ABSTRACT

DACUS III, ROBERT WARREN. Development of a Multiphase Adjoint Capability in OpenFOAM. (Under the direction of Paul Turinsky.)

A multiphase adjoint capability was developed in the OpenFOAM computational fluid dynamics platform using an existing sub-cooled nucleate boiling model developed at NC State University. Although this solver contains a boiling model, the adjoint problem examined an adiabatic case which solves the multiphase Reynolds Averaged Navier-Stokes equations and $k - \epsilon$ turbulence equations. This capability constructs Jacobian matrices using automatic differentiation in the forward sense. These Jacobian matrices contain discrete coefficients that capture the dependency of all flow field variables. The Jacobian is required for the adjoint problem derivation with respect to sensitivity analysis. The sensitivity analysis examined perturbations in the dispersed phase void fraction with respect to changes in the drag, lift, and wall lubrication coefficients. Functional responses for void fraction perturbations within various regions of interest in the flow field were calculated using the adjoint solution and compared to the exact perturbation response. Overall, the three cases showed the adjoint problem was able to return useful perturbation responses in various regions of interest. For the drag coefficient case, the near-wall region adjoint response was found to approximate exact responses with 1% relative error for a perturbation of 2% the original void fraction. For the lift coefficient case, the bulk flow region adjoint response was found to approximate exact responses with 21% relative error for a perturbation of 0.04% of the original void fraction. The large perturbation in lift coefficient was required in order to see any perturbations in void fraction. For the wall lubrication coefficient case, the entrance flow region adjoint response was found to approximate exact responses with 6% relative error for a perturbation of 0.03% of the original void fraction. The forward computation time for the initial and perturbed cases took 13.7s while the adjoint response calculation time for five regions of interest only took 0.2s. This shows that multiphase adjoints using automatic differentiation for Jacobian construction can be useful with regards to sensitivity analysis.

Development of a Multiphase Adjoint Capability in OpenFOAM

by
Robert Warren Dacus III

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Nuclear Engineering

Raleigh, North Carolina

2016

APPROVED BY:

Nam Dinh	Igor Bolotnov
Sharon Lubkin	Greg DeWitt

Paul Turinsky
Chair of Advisory Committee

DEDICATION

To Laura. Soon you'll be stuck with me.

BIOGRAPHY

The author was born in Nashville, TN to Linda and Tad Dacus, later moving to Chattanooga, TN. He completed his undergraduate degree at the University of Tennessee at Chattanooga in chemical engineering. He moved to Raleigh, NC in order to pursue his Ph.D. in nuclear engineering from North Carolina State University under the direction of Paul Turinsky.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Paul Turinsky, for his patience and help with this difficult and oftentimes stubborn project. I would also like to thank the Rickover Fellowship staff in South Carolina as well as coworkers at both Knolls Atomic Power Lab (KAPL) and the Nuclear Engineering Department at North Carolina State University for their willingness to listen and offer advice in technical and non-technical areas alike. My family and friends were instrumental in supplying the much needed moral support and prayers necessary to complete this degree. Finally, I'd like to thank my dear fiancé Laura Cline for her constant encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

In computational engineering, there is often a tension between predicting physical phenomena at high levels of accuracy and maintaining a reasonable computational resource requirement. Oftentimes, the interests of accuracy and resource demand are directly opposed, and designers find themselves sacrificing one advantage for another. In many areas of engineering design, previous use of low fidelity models has proven acceptable for satisfying given design requirements, but the importance of efficiency and safety as they relate to complicated geometry and transients necessitates powerful and accurate methods.

Thermal hydraulic predictions of flow regimes within nuclear reactor cores require significant computational resources and accuracy to ensure that the core design does not violate the thermal limits of materials. Nuclear Regulatory Commission (NRC) requirements dictate the accurate simulation of plant performance during accident scenarios. A wide array of models at various levels of complexity are available to satisfy these design requirements.

Direct numerical simulation (DNS) models resolve flow field phenomena at the smallest physical and temporal length scales. These simulations are able to deterministically calculate velocity and pressure fields of turbulent flow but at a high computational cost. Multiphase computational fluid dynamics (CFD) can calculate three dimensional velocity and pressure fields for turbulent flow but require correlations for understanding the interactions between the fluid and vapor phases as well as averaging techniques to handle turbulence. These Reynolds Average Navier Stokes (RANS) methods, which average the time dependent behavior of turbulent oscillations, are able to approximate system wide pressures and velocities but are unable to determine local eddy configurations at the same level of detail as DNS methods.

Other less sophisticated methods like drift-flux or homogeneous equilibrium mixture model-

ing in conjunction with subchannel methods are often popular for system wide simulations due to their significantly lower computational burden. These methods employ area averaging and are often one dimensional in space, approximating missing dimensions' effects via correlations, and therefore have no ability to resolve local flow field phenomena that may be pertinent to thermal hydraulic design.

While one dimensional subchannel methods are useful in system wide simulations, emerging problems in nuclear design and safety require resolving three dimensional flow fields for complex geometry and transients. DNS techniques can resolve these fields, but the resource requirements for a single subchannel are enormous. For this reason, multiphase CFD has become a valuable computational capability for predicting flow characteristics as they relate to applications such as nuclear reactor safety and mixing phenomena around spacer grids within fuel assemblies [1][2]. Multiphase CFD still requires closure relationships for both the effects of turbulence as well as the transfer of mass, momentum, and energy between phases. Understanding the sensitivity of the parameters used in closure relationships is important for ensuring accuracy, but performing rigorous sensitivity analysis is computationally expensive.

Adjoint mathematics have been utilized for reactor physics and fluid dynamics applications with respect to verification, sensitivity analysis, adaptive mesh refinement, and optimization [3]-[5]. The use of adjoint methods can predict the sensitivities of a variety of physical parameters within a given problem, and they have been shown by Cacuci to return sensitivity responses for first and second order methods with significantly fewer computations than normal "large-scale" methods [6]. Adjoints have also been used for the optimization of flow geometry constrained to minimize pressure drop or drag for ducted flow. As the complexity of multiphase CFD methods increases, the demand for accurate verification tools and improvements in computational efficiency also increases, making the development of adjoint capabilities an attractive option.

This thesis builds a multiphase adjoint capability within the open source CFD code OpenFOAM. It develops a multiphase adjoint capability, the first of its kind, for the Eulerian-Eulerian two field, two phase sub-cooled nucleate boiling code `boilEulerFoam` developed by Dr. Nam Dinh of NC State Univerisity in OpenFOAM version 2.1.1 [7]. `BoilEulerFoam` capability includes forward solutions to two-field, steam-water momentum, mass, and energy equations with inter-phase momentum and mass transfer, wall boiling, interfacial area transfer, and liquid phase turbulence.For reference, the term "forward" is used to describe the problem set for which the adjoint capability is developed, as opposed to the term "adjoint" which describes the problem set used for functional response calculations. This thesis will develop a first look application of appropriate adjoint methodology within the two-field, two-phase fluid model with application

to sensitivity analysis.

This thesis covers three primary areas governing the development of multiphase adjoint capability. These areas are 1) the physics modeling of the thermal hydraulics computational methodology, 2) the computer science of the development of automatic differentiation tools within OpenFOAM, and 3) the mathematics of the adjoint problem definition and derivation. A final section draws conclusions and lays out future work for multiphase adjoint development.

1.1 Thermal Hydraulic Design and Simulation

For safe and efficient operation of a thermal nuclear power core, a fluid must effectively cool the reactor to appropriate temperatures in order to maintain the functional integrity of materials without compromising the configuration necessary to sustain a self-propagating chain reaction of the nuclear fuel. It is necessary for designers to understand in detail the behavior of system wide pressures, void fractions, and velocities and how they affect overall plant performance as well as local fluid behavior that can influence the characteristics of materials' corrosion and neutron flux. A wide variety of numerical tools are necessary in order to properly design and simulate a nuclear reactor and its plant and safety components. Thermal hydraulic tools present solutions to mass, momentum, and energy balance equations for single or multiphase fluid flow. The level of simplification and estimation of these equations coincides with the demand for accuracy that a designer requires.

One dimensional or three dimensional techniques for solving the two phase mixture equations are typical for basic system wide simulations. Design codes such as TRACE, RELAP and TRAC implement these methods for best-estimate thermal hydraulic design [8][9]. Correlations are chosen in order to close the six-equation, two-phase system describing the mass, momentum, and internal energy of a flow field. For simulation and analysis focused on core internal thermal hydraulic behavior, either closed-channel methods or sub-channel methods as incorporated by COBRA and VIPRE codes are employed [10][11]. The two-phase drift flux model, which considers only mixture momentum instead of separate phasic momentum equations and uses a correlation for determining the relative velocity of phases, may be employed for similar simulation conditions. Alternatively, the homogeneous equilibrium mixture (HEM) model assumes that both phases are at saturation and moving at the same velocity; therefore it is only necessary to solve for mixture momentum, mass, and energy. Although these methods are efficient, they typically are unable to resolve the flow mechanics near to the wall of the system where oftentimes safety criteria such as critical heat flux (CHF) are a concern.

For simulating the wall resolved effects of turbulent flow, multiphase computational fluid dynamics methods can be used to investigate flow phenomena that cannot be resolved by using simplified equations. In most cases, momentum, mass, and energy are solved explicitly for each phase rather than using correlations for relating pressure and velocity. Equations for turbulent kinetic energy and turbulent dispersion are also necessary in order to describe the Reynolds stresses on the system for Reynolds Averaged Navier Stokes (RANS) methods. Models such as k-ϵ or k-ω are typical for RANS equations and use correlations to describe the turbulence effects near the wall. As a consequence, the averaging technique used by RANS loses information regarding small eddy formation and dissipation within the flow field. Large eddy simulation (LES) can be used in its stead in order to retain the turbulence induced time dependent perturbation in the flow field. However, this is often computationally limiting due to the need for finer spatial meshing and time dependent ensemble average solutions. Codes such as STAR-CCM+, HYDRA-TH, and OpenFOAM employ RANS and LES methods for single and multiphase models [12][13][14].

DNS methods have the capability of resolving all micro scales of fluid flow. These methods employ a variety of techniques for tracking the interface between liquid and vapor. Examples are front tracking and level set methods which both have the capability of simulating individual bubble or droplet interactions within a fluid. In this case, no wall models or correlations are necessary due to the fact that the length scale of individual bubbles can be resolved. The computational resources needed to implement these methods for a single reactor core flow channel containing thousands of bubbles are vast, and DNS techniques are not typically used for large scale design. Computational codes such as PHASTA and FTC3D employ these DNS methods [15] [16]. They have proven useful for gaining insights and developing closure relationships for LES and RANS models, e.g. bubble lift and drag forces.

All of these fidelity levels are used in concert with one another to ensure proper design of a nuclear reactor core and its supporting thermal hydraulic systems. Within each method, various closure models or equation parameters can help improve the physical accuracy of the problem at the cost of additional computational resources. It is often difficult to resolve large geometries such as spacer grids with mixing vanes using DNS methods due to the computational cost. However, HEM and drift flux models cannot resolve the three dimensional flow behavior along the wall. Therefore, multiphase CFD has become a favorable choice with regards to complex boiling problems.

1.2 Adjoint Methods

An adjoint is a mathematical construct that mirrors behavior found in physical or forward problems and helps to describe the importance of functionals with respect to a specific quantity of interest. Adjoint solutions can be thought of as an importance weighting function that describes the spatial and temporal importance of a system response. It can be used to quantify the importance of functional responses such as the sensitivities of closure parameters, drag, or the effect of localized boundary conditions.

The derivation of an adjoint requires linear operators. This thesis considers the solution to pressure implicit multiphase flow equations, which is highly nonlinear. A common objection to adjoint methods is that they require linear operators to predict functional responses. Nonetheless, to obtain response perturbations, the adjoint approach can be accurate. Linearization techniques are required to build an adjoint problem, but these are often the same techniques used in Newtonian iteration methods.

Adjoint methods have found widespread use in the field of radiation transport due to its usefulness in perturbation analysis [17] - [19]. For small changes in specific parameters, adjoint solutions can describe the influence of perturbations within specific regions of interest without needing to re-solve the forward system of equations. In neutronics, these perturbed parameters include material properties, source distributions, and cross sections. In thermal hydraulics, perturbed parameters include material properties, closure relationships, and wall models [20] - [24].

Other adjoint applications include solution verification when exact correlations for complicated multiphase flow phenomena are unavailable. Adjoint methods are also used for adaptive grid refinement and provide error controlled localized grid refinement as an attempt to reduce the numerical error [25]. Adjoints have also been used by Othmer, Mani, and Mavriplis to describe the sensitivities of topological surface sensitivities for steady and unsteady systems [26][27].

Previous use of adjoint methods have primarily applied only to single phase CFD. With regards to multiphase CFD, the majority of adjoint capability has been limited to creeping flow with application to sub-surface oil reservoir optimization [28][29]. These models attempt to optimize design characteristics for the extraction of oil-water mixtures from porous media. They ignore inertial effects and use an empirical relationship between pressure drop and flow velocity. Multiphase flow in nuclear reactor design is highly turbulent and these empirical relationships are not appropriate for the CFD models examined by this thesis, therefore this thesis continues

adjoint development to include full solutions of the multiphase momentum equations.

1.2.1 Detector Example

In adjoint problems, one has the freedom to describe boundary conditions and source terms such that the evaluation of a functional yields a desired quantity of interest. A simple example is the detector response problem which is applicable to both thermal hydraulics and neutronics problems. In this problem, the response of a detector is desired as a function of the location of a source term. Typically, moving a source term anywhere within the geometry of the problem requires the equations to be solved a second time. Adjoint methods present an alternative approach. An adjoint problem is solved once and then its solution can be used multiple times to describe responses for various source term locations.

Starting with the following forward and adjoint linear equations

$$\mathbf{A}\left[\phi\right] = Q, \qquad \mathbf{A}^{*}\left[\phi^{*}\right] = Q^{*} \tag{1.1}$$

the adjoint operator $\mathbf{A}^{*}$ is defined such that the following inner product equality holds for all ϕ and $\phi*$ within the solution space

$$\left\langle \mathbf{A}\left[\phi\right], \phi^{*} \right\rangle = \left\langle \mathbf{A}^{*}\left[\,\phi^{*}\,\right], \phi \right\rangle \tag{1.2}$$

Ensuring equality of (1.2) imposes restrictions on the initial and boundary conditions of the adjoint problem. If the response function desired is defined as $R = \langle \Sigma, \phi \rangle$ then one can provide an exact evaluation of the response when the adjoint source term is given by $Q^{*} = \Sigma$ according to

$$\langle \Sigma, \phi \rangle = \langle Q^{*}, \phi \rangle = \langle \mathbf{A}^{*}[\phi^{*}], \phi \rangle = \left\langle \mathbf{A}\left[\phi\right], \phi^{*} \right\rangle = \langle Q, \phi^{*} \rangle \tag{1.3}$$

Using (1.3), one is able to determine the response of a detector for various Q locations without having to solve for ϕ each time Q is moved. Instead, ϕ^{*} is determined once and then used to provide the response for any value of Q in the solution space. This method of functional response prediction with respect to adjoint solutions can provide a means for performing sensitivity analysis for a wide array of perturbations, or for evaluating responses for various boundary conditions, etc.

1.2.2 Adjoint Operators

For each linear operator contained in a partial differential equation, there is a subsequent adjoint operator that describes the appropriate adjoint behavior. Using the nomenclature from

(1.1), operators are said to be self adjoint if $\mathbf{A}^* = \mathbf{A}$. Second derivatives and constant multipliers are examples of self adjoint operators while first order derivatives are not self adjoint [17].

There are several ways to derive adjoint operators for a given system of equations. A physical adjoint is defined as an operator that is derived from the continuous set of forward equations. One constrains the adjoint to satisfy equation (1.2) and, using integration by parts, derives an appropriate adjoint operator. The application of necessary boundary conditions and initial conditions ensures that the equality in (1.2) holds. Another approach to deriving physical adjoint operators uses cost functions and the method of Lagrange multipliers. This method "adjoins" a Lagrange multiplier to a general cost function for a given problem. One can then describe the maxima and minima of this cost function without needing to substitute the solution into the cost function.

A third method for deriving adjoint operators uses the coefficient matrix from the linear solve of a discretized system. Since the adjoint of a matrix operator is its conjugate transpose, then for a discretized forward matrix $\mathbf{A}$ operating on the forward solution vector $\vec{\phi}$, we have the following forward and adjoint problem

$$\mathbf{A}\left[\phi\right] = \mathbf{Q}, \qquad \mathbf{A^T}\left[\phi^*\right] = \mathbf{Q}^* \tag{1.4}$$

This method requires no integration by parts or Lagrange multipliers and is only applicable to linear problems. This is called a mathematical or discrete adjoint. This is the most common continuous adjoint equation formulation used in reactor physics applications. Most adjoint operators are typically discretized using the same methodology as the forward problem, and similar solution techniques are applied to both systems of equations. There is, however, no guarantee that the discretized physical adjoint solution will be the same as the mathematical adjoint solution. If the problems are defined correctly, the functional as predicted by the adjoint solutions should be the same for both the physical and mathematical problems within the numerical methods accuracy.

1.3 Automatic Differentiation

Adjoint techniques are only applicable to linear problems. Multiphase CFD equations, however, are highly nonlinear. In order to linearize these equations, a simple Taylor expansion of a given nonlinear operator $\mathbf{F}[\vec{\phi}] = \vec{d}$ can be written to first order accuracy as

$$\left.\frac{\partial \mathbf{F}}{\partial \vec{\phi}}\right|_{\vec{\phi_o}} \left(\vec{\phi} - \vec{\phi_o}\right) = \mathbf{F}[\vec{\phi}] - \vec{d}$$

the partial derivative $\frac{\partial \mathbf{F}}{\partial \vec{\phi}}$ defines the Jacobian matrix

$$\frac{\partial \mathbf{F}}{\partial \vec{\phi}} = \begin{bmatrix} \frac{\partial F_1}{\partial \phi_1} & \frac{\partial F_1}{\partial \phi_2} & \cdots & \frac{\partial F_1}{\partial \phi_n} \\ \frac{\partial F_2}{\partial \phi_1} & \frac{\partial F_2}{\partial \phi_2} & & \\ \vdots & & \ddots & \\ \frac{\partial F_n}{\partial \phi_1} & & & \frac{\partial F_n}{\partial \phi_n} \end{bmatrix}$$

This Jacobian matrix is often present in Newton methods for finite-element solutions to the Navier-Stokes equations [30]. For these methods, it is straightforward to linearize the problem and obtain an adjoint capability. OpenFOAM, however, uses a finite volume based solver with pressure implicit splitting of operators (PISO) and contains no Jacobian. In order to linearize the discrete equations, this thesis developed an automatic differentiation (AD) capability in the forward sense that expanded on existing capability and made AD objects available to all operations pertaining to the subcooled nucleate boiling model.

Automatic differentiation is a method with which derivatives of functions can be calculated automatically based on existing computational algorithms. AD includes a forward mode and a reverse mode for calculating derivatives or gradients. The forward mode uses the chain rule in order to explicitly calculate required derivatives. Operators can be overloaded such that derivatives can be calculated exactly according to the called algebraic functions.

As an example, we motivate an example fro Rall et. al.[33]. Let us introduce a function $f(\vec{x}) = f(x_1, \ldots, x_n)$. Let t_{n+i} with $i \in (1, \ldots, m)$ be a set of intermediate values generated as a part of the algorithm used to determine f. Using the chain rule, we can determine the derivative of an intermediate value t according to

$$\frac{\partial t_j}{\partial x_i} = \sum_{k \in K_j} \frac{\partial t_i}{\partial t_k} \frac{\partial t_k}{\partial x_i}$$

for $j = n+1, n+2, \ldots, n+m$ and where K_j is the $k < i$ set of indices that t_i depends on t_k. When the final operation t_{n+m} returns f, the derivative of the algorithm is also returned. Note that the number of operations required to calculate ∇f is roughly $n \cdot m$ using the forward mode of automatic differentiation.

The reverse mode of automatic differentiation calculates derivatives by evaluating the function f and then working backwards such that

$$\frac{\partial f_i}{\partial t_k} = \sum_{j \in I_k} \frac{\partial f_i}{\partial t_j} \frac{\partial t_j}{\partial t_k},$$

for $k = n+m, n+m-1, \ldots, i+1$ and where I_k is the $k > j$ set of indices where t_j depends on t_k. This requires storage of all $\frac{\partial t_j}{\partial t_k}$ derivatives required to calculate a derivative of f. The final $\frac{\partial f_i}{\partial t_{n+1}}$ will return the derivative with respect to the first independent variable, which is often space or time. The gradient of a product operator was shown by Griewank to require $\frac{1}{2}n^2$ non-trivial multiplications using the forward mode and $3n-3$ using the reverse mode [34]. The reverse mode is therefore said to be much more efficient than the forward mode, however, storage of intermediate operations can make the reverse mode more expensive with regards to memory. Due to simplicity and the need to store full Jacobian matrices as explained in section 3.3.1, the forward mode of automatic differentiation was selected for use in the derivation of multiphase adjoint capability in OpenFOAM.

Various black-box AD tools are available, such as ADIFOR, TAMC, TAF, and Tapenade [35]. These methods employ AD by either overloading existing operators or performing source code transformations. Due to OpenFOAM's object oriented nature and the existing work done by Jasak, AD capability was coded directly into OpenFOAM and takes full advantage of its hierarchy, operators, and templates. This capability is termed FadOne, a first look at FOAM AD, and is used for the construction of Jacobian matrices necessary for adjoint calculation.

1.4 OpenFOAM

OpenFOAM is an open source, community based computational fluid dynamics software developed in C++. First called FOAM (Field Operations and Manipulations), it was created by Dr. Hrvoje Jasak and Dr. Henry Weller at Imperial College in London during the late 1980s as an object oriented alternative to existing CFD methodology. The official OpenFOAM release and open source distribution is handled by the OpenFOAM Foundation. The FOAM-Extend project is a subset of OpenFOAM that formalizes the open spirit of collaboration and makes a wide variety of CFD capability available to the OpenFOAM community.

OpenFOAM was selected as the platform for multiphase adjoint development due to its open source nature and the subsequent unlimited access to source code. Thermal hydraulic development groups traditionally have had the option of either developing codes in house, or they have licensed the use of commercial off the shelf (COTS) software such as ANSYS CFX or FLOW-3D. Both options are considered expensive with respect to development time or licensing costs.

The CFD utility OpenFOAM is regarded as "open source" in that all source code pertaining to releases are published in full and are available to any user of OpenFOAM without charge. Subsequent user demand drives further development of OpenFOAM, and additional releases are made available per the requests and contracts of the OpenFOAM community. As a result, OpenFOAM can be considered a middle ground between full in-house CFD code development and the licensing of COTS. OpenFOAM's primary advantage over COTS is its lack of upfront licensing costs as well as full user access to all source code. OpenFOAM also has substantial professional development, and does not require the development time effort of in-house codes. In order to tailor OpenFOAM to specific problems, there is a man-hour requirement associated with training and development, and effective use of OpenFOAM may also necessitate support and documentation licenses from external companies such as Engys or OpenCFD.

The C++ methodology used in OpenFOAM is a powerful and consistent representation of object oriented programming. C++ encapsulation in OpenFOAM can be thought of as a toolkit of building blocks that, when assembled correctly, can be used for many CFD applications without a detailed understanding of how each building block functions. The primary functionality of OpenFOAM objects is encapsulated underneath easily implemented code that reads like the equation itself. For example, the scalar transport equation is as follows

$$\frac{\partial T}{\partial t} + \nabla \cdot \left(T\vec{\phi} \right) = \nabla \cdot (D \, \nabla T) \tag{1.5}$$

In OpenFOAM C++, this same equation is written as

```
solve(
      fvm::ddt(T) + fvm::div(phi, T)
      == fvm::laplacian(D, T)
      );
```

where OpenFOAM uses the face interpolated flux ϕ rather than velocity according to finite volume methods.

Systems of equations in OpenFOAM closely resemble the solved equations which helps the readability of the code. OpenFOAM also utilizes advanced templates and polymorphism so that the variable `T` from the previous example can be any type of parameter, such as a vector or tensor, and the functionality remains the same. This encapsulation and polymorphism is what helps make OpenFOAM such a powerful platform for CFD development.

The OpenFOAM community has previously performed verification case studies for single phase and multiphase, adiabatic and heated OpenFOAM applications and has demonstrated the codes initial viability as a supplement to existing COTS CFD software [36]. Dr. Nam Dinh and his research team have enhanced existing multiphase capability available in OpenFOAM release 2.2.1 to include a subcooled nucleate boiling model called `boilEulerFoam`. Recent development has added wall lubrication, turbulent dispersion, and interfacial area transfer equations to the existing model. This thesis uses OpenFOAM release 2.2.1 and the most recent version of `boilEulerFoam` as the bases for adjoint development.

Chapter 2

Physics Modeling and Methodology

This chapter explains the subcooled nucleate boiling methodology within the OpenFOAM solver `boilEulerFoam`. This solver simulates sub-cooled nucleate boiling and multiphase behavior by implementing an Eulerian-Eulerian two-field, two-phase flow model. This model considers each phase to be inter-penetrating the other in two distinct fields with appropriate interphase momentum and mass transfer terms. Void fractions are weighted according to a conditional averaging technique as developed by Weller [14]. The effect of turbulence on the flow field is simulated using a Reynolds Averaged Navier-Stokes (RANS) k-ϵ model and appropriate wall functions[53].

This section examines the forward multiphase RANS equations solved by `boilEulerFoam`, discusses some initial results using the `boilEulerFoam` solver, and examines some basic verification cases including a correlation for the onset of nucleate boiling. It also describes in detail the discretization and solution procedure for an adiabatic case to be used for adjoint construction.

2.1 Continuous Forward Equations for `boilEulerFoam`

The following are the continuous equations implemented in the sub-cooled boiling model.

Continuity

$$\frac{\partial \alpha_k}{\partial t} + \nabla \cdot (\alpha_k \vec{u}_k) = \frac{\Gamma_{ki} - \Gamma_{ik}}{\rho_k}$$

$$\alpha_l = 1 - \alpha_g$$

where g and l represent the dispersed vapor and continuous liquid phases respectively, and α, ρ, and $\vec{u}$ represent the phase volume fraction, density, and velocity. Γ_{ki} denotes the evaporation or condensation rate per unit volume from phase k to phase i.

Momentum

$$\frac{\partial \alpha_k \vec{u}_k}{\partial t} + \nabla \cdot (\alpha_k \vec{u}_k \vec{u}_k) =$$
$$-\nabla \cdot \left(\alpha_k \left(\mathbf{R}_k + \mathbf{R}_k^t\right)\right) - \frac{\alpha_k}{\rho_k}\vec{\nabla} p + \alpha_k \vec{g} + \frac{\vec{M}_k}{\rho_k} + \frac{\Gamma_{ki}\vec{u}_i - \Gamma_{ik}\vec{u}_k}{\rho_k}$$

where $\vec{M}_k$ denotes the interphase momentum transfer for phase k per unit volume. The stress tensor $\mathbf{R}_k^{\text{eff}} = \mathbf{R}_k + \mathbf{R}_k^t$ is given by

$$\mathbf{R}_k^{\text{eff}} = \mathbf{R}_k + \mathbf{R}_k^t = -\left(\nu_k + \nu_k^t\right)\left(\nabla \vec{u}_k + (\nabla \vec{u}_k)^T - \frac{2}{3}\mathbf{I}\nabla \cdot \vec{u}_k\right) + \frac{2}{3}\mathbf{I}k_k$$

where k_k is the turbulent kinetic energy for phase k as described by the k-ϵ equations. The interphase momentum transfer term $\vec{M}_k$ contains expressions describing drag, lift, virtual mass, turbulent dispersion, and wall lubrication. Since interfacial momentum is the transfer of momentum from the dispersed phase to the continuous phase, we have $\vec{M}_g = -\vec{M}_l$. We therefore describe the interfacial momentum transfer as it relates to the dispersed phase.

The expression for the drag force interphase momentum transfer according to Ishii-Zuber is as follows [37]

$$\vec{M}_g^D = -C_D \frac{3}{4}\frac{\rho_l}{D_S}\alpha_g \left|\left|\vec{u}_g - \vec{u}_l\right|\right| (\vec{u}_g - \vec{u}_l)$$

with the following correlations and constants

$$C_D = \max\left(\frac{24 + 3.6(Re_{bm})^{0.687}}{Re_{bm}},\ 0.44\right)$$
$$Re_{bm} = -\frac{\rho_l \left|\left|\vec{u}_g - \vec{u}_l\right|\right| D_s}{\mu_m}$$
$$\mu_m = \mu_l \left(1 - \frac{\alpha_g}{0.52}\right)^{-1.3\mu^*}$$
$$\mu^* = \frac{\mu_g + 0.4\mu_l}{\mu_g + \mu_l}$$

The above Ishii-Zuber drag model uses the mean bubble diameter D_S calculated from the definition of interfacial area and assumes that the bubbles are spherical, $\alpha_g < 0.52$, and $0.2 < Re_b < 1.0 \times 10^5$ [37]. This closure relationship simulates the drag experienced by the dispersed phase as it travels through the continuous phase. Extensive research has examined the accuracy of drag closure relationships, and `boilEulerFoam` has the option of using several models developed by Wen-Yu, Ergun, Schiller-Naumann, Gibilaro, and Syamlal-OBrien [38] -

[40]. For simplicity, this thesis keeps the lift force coefficient constant $C_L = 0.01$

The expression for the lift force interphase momentum transfer term is as follows

$$\vec{M}_g^L = C_L \alpha_g \left(\vec{u}_g - \vec{u}_l\right) \times \left(\nabla \times \vec{u}_l\right)$$

For two dimensional pipe flow, lift forces affect the radial void distribution and can either push bubbles towards or away from the wall. This behavior mimics observations seen in experiments and is thought to be related to the liquid velocity at the surface of a bubble. For small spherical bubbles, the lift force pushes bubbles towards the wall of a pipe. `boilEulerFoam` has the capability of accounting for bubble size and deformation using the Rusche and Tomiyama models to calculate the lift force coefficient [41][42].

The expression for the virtual mass interphase momentum transfer term is as follows

$$\vec{M}_g^{VM} = -C_{vm} \alpha_g \alpha_l \rho_l \left(\frac{D\vec{u}_g}{Dt} - \frac{D\vec{u}_l}{Dt}\right)$$

where $C_{vm} = 0.5$ is the virtual mass coefficient. Virtual mass is the interfacial momentum force that resists the acceleration of bubbles as they pass through the continuous phase. Since these phases are considered to be interpenetration, virtual mass attempts to account for the physical space left behind by a bubble as it travels through a liquid. For steady state solutions where the dispersed phase is no longer accelerating, $\vec{M}_g^{VM} = 0$. $\dfrac{D}{Dt}$ is the total material derivative and is given by

$$\frac{D\phi}{Dt} = \frac{\partial \phi}{\partial t} + \vec{u} \cdot \vec{\nabla} \phi$$

The expression for the turbulent dispersion interfacial momentum transfer term is as follows

$$\vec{M}_g^T = -C_{td} \rho_l k_l \nabla \alpha_g$$

where $C_{td} = 0.01$ is the turbulent dispersion coefficient. Turbulent dispersion force describes the force that resists the aggregation of void fraction as a function of turbulent kinetic energy. `boilEulerFoam` has the capability of using the Burns and Gosman models which describes the relationship of turbulent dispersion and velocity rather than turbulent kinetic energy [43][44].

The expression for the wall lubrication interphase momentum transfer term is as follows

$$\vec{M}_g^{WL} = C_W \alpha_g \rho_l \left| \vec{u}_r - (\vec{u}_r \cdot \vec{n}_w)\, \vec{n}_w \right|^2$$

where $\vec{u}_r = \vec{u}_g - \vec{u}_l$ is the relative velocity. $|\vec{u}_r - (\vec{u}_r \cdot \vec{n}_w)\, \vec{n}_w|$ is the projection of the relative velocity parallel to the wall normal vector n_w. According to Frank, C_w is defined as

$$C_W = C_{wl} \max\left(0,\ \frac{(1 - y_r)}{C_{wd} y_w y_r^{p-1}}\right)$$

with y_w defined as the distance to the wall, $C_{wd} = 6.8$, $p = 1.7$, and the ratio y_r defined as

$$y_r = \frac{y_w}{C_{wc} D_S}$$

with $C_{wc} = 10.0$. The wall lubrication force describes the experimental phenomenon that pushes bubbles slightly off of the wall [46].

Energy

It is assumed that all vapor is at saturation, and transfer from the vapor to the liquid phase is solely due to condensation. Therefore, the energy conservation equation is written for the liquid phase only in terms of specific enthalpy

$$\frac{\partial \alpha_l h_l}{\partial t} + \nabla \cdot (\alpha_l h_l \vec{u}_l) = -\frac{1}{\rho_l} \nabla \cdot \left[\alpha_l \left(\vec{q}_l + \vec{q}_{l,t}\right)\right] + \frac{\alpha_l}{\rho_l} \frac{Dp}{Dt} + \frac{\Gamma_{lg} h_{g,sat} - \Gamma_{gl} h_l}{\rho_l} + \frac{q_w'' A_w''}{\rho_l}$$

where h_l denotes the liquid specific enthalpy, $\vec{q}_l$ and $\vec{q}_{l,t}$ represent the molecular and turbulent heat fluxes for the liquid phase, A_w'' represents the wall contact area per unit volume, and q_w'' represents the wall heat flux density. The molecular and turbulent heat fluxes are given by

$$\vec{q_{l,(t)}} = -\frac{\lambda_{l,(t)}}{c_{pl}} \vec{\nabla} h_l$$

where λ denotes thermal conductivity and c_p denotes specific heat.

Interfacial Area

Interfacial area is defined as the surface area of void bubbles per unit volume and is used to calculate the mean bubble diameter according to

$$a_i = \frac{6\alpha_g}{D_S}$$

The transport of interfacial area is modeled according to

$$\frac{\partial a_i}{\partial t} + \nabla \cdot (a_i \, \vec{u}_g) = -\frac{2}{3}\frac{a_i}{\alpha_g \rho_g}\Gamma_{lg} + \Phi_{BB} + \Phi_{BC} + \Phi_{NUC}$$

where Φ_{BB}, Φ_{BC}, and Φ_{NUC} are the bubble breakup, coalescence, and nucleation terms respectively. The nucleation term is relevant in near wall cells only. `boilEulerFoam` uses the Hibiki and Ishii model to describe the bubble sink and source terms [37]. The interfacial area transfer equation is used to solve for the bubble diameter which is required by the interfacial momentum transfer models.

Turbulence

Turbulence modeling follows typical RANS k-ϵ methodology where the effective viscosity of the liquid phase is given by the sum of the molecular and turbulent viscosities.

$$\nu_l^{\text{eff}} = \nu_l + \nu_l^t$$

The turbulent viscosity ν_l^t for multiphase flows is equal to the sum of shear induced and bubble induced turbulent viscosities. Using this methodology, the equation for turbulent viscosity is as follows

$$v_l^t = C_\mu \frac{k_l^2}{\epsilon_l} + \frac{1}{2} C_{\mu b} D_S \alpha_g ||\vec{u}_g - \vec{u}_l||$$

where $C_\mu = 0.09$ and $C_{\mu b} = 1.2$.

The effective viscosity of the vapor phase is dependent on the effective liquid viscosity and is given by

$$\nu_g^{\text{eff}} = \nu_g + C_t^2 v_l^t$$

where C_t is the turbulent response coefficient defined by the ratio of velocity fluctuations of the dispersed phase to the continuous phase, or

$$C_t = \frac{u_g'}{u_l'}$$

By assumption, the turbulence of the vapor phase depends on the turbulence of the liquid phase. Therefore, only the continuous phase turbulence k - ϵ equations are solved.

$$\frac{\partial \alpha_l k_l}{\partial t} + (\vec{u}_l \cdot \nabla)\, \alpha_l k_l - \nabla \cdot \left(\frac{\nu_l^{\text{eff}}}{\sigma_k} \vec{\nabla} k_l \right) = \alpha_l P_l - \alpha_l \epsilon_l$$

$$\frac{\partial \alpha_l \epsilon_l}{\partial t} + (\vec{u}_l \cdot \nabla)\, \alpha_l \epsilon_l - \nabla \cdot \left(\frac{\nu_l^{\text{eff}}}{\sigma_k} \vec{\nabla} \epsilon \right) = \frac{\epsilon_l}{k_l} \alpha_l \left(C_1 P_l - C_2 \epsilon_l \right)$$

with the following constants

$$\begin{aligned} C_1 &= 1.44 \\ C_2 &= 1.92 \\ \sigma_k &= 1.0 \\ \sigma_\epsilon &= 1.3 \end{aligned}$$

The kinetic energy production term P_l is defined as

$$P_l = 2\nu_l^{\text{eff}} \left(\nabla \vec{u}_l \cdot \text{dev} \left(\nabla \vec{u}_l + (\nabla \vec{u}_l)^T \right) \right)$$

Summary

The unknown variables to be solved for include

1. Void fraction α_g (or α_l)
2. Liquid velocity $\vec{u}_l$
3. Vapor velocity $\vec{u}_g$
4. Liquid specific enthalpy h_l
5. Liquid turbulent kinetic energy k_l
6. Liquid turbulent dissipation ϵ_l
7. Pressure p
8. Interfacial area a_i

totaling 8 dependent variables. The given equations by the forward two-field, two-phase model include

1. Liquid mass conservation equation
2. Vapor mass conservation equation
3. Liquid momentum conservation equation
4. Vapor momentum conservation equation
5. Liquid energy conservation equation

6. Liquid turbulent kinetic energy equation

7. Liquid turbulent dissipation equation

8. Interfacial area transport equation

totaling 8 equations which closes the system.

2.1.1 Forward Results Using `boilEulerFoam`

The following section contains results of a base case sub-cooled nucleate boiling simulation as predicted by `boilEulerFoam`.

Flow Field Distributions

The base case is a vertical two dimensional axi-symmetric pipe geometry provided with the original `boilEulerFoam` source code. It demonstrates the solvers ability to simulate void generation and its impact on flow field predictions for constant heat flux. Table 2.1 contains the input flow field and geometry parameters for the base case simulation.

Table 2.1: **Base case input parameters**

Parameter	**OF Variable**	**Value**
Inlet Pressure	p	4.5 MPa
Inlet Velocity	Uwater	1.0 m/s
Inlet Sub-cooling	Tsub	26.6^{o} C
Inlet Void	alphaair	0
Wall Heat Flux	qWall	3.8×10^2 kW/m^2
Pipe Length	L	2 m
Pipe Radius	r	1.2 cm
Lift Coefficient	Cl	0.01
Virtual Mass Coefficient	Cvm	0.5
Turbulent Dispersion Coefficient	Ct	0.01
Minimum Bubble Diameter	D_o	0.1 mm
Maximum Bubble Diameter	D_1	1 mm
Frank Wall Model Coefficients	Cwc, Cwd, p	10.0, 6.8, 1.7
Drag Model	Ishii-Zuber	

Figure 2.1 illustrates the mesh used in the axi-symmetric OpenFOAM multiphase simula-

tion. The pipe models vertical flow and $\vec{g}$ acts in the negative z direction, but the mesh is shown horizontally for convenience. The top boundary condition is a solid wall with zero slip and a uniform profile heat flux. Heat loss is ignored. The bottom boundary condition is a line of rotational symmetry that runs through the center of the pipe. The axi-symmetric pipe mesh used a wall spacing ratio such that $\frac{\Delta_{wall}}{\Delta_{symm}} = 0.25$. There were a total of 3,171 grid nodes within the mesh, and spacing in the axial direction was uniform. The Figures containing computational results are scaled radially by a factor of 20 such that pertinent field information is visible.

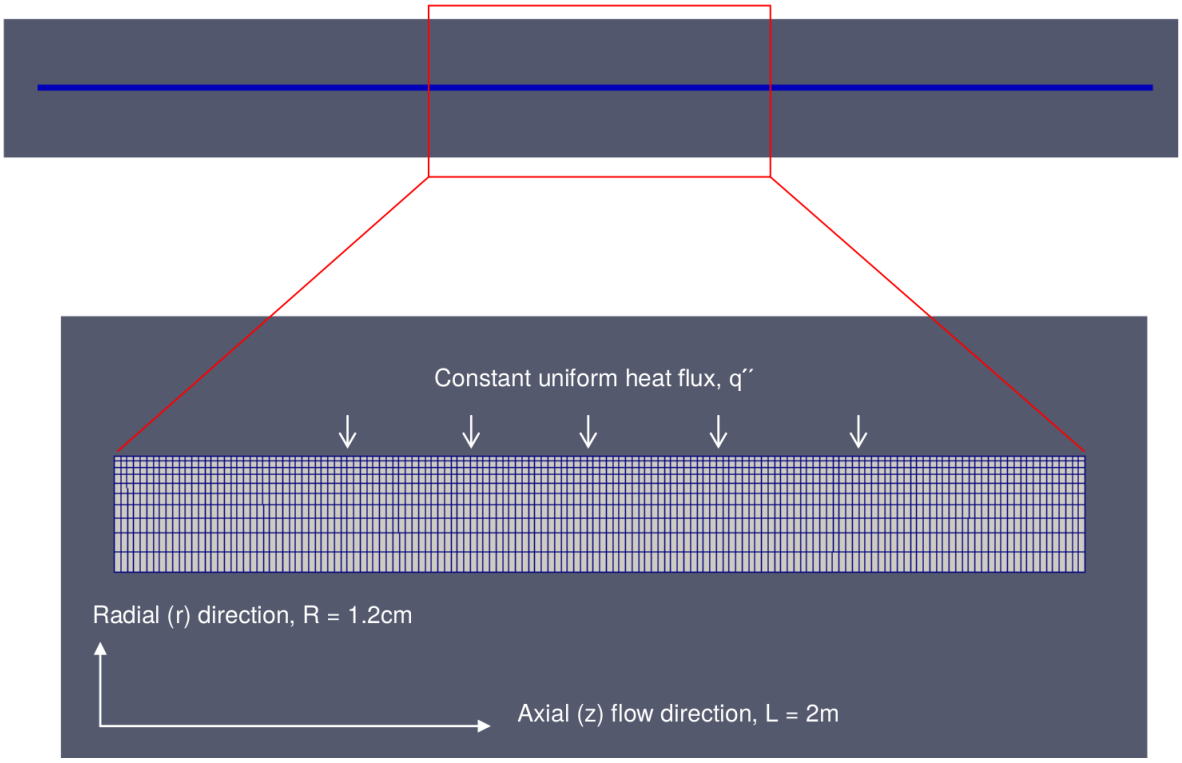


Figure 2.1: Representative mesh used in `boilEulerFoam`'s simulation of multiphase flow through an axially symmetric pipe

Figures 2.2 through 2.7 contain the axi-symmetric simulation results for various flow phenomena as predicted by `boilEulerFoam`.

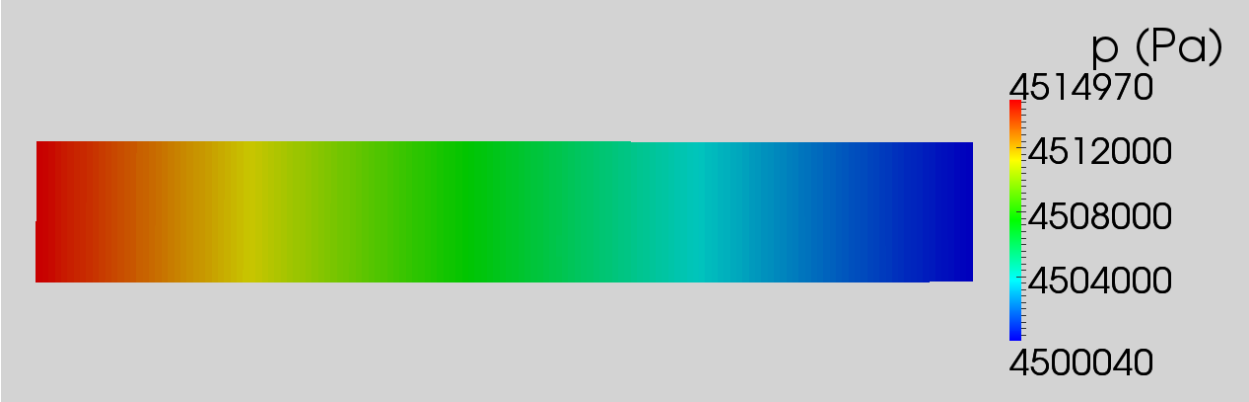


Figure 2.2: Pressure field results for `boilEulerFoam` multiphase base case

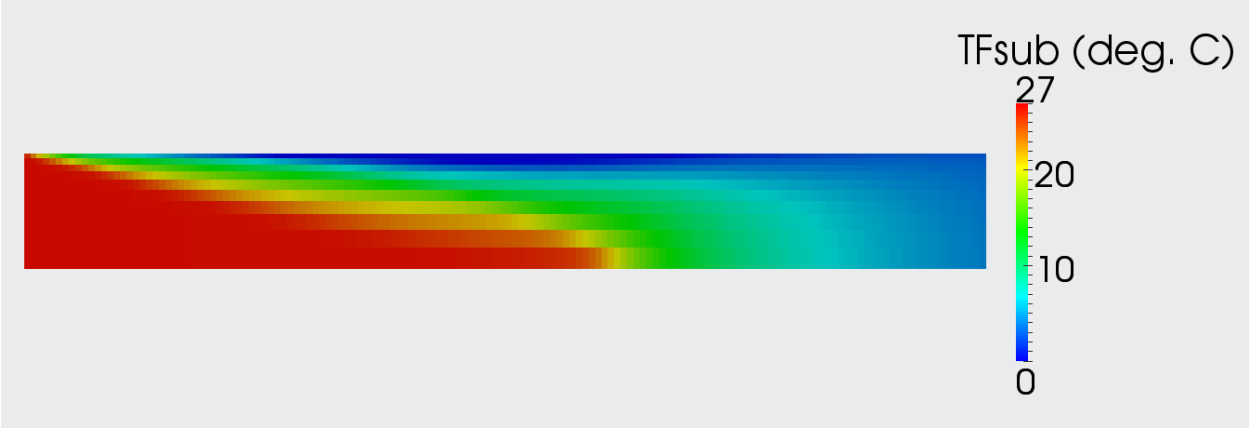


Figure 2.3: Degrees of sub-cooling results for `boilEulerFoam` multiphase base case

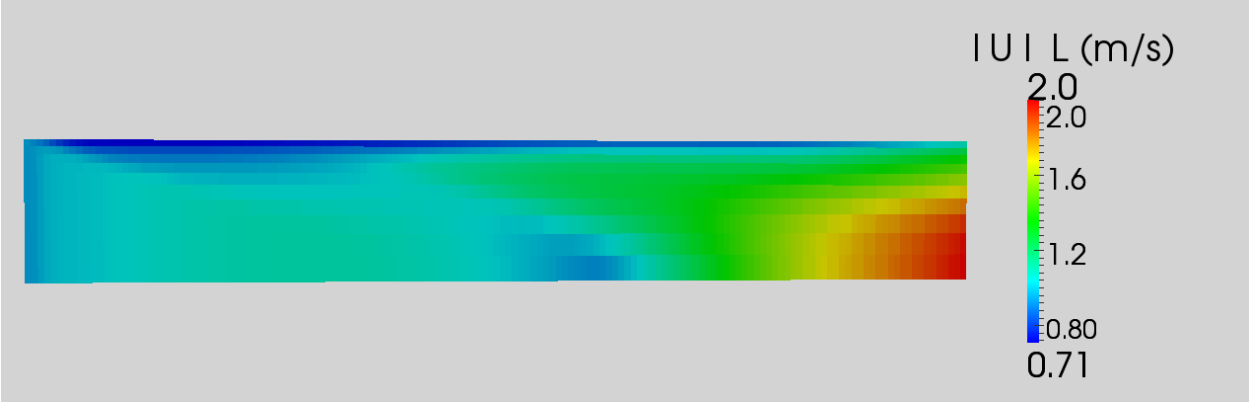


Figure 2.4: Liquid velocity magnitude field results for `boilEulerFoam` multiphase base case

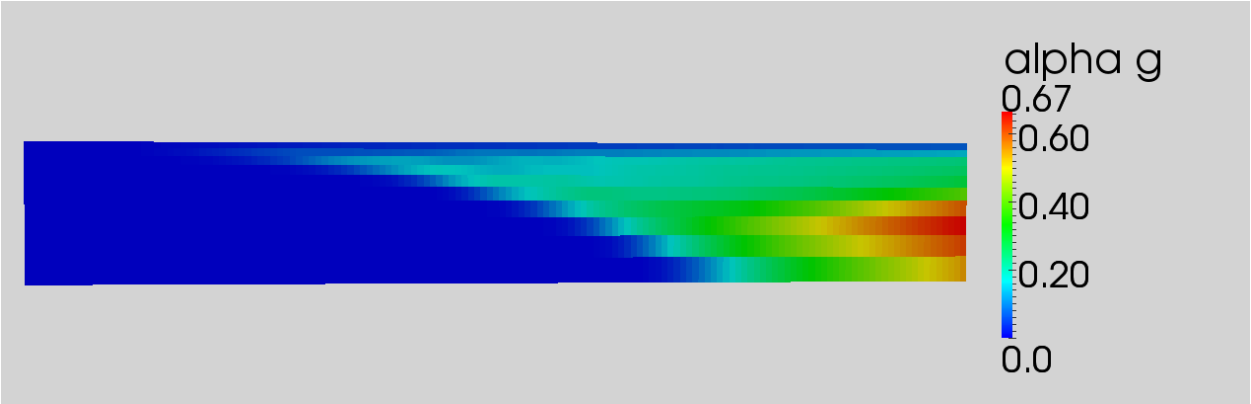


Figure 2.5: Void fraction results for `boilEulerFoam` multiphase base case

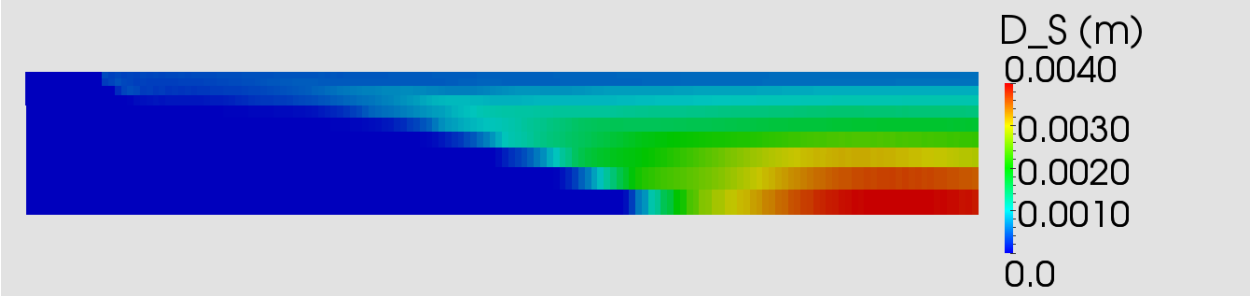


Figure 2.6: Bubble diameter results for `boilEulerFoam` multiphase base case

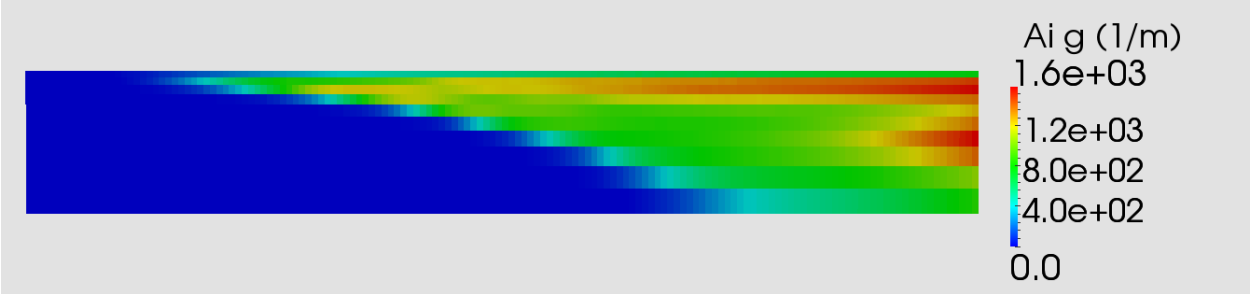


Figure 2.7: Interfacial area results for `boilEulerFoam` multiphase base case

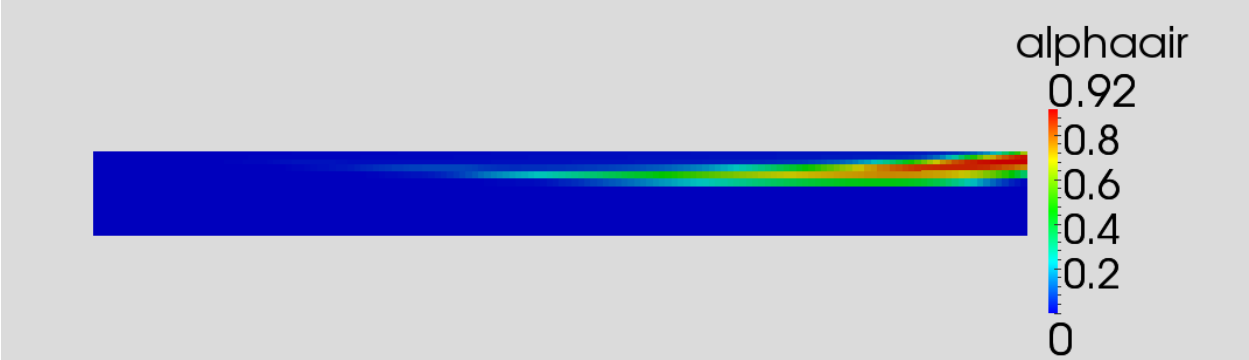


Figure 2.8: Void fraction results for old version of `boilEulerFoam` without wall lubrication and turbulent dispersion interphase momentum transfer terms

Figure 2.2 indicates a linear pressure drop of 15 kPa from the inlet of the pipe to its exit. Because the flow is friction dominated, this is an expected result. In Figure 2.3, the fluid enters at the prescribed inlet sub-cooling value and increases in temperature first along the wall. The temperature continues to increase until the bulk flow is only several degrees below saturation.

Initially, the radial velocity along the boundary is distributed according to typical turbulent flow behavior as prescribed by the k-ϵ turbulence model, shown in Figure 2.4. However, as the void fraction increases, the differences in density between the liquid and vapor phases result in an accelerative buoyancy force, and the location coinciding with increased void fraction also increases in velocity magnitude.

According to the wall boiling model, void is only produced in the near wall cell values shown in Figure 2.5. This is consistent with the initial void production near the wall for $L = 0.5$ m. As boiling continues, the interfacial forces present dictate the distribution of dispersed vapor shown in Figure 2.5. Wall lubrication, according to the Frank model, ensures that void is pushed away from (negative $\vec{r}$ direction) the near wall region. Turbulent dispersion acts on the larger gradients of α_g and resists sharp changes in void fraction. Drag resists the propensity for the dispersed phase to move faster than the continuous phase due to differences in density. All of these forces in conjunction with the transport of mass and momentum result in the void distribution shown in Figure 2.5.

The mean bubble diameter shown in Figure 2.6 is the largest in regions with higher concentration of void and is directly related to the interfacial area of the solution, shown in Figure 2.7. Bubble diameter is used in interphase momentum transfer calculation and will also have

an impact on the overall void distribution.

Figure 2.8 shows results from a previous version of `boilEulerFoam` that does not include wall lubrication and turbulent dispersion interphase momentum transfer. In the absence of wall lubrication, there is no wall normal force to counteract lift and the system will not converge. Because of this, the lift force in Figure 2.8 is reversed, and the void congregates in a region off of the wall in a way that is likely unphysical.

From the results shown in Figures 2.5 and 2.8, it can be said that the void fraction distribution is strongly dependent on interphase momentum transfer closure relationships. Future development of `boilEulerFoam`'s subcooled nucleate boiling capability must rely on detailed numerical studies and experimental comparisons to ensure that the distribution shown in Figure 2.5 is accurate. Due to the importance of these closure relationships, the numerical adjoint perturbations contained in section 4.3 will focus on aspects of interphase forces.

Onset of Nucleate Boiling Correlation

The Thom and Dittus-Boelter correlations can estimate the onset of nucleate boiling or z_{ONB}, and results were compared to `boilEulerFoam`. The Dittus-Boelter correlation for wall temperature $T_{w,D-B}$ is applicable to single phase convective turbulent heat transfer and is given by [49]

$$T_{w,D-B} = T_m(z) + \frac{q(z)}{h_{D-B}}$$

where $T_m(z)$ is determined according to the relationship

$$\dot{m} c_p \int_{T_{in}}^{T_m(z)} dT = 2\pi D \int_0^z q(z) dz$$

where z is the height or axial coordinate of the pipe and c_p is a function of the inlet conditions. Because the heat flux is uniform along the wall, the previous equation can be rewritten as

$$T_m(z) = \frac{qz}{\rho A_x u_z c_p} + T_{in}$$

The heat transfer coefficient of the Dittus-Boelter correlation is calculated according to

$$h_{D-B} = \frac{k_l Nu}{D_h}$$

where the Nu is given by

$$Nu = 0.023 Re^{0.8} Pr^{0.4}$$

All of this together creates a function for wall temperature that is dependent on z.

The Thom correlation describes the wall temperature for sub-cooled boiling flows. It gives a wall temperature $T_{w,Thom}$ according to the following expression

$$T_{w,Thom} = \sqrt{\frac{q(q_o)^2}{\exp(2p/p_o)}} + T_{sat}$$

where $p_o = 8.7$ MPa, $q_o = 227$ m/(MW$^{0.5}$), p is in MPa and q is in MW/m^2. The point $T_{w,D-B} = T_{w,Thom}$ is the axial location at which the onset of nucleate boiling will occur. This correlation was used with a given `boilEulerFoam` run using the following input parameters. Two sets of results show the z_{ONB} values for the previous and current versions of the solver.

Table 2.2: z_{ONB} **Dittus-Boelter Correlation Parameters**

Parameter	**OF Variable**	**Value**
Inlet Pressure	p	4.5 MPa
Inlet Velocity	Uwater	1.0 m/s
Reynolds Number	Re	2.6×10^5
Inlet Sub-cooling	Tsub	26.6^o C
Inlet Void	alphaair	0
Wall Heat Flux	qWall	3.2×10^2 kW/m^2
Correlation	z_{ONB}	*0.64 m*
Calculated (Old)	z_{ONB}	*1.5 m*
Calculated (New)	z_{ONB}	*1.48 m*

From Table 2.2, the updated version of `boilEulerFoam` predicts boiling to occur much later than that of the Dittus-Boelter and Thom correlation, although there is a slight improvement of 0.02 m from the old version of `boilEulerFoam`. This discrepancy in z_{ONB} could be due to the boiling model present in `boilEulerFoam`, and it is recommended either that this model is investigated or other correlations for z_{ONB} are also examined.

2.2 Adiabatic Multiphase Equations for the Forward Problem

For adjoint development, this thesis is primarily concerned with the incompressible adiabatic system; therefore, the equations of state and the energy equations are removed from the forward problem. For reference, the term "forward" is used to describe the problem set for which the adjoint capability is developed, as opposed to the term "adjoint" which describes the problem set used for functional response calculations. Making the system adiabatic dramatically reduces the complexity of the forward system while still furthering the development of technology with respect to adjoint capability. The adiabatic case is constructed from the previously demonstrated `boilEulerFoam` such that wall boiling, interphase mass transfer, and energy dependence are removed. The physical properties in adiabatic `boilEulerFoam` are constant.

2.2.1 Discretization of the Adiabatic Multiphase Equations

This section describes in detail the discretization of the forward multiphase equations in `boilEulerFoam` and its solution algorithm. The previously written momentum equations are elliptic when seeking a steady solution [53]. The gradient source term makes them weakly dependent on pressure, and splitting of operators is required to couple the linearized momentum equations with pressure. These section develops the Pressure Implicit Splitting of Operators (PISO) algorithm as developed by Issa [51]. PISO is developed from implicit Navier-Stokes solution methods and a predictor-corrector methodology for calculating velocity. Issa demonstrates the accuracy of PISO methods as well as motivates stability based on the implicit solution of field variables, although the specific nature of stability is dependent on the exact discretization and interpolation scheme used. Issa does motivate a general stability for the velocity calculated using the PISO specific predictor-corrector scheme that has a time-step dependent error amplicfication term less than unity. This is however for a single-phase, compressible system, and Issa notes that increasing the nonlinearity of the system will affect the restriction of time-step size.

Continuity Equation Discretization

Ignoring evaporation and condensation, the phasic continuity equation is written as

$$\frac{\partial(\alpha_k \rho_k)}{\partial t} + \nabla \cdot (\alpha_k \rho_k \vec{u}_k) = 0$$

where k denotes either the l liquid phase or g vapor phase. Rewriting the equation, we have

$$\rho_k \frac{\partial \alpha_k}{\partial t} + \alpha_k \frac{\partial \rho_k}{\partial t} + (\alpha_k \vec{u}_k) \cdot \nabla \rho_k + \rho_k \nabla \cdot (\alpha_k \vec{u}_k) = 0$$

Assuming incompressibility such that

$$\alpha_k \left(\frac{\partial \rho_k}{\partial t} + \vec{u}_k \cdot \nabla \rho_k \right) = \alpha_k \frac{D\rho_k}{Dt} = 0$$

and dividing through by ρ_k, the continuity equation for phase k can be written as

$$\frac{\partial \alpha_k}{\partial t} + \nabla \cdot (\alpha_k \vec{u}_k) = 0$$

Introducing the total and relative velocities

$$\vec{u}_t = \alpha_l \vec{u}_l + \alpha_g \vec{u}_g$$
$$\vec{u}_r = \vec{u}_g - \vec{u}_l$$

the velocity for the gaseous phase can be written as

$$\begin{aligned} \vec{u}_g = &\quad \vec{u}_r + \vec{u}_l \\ \vec{u}_g = &\quad \vec{u}_r + \left(\frac{\vec{u}_t}{\alpha_l} - \frac{\alpha_g}{\alpha_l} \vec{u}_g \right) \\ \alpha_l \vec{u}_g = &\quad \alpha_l \vec{u}_r + \vec{u}_t - \alpha_g \vec{u}_g \\ (\alpha_l + \alpha_g)\, \vec{u}_g = &\quad \alpha_l \vec{u}_r + \vec{u}_t \end{aligned}$$

Noting that $\alpha_g + \alpha_l = 1$, we then have for the gaseous phase velocity

$$\vec{u}_g = \alpha_l \vec{u}_r + \vec{u}_t$$

Similarly for the liquid phase velocity

$$\begin{aligned} \vec{u}_l = &\quad -\vec{u}_r + \vec{u}_g \\ \vec{u}_l = &\quad -\alpha_g \vec{u}_r + \vec{u}_t \end{aligned}$$

Substituting this into the continuity equation for each phase

$$\frac{\partial \alpha_g}{\partial t} + \nabla \cdot (\alpha_g \, (\alpha_l \vec{u}_r + \vec{u}_t)) = 0$$

$$\frac{\partial \alpha_l}{\partial t} + \nabla \cdot (\alpha_l \, (-\alpha_g \vec{u}_r + \vec{u}_t)) = 0$$

Adding the continuity equations together

$$\frac{\partial(\alpha_g + \alpha_l)}{\partial t} + \nabla \cdot (\alpha_g \alpha_l \vec{u}_r - \alpha_g \alpha_l \vec{u}_r) + \nabla \cdot ((\alpha_g + \alpha_l)\, \vec{u}_t) = 0$$

$$\frac{\partial(1)}{\partial t} + \nabla \cdot (\vec{u}_t) = 0$$

$$\nabla \cdot \vec{u}_t = 0$$

Therefore, for the incompressible system of equations, we have the following constraints

$$\frac{D\rho_k}{Dt} = 0$$

$$\nabla \cdot \vec{u}_t = 0$$

For finite volume methods, the spatial domain is broken into control volumes or cells. We represent these control volumes as ΔV_i with $i \in (0, 1, \ldots, N)$ where N is the total number of cells. Each cell has a set of associated face area vectors $\vec{S_f}$. For a two dimensional structured mesh, each cell (labeled P) will have only four face area vectors associated with four neighbors (labeled N, S, E, and W after North, South, East, and West) and $f \in (1, \ldots, 4)$. A sample mesh cell can be found in Figure 2.9. The EE cell is also shown for reference, and this Figure is also used in the discussion of cell face interpolation. This P cell along with its four cell neighbors is called a compact computational molecule.

The continuity equation for the gaseous phase is integrated over a given control volume or cell ΔV. For simplicity, the cell reference subscript i has been removed for the rest of the derivation.

$$\int_{\Delta V} \frac{\partial \alpha_g}{\partial t} dV + \int_{\Delta V} \nabla \cdot (\alpha_g \vec{u}_t)\, dV + \int_{\Delta V} \nabla \cdot (\alpha_g \alpha_l \vec{u}_r)\, dV = 0$$

Writing Gauss's law

$$\int_{\Delta V} (\nabla \cdot \psi)\, dV = \int_{\delta S} (\psi \cdot \vec{n})\, dS$$

where δS is the surface of ΔV and $\vec{n}$ is the outward normal of δS.

For a two dimensional co-located structured mesh, discretized phase fractions α_k, velocity vectors $\vec{u}_k$, pressure p, liquid phase turbulent kinetic energy k_l, and liquid phase turbulent dissipation ε_l are solved at the cell centers given by the centroid of the control volume ΔV. The

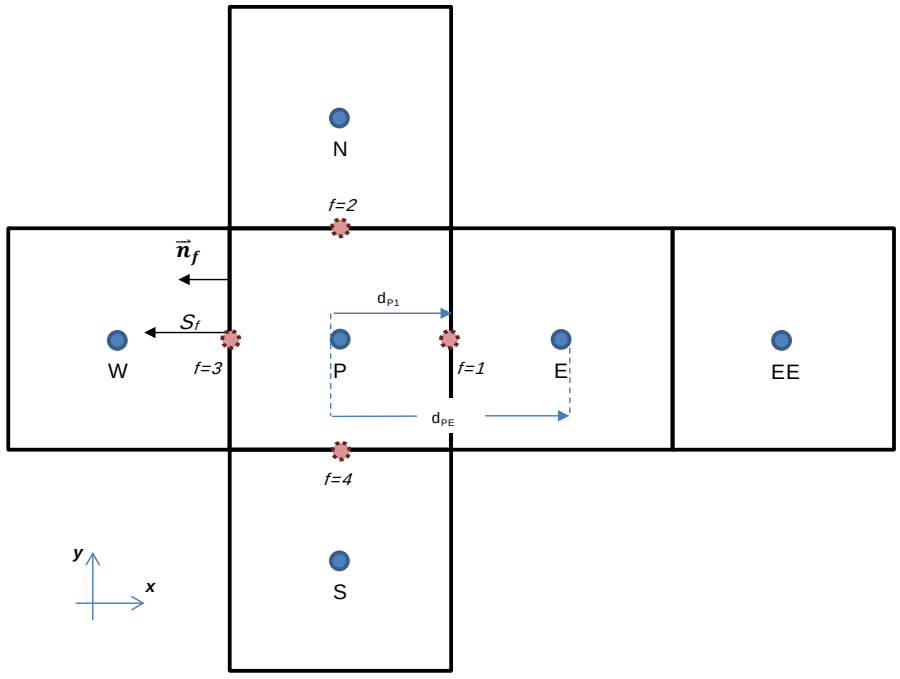


Figure 2.9: Two dimensional co-located mesh showing neighbor and face indexing and spacing for a given point P of interest

face volumetric flux $\phi_{k,f}$ is defined as

$$\phi_{k,f} = \vec{S}_f \cdot \vec{u}_{kf}$$

where $\vec{u}_{kf}$ is the face interpolated velocity. Since Gausses law requires integrating around cell surfaces, OpenFOAM calculates divergence and Laplacian terms using the face centered flux of each cell. Various methods of face interpolation are available in OpenFOAM, and a detailed analysis of the ones implemented in `boilEulerFoam` can be found in section 2.2.2

Using the aforementioned integrations, the following approximations are made for the integrated terms in the vapor phase continuity equation

$$\int_{\Delta V} \nabla \cdot (\alpha_g \vec{u}_t)\, dV = \int_{\delta S} \vec{n} \cdot (\alpha_g \vec{u}_t)\, dS \approx \sum_f \alpha_g \phi_{t,f}$$

$$\int_{\Delta V} \nabla \cdot (\alpha_g \alpha_l \vec{u}_r)\, dV = \int_{\delta S} \vec{n} \cdot (\alpha_g \alpha_l \vec{u}_r)\, dS \approx \sum_f \alpha_g \alpha_l \phi_{r,f}$$

$$\int_{\Delta V} \frac{\partial \alpha_g}{\partial t} dV \approx \frac{\partial \alpha_g}{\partial t} \Delta V$$

For a given time interval Δt, the time discretization is performed by evaluating values at old times n such that $\psi^n = \psi(t_n)$ and new times $n+1$ such that $\psi^{n+1} = \psi(t_n + \Delta t)$. The Euler implicit time differencing scheme is written as

$$\frac{\partial}{\partial t} \int_{\Delta V} \psi dV \bigg|_{t_n + \Delta t} \approx \frac{(\psi \Delta V)^{n+1} - (\psi \Delta V)^n}{\Delta t}$$

If the mesh size is constant with respect to time, then ΔV can be factored out of the time discretization term.

The solution algorithm builds a coefficient matrix with respect to α_g^{n+1} and solves for updated time values. The end result is a matrix of coefficients operating on α_g^{n+1} discretized variables and will have the form

$$a_{ii} \alpha_i^{n+1} + \sum_{j \neq i} a_{ij} \alpha_j^{n+1} = S_i$$

where i, j denote the row and column of the matrix of discretized coefficients, and S is the source term. All finite volume operators in OpenFOAM construct linear equations of this form. The superscripts n and $n+1$ are used to denote previous time and current time values. Variables without a superscript are assumed to have been calculated during previous times or iterations. This linearized set of coefficients for α_g^{n+1} is written as

$$\frac{\Delta V}{\Delta t} \alpha_g^{n+1} - \frac{\alpha_g \Delta V}{\Delta t} + \sum_f \left(\phi_t \alpha_g^{n+1} \right)_f + \sum_f \left(\alpha_l \phi_r \alpha_g^{n+1} \right)_f = 0$$

OpenFOAM's discrete solution algorithm uses a Pressure Implicit Splitting of Operators method to create an artificial pressure equation which helps enforce continuity. As a result, an

additional pressure equation is manufactured and the liquid continuity equation is not solved. Continuous phase void fraction is instead simply given by the expression

$$\alpha_l^{n+1} = 1 - \alpha_g^{n+1}$$

Momentum Equation Discretization

The momentum equation for phase k is written as

$$\frac{\partial \alpha_k \rho_k \vec{u}_k}{\partial t} + \left(\vec{u}_k \cdot \nabla\right) \alpha_k \rho_k \vec{u}_k = -\alpha_k \nabla p + \nabla \cdot \left(\alpha_k \boldsymbol{\tau}_k^{\text{eff}}\right) + \alpha_k \rho_k \vec{g} + \vec{M}_k$$

where $\boldsymbol{\tau}_k^{\text{eff}}$ is the total effective Reynolds stress for phase k. This term is expressed as

$$\boldsymbol{\tau}_k^{\text{eff}} = \rho_k \nu_k^{\text{eff}} \left(\nabla \vec{u}_k + \left(\nabla \vec{u}_k\right)^T - \frac{2}{3}\mathbf{I}\nabla \cdot \vec{u}_k\right) - \frac{2}{3}\mathbf{I}\rho_k k_k$$

Re-writing the momentum equation time and convection derivatives produces

$$\begin{aligned}
&\rho_k \frac{\partial \alpha_k \vec{u}_k}{\partial t} + \alpha_k \vec{u}_k \frac{\partial \rho_k}{\partial t} + \alpha_k \vec{u}_k \left(\vec{u}_k \cdot \nabla\right) \rho_k + \rho_k \left(\vec{u}_k \cdot \nabla\right) \alpha_k \vec{u}_k \\
&+ \alpha_k \nabla p - \nabla \cdot \left(\alpha_k \boldsymbol{\tau}_k^{\text{eff}}\right) - \alpha_k \rho_k \vec{g} - \vec{M}_k = 0
\end{aligned}$$

$$\begin{aligned}
&\alpha_k \vec{u}_k \left(\frac{\partial \rho_k}{\partial t} + \vec{u}_k \cdot \nabla \rho_k\right) + \rho_k \frac{\partial \alpha_k \vec{u}_k}{\partial t} + \rho_k \left(\vec{u}_k \cdot \nabla\right) \alpha_k \vec{u}_k + \\
&\alpha_k \nabla p - \nabla \cdot \left(\alpha_k \boldsymbol{\tau}_k^{\text{eff}}\right) - \alpha_k \rho_k \vec{g} - \vec{M}_k = 0
\end{aligned}$$

Using the incompressibility assumption and dividing through by ρ_k results in the following expression

$$\frac{\partial \alpha_k \vec{u}_k}{\partial t} + \left(\vec{u}_k \cdot \nabla\right) \alpha_k \vec{u}_k + \frac{\alpha_k}{\rho_k} \nabla p - \frac{1}{\rho_k} \nabla \cdot \left(\alpha_k \boldsymbol{\tau}_k^{\text{eff}}\right) - \alpha_k \vec{g} - \frac{\vec{M}_k}{\rho_k} = 0$$

The effective phasic stress $\boldsymbol{\tau}_k^{\text{eff}}$ is written explicitly as

$$\frac{1}{\rho_k}\nabla\cdot\left(\alpha_k\rho_k\nu_k^{\text{eff}}\left(\nabla\vec{u}_k+(\nabla\vec{u}_k)^T-\frac{2}{3}\mathbf{I}\nabla\cdot\vec{u}_k\right)-\frac{2}{3}\mathbf{I}\alpha_k\rho_k k_k\right)$$

$$=\frac{1}{\rho_k}\left[\begin{array}{l}\rho_k\nabla\cdot\left(\alpha_k\nu_k^{\text{eff}}\left(\nabla\vec{u}_k+(\nabla\vec{u}_k)^T-\frac{2}{3}\mathbf{I}\nabla\cdot\vec{u}_k\right)-\frac{2}{3}\mathbf{I}\alpha_k k_k\right)\\+\left(\alpha_k\nu_k^{\text{eff}}\left(\nabla\vec{u}_k+(\nabla\vec{u}_k)^T-\frac{2}{3}\mathbf{I}\nabla\cdot\vec{u}_k\right)-\frac{2}{3}\mathbf{I}\alpha_k k_k\right)\nabla\cdot\rho_k\end{array}\right]$$

Assuming that $\nabla\cdot\rho_k=0$, the effective stress becomes

$$\nabla\cdot\left(\alpha_k\nu_k^{\text{eff}}\nabla\vec{u}_k\right)+\nabla\cdot\left(\alpha_k\nu_k^{\text{eff}}\left[(\nabla\vec{u}_k)^T-\frac{2}{3}\mathbf{I}\nabla\cdot\vec{u}_k\right]\right)-\nabla\cdot\left(\alpha_k\frac{2}{3}\mathbf{I}k_k\right)$$

The interphase momentum transfer considered by this thesis includes drag, lift, virtual mass, turbulent dispersion, and wall lubrication. The drag force opposes the motion of the vapor through the fluid and is opposite to the liquid's relative velocity. It is written as

$$\vec{M}_g^D=-C_D\frac{3}{4}\frac{\rho_l}{D_S}\alpha_g\left|\vec{u}_r\right|\vec{u}_r$$

where $|\vec{u}_r|$ is the magnitude of the relative velocity and C_D is a drag coefficient based on a given momentum transfer closure relationship. There are formulations of drag available to OpenFOAM that attempt to minimize the value when dispersed phase void fraction is small by including the phase fractions for both phases [14] [41] [52]. This phase treatment is also present in the lift and virtual mass forces. Lumping terms into a single constant C_{dr}, we have

$$\vec{M}_g^D=-C_{dr}\left(\vec{u}_g-\vec{u}_l\right)$$
$$\vec{M}_l^D=-C_{dr}\left(\vec{u}_l-\vec{u}_g\right)$$

with $C_{dr}=C_D\rho_l\frac{3}{4}\frac{\alpha_g}{D_S}|\vec{u}_r|$. The drag term is treated "semi-implicitly" such that opposite phase velocity terms are moved to the pressure equation. The remaining terms are treated to either increase the diagonal dominance of the coefficient matrix or reduce the source term which promotes stability.

The lift interphase momentum transfer term acting on the vapor phase is written as

$$\vec{M}_g^L=-C_L\alpha_g\vec{u}_r\times\nabla\times\vec{u}_t$$

The curl in OpenFOAM is calculated using the following identity

$$\nabla \times \vec{a} = 2\left(\star \text{skew} \nabla \vec{a}\right) = \star\left(\nabla \vec{a} - \left(\nabla \vec{a}\right)^T\right)$$

where the $\star$ operator is the Hodge Dual of the skew-symmetric matrix. The Hodge Dual is defined as

$$\star \begin{bmatrix} 0 & a_3 & -a_2 \\ -a_3 & 0 & a_1 \\ a_2 & -a_1 & 0 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

or for a two dimensional matrix, as is the case for `boilEulerFoam`, the Hodge Dual is

$$\star \begin{bmatrix} 0 & -a \\ a & 0 \end{bmatrix} = a$$

The curl and cross product expressions in the lift term are calculated explicitly using velocity values from the previous time step. The lift constant C_l is typically dependent on bubble size and deformation, but for this thesis C_l remains constant.

Cross products for 2D geometry are not well defined. The 2D cross product analogue used in `boilEulerFoam` takes one argument, $\vec{u}_r$, and returns the appropriate orthogonal vector in 2D space, $\vec{u}_{\perp r}$, such that $\vec{u}_r \times = [u_{r,1}\ u_{r,2}]^T \times = [u_{r,2} \quad -u_{r,1}]^T$.

The virtual mass interphase momentum transfer term acting on the vapor phase is written as

$$\vec{M}_g^{VM} = -C_{vm}\alpha_g\alpha_l\rho_l\left(\frac{D\vec{u}_g}{Dt} - \frac{D\vec{u}_l}{Dt}\right)$$

where $\dfrac{D}{Dt}$ is the material derivative, defined again for reference as

$$\frac{D\vec{u}_k}{Dt} = \frac{\partial \vec{u}_k}{\partial t} + \vec{u}_k \cdot \nabla \vec{u}_k$$

Virtual mass is also determined semi-implicitly such that matrix coefficients are either added to the diagonal if they are positive or subtracted from the source if they are negative in an attempt to improve stability.

The turbulent dispersion interphase momentum transfer term acting on the dispersed phase is written as

$$\vec{M}_g^T = -C_{td}\nabla\alpha_g$$

where C_{td} is calculated as a function of turbulent kinetic energy, density, and relative velocity. For this simplified case, C_{td} is written as

$$C_{td} = (0.01)k_l\rho_l$$

The wall lubrication interphase momentum transfer term acting on the dispersed phase is written as

$$\vec{M}_g^{WL} = C_W\alpha_g\rho_l\left|\vec{u}_r - (\vec{u}_r\cdot\vec{n}_w)\,\vec{n}_w\right|^2$$

The expression $|\vec{u}_r - (\vec{u}_r\cdot\vec{n}_w)\,\vec{n}_w|$ returns the projection of the relative velocity parallel to the wall of the pipe. For simplicity, C_W is constant for the adiabatic case.

The solution method in `boilEulerFoam` calculates a "turbulent pressure" rather than an actual pressure such that $p_t = \frac{2}{3}\rho_k k_k$. Therefore, the turbulent kinetic energy term present in the Reynolds stress is removed from the momentum equation and $p = p_t$ is assumed. In order to apply Gauss's theorem, the momentum advection term is rewritten in terms of a dyadic product of vectors such that $(\vec{u}_k\cdot\nabla)\,\alpha_k\vec{u}_k = \nabla\cdot(\alpha_k\vec{u}_k\otimes\vec{u}_k)$, where $\otimes$ designates the dyadic product. Ignoring the pressure term and the gravity term and integrating over a cell volume as before with the continuity equations, we have

$$\begin{aligned}
&\frac{\partial}{\partial t}\int_{\Delta V}\alpha_g\vec{u}_g dV + \int_{\delta S}\vec{n}\cdot(\alpha_g\vec{u}_g\otimes\vec{u}_g)\,dS - \int_{\delta S}\vec{n}\cdot\left(\alpha_g\nu_g^{\text{eff}}\nabla\vec{u}_g\right)dS \\
&+\int_{\delta S}\vec{n}\cdot\left(\alpha_g\nu_g^{\text{eff}}\left[(\nabla\vec{u}_g)^T - \frac{2}{3}\mathbf{I}\nabla\cdot\vec{u}_g\right]\right)dS + \int_{\Delta V}(-C_{dr}\vec{u}_g)\,dV \\
&+\int_{\Delta V}\frac{\vec{M}_g}{\rho_g}dV = 0
\end{aligned}$$

The discretized momentum equations are linearized with respect to $\vec{u}_k^{n+1}$, and the coefficients operating on the dispersed phase variable can be written as

$$
\begin{aligned}
&\frac{\left(\alpha_g^{n+1}\Delta V\right)}{\Delta t}\vec{u}_g^{n+1} - \frac{\alpha_g\vec{u}_g\Delta V}{\Delta t} - \left(\frac{\alpha_g^{n+1}\Delta V}{\Delta t} - \frac{\alpha_g\Delta V}{\Delta t}\right)\vec{u}_g^{n+1} \\
&+ \sum_f \left(\phi_g\alpha_g\vec{u}_g^{n+1}\right)_f - \sum_f \left(\phi_g\alpha_g\right)_f \vec{u}_g^{n+1} \\
&- \sum_f \left(\vec{S}_f\alpha_g\nu_g^{\text{eff}}\right)_f \left(\nabla\vec{u}_g^{n+1}\right)_f \\
&+ \sum_f \left[\vec{S}_f\alpha_{g,f}\left(\nu_{g,f}^{\text{eff}}\, trace\left(\left(\nabla\vec{u}_g\right)_f^T\right)\right)\cdot\frac{2}{3}\mathbf{I} - \nu_{g,f}^{\text{eff}}\left(\nabla\vec{u}_g\right)_f^T\right] \\
&+ \sum_i \frac{\vec{M}_{g,i}}{\rho_g} = 0
\end{aligned}
$$

with the discretized interphase momentum transfer terms written as

$$
\begin{aligned}
&-\left(\frac{1}{\rho_g}C_{dr}\vec{u}_g\Delta V\right) + \left[C_l\frac{\rho_t}{\rho_g}\alpha_g\alpha_l\Delta V\begin{bmatrix}\frac{\partial u_{t,2}}{\partial x_1} & \frac{\partial u_{t,1}}{\partial x_2}\end{bmatrix}\begin{bmatrix}u_{r,2}\\ -u_{r,1}\end{bmatrix}\right] \\
&- C_{vm}\frac{\rho_l}{\rho_g}\alpha_g\alpha_l\begin{bmatrix}\frac{(\Delta V)}{\Delta t}\vec{u}_g^{n+1} - \frac{\left(\vec{u}_g^{n}\Delta V\right)}{\Delta t} + \sum_f\left(\phi_g\vec{u}_g^{n+1}\right)_f \\ -\frac{\left(\vec{u}_l^{n}\Delta V\right) - \left(\vec{u}_l^{n-1}\Delta V\right)}{\Delta t} - \sum_f\left(\phi_l\right)_f\vec{u}_l\end{bmatrix} \\
&- C_{td}\nabla\alpha_g + C_W\alpha_g\rho_l\left|\left(\vec{u}_r - \left(\vec{u}_r\cdot\vec{n}_w\right)\vec{n}_w\right|^2\right.
\end{aligned}
$$

The $\left(\frac{\alpha_g^{n+1}\Delta V}{\Delta t} - \frac{\alpha_g\Delta V}{\Delta t}\right)\vec{u}_g^{n+1} - \sum_f\left(\phi_g\alpha_g\right)_f\vec{u}_g^{n+1}$ term is referred to in OpenFOAM as the "compressibility correction" and is present to help dampen sensitivities of the momentum equation with respect to changes in α_k.

For the continuous phase, the discretized momentum equation is

$$
\begin{aligned}
&\frac{\left(\alpha_l^{n+1}\Delta V\right)}{\Delta t}\vec{u}_l^{n+1} - \frac{\left(\alpha_l\vec{u}_l\Delta V\right)}{\Delta t} - \left(\frac{\alpha_l\Delta V}{\Delta t} - \frac{\alpha_l\Delta V}{\Delta t}\right)\vec{u}_l^{n+1} \\
&+ \sum_f \left(\phi_l\alpha_l\vec{u}_l^{n+1}\right)_f - \sum_f \left(\phi_g\alpha_g\right)_f \vec{u}_l^{n+1} \\
&- \sum_f \left(\vec{S}_f\alpha_{l,f}\nu_l^{\text{eff}}\right)_f \left(\nabla\vec{u}_l\right)_f^{n+1} \\
&+ \sum_f \left[\vec{S}_f\alpha_l\left(\nu_l^{\text{eff}}\, trace\left(\left(\nabla\vec{u}_l\right)_f^T\right)\right)\cdot\frac{2}{3}\mathbf{I} - \nu_l^{eff,n+1}\left(\nabla\vec{u}_l\right)_f^T\right] \\
&+ \sum_i \frac{\vec{M}_{l,i}}{\rho_g} = 0
\end{aligned}
$$

with the discretized interphase momentum transfer terms written as

$$
\begin{aligned}
&- \left(C_{dr}\vec{u}_l\Delta V\right) - \left[C_l\rho_t\alpha_g\alpha_l\Delta V\begin{bmatrix}\frac{\partial u_{t,2}}{\partial x_1} & \frac{\partial u_{t,1}}{\partial x_2}\end{bmatrix}\begin{bmatrix}u_{r,2}\\ -u_{r,1}\end{bmatrix}\right] \\
&+ C_{vm}\frac{\rho_l}{\rho_g}\alpha_g\alpha_l\begin{bmatrix}\frac{\left(\Delta V\right)}{\Delta t}\vec{u}_l^{n+1} - \frac{\left(\vec{u}_l^n\Delta V\right)}{\Delta t} + \sum_f\left(\phi_l\vec{u}_l^{n+1}\right)_f \\ -\frac{\left(\vec{u}_g^n\Delta V\right) - \left(\vec{u}_g^{n-1}\Delta V\right)}{\Delta t} - \sum_f\left(\phi_g\right)_f\vec{u}_g\end{bmatrix} \\
&+ C_{td}\nabla\alpha_g - C_W\alpha_g\rho_g\left|\left(\vec{u}_r - \left(\vec{u}_r\cdot\vec{n}_w\right)\vec{n}_w\right|^2\right.
\end{aligned}
$$

Following this formulation, the momentum equation is a linear system with respect to $\vec{u}_k^{n+1}$, and the discretized momentum equation can be written generally as

$$\mathbf{A}\vec{u}_k^{n+1} = \vec{R}$$

where $\mathbf{A}$ is the matrix containing the discretized coefficients operating on cell centered velocities for the momentum equation. An approximate solution for $\vec{u}_k^{n+1}$ can be constructed such that

$$\vec{u}_k^{n+1} \approx \mathbf{A}_D^{-1}\vec{R} - \mathbf{A}_D^{-1}\mathbf{A}_H\vec{u}_k^n$$

where $\mathbf{A}_D$ is the matrix of diagonal coefficients of $\mathbf{A}$ and $\mathbf{A}_H$ is the matrix of off-diagonal coefficients of $\mathbf{A}$.

Pressure Equation Discretization

The splitting of operators and pressure equation formulation is performed according to the PISO algorithm methodology, a derivative of the predictor-corrector scheme for velocity. This methodology predicts the phase flux ϕ_k^* and the velocity $\vec{u}_k^*$ according to

$$\vec{u}_k^* = \mathbf{A}_{Dk}^{-1}\vec{R}_k - \mathbf{A}_{Dk}^{-1}\mathbf{A}_{Hk}\vec{u}_k^n$$

$$\phi_k^* = \vec{S}_f \cdot \vec{u}_{k,f}^*$$

where $\vec{u}_{k,f}^*$ is the predicted velocity interpolated to the cell face. Although this is the typical formulation for the flux prediction, in OpenFOAM there is a flux correction term present such that ϕ_k^* is determined by

$$\phi_k^* = \vec{S}_f \cdot \vec{u}_{k,f}^* + \phi_k^d \left[1 - \min\left(\frac{|\phi_k^d|}{|\phi_k^n| + \epsilon},\ 1\right)\right] \frac{1}{\Delta t}$$

with

$$\phi_k^d = \left(\phi_k^n - \vec{S}_f \cdot \vec{u}_{k,f}^n\right)$$

where ϕ_k^n and $\vec{u}_{k,f}^n$ are the fluxes and velocities for the previous n time step. This expression attempts to correct flux predictions in order to balance values at the faces of a given cell. This term is not always necessary for the PISO algorithm, and other subsequent applications do not include it.

For the flux corrections, the matrix diagonal $\left(\mathbf{A}_D^{-1}\right)_f$ is interpolated so that it operates on cell face values. The flux is then corrected to include the gravity, pressure, and drag terms such that

$$\phi_g^{**} = \phi_g^* + \vec{S}_f \cdot \left(\left(\mathbf{A}_{Dg}^{-1}\right)_f \alpha_g \vec{g}_f\right) + \vec{S}_f \cdot \left((\mathbf{A}_{Dg}^{-1})_f \frac{C_{dr}}{\rho_g}\vec{u}_l\right)$$

$$\phi_l^{**} = \phi_l^* + \vec{S}_f \cdot \left(\left(\mathbf{A}_{Dl}^{-1}\right)_f \alpha_l \vec{g}_f\right) + \vec{S}_f \cdot \left((\mathbf{A}_{Dl}^{-1})_f \frac{C_{dr}}{\rho_l}\vec{u}_g\right)$$

The mixture flux includes both continuous and dispersed phases such that

$$\phi = \alpha_{g,f}\phi_g + \alpha_{l,f}\phi_l$$

We write the total flux using the corrected flux ϕ_k^{**} and the pressure term, producing the expression

$$\phi = \alpha_{g,f}\left[\phi_g^{**} - \vec{S}_f \cdot \left(\left(\mathbf{A}_{Dg}^{-1}\right)_f \frac{\alpha_{g,f}}{\rho_g}\nabla p\right)\right] + \alpha_{l,f}\left[\phi_l^{**} - \vec{S}_f \cdot \left(\left(\mathbf{A}_{Dl}^{-1}\right)_f \frac{\alpha_{l,f}}{\rho_l}\nabla p\right)\right]$$

Enforcing the continuity constraining $\nabla \cdot \phi = 0$ results in the following pressure equation

$$\nabla \cdot \phi^{**} - \nabla \cdot \left[S_f \cdot \left(\alpha_{g,f}\left(\mathbf{A}_{Dg}^{-1}\right)_f \frac{\alpha_{g,f}}{\rho_g} + \alpha_{l,f}\left(\mathbf{A}_{Dl}^{-1}\right)_f \frac{\alpha_{l,f}}{\rho_l}\right)\nabla p\right] = 0$$

Integrating over a cell volume and applying Gauss's law, we have an integrated pressure equation that is then solved for a discretized pressure field p^*

$$\int_{\delta S} \phi^{**} dS - \int_{\delta S} S_f \left(\alpha_{g,f}\left(\mathbf{A}_{Dg}^{-1}\right)_f \frac{\alpha_{g,f}}{\rho_g} + \alpha_{l,f}\left(\mathbf{A}_{Dl}^{-1}\right)_f \frac{\alpha_{l,f}}{\rho_l}\right)\nabla p^* \; dS = 0$$

$$\sum_f S_f \phi^{**} - \sum_f S_f S_f \left(\alpha_{g,f}\left(\mathbf{A}_{Dg}^{-1}\right)_f \frac{\alpha_{g,f}}{\rho_g} + \alpha_{l,f}\left(\mathbf{A}_{Dl}^{-1}\right)_f \frac{\alpha_{l,f}}{\rho_l}\right)\left(\nabla p^*\right)_f = 0$$

For the SIMPLE and PIMPLE methods, this pressure equation can undergo relaxation and be solved multiple times. A detailed description of this solution algorithm can be found in section 2.2.3. The velocity terms are then corrected using the following expression

$$\vec{u}_k^{**} = \vec{u}_k^{*} + \text{reconstruct}\left[\vec{S}_f \cdot \left(\left(\mathbf{A}_{Dk}^{-1}\right)_f \alpha_k \vec{g}_k + \left(\mathbf{A}_{Dk}^{-1}\right)_f \frac{C_{dr}}{\rho_k}\vec{u}_j^n - \left(\mathbf{A}_{Dk}^{-1}\right)_f \frac{\alpha_{k,f}}{\rho_k}\nabla p^*\right)\right]$$

where $\vec{u}_j$ denotes the flux for the alternate phase. The correction term is calculated at the face of the cell and reconstructed as a cell center velocity term using the following methodology

$$\text{reconstruct}(\vec{u}) = \left(\sum_f \frac{1}{|\vec{S}_f|}\vec{S}_f \otimes \vec{S}_f\right)^{-1} \cdot \left(\sum_f \frac{1}{|\vec{S}_f|}\vec{S}_f \phi_f\right)$$

The end result is then a corrected velocity, flux, and pressure field.

k-ϵ Equation Discretization

`BoilEulerFoam` uses a k-ϵ Reynolds averaged Navier-Stokes or RANS model. This model time-averages the Navier-Stokes equations for turbulent flows and ignores local eddy formation, considering instead the time averaged and fluctuating terms $\bar{u}$ and $\bar{u}'$. The resulting derivation requires closure models due to the Reynolds stress term $\tau = -\rho\bar{u}'\bar{u}'$ present in the RANS expression for momentum. For the k-ϵ model, closure relationships are derived for turbulent kinetic energy, or $k = (||\bar{u}'||_2)^2$, and the rate of turbulent dissipation, or $\epsilon = \frac{\nu}{2}\left(\nabla\bar{u}' + (\nabla\bar{u}')^T\right)$,

where turbulent dissipation can be thought of as the work done by the smallest eddies on the viscous stresses [53].

The turbulent viscosity ν_k^t for multiphase flow is equal to the bubble induced turbulent viscosities. The equation for turbulent viscosity for the continuous phase is as follows

$$\nu_l^t = C_\mu \frac{k_l^2}{\epsilon_l}$$

where $C_\mu = 0.09$. The effective viscosity for the continuous phase is then the summation of the continuous phase viscosity, the turbulent viscosity, and the shear induced viscosity and is written as

$$\nu_l^{\text{eff}} = \nu_l + \nu_l^t + \frac{1}{2} C_{\mu b} D_S \alpha_g |\vec{u}_g - \vec{u}_l|$$

where $C_{\mu b} = 1.2$ is the factor of enhanced turbulence in the continuous phase due to the dispersed phase, and D_S is the bubble diameter of the dispersed phase (set to 0.1 mm). The effective viscosity of the vapor phase is also dependent on the turbulent liquid viscosity and is given by

$$\nu_g^{\text{eff}} = \nu_g + C_t^2 \nu_l^t$$

where C_t is the turbulent response coefficient. This is often defined as the ratio of velocity fluctuations of the dispersed phase to the continuous phase, but for the adiabatic case, $C_t = 1$.

As mentioned previously, the turbulent kinetic energy and turbulent dissipation equations for the dispersed phase are ignored. The liquid k-ϵ equations are then

$$\frac{\partial \alpha_l k_l}{\partial t} + \nabla \cdot (\vec{u}_l \alpha_l k_l) - \nabla \cdot \left(\nu_l^{\text{eff}} \alpha_l \nabla k_l \right) = \alpha_l P_l - \alpha_l \epsilon_l$$

$$\frac{\partial \alpha_l \epsilon_l}{\partial t} + \nabla \cdot (\vec{u}_l \alpha_l \epsilon_l) - \nabla \cdot \left(\nu_l^{\text{eff}} \alpha_l \nabla \epsilon_l \right) = \alpha_l \frac{\epsilon_l}{k_l} \left(C_1 P_l - C_2 \epsilon_l \right)$$

where $C_1 = 1.44$ and $C_2 = 1.92$. The kinetic energy production term P_l is defined as

$$2\nu_l^t \left[\nabla \vec{u}_l \cdot \text{dev} \left(\frac{1}{2} \nabla \vec{u} + \frac{1}{2} (\nabla \vec{u})^T \right) \right]$$

where the deviatoric operation for a given 2×2 tensor $\mathbf{T}$ is

$$\text{dev}(\mathbf{T}) = \text{dev} \left(\begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} \right) = \begin{bmatrix} T_{11} - \frac{1}{3} T_{11} & T_{12} \\ T_{21} & T_{22} - \frac{1}{3} T_{22} \end{bmatrix}$$

Integrating over a cell volume and applying Gauss's law, we have for the turbulent kinetic energy equation

$$\frac{\partial}{\partial t}\int\limits_{\Delta V}\alpha_l k_l + \int\limits_{\delta S}\vec{n}\cdot\left(\vec{u}_l\alpha_l k_l\right)dS - \int\limits_{\delta S}\vec{n}\cdot\left(\nu_l^{\text{eff}}\alpha_l\nabla k_l\right) = \int\limits_{\Delta V}\left(\alpha_l P_l - \alpha_l\epsilon_l\right)dV$$

$$\begin{aligned}
&\frac{\left(\alpha_l^{n+1}\Delta V\right)}{\Delta t}k_l^{n+1} - \frac{\left(\alpha_l k_l\Delta V\right)}{\Delta t} - \left(\frac{\alpha_l^{n+1}\Delta V}{\Delta t} - \frac{\alpha_l^{n}\Delta V}{\Delta t}\right)k_l^{n+1} \\
&+ \sum_f\left(\phi_l^n\alpha_l k_l^{n+1}\right)_f - \sum_f\left(\phi_l\alpha_l\right)_f k_l^{n+1} \\
&- \sum_f\vec{S}_f\nu_l^{\text{eff}}\alpha_l^{n+1}\left(\nabla k_l\right)_f^{n+1} = \left[\Delta V\left(\alpha_l P_l - \alpha_l\epsilon_l\right)\right]
\end{aligned}$$

and for the turbulent dissipation equation

$$\begin{aligned}
&\frac{\left(\alpha_l^{n+1}\Delta V\right)}{\Delta t}\epsilon_l^{n+1} - \frac{\left(\alpha_l\epsilon_l\Delta V\right)}{\Delta t} - \left(\frac{\alpha_l^{n+1}\Delta V}{\Delta t} - \frac{\alpha_l^{n}\Delta V}{\Delta t}\right)\epsilon_l^{n+1} \\
&+ \sum_f\left(\phi_l^n\alpha_l\epsilon_l^{n+1}\right)_f - \sum_f\left(\phi_l\alpha_l\right)_f\epsilon_l^{n+1} \\
&- \sum_f\vec{S}_f\nu_l^{n,eff}\alpha_l^{n+1}\left(\nabla\epsilon_l\right)_f^{n+1} = \left(\alpha_l\frac{\epsilon_l}{k_l}C_1 P_l\Delta V\right) - \left(\alpha_l\frac{\epsilon_l}{k_l}C_2\epsilon_l\right)^{n+1}
\end{aligned}$$

2.2.2 Cell Face Interpolation Schemes

For co-located mesh configurations such as the one implemented in OpenFOAM, a face interpolation scheme is necessary in order to solve the finite volume discretized equations. OpenFOAM makes many interpolation schemes available to its users, such as simple linear interpolation or higher resolution total variation diminishing (TVD) schemes. The code inherited from `boilEulerFoam` makes use of TVD schemes in order to limit oscillations and insure stability for solutions with sharp variations in void fraction.

In section 2.2.1, expressions involving flux, divergence, Laplacian, and gradient operators require interpolation as designated by the subscript $_f$. The interpolation methods described in this section develop methods for these interpolated parameters and reference the previously shown 2D structured mesh pictured in Figure 2.9.

Two interpolation schemes are specified for calculating the divergence of convection terms.

These interpolation schemes are called `vanLeer` and `limitedLinear` in the OpenFOAM systems file `fvSchemes`. In order to derive these interpolation methods, we start with the general formula for an interpolated value θ_f at the east face of Figure 2.9 with $f = 1$ given by the following expression

$$\theta_{f=1} = \theta_P + \psi\,(\theta_E - \theta_P) \tag{2.1}$$

where ψ is defined according to the interpolation scheme. If linear interpolation is used, then ψ is given by

$$\psi = \frac{\vec{d}_{p1}}{\vec{d}_{PE}}$$

If spacing in the x direction is uniform, or $\vec{d}_{PE} = \vec{d}_{EE}$, then $\psi = \dfrac{1}{2}$ and θ_f is simply the average of cell P its face neighbor values. Linear interpolation is used to calculate face centered flux ϕ_f.

The `vanLeer` high resolution TVD scheme interpolates to the eastward face of Figure 2.9 using equation 2.1 with ψ defined by [47]

$$\psi(r) = \frac{1}{2}\left(\frac{r + |r|}{r + 1}\right)$$

and the upwinded flux splitting ratio given by

$$r = \frac{\theta_W - \theta_P}{\theta_P - \theta_E}$$

This interpolation scheme can help limit spurious oscillations in the void fraction across cell faces according to the TVD methodology [63].

The `limitedLinear` interpolation scheme is used in the interpolation of velocity to cell faces, and weights ψ calculated using linear interpolation with either 0 or 1. Since RANS pipe bulk flow convects mass in only one direction, `limitedLinear` simply uses upwind face values and $\psi = 1$.

The `vanLeer` interpolation scheme is used for the following discretized expressions

- α_g^{n+1} in the convection term $\sum_f (\phi_{t,f})^n\, \alpha_g^{n+1}$ and $\sum_f (\alpha_l \phi_{t,f})^n\, \alpha_g^{n+1}$ from the dispersed phase continuity equation
- α_l^n in the convection term $\sum_f (\alpha_l \phi_{t,f})^n$ from the dispersed phase continuity equation

- α_k^{n+1} in the convection term $\sum_f \phi_{k,f}^n \alpha_k^{n+1} \vec{u}_k^{n+1}$ from the momentum equations
- α_l^{n+1} in the convection terms $\sum_f \phi_{l,f}^n \alpha_l^{n+1} k_l^{n+1}$ and $\sum_f \phi_{l,f}^n \alpha_l^{n+1} \epsilon_l^{n+1}$ from the k-ϵ turbulence equations
- α_l^{n+1} from the continuity constraint for the pressure equation

The `limitedLinear` interpolation scheme is used for the following discretized expressions High Resolution Schemes Implementation in OpenFOAM

- $\vec{u}_k^{n+1}$ in $\sum_f \phi_{k,f}^n \alpha_k^{n+1} \vec{u}_k^{n+1}$ from the convection term in the momentum equations
- $\vec{u}_k^{n+1}$ in $\sum_f \phi_{k,f}^n \vec{u}_k^{n+1}$ from the virtual mass interphase momentum transfer term in the momentum equations
- ϵ_l^{n+1} in the convective term $\sum_f \phi_l^n \alpha_l^{n+1} \epsilon_l^{n+1}$ from the ϵ turbulence equation
- k_l^{n+1} in the convective term $\sum_f \phi_l^n \alpha_l^{n+1} k_l^{n+1}$ from the k turbulence equation

The general form for Laplacian terms is $\int_{\Delta V} \nabla \cdot (\nu \nabla \theta)\, dV \approx \sum_f \nu_f \vec{S}_f \cdot (\nabla \theta)_f$. Looking at only the eastward face in Figure 2.9, we have the following expression for the discretized Laplacian operator

$$\vec{S}_1 \cdot (\nabla \theta)_1 = |S_1| \frac{\theta_E - \theta_P}{|\vec{d_{PE}}|}$$

This equation is only valid if $\vec{n}_E$ is orthogonal to $\vec{S}_1$, which is the case for the structured mesh used in `boilEulerFoam`. The value ν_f in the discretized Laplacian expression is linear interpolated from the ν_P and ν_E values. This Laplacian scheme is used for the following discretized expressions

- $\sum_f \vec{S}_f \alpha_{k,f} \nu_{k,f}^{eff,n+1} (\nabla \vec{u}_k)_f^{n+1}$ from the discretized momentum equations
- The Laplacian term in the pressure equation. Note that the $\alpha_{k,f}^{n+1}$ terms are interpolated using the vanLeer scheme. $(\mathbf{A}_D)_f$ and the ν term are already a surface scalar field and have been interpolated using the default linear scheme.

2.2.3 Solution Algorithm Overview

The discretized Reynolds averaged Navier-Stokes equations are solved using a Pressure Implicit Splitting of Operators or PISO algorithm. This methodology guesses phasic velocity values that do not initially satisfy the continuity equation. A pressure equation is constructed independent from the momentum equations based on predicted fluxes that are forced to obey continuity. This pressure field is then used to correct the initial velocity prediction.

For unsteady and incompressible flows, the PISO algorithm uses an additional corrector step that includes a second pressure equation to account for compressibility effects where Mach numbers are close to or greater than 1. The fluid model considered by `boilEulerFoam` is incompressible, therefore the PISO algorithm resembles that of the Semi-Implicit Method for Pressure-Linked equations or the SIMPLE algorithm. SIMPLE is used for incompressible, steady-state equations that often require relaxation.

Figure 2.10 shows the pseudo-code for the PISO algorithm used by `boilEulerFoam`. First, the dispersed phase continuity equation is solved for α_g and the k-ϵ turbulence equations are solved for k_l and ϵ_l. The momentum equations are then constructed without gravity, pressure, and certain interphase momentum transfer terms. These split momentum equations are then used to predict $\vec{u}_k^*$ and ϕ_k^*.

The face volumetric flux ϕ_k^{**} is corrected once to include gravity and interphase momentum transfer terms previously removed from the momentum equations. The total flux ϕ is then constructed using the corrected fluxes ϕ_k^{**} and the pressure gradient previously removed from the momentum equations. This total flux is forced to obey continuity, resulting in an implicit equation for pressure. This pressure equation is solved for the pressure, p^*, and corrected velocities $\vec{u}_k^{**}$ are reconstructed using p^*, gravity, and interphase momentum transfer terms.

After pressure p^* and corrected velocity $\vec{u}_k^{**}$ are calculated, the PISO algorithm advances the time-step and the algorithm is continued for $t + \Delta t$. The SIMPLE algorithm, instead of advancing the time-step, iterates on the predictor-correction scheme until the system reaches steady state. Figure 2.11 shows a typical pseudo-code for the SIMPLE algorithm. According to this methodology, SIMPLE will relax solutions to the predictor-correction iterations until a normalized tolerance is reached for solutions to the flow equations. However, PISO assumes that for one predictor-correction iteration, the calculated pressure and velocity are time value correct, given a stable numerical configuration. For flow systems that have steady state solutions, there is no real difference between PISO and SIMPLE other than the relaxation that occurs in SIMPLE.

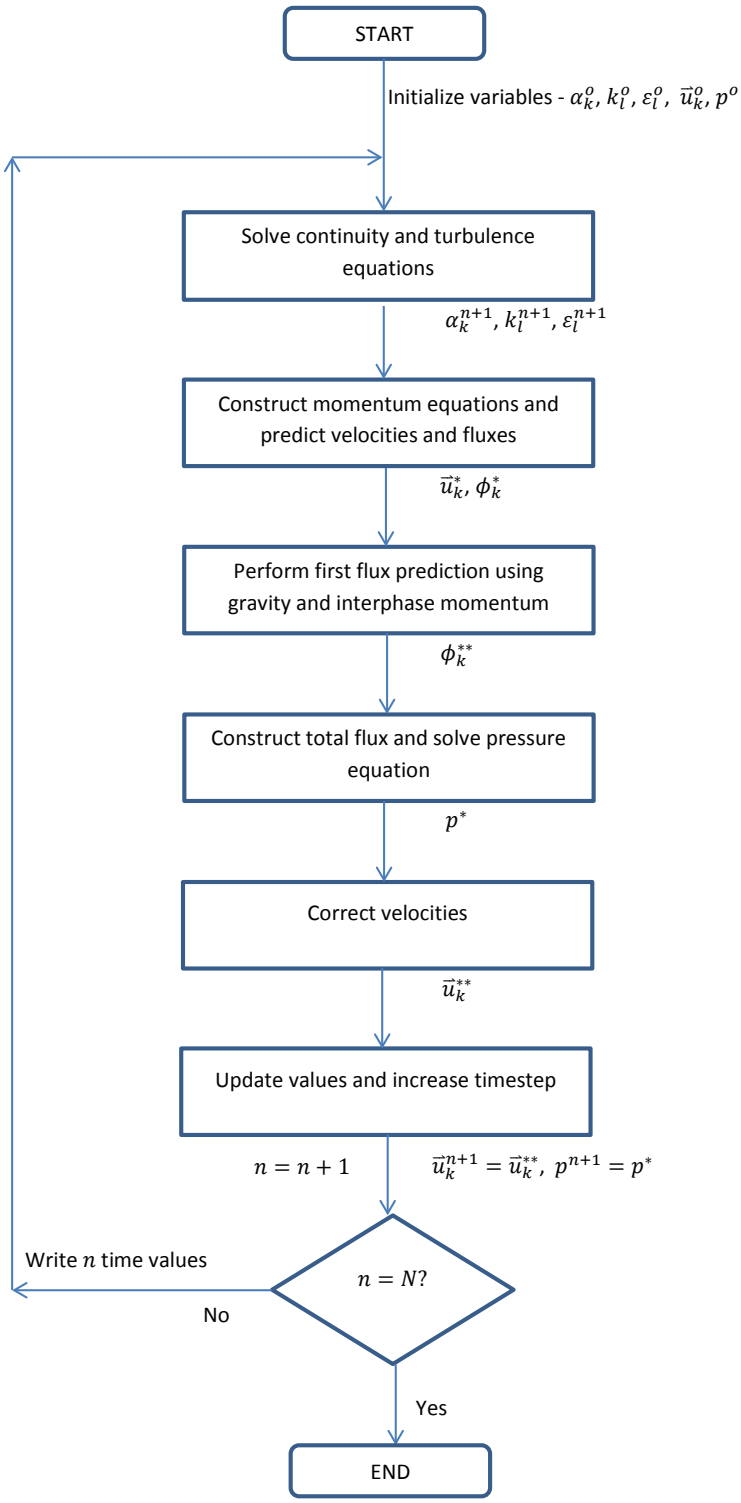


Figure 2.10: Pseudo-code for the Pressure Implicit Splitting of Operators or PISO algorithm

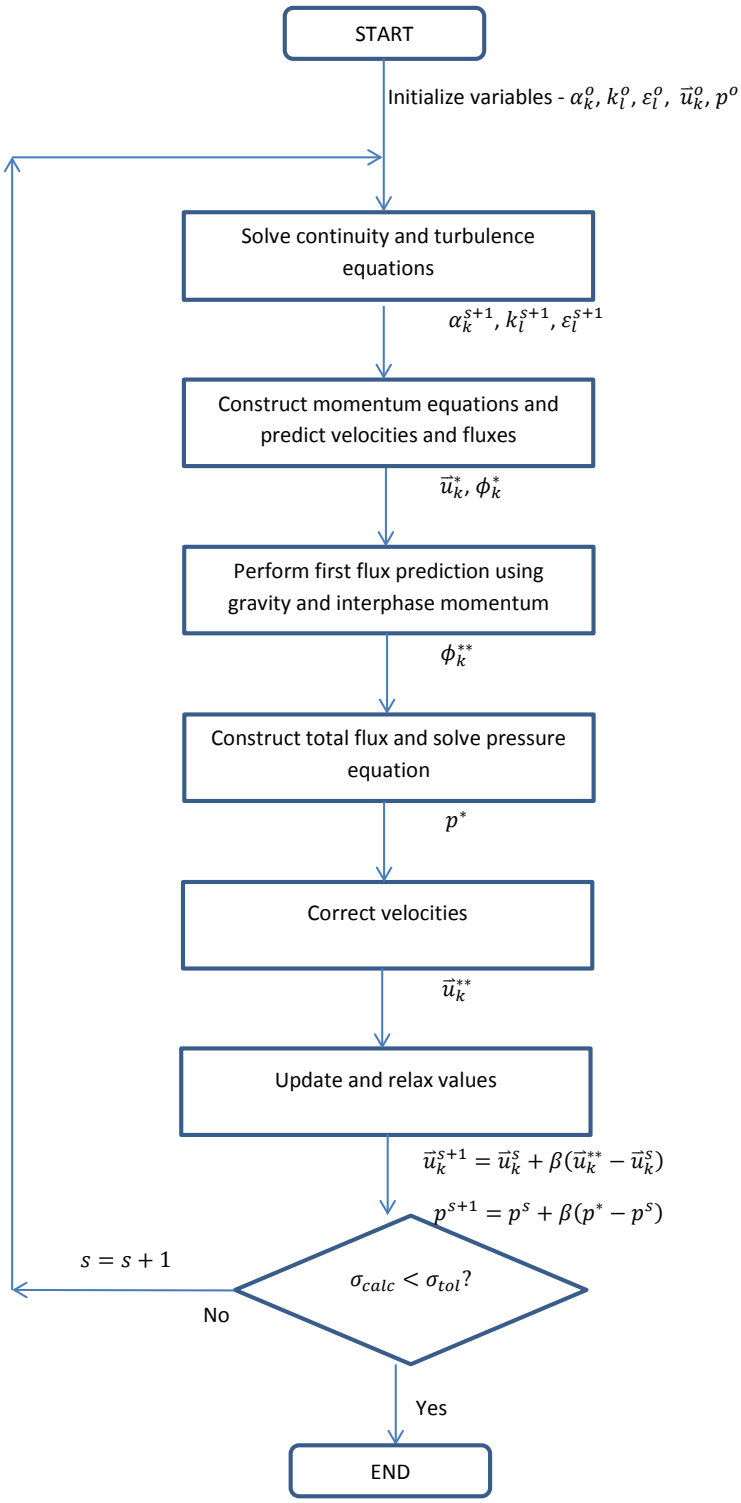


Figure 2.11: Pseudo-code for the Semi-Implicit Method for Pressure-Linked Equations or SIMPLE algorithm

OpenFOAM provides an enhanced PISO algorithm that combines the methods from both the SIMPLE and PISO algorithms. This enhanced method, called PIMPLE, includes additional corrector steps within the PISO loop for each time step and allows for relaxation. PIMPLE is therefore able to use large time-steps (Courant No. $Co > 3$) for complex geometries. Figure 2.12 shows a typical pseudo-code for the PIMPLE algorithm. This algorithm is similar to PISO, except that there are two additional loops that iterate from $s \in (1,\ S)$ for solutions to p^s and from $r = (1,\ R)$ for predicted velocities $\vec{u}_k^r$ and fluxes ϕ_k^r. Additionally, there is a provision to relax either pressure, velocity, void fraction, or turbulence variables such that $x^{n+1} = x^n + \beta\,(x^r - x^n)$, where n designates current time, and r designates an internal PIMPLE loop iterate. The final inner and outer loops for PIMPLE do not relax variables, and $\beta = 1$ so that the time dependent values remain accurate.

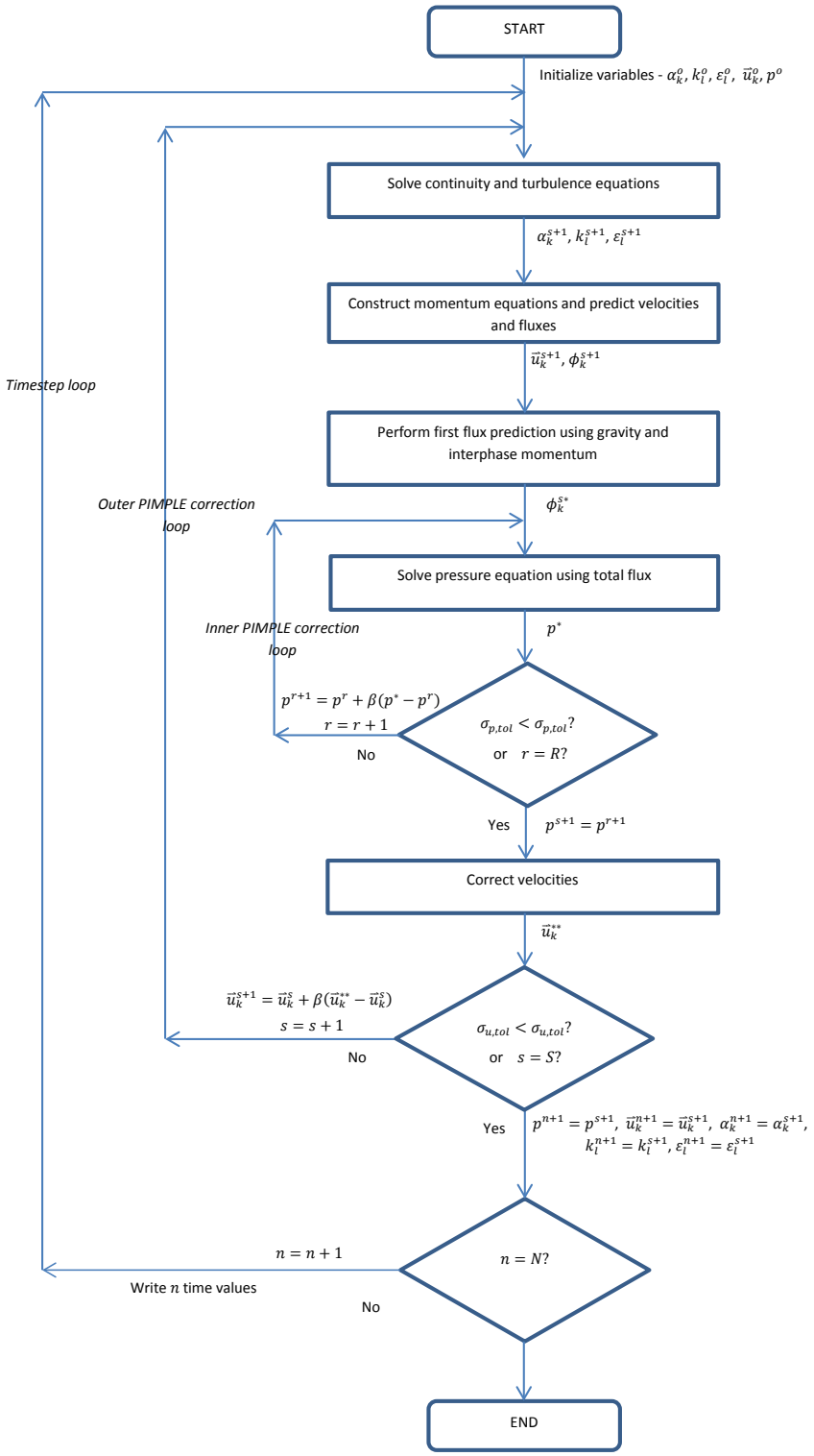


Figure 2.12: Pseudo-code for the combined SIMPLE and PISO algorithms, or the PIMPLE algorithm

2.2.4 Forward Solution Analysis

The following section shows field variable solutions using the `boilEulerFoam` forward equations, simplified from the original `boilEulerFoam` equations. Input parameters are shown in Table 2.3.

Table 2.3: **Adiabatic Input Parameters**

Parameter	OF Variable	Value
Pipe Diameter	Dh	51.2 mm
Inlet Pressure	p	4.5 MPa
Inlet Velocity	Uwater	1.01 m/s
Bubble Diameter	Ds	4.5 mm
Inlet Void	alphaair	0.1
Wall Heat Flux	qWall	0 kW
Constant Lift Coefficient	Cl	0.001
Constant Turb. Disp. Coefficient	Ctd	0.01
Constant Virtual Mass Coefficient	Cvm	0.5
Frank Wall Model Coefficients	Cwc, Cwd, p	10.0, 6.8, 1.7
Drag Model	Ishii-Zuber	

Figures 2.17 through 2.13 that follow contain the axi-symmetric simulation results for various flow phenomena as predicted by the adiabatic case of `boilEulerFoam`.

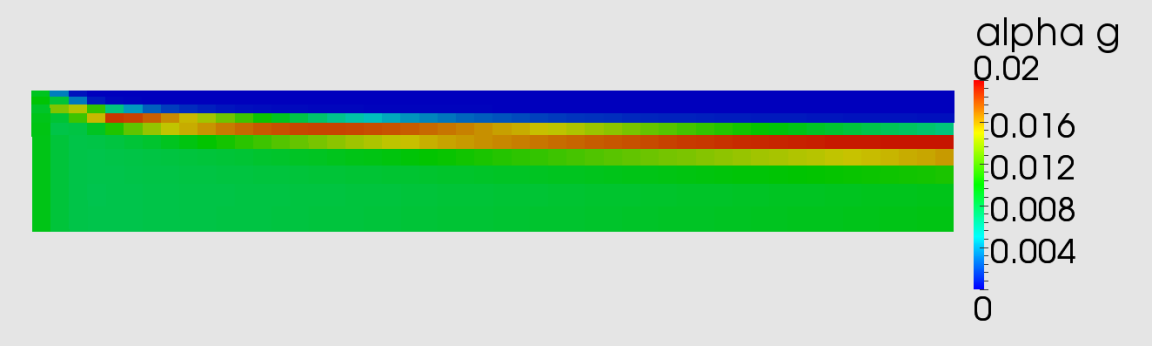


Figure 2.13: Vapor void fraction results for adiabatic `boilEulerFoam` multiphase base case

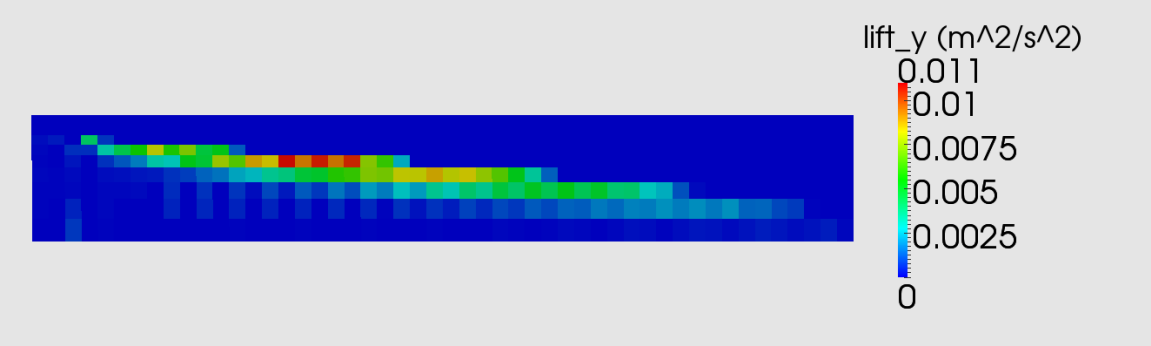


Figure 2.14: Lift force in the radial direction for adiabatic `boilEulerFoam` multiphase base case

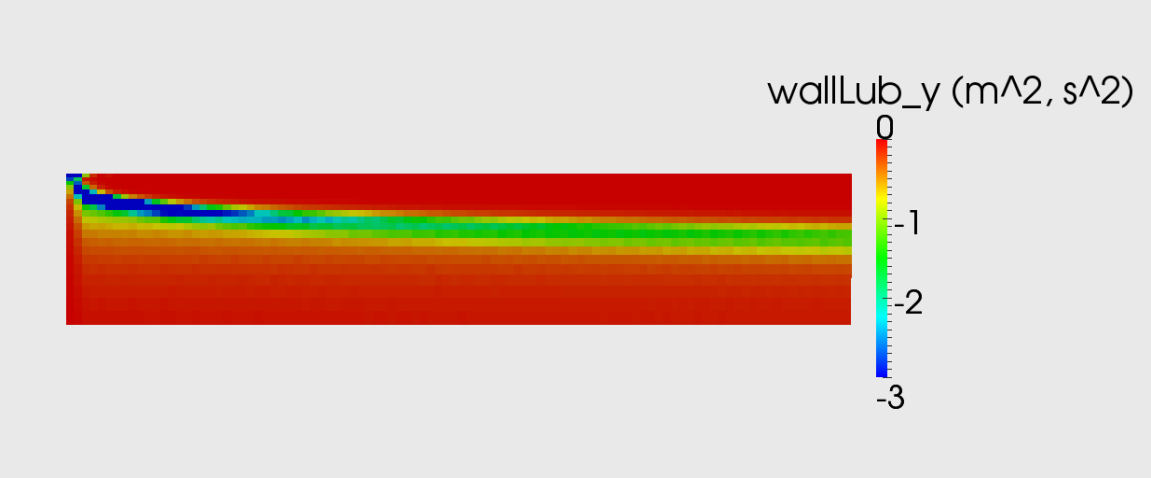


Figure 2.15: Wall lubrication force in the radial direction for adiabatic `boilEulerFoam` multiphase base case

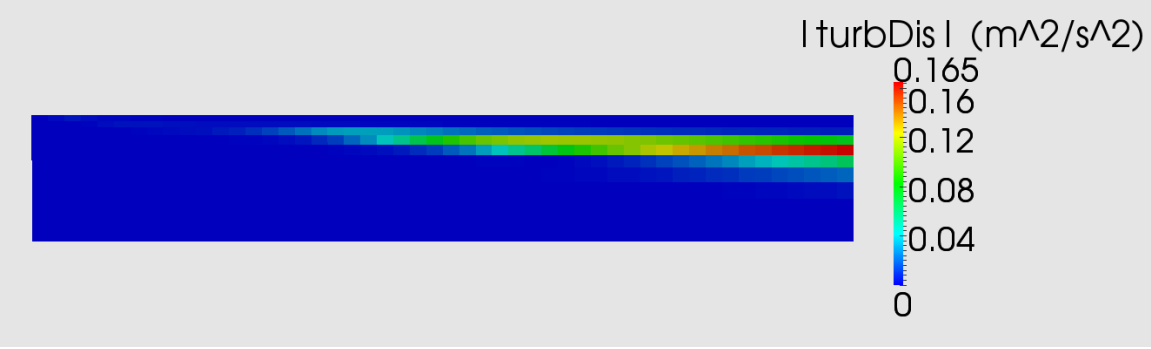


Figure 2.16: Magnitude of turbulent dispersion force for adiabatic `boilEulerFoam` multiphase base case

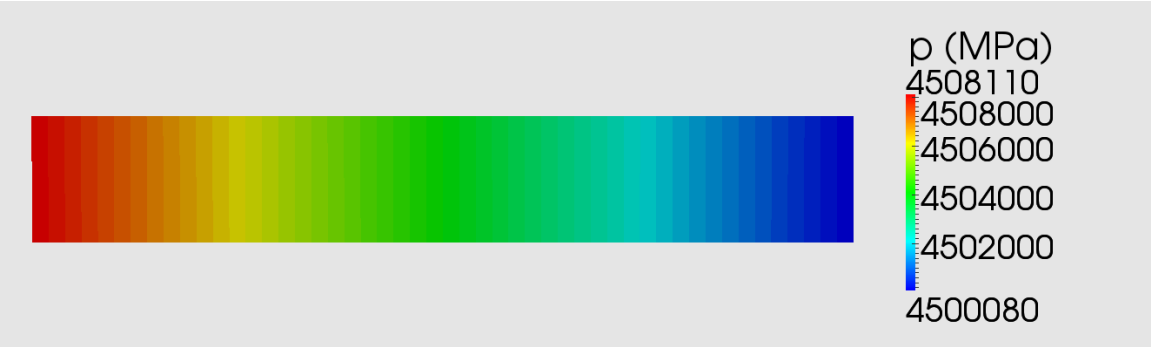


Figure 2.17: Pressure field results for adiabatic `boilEulerFoam` multiphase base case

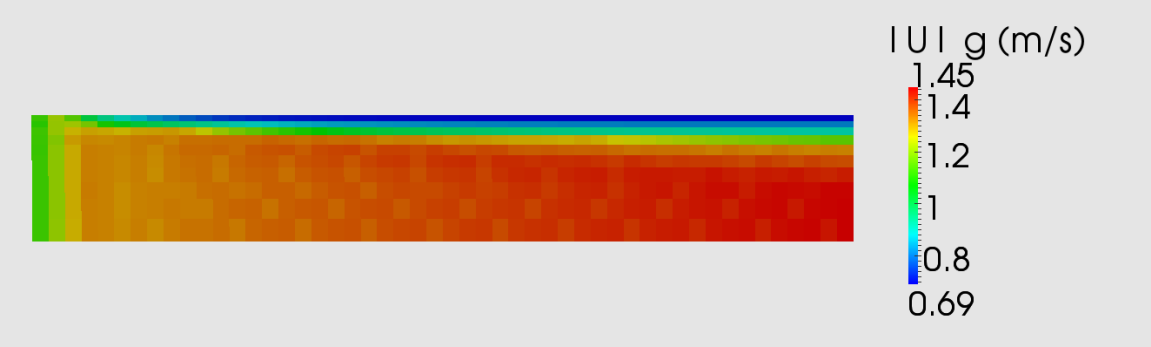


Figure 2.18: Vapor velocity magnitude field results for adiabatic `boilEulerFoam` multiphase base case

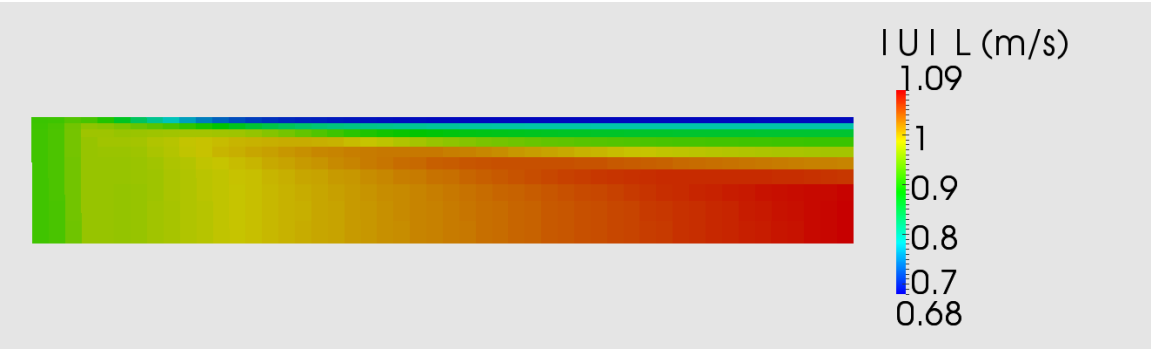


Figure 2.19: Liquid velocity magnitude field results for adiabatic `boilEulerFoam` multiphase base case

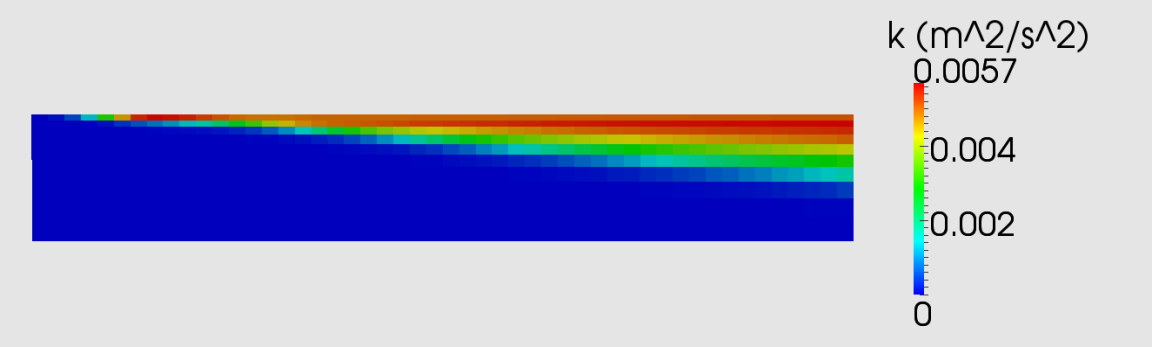


Figure 2.20: Turbulent kinetic energy for adiabatic `boilEulerFoam` multiphase base case

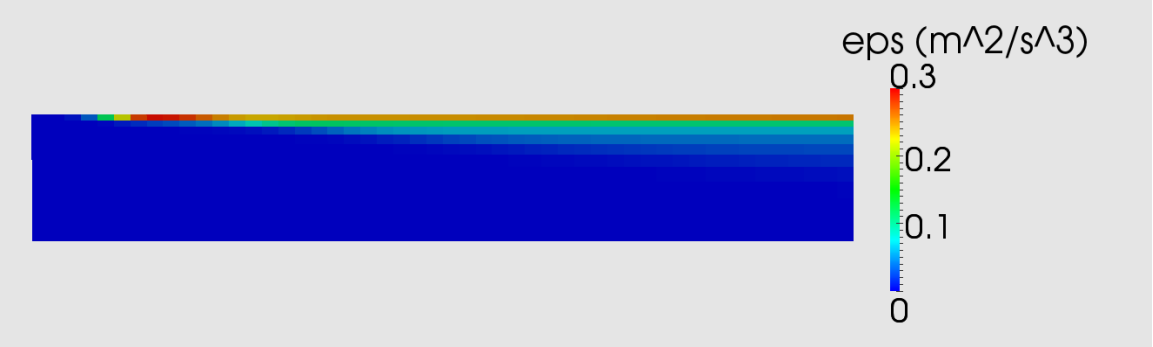


Figure 2.21: Turbulent dissipation for adiabatic `boilEulerFoam` multiphase base case

In Figure 2.13, the void profile enters the pipe at a uniform value of 0.1. As the velocity profile is established, interphase momentum transfer terms in conjunction with diffusion and dissipation set up the void profile. The lift force in Figure 2.14 acts in a positive radial direction (upwards) and pushes void towards the wall as a function of radial velocity. The wall lubrication force shown in Figure 2.15 acts in the negative radial direction (down) and counteracts lift by pushing void away. The magnitude of turbulent dispersion shown in Figure 2.16 acts on the sharpest gradients of the dispersed phase void profile.

Figure 2.17 indicates a linear pressure drop of 15 kPa from the inlet of the pipe to its exit, similar to that shown previously in Figure 2.2, which is expected for the friction dominated flow. From Figures 2.18 and 2.19, the vapor and liquid phase velocity distributions closely resemble one another with the exception that the vapor phase velocity magnitude is slightly greater than the liquid magnitude throughout the pipe. This is due to the buoyancy of the less dense vapor phase. However, the drag force limits the acceleration of the vapor phase and the resulting maximum relative velocity is 0.36 m/s.

The turbulent kinetic energy and turbulent dissipation solution fields are shown in Figures 2.20 and 2.21. The turbulent kinetic energy increases along the wall of the pipe and partially spreads into the bulk flow as the velocity profile develops. The turbulent dissipation coefficient remains fairly isolated along the wall where the steepest changes in velocity are present.

Some numerical oscillation is visible in Figure 2.18 and resembles the oscillation shown in the radial lift force in Figure 2.14. The explicit treatment of interphase momentum transfer terms typically decrease the diagonal dominance of the linearized momentum and pressure equations, which is why several terms are removed from the discretized momentum equation formulation. Since the relative velocity is high for this case (36% of inlet continuous velocity), interphase momentum transfer may be causing some small numerical instabilities, such as the ones visible in Figure 2.14. Increasing the drag force or changing the bubble size, either artificially or by changing drag models, will likely help limit some of these numerical oscillations since the numerical treatment increases stability.

In the PhD thesis conducted by Alali, `boilEulerFoam` predicted void distributions without phase change were compared to experimental MT-LOOP test results [48]. MT-LOOP data was collected for bubbly flows in a vertical pipe with an inner diameter of 51.2 mm. Data was collected at $z = 3.5$ m for various bubble sizes, void fractions, and superficial gas and liquid velocities. Alali's comparison is shown in Figure 2.22.

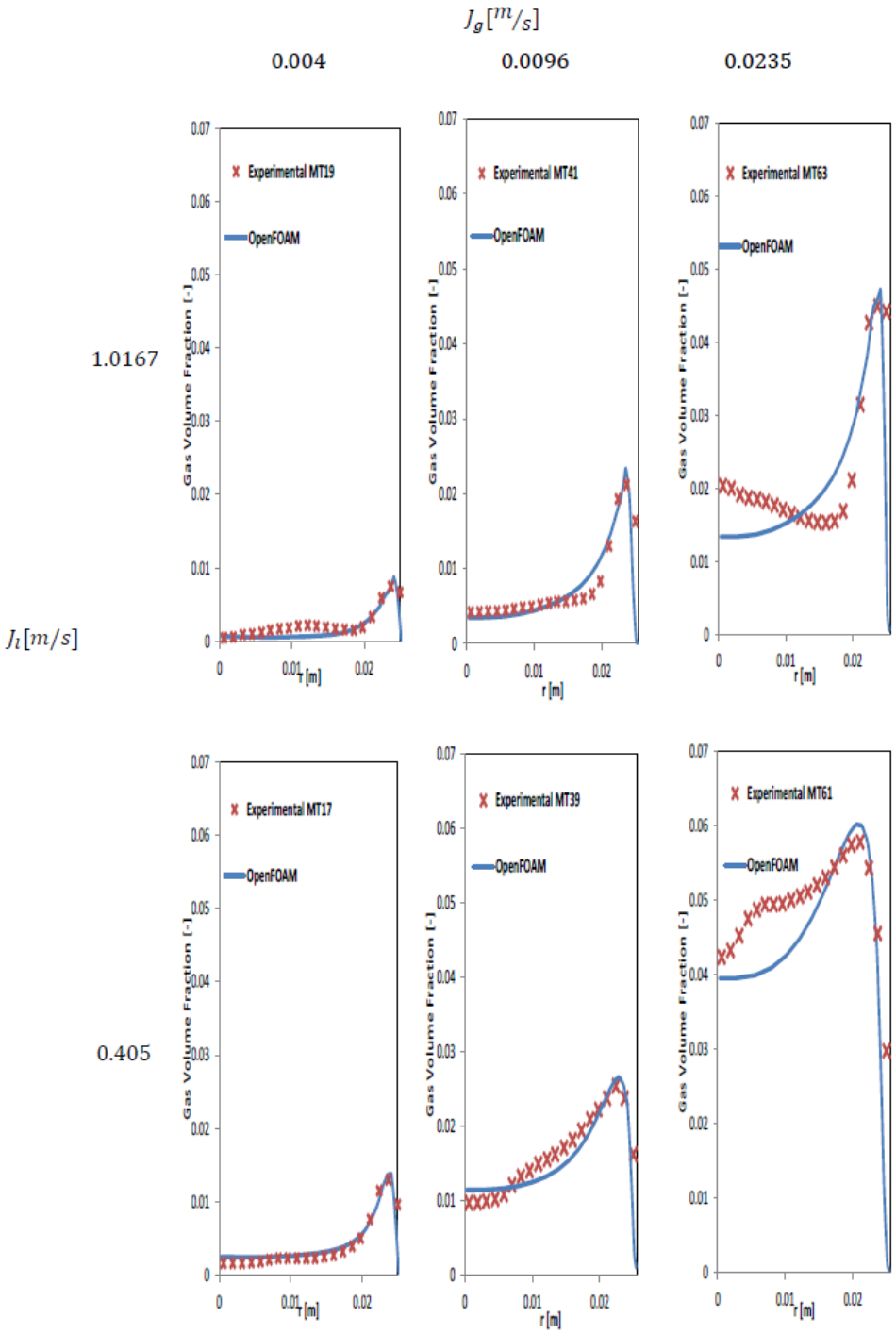


Figure 2.22: Comparison between MT-Loop radial void profile distributions and results from `boilEulerFoam` without phase change, as presented in Alali's PhD Thesis pp. 33-52.

Radial void profiles resemble MT-LOOP data in predicting the accumulation of void close to the wall of the pipe, and general bulk flow average values show good agreement with experimental results. The general shape of the void profile predicted by `boilEulerFoam` is the same for all of the cases, and for higher superficial gas velocities ($J_G = 0.0235\text{m/s}$), some of the experimental bulk flow profile behavior is missed in the calculation. The shape of these void profiles is highly dependent on interphase momentum transfer closure models, and these will be the primary focus of adjoint sensitivity studies.

Chapter 3

Automatic Differentiation Implementation in OpenFOAM

This chapter details the automatic differentiation capability built for OpenFOAM in order to create Jacobian matrices required for adjoint calculations. The AD capability was built using an existing object called FadOne developed by Jasak [54]. Fad stands for Foam automatic differentiation, and One indicates its first implementation in OpenFOAM. The prefix `fad` is used as a naming convention for AD implementations of existing OpenFOAM objects. For example, the AD counterpart to `scalarField`, an object designed to hold a list of scalar variables, would be `fadScalarField`.

Typical use of OpenFOAM does not require an in depth understanding of the objects required to build solvers. These objects are meant to be building blocks, where the functionality is utilized without understanding or having access to the internals of an object. This premise is referred to in computer science theory as encapsulation. However, in order to implement the FadOne class to automatically differentiate flow field solutions, all necessary operators and functions must be able to handle FadOne objects. It was therefore necessary to alter much of the source code that otherwise would be unnecessary to alter during normal solver construction. Performing a search of all OpenFOAM directories provides a primitive tally of over 6,000 instances of code pertaining to `fad` objects, and implementing automatic differentiation in OpenFOAM was not trivial.

This chapter provides a general explanation of the application of automatic differentiation in the forward sense and discusses it's implementation in OpenFOAM's object hierarchy. This chapter demonstrates AD capability for a basic Laplacian solver and verifies its created Jacobian. The final section demonstrates AD with respect to the nonlinear multiphase CFD

equations by verifying Jacobian matrices for a one dimensional adiabatic test case.

3.1 Templates and Operator Overloading

The most fundamental objects in OpenFOAM are called "primitive types" and include scalars, vectors, and tensors. These act as the fundamental variables which all finite volume functions use for their calculations. These functions then call operators that multiply, divide, etc. variables appropriately according to finite volume methodology. The primary functionality of automatic differentiation in OpenFOAM replaces existing primitive type variables with an analogous `fad` variable that intrinsically calculates both the variable values and derivatives. In an ideal scenario, operators and functions that take primitive types are blind to what they have been passed. The specifics of primitive type calculations are handled by a base class so that finite volume functions can be reused for many different implementations.

This functionality is accomplished in C++ by using templates and operator overloading. Templates allow the compiler to treat a function or object generally, trusting that specific definitions for specific variables is contained somewhere else in the code. We can demonstrate this principle using the divergence function taken from a basic transport equation

```
1   fvm::div(phi,T)
```

This function uses the cell face fluxe ϕ to calculate the divergence of field `T`. The `div()` function, however, does not know whether `T` is a field of scalars, a field of vectors, or a field of `fadScalar`s, etc. The appropriate declaration of the divergence function will therefore look like

```
2   template<class Type>
3   fvMatrix<Type> div
4   (
5       GeometricField<Type, ...
6   )
```

where the expression on line 2 lets the compiler know that the following function can be general for any type. Through templates, the definition of `div()` is general for any variable, and the function does not care what type of variable `T` is. The implementation of the `div()` divergence function will boil down to basic algebraic operations on `T`. These algebraic operators can accept floating point numbers or integers and use the existing functionality of C++ to return a value. A developer also has the freedom to define their own implementation of these same operators

that can accept and return instances of user created classes. In other words, a multiplication involving `scalarA * scalarB` and a multiplication involving `fadScalarA * fadScalarB` can both use the same $*$ operator, but their returned value is dependent on the specific definition of $*$ contained within the `scalar` and `fadScalar` classes. In computer science, this premise is known as operator overloading.

The combination of templates and operator overloading is the primary means by which automatic differentiation is implemented in OpenFOAM. The functions used to construct CFD solvers do not change, but added capability allows these functions to calculate both solutions and derivatives of solutions with respect to user defined values.

3.1.1 FadOne Implementation and Overloading Example

The FadOne class is an object designed to store both a value and all user defined derivatives of a value. If there is only one dependent variable $x = 5$ for an example calculation, then the FadOne implementation of this `scalar` will be

```
1   fadScalar x(0.0);
2   x.value() = 5.0;
3   x.deriv() = 1.0;
```

First, the `fadScalar` object `x` is declared and initialized on line 1. At the time of initialization, all values and derivatives are set equal to 0.0. Next, the `value()` function is called, setting the variable value equal to 5.0. Finally, the `deriv()` function is called and sets the derivative of x with respect to itself, which is 1.0. Basic operations on x can then be performed according to the following example

```
4   fadScalar y(0.0);
5   y = 3*x - 5;
```

at the end of the calculation on line 5, the `fadScalar` object `y` in this example contains 10 for its value and 3 for its derivative with respect to `x`. In this basic example, only one derivative is stored in a given `fadScalar` object. FadOne objects may contain as many derivatives as are needed by specifying an integer value for `nVars` according to the following typename definition

```
6   template<class Cmpt, int nVars>
7   class FadOne
8   {
...
9   typename FadOne<scalar, 1> fadScalar;
```

where `Cmpt` designates a primitive type, and `nVars` designates the number of derivatives. Therefore, the typename for `fadScalar` on line 9 is a templated FadOne object that holds a scalar value and one scalar derivative. This implementation requires the user to set the number of derivatives at compile time. Ideally, the one would like to set the number of derivatives at run time, but it is computationally expensive to build arrays of unknown size in C++, especially if these arrays are to be used as primitive types.

We now demonstrate operator overloading using the basic definition of multiplication for automatically differentiable objects. According to basic calculus, the derivative of a multiplication operation is given by

$$\frac{d}{dx}(f_1 \times f_2) = f_1 \frac{df_2}{dx} + f_2 \frac{df_1}{dx}$$

The multiplication operation of the `fadScalar` object `x` shown on line 5 in the previous example automatically performs this calculation according to the following overloaded $*=$ operator

```
6   template<class Cmpt, int nVars>
7   void FadOne<Cmpt, nVars>::operator*=(FadOne<Cmpt, nVars> v)
8   {
9       for (int i = 0; i < nVars; i++)
10      {
11          this->deriv(i) = this->deriv(i)*v.value() + this->value()*v.deriv(i);
12      }
13
14      this->value() *= v.value();
15  }
```

This operator is defined for a templated FadOne class and can accept any primitive type `Cmpt` and any integer number of derivatives `nVars` as represented by the template declaration on line 6. When a multiplication operation is called using a FadOne object, first the derivative is set on lines 9 through 12. For reference, `this` is a pointer that refers to the object on the left hand side of the `*=` operator. The `for` loop iterates through all the derivatives of the FadOne object

and sets the derivative of `this` on line 11 according to the multiplication rule. Finally, the value of `this` is set on line 14, and the $*=$ carries out the appropriate multiplication and sets all derivatives automatically.

All algebraic operators required by the finite volume functions are defined similarly within the FadOne class. As a result, instantiation of FadOne objects can use existing OpenFOAM solvers to automatically return derivatives of user defined variables.

3.2 OpenFOAM Object Hierarchy

The primary function of any OpenFOAM solver is to take a mesh and a given field of variables and construct matrices according to finite volume methodology. These matrices are then solved according to a supplied algorithm, whether it be PISO or SIMPLE or another user defined method. Objects in OpenFOAM are designed in such a way to facilitate this methodology as efficiently and succinctly as possible. Figure 3.1 on the following page shows a rough flow chart demonstrating the use of objects to build finite volume matrices and perform calculations.

Primitive type objects such as scalars, vectors, and tensors are the starting point for this hierarchy. Within each of these primitive classes are hidden all of the necessary operators for performing calculations such as dot products, cross products, and algebraic operations, as well as function calls like max or min.

The FadOne class created by Jasak was originally designed to be a stand alone functionality. To allow OpenFOAM to perform finite volume operations on FadOne objects, it was necessary to add a FadOne primitive analogous to the scalar primitive type. This FadOne primitive type is called `fadScalar`, and its implementation is shown in section 3.1.1 on lines 6-9 of example source code. If a scalar holds a floating point number, then a `fadScalar` holds floating point values and floating point derivatives. In the same way that vectors hold dimensional arrays of scalars, `fadVector`s must therefore be able to hold dimensional arrays of `fadScalar`s. Similarly, all operations and classes defined for scalars must also have an equivalent definition for `fadScalar`s. Ideally these operations should be able to operate on any user defined primitive type if objects are templated correctly. However, much of OpenFOAM is hard coded for scalars and a significant amount of functionality was added to facilitate the use of automatically differentiable variables.

The next level of hierarchy shown in Figure 3.1 shows the implementation of the `Field` class. This class stores lists of `scalar`s, `vector`s, `fadScalar`s, etc, and provides access to stored

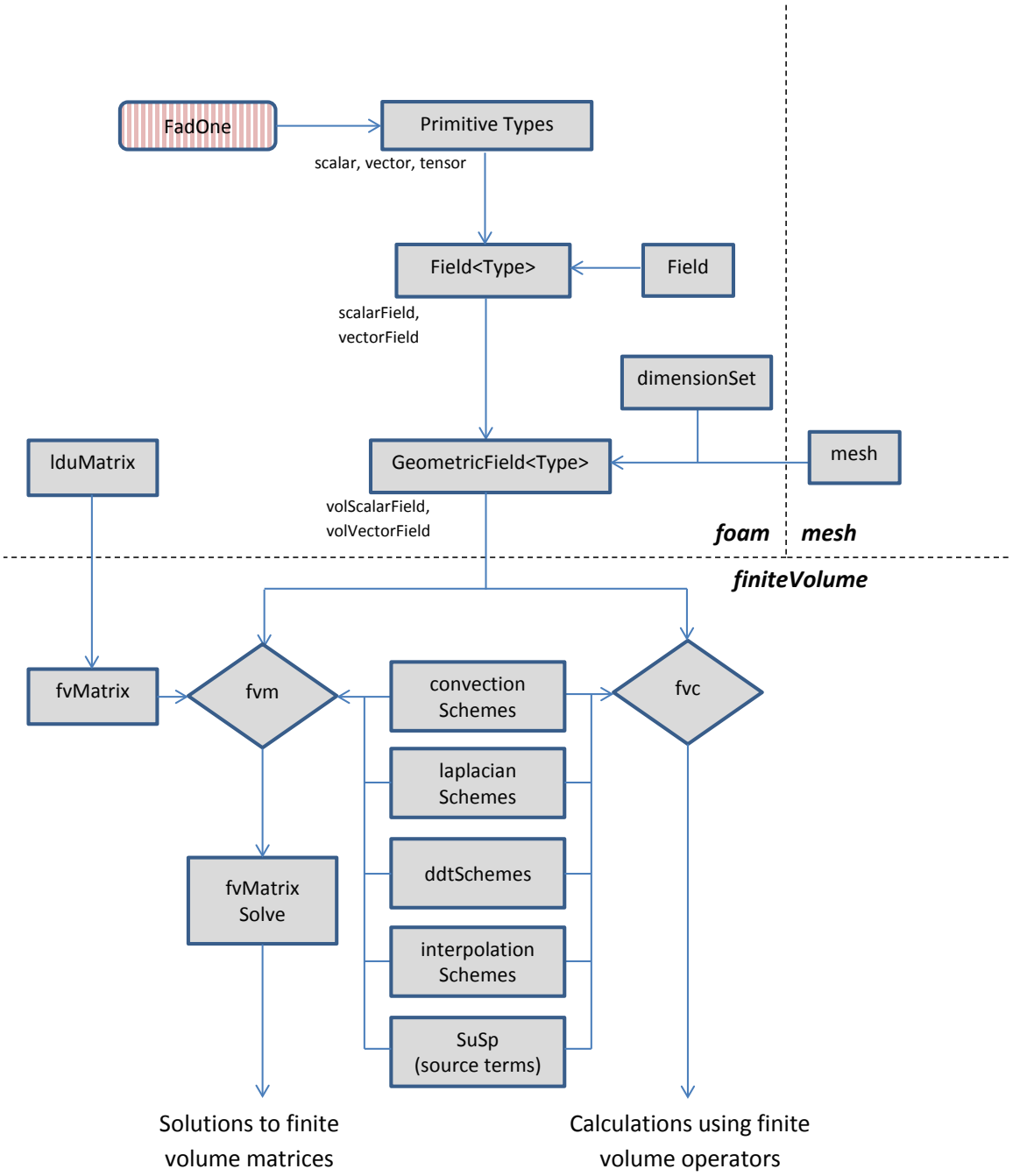


Figure 3.1: Flow chart demonstrating object dependency of finite volume matrices (fvm) and finite volume calculations (fvc)

values to be utilized by operators such as $*$ or $+$ and functions such as `max` or `min`. `Field`s for carrying specific types of primitive variables are defined using typenames like `scalarField` and `fadScalarfield`. `Field`s then incorporate dimension and a volume mesh in order to build `GeometricField` objects - the next box in the flow chart in Figure 3.1. Specific instances of `GeometricField` objects for carrying specific primitive types are defined using typenames such as `volScalarField` for cell centered values and `surfaceScalarField` for cell face values. The `Geometricfield` class matches a given `Field` with mesh locations for both internal fields and boundary conditions.

The objects referred to up to this point are defined within the $foam$ directory of OpenFOAM and can be thought of as the most basic building blocks of the finite volume solvers. The only exception to this is the `mesh` object, which is relegated to its own *mesh* directory. The directory division is noted in Figure 3.1 using a dashed line and appropriate label. The final directory described in this section is the $finiteVolume$ directory. It contains classes for performing finite volume operations on `GeometricField` objects.

There are two namespaces under which OpenFOAM performs finite volume calculations - `fvm` (finite volume matrix) and `fvc` (finite volume calculation). Both namespaces contain functions and classes used for convection, Laplacian, time difference, and interpolation operators. The `fvc` namespace returns finite volume calculations using the supplied `GeometricField`. The `fvm` namespace uses the same function calls to construct matrix coefficients used to solve for a given `GeometricField`. `fvm` operations therefore also require the `fvMatrix` class in order to transfer coefficients of operations like divergence or diffusion to a matrix and vector source term. The `fvMatrix` class is inherited from the `lduMatrix` class defined in the foam directory and stores non-zero matrix values in diagonal, lower, and upper arrays.

This brief discussion of OpenFOAM's object hierarchy was intended to illustrate the flow of primitive type object dependence. In order to develope an AD capability using FadOne, OpenFOAM objects used for finite volume calculations must be altered so that they are able to operate on the new `fadScalar` primitive type. All definitions for scalar operations must also have definitions for the `fadScalar` class, and all vector operations must also have definitions for the `fadVector` class, etc. In this way, functions defined in the `fvc` namespace that return calculations such as divergence, Laplacian, and interpolation will return the appropriate finite volume calculation as well as the derivative of each calculation with respect to user defined variables.

Although `fvm` functions must also be able to operate on FadOne objects, no linear solves are

performed using `fadScalar` variables. The only operations that return derivatives are contained within the `fvc` namespace. In order to construct the pressure equation, discrete matrix diagonal and off-diagonal arrays are necessary according to the methodology shown in section 2.2.1. As a result, matrices of `fadScalar` variables are constructed but not solved.

3.3 Jacobian Implementation in Basic Laplacian Solver

In order to verify automatic differentiation capability, a Jacobian matrix was built for a basic Laplacian solver. The finite volume Laplacian function was thought to be the most complex of the solver functions with regards to the OpenFOAM algorithm, and successful Laplacian implementation would indicate the ability to automatically differentiate other functions. The solver `laplacianFoam` solves the following equation

$$\frac{\partial T}{\partial t} - \nabla \cdot (D_T \, \nabla T) = 0 \tag{3.1}$$

$$T|_{t=0} = T_{initial} \quad T|_{\delta S} = T_S \tag{3.2}$$

This equation is analogous to the conduction of temperature according to a diffusion coefficient D_T. For simplicity, D_T is constant. The OpenFOAM implementation of this equation is

```
1   solve
2   (
3       fvm::ddt(T) - fvm::laplacian(DT, T)
4   );
```

If boundary conditions do not vary with time, this equation has a steady state solution. A simple 5×5, 2 dimensional mesh was used, and the solution to this Laplacian problem is shown in Figure 3.2. The left and right hand boundary conditions are equal to 100, and the top and bottom boundary conditions are equal to 500. The discrete solution shown in Figure 3.2 is represented by $\vec{T_o} = [T_1 \, T_2 \cdots T_{25}]_o^T$.

For constant diffusion coefficients, the Laplacian operator is simply a second derivative with respect to space and is self adjoint. The discrete matrix constructed using `fvm::laplacian(DT,T)` will be symmetric and is also self adjoint since $\mathbf{A} = \mathbf{A}^T$.

To develop a Jacobian for the discrete Laplacian matrix operator using automatic differentiation, we perturb the original Laplacian equation by introducing a source term $\vec{q} = [100 \; 100 \ldots 100]^T$.

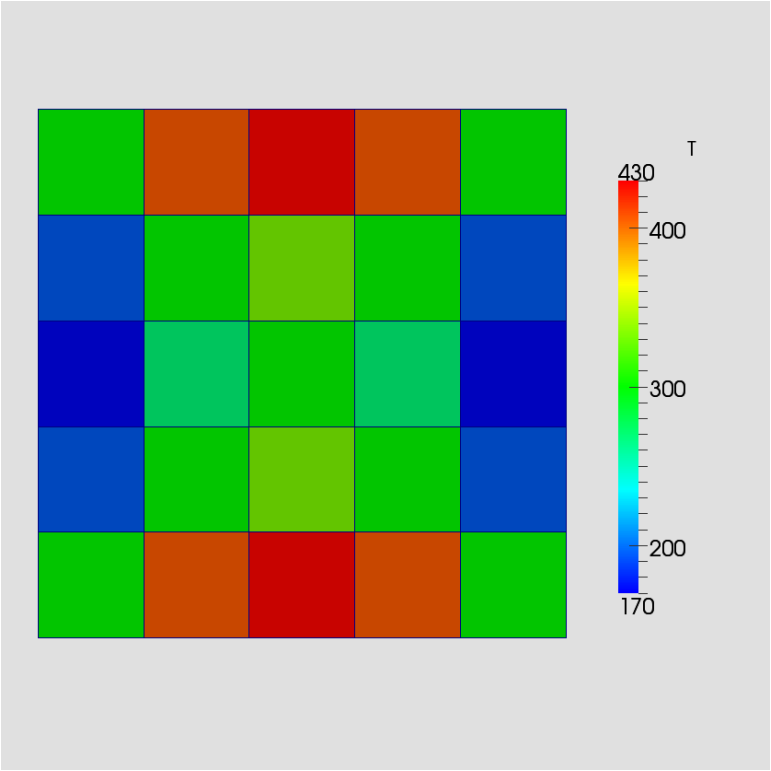


Figure 3.2: Steady state solution to the Laplacian equation using a constant diffusion coefficient

The Taylor series expansion of the discrete Laplacian operator $\mathbf{F}$ operating on $\Delta\vec{T} = \vec{T} - \vec{T_o}$ is as follows

$$\frac{\partial \mathbf{F}}{\partial \vec{T}}\left(\vec{T} - \vec{T_o}\right) = \mathbf{F}(\vec{T}) - \mathbf{F}(\vec{T_o}) = \vec{q} \tag{3.3}$$

Since $\mathbf{F}$ is linear with respect to $\vec{T}$, the Jacobian matrix is trivial and is simply given by $\frac{\partial \mathbf{F}}{\partial \vec{T}} = \mathbf{F}$. This presents an easy to verify case to test FadOne's ability to calculate derivatives and build Jacobian matrices.

In order to calculate derivatives of $\mathbf{F}$ with respect to $\vec{T}$, a `volFadScalarField` called `fT` is built and operated on using `fvc` functions to propagate the Jacobian matrix. The `fadScalar` primitive type is defined at compilation as

```
5   typedef FadOne<scalar, 25> fadScalar
```

The `fadScalar` field `fT` is initialized using the following code

```
5   for(i = 0; i < fT.size(); i++)
```

```
6   {
7       fT[i] = T[i];
8       fT[i].deriv(i) = 1.0;
9   }
```

This sets the discrete value of `fT` equal to the value of T on line 7, and also sets the derivatives of fT on line 8 such that $\frac{\partial T_i}{\partial T_i} = 1$ and $\frac{\partial T_i}{\partial T_j} = 0$. With the `fvc::laplacian()` function properly defined to handle FadOne primitive types, the function call to propagate the Jacobian matrix is simply

```
10  volFadScalarField J
11  (
12      -fvc::laplacian(DT, fT)
13  );
```

This demonstrates the proper templating and operator overloading of this `fvc` function in that the same function call is made for both the scalar field `T` and the `fadScalar` field `fT`. The `volFadScalarField` J now contains the residual of the $\mathbf{F}\vec{T} = \vec{0}$ operation and all derivatives with respect to each spatial location. This field of derivatives will be the 25×25 Jacobian matrix, and J should be equal to $\mathbf{F}$. The calculation using the Taylor series expansion reveals this to be true, and the solution to

$$\mathbf{J}\left(\vec{T} - \vec{T_o}\right) = \vec{q}$$

is exactly equal to

$$\mathbf{F}\left(\vec{T} - \vec{T_o}\right) = \vec{q}$$

This is visible in figures 3.3 and 3.4, and the solutions in each figure are the exact same. The physical implication of introducing a source term to the Laplacian equation is the addition of volumetric heat generation to a heat conduction problem. As a result, the increase in scalar field T is dramatic and follows a parabolic shape with the maximum temperature located at the center cell. The difference between the fields calculated using the J and F was no greater than 5×10^{-4}

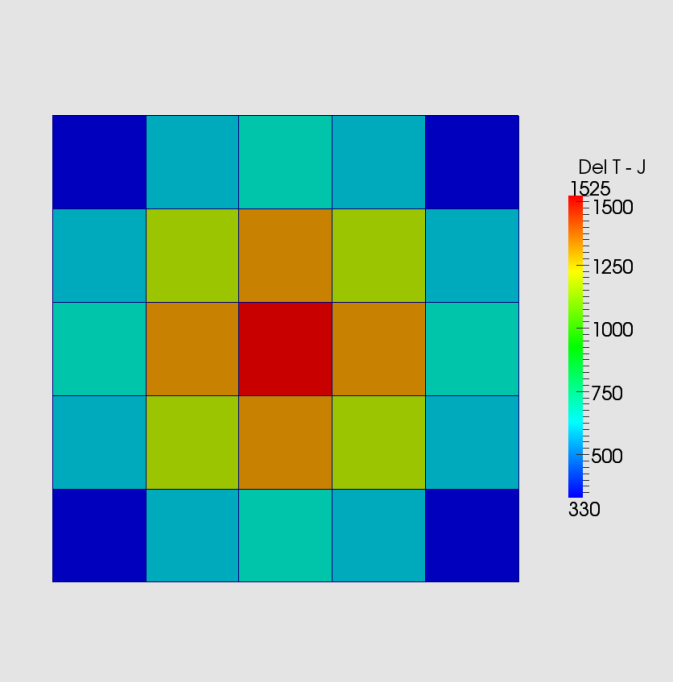


Figure 3.3: Solution to the perturbed Laplacian equation using the exact Jacobian matrix calculated from automatic differentiation

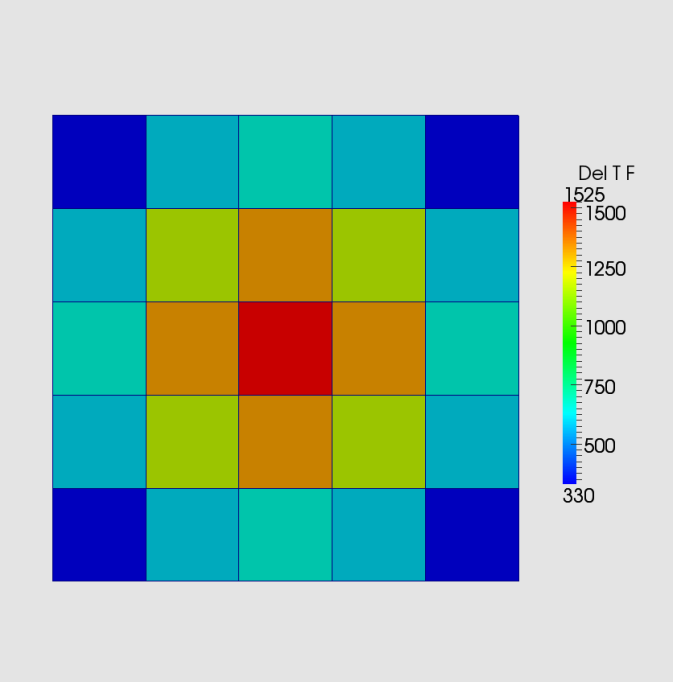


Figure 3.4: Solution to the perturbed Laplacian equation using the discrete Laplacian matrix

The internal `fadScalarField` for J returned by the `fvc::laplacian` operation is tridagonal with fringes and is shown below.

```
2.7e-06 | 0.75 -0.125 0 0 0 -0.125 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
-1.5e-05| -0.125 0.625 -0.125 0 0 0 -0.125 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
9.0e-06 | 0 -0.125 0.625 -0.125 0 0 0 -0.125 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
8.3e-06 | 0 0 -0.125 0.625 -0.125 0 0 0 -0.125 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
3.3e-06 | 0 0 0 -0.125 0.75 0 0 0 0 -0.125 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
1.2e-05 | -0.125 0 0 0 0 0.625 -0.125 0 0 0 -0.125 0 0 0 0 0 0 0 0 0 0 0 0 0]
-3.7e-06| 0 -0.125 0 0 0 -0.125 0.5 -0.125 0 0 0 -0.125 0 0 0 0 0 0 0 0 0 0 0 0]
-1.7e-05| 0 0 -0.125 0 0 0 -0.125 0.5 -0.125 0 0 0 -0.125 0 0 0 0 0 0 0 0 0 0 0]
-1.0e-05| 0 0 0 -0.125 0 0 0 -0.125 0.5 -0.125 0 0 0 -0.125 0 0 0 0 0 0 0 0 0 0]
1.6e-05 | 0 0 0 0 -0.125 0 0 0 -0.125 0.625 0 0 0 0 -0.125 0 0 0 0 0 0 0 0 0]
-1.1e-05| 0 0 0 0 0 -0.125 0 0 0 0 0.625 -0.125 0 0 0 -0.125 0 0 0 0 0 0 0 0]
1.4e-05 | 0 0 0 0 0 0 -0.125 0 0 0 -0.125 0.5 -0.125 0 0 0 -0.125 0 0 0 0 0 0 0]
-6.1e-06| 0 0 0 0 0 0 0 -0.125 0 0 0 -0.125 0.5 -0.125 0 0 0 -0.125 0 0 0 0 0 0]
5.4e-06 | 0 0 0 0 0 0 0 0 -0.125 0 0 0 -0.125 0.5 -0.125 0 0 0 -0.125 0 0 0 0 0]
4.7-06 0| 0 0 0 0 0 0 0 0 0 -0.125 0 0 0 -0.125 0.625 0 0 0 0 -0.125 0 0 0 0]
-1.1e-05| 0 0 0 0 0 0 0 0 0 0 -0.125 0 0 0 0 0.625 -0.125 0 0 0 -0.125 0 0 0]
-4.2e-06| 0 0 0 0 0 0 0 0 0 0 0 0 -0.125 0 0 0 -0.125 0.5 -0.125 0 0 0 -0.125 0 0]
1.9e-06 | 0 0 0 0 0 0 0 0 0 0 0 0 0 -0.125 0 0 0 -0.125 0.5 -0.125 0 0 0 -0.125 0]
4.9e-06 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -0.125 0 0 0 -0.125 0.5 -0.125 0 0 0 -0.125]
4.6e-06 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -0.125 0 0 0 -0.125 0.625 0 0 0 0 -0.125]
-2.4e-06| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -0.125 0 0 0 0 0.75 -0.125 0 0 0]
-1.4e-05| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -0.125 0 0 0 -0.125 0.625 -0.125 0 0]
2.4e-05 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -0.125 0 0 0 -0.125 0.625 -0.125 0]
1.5e-07 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -0.125 0 0 0 -0.125 0.625 -0.125]
2.7e-06 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -0.125 0 0 0 -0.125 0.75]
```

The residual in the first column is the same as the tolerance of the linear solver used by OpenFOAM. A symmetric matrix is clearly visible and is exactly equal to the coefficients used to build **F**.

3.3.1 OpenFOAM Matrix Methodology

This implementation of automatic differentiation requires derivative values for all spatial locations, which are mostly zero. The subsequent Jacobian must store a full matrix of coefficients, which are mostly zero as well. If there are n cells and m variables for a given solver, then the Jacobian matrix will contain $(nm)^2$ values. Storage of full matrices is not feasible for meshes with $n > 10^2$; this is a major shortcoming of the current Jacobian implementation.

OpenFOAM stores sparse matrices using an LDU methodology and addressing array. Each cell will have one diagonal coefficient for cell centered values and one off diagonal coefficient for each cell face. The lower and upper triangles are therefore addressed using cell faces, while the matrix diagonal is addressed using cell centers. This coefficient addressing scheme is the exact same array used by the mesh to assign cell volumes and faces to appropriate cell locations. Because OpenFOAM uses a compact computational molecule, the matrix structure of discretized divergence, Laplacian, and time differencing operators is known, and their appropriate matrix structure is based solely on the mesh.

Jacobian matrices used for adjoint problems do not use this compact computational stencil. Proper Jacobian structure involves derivatives of all discrete matrix coefficients with respect to all dependent variables. For coupled systems of equations involving convection, these coefficients are dependent both on the flow field flux and the convected value. OpenFOAM uses past iterate values for flux, but we desire to capture current time dependence on both flux and the convected field. This computational molecule is not known, and Jacobian matrices cannot propagate discrete values intuitively using the mesh using the LDU methodology inherent in OpenFOAM.

As a result, current adjoint capability constructed for the multiphase flow equations is limited based on the virtual memory of the computational platform. Storing full matrices is costly, and two dimensional pipe mesh size for multiphase CFD Jacobian construction is limited to around 200 cells. It may be possible in the future to construct Jacobian matrices in the intuitive way that OpenFOAM builds its discrete matrices such that the computational molecule for finite volume operations is known beforehand. This will require the current mesh addressing array implementation and automatic differentiation to work in concert, and it could be a future

method of building sparse Jacobian matrices without a large memory requirement.

3.4 Jacobian Implementation in `boilEulerFoam`

The multiphase flow solver `boilEulerFoam` solves for void fraction, velocity, pressure, turbulent dispersion, and turbulent kinetic energy. The Jacobian of this system of equations must take derivatives with respect to all variables at all spatial locations. The Jacobian operator for `boilEulerFoam` in two dimensional geometry is written as

$$\mathbf{J} = \frac{\partial \mathbf{F}}{\partial \vec{\phi}} \tag{3.4}$$

with the dependent variables

$$\vec{\phi} = \left[\vec{\alpha}_g \; \vec{\alpha}_l \; \vec{u_g}_x \; \vec{u_g}_y \; \vec{u_l}_y \; \vec{u_l}_y \; \vec{p} \; \vec{k}_l \; \vec{\epsilon}_l\right]^T \tag{3.5}$$

Vectors are used to denote discrete values, and the $\vec{\phi}$ uses phasic subscripts consistent with the discrete equations derived in section 2.2.1. In this way, the numerical dependence of all variables is contained within $\mathbf{J}$. The implementation in `boilEulerFoam` is the same as the previous implementation in the basic Laplacian solver such that `volFadScalarfield`s are constructed for each variable and `fvc` calculations performed to compute the residuals and propagate derivatives.

Due to the necessity to store derivatives for all spatial locations, a simplified one dimensional problem was examined using a ten cell mesh. For seven dependent variable fields solved using a ten cell mesh, the Jacobian matrix is 70×70 and the appropriate templating of FadOne for Jacobian construction is

```
1   typedef FadOne<scalar, 70> fadScalar
```

A test similar to the Laplacian operator case was conducted to verify the Jacobian and test its accuracy. For the nonlinear problem

$$\mathbf{F}[\vec{\phi}] = \vec{d}$$

we can write a Jacobian problem using the Taylor expansion

$$\mathbf{F}[\vec{\phi}] = \mathbf{F}[\vec{\phi}_o] + \frac{\partial \mathbf{F}}{\partial \vec{\phi}} \left(\vec{\phi} - \vec{\phi}_o\right) + \mathcal{O}(\Delta^2)$$

where $\mathcal{O}(\Delta^2)$ represents the higher order terms. Using this equation, we then have a first order approximation of a perturbation in solution field $\vec{\phi}$

$$\frac{\partial \mathbf{F}}{\partial \vec{\phi}} \left(\vec{\phi} - \vec{\phi}_o \right) \approx \mathbf{F}[\vec{\phi}] - \mathbf{F}[\vec{\phi}_o] \tag{3.6}$$

With equation 3.6, we have a way of checking solutions to a perturbed field $\Delta\vec{\phi} = \vec{\phi} - \vec{\phi}_o$ using the Jacobian determined by automatic differentiation. This is done by calcuating the right hand side of equation 3.6 using the values of $\vec{\phi}$ and $\vec{\phi}_o$ and then solving for $\Delta\vec{\phi}$. Figures 3.5 through 3.9 show initial and perturbed solutions to void fraction, velocity (dispersed and continuous phases), and pressure superimposed with the Jacobian solution calculated according to equation 3.6. For this case, $\mathbf{F} \neq \frac{\partial \mathbf{F}}{\partial \vec{\phi}}$ as in the previous Laplacian problem, and the Jacobian matrix includes off-diagonal terms that are dependent on other discrete variables. Wall lubrication was removed from the momentum equation and wall dissipation was removed from the $k - \epsilon$ equations for this case to be consistent with a one dimensional model. All physical properties were held constant according to the adiabatic test cases shown in section 2.3.

The perturbation and Jacobian prediction of discrete void fraction $\vec{\alpha}_g$ is shown in Figure 3.5. The red line denotes the initial solution, the dashed green line denotes the perturbed solution, and the blue dots show the predicted Jacobian solution. The inlet value for dispersed phase void fraction was 0.1 and the inlet value for both dispersed and continuous velocity was 1m/s. A step function of approximately 20% of the inlet value of the dispersed phase void fraction was used as a perturbation for four of the cells in the middle of the pipe. This perturbation step function was thought to be irregular and drastic and could help reveal potential problems in the Jacobian calculation.

Due to the use of micro-bubbles ($D_S = 0.001$ mm), the solution of dispersed phase void fraction is trivial and a constant value of 0.01 is shown in Figure 3.5, denoted by the red line. This bubble diameter was used to increase the stability of the numerical solution. The perturbation of approximately of 0.02 for $\vec{\alpha}_g$ is shown for the four middle cells using the dashed green line Using equation 3.6 and the automatically differentiated Jacobian matrix, the approximated field is shown using the blue circles, and the maximum relative error between the exact perturbation and the Jacobian predicted perturbation is 0.49%. A small undershoot of the predicted field is visible in Figure 3.5, but otherwise it can be said that the Jacobian matrix is closely approximating the $\vec{\alpha}$ perturbation and the automatic differentiation capability for the nonlinear multiphase system of equations is working as expected. An equivalent figure for the dispersed phase void fraction perturbation is not shown since the calculation $\vec{\alpha}_g + \vec{\alpha}_l - 1 = 0$ for this field

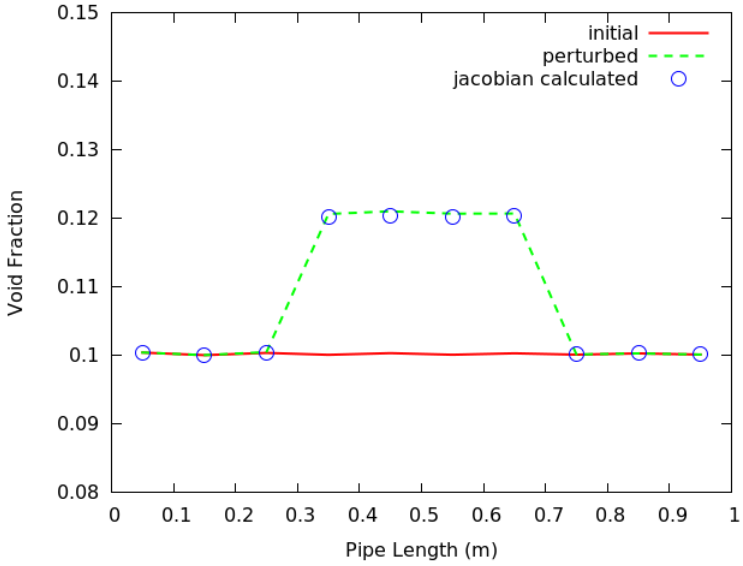


Figure 3.5: Initial, perturbed, and Jacobian calculated dispersed phase void fraction solutions to the one dimensional multiphase flow problem

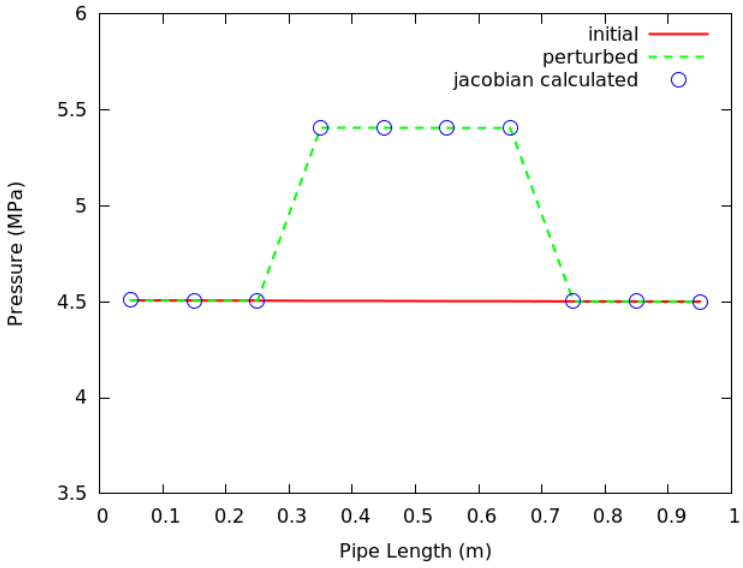


Figure 3.6: Initial, perturbed, and Jacobian calculated pressure solutions to the one dimensional multiphase flow problem

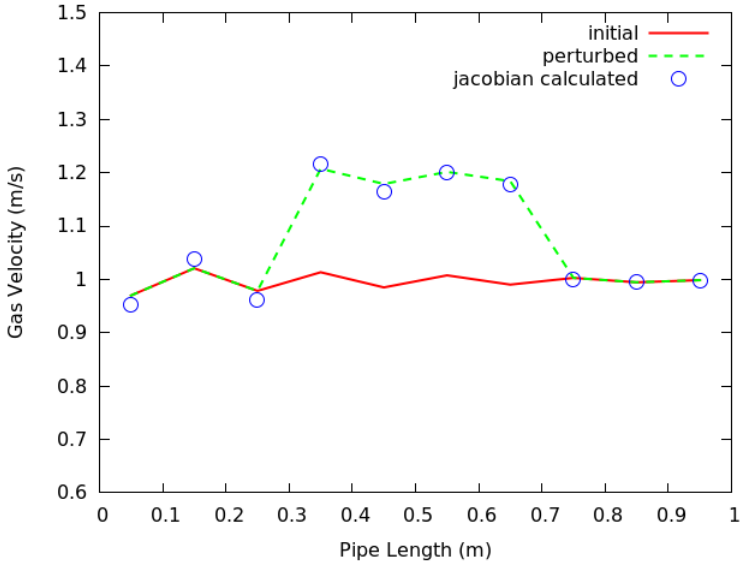


Figure 3.7: Initial, perturbed, and Jacobian calculated dispersed phase velocity solutions to the one dimensional multiphase flow problem

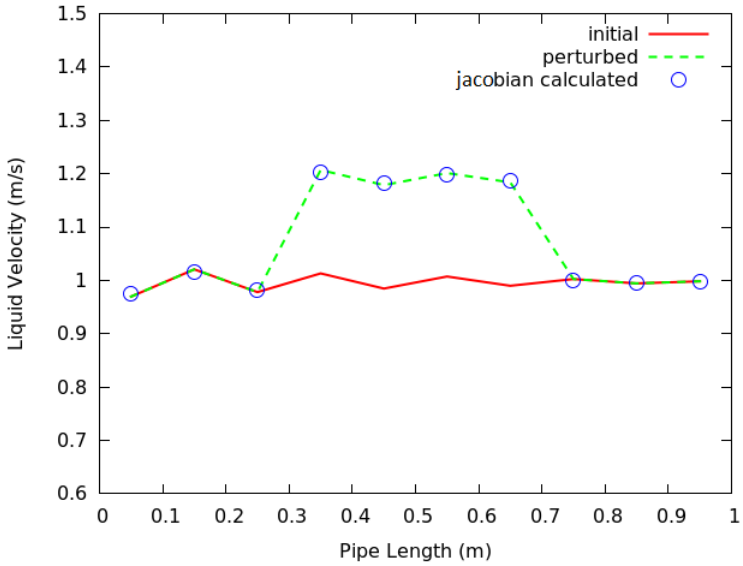


Figure 3.8: Initial, perturbed, and Jacobian calculated continuous phase velocity solutions to the one dimensional multiphase flow problem

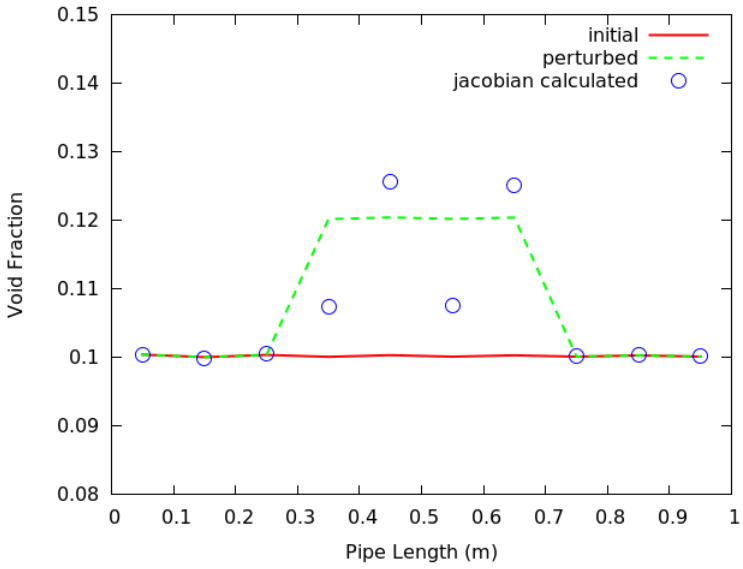


Figure 3.9: Initial, perturbed, and Jacobian calculated dispersed phase void fraction using the total variation diminishing interpolation scheme

is trivial.

Figure 3.6 shows similar results for pressure where now pressure is perturbed by 20% of the initial value in the four middle cells. The pressure drop along the length of the one dimensional channel is $7 \times 10^{-3} MPa$ and is hardly visible from the red line of the figure. The difference between the perturbed green solution and the Jacobian calculated perturbation is close to zero (4×10^{-6}). This high degree of accuracy is likely due to the linear nature of the Laplacian operator used in the pressure equation.

In Figure 3.7, the gas velocity is again perturbed in the same locations by 20% of the inlet value. The Jacobian approximates the perturbation fairly well with a maximum relative error equal to 1.6%, slightly more than the results in Figure 3.5. Figure 3.8 shows similar results for a continuous phase velocity perturbation of 20%, but the relative error in continuous phase velocity prediction, 0.4%, is less than the continuous phase velocity prediction. The predictor-corrector scheme used to calculate liquid and vapor flow velocity is highly non-linear, and this is likely the reason velocity field predictions demonstrated the highest percent error.

Visible also in figure 3.7 are slight numerical oscillations that begin at the inlet and dampens towards the exit. This is likely due to instabilities that were added by the solution algorithm with regards to implicit and explicit treatment of interphase momentum transfer, or this could

be due to other structural instabilities contained in the discretized scheme, although the dampening of the solution suggests that the overall system is stable. For the velocity prediction and correction algorithm, special treatment was needed for the momentum equations in order to build a Jacobian that accurately predicted velocity perturbations. It was found that treating the drag term implicitly in the pressure and velocity equations as described in section 2.2.1 resulted in inaccurate predictions of velocity fields. Therefore, drag terms were removed from the pressure equation, and a portion of the drag interphase momentum term was calculated explicitly and added to the source term of the one dimensional multiphase equations. It is unclear why the implicit treatment of drag causes inaccurate approximations of velocity. One possible hypothesis pertains to the predictor-corrector scheme for velocity included in the PISO algorithm. The variable dependence of the Jacobian matrix treats the predicted and corrected velocities the same with respect to their derivatives, when numerically this is not the case. Future work should investigate the numerical dependence and response of Jacobian calculations with respect to predicted and corrected velocities separately.

The previous figures 3.5 through 3.8 used upwind divergence schemes to calculate face interpolated values for the convection of void fraction. Upwind values interpolated to the face are all that is necessary for a one dimensional problem with mass transport occurring only along one axis. However, the two dimensional solutions use the TVD vanLeer interpolation scheme for calculating face interpolated values of dispersed phase void fraction, and so Figure 3.9 shows the results using this interpolation scheme for the perturbed dispersed phase void fraction scheme. While the initial and perturbed solutions are the same as that of Figure 3.5 the Jacobian approximated values oscillate above and below the exact value. The largest relative difference for this case is 11% and the L_2 norm for the perturbed region is $|\vec{\alpha_g}|_2 = 0.02$. This difference is likely due to the highly non-linear nature of the flux weighted vanLeer total variation diminished interpolation scheme.

Figures 3.10 and 3.11 show log-log plots of perturbations in $\vec{\phi}$ verses the difference between the Jacobian calculated solution and the exact value. Upwinding was used for interpolation, and perturbations include all field values. For Figure 3.10, the final cell was omitted in the calculation of the L_2 norm due to some irregularities in the interpolation of the exit value boundary condition. Perturbations only occur in the field of interest for each of the cases. Regressions of the L_2 norm in both figures show a decrease in the error associated with linearizing the operator, i.e. using a Jacobian, as the amount perturbed decreases. The order of error for perturbations in $\vec{\alpha}_g$ is approximately 1.6, and the order of L_2 norm error for perturbations in $\vec{u}_g$ is approximately 0.9. Using a Taylor series introduces a $\mathcal{O}(\Delta^2)$ error in the conservation equations, implying one would have expected an order of L_2 norm error of 2.

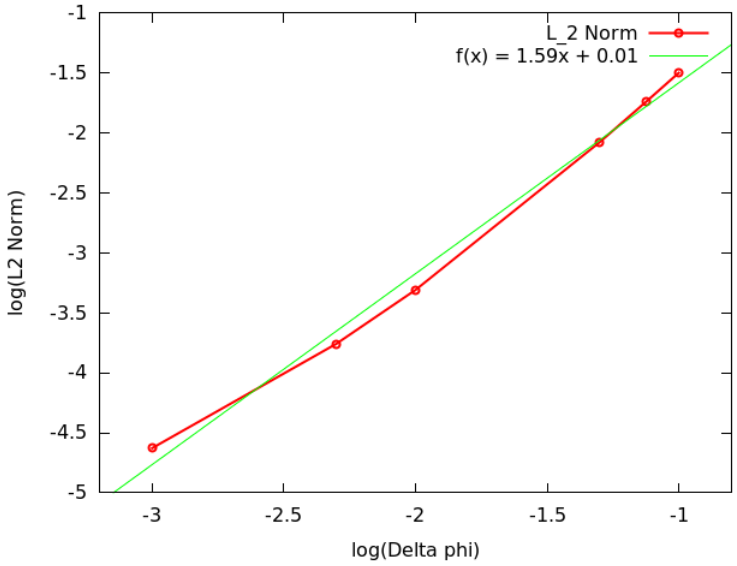


Figure 3.10: Log-log plot of L2 norms verses perturbation size for dispersed phase void fraction solutions with linear regression

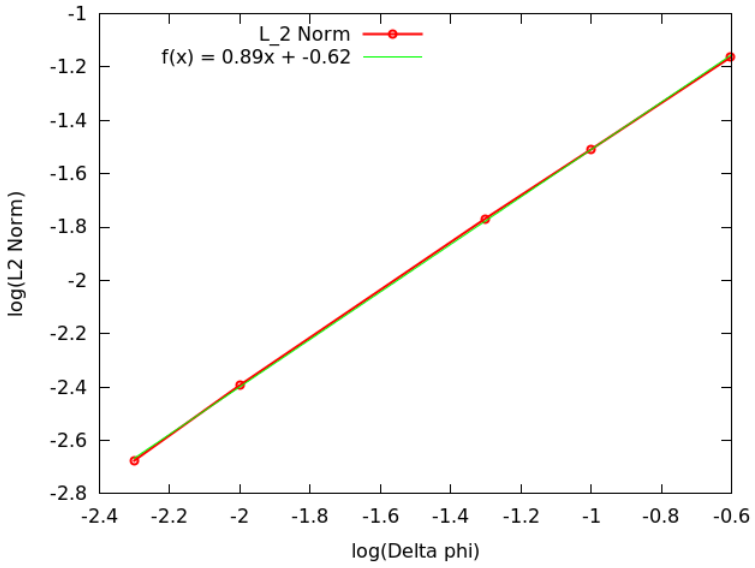


Figure 3.11: Log-log plot of L2 norms verses perturbation size for dispersed phase velocity solutions with linear regression

The results presented in this section give us good confidence that the automatic differentiation capability for calculating Jacobians of nonlinear multiphase systems is producing expected results and is suited for use in calculating multiphase adjoints.

Chapter 4

Adjoint Methodology and Results

This chapter presents the adjoint methodology used with application to the multiphase RANS system of equations and applies it to a preliminary sensitivity analysis of interphase momentum closure relationships. First, we demonstrate two separate methods for deriving adjoints for nonlinear systems of equations - the method of Lagrange multipliers or fluids approach and the generalized functional response or physics approach. These methods are shown to be the same, and both derive the same adjoint operator. Following this, a specific adjoint problem is derived as it relates to the multiphase CFD system of equations with application to sensitivity analysis, and adjoints are used to approximate sensitivity responses for perturbations in interphase momentum transfer coefficients.

4.1 Adjoint Problem Derivation - Fluids vs. Physics Methods

In the derivation of an adjoint problem, there are many degrees of freedom left up to designers that must be properly defined in order to return desired responses. These responses could be the sensitivity of void distribution to interphase momentum constants, nuclear crosssections, or the porosity of a boundary condition made to minimize pressure drop. In either case, the fluids community and neutron physics community have separate methodologies for deriving adjoint problems. The fluids community uses the method of Lagrangian multipliers, while the physics community develops the adjoint directly from the definition of adjoint operators and functional responses.

Because this thesis examines a fluids problem with application to nuclear reactor design, both methods are examined. The first section presents the method of Lagrange multipliers, traditionally used by the fluids community with application to optimization. The second section presents the direct mathematical derivation used in most neutron physics applications and is

applied to a finite-element CFD code.

4.1.1 The Method of Lagrange Multipliers

Consider a two dimensional problem subject to the following constraints

- minimize $f(x, y)$
- $g(x, y) = c$

Points (x^*, y^*) are sought such that $f(x^*, y^*)$ does not change as one incrementally moves while following the contour given by the state equation $g(x, y) = c$. Under such conditions, there is a contour of f such that $f(x, y) = d$ in the vicinity of (x^*, y^*), indicating that the contours of f and g are parallel. The second possibility is that f is level and does not change regardless of the direction.

If both contours are parallel, then the gradients of f and g share the same direction since, by definition, the gradient of a function is perpendicular to its contour. We then have the following expression

$$\nabla f = -\phi^* \nabla g$$

The constant ϕ^* is the Lagrangian multiplier and is necessary to ensure equality due to the fact that the magnitudes of the gradients are not always equal. Writing the Lagrangian or adjoined equation

$$\Lambda(x, y) = f(x, y) + \phi^* \cdot (g(x, y))$$

the point (x, y) can be solved using the previous two expressions, noting that at the stationary point we have

$$\nabla \Lambda(x, y) = 0$$

This gives us an initial motivation for the derivation of an adjoined or adjoint problem using Lagrange multipliers.

Lagrangian Formulation as Derived by Stengel in Stochastic Optimal Control[58]
Starting with a scalar cost function $C(\vec{u})$, we define the following:

- $C(\vec{u})$, scalar cost function

- $\vec{u}$, $m \times 1$ control vector
- $A(\vec{u}) = 0$, scalar equality constraint

where m components of $\vec{u}$ must be specified to constrain the minimum of $C(\vec{u})$. Using the scalar equality constraint, there are now $m - 1$ independent components of $\vec{u}$.

For the Lagrange-Euler equation, we define:

- $\mathbf{A}(\vec{u}')$, $n \times 1$ vector equality constraint
- $\vec{u}'$, $(m + n) \times 1$ vector

The n components of $\vec{u}'$ are specified as functions of the remaining m terms. These remaining m terms are varied until $C(\vec{u}')$ is at a minimum.

Partitioning $\vec{u}$, we have

$$C(\vec{u}') = C(\vec{u}_a, \vec{u}_b), \text{ to be minimized}$$

$$\mathbf{A}(\vec{u}') = \mathbf{A}(\vec{u}_a, \vec{u}_b) = \vec{0}, \quad \vec{u}' = [\vec{u}_a \ \vec{u}_b]^T$$

The n components of $\vec{u}_a$ are specified by $\mathbf{A}(\vec{u}') = \vec{0}$. The m terms of $\vec{u}_b$ are varied to obtain the minimum of $C(\vec{u})$

Let

$$\vec{\phi} = \vec{u}_a$$

$$\vec{u} = \vec{u}_b$$

where $\vec{\phi}$ is the state vector and $\vec{u}$ is the control vector.

In order to find the minimum of the cost function, $A(\vec{u})$ must be substituted into $C(\vec{u})$, which is often difficult. An alternative approach introduces the method of Lagrange Multipliers. Motivated by the previous discussion on function contours, we consider the augmented cost function C_A such that

$$C_A(\vec{\phi}, \vec{u}) = C(\vec{\phi}, \vec{u}) + (\vec{\phi}^*)^T \mathbf{A}(\vec{\phi}, \vec{u})$$

which will be minimized with respect to $\vec{u}$. Here, the inner product of the $n \times 1$ vector of Lagrange multipliers, $\vec{\phi}^*$, and the equations of state is *adjoined* to the original cost function. The stationary condition $\mathbf{A}(\vec{\phi}, \vec{u}) = \vec{0}$ ensures the inner product with the Lagrange multipliers or adjoint solution $\vec{\phi}^*$ always equals zero for $\vec{\phi}$ and $\vec{u}$ that satisfy the state function. The n values of the adjoint $\vec{\phi}^*$ must be found to minimize C_A and subsequently C.

The following specifies the $(m + 2n)$ system of equations

- n for $\vec{\phi}^*$
- n for $\vec{\phi}$
- m for $\vec{u}$

Performing the first order Taylor expansion of the perturbed optimum C_A^* around $\vec{\phi}, \vec{u} = \vec{\phi_o}, \vec{u_o}$

$$\Delta C_A = \left.\frac{\partial C_A}{\partial \vec{\phi}}\right|_{\vec{\phi_o}, \vec{u_o}} \Delta\vec{\phi} + \left.\frac{\partial C_A}{\partial \vec{u}}\right|_{\vec{\phi_o}, \vec{u_o}} \Delta\vec{u} + \mathcal{O}(\Delta^2)$$

with the following partial derivatives

$$\frac{\partial C_A}{\partial \vec{\phi}} = \frac{\partial C}{\partial \vec{\phi}} + (\vec{\phi}^*)^T \frac{\partial \mathbf{A}}{\partial \vec{\phi}}$$
$$\frac{\partial C_A}{\partial \vec{u}} = \frac{\partial C}{\partial \vec{u}} + (\vec{\phi}^*)^T \frac{\partial \mathbf{A}}{\partial \vec{u}}$$

where the partial derivative $\dfrac{\partial \mathbf{A}}{\partial \vec{\phi}}$ defines the $n \times n$ Jacobian matrix

$$\frac{\partial \mathbf{A}}{\partial \vec{\phi}} = \begin{bmatrix} \dfrac{\partial A_1}{\partial \phi_1} & \dfrac{\partial A_1}{\partial \phi_2} & \cdots & \dfrac{\partial A_1}{\partial \phi_n} \\ \dfrac{\partial A_2}{\partial \phi_1} & \dfrac{\partial A_2}{\partial \phi_2} & & \\ \vdots & & \ddots & \\ \dfrac{\partial A_n}{\partial \phi_1} & & & \dfrac{\partial A_n}{\partial \phi_n} \end{bmatrix}$$

and $\dfrac{\partial \mathbf{A}}{\partial \vec{u}}$ defines the $n \times m$ Jacobian matrix

$$\frac{\partial \mathbf{A}}{\partial \vec{u}} = \begin{bmatrix} \dfrac{\partial A_1}{\partial u_1} & \dfrac{\partial u_1}{\partial u_2} & \cdots & \dfrac{\partial A_1}{\partial u_m} \\ \dfrac{\partial A_2}{\partial u_1} & \dfrac{\partial u_2}{\partial u_2} & & \\ \vdots & & \ddots & \\ \dfrac{\partial A_n}{\partial u_1} & & & \dfrac{\partial A_n}{\partial u_m} \end{bmatrix}$$

We require the Lagrangian or adjoined cost function to be stationary with respect to the $\vec{\phi}$ state vector. Starting with the previously defined partial derivative of C_A, we derive an expression for the adjoint vector.

$$\frac{\partial C_A}{\partial \vec{\phi}} = \frac{\partial C}{\partial \vec{\phi}} + (\vec{\phi^*})^T \frac{\partial \mathbf{A}}{\partial \vec{\phi}} = \vec{0}$$

$$(\vec{\phi^*})^T \frac{\partial \mathbf{A}}{\partial \vec{\phi}} = -\frac{\partial C}{\partial \vec{\phi}}$$

$$(\vec{\phi^*})^T = -\left[\left(\frac{\partial C}{\partial \vec{\phi}}\right)\left(\frac{\partial \mathbf{A}}{\partial \vec{\phi}}\right)^{-1}\right]$$

$$\vec{\phi^*} = -\left[\left(\frac{\partial \mathbf{A}}{\partial \vec{\phi}}\right)^{-1}\right]^T \left(\frac{\partial C}{\partial \vec{\phi}}\right)^T$$

This establishes n equations to find $\vec{\phi^*}$. The partial derivatives are evaluated in the vicinity of the stationary point $\vec{\phi} = \vec{\phi}_o,\ \vec{u} = \vec{u}_o$. The first order approximation of perturbations in the cost function now becomes

$$\Delta C_A = \frac{\partial C_A}{\partial \vec{u}} \Delta \vec{u} = \left(\frac{\partial C}{\partial \vec{u}} + (\vec{\phi^*})^T \frac{\partial \mathbf{A}}{\partial \vec{u}}\right) \Delta \vec{u} = 0$$

Note that $\Delta C_A = 0$ when $\vec{u}_o,\ \vec{\phi}_o$ indicates the minimum of C. We then have for arbitrary variations in the control

$$\frac{\partial C}{\partial \vec{u}} + (\phi^*)^T \frac{\partial \mathbf{A}}{\partial \vec{u}} = \vec{0}$$

which provides m equations for the control vector. The state equations

$$\mathbf{A}(\vec{\phi}, \vec{u}) = \vec{0}$$

provides the last n equations to close the system. This method returns the minimum of a cost function subject to state equations without substituting these state equations back into the

original cost function, which in practice is most times not feasible.

4.1.2 Direct Adjoint Derivation - Implementation for Error Estimates in Drekar:CFD [30] - [32]

Drekar is a finite element based CFD code developed by Sandia National Laboratories to address single phase flow problems. This section investigates the adjoint method applied within the code for the purpose of a posteriori error estimation. It follows a set of unlimited release reports authored by Timothy M. Wildey et. al.

The system of partial differential equations is defined as

$$\mathbf{A}(\vec{\phi}) = \vec{0}$$

We define $\vec{e} = \vec{\phi} - \vec{\phi}_o$, and, applying the integral mean value theorem, we have a set of equations for $\mathbf{A}(\bar{\vec{\phi}})$ that lies on the line $\vec{e}$ such that

$$\mathbf{A}'(\bar{\vec{\phi}}) = \frac{\mathbf{A}(\vec{\phi}) - \mathbf{A}(\vec{\phi_o})}{\vec{e}}$$

Taking the Taylor Expansion of $\mathbf{A}(\vec{\phi})$ about $\vec{\phi}_o$, we have

$$\mathbf{A}(\vec{\phi}) = \mathbf{A}(\vec{\phi}_o) + \mathbf{A}'(\vec{\phi}_o)(\vec{\phi} - \vec{\phi}_o) + \mathcal{O}(\Delta^2)$$

Substituting $\vec{e}$ and rearranging terms yields

$$\mathbf{A}'(\vec{\phi}_o)\vec{e} = \mathbf{A}(\vec{\phi}) - \mathbf{A}(\vec{\phi}_o) + \mathcal{O}(\Delta^2)$$

which supplies a linear operator $\mathbf{J}$ such that

$$\mathbf{J}\vec{e} = \mathbf{A}'(\vec{\phi}_o)\vec{e} = \mathbf{A}(\vec{\phi}) - \mathbf{A}(\vec{\phi}_o) + \mathcal{O}(\Delta^2)$$

In this example, $\mathbf{J}$ is the Jacobian matrix $\dfrac{\partial \mathbf{A}}{\partial \vec{\phi}}$ and is used in the computing step for Newtons whose availability is often taken advantage of for the discrete adjoint formulation.

Equating $\vec{\phi}_o$ to the true solution and $\vec{\phi}$ to an approximate solution, we can now solve for the error $\vec{e}$ using the linear operator along with the adjoint expression

$$\mathbf{J}^*\vec{\phi}^* = \vec{Q}^*$$

where

$$\vec{Q}^* = \frac{\partial}{\partial \vec{e}}\left(\vec{f}_R, \vec{e}\right) = \vec{f}_R$$

and $\vec{f}_R$ denotes the response vector. It then follows that

$$\left(\vec{f}_R, \vec{e}\right) = \left(\vec{Q}^*, \vec{e}\right) = \left(\mathbf{J}^*\vec{\phi}^*, \vec{e}\right) = \left(\vec{\phi}^*, \mathbf{J}\vec{e}\right) = \left(\vec{\phi}^*, \mathbf{A}(\vec{\phi}) - \mathbf{A}(\vec{\phi}_o) + \mathcal{O}(\Delta^2)\right) = \left(\vec{\phi}^*, \vec{r}\right) + \mathcal{O}(\Delta^2) \tag{4.1}$$

where the residual is defined as

$$\vec{r} = \mathbf{A}(\vec{\phi}) - \mathbf{A}(\vec{\phi}_o)$$

Similar to the generalized physics approach, the result is a weighted residual given by the inner product $\left(\vec{\phi}^*, \vec{r}\right)$ that approximates the response.

Using the definition of the adjoints of matrices, the adjoint linear operator $\mathbf{J}^*$ is simply the transpose of the Jacobian matrix [65].

$$J^* = J^T$$

Therefore, solving the adjoint equation with the appropriate Jacobian matrix leads to a first order accurate approximation of the error in the response of approximating $\mathbf{A}(\vec{\phi})$ about an initial solution $\vec{\phi}_o$.

The adjoint approach can be used both for a posteriori error estimation as well as sensitivity parameter analysis. In each case, the adjoint formulation is only dependent on the definition of the response vector $\vec{f}_R$. The adjoint solution is not dependent on the residual, and a variety of conditions and quantities of interest can be estimated using a single adjoint solution.

4.1.3 Comparison of Lagrangian and Physics Methodology

The physics and fluids communities each derive the adjoint problem using two separate methodologies, described here as the Physics and Lagrangian methods respectively. The method presented in the previous section is the generally accepted adjoint problem derivation for nonlinear systems as pioneered by the neutron physics community. Though this is the commonly accepted

methodology, the computational fluids community often prefers the method of Lagrange Multipliers for optimizing a cost function. Here, we compare the Lagrangian and Physics approach and show that the methods are one in the same

Lagrangian:

$$(\vec{\phi}^*)^T \frac{\partial \mathbf{A}}{\partial \vec{\phi}} = -\frac{\partial C}{\partial \vec{\phi}}$$

Physics:

$$\left(\frac{\partial \mathbf{A}}{\partial \vec{\phi}}\right)^T \vec{\phi}^* = \vec{Q}^*$$

Starting with the Lagrangian form of the adjoint equation, we can derive the physics form with a given right hand side for the optimization problem.

$$(\vec{\phi}^*)^T \frac{\partial \mathbf{A}}{\partial \vec{\phi}} = -\frac{\partial C}{\partial \vec{\phi}}$$

$$\left((\vec{\phi}^*)^T \frac{\partial \mathbf{A}}{\partial \vec{\phi}}\right)^T = \left(-\frac{\partial C}{\partial \vec{\phi}}\right)^T$$

$$\left(\frac{\partial \mathbf{A}}{\partial \vec{\phi}}\right)^T \vec{\phi}^* = \left(-\frac{\partial C}{\partial \vec{\phi}}\right)^T$$

This suggests that for quantities of interest relating to optimization, e.g. the extrema of cost function derivatives, the right hand side of the physics formulation must be such that

$$\vec{Q}^* = \left(-\frac{\partial C}{\partial \vec{\phi}}\right)^T$$

Regardless the nature of $\vec{Q}^*$, both methods require the transpose of the Jacobian matrix of the forward state equations in order to solve for the adjoint solution.

4.2 Derivation of Multiphase Adjoint Problem with Application to Sensitivity Analysis

We use a similar methodology to the physics approach in order to derive the adjoint problem for the multiphase CFD equations with application to sensitivity analysis.

The nonlinear multiphase RANS system of equations is written as

$$\mathbf{F}(\vec{\phi}) = \vec{d} \tag{4.2}$$

where $\mathbf{F}$ contains the multiphase continuity, momentum, and turbulence equations for adiabatic flow, and the $\vec{\phi}$ contains solutions for void fraction, velocity, pressure, turbulent kinetic energy, and turbulent dispersion according to

$$\vec{\phi} = \left[\vec{\alpha}_g \; \vec{\alpha}_l \; \vec{u_g}_x \; \vec{u_g}_y \; \vec{u_l}_y \; \vec{u_l}_y \; \vec{p} \; \vec{k}_l \; \vec{\epsilon}_l\right]^T$$

For sensitivity analysis, we are interested in a perturbed problem

$$\mathbf{F}'(\vec{\phi}') = \vec{d}' \tag{4.3}$$

with perturbed solution

$$\vec{\phi}' = \vec{\phi} + \Delta\vec{\phi} \tag{4.4}$$

and perturbed source

$$\vec{d}' = \vec{d} + \Delta\vec{d} \tag{4.5}$$

Writing

$$\mathbf{F}'(\vec{\phi}') = \mathbf{F}(\vec{\phi}) + \Delta\mathbf{F}(\Delta\vec{\phi}) \tag{4.6}$$

and substituting 4.2 and 4.3 into equation 4.6, we have

$$\Delta\mathbf{F}(\vec{\phi}) = \Delta\vec{d} \tag{4.7}$$

Using a Taylor's series expansion, we have a first order approximation of $\Delta\vec{d}$ given by

$$\Delta\vec{d} = \frac{\partial\vec{d}}{\partial a}\Delta a + \mathcal{O}(\Delta^2) \tag{4.8}$$

where Δa is some perturbation parameter a appearing in the source term. Because we are seeking the sensitivity with respect to parameter a, there is only one partial derivative in equation 4.8.

In the same way, if the nonlinear operator $\mathbf{F}$ is also dependent upon parameter a, then a first order approximation of $\Delta\mathbf{F}$ is given by

$$\Delta\mathbf{F} = \frac{\partial\mathbf{F}(\vec{\phi})}{\partial a}\Delta a + \frac{\partial\mathbf{F}(\vec{\phi})}{\partial\vec{\phi}}\Delta\vec{\phi} + \mathcal{O}(\Delta^2) \tag{4.9}$$

Rearranging equation 4.9, ignoring higher order terms, and substituting in equations 4.7 and

4.8, we have the following expression

$$\frac{\partial \mathbf{F}(\vec{\phi})}{\partial \vec{\phi}}[\Delta\vec{\phi}] \approx \frac{\partial \vec{d}}{\partial a}\Delta a - \frac{\partial \mathbf{F}(\vec{\phi})}{\partial a}\Delta a \tag{4.10}$$

Equation 4.10 is a linear problem that solves for perturbed solutions of $\vec{\phi}$ using a right hand side that is not dependent on $\Delta\vec{\phi}$.

Let $\mathbf{J}$ be the Jacobian matrix of $\mathbf{F}$ with respect to $\vec{\phi}$. We then have the following forward problem

$$\mathbf{J}[\Delta\vec{\phi}] = \vec{Q} \tag{4.11}$$

with

$$\vec{Q} = \frac{\partial \vec{d}}{\partial a}\Delta a - \frac{\partial \mathbf{F}(\vec{\phi})}{\partial a}\Delta a \tag{4.12}$$

and adjoint problem

$$\mathbf{J}^*[\vec{\phi}^*] = \vec{Q}^* \tag{4.13}$$

where $\mathbf{J^T} = \mathbf{J}^*$ according to the definition of the adjoint of a matrix [65]. In order to properly define $\vec{Q}^*$ for the adjoint problem, we define the following inner product of a response function

$$\Delta R = \left(\vec{f}_R, \Delta\vec{\phi}\right) \tag{4.14}$$

This response is used to define $\vec{Q}^*$. For example, if the desired ΔR response is the change in void fraction α_g in a specific control volume, then the response function is

$$\vec{Q}^* = \vec{f}_R \tag{4.15}$$

where the discrete response vector is

$$(\vec{f}_R)_i = \begin{cases} 1, & \text{index } i \text{ corresponds to } \alpha_g \text{ for specific control volume} \\ 0, & \text{otherwise} \end{cases}$$

Examining the inner product of $\vec{Q}^*$ and $\Delta\vec{\phi}$ and applying the definition of mathematical adjoints

$$\Delta R = \left(\vec{f}_R, \Delta\vec{\phi}\right) = \left(\vec{Q}^*, \Delta\vec{\phi}\right) = \left(\mathbf{J}^*[\vec{\phi}], \Delta\vec{\phi}\right) = \left(\vec{\phi}^*, \mathbf{J}[\Delta\vec{\phi}]\right) = \left(\vec{\phi}^*, \vec{Q}\right) \tag{4.16}$$

which gives us the response ΔR in terms of $\vec{\phi}^*$ and $\vec{Q}$ only. Because $\vec{Q}$ is dependent on perturbed operators as shown in equation 4.12, we can estimate responses for various perturbed parameters of $\mathbf{F}$ without needing to resolve for $\Delta\vec{\phi}$. The equality in equation 4.16 is exact, and the only approximation occurs in the linearization of the $\mathbf{F}$ operator. Therefore, adjoint approximations of sensitivities is limited by the linearization of the $\mathbf{F}$ operator.

4.3 Adjoint Sensitivity Analysis of Interphase Momentum Transfer Terms

This section investigates perturbations in interphase transfer coefficients and uses Jacobian matrices and adjoint calculations to approximate the response of these perturbations with respect to dispersed phase void profiles. As shown previously in section 2.3, dispersed phase void profiles are dependent on interphase momentum transfer closure laws, and sensitivities of these parameters are of interest to multiphase CFD developers.

The perturbations are carried out on the multiphase equations adjusted for highest Jacobian accuracy as described in section 3.3.1. Specifically, drag terms are removed from the pressure equation and placed into the momentum equation source term. Due to the large memory requirements of automatic differentiation and the storage of full Jacobian matrices, the size of the channel was drastically reduced. Channel size is, however, long enough to establish a preliminary void profile for use in adjoint sensitivity analysis.

Table 4.1 shows input parameters for adjoint sensitivity calculations of interphase momentum transfer terms. The perturbed case using altered interphase momentum coefficients is solved iteratively along with the initial case. Upwinding is used to calculate the interpolated convection terms with regards to dispersed phase void fraction.

Figure 4.1 shows the two dimensional pipe mesh used for the interphase momentum perturbation response studies. This figure is to scale. Flow still occurs from left to right, and gravity acts in the negative z direction. The mesh size is reduced to 10 radial cells by 20 axial cells due to the memory requirements of Jacobian construction. This reduced mesh size is still able to resolve initial flow fields and perturbations in void fraction with respect to interphase momentum transfer closure relationships. Equation 4.10 is used to calculate an approximate $\Delta\tilde{\phi}$ response, and this approximation is compared to the exact difference between the initial and perturbed cases. An intrinsic to OpenFOAM LU solver with partial pivoting is used to solve for the Jacobian approximated solution.

Table 4.1: **Adjoint Sensitivity Response Input Parameters**

Parameter	**OF Variable**	**Value**
Pipe Diameter	Dh	1.5cm
Pipe Length	L	15cm
Inlet Pressure	p	4.5 MPa
Inlet Void	alphaair	0.1
Inlet Liquid Velocity	Uwater	1.01 m/s
Inlet Vapor Velocity	Uair	1.0 m/s
Bubble Diameter	Ds	0.7mm
Wall Heat Flux	qWall	0 kW

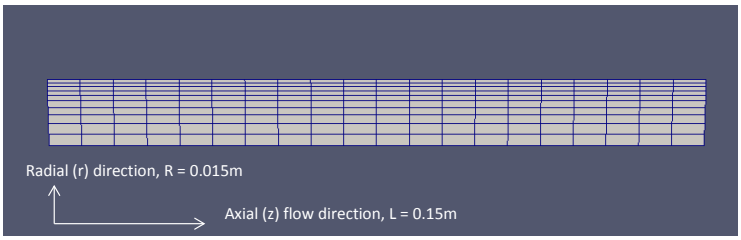


Figure 4.1: Representative mesh for the 2D pipe with radial symmetry used in the interphase momentum transfer coefficient perturbation response study

Each interphase momentum transfer coefficient was held constant for its respective perturbation study. This is required according to the formulation shown in equation 4.12, since it is assumed that Δa is a scalar value. The right hand side $\vec{Q}$ for the forward problem is calculated using automatic differentiation with respect to the perturbed parameter Δa according to equation 4.12. If a were a vector of values, then partial derivatives with respect to $\vec{a}$ would require additional Jacobian construction, further limiting available memory,. Alternatively, $\vec{Q}$ could be expressed by $\vec{Q} = \Delta\vec{d} + \left(\mathbf{F}'[\vec{\phi}] - \mathbf{F}[\vec{\phi}]\right)$. This preliminary sensitivity analysis examines changes in the void profile for perturbations in drag, lift, and wall lubrication coefficients. Perturbations are carried out on steady state solutions.

This sensitivity study only examines perturbations in void $\Delta\vec{\alpha}_g$ for specific regions of interest as defined by $\vec{Q}^*$ in equation 4.15. For simplicity,

$$(\vec{Q}^*)_i = \begin{cases} 1, & \text{index } i \text{ corresponds to } \alpha_g \text{ for specific control volumes} \\ 0, & \text{otherwise} \end{cases}$$

and the inner product $(\vec{Q}^*, \Delta\vec{\phi})$ returns the summation of $\Delta\vec{\alpha}_g$ for various spatial regions of interest within the pipe. The response for various $\vec{Q}^*$ functionals is calculated using the exact $\Delta\vec{\phi}$ perturbed values and the adjoint $\vec{\phi}^*$ solution according to equation 4.16.

4.3.1 Adjoint Sensitivity Response Prediction for Drag Interphase Momentum Transfer Coefficient Perturbations

This section investigates perturbations in the drag interphase momentum transfer coefficient and approximated adjoint responses. The modified drag coefficient used to calculate the sensitivity response is given by the following equation

$$\vec{M}_g^D = -C_D \frac{3}{4} \frac{\rho_l}{D_S} \alpha_g \left|\vec{u}_r\right| (\vec{u}_r) \tag{4.17}$$

where the relative velocity is $\vec{u}_r = \vec{u}_g - \vec{u}_l$ and C_D was held constant for the initial case and perturbed by a percentage for the sensitivity responses. This drag expression returns the actual term added to the dispersed phase momentum equation and therefore includes the $\frac{1}{\rho_g}$ term. Table 4.2 lists the physical parameters and interphase momentum transfer models used for this drag sensitivity study.

Table 4.2: **Drag Sensitivity Response Parameters and Interphase Momentum Models**

Parameter	**OF Variable**	**Value**
Bubble Diameter	Ds	0.7mm
Frank Wall Model Coefficients	Cwc, Cwd, p	10.0, 6.8, 1.7
Constant Lift Coefficient	Cl	0.001
Constant Turbulent Dispersion Coefficient	Ctd	0.01
Constant Virtual Mass Coefficient	Cvm	0.5
Constant Base Drag Coefficient	Cd	1.6

This drag sensitivity response analysis increased the drag coefficient by 10% and solved the per-

turb case iteratively alongside of the initial case using listed parameters. Figures 4.2 through 4.4 show initial and perturbed void profiles for three axial locations.

In Figure 4.2, the initial $\vec{\alpha}_g$ profile at 2cm is shown using a red line and the perturbed profile is shown using a dashed green line. The bulk void is 0.002 lower than the inlet value of 0.1, and the void peaks at 0.12 in the near-wall region. The sharpness of this peak distribution is due to the coarseness of the mesh. There is little visible difference at this axial location between the initial and perturbed solutions, although the peak void for the perturbed case is slightly lower than the initial case.

In Figure 4.3, the difference at 7cm between the initial red line and perturbed green line is noticeable in the near-wall peak region. The initial case still shows a sharp peak in void while the perturbed case is rounded. Figure 4.4 shows the pipe exit void profile, and again the differences between the perturbed profile and initial profile are small but still visible in the near-wall peak.

Figure 4.5 plots the exact $\Delta\vec{\alpha}_g$ perturbation alongside the approximated $\Delta\tilde{\alpha}_g$ perturbation. The red line denotes the exact perturbation and the green dashed line denotes the approximation calculated using the Jacobian matrix. The vectors are indexed according to OpenFOAM's mesh cell ordering scheme, which is specified by the cell index vector $\vec{C}_{r,l}$ where r is the radial index and l is the axial index of the pipe. If there are R radial cells and L axial cells, then $\vec{C}_{r,l}$ is indexed according to

$$\vec{C}_{r,l} = [C_{0,0}\ C_{0,1}\ C_{0,2}\ \ldots\ C_{0,L}\ C_{1,0}\ C_{1,1}\ \ldots\ C_{R,L}]^T$$

Figure 4.5 shows that the largest perturbations in $\vec{\alpha}_g$ occur along the wall. This is consistent with earlier plots of perturbed void profiles. The Jacobian approximation replicates the general shape of the exact perturbation, especially in areas where the magnitude is larger than 0.001. At index locations 0 through 100, the Jacobian returns a reflected response across the $x = 0$ axis. The reason for this numerical behavior is unknown. For indexes larger than 100, the approximated perturbation follows the exact value above and below the $x = 0$ axis, undershooting the highest perturbation of $\Delta\vec{\alpha}_g = 0.01$ by around 20%.

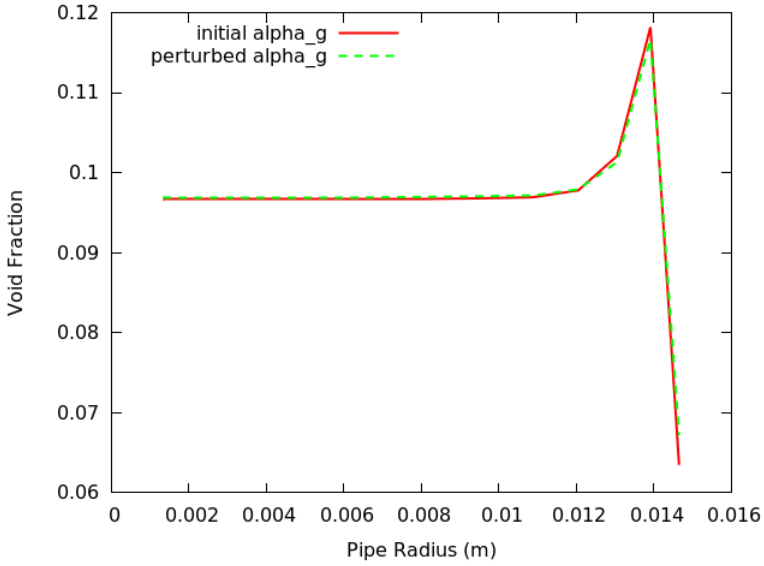


Figure 4.2: Initial and Perturbed void profiles for a 10% increase in the drag coefficient for axial location $L = 2cm$

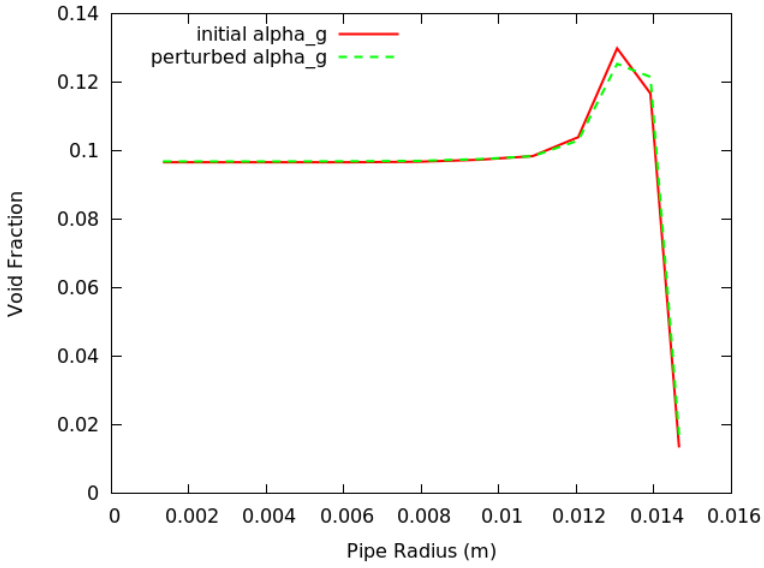


Figure 4.3: Initial and Perturbed void profiles for a 10% increase in the drag coefficient for axial location $L = 7cm$

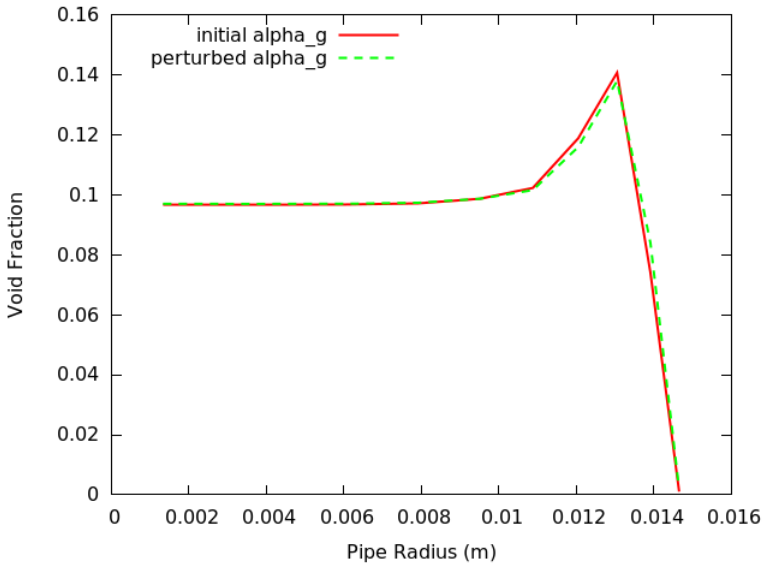


Figure 4.4: Initial and Perturbed void profiles for a 10% increase in the drag coefficient for axial location $L = 15cm$

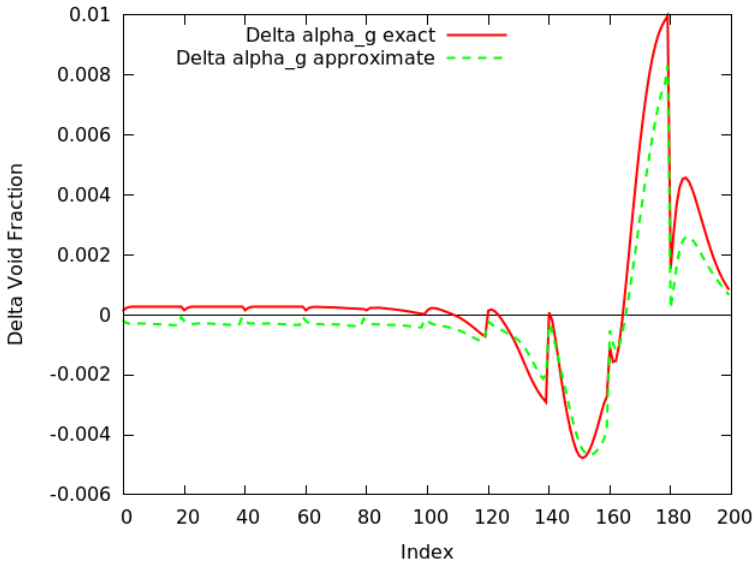


Figure 4.5: Exact perturbations $\Delta\vec{\alpha}_g$ and Jacobian approximated $\Delta\tilde{\alpha}_g$ for pipe mesh cell indices for perturbed drag coefficients

Figure 4.6 shows the log-log plot of Δa perturbation size verses the L_2 normal of the difference between the exact $\Delta\vec{\alpha}_g$ perturbation and the approximate $\Delta\tilde{\alpha}_g$ perturbation. For the drag perturbed case, $\Delta a = \Delta C_D$ where C_D is the drag coefficient. There is a strong log-linear

relationship between the L_2 norm and ΔC_D whose slope is close to 1. A slope of 1 suggests a linear relationship between the L_2 norm and ΔC_D. An order of 2 is expected for Taylor series approximations that ignore $\mathcal{O}(\Delta^2)$ terms.

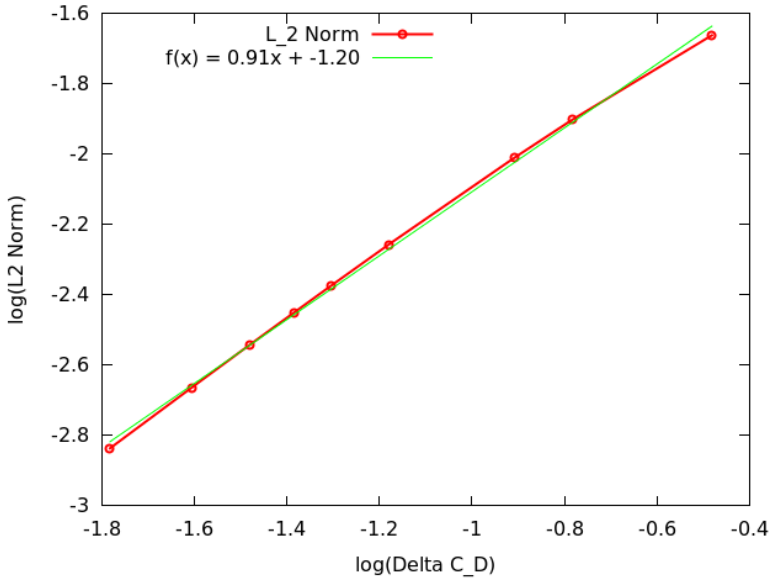
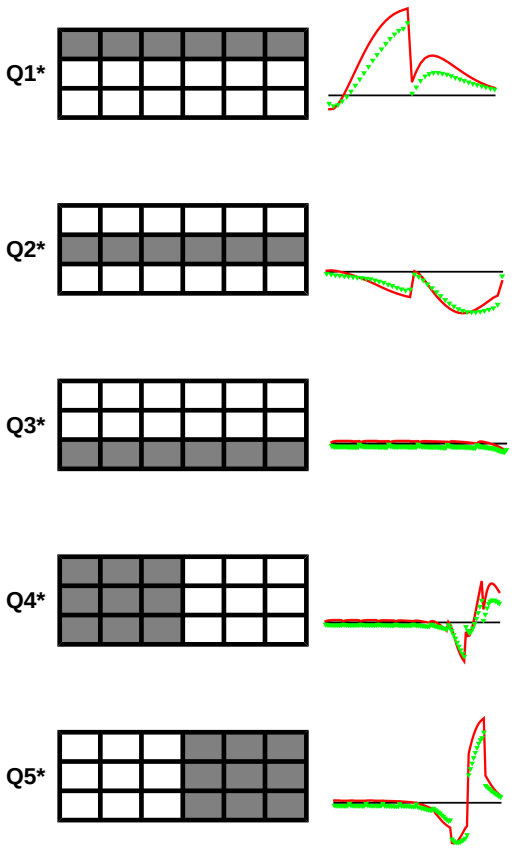


Figure 4.6: Log-log plot of $\log(\Delta C_D)$ vs. $\log(||\Delta\vec{\alpha}_g - \Delta\tilde{\alpha}_g||_2)$ and linear regression

From Figures 4.5 and 4.6, it can be said that the Jacobian matrix is approximating perturbations in void with a reasonable order of error for a 10% increase in drag coefficient. These exact and approximate values were then used to calculate perturbation responses for various locations of interest as defined by the functional $\vec{Q}^*$. It's clear from Figure 4.5 that functionals capturing near wall locations will not only capture the largest of the perturbations, but will also return more favorable responses. Bulk flow regions, however, will not return accurate responses if judged by percntage error in predicting the perturbation, but in terms of absolute error this error is small.

Figure 4.7 shows the exact and adjoint calculated responses for five regions of interest within the pipe. Note that for a linear operator where only the source term is perturbed, the adjoint method is exact, and the approximate (Jacobian based) and adjoint approximated perturbation values are equal. The graphic on the left hand side shows a primitive diagram of $\vec{Q}^*$ spatial locations in the pipe that were used to generate the response. The mesh used is still the 10×20



Exact Response	Adjoint Response
0.149	0.094

Exact Response	Adjoint Response
-0.087	-0.086

Exact Response	Adjoint Response
0.021	-0.042

Exact Response	Adjoint Response
0.034	-0.024

Exact Response	Adjoint Response
0.049	-0.009

Figure 4.7: Exact and adjoint responses with respect to drag coefficient perturbations for various $\vec{Q}^*$ regions of interest

axially symmetric pipe from Figure 4.1, and the simplified drawing only illustrates the general location of $\vec{Q}^*$ used for the response. The plot in the middle shows the exact and adjoint $\Delta\vec{\alpha}_g$ used to generate the response along with the $x = 0$ axis for reference. These reference plots utilize a consistent scale to show identical ranges of perturbations.

All responses are calculated by using the summation of the products of $\left(\Delta\vec{\phi}, \vec{Q}^*\right)$ or $\left(\vec{\phi}^*, \vec{Q}\right)$ as defined by equation 4.16. $\vec{Q}1^*$ returns a response for the radial mesh rings against the wall of the pipe (indices 160-200). This region captures the largest perturbations in $\Delta\vec{\alpha}_g$. The sign and order of magnitude of the adjoint response are correct compared to the exact response, and the adjoint approximates the value with 37% relative error. $\vec{Q}2^*$ returns a response for near-wall radial mesh rings (indices 120-160) and captures the peak void fraction distribution according to Figures 4.2 through 4.4. This region of interest returns the most accurate response approximation with only 1.1% relative error. $\vec{Q}3^*$ returns a response for the bulk flow region of the pipe. This is the region where approximated $\Delta\vec{\alpha}_g$ is a reflection of the exact perturbation. Here, the sign of the approximated response is incorrect due to this reflection, though the magnitude of the perturbation in this region is small. $\vec{Q}4^*$ and $\vec{Q}5^*$ return responses for the entrance and exit of the pipe, and both also return incorrect responses due to inaccuracies in the bulk flow. Responses in these regions are likely not valuable due to positive-negative summation of $\Delta\vec{\alpha}_g$ terms.

It could be possible to use the magnitude of $|\Delta\tilde{\phi}|$ for calculating responses, especially because positive-negative summation can mask perturbations. However, properly defining $\vec{Q}^*$ to return magnitudes is problematic. The inner product relationship shown in equation 4.16 must hold for the adjoint response to be accurate. It is possible to force $\vec{Q}^*$ to be positive or negative depending on the value of the approximated $\Delta\tilde{\phi}$, but this assumes prior knowledge of perturbation profiles, and the exact $(\vec{Q}^*, \Delta\vec{\phi})$ response will not be correct.

For all five functionals, the adjoint calculated response is exactly equal to the approximate response according to equation 4.16, as noted earlier. This is because we are using the mathematical adjoint, and $\vec{\phi}^*$ returns the exact response when using the transpose of the Jacobian matrix. In this way, we can calculate approximations for various parameter Δa perturbation sizes for a fixed location of interest without performing any additional linear solves. The only necessary calculation is the calculation of $\vec{Q} = \dfrac{\partial\vec{d}}{\partial a}\Delta a - \dfrac{\partial\mathbf{F}(\vec{\phi})}{\partial a}\Delta a$, which is trivial using automatic differentiation with a single derivative.

Table 4.3 shows execution times for various calculations performed for this adjoint pertur-

bation sensitivity analysis.

Table 4.3: **Execution Times for Drag Perturbation Response**

Initial and Perturbed Solution Calculation	13.7s
Jacobian Initialization	93.2s
Jacobian Transpose	18.3s
$Q(\Delta a)$ **Calculation**	1.4s
Adjoint Solution Using Five $\vec{Q}^*$ **Locations**	0.2s

The two shortest calculations are the determination of $\vec{Q}$ which is required to evaluate the $(\vec{\phi}^*,\ \vec{Q})$ response and adjoint linear solves, the latter of which takes only two-tenths of a second due to the fact that $\vec{Q}^*$ is mostly zero. The initialization and transposition of the Jacobian takes the longest amount of time due to the large number of calculations required by automatic differentiation in the forward sense. Because the response $(\vec{Q},\ \vec{\phi}^*)$ requires only the adjoint solution, multiple responses of perturbed parameters can be determined at a fraction of the computational cost of the forward solve. In a robust sensitivity analysis that requires a distribution of responses for a given perturbation, this computational efficiency is highly advantageous.

It could be possible to overcome the large memory requirement of Jacobian initialization by altering automatic differentiation to store sparse matrices instead of full matrices in the same fashion as the existing OpenFOAM LDU matrix methodology.

4.3.2 Adjoint Sensitivity Response Prediction for Lift Interphase Momentum Transfer Coefficients

This section investigates perturbations in the lift interphase momentum transfer coefficient and approximated adjoint responses. The same parameters contained in table 4.4 and the mesh shown in Figure 4.1 are used. The modified lift interphase momentum transfer term is given by the following equation

$$\vec{M}_L = C_L \alpha_g \left(\vec{u}_r \times \nabla \times \vec{u}_l\right)$$

where C_L is held constant and perturbed by a factor. Table 4.4 shows interphase coefficients used for the lift perturbation response study.

Table 4.4: **Lift Sensitivity Response Parameters and Interphase Momentum Models**

Parameter	**OF Variable**	**Value**
Bubble Diameter	Ds	0.7mm
Frank Wall Model Coefficients	Cwc, Cwd, p	10.0, 6.8, 1.7
Constant Lift Coefficient	Cl	0.8
Constant Turbulent Dispersion Coefficient	Ctd	0.01
Constant Virtual Mass Coefficient	Cvm	0.5
Drag Model	Ishii-Zuber	

In order to create perturbations large enough for analysis, this lift response study increased the base lift coefficient by several orders of magnitude to 0.8 and perturbed the lift coefficient by a factor of 5 or by 500%. This is significantly larger than the previous drag coefficient study which only perturbed the coefficient by a factor of 0.1 or 10%. Figures 4.8 through 4.10 show initial and perturbed void profiles at three axial locations. The red line denotes the initial case void profile, and the dashed green line denotes the perturbed void profile using the increased lift coefficient.

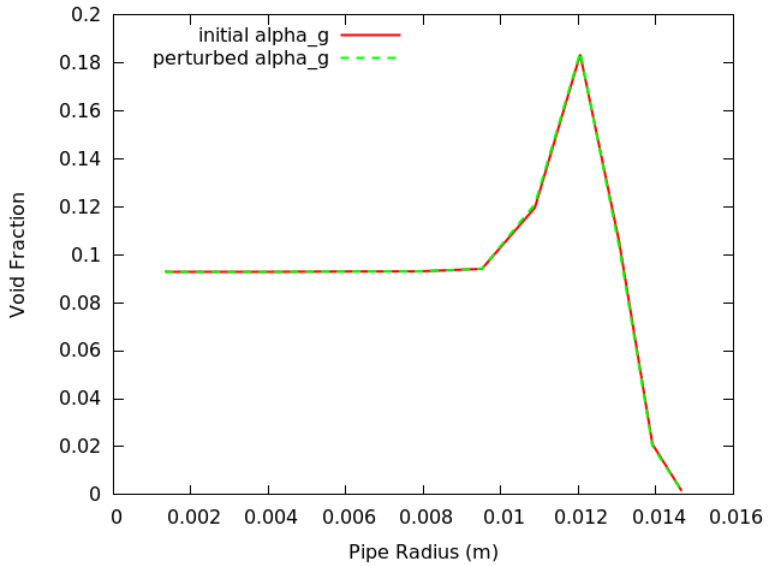


Figure 4.8: Initial and Perturbed void profiles for a 500% increase in the lift coefficient for axial location $L = 2cm$

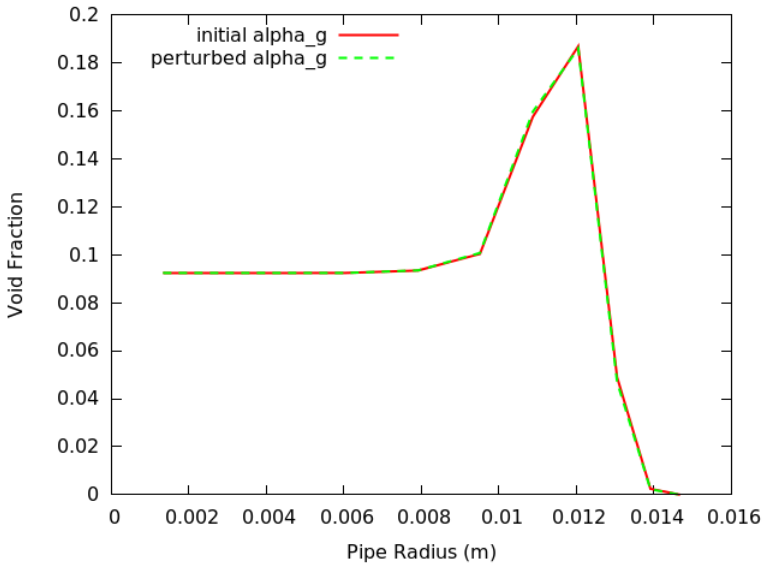


Figure 4.9: Initial and Perturbed void profiles for a 500% increase in the lift coefficient for axial location $L = 7cm$

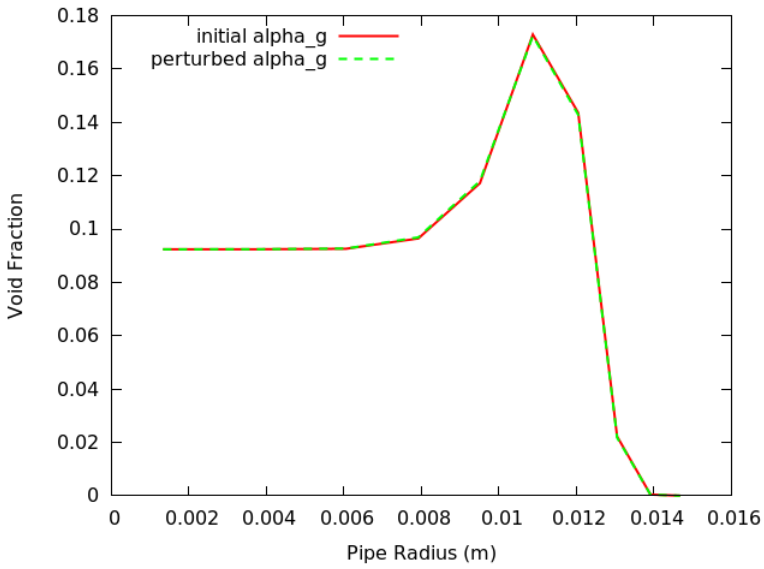


Figure 4.10: Initial and Perturbed void profiles for a 500% increase in the lift coefficient for axial location $L = 15cm$

The void profiles shown in Figures 4.8 through 4.10 show a similar near-wall peak as shown previously in Figures 4.2 through 4.4. Again, the bulk flow is slightly lower than the initial value of 0.1, and there is a near-wall peak in void with a sharp drop in void at the wall. The difference in shape compared to the constant drag analysis is due to the Ishii-Zuber drag coefficient model and the larger constant lift coefficient, the latter of which can account for the larger near-wall peak region. Even though the lift coefficient has been perturbed by 500%, the initial and perturbed void profiles look almost identical at all three axial locations. In Figure 4.9, there is a small but noticeable perturbation at $r = 0.011cm$. There are differences at other locations which is demonstrated by the next figure, but they are small compared to the previous drag analysis.

Figure 4.11 plots the exact $\Delta\vec{\alpha}_g$ perturbation alongside the approximated $\Delta\tilde{\alpha}_g$ perturbation. The red line denotes the exact perturbation and the green dashed line denotes the approximation calculated using the Jacobian matrix. The cell indexing scheme is the same as used in section 4.3.1.

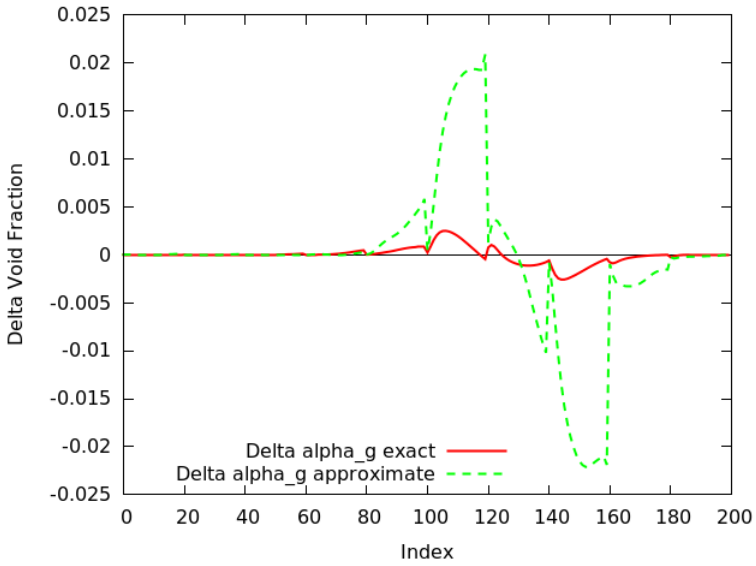


Figure 4.11: Exact perturbations $\Delta\vec{\alpha}_g$ and Jacobian approximated $\Delta\tilde{\alpha}_g$ for pipe mesh cell indices

Figure 4.11 shows that the largest perturbations occur in the near-wall region of the pipe. This is consistent with the only visible perturbation in Figure 4.9. The red line in Figure 4.11

shows the maximum exact perturbation to be only 0.0025, significantly lower than the previous drag coefficient analysis. The green dashed line shows the Jacobian approximated perturbation overshoots the exact perturbation by a factor of 9. The Jacobian approximated perturbation does, however, follow the general shape and sign of the exact perturbation in the near wall region.

Figure 4.12 shows the log-log plot of ΔC_L perturbation size verses the L_2 normal of the difference between the exact $\Delta\vec{\alpha}_g$ perturbation and the approximate $\Delta\tilde{\alpha}_g$ perturbation. There is again a strong log-linear relationship for larger ΔC_L perturbations, and the slope is equal to 1.

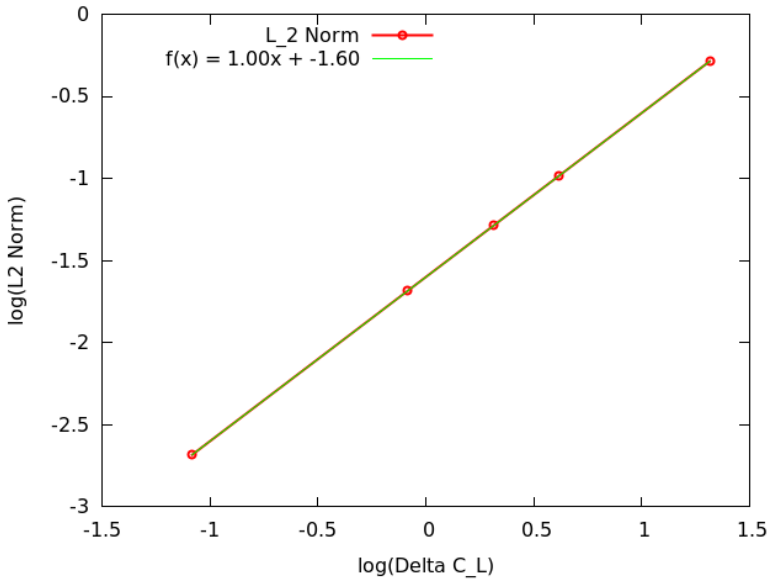


Figure 4.12: Log-log plot of $\log(\Delta C_L)$ vs. $\log(||\Delta\vec{\alpha}_g - \Delta\tilde{\alpha}_g||_2)$ and linear regression

Figures 4.11 and 4.12 show that the Jacobian approximated calculation of $\vec{\alpha}_g$ with respect to perturbed lift coefficients still can return some information about the perturbation response, but it is less accurate than the drag perturbation study. While the magnitude of the Jacobian approximated perturbation is too large, it does return the correct sign and shape of the perturbations, and the reflection across the $x = 0$ axis from figure 4.5 does not occur.

Figure 4.13 shows the exact and adjoint calculated responses for five regions of interest within the pipe. The wall and near-wall regions for $\vec{Q}_1^*$ and $\vec{Q}_2^*$ have been adjusted to capture

the largest positive and negative perturbed solutions from Figure 4.11 to avoid positive-negative summation. Again, the graphic on the left hand side shows a primitive diagram of $\vec{Q}^*$ spatial locations in the pipe that were used to generate the response. The plot in the middle shows the exact and adjoint $\Delta\vec{\alpha}_g$ used to generate the response along with the $x = 0$ axis for reference. These reference plots are consistently scaled to show identical ranges of perturbations.

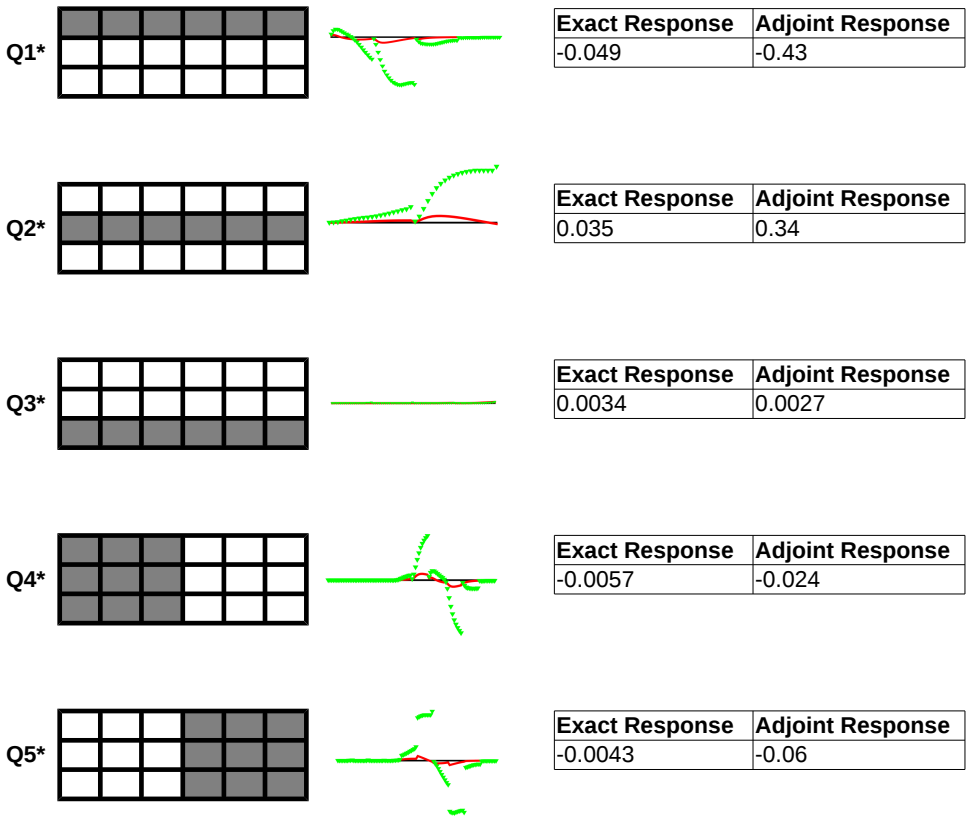


Figure 4.13: Exact and adjoint responses with respect to lift coefficient perturbations for various $\vec{Q}^*$ regions of interest

All of the adjoint approximated responses in Figure 4.13 with exception to $\vec{Q}_3^*$ return values that are an order of magnitude larger than the exact response. This is consistent with results

shown in Figure 4.11. Although the order of magnitude is incorrect, the sign is correct and the approximated responses stay around a factor of 10 times the exact response. The adjoint approximated response of 0.0027 returned by $\vec{Q}_3^*$ for the bulk flow region (indices 0-80) is the closest to the exact value of 0.0034 with a 21% relative error.

Possible explanations for the discrepancy in magnitude could include the significantly larger perturbation percentage (500%) as compared to the perturbed drag coefficient case. Also, the expression used to calculate the lift force is significantly less linear than the expression used to calculate drag. Lift requires the calculation of $\vec{u}_r \times \nabla \times \vec{u}_l$, where drag is simply a factor of $|\vec{u}_r|\,\vec{u}_r$. It is curious that the magnitude difference remains fairly constant in Figure 4.11, but the linear relationship between the L_2 norm and ΔC_L may indicate that the higher order terms are more significant with respect to lift force calculations.

4.3.3 Adjoint Sensitivity Response Prediction for Wall Lubrication Interphase Momentum Transfer Coefficients

This section investigates perturbations in the wall lubrication interphase momentum transfer coefficient and approximated adjoint responses. The same parameters contained in table 4.2 and the mesh shown in Figure 4.1 are used. The modified wall interphase momentum transfer term is given by the following equation

$$\vec{M}_W = C_W \alpha_g \rho_l \left|\vec{u}_r - (\vec{u}_r \cdot \vec{n}_w)\,\vec{n}_w\right|^2$$

where $\vec{n}_w$ is the wall normal direction. Table 4.5 shows interphase coefficients used for the lift perturbation response study.

Table 4.5: **Wall Lubrication Sensitivity Response Parameters and Interphase Momentum Models**

Parameter	**OF Variable**	**Value**
Bubble Diameter	Ds	0.7mm
Constant Wall Lubrication Coefficient	Cw	5.0
Constant Lift Coefficient	Cl	0.001
Constant Turbulent Dispersion Coefficient	Ctd	0.01
Constant Virtual Mass Coefficient	Cvm	0.5
Drag Model	Ishii-Zuber	

The constant wall lubrication coefficient $C_W = 5.0$ was perturbed by 75% to generate an appropriately perturbed $\vec{\alpha}_g$ solution for the response analysis. This is a larger perturbation than what was used in the drag response analysis, but it's an order of magnitude smaller than what was used in the lift response analysis. Figures 4.14 through 4.16 show initial and perturbed void profiles at three axial locations.

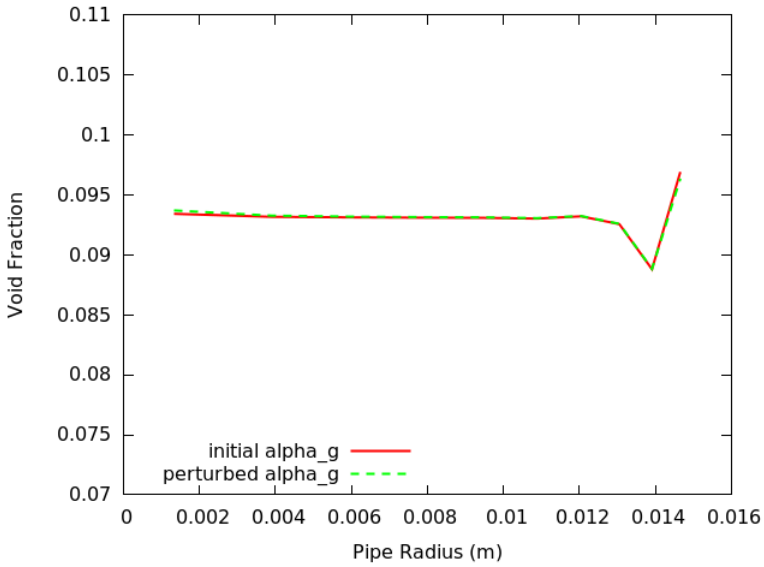


Figure 4.14: Initial and Perturbed void profiles for a 75% increase in the wall lubrication coefficient for axial location $L = 2cm$

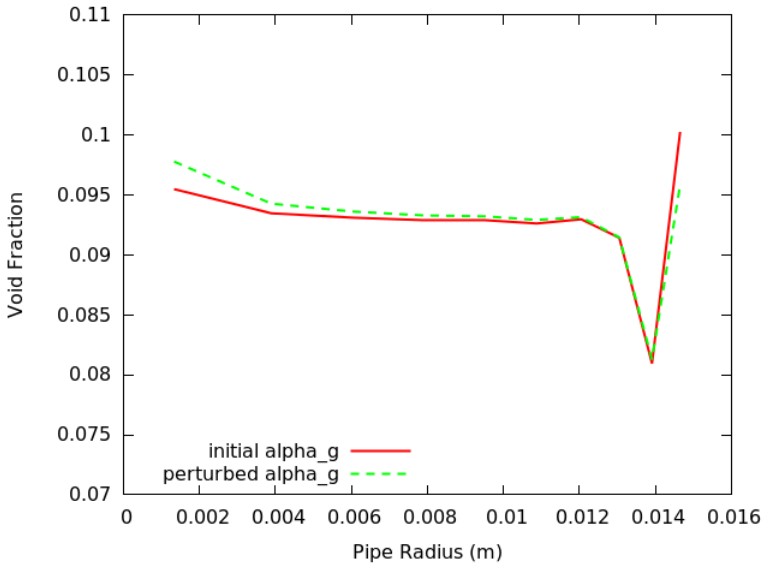


Figure 4.15: Initial and Perturbed void profiles for a 75% increase in the wall lubrication coefficient for axial location $L = 7cm$

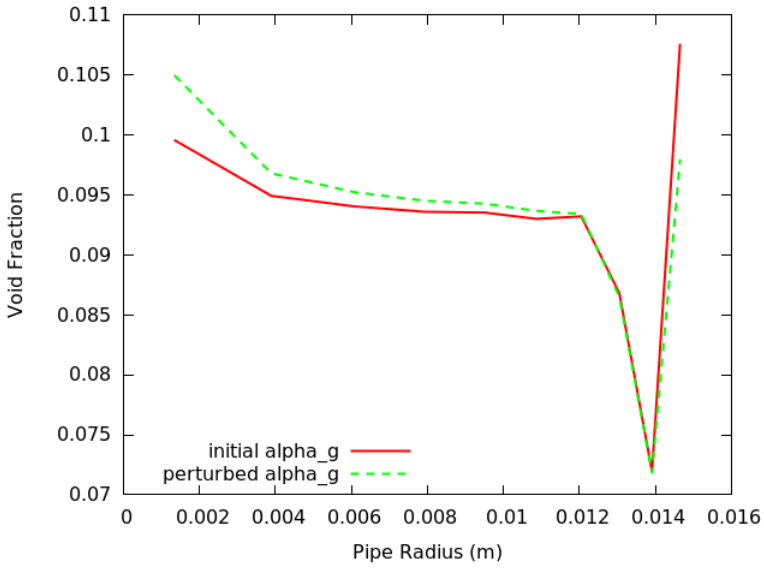


Figure 4.16: Initial and Perturbed void profiles for a 75% increase in the wall lubrication coefficient for axial location $L = 15cm$

The void profiles shown in Figures 4.14 through 4.16 are inverted compared to drag perturbed profiles in Figures 4.2 through 4.4 and lift perturbed profiles in Figures 4.8 through 4.10. This is likely due to the constant wall lubrication coefficient as opposed to the Frank model

used in the drag and lift perturbation study. The profiles shown in Figures 4.14 through 4.16 are likely not physical, but it still presents an opportunity to test adjoint accuracy and obtain an initial understanding of wall lubrication sensitivities.

The difference in perturbed and initial void profiles in Figure 4.14 is hardly noticeable, but Figures 4.15 and 4.16 show the difference increases dramatically towards the exit of the pipe. The largest regions of difference occur in the bulk flow and against the wall of the pipe, while the near-wall void trough shows almost no perturbation.

Figure 4.17 plots the exact $\Delta\vec{\alpha}_g$ perturbation alongside the approximated $\Delta\tilde{\alpha}_g$ perturbation. The red line denotes the exact perturbation and the green dashed line denotes the approximation calculated using the Jacobian matrix. The cell indexing scheme is the same used in section 4.3.1.

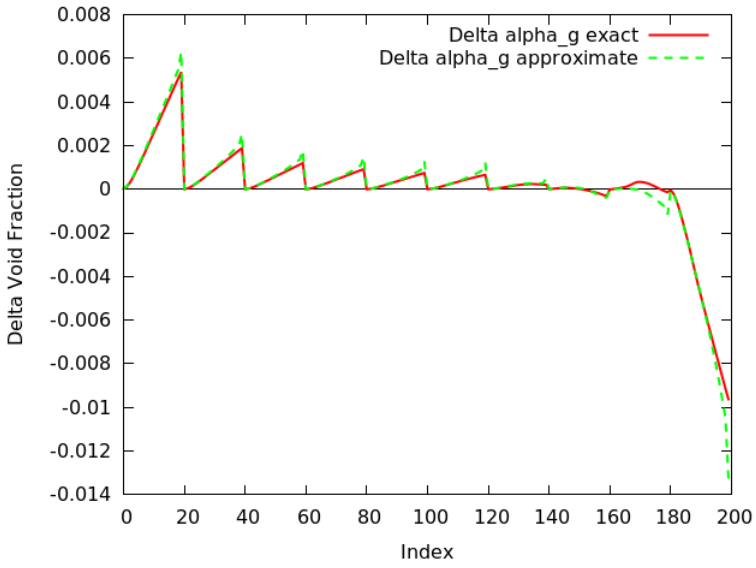


Figure 4.17: Exact perturbations $\Delta\vec{\alpha}_g$ and Jacobian approximated $\Delta\tilde{\alpha}_g$ for pipe mesh cell indices

Figure 4.17 shows that the largest perturbations occur at the pipe exit (indices 20, 40, 60, etc), and there is almost no change in the near-wall region of the pipe (indices 120-180). This is consistent with the void profiles shown in Figures 4.14 through 4.16. The shape, magnitude, and sign of the Jacobian approximated perturbed solution follow the exact solution closely. There is no reflected response across the $x = 0$ axis as there was in the drag case, and there is

no factor of 9 difference as there was in the lift case.

Figure 4.18 shows the log-log plot of ΔC_W perturbation size verses the L_2 normal of the difference between the exact $\Delta\vec{\alpha}_g$ perturbation and the approximate $\Delta\tilde{\alpha}_g$ perturbation. There is again a strong log-linear relationship for larger ΔC_W perturbations, and the slope is also close to 1.

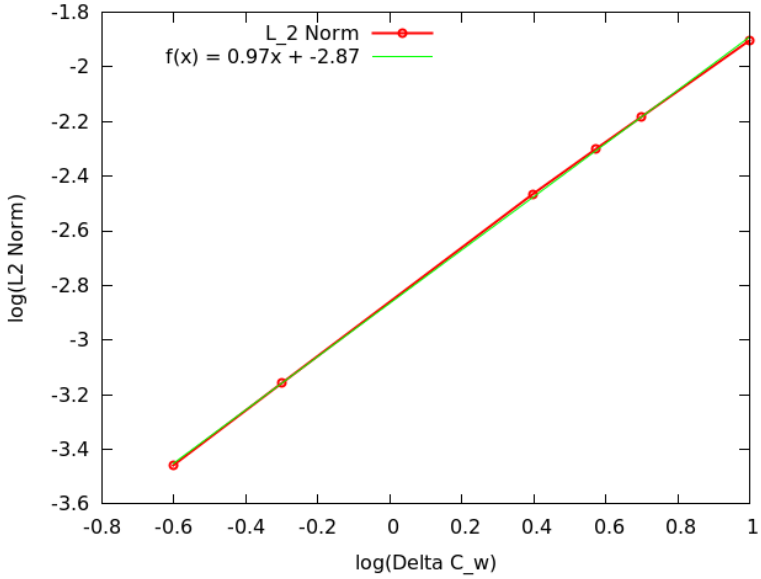


Figure 4.18: Log-log plot of $\log(\Delta C_W)$ vs. $\log(||\Delta\vec{\alpha}_g - \Delta\tilde{\alpha}_g||_2)$ and linear regression

Figures 4.17 and 4.18 show that the Jacobian matrix approximates the $\vec{\alpha}_g$ solution well with respect to perturbed wall lubrication coefficients. Figure 4.19 shows the exact and adjoint calculated responses for five regions of interest within the pipe. The wall and near-wall regions for $\vec{Q}_1^*$ and $\vec{Q}_2^*$ are the same as those used in section 4.3.1. Again, the graphic on the left hand side shows a primitive diagram of $\vec{Q}^*$ spatial locations in the pipe that were used to generate the response. The plot in the middle shows the exact and adjoint $\Delta\vec{\alpha}_g$ used to generate the response along with the $x = 0$ axis for reference. These reference plots are consistently scaled to show identical ranges of perturbations.

Consistent with previous results, all of the adjoint approximated responses in Figure 4.19 are the same sign and magnitude as the exact responses. The largest relative error between the

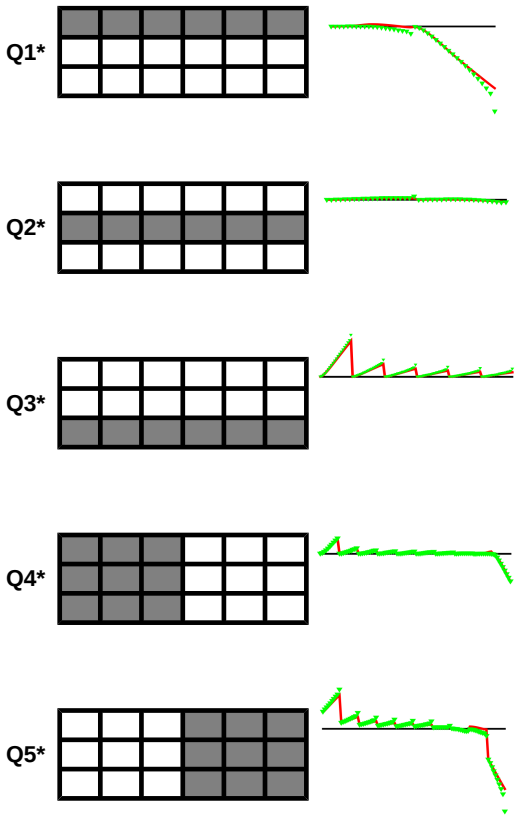


Figure 4.19: Exact and adjoint responses with respect to wall lubrication coefficient perturbations for various $\vec{Q}^*$ regions of interest

exact and adjoint approximated response is 23% for the $\vec{Q}_5^*$ or pipe exit region of interest, and the smallest relative error is 6% for the $\vec{Q}_4^*$ or pipe entrance region. This is the only interphase coefficient study that returns accurate responses for the pipe entrance and exit regions of interest. Unlike in the drag and lift response analysis, all $\vec{Q}^*$ regions of interest return responses that can indicate useful information about the exact perturbed $\vec{\alpha}_g$ solution.

It is unknown why this particular perturbed parameter returns the most accurate responses. The $|\vec{u}_r - (\vec{u}_r \cdot \vec{n}_w)\,\vec{n}_w|^2$ wall lubrication term is nonlinear with respect to relative velocity similar to the nonlinear $|\vec{u}_r|\,\vec{u}_r$ drag term, yet the wall lubrication analysis performs better than

the drag analysis. A possibility could be the semi-implicit treatment of the drag force. The drag vapor velocity is transfered to the diagonal of the discretized vapor momentum matrix and vice versa for the liquid velocity, while both the lift and the wall lubrication forces are calculated explicitly and moved to the source term of the discretized momentum equations. This will have an impact on the calculation of the mathematical adjoint, but the precise nature of this impact is unclear. Nonetheless, this section demonstrates that adjoint approximated responses can be a feasible means to approximate the perturbed $\vec{\alpha_g}$ void profiles with respect to perturbations in interphase momentum transfer coefficients.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The multiphase CFD solver `boilEulerFoam` implemented in OpenFOAM was found to be a suitable platform for the development of a multiphase adjoint capability. It was decided to remove the subcooled nucleate boiling model and consider an adiabatic case for adjoint development due to simplicity. Also, `boilEulerFoam` employs interphase momentum transfer closure relationships that are able to closely approximate experimental values for adiabatic cases. These closure relationships employ coefficients that are of particular interest with regards to sensitivity analysis.

Another attractive feature of `boilEulerFoam` and the OpenFOAM environment is that it is written in C++ and makes full use of a logical object hierarchy. Development in this hierarchy is desirable and OpenFOAM is designed in such a way as to facilitate implementation of new capability. Having full access to the source code was also necessary in order to build an adjoint solver within `boilEulerFoam`. In order to build an adjoint problem, the discrete equations that solve for void fraction, velocity, pressure, and turbulence in `boilEulerFoam` must be linearized. The linearization of these discrete equations requires a Jacobian matrix. Automatic differentiation in the forward sense was used to construct a discrete Jacobian matrix containing derivatives of all forward matrix coefficients.

Automatic differentiation was developed using the stand-alone object FadOne created by Jasak and was implemented within the OpenFOAM finite volume framework using existing solver methodology. A primitive count of FadOne instances found over 6,000 lines of altered code within OpenFOAM. Finite volume solvers that return solutions to scalar and vector fields can, using FadOne, also return `fadScalar` and `fadVector` fields that contain both values and

derivatives of values. This is done automatically using templates and operator overloading. Automatic differentiation implemented in OpenFOAM was found to build a Jacobian for a Laplacian operator that returned the exact solution for a Laplacian equation perturbed by introducing a source term. For non-coupled systems of equations, this Jacobian matrix is the same as the discrete finite volume matrix, and the Laplacian example is trivial, but this Laplacian test case provided verification that automatic differentiation was working correctly.

The Jacobian matrix for a one dimensional multiphase RANS problem was also examined and a perturbation test case was used to further verify the automatic differentiation capability. This Jacobian was built using the coupled system of equations, and it contained appropriate off-diagonal coefficients that captured the dependency of all variables. A perturbation of approximately 20% of the field of interest, e.g. pressure, was introduced in the four central cells of the uniform 10 cell mesh to create a perturbed system of linear equations. The Jacobian was found to appropriately approximate the perturbation with 4×10^{-4}% error, with respect to pressure, the smallest relative error, and 1.6% error with respect to dispersed phase velocity, the largest relative error. When using a vanLeer total variation diminishing scheme for void fraction convection, the perturbed approximation for a 20% perturbation in dispersed phase void fraction undershoots $\vec{\alpha}_g$ by 11%. The upwinded convection approximation, however, returns a relative error of 0.49%. Thus, the constructed Jacobian was found to give a satisfactory approximation of perturbed values for the one dimensional multiphase test problem.

An adjoint problem was derived for the nonlinear system of multiphase RANS equations with application to sensitivity analysis. This derivation returns the response of a given variable to perturbations in scalar operators for a particular field of interest. Once an adjoint solution is found, it can be used many times for various perturbations as long as the response of interest remains the same. This sensitivity analysis was performed on axi-symmetric pipe geometry using a coarse 20×10 mesh due to memory constraints imposed by automatic differentiation and Jacobian construction.

The specific sensitivity evaluation examined the responses of void fraction to perturbations in drag, lift, and wall lubrication coefficients. The drag case perturbed the drag coefficient by 10% and calculated a perturbed void fraction. The largest perturbations in $\vec{\alpha}_g$ were observed along the wall coinciding with the largest gradients of void fraction. For these regions, the Jacobian approximated the perturbations with a reasonable level of accuracy. The exact response for the wall region was 0.149, and the adjoint response was 0.094 with 37% relative error for a perturbation of 4% of the original void fraction. The exact response for the near-wall region was -0.087, and the adjoint response was -0.086 with a 1% relative error for a perturbation

of 2% of the original void fraction. The approximated perturbation in the bulk region, when compared to the exact perturbation, returned reflected values across the $x = 0$ axis. Therefore, the exact response for the bulk flow region was 0.021, and the adjoint response was -0.042 for a perturbation of 0.2% of the original void fraction.

The lift case perturbed the lift coefficient by a factor of 5 or by 500% and calculated a perturbed void fraction. The largest perturbations in $\vec{\alpha}_g$ were observed in the near-wall void peak region. These perturbations were an order of magnitude smaller than those calculated in the drag coefficient analysis. The Jacobian approximated perturbation was around a factor of 9 larger than the exact perturbed solution, although there is no reflection across the $x = 0$ axis as there was in the drag coefficient case. All of the approximated responses with exception to the region corresponding to the bulk flow of the pipe were an order of magnitude larger than the exact response, although the sign of the summation was correct. The exact response for the bulk flow region was 0.0034, and the adjoint response was 0.0027 with a 21% relative error for a perturbation of 0.04% of the original void fraction. The exact response for the near wall region was 0.035, and the adjoint response was 0.34 with a factor of 9.7 difference for a perturbation of 0.8% of the original void fraction.

The wall lubrication case perturbed wall lubrication coefficients by 75% and calculated a perturbed void fraction. The largest perturbations in $\vec{\alpha}_g$ were observed towards the exit in both the bulk flow region and against the wall of the pipe. The Jacobian approximated perturbation closely followed the shape, magnitude, and sign of the exact perturbed solution. The exact response for the pipe exit region was 0.0095, and the adjoint response was 0.0073 with a 23% relative error for a perturbation of 0.1% of the original void fraction. This was the largest relative error of these adjoint responses. The exact response for the pipe entrance was 0.0036, and the adjoint response was 0.0038 with a relative error of 6% for a perturbation of 0.03% of the original void fraction. This was the smallest relative error of these adjoint responses. The wall lubrication coefficient case was the only case to return reasonable response approximations for the pipe entrance and exit regions.

The computation time for the adjoint solution for five separate regions of interest was substantially less than the computation time for the initial and perturbed solution - 0.2s verses 13.7s. This is an attractive advantage for future sensitivity analysis calculations. The computational time for Jacobian construction was the largest at 93.2s. This is due to the fact that automatic differentiation performs operations on full matrices. However, adjoint sensitivity analysis only requires that the Jacobian be constructed once for successive perturbation values, and it is still possible to take advantage of the efficiency of adjoint calculations for a more intensive

sensitivity analysis.

This multiphase adjoint analysis shows that automatic differentiation can be used to construct Jacobian matrices and adjoint problems that return reasonable void fraction responses with respect to perturbations in some but not all of the interphase momentum coefficients. While some of the numerical behavior still requires investigation, the relative accuracy and computational advantage of these adjoint approximations shows their potential viability as a tool for robust sensitivity analyses.

5.2 Future Work

The future work of the adjoint capability presented in this thesis could be taken in many different directions. Because adjoint problem construction calculates derivatives automatically, there is the possibility of adding a wide array of capability to the existing adiabatic multiphase equations examined in this thesis. Any number of interphase momentum transfer models or turbulence models could be included without significant change to the source code. The addition of a boiling model would be a significant improvement of the existing adiabatic capability. This, however, would require significant verification of the Jacobian matrix due to the highly nonlinear nature of the subcooled nucleate boiling simulation. Particularly problematic for methodologies that utilize a Jacobian is when the closure relationship is flow regime dependent, implying if a parameter perturbation triggers a change in a closure relationship, the generated Jacobian is not appropriate.

It was found that the Jacobian problem approximates velocities only when drag terms have been removed from the pressure equation. The exact reason for this numerical phenomenon is not understood, although it is hypothesized to be related to the predictor-corrector algorithm used in the calculation of velocity. Future development can test alternate Jacobian matrix construction that accounts for both predicted and corrected values and improve the linear approximation. Future work could also examine the numerical behavior of approximated perturbations and understand why the perturbed case solution errors presented in this thesis behave as they do. This includes increasing the understanding of sign, magnitude, and order of the solution errors for not only dispersed phase void fraction but also other fields and closure relationships.

Future work could construct Jacobian matrices using OpenFOAM's sparse LDU matrix methodology instead of storing and operating on full matrices. Such a technique will require the implicit understanding of the off-diagonal structure of finite volume operations and their

dependence on other field variables. This is a significant undertaking since the algorithm to perform such an operation is not obvious. Furthermore, it may significantly differ from the automatic differentiation methodology because Jacobians will be constructed intuitively rather than automatically.

Even if Jacobian matrices are full, the successful calculation of adjoint solutions could still facilitate a more in depth sensitivity study of interphase momentum transfer terms. Due to the computational efficiency of an adjoint approach, it is possible to perform data assimilation using Bayesian statistics that can return frequency distributions of interphase momentum transfer terms as derived from experimental data. Similar techniques use adjoint responses to generate distributions of macroscopic cross sections for use in neutronics calculations [55]. Such an analysis could be done at a fraction of the computational cost of full forward solutions.

REFERENCES

[1] E Krepper, B Koncar, Y Egorov: "CFD Modeling of Subcooled Boiling - Concept, Validation and Application to Fuel Assembly Design", Nuclear Engineering and Design, Vol. 237, pages 716-731, (2007).

[2] D Bestion: "Applicability of Two-Phase CFD to Nuclear Reactor Thermal Hydraulics and Elaboration of Best Practice Guidelines" Nuclear Engineering and Design, Vol. 253, pages 311-321, (2012).

[3] HS Abdel-Khalik. "Adjoint-based sensitivity analysis for multi-component models", Nuclear Engineering and Design, Vol. 245, pages 49-54, (2012).

[4] TM Wildey, EC Cyr, JN Shadid, R Pawlowski, T Smith. "A comparison of Adjoint and Data-Centric Verification Techniques", Sandia Report Unlimited Release, Sandia National Laboratories, March (2013).

[5] R Roth, S Ulbrich. "A Discrete Adjoint Approach for the Optimization of Unsteady Turbulent Flows", Flow Turbulence and Combustion, Vol. 90, pages 763-783, (2013).

[6] D Cacuci: "Second-order ADJOINT Sensitivity analysis procedure (SO-ASAP) for computing exactly and efficiently first- and second-order sensitivities in large-scale linear systems: II. Illustrative application to a paradigm particle diffusion problem", Journal of Computational Physics, (2014).

[7] OpenFOAM - The Open Source CFD Toolbox, User Guide. Version 2.2.1. December (2012).

[8] Mesina, G. L.,"Reformulation RELAP5-3D in FORTRAN 95 and Results." *Proceedings of the ASME 2010 Joint US-European Fluids Engineering Summer Meeting and 8th International Conference on Nanochannels Microchannels, and Minichannels* , FEDSM2010-ICNMM2010, Montreal, Quebec, Canada, Aug 1-5 (2010).

[9] Martin, R. P. "TRAC-B Thermal-Hydraulic Analysis of the Black Fox Boiling Water Reactor." *National Technical Information Service, 77H-Reactor Engineering and Nuclear Power Plants*. Issue 9318, (1993).

[10] Areva NP Inc, *COBRA-FLX: A Core Thermal-Hydraulics Analysis Code.* ANP-10311NP. pbadupws.nrc.gov, (2010).

[11] Sung, Y., R. L. Oelrich Jr., C. C. Lee, N. Ruiz-Esquide, M. Gambetta, and C. M. Mazufri, "Benchmark of Subchannel Code VIPRE-W with PSBT Void and Temperature Test Data." *Science and Technology of Nuclear Instalations*, Vol. 2012, (2012).

[12] Cardoni, J. N., and Rizwan-uddin. "Nuclear Reactor Multi-Physics Simulations with Coupled MCNP5 and Star-CCM+." *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, (2011).

[13] Christon, M. A., J. Bakosi, M. M. Francois, R. B. Lowrie, R. Nourgaliev. "Multiphase Flow Analysis in Hydra-TH." *Conference: CASL Virtual Roundtable*, (2012).

[14] HG Weller, "Derivation, Modelling and Solution of the Conditionally Averaged Two-Phase Flow Equations", OpenCFD, `<http://www.opencfd.co.uk>`. February (2005).

[15] Sussman, M., A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell and M. L. Welcome., "An Adaptive Level Set Approach for Incompressible Two-Phase Flows.", *Journal of Computational Physics*, Vol. 148, Issue 1, pages 81-124, (1999).

[16] Popinet, S. and Zaleski S. "A Front-Tracking Algorithm for the Accurate Representation of Surface Tension." *International Journal of Numerical Methods in Fluids*, Vol. 30 No. 775, (1999).

[17] Jackson, C. J., D. G. Cacuci, and H. B. Finnemann. "Dimensionally Adaptive Neutron Kinetics for Multdimensional Reactor Safety Transients-I: New Features of RELAP5/PANBOX." *Nuclear Science and Engineering* Vol. 131 No. 2, pages 143-163, (1999).

[18] Jackson, C. J., D. G. Cacuci, and H. B. Finnemann. "Dimensionally Adaptive Neutron Kinetics for Multdimensional Reactor Safety Transients-II: Dimensionally Adaptive Switching Algorithms." *Nuclear Science and Engineering* Vol. 131 No. 2, pages 164-186, (1999).

[19] Williams, Mark L. "Perturbation Theory for Nuclear Reactor Analysis." *CRC Handbook of Nuclear Reactors Calculations* Vol. 3, pages 63-188, (1987).

[20] Pironneau, O. "On Optimum Design in Fluid Mechanics." *Journal of Fluid Mechanics* Vol. 64 part 1, pages 97-110, (1974).

[21] Pupko, V. Y. "Use of Adjoint Functions in Investigations of Heat Conduction and Transfer Processes." *Inzhenerno-Fizicheskii Zhurnal* Vol. 11 No. 2, pages 242-249, (1966).

[22] Dam, H. van, and J. E. Hoogenboom. "The Adjoint Space in Heat Transport Theory." *International Journal of Heat and Mas Transfer* Vol. 23, pages 349-353, (1980).

[23] Huang, C. H., and M. N. Ozisik. "Inverse Problem of Determining Unknown Wall Heat Flux in Laminar Flow Through a Parallel Plate Duct." *Numerical Heat Transfer* Part A Vol. 21, pages 55-70, (1992).

[24] Huang, C., S. Wang. "A Three-Dimensional Inverse Heat Conduction Problem in Estimating Surface Heat Flux by Conjugate Gradient Method." *International Journal of Heat and Mass Transfer*, Vol. 42, pages 3387-3403, (1999).

[25] *An Optimal Control Approach to A Posteriori Error Estimation in Finite Element Methods.* Institut fur Angewandte Mathematik, Universitat Heidelburg, (2001).

[26] C Othmer: "A Continuous Adjoint Formulation for the Computation of Topological and Surface Sensitivities of Ducted Flows", International Journal for Numerical Methods in Fluids, Vol. 58, pages 861-877, (2008).

[27] K Mani, DJ Mavriplis. "Unsteady Discrete Adjoint Formulation for Two-Dimensional Flow Problems with Deforming Meshes", American Institute of Aeronautics and Astronautics Journal, Vol. 46, Number 6, June (2008).

[28] JD Jansen, OH Bosgra, PMJ Van den Hof. "Model-base control of multiphase flow in subsurface oil reservoirs", Journal of Process Control, Vol. 18, pages 846-855, (2008).

[29] JD Jansen. "Adjoint-Base Optimization of Multi-Phase Flow Through Porous Media - A Review", Computers & fluids, Vol. 46, pages 40-51, (2011).

[30] T Wildey, E Cyr, R Pawlowski, J Shadid, T Smith: "Adjoint Based a Posteriori Error Estimation in Drekar:CFD ", Sandia National Laboratories Unlimited Release, October (2012).

[31] T Wildey, E Cyr, J Shadid, R Pawlowski, T Smith. "A Comparison of Adjoint and Data-Centric Verification Techniques", Sandia National Laboratories Unlimited Release, March (2013).

[32] T Smith, J Shadid, R Pawlowski, E Cyr, and T Wildey: "Thermal Hydraulic Simulations, Error Estimation, an Parameter Sensitivity Studies in Drekar:CFD", Sandia National Laboratories Unlimited Release, September (2013).

[33] L B Rall, G F Corliss. "An Introduction to Automatic Differentiation."

[34] A Grienwank: "On Automatic Differentiation", Argonne National Laboratory, November (1988).

[35] CA Mader, JRRA Martins. "ADjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers", American Institute of Aeronautics and Astronautics, Vol. 46, Number 4, April (2008).

[36] The OpenFOAM Extend Project `www.extend-project.de`

[37] M Ishii and N Zuber. "Drag Coefficient and Relative Velocity in Bubbly, Droplet or Particulate Flows". AIChE J., pages 25-843, (1979).

[38] H, Enwald, E Peirano, A-E Almstedt "Eulerian Two-Phase Flow Theory Applied to Fluidization". Int. J. Multiphase Flow, Vol. 22, Suppl, pages 21-66, (1996)

[39] L. Schiller and Z. Naumann. “A Drag Coefficient Correlation”. Z. Ver. Deutsch. Ing., pages 77-318, (1935).

[40] Syamlal, M., Rogers, W. and O’Brien, T. J. (1993) MFIX documentation, Theory Guide. Technical Note DOE/METC-94/1004. Morgantown, West Virginia, USA.

[41] H Rusche: ”Computational Fluid Dynamics of Dispersed Two-Phase Flows at High Phase Fractions”, PhD Thesis for Imperial College of Science, Technology, and Medicine, London, England,(2002).

[42] D. Lucas and A. Tomiyama. “On the Role of the Lateral Lift Force in Poly-Dispersed Bubbly Flows.” International Journal of Multiphase Flow, (2011).

[43] The Favre Averaged Drag Model for Turbulent Dispersion in Eulerian Multi-Phase Flows, (2004).

[44] A. D. Gosman, C. Lekakou, S. Politis, R. I. Issa, and M. K. Looney. “Multidimensional Modeling of Turbulent Two-Phase Flow in Stirred Vessels”. AIChE J., pages 38-1946, (1992).

[45] T Frank. “Advances in Computational Fluid Dynamics (CFD) of 3-Dimensional Gas-Liquid Multiphase Flows.” In NAFEMS Seminar Simulation of Complex Flows (CFD), page 1, Wiesbaden, Germany, (2005).

[46] R. Rzehak, E. Krepper, and C. Lifante. “Comparative Study of Wall-Force Models for the Simulation of Bubbly Flows”. Nuclear Engineering and Design, (2012).

[47] B van Leer. “Towards the Ultimate Conservative Difference Scheme. IV A New Approach to Numerical Convection”, Journal of Computational Physics, Vol. 23, pages 276-299, (1977).

[48] A Alali, Development and Validation of a New Solver Based on the Interfacial Area Transport Equation for the Numerical Simulation of Sub-cooled Boiling with OpenFOAM CFD Code for Nuclear Safety Applications. PhD Thesis, pages 33-52.

[49] N Todreas, M Kazimi: *Nuclear Systems*, 2nd Edition, Vol. 1, pages 851-859. CRC Press, (2012).

[50] Versteeg, H. K., and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Harlow, England: Pearson-Prentice Hall, 2007.

[51] Issa, R. I. "Solution of the Implicitly Discretised Fluid Flow Equations by Operator-Splitting." *Journal of Computational Physics* Vol 62, pages 40-65, (1985).

[52] D Drew, R T Lahey, "Application of General Constitutive Principles to the Derivation of Multidimensional Two-Phase Flow Equations", International Journal of Multiphase Flow, Vol. 5, pages 243-264, (1979).

[53] HK Versteeg, W Malakasekera: *An Introduction to Computational Fluid Dynamics*, 2nd Edition, Prentice Hall, Essex, England (2007).

[54] H Jasak. 6th OpenFOAM Workshop, Penn State University, June 2011.

[55] A Hernandez-Solis. "Uncertainty and Sensitivity analysis Applied to LWT Neutronic and Thermal-Hydraulic Calculations". Gteborg : Chalmers University of Technology, (2012).

[56] E Mictha: "Modeling of Subcooled Nucleate Boiling with OpenFOAM", Masters of Science Thesis for the Royal Institute of Technology, Stockholm, Sweden, February (2011).

[57] A Ghione: "Development and Validation of a Two-Phase CFD Model Using OpenFOAM", Masters of Science Thesis for the Royal Institute of Technology, Stockholm, Sweden, December (2012).

[58] R Stengel: *Stochastic Optimal Control, Theory and Application*, John Wiley & Sons, Inc, Princeton, NJ (1986).

[59] H Schlichting: *Boundary Layer Theory*, 7th Edition, McGraw Hill, Inc. (1987).

[60] F Incropera, D Dewitt, T Bergman, A Lavine: *Fundamentals of Heat and Mass Transfer*, 6th Edition, John Wiley & Sons, Hoboken, NJ (2007).

[61] CM Rhie, W L Chow: "Numerical Study of the Turbulent Flow Past an Airfoil with Trailing Edge Separation", AIAA Journal, Vol. 21, No. 11, November (1983).

[62] R Kunz, B Siebert, W K Cope, N Foster, S Antal, S Ettorre: "A Coupled Phasic Exchange Algorithm for Three-Dimensional Multi-Field Analysis of Heated Flows with Mass Transfer", Computers and Fluids, Vol. 27, Number 7, pages 741-768, (1998).

[63] SM Damian. "High Resolution Schemes Implementation in OpenFOAM", Simulation Internal Report.
<https://openfoamwiki.net/index.php/OpenFOAM_guide/NVD_TVD_formulation>.
March (2007).

[64] N Dinh, R Nourgaliev. "An Intial VU-Assessed Code Development Effort", Idaho National Laboratory, December (2010).

[65] ML Williams. *CFC Handbook of Nuclear Reactors Calculations.* "III. Adjoint Operators - The Differential and Variational Methods", Vol. III, pages 72-84, (1988).

[66] E Furbo. "Evaluation of RANS Turbulence Models for Flow Problems with Significant Impact of Boundary Layers", Masters of Science Thesis, Uppsalal Universitet, December (2010).

[67] MS Darwish, "A New High-Resolution Scheme Based on the Normalized Variable Formulation", Numerical Heat Transfer, Part B: Fundamentals: An International Journal of Computation and Methodology, Vol. 24, pages 353-371, (1993).

[68] SK Nadarajah, A Jameson, "A Comparison of the Continuous and Discrete Adjoint Approach to Automatic Aerodynamic Optimization", American Institute of Aeronautics and Astronautics, (1999).

[69] N Safiran, U Naumann. "Toward Adjoint OpenFOAM", Department of Computer Science of RWTH Aachen University, Germany, July (2011).

[70] B Yu, WQ Tao, JJ Wei, Y Kawaguchi, T Tagawa, H Ozoe. "Discussion on Momentum Interpolation Method for Collocated Grids of Incompressible Flow", Numerical Heat Tranfer, Part B, Vol. 42, pages 141-166, (2002).

[71] PM Morse, H Feshback. *Methods of Theoretical Physics*. New York: McGraw-Hill, (1953).

[72] Petersdorff, T. Von, and R. Leis. "Boundary Integral Equations for Mixed Dirichlet, Neumann and Transmission Problems." *Mathematical Methods in the Applied Sciences* Vol. 11, pages 185-213, (1989).

[73] Carey, V., D. Estep, A. Johansson, M. Larson, and S. Tavener. "Blockwise Adaptivity for Time Dependent Problems Based on Coarse Scale Adjoint Solutions." *SIAM Journal on Scientific Computing* Vol. 342 No. 4, (2010).

[74] Estep, D., S. Tavener, and T. Wildey. "A Posteriori Analysis and Improved Accuracy for an Operator Decomposition Solution of a Conjugate Heat Transfer Problem." *SIAM Journal on Numerical Analysis* Vol. 46 No. 4, pages 2068-2089, (2008).

[75] Estep, D., V. Ginting, D. Ropp, J. N. Shadid, and S. Tavener. "An A Posteriori - A Priori Analysis of Multiscale Operator Splitting." *Siam Journal on Numerical Analysis* Vol 46 No. 3, pages 1116-1146, (2008).

[76] Estep, D. M. Pernice, D. Pham, S. Tavener, and H. Wang. "A Posteriori Error Analysis of a Cell-Centered Finite Vol. Method for Semilinear Elliptic Problems." *Journal of Computational and Applied Mathematics* Vol. 233, pages 459-472, (2009).

[77] Estep, D. M. Pernice, D. Pham, S. Tavener, and H. Wang. "A Posteriori Error Analysis for a Cut Cell Finite Vol. Method." *Computational Methods in Applied Mechanics and Engineering* Vol. 200, pages 2768-2781, (2011).

[78] Yen, D. H. Y., J. V. Beck. "Green's Functions for Non-Self-Adjoint Problems in Heat Conduction with Steady Motion." *Journal of Engineering Mathematics* Vol. 57, pages 115-132, (2007).

[79] Jarny, Y., M. N. Ozisik, and J. P. Bardon. "A General Optimization Method Using Adjoint Equation for Solving Multidimensional Inverse Heat Conduction." *International Journal of Heat and Mass Transfer* Vol. 34 No. 11, 2911-2919 (1991).