatives can be made to flow along each step of the evaluation tree for the functions f_1 and f_2 (Eq. 3.1 and Figure 3.1) according to the Eqs. 3.9 and 3.10. These equations present the computation of derivative of each term $\partial_{x_1} f_1^{(i)}$ and $\partial_{x_1} f_2^{(i)}$ with respect to x_1 from the beginning to end in the evaluation tree depicted in Figure 3.2(a). Prime ($'$) represents the derivative with respect to whatever argument is present in the elementary function. If we choose $x_1^{'} = 1$ and $x_2^{'} = 0$, we effectively set up x_1 and x_2 as the independent active and inactive variable.

$$
\begin{aligned}
x_1^{'} &= 1;\ \text{Or}\ \partial_{x_1}(x_1) = 1 \\
x_2^{'} &= 0;\ \text{Or}\ \partial_{x_2}(x_2) = 0 \\
\Longrightarrow \partial_{x_1}\left(f_1^{(1)}\right) &= \partial_{x_1}(x_1 * x_2) = \partial_{x_1}(x_1) * x_2 + \partial_{x_2}(x_2) * x_1 \\
&= x_1^{'} * x_2 + x_2^{'} * x_1 = x_2 + 0 = x_2 \\
\Longrightarrow \partial_{x_1}\left(f_1^{(2)}\right) &= \partial_{x_1}(\sin(x_1)) = (\sin(x_1))^{'} \cdot \partial_{x_1}(x_1) = \cos(x_1) \cdot 1 = \cos(x_1) \\
\Longrightarrow \partial_{x_1}\left(f_1^{(3)}\right) &= \partial_{x_1}\left(f_1^{(1)} + f_1^{(2)}\right) = \partial_{x_1}\left(f_1^{(1)}\right) + \partial_{x_1}\left(f_1^{(2)}\right) = x_2 + \cos(x_1) \\
\Longrightarrow \partial_{x_1}\left(f_1^{(4)}\right) &= \partial_{x_1}\left(\exp\left(f_1^{(3)}\right)\right) = \left(\exp\left(f_1^{(3)}\right)\right)^{'} \partial_{x_1}\left(f_1^{(3)}\right) \\
&= \log\left(f_1^{(3)}\right) \cdot (x_2 + \cos(x_1)) \\
&= \log(x_1 x_2 + \sin(x_1)) \cdot (x_2 + \cos(x_1))
\end{aligned}
\tag{3.9}
$$

$$
\begin{aligned}
x_1^{'} &= 1;\ \text{Or}\ \partial_{x_1}(x_1) = 1 \\
x_2^{'} &= 0;\ \text{Or}\ \partial_{x_2}(x_2) = 0 \\
\implies \partial_{x_1}\left(f_2^{(1)}\right) &= \partial_{x_1}\left(x_1^2\right) = \left(x_1^2\right)^{'} \partial_{x_1}(x_1) = 2x_1 \cdot 1 = 2x_1 \qquad (3.10) \\
\implies \partial_{x_1}\left(f_2^{(1)}\right) &= \partial_{x_1}\left(x_2^2\right) = \left(x_2^2\right)^{'} \partial_{x_1}(x_2) = 2x_1 \cdot 0 = 0 \\
\implies \partial_{x_1}\left(f_2^{(2)}\right) &= \partial_{x_1}\left(f_2^{(1)} + f_2^{(2)}\right) = \partial_{x_1}\left(f_2^{(1)}\right) + \partial_{x_1}\left(f_2^{(2)}\right) = 2x_1 + 0 = 2x_1
\end{aligned}
$$

Similarly x_2 can be set as the independent active variable by numerically setting $x_1^{'} = 0$ (implying $\partial_{x_1}(x_1) = 0$) and $x_2^{'} = 1$ (implying $\partial_{x_2}(x_2) = 1$). A second sweep all the way from the beginning to the end with such a setting will make the derivatives flow through all elementary functions with respect to x_2 as depicted in Figure 3.2(b). The equations corresponding to the flow are,

$$
\begin{aligned}
x_1^{'} &= 0;\ \text{Or}\ \partial_{x_1}(x_1) = 0 \\
x_2^{'} &= 1;\ \text{Or}\ \partial_{x_2}(x_2) = 1 \\
\implies \partial_{x_2}\left(f_1^{(1)}\right) &= \partial_{x_2}(x_1 * x_2) = \partial_{x_2}(x_1) * x_2 + \partial_{x_2}(x_2) * x_1 \\
&= x_1^{'} x_2 + x_2^{'} x_1 = x_1 \\
\implies \partial_{x_2}\left(f_1^{(2)}\right) &= \partial_{x_2}(\sin(x_1)) = (\sin(x_1))^{'} \cdot \partial_{x_2}(x_1) = \cos(x_1) \cdot 0 = 0 \\
\implies \partial_{x_2}\left(f_1^{(3)}\right) &= \partial_{x_2}\left(f_1^{(1)} + f_1^{(2)}\right) = \partial_{x_2}\left(f_1^{(1)}\right) + \partial_{x_2}\left(f_1^{(2)}\right) = x_1 + 0 = x_1 \\
\implies \partial_{x_2}\left(f_1^{(4)}\right) &= \partial_{x_2}\left(\exp\left(f_1^{(3)}\right)\right) = \left(\exp\left(f_1^{(3)}\right)\right)^{'} \partial_{x_2}\left(f_1^{(3)}\right) \\
&= \log\left(f_1^{(3)}\right) \cdot (x_1) \\
&= \log(x_1 x_2 + \sin(x_1)) \cdot (x_1) \qquad (3.11)
\end{aligned}
$$

$$
\begin{aligned}
x_1^{'} &= 0;\ \text{Or}\ \partial_{x_1}(x_1) = 0 \\
x_2^{'} &= 1;\ \text{Or}\ \partial_{x_2}(x_2) = 1 \\
\implies \partial_{x_2}\left(f_2^{(1)}\right) &= \partial_{x_2}(\text{pow}(x_1, 2)) = (\text{pow}(x_1, 2))^{'} \cdot \partial_{x_1}(x_1) \\
&= 2x_1 \cdot 0 = 0 \\
\implies \partial_{x_2}\left(f_2^{(2)}\right) &= \partial_{x_2}(\text{pow}(x_2, 2)) = (\text{pow}(x_2, 2))^{'} \cdot \partial_{x_2}(x_1) \\
&= 2x_2 \cdot 1 = 2x_2 \qquad (3.12) \\
\implies \partial_{x_2}\left(f_2^{(2)}\right) &= \partial_{x_2}\left(f_2^{(1)} + f_2^{(2)}\right) = \partial_{x_2}\left(f_2^{(1)}\right) + \partial_{x_2}\left(f_2^{(2)}\right) = 0 + 2x_2 = 0
\end{aligned}
$$

We can discern a pattern in such a mechanical process of differentiating

the functions illustrated above. A need for a structure to hold the data that facilitates such a process of mechanical differentiation becomes evident.

3.5.2.2 Tangent data structure

The first step in implementing the AD framework is to design a proper '*data-structure*' that holds the relevant information [71]. From Figure 3.2 and Eqs. 3.9-3.12, it is clear that each variable needs two fields to store - a function value and a derivative value. Since each is a real value, each can be of the '*double'* data type. Thus any variable of type *Tangent* will inherently have two fields. A class '*Tangent'* as listed in Listing 2 is therefore introduced that contains a component for storing its value ('double _v') and a second component for storing its derivative value ('double _dv').

3.5.2.3 Building the *Tangent* Class

The function and the derivative values for a forward mode AD need to be computed simultaneously. Member functions have to be built into the class acting on the data structure to accomplish this simultaneous computation. The operators in $\mathcal{F}$ are re-implemented in the library for simultaneous computation of function values and derivatives. The operator overloading feature of C++ enables us to accomplish this task.

In our implementation, some operators are re-implemented in the '*Tangent'* class with class member functions and the remaining as non-member functions. Listings 1 defines the data members and the bare minimum mem-

Program Listing 1 Tangent class definition

```
class Tangent {
 public:
  // Constructors
  Tangent() {}
  Tangent(const double v, const double dv):_v(v),_dv(dv) {}
  Tangent(const double v):_v(v){}
  Tangent(const Tangent& o):_v(o._v),_dv(o._dv){}
  ~Tangent() {}
  // Member functions
  Tangent& operator=(const Tangent& o);
  Tangent& operator+=(const Tangent& o);
  Tangent& operator*=(const Tangent& o);
  //display
  void print() {cout << "_v=" << _v << "," << "_dv=" << _dv;}
  // Data members
  double _v;
  double _dv;
};
```

ber functions. Lines 4-8 define few constructors including the copy constructor for various types of initializations of the object of the class. Note that only the '$=, +=, *=$' operators are re-implemented as class member functions while the '$+, *, \text{pow}, \sin, \exp$' operators are implemented as non-class member functions[2].

Program Listing 2 Overloading assignment operator '='

```
1 Tangent& Tangent::operator=(const Tangent& o) {
2   if (this == &o)
3     return *this;
4   _v = o._v;
5   _dv = o._dv;
6   return *this;
7 }
```

Listing 2 implements the assignment operator where the two data members *_v* and *_dv* are correspondingly assigned. Listing 3 shows the implementation of '+=' as a class member function and '+' as non-class member function. Note that both these operators are necessary and work in unison to realize the addition operator of two objects of the *Tangent* type. It is in the '+=' member function that the differentiation rules for addition are programmed.

Listing 4 implements multiplication operator. Again we need the '*=' and '*' operators to be overloaded as a class and a non-class member functions respectively. The multiplication rules of calculus are coded in '*=' class member function.

[2]This is one way of a simple implementation

Program Listing 3 Implementing addition by overloading '+=' and '+' operators

```
1 Tangent& Tangent::operator+=(const Tangent& o) {
2   _v += o._v;
3   _dv += o._dv;
4   return *this;
5 }
6 Tangent operator+(const Tangent& a, const Tangent& b) {
7   return Tangent(a) += b;
8 }
```

Such a combination of ('+=,+') and ('*=,*') are necessary to realize a general function evaluation which will have multiple elementary operations (Figure 3.2). Only then would the derivatives propagate along the function evaluation tree in accordance with the chain rule of differentiation. It is also noted that the copy constructor plays an important role in the implementation. For instance, the '+' produces a temporary object (Listing 3 line 7) with a copy constructor and becomes the calling object for the '+='. This temporary object is destroyed as soon as it is no longer needed.

Program Listing 4 Implementing multiplication operation by overloading '*=' and '*' operators

```
1 Tangent& Tangent::operator*=(const Tangent& o) {
2   _dv = _dv*o._v + _v*o._dv;
3   _v *= o._v;
4   return *this;
5 }
6 Tangent operator*(const Tangent& a, const Tangent& b) {
7   return Tangent(a) *= b;
8 }
```

In-built mathematical functions like 'pow', 'sin' and 'exp' etc. are re-implemented in accordance with rules of differential calculus as non-member functions. Listing 5 should be fairly self-explanatory.

Program Listing 5 Implementation of 'pow', 'sin' and 'exp' mathematical functions.

```
1  Tangent pow(const Tangent& a, const Tangent& b) {
2    double arg1 = a._v == 0.0 ? 0 : pow(a._v,b._v);
3    double arg2 = a._v == 0.0 ? 0 : a._dv*b._v*pow(a._v,b._v-1);
4    return Tangent(arg1, arg2);
5  }
6  Tangent sin(const Tangent& a){
7    return Tangent(sin(a._v), a._dv*cos(a._v));
8  }
9  Tangent exp(const Tangent& a){
10   return Tangent(exp(a._v), a._dv*exp(a._v));
11 }
```

With this infrastructure, we now proceed to compute the two example functions discussed in Figure 3.1 using the *Tangent* type. The function 'testExample1()' in Listing 6 computes the derivative of functions f_1 and f_2 given in Eq. 3.1 with respect to x_1. The independent variables x_1 and x_2 are defined as '*Tangent*' objects in lines 3-4. Setting the '$\_dv$' field of object x_1 signals that we wish to compute the derivative of all functions with respect to x_1. At the same time, the corresponding field of x_2 is set to 0. We can obtain the derivatives of the functions with respect to x_2 by setting the $\_dv$ fields the reverse way as shown in 'testExample2()' in the same listing. It is noted that in such an infrastructure, if there are n independent variables, the $\_dv$ field

of only one of them should be set to 1 and all the remaining $n-1$ variables should be set to 0 to obtain derivatives correctly.

Program Listing 6 Computing derivatives of functions in Eq. 3.1 using *Tangent* class.

```
1  //Computing df1/dx1, df2/dx1
2  void testExample1() {
3    Tangent x1(2.0,1);
4    Tangent x2(3.0,0);
5    Tangent f1 = exp(x1*x2+sin(x1)); f1.print();
6    Tangent f2 = pow(x1,2)+pow(x2,2); f2.print();
7  }
8  //Computing df1/dx2, df2/dx2
9  void testExample2() {
10   Tangent x1(2.0,0);
11   Tangent x2(3.0,1);
12   Tangent f1 = exp(x1*x2+sin(x1)); f1.print();
13   Tangent f2 = pow(x1,2)+pow(x2,2); f2.print();
14 }
```

The infrastructure can now be made more general and multipurpose in various ways using various programming techniques. Expression templates[81] as discussed earlier can be used to enhance computational efficiency. Type traits is another useful technique for compile time type conversion [93]. Interested readers are pointed to [94, 50].

We also mention that the FAD package [81] available in SACADO [94] does have the mechanism to specify more than one independent variable with which the derivatives need to be computed. Line 4 in Listing 7 is a multiple array object capable of holding derivatives with respect to multiple

independent variables.

Program Listing 7 Data-structure of FAD class.

```
template <class T> class Fad {
protected:
  T val_;
  Vector<T> dx_;
public:
  ...
};
```

However for reasons mentioned in Section 3.4, we follow a different approach to obtain all the partial derivatives in a single sweep.

3.5.3 $\mathcal{R}$apid Library

3.5.3.1 $\mathcal{R}$apid data structure

A $\mathcal{R}$apid data structure has two data members. The first one is of data type double named *'_v'* and stores the value of the function that it represents. The second data member is an object of *'STL map data structure'* named *_dv* that stores all the partial derivatives of the function with respect to each independent variable. Figure 3.7 shows the abstract definition of $\mathcal{R}$apid data structure.

C++ STL maps[77] are associative containers that store elements formed by a combination of a *'key value'* and a '*mapped value*' [95]. The key value is used to sort and uniquely identify the element, while the mapped value is a data associated with this key. The data type of the key and mapped value can be chosen based on the purpose. The mapped value is accessed directly by its

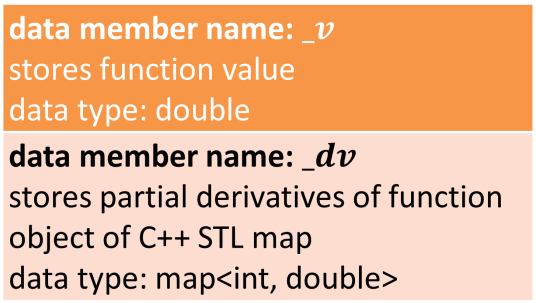


Figure 3.7: $\mathcal{R}$apid data structure.

corresponding key using the bracket operator '[]'. The number of elements in the map can grow dynamically. Maps also have built in iterators that facilitate traversal of the elements. The elements in a map are always sorted by their keys following a specific strict weak ordering criterion (in our case, ascending order) [95].

For $\mathcal{R}$apid, the key value of the data member *_dv* is of data type *int* while the mapped value is of type *double*, i.e. *map¡int, double¿*. The number of elements of *_dv* depend on the number of independent variables that function *_v* depends on. (Later an alias '*Gradient*' for the combination *map¡int, double¿* would be used.)

3.5.3.2 Setting up independent and dependent variables

All the variables in X and F are objects of the $\mathcal{R}$apid class shown in Figure 3.7. The classification of variables into independent active and inactive and dependent variables have to be built into the $\mathcal{R}$apid construct.

In the construct, all the elements of set X_a are assigned a unique iden-

tifier. The integer key of the data member $\_dv$ is used for this purpose. The corresponding mapped value of $\_dv$ is set to 1.0, essentially capturing the fact that the derivative of a variable with itself is unity. Also, the map object $\_dv$ for an active variable strictly contains only one element. The map object $\_dv$ for the inactive independent variable is kept empty by leaving it untouched and therefore does not need an identifier/key. An inactive independent variable thus does not have any elements in $\_dv$. This is in contrast to our *Tangent* class where the $\_dv$ field was set to 0 for the inactive variables.

The data member $\_dv$ of a dependent variable (function) is also left untouched. However the construct of the $\mathcal{R}$apid class facilitates populating the elements of the map dynamically. The load is completely taken up by the the compiler. The number of elements in the map depends on how many independent active variables r_i the function variable f_i depends on. This allows optimal use of storage space akin to various sparse matrix storage data structures. The key of each element of $\_dv$ signals the partial derivative of f_i with respect to the particular active variable with the matching key. The associated value stores the value of the corresponding derivative.

From a user's perspective, one just needs to set up the active variable set by assigning a unique integer key after which the $\mathcal{R}$apid data structure propagates all the partial derivatives along the function evaluation trace. An example is presented next illustrating the how the data structure holds the function and derivative values.

3.5.3.3 Example illustrating $\mathcal{R}$apid data structure

Let $X = \{x_1, x_2, \ldots x_9\}$ be the set of 9 independent variables associated with each cell in Figure 3.8(a). Let $F = \{f_1, f_2, \ldots f_9\}$ be a set of functions, with each function defined on set X given by the following test function

$$f_i = \sin\left(\prod_j \left((x_i + c * (x_i + x_j)\right)^2\right) \tag{3.13}$$

where j represents a neighbor of the element i in Figure 3.8(a). Some representative functions are

$$\begin{aligned} f_1 &= \sin\left((x_1 + c * (x_1 + x_2)\right)^2 * (x_1 + c * (x_1 + x_4))^2\right) \\ f_2 &= \sin\left((x_2 + c * (x_2 + x_1)\right)^2 * (x_2 + c * (x_2 + x_3))^2 * (x_2 + c * (x_2 + x_5))^2\right) \\ f_5 &= f_5\,(x_2, x_4, x_5, x_6, x_8) \end{aligned} \tag{3.14}$$

Evaluation of these functions is accompanied by the evaluation of the derivatives with respect to only the independent variables on which they depend. For instance, evaluation of f1 is accompanied by the evaluation of only the following three derivatives,

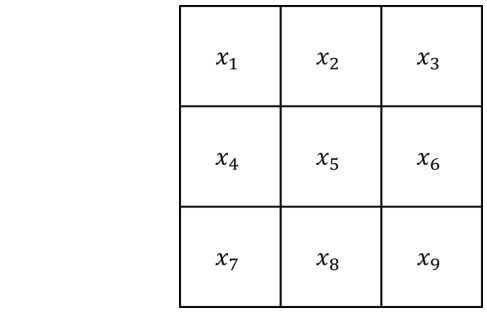
$$
\begin{aligned}
\frac{\partial f_1}{\partial x_1} &= \cos\left((x_1 + c*(x_1+x_2))^2 * (x_1 + c*(x_1+x_4))^2\right) \times \\
&\quad \left((2\,(x_1 + c*(x_1+x_2)) \times (1+c)) * (x_1 + c*(x_1+x_4))^2\right) \times \\
&\quad \left((x_1 + c*(x_1+x_2))^2 * (2\,(x_1 + c*(x_1+x_4)) \times (1+c))\right) \quad (3.15) \\
\frac{\partial f_1}{\partial x_2} &= \ldots \\
\frac{\partial f_1}{\partial x_4} &= \ldots
\end{aligned}
$$

Similarly f_2 and f_5 evaluations are accompanied by the evaluation of four and five derivative components respectively as is shown in Figures 3.8(b).

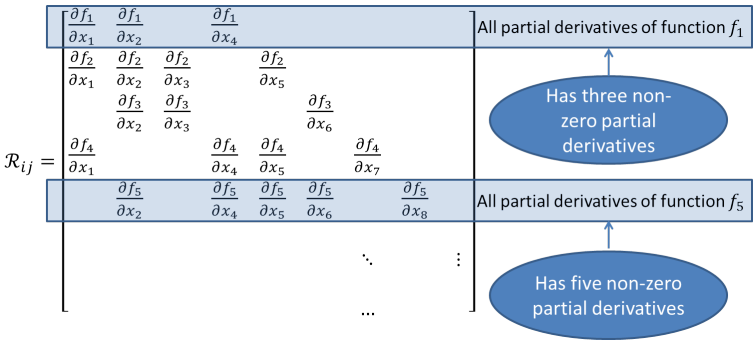
Figure 3.8(c) illustrates the function and derivative values being stored in *_v* and *_dv* fields for some of the independent and dependent $\mathcal{R}$apid variables. As shown in Eq. 3.15, one can see that the derivatives are tedious to compute. We now proceed to their computation and storage in the data structure discussed above.

3.5.3.4 Building the $\mathcal{R}$apid Class

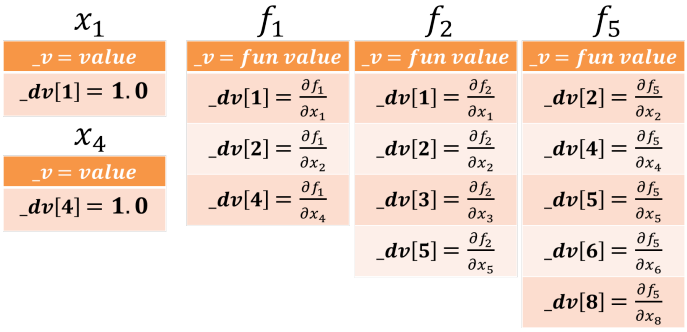
We list a few modified snippets of the C++ $\mathcal{R}$apid implementations that are counterparts of the listings of *Tangent* class. The $\mathcal{R}$apid library consists of a $\mathcal{R}$apid class with member functions and non-member functions acting on $\mathcal{R}$apid objects. Similar to the *Tangent* class, the $\mathcal{R}$apid library overloads (or re-implement) operators in the set $\mathcal{F} = \mathcal{F}_A \cup \mathcal{F}_M \cup \mathcal{F}_L$ using *operator overloading*, some with member functions and others with non-member functions.



(a)



(b)



(c)

Figure 3.8: Illustration of independent and dependent $\mathcal{R}$apid objects. Here we define $f_i = \sin\left(\prod_j \left((x_i + c * (x_i + x_j)\right)^2\right)$ for each cell i in (a) where j represents a neighbor of the element i.

Program Listing 8 Rapid class definition

```
class Rapid
{
 public:
  typedef map<int, double> Gradient;
  typedef const Gradient::value_type GradientIterator;
  typedef map<int, double>::iterator Iter;
  typedef map<int, double>::const_iterator CIter;
  // Constructors
  Rapid() {}
  Rapid(const double v, const Gradient& dv):_v(v),_dv(dv) {}
  Rapid(const double v):_v(v){}
  Rapid(const Rapid& o):_v(o._v),_dv(o._dv){}
  ~Rapid() {}
  // Member functions
  void setAsIndependentVariable(const int key){_dv[key] = 1.0;}
  Rapid& operator=(const Rapid& o);
  Rapid& operator+=(const Rapid& o);
  Rapid& operator*=(const Rapid& o);
  // Display
  void print() {
  cout << "_v=" << _v;
  foreach(GradientIterator& ii, _dv)
    cout << "," << "_dv[" << ii.first << "]=" << ii.second;
  }
  // Data members
  double _v;
  Gradient _dv;
};
```

Listing 8 shows the definition of $\mathcal{R}$apid class with the data members, constructors and member functions. Lines 9-13 lists various constructors with the copy constructor in line 12. The member function to set a $\mathcal{R}$apid variable as an independent active variable is listed in line 15 as described in Section 3.5.3.2.

Program Listing 9 Overloading assignment operator '='

```
1 Rapid& Rapid::operator=(const Rapid& o) {
2   if (this == &o)
3     return *this;
4   _v = o._v;
5   _dv = o._dv;
6   return *this;
7 }
```

The assignment operator '=' is overloaded as shown in Listing 9. The two data members $\_v$ and $\_dv$ of the $\mathcal{R}$apid operand are assigned the corresponding data members of the $\mathcal{R}$apid operand following the assignment operator. The rest of the implementation of the minimal infrastructure is explained with the example function f_1 (Eq. 3.14) discussed in Section 3.5.3.3 given by,

$$f_1 = \sin\left((x_1 + c * (x_1 + x_2))^2 * (x_1 + c * (x_1 + x_4))^2\right)$$

As before, the compiler will break down this function evaluation into a series of elementary ones. Let the evaluation tree be as shown in Figure 3.9 (the order can depend slightly on the compiler) where the superscripts represents the evaluation step,

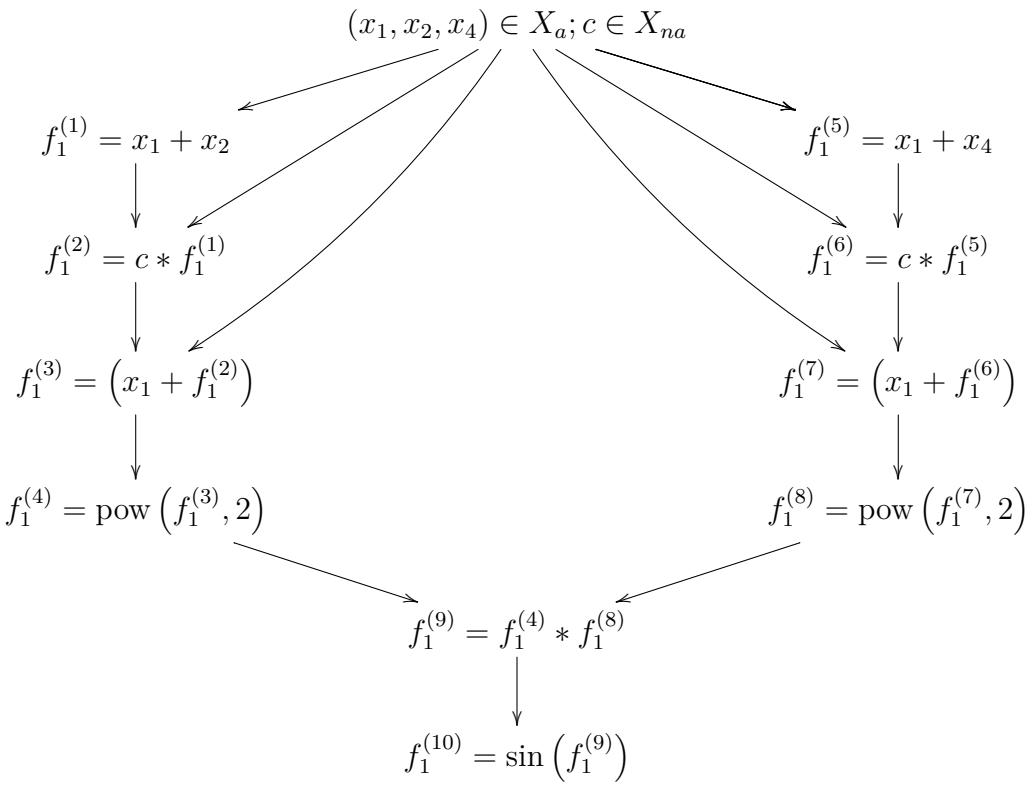


Figure 3.9: Evaluation tree for the example to illustrate $\mathcal{R}$apid.

Program Listing 10 Test example to illustrate use of $\mathcal{R}$apid library.

```
1 void testRapid() {
2   Rapid x1(1.0); x1.setAsIndependentVariable(1);
3   Rapid x2(3.0); x2.setAsIndependentVariable(2);
4   Rapid x4(2.0); x3.setAsIndependentVariable(3);
5   Rapid c(2.0);
6   Rapid f1 = sin(pow(x1+c*(x1+x2),2)*pow((x1+c*(x1+x4)),2));
7   f1.print();
8 }
```

The first step is to set the required variables as independent variables. The active and inactive independent variable are assigned values. The active variables are then assigned unique identifiers, while the inactive are left untouched. This is accomplished by lines 2-5 in Listing 10. The data structure of the independent $\mathcal{R}$apid variables can now be visualized as,

$$x_1 = \begin{bmatrix} \_v = 1 \\ \_dv[1] = 1 \end{bmatrix}; x_2 = \begin{bmatrix} \_v = 3 \\ \_dv[2] = 1 \end{bmatrix}; x_4 = \begin{bmatrix} \_v = 2 \\ \_dv[4] = 1 \end{bmatrix}; c = \begin{bmatrix} \_v = 2 \\ \_dv = <> \end{bmatrix} \tag{3.16}$$

Program Listing 11 Implementing addition by overloading '+=' and '+' operators in $\mathcal{R}$apid class.

```
1 Rapid& Rapid::operator+=(const Rapid& o) {
2   _v += o._v;
3   foreach(GradientIterator& ii, o._dv)
4     _dv[ii.first] += ii.second;
5   return *this;
6 }
7 Rapid operator+(const Rapid& a, const Rapid& b) {
8   return Rapid(a) += b;
9 }
```

The addition of $\mathcal{R}$apid variables is accomplished with Listing 11. The addition operator is illustrated with the computation of $f_1^{(1)}$ given by,

$$f_1^{(1)} = x_1 + x_2 = \begin{bmatrix} \_v = 1 \\ \_dv[1] = 1 \end{bmatrix} + \begin{bmatrix} \_v = 3 \\ \_dv[2] = 1 \end{bmatrix} = \begin{bmatrix} \_v = 4 \\ \_dv[1] = 1 \\ \_dv[2] = 1 \end{bmatrix} \tag{3.17}$$

First, references of $\mathcal{R}$apid objects x_1 and x_2 are passed to 'a' and 'b' in line 7 after which a copy of 'a' is constructed. The copy becomes the output object (line). For explanation we call this copied object $f_1^{(1)}$, though this does not appear explicitly in the listing. This copied object $f_1^{(1)}$ in turn calls the '+=' operator in line 1 with a reference of 'b' passed to object 'o'. After the function values of $f_1^{(1)}$ and 'o' are added to $\_v$ field of $f_1^{(1)}$, computation of its derivative components starts by traversing through the $\_dv$ map of the 'o'. The traversal is accomplished with the 'boost' library (line 3). Here it is important to note that the derivative values with common keys of $f_1^{(1)}$ and 'o' get added up and assigned to the corresponding field in $f_1^{(1)}$. Derivative values with disjoint keys of $f_1^{(1)}$ and 'o' just gets carried over to the $f_1^{(1)}$ object. These processes result in the final values for $f_1^{(1)}$ as in Eq. 3.17. There are no common keys between $f_1^{(1)}$ and 'o' specific to the computation of Eq. 3.17 .

Multiplication of two $\mathcal{R}$apid objects listed in Listing 12 is slightly more involved and is illustrated with the computation of $f_1^{(2)}$ in Figure 3.9 as follows,

$$f_1^{(2)} = c{*}f_1^{(1)} = \begin{bmatrix} \_v = 2 \\ \_dv = <> \end{bmatrix} * \begin{bmatrix} \_v = 4 \\ \_dv[1] = 1 \\ \_dv[2] = 1 \end{bmatrix} = \begin{bmatrix} \_v = 2 \times 4 \\ \_dv[1] = 1 \times 2 \\ \_dv[2] = 1 \times 2 \end{bmatrix} = \begin{bmatrix} \_v = 8 \\ \_dv[1] = 2 \\ \_dv[2] = 2 \end{bmatrix} \tag{3.18}$$

Here references of objects 'c' and $f_1^{(1)}$ are passed to 'a' and 'b' (line 20) after which a copy of 'a' is assigned to the temporary object which we call $f_1^{(2)}$. Next $f_1^{(2)}$ calls the operator '*=' with the argument 'b' (or $f_1^{(1)}$) being passed to 'o'. In the implementation of the multiplication operation, we need

Program Listing 12 Implementing multiplication operation by overloading '*=' and '*' operators in $\mathcal{R}$apid class.

```
Rapid& Rapid::operator*=(const Rapid& o) {
  foreach(GradientIterator& ii, o._dv) {
    Iter key;
    key = _dv.find(ii.first);
    if (key != _dv.end())
      _dv[ii.first] = _dv[ii.first]*o._v + _v*ii.second;
    else
      _dv[ii.first] = _v*ii.second;
    }
  }
  foreach(GradientIterator& ii, _dv) {
    CIter key;
    key = o._dv.find(ii.first);
    if (key == o._dv.end())
      _dv[ii.first] = _dv[ii.first]*o._v;
  }
  _v *= o._v;
  return *this;
}
Rapid operator*(const Rapid& a, const Rapid& b) {
  return Rapid(a) *= b;
}
```

to traverse map field $\_dv$ of both objects $f_1^{(2)}$ and 'o'. During the traversal of $\_dv$ of 'o' (lines 2-10), the derivatives of $f_1^{(2)}$ and 'o' with common keys are computed in line 6, following the rules of calculus. Line 8 implements the computation of derivatives for disjoint keys in 'o' but not in $f_1^{(2)}$. The second traversal of $\_dv$ of $f_1^{(2)}$ (lines 11-16) computes the derivatives of the disjoint key in $f_1^{(2)}$ but not in 'o'. Specifically for this example computation of Eq. 3.18, only line 8 is executed.

A similar methodology follows for the computation of the remaining elementary functions $f_1^{(3)}$ to $f_1^{(10)}$. Here, only relevant differences are explained. In the computation of $f_1^{(3)}$, the derivatives of objects have common key (here '[1]') and are therefore added to $f_1^{(3)}$. Again the disjoint key is just carried over to the output $f_1^{(3)}$.

$$f_1^{(3)} = x_1 + f_1^{(2)} = \begin{bmatrix} \_v = 1 \\ \_dv\,[1] = 1 \end{bmatrix} + \begin{bmatrix} \_v = 8 \\ \_dv[1] = 2 \\ \_dv[2] = 2 \end{bmatrix} = \begin{bmatrix} \_v = 1 + 8 \\ \_dv\,[1] = 1 + 2 \\ \_dv\,[2] = 2 \end{bmatrix} = \begin{bmatrix} \_v = 9 \\ \_dv\,[1] = 3 \\ \_dv\,[2] = 2 \end{bmatrix} \tag{3.19}$$

Overloading of mathematical functions '*pow*' and '*sin*' for $\mathcal{R}$apid objects are listed in Listing . The computation of $f_1^{(4)}$ is self explanatory, except that the object 'b' is assumed to be constant or an inactive variable.

$$f_1^{(4)} = \mathrm{pow}\left(f_1^{(3)}, 2\right) = \begin{bmatrix} \_v = 9^2 \\ \_dv\,[1] = 3 \times (2 \times 9^1) \\ \_dv\,[2] = 2 \times (2 \times 9^1) \end{bmatrix} = \begin{bmatrix} \_v = 81 \\ \_dv\,[1] = 54 \\ \_dv\,[2] = 36 \end{bmatrix} \tag{3.20}$$

Program Listing 13 Implementing of pow and sin functions

```
Rapid pow(const Rapid& a, const Rapid& b) {
  //Assumes b is a constant
  Rapid o;
  o._v = pow(a._v,b._v);
  foreach(Rapid::GradientIterator& ii, a._dv)
    o._dv[ii.first] = ii.second*b._v*pow(a._v,b._v-1);
  return o;
}
Rapid sin(const Rapid& a) {
  Rapid o;
  o._v = sin(a._v);
  foreach(Rapid::GradientIterator& ii, a._dv)
    o._dv[ii.first] = ii.second*cos(a._v);
  return o;
}
```

Computations of the $f_1^{(5)}$ to $f_1^{(8)}$ proceed exactly the same way $f_1^{(1)}$ to $f_1^{(4)}$, except with a new independent variable x_4.

$$f_1^{(5)} = x_1 + x_4 = \begin{bmatrix} \_v = 1 \\ \_dv[1] = 1 \end{bmatrix} + \begin{bmatrix} \_v = 2 \\ \_dv[4] = 1 \end{bmatrix} = \begin{bmatrix} \_v = 3 \\ \_dv[1] = 1 \\ \_dv[4] = 1 \end{bmatrix} = \begin{bmatrix} \_v = 3 \\ \_dv[1] = 1 \\ \_dv[4] = 1 \end{bmatrix} \tag{3.21}$$

$$f_1^{(6)} = c * f_1^{(5)} = \begin{bmatrix} \_v = 2 \\ \_dv = <> \end{bmatrix} * \begin{bmatrix} \_v = 3 \\ \_dv[1] = 1 \\ \_dv[4] = 1 \end{bmatrix} = \begin{bmatrix} \_v = 6 \\ \_dv\,[1] = 2 \\ \_dv\,[4] = 2 \end{bmatrix} \tag{3.22}$$

$$f_1^{(7)} = x_1 + f_1^{(6)} = \begin{bmatrix} \_v = 1 \\ \_dv\,[1] = 1 \end{bmatrix} + \begin{bmatrix} \_v = 6 \\ \_dv\,[1] = 2 \\ \_dv\,[4] = 2 \end{bmatrix} = \begin{bmatrix} \_v = 7 \\ \_dv\,[1] = 3 \\ \_dv\,[4] = 2 \end{bmatrix} \tag{3.23}$$

$$f_1^{(8)} = \text{pow}\left(f_1^{(7)}, 2\right) = \begin{bmatrix} \_v = 7^2 \\ \_dv\,[1] = 3 \times (2 \times 7^1) \\ \_dv\,[4] = 2 \times (2 \times 7^1) \end{bmatrix} = \begin{bmatrix} \_v = 49 \\ \_dv\,[1] = 42 \\ \_dv\,[4] = 28 \end{bmatrix} \quad (3.24)$$

Computation of $f_1^{(9)}$ invokes all the possibilities of multiplication and deserves mention. '[1]' is a common key and hence the computation of $\_dv[1]$ is accomplished by 6. Derivatives of the disjoint key in $f_1^{(4)}$ but not in $f_1^{(8)}$ $\_dv[2]$ is computed in line 15 while $\_dv[4]$ is computed in 8. The order of computation can be different from what is explained here. However, the compiler takes the burden of arranging the map in increasing order (this is a property of map stl).

$$\begin{aligned} f_1^{(9)} = f_1^{(4)} * f_1^{(8)} &= \begin{bmatrix} \_v = 81 \\ \_dv\,[1] = 54 \\ \_dv\,[2] = 36 \end{bmatrix} * \begin{bmatrix} \_v = 49 \\ \_dv\,[1] = 42 \\ \_dv\,[4] = 28 \end{bmatrix} \\ &= \begin{bmatrix} \_v = 81 \times 49 \\ \_dv\,[1] = 54 \times 49 + 42 \times 81 \\ \_dv\,[2] = 36 \times 49 \\ \_dv\,[4] = 28 \times 81 \end{bmatrix} = \begin{bmatrix} \_v = 3969 \\ \_dv\,[1] = 6048 \\ \_dv\,[2] = 1764 \\ \_dv\,[4] = 2268 \end{bmatrix} \end{aligned} \quad (3.25)$$

Finally the computation of $f_1^{(10)}$ is shown in Eq. 3.26 and is self explanotory.

$$f_1^{(10)} = \sin\left(f_1^{(7)}\right) = \begin{bmatrix} \_v = \sin(3969) \\ \_dv\,[1] = 6048 \times \cos(3969) \\ \_dv\,[2] = 1764 \times \cos(3969) \\ \_dv\,[4] = 2268 \times \cos(3969) \end{bmatrix} = \begin{bmatrix} \_v = -0.9202 \\ \_dv\,[1] = -2368.1073 \\ \_dv\,[2] = -690.6979 \\ \_dv\,[4] = -888.0402 \end{bmatrix} \quad (3.26)$$

The reader might need to go back and forth between examples and the listings before the process is completely understood. All the operators in the set $\mathcal{F}$ can be similarly implemented with slight modifications. After having developed the library, we concisely present the procedure to perform topology optimization using Memosa in conjunction with $\mathcal{R}$apid library.

3.6 Using MEMOSA with $\mathcal{R}$apid library

As discussed earlier, C++ allows the variable type to be templated and suitably type-cast at compile time. All classes in MEMOSA implementing scalar transport equations, including the diffusion, convection, source and transient operators, have been templated at different levels. To use the $\mathcal{R}$apid library with MEMOSA, only small modifications are needed in the source code. With the help of a representative listing, we give an idea of the type of edits needed.

For illustrative purposes, we list a highly pruned version of a class implementing the '*diffusion discretization*' of the PDE in Listing 14. The types of three variables *X, Diag, OffDiag* as listed in line 1 are specified at compile time. These are templated to facilitate discretization of the diffusion operator arising in various physical models and are beyond the scope of the discussion here. For heat conduction problems, these are generally passed with scalar double type for each. The entire code gets compiled when passed with the type *double* (or any other type except $\mathcal{R}$apid).

As far as sensitivity computations are concerned, we are interested only

Program Listing 14 Using the templated diffusion discretization class with $\mathcal{R}$apid.

```
template<class X, class Diag, class OffDiag>
  class DiffusionDiscretization:public Discretization
{
 public:
  typedef typename NumTypeTraits<X>::T_Scalar T_Scalar;
  typedef CRMatrix<Diag,OffDiag,X> CCMatrix;
  typedef typename CCMatrix::DiagArray DiagArray;
  typedef typename CCMatrix::PairWiseAssembler CCAssembler;
  typedef Gradient<X> XGrad;
  typedef Array<X> XArray;
  typedef Array<T_Scalar> TArray;
  typedef Vector<T_Scalar,3> VectorT3;
  typedef Array<VectorT3> VecT3Array;

 DiffusionDiscretization(...) :
  Discretization(...){}
  void discretize(...)
  {
    ...
#if !(defined(USING_ATYPE_RAPID))
    CCMatrix& matrix=dynamic_cast<CCMatrix&>
                        (mfmatrix.getMatrix(cVarIndex, cVarIndex));
    DiagArray& diag = matrix.getDiag();
#endif
    const VecT3Array& cellCentroid=dynamic_cast<const VecT3Array&>(_geomFields.coordinate[cells]);
    const TArray& cellVolume = dynamic_cast<const TArray&>(_geomFields.volume[cells]);
    XArray& xCell = dynamic_cast<const XArray&>(xField[cVarIndex]);
    XArray& rCell = dynamic_cast<XArray&>(rField[cVarIndex]);
    ...
    foreach(const FaceGroupPtr fgPtr, mesh.getAllFaceGroups()) {
        ...
#if !(defined(USING_ATYPE_RAPID))
        DiagArray& diag = matrix.getDiag();
        CCAssembler& assembler = matrix.getPairWiseAssembler(faceCells);
#endif
        for(int f=0; f<nFaces; f++) {
            const int c0 = faceCells(f,0);
            const int c1 = faceCells(f,1);
            T_Scalar vol0 = cellVolume[c0];
            T_Scalar vol1 = cellVolume[c1];
            VectorT3 ds=cellCentroid[c1]-cellCentroid[c0];
            faceDiffusivity = harmonicAverage(diffCell[c0],diffCell[c1]);
            const T_Scalar diffMetric = faceAreaMag[f]*faceAreaMag[f]/dot(faceArea[f],ds);
            const T_Scalar diffCoeff = faceDiffusivity*diffMetric;
            const VectorT3 secondaryCoeff = faceDiffusivity*(faceArea[f]-ds*diffMetric);
            const XGrad gradF = (xGradCell[c0]*vol0+xGradCell[c1]*vol1)/(vol0+vol1);
            const X dFluxSecondary = gradF*secondaryCoeff;
            X dFlux=diffCoeff*(xCell[c1]-xCell[c0])+dFluxSecondary;
            rCell[c0] += dFlux; rCell[c1] -= dFlux;
#if !(defined(USING_ATYPE_RAPID))
            assembler.getCoeff01(f) += diffCoeff;
            assembler.getCoeff10(f) += diffCoeff;
            diag[c0] -= diffCoeff; diag[c1] -= diffCoeff;
#endif
         }
      }
  }
  ...
};
```

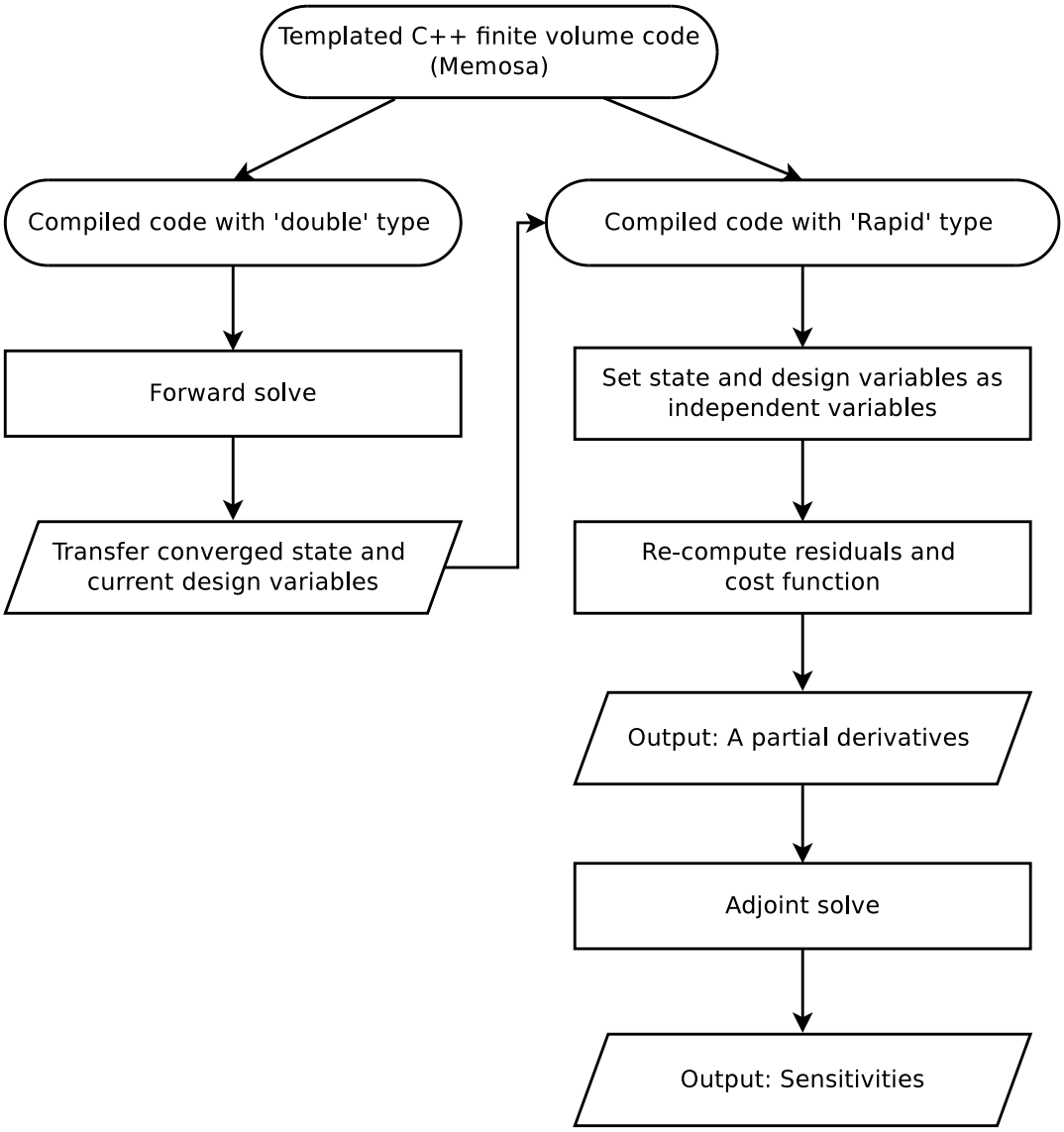


Figure 3.10: Methodology for obtaining sensitivities of numerical models in MEMOSA using $\mathcal{R}$apid.

in residual computation with the converged variables in this code snippet. We do not form the Jacobian for the linear system of the forward solution. A separate object file is created by passing the type $\mathcal{R}$apid for *X, diag* and *OffDiag.* Technically it does not matter what we pass for *diag* or *offDiag* since they do not get used for residual computation. The parts of the code that should be precluded from compilation are enclosed within 'macros' (other methods can also be used). Notice how manual Jacobian computation required for the forward solve is avoided with the help of lines 20-24, 32-35 and 50-54 (color coded in pink) in Listing 14 when run in $\mathcal{R}$apid mode. This will ensure that only the residual *rCell*[ ] is computed. At the end of the program function, the $\mathcal{R}$apid type computes residuals *rCell* in its field *_v* and all its derivatives in its field *_dv*. The residual field *_dv* may be checked for correctness by making sure that it is close to zero within tolerance (or whatever the forward solve achieved at convergence for residuals). Following the example all the necessary minor edits were made in various other classes.

Figure 3.10 shows such a C++ finite volume templated code after making necessary edits to all the *classes* being used in $\mathcal{R}$apid mode. Two compiled versions of the complete code are generated, one type-cast in the basic type – *double* and the other one in the $\mathcal{R}$apid type. For a given distribution of design variables $\boldsymbol{\beta}$, the residual equations are solved for the state variables (temperature or velocities/pressure) using the code in *double* mode. The converged state and design variables are transferred to the code compiled in $\mathcal{R}$apid mode and then both are set as independent variables. Residuals and cost function are

re-calculated in $\mathcal{R}$apid mode, which automatically generates all the required derivatives for sensitivity calculation, as shown in the flowchart depicted in Figure 3.10. This automation is critical in making the code independent of the cost function definition, and also independent of the underlying physics. If a new governing equation is added to the parent code, the $\mathcal{R}$apid infrastructure will automatically generate all necessary derivatives without any new coding.

Going back to the model mesh shown in Figure 3.6, we briefly describe the process of setting the state and design variables as independent variables. There are 31 state variables and 15 design variables. Thus, 46 unique numbers (say 0-30 for temperature state variables and 31-45 for design variables) are set, making them active independent variables. Henceforth, the residuals and cost functions will have the appropriate derivatives in $\_dv$ field after necessary computation. Populating the adjoint linear system is very easy with the $\_dv$ map field. The adjoint system is solved with a direct solver available in PETSc[96].

We demonstrated topology optimization for heat conduction problems in Chapter 2, where the derivatives of residuals and cost functions, required for discrete adjoint sensitivity computation, were hand coded. All the topology optimization test cases with Cartesian meshes presented in Chapter 2 were re-run again, but with sensitivities obtained from $\mathcal{R}$apid mode. We obtained exactly the same topologies for all the cases, thus testing and validating the methodology of using MEMOSA with $\mathcal{R}$apid library to obtain the sensitivities.

3.7 Closure

In this chapter, we explained in detail the implementation of the AD library $\mathcal{R}$apid. One of the pioneers in Automatic Differentiation (AD), Dr. Andreas Griewank says [67]: "AD has been rediscovered and implemented many times, yet its application still has not reached its full potential". Implementing the new AD library $\mathcal{R}$apid for obtaining sensitivities for TO applications has been very effective from our experience, though bench-marking of the library has to be done with other available libraries.

Programs that are C++ templated can be compiled with $\mathcal{R}$apid and used for obtaining sensitivities. We believe that the $\mathcal{R}$apid library will help the scientific community to use AD in for a variety of applications. One area of interest could be dynamic mesh adaptation in response to the evolving flow solution. It is useful to carry out dynamic mesh adaptation based on the instantaneous sensitivities of the various variables of interest.

In the next three chapters, we use $\mathcal{R}$apid extensively in TO for thermal-fluid problems in both laminar and turbulent regimes.

Chapter 4

Laminar flow applications

Following in the footsteps of Chapter 2, the overall objective of this chapter is to develop and demonstrate a topology optimization methodology for steady state flow problems using an un-structured cell-centered finite volume method. Topologies that maximize or minimize flow QoIs for both internal and external flow applications spanning a range of Reynolds numbers are presented to demonstrate the applicability of the methodology. We consider topology optimization for laminar flow applications in this chapter; turbulent flow applications are presented in the next chapter. We lay special emphasis on the development of a novel technique to compute various derivative ingredients required for the discrete adjoint method for flow solvers based on sequential pressure-based incompressible flow algorithms (for example, the SIMPLE algorithm [97, 31]), using the $\mathcal{R}$apid AD library. The technique also forms for the basis for sensitivity computations for flow problems involving new physics like turbulence or heat transfer, which are discussed in forthcoming chapters.

4.1 Introduction

Optimization methods have been extensively applied to various viscous fluid flow problems. For instance, optimal control theory has become popular in designing practical flow control systems [98, 99, 100]. Shape optimization has been applied to determine optimal geometries for minimum drag bodies, diffusers, valves, airfoils etc. [6, 101]. The objective in fluid flow applications is typically to minimize drag, minimize energy rate dissipation or minimize the pressure drop across the domain.

However, as discussed in Chapter 1, relatively few papers have been published on topology optimization for fluid flow applications. Published work on topology optimization using the finite volume method is even more scarce. Most publications on TO employ a SIMP based approach, though level set and lattice Boltzmann based methods have also been used. To begin this chapter, we review the literature on topology optimization for pure fluid flow applications. The discussion mainly pertains to papers based on the SIMP approach.

Topology optimization for fluid flow can be stated as a problem of subdividing the design domain into a solid region and a flow region as illustrated in Figure 4.1. For pure conduction problems, a single heat conduction equation with an interpolation function for thermal conductivity was all that was necessary for distributing the two materials on the design domain. Fluid flow is governed by the Navier-Stokes equations with the no-slip boundary conditions on the solid-fluid interfaces. Since the interfaces are not known

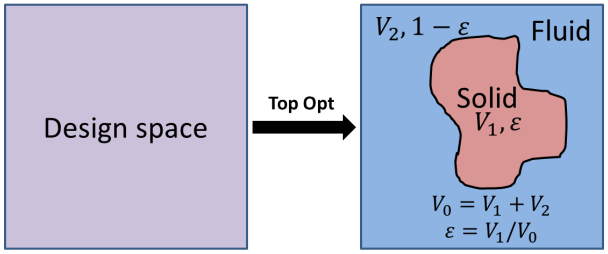


Figure 4.1: Statement of the problem of topology optimization for flow problems.

a priori, topology optimization for fluid flow cannot be performed with the Navier-Stokes equations in their pristine form. The presence of the solid is signaled through the use of momentum sinks in the Navier-Stokes equations, and a variety of different forms have been proposed in the literature[21, 22, 34]. Interpolation of these source terms, as well as interpolation of viscosity, and the numerical consequences of these choices are additional issues for fluid flow. These are discussed briefly, following the historical evolution of the field.

The breakthrough work that introduced topology optimization for fluid flow problems was published by Borrvall and Petersson [21] for Stokes flow (low Reynolds number flow). The Stokes equations that govern creeping flows are obtained by dropping the convective term in Navier-Stokes equations. The Stokes equation is then augmented with a friction term proportional to the velocity to allow the damping of velocity in solid regions. The governing continuity and momentum equations are given by:

$$\begin{aligned} \nabla \cdot (\mu \nabla \vec{v}) - \nabla p - \alpha(\beta)\, \vec{v} &= \mathbf{0} \\ \nabla \cdot (\rho \vec{v}) &= \mathbf{0} \end{aligned} \tag{4.1}$$

where $\vec{v}$ is the velocity field, ∇p is the pressure gradient, μ is the viscosity of the fluid and α is called the inverse permeability function and is used to damp velocities in solid elements. The differentiation of a given spatial point into either solid or fluid region is achieved by virtue of the friction term $\alpha \cdot \vec{v}$, that acts as a momentum sink. This term has its origin in the modeling of fluid flow through porous media. Solid regions which are impermeable can be thought of having extremely high values of α. In such a case, the pressure drop is completely balanced by the friction term, thereby implying vanishing velocities of fluid through solid regions. This emulates the no-slip condition on the solid-fluid interfaces. The impermeability α of the fluid region is negligible, thereby enabling the pressure drop to be balanced solely by the viscous terms, and recovering the Stokes equation. Similar to the interpolation functions of thermal conductivity [Eq. 2.18] and heat source terms [Eq. 2.19] in Chapter 2, α is parameterized in terms of the design variable β. Borrvall and Petersson used the RAMP function to parameterize effective α_{eff}, given by

$$\alpha_{eff}(\beta, \gamma) = \alpha_s + (\alpha_f - \alpha_s)\, \beta \frac{1+\gamma}{\beta+\gamma} \tag{4.2}$$

where $\alpha_f \sim 0$ is the fluid impermeability and $\alpha_s \gg 0$ (Borrvall and Petersson [21] used 10^5) is the solid impermeability. γ is the penalization

factor that controls the convexity of the interpolation function. Extremely high values of α_s may lead to numerical instability.

A paper similar to [21] with minor modifications was published by Guest *et al.* [102], where the authors use the following governing equations, termed the Darcy-Stokes equations, instead of Eq. 4.1.

$$\begin{aligned} (1-\rho(\beta))\,\nabla\cdot(\mu\nabla\vec{v}) - \nabla p - \rho(\beta)\,\frac{\mu}{k}\vec{v} &= \mathbf{0} \\ \nabla\cdot(\rho\vec{v}) &= \mathbf{0} \end{aligned} \tag{4.3}$$

Here $\rho(\beta)$ is interpolated between 0 and 1 forcing the governing equation to switch between Stokes and Darcy's equations in the limits. Both the papers present topologically derived 2D fluidic channels (like diffusers) using the finite element, by minimizing the pressure difference between inlet and outlet for various inlet and outlet configurations. Flow past an obstacle was modelled as well to come up with the optimal shape of the obstacle that minimized the pressure drop. The same methodology was extended to 3D and similar problems presented in [45]. The types of problems solved in these papers have become standard problems for bench-marking topology optimization of fluid flow problems. The addition of the friction term is termed the Brinkman approach to fluid topology optimization. Wiker *et al.* extended Eq. 4.1 by also interpolating viscosity using SIMP. Such a methodology partitions the design space into a fluid region and a permeable region (allowing seeping flow); it is used to design conceptual filters [103].

The Brinkman approach was later extended to the Navier-Stokes equations (i.e. without truncating the convection terms) [22, 104, 105]. The governing equations are

$$\begin{aligned} \nabla \cdot (\rho \vec{v} \vec{v}) &= \nabla \cdot (\mu \nabla \vec{v}) - \nabla p - \alpha(\beta)\, \vec{v} \\ \nabla \cdot (\rho \vec{v}) &= \mathbf{0} \end{aligned} \tag{4.4}$$

In [22, 104], simple 2D fluid channels were designed to minimize the pressure drop for low-to-moderate Reynolds numbers. Hansen *et al.* also introduced various other cost functions in their paper and demonstrated conceptual designs of a flow switch and a fluid mechanism (finding the channel topology that reverses the flow in the direction of interest at the center of the device). Again the Brinkman penalized Navier-Stokes equations were solved using the finite element method. The next natural extension is to add body forces to the Navier Stokes equations which was done by [106]. Here a body force term $\vec{f}(\beta)$ was added to the momentum equations and used a SIMP interpolation to restrict the body forces on to fluid elements. Two-dimensional fluid channels with centrifugal and/or Coriolis forces were investigated.

Topology Optimization has been applied to many passive transport problems, where QoIs are based on convected scalar quantities (thermal or mass transfer) in the presence of an underlying flow field. Forced convection [107], natural convection[54], micro-fluidic mixers [29], reactive flows [108] etc.,

have been explored recently and are an active area of research. To the knowledge of the authors, the only finite volume work in the literature, based on SIMP and using Brinkman's Navier-Stokes equations, published by Marck *et al.* [33]. Marck *et al.* used a staggered sequential algorithm (SIMPLER) on 2D structured meshes for laminar flows. The authors also presented a coupled flow and heat transfer problems with bi-objective function from both flow and thermal variables. We will discuss such coupled problems in Chapter 6.

For completeness, we briefly touch upon other methods employed for topology optimization for fluid flow problems. Almost all the above papers employ a discrete adjoint method for sensitivity computation. Topology optimization based on a continuous adjoint method in the finite volume framework was presented by [109], again based on Eq. 4.4. Pingen *et al.* [110] used the lattice Boltzmann method instead of a Navier-Stokes formulation for solving fluid topology optimization problems. Level set methods for the flow solution have been used for topology optimization have also been used [111, 112, 113]. Here, the solid-fluid interface is implicitly characterized by the zero-level contour of a level set function. The authors claim that level set methods produce interfaces that are smooth compared to density-based approaches. However, level set methods are not as popular in the topology optimization community as SIMP. Kreissl *et al.* [113] employed the lattice Boltzmann method in combination with a level set-based geometric interface representation for generalized shape optimization for fluid flows.

Over the years, finite volume solvers have been developed for incom-

pressible flow employing co-located pressure-velocity schemes on unstructured meshes [31]. Most commercial CFD software vendors, including ANSYS Fluent [114], OpenFOAM [88], STAR-CCM+ [115] etc. employ such methodologies for incompressible flow. As we mentioned before, to our knowledge, topology optimization has not been explored in a co-located unstructured finite volume framework in the published literature. Here we attempt to address this issue, and present detailed methodologies to perform topology optimization in such industry-standard finite volume frameworks.

It has been mentioned earlier that one of the biggest challenges in addressing this issue is the process of computing sensitivities using the adjoint method. Finite volume schemes (like SIMPLE) do not assemble the complete Jacobian during the solution process. For a forward solution, approximate Jacobians are sufficient in guiding the iterative procedure. However a complete Jacobian is necessary for the adjoint solution of the problem. The problem therefore boils down to obtaining complete Jacobians while using partial Jacobians for forward solution. We achieve this using the $\mathcal{R}$apid library mentioned in Chapter 3, as explained in detail in this chapter.

This chapter is composed of two parts. Sections 4.2, 6.2.2 and 4.4 form the first part. Here we present the sequential solution algorithm for flow equations using unstructured co-located finite volume schemes in a residual formulation, with the goal of computing adjoint based sensitivities using automatic differentiation. Finite volume practitioners who wish to obtain sensitivities using a discrete adjoint method for any custom application can use this part

of the chapter independently. The remaining sections form the second part of the chapter. It is here that we present most of material on topology optimization for flow applications. We describe the associated $\mathcal{R}$apid infrastructure in Section 4.5. Finally we present the results obtained by performing topology optimization for flow applications with a variety of test cases in Section 4.6. We believe that the topology optimization procedures described in this chapter, will guide in the advancement of the field for flow applications.

4.2 Governing equations and boundary conditions

The equations governing viscous flow are given by Navier Stokes equations. In all the discussions that follow, the finite volume formulation and the solution of these flow field equations are meant for general 3D flows on unstructured meshes. However for reasons of simplicity we explicitly mention only two independent spatial coordinates during formulation. The flow field variables to be determined are the velocity vector and the pressure fields. We restrict our discussions to Newtonian flow, the governing equations for which are given by

$$\nabla \cdot (\rho \vec{v} u) - \nabla \cdot (\mu \nabla u) = -\nabla p \cdot \vec{i} + S_u \tag{4.5}$$

$$\nabla \cdot (\rho \vec{v} v) - \nabla \cdot (\mu \nabla v) = -\nabla p \cdot \vec{j} + S_v \tag{4.6}$$

$$\nabla \cdot (\rho \vec{v}) = 0 \tag{4.7}$$

where $\vec{v} = [u, v]$ is the velocity vector field and p is the pressure field

to be determined. The diffusion coefficient is the viscosity of the fluid μ. ρ is the density of the fluid.

Eqs. 4.5 and 4.6 are the momentum equations in the x and y directions cast in the form of general scalar transport equation and express the statement of conservation of linear momentum. A part of the stress tersor forms the diffusion term of the momentum equations. The source terms S_u and S_v in the x and y direction contain the body forces and the remaining part of the stress tensor for a general Newtonian fluid. For an incompressible flow of a constant viscosity fluid this part of the stress tensor drops out[48]. The source/sink terms are also used to accommodate additional physical models like Darcy's term for flow through porous media (which we will use for topology optimization) or the Boussinesq approximation for natural convection.

The Eq. 4.7 represents steady state continuity equation expressing conservation of mass.

The boundary is partitioned into a solid boundary(wall), and inlet and outlet regions. A no slip boundary condition is applied to solid boundaries. Either a given velocity or a given pressure condition can be applied to inlets and outlets.

4.3 Numerical method

We discussed the discretization of the diffusion terms in the scalar transport equation in Section 2.4.2, Chapter 2. Momentum equations (Eqs. 4.5 and

4.6) have additional convection terms and pressure gradient terms. We briefly mention the discretization of these terms in the framework of finite volume method for non-orthogonal meshes. Details may be found in [31, 48].

We employ a co-located or non-staggered finite volume scheme, again described in [31] for solving governing Eqs. 4.5-4.6. Consider the cells $C0$ and $C1$ in Figure 4.2. The Cartesian velocities u and v and pressure p are stored at the cell centroids (i.e. at the same location, unlike in staggered mesh schemes). However in the process of computation of these cell-centroid variables, other derived variables need to be computed. The face pressure p_f is associated with the face centroid of face f. The massflux $\mathcal{F}_f$ is also associated with the faces. Recall that the direction the direction $\vec{e}_\xi$ is aligned with the line joining cell centroids, and for general non-orthogonal meshes, is not parallel to face area vector $\vec{A}_f$. The vector $\vec{e}_\eta$ is any direction tangential to the face.

As mentioned, the unknowns to be determined are u, v and p. The momentum equations 4.5 and 4.6 each serve as the scalar transport equation for variables u and v respectively. The continuity equation is also expressed in terms of the velocity components u and v. Thus we see that, for incompressible flow, there is no explicit equation for pressure variable p. Pressure only appears in the pressure gradient terms in the u and v momentum equations. An equation for pressure has to be derived suitably from continuity equation. All these equations are coupled to each other which makes the solution of flow variables even more complicated. However, over the course of time, the finite volume community have developed established algorithms to solve these

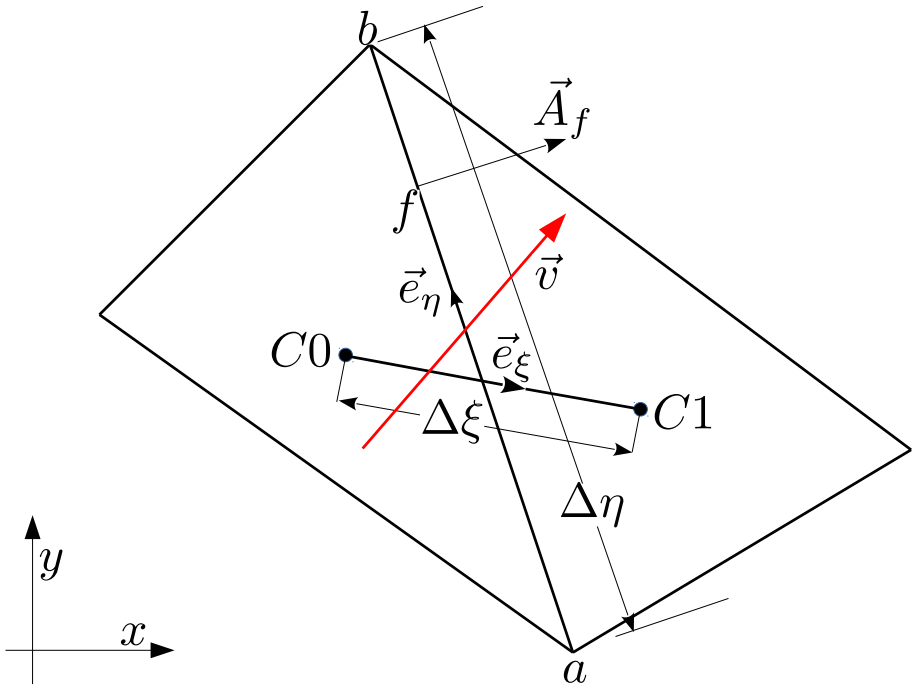


Figure 4.2: Two representative neighboring cells $C0$ and $C1$ sharing a common face f in an unstructured mesh.

coupled equations for the flow variables.

4.3.1 SIMPLE algorithm

The SIMPLE (Semi-Implicit Method for Pressure Linked Equation) algorithm and its variants - SIMPLER (SIMPLE-Revised), SIMPLEC (SIMPLE-Corrected) and others, termed pressure-based methods, are among the most widely used algorithms in the incompressible flow community [97]. The central idea of these algorithms is to create discrete equations for pressure from the continuity equation by introducing a relationship between pressure and velocity from the discrete momentum equation.

MEMOSA uses an unstructured co-located SIMPLEC algorithm for solving flow equations. We discuss these procedures briefly, the main objective being able to devise procedures for obtaining sensitivities of the relevant QoIs using the adjoint method using $\mathcal{R}$apid AD library. Some familiarity with SIMPLE algorithm is assumed; readers unfamiliar with these algorithms are directed to [48].

Figure 4.3 depicts the flowchart of a typical SIMPLE/SIMPLEC algorithm for a co-located unstructured scheme. For the current set of cell velocities and pressure fields, discrete momentum equations are solved for new cell velocities. Next the continuity equation is solved for pressure correction fields (discussed in subsequent sub-sections) with velocities obtained from the most recent momentum equations. Then, cell velocities and pressures are updated based on the pressure correction field. This forms one complete iteration of

the SIMPLE algorithm. The procedure is repeated until convergence as shown in the flowchart. We discuss briefly each of the steps in the flowchart.

To facilitate reading the all the subsequent flow charts, the notations used for the variables are mentioned here. Variables with sub-scripts 0 or 1 associates the variable to cell centroids of the cell $C0$ and cell $C1$ respectively. Variables with the sub-script f associates the variable to the face centroids of the common face between cell $C0$ and cell $C1$.

4.3.2 Discretization of momentum equations

The general transport equation of a scalar quantity ϕ quantifies the net diffusive transport and the convective transport of the scalar quantity ϕ in the presence of an underlying flow field $\vec{v}$ (Add a small section about the general scalar transport equation in Chapter 1). By the same token, the transport equation for u (or v) is viewed as its net diffusive and convective transport with the underlying flow field $\vec{v}$. We start discretizing the the transport equation for u given by,

$$\nabla \cdot \vec{J^u} = -\nabla p \cdot \vec{i} + S_u \tag{4.8}$$

where the total convective and diffusive flux $\vec{J^u}$ is

$$\vec{J^u} = (\rho \vec{v} u - \mu \nabla u) \tag{4.9}$$

The usual process of integration of Eq. 4.8 over cell $C0$ followed by the

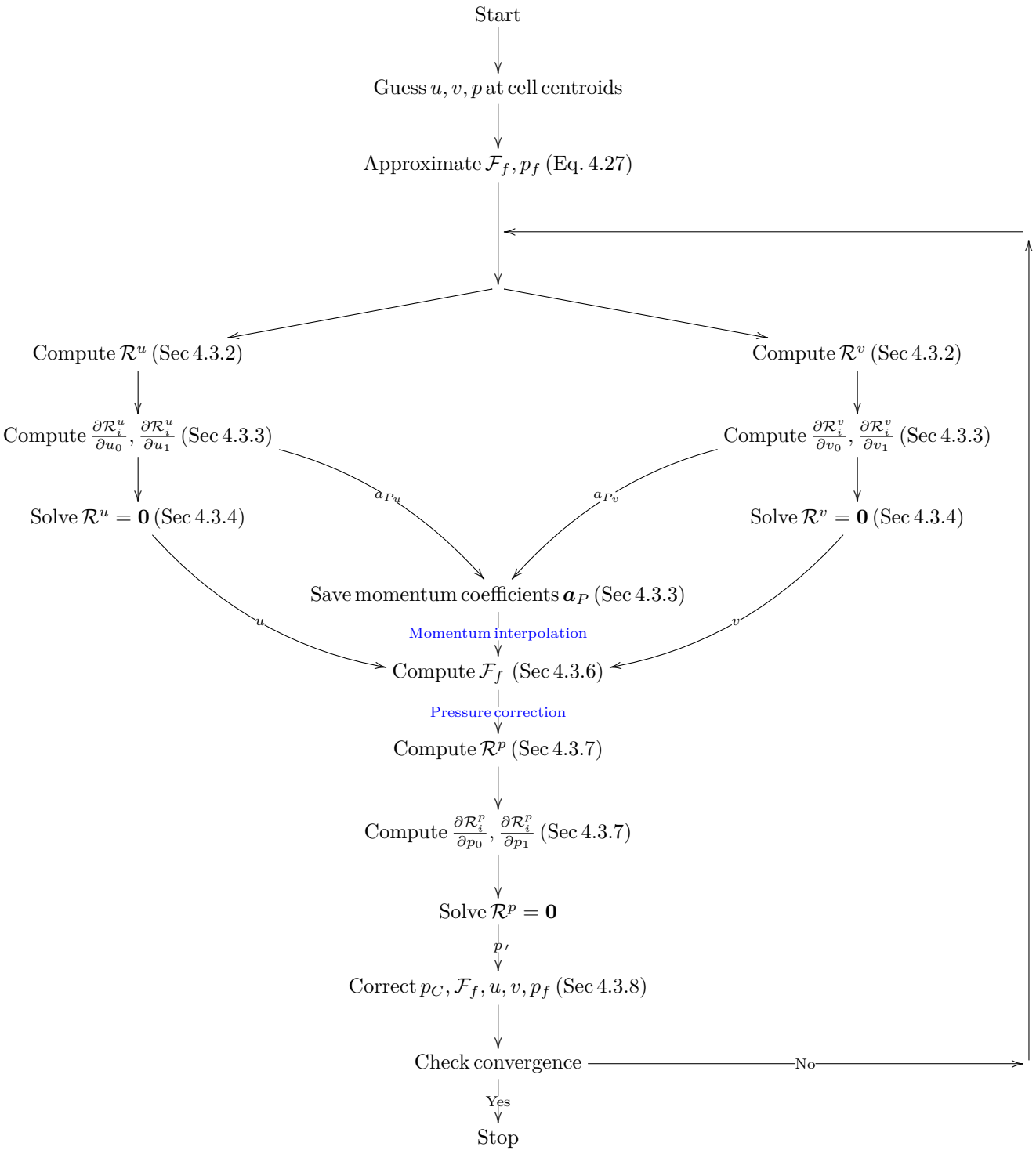


Figure 4.3: Flowchart for a typical SIMPLE algorithm.

application of divergence theorem yields,

$$\int_A \vec{J^u} \cdot d\vec{A} = \int_{\Delta\mathscr{V}_0} \left(-\nabla p \cdot \vec{i}\right) d\mathscr{V} + \int_{\Delta\mathscr{V}_0} S_u d\mathscr{V} \tag{4.10}$$

Source discretization yields,

$$\int_{\Delta\mathscr{V}_0} S_u d\mathscr{V} = \left(\mathcal{S}_u^C + S_u^P \phi_0\right) \Delta\mathscr{V}_0$$

The pressure gradient discretization under the assumption that the face pressure p_f prevails over a entire given face is obtained as,

$$\begin{aligned} \int_{\Delta\mathscr{V}_0} \left(-\nabla p \cdot \vec{i}\right) d\mathscr{V} &= \int_{\Delta\mathscr{V}_0} (-\nabla p d\mathscr{V}) \cdot \vec{i} \\ &= \int_{A_f} \left(-p d\vec{A}_f\right) \cdot \vec{i} \\ &= -\vec{i} \cdot \left(\sum_f p_f \vec{A}_f\right) \end{aligned} \tag{4.11}$$

We denote a discrete field for the pressure gradient associated with each cell $C0$ as ∇p_0. The discretization of pressure gradient term in terms of ∇p_0 is then given by,

$$\int_{\Delta\mathscr{V}_0} \left(-\nabla p \cdot \vec{i}\right) d\mathscr{V} = -\nabla p_0 \Delta\mathscr{V}_0 \tag{4.12}$$

thus discretely implying,

$$-\nabla p_0 \Delta \forall_0 = -\sum_{f}^{nbs} p_f \vec{A}_f \tag{4.13}$$

If the face pressures p_f of all the bounding faces of a cell $C0$ are known, its pressure gradient ∇p_0 can be computed using Eq. 4.13. On the other hand, if neighbor cell pressures and cell velocities of neighbor cells $C0$ and $C1$ are known, then the face pressure p_f of the common face f can be computed. The functional form for computation of p_f is deferred to later sub-sections.

Now, assuming the $x-$momentum flux on the face $\vec{J}^u$ may be written in terms of the face centroid value $\vec{J}^u_f$, Eq. 4.10 approximates to

$$\sum_f \vec{J}^u_f \cdot \vec{A}_f = -\vec{i} \cdot \left(\sum_f p_f \vec{A}_f \right) + \left(\mathcal{S}^C_u + S^P_u \phi_0 \right) \Delta \forall_0 \tag{4.14}$$

where the summation is over the faces f of the cell. The flux is given by

$$\vec{J}^u_f = \left(\rho \vec{v} u\right)_f - \mu_f \left(\nabla u\right)_f \tag{4.15}$$

The transport of u at the face f may thus be written as

$$\vec{J}^u_f \cdot \vec{A}_f = \left(\rho \vec{v}\right)_f \cdot \vec{A}_f u_f - \mu_f \left(\nabla u\right)_f \cdot \vec{A}_f \tag{4.16}$$

Physically the above term is the rate of $x-$momentum transported out of face f.

We define the face mass flow rate $\mathcal{F}_f$ out of the cell $C0$ associated with a face f as

$$\mathcal{F}_f = (\rho\vec{v})_f \cdot \vec{A}_f \tag{4.17}$$

Similar to the diffusion discretization heat equation in Chapter 2, the diffusion transport of the $x-$momentum at the face may be written as

$$-\mu_f (\nabla u)_f \cdot \vec{A}_f = \frac{\mu_f}{\Delta\xi} \frac{\vec{A}_f \cdot \vec{A}_f}{\vec{A}_f \cdot \vec{e}_\xi} (u_1 - u_0) + \mathscr{S}_f^u \tag{4.18}$$

Defining the quantity $\mathcal{D}_f$ as,

$$\mathcal{D}_f = \frac{\mu_f}{\Delta\xi} \frac{\vec{A}_f \cdot \vec{A}_f}{\vec{A}_f \cdot \vec{e}_\xi} \tag{4.19}$$

the net transport of u across the face f is

$$\vec{J}_f^u \cdot \vec{A}_f = \mathcal{F}_f u_f - \mathcal{D}_f (u_1 - u_0) + \mathscr{S}_f^u \tag{4.20}$$

The secondary gradient and the face diffusivity are,

$$\mathscr{S}_f^u = -\mu_f (\nabla u)_f \cdot \vec{A}_f + \frac{\mu_f}{\Delta\xi} \frac{\vec{A}_f \cdot \vec{A}_f}{\vec{A}_f \cdot \vec{e}_\xi} (\nabla\mu)_f \cdot \vec{e}_f \Delta\xi \tag{4.21}$$

$$\mu_f = \frac{2\mu_0\mu_1}{\mu_0 + \mu_1} \tag{4.22}$$

The convective transport of u at the face requires the evaluation of the face velocity component u_f.

A general transport quantity ϕ at the face ϕ_f can be interpolated using a central difference approximation, an upwind difference approximation and a higher order scheme[31, 48]. We use an upwind scheme throughout the dissertation. Under the upwind difference approximation,

$$\begin{aligned} \phi_f &= \phi_0 \quad \text{if } \mathcal{F}_f > 0 && (4.23) \\ &= \phi_1 \quad \text{otherwise} && (4.24) \end{aligned}$$

The same upwinding is performed for the face velocity component u_f. Though upwinding as above is a only a first-order approximation, we believe that this level of accuracy is sufficient for topology optimization, and for determining the conceptual designs that result from it.

Thus the discrete residual of the $x-$momentum equation of cell $C0$ transporting u becomes,

$$\mathcal{R}_i^u = \left(\sum_f - \left[\mathcal{F}_f u_f - \mathcal{D}_f \left(u_1 - u_0 \right) + \mathscr{S}_f^u \right] - \vec{i} \cdot \left(\nabla p_0 \Delta \mathscr{V}_0 \right) + \left(\mathcal{S}_u^C + S_u^P u_0 \right) \Delta \mathscr{V}_0 \right)_i \tag{4.25}$$

Similarly the discrete residual for $y-$momentum equation of cell $C0$ transporting v becomes,

$$\mathcal{R}_i^v = \left(\sum_f - \left[\mathcal{F}_f v_f - \mathcal{D}_f \left(v_1 - v_0 \right) + \mathscr{S}_f^v \right] - \vec{j} \cdot \left(\nabla p_0 \Delta \mathscr{V}_0 \right) + \left(\mathcal{S}_v^C + S_v^P u_0 \right) \Delta \mathscr{V}_0 \right)_i \tag{4.26}$$

At this juncture, the computation of massflux at the faces $\mathcal{F}_f$ given by Eq. 4.17 requires further clarification. In a co-located formulation we compute the velocities and the pressure variables associated with the cell-centroids. However $\mathcal{F}_f$ requires the velocities associated with the faces (or face centroids). The face velocities thus needs to be interpolated appropriately, as discussed in Section 4.3.6. However, during the first iteration of the SIMPLE algorithm where momentum equations are solved, the face velocities and face pressure are obtained as the average of the initial guess of neighboring cell velocities and pressure given by,

$$\begin{aligned} \boldsymbol{V}_f &= 0.5 \cdot \left(\boldsymbol{V}_0 + \boldsymbol{V}_1 \right) \\ p_f &= 0.5 \cdot \left(p_0 + p_1 \right) \end{aligned} \tag{4.27}$$

4.3.3 Jacobians of momentum residual

During the forward mode solution of flow variables, the following two Jacobian terms for the derivative of the $x-$momentum residual *w.r.t* u, the first one being the diagonal term and the second one being the off diagonal term, are computed as

$$\frac{\partial \mathcal{R}_i^u}{\partial u_0} = \sum_f \begin{pmatrix} -\left[\mathcal{F}_f + D_f + S_u^P \Delta \mathscr{V}_0\right] & \text{if } \mathcal{F}_f > 0 \\ -\left[D_f + S_u^P \Delta \mathscr{V}_0\right] & \text{otherwise} \end{pmatrix} \tag{4.28}$$

$$\frac{\partial \mathcal{R}_i^u}{\partial u_1} = \sum_f \begin{pmatrix} -\left[-D_f\right] & \text{if } \mathcal{F}_f > 0 \\ -\left[-\mathcal{F}_f - D_f\right] & \text{otherwise} \end{pmatrix} \tag{4.29}$$

In principle, the convection term ($\mathcal{F}_f u_f$) in the momentum equation is non-linear since the mass flux term $\mathcal{F}_f$ has velocity components within it. However, in the standard implementation of the SIMPLE algorithm, while forming the Jacobian terms in Eqs. 4.28 and 4.29, the convection term is linearized assuming mass flux $\mathcal{F}_f$ is a constant coefficient. Furthermore, the terms $\frac{\partial \mathcal{R}_i^u}{\partial v_0}$ and $\frac{\partial \mathcal{R}_i^u}{\partial v_1}$ of the Jacobian are neglected as a consequence of this assumption. Also residual $\mathcal{R}_i^u$ has the pressure gradient term which depends on the face pressure of its faces which in turn depends on the cell pressure of the neighbors. The derivative of $\mathcal{R}_i^u$ *w.r.t.* cell pressures is not computed while forming its Jacobian because of the sequential nature of solution algorithm. Similarly only the terms $\frac{\partial \mathcal{R}_i^v}{\partial v_0}$ and $\frac{\partial \mathcal{R}_i^v}{\partial v_1}$ are computed for $y-$momentum residual assuming constant $\mathcal{F}_f$, in turn also leading to omission of the terms $\frac{\partial \mathcal{R}_i^v}{\partial u_0}$ and $\frac{\partial \mathcal{R}_i^v}{\partial u_1}$. The pressure dependence is also omitted. In addition, the secondary gradient flux terms of the diffusion discretization terms are not included in the Jacobian for all momentum equations. Thus a complete Jacobian is never computed. All these facts will have implications when we find sensitivity derivatives using $\mathcal{R}$apid AD library.

4.3.4 Solution of momentum equations for u, v and w

Though we restricted to variables in 2D formulation for simplicity until now, we start explicitly mentioning the variables in 3D formulation from now on. The following three nominally linear systems in delta form are individually solved for obtaining cell velocities $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{w}$.

$$\begin{bmatrix} \frac{\partial \boldsymbol{\mathcal{R}}_u}{\partial \boldsymbol{u}} & & \\ & \frac{\partial \boldsymbol{\mathcal{R}}_v}{\partial \boldsymbol{v}} & \\ & & \frac{\partial \boldsymbol{\mathcal{R}}_w}{\partial \boldsymbol{w}} \end{bmatrix} \begin{Bmatrix} \delta_u \\ \delta_v \\ \delta_w \end{Bmatrix} + \begin{Bmatrix} \mathcal{R}_u \\ \mathcal{R}_v \\ \mathcal{R}_w \end{Bmatrix} = \begin{Bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{Bmatrix} \tag{4.30}$$

In practice both momentum equations and continuity equations are under-relaxed during iterative solution[31]. If α_m is the under-relaxation factor chosen for momentum equations and and $\vec{\boldsymbol{V}}^*$ the current velocity of the cell, then the cell velocities are updated based on the solution $\vec{\delta} = \{\delta_u, \delta_v, \delta_w\}$ of Eq. 4.30 as,

$$\boldsymbol{V} \leftarrow \boldsymbol{V}^* + \alpha_m \vec{\delta} \tag{4.31}$$

From implementation perspective, we realize the correction by scaling the Jacobian diagonal term instead of explicitly correcting the velocity after the solve.

4.3.5 Discretization of continuity equation

To discretize the continuity equation, we integrate Eq. 4.6 over the control volume and apply divergence theorem to obtain,

$$
\begin{aligned}
\int_{\mathscr{V}_0} \nabla \cdot (\rho \vec{v})\, d\mathscr{V} &= 0 \\
\implies \int_A (\rho \vec{v}) \cdot d\vec{A} &= 0 \\
\implies \sum_f (\rho \vec{v})_f \cdot \vec{A}_f &= 0 \\
\implies \sum_f \mathcal{F}_f &= 0
\end{aligned} \tag{4.32}
$$

The residual for the continuity equation is therefore

$$
\mathcal{R}_i^p = \left(\sum_{faces} -\mathcal{F}_f \right)_i \tag{4.33}
$$

4.3.6 Momentum interpolation of face velocities

The massflux at the faces $\mathcal{F}_f$ for Eq. 4.33 requires normal velocities at the faces. One way is to obtain face velocity is to linearly interpolate from the cell velocities obtained from the solution of momentum equations. However such an interpolation if used for continuity equations can support pressure checker-boarding in turn supporting velocity checker-boarding in the final solution[48]. The root cause for checker boarding is the discretization of the pressure gradient term in the discretized momentum equations (Eqs. 4.25 and 4.26). The linearly interpolated normal velocity at the face $\overline{V}_f^n$ would have the following functional dependence.

$$
\overline{V}_f^n = \overline{V}_f^n \left(\boldsymbol{V}_0, \boldsymbol{V}_1, \nabla p_0, \nabla p_1 \right)
$$

As a consequence of this linear interpolation involving pressure gradients, face velocities finally get expressed in terms of alternate cell pressure values causing pressure checker-boarding [31]. To prevent such checker-boarding, co-located formulations employ interpolation procedures that express face velocities in terms of the adjacent pressure values rather than alternate pressure values. Such an interpolation is known as '*momentum interpolation*' or as an '*added dissipation scheme*' in the literature[31, 48].

Momentum interpolation prevents checker-boarding of the velocity field by not interpolating the velocities linearly. Here the pressure gradient term resulting from linear interpolation of face velocity is subtracted and a new pressure gradient term written in terms of the pressure difference of the adjacent pressure values $p_0 - p_1$ is added [31]. Therefore we are seeking an momentum-interpolated face normal velocity of the form given by,

$$V_f^n = V_f^n \left(\boldsymbol{V}_0, \boldsymbol{V}_1, \nabla p_f \left(p_0, p_1 \right) \right) \tag{4.34}$$

where the pressure gradient associated with the face ∇p_f depends functionally on adjacent pressure values p_0 and p_1.

We now briefly describe the process of interpolation of the velocities at the face for an unstructured co-located scheme in the most generic case where the coefficients of u, v or w momentum equations can be different. Again the primary intent is to present the flow of function evaluations for the purpose of obtaining sensitivities. For detailed understanding and rationale of these

formulas, the reader is referred to [31, 48].

The diagonal terms of the Jacobian matrices $\left[\frac{\partial \boldsymbol{\mathcal{R}}_u}{\partial \boldsymbol{u}}\right]$, $\left[\frac{\partial \boldsymbol{\mathcal{R}}_v}{\partial \boldsymbol{v}}\right]$ and $\left[\frac{\partial \boldsymbol{\mathcal{R}}_w}{\partial \boldsymbol{w}}\right]$ formed for solving the linear system for $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{w}$ are saved for momentum interpolation. These diagonal terms are commonly referred as the $\boldsymbol{a}_P$ coefficients. The coefficients for cell $C0$ and cell $C1$ are given by,

$$\boldsymbol{a}_{P_i} = \left\{a^u_{P_i}, a^v_{P_i}, a^w_{P_i}\right\} = \left\{\frac{\partial \mathcal{R}^u_i}{\partial u_i}, \frac{\partial \mathcal{R}^v_i}{\partial v_i}, \frac{\partial \mathcal{R}^w_i}{\partial w_i}\right\} \tag{4.35}$$

We define a mean coefficient for each cell referred as to as $\overline{a}_P$ given by,

$$\overline{a}_{P_i} = \frac{\left(a^u_{P_i} + a^v_{P_i} + a^w_{P_i}\right)}{3} \tag{4.36}$$

Consequently, the corresponding coefficient associated with the common face f for cells $C0$ and $C1$ can then be given by,

$$\overline{a}_{P_f} = \overline{a}_{P_0} + \overline{a}_{P_1} \tag{4.37}$$

As discussed before, the momentum interpolation procedure is applied to the normal velocity at the face. The linearly interpolated face normal velocity $\overline{V}^n_f$ from the discretized momentum equations is given by,

$$\overline{V}^n_f = \frac{((\boldsymbol{V}_0 \cdot \boldsymbol{A}_f) \times \overline{a}_{P_0} + (\boldsymbol{V}_1 \cdot \boldsymbol{A}_f) \times \overline{a}_{P_1})}{\overline{a}_{P_0} + \overline{a}_{P_1}} \tag{4.38}$$

where $\boldsymbol{V}_0$ and $\boldsymbol{V}_1$ are the velocities from the converged momentum equations of the current iteration. The mean pressure gradient of the face dotted the unit vector $\vec{e}_f$, termed as $\overline{\nabla p}_f$, is computed from the pressure gradient of the cells ∇p_0 and ∇p_1 as,

$$\overline{\nabla p}_f = \Delta\forall_0 \left(\nabla p_0 \cdot \Delta\xi\vec{e}_f\right) + \Delta\forall_1 \left(\nabla p_1 \cdot \Delta\xi\vec{e}_f\right) \tag{4.39}$$

Implementing the idea of momentum interpolation, we subtract this component of pressure gradient $\overline{\nabla p}_f$, in the direction of cell centroids, from $\overline{V}_n^f$ to obtain $\hat{V}_f^n$

$$\hat{V}_f^n = \frac{\left(\left(\boldsymbol{V}_0 \cdot \boldsymbol{A}_f\right) \times \overline{a}_{P_0} + \left(\boldsymbol{V}_1 \cdot \boldsymbol{A}_f\right) \times \overline{a}_{P_1} - \overline{\nabla p}_f \frac{\vec{A}_f \cdot \vec{A}_f}{\vec{A}_f \cdot \Delta\xi\vec{e}_\xi}\right)}{\overline{a}_{P_0} + \overline{a}_{P_1}} \tag{4.40}$$

A term d_f, which is the coefficient for the new pressure gradient term $\nabla p_f\left(p_0, p_1\right)$, is computed from the a_p coefficients of the cells as,

$$d_f = \left(\frac{A_f^x A_f^x}{a_{P_0}^u + a_{P_1}^u} + \frac{A_f^y A_f^y}{a_{P_0}^v + a_{P_1}^v} + \frac{A_f^z A_f^z}{a_{P_0}^w + a_{P_1}^w}\right) \frac{\left(\Delta\forall_0 + \Delta\forall_1\right)}{\vec{A}_f \cdot \vec{e}_\xi} \tag{4.41}$$

The manipulation results in adding a pressure gradient term associated with the gradient $\frac{\partial p}{\partial \xi}$ rather than $\frac{\partial p}{\partial n}$, since this is the only gradient that can be directly associated with the adjacent pressure difference $p_1 - p_0$. Thus the pressure gradient term $\nabla p_\xi\left(p_0, p_1\right)$ is given by,

$$\nabla p_\xi \left(p_0, p_1\right) = -d_f \left(p_0 - p_1\right) \tag{4.42}$$

After adding the above pressure gradient to $\hat{V}_f^n$, we obtain the interpolated face velocity V_f^n as

$$V_f^n = \hat{V}_f^n + \nabla p_\xi \left(p_0, p_1\right) = \hat{V}_n^f - d_f \left(p_0 - p_1\right) \tag{4.43}$$

Finally we obtain the mass flux at the face obtain through the process of momentum interpolation given by,

$$\mathcal{F}_f = \rho_f \left[\hat{V}_f^n - d_f \left(p_0 - p_1\right)\right] \tag{4.44}$$

Here the mean density ρ_f for the the face is

$$\rho_f = \frac{\left(\rho_0 + \rho_1\right)}{2} \tag{4.45}$$

Again in practice, the momentum interpolated mass flux at the faces are under-relaxed based on the momentum under-relaxation factor α_m. If $\mathcal{F}_f^\dagger$ is the most recent mass flux at the face, then the mass flux is computed as

$$\mathcal{F}_f \leftarrow \rho_f \left[\hat{V}_f^n - d_f \left(p_0 - p_1\right)\right] + \left(1 - \alpha_m\right) \mathcal{F}_f^\dagger \tag{4.46}$$

4.3.7 Solution of pressure correction equation

Before convergence the momentum interpolated mass flux (Eq. 4.44/4.46) does not satisfy continuity field. The residual of the continuity equation (Eq. 4.33) using the interpolated mass flux physically represents the total mass source of the cell and is not zero. Therefore the algorithm should drive the residual to zero thus achieving the continuity satisfying velocity field. Our intent is to determine a mass flux correction term $\mathcal{F}_f^{'}$ that when added to the most recent momentum interpolated mass flux $\mathcal{F}_f^*$(Eq. 4.44/4.46), would result in a continuity satisfying velocity field in the cell, i.e.,

$$\left(\sum_{faces} -\mathcal{F}_f\right)_i = 0 \tag{4.47}$$

where $\mathcal{F}_f$ is the corrected mass flux given by,

$$\mathcal{F}_f \leftarrow \mathcal{F}_f^* + \mathcal{F}_f^{'} \tag{4.48}$$

Based on Eq. 4.44 and keeping with the SIMPLE philosophy, the mass flux correction term is proposed solely in terms of pressure correction field $\boldsymbol{p}^{'}$ and dropping the velocity term $\hat{V}_f^n$. The mass flux correction $\mathcal{F}_f^{'}$ and its relation to the change in pressure difference across the face f is assumed to be:

$$\mathcal{F}_f^{'} = \rho_f \left[-d_f \left(p_0^{'} - p_1^{'}\right)\right] \tag{4.49}$$

To solve for $\boldsymbol{p}^{'}$, we may recast the nominally linear system in delta form as,

$$\left[\frac{\partial \boldsymbol{\mathcal{R}}_p}{\partial \boldsymbol{p}}\right]\left\{\boldsymbol{p}^{'}\right\} + \boldsymbol{\mathcal{R}}_p = 0 \tag{4.50}$$

We also note that instead of using the variable δ as the vector to be solved, we term it as $\boldsymbol{p}^{'}$ itself, since after all we are interested in a correction field. The partial Jacobian terms for cells $C0$ and $C1$ with respect to pressure variables used in the pressure-correction linear system are,

$$\frac{\partial \mathcal{R}_p^0}{\partial p_0} = \frac{\partial \mathcal{R}_p^1}{\partial p_1} = \rho_f d_f \tag{4.51}$$

$$\frac{\partial \mathcal{R}_p^0}{\partial p_1} = \frac{\partial \mathcal{R}_p^1}{\partial p_0} = -\rho_f d_f \tag{4.52}$$

Notice that the terms $\left[\frac{\partial \boldsymbol{\mathcal{R}}^p}{\partial \boldsymbol{u}}\right]$, $\left[\frac{\partial \boldsymbol{\mathcal{R}}^p}{\partial \boldsymbol{v}}\right]$ and $\left[\frac{\partial \boldsymbol{\mathcal{R}}^p}{\partial \boldsymbol{w}}\right]$ are not used in the pressure correction equation because of the nature of the sequential algorithm.

4.3.8 Correction of pressure and velocity fields

Once the pressure correction equation is solved and the $\boldsymbol{p}^{'}$ field is obtained (Eq. 4.50), the cell pressure is corrected as,

$$p_i \leftarrow p_i^* + \alpha_p p_i^{'} \tag{4.53}$$

where α_p is the under-relaxation factor for pressure correction equation. The face pressure is corrected based on the corrected cell pressures as,

$$p_f = \left(\frac{d_0 p_0 + d_1 p_1}{d_0 + d_1} \right) \tag{4.54}$$

where d_0 and d_1 are computed based on momentum a_P coefficients as [31, 48],

$$d_0 = \left(\frac{A_f^x}{a_{P_0}^u} + \frac{A_f^y}{a_{P_0}^v} + \frac{A_f^z}{a_{P_0}^w} \right) \frac{(\Delta \mathscr{V}_0 \rho_0)}{\vec{A}_f \cdot \vec{e}_\xi}; \;\; d_1 = \left(\frac{A_f^x}{a_{P_1}^u} + \frac{A_f^y}{a_{P_1}^v} + \frac{A_f^z}{a_{P_1}^w} \right) \frac{(\Delta \mathscr{V}_1 \rho_1)}{\vec{A}_f \cdot \vec{e}_\xi} \tag{4.55}$$

The massflux at the faces is updated after computing $\mathcal{F}_f^{'}$ with $\boldsymbol{p}^{'}$ as,

$$\mathcal{F}_f^{'} = - \left[\frac{\partial \mathcal{R}_p^0}{\partial p_1} \right] p_1^{'} - \left[\frac{\partial \mathcal{R}_p^1}{\partial p_0} \right] p_0^{'} \tag{4.56}$$

$$\mathcal{F}_f \leftarrow \mathcal{F}_f^* + \mathcal{F}_f^{'} \tag{4.57}$$

For the co-located version of the SIMPLE algorithm, the face pressure and the cell velocities are also corrected [31, 48]. The face pressure is corrected as

$$p_f^{'} = \left(\frac{d_0 p_0^{'} + d_1 p_1^{'}}{d_0 + d_1} \right) \tag{4.58}$$

The cell velocity corrections are computed as [31, 48],

$$\vec{V}_0^{'} = \left\{ \frac{p_f^{'} A_f^x}{a_{P_0}^u}, \frac{p_f^{'} A_f^y}{a_{P_0}^v}, \frac{p_f^{'} A_f^z}{a_{P_0}^w} \right\}; \;\; \vec{V}_1^{'} = \left\{ \frac{p_f^{'} A_f^x}{a_{P_1}^u}, \frac{p_f^{'} A_f^y}{a_{P_1}^v}, \frac{p_f^{'} A_f^z}{a_{P_1}^w} \right\} \tag{4.59}$$

Finally the cell velocities are corrected as [31, 48],

$$\vec{V_0} \leftarrow \vec{V_0} + \vec{V_0}'; \ \vec{V_1} \leftarrow \vec{V_1} - \vec{V_1}' \tag{4.60}$$

The above procedure has described the SIMPLE algorithm for co-located pressure-velocity schemes. The SIMPLEC algorithm, a variant of SIMPLE[48], which converges somewhat faster than SIMPLE, is easily implemented using the same framework. This is accomplished simply by choosing the under-relaxation factors for momentum and continuity equations such that $\alpha_p = 1 - \alpha_m$.

The operations mentioned in this subsection are needed only for the forward solution of the flow problem. They are not needed for sensitivity computation.

4.4 Flow model sensitivities with $\mathcal{R}$apid AD library

By now we understand that the co-located SIMPLE algorithm for pressure-velocity coupling does not assemble the complete Jacobian during the solution process. However sensitivity computations using the adjoint method need the complete Jacobian. We intend to re-compute residuals and cost functions in the $\mathcal{R}$apid mode with chosen independent variables that would automatically compute the complete Jacobian with respect to all the independent variables.

We follow the same procedure outlined in Section 3.6, Chapter 3 for

obtaining residual derivatives of flow equations. We first obtain the converged values of the flow variables from the forward solution of the flow model (Figure 4.3) in *double* mode. The converged flow variables are then transferred to the code compiled with the $\mathcal{R}$apid type.

A preferred situation would be to able to blindly follow the sequence of steps of SIMPLE algorithm used for forward solution to compute residuals in $\mathcal{R}$apid mode, automatically leading to the computation of the complete Jacobians. However in retrospect, one would understand that such a blind procedure results in inaccurate sensitivities. This is primarily because of non-linear coupled nature of flow equations and the methodology devised in SIMPLE for obtaining equations for pressure. We have to devise algorithm specifically to obtain sensitivities accurately. We present three methods which illustrate the underlying issues.

4.4.1 Method 1: Traversal of SIMPLE algorithm with independent cell variables

The first step while using the $\mathcal{R}$apid mode is to choose the set of independent variables with respect to which derivatives are desired. As mentioned, in a co-located formulation, flow variables are associated with cell centroids. Variables associated with faces, such as face velocities (or face mass flux) and face pressure, can be derived from their cell counterparts. In obtaining sensitivities, there may be imperatives such as minimizing code-intrusion and obtaining problem-agnostic implementations.

For the SIMPLE algorithm depicted in Figure 4.3, the residual variables $\mathcal{R}^u, \mathcal{R}^v$, $\boldsymbol{\mathcal{R}}^w$ and $\boldsymbol{\mathcal{R}}^p$ are dependent on the cell velocity components (u, v, w) and cell pressures $\boldsymbol{p}$. It is therefore natural to select cell velocities and pressure variables as the independent state variables. Face variables can be dependent functions of their cell counterparts.

Figure 4.4 depicts the process of obtaining derivatives of residuals following the SIMPLE algorithm depicted in Figure 4.3 in $\mathcal{R}$apid mode. As mentioned in Section 3.6, Chapter 3, minimal modifications are needed while executing the code in $\mathcal{R}$apid mode. This is pictorially represented with the color coded arrows in the flowchart in Figure 4.4. Here, we traverse the flow chart following the red arrows for both the forward solution with double mode and derivative computation with the $\mathcal{R}$apid mode. However we skip the steps depicted in the black arrows in the $\mathcal{R}$apid mode. We follow the green arrows only in the $\mathcal{R}$apid mode. Steps related to solution of the linear system and the correction steps that follow are skipped.

While performing automatic differentiation, it is important that the functional dependencies are properly captured. Consider the computation of the u-momentum residual $\mathcal{R}_i^u$ (Eq. 4.25) for a particular cell, using the flowchart in Figure 4.4. We need cell velocities for the diffusion term, face and cell velocities for convection term, face pressures for pressure gradient term as well as linearized source terms for the computation of $\mathcal{R}_i^u$. The face velocities and pressure are approximated with neighbor cell velocities and pressures with Eq. 4.27. Computation of $\mathcal{R}_i^u$ will produce its complete Jacobian terms

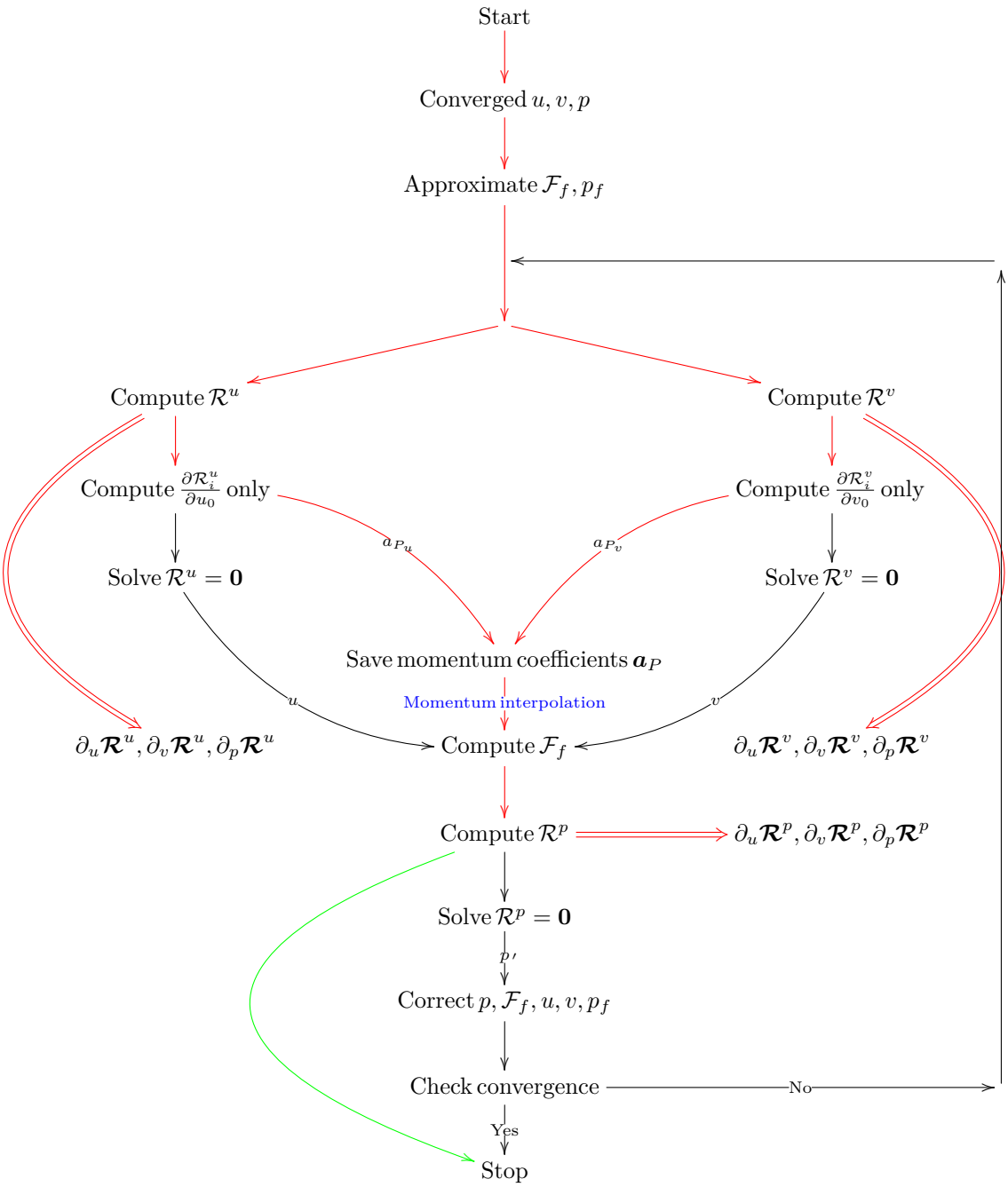


Figure 4.4: Method 1: Sequence of steps of the SIMPLE algorithm depicted in Figure 4.3 in $\mathcal{R}$apid mode. Traversal along red arrows is performed both in the forward solve in double mode and in the $\mathcal{R}$apid mode to get the desired sensitivities. We do not follow the black arrows in $\mathcal{R}$apid mode. Traversal along the green arrow is performed only in $\mathcal{R}$apid mode.

$\partial_u \boldsymbol{\mathcal{R}}^u, \partial_v \boldsymbol{\mathcal{R}}^u, \partial_p \boldsymbol{\mathcal{R}}^u$ per the defined functional dependencies.

It is important to address the discretization of convection term and the pressure gradient term. The dependency of face velocity solely being a function of neighbor velocities (Eq. 4.27) is reflected while computing the convection term $\mathcal{F}_f u_f$ of $\mathcal{R}_i^u$. However the face mass flux $\mathcal{F}_f$ has much more complex dependence on more neighbor cell velocities and cell pressures through the process of momentum interpolation (Eq. 4.44). The pressure gradient term of a cell is computed based on the face pressure of its faces. Though we approximate the face pressure with neighbor cell pressures, it has much more complex dependence of the neighbor cell pressures and velocities as shown in Eq. 4.54. These dependencies are not captured while computing $\mathcal{R}_i^u$ in Figure 4.4. Thus the Jacobian terms $\partial_u \boldsymbol{\mathcal{R}}^u, \partial_v \boldsymbol{\mathcal{R}}^u, \partial_p \boldsymbol{\mathcal{R}}^u$ would be not only inaccurate but will also be missing the dependencies on many cell and pressure variables. The same is true for $\mathcal{R}_i^v$.

Going forward with the algorithm, we re-compute the face mass flux $\mathcal{F}_f$ using momentum interpolation with the a_P momentum coefficients obtained from the momentum residual computations. This mass flux is used to compute the continuity residual $\mathcal{R}_i^p$. This re-computed mass flux reflects the true dependence on all neighbor cell velocities and pressure variables with the right functional form. This residual computation is followed by the computation of all its Jacobian terms $\partial_u \boldsymbol{\mathcal{R}}^p, \partial_v \boldsymbol{\mathcal{R}}^p, \partial_p \boldsymbol{\mathcal{R}}^p$.

4.61Once we get all the Jacobian terms, the adjoint system for solving the adjoint variables associated with each flow variable can be assembled in

the following way. Here the RHS is the partial derivative of the cost function (QoI) $w.r.t$ each flow variable u, v, w and p.

$$\begin{bmatrix} \underbrace{\partial_u \boldsymbol{\mathcal{R}}^u}_{n_C \times n_C} & \partial_v \boldsymbol{\mathcal{R}}^u & \partial_w \boldsymbol{\mathcal{R}}^u & \partial_p \boldsymbol{\mathcal{R}}^u \\ \partial_u \boldsymbol{\mathcal{R}}^v & \partial_v \boldsymbol{\mathcal{R}}^v & \partial_w \boldsymbol{\mathcal{R}}^v & \partial_p \boldsymbol{\mathcal{R}}^v \\ \partial_u \boldsymbol{\mathcal{R}}^w & \partial_v \boldsymbol{\mathcal{R}}^w & \partial_w \boldsymbol{\mathcal{R}}^w & \partial_p \boldsymbol{\mathcal{R}}^w \\ \partial_u \boldsymbol{\mathcal{R}}^p & \partial_v \boldsymbol{\mathcal{R}}^p & \partial_w \boldsymbol{\mathcal{R}}^p & \partial_p \boldsymbol{\mathcal{R}}^p \end{bmatrix} \begin{Bmatrix} \psi_u \\ \psi_v \\ \psi_w \\ \psi_{p_C} \end{Bmatrix} = \begin{Bmatrix} \partial_u c \\ \partial_v c \\ \partial_w c \\ \partial_p c \end{Bmatrix} \tag{4.61}$$

Here n_C is the number of cells. All terms in the Jacobian (Eq. 4.61) sparse matrices of size $n_C \times n_C$.

4.4.2 Method 2: Algorithm with independent cell and face variables

Having conceptually understood that the Jacobians of momentum residuals can be inaccurate in Method 1 because of the inability to use the momentum interpolation functional form for mass flux (Eq. 4.25) and face pressure (Eq. 4.54), we can conceptually device an algorithm as depicted in Figure 4.5.

At the end of the forward solution we have accurate values for cell velocities, cell pressures, face mass fluxes and face pressures. We can set each of these cell and face variables as independent variables. As depicted in Figure 4.5, we start with converged values of $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{p}, \mathcal{F}_f$ and $\boldsymbol{p}_f$ that have been defined as independent variables in $\mathcal{R}$apid mode. We can right away compute all the momentum residuals (Eqs. 4.25 and 4.26) and continuity residual (Eq. 4.33) in $\mathcal{R}$apid mode. Since the $u-$momentum residual $\boldsymbol{\mathcal{R}}^u$ is now a function of five independent variables $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{p}, \mathcal{F}_f$ and $\boldsymbol{p}_f$, $\mathcal{R}$apid generates all the

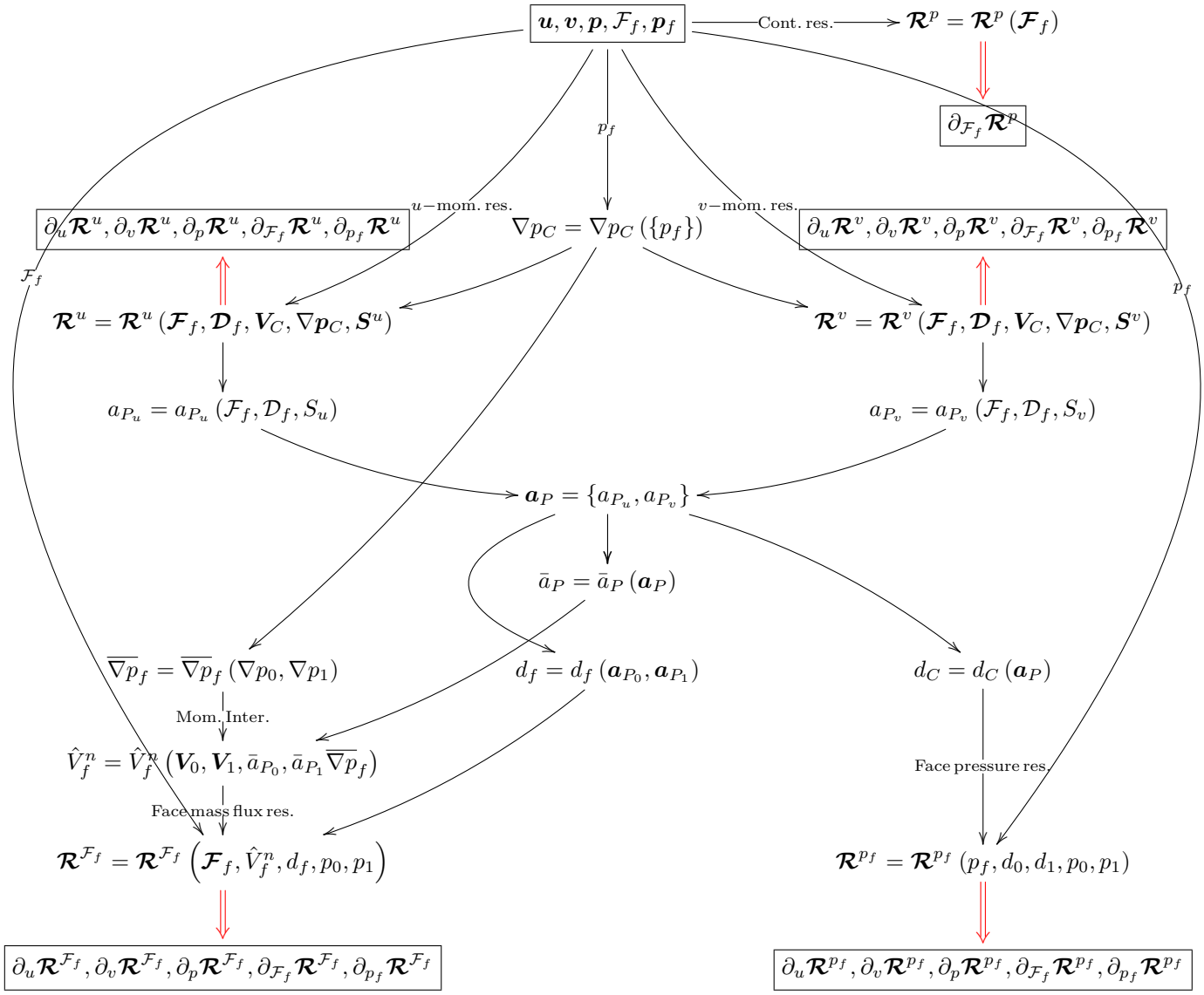


Figure 4.5: Method 2: A method (Section 4.4.2) of obtaining the the accurate complete Jacobian of the coupled momentum and continuity equations, required for adjoint based sensitivity computation. Here the cell velocities, cell and face pressures and face mass fluxes are defined as independent variables. The figure illustrates the sequence of steps needed to be performed to obtain the complete Jacobian in $\mathcal{R}$apid mode.

Jacobian terms $\partial_u \boldsymbol{\mathcal{R}}^u, \partial_v \boldsymbol{\mathcal{R}}^u, \partial_p \boldsymbol{\mathcal{R}}^u, \partial_{\mathcal{F}_f} \boldsymbol{\mathcal{R}}^u$ and $\partial_{p_f} \boldsymbol{\mathcal{R}}^u$ as per the functional dependence of the discrete momentum equation. The same is the case with the $v-$momentum residual $\boldsymbol{\mathcal{R}}^v$. Since face massflux $\mathcal{F}_f$ is an independent variable, the definition for the continuity residual is directly in terms of the mass flux given by Eq. 4.33. Its computation is accompanied by the generation of the derivative term $\partial_{\mathcal{F}_f} \boldsymbol{\mathcal{R}}^p$.

In addition, new residual variables, termed $\boldsymbol{\mathcal{R}}^{\mathcal{F}_f}$ and $\boldsymbol{\mathcal{R}}^{p_f}$, must be defined and properly computed for the new additional independent variables $\mathcal{F}_f$ and $\boldsymbol{p}_f$.

We discuss the definition of $\boldsymbol{\mathcal{R}}^{\mathcal{F}_f}$ first. As shown in Figure 4.5, cell pressure gradient is computed from face pressure using Eq. 4.13. Subsequently, the average face pressure gradient is computed using Eq. 4.39. The velocity term $\hat{V}_f^n$(Eq. 4.40) and the term d_f(using Eq. 4.41 from the momentum $\boldsymbol{a}_p$ coefficients) are computed thereafter. We are now in a position to define the residual $\mathcal{R}^{\mathcal{F}_f}$ of a given face f by,

$$\mathcal{R}^{\mathcal{F}_f} = \mathcal{F}_f - \rho_f \left[\hat{V}_f^n - d_f \left(p_0 - p_1\right)\right] \tag{4.62}$$

It is specifically noted that the face flux term $\mathcal{F}_f$ used in Eq. 4.62 is the converged independent variable as depicted in the figure. The computation of the above residual generates the terms $\partial_u \boldsymbol{\mathcal{R}}^{\mathcal{F}_f}, \partial_v \boldsymbol{\mathcal{R}}^{\mathcal{F}_f}, \partial_p \boldsymbol{\mathcal{R}}^{\mathcal{F}_f}, \partial_{\mathcal{F}_f} \boldsymbol{\mathcal{R}}^{\mathcal{F}_f}$ and $\partial_{p_f} \boldsymbol{\mathcal{R}}^{\mathcal{F}_f}$.

Similarly the residual for face pressure $\mathcal{R}^{p_f}$ of a face f is defined as

$$\mathcal{R}^{p_f} = p_f - \left(\frac{d_0 p_0 + d_1 p_1}{d_0 + d_1} \right) \tag{4.63}$$

where the coefficient for the cell d_C is computed using 4.55 from momentum $\boldsymbol{a}_P$ coefficients as depicted in the figure. The face pressure p_f used in $\mathcal{R}^{p_f}$ is the independent converged value of face pressure as shown in the figure. The computation of residual $\boldsymbol{\mathcal{R}}^{p_f}$ generates its complete Jacobian terms $\partial_u \boldsymbol{\mathcal{R}}^{p_f}, \partial_v \boldsymbol{\mathcal{R}}^{p_f}, \partial_p \boldsymbol{\mathcal{R}}^{p_f}, \partial_{\mathcal{F}_f} \boldsymbol{\mathcal{R}}^{p_f}$ and $\partial_{p_f} \boldsymbol{\mathcal{R}}^{p_f}$.

Introducing new independent variables also affects the adjoint system used for sensitivity computation. New adjoint variables $\psi_{\mathcal{F}_f}$ and ψ_{p_f} must be added. The Jacobian also increases in size. If n_C and n_f are the number of cells and number of faces in the mesh, then the size of Jacobian for a 3D flow problem would be $(4n_C + 2n_f) \times (4n_C + 2n_f)$.

$$\begin{bmatrix} \underbrace{\partial_u \boldsymbol{\mathcal{R}}^u}_{n_C \times n_C} & \partial_v \boldsymbol{\mathcal{R}}^u & \partial_w \boldsymbol{\mathcal{R}}^u & \partial_p \boldsymbol{\mathcal{R}}^u & \underbrace{\partial_{\mathcal{F}_f} \boldsymbol{\mathcal{R}}^u}_{n_f \times n_f} & \partial_{p_f} \boldsymbol{\mathcal{R}}^u \\ \partial_u \boldsymbol{\mathcal{R}}^v & \partial_v \boldsymbol{\mathcal{R}}^v & \partial_w \boldsymbol{\mathcal{R}}^v & \partial_p \boldsymbol{\mathcal{R}}^v & \partial_{\mathcal{F}_f} \boldsymbol{\mathcal{R}}^v & \partial_{p_f} \boldsymbol{\mathcal{R}}^v \\ \partial_u \boldsymbol{\mathcal{R}}^w & \partial_v \boldsymbol{\mathcal{R}}^w & \partial_w \boldsymbol{\mathcal{R}}^w & \partial_p \boldsymbol{\mathcal{R}}^w & \partial_{\mathcal{F}_f} \boldsymbol{\mathcal{R}}^w & \partial_{p_f} \boldsymbol{\mathcal{R}}^w \\ & & & & \partial_{\mathcal{F}_f} \boldsymbol{\mathcal{R}}^p & \\ \partial_u \boldsymbol{\mathcal{R}}^{\mathcal{F}_f} & \partial_v \boldsymbol{\mathcal{R}}^{\mathcal{F}_f} & \partial_w \boldsymbol{\mathcal{R}}^{\mathcal{F}_f} & \partial_p \boldsymbol{\mathcal{R}}^{\mathcal{F}_f} & \partial_{\mathcal{F}_f} \boldsymbol{\mathcal{R}}^{\mathcal{F}_f} & \partial_{p_f} \boldsymbol{\mathcal{R}}^{\mathcal{F}_f} \\ \partial_u \boldsymbol{\mathcal{R}}^{p_f} & \partial_v \boldsymbol{\mathcal{R}}^{p_f} & \partial_w \boldsymbol{\mathcal{R}}^{p_f} & \partial_p \boldsymbol{\mathcal{R}}^{p_f} & \partial_{\mathcal{F}_f} \boldsymbol{\mathcal{R}}^{p_f} & \partial_{p_f} \boldsymbol{\mathcal{R}}^{p_f} \end{bmatrix} \begin{Bmatrix} \psi_u \\ \psi_v \\ \psi_w \\ \psi_{p_C} \\ \psi_{\mathcal{F}_f} \\ \psi_{p_f} \end{Bmatrix} = \begin{Bmatrix} \partial_u c \\ \partial_v c \\ \partial_w c \\ \partial_p c \\ \partial_{\mathcal{F}_f} c \\ \partial_{p_f} c \end{Bmatrix} \tag{4.64}$$

It is clear that the implementation depicted in Figure 4.5 produces all the required derivatives accurately to compute sensitivities based on the adjoint method. However this approach has larger computational overheads. Generally the number of faces n_f are approximately 1.5 times that of number

of cells n_C. Thus the number of independent variables is nearly doubled, increasing the time required for automatic differentiation by the $\mathcal{R}$apid library. Moreover, two more residual vectors $\boldsymbol{\mathcal{R}}^{\mathcal{F}_f}$ and $\boldsymbol{\mathcal{R}}^{p_f}$, associated with each face of the mesh, must be added. These are only to be used in the code when compiled in $\mathcal{R}$apid mode. The increased size of the Jacobian for the adjoint system also increases the time required for the the adjoint solution.

In addition to the computational overhead, this method requires considerable modification or addition of code to the existing forward solution infrastructure. This can be inferred to an extent by comparing the Figures 4.5 and 4.4. Our motivation was again to obtain a problem-agnostic method for obtaining accurate enough sensitivities with minimal modification of the existing infrastructure. Ideally we still desire an infrastructure similar to the one in Figure 4.4. This brings us to the third method described next. The design of the $\mathcal{R}$apid library actually facilitates this objective very effectively. In fact this was one of the main guiding principles for designing the library.

4.4.3 Method 3: Modified algorithm with independent cell variables

Similar to Method 1, we only set the converged values of cell variables $\boldsymbol{u}, \boldsymbol{v}$ and $\boldsymbol{p}$ as the independent variables. Residuals are required for these variables solely, which already exist for a typical flow solver depicted in Figure 4.3. The modified algorithm employing independent cell variables is depicted in Figure 4.6. Most of the steps in Figure 4.3 are followed here, however with

changes in the executing the sequence of steps.

We explained the significance of properly capturing the functional dependence of face mass flux and face pressure on neighbor (and next-neighbor) cell velocities and cell pressures arising from the momentum interpolation (Eq. 4.44), for use in convection and pressure gradient discretized terms of momentum equations. This was not obtained in Figure 4.4. To account for such dependency here, we swap the order of computation of the momentum and continuity residuals. Unlike Method 1 (Figure 4.4), we first compute continuity residual followed by computation of momentum residual. We describe the steps depicted in Figure 4.6 in order.

To start, we compute an approximate face mass flux termed $\mathcal{F}_f^*$ based on just average neighbor cell velocities (Eq. 4.27). The momentum $\boldsymbol{a}_P$ coefficients require only discretized convection, diffusion and source terms. The discretized pressure gradient discretized term does not enter the computation at this stage. There is no need for an approximate face pressure here. Thus based on the approximate mass flux term $\mathcal{F}_f^*$, diffusion coefficient and S_P source terms, momentum coefficients a_{P_u} and a_{P_v} are computed for the u and v momentum equations based on Eq. 4.28. An average momentum coefficient term $\bar{a}_P$ associated with each cell is subsequently computed based on Eq. 4.36. By now it is important to understand that the $\bar{a}_P$ of a cell will have functional dependencies of the various operators of momentum equation in terms of cell and its neighbor cell velocities. For non-orthogonal meshes, next-neighbor cell velocity dependencies are also captured arising from secondary gradient flux

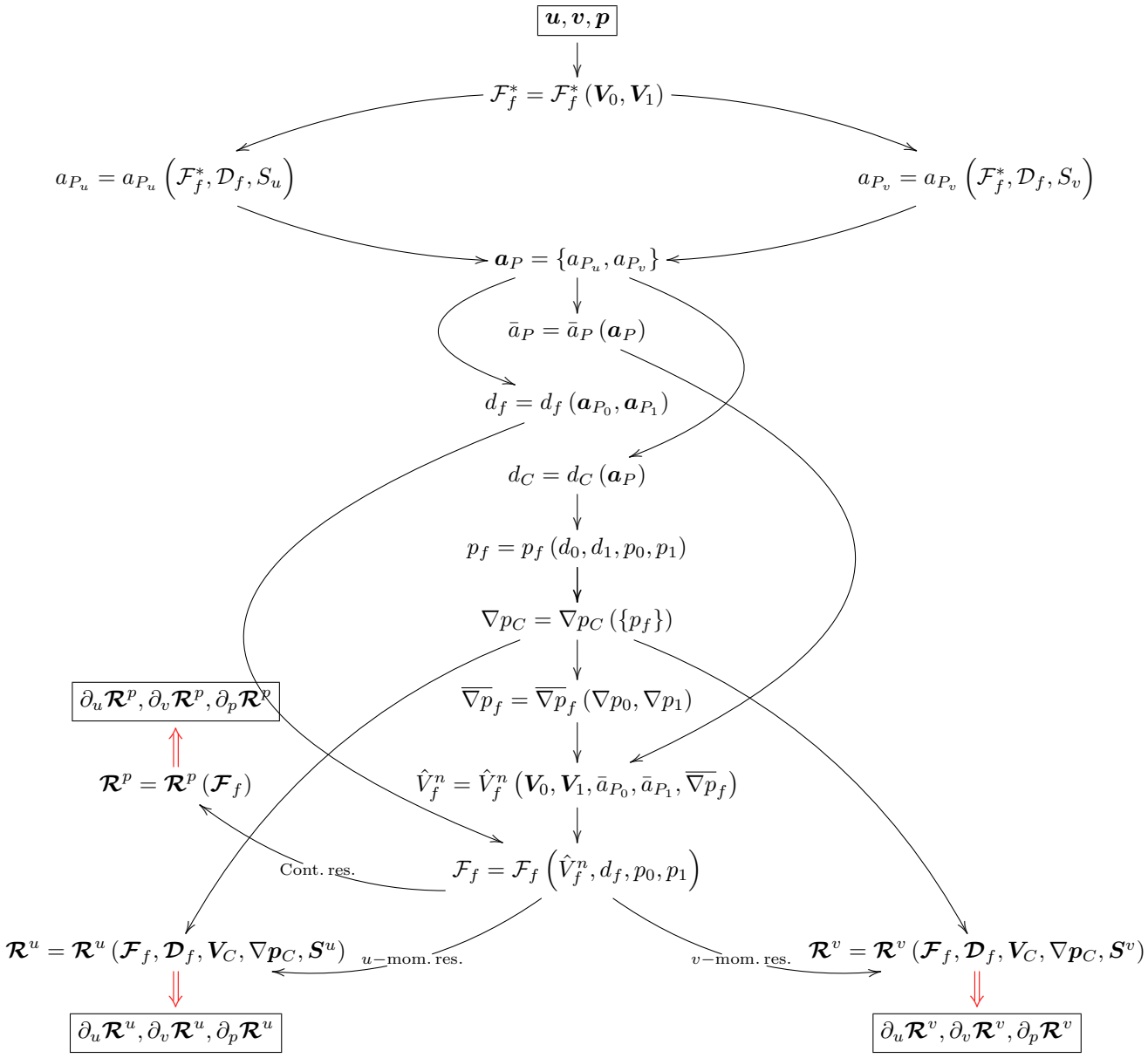


Figure 4.6: Method 3: Computation of complete Jacobian of the coupled momentum and continuity equations (Section 4.4.3). Here only cell velocities and pressure variables are defined as independent variables. The figure illustrates the sequence of steps needed to be performed to obtain the complete Jacobian in $\mathcal{R}$apid mode.

computation [Eq. 2.6]. $\mathcal{R}$apid captures all these dependencies while computing derivatives and stores them in its derivative field, given by,

$$\begin{aligned} \bar{a}_P &= \bar{a}_P\left(u_P, u_{nb1}, u_{nb2} \ldots v_P, v_{nb1}, v_{nb2} \ldots\right) \\ \partial\left(\bar{a}_P\right) &= \left\{\partial_{u_P}\left(\bar{a}_P\right), \partial_{u_{nb1}}\left(\bar{a}_P\right), \partial_{u_{nb2}}\left(\bar{a}_P\right) \ldots, \partial_{v_P}\left(\bar{a}_P\right), \partial_{v_{nb1}}\left(\bar{a}_P\right), \partial_{v_{nb2}}\left(\bar{a}_P\right)\right\} \end{aligned} \tag{4.65}$$

We need face pressure p_f for both momentum interpolation of the massflux and later for pressure gradient discretization in the momentum equations. The term d_C associated with each cell is computed from the a_P coefficients using Eq. 4.55. The face pressure p_f is interpolated with pressure and the d_C values of neighbor cells using Eq. 4.54. At this juncture, we note the dependencies that $\mathcal{R}$apid has captured for p_f,

$$\begin{aligned} p_f &= p_f\left(u_0, u_1, u_{nb1} \ldots v_0, v_1, v_{nb1} \ldots p_0, p_1\right) \\ \partial\left(p_f\right) &= \left\{\partial_{u_0}\left(p_f\right), \partial_{u_1}\left(p_f\right), \partial_{u_{nb1}}\left(p_f\right) \ldots, \partial_{v_0}\left(p_f\right), \partial_{v_1}\left(p_f\right), \partial_{v_{nb1}}\left(p_f\right) \ldots p_0, p_1\right\} \end{aligned} \tag{4.66}$$

The pressure gradient of a cell $\overline{\nabla p}_C$ is computed from the face pressure of all its bounding faces denoted by $\{p_f\}$. Notice how the effect is cascaded in the $\mathcal{R}$apid variable for $\overline{\nabla p}_C$ capturing the dependencies on neighbor cell velocities and cell pressures. The cell pressure gradients thus computed will be later used for momentum residual computation.

Next we proceed to compute the face massflux using momentum interpolation. The face velocity term $\hat{V}_f^n$ (Eq. 4.40) is computed using the average

face pressure gradient $\overline{\nabla p}_f$(Eq. 4.13) and average momentum coefficients $\bar{a}_P$ (Eq. 4.36). Finally a pressure coefficient term d_f associated with the face is computed using Eq. 4.41. Face mass flux $\mathcal{F}_f$ is thus interpolated with all these computed terms using Eq. 4.44. We visualize the $\mathcal{R}$apid data stucture for $\mathcal{F}_f$ for a face f with a collection of velocities $\{u\}, \{v\}$ and pressure $\{p\}$variables, where $\{\ \}$ represent the neighbor set cell variables of face f.

$$\begin{aligned} \mathcal{F}_f &= \mathcal{F}_f\left(\{u\}, \{v\}, \{p\}\right) \\ \partial\mathcal{F}_f &= \partial_{\{u\}}\mathcal{F}_f, \partial_{\{v\}}\mathcal{F}_f, \partial_{\{p\}}\mathcal{F}_f \end{aligned} \tag{4.67}$$

Computation of continuity residual generates all the desired derivatives of $\mathcal{R}_i^p$ *w.r.t.* all the neighbor independent variables that it depends upon.

Convection, diffusion, source and pressure gradient terms are used to computed the momentum residuals using Eqs. 4.25 and 4.26. Notice that we use momentum interpolated mass fluxes, cell velocities and interpolated pressure gradient terms computed until now to computed the various terms of momentum residuals. The complete Jacobian terms can be used to solve the adjoint linear system (Eq. 4.61) for adjoint variables, which in turn, are used for obtaining sensitivities.

Unlike the complete Jacobians generated with Method 1 (Figure 4.4), the complete Jacobians generated using Method 3 (Figure 4.6) should in principle be accurate enough to compute accurate sensitivities for adjoint method. We demonstrate the validity of such a statement using an illustrated example

in the next sub-section. A rigorous mathematical analysis to determine the accuracy of method 3, in the context of the solution of the adjoint system, is beyond the scope of this dissertation.

4.4.4 Illustrative example

We consider a 2D rectangular channel illustrated in Figure 4.7 with dimensions $a = 3\,\text{m}$ and $L = 5\,\text{m}$. A fully developed velocity profile (parabolic profile) is specified at the inlet with mean velocity of $\overline{V} = 2/3\,\text{m/s}$. A zero pressure is specified at the outlet. No slip conditions are specified at the remaining two boundaries of the domain. The objective is to determine the sensitivity of total pressure drop in the channel with respect to the viscosity μ of the fluid. The pressure gradient for a fully developed flow for a Newtonian fluid in a rectangular channel is given by [116],

$$\frac{\partial p}{\partial x} = -\frac{12\mu\overline{V}}{a^2} \quad \left[\frac{\text{N}}{\text{m}^3}\right] \tag{4.68}$$

The total pressure drop in the $x-$direction for unit depth into the page is given by

$$F_{\Delta P} = -\frac{12\mu\overline{V}}{a}L \quad [\text{N}] \tag{4.69}$$

The sensitivity of the total pressure drop with viscosity is therefore given by,

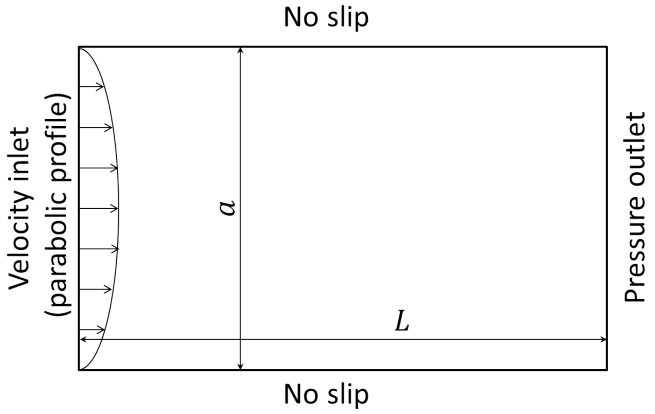


Figure 4.7: Laminar Newtonian flow in a channel. The objective is to compute the sensitivity of total pressure drop in a channel with respect to the viscosity of fluid.

Table 4.1: Comparison of sensitivities

Method ↓	Sensitivity $\frac{\partial(F_{\Delta P})}{\partial \mu}\left[\frac{\mathrm{m}^2}{\mathrm{s}}\right]$			
Mesh size (No. of cells) →	3X5=15	6X10=60	15X25=375	90X150=13500
Finite Difference method	-10.7604	-12.4105	-13.0913	-13.6026
Adjoint method (Method 1)	-10.8438	-9.3053	-7.8497	-6.8861
Adjoint method (Method 3)	-10.7950	-12.4216	-13.0931	-13.3286
Analytical	13.3333			

$$\frac{\partial\left(F_{\Delta P}\right)}{\partial \mu} = -\frac{12\overline{V}}{a} L \quad \left[\frac{\mathrm{m}^2}{\mathrm{s}}\right] \tag{4.70}$$

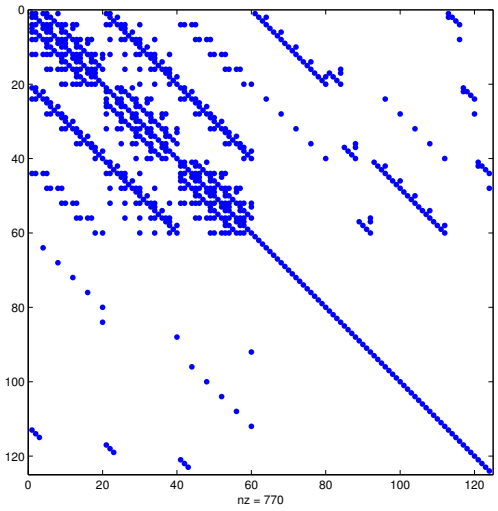
Next we consider a domain of dimensions $3\,\mathrm{m} \times 5\,\mathrm{m}$ discretized with four different Cartesian meshes of varying cell numbers as outlined in Table 4.1. We compare the analytical value of sensitivity from Eq. 4.70 with the sensitivities obtained through finite differencing and the discrete adjoint method based on Method 1 (Section 4.4.1) and Method 3 (Section 4.4.3) for all mesh sizes.

Converged flow variables (cell velocities and pressure) from the forward solution are transferred to $\mathcal{R}$apid mode. These flow variables are set as independent state variables. Viscosity is set as the independent design variable. Residuals are computed using both Method 1 and Method 3 and the sensitivity of total pressure drop with respect to viscosity determined using adjoint method with derivatives obtained from $\mathcal{R}$apid.

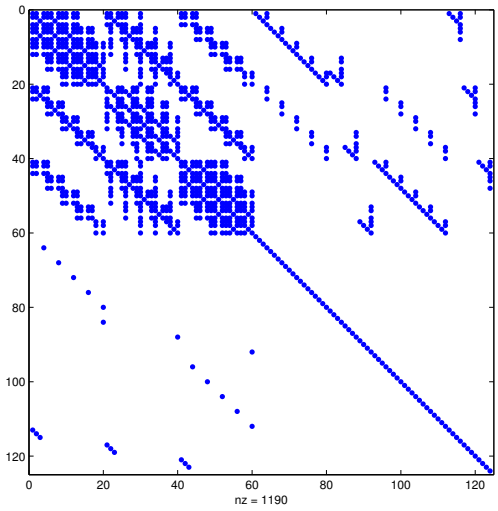
The adjoint sensitivity from Method 1 and 3 for the coarsest mesh are approximately the same but quite different from the analytical value. As the mesh gets finer, sensitivity from Method 3 (and from finite differencing) approaches the analytical value, while that of Method 2 diverges slowly.

The rationale for such a behavior with Method 1 is that the mass flux and face pressure of a particular face under consideration is not interpolated with the right functional dependence on the cell velocities and cell pressures that influence these face variables. Method 3 is able to bring in the functional dependence of the independent variables on the mass flux and face pressure used for momentum residual computation. We try to qualitatively express these behavior with the sparse distribution diagram for 3X5 mesh and 6X10 mesh in Figure 4.8.

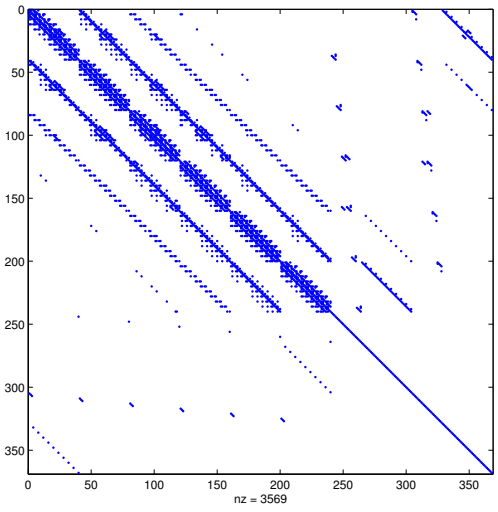
Figures 4.8 (a) and (b) depict the sparse distribution of the complete Jacobian of the momentum and continuity residuals with respect to cell velocities and pressure variables obtained using Method 1 and Method 3 with the $\mathcal{R}$apid library. The difference in the number of elements is visually obvious from the figure. For 3X5 mesh, there are a total of n =134 flow variables.



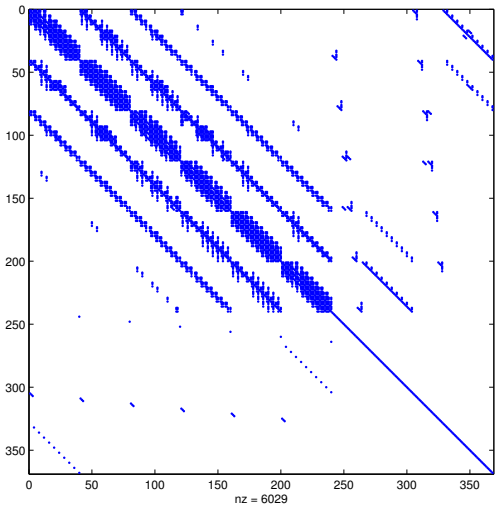
(a) Method 1 on 3×5 mesh; $nz = 770$



(b) Method 3 on 3×5 mesh; $nz = 1190$



(c) Method 1 on 6×10 mesh; $nz = 3569$



(d) Method 3 on 6×10 mesh; $nz = 6029$

Figure 4.8: Jacobian of the momentum and continuity residuals with respect to velocity and pressure variables using the $\mathcal{R}$apid library and following Method 1 and Method 3.

The number of non-zero (nz) elements can be thought of as a measure of how each method captures the dependence of the right number of cell velocities and pressure. Method 1 has lower nz compared to Method 3. We define a normalized metric M to quantify the number of dependencies given by

$$M = \frac{\Delta nz}{n} \tag{4.71}$$

M for a 3X5 mesh is ˜ 3.13. Similarly Figures 4.8 (c) and (d) shows the Jacobian matrices on 6X10 mesh. Again the difference is distribution is quite noticeable. The mesh has $n = 368$ flow variables. The value for metric M for this mesh is 6.68. The values for the metric for finer meshes with elements 15X25 and 90X150 are 9.85 and 12.82 respectively. A high value of this metric represents the decrease in capturing the dependencies of residuals on independent neighbor flow variables. The divergence of sensitivity values obtained from Method 1 can be attributed to increase in such a metric for the meshes.

We now summarize our presentation in the last three sections. A technique is devised (Method 3) to compute all the partial derivatives of the momentum residuals, continuity residuals and the desired cost functions with respect to all the flow and design variables, to be used for adjoint based sensitivity computation. The methodology achieves this objective by traversal of the sequence of steps of the SIMPLE algorithm in $\mathcal{R}$apid mode, albeit with minor modifications (as opposed to Method 2). The methodology can be used

to compute sensitivity for any flow problem solved using the finite volume method and employing the SIMPLE algorithm. This methodology is used to obtain the sensitivities of arbitrary cost functions with respect to the design variables $\boldsymbol{\beta}$ to perform topology optimization for flow applications.

4.5 Topology optimization for flow problems

4.5.1 Formulation and algorithm

Though the formalism and the algorithm of topology optimization for flow applications remains exactly the same as was presented for heat conduction applications (see Section 2.5.2, Chapter 2), we briefly highlight the differences. As mentioned in the introduction of this chapter, the governing equations are Brinkman's penalized Navier-Stokes equation (Eq. 4.4), obtained by choosing the source terms of Eqs. 4.5 and 4.6 to be a Brinkman's term.

The state variables for flow equations are the cell velocity vector and pressure. The cost function or QoI for a flow problem is generally a function of these flow state variables and the design variable β. The cost function is thus $c = c\left(\{u, v, w, p\}, \beta\right)$.

The goal of topology optimization is to partition the design space into specified volumes of solid and fluid. If ε and $1 - \varepsilon$ be the specified volume fractions of fluid and solid with which we wish to fill the design space of volume $\mathscr{V}_0$, then we seek appropriate values for the design variable β that minimizes the functional c by satisfying all the constraints of the problem. The topology

optimization problem in continuous and discrete form and shown side to side below,

$$
\begin{array}{rc}
\min : & c = c\left(\{u, v, p\}, \beta\right) \\
\text{subject to :} & \frac{\mathscr{V}_f(\beta)}{\mathscr{V}_0} \leq \varepsilon \\
& \nabla \cdot (\rho \vec{v} u) - \nabla \cdot (\mu \nabla u) = -\nabla p \cdot \vec{i} - \alpha(\beta)\, u \\
& \nabla \cdot (\rho \vec{v} v) - \nabla \cdot (\mu \nabla v) = -\nabla p \cdot \vec{j} - \alpha(\beta)\, v \\
& \nabla \cdot (\rho \vec{v}) = \mathbf{0} \\
& 0 \leq \beta \leq 1
\end{array}
\qquad
\begin{array}{rc}
\min : & c = c\left(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{p}, \boldsymbol{\beta}\right) \\
\text{subject to :} & g := \frac{\sum_i^n \beta_i}{n} - \varepsilon \leq 0 \\
& \boldsymbol{\mathcal{R}}^u(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{p}, \boldsymbol{\beta}) = \mathbf{0} \\
& \boldsymbol{\mathcal{R}}^v(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{p}, \boldsymbol{\beta}) = \mathbf{0} \\
& \boldsymbol{\mathcal{R}}^p(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{p}, \boldsymbol{\beta}) = \mathbf{0} \\
& 0 \leq \boldsymbol{\beta} \leq 1
\end{array}
\tag{4.72}
$$

Here $\mathscr{V}_f(\beta)$ is equal to the specified volume of the fluid. The coefficient of the Brinkman's term, $\alpha(\beta)$ is expressed in terms of the design variable β. Various interpolations have been used in the literature. We have tested both the RAMP (Eq. 4.2) and SIMP functional forms for interpolating α and found negligible differences. We use both the RAMP and SIMP functional form given by respectively,

$$
\alpha_u(\beta) = \alpha_f + (\alpha_s - \alpha_f)\frac{(1-\beta)}{(1+q\cdot\beta)} \tag{4.73}
$$

$$
\alpha(\beta) = (\alpha_s - \alpha_f)(1-\beta)^p + \alpha_f \tag{4.74}
$$

We should be aware of the consequences of the functional form we use. In Eqs. 4.73 or 4.74, the cell signals the presence of fluid element when corresponding $\beta = 1$, while signal solid element if $\beta = 0$. We discuss this in more

detail in Chapter 6, where we present coupled thermal and flow problems.For all the results presented in this chapter we choose $\alpha_s = 100$ and $\alpha_f = 0$.

We can also have other constraints in addition to the volume constraint. In all the figures presented in the Results section, the flow region ($\beta = 1$) is shown in red region, while the solid region ($\beta = 0$) is represented by red regions. The optimization problem 4.72 is solved using the nested formulation as shown in Figure 2.2, Chapter 2, where the discretized system of equations for the flow fields are solved separately from the design problem. The figure is self-explanatory since it had been described earlier for heat conduction applications. We have found that filtering is much more essential in flow problems than heat conduction problems, probably because of the more complex nature of the flow equations.

4.5.2 Flow topology optimization using $\mathcal{R}$apid

We invoke the model mesh introduced in Chapter 3 to illustrate the process of performing topology optimization for flow problem with the $\mathcal{R}$apid library (Figure 4.9). There are 31 cells partitioned into 15 normal cells and 16 ghost cells. Since there are 4 flow variables u, v, w and p associated with each of the 31 cells, there are a total of 124 independent flow variables. The design variables $\boldsymbol{\beta}$ is associated only with interior cells, and so we have 15 design variables.

The same procedure is followed for obtaining sensitivities for flow problems as presented the flowchart (Chapter 3 Figure 10). For a given value of

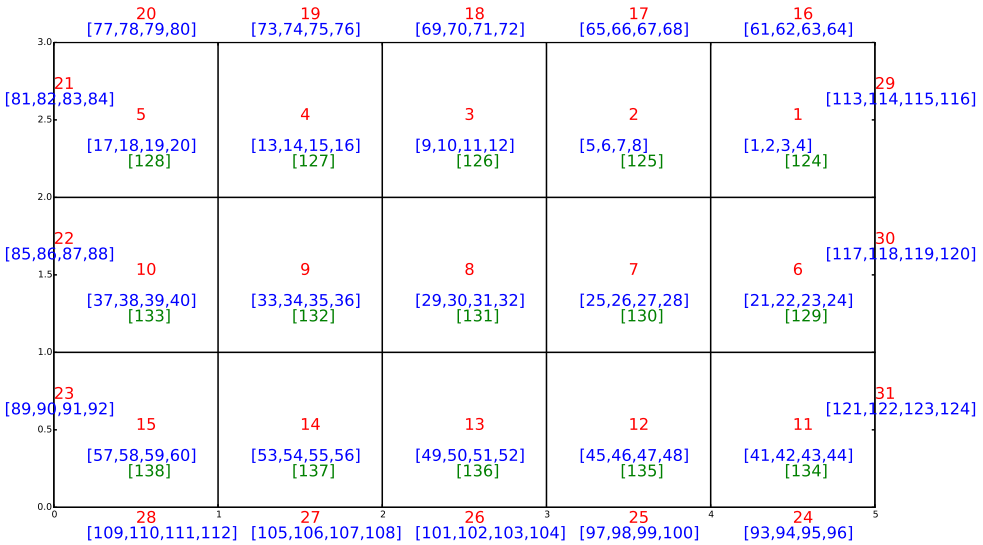


Figure 4.9: Model mesh illustrating the numbering schemes for computing sensitivities using $\mathcal{R}$apid.

design variables, the forward solution is performed to obtain the converged values for flow variables. These are then transferred to the code compiled in $\mathcal{R}$apid mode. The 124 flow variables and the 15 design variables are set as independent variables. As mentioned, unique numbers have to be used to set the independent variables. This is illustrated in Figure 4.9. For example, keys from 1-4 are assigned to velocity and pressure for cell 1, while 57-60 are used for cell 15. Numbers from 125-139 are used to identify each of the 15 design variables. Afterwards, the residuals are recomputed and the adjoint system solved to obtain sensitivities to be fed into the optimizer to determine new values of $\boldsymbol{\beta}$.

We present the customized version of Figure 4.6 for computing sensi-

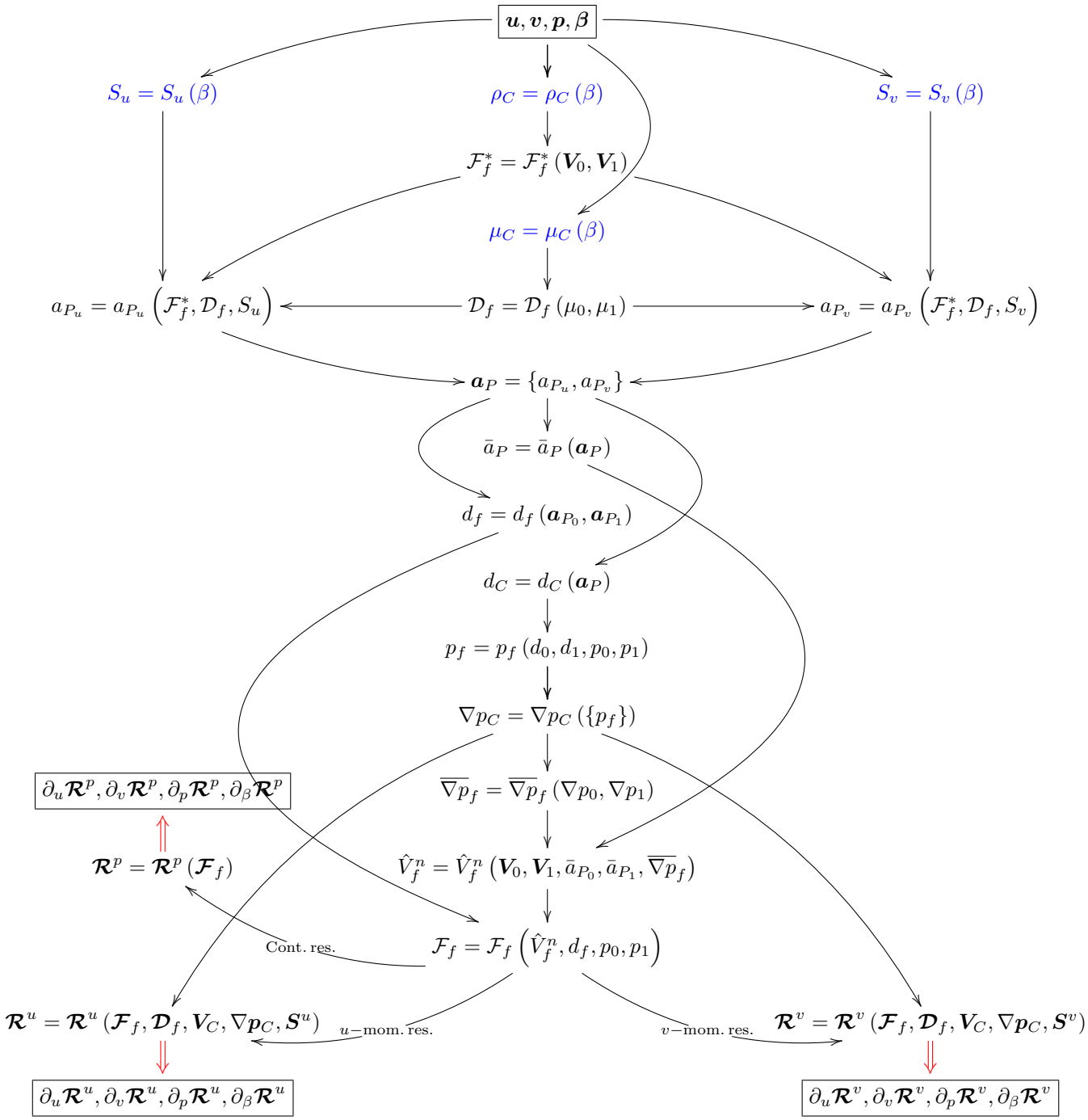


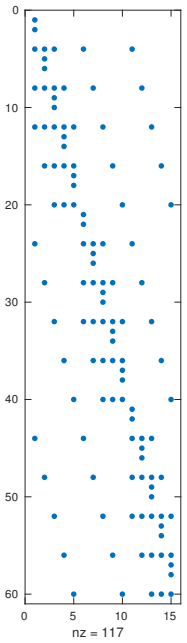
Figure 4.10: Flowchart depicting the generation of complete Jacobians with respect to flow $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{p})$ and design variables $(\boldsymbol{\beta})$ for SIMP based topology optimization with $\mathcal{R}$apid library.

tivities specifically for topology optimization of flow problems in Figure 4.10. For flow problems, in the most general way, the design variables enter the governing equations through the source term or material properties like density or viscosity (we only employ a design variable dependent source term for our work in this chapter). These are coded blue color in Figure 4.10. We note that adding design variables as independent variables also effortlessly generates the Jacobians of momentum and continuity residuals $\partial_\beta \boldsymbol{\mathcal{R}}^u$,$\partial_\beta \boldsymbol{\mathcal{R}}^v$ and $\partial_\beta \boldsymbol{\mathcal{R}}^p$ with respect to design variables β. Figure 4.11 illustrates the Jacobians of the momentum and continuity residuals obtained using Method 1 and Method 3. Notice how $\mathcal{R}$apid captures more dependencies with Method 3, thereby computing accurate sensitivities.

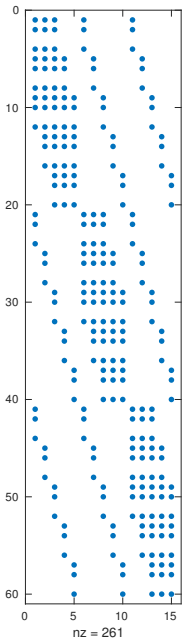
In the next chapters, we build methodologies for new physical models where the infrastructure presented in Figure 4.6 always serves as the base. An encapsulated version of Figure 4.10 is presented in Figure 4.12 to avoid repetition of the former, in the forthcoming chapters.

4.6 Results

We consider a number of test cases for both internal and external flows demonstrating the applicability of the topology optimization algorithm to flow problems. Both Cartesian and un-structured meshes are considered. A volume fraction constraint for the solid is stated as an inequality for all the cases, as in Eq. 4.72. At the end of optimization, the optimal topology for all other problems is found to have a volume fraction of the fluid equal to the specified



(a) Method 1 - $nz = 117$



(b) Method 3 - $nz = 261$

Figure 4.11: Sparse matrix representation of Jacobian of momentum and continuity residuals with respect to design variables $\boldsymbol{\beta}$.

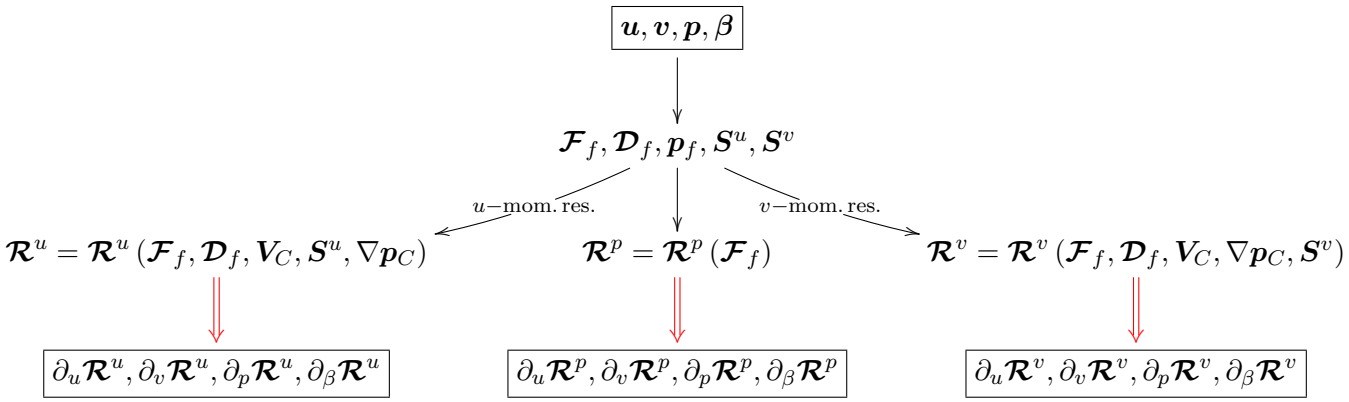


Figure 4.12: Encapsulated version of the flowchart depicting the generation of complete Jacobians with respect to flow $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{p})$ and design variables $(\boldsymbol{\beta})$ for SIMP based topology optimization with $\mathcal{R}$apid library

maximum, ε.

4.6.1 Test cases for internal flow

4.6.1.1 Channels

Almost all the papers published on topology optimization for flow applications present a so-called diffuser problem[21, 102, 33]. This is a typically a 2D square design space with a fully developed velocity inlet spanning the entire width on the left boundary and part of the exit boundary specified with another fully developed profile, respecting continuity. The top and bottom sides are set to no-slip condition.

We consider a more realistic variant of this test case in this section by specifying a velocity inlet - pressure outlet combination. The schematic diagram of the test case is the same rectangular channel used for the illustrative

example in Figure 4.7 but now with $a = L$. A fully developed velocity profile is specified at the left inlet boundary and the flow exits to zero pressure value at the right boundary. No-slip wall conditions are specified at the top and bottom boundaries. The objective is to minimize the total pressure drop across the channel in horizontal flow direction. Thus the cost function is:

$$c = \min \left(\int_{\Gamma_{in}} pdA - \int_{\Gamma_{out}} pdA \right) \tag{4.75}$$

The Reynolds number based on the inlet channel width is 10. The volume constraint of fluid ε is set to 0.5, implying that the optimizer fill the design space with at-least 50% fluid. The design domain is made of a mesh of 100 X 100 quadrilateral cells. A uniform distribution for $\boldsymbol{\beta} = 0.5$ is chosen as the initial condition for the design variable.The value for α_s and α_f are chosen to be 100 and 0. Figure 4.13 (a)-(i) shows the process of the topology optimization and the steps in the evolution $\boldsymbol{\beta}$ to a shape akin to a converging-diverging channel. The red region represents the fluid while the blue region represents solid.

Figure 4.14 shows the plots of cost function normalized with a scale factor based on the dynamic pressure, and given by,

$$c^* = \frac{1}{2}\rho \left(\overline{V}_{in} \right)^2 A_{in} \tag{4.76}$$

where ρ is the density of the fluid, $\overline{V}_{in}$ is the average inlet velocity and A_{in} is the inlet area. We observe that the normalized total pressure

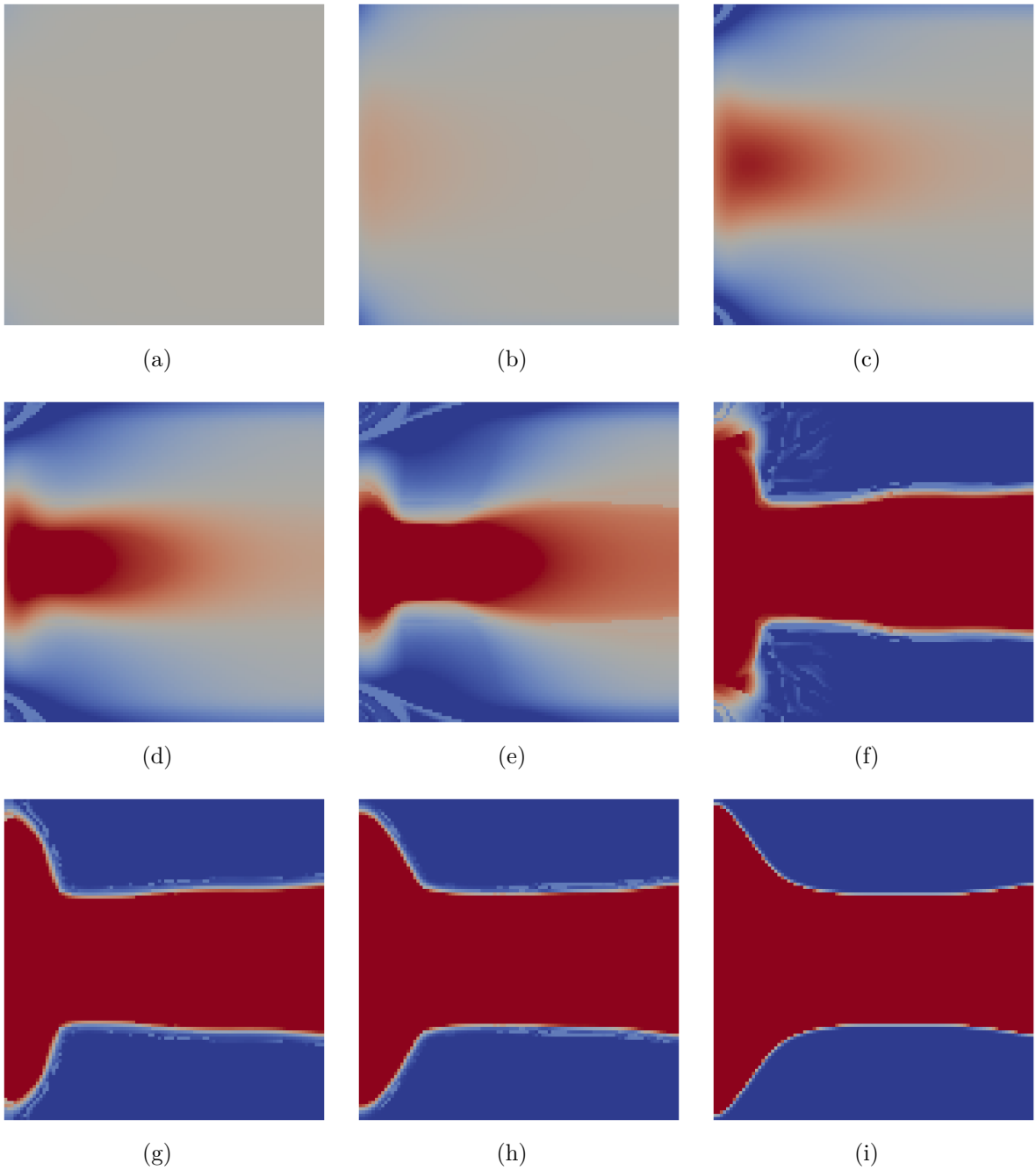


Figure 4.13: Topology optimization in test case 1. (a) Initial random distribution of $\boldsymbol{\beta}$ (b) & (c) Representative steps leading to optimized geometry. (d) One possible optimal topology. $\alpha_s = 100, \alpha_f = 0, \varepsilon = 0.5$

drop between the inlet and outlet has been reduced from an initial value of ˜2400 for the initial geometry to a final value of ˜20 for the final topology. Though the initial value of the normalized cost function depends on the value of α_s chosen, the final value is independent of it. Figure 4.14 also shows the volume fraction of the solid material and filter radius normalized with domain length for each iteration. The filter radius is gradually reduced during the optimization process. The volume fraction of the solid reaches the set value at the end, making the volume fraction constraint active (i.e., satisfied in its equality). Similar to heat conduction test cases, it is observed that an approximate geometry is evolved quickly in the first twenty iterations, and the optimizer fine-tunes the geometry to the final optimal solution at a slower pace.

Next we perform the same problem on a longer design domain with $L = 2a$. The domain is composed of 100X200 quadrilateral cells. Various Reynolds numbers ranging from $Re = 1$ to $Re = 200$ were considered. Figure 4.15(a) and (b) shows the final topologies for $Re = 1$ and $Re = 100$ respectively. The channel converges at the entrance to a near constant cross-section down-stream for the low Reynolds numbers. For high Reynolds number, the channel again converges near the entrance but starts opening up towards the outlet.

It is useful to understand the design obtained through topology optimization. The initial converging section is necessary since we are given a certain solid volume that we must use, and the best use for it would be to architect a smoothly converging section with low losses. The diverging portion

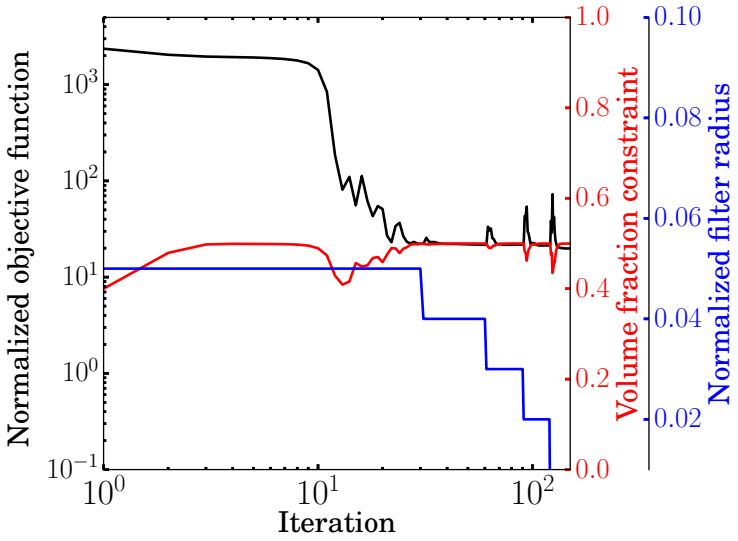
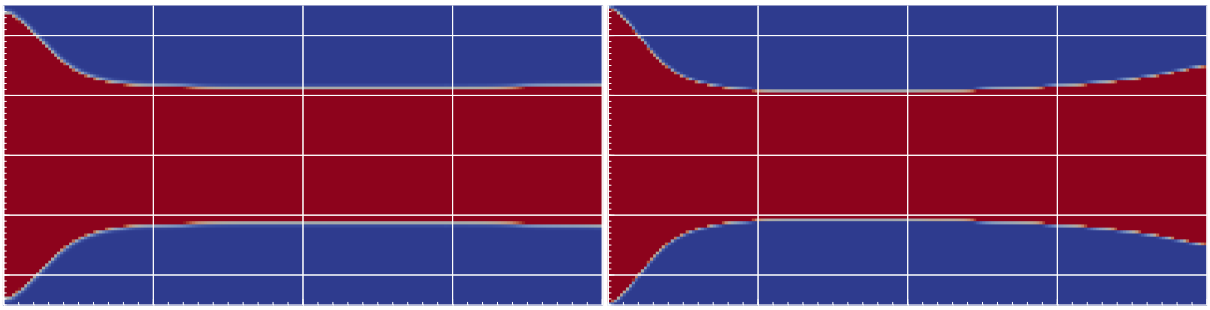


Figure 4.14: Normalized objective function and the volume fraction of fluid material versus iteration number. The plot also shows the normalized filter radius applied at various phases of the topology optimization process.

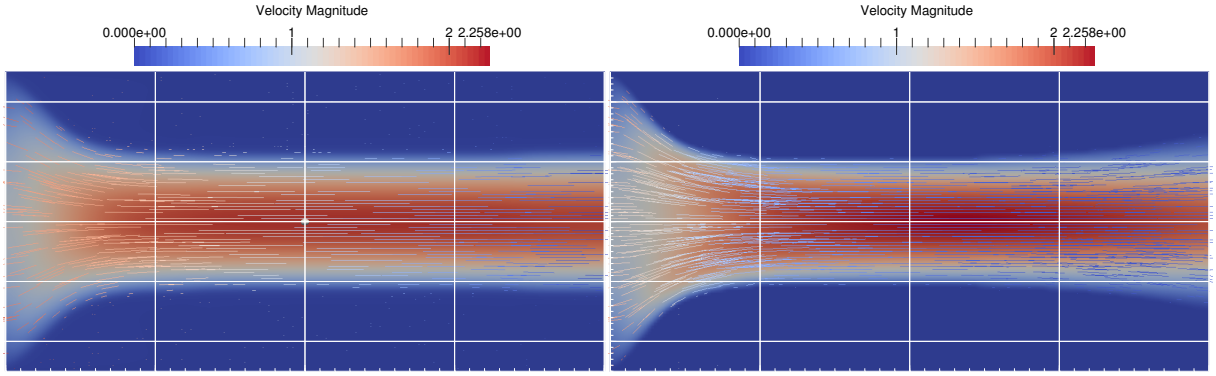
allows pressure recovery so that the pressure difference between inlet and outlet can be minimized. At high Reynolds number, the pressure recovery is large because of the quadratic dependence on velocity in Bernoulli's equation. On the other hand, in a converging-diverging geometry, the narrow throat means greater viscous losses. These are greater for low Reynolds number flows. The balance of these considerations means that we see a nearly straight channel design for low Re flows, but a diverging section for high Re flows.

4.6.1.2 Diffusers

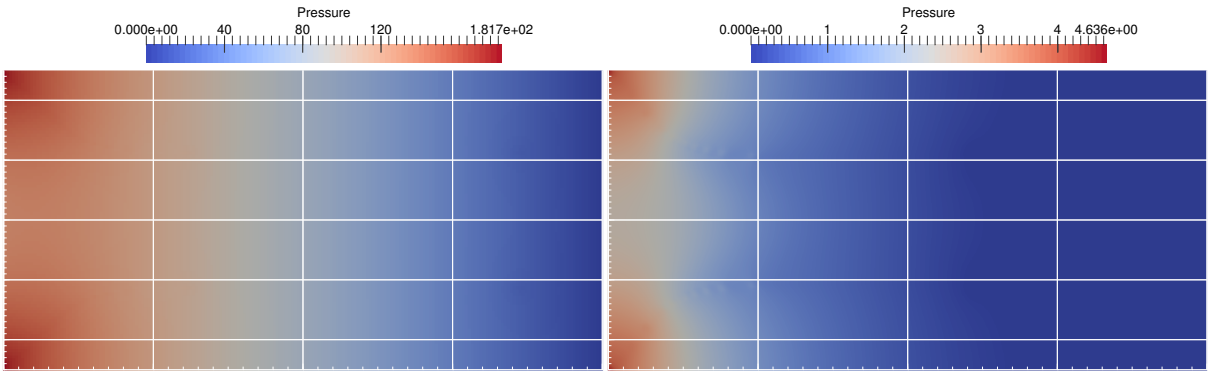
We consider next a variant of the above problem, with boundary conditions that lead to diffuser-like geometries. The schematic for design of diffuser



(a) Optimal topology for $Re = 1$ (b) Optimal topology for $Re = 100$

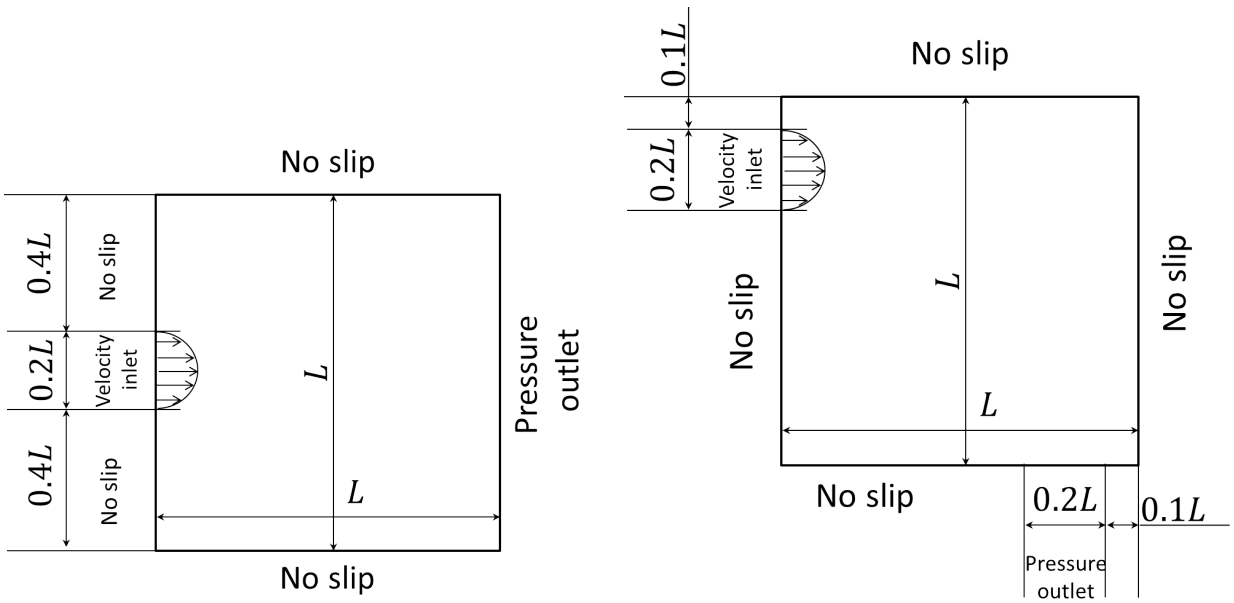


(c) Velocity magnitude distribution for $Re = 1$ (d) Velocity magnitude distribution for $Re = 100$



(e) Pressure distribution for $Re = 1$ (f) Pressure distribution $Re = 100$

Figure 4.15: Effect of Reynolds number on the optimal topologies for the channel test case with $L = 2a$.

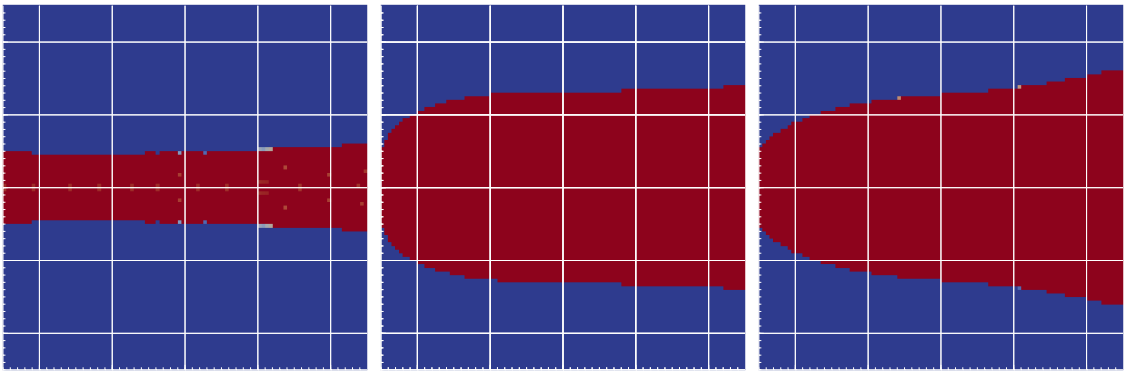


(a) (b)

Figure 4.16: Problem statement for (a) diffuser, and (b) bend design.

test case is shown in Figure 4.16(a). A portion of the inlet boundary is specified with a velocity profile. The outlet is specified as a zero- pressure boundary. The top and bottom boundaries are no-slip, as shown in the figure. We perform topology optimization for a variety of combinations of solid volume fraction and Reynolds number. The Reynolds number is based on the inlet width. Again the cost function is to minimize the total pressure drop between inlet and outlet.

Figure 4.17(a) is the final topology for $Re = 2$ and $\varepsilon = 0.2$, i.e. with the specified volume fraction of fluid $\mathscr{V}_f/\mathscr{V}_0 \leq 0.2$. The dimensions of the inlet width is 20% of the total width of the left boundary. The chosen volume

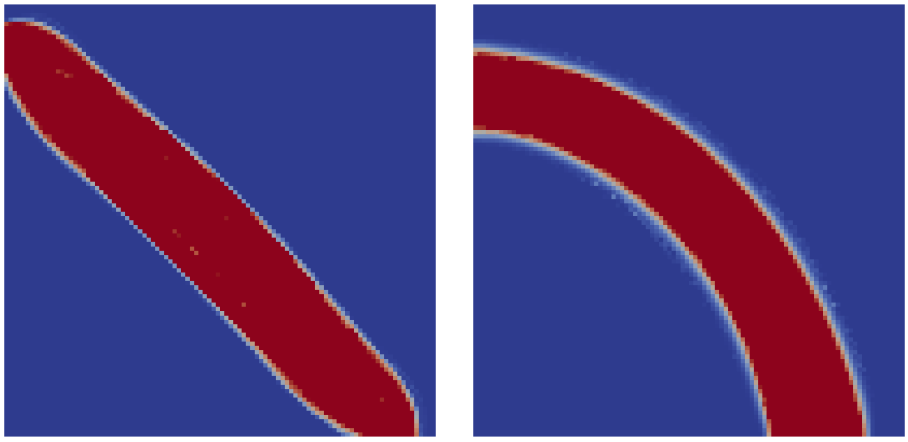


(a) $Re = 2, \varepsilon = 0.8$ (b) $Re = 2, \varepsilon = 0.5$ (c) $Re = 20, \varepsilon = 0.5$

Figure 4.17: Topology optimization of an open diffuser.

constraint should thus ideally prevent the channel from expanding. This is a test case to check whether the optimizer realizes the intended purpose. As can be seen in 4.17(a), this is indeed the case, but again the channel slightly converge and diverges, since it is not allowed to open up from the inlet.

We set the volume fraction of fluid $\varepsilon = 0.5$ and obtain the results for two Reynold's numbers $Re = 2$ and 20. Both the topologies open up as intended as shown in Figure 4.17, but at different rates. For low Re the channel mostly opens up close to the inlet as seen in Figure 4.17(b. Higher Re delays the rate of opening up the channel, basically to prevent adverse pressure gradient as shown in Figure 4.17(c). The volume constraint is satisfied to equality for all the three sub-cases.



(a) Optimal topology for $Re = 2$ (b) Optimal topology for $Re = 200$

Figure 4.18: Design of pipe bends.

4.6.1.3 Pipe bend

Design of a pipe bend is a has been a simple benchmark problem in topology optimization[21, 102, 33, 22], the schematic of which is being shown in Figure 4.18 (a). A part of the left boundary serves as the velocity inlet while a part of the right bottom region serves as the outlet. We again specify a fully developed velocity profile at the inlet and a zero pressure at the outlet. There are 100X100 elements in the design domain. We specify the fluid volume constraint ε to be 0.25. To demonstrate the effect of Reynolds number, topology optimization was performed with two Reynolds numbers $Re = 2$ and $Re = 200$. The corresponding optimal topologies are displayed in Figure 4.18(a) and (b) respectively.

There are two competing imperatives in arriving at the final design. The incoming flow, which enters horizontally, must turn in order to exit. The

slower the turn, the smaller are the pressure losses. However, the slower the turn, the longer the path length and therefor, the greater are the viscous losses. For low Re, the turning of the flow is more easily accomplished because of low inertia, but the potential for viscous loss is high. Therefore the design focuses on creating a direct straight-line path from inlet to outlet. At high Re, it is more difficult to turn the flow, and abrupt turns lead to high losses. Therefore, the design focuses on a more gentle turn, albeit at the cost of a longer path.

4.6.2 Test cases for external flow

The test cases presented in this section are for external flows past bluff bodies. The objective is to obtain shapes that minimize the pressure drop across the domain. The schematic of the test case is shown in Figure 4.19. Constant velocity profiles (plug flows) at specified angles are specified at the inlet while pressure is specified at the outlet. Symmetry conditions are specified at the top and bottom boundaries. Small obstacles are placed along the center-line axis of the design space at specified distances from the inlet boundary to encourage the formation of streamlined bodies around them to minimize the overall pressure drop from inlet to outlet. For test cases for external flow we consider Eq. 4.74 for interpolation of α. We now consider various cases.

4.6.2.1 Structured mesh

We first consider plug flow with a zero angle of attack. The design space is square-shaped with side $L = L_1 = L_2$. A structured mesh of size

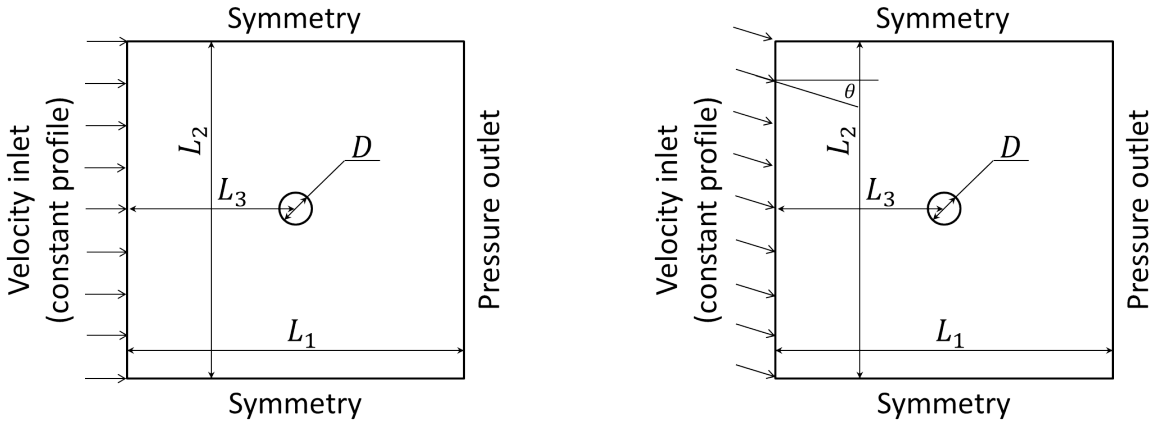
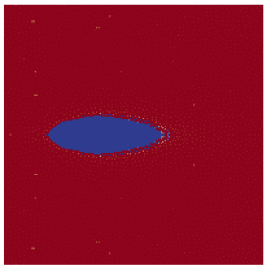


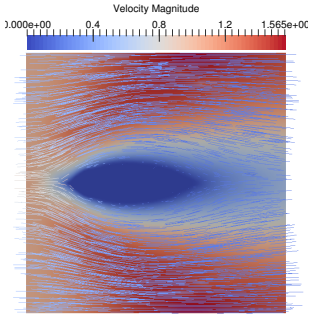
Figure 4.19: Schematic for test case for external flows

200X200 cells is used. A circular obstacle of diameter $D = 0.02L$ is placed at a distance of $L_3 = 0.25L$ from the inlet along the center line. The circular object is realized by setting the appropriate cells within the circular region to have values of $\boldsymbol{\beta} = 0$. They are held at this value and not therefore considered design variables. The fluid volume constraint is set to $\varepsilon = 0.95$ i.e. $\mathscr{V}_f/\mathscr{V}_0 \leq 0.95$. In other words, the goal is to place 5% of solid material around the obstacle so as to minimize the pressure drop between the inlet and outlet. We present the final topologies for three Reynolds number, namely $Re = 10, 100$ and 200 in Figure 4.20. The corresponding velocity plots are also plotted in the figure.

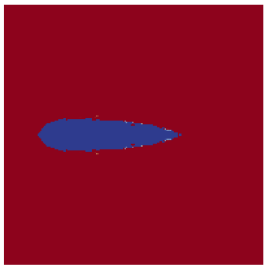
We observe that the shape of the body takes that of a back-to-front symmetric streamlined rugby ball at $Re = 10$. As Re increases, the shape remains streamlined, but the cross-sectional area exposed to the inflow becomes smaller, and the trailing edge of the blade becomes thinner. This is because a



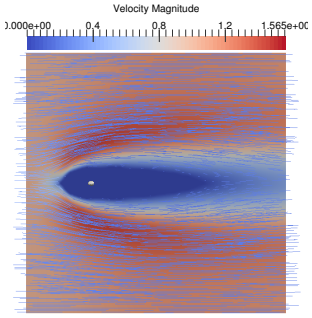
(a) Optimal topology for $Re = 10$



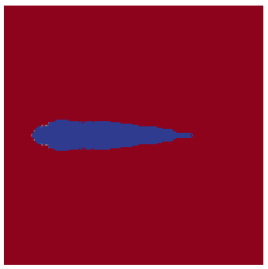
(b) Velocity mag. distribution for (a)



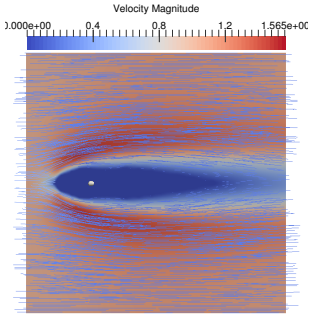
(c) Optimal topology for $Re = 100$



(d) Velocity mag. distribution for (c)



(e) Optimal topology for $Re = 200$



(f) Velocity mag. distribution for (e)

Figure 4.20: Optimal topologies for test cases described in Section 4.6.2.1.

blunt trailing edge would lead to separation and pressure loss.

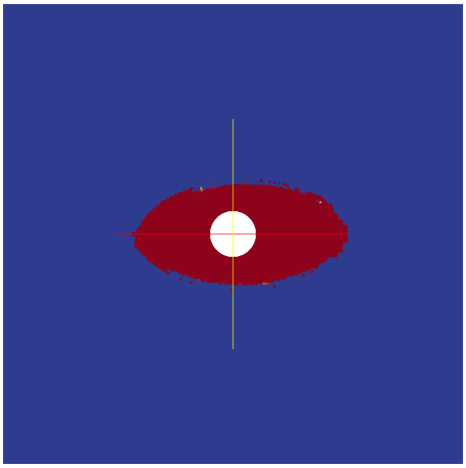
4.6.2.2 Unstructured mesh

A similar problem is presented here, but the obstacle is explicitly created in the geometry as in shown in Figure 4.21. Again the domain is square with side $L = L_1 = L_2$. The obstacle is placed at a distance of $L_3 = 0.5L$ along the center line axis from the inlet boundary. The diameter of the obstacle is $D = 0.1L$. The design domain is discretized with un-structured meshes. The mesh is not symmetrically arranged about the horizontal center-line. The fluid volume constraint is set to $\varepsilon = 0.8$.

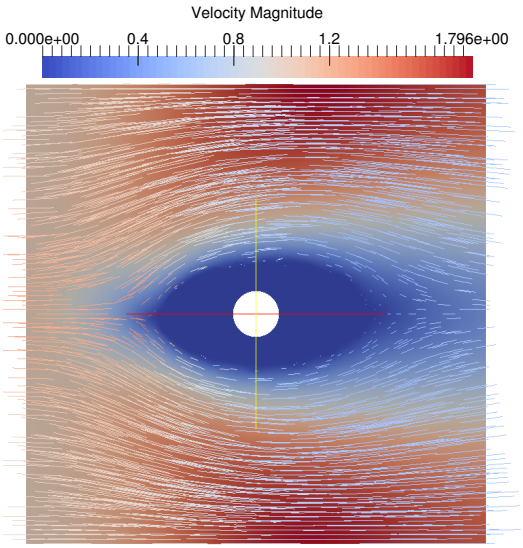
We consider two inlet flows with same magnitude of velocity, one with a zero angle of attack and another with $\theta = -45°$. We consider only a low Reynolds number, $Re = 0.1$ based on the diameter of the obstacle. Figure 4.21 shows the shapes of the two bodies that minimize the pressure drop for the given configuration. We can see the asymmetry in the shape of the object when the velocity direction is inclined. This is more conspicuous in the velocity plots shown in the figure. The symmetry condition influences the topology significantly well.

4.7 Closure

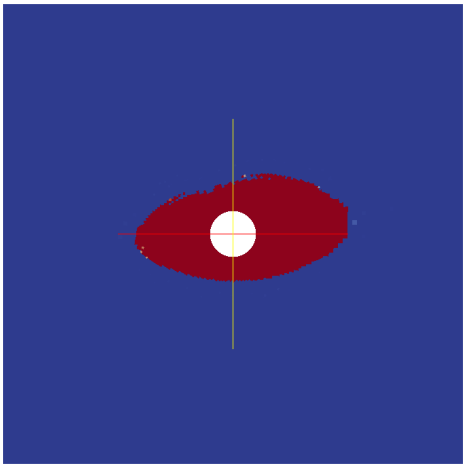
In this chapter, we developed and demonstrated topology optimization for incompressible Newtonian flow problems in the laminar regime. The underlying numerical technique is the finite volume method employing a sequential



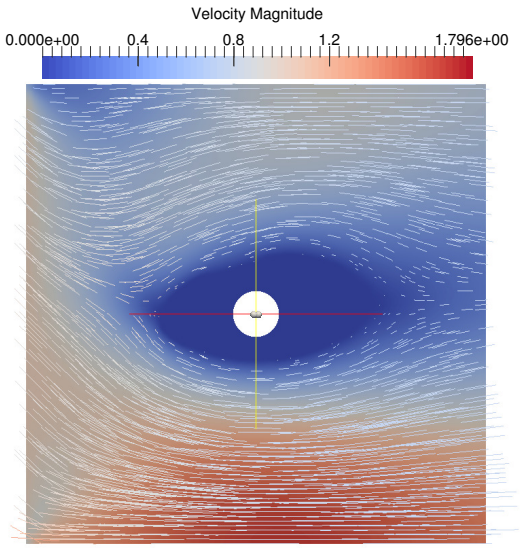
(a) Angle of attack $\theta = 0°$



(b)



(c) Angle of attack $\theta = -45°$



(d)

Figure 4.21: Topology optimization for flow past an obstruction using unstructured meshes.

co-located pressure-velocity scheme for unstructured meshes. We also developed a novel technique to compute discrete adjoint sensitivities within such a finite volume framework using the $\mathcal{R}$apid AD library. The central idea was to capture the full Jacobian accurately in the $\mathcal{R}$apid mode accounting for the right dependencies of face mass flux and face pressure on the relevant cell velocities and cell pressures.

The methodology described in this chapter is essentially problem-agnostic in nature. Similar strategies can be extended systematically to compute sensitivities in flow problems coupled with new physics. For example, one can obtain sensitivities for Reynolds-Averaged Navier-Stokes (RANS) based turbulent flow models or coupled flow and thermal problems. We consider topology optimization for turbulent flows in Chapter 5. Coupled flow and thermal problems are described in Chapter 6. In both chapters, we build methodologies for new physical models using the infrastructure presented in Figure 4.6.

Chapter 5

Turbulent flow applications

In this chapter, we extend topology optimization for flow problems to the turbulent regime. We choose the Spalart-Allmaras RANS turbulence model, though other models may be considered as well. Based on this model, the topology optimization methodology for steady state turbulent flow problems is developed in the frameworks of unstructured cell-centered finite volume schemes and automatic differentiation for computations of discrete adjoint sensitivities. The use of the $\mathcal{R}$apid library makes the arduous task of differentiating the various terms of the Spalart-Allmaras governing equations virtually effortless. Finally, as proof of concept, we demonstrate the methodology with a channel flow test case.

5.1 Introduction

The definition of turbulence by Peter Bradshaw in his book is very relevant here [117]. '*Turbulence is a three-dimensional time-dependent motion in which vortex stretching causes velocity fluctuations to spread to all wavelengths between a minimum determined by viscous forces and a maximum determined by the boundary conditions of the flow. It is the usual state of fluid motion*

except at low Reynolds numbers.' Thus even though there are quite a few industrial applications of laminar flow such as in microfluidics, food processing, polymer and glass processing etc., the majority of flows in industry (and in nature) are turbulent. Therefore it is necessary to consider turbulence when designing airplane wings, turbine blades, heat exchangers, chemical reactors, buildings/bridges etc. Therefore topology optimization must be extended to turbulent flow before it can become an accepted design tool in industry.

There exist various computational methods to solve turbulent flow. Direct Numerical Simulation (DNS) employs the solution of the Navier-Stokes equations by resolving all scales of turbulence [118]. Though most accurate, this method can be only used to solve relatively low Reynolds number problems even with the most powerful computers available today. DNS has been used to understand the physics of turbulence and to validate various turbulence models. To make simulation of turbulent flows computationally tractable, many turbulent models have been developed over the last few decades with varying degrees of success. Reynolds Averaged Navier Stokes (RANS) based models and Large Eddy Simulation (LES) are among the most important [119, 120], though the latter are still computationally intensive. The most widely used models in the commercial CFD codes are RANS based models.

Of the large variety of RANS models[119, 121] available, the linear eddy viscosity models are the most prevalent. These are again sub-classified to algebraic models, one-equation models, two-equation models etc., and offer increasing applicability but with increasing computational complexity. The

Spalart-Allmaras (SA) model is a popular one equation model which has been used with a fair amount of success [122]. Here one scalar transport equation for a new transport variable related to the eddy viscosity is solved in conjunction with the RANS equations. Among the most widely used two-equations models are the are $k-\varepsilon$ model and $k-\omega$ models. Here two extra scalar transport equations, for the turbulent kinetic energy and either the eddy dissipation or the specific dissipation, are solved along with equations [119]. To demonstrate topology optimization for turbulent flow we choose the SA model here, though the same methodology can be applied to other RANS based models as well.

Before discussing the SA model in detail, we present the literature on topology optimization for turbulent flows.

As was mentioned before, a significant amount of work has been done on shape optimization for flows in turbulent regime. However the work on topology optimization for turbulent flow has just started to appear recently and is very much in its infancy[23]. The few published works on the topic use a continuous adjoint method. The authors used the Spalart-Allmaras model using a continuous adjoint method in a finite volume framework [123, 124, 125]. They solved coupled flow and thermal problems as well. To the best of our knowledge, we know of only one very recent paper in the literature on topology optimization for turbulent flow using SIMP and a discrete adjoint method [34]. Here, the SA model is used within a finite element framework.

We briefly outline the contents of chapter. To begin, the governing equations underlying turbulent flows using the Spalart-Allmaras model are

described. The numerical procedure to solve these equations based on the SIMPLE algorithm is described next in Section 5.3. The description of topology optimization for turbulent flow starts in Section 5.4 and includes a detailed description of the relevant sensitivity computation using the $\mathcal{R}$apid library. Finally a test case is presented as a proof of concept.

5.2 Governing equations and boundary conditions

5.2.1 Reynolds Averaged Navier-Stokes (RANS) equation

For many practical situations of engineering interest, it is enough to know the mean velocity of the turbulent flow. To obtain an equation for mean velocity, the time-dependent flow velocity $\boldsymbol{v} = v_i = [u, v, w]$ is decomposed into the mean velocity term $\bar{v}_i$ and the fluctuating velocity term $v_i^{'}$ and then substituted into the Navier-Stokes equations. The Navier-Stokes equations are then time-averaged to obtain the RANS equations given by,

$$\begin{aligned} \frac{\partial\left(\rho\bar{v}_i\right)}{\partial t} + \frac{\partial\left(\rho\bar{v}_j\bar{v}_i\right)}{\partial x_j} &= -\frac{\partial\bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\mu\frac{\partial\bar{v}_i}{\partial x_j} - \rho\overline{v_i^{'}v_j^{'}}\right) \\ \frac{\partial\left(\rho\bar{v}_i\right)}{\partial x_i} &= 0 \end{aligned} \tag{5.1}$$

where $\bar{p}$ is the mean pressure field (details in [119]). The unknowns being solved in the RANS equations are the mean velocity $\bar{v}_i$ and pressure $\bar{p}$. The term $\overline{v_i^{'}v_j^{'}}$ is the correlation of turbulent fluctuations termed as Reynolds stress $r_{ij} = -\overline{v_i^{'}v_j^{'}}$, and is not known in terms of $\bar{v}_i$. Therefore to close the

RANS equations, one needs to model the Reynolds stress r_{ij} term. A number of closure models have been proposed in the literature[119, 121].

Eddy viscosity models are one such class of models where the effect of turbulence is modelled with an effective viscosity term ν_T. In this approach, one writes

$$r_{ij} = \nu_T \left(\frac{\partial \bar{v}_i}{\partial x_j} + \frac{\partial \bar{v}_j}{\partial x_i} \right) - \frac{2k}{3}\delta_{ij} = -\nu_T S_{ij} + \frac{2k}{3}\delta_{ij} \tag{5.2}$$

The term ν_T is termed as eddy viscosity and k is the mean turbulent kinetic energy given by

$$k = \frac{1}{2}\overline{\left(v_i' v_i'\right)} \tag{5.3}$$

If ν_T and k are specified, then the RANS equation (Eq. 5.1) can be closed.

5.2.2 Spalart Allmaras (SA) turbulence model

In the SA model, the last term of Eq. 5.2 is generally ignored since k is not readily available[126]. Therefore the Reynolds stress is approximated as,

$$r_{ij} = \nu_T \left(\frac{\partial \bar{v}_i}{\partial x_j} + \frac{\partial \bar{v}_j}{\partial x_i} \right) = \rho \mu_T \left(\frac{\partial \bar{v}_i}{\partial x_j} + \frac{\partial \bar{v}_j}{\partial x_i} \right) \tag{5.4}$$

Substituting in Eq. 5.1, and assuming steady state yields the following scalar transport equations with source terms for $\bar{u}, \bar{v}$ ($\bar{w}$ implied)and $\bar{p}$ in

conservative form as,

$$\nabla \cdot (\rho \bar{\boldsymbol{v}} \bar{u}) = -\nabla \bar{p} \cdot \vec{i} + \nabla \cdot ((\mu + \mu_T) \nabla \bar{u}) + S_u \tag{5.5}$$

$$\nabla \cdot (\rho \bar{\boldsymbol{v}} \bar{v}) = -\nabla \bar{p} \cdot \vec{j} + \nabla \cdot ((\mu + \mu_T) \nabla \bar{v}) + S_v \tag{5.6}$$

$$\nabla \cdot (\rho \bar{\boldsymbol{v}}) = 0$$

To determine the eddy viscosity $\nu_T = \mu_T/\rho$, a new transport equation is proposed for a variable $\tilde{\nu}$, called the SA variable. The two variables are related by [122],

$$\nu_T = \tilde{\nu} f_{v1} \tag{5.7}$$

where

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}; \quad \chi \coloneqq \frac{\tilde{\nu}}{\nu} \tag{5.8}$$

Here $\nu = \mu/\rho$ is the molecular kinematic viscosity.

The transport equation for $\tilde{\nu}$ may be written as,

$$\underbrace{\nabla \cdot (\rho \bar{\boldsymbol{v}} \tilde{\nu})}_{\text{convection}} = \underbrace{\nabla \cdot \left(\frac{\rho (\nu + \tilde{\nu})}{\sigma} \nabla \tilde{\nu} \right) + \frac{\rho c_{b2}}{\sigma} (\nabla \tilde{\nu})^2}_{\text{diffusion}} + \underbrace{c_{b1} (1 - f_{t2}) \tilde{S} \tilde{\nu}}_{\text{production}} - \underbrace{\left(c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2} \right) \left[\frac{\tilde{\nu}}{d} \right]^2}_{\text{wall destruction}} \tag{5.9}$$

Here

$$(\nabla\tilde{\nu})^2 = \frac{\partial\tilde{\nu}}{\partial x_j}\frac{\partial\tilde{\nu}}{\partial x_j} \tag{5.10}$$

Eq. 5.9, cast in the form of a scalar transport equation for finite volume discretization and is given by,

$$\nabla\cdot(\rho\bar{\boldsymbol{v}}\tilde{\nu}) = \nabla\cdot\left(\frac{\rho\left(\nu+\tilde{\nu}\right)}{\sigma}\nabla\tilde{\nu}\right)+\underbrace{\frac{\rho c_{b2}}{\sigma}\left(\nabla\tilde{\nu}\right)^2 + c_{b1}\left(1-f_{t2}\right)\tilde{S}\tilde{\nu} - \left(c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right)\left[\frac{\tilde{\nu}}{d}\right]^2}_{\text{Source term}} \tag{5.11}$$

There are many closure functions in Eq. 5.9, which are systematically given by,

$$\tilde{S} \equiv S + \frac{\tilde{\nu}}{\kappa^2 d^2}f_{v2}, \quad f_{v2} = 1 - \frac{\chi}{1+\chi f_{v1}} \tag{5.12}$$

$$f_w = g\left[\frac{1+C_{w3}^6}{g^6+C_{w3}^6}\right]^{1/6}, \quad g = r + C_{w2}\left(r^6 - r\right), \quad r \equiv \min\left(\frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2}, 10\right) \tag{5.13}$$

$$f_{t2} = C_{t3}\exp\left(-C_{t4}\chi^2\right) \tag{5.14}$$

$$S = \sqrt{2\Omega_{ij}\Omega_{ij}} \tag{5.15}$$

The rotation tensor Ω_{ij} is given by

$$\Omega_{ij} = \frac{1}{2}\left(\frac{\partial \bar{v}_i}{\partial x_j} - \frac{\partial \bar{v}_j}{\partial x_i}\right) \tag{5.16}$$

The constants in the above closure functions are given by,

$$\begin{aligned}
\sigma &= 2/3 \\
C_{b1} &= 0.1355 \\
C_{b2} &= 0.622 \\
\kappa &= 0.41 \\
C_{w1} &= \frac{C_{b1}}{\kappa^2} + \frac{(1 + C_{b2})}{\sigma} \\
C_{w2} &= 0.3 \\
C_{w3} &= 2 \\
C_{v1} &= 7.1 \\
C_{t1} &= 1 \\
C_{t2} &= 2 \\
C_{t3} &= 1.1 \\
C_{t4} &= 2
\end{aligned} \tag{5.17}$$

5.2.3 PDE-based wall distance model

Computation of the wall destruction term in Eq. 5.9 requires specifying the distance d of a field point to the nearest wall. It is easiest to compute the distance to the nearest wall by solving a partial differential equation (PDE),

though other methods exist. Various PDEs have been postulated to serve the purpose [127]. For example, the Eikonol function approach was proposed for which the solution to the PDE is the wall distance [34]. Spalding proposed a simple PDE for computing a field ϕ from which wall distance d can be post-computed [128]. ϕ is henceforth referred as the wall distance function. We employ Spalding's approach in this dissertation.

The equation to be solved is Poisson's equation,

$$\nabla^2\phi + 1 = 0 \tag{5.18}$$

The boundary conditions are

$$\phi = 0 \,\text{at walls} \tag{5.19}$$

$$\frac{\partial\phi}{\partial n} = 0 \,\text{every other boundary} \tag{5.20}$$

After solving the field ϕ for the given domain, the distance of any field point to the nearest wall is given by

$$d = \sqrt{\nabla\phi\cdot\nabla\phi + 2\left(|\phi|\right)} - \sqrt{\nabla\phi\cdot\nabla\phi} \tag{5.21}$$

5.2.4 Boundary conditions

All the boundary conditions for velocities and pressure variables specified at inlet, outlet and wall boundaries for the Navier-Stokes equations apply

for the RANS equations as well, but operate on the corresponding mean velocity and pressure variables.

At the inlet boundary, the value of the SA variable $\tilde{\nu}$ is usually specified as a multiple of molecular kinematic viscosity, generally between $3\nu \leq \tilde{\nu} \leq 5\nu$ [126]. The values of $\tilde{\nu}$ are generally extrapolated from the interior at the outlet boundary. The symmetry condition is achieved by applying the condition $\frac{\partial \tilde{\nu}}{\partial n} = 0$, where n is the normal to the symmetry boundary.

The wall boundary condition is

$$\tilde{\nu} = 0 \tag{5.22}$$

We will be revisiting the wall boundary condition (Eq. 5.22), as it is of specific interest to us from the perspective of topology optimization.

It is also noted that the boundary conditions for $\tilde{\nu}$ are generally specified in terms of the turbulent viscosity ratio I, defined as the ratio of eddy viscosity and molecular viscosity and is given by,

$$I \coloneqq \frac{\mu_T}{\mu} = \frac{\nu_T}{\nu} = \frac{\tilde{\nu} f_{v1}}{\nu} = \chi f_{v1} \tag{5.23}$$

$$\implies f_{v1} = \frac{I}{\chi} \tag{5.24}$$

Also,

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3} \tag{5.25}$$

Therefore,

$$\frac{I}{\chi} = \frac{\chi^3}{\chi^3 + c_{v1}^3} \tag{5.26}$$

Therefore given a value for I one can find χ by solving the non linear equation (Eq. 5.26), from which $\tilde{\nu}$ can be determined for a given ν.

5.2.5 Spalding's wall function

For turbulent flow, the near-wall mesh must be extremely refined to resolve the thin turbulent boundary layer. To circumvent this, it is a general practice to correct the eddy viscosity near the wall using wall functions based on the law of the wall. Among the many wall functions available in the literature, we use Spalding's formula [129] given by:

$$y^+ = u^+ + \exp\left(-\kappa B\right)\left[\exp\left(\kappa u^+\right) - 1 - \kappa u^+ - \frac{1}{2}\left(\kappa u^+\right)^2 - \frac{1}{6}\left(\kappa u^+\right)^3\right] \tag{5.27}$$

where y^+ and u^+ are scaled wall distance and velocity respectively, and are given by,

$$y^+ := \frac{y u_\tau}{\nu} \text{ and } u^+ := u/u_\tau \tag{5.28}$$

Here y is the wall distance of the cell adjacent to the wall boundary. u_τ is the friction velocity based on the shear stress on the wall which is an unknown. Our goal is to determine u_τ from Eq. 5.27 for the given y. The non-linear Eq. 5.27 is solved using Newton Raphson iteration. Once u_τ is solved, the eddy viscosity of the cell near the wall is found using,

$$\nu_T = \frac{u_\tau^2}{|\nabla \boldsymbol{v}_{wall} \cdot \vec{n}|} - \nu \tag{5.29}$$

where $\nabla \boldsymbol{v}_{wall}$ is the velocity gradient of the wall adjacent cell under consideration and $\vec{n}$ is the normal to the wall.

5.3 Numerical method - SIMPLE for RANS-SA model

There are three groups of scalar transport equations to be solved for RANS based SA turbulent model -

1. Momentum and continuity equation,
2. Wall distance model and
3. Spalart-Allmaras equation for $\tilde{\nu}$.

From an implementation perspective, the numerical solution of RANS equation is identical to the solution of the Navier-Stokes equation that were discussed in Chapter 4 using the SIMPLE algorithm. We solve for the mean velocity and pressure variables in the RANS equations as opposed to solving

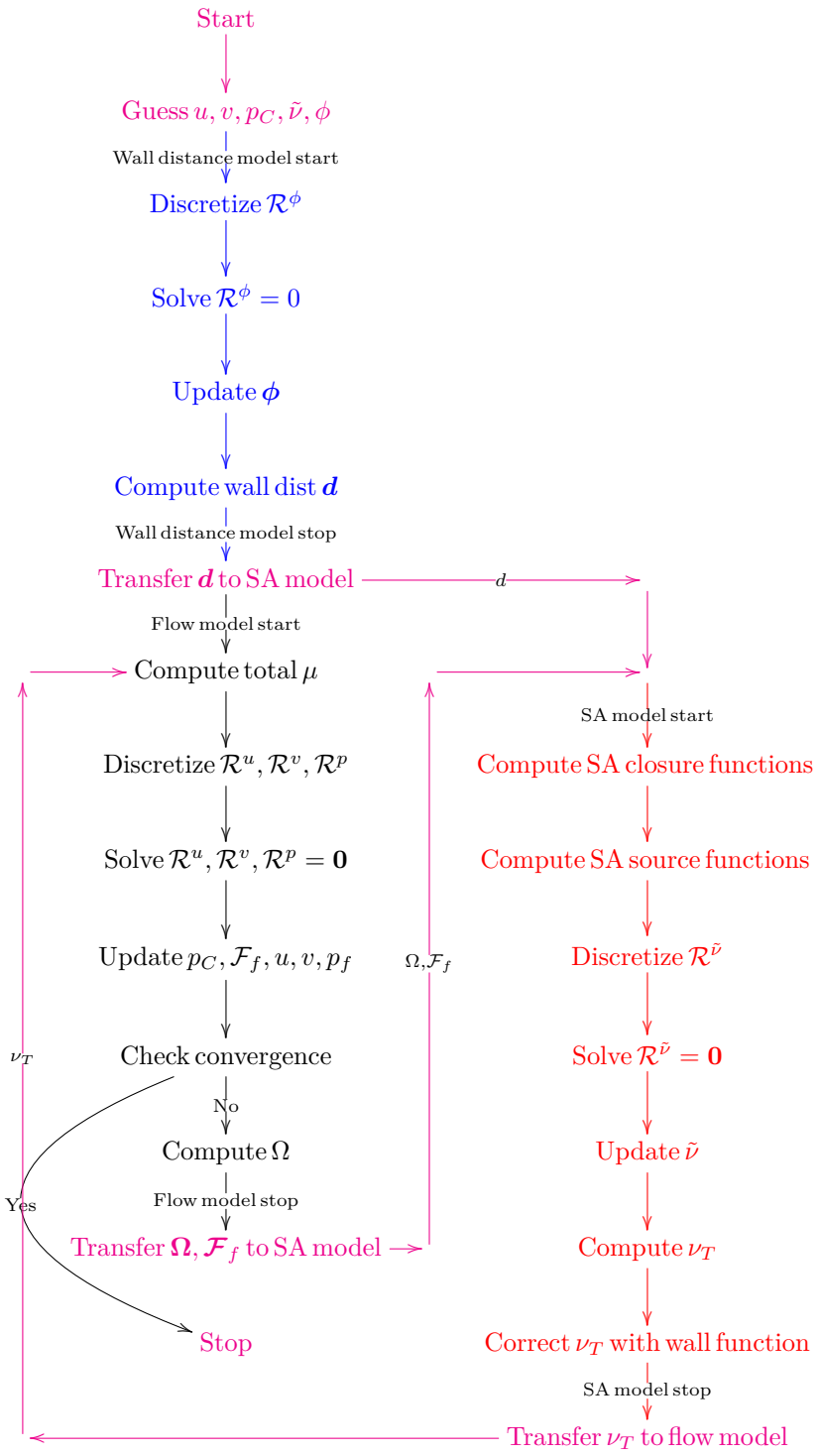


Figure 5.1: Flowchart for a Spalart-Allmaras turbulence model employing SIMPLE algorithm. The governing equations include the wall distance model (blue), the RANS equations (black) and the Spalart-Allmaras model (red).

the intrinsic velocities in laminar flow. The only additional operation that needs to be done is to compute an effective viscosity termed μ_{total},

$$\mu_{total} = \mu + \mu_T \tag{5.30}$$

which is the sum of molecular viscosity and the eddy viscosity μ_T. The rest of the procedure is same until we finish the post-correction part (Section 4.3.1, Chapter 4) of the SIMPLE algorithm.

The wall distance model (Eq. 5.18) is a pure diffusion equation with a constant source term and requires no further explanation. It is solved using methods similar to those for the heat conduction equation described in Chapter 2.

Solution of the transport equation for the SA variable (Eq. 5.11) is also straightforward By the time one solves the SA equation, the underlying flow field variables are known from the solution of the momentum and continuity equations. The mass flux term $\boldsymbol{\mathcal{F}}_f$ thus obtained from the current flow solution is used to discretize the convection part of the SA equation. The source terms are algebraically complicated and need care while discretization.

Thus the solution to the RANS-SA using a SIMPLE based finite volume scheme is a sequential iteration between the three models in the order depicted in Figure 5.1. For enhanced readability, the three models are color coded - blue for wall distance model, black for flow model and red for SA $\tilde{\nu}$ model. In MEMOSA the iteration between the various physical models implemented in

C++ is controlled at a higher level using Python scripting (shown in magenta).

As depicted in Figure 5.1, the wall distance is solved only once to obtain the field ϕ, from which the wall distance field $\boldsymbol{d}$ is is computed using Eq. 5.21.. For a given geometry, the field $\boldsymbol{d}$ is transferred to SA model once for all.

We now consider the flow solver part of the algorithm. Based on a guessed value of $\tilde{\nu}$, total viscosity μ_{total} is computed using Eq. 5.30. The momentum and continuity equations are then solved. Next, from the current solution of velocity field, the vorticity field given by Eq. 5.16 is computed. The current mass flux field and vorticity field are transferred to the SA model.

All the closure functions from Eqs. 5.12 to Eqs. 5.15 are computed in the proper sequence. The source term as given in Eq. 5.11 is discretized with cell center values. The new values of the SA variable $\tilde{\nu}$ are then computed. Based on Eq. 5.7, eddy viscosity ν_T is computed. The current value of ν_T near the wall is corrected using the the procedure outlined in Section 5.2.5.

The latest values of eddy viscosity are transferred back to the flow model and the new μ_{total} computed. The algorithm iterates between flow and SA solvers until convergence of mean quantities in RANS is obtained.

We end with a note reminding the reader that due to the sequential nature of the algorithm described above, only partial Jacobians are computed during the forward solution, both for the flow equations and the SA equations. However, complete Jacobians are needed for sensitivity computations for topology optimization.

5.4 Topology optimization using RANS-SA model

5.4.1 Formulation

The governing equations for RANS based SA turbulence model are Eqs. 5.5, 5.11 and 5.18.Topology optimization for fluid flow is accomplished by partitioning the design domain into fluid and solid regions. The boundary conditions for all the three governing equations that must be satisfied on the solid-fluid interface Γ are

$$u_\Gamma = 0; v_\Gamma = 0; \phi_\Gamma = 0; \tilde{\nu}_\Gamma = 0 \tag{5.31}$$

We modify the governing equations to achieve this condition.

We first modify the wall distance Eq. 5.18 by adding a term $\alpha_\phi \phi$ as follows,

$$\nabla^2 \phi + 1 + \alpha_\phi \phi = 0 \tag{5.32}$$

If the $\alpha_\phi \gg 0$, then the laplacian term becomes negligible reducing the wall distance PDE to $\alpha_\phi \phi = 0$, thereby implying $\phi = 0$ and in turn signaling that the field point under consideration is solid. On the other hand if $\alpha_\phi = 0$, we recover the original wall distance PDE signaling the field point to be present in fluid region. Thus we interpolate α_ϕ between zero and a chosen high value using a SIMP function designed to achieve the proper limiting values.

Based on the same argument, the equation for the SA variable $\tilde{\nu}$ is also augmented with the term $\alpha_{\tilde{\nu}} \tilde{\nu}$, where the coefficient $\alpha_{\tilde{\nu}}$ is interpolated between

two limits - zerop and a high value. The algorithm is intended to drive the coefficient value to one of the limits, consistent with whether the cell under consideration is fluid or solid. The RANS equations is augmented with the Brinkman's penalized term as before. Thus the modified governing equations using the SIMP methodology takes the following final form.

$$
\begin{aligned}
\nabla \cdot (\rho \bar{\boldsymbol{v}} \bar{u}) &= \nabla \cdot ((\mu + \mu_T) \nabla \bar{u}) - \nabla \bar{p} \cdot \vec{i} - \alpha_u (\beta) \bar{u} \\
\nabla \cdot (\rho \bar{\boldsymbol{v}} \bar{v}) &= \nabla \cdot ((\mu + \mu_T) \nabla \bar{v}) - \nabla \bar{p} \cdot \vec{j} - \alpha_v (\beta) \bar{v} \\
\nabla \cdot (\rho \bar{\boldsymbol{v}}) &= 0 \\
0 &= \nabla \cdot (\nabla \phi) + 1 - \alpha_\phi (\beta) \phi \\
\nabla \cdot (\rho \bar{\boldsymbol{v}} \tilde{\nu}) &= \nabla \cdot \left(\frac{\rho (\nu + \tilde{\nu})}{\sigma} \nabla \tilde{\nu} \right) + \frac{\rho c_{b2}}{\sigma} (\nabla \tilde{\nu})^2 + \ldots \\
&\qquad c_{b1} (1 - f_{t2}) \tilde{S} \tilde{\nu} - \left(c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2} \right) \left[\frac{\tilde{\nu}}{d} \right]^2 - \alpha_{\tilde{\nu}} (\beta) \tilde{\nu}
\end{aligned}
\tag{5.33}
$$

We choose the same functional forms for all the SIMP functions $\alpha_u (\beta) \bar{u}, \alpha_v (\beta) \bar{v}$, $\alpha_\phi (\beta) \phi$ and $\alpha_{\tilde{\nu}} (\beta) \tilde{\nu}$ as in previous chapters given by,

$$
\alpha (\beta) = (\alpha_s - \alpha_f) (1 - \beta)^p + \alpha_f \tag{5.34}
$$

where values for α_s and α_f for each governing equation are different and must be chosen judiciously.

The state variables for turbulent flow equations are thus the mean velocity vector, the mean pressure, the SA variable, and the wall distance function

ϕ. The cost function can be therefore a function of these state variables and the design variable β, and is thus denoted by $c = c\left(\{\bar{u}, \bar{v}, \bar{w}, \bar{p}, \phi, \tilde{\nu}\}, \beta\right)$. The optimization problem may thus be stated as:

Following the same pattern (Chapter 4, Eq. XX) and with the volume constraint for the fluid $\frac{\mathscr{V}_f(\beta)}{\mathscr{V}_0} \leq \varepsilon$, the topology optimization problem in discrete form is,

$$\begin{aligned}
\min: \quad & c = c\left(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}, \bar{\boldsymbol{p}}, \boldsymbol{\phi}, \tilde{\boldsymbol{\nu}}, \boldsymbol{\beta}\right) \\
\text{subject to}: \quad & g := \frac{\sum_i^n \beta_i}{n} - \varepsilon \leq 0 \\
& \boldsymbol{\mathcal{R}}^u\left(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}, \bar{\boldsymbol{p}}, \boldsymbol{\phi}, \tilde{\boldsymbol{\nu}}, \boldsymbol{\beta}\right) = \mathbf{0} \\
& \boldsymbol{\mathcal{R}}^v\left(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}, \bar{\boldsymbol{p}}, \boldsymbol{\phi}, \tilde{\boldsymbol{\nu}}, \boldsymbol{\beta}\right) = \mathbf{0} \\
& \boldsymbol{\mathcal{R}}^p\left(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}, \bar{\boldsymbol{p}}, \boldsymbol{\phi}, \tilde{\boldsymbol{\nu}}, \boldsymbol{\beta}\right) = \mathbf{0} \\
& \boldsymbol{\mathcal{R}}^\phi\left(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}, \bar{\boldsymbol{p}}, \boldsymbol{\phi}, \tilde{\boldsymbol{\nu}}, \boldsymbol{\beta}\right) = \mathbf{0} \\
& \boldsymbol{\mathcal{R}}^{\tilde{\nu}}\left(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}, \bar{\boldsymbol{p}}, \boldsymbol{\phi}, \tilde{\boldsymbol{\nu}}, \boldsymbol{\beta}\right) = \mathbf{0} \\
& 0 \leq \boldsymbol{\beta} \leq 1
\end{aligned} \tag{5.35}$$

Here $\mathscr{V}_f\left(\beta\right)$ is equal to the specified volume of the fluid and $\mathscr{V}_0$ the total volume of the design domain. The optimization problem 4.72 is again solved using the nested formulation as shown in Figure 2.2, Chapter 2, where the discretized system of equations for the turbulent flow field are solved separately from the design problem. We now turn to the problem of obtaining adjoint based sensitivities for topology optimization for the RANS-SA model.

5.4.2 Sensitivities for RANS-SA using $\mathcal{R}$apid

Again, it is easiest to explain the algorithm and its implementation using the model mesh in Chapter 3. There are six variables $\bar{u}, \bar{v}, \bar{w}, \bar{p}, \phi$ and $\tilde{\nu}$

associated with each of the 31 cells bringing the total number of independent state variables to 186. In addition for β, we have 15 design variables. There are thus a total of 201 independent variables.

The same procedure is followed for obtaining sensitivities for the RANS-SA turbulent flow problems as was presented in the flowchart in Figure 3.10, Chapter 3. For given values of the design variable, the forward solution is performed to obtain the converged values of the flow variables. These are then transferred to the code compiled in $\mathcal{R}$apid mode. The 201 variables are set as independent variables. As mentioned, unique numbers have to be used to set the independent variables. For example, keys from 1-6 may be assigned to the mean velocities, pressure, ϕ and $\tilde{\nu}$ the for state variables of cell 1, keys 7-12 can be assigned to the state variables of cell 2 etc. (Figure 4.9, Chapter 4). The residuals are then recomputed and the adjoint system solved to obtain sensitivities to be fed into the optimizer to determine new values of $\boldsymbol{\beta}$.

The methodology for computing sensitivities for the RANS-SA model is shown in Figure 5.2 and is built based on Method 3 described in Section 4.4.3, Chapter 4. The process generates the complete Jacobian of all residuals with respect to the state and design variables. We set the converged values of cell variables $\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}, \bar{\boldsymbol{w}}, \bar{\boldsymbol{p}}, \boldsymbol{\phi}$ and $\tilde{\boldsymbol{\nu}}$ as the independent variables. The eddy viscosity computation (Eq. 5.7) followed by computation of total viscosity (5.30), brings in the dependence on the SA variable $\tilde{\nu}$ into the diffusion part of the momentum residual. The face mass flux is then computed in exactly the same way as presented in Method 3, and in addition captures the dependence

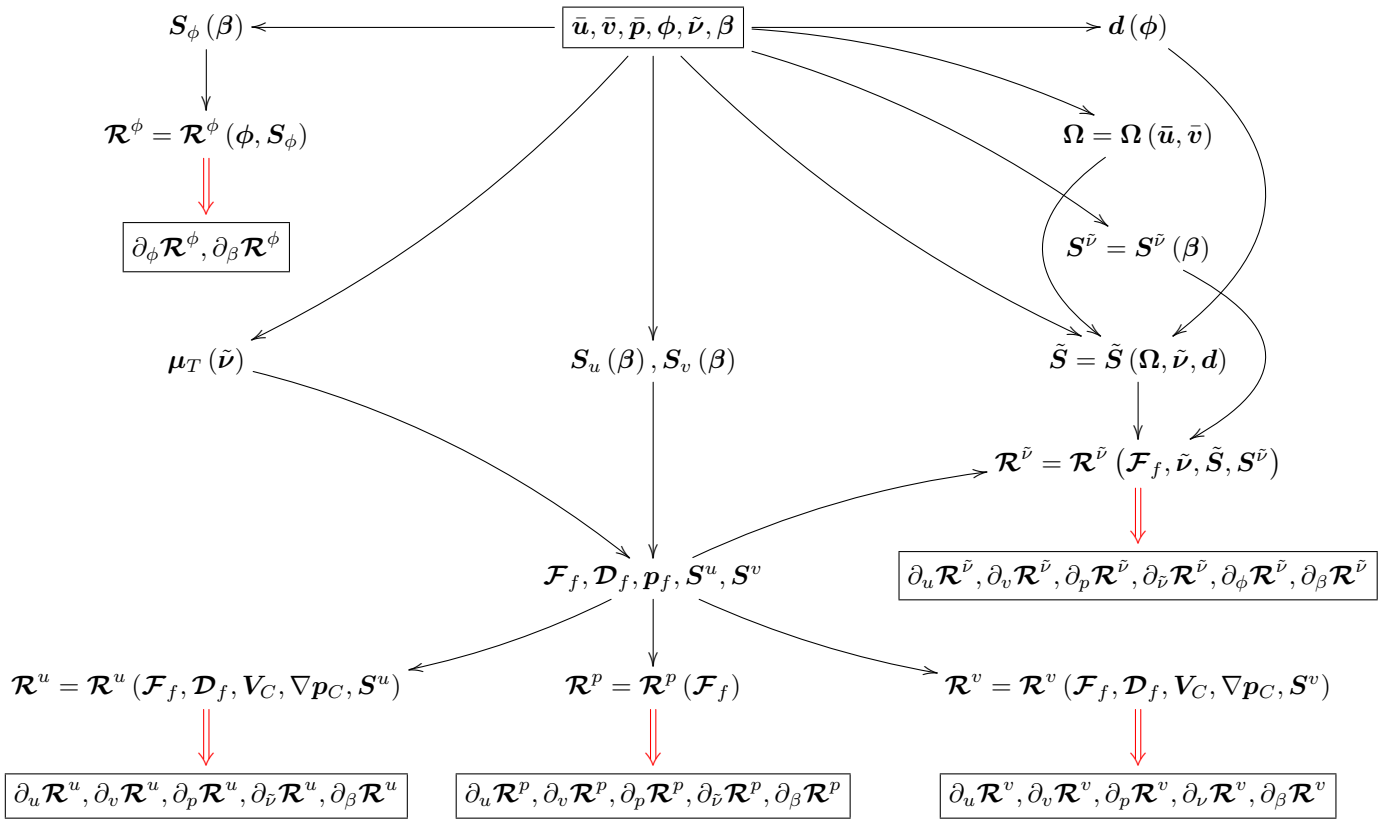


Figure 5.2: Computation of various derivative terms required to compute discrete adjoint sensitivities $dc/d\boldsymbol{\beta}$ for the RANS-SA turbulent model. The figure illustrates the sequence of steps needed to be performed to obtain the complete Jacobian of all residuals in $\mathcal{R}$apid mode.

on $\tilde{\nu}$ here. Subsequently, computation of all the momentum residuals and continuity residual generates the complete Jacobian as shown in Figure 5.2.

The magnitude of the vorticity (Eq. 5.15) and the face mass flux from the flow model feed into the residual computation for $\tilde{\nu}$. The wall distance d as a function of ϕ (Eq.5.21) is fed in from the wall function model. Thus the computation of $\boldsymbol{\mathcal{R}}^{\tilde{\nu}}$ generates the derivatives with respect to all the variables as shown in Figure 5.2. Computing the derivatives of the source terms (production, destruction and part of diffusion) in Eq. 5.11 is very laborious.

However the $\mathcal{R}$apid AD library computes all these derivatives effortlessly and automatically. Since the wall distance is computed based on the variable ϕ, the computation of $\boldsymbol{\mathcal{R}}^{\phi}$ is also required to include $\partial_{\phi}\boldsymbol{\mathcal{R}}^{\phi}$ in the adjoint computation.

The adjoint system is assembled with the complete Jacobian terms as shown in Eq. 5.36. Once the adjoint linear system is solved, the sensitivities $dc/d\beta$ can be computed from Eq. 2.32 (Chapter 2).

$$\begin{bmatrix} \partial_u \boldsymbol{\mathcal{R}}^u & \partial_v \boldsymbol{\mathcal{R}}^u & \partial_w \boldsymbol{\mathcal{R}}^u & \partial_p \boldsymbol{\mathcal{R}}^u & \partial_{\tilde{\nu}} \boldsymbol{\mathcal{R}}^u & \\ \partial_u \boldsymbol{\mathcal{R}}^v & \partial_v \boldsymbol{\mathcal{R}}^v & \partial_w \boldsymbol{\mathcal{R}}^v & \partial_p \boldsymbol{\mathcal{R}}^v & \partial_{\tilde{\nu}} \boldsymbol{\mathcal{R}}^v & \\ \partial_u \boldsymbol{\mathcal{R}}^w & \partial_v \boldsymbol{\mathcal{R}}^w & \partial_w \boldsymbol{\mathcal{R}}^w & \partial_p \boldsymbol{\mathcal{R}}^w & \partial_{\tilde{\nu}} \boldsymbol{\mathcal{R}}^w & \\ \partial_u \boldsymbol{\mathcal{R}}^p & \partial_v \boldsymbol{\mathcal{R}}^p & \partial_w \boldsymbol{\mathcal{R}}^p & \partial_p \boldsymbol{\mathcal{R}}^p & \partial_{\tilde{\nu}} \boldsymbol{\mathcal{R}}^p & \\ \partial_u \boldsymbol{\mathcal{R}}^{\tilde{\nu}} & \partial_v \boldsymbol{\mathcal{R}}^{\tilde{\nu}} & \partial_w \boldsymbol{\mathcal{R}}^{\tilde{\nu}} & \partial_p \boldsymbol{\mathcal{R}}^{\tilde{\nu}} & \partial_{\tilde{\nu}} \boldsymbol{\mathcal{R}}^{\tilde{\nu}} & \partial_{\phi} \boldsymbol{\mathcal{R}}^{\tilde{\nu}} \\ & & & & & \partial_{\phi} \boldsymbol{\mathcal{R}}^{\phi} \end{bmatrix} \begin{Bmatrix} \psi_u \\ \psi_v \\ \psi_w \\ \psi_{p_C} \\ \psi_{\tilde{\nu}} \\ \psi_{\phi} \end{Bmatrix} = \begin{Bmatrix} \partial_u c \\ \partial_v c \\ \partial_w c \\ \partial_p c \\ \partial_{\tilde{\nu}} c \\ \partial_{\phi} c \end{Bmatrix} \tag{5.36}$$

5.5 Results

We consider a single test case to demonstrate topology optimization for turbulent flow. The schematic for the test case is shown in Figure 5.3 with $a = L$. The design domain is made of a 100X100 Cartesian mesh.

A velocity inlet with a fully developed turbulent velocity profile is specified. The velocity profile is generated by simulating a channel flow with same width but with a considerably longer length using the SA model, where a plug flow at $Re = 5000$ is specified at inlet and pressure at the outlet. The fully developed turbulent profile near the outlet of this channel is used as the inlet

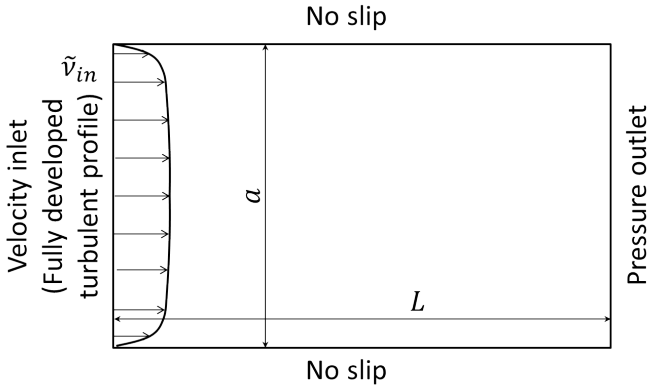


Figure 5.3: Sensitivity of total pressure drop in a channel with respect to viscosity of fluid.

velocity profile for the topology optimization problem.A turbulent viscosity ration of 1 was applied at the inlet, that corresponds to $\tilde{\nu}_{in} = 4.62284\nu$ where ν is the molecular kinematic viscosity of the fluid. Zero pressure is specified at the outlet and no-slip at the top and bottom boundaries. The cost function is to minimize the total pressure drop across the channel in horizontal flow direction.

$$c = \min \left(\int_{\Gamma_{in}} pdA - \int_{\Gamma_{out}} pdA \right) \tag{5.37}$$

The goal is to fill 50% of the design space with solid material i.e. $\varepsilon = 0.5$. A uniform distribution for $\boldsymbol{\beta} = 0.5$ is chosen as the initial condition for the design variable. The limiting values for the SIMP function α^s and α^f for the flow model, SA model and wall distance model, are listed in order.

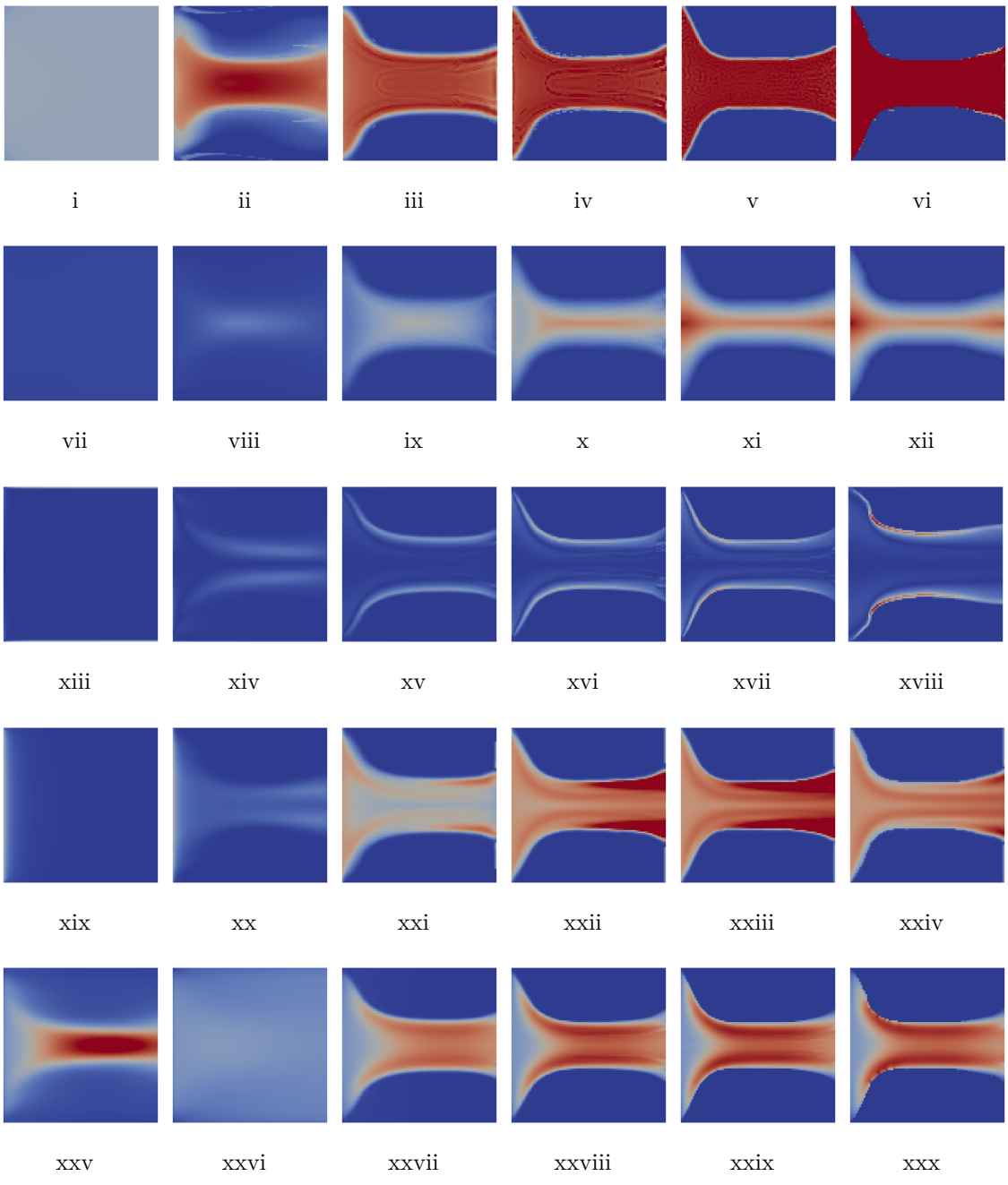


Figure 5.4: (i)-(vi) Illustration of the evolution of the topologies for turbulent flow in a channel at $Re = 5000$. Corresponding evolution of wall distance d in (vi)-(xii), vorticity magnitude in (xiii)-(xviii), eddy viscosity ν_T in (xix)-(xxiv) and velocity magnitude in (xxv)-(xxx).

$$\begin{aligned} \alpha^s_{u,v,w} &= 100, \quad \alpha^f_{u,v,w} = 0 \\ \alpha^s_{\tilde{\nu}} &= 10, \quad \alpha^f_{\tilde{\nu}} = 0 \\ \alpha^s_{\phi} &= 10, \quad \alpha^f_{\phi} = 0 \end{aligned} \tag{5.38}$$

These parameters were obtained by trial and error. A proper analysis to determine the range of these parameters must be investigated to extend the methodology to any general problem. With the current parameters as in Eq. 5.38, the topology optimization was carried out. Figure 5.4 shows the evolution of various variables of interest along the optimization process. In Figure 5.4, (i)-(vi) illustrates the evolution of β, where the red region represents the solid and blue the fluid. We finally get a very similar topology as the one we got for laminar flow, but with a wider exit mouth. The evolution of wall distance d is shown in sub-figures (vi)-(xii). The center-point at the inlet is indeed the point with the largest value of d in the fluid domain, which is exactly captured towards the end of the optimization as is shown in the last sub-figure. The corresponding values for vorticity magnitude at various steps is shown in (xiii)-(xviii). Next we show the values for eddy viscosity ν_T in sub-figures (xix)-(xxiv). We note that the values for eddy viscosity are high near the inlet and at the exit. We also show the velocity distribution at the same steps in sub-figures (xxv)-(xxx). While comparing the eddy viscosity figures to the velocity figures, one can see the influence of the particular distribution of eddy viscosity on the velocity distribution.

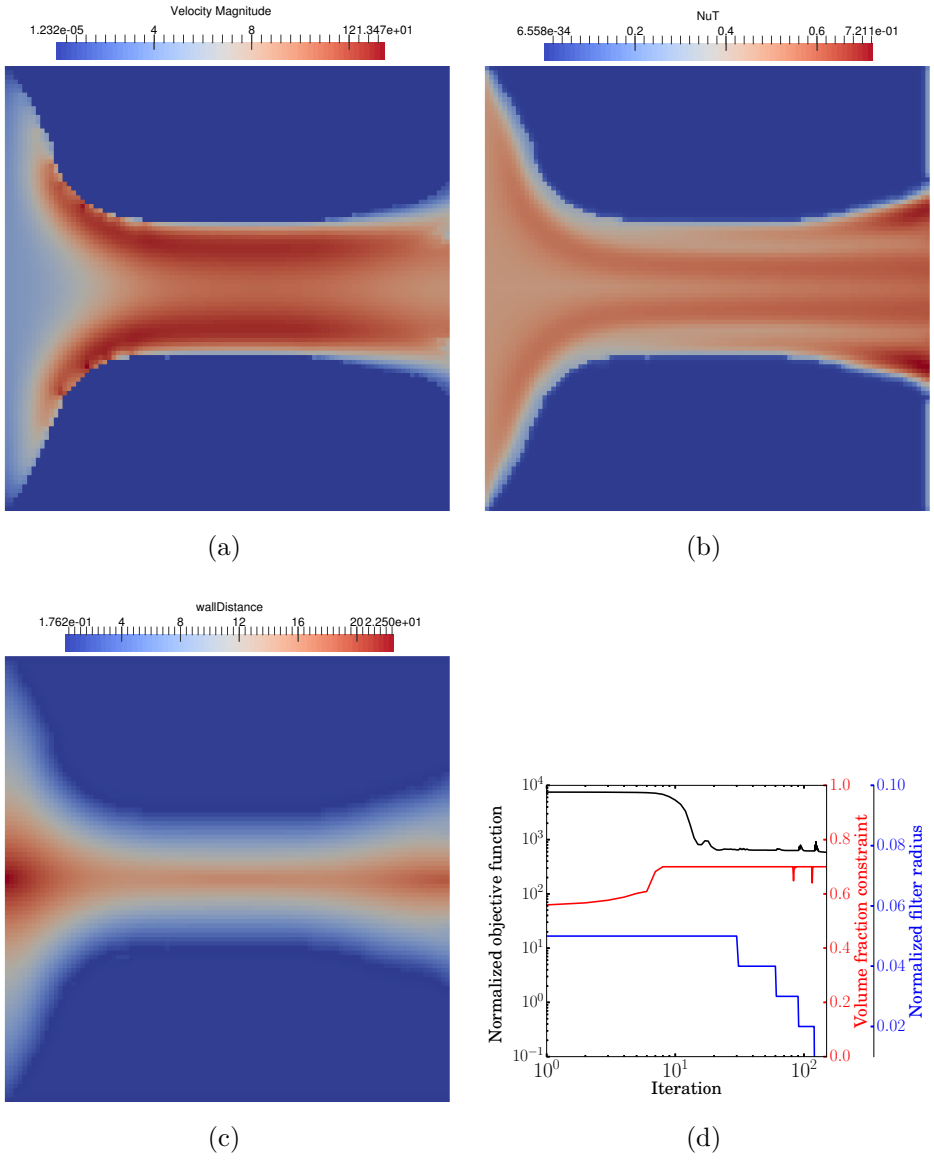


Figure 5.5: Distribution of state variables – velocity magnitude (a), eddy viscosity ν_T (b) and wall distance d (c) for the final topology presented in Figure 5.4 (vi). (d) Normalized objective function, volume fraction of solid and filter radius versus iteration number during the process of optimization.

Figure 4.21 shows the distribution of different state variables, i.e. velocity magnitude, eddy viscosity and wall distance d for the final topology shown in Figure 5.4 (vi). Figure 4.21 (d) also shows the evolution of normalized cost function, i.e. the total pressure drop across the inlet and and the outlet normalized with the dynamic pressure at the inlet, against iteration number during the process of optimization. The volume fraction of the solid material and the filter radius are also shown.

The final optimal shape may be rationalized in the much the same was as for laminar flow. Given that the domain must be filled with a minimum of 50% solid, the optimization algorithm produces a gently contoured converging diverging nozzle. The diverging section is produced because pressure recovery at the exit helps reduce the overall pressure drop. However, too large an exit area would make it difficult to fill the domain with the necessary amount of solid. Furthermore, too narrow a throat increases viscous losses. A balance between these competing imperatives produces the final optimal geometry.

5.6 Closure

In this chapter, we extended the infrastructure built in Chapters 3 and 4 to compute sensitivities and perform topology optimization for turbulent flows using the Spalart-Allmaras turbulence model. Even though the governing equations have grown significantly in complexity, The Rapid infrastructure allows relatively straightforward problem-agnostic implementation. A test case that serves as the proof of concept of the applicability of topology optimization

for turbulent flow was presented. In the next chapter, we consider combined flow and heat transfer for both laminar and turbulent flow.

Chapter 6

Convective heat transfer applications

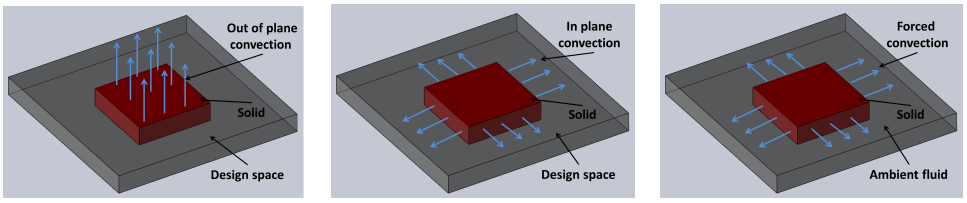
In this chapter, we develop the topology optimization methodology for forced convection applications in which the Navier-Stokes and energy equations are coupled. As before, the procedure is implemented in an unstructured co-located finite volume framework employing a sequential pressure-based algorithm. Sensitivities are obtained using the $\mathcal{R}$apid AD library. The coupled nature of the problem requires the use of a multi-objective cost function reflecting both flow and heat transfer components. The techniques developed here are valid for both laminar and turbulent flows.

6.1 Introduction

Convective heat transfer is central to virtually every industry application in which thermal transport occurs. Here, the focus is on heat transfer occurring between a fluid in motion and an adjacent solid surface[130]. The goal is generally to enhance the heat transfer between two media. Convection heat transfer is generally classified into forced and natural convection. When the flow of fluid is induced by external forces it is termed as forced convection. When the flow arises due to buoyancy caused by spatial variations in

density, it is termed natural convection. We start this chapter by presenting a literature survey on topology optimization for forced convection.

Approximate representations of forced convection heat transfer have been published in conjunction with topology optimization. For plane problems where the out-of-plane dimension is small, in-plane convection was neglected and a constant out-of-plane convective boundary condition applied, e.g. [131, 132, 42]. Here, the actual fluid flow is not solved; only its effect is included through a convection heat transfer coefficient. This is depicted in Figure 6.1(a). In order to include the effects of in-plane convective heat transfer on the topology, convective boundary conditions are applied in different ways [133, 132, 15, 42]. This is shown in Figure 6.1(b). Both the methods rely on specifying a heat transfer coefficient. Since the convective heat transfer coefficient is a strong function of geometry, which is not known *a priori*, this approach is not physically correct and can give incorrect designs. To obtain realistic designs, it is necessary to couple flow and heat transfer together and to explicitly resolve the flow and temperature fields, as shown in Figure 6.1(c). Here, both the fluid and solid design regions are explicitly modelled to obviate the need for specifying a heat transfer coefficient.



(a) (b) (c)

Figure 6.1: Various ways of incorporating convection into topology optimization (diagrams re-created based on [5]). (a) Constant surface (out of plane) convection using heat transfer coefficient to model convective heat loss (b) Constant side (in-plane) convection using heat transfer coefficient to model convective heat loss (c) Forced convection with explicit solution of combined flow and heat transfer to model forced convection.

For coupled flow and heat transfer problems, it is necessary to solve the Navier-Stokes equations with the Brinkman formulation coupled with the energy equation (Section 6.2.1). Only a few papers have been published with this methodology. Yoon [5] implemented topology optimization to design heat dissipating structures subjected to forced convection, with an objective to minimize thermal compliance. The Navier-Stokes equations with the Brinkman formulation were coupled with the energy equation and were numerically solved using the finite element method. All material properties including thermal conductivity, specific heat capacity, macroscopic density and inverse permeability, were modelled with the SIMP formulation. A few simple 2D examples were presented. The intent was to come up with optimized structures that would maximize heat dissipation. Lee [134] also presented results for simple 2D and 3D convective heat exchanger geometries design using topology optimization within a finite element framework for Reynolds number up to

1000. The author discussed some of the numerical issues such as velocity and pressure oscillations and ways to circumvent them.

Koga *et al.* [135] performed topology optimization with coupled flow and energy equations on a 2-D planar domain to design a micro-channel heat sink using the finite element method. The authors assumed Stokes flow, while retaining both conduction and convection terms in the energy equation. In the internal flow problem that they solved, flow enters and exits the domain through opposing faces convecting the heat generated in the solid portions of the domain. The objective function used is a weighted combination of two opposing functionals, one minimizing the fluid power dissipation and the other maximizing the overall heat dissipation. The combined cost function was formulated as the weighted sum of the log of the two individual functions. The log function was used to bring the objective functions to approximately same scale. The extruded geometry of the optimized topology was fabricated with aluminum and experimental results were presented with water flow on the channels. Similar work was done by Matsumori *et al.* for various Reynolds number (10-100) [136]. However the authors assume both solid and fluid regions to have same thermal conductivity citing simplification.

Topology optimization with a multi-objective cost function was also done by Marck *et al.* [33] using the finite volume method based on a staggered-mesh sequential algorithm (SIMPLER) on 2D structured meshes to design a simple heat exchanger in a 2-D domain. In this internal laminar flow problem, fluid entering through a portion of the left boundary is heated by top and

bottom walls maintained at specified temperatures, before exiting through a portion of the right boundary. The governing equations are the Brinkman penalized Navier-Stokes equations coupled with the energy equation. The conductivity and inverse permeability are parameterized in terms of element density using RAMP functions. The objective function consists of the difference in the pressure and the enthalpy between the inlet and the outlet. By weighting these two parts differently, the authors arrived at different optimal topologies.

We close the survey with some related works. As mentioned in Chapter 5, Papoutsis *et al.* presented topology optimization of forced convection using a continuous adjoint method in finite volume framework [123, 137, 125]. Alexandersen *et al.* published work on natural convection, where both flow and thermal equations have two way coupling[54, 23]. They used a single thermal objective function.

In this chapter we extend our topology optimization methodology to combined flow and heat transfer. The chapter is sub-divided into two parts. The first part deals with topology optimization for forced convection applications with laminar flow with all the infrastructure developed until now. A brief discussion of the governing equations and numerical methods emphasizing the coupling of two physical models is presented. The new concept introduced in this chapter is multi-objective optimization and is dealt with in detail. The changes needed for the sensitivity computation are also presented. A test case is chosen to demonstrate the methodology as a proof of concept. The second

part discusses the implementation of forced convection with turbulent flow. Example forced convection test problems are presented to demonstrate the efficacy of the implementation.

6.2 Forced convection with laminar flow

6.2.1 Governing equations and boundary conditions

The governing equations governing laminar fluid flow and heat transfer are:

$$\begin{aligned} \nabla \cdot (\rho \vec{v} u) &= \nabla \cdot (\mu \nabla u) - \nabla p \cdot \vec{i} - \alpha_u (\beta) u \\ \nabla \cdot (\rho \vec{v} v) &= \nabla \cdot (\mu \nabla v) - \nabla p \cdot \vec{j} - \alpha_v (\beta) v \\ \nabla \cdot (\rho \vec{v}) &= 0 \\ \nabla \cdot (\rho C_p \vec{v} T) &= \nabla \cdot (k (\beta) \nabla T) + S^T (\beta) \end{aligned} \tag{6.1}$$

Here, the notation for fluid flow is the same as that in Chapter 4. T is temperature, C_p is the specific heat capacity, and k is the thermal conductivity.

As usual, the boundary is partitioned into solid boundaries, and inlet and outlet regions. A no-slip boundary condition is applied at solid boundaries for the flow equations. For the energy equation, either a temperature (Dirichlet) or a heat flux (Neumann) boundary condition may be applied at solid boundaries. Furthermore, for solid regions embedded in a fluid, the formulation permits conjugate heat transfer using the SIMP/RAMP formulation, as described in Chapter 2.

As before, either velocity or pressure boundary may be applied at both inlets and outlets. For the energy equation, temperature values must be specified at the inlet. Under the assumption that there is no reverse flow at the outlet, temperature at outlets is extrapolated from the interior.

For topology optimization, the momentum equations have the usual Brinkman's term interpolated in terms of the design variable. In addition, thermal conductivity is also interpolated here, similar to what was done for topology optimization of heat conduction applications. A RAMP formulation is used here [CITE]. The quantities $\alpha_u(\beta)$, $\alpha_v(\beta)$ and $k(\beta)$ are interpolated within the two limits with the interpolation formula given by,

$$\begin{aligned} \alpha_u(\beta) &= \alpha_f + (\alpha_s - \alpha_f)\frac{(1-\beta)}{(1+q\cdot\beta)} \\ k(\beta) &= k_f + (k_s - k_f)\frac{(1-\beta)}{(1+q\cdot\beta)} \end{aligned} \tag{6.2}$$

This interpolation formula ensures that $\beta = 1$ signifies fluid, and $\beta = 0$ signifies solid.

6.2.2 Numerical method

Figure 6.2 depicts the sequential algorithm for forced convection. We have discussed in detail how to solve flow equations using SIMPLE algorithm in Chapter 4. After the flow variables get converged, the face mass flux $\mathcal{F}_f$ is transferred to the thermal model as shown in the figure. The term $\mathcal{F}_f C_p$ is used to discretize the convective term in the energy equation, following procedures

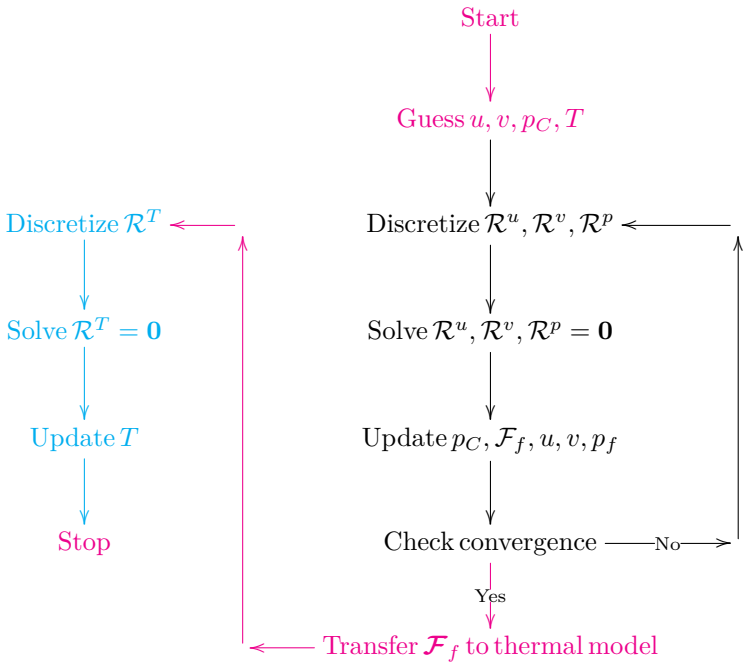


Figure 6.2: Flowchart for solving forced convection sequentially using the SIMPLE algorithm.

discussed in [31]. Accordingly the convection term can be discretized using upwinding scheme. If properties are temperature independent, the convection term is linear, unlike for the flow equations. In this event, the energy equation is thus solved for temperature after the flow field has converged, as shown in the figure.

6.2.3 Multi-objective cost function

Until now we have dealt with only one objective or cost function that was minimized. That was primarily because we solved problems governed by a single physical model. It is noted that it is possible to define more than one

cost function (which can be opposing as well) from a single physical model [16]. The goal of doing forced convection is to maximize some metric of heat transfer. One common goal is to maximize fluid enthalpy changed ΔH from the inlet to the outlet: the bulk (or mean) temperature T_m of the fluid at the outlet boundary or the enthalpy change ΔH from the inlet to the outlet of the fluid, respectively given by,

$$\Delta H = \int_{\Gamma_{out}} \rho \vec{v} C_p T \cdot d\vec{A} - \int_{\Gamma_{in}} \rho \vec{v} C_p T \cdot d\vec{A} = \sum_{outlet} \mathcal{F}_f C_p T A_f - \sum_{inlet} \mathcal{F}_f C_p T A_f \tag{6.3}$$

At the same time, we would like to minimize the energy expended in driving the fluid through the domain. A good metric of this energy is the total pressure drop of the fluid across the boundaries,

$$\Delta P = \left(\int_{\Gamma_{in}} pdA - \int_{\Gamma_{out}} pdA \right) \tag{6.4}$$

Most often trying to minimize pressure drop reduces the gain in heat transfer and vice versa. Therefore the cost function for such problems is generally chosen as a weighted sum of two cost functions, one from each physical model, and is termed as a multi-objective cost function. The weights determine the preference levels of one cost function over other. For forced convection problems, our objective can be thus to determine topologies that would minimize the total pressure drop of the fluid across the boundary while maximizing the mean temperature of the fluid at the outlet formulated as,

$$c = \gamma_F \cdot \Delta P + \gamma_T \cdot \Delta T_b \tag{6.5}$$

where γ_F and γ_T are the weights for the pressure drop and the bulk temperature respectively. In Eq. 6.5, ΔT_b is the difference in bulk temperature between the inlet and the outlet. The bulk temperature T_b for a given boundary Γ is defined as,

$$T_b = \frac{\int_\Gamma \rho \vec{v} C_p T \cdot d\vec{A}}{\int_\Gamma \rho \vec{v} C_p \cdot d\vec{A}} = \frac{\sum_\Gamma \mathcal{F}_f C_p T A_f}{\sum_\Gamma \mathcal{F}_f C_p A_f} \tag{6.6}$$

$$\implies \Delta T_b = T_{b,outlet} - T_{b,inlet} \tag{6.7}$$

Note that the units of the weights γ_F and γ_T are different. For any generic cost function for flow and thermal problems given by c_F and c_T as functions for state and design variables, the topology optimization problem for forced convection may be stated as,

$$\begin{aligned}
\min: \quad & c = \gamma_F \cdot c_F\left(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{p}_C, \boldsymbol{\beta}\right) + \gamma_T \cdot c_T\left(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{p}_C, \boldsymbol{T}, \boldsymbol{\beta}\right) \\
\text{subject to}: \quad & g := \frac{\sum_i^n \beta_i}{n} - \varepsilon \leq 0 \\
& \boldsymbol{\mathcal{R}}^u\left(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{p}_C, \boldsymbol{T}, \boldsymbol{\beta}\right) = \boldsymbol{0} \\
& \boldsymbol{\mathcal{R}}^v\left(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{p}_C, \boldsymbol{T}, \boldsymbol{\beta}\right) = \boldsymbol{0} \\
& \boldsymbol{\mathcal{R}}^p\left(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{p}_C, \boldsymbol{T}, \boldsymbol{\beta}\right) = \boldsymbol{0} \\
& \boldsymbol{\mathcal{R}}^T\left(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{p}_C, \boldsymbol{T}, \boldsymbol{\beta}\right) = \boldsymbol{0} \\
& 0 \leq \boldsymbol{\beta} \leq 1
\end{aligned} \tag{6.8}$$

The volume constraint ε is specified for the total volume fraction of the fluid. The procedure for performing topology optimization with the nested

formulation remains the same. We only discuss the sensitivity computation for forced convection in detail, due to the multi-objective cost function.

6.2.4 Sensitivity computation for forced convection

By the end of forward solution for a given set of design variable β, we have converged values of the state variables - i.e. the flow variables u, v, p_C and temperature T. These values are transferred to the $\mathcal{R}$apid code. Figure 6.3 shows how to compute the necessary derivatives for sensitivity computation. As usual we compute the residuals associated with the various state variables as described briefly.

The momentum and continuity residuals are computed as usual (Chapter 4). The temperature does not influence the flow for the forced convection problems considered here. Thus we do not expect any dependence of momentum or continuity residuals on temperature T and hence do not find these residual derivatives with respect to temperature. The mass flux computed in the process gets transferred to the computation of temperature residual, as shown in Figure 6.3. Therefore the temperature residual computation is accompanied by derivative computation with respect to all variables.

The adjoint system is assembled with the complete Jacobian terms as shown in Eq. 6.9. Notice that only one-way coupling between flow and temperature variables is reflected in the Jacobian.

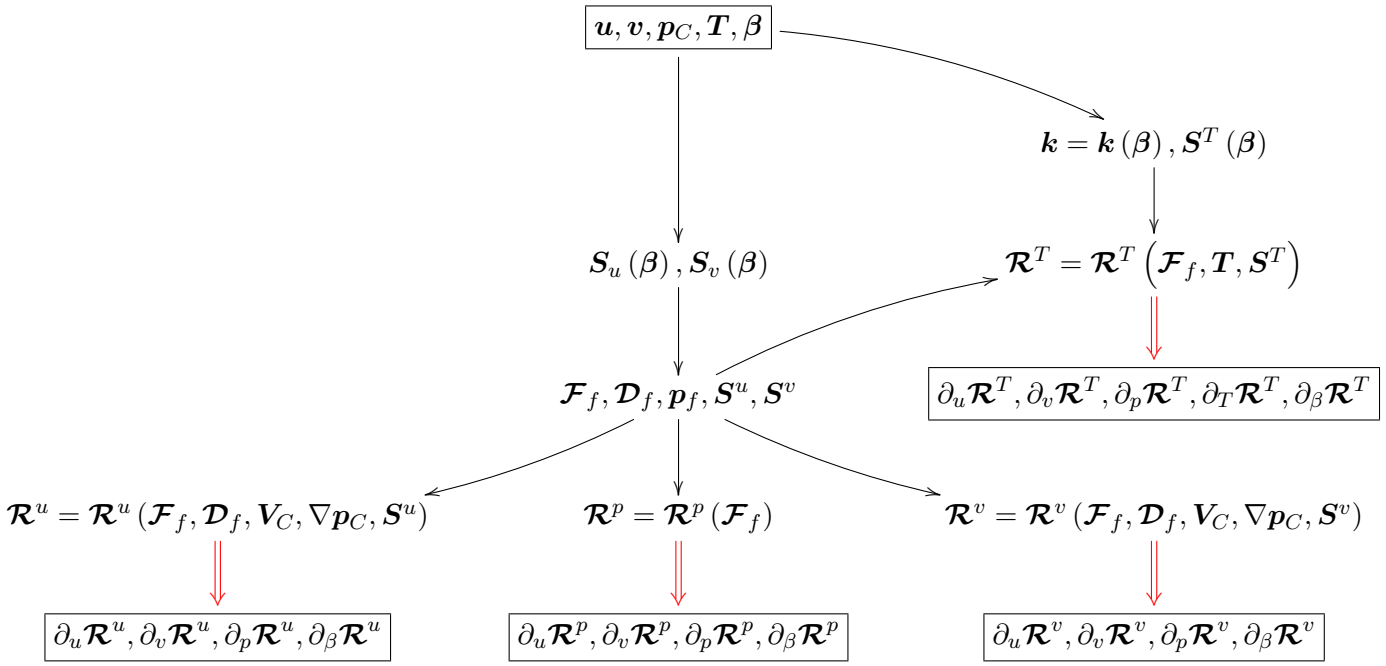


Figure 6.3: Figure depicts the computational of various derivative terms required to compute discrete adjoint sensitivities $dc/d\boldsymbol{\beta}$ for the RANS-SA turbulent model for used in topology optimization. The figure illustrates the sequence of steps needed to be performed to obtain the complete Jacobian of all residuals in $\mathcal{R}$apid mode.

$$\begin{bmatrix} \partial_u \boldsymbol{\mathcal{R}}^u & \partial_v \boldsymbol{\mathcal{R}}^u & \partial_w \boldsymbol{\mathcal{R}}^u & \partial_p \boldsymbol{\mathcal{R}}^u & \\ \partial_u \boldsymbol{\mathcal{R}}^v & \partial_v \boldsymbol{\mathcal{R}}^v & \partial_w \boldsymbol{\mathcal{R}}^v & \partial_p \boldsymbol{\mathcal{R}}^v & \\ \partial_u \boldsymbol{\mathcal{R}}^w & \partial_v \boldsymbol{\mathcal{R}}^w & \partial_w \boldsymbol{\mathcal{R}}^w & \partial_p \boldsymbol{\mathcal{R}}^w & \\ \partial_u \boldsymbol{\mathcal{R}}^p & \partial_v \boldsymbol{\mathcal{R}}^p & \partial_w \boldsymbol{\mathcal{R}}^p & \partial_p \boldsymbol{\mathcal{R}}^p & \\ \partial_u \boldsymbol{\mathcal{R}}^T & \partial_v \boldsymbol{\mathcal{R}}^T & \partial_w \boldsymbol{\mathcal{R}}^T & \partial_p \boldsymbol{\mathcal{R}}^T & \partial_T \boldsymbol{\mathcal{R}}^T \end{bmatrix} \begin{Bmatrix} \psi_u \\ \psi_v \\ \psi_w \\ \psi_{p_C} \\ \psi_T \end{Bmatrix} = \gamma_F \begin{Bmatrix} \partial_u c_F \\ \partial_v c_F \\ \partial_w c_F \\ \partial_p c_F \\ 0 \end{Bmatrix} + \gamma_T \begin{Bmatrix} \partial_u c_T \\ \partial_v c_T \\ \partial_w c_T \\ \partial_p c_T \\ \partial_T c_T \end{Bmatrix} \tag{6.9}$$

The RHS of the adjoint system for multi-objective optimization problems requires special mention. The RHS is comprised of the discrete partial derivatives of the cost function with respect to all the state variables. Since there are two weighted cost functions, the weights also appear in the RHS as shown in Eq. 6.9.. Once this adjoint linear system is solved, the sensitivities $dc/d\beta$ can be computed from Eq. 2.32 (Chapter 2).

6.3 Forced convection with turbulent flow

Turbulent forced convection involves all the four physical models discussed thus far - the fluid flow solver, the SA turbulence model, the thermal model and wall distance model. The solution process using a sequential algorithm is depicted in Figure 6.4. The flow chart is self explanatory for the most part. We focus on the concept of eddy conductivity.

Forced convection involving turbulent flows involves all the four physical models discussed until now - flow solver, SA turbulence model, thermal model and wall distance model. The solution process, for a forced convection with underlying turbulent flow field, using a sequential algorithm is depicted in Figure 6.4. The flow chart is self explanatory for most of its parts from

our previous chapters, with the need to only discuss the the concept of eddy conductivity.

Similar to the quantity eddy viscosity enhancing fluid diffusion, a eddy conductivity component also gets introduced into the total conductivity of the thermal model. The rationale behind adding an eddy conductivity component makes sense because turbulence enhances heat transfer. Therefore

$$k_{total} = k_f + k_T \tag{6.10}$$

where k_T is the eddy conductivity. Based on the Prandtl number, once can express the the conductivity in terms of other material properties as as follows,

$$\begin{aligned} Pr &= \frac{\nu}{\alpha} = \frac{\rho C_p \nu}{k} \\ \implies k &= \frac{\rho C_p \nu}{Pr} \end{aligned} \tag{6.11}$$

Based on the same logic, the eddy conductivity is defined as

$$k_T = \frac{\rho C_p \nu_T}{Pr} \tag{6.12}$$

Here Prandtl number in Eq. 6.12 is a model parameter, which needs to be specified by the user. Therefore one can determine the value for k_T from a value for ν_T obtained from a turbulent model, say the SA model.

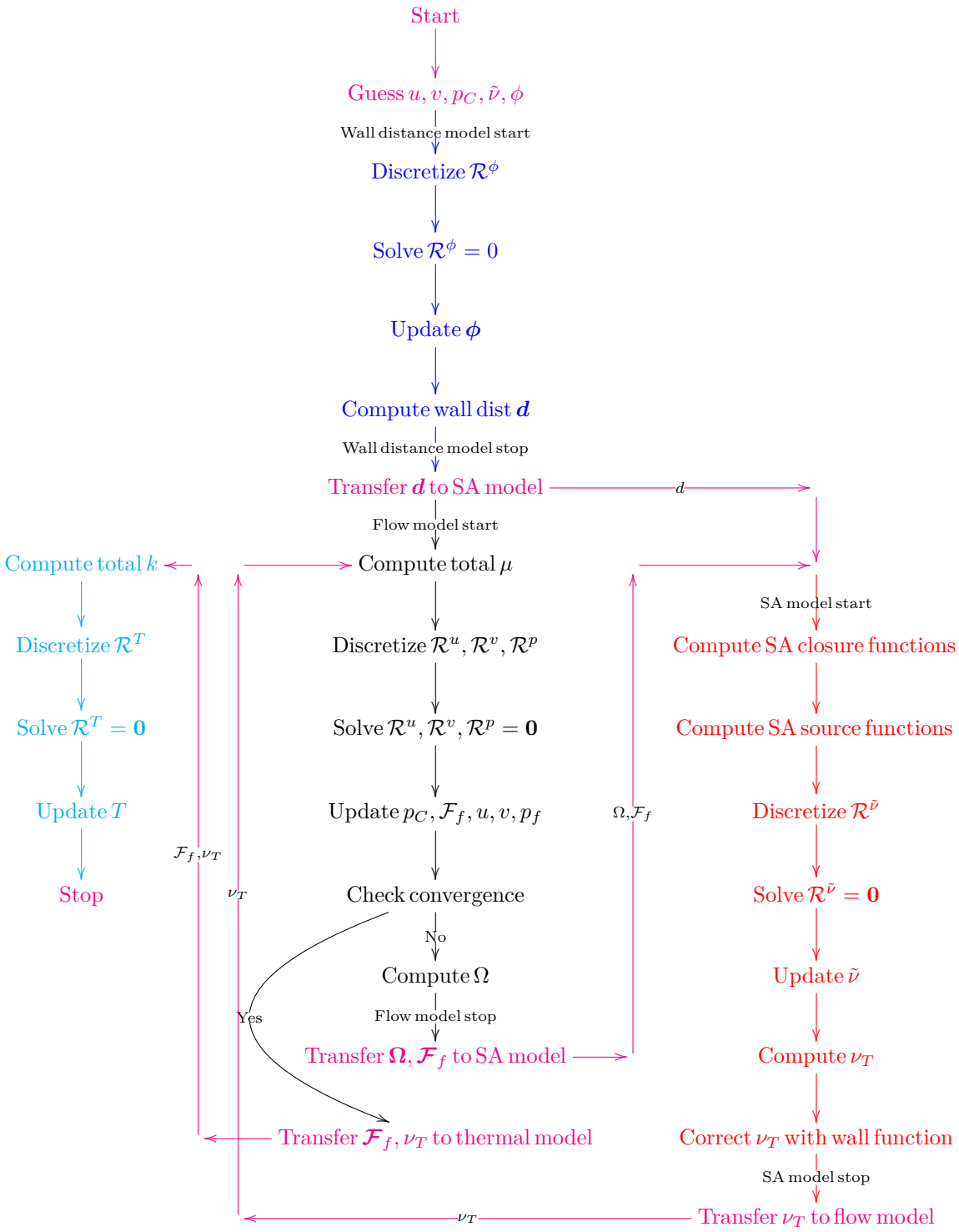


Figure 6.4: Flowchart for a forced convection for turbulent flow turbulence model employing SIMPLE algorithm with RANS-SA model. Here the algorithm is an interplay of four different models.

As can be seen in Figure, 6.4 once face mass flux and the eddy viscosity is transferred to thermal model, eddy conductivity is computed. This is followed by the usual process of discretization and solution of temperature equations.

Figure 6.5 depicts the flow diagram for computing the sensitivities for turbulent flow. It is mostly self explanatory, except that the eddy conductivity computed from converged variables feed into the temperature residual computation.

The adjoint system with the Jacobian now takes the following form.

$$\begin{bmatrix} \partial_u \boldsymbol{\mathcal{R}}^u & \partial_v \boldsymbol{\mathcal{R}}^u & \partial_w \boldsymbol{\mathcal{R}}^u & \partial_p \boldsymbol{\mathcal{R}}^u & \partial_{\tilde{\nu}} \boldsymbol{\mathcal{R}}^u & & \\ \partial_u \boldsymbol{\mathcal{R}}^v & \partial_v \boldsymbol{\mathcal{R}}^v & \partial_w \boldsymbol{\mathcal{R}}^v & \partial_p \boldsymbol{\mathcal{R}}^v & \partial_{\tilde{\nu}} \boldsymbol{\mathcal{R}}^v & & \\ \partial_u \boldsymbol{\mathcal{R}}^w & \partial_v \boldsymbol{\mathcal{R}}^w & \partial_w \boldsymbol{\mathcal{R}}^w & \partial_p \boldsymbol{\mathcal{R}}^w & \partial_{\tilde{\nu}} \boldsymbol{\mathcal{R}}^w & & \\ \partial_u \boldsymbol{\mathcal{R}}^p & \partial_v \boldsymbol{\mathcal{R}}^p & \partial_w \boldsymbol{\mathcal{R}}^p & \partial_p \boldsymbol{\mathcal{R}}^p & \partial_{\tilde{\nu}} \boldsymbol{\mathcal{R}}^p & & \\ \partial_u \boldsymbol{\mathcal{R}}^{\tilde{\nu}} & \partial_v \boldsymbol{\mathcal{R}}^{\tilde{\nu}} & \partial_w \boldsymbol{\mathcal{R}}^{\tilde{\nu}} & \partial_p \boldsymbol{\mathcal{R}}^{\tilde{\nu}} & \partial_{\tilde{\nu}} \boldsymbol{\mathcal{R}}^{\tilde{\nu}} & \partial_\phi \boldsymbol{\mathcal{R}}^{\tilde{\nu}} & \\ & & & & & \partial_\phi \boldsymbol{\mathcal{R}}^{\phi} & \\ \partial_u \boldsymbol{\mathcal{R}}^T & \partial_v \boldsymbol{\mathcal{R}}^T & \partial_w \boldsymbol{\mathcal{R}}^T & \partial_p \boldsymbol{\mathcal{R}}^T & \partial_{\tilde{\nu}} \boldsymbol{\mathcal{R}}^T & & \partial_T \boldsymbol{\mathcal{R}}^T \end{bmatrix} \begin{Bmatrix} \psi_u \\ \psi_v \\ \psi_w \\ \psi_{p_C} \\ \psi_{\tilde{\nu}} \\ \psi_\phi \\ \psi_T \end{Bmatrix} = \begin{Bmatrix} \partial_u c \\ \partial_v c \\ \partial_w c \\ \partial_p c \\ \partial_{\tilde{\nu}} c \\ \partial_\phi c \\ \partial_T c \end{Bmatrix} \tag{6.13}$$

6.4 Results

We consider a test case that the demonstrates topology optimization for forced convection under laminar flow. A similar problem has been presented in [33] and [136]. A square domain discretized with a 100X100 Cartesian mesh is considered, as shown in Figure 6.6. A small portion of the left boundary serves as the inlet and a similar region on the right boundary serves as the

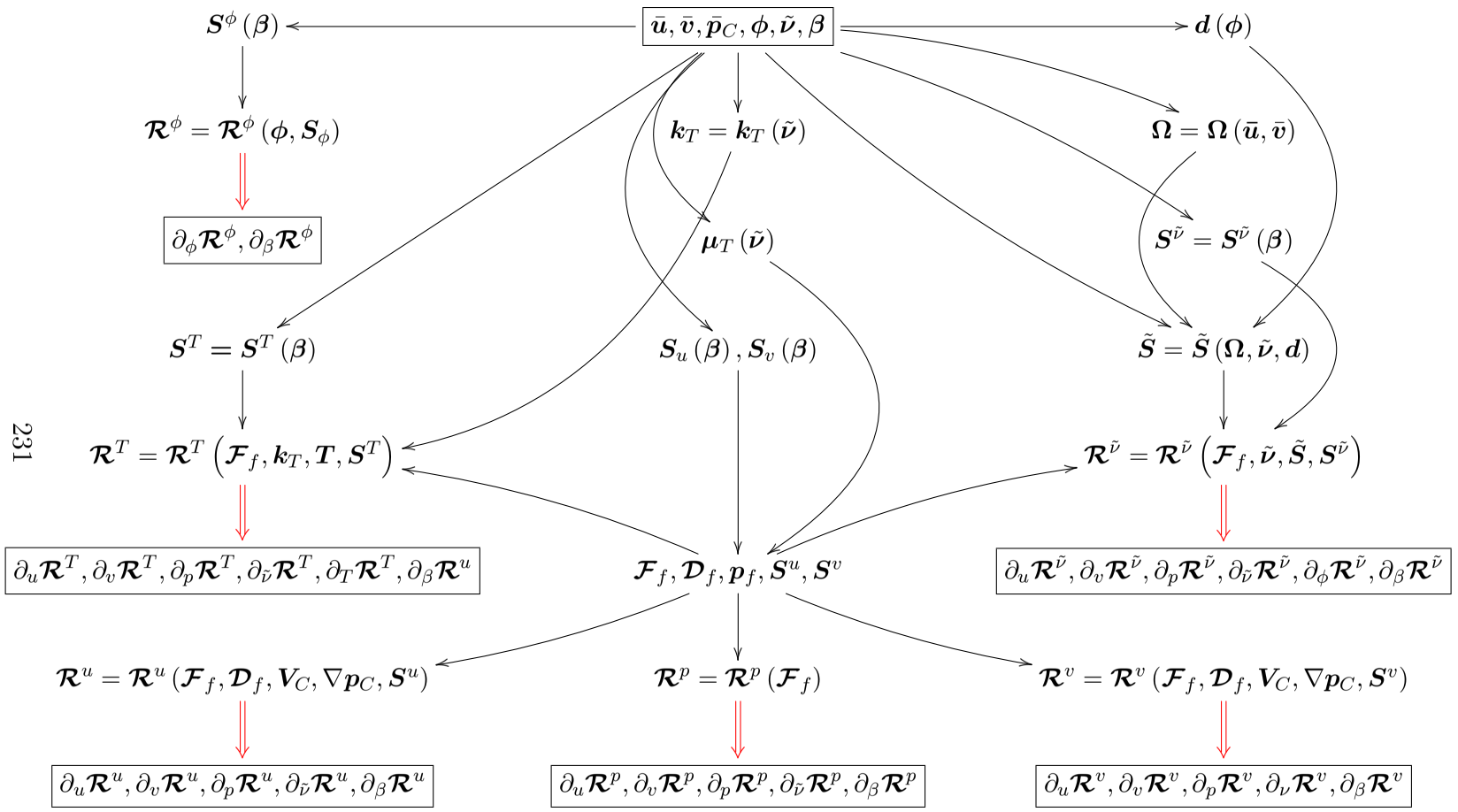


Figure 6.5: The computation of sensitivities for forced convection with turbulent flow using $\mathcal{R}$apid library.

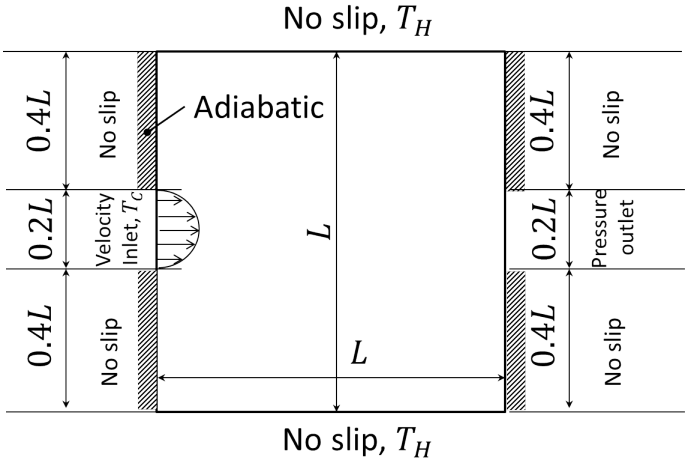


Figure 6.6: Test case for demonstrating topology optimization for forced convection.

outlet. A fully-developed laminar velocity profile is specified at the inlet and a zero pressure boundary at the outlet. A no-slip condition is specified on all the remaining boundaries. The incoming fluid at the inlet is set at a cold temperature, T_C. The top and bottom boundaries are specified at a higher temperature, T_H. Adiabatic boundaries are specified as shown in the figure. The temperature at the outlet need not be specified; it is extrapolated from the interior.

The thermal cost function maximizes the bulk temperature rise ΔT_b at the outlet (Eq. 6.7). At the same time, the total pressure drop across the channel should also be minimized. Therefore we have a multi-objective cost function, as in Eq. 6.5 and 4.72. For convenience, we rewrite the cost function in Eq. 6.5 with a single weightage factor γ without any loss of generality as follows,

$$c = \Delta P - \gamma \Delta T_b \tag{6.14}$$

The parameters of the problem are now described. The conductivity ratio is chosen to be $k_s/k_f = 100$, where $k_s = 10$ W/mK and $k_f = 0.1$ W/mK. The limits of α are kept at $\alpha_s = 100$ and $\alpha_f = 0.0001$ for the momentum equations. Density ρ , specific heat capacity C_p and viscosity of the fluid are set to unity. The Prandtl number for the fluid is thus given by,

$$Pr = \frac{C_p \mu}{k_f} = 10 \tag{6.15}$$

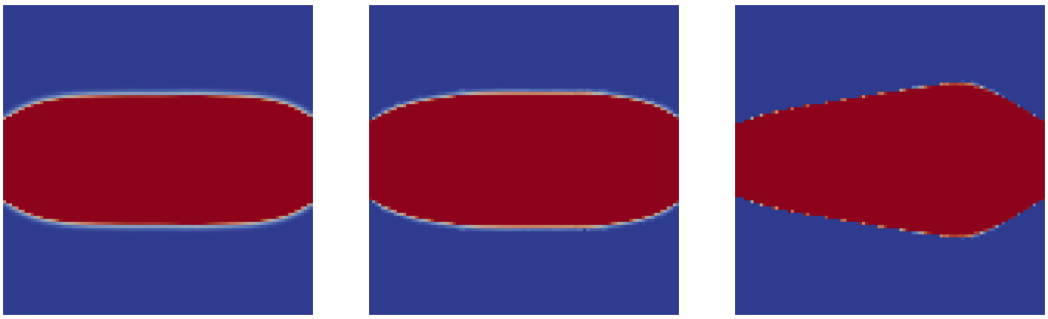
The Reynolds number based on the center line inlet velocity and the width of the inlet is $Re = 20$. Therefore the Peclet number for the problem is,

$$Pe = Re \times Pr = 20 \times 10 = 200 \tag{6.16}$$

The walls are kept at $T_H = 100$ K, while the incoming fluid is at $T_C = 0$ K.

For the volume of domain $\mathscr{V}_0$ and volume occupied the fluid $\mathscr{V}_f$, a volume fraction constraint for the fluid is stated as an inequality for all the cases, i.e,

$$\frac{\mathscr{V}_f}{\mathscr{V}_0} = \varepsilon \leq 0.4 \tag{6.17}$$

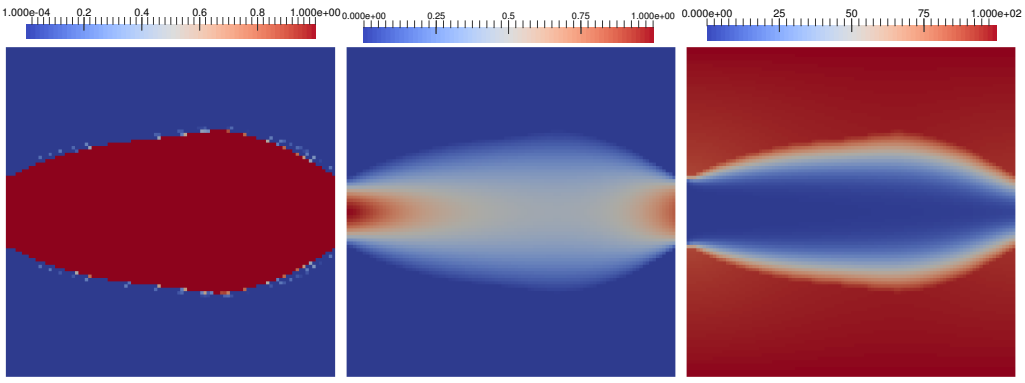


(a) (b) (c)

Figure 6.7: Topologies for test case obtained by solely minimizing pressure drop i.e. $\gamma = 0$ for various Reynolds numbers (a) $Re = 2$ (b) $Re = 20$ (c) $Re = 100$

For comparison with later results, we first present the topologies obtained by solely minimizing the pressure drop. The thermal cost function does not enter the total cost function and hence $\gamma = 0$ in Eq. 6.14. Figure 6.7 illustrates the three topologies for three different Reynolds numbers $Re = 2, 20$ and 100. For $Re = 2$, the geometry is front-to-back symmetric while the topology for $Re = 100$ gets skewed towards the right. This shape can be rationalized as being due to the higher inertia of the incoming fluid in Figure 6.7 (c).

Next we perform topology optimization with both the cost functions, by choosing four different values for scale variable $\gamma = 1, 2.5, 5$ and 10. The cases for the different values of γ are termed Case 1, Case 2, Case 3 and Case 4 respectively. As proof of concept, we choose arbitrary scales with an interpretation that higher values of γ would give more preference to thermal cost function. However the two cost functions have different units and have



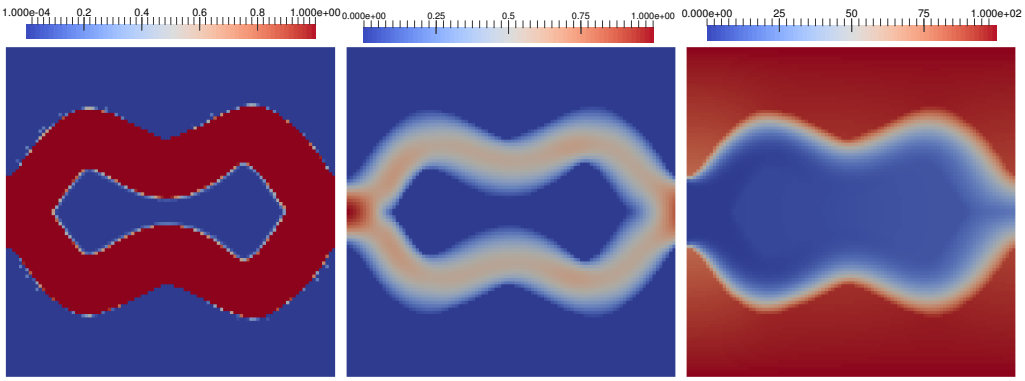
(a) (b) (c)

Figure 6.8: Case 1: Final topology obtained obtained by minimizing the multi-objective cost function for $Re = 20$. Here the scale factor $\gamma = 1$. (a) Final topology, (b) velocity distribution for the final topology, and (c) corresponding temperature distribution.

to be scaled by proper non-dimensionalization with the characteristic values of the QoIs of the problem in order to extend the formulation to real-world applications.

Figure 6.8 shows the final topology and velocity and temperature distribution for for $\gamma = 1$ and $Re = 20$ and $Pe = 200$. Contrast the difference in topologies by comparing Figure 6.8(a) with 6.7(b) for the same Reynolds number. Inclusion of the thermal cost function has skewed the topology to the right. Such a skewing is a result of the thermal inertia present at $Pe = 200$, causing the area expansion to occur further downstream then if the Peclet number were lower.

Figures 6.9, 6.10 and 6.11 show the corresponding figures for for $\gamma = 2.5, 5$ and 10. Note that the cost function becomes more thermally dominated

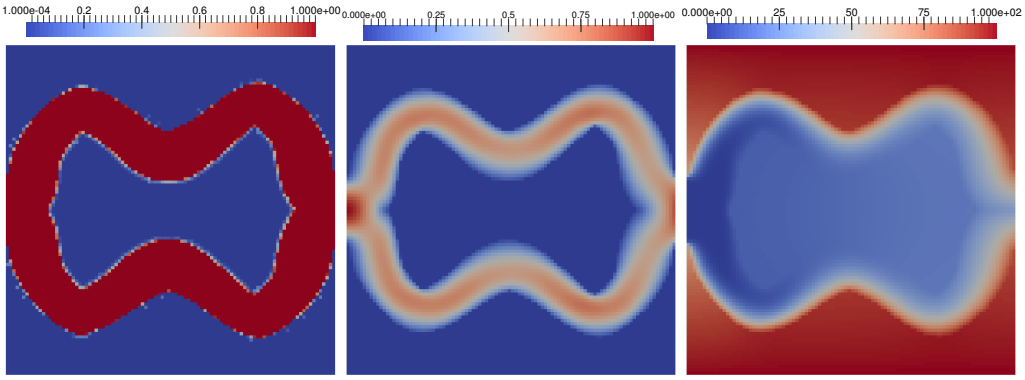


(a) (b) (c)

Figure 6.9: Case 2: The scale factor here is $\gamma = 2.5$. (a) Final topology, (b) velocity distribution for the final topology, and (c) corresponding temperature distribution.

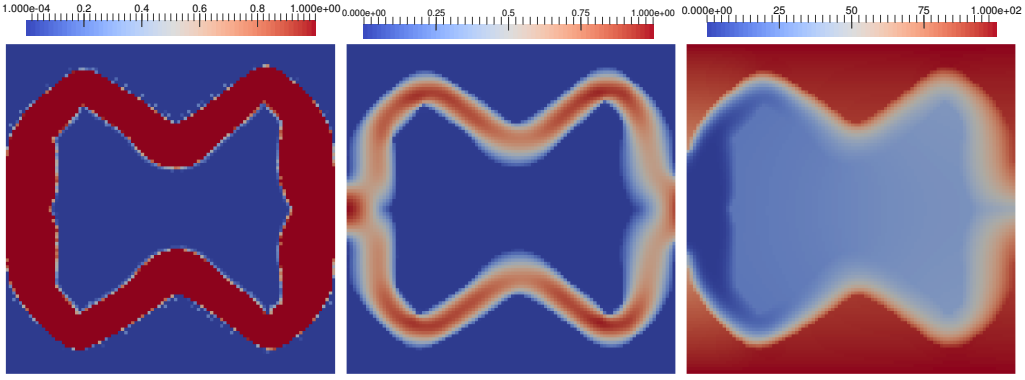
for higher values of γ. In the topologies that are generated, a solid region is deposited in the middle of the domain, forcing the fluid to take convoluted long-residence-time path close to the hot boundary. This facilitates higher surface area of contact with the solid regions connected to the hot walls. The contact surface area with the lower and upper solid regions increase with increasing values for γ. The fact that the outgoing fluid attains higher temperatures with increasing γ can be seen visually in the temperature plots.

Table 6.1, shows the data more quantitatively, with normalized individual cost functions for the final topologies for various scale factors. The pressure drop is normalized by the a characteristic pressure scale which is chosen to be the dynamic pressure at the inlet,



(a) (b) (c)

Figure 6.10: Case 2: The scale factor here is $\gamma = 5$. (a) Final topology, (b) velocity distribution for the final topology, and (c) corresponding temperature distribution.



(a) (b) (c)

Figure 6.11: Case 2: The scale factor here is $\gamma = 10$. (a) Final topology, (b) velocity distribution for the final topology, and (c) corresponding temperature distribution.

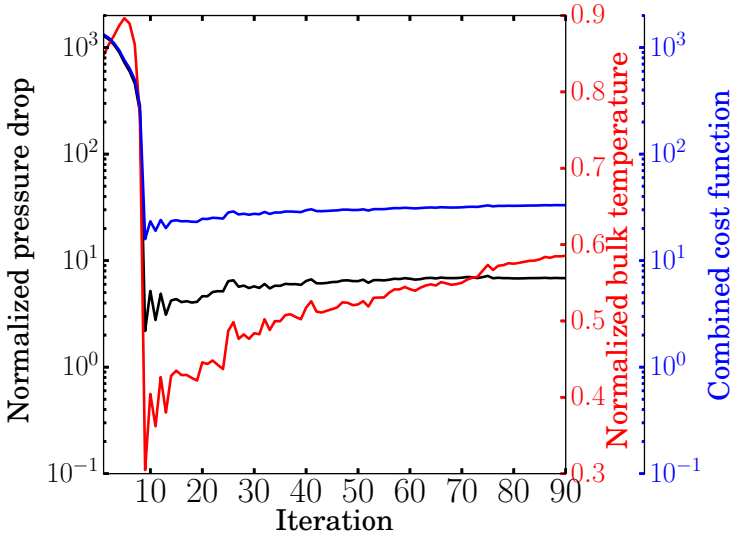
Table 6.1: Normalized cost functions, pressure drop and bulk temperature rise, for the various cases considered in Section 6.4. For all cases, $Re = 20$, $Pr = 10$ and therefore $Pe = 200$.

Case #	γ	Normalized pressure drop	Normalized bulk temperature rise
Case 1	1	1.6211	0.2190
Case 2	2.5	8.5177	0.3661
Case 3	5	18.3236	0.4820
Case 4	10	34.3108	0.5854

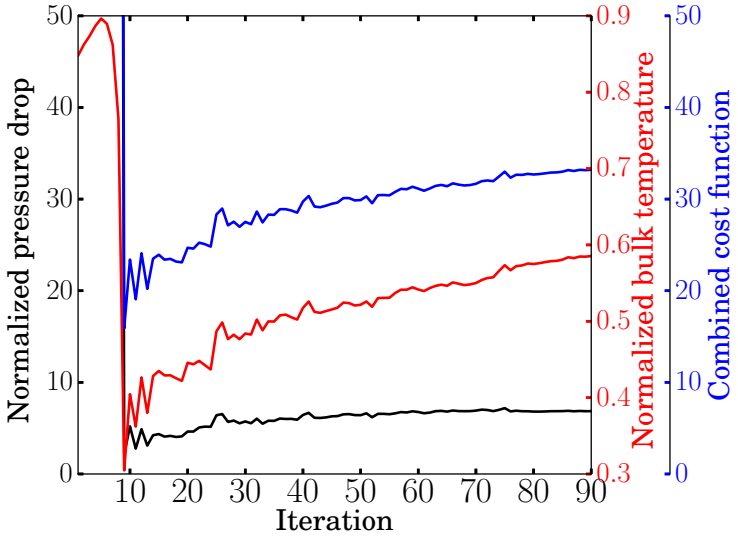
$$c^* = \frac{1}{2}\rho\left(\overline{V}_{in}\right)^2 A_{in} \tag{6.18}$$

The bulk temperature rise ΔT_b is normalized with the difference $T_H - T_C$. We observe that ΔT_b increases significantly by choosing larger scale values, but at the expense of increasing increasing pressure drop.

Finally, in Figure 6.12, we plot the evolution of the flow cost function, the thermal cost function and the total cost function for the Case 4, where $\gamma = 10$. As the last part of the section, we plot two figures as shown in Figure 6.12, depicting the evolution of the flow cost function, thermal cost function and the total cost function. The pressure drop and the mean temperature are normalized as discussed above. The total cost function is normalized with the characteristic pressure scale as in Eq. 6.18. Figure 6.12(a), shows the evolution of the cost functions with the complete range of their values. Figure 6.12(b) is selectively zoomed for enhanced viewing by plotting the pressure drop and combined cost function in linear scale as opposed to log scale in Figure 6.12(a).



(a)



(b)

Figure 6.12: Evolution of flow and thermal cost functions for $\gamma = 10$.

The optimizer performs optimization solely based on the combined cost function. In the process, one can observe that final value of pressure drop is attained relatively early in the optimization stages. The optimizer tries to enhance thermal cost function at a slower pace but spread across widely. These trends in the individual cost functions gets reflected in the combined cost function.

6.5 Challenges with Multi-Objective topology optimization

Multi-objective topology optimization poses unique challenges not encountered in previous chapters. Consider, for example, the test case in Section 4.6.1.1, Chapter 4, but with a combined cost function as in Eq. 6.14. The top and bottoms walls are maintained at a higher temperature T_H while the incoming fluid is set to a cold temperature T_C. The Reynolds number and Prandtl numbers are 10 and 100 respectively. The problem is therefore convective driven due to the high Peclet number, $Pe = 1000$. We choose a particular value of $\gamma = 200$ (cost function is almost entirely dependent on the temperature increase) and contrast the evolution of topology with the case for $\gamma = 0$ (cost function is purely dependent on pressure drop). Recall that the liquid volume fraction is constrained to be less than or equal to 0.5; i.e., the solid fraction is constrained to be greater than or equal to 0.5.

Figure 6.13(a) shows the topology for minimizing the pressure drop solely, i.e., for $\gamma = 0$. A unique stable solution is reliably obtained, and

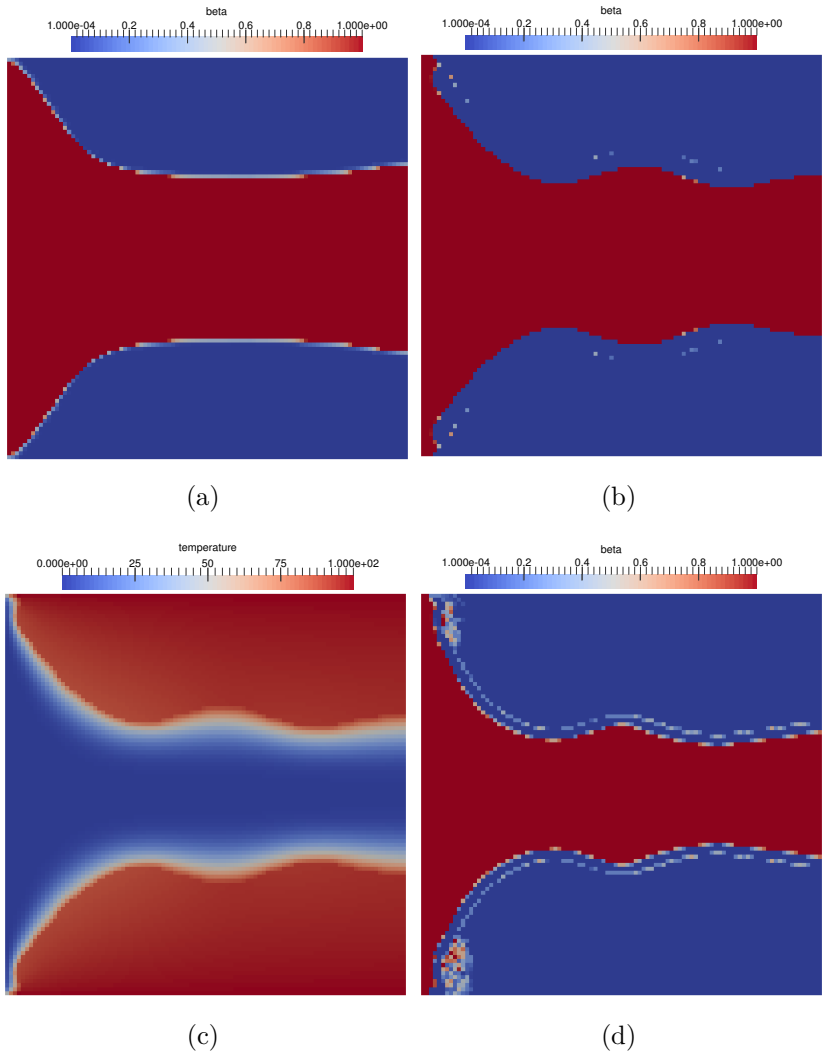


Figure 6.13: A test case to highlight the challenges of multi-objective optimization problem in realizing an active volume constraint.

the solid fraction at convergence is found to be exactly 0.5, i.e., the volume constraint is found to be satisfied in the equality. This is as expected: filling the domain with more than the minimum amount of solid would make it harder to push the fluid through the domain because of the reduction in cross-sectional area. Thus the optimizer correctly picks the minimum solid volume as the final converged value.

Consider now the case $\gamma = 200$. Here the cost function is determined almost entirely by the bulk temperature difference between the inlet and the outlet. (Only the outlet fluid temperature appears in the cost function because of the definition in Eq. 6.6). Here, increasing the amount of solid in the domain decreases the area of cross section of the fluid channel, increasing the fluid velocity and the convective heat transfer rate from the solid to the fluid, and causing the the fluid to heat up more. Thus, the optimizer moves in the direction of filling up the domain with solid. The volume fraction constraint can no longer be met in the equality. The solid volume increases without bound and a stable convergent solution cannot be found. Figure 6.13(d) shows an intermediate topology in the evolution process. As is evident, the fluid channel has become narrower, and the process will continue indefinitely until the domain is filled with solid.

In all the problems solved in this dissertation, we have posed the volume fraction constraint as an inequality constraint. The problem encountered here could be circumvented by posing the constraint as an equality constraint. For more complex problems, care must be taken in posing the problem correctly.

In topology optimization, a guiding principle is that "you get what you ask for." Posing the right problem is central to obtaining the right solution.

6.6 Closure

In this chapter we demonstrated the topology optimization methodology for forced convection. We saw that consideration of forced convection naturally introduces a multi-objective cost function into the problem formulation. Topology optimization for forced convection problems is more complex due to the multi-objective nature of the cost function. Unless care is taken with the problem statement, unexpected (though mathematically defensible) solutions may be obtained. Furthermore, we have taken a relatively simple approach to multi-objective optimization, using a simple linear weighted sum of objective functions to drive the problem. More sophisticated methods are available, though these have not, to our knowledge, been explored in the context of topology optimization [138]. This represents a rich area for future work, and is essential if industrially-relevant problems are to be solved using topology optimization.

Chapter 7

Summary and future directions

In this chapter, we briefly summarize the major contributions of this dissertation. We also outline possible future work, classified into short term and long term extensions. In the process, we identify the major areas of research that need to be done for taking the field of topology optimization for flow and transport applications to the next level.

7.1 Major contributions of the dissertation

The objective of this dissertation has been to develop topology optimization procedures for thermal-fluid problems in the framework of unstructured finite volume methods. The topology optimization employed here is based on the SIMP formulation in conjunction with a gradient based optimization algorithm. A central contribution of the dissertation has been the development of methodologies for obtaining discrete adjoint sensitivities for the finite volume scheme employing co-located pressure-velocity storage and a sequential solution algorithm. A novel Automatic Differentiation library was built to enable the computation of sensitivities in a problem-agnostic way. After developing and demonstrating the topology optimization procedures for

heat conduction applications, we systematically developed the procedures for fluid flow applications in the laminar regime. Building on laminar flow procedures, the methodology was expanded to turbulent flow applications modelled using the Spalart-Allmaras RANS turbulent model. Finally topology optimization was developed for forced convection applications for both laminar and turbulent flows. We demonstrated the evolution of optimal topologies for flow and heat transfer applications with a variety of realistic cost function for both laminar and turbulent flows. We believe this is the first time that a complete SIMP based topology optimization framework using an unstructured finite volume method, fully generalizable to practical use in commercial solvers and for industrial applications, has been demonstrated.

7.2 Short term extensions

We first identify extensions that can be implemented easily within a short period of time based on the work carried out in this dissertation.

7.2.1 Parallelization of $\mathcal{R}$apid library and adjoint linear system

Topology optimization algorithms should be able to run in parallel for obtaining solutions to real-life industrial applications [62, 23]. Having developed methodologies in this dissertation for topology optimization using finite volume method, the natural extension would be to parallelize the infrastructure. MEMOSA is a finite volume software suite inherently written for parallel architectures using domain decomposition and therefore the forward solution

part of topology optimization is already taken care of. The $\mathcal{R}$apid library has to be extended for parallelization with domain decomposition.

There are a series of independent state variables and a design variable associated with every finite volume cell in $\mathcal{R}$apid mode. All these variables have to be uniquely identified with certain numbering scheme as discussed in Sections 3.4/4.5.2, Chapters 3/4 for the residuals to be computed to obtain derivatives. In domain decomposition, the mesh is decomposed and distributed to the the various processors. It is noted that the process of computing residuals in $\mathcal{R}$apid mode is 'embarrassingly parallel', since the converged values of variables are already available and residual computation of one cell does not depend on any other. It is necessary to devise a proper numbering schemes for the variables to be assigned to each finite volume cell of every decomposed domain, so that the construct for computing the derivatives using MAP-STL works as desired. Some methods in the $\mathcal{R}$apid class might need modifications to realize parallelization.

The adjoint linear system underlying topology optimization is very stiff and sometimes requires direct solvers to solve. There are many powerful parallel sparse direct solvers available in the open source [96, 139]. MEMOSA can be used in tandem with these open source libraries for parallelization of the topology optimization methodology using domain decomposition methods.

7.2.2 Extension to other physical models

We demonstrated the topology optimization procedure for the flow and energy equation. The procedures can easily be extended to various models, say for example, the species transport equations and chemical reactions incorporating various chemistry models. Other possible applications include the design of mixers, chemical reactors, 3D batteries etc. The methodology can also be extended to sub-micron transport models using phonon-BTE transport models, to design particulate composites and nano-structures for engineering the desired thermal and electrical properties. Coupling of various physical models can also be accomplished by following the methodology presented for coupling flow and energy equations in the dissertation.

The advent of 3D printing is a particular opportunity for topology optimization. Indeed, one may view it as an essential design tool for this emerging technology. Development of a user-friendly CAD system based on topology optimization, and the ultimate translation of 3D printed designs to a laser rastering scheme for, for example, selective laser sintering [140, 30] is an intriguing possibility. Coupling of various physical models can also be accomplished by following the methodology presented for coupling flow and energy equations in the dissertation.

7.2.3 Quantities of Interest

Many new interesting problems could be solved within the existing infrastructure. We have explored only a few QoIs for the various applications

described in the dissertation. For instance, the only QoI used for fluid flow application was the the pressure drop across the boundaries. Also it is noted that these QoIs were based on boundary values of pressure and temperature. Various other boundary and volume–related QoIs can be explored to solve new problems effortlessly. $\mathcal{R}$apid facilitates computation of the sensitivities of any newly specified cost functions in a problem-agnostic way. This powerful functionality must be explored further for realistic industrial problems.

7.2.4 Modifying boundary conditions and initial design domain space

It has to be understood that the designer cannot use topology optimization to come up with the best geometrical designs in the first attempt for every new problem that is specified. The process is iterative in nature, where the user can try various the boundary conditions for the problem, or modify the geometries of initial domain space or change constraints of the problem, change the various parameters of the problem etc., based on previous trials. One has to understand that the process is also an art. In optimization, you only get what you ask for, problem definition is key to obtaining useful designs.

7.3 Future directions

We now briefly discuss some of the non-trivial extensions that might need significant research to make noteworthy strides.

7.3.1 Multi-objective cost functions and constraints

We demonstrated topology optimization for forced convection applications that involved multi-objective cost functions in Chapter 6. As mentioned, there are various issues associated with multi-objective cost functions in the current formulation. The major one is the inability of the currently implemented optimizer to satisfy the volume constraint in the equality. Either new paths to solution must be devised so that the constraint is active, or the optimizer must be modified to accommodate equality constraints. Correct scaling of competing cost functions is also an issue that must be addressed. The literature is rich in optimization techniques for multi-objective optimization, and it would be interesting to see if these methods could be married to topology optimization to address more complex industrially-relevant problems.

In this dissertation, we have explored relatively simple constraints, typically only on the total volume fraction. For topology optimization to be truly useful to the industrial practitioner, it would be important to address more complex constraints, for example on manufacturability. Constraints on material connectivity (to prevent the formation of holes) or on feature size would be some examples.

7.3.2 Filtering

Filtering of either sensitivity variables or design variables is critical in topology optimization, especially in the early part of the process, in order to avoid getting trapped at local minima (Chapter 2, Section 2.5.6). The process

is also necessary to mitigate mesh dependence. There has been substantial on-going research in the area of filtering and design for manufacturability. However the majority of the papers have focused on structural applications [65, 141, 142]. Filtering of sensitivities or design variables for flow problems must be explored, since flow problems have fundamental differences when compared to structural mechanics.

7.3.3 Discrete and continuous adjoint methods and level sets

As mentioned previously, the vast majority of the work in the literature, including this work, has been done using the discrete adjoint method. Here, sensitivities are computed based on the adjoint formed from the discretized PDE. Recently, some interesting results have been published on topology optimization for flow and heat transfer problems using continuous adjoint methods [109, 123, 137, 125]. In the continuous adjoint method, the adjoint PDE is formulated from the original PDE, and then discretized and solved. Evgrafov *et al.* present a mathematical analysis of the convergence of finite volume schemes for design problems and in the process compare the discrete and adjoint approaches [143].

A few papers have been also published on topology optimization using level set methods. This methodology may allow the final optimal geometry to be defined with smooth boundaries rather than with a stair-step geometry. Of course, this is also easily achieved *posteriori* by identifying a bounding contour surface based on the stair-stepped geometry.

The authors cite different advantages for the the methods they present. However a more quantitative comparative study between discrete and continuous adjoint methods and level set methods will help to move the field of topology optimization forward.

7.3.4 Acceleration of adjoint linear system

It is the solution of adjoint system used for computing sensitivities that take the most amount of time in topology optimization. Because of significant research in the last several decades, the forward solution of the governing PDEs is quite efficient both in serial and parallel. Similar research for accelerating the solution of the adjoint linear system, especially for the systems arising in topology optimization, would be useful to move the field forward.

7.3.5 Amalgamation of topology and shape optimization

We started Chapter 1 with an example illustrating the use of topology optimization for a design work-flow in the industry. For convenience the figure is reproduced here as Figure 7.1. It is interesting to see how the designer uses the techniques of topology optimization and shape optimization in tandem to come up with an optimally stiff structure with minimum weight.

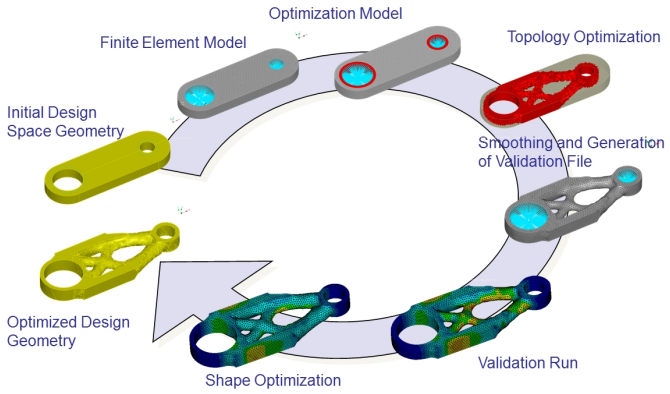


Figure 7.1: Illustration of topology and shape optimization being used in industry in a design work flow (Courtesy - TOSCA [4]).

Topology optimization is best at realizing initial conceptual designs in an initial arbitrary design space. Once an initial geometry is conceptualized from topology optimization, shape optimization can be used to optimize these template shapes. As mentioned in Chapter 1, shape optimization for fluid applications is a well developed field [6, 7, 8]. Thus, following in the footsteps of optimization for structural applications described in Figure 7.1, implementation of such a combined methodology for flow and thermal applications may pay significant dividends as well.

7.4 Closure

We thus come to the end of this dissertation. We believe that this work will help motivate the thermal fluids community to not only use topology optimization for their various applications, but also will propel future research and move the field forward. The avenues are infinite!

Bibliography

[1] http://altairenlighten.com/in-depth/shape-optimization/.

[2] Jens Howard Peter Buckley. *Numerical Shape Optimization of Airfoils With Practical Aerodynamic Design Requirements*. PhD thesis, University of Toronto, 2009.

[3] Ole Sigmund. A 99 line topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 21(2):120–127, 2001.

[4] http://www.3ds.com/products-services/simulia/products/tosca/.

[5] Gil Ho Yoon. Topology optimization for stationary fluid–structure interaction problems using a new monolithic formulation. *International journal for numerical methods in engineering*, 82(5):591–616, 2010.

[6] Jan Sokolowski and Jean-Paul Zolesio. *Introduction to Shape Optimization*. Springer, 1992.

[7] Olivier Pironneau. On optimum design in fluid mechanics. *Journal of Fluid Mechanics*, 64(01):97–110, 1974.

[8] Jeff Borggaard and John Burns. A pde sensitivity equation method for optimal aerodynamic design. *Journal of Computational Physics*, 136(2):366–384, 1997.

[9] Martin Philip Bendsoe and Ole Sigmund. *Topology Optimization: Theory, Methods and Applications*. Springer, 2003.

[10] Martin P Bendsøe and Ole Sigmund. Material interpolation schemes in topology optimization. *Archive of applied mechanics*, 69(9-10):635–654, 1999.

[11] Hans A Eschenauer and Niels Olhoff. Topology optimization of continuum structures: A review*. *Applied Mechanics Reviews*, 54(4):331–390, 2001.

[12] Claus BW Pedersen and Peter Allinger. Industrial implementation and applications of topology optimization and future needs. In *IUTAM Symposium on Topological Design Optimization of Structures, Machines and Materials*, pages 229–238. Springer, 2006.

[13] George IN Rozvany. A critical review of established methods of structural topology optimization. *Structural and Multidisciplinary Optimization*, 37(3):217–237, 2009.

[14] Andrew T Gaynor, Nicholas A Meisel, Christopher B Williams, and James K Guest. Multiple-material topology optimization of compliant mechanisms created via polyjet three-dimensional printing. *Journal of Manufacturing Science and Engineering*, 136(6):061015, 2014.

[15] Tyler E Bruns. Topology optimization of convection-dominated, steady-state heat transfer problems. *International Journal of Heat and Mass Transfer*, 50(15):2859–2873, 2007.

[16] Gilles Marck, Maroun Nemer, Jean-Luc Harion, Serge Russeil, and Daniel Bougeard. Topology optimization using the simp method for multiobjective conductive problems. *Numerical Heat Transfer, Part B: Funda-*

mentals, 61(6):439–470, 2012.

[17] Ajay Vadakkepatt, Bradley L Trembacki, Sanjay Mathur, and Jayathi Y Murthy. Topology optimization for heat conduction applications. In *ASME 2014 International Mechanical Engineering Congress and Exposition*, pages V08BT10A035–V08BT10A035. American Society of Mechanical Engineers, 2014.

[18] James Madigan Loy. An efficient solution procedure for simulating phonon transport in multiscale multimaterial systems. 2013.

[19] Bradley L Trembacki, Jayathi Y Murthy, and Scott Alan Roberts. Fully coupled simulation of lithium ion battery cell performance. Technical report, Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), 2015.

[20] Matthew Roberts, Phil Johns, John Owen, Daniel Brandell, Kristina Edstrom, Gaber El Enany, Claude Guery, Diana Golodnitsky, Matt Lacey, Cyrille Lecoeur, et al. 3d lithium ion batteriesâĂŤfrom fundamentals to fabrication. *Journal of Materials Chemistry*, 21(27):9876–9890, 2011.

[21] Thomas Borrvall and Joakim Petersson. Topology optimization of fluids in stokes flow. *International journal for numerical methods in fluids*, 41(1):77–107, 2003.

[22] Allan Gersborg-Hansen, Ole Sigmund, and Robert B Haber. Topology optimization of channel flow problems. *Structural and Multidisciplinary Optimization*, 30(3):181–192, 2005.

[23] Joe Alexandersen, Ole Sigmund, and Niels Aage. Large scale three-

dimensional topology optimisation of heat sinks cooled by natural convection. *International Journal of Heat and Mass Transfer*, 100:876–891, 2016.

[24] Aliaa I Shallan, Petr Smejkal, Monika Corban, Rosanne M Guijt, and Michael C Breadmore. Cost effective 3d-printing of visibly transparent microchips within minutes. *Analytical chemistry*, 2014.

[25] Gwo-Bin Lee, Bao-Herng Hwei, and Guan-Ruey Huang. Micromachined pre-focused m× n flow switches for continuous multi-sample injection. *Journal of Micromechanics and Microengineering*, 11(6):654, 2001.

[26] Robert DK Templeton and William W Arnott. Fluid flow switches with low flow resistance, May 28 1991. US Patent 5,019,678.

[27] Marc Bessler and Timothy AM Chuter. Artificial heart valve and method and device for implanting the same, January 5 1999. US Patent 5,855,601.

[28] Karthik K Bodla, Jayathi Y Murthy, and Suresh V Garimella. Direct simulation of thermal transport through sintered wick microstructures. *Journal of heat transfer*, 134(1):012602, 2012.

[29] Casper Schousboe Andreasen, Allan Roulund Gersborg, and Ole Sigmund. Topology optimization of microfluidic mixers. *International Journal for Numerical Methods in Fluids*, 61(5):498–513, 2009.

[30] Carolyn Conner Seepersad. Challenges and opportunities in design for additive manufacturing. *3D Printing and Additive Manufacturing*, 1(1):10–13, 2014.

[31] SR Mathur and JY Murthy. A pressure-based method for unstructured meshes. *Numerical Heat Transfer*, 31(2):195–215, 1997.

[32] Suhas Patankar. *Numerical heat transfer and fluid flow*. CRC Press, 1980.

[33] Gilles Marck, Maroun Nemer, and Jean-Luc Harion. Topology optimization of heat and mass transfer problems: laminar flow. *Numerical Heat Transfer, Part B: Fundamentals*, 63(6):508–539, 2013.

[34] Gil Ho Yoon. Topology optimization for turbulent flow with spalart–allmaras model. *Computer Methods in Applied Mechanics and Engineering*, 303:288–311, 2016.

[35] http://www.purdue.edu/discoverypark/prism/.

[36] James Ahrens, Berk Geveci, Charles Law, CD Hansen, and CR Johnson. 36-paraview: An end-user tool for large-data visualization, 2005.

[37] Washington Bellevue. Tecplot user's manual. *Amtec Engineering Inc*, 2003.

[38] Allan Gersborg-Hansen, Martin P Bendsøe, and Ole Sigmund. Topology optimization of heat conduction problems using the finite volume method. *Structural and multidisciplinary optimization*, 31(4):251–259, 2006.

[39] T Gao, WH Zhang, JH Zhu, YJ Xu, and DH Bassir. Topology optimization of heat conduction problem involving design-dependent heat load effect. *Finite Elements in Analysis and Design*, 44(14):805–813, 2008.

[40] Jaco Dirker and Josua P Meyer. Topology optimization for an internal heat-conduction cooling scheme in a square domain for high heat flux applications. *Journal of Heat Transfer*, 135(11):111010, 2013.

[41] Kurt Maute. Topology optimization of diffusive transport problems. In *Topology Optimization in Structural and Continuum Mechanics*, pages 389–407. Springer, 2014.

[42] A Iga, S Nishiwaki, K Izui, and M Yoshimura. Topology optimization for thermal conductors considering design-dependent effects, including heat conduction and convection. *International Journal of Heat and Mass Transfer*, 52(11):2721–2732, 2009.

[43] Mingdong Zhou, Joe Alexandersen, Ole Sigmund, and Claus BW Pedersen. Industrial application of topology optimization for combined conductive and convective heat transfer problems. *Structural and Multidisciplinary Optimization*, pages 1–16, 2016.

[44] Tomás Zegard and Glaucio H Paulino. Toward gpu accelerated topology optimization on unstructured meshes. *Structural and Multidisciplinary Optimization*, 48(3):473–485, 2013.

[45] Niels Aage, Thomas H Poulsen, Allan Gersborg-Hansen, and Ole Sigmund. Topology optimization of large scale stokes flow problems. *Structural and Multidisciplinary Optimization*, 35(2):175–180, 2008.

[46] Jean-Christophe Cuillière, Vincent Francois, and Jean-Marc Drouet. Towards the integration of topology optimization into the cad process. *Computer-Aided Design and Applications*, 11(2):120–140, 2014.

[47] Cameron Talischi, Glaucio H Paulino, Anderson Pereira, and Ivan FM Menezes. Polytop: a matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. *Structural and Multidisciplinary Optimization*, 45(3):329–357, 2012.

[48] Jayathi Y Murthy and SR Mathur. Numerical methods in heat, mass, and momentum transfer. *School of Mechanical Engineering Purdue University*, 2002.

[49] JY Murthy and SR Mathur. Computation of anisotropic conduction using unstructured meshes. *Journal of heat Transfer*, 120(3):583–591, 1998.

[50] https://github.com/c-primed/fvm.git.

[51] Sanjay R Mathur and Jayathi Y Murthy. A multigrid method for the poisson–nernst–planck equations. *International Journal of Heat and Mass Transfer*, 52(17):4031–4039, 2009.

[52] Martin Philip Bendsøe and Noboru Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computer methods in applied mechanics and engineering*, 71(2):197–224, 1988.

[53] Martin Philip Bendsøe. Optimal shape design as a material distribution problem. *Structural optimization*, 1(4):193–202, 1989.

[54] Joe Alexandersen, Niels Aage, Casper Schousboe Andreasen, and Ole Sigmund. Topology optimisation for natural convection problems. *International Journal for Numerical Methods in Fluids*, 76(10):699–721, 2014.

[55] Anton Evgrafov, Kurt Maute, RG Yang, and Martin L Dunn. Topology optimization for nano-scale heat transfer. *International journal for numerical methods in engineering*, 77(2):285–300, 2009.

[56] Antony Jameson. Aerodynamic shape optimization using the adjoint method. *Lectures at the Von Karman Institute, Brussels*, 2003.

[57] Ole Sigmund. On the design of compliant mechanisms using topology optimization*. *Journal of Structural Mechanics*, 25(4):493–524, 1997.

[58] Qing Li, Grant P Steven, YM Xie, and Osvaldo M Querin. Evolutionary topology optimization for temperature reduction of heat conducting fields. *International Journal of Heat and Mass Transfer*, 47(23):5071–5083, 2004.

[59] Zhihao Zuo. Topology optimization of periodic structures. 2009.

[60] Krister Svanberg. The method of moving asymptotes-a new method for structural optimization. *International journal for numerical methods in engineering*, 24(2):359–373, 1987.

[61] Krister Svanberg. Mma and gcmma. *www.math.kth.se/ krille/gcmma07.pdf*, 2007.

[62] Niels Aage and Boyan S Lazarov. Parallel framework for topology optimization using the method of moving asymptotes. *Structural and Multidisciplinary Optimization*, 47(4):493–505, 2013.

[63] Krister Svanberg. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM Journal on Optimization*, 12(2):555–573, 2002.

[64] Erik Andreassen, Anders Clausen, Mattias Schevenels, Boyan S Lazarov, and Ole Sigmund. Efficient topology optimization in matlab using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, 2011.

[65] Weisheng Zhang, Wenliang Zhong, and Xu Guo. An explicit length scale control approach in simp-based topology optimization. *Computer Methods in Applied Mechanics and Engineering*, 282:71–86, 2014.

[66] Fengwen Wang, Boyan Stefanov Lazarov, and Ole Sigmund. On projection methods, convergence and robust formulations in topology optimization. *Structural and Multidisciplinary Optimization*, 43(6):767–784, 2011.

[67] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Siam, 2008.

[68] Uwe Naumann. *The art of differentiating computer programs: an introduction to algorithmic differentiation*, volume 24. Siam, 2012.

[69] Christian Bischof, Alan Carle, George Corliss, Andreas Griewank, and Paul Hovland. Adifor–generating derivative codes from fortran programs. *Scientific Programming*, 1(1):11–29, 1992.

[70] Ralf Giering and Thomas Kaminski. Applying taf to generate efficient derivative code of fortran 77-95 programs. *PAMM*, 2(1):54–57, 2003.

[71] Andrea Walther, Andreas Griewank, and Olaf Vogel. Adol-c: Automatic differentiation using operator overloading in c++. *PAMM*, 2(1):41–44, 2003.

[72] Robin J Hogan. Fast reverse-mode automatic differentiation using expression templates in c+ &plus. *ACM Transactions on Mathematical Software (TOMS)*, 40(4):26, 2014.

[73] Claus Bendtsen and Ole Stauning. Fadbad, a flexible c++ package for automatic differentiation. *Department of Mathematical Modelling, Technical University of Denmark*, 1996.

[74] ET Phipps and DM Gay. Sacado automatic differentiation package, 2011.

[75] Eric T Phipps, Roscoe A Bartlett, David M Gay, and Robert J Hoekstra. Large-scale transient sensitivity analysis of a radiation-damaged bipolar junction transistor via automatic differentiation. In *Advances in automatic differentiation*, pages 351–362. Springer, 2008.

[76] www.coin-or.org/cppad/.

[77] Bjarne Stroustrup. *The C++ programming language*. Pearson Education, 2013.

[78] Andreas Griewank. On automatic differentiation and algorithmic linearization. *Pesquisa Operacional*, 34(3):621–645, 2014.

[79] Todd Veldhuizen. Expression templates. *C++ Report*, 7(5):26–31, 1995.

[80] Pierre Aubert, Nicolas Di Césaré, and Olivier Pironneau. Automatic differentiation in c++ using expression templates and. application to a flow control problem. *Computing and Visualization in Science*, 3(4):197–208, 2001.

[81] Eric Phipps and Roger Pawlowski. Efficient expression templates for operator overloading-based automatic differentiation. In *Recent Advances in Algorithmic Differentiation*, pages 309–319. Springer, 2012.

[82] Markus Towara and Uwe Naumann. A discrete adjoint model for openfoam. *Procedia Computer Science*, 18:429–438, 2013.

[83] Thomas F Coleman and Arun Verma. The efficient computation of sparse jacobian matrices using automatic differentiation. *SIAM Journal on Scientific Computing*, 19(4):1210–1233, 1998.

[84] Anders Logg, Kent-Andre Mardal, and Garth Wells. *Automated solution of differential equations by the finite element method: The FEniCS book*, volume 84. Springer Science & Business Media, 2012.

[85] Nicolas R Gauger, Andrea Walther, Carsten Moldenhauer, and Markus Widhalm. Automatic differentiation of an entire design chain for aerodynamic shape optimization. In *New Results in Numerical and Experimental Fluid Mechanics VI*, pages 454–461. Springer, 2007.

[86] Francisco Palacios, Michael R Colonno, Aniket C Aranake, Alejandro Campos, Sean R Copeland, Thomas D Economon, Amrita K Lonkar, Trent W Lukaczyk, Thomas WR Taylor, and Juan J Alonso. Stanford university unstructured (su2): An open-source integrated computational environment for multi-physics simulation and design. *AIAA Paper*, 287:2013, 2013.

[87] Nicolas R Gauger, Michael Giles, Max Gunzburger, and Uwe Naumann. Adjoint methods in computational science, engineering, and finance.

2015.

[88] Henry G Weller, G Tabor, Hrvoje Jasak, and C Fureby. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in physics*, 12(6):620–631, 1998.

[89] Johannes Lotz, Klaus Leppkes, and Uwe Naumann. dco/c++-derivative code by overloading in c++. *Aachener Informatik Berichte (AIB-2011-06)*, 2011.

[90] Shankhadeep Das. Fluid-structure interactions in microstructures. 2013.

[91] Prabhakar Marepalli. *Thermal transport in low-dimensional materials*. PhD thesis, 2015.

[92] Nicolas Di Cesare. Fad: Automatic differentiation library in forward mode using expression template, 1999.

[93] Nathan C Myers. Traits: a new and useful template technique. *C++ Report*, 7(5):32–35, 1995.

[94] Michael Heroux, Roscoe Bartlett, Vicki Howle Robert Hoekstra, Jonathan Hu, Tamara Kolda, Richard Lehoucq, Kevin Long, Roger Pawlowski, Eric Phipps, Andrew Salinger, Heidi Thornquist, Ray Tuminaro, James Willenbring, and Alan Williams. An Overview of Trilinos. Technical Report SAND2003-2927, Sandia National Laboratories, 2003.

[95] http://www.cplusplus.com/reference/map/map/.

[96] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes,

Karl Rupp, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.7, Argonne National Laboratory, 2016.

[97] Suhas V Patankar and D Brian Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *International journal of heat and mass transfer*, 15(10):1787–1806, 1972.

[98] Antony Jameson. Aerodynamic design via control theory. *Journal of scientific computing*, 3(3):233–260, 1988.

[99] Omar Ghattas and Jai-Hyeong Bark. Optimal control of two-and three-dimensional incompressible navier–stokes flows. *Journal of Computational Physics*, 136(2):231–244, 1997.

[100] M Abdelwahed, M Hassine, and Mohamed Masmoudi. Optimal shape design for fluid flow using topological perturbation technique. *Journal of Mathematical Analysis and applications*, 356(2):548–563, 2009.

[101] Bijan Mohammadi and Olivier Pironneau. *Applied shape optimization for fluids*. Oxford University Press, 2010.

[102] James K Guest and Jean H Prévost. Topology optimization of creeping fluid flows using a darcy–stokes finite element. *International Journal for Numerical Methods in Engineering*, 66(3):461–484, 2006.

[103] Niclas Wiker, Anders Klarbring, and Thomas Borrvall. Topology optimization of regions of darcy and stokes flow. *International journal for numerical methods in engineering*, 69(7):1374–1404, 2007.

[104] Laurits Højgaard Olesen, Fridolin Okkels, and Henrik Bruus. A high-

level programming-language implementation of topology optimization applied to steady-state navier–stokes flow. *International Journal for Numerical Methods in Engineering*, 65(7):975–1001, 2006.

[105] Ercan M Dede. Multiphysics topology optimization of heat transfer and fluid flow systems. In *proceedings of the COMSOL Users Conference*, 2009.

[106] Yongbo Deng, Zhenyu Liu, and Yihui Wu. Topology optimization of steady and unsteady incompressible navier–stokes flows driven by body forces. *Structural and Multidisciplinary Optimization*, 47(4):555–570, 2013.

[107] Gil Ho Yoon. Topological design of heat dissipating structure with forced convective heat transfer. *Journal of Mechanical Science and Technology*, 24(6):1225–1233, 2010.

[108] Fridolin Okkels and Henrik Bruus. Scaling behavior of optimally structured catalytic microfluidic reactors. *Physical Review E*, 75(1):016301, 2007.

[109] C Othmer. A continuous adjoint formulation for the computation of topological and surface sensitivities of ducted flows. *International Journal for Numerical Methods in Fluids*, 58(8):861–877, 2008.

[110] Georg Pingen, Anton Evgrafov, and Kurt Maute. Topology optimization of flow domains using the lattice boltzmann method. *Structural and Multidisciplinary Optimization*, 34(6):507–524, 2007.

[111] Shiwei Zhou and Qing Li. A variational level set method for the topology

optimization of steady-state navier–stokes flow. *Journal of Computational Physics*, 227(24):10178–10195, 2008.

[112] Vivien J Challis and James K Guest. Level set topology optimization of fluids in stokes flow. *International journal for numerical methods in engineering*, 79(10):1284–1308, 2009.

[113] Sebastian Kreissl and Kurt Maute. Levelset based fluid topology optimization using the extended finite element method. *Structural and Multidisciplinary Optimization*, 46(3):311–326, 2012.

[114] ANSYS Fluent. 14.0 user's manual. *ANSYS Inc., Canonsburg, PA*, 2011.

[115] User Guide. Star-ccm+ version 8.04. *CD-adapco-2013*, 2013.

[116] Robert W Fox, Alan T McDonald, and Philip J Pritchard. *Introduction to fluid mechanics*, volume 7. John Wiley & Sons New York, 1985.

[117] Peter Bradshaw. *An Introduction to Turbulence and Its Measurement: Thermodynamics and Fluid Mechanics Series*. Elsevier, 2013.

[118] Robert D Moser, John Kim, and Nagi N Mansour. Direct numerical simulation of turbulent channel flow up to re= 590. *Phys. Fluids*, 11(4):943–945, 1999.

[119] Stephen B Pope. Turbulent flows, 2001.

[120] Robert D Moser. Lecture notes on turulence. 2015.

[121] Paul A Durbin and BA Pettersson Reif. *Statistical theory and modeling for turbulent flows*. John Wiley & Sons, 2011.

[122] Philipe R Spalart and Stephen R Allmaras. A one equation turbulence model for aerodinamic flows. *AIAA journal*, 94, 1992.

[123] EM Papoutsis-Kiachagias, EA Kontoleontos, AS Zymaris, DI Papadimitriou, and KC Giannakoglou. Constrained topology optimization for laminar and turbulent flows, including heat transfer. *CIRA, editor, EUROGEN, Evolutionary and Deterministic Methods for Design, Optimization and Control, Capua, Italy*, 2011.

[124] E. A. Kontoleontos, E. M. Papoutsis-Kiachagias, A. S. Zymaris, D. I. Papadimitriou, and K. C. Giannakoglou. Adjoint-based constrained topology optimization for viscous flows, including heat transfer. *Engineering Optimization*, 45(8):941–961, 2013.

[125] EM Papoutsis-Kiachagias and KC Giannakoglou. Continuous adjoint methods for turbulent flows, applied to shape and topology optimization: industrial applications. *Archives of Computational Methods in Engineering*, 23(2):255–299, 2016.

[126] http://turbmodels.larc.nasa.gov/spalart.html.

[127] Paul G Tucker, Chris L Rumsey, Philippe R Spalart, Robert B Bartels, and Robert T Biedron. Computations of wall distances based on differential equations. *AIAA journal*, 43(3):539–549, 2005.

[128] DB Spalding. Calculation of turbulent heat transfer in cluttered spaces. In *Proc. 10^{th} Int. Heat Transfer Conf*, 1994.

[129] DB Spalding. A single formula for the "law of the wall". *Journal of Applied Mechanics*, 28(3):455–458, 1961.

[130] Theodore L Bergman, Frank P Incropera, David P DeWitt, and Adrienne S Lavine. *Fundamentals of heat and mass transfer*. John Wiley & Sons, 2011.

[131] Ole Sigmund. Design of multiphysics actuators using topology optimization–part i: One-material structures. *Computer methods in applied mechanics and engineering*, 190(49):6577–6604, 2001.

[132] Gil Ho Yoon and Yoon Young Kim. The element connectivity parameterization formulation for the topology design optimization of multiphysics systems. *International Journal for Numerical Methods in Engineering*, 64(12):1649–1677, 2005.

[133] Luzhong Yin and GK Ananthasuresh. A novel topology design scheme for the multi-physics problems of electro-thermally actuated compliant micromechanisms. *Sensors and Actuators A: Physical*, 97:599–609, 2002.

[134] Kyungjun Lee. *Topology Optimization of Convective Cooling System Designs*. PhD thesis, The University of Michigan, 2012.

[135] Adriano A Koga, Edson Comini C Lopes, Helcio F Villa Nova, Cícero R de Lima, and Emílio Carlos Nelli Silva. Development of heat sink device by using topology optimization. *International Journal of Heat and Mass Transfer*, 64:759–772, 2013.

[136] Tadayoshi Matsumori, Tsuguo Kondoh, Atsushi Kawamoto, and Tsuyoshi Nomura. Topology optimization for fluid–thermal interaction problems under constant input power. *Structural and Multidisciplinary Optimiza-*

tion, 47(4):571–581, 2013.

[137] EA Kontoleontos, EM Papoutsis-Kiachagias, AS Zymaris, DI Papadimitriou, and KC Giannakoglou. Adjoint-based constrained topology optimization for viscous flows, including heat transfer. *Engineering Optimization*, 45(8):941–961, 2013.

[138] Kaisa Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media, 2012.

[139] Patrick R Amestoy, Iain S Duff, and J-Y L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer methods in applied mechanics and engineering*, 184(2):501–520, 2000.

[140] Mukesh Agarwala, David Bourell, Joseph Beaman, Harris Marcus, and Joel Barlow. Direct selective laser sintering of metals. *Rapid Prototyping Journal*, 1(1):26–36, 1995.

[141] Mingdong Zhou, Boyan S Lazarov, Fengwen Wang, and Ole Sigmund. Minimum length scale in topology optimization by geometric constraints. *Computer Methods in Applied Mechanics and Engineering*, 293:266–282, 2015.

[142] Tomás Zegard and Glaucio H Paulino. Bridging topology optimization and additive manufacturing. *Structural and Multidisciplinary Optimization*, 53(1):175–192, 2016.

[143] Anton Evgrafov, Misha Marie Gregersen, and Mads Peter Sørensen. Convergence of cell based finite volume discretizations for problems of control in the conduction coefficients. *ESAIM: Mathematical Modelling*

and Numerical Analysis, 45(6):1059–1080, 2011.