Design Optimization and Combustion Simulation of Two Gaseous and Liquid-Fired Combustors

by

Sina Hajitaheri

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical Engineering

Waterloo, Ontario, Canada, 2012

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The growing effect of combustion pollutant emission on the environment and increasing petroleum prices are driving development of design methodologies for clean and efficient industrial combustion technologies. The design optimization methodology employs numerical algorithms to find the optimal solution of a design problem by converting it into a multivariate minimization problem. This is done by defining a vector of design parameters that specifies the design configuration, and an objective function that quantifies the performance of the design, usually so the optimal design outcome minimizes the objective function. A numerical algorithm is then employed to find the design parameters that minimize the objective function; these parameters thus specify the optimal design. However this technique is used in several other fields of research, its application to industrial combustion is fairly new.

In the present study, a statistical optimization method called response surface methodology is connected to a CFD solver to find the highest combustion efficiency by changing the inlet air swirl number and burner quarl angle in a furnace. OpenFOAM is used to model the steady-state combustion of natural gas in the 300 KW BERL combustor. The main barrier to applying optimization in the design of industrial combustion equipment is the substantial computational effort needed to carry out the CFD simulation every time the objective function needs to be evaluated. This is intensified by the stiffness of the coupled governing partial differential equations, which can cause instability and divergent simulations. The present study addresses both of these issues by initializing the flow field for each objective function evaluation with the numerical results of the previously converged point. This modification dramatically reduced computation time.

The combustion of diesel spray in the GenTex 50M process heater is investigated in the next part of this thesis. Experimental and numerical studies were carried out for both the cold spray and the diesel combustion where the numerical results satisfactorily predicted the observations. The simulation results show that, when carrying out a parametric design of a liquid fuel-fired combustor it is necessary to consider the effect of design parameters on the spray aerodynamic characteristics and size distribution, the air/spray interactions, and the size of the recirculation zones.

Acknowledgements

I would like to thank my supervisors, Prof. Kyle Daun, and Prof. John Wright for all of their help and support in my Masters research. Their guidance during the research meetings is appreciable as well as their collaboration and care in my studies. Besides, I thank my committee members, Prof. Fraser and Prof. Peterson, for reviewing my thesis and providing helpful suggestions.

I am also thankful to David Burr and Adam Horsman, my fellow graduate students. They were supportive and listening, while offering advice when I face difficulties in my research. They also provided much needed break in the office, such as Sporcle games and sword fights. Moreover, I am obliged to many of my friends for all the memorable moments we made together. I greatly value their friendship and deeply appreciate their belief in me.

I am truly grateful to my wonderful brother Saeed, and my treasurable sisters Sahar and Setareh for their kindness, understanding, and support throughout my Masters research. Their help and encouragement taught me many valuable life lessons. I wish them the loveliness of dawn, the splendour of stars, and the essence of happiness.

Above all, I owe sincere and earnest gratitude to my beloved parents. Despite the distance separating us, I always felt their presence during my journey. They have been my eternal heroes who provide me love, joy, and happiness in my life. I am honoured to dedicate this thesis to my elegant parents.

Table of Contents

List of Figures

List of Tables

We think too much and feel too little. More than machinery we need humanity. More than cleverness, we need kindness and gentleness. Without these qualities, life will be violent and all will be lost.

"The Great Dictator"
Charles Chaplin

Chapter 1
Introduction and Literature Review

1.1 Motivation and Objective of the Thesis

The development of industrial burners, boilers, and furnaces with higher performance, (particularly higher efficiency, and lower pollutant emissions) is typically the main goal in the design of combustion equipment. Improving the present day performance in industrial combustion devices will require significant advancement in the knowledge of the physics and chemistry underlying combustion. In addition, more organized and efficient tools, such as numerical simulation methods should be applied on the design process to enhance the performance of the new combustion equipment.

Compared to the conventional experimental techniques and analytical methods, numerical modeling has been mostly selected as a faster, cheaper and more convenient tool. Additionally, owing to considerable improvement in computational performance and calculation speed in this past recent decade, it is possible to perform a comprehensive computer simulation using numerical methods and mathematical models for many complex industrial problems. However, some discrepancies are expected due to the assumptions made to make combustion simulations computationally tractable. Moreover, the numerical schemes used in these problems may encounter numerical instability and divergence. Hence, in order to overcome simulation difficulties and to speed up the simulation, it is necessary to implement a methodology which could meticulously address all these issues.

This thesis aims to demonstrate how CFD technology can be used to simulate and optimize two different combustion chambers: a laboratory-scale methane-fired furnace; and a diesel process heater. To do so, a series of mathematical models have been used to represent the fluid flow, gas phase combustion, turbulent reacting flow, two-phase flow, and spray combustion in these two combustion systems.

In the first part of the thesis, a statistical optimization study called response surface modeling (RSM) [1] is coupled to OpenFOAM [2] to maximize fuel conversion efficiency of the BERL natural gas (methane)-fired furnace [3] by changing the inlet air swirl number and burner quarl angle. The main barrier to implementing optimization in the design of industrial combustion equipment is the calculation time needed to carry out the full CFD simulation for each objective function evaluation. This is exacerbated by the stiffness of the coupled partial differential equations (momentum, energy, radiation, and chemical kinetics), which can lead to instability and divergent simulations. Since the optimization algorithm must evaluate the objective function without user intervention, divergence is often avoided through excessive under-relaxation, which further adds to the computational effort needed to evaluate the final design objective and overall design time. The present study shows how these issues can be addressed by initializing the flow field for each objective function evaluation with the numerical results of the previously converged point.

The second part of the thesis presents a more complicated combustion problem where the diesel oil combustion flow in the GenTex process heater [4] is studied numerically and experimentally. Since many complex phenomena occur in a typical liquid fuel combustion problem, insight provided by numerical simulations is invaluable for understanding the burner functionality, and where the designer should focus when seeking to improve the design

performance. At the end, methods to optimize the capabilities of the process heater using a design optimization technique are proposed.

1.2 Outline of the Thesis

This thesis contains five chapters. Chapter 1 presents a review of the literature regarding the design optimization in combustion problems. The combustion models used in the current simulation for the gaseous and liquid fuel combustion chambers are described in the next chapter. This includes the governing equations, turbulence-chemistry interactions, CFD solvers, injection condition, turbulent spray, and the boundary conditions.

Chapter 3 explains the optimization algorithm used in this research. First, a description of the design optimization including basic gradient-based methods is given. Then, the major framework of the optimization algorithm used in the current research, the Response Surface Modeling (RSM) method, is explained in detail for multivariate problems and the related challenges are discussed further in the chapter.

The implementation of the combustion models and the optimization algorithm are presented in Chapter 4. The results obtained from the design optimization of the BERL burner and the numerical simulation and experimental study of the reacting flow in the GenTex diesel oil process heater are comprehensively studied in this chapter. Finally, the concluding results are summarized in Chapter 5, along with the recommendations for future work regarding the developments on the current optimization algorithm and the suggestions for implementing new optimizing methods.

1.3 Working Environment

This thesis work was carried out using the computational continuum mechanics source code called OpenFOAM [2]. OpenFOAM is an object-oriented C++ library which is used to build executable code, called "applications". The applications are divided into two groups; "solvers" that are each created to solve a particular problem in continuum mechanics mostly by using computational fluid dynamics, and "utilities" that are created to perform tasks that involve data operation. This source code is comprised of numerous solvers and utilities which can solve many types of problems as described in Ref. [5]. OpenFOAM's interfaces to the pre-processing and post-processing environments are also its utilities which ensure consistent data handling across all environments.

One of the advantages of this source code is that the solvers and utilities can be modified and updated by the users who understand the original method, the physics of the problem and the knowledge of computer programming techniques. Accordingly, the user is able to derive and define desired solvers and functions specific to the problem at hand. On the other hand, the main disadvantage is that the computer-programmed codes are not commented, and there is no comprehensive documentation for many of its functional solvers of OpenFOAM. In addition, adding a new computer code requires the user to do a lot of configuration work with little support.

The solvers used in the current research are extended by the user to consider the swirling inlet air, axisymmetrical geometries, fuel ignition, different chemical solvers, steady-state simulation, radiation heat transfer, etc. Then, the solvers are connected to an optimization algorithm with an interface module written in C++ as presented in Appendix B.

1.4 Literature Survey on the Optimization in Combustion Studies

This chapter presents the literature related to the current research. This review shows that there have been very few applications of design optimization to solve industrial combustion problems.

Many studies have conducted experiments to heuristically improve the design of combustion equipment, usually through univariate parametric studies [6-10]. While the majority of these studies claim that their efforts have found an "optimal" solution, since the majority of these procedures consist of a series of univariate parametric studies, which do not guarantee optimality (optimality conditions are explained in Ref. [11]). Accordingly, the obtained results could be only considered as an improved solution. Although these works are not oriented in the design optimization field, however, they generally give the optimization designer/programmer some insight into choosing an appropriate optimization technique.

Several other studies combined the experimental apparatus and numerical calculations in order to optimize objective(s) in a combustion problem [12-16]. In these studies, an on-site automatic analyzer which is programmed based on an optimization technique (mostly evolutionary algorithms, e.g. genetic algorithm and neural network), is used. Using this analyzer, some input variables of the combustion equipment, e.g. inlet conditions, are changed in order to enhance the performance of certain output data, e.g. pollutant contents. For certain input variables, the combustion device is run and the output data are collected. The on-site automatic analyzer then analyzes the output data and makes the necessary changes to the input variables with the aim of enhancing the output data. The combustion equipment is then rerun and the output data are measured. Again, the new input variables are specified by the analyzer based on

the collected output data. The procedure is repeated continuously until the best output data are measured from the combustion equipment. Although in these works, the controllers are fed by experimental data, several of them are also compatible with CFD simulations. For instance, Chu et al. [16] presents a constrained optimization algorithm using an artificial neural network with the given experimental data as an input, with the aim of minimizing NO_x and CO emissions while maximizing the thermal efficiency in a coal-fired boiler. The overall algorithm can be changed into an automatic fully numerical optimization by only replacing the experimental tests with CFD simulations. The other example of this approach is the work presented by Buche et al. [17] who studied the automatic optimization of the spatial distribution of fuel injection rates in a gas turbine burner. Buche et al. [17] employed an evolutionary optimization algorithm and an automated interface to modify the parameters in the experimental setup for the fuel injection as well as for the post-processing. The evolutionary algorithm worked for multiple objectives using a Pareto front. It also considered the effect of noise in the objective function. The design parameters comprised of eight analogue valves for controlling the fuel distribution, and the evaluation tool used was an experimental test-rig for a gas turbine burner. The emissions and the pulsation of the burner were taken as the two objectives for the evolutionary algorithm. Their results showed that the implemented algorithm was successfully converged to a Pareto front and their analysis of the resulting parameters clarified the relevant physical processes.

The costs coming from the implementation of experimental apparatus are usually very high, in addition to the expenses due to performing many on-site tests, let alone that sometimes the devices might not be available or difficult to access. Given the high costs of carrying out experiments and since the state-of-the-art in combustion modeling is advancing rapidly, both in terms of accuracy and computational efficiency, there is a pressing need to develop model-based

optimization techniques for industrial combustion. This technique works by coupling a numerical model of the system to a numerical minimization algorithm; the system state is defined by a vector of design parameters, **x**, and the design performance is quantified by an objective function, $F(\mathbf{x})$. This transforms the design problem into one of multivariate minimization, which can be solved numerically to find the vector $\mathbf{x}^*$ that minimizes $F(\mathbf{x})$. These design parameters thus specify optimal design outcome. Numerical optimization techniques are ubiquitous in other engineering disciplines, however studies in the field of CFD simulation has been very limited to date, where these studies are mostly associated with aerodynamic and heat transfer problems [18-22]. In particular, Thévenin and Janiga [23] assert that the application of optimization to industrial combustion is a "*fairly new field of research*".

One of the very early works in this field was done by Aizenbud and Band [24]. They defined a simple model for an internal combustion engine in order to achieve the highest efficiency by changing different engine parameters such as compression ratio. Their results cannot be considered reliable as they hadn't considered several important operating parameters in their calculations, however their effort was a relatively new venture into the optimization in the field of industrial combustion.

The first sophisticated study in the combustion-oriented model-based optimization area was presented by Smith et al. [25]. In this research, a local optimization technique called response surface methodology (RSM) is incorporated to a CFD solver with the objective of maximizing the highest cold gas efficiency of a premixed and a diffusion type injectors through manipulating injectors inlet conditions. Afterward, their studies were continued by investigating the 3-D constrained optimization of combustion efficiency of a simulated pulverized coal combustor. The results appear to be reasonable, although there was no verification by any

experimental data. In general, their work can be recognized as a relatively comprehensive optimization study in which this study investigates the optimization application scheme along with the CFD solver in some details, however there are still some uncertainties and lack of physical interpretation to their final results.

Correa and Smith [26] compared two different optimization schemes: a parallel direct search method and a quasi-Newton method, the aim of obtaining a specified velocity profile at the outlet of a square pipe using the inlet flow rate as the design parameter. They concluded that quasi-Newton method is a more efficient technique, so they used to carry out a more comprehensive optimization of coil outlet temperature in an ethylene furnace. Their results show a very high improvement in the objective function value. At the next step, the burner was split into four different zones in order to improve the outlet temperature; nevertheless this effort didn't result in any considerable improvement.

Thévenin et al. [27] applied the simplex optimization method to obtain a homogeneous temperature profile at a certain cross-section from the injector of a laminar burner. The only design variable in this single-objective problem was the fuel/air ratio at the primary and secondary inlet. Subsequently, Janiga and Thévenin [28] studied a problem with the same geometry and design variable, however this time they compared their results obtained from the simplex method those obtained with a genetic algorithm (GA). In addition, in the recent research, the main objective was set to be the reduction of CO emission. It was concluded that the genetic algorithm improves the objective function more than simplex method, however the GA was also more time-consuming.

Catalano et al. [29] suggested a comprehensive gradient-based optimization procedure for black-box simulation codes. They implemented their method to optimize a duct-burner in combined-cycle and cogenerative plants. First, they used a commercial CFD code to simulate a new enhanced-mixing duct-burner. They validated the code results versus the experimental data. Then, they applied a gradient-based optimization procedure called "progressive optimization", and incorporated it in the CFD codes. This method is very efficient, since the convergence of the flow solution and of the optimization process occurs simultaneously. At the next step, the proposed optimization method was used to solve two optimization problems involving a duct-burner. The goal of the first application was to minimize the outlet temperature gradient, while the second application aimed to reduce the near-wall temperatures and to shorten the flame. Moreover, they discussed some criteria that should be considered in design optimization schemes, although these criteria had been ignored in previous studies.

Motsamai et al. [30] presented a technique for design optimization of a liquid fuel combustor. Similar to the previous works mentioned earlier, they combined an automatic design optimization technique with a CFD solver, in this case the Fluent commercial software. The objective was to minimize the combustor exit temperature profile through changing the combustor parameters. They used the dynamic-Q optimization algorithm [31] since this method was specifically designed for constrained problems where the objective and constraint functions are expensive to evaluate. The optimization technique resulted in a more uniform combustor exit temperature profile compared to the original case.

1.5 Summary

Rising fuel costs and the increasing impact of combustion pollutants on the environment make development of design methodologies for efficient and clean industrial combustion technologies a priority. In this regard, many studies have used experiments and numerical simulations to heuristically improve the design of combustion devices. A more sophisticated approach is model-based optimization, but its application to industrial combustion equipment has been limited to date due to the computational expense of combustion simulations and the numerical stiffness of governing equations [23]. In the next few chapters this thesis will show how the performance of a methane-fired furnace can be improved through multivariable model-based optimization.

Chapter 2
Combustion Modeling

2.1 Introduction

This chapter presents the combustion models used in the current research. The methods used to model the combustion of methane in a laboratory-scale furnace, as well as the combustion inside a diesel-fired process heater are given in the present chapter. The working fuels in these combustion chambers are in two different phases which behave distinctively, and consequently the chapter is divided into two main sections; *a*) combustion modeling of gaseous fuels and *b*) combustion modeling of liquid fuels. In gaseous fuel combustion chambers, when the fuel is inserted into the chamber (normally through a burner), it mixes with an oxidant gas (usually air) and then the fuel receives the heat required to initiate combustion through an igniter. In liquid fuel combustion chambers, the fuel is introduced through an injector. The droplets of the fuel spray evaporate and then react with air. While the whole procedure in gaseous combustion systems deals with various complex concepts such as chemical kinetics, turbulence-chemistry interactions, and high temperature gradients, the combustion of liquid fuel sprays is significantly more complicated due to the many two-phase flow phenomena, e.g. atomization, evaporation, collision, break-up, and numerous other gas-liquid interactions, which requires the implementation of several precise and comprehensive models. In addition, since the governing equations should also be solved for the dispersed phase, these nonlinear equations are much

stiffer and the computational time is considerably higher compared to the combustion modeling of gaseous fuels which asks for more considerations on the simulation.

Each section of this chapter is followed by the governing equations and the necessary boundary and initial conditions along with the corresponding numerical algorithm. Also, properties, correlations, and submodels pertaining to the gas phase and liquid phase are discussed in the related subsections.

2.2 Combustion of Gaseous Fuels

In the past two decades the use of CFD codes for modeling the reacting flow in boilers, heaters, and combustion chambers has become a helpful tool to predict the performance of these combustion devices and has gained increased acceptance by the scientific and industrial communities. Modeling helps engineers to optimize the operating conditions, reduce pollutant emissions, investigate the negative points of the equipment, evaluate their measurements and improve the design of new combustion devices. Many of the combustion chambers involve natural gas, since it is readily available.

Since several complex phenomena such as mixing, radiative heat transfer, chemical kinetics, and turbulence occur during the combustion process, a rational approach with appropriate submodels should be used in order to obtain a reasonable prediction. In this section the modeling approach to simulate the combustion of methane in the BERL furnace is defined.

2.2.1 Governing Equations

In a homogenous Newtonian fluid flow, mathematical modeling is done by solving a set of equations governing the transport of mass, momentum, energy, and chemical species, along with the state equations of the fluidic system. In addition, since the flows in most industrial combustion applications are turbulent, the conservation equations should be written in time-averaged or spatial forms, which need to be closed by using additional turbulent models. The set of governing differential equations consist of conservation of mass,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \left(\rho \vec{\mathbf{v}} \right) = 0 \tag{2.1}$$

conservation of species

$$\frac{\partial \left(\rho Y_i \right)}{\partial t} + \nabla \cdot \left(\rho \vec{\mathbf{v}} Y_i \right) = \nabla \cdot \left(\mu_{eff} \nabla \cdot Y_i \right) + \kappa \dot{r}_i \tag{2.2}$$

conservation of momentum

$$\frac{\partial \left(\rho \vec{\mathbf{v}} \right)}{\partial t} + \nabla \cdot \left(\rho \vec{\mathbf{v}} \vec{\mathbf{v}} - \tau_{eff} \right) = -\nabla P + \rho \vec{\mathbf{g}} \tag{2.3}$$

and conservation of energy

$$\frac{\partial \left(\rho h \right)}{\partial t} + \nabla \cdot \left(\rho \vec{\mathbf{v}} h \right) = \nabla \cdot \left(\alpha_{eff} \nabla h \right) + \frac{\partial \rho}{\partial t} h_T - \nabla \cdot \mathbf{q}_r \tag{2.4}$$

where $\vec{\mathbf{v}}$, Y, h, P, and ρ are the velocity vector, mass fraction, enthalpy, pressure, and the flow density respectively, and $\nabla \cdot \mathbf{q}_r$ is the radiative source term provided by the P_1 model (see Section 2.2.3).

Turbulence is simulated using the RNG k-ε model. The tensor τ_{eff} is the summation of the viscous and the turbulent stresses. In the same way, the effective thermal diffusivity and

dynamic viscosity are approximated as the summation of viscous and turbulent components, α_{eff} = α + α_t and μ_{eff} = μ + μ_t, respectively. The turbulent viscosity, μ_t, is calculated by

$$\mu_t = \mu\left(1+\sqrt{\frac{C_\mu \rho k^2/\varepsilon}{\mu}}\right)^2 \tag{2.5}$$

The constant C_μ is equal to 0.0845. The solver assumes that the Lewis number for each of the species is equal to unity. The RNG k-ε model includes transport equations for k and ε

$$\frac{D(\rho k)}{Dt} = \nabla\cdot\left[\left(\mu+\frac{\mu_t}{\sigma_k}\nabla k\right)\right] + G_k - \rho\varepsilon \tag{2.6}$$

and

$$\frac{D(\rho \varepsilon)}{Dt} = \nabla\cdot\left[\left(\mu+\frac{\mu_t}{\sigma_k}\nabla \varepsilon\right)\right] + C_{1\varepsilon}\frac{\varepsilon}{k}G_k - C_{2\varepsilon}^{*}\rho\frac{\varepsilon^2}{k} \tag{2.7}$$

respectively, where G_k is the turbulent kinetic energy production rate caused by average velocity gradients.

$$C_{2\varepsilon}^{*} = C_{2\varepsilon} + \frac{C_\mu \eta^3(1-\eta/\eta_0)}{1+\beta\eta^3} \tag{2.8}$$

where $\eta = Sk/\varepsilon$ and S, the strain tensor, is equal to $(2S_{ij}S_{ij})^{1/2}$. The remaining RNG modeling parameters are $\sigma_k = 0.7194$, $\sigma_\varepsilon = 0.7194$, $C_{1\varepsilon} = 1.42$, $C_{2\varepsilon} = 1.68$, $\beta = 0.012$, and $\varepsilon_o = 4.38$ [32].

2.2.2 Turbulence-Chemistry Interactions

Modeling a combustion process requires a turbulence-chemistry interaction model that can consider different types of turbulent and chemical time scales, varying from dispersed and slow chemical reactions to turbulent and fast chemical reactions. In addition, the model must account for both premixed and diffusion flames as well as partially premixed combustion. The

local stirred reactor model satisfies these conditions. Although this model doesn't calculate the molecular fluxes directly, it considers their effects by including the Kolmogorov scale mixing in the interaction and turbulent motion of the reacting regions. The Chalmers PaSR (Partially Stirred Reactor) model [33], used in the current simulation, was originally derived from this approach.

The PaSR model is based on the theory that real flames are much thinner than any computational cell, thus assuming that an entire cell is a perfect reactor would severely overestimate the burning rate. Therefore, in the PaSR model, the cells are divided into a reacting zone and a non-reacting zone. The reacting zone is treated like a perfectly-stirred reactor, in which all present species are homogeneously mixed. After reactions occur, the species are assumed to be mixed due to turbulence for the mixing time, t_{mix}, and the subsequent concentration gives the final concentration in the entire cell. The interaction among all the cells takes place by exchange with the mean. For the mean value we have

$$Y_i = (1-\kappa)Y_i^N + \kappa Y_i^R \tag{2.9}$$

where κ is the reactive volume fraction of the cell available for chemical processes, and the superscripts *R* and *N* account for the reacting and the nonreacting volume, respectively.

For a steady state problem the relative sizes of the zones of the computational cell forming the reactor and the rest of the cell, are governed by the flow time scale, turbulent mixing time and residence. Hence, the reaction rate of the i^{th} species is scaled by the reactive volume fraction, κ [34] as follows:

$$\kappa = \frac{t_f + t_c}{t_f + t_c + t_{mix}} \tag{2.10}$$

where t_f, t_c, and t_{mix} are the time scales of flow, chemical kinetics, and turbulent mixing, respectively. The way of finding the time scales is explained in Ref. [34].

The Chalmers PaSR turbulent combustion model is then coupled with a two-step chemical reaction mechanism [35] employing CHEMKIN thermophysical data (see Appendix A).

2.2.3 Radiation Heat Transfer Modeling

Most flames/fires in combustion equipment, such as boilers, gas turbines, internal combustion engines, etc., involve high temperatures. Therefore, thermal radiation potentially plays a very important role in the overall combustion physics of flames, such as the one in the BERL burner.

Thermal radiation in a participating medium is expressed by the radiative transfer equation (RTE) [36],

$$\frac{dI_\lambda\left[\mathbf{r}(u),\mathbf{s}\right]}{du} = -\left\{a_\lambda\left[\mathbf{r}(u)\right]+\sigma_{s\lambda}\left[\mathbf{r}(u)\right]\right\} I_\lambda\left\{\left[\mathbf{r}(u)\right],\mathbf{s}\right\} + a_\lambda\left[\mathbf{r}(u)\right] I_{\lambda,b}\left\{T\left[\mathbf{r}(u)\right]\right\} + \frac{\sigma_{s\lambda}\left[\mathbf{r}(u)\right]}{4\pi}\int_{4\pi} I_\lambda\left\{\left[\mathbf{r}(u)\right],\mathbf{s}'\right\}\Phi_\lambda\left[\mathbf{r}(u),\mathbf{s},\mathbf{s}'\right]d\omega' \tag{2.11}$$

where $dI_\lambda[\mathbf{r}(u), \mathbf{s}]/u$ is the rate of increase in spectral intensity at a location $\mathbf{r}(u)$ along a ray in the direction of the unit vector $\mathbf{s}$, α_λ is the spectral absorption coefficient, $\sigma_{s\lambda}$ is the scattering coefficient, $I_b\{T[\mathbf{r}(u)]\}$ is the local blackbody intensity, and $\Phi_\lambda[\mathbf{r}(u), \mathbf{s}, \mathbf{s}']$ is the scattering phase function which defines the fraction of intensity scattered from the $\mathbf{s}'$direction into the $\mathbf{s}$ direction. The geometry corresponding to Eq. (2.11) is shown in Figure 2.1.

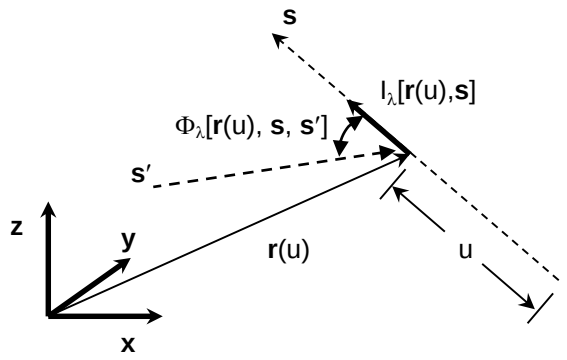


Figure 2.1: Geometry corresponding to the radiative transfer equation, Eq. (2.11)

In principle the spectral intensity can be found at all wavelengths, directions, and locations in the medium by solving the RTE with boundary conditions. The total intensity $I(\mathbf{r}, \mathbf{s})$ would be found by integrating the spectral intensity over all wavelengths, and the components of a radiative flux vector, $\boldsymbol{q}_r$, could then be found by

$$q_{r,n}(\mathbf{r}) = \int_{4\pi} I(\mathbf{r},\mathbf{s})\mathbf{s}\cdot\mathbf{n}\, d\omega \tag{2.12}$$

where $\boldsymbol{n}$ is the unit vector corresponding to the n^{th} component of $\boldsymbol{q}_r$. Finally, the radiative source term is equal to the negative of the divergence of the radiative flux vector,

$$q_{rad}(\mathbf{r}) = -\nabla\cdot q_r(\mathbf{r}) \tag{2.13}$$

which appears in the energy equation used to solve for the temperature of the combustion gases.

Using the process explained above results in a complete solution for the radiation heat transfer; it is not a practical way of studying radiation in many real-world problems, however, since it is extremely computationally expensive.

Accordingly, OpenFOAM simplifies the radiation subproblem by assuming that the participating medium is a homogeneous grey gas, meaning that the radiative properties are uniform with respect to wavelength and isotropic over the problem domain. In some combustion

simulations this treatment would be grossly inaccurate due to the highly spectral nature of gas radiation, and the large spatial variation in gas composition. In the current problem this approximation is reasonable because the turbulent flow produces a well-mixed gas, and radiation is dominated by soot particles that emit radiation approximately uniformly over the wavelength band important to thermal radiation. As a first approximation, it is also reasonable to neglect scattering, since $\sigma_{s\lambda} << a_\lambda$ over the wavelengths of interest [37]. (Scattering becomes important in the presence of larger particles, e.g. cenospheres typical of heavy-oil droplet combustion.) With these assumptions Eq. (2.11) simplifies to an ordinary differential equation

$$\frac{dI\left(\mathbf{r}\left(u\right),\mathbf{s}\right)}{du} = -aI\left\{\left[\mathbf{r}\left(u\right)\right],\mathbf{s}\right\} + a\left\{\left[\mathbf{r}\left(u\right)\right],\mathbf{s}\right\} I_{\lambda,b}\left\{T\left[\mathbf{r}\left(u\right)\right]\right\} \tag{2.14}$$

which are solved using the P_1 method [38].

To carry out this solution, it is necessary to specify the total absorption coefficient, *a*, which for a luminous flame depends on the size and concentration of particles present in the flame. Since these attributes are unknown for the current burner, we estimate *a* from optically-determined emissivities reported in the literature for similar liquid fuel flames [39]; a typical experimental apparatus for measuring ε_{flame} is shown in Figure 2.2. Flame emissivity is related to the absorption coefficient by

$$\varepsilon_{flame} = \frac{I_{detect}}{I_b\left(T_{flame}\right)} = \frac{I_{detect}}{\sigma T_{flame}^4/\pi} = 1 - \exp\left[-aL_e\right] \tag{2.15}$$

where L_e is the detection path length through the flame, $\sigma = 5.67\times10^{-8}$ W/(m^{2}K^4) is the Stefan-Boltzmann constant, and T_{flame} is the flame temperature, usually determined by multi-wavelength pyrometry. This expression can be obtained by solving Eq. (2.14) along the detection path length, assuming the incident intensity is zero. Rearranging Eq. (2.15) results in

$$a = \frac{1}{L_e} \ln(1 - \varepsilon_{flame}) \tag{2.16}$$

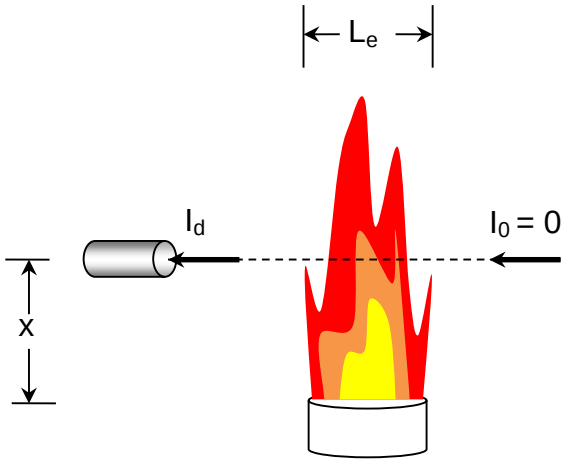


Figure 2.2: Experimental apparatus for determining flame emissivity [4]

2.2.4 Chemistry Solver

Many chemical reactions take place in a typical combustion process. For any reaction which occurs, the reaction rate constant, k , is calculated by the Arrhenius equation,

$$k = AT^{\alpha} \exp\left(-\frac{E_{act}}{RT}\right) \tag{2.17}$$

where A is frequency (steric) factor, α is the temperature exponential constant, and E_{act} is the activation energy for that specific reaction.

Based on the reaction mechanism that occurs during the combustion process the source term for species i can be defined as:

$$\dot{r}_i = \frac{W_i}{\rho} \sum_{j=1}^{N_r} \left\{ \left(\nu_{ij}^{f} - \nu_{ij}^{b} \right) . \left(k_j^{f} \prod_{i=1}^{N_s} [X_i]^{\nu_{ij}^{f}} - k_j^{b} \prod_{i=1}^{N_s} [X_i]^{\nu_{ij}^{b}} \right) \right\} \tag{2.18}$$

where N_r, ν, and X_i represent the number of reactions, stoichiometric coefficients, and species molar fraction respectively, while notations *j*, *f*, and *b* correspond to the reaction number, reactants, and products respectively.

This equation is formulated for every species included in the chemical mechanism, as well as for every reaction, resulting in an equation system consisting of $N_r \times N_s$ equations. As can be seen from the above equation, it is a system of ordinary differential equations (ODEs), which can be solved using an ODE solver, using sequential method presented by [40], or by an Euler-Implicit approach or to solve the equations using an ODE solver [41]. In the current research, the Euler-Implicit method is used due to its robustness in solving various stiff differential equations.

2.2.5 CFD Solver Implementation

As mentioned before, OpenFOAM [2], a finite-volume based CFD open source code, is employed to solve the time-averaged Eulerian equations for the conservation of mass, momentum, species mass fraction, and enthalpy along with the transport equations for k and ε for the gas phase. The solver implemented in the current work is mainly written based on the original solver called "reactingFoam". Since it is a transient chemical reaction solver, some modifications are necessary to be made to change it into a steady-state solver. By coupling of the chemistry to the flow time, using the SIMPLE algorithm for P-U coupling [42], and stabilization of the solution by the reduction of κ (if $\left|Y_i^{old} - Y_i^{new}\right|$ is too big), the steady-state solver is created from "reactingFoam". Here, the general algorithm in order to run the steady-state solver is shown in Figure 2.3 which continues until it reaches specific convergence criterion.

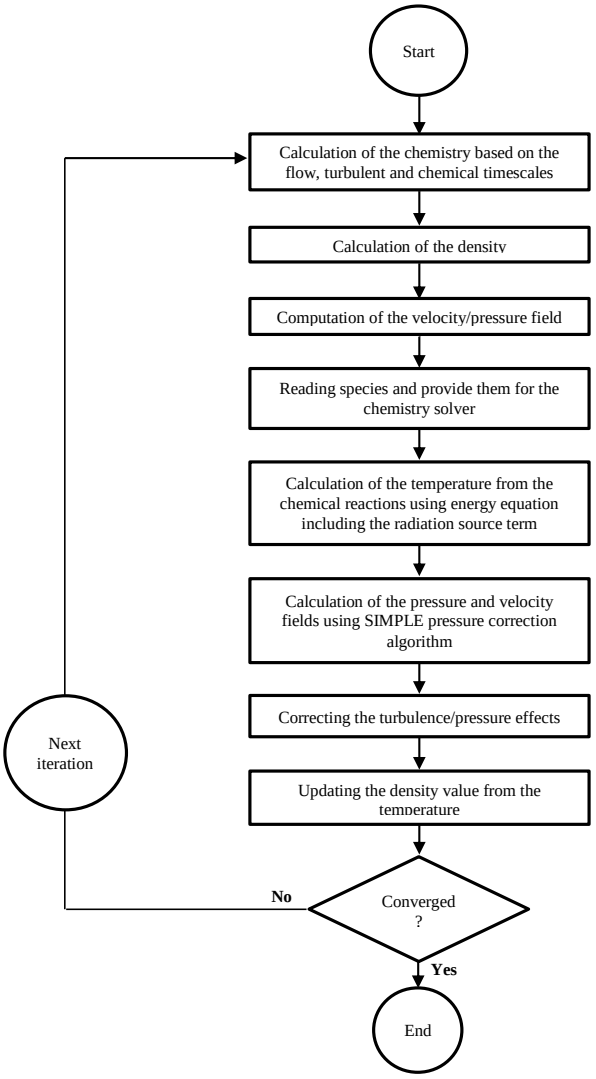


Figure 2.3: The flowchart of the CFD solver used to model the gaseous fuel combustion

2.2.6 Boundary Conditions

The BERL furnace is vertically fired and considered to have an axisymmetric cylindrical geometry. The exhaust gas exits the conical hood through a cylindrical duct. Dirichlet boundary conditions are applied at the inlet of the chamber, excluding the pressure where is found using zero gradients along the flow direction. At the outlet, the atmospheric pressure is fixed, while for other variables, by being sufficiently away from the turbulent region, the boundary conditions

are satisfied assuming zero gradients along the length of the furnace, except for the velocity which is calculated by satisfying the law of continuity. The wall function method [43] is employed to model the flow near the wall. On the walls, the velocity is set to zero, and the experimental data from [3] are put as wall temperature distribution, while the rest of the problem parameters, e.g. species mass fraction, are considered to be zero gradient along the furnace diameter. The boundary conditions applied on the furnace are sketched in Figure 2.4. Detailed geometry of the furnace as well as the inlet conditions are given in Section 4.2.

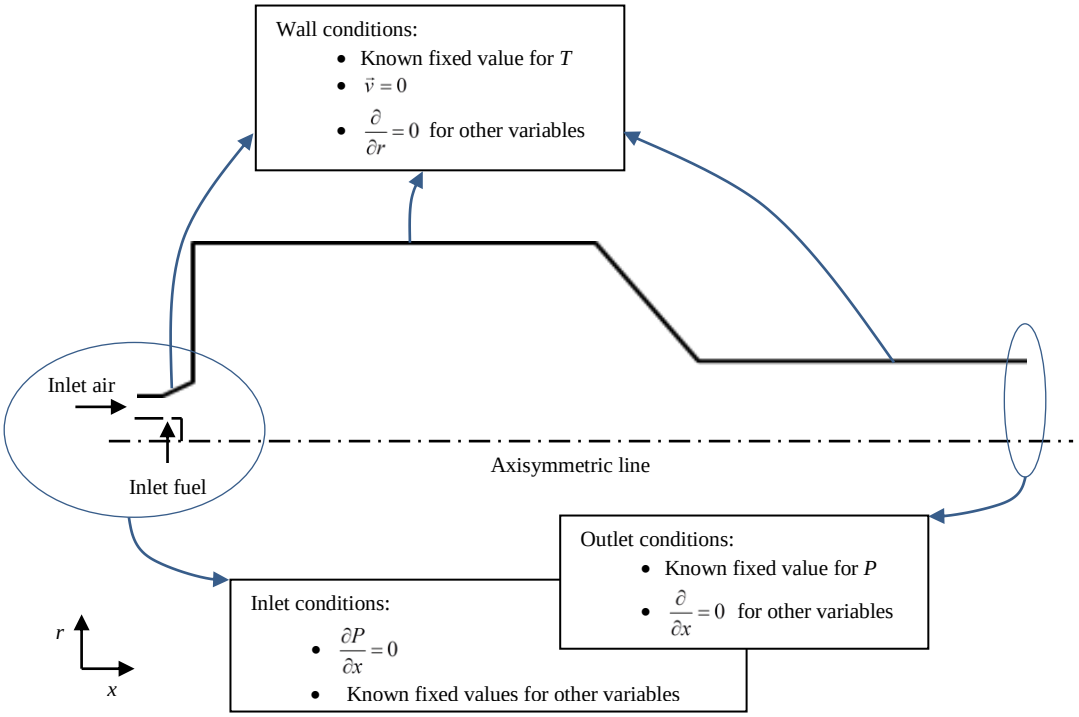


Figure 2.4: Schematic of the boundary conditions used for the BERL furnace

2.3 Combustion of Liquid Fuel Sprays

Many industrial processes involve multi-phase flow, phase transformation and complex chemical reactions linked with heat transfer. This is particularly true of the power generation and processing industries. Combustion of liquid fuels is one of the largest sources of energy production. Accordingly, studying the behaviour of reacting flow inside fuel oil combustors is

one of most important fields of combustion modeling. This is the main reason that combustion of diesel fuel inside the GenTex process heater is studied here.

In a typical liquid fuel combustion problem many complex phenomena take place such as evaporation, collision, break-up, and several other gas-liquid interactions. Therefore, having a deep understanding of these multi-phase processes is necessary to deliver well-validated simulation results.

Accordingly, an unsteady Eulerian-Lagrangian approach is implemented in order to model the spray-gas phase interactions inside the GenTex process heater. Since modeling the near-nozzle flow using only Eulerian cells is complicated and time-consuming, in the present work the spray is modelled in Lagrangian coordinates. Several sub-models are used to model the different physical phenomena occurring at the spray droplets. These sub-models are described following the current chapter.

2.3.1 Governing equations of the gas phase

The Eulerian coordinate system is used to discretize the governing equations of the gas phase. The conservation of mass for a gaseous system including N_s different species can be written as

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \left(\rho \vec{\mathbf{v}} \right) = S_{mp} \tag{2.19}$$

The evaporation source term, S_{mp}, is calculated from the evaporation model (see Section 2.3.6.1).

Conservation of species requires that each species is transported by diffusion and advection or is formed or consumed by chemical reactions. Thus, the, the conservation equation for the i^{th} species is

$$\frac{\partial(\rho Y_i)}{\partial t} + \nabla \cdot (\rho \vec{\mathbf{v}} Y_i) = \nabla \cdot (\mu_{eff} \nabla \cdot Y_i) + \kappa \dot{r}_i + S_{yp} \tag{2.20}$$

where Y_i is the mass fraction and $\kappa \dot{r}_i$ is the reaction rate for species *i*. The value of $\dot{r}_i$ is found from Eq. (2.18) and the reactive volume fraction, κ, is defined in section 2.3.4. S_{yp} represents the spray mass fraction source term.

The Lagrangian particles bring a modification to the momentum equation as well. By making the Reynolds Average Navier-Stokes assumption the momentum-force balance is

$$\frac{\partial(\rho \vec{\mathbf{v}})}{\partial t} + \nabla \cdot (\rho \vec{\mathbf{v}} \vec{\mathbf{v}} - \tau_{eff}) = -\nabla P + \rho \vec{\mathbf{g}} + S_{pp} \tag{2.21}$$

where S_{pp} is the spray momentum source term.

Similarly, conservation of energy is

$$\frac{\partial(\rho h)}{\partial t} + \nabla \cdot (\rho \vec{\mathbf{v}} h) = \nabla \cdot (\alpha_{eff} \nabla h) + \frac{\partial \rho}{\partial t} h_T - \nabla \cdot \mathbf{q}_r + S_{ep} \tag{2.22}$$

where $\vec{\mathbf{v}}$, *Y*, *h*, *P*, and ρ are the velocity vector, mass fraction, enthalpy, pressure, and the flow density respectively, $\nabla \cdot \mathbf{q}_r$ is the radiative source term provided by the P_1 model, and S_{ep} is the spray energy source term. As with gas-phase combustion, turbulence is modelled using the RNG *k*-ε model.

2.3.2 Turbulent Spray

The gas turbulence equations, Eqs. (2.6) and (2.7), must be modified to account for turbulence-droplet interaction. One way to do this would be to use a source term in the governing equations as mentioned in the previous section, however there are some other ways to model the effect of the droplets. The method which is used in the current simulation is to confine the turbulent length scale to the diameter of the orifice in the grids which contain the droplets. Actually, the spray is defined as a group of parcels, as a consequence, all of the cells that have parcels in them will be the representative of this group and since in the usual spray combustion problems many of the cells contain the parcels, determining the jet diameter at every time-step would be very expensive. Thus, as an alternative, the orifice diameter is fixed. First, the length scale of the turbulence is defined as:

$$l_t = C_\mu k^{3/2} / \varepsilon \tag{2.23}$$

Next, a limit is forced on ε,

$$\varepsilon > C_\mu k^{3/2} / l_t \tag{2.24}$$

where, here, l_t is set to the nozzle diameter. Since the diesel sprays have fairly high momentum they influence the gas flow and increase the gas momentum. Therefore, based on this, the length scale of the turbulence is imposed by the fuel spray and consequently by the orifice diameter.

2.3.3 Chemical Kinetics

After the fuel droplets evaporate (see Section 2.3.6.1), the fuel mixes and reacts with the air in the gas phase. Diesel fuel includes many different species but it is normally modeled by a

single species which shows the closest behaviour to the diesel. In most studies “n-heptane” is used to represent diesel fuel [44].

Based on the reaction mechanism which occurs during the combustion process the source term for each species is determined in the same way described in Section 2.2.4. Solving a detailed chemical mechanism will make the governing equations stiff and also since these equations must be solved for every time-step the calculation time would be very large.

One of the reasons of the numerical stiffness is that the chemical reactions in combustion have small timescales which normally do not meet with flow timescales [43]. Therefore, solving an additional transport equation for each chemical species would strictly limit the simulation. In addition to the first reason, unresolved turbulence-chemistry interactions due to the fluctuations of scalar fields are substantial [45]. In the current study, these interactions are modelled by the Favre averaged transport equation [46] where the average reaction rate is highly nonlinear, hence the average rate cannot be calculated using the average quantities it depends on [47]. In fact, the only condition that the average rate could be set equal to the related average quantities is when the Damkohler number (indication of the relative magnitude of gas diffusional and surface kinetic resistance) is very small, meaning that the reaction timescales are much greater than turbulent timescales; however, since the current studied furnace is a diffusion-limited combustion problem (large Damkohler number), this condition doesn’t hold and the reaction rate is highly nonlinear [48]. As a consequence, in the current simulation, only a two-step reduced mechanism is used to model the chemical mechanism.

2.3.4 Turbulent Combustion Model

Similar to the BERL burner combustion, the Chalmers PaSR (Partially Stirred Reactor) model is used to model the turbulence/chemistry interactions, but since here the problem is unsteady-state, the reactive volume fraction is defined differently. It is, in fact, presumed that the reaction rate is controlled by the turbulent mixing time and the residence time [33]. The reaction rate term for species i is approximated as $\kappa \dot{r}_i$ from Eq. (2.20) and κ, the reactive volume fraction, is calculated as:

$$\kappa = \frac{t_c}{t_c + t_{mix}} \tag{2.25}$$

where t_c is the chemical timescale, and t_{mix} is the turbulence timescale which is calculated as:

$$t_{mix} = C_{mix} \frac{k}{\varepsilon} \tag{2.26}$$

where C_{mix} = 0.03. Other variations and detailed a description of the Partially-Stirred model can be found in Refs. [43, 49].

2.3.5 Radiation Heat Transfer

A previous analysis on a GenTex process heater estimated that radiation heat transfer accounts for approximately 80% of the heat transferred from the hot combustion gases to the inner surface of the inner coil [50]. The combustion of fuel oils results in the formation of soot and char particles. In the hot region of the combustor, soot and other particulates along with the fuel oil droplets (which are relatively large in size) absorb heat from the combustion gases and emit the heat to the surrounding surfaces, resulting in significant temperature reduction of the combustion gases. In other words, radiation heat transfer from the hot combustion gases to the

surrounding cold surfaces acts like a volumetric heat sink, which reduces the temperature of the combustion gases sometimes by as much as 500°C [51]. It is therefore imperative to include radiation heat transfer in the CFD model of the GenTex burner.

Here again, since the flame is luminous, and because fuel and air are well-mixed, the participating medium could be treated as a homogeneous grey gas. Once more, the P_1 model is implemented to solve RTE. Since the radiative properties of the gas mixture are unknown for the current burner, similar to the BERL burner, the total absorption coefficient, *a*, is estimated from optically-determined emissivities reported in the literature. Results from [52] show flame emissivities that vary with height along the flame, but typically $0.3 \leq \varepsilon \leq 0.5$ for $L_e \approx 0.1$ m, which is consistent with other published values for similar flames [53, 54]. Therefore, substituting these emissivity values in Eq. (2.16) gives $4\ \text{m}^{-1} \leq a \leq 7\ \text{m}^{-1}$, where the value of $a = 5\ \text{m}^{-1}$ is used in the current research.

2.3.6 Governing equations of the liquid phase

2.3.6.1 Evaporation Model

The evaporation rate of the droplet is assumed to follow the D^2-law [55] where it is assumed that the square of the droplet diameter, *D*, decays linearly with time

$$D^2 - D_0^2 = C_{evap}\ t_e \tag{2.27}$$

where C_{evap} is a constant found from

$$C_{evap} = -4Sh\frac{\rho_v D_{AB}}{\rho_p}\ln\left(1+\frac{X_{v,0}-X_{v,\infty}}{1-X_{v,0}}\right) \tag{2.28}$$

where D_{AB} is binary diffusion coefficient, *Sh* is Sherwood number and X_v stands for the molar fraction of the fuel vapor. The subscripts *0* and ∞ represent location on the droplet surface and

in the ambient gas, respectively. The Sherwood number is calculated using the Ranz-Marshall correlation,

$$Sh = 2 + 0.6\,\mathrm{Re}^{1/2}\,Sc^{1/3} \tag{2.29}$$

Therefore, using the D^2 assumption gives the droplet evaporation rate

$$\frac{dm_p}{dt} = \frac{\pi}{2}\rho_p D^2 \frac{dD}{dt} = -\frac{m_p}{t_e} \tag{2.30}$$

where m_p is the droplet's mass. The lifetime of droplet, t_e, is then calculated as

$$t_e = \frac{\rho_p D^2}{6\,D_{AB} Sh\,\rho_v \ln\left(1 + \dfrac{X_{v,0} - X_{v,\infty}}{1 - X_{v,0}}\right)} \tag{2.31}$$

The relaxation time, t_e, is introduced to characterise the evolution of the particle size. It is important to know that if we use an explicit method, the time-step (a function of the relaxation timescale) must not be larger than the relaxation time, otherwise the mass will become negative. On the other hand, if an implicit method is used, the method would be unconditionally stable.

2.3.6.2 Spray Momentum Equation

The motion of a Lagrangian particle, moving in the Eulerian cells, is

$$\frac{\partial u_p}{\partial t} = -\frac{3}{4}\frac{\rho_g}{\rho_p}\frac{1}{D} C_D \left(u_p - U\right)\left|u_p - U\right| + g \tag{2.32}$$

where C_D, and D are the drag coefficient and droplet's diameter respectively. The right-hand side terms represent the effect of drag force and gravity respectively. Also, it should be mentioned that the effect of temporal variation of droplet mass on drag coefficient has been neglected.

2.3.6.3 Spray Heat Transfer Equation

The heat transfer model is based on the convective heat transfer of a particle with a uniform temperature. Also a modification is made to consider the latent heat transfer because of the mass transfer due to evaporation

$$\frac{dT_p}{dt} = \frac{\pi D \kappa_c Nu}{m_p c_1}\left(T_g - T_p\right) f_{heat} - \frac{1}{c_1}\frac{h_v\left(T_p\right)}{t_e} \tag{2.33}$$

This modification is in the factor f_{heat} which is

$$f_{heat} = \frac{-\frac{c_{p,v}\dot{m}_p}{\pi D \kappa_c Nu}}{\left(e^{-\frac{c_{p,v}\dot{m}}{\pi D \kappa_c Nu}} - 1\right)} \tag{2.34}$$

The first term of Eq. (2.33) stands for the heat transfer to the liquid, and the second part represents latent heat coming from mass transfer. Since the evaporation of a droplet also transfers heat to the surrounding gas, the heat transfer model also uses the evaporation relaxation time. For more details see [56].

2.3.6.4 Particle Tracking

Tracking the particles is the most fundamental action in Lagrangian simulation. In the current study, the Face-to-Face approach (F2F) [57] is used for tracking the parcels. In this approach, the parcel is moved towards its path until it reaches the boundary of the current cell or for the entire time step if the parcel stays in the same cell. When the parcel reaches the face of a cell, the code calculates the source term due to this trip inside the cell and exchanges it with the Eulerian grid, then the parcel continues the trip for the remaining part of the time-step. This procedure is repeated at every new cell until the droplet is totally evaporated or exits the problem

domain. The benefit of this method is that the parcel exchanges mass, momentum and energy with every cell that it passes through for any value of time-step. Accordingly, this approach brings more accuracy and better stability conditions compared to the methods that only consider the parcel's path without considering the number of cells that it passes where, in this case, the parcel could move far away from the previous point by passing through several cells. In addition, the F2F method doesn't need any search algorithm to find the parcels, which saves considerable amount of computational time.

2.3.6.5 Injection Model

A multi-hole injector is used to spray the diesel fuel. This injector contains 6 orifices which are spread circumferentially. The details of the injector configuration are given in Section 4.3.1. Here, the Rosin-Rammler probability density function (PDF) is used to model the spray size distribution. The parameters for the model are obtained as shown in Section 4.3.3 and are summarized in Table 2.1.

Table 2.1: Rosin-Rammler constants

Model Parameter	Value
$\tilde{D}$ (μm)	25.857
n (Spread parameter)	1.362

The spray exiting each of the orifices is shaped like a solid-cone, thus the injection model used in the Lagrangian spray simulations is set to be a solid-cone spray. The apex angle of the spray cone is measured to be equal to 45°. In order to find the angle between the set spray direction and the direction of the injected droplet, the spray angle is multiplied by a random

number between 0 and 1. Consequently, the obtained angle defines the injection direction of a droplet into the domain.

By applying the Bernoulli equation [58] across the nozzle of the injector, the velocity of the injected parcel is found; however due to the friction across the nozzle (no-friction is one of the conditions that holds the Bernoulli equation), the final velocity value is multiplied by the discharge coefficient, C_d, in order to consider the effect of friction on the reduction of the velocity of the flow leaving the nozzle. C_d <1 depends on the type and size of the nozzle. As a result, the velocity of the injected parcel, which is based on the injection pressure and the pressure in the domain, is calculated as

$$u_{inj} = C_d \sqrt{2\left(p_{inj} - p_{amb}\right)/\rho} \tag{2.35}$$

where P_{inj} is the injection pressure of the fuel at the tip of the nozzle, P_{amb} is the average pressure in the computational domain which is updated at each injection, and ρ is the density of the injected droplets.

2.3.6.6 Breakup Model

The breakup model used is the Kelvin-Helmholtz-Rayleigh-Taylor (KHRT) model. This model is one of the most widely used models in Lagrangian spray simulations today [59, 60], and was chosen for its proven performance with diesel fuel [61]. The KHRT model includes two modes of breakup: Kelvin-Helmholtz breakup which accounts for unstable waves growing on the liquid jet due to the differences in velocity between the gas and liquid; and Rayleigh-Taylor breakup which accounts for waves growing on the droplets' surface due to acceleration normal to the droplet-gas interface [60]. Studies performed on a round liquid jet has given a fastest growing wave with a wavelength Λ_{KH} and a growth rate Ω_{KH} as follows:

$$\Lambda_{KH} = 9.02 \frac{r(1+0.45Oh^{1/2})(1+0.4Ta^{0.7})}{\left(1+0.865We^{1.67}\right)^{0.6}} \tag{2.36}$$

$$\Omega_{KH} = \frac{\left(0.34+0.38We^{3/2}\right)}{\left(1+Oh\right)\left(1+1.4Ta^{0.6}\right)} \sqrt{\frac{\sigma}{\rho_p r^3}} \tag{2.37}$$

The critical droplet radius r_c, is defined as the radius of the droplet which will be made when the liquid jet is broken. It is assumed to depend linearly to the Λ_{KH} and the stripping rate. Also, the controlling factor for break-up rate, t_{KH} , is assumed to be a function of the growth rate, , Ω_{KH} , wave-length, Λ_{KH} , and the droplet radius, r,

$$r_c = B_0 \Lambda_{KH} \tag{2.38}$$

$$t_{KH} = \frac{3.726 B_1 r}{\Omega_{KH} \Lambda_{KH}} \tag{2.39}$$

The values of the constants with detailed descriptions are available in Ref. [49].

Rayleigh-Taylor breakup is governed by how quickly disturbances grow on the surface of the droplet. For real droplets these disturbances are created from the droplets' trailing edges [62]. If these disturbances are assumed to be linear, the frequency of the fastest growing wave, and the corresponding wavelength, will be respectively equal to

$$\Omega_{RT} = \sqrt{\frac{2\left|g_t\left(\rho_l - \rho_g\right)\right|^{3/2}}{3\sqrt{3\sigma}\left(\rho_l + \rho_g\right)}} \tag{2.40}$$

$$\Lambda_{RT} = 2\pi C_{RT} \sqrt{\frac{3\sigma}{\left|g_t\left(\rho_l - \rho_g\right)\right|}} \tag{2.41}$$

where C_{RT} is a modelling parameter. If this wavelength of the fastest growing wave is smaller than the diameter of the droplet diameter and the perturbations are allowed to grow for a period

of time, the droplet will be immediately converted into a parcel with smaller size. For more detailed information see Ref. [63].

2.3.6.7 Drag Model

The drag coefficient is defined using the following relation

$$C_D = \begin{cases} \frac{24}{\mathrm{Re}_p} + \frac{4}{\mathrm{Re}_p^{1/3}} & \mathrm{Re}_p < 1000 \\ 0.44 & \mathrm{Re}_p > 1000 \end{cases} \tag{2.42}$$

The current open source [2] also offers the possibility of changes in drag due to oscillations of the droplet surface. These oscillations are the instability waves which eventually result in droplet breakup. The breakup model will be used to calculate the oscillations and the resulting drag will be added to the drag described above. Thus the modification will be:

$$C_{D,\mathrm{modified}} = C_D \left(1 + C_{D,distort} \min(y_{\lim}, y)\right) \tag{2.43}$$

where $C_{D,distort}$ and $y_{\lim}$ are constants based on the phenomenon of spray injection and fuel properties, and y is the relative deviation from the equator of the droplet if it was spherical.

2.3.6.8 Collision

The Trajectory model is picked in the current simulation to model collisions among the droplets. In this model, the direction of the droplets is taken into account (see Figure 2.5).

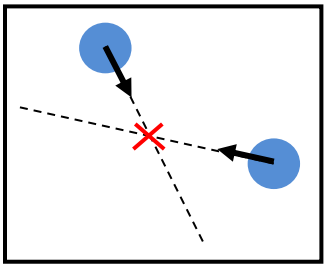


Figure 2.5: Collision modeling using Trajectory method

In addition, the Trajectory model checks if a collision is possible within the current time step (see Eq. (2.44)), meaning that the parcels need to be close enough to collide. Hence, the distance between the parcels must be smaller than the maximal distance defined by the product of relative velocity between the parcels and the time-step. This gives the following criterion

$$u_{align}\Delta t > \left|x_2 - x_1\right| - \frac{d_{\max} + d_{\min}}{2} \tag{2.44}$$

where

$$u_{align} = u_{rel} \cdot \frac{x_2 - x_1}{\left|x_2 - x_1\right|} \tag{2.45}$$

This confirms that the parcels can reach each other within the given numerical timestep. The other important criterion that should be considered is whether or not the parcels reach the intersection at the same time. Therefore, the algorithm needs to determine the times where the parcels reach the intersection. According to this criterion and by using Eqs. (2.44) and (2.45), the probability for a collision is found as [64]

$$\mathrm{P}_{collision} = \left(\frac{\left(d_1 + d_2\right)/2}{\max\left(\left(d_1 + d_2\right)/2,\ \left|x_{1,new}\left(t_1\right) - x_{2,new}\left(t_2\right)\right|\right)} \right)^{C_{space}} e^{-C_{time}\left|t_1 - t_2\right|/\Delta t} \tag{2.46}$$

where C_{space} and C_{time} are the model constants that control the collision rate in time and space, and t_1 and t_2 are the times that the parcels reach the intersection. More details of collision modeling are given in Ref. [64].

2.3.7 CFD Solver Implementation

The solver implemented in the current work is mainly written based on the original unsteady liquid fuel combustion solver called “dieselFoam”. The general algorithm in order to run “dieselFoam” is shown in Figure 2.6.

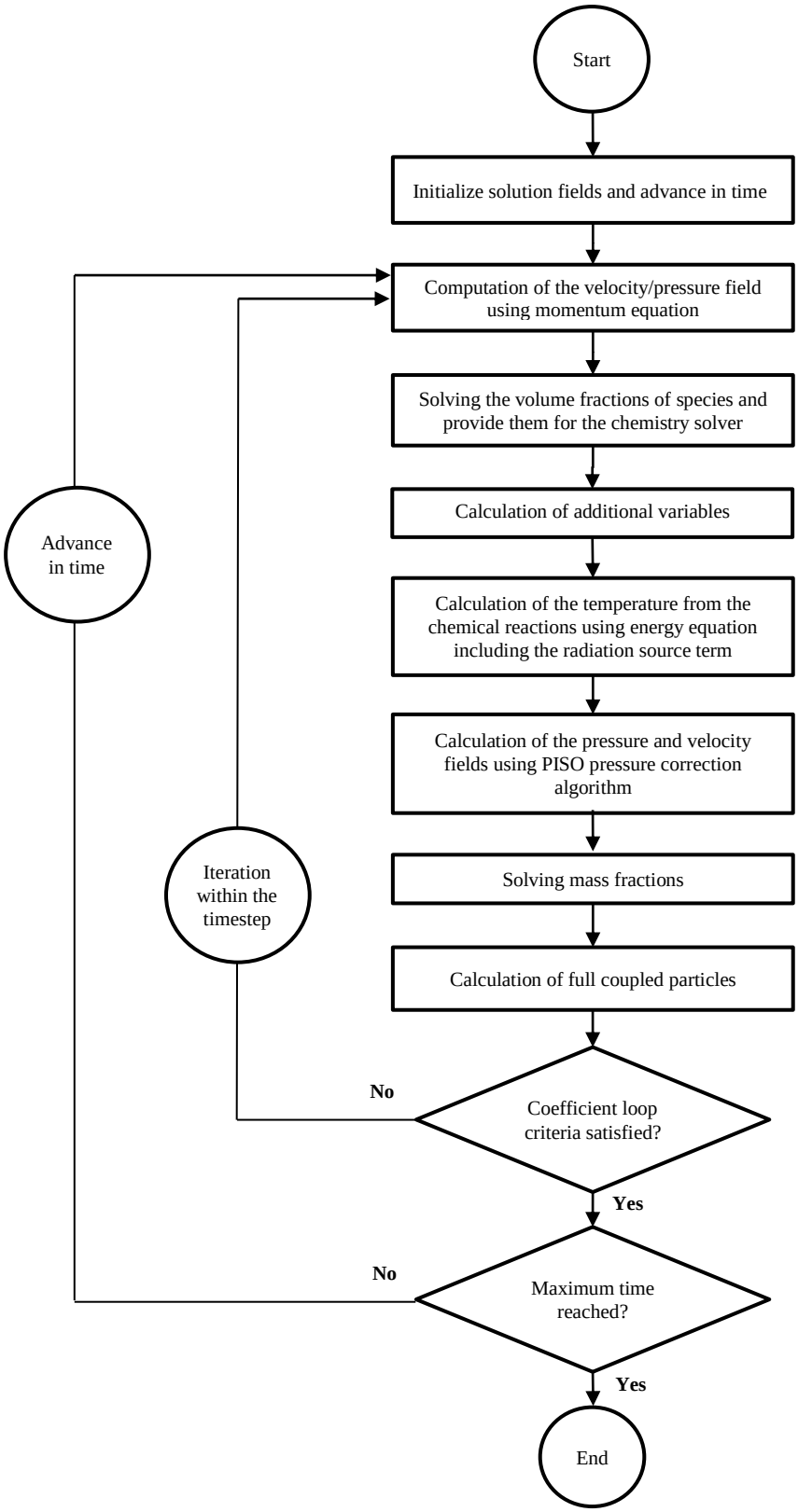


Figure 2.6: The flowchart of the CFD solver used to model the liquid fuel combustion [65]

2.3.8 Boundary Conditions

The GenTex process heater is a cylindrical horizontally fired combustor. It consists of three layers of water coils, i.e. innermost, intermediate and outermost, as the combustion gases pass through them and heat the water inside the coils. The details of the heater configuration are given in Section 4.3.2. For the current study, the reacting flow inside the innermost layer is modelled as shown in Figure 4.7. The type of the boundary conditions applied on the GenTex heater was the same as the BERL furnace as explained in Section 2.2.6. The operating conditions of the process heater are summarized in Table 4.2.

Chapter 3
Optimization Method

3.1 Introduction

The optimization method used in the current study is explained in the present chapter. First, a brief description of the design optimization is given, then two principal gradient-based methods are studied and integrated in order to be employed as a part of the optimization procedure. The major framework of the optimization algorithm used in the current research is established through the Response Surface Methodology (RSM) method. Finally, this chapter gives the details of using RSM in multivariate problems along with the challenges of this technique. The advantages of Response Surface Methodology over conventional methods for the current problem are also mentioned in the context.

3.2 Concept of Optimization

Many problems aim to minimize or maximize a mathematical function which is made of one or more variables, and is subjected to certain constraints. These comprise a class of problems called optimization problems. Many actual and theoretical problems can be modelled in this general structure.

The term optimization is usually used to replace the terms minimization or maximization. The mathematical function to be optimized is known as the objective function, which usually

depends on several variables. An objective function can be a function of a single variable for simple problems; however a single-variable function may not be able to satisfy some optimization problems. Indeed, many practical optimization problems involve more than one variable and require multivariate optimization. Optimization problems are typically classified based on the types of the objective function (single or composite), the problem boundary (unconstrained or constrained), the variable types (continuous or discrete), and the function behaviour (linear or nonlinear).

As mentioned before, the objective of this research is to demonstrate how model-based optimization methodology can be applied to design industrial combustion equipment. As noted above, this technique transforms the design problem into a multivariate minimization problem by defining a vector **x** of design parameters which specify the design, and an objective function, $F(\mathbf{x})$, which quantifies the final design quality. It is also possible to impose inequality constraints on the design parameters having the form $\mathbf{c}(\mathbf{x}) \leq 0$. The objective, then, is to solve

$$\mathbf{x}^* = \arg\min \left\{ F\left(\mathbf{x}\right) \right\} \; s.t. \; \mathbf{c}\left(\mathbf{x}\right) \leq 0 \tag{3.1}$$

3.3 Optimization Search Approach

The numerical optimization of general nonlinear multivariate objective functions requires efficient and robust techniques. Efficiency is important because these problems are solved by iteration, involving many function evaluations. If the cost of evaluating every $F(\mathbf{x})$ is high for a typical optimization problem, therefore trial and error (method of manipulating the variables with no specific rule in order to sort through possibilities which may result in a better outcome) would not be practical for more than one independent variable. Robustness (the ability to reliably achieve a solution) is a key point as well, because a general nonlinear function is unpredictable

in its behaviour; there may be a relative maximum or minimum, saddle points, regions of convexity, concavity, and so on. In some regions the optimization algorithm may progress very gradually toward the optimum point, demanding excessive computer time.

Gradient-based minimization works by generating a search direction, $\mathbf{p}^k$, and a step size, α^k, at each iteration. The design parameters are then updated by

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{p}^k \tag{3.2}$$

A good search direction should reduce (for minimization) the objective function so that if $\mathbf{x}^k$ is the current point and $\mathbf{x}^{k+1}$ is the new point,

$$F(\mathbf{x}^{k+1}) < F(\mathbf{x}^k) \tag{3.3}$$

Such a direction is called a descent direction only when it satisfies the following requirement at any point,

$$\nabla F(\mathbf{x}^k) \,.\, \mathbf{p}^k < 0 \tag{3.4}$$

where $\nabla F(\mathbf{x}^k)$ is the gradient of the objective function containing the first order objective function variation with respect to the unknowns in **x**. There are several methods for finding the search direction, $\mathbf{p}^k$, but for this research, a model combined of Newton's method and steepest descent direction is implemented as a part of the optimization coding.

3.3.1 Newton's Method

Newton's method uses a quadratic approximation to a function at the current point. For each optimization iteration, a quadratic function fit is applied and its minimum is found. This process continues until the optimization procedure converges to an optimal solution. The original

form of Newton's method is derived by finding the vector $\mathbf{p}^k$ that generates the largest possible drop in objective function. Therefore, the new step is found as

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{p}^k \tag{3.5}$$

A Taylor series expansion of the objective function, $F(\mathbf{x})$, at the $(k+1)^{th}$ point provides

$$F(\mathbf{x}^{k+1}) = F(\mathbf{x}^k) + \nabla F(\mathbf{x}^k)^T(\mathbf{x}^{k+1} - \mathbf{x}^k) + \frac{1}{2}(\mathbf{x}^{k+1} - \mathbf{x}^k)^T H(\mathbf{x}^k)(\mathbf{x}^{k+1} - \mathbf{x}^k) + \text{H.O.T.} \tag{3.6}$$

where H.O.T. stands for the higher order terms, and $H(\mathbf{x}^k) = \nabla^2 F(\mathbf{x}^k)$ is the Hessian matrix of the objective function (the matrix of second partial derivatives with respect to $\mathbf{x}$ at $\mathbf{x}^k$).

As mentioned before, Newton's method is used to make the quadratic approximation of $F(\mathbf{x}^k)$ which means it approximates the function with the first three terms on the right hand side of Eq. (3.6) employing second-order information about $F(\mathbf{x}^k)$ obtained from the Hessian matrix. So, based on this matrix, the curvature of $F(\mathbf{x}^k)$ is taken into account in identifying a search direction.

The minimum of the quadratic approximation of $F(\mathbf{x}^k)$ in Eq. (3.6) is obtained by differentiating it with respect to the $\mathbf{x}^k$ and equating the resulting expression to zero which yields

$$\mathbf{x}^{k+1} - \mathbf{x}^k = -\left[H(\mathbf{x}^k)\right]^{-1} \nabla F(\mathbf{x}^k) \tag{3.7}$$

where $\left[H(\mathbf{x}^k)\right]^{-1}$ is the inverse of the Hessian matrix. By comparing Eqs. (3.5) and (3.7) the vector of search direction is found

$$\mathbf{p}^k = -\left[\mathbf{H}(\mathbf{x}^k)\right]^{-1} \nabla F(\mathbf{x}^k) \tag{3.8}$$

If $F(\mathbf{x}^k)$ is actually quadratic, Newton's method will find the minimum of $F(\mathbf{x}^k)$ in a single step. This is actually the condition considered in the current research.

Note that in order to evaluate $\mathbf{p}^k$ in Eq. (3.7), a matrix inversion is not necessary. Its precursor can be taken, and the following set of linear equations can be solved for $\mathbf{p}^k$,

$$H(\mathbf{x}^k)\mathbf{p}^k = -\nabla F(\mathbf{x}^k) \tag{3.9}$$

In the current study, LU decomposition is used to find the inverse of the Hessian matrix, since this algorithm is very efficient for solving multiple linear equations compared to other direct methods [66].

3.3.1.1 Benefits and Limitations of Newton's Method

Although real objective functions are more complex than a second-order function, most of them can be modelled correctly as quadratic when they are sufficiently close to the minimum point, consequently, within the region of the minimum, Newton's method usually brings rapid convergence, and is generally regarded as a highly efficient minimization algorithm for problems in which $F(\mathbf{x})$ is continuous and not noisy [11].

On the other hand, there is a known flaw in finding the minimum point using this method. Since the search direction in Newton's method is obtained from setting the gradient of the objective function equal to zero, the point found is a local extreme point which is not necessarily the minimum. If the Hessian of the matrix is not positive definite (matrix H is positive definite if $\mathbf{p}^T H(\mathbf{x}^k)\,\mathbf{p} > 0$ for all $\mathbf{p} \neq 0$), the extreme point could be a maximum point or a saddle point which, in this case, Newton's method will take you to a wrong direction resulting in the increase

of the value of the objective function of the new point compared to the current point. To resolve this problem, Newton's method needs to be modified.

3.3.2 Steepest Descent Direction

When Eq. (3.2) does not hold or the Hessian matrix is not positive definite, the step determined by Newton's method is an ascent step and should be replaced by an alternative method. Steepest descent is the method used for the current study under this condition. In this method, the search direction is defined as the negative of the gradient of the function (steepest descent)

$$\mathbf{p}^k = -\nabla F(\mathbf{x}^k) \tag{3.10}$$

The gradient shows the ascent direction and therefore the negative value of it gives the descent direction required for minimization. It is assumed that the value of $F(\mathbf{x}^k)$ continuously decreases.

3.3.3 Integrated method

Convergence speed and global convergence are two important characteristics of search algorithms. Sometimes these two characteristics conflict with each other, their effects should be carefully considered before picking an algorithm for a specific problem. For instance, the global convergence of the steepest descent method is ideal, but this algorithm is very slow. On the other hand, the original Newton iteration usually converges rapidly when it is close enough to the solution, but sometimes its steps may not even be descent directions especially when they are not close enough to the solution. Therefore, the challenge here is to create algorithms that

incorporate both these characteristics. Accordingly, for the current research when Newton's method moves in the wrong direction, the steepest descent is used. This condition could happen at the early search steps as the steps are not close enough to the solution.

3.3.3.1 Negative Points of the Integrated Method

If the objective function were known, the values of the first and second order derivatives in Eqs. (3.8) and (3.10), for calculating $\mathbf{p}^k$, could simply be solved analytically. However, since the function for the current research is unknown, the derivatives must be solved numerically through finite differencing. In order to make a finite difference calculation, two or three function evaluations are required for each variable for every derivative(s) at each optimization step. Since two design variables are studied in the current research and the objective function calculation is expensive for the studied cases, this method can be very time consuming. Therefore, employing a method which is more time efficient with a higher robustness is necessary for the current study. This is the reason that a new approach called Response Surface Methodology (RSM) is used here, however, as a part of the RSM model, the integrated method is also used only when it is required to find a minimum of an analytical function.

3.4 Response Surface Methodology

Response surface methodology (RSM) [1] is a statistical-mathematical technique used for finding the optimum of expensive, noisy and complex objective functions. RSM is applied mostly in industrial problems where there are several design variables influencing a certain design quality (objective function).

The original RSM was developed to optimize physical experiments in a stochastic way. In this method, instead of minimizing $F(\mathbf{x})$ directly, RSM works by minimizing a sequence of second-order approximating functions, $\tilde{F}(\mathbf{x})$, fit to objective functions sampled over a subspace, Ω^k, of the feasible region that surrounds the current set of design parameters, $\mathbf{x}^k$. In simpler words, the solution starts with selecting a design space based on the design parameters, $\mathbf{x}^k$. From this design space, RSM defines a number of sample points in the search space (model region) in order to characterize the sensitivity of the objective functions to the design parameters. Then a polynomial regression function is fit to the response function value corresponding to each of the sample points. These regressions produce a polynomial surface which represents the objective function behaviour inside the model region. Given that the regression functions are simple low-order polynomials, they are solved very quickly using the integrated method explained in Section 3.3.3. Since the surface function approximates the real objective function, the minimum found won't necessarily be the minimum of the real function, therefore based on this point a new model region is defined and the procedure is repeated until the results are satisfactory within a reasonable tolerance (convergence is defined as the norm of the difference between the new point and the current point drops below an empirical threshold value). The RSM algorithm used in the current research is implemented from the previous work presented by Horsman [67]. In the present study, the CFD solvers explained in Chapter 2 are connected to this RSM algorithm with an interface module written in C++ (see Appendix B).

There are several advantages to this approach. One of the major benefits of this technique is that it reduces the number of simulations run with the CFD solver to a reasonable number and this significantly decreases the computation time. The other benefit is that since the response

surface is calculated as an analytical function, the gradient and the Hessian matrices can be found at once which avoids time-consuming numerical computation.

The following subsections describe the specific type of RSM method used in this research where selecting the model region, fitting, creating the surface from the regressors, and minimizing the objective function are explained in detail. The explanations are for an objective function with two design variables, however this method works for more variables as well.

3.4.1 Model Region Design

There are many practical situations where the designers should select ranges for the design variables. This is the region of interest and is called the feasible region where in a two-dimensional (two design variables) design optimization problem it can be shown as a rectangular region. The model region, which contains the sample points, should obviously be defined inside this feasible region. Several practical arrangements are available in the literature) [1] for selecting the sample points within the model region, the face-centered central composite design (FCCCD) method is the arrangement used in the current study [68]. Theoretically, in order to model a 2D optimization problem, having the information of six different sample points is enough for generating a quadratic function. However, in order to find a surface function which can model the real function better, more sample points are taken inside the model region. For the current case, three more points giving the total of nine interpolating points are picked. In addition to transferring more information from $F(\mathbf{x})$ to $\tilde{F}(\mathbf{x})$, implementing more sample points also helps to reduce the possible noises in the function. As shown in Figure 3.1, the current point, $\mathbf{x}^k$, is located exactly in the centre of the model region. The concept and nature of the problem are the factors should be considered by the designer in order to specify the initial size of this region.

The designer should always keep in mind that the model region should be defined in a way that the procedure results in a stable and time-efficient manner.

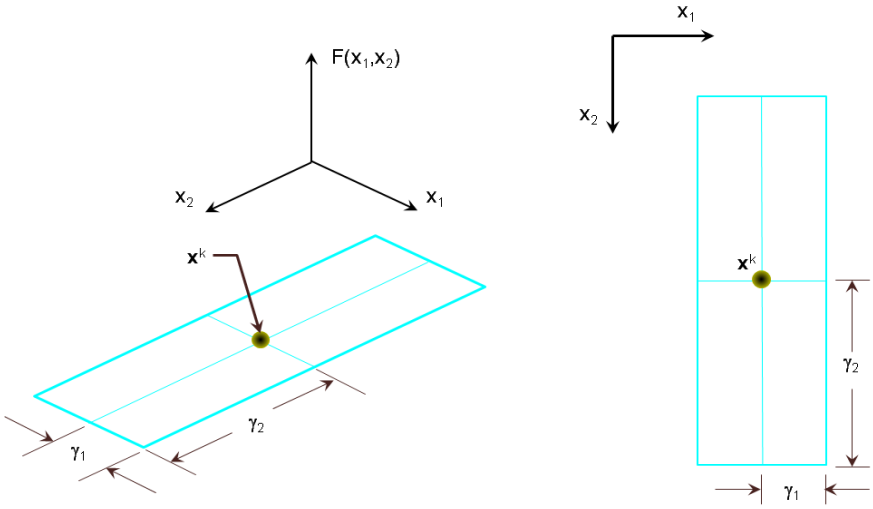


Figure 3.1: Model region definition

The rest of the points are placed uniformly on the boundary of the region. The right side of Figure 3.2 shows how the sample points are spread out throughout the domain of the model region. As can be seen, the rectangular model region is defined with the dimension of $2\gamma_1 \times 2\gamma_2$ which shows that the sample points are arranged equally at a distance of γ_1 and γ_2 along the $\mathbf{x}_1$ and $\mathbf{x}_2$ directions, respectively.

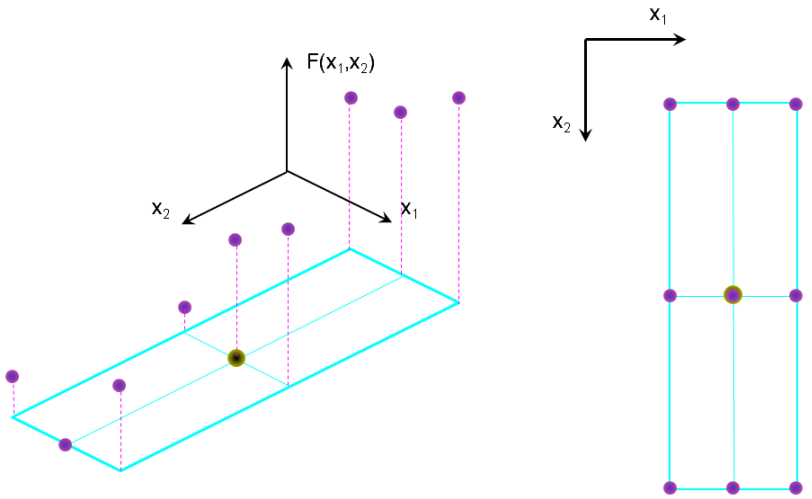


Figure 3.2: Sample points arrangement and the corresponding $F(x)$

3.4.2 Generating the Response Surface Function

When the sample points are selected, the CFD solver is run for each of the samples meaning that nine CFD simulations are performed per optimization iteration. The function value for an arbitrary model region is shown on the left side of Figure 3.2. One of the challenges of RSM is to approximate the unknown function based on the known points. This is usually done with a low-order polynomial over a well-designed model region. For many types of objective functions, either a first-order or a second-order polynomial is employed [1]. The first-order approximation is often reasonable when the model region is small and/or where the true objective function, $F(\mathbf{x})$, has a little curvature. This model is also called a "main effects model" since it only shows the main effects of the independent variables. However, in most cases, the curvature of $F(\mathbf{x})$ is strong enough make the first-order assumption inaccurate. Under this condition a second-order model function is required. In general, the second-order regression model is shown,

$$\tilde{F}(\mathbf{x},\beta) = \beta_0 + \sum_{j=1}^{k} \beta_j \mathbf{x}_j + \sum_{j=1}^{k} \beta_{jj} \mathbf{x}_j^2 + \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \beta_{ij} \mathbf{x}_i \mathbf{x}_j \tag{3.11}$$

where the β's are the set of regression coefficients. Since the current study applies to two independent variables, Eq. (3.11) can be written as

$$\tilde{F}(\mathbf{x},\beta) = \beta_0 + \beta_1 \mathbf{x}_1 + \beta_2 \mathbf{x}_2 + \beta_{11} \mathbf{x}_1^2 + \beta_{22} \mathbf{x}_2^2 + \beta_{12} \mathbf{x}_1 \mathbf{x}_2 \tag{3.12}$$

The corresponding coefficients for Eq. (3.12) are calculated by the method of "least squares" where the best fit is found by minimizing the sum of the squares of the deviation between the real function and the response surface,

$$\beta^* = \operatorname{argmin}_\beta \left\| F(\mathbf{x}) - \tilde{F}(\mathbf{x},\beta) \right\|_2^2 \tag{3.13}$$

The second-order model is widely used in RSM since it is very flexible. This model, in fact, is able to engage a wide range of different functional forms, thus it often works well as an approximator to the real function. In addition, it is easy to estimate the values of β's for the second-order model using least squares. Implementation of higher-order polynomials would require more sample points: for example, fifteen points are required to provide a third-order fit and this would significantly increase the computational time yet it may not deliver more precise results. Figure 3.3 shows a typical second-order least squares regression function fit to nine sample points.

Since for this regression function the gradient and Hessian matrices can be calculated analytically, the minimum point can also be found analytically. This is a considerable advantage over the integrated method which takes a relatively large time to find the minimum yet this method deals with possible numerical errors and instability issues (see section 3.3.3.1).

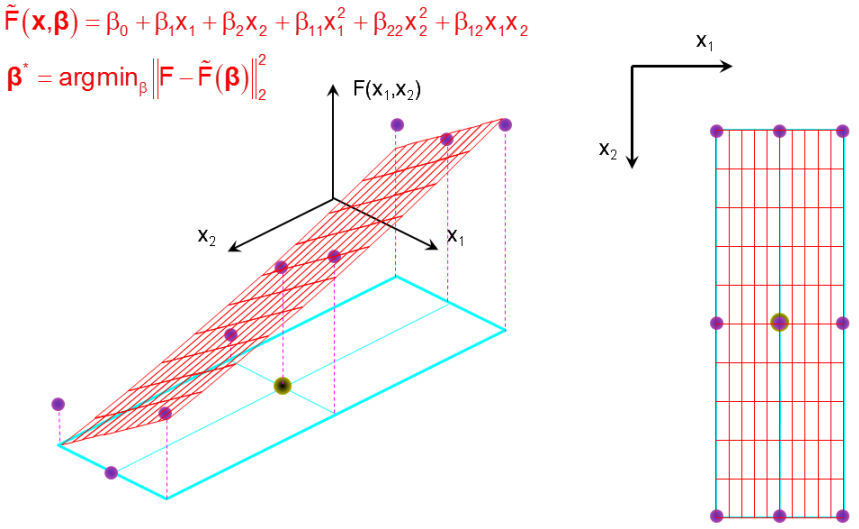


Figure 3.3: Fitting response surface using second-order least squares regression

3.4.3 Calculation of Search Direction

After finding the second-order surface function, it is easy to pinpoint the minimum of the surface function using the integrated method comprised of Newton's method and the steepest descent direction (see Section 3.3.3). Since the surface is a quadratic function, Newton's method finds the stationary point in a single step, but if this point is not a local minimum this method is replaced with the steepest descent. However, as the designated surface function is only valid inside the model region, the minimum point found outside the boundaries of the model region cannot be accepted. This means that the model region might constrain the answers given by the integrated method; therefore it is necessary to calculate the step length in Eq. (3.2). According to this methodology, there are three possible conditions to calculate α^k: (*i*) if Newton's method calculates a descent direction and $\mathbf{x}^k+\mathbf{p}^k$ lies within the model region, then $\alpha^k=1$; (*ii*) if Newton's method calculates a descent direction and $\mathbf{x}^k+\mathbf{p}^k$ lies outside the model region, then α^k is chosen to project $\mathbf{x}^k+\alpha^k\mathbf{p}^k$ onto Ω^k; and (*iii*) if the stationary point found by Eq. (3.7) is an ascent direction then $\mathbf{p}^k$ follows the steepest descent direction and α^k is chosen to project $\mathbf{x}^k+\alpha^k\mathbf{p}^k$ onto Ω^k.

Figure 3.4 shows how the search direction is found for an arbitrary surface function in a model region. For the surface function shown in this figure, Newton's method finds the minimum outside the model region, thus the step length, α^k, is determined by projecting the direction of the Newton's step to the edge of the model region where in this case condition (*i*) holds.

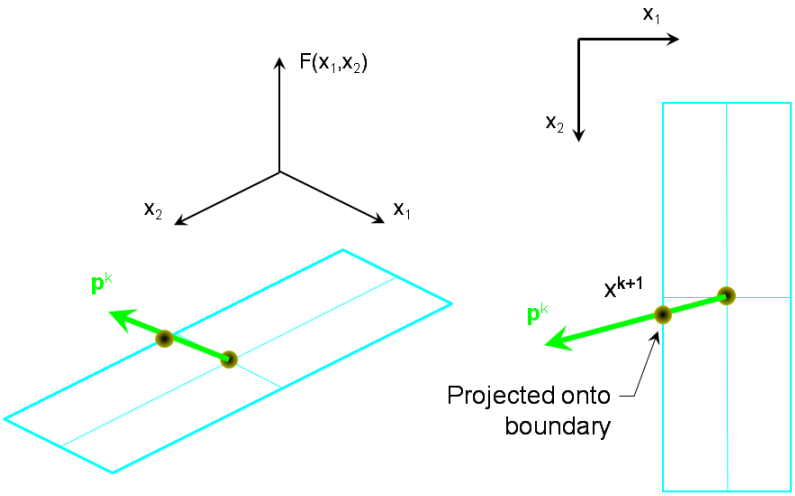


Figure 3.4: Calculating search direction and step length

3.4.4 Next Optimization Iteration

When the value of the step length, α^k, is found, Eq. (3.2) is used to determine the new point, $\mathbf{x}^{k+1}$, and the value of the corresponding real objective function, $F(\mathbf{x}^k)$. Optimization is terminated once the norm of the difference between the new point and the current point, $\left\|\mathbf{x}^{k+1}-\mathbf{x}^k\right\|$, drops below an empirical threshold value; otherwise the program generates a new model region, centered on $\mathbf{x}^{k+1}$, as shown in Figure 3.5, and proceeds until convergence is obtained.

Depending on the location of the new point, there are two possible conditions to design the new model region: (*i*) if the new point is located on the boundary of the model region, then the size of the model region remains unchanged and it is shifted to be centered on the new point; and (*ii*) if the new point is inside the model region, the model region is reduced in size by a factor of two in each dimension and then centered on the new point. The reason of shrinking the region is that there is a good possibility that the optimum point, $\mathbf{x}^*$, lies within the existing

region; so, in this case, the model region is shrunk in order to focus more on the area of interest to make sure it is on the right track to find the optimum point.

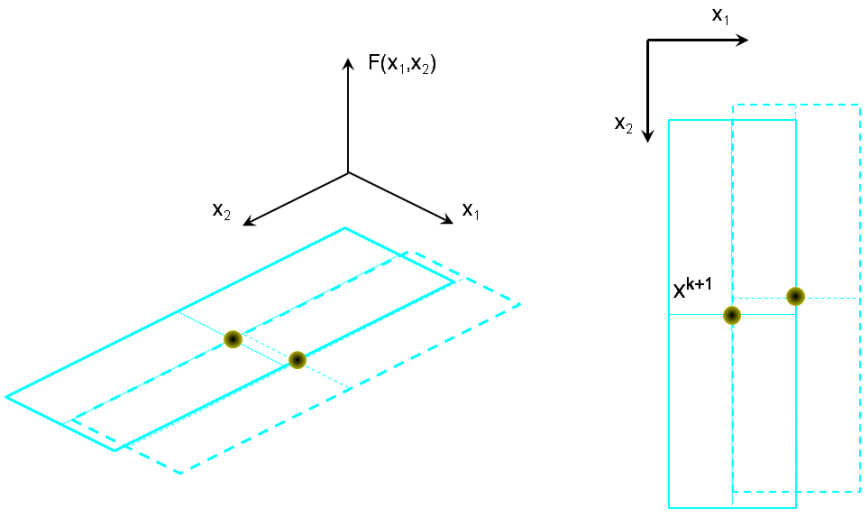


Figure 3.5: Updating design parameters

3.4.5 Effect of Constraints on RSM

Since most practical optimization problems are constrained, meaning that they are reliable only within a feasible region, some modifications are required to the RSM algorithm. Corrections on the size and location of the model region, and, in some cases, a modified search along the feasible boundary are the major required adaptations caused by the constrained RSM. These modifications are specifically devised by Horsman [67] and are presented in the next two subsections.

3.4.5.1 Resizing and Relocation of Model Region

There are two conditions that might require the model region be resized or relocated during the constrained optimization procedure:

i) If the distance of $\mathbf{x}^k$ to the boundary of the constrained region is shorter than γ_1 or γ_2, which means that a part of the standard model region will go outside the feasible region, the sample points which are stepped out of the constrained region are shifted so that they are located on the edge of the region, as shown in Figure 3.6. As a result, by shrinking the model region, all function evaluations are performed inside the feasible region.

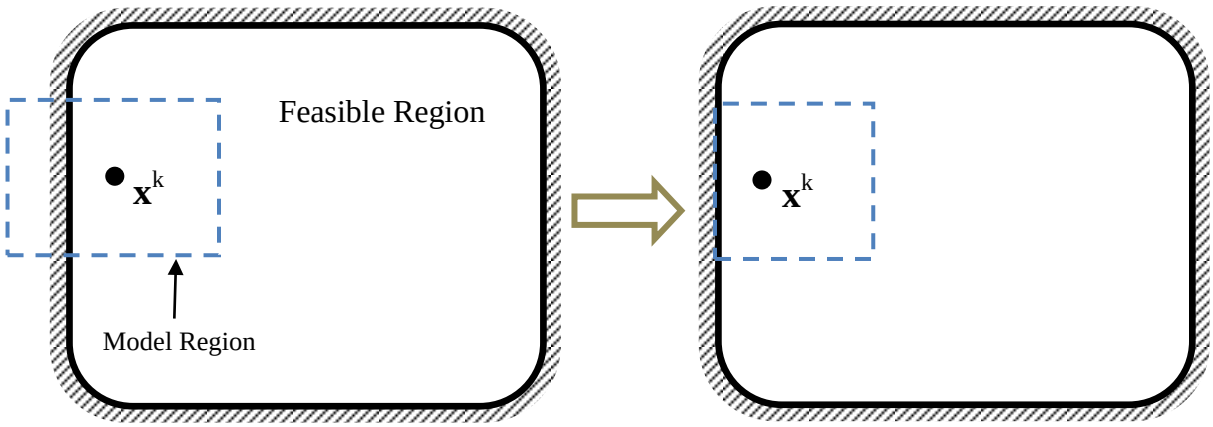


Figure 3.6: Correction of model region intersecting a constraint

ii) If the search calculation appoints $\mathbf{x}^k$ to be exactly on the boundary of the feasible region, the model region should shrink. Under this condition, where the edge of the model region and the constrained region are aligned, the model region is reduced in size in all the directions by the factor of two, as shown in Figure 3.7. In fact, having the current point on the constraint means that the minimum point lies outside the feasible region, but since the real objective function is only approximated by a low-order regression function, the model region shrinks around the point to improve the accuracy of the model function. By focusing more on that area through shrinking the model region and then calculating a new search direction and step length, it can be determined whether the minimum is outside the feasible region.

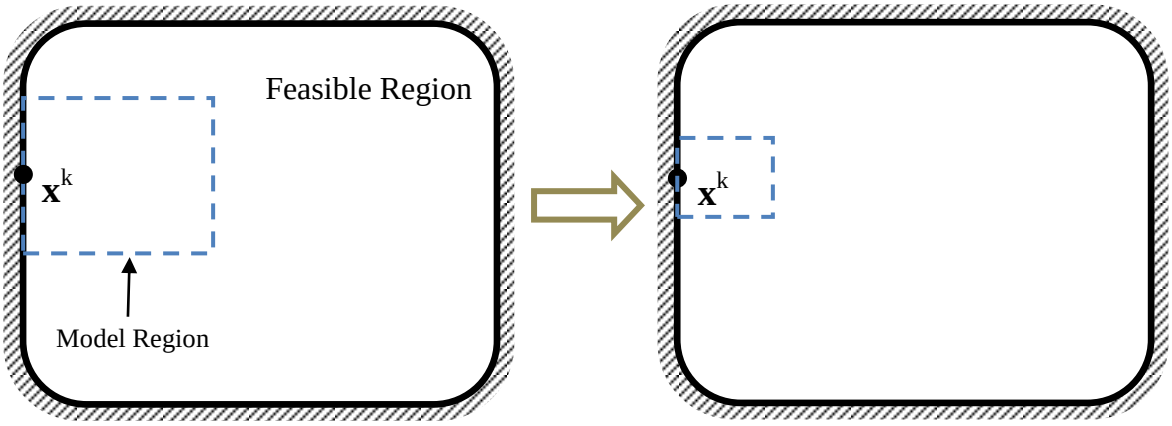


Figure 3.7: Change to model region when $\mathbf{x}^k$ is on a constraint

3.4.5.2 Finding the Minimum on a Constraint

If condition (*ii*) explained in Section 3.4.5.1 occurs, a method called the Generalized Reduced Gradient is used in order to reduce the dimensionality of the problem by one through treating the inequality constraint of the boundary as an equality constraint [69]. By doing so, further improvements can be found by searching along the boundary of the feasible region to find the minimum, and after that the algorithm switches back to its original dimensionality, shrinks the model region and follows a new search step. This optimization process along the boundary continues until the convergence condition is met or the search direction steps back into the feasible region [70].

Chapter 4
Results and Discussion

4.1 Introduction

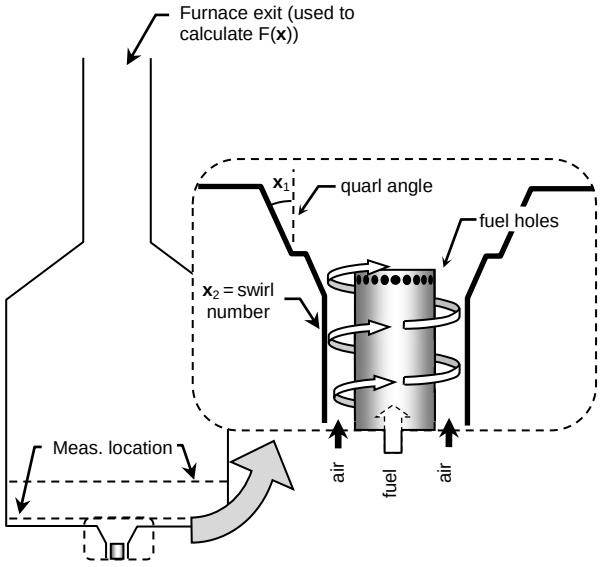
The methods explained in Chapters 2 and 3 were implemented to simulate two different combustion chambers. The results are explained in the current chapter. First, the CFD simulation of the BERL burner is studied and validated with the experimental data. Then, the RSM two-dimensional optimization algorithm is applied with the goal of enhancing the fuel utilization. The technique's capability to find the optimal solution is studied further in this chapter. In a separate study, the two-phase combustion flow in the GenTex diesel oil process heater was studied numerically and experimentally, and the effects of physical phenomena on the performance of the process heater are discussed.

4.2 Design Optimization of the BERL Furnace

The objective of this research is to demonstrate how the model-based optimization methodology can be applied to design industrial combustion equipment. We have chosen to optimize the design of the 300 kW BERL furnace [3] shown in Figure 4.1, since the burner is well-characterized and has been used to validate other CFD studies [71, 72]. The furnace is vertically fired and the exhaust gas exits the conical hood through a cylindrical duct. Fuel is injected through 24 circumferential holes, and combustion air is swirled through swirl blocks and

blown through an annular zone. Neither flue gas recirculation nor natural gas staging are used in this burner. Temperature distributions on the walls are given in Ref. [3]. The burner inlet conditions are summarized in Table 4.1.

The goal of the current study is to optimize the fuel utilization, which is equivalent to maximizing the mass flux of CO_2 along the exit plane of the furnace. Since by convention the objective function is defined so its minimum corresponds to the optimal design, we define the objective function, $F(\mathbf{x})$,



$$F(\mathbf{x}) = -Y_{CO_2}(\mathbf{x})\dot{m}_{total} \tag{4.1}$$

Figure 4.1: BERL burner geometry and design

The design parameters include the quarl angle, $\mathbf{x}_1$, which is between 0° and 30°, and the swirl number, $\mathbf{x}_2$, which is allowed to vary between 0.1 and 0.8. In practice the swirl number can be changed by altering the swirl vane angle.

Table 4.1: Burner inlet condition

Problem parameter	Inlet swirling air	Inlet fuel
Mass flow rate ($\boldsymbol{kg/hr}$)	436.2	22.7
Temperature ($°\boldsymbol{K}$)	312.15	308.15
Swirl Number	0.56	0
Turbulent intensity	17%	5%
Turbulent kinetic energy ($\boldsymbol{J/kg}$)	61.29	94.21
Turbulent dissipation rate ($\boldsymbol{m^2/s^3}$)	1.479×10^5	2.385×10^6

4.2.1 CFD Validation

Before implementing the optimization algorithm, it is necessary to accurately model the combustion inside the furnace. The physical domain of the furnace was treated as axisymmetric and discretized into 37×195 elements. (A grid independence study was carried out to ensure this level of refinement was sufficient.) The fuel injection holes were modelled as a single annular slot in a manner that preserves the mass flow rate and momentum of methane entering the furnace.

The CFD simulation is validated against the experimental data reported in Ref. [3]. A selection of results obtained at 0.027 m downstream from the burner throat is shown in Figure 4.2. The predicted axial velocity, temperature, and CO_2 mole fraction show reasonable agreement with the experiment implying accurate predictions of the flow and the concentration fields. Negative values for axial velocity indicate that there is an internal recirculation zone in front of the fuel gun due to the sudden expansion and swirl velocity of air exiting the quarl. This zone entrains most of the fuel exiting the gun, resulting in a very high reaction rate; this produces a higher temperature and higher concentration of product gases compared to the other parts of the furnace, as reflected by the CO_2 concentration and temperature profiles in Figure 4.2.

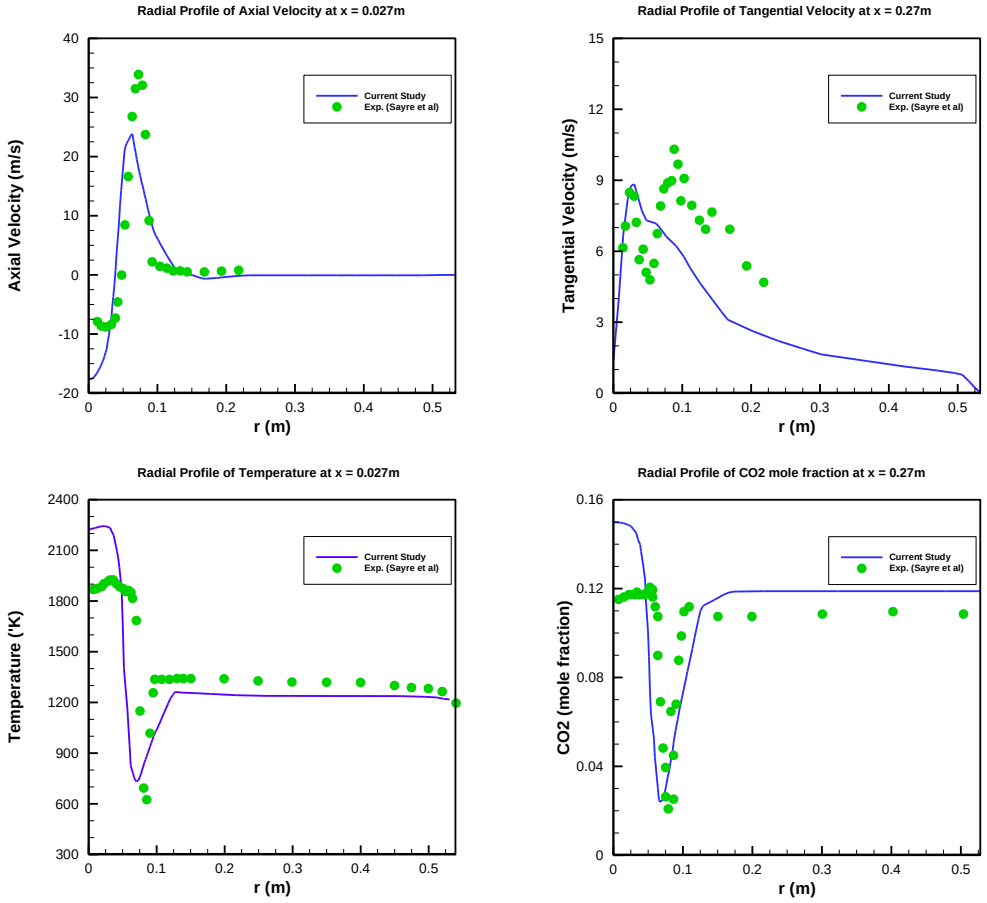


Figure 4.2: Radial profiles of velocity, temperature, and CO2 mole fraction at 0.027m downstream of the burner throat

Similar results were obtained for 0.343 m downstream form the burner exit as shown in Figure 4.3. Figures 4.2 and 4.3 indicate that the largest discrepancy between the model and experimental results is in the reaction zone. This is likely due to the two-step reduced mechanism and PaSR model, which overestimate the reaction rates. The other sources of error are the underprediction in the turbulent viscosity of the RNG k-ε, the approximation of constant radiative properties in the radiation model, and the modeling of the fuel inlet holes as an annular slot.

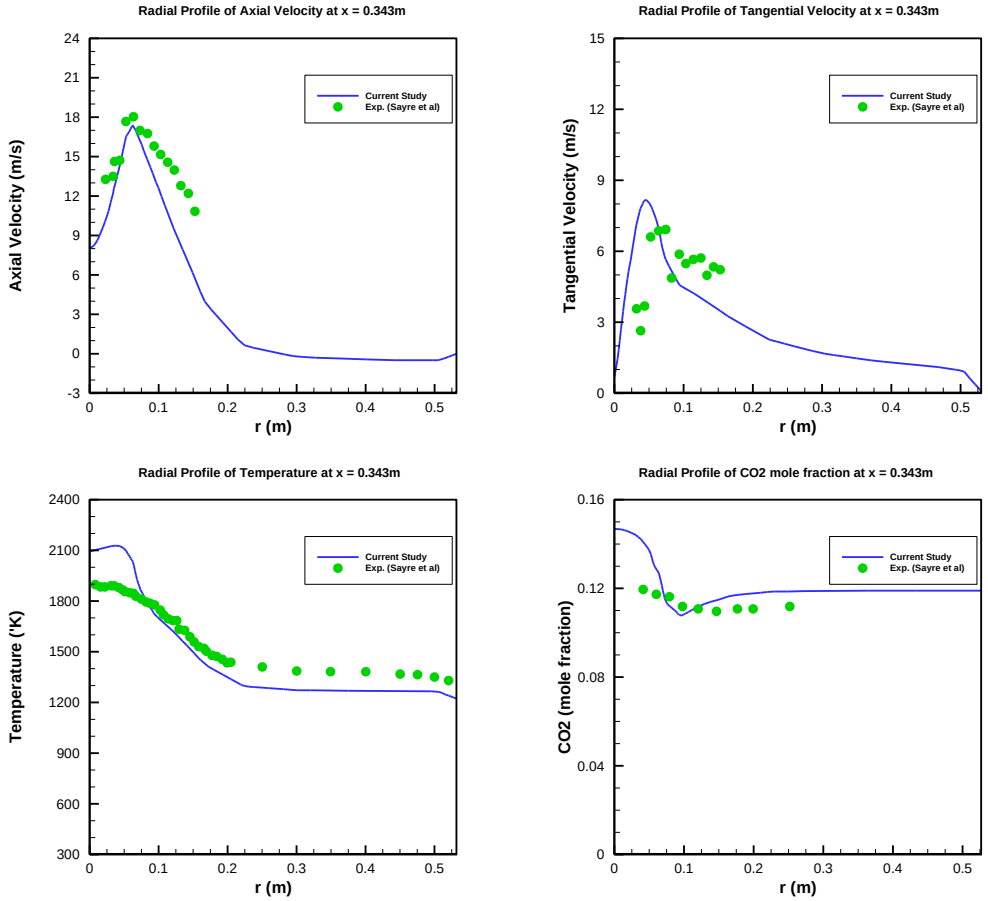


Figure 4.3: Radial profiles of velocity, temperature, and CO2 mole fraction at 0.343m downstream of the burner throat

4.2.2 Optimization Implementation

Once the CFD solver satisfactorily predicts the reacting flow inside the furnace, it connects to the RSM algorithm with the interface module written in C++ (see Appendix B). The initial design configuration is chosen to be $\mathbf{x}^0 = [20°, 0.6727]^T$. Figure 4.4 shows the steps followed by the RSM algorithm to maximize the CO_2 mass generation. Sampled points used to construct the response surface are shown in circles while the larger circles represent the optimum point found at the corresponding iteration. Note that the contours shown in this figure correspond to the response surface, and not the true objective function. In the first iteration, Eq. (3.2)

produces a new set of points, $\mathbf{x}^1$, located on the model region boundary showing that the extreme point is outside the model region and since the calculated extreme point gives an ascent direction, the model region keeps its size and takes the steepest descent direction to the point (condition (*iii*) from Section 3.4.3).

Since the upper points of the next model region would lie outside of the feasible region, the model region is shrunk so that the new model region is bounded by the upper bound constraint of x_2. The second optimization iteration follows condition (*iii*) resulting in point on the boundary. This means that the minimum point is located outside the feasible region, but since the real objective function is only approximated by a low-order regression function, the model region shrinks around the point to improve the accuracy of the model function. The next point falls in the feasible region following condition (*i*) (explained in Section 3.4.3) and the model region steps back to the inside of the feasible region. For the next iterations, condition (*i*) takes place which makes the model shrink around the point and recalculate to make sure the real optimum point is inside the model region.

At the final iteration, a small region is left which determines the value of the objective function slightly lower than the earlier point, thus the optimization is terminated offering the previous point as the optimum solution; $\mathbf{x}^* = [23.2149, 0.5098]^T$ with the CO_2 mass generation of 52.6976 kg/hr. The value of the objective function for the original case was calculated as 51.3128 kg/hr which reveals that the original burner could be modified to improve its performance.

One of the main impediments to model-based optimization of industrial combustion equipment is the overall calculation time. Since each optimization step requires multiple

evaluations of $F(\mathbf{x})$ to calculate $\mathbf{p}^k$, and sometimes α^k, many CFD simulations are executed during optimization, a number that increases geometrically with the number of design parameters. Furthermore, due to its long running time the procedure must be carried out automatically without user interaction; this is especially challenging since, due to inherent numerical stiffness of the governing equations, CFD simulations of combustion are especially prone to divergence.

To overcome these challenges, at each iteration the field variables (velocity, temperature, species, etc.) are initialized using the results of the previous iteration, mapped to the adjusted geometry. This dramatically reduces the computational time needed to converge the solutions, since the CFD algorithm starts with small residuals due only to the (relatively small) change in furnace geometry across iterations. In addition, since each CFD simulation is initialized from a physically realistic solution, the chance of divergence becomes much lower; this allows the user to employ higher values of under-relaxation factors, resulting in even faster convergence. By doing so the number of the iterations needed to converge the governing equations for each evaluation of $F(\mathbf{x})$ is generally reduced to below 1000 (except for the first simulation), resulting in a total calculation time for this problem of about 10 hours running on a single core computer.

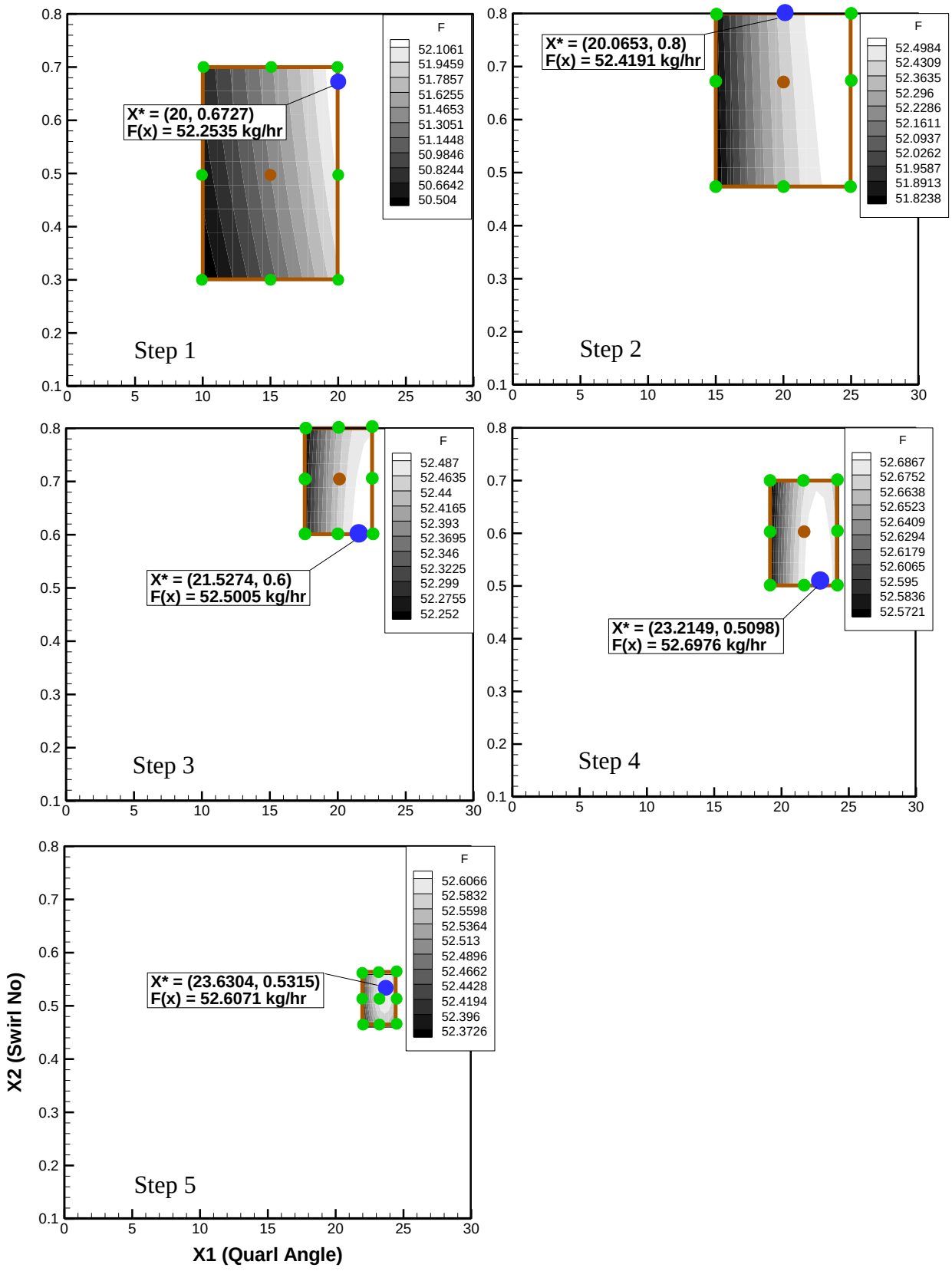


Figure 4.4: Optimization steps followed by the RSM algorithm

4.3 Measurements and Modeling of Diesel Combustion in a Cylindrical Industrial Process Heater

In this section the combustion of diesel spray in the GenTex 50M process heater is studied. The process heater belongs to GenTex Oilfield Manufacturing Inc. located in Red Deer, Alberta. Since the numerical results of liquid fuel combustion are very sensitive to the fuel spray distribution coming out of the injector [73], collecting sufficient information on the behaviour of the spray was necessary. Thus, the spray distribution of the operating injector was studied numerically and experimentally at National Research Council of Canada (NRC). Next, the temperature distribution along the centerline of the process heater was measured in order to validate the simulation of the combustion field. Numerical simulations for both the cold spray and the diesel combustion were then carried out and results agreed with the experimental data.

4.3.1 Cold Spray Measurements and Injector Characteristics

The diesel fuel is introduced into the burner through a Y-type multi-hole injector. As shown in Figure 4.5, the injector contains 6 orifices which are spread circumferentially. The atomizing air is blown from the middle hole and entrains the fuel which is being injected through the six circumferentially located fuel channels surrounding it.

The measurements were performed only for one orifice of the injector while the other orifices were sealed because observations showed each of the orifices produce an almost identical solid-cone spray. Accordingly, instead of making the measurements for the entire area of the spray which requires a large testing area and takes an extensive time for testing, only the flow from one orifice was studied.

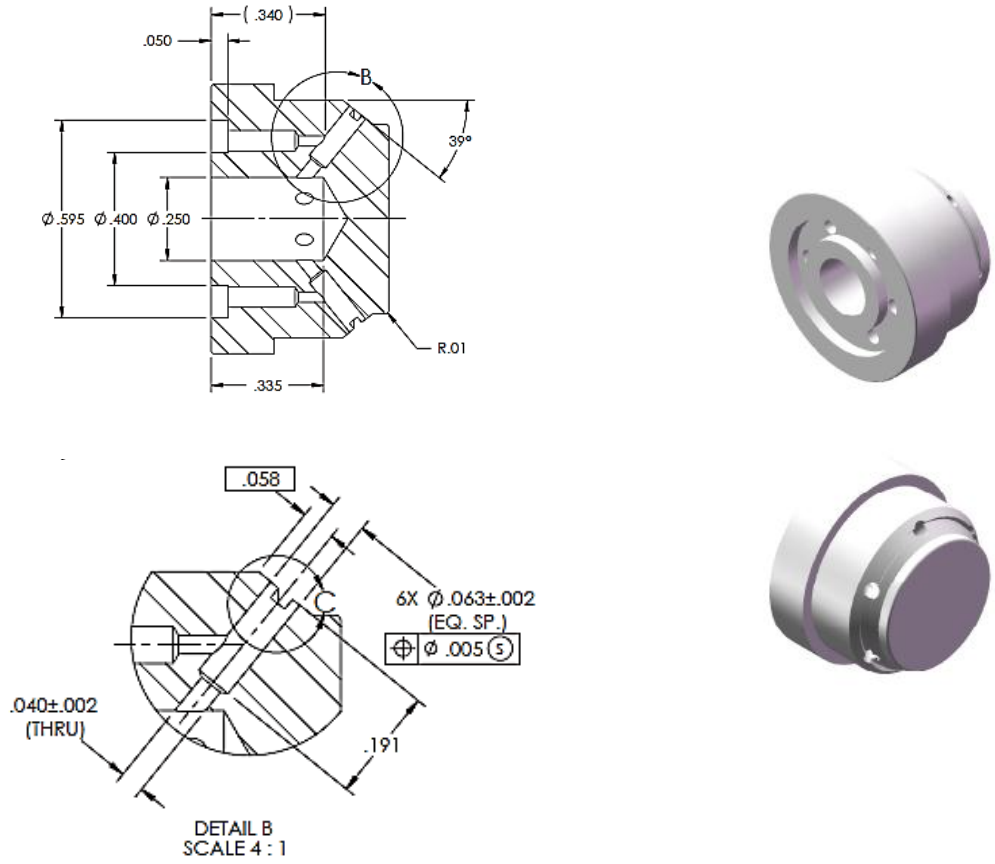


Figure 4.5: The multi-hole injector (units are in inches) [74]

The technique applied to analyze the droplet sizes is Laser Diffraction Spectrometry (LDS). The instrument used in the current study is a Sympatec HELOS/KR, which combines a laser-based optical transmitter, an optical receiver, an electronic signal processor and software for capturing and analyzing data. This device provides a laser beam with thickness of 2 inches and measures droplets from 0.1 µm to 8750 µm in diameter. The diffraction of the laser light results from the interaction of the light with the droplets and this interaction can be described mathematically using Mie theory [75]. The measured diffraction signals are due to contributions from all droplet size classes, so the droplet size distribution must be involved using inverse

analysis. The measurement precision is typically ±1% deviation with respect to the standard metre [76].

Due to the NRC safety rules, instead of a real fuel which is flammable, water is studied as the working fluid. Water is discharged through a pump rated at 80 psi with the volumetric flow rate of 41 gallons per hour. As it passes through a nozzle, it produces a distinctive solid-cone spray pattern. As soon as the droplets confront the laser beam of the LDS system, the equipment measures the size distribution of the droplets as well as the Sauter mean diameter (D_{32}) and the arithmetic mean diameter (D_{10}). The Sauter mean diameter (SMD) is defined as the diameter of a sphere which has the same volume/surface area ratio of the real droplet. SMD is typically defined as

$$\mathrm{SMD} = d_v^3 / d_s^2 \tag{4.2}$$

where

$$d_s = \sqrt{A_p / \pi} \tag{4.3}$$

$$d_v = \left(6V_p / \pi\right)^{1/3} \tag{4.4}$$

where A_p and V_p are the surface area and volume of the particle, respectively. d_s and d_v are called surface diameter and volume diameter and are usually measured directly by other means without the knowledge of A_p or V_p. The Sauter mean diameter for a group of n particles, D_{32}, is calculated as

$$D_{32} = \frac{\sum_{i=1}^{n} d_{v,i}^3}{\sum_{i=1}^{n} d_{s,i}^2} \tag{4.5}$$

The arithmetic mean diameter, D_{10}, is the average of the sizes of each of the droplets,

$$D_{10} = \frac{1}{n}\sum_{i=1}^{n} d_i \tag{4.6}$$

The nozzle was set up in a vertical position and the measurements were conducted at five axial locations, beginning at 101.6 mm and continuing to 228.6, 304.8, 457.2, and 635.0 mm downstream of the spray nozzle. The measuring distance and the position of the laser beam and the spray cone are shown in Figure 4.6. These distances correspond to 63.5, 142.8, 190.5, 285.7 and 396.8 in terms of distances non-dimensionalized by the nominal nozzle diameter which is 1.6 mm [77].

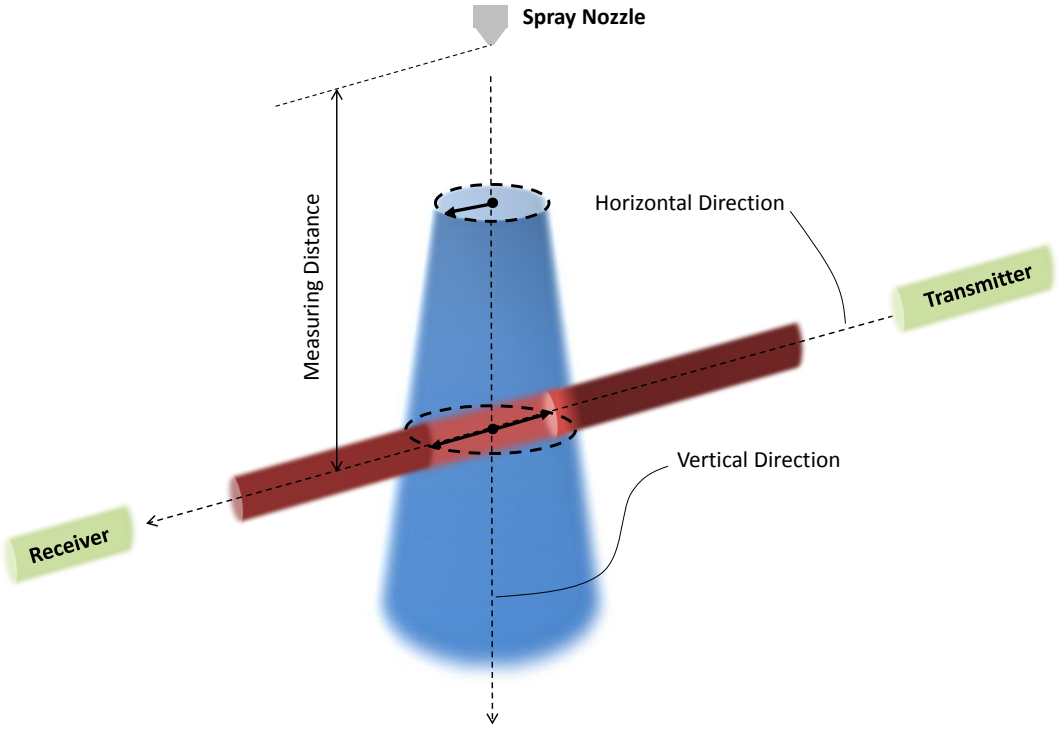


Figure 4.6: The spray cone alignment versus the LDS laser beam

4.3.2 GenTex Process Heater and Temperature Measurements

The 50M GenTex process heater is 0.928 m in outside diameter and 2.040 m in length of the casing with a total thermal input of 5 Mbtu/hr. It consists of three layers of water coils as are shown in Figure 4.7. The combustion gases pass through the innermost, intermediate and

outermost passages adjacent to water coils and heats water inside the coils. Figure 4.8 shows the burner used in the heater being studied. The burner consists of the multi-hole injector located at the center, a tangential vane swirler with a vane angle of 22º, and a firepot made from a perforated drum. The fuel is pumped to the burner and a blower feeds the air to a swirler and the perforated base in order to create recirculation zones required for stabilizing the flame. Ignition occurs a short distance from the injector. The hot combustion gases heat the water by convection and radiation with the water coils, and finally leaves the process heater through a stack. The operating conditions of the process heater are given in Table 4.2.

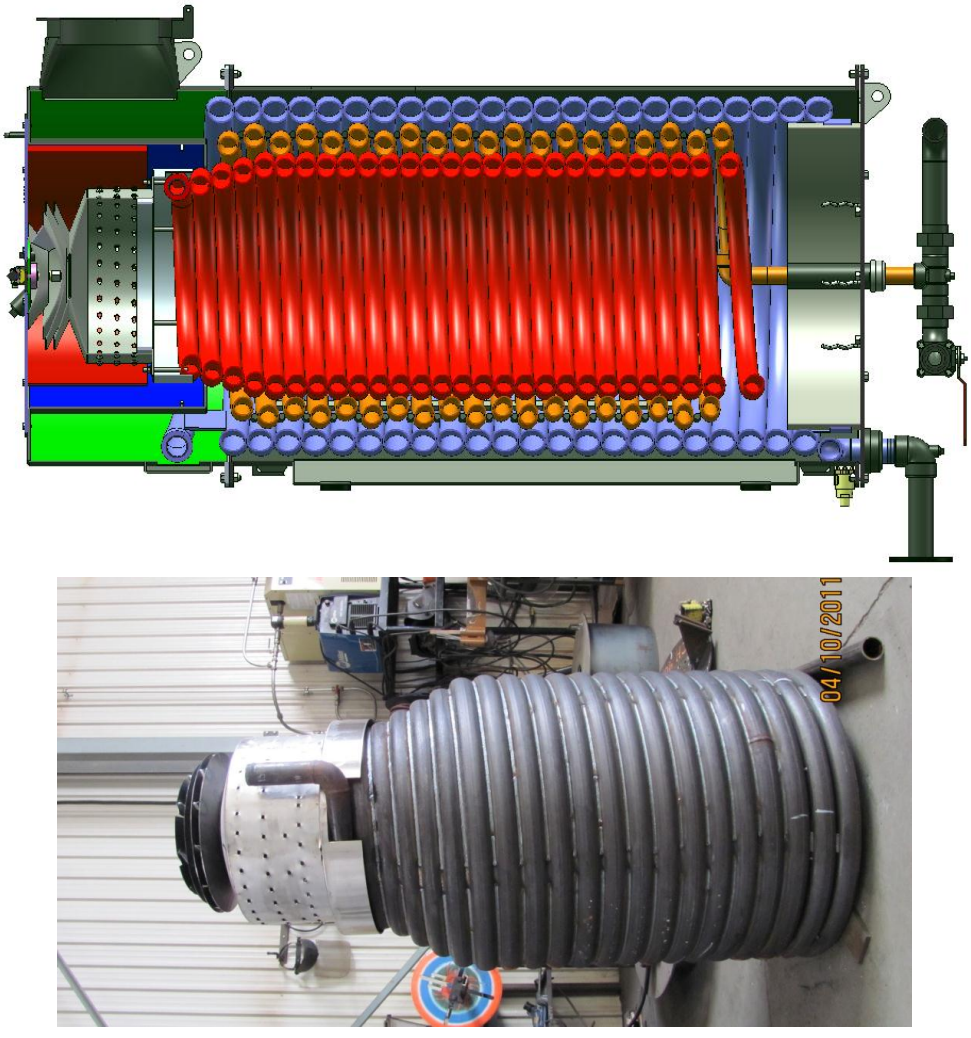


Figure 4.7: 50M heater sketch (top figure) [78] and the innermost layer (bottom picture)

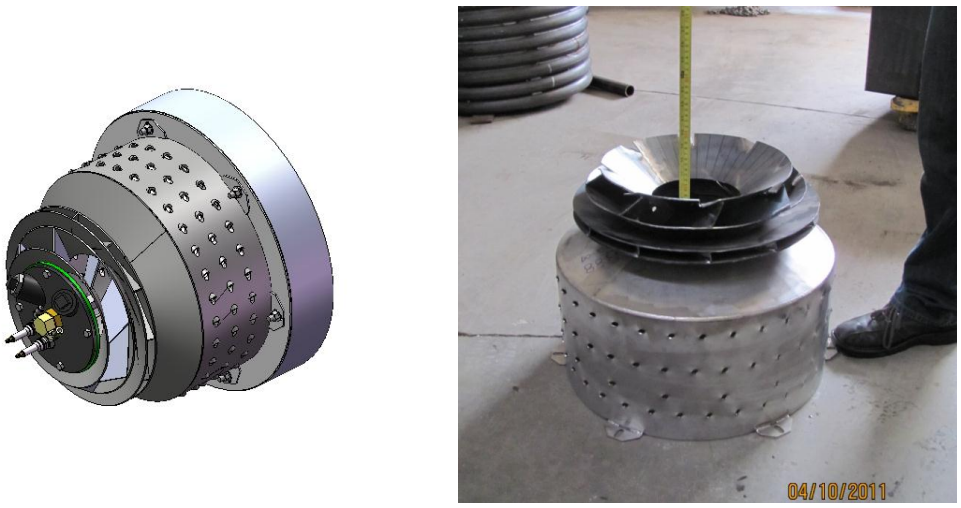


Figure 4.8: Burner configuration [78]

Table 4.2: GenTex burner inlet conditions	
Air/fuel ratio	23.535
Volumetric flow rate of air (m^3/sec)	1.0146
Swirl number	0.85
Vane angle	22º
Temperature of fuel and air (ºC)	25

The temperature of the combustion gases inside the innermost zone was measured using a K-type thermocouple (see Figure 4.9). The specific thermocouple has an aluminum head and a grounded junction to protect the effect of radiation heat transfer on the metal wires. The protection tube with a 0.5 inch diameter covers the metal wires which are configured in a four-bore rounded 3 inch long insulator and are parallel-welded.

Figure 4.9: K-type thermocouple probe inserted from a hole at the end-side of the heater

4.3.3 Validation of Spray Injection Tests at NRC

There are many known droplet size distribution functions such as normal, log-normal, root-normal, Nukiyama-Tanasawa, and Rosin-Rammler. For the majority of the simulations, initial droplet sizes at the injector location were chosen from prescribed probability density function using the Rosin-Rammler distribution, *f(D)*,

$$f(D) = \frac{nD^{n-1}}{\tilde{D}^n} \exp\left[-\left(\frac{D}{\tilde{D}}\right)^n\right] \tag{4.7}$$

where *n* is the spread parameter, and $\tilde{D}$ is the characteristic droplet size. The Rosin-Rammler cumulative distribution function, *CDF*, is then

$$CDF = \int_0^D f(D)dD = 1 - \exp\left[-\left(\frac{D}{\tilde{D}}\right)^n\right] \tag{4.8}$$

However, it should be noted that the parcel size distribution is different from the particle size distribution described by Eq. (4.8). Indeed, there is a factor of D^3 difference between the size distribution PDF of individual particles and modeled parcels of particles; $f(D)_{parcel} = D^3 f(D)_{particle}$. The reason is that the submodel parameter, PPP (number of particles per parcel), is based on a fixed mass per parcel which weights the droplet distribution by a factor proportional to $1/D^3$. Accordingly, in order to obtain the desired particle size distribution, the scaled parcel distribution must be implemented in the spray model which brings the following parcel distribution [79],

$$CDF = \frac{1}{N}\int_0^D f(D)_{parcel}\, dD \tag{4.9}$$

where

$$f(D)_{parcel} = D^3 \frac{nD^{n-1}}{\tilde{D}^n} \exp\left[-\left(\frac{D}{\tilde{D}}\right)^n\right] \tag{4.10}$$

and N is a normalization factor, which is calculated as follows

$$N = \int_0^\infty f(D)_{parcel}\, dD = \tilde{D}^3 \Gamma\left(\frac{3}{n}+1\right) \tag{4.11}$$

where Γ is the Gamma function. Thus based on the above relations and by performing certain calculations, the arithmetic and Sauter mean diameter are respectively calculated as

$$D_{10} = \tilde{D}\Gamma\left(\frac{1}{n}+1\right) \tag{4.12}$$

$$D_{32} = \tilde{D}\frac{\Gamma\left(\frac{3}{n}+1\right)}{\Gamma\left(\frac{2}{n}+1\right)} \tag{4.13}$$

By using Eqs. (4.12) and (4.13) the variation of D_{32}/D_{10} versus the spread parameter, n, is shown in Figure 4.10. Accordingly, by using, D_{32}, and D_{10}, which are obtained experimentally, the value of n can be found from Figure 4.10 and subsequently the value of $\tilde{D}$ can be specified.

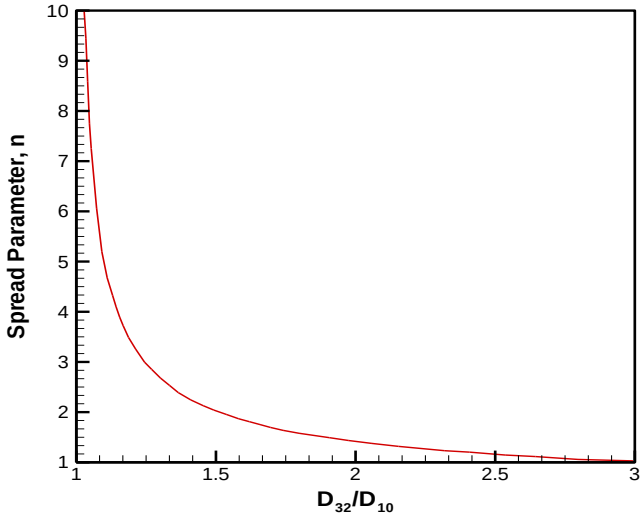


Figure 4.10: Variation of spread parameter, n, with respect to the experimental data, D_{32} and D_{10}

Very close to the nozzle, since the break-up of the main jet takes place with a complex unstable behavior and also as the optical thickness is considerably high, measuring the spray distribution is not accurate using the current equipment. Therefore, the experimental studies at NRC for a vertical nozzle test were brought to the closest reliable location at 4 inches downstream of the tip of the injector. Correspondingly, the experiments at this location provided the values of $D_{32} = 48.29\ \mu m$ and $D_{10} = 23.69\ \mu m$. The Rosin-Rammler parameters were obtained from Eqs. (4.12) and (4.13) and are presented in Table 2.1.

In order to test the capability of this Rosin-Rammler function, it was compared with the experimentally derived size distribution of a spray as shown in Figure 4.11. The experimental size distribution of the spray was acquired from a single orifice with the injection pressure of 50 psi at the distance of 4 inches downstream of the tip of the nozzle, and was compared with the corresponding Rosin-Rammler function. Although, there are some discrepancies for small droplets, the Rosin-Rammler function is generally a good fit to the data. The histogram doesn't show the droplets with the size of smaller than 18 μm, a limitation of the lens used in the LDS device where it is able to detect only the sizes of the droplets which are above 18 μm.

Due to the symmetry of the injector, the simulation was performed on only one wedge with the angle of 60 degrees corresponding to one injection orifice. The total run-time is set to be 1 second and the Cartesian grid size of 66×116×11 was applied on the wedge with length and radius of 4 and 6 inches respectively. The working fluid is water, and the water spray initial condition is defined using the data (Table 2.1) obtained from the Rosin-Rammler relation described earlier.

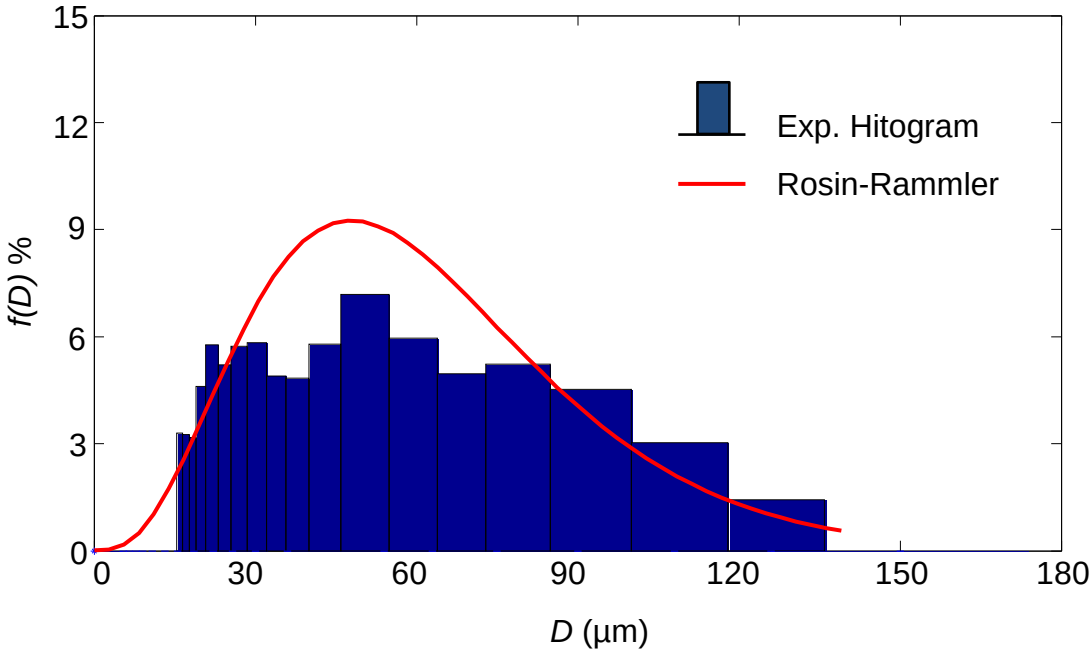


Figure 4.11: Rosin-Rammler distribution versus experimental data

Several experiments were performed at NRC's Sands and Oil Laboratory on the size distribution of the droplets [77] and compared with the simulation results. Figure 4.12 shows the experimental and numerical arithmetic mean diameter distribution of the spray, D_{10}, along the centerline of the innermost coil of the heater. The numerical curve follows reasonably the experimental data. As can be seen, the droplet size doesn't greatly change which means that the effect of evaporation is not very high especially at upstream locations. While evaporation does not play a major role at upstream locations, its effect cannot be neglected. Although the evaporation takes place weakly in room temperature indeed, still some part of the droplets masses are transferred to the environment due to considerable convective heat transfer rate between the air and the spray as a result of high velocity of the droplets. Also, since an increase in the mean diameter is observed, it can be concluded that the role of coalescence is not negligible either.

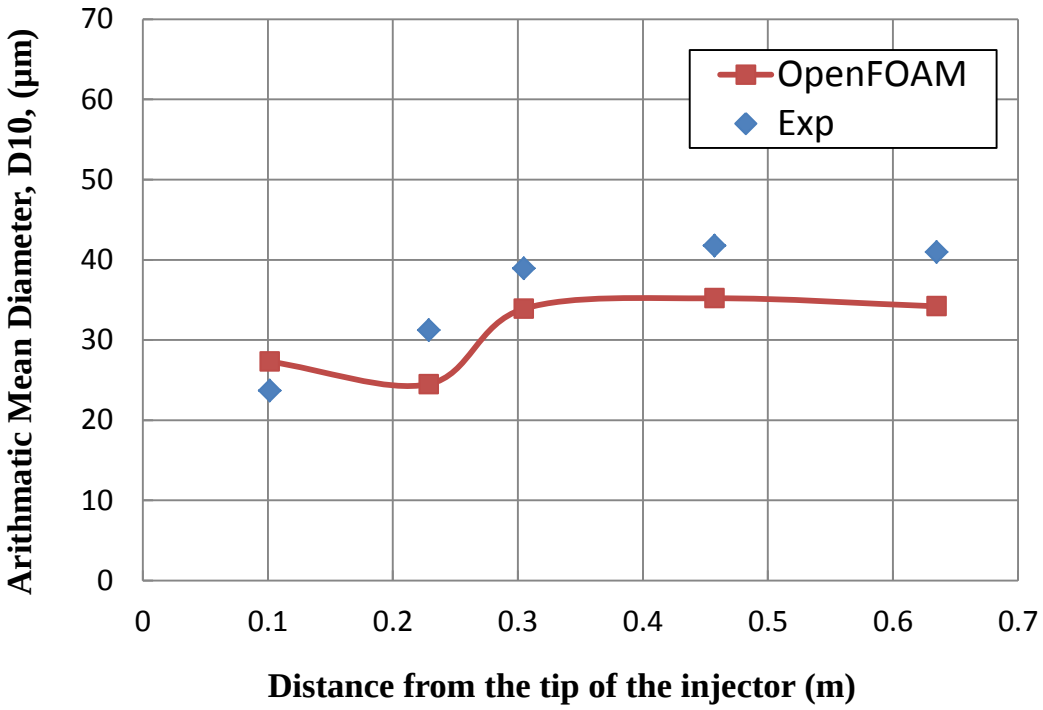


Figure 4.12: Distribution of numerical and experimental arithmetic mean diameter of spray at various axial locations

Figure 4.13 shows the spray distribution with their velocity magnitudes inside the computational domain. A gradation of droplets sizes in the radius of each of the cones is observed which is due to the different dynamic behaviour of small and large droplets. Principally, the reason is that large droplets are dispersed by the spray initial cone angle and subsequent interactions with turbulent fluctuations in the entrained air, while smaller droplets are generally brought to the spray centerline by aerodynamic drag interactions with the entrained air.

From Figure 4.12 and Figure 4.13, it can be observed that the droplet size distribution becomes more uniform further downstream because most flow mechanisms (e.g. turbulent mixing and coalescence) become spatially uniform.

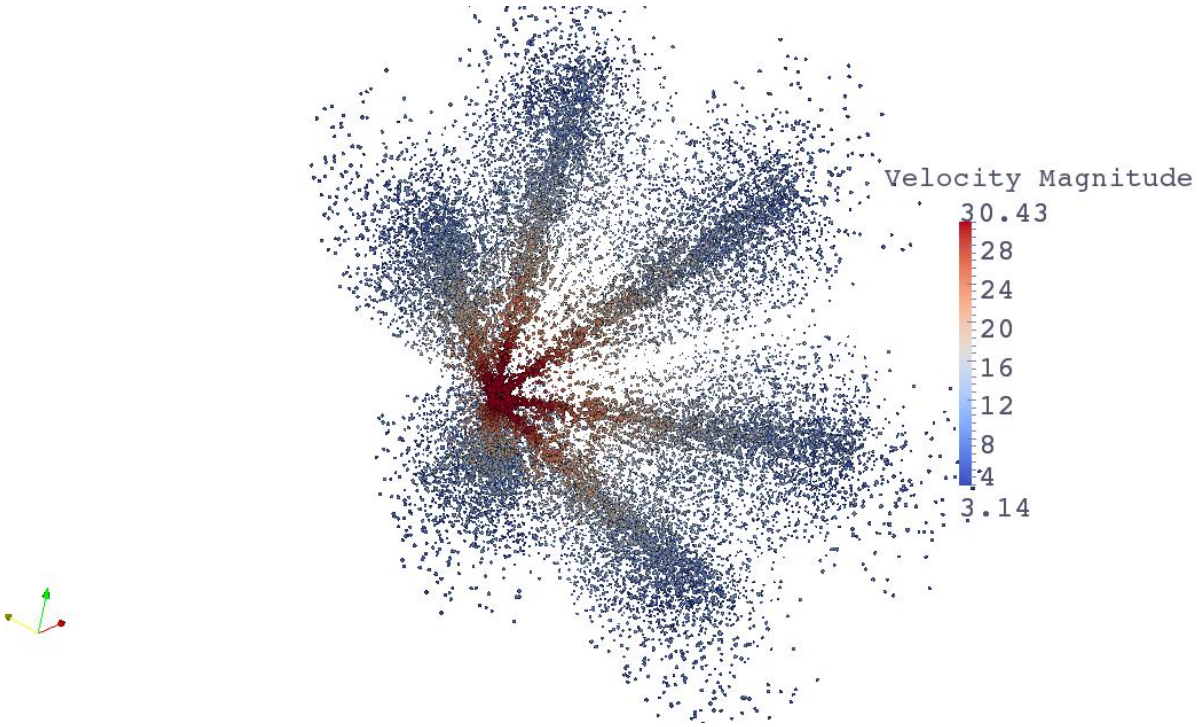


Figure 4.13: Spray distribution with droplets velocity magnitudes (m/sec)

4.3.4 Simulation of Combustion inside the Process Heater

In this section, the results of the simulation of reacting flow in the GenTex diesel fired cylindrical process heater are presented. The calculations were performed on a finite-difference grid of 70×19×11 nodes. 1500 parcels were defined to represent the spray coming out of the injector. It was assumed that the droplets evaporate first (evaporation process) and then oxidizes (burning process). The unsteady solver simulation converged to steady-state condition less than 1.0 second after the injection of the spray.

The presence of air coming through the perforated drum was neglected. Inlet air conditions are thoroughly studied and presented by Ref. [4] and summarized in Table 4.2 along with other inlet conditions.

Figure 4.14 shows the predicted velocity vectors of the flow inside the innermost region. As expected, it indicates two features of swirling flow; central recirculation zones oriented in front of the burner and wall-bounded vortices formed in the corner of the heater. Near the burner, due to the geometry conditions and also the inlet flow conditions such as highly-swirling inlet air, the flow moves toward the wall [53]. Therefore, this motion makes the central recirculation zone larger (resulting in a low fully development rate) and the corner vortex smaller, which helps the flame become stable. If no recirculation zone were created in front of the burner, no fuel would be trapped in the near burner region and the flame would quench. Moving further downstream, the recirculation region gets thinner and the flow continuously gets closer to the fully-developed condition, all of the flow in the same direction.

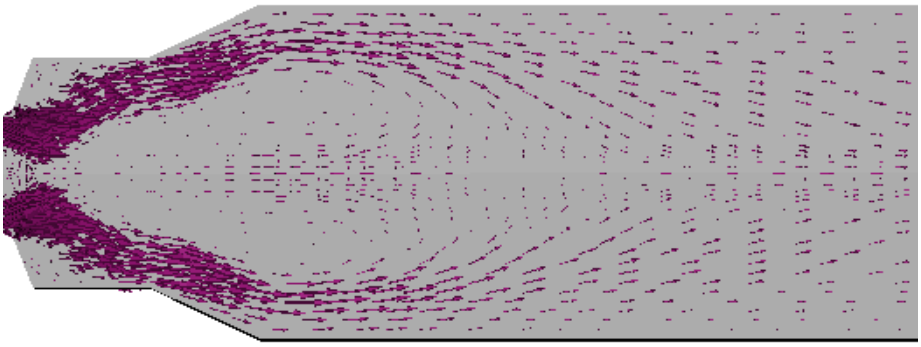


Figure 4.14: Velocity vectors of the combusting gases inside the heater

Figure 4.15 indicates how the fuel has been spread through the domain. To avoid confusion the spray distribution of only one orifice is shown here. It can be observed that the majority of droplets disappear when they intersect the air flow coming from the swirlers. Observations from Figure 4.17 support this idea, which results in the combustion zone being very close to the interaction of the droplet and swirling air.

Only a minority of droplets persist further into the domain. These are normally larger droplets with high momentum which can pass the spray/air intersection area and follow the air flow path with a relatively high speed (as shown in Figure 4.15) and get closer to the walls. The presence of a small number of droplets near the walls is normally not a considerable concern, but if this amount increases, it will spread the flame from the center which might cause corrosion of the coils, pollutant formation, incomplete combustion and consequently lower combustion efficiency. The spray aerodynamic characteristics and the air/spray interactions play the major role in this phenomenon. Therefore, these factors should be taken into account in cases of shape designing the injector, and computing air/fuel ratio and swirling/axial velocities.

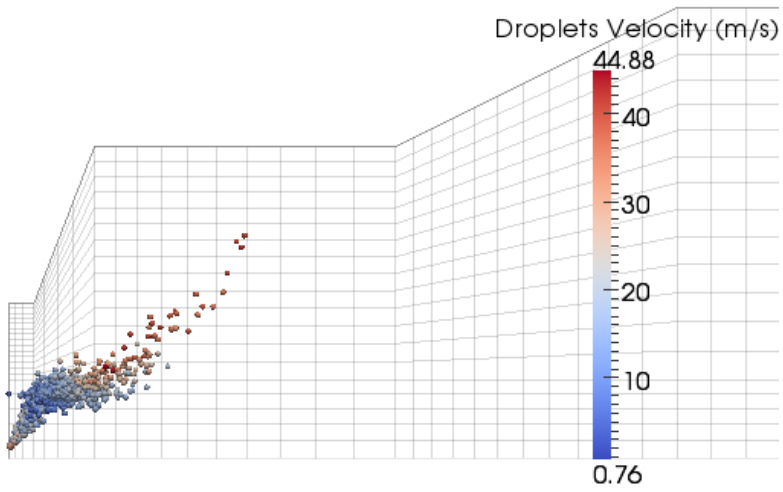


Figure 4.15: Fuel spray distribution

The results of the current numerical simulation were validated against the previously-described experimental results. Figure 4.16 shows the temperature distribution of the combustion gases along the centerline of the heater. The numerical and experimental results are in satisfying agreement. As can be realized, in a very short distance from the tip of the nozzle, the temperature is not high which shows that most of the evaporation and combustion hasn't taken place yet. The

sudden rise in the temperature indicates where, axially, combustion takes place and the simulation satisfactorily predicts this location. In the downstream direction, the temperature changes smoothly with little variation along the centerline. This comes as a consequence of the relatively uniform heat transfer of the combustion gases owing to the large size recirculation zones. In this case, the flow transfers heat uniformly to the walls as a result of flow recirculation inside this area. This result shows the importance of the swirling velocity and the burner/heater geometry on the uniformity of heat transfer inside the process heater.

As mentioned in Chapter 2, since combustion problems deal with high temperatures, the radiation heat transfer plays a major role on the total phenomena [80]. While the absorption/emission coefficients of the gas varies inside the heater, in modeling the radiation for the current fuel oil combustion problem, it is assumed that the radiative properties of the species are constant regardless of the species concentration and the flow temperature. This assumption causes the temperature to be over-predicted in some areas such as along the centerline as shown in Figure 4.16, and under-predicted in others. The reason is that, in reality, the areas with higher temperature such as the flame should have a higher absorption/emission values which means they radiate more energy to the surrounding compared to the predicted values. Therefore, the actual temperature of these areas should be lower than the numerical results. Moreover, the uncertainty of the temperature values obtained from the k-type thermocouple rises in high temperatures; however this should result in a very slight change in the experimental data.

Other causes of the difference between the numerical and experimental results can be identified. These include neglecting the soot formation, implementation of reduced chemical mechanisms instead of considering the fully-complex chemical reactions, experimental errors in

measuring the droplet size distribution caused by both the apparatus and the operators, and using a water spray distribution instead of a diesel fuel distribution.

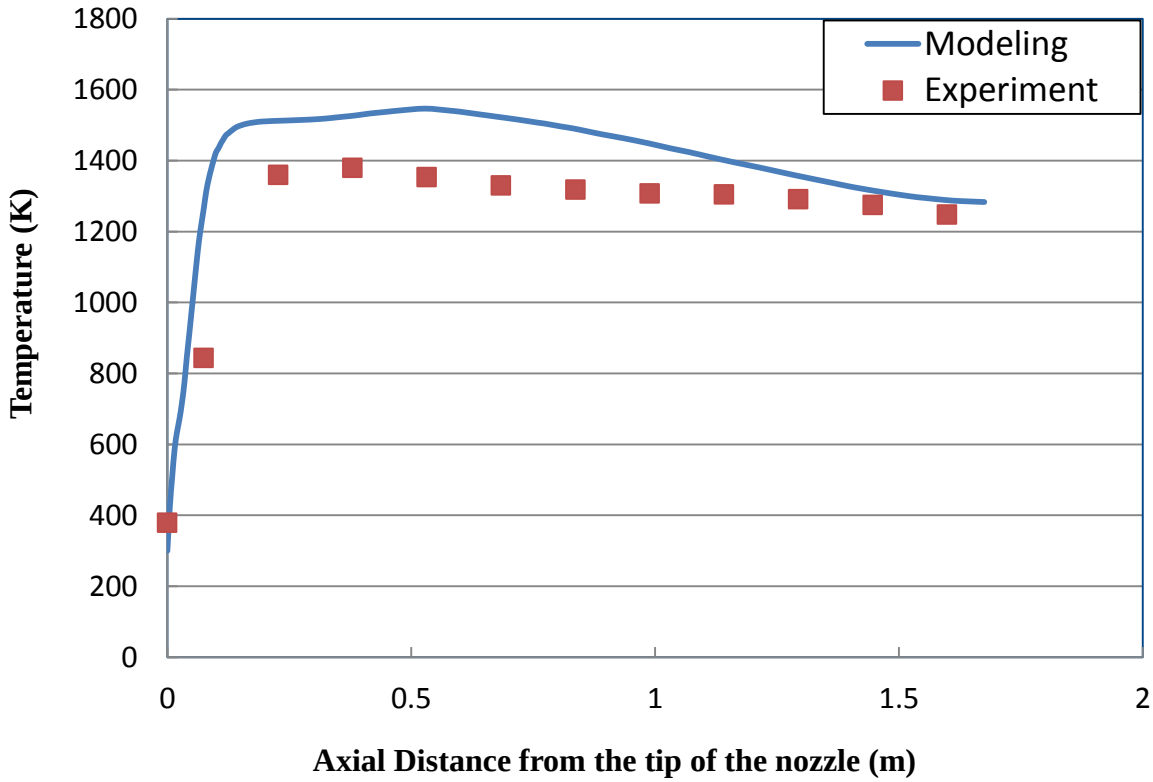


Figure 4.16: Comparison of numerical and experimental temperature profile along the centerline

The calculated temperature contours of the flow are illustrated in Figure 4.17. As the fuel is injected with an angle with respect to the centerline and it mixes and reacts with the air followed by the main flow path on the recirculation zone, the flame doesn't take place along the centerline (off-centre flame). Accordingly, most of the combusting gases go to the recirculation zone resulting in the temperature rise in that area.

The size of the recirculation zone has a great effect on the performance of the process heater. If it is too small, all the fuel vapour is not able to enter the recirculation zone which causes lower efficiency and more incomplete combustion. On the other hand, if the recirculation zone is too large, the flame gets closer to the walls causing corrosion and a lower mixing rate. While the current off-centre flame has brought reasonable efficiency for the present heater [4], the optimal size of the recirculation zone could be still studied to enhance the heat transfer to the coils.

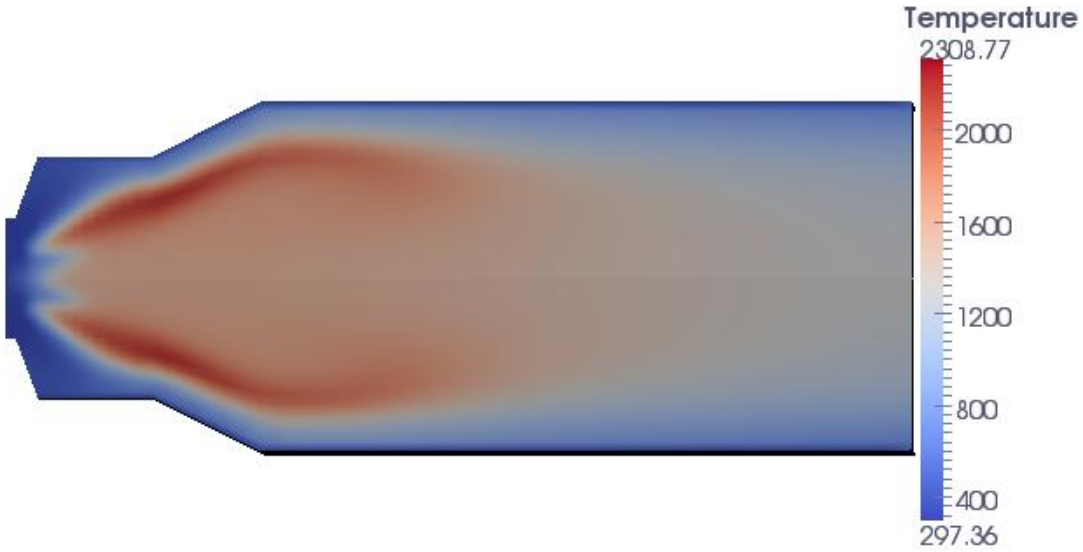


Figure 4.17: Temperature contours (units are in Kelvin)

Figure 4.18 reveals the influence of recirculation regions on the temperature distribution and the flame shape (In this figure the arrows do not represent the magnitude of the velocity vectors, but only show the direction of the flow). It can be observed from the figure that the major hot combustion gas diffuses to the recirculation zone and makes this area a relatively uniform hot region, while the area near the walls remain fairly cold.

Four basic standard types of diffusion flames are defined and categorized based on their applications by the IFRF flame classifications [81]. From Figure 4.18, the corresponding flame type can be recognized. Based on the IFRF definition of flame types, the current flame best fits the “flame type-1” category because the swirl velocity of inlet air is in a medium range and the fuel jet penetrates partially or fully into the recirculation zone, while the most of the combustion process occurs within this recirculation zone. This type of flame seems to be appropriate for the current process heater since, based on its geometry, the other types would bring too short or too long flames making them intense or unstable which will yield lower efficiency and higher pollutant content.

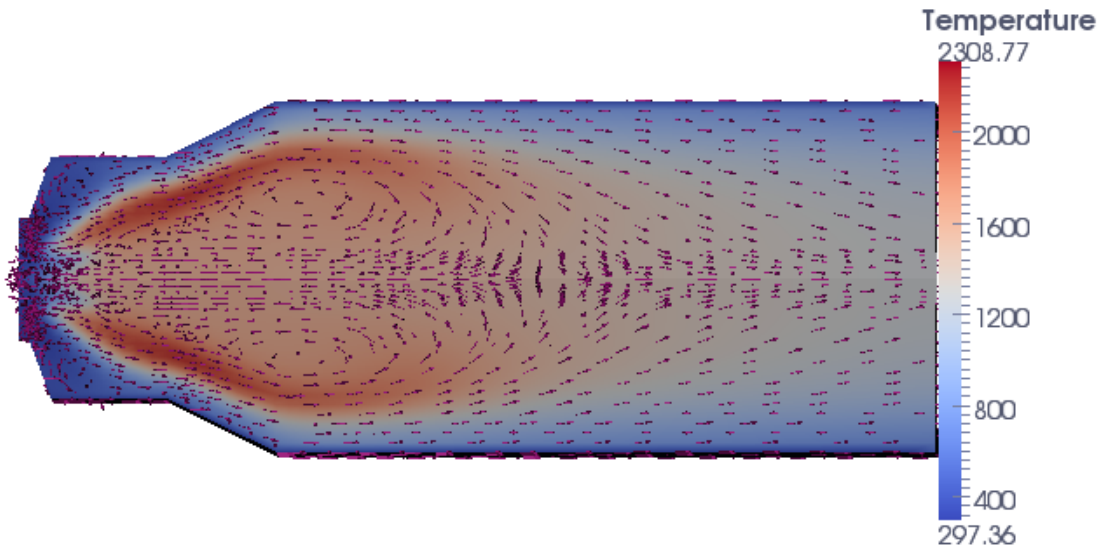


Figure 4.18: Effect of recirculation zones on the shape of temperature distributions (temperature in kelvin)

4.4 Optimization of the Performance of the GenTex Process Heater

Although the studies on the current heater illustrate its good performance, it is believed that still a number of modifications could be made to improve the performance, e.g. uniform heat

transfer to the coils. The relevant design parameters and the optimization objective function, along with the prescribed optimization method, are briefly explained in the next chapter and offered as a suitable extension to the current work.

Chapter 5
Conclusions and Future Work

5.1 Major Conclusions

In this section, a summary of the major outcomes of the current research along with the challenges in the modeling and the advantages over the other methods are presented for each of the combustion design problems.

5.1.1 Optimization of the BERL Furnace

This thesis showed how model-based design optimization, in this case based on response surface methodology, can be applied to optimize the design of a natural-gas fired furnace. As the first step, the CFD model satisfactorily predicted details of the reacting flow inside the furnace. The model was then linked to the response surface optimization algorithm with the interface module written in C++. The objective of this optimization problem was to maximize the conversion of fuel to carbon dioxide, by changing the quarl angle and swirl number.

The initial design configuration of the furnace was chosen to have the quarl angle of 20° and swirl number of 0.67 with CO_2 mass generation of 51.313 kg/hr whereas the optimization method offered the optimum solution equal to 23.2° and 0.51 for quarl angle and swirl number respectively. The value of the objective function for the optimal solution resulted in 52.698 kg/hr CO_2 production, leading to an improvement in fuel utilization.

As expressed in the context of this thesis, a major challenge of model-based design optimization lies in the computationally-intense nature of CFD combustion simulations, and unexpected divergence caused by the stiffness of the governing equations. Initializing the field variables of each CFD evaluation using values from the previous design iteration avoided divergence and greatly reduced the overall computation time. The number of iterations needed to converge the governing equations for each objective function evaluation was generally reduced to below 1000, resulting in a total calculation time of 10 hours and 18 minutes running on a single core computer.

All of the mentioned advantages together lead to a suitable algorithm for combustion equipment, which is also computationally inexpensive compared to the other methods. Furthermore, this algorithm is not restricted to the furnace studied here only; it indeed can be simply extended to many other types of combustion devices.

5.1.2 Modeling the GenTex Diesel-Fired Process Heater

The combustion of diesel spray in the GenTex 50M process heater was studied in the next part of this thesis. At the first step, the injection spray pattern was characterized experimentally using laser diffraction spectrometry. Then the temperature distribution along the centerline of the heater was measured and compared with the results of combustion modeling. Numerical simulations were carried out for both the cold spray and the diesel combustion and the outcome was in good agreement with the experimental data.

It was shown how several two-phase phenomena, e.g. evaporation, turbulent mixing and coalescence influence the droplet size distribution throughout the spray field, consequently

indicating the importance of the geometry and shape of the injector on the overall performance of the process heater.

The other important parameter affecting the performance of the process heater was found to be the size of the central recirculation zone. If it is too small, all the fuel vapour is not able to enter the recirculation zone which causes lower efficiency and incomplete combustion, while if the recirculation zone is too large, the flame gets closer to the walls causing corrosion and a lower mixing rate. This result showed the importance of the swirling velocity and the burner/heater geometry on the uniformity of heat transfer inside the process heater.

On the whole, it was concluded that in a successful parametric design of a liquid fuel-fired combustor it is necessary to consider the effect of design parameters on the spray aerodynamic characteristics and size distribution, the air/spray interactions, and the size of the recirculation zones.

While the present study illustrated good functionality of the process heater in question, it is presumed that it would be worthwhile to study some of the other capabilities of this heater using the model-based optimization algorithm explained in Section 5.2.3.

5.2 Suggestions for Future Work

Although the optimization presented in this thesis represents a step toward model-based optimization, in order to reach the final step, an optimization scheme widely-applicable to complex industrial combustion problems, significant research is still needed. This section suggests some research directions that could provide for the next steps.

5.2.1 Optimization of BERL Furnace by Improving the Radiation Model

As mentioned in Section 2.3.5, the radiant properties of the combusting gas inside the BERL furnace domain are approximated as uniform, while the property values actually change throughout the domain. In order to make precise radiative transfer calculations in absorbing/emitting gases such as the BERL furnace, it is recommended that the Full Spectrum Correlated-k Distribution (FSCK) model [82] be added to the current radiation library. The FSCK distributions are collected from pre-determined narrow band k-distributions with the aim of reducing the computation time. Using this method decreases the CPU time considerably compared to the narrow band methods [82].

By modifying the radiation model, it would be possible to investigate other objectives such as, the effect of heat transfers on the walls. Since the radiation heat transfer mode plays a major role in the temperature distribution of the walls, seeking the most uniform radiant heat load on the walls using the FSCK model could be the next goal in further optimizing the furnace.

5.2.2 Multi-Objective Optimization of the BERL Furnace

In the current work, the optimization of a single-objective function was studied; while the combination of several objectives, e.g. minimizing NO_x or CO while producing a uniform outlet temperature distribution, could be investigated as a further step. To model the multi-objective optimization of the BERL furnace two major methods are offered to enhance the current RSM model: the PRESS weighted objective function; and a Pareto front analysis.

5.2.2.1 PRESS Weighted (PWS)

In this approach, a weighted average of objective functions combines information from multiple individual objective functions using a weighting scheme [83]. There are several weighting techniques available in Ref. [83]. On occasion, based on the nature of the problem, it is also possible to heuristically define the weights of each of the objective functions.

5.2.2.2 Pareto Front

In this method, Pareto-optimal solutions combine the set of objectives that are not dominated by any other objective. The set of Pareto-optimal solutions are then used to create a Pareto front [84] which represents all optimal combinations of the objectives when their relative importance is unknown. In a two-objective problem, for instance, the Pareto front delivers a single line in a two dimensional geometry where the optimum point could be recognized simply by the user/designer.

5.2.3 Surrogate-Based Model Optimization of the GenTex 50M Process Heater

Surrogate-based modeling [85, 86] is a robust method for finding an optimal solution in optimization problems which involve complex objective functions and a large number of design variables. This model combines different minimization techniques, e.g. polynomial response surface [85], Kriging [86], radial-basis neural network [87], and PRESS weighted surrogates [86]. At the first step of the optimization procedure, each of the minimization models are solved for some different sample points and then the error assessment (difference between the value of the actual and the approximated objective functions) for each of the minimization models is made. Then the error values are compared to each other and one best method is selected. Since

the best method, in most of minimization problems, is problem-dependant, using this step helps to choose the most accurate method for optimization.

In the next step, a global sensitivity analysis [88] is performed on the selected method with the purpose of recognizing the relative effect of design variables on the objective functions. By doing so, some design variables which have weak effects on the total performance of the objective function are removed, consequently resulting in a reduction of the dimension of the design space. Finally, the multi-objective optimization problem is solved for the effective design variables using the Pareto Front technique. This provides a set of optimal solutions, from which the designer identifies the most desirable one.

5.2.4 Application of Metaheuristic Optimization Algorithms

Design optimization studies associated with non-gradient methods which use metaheuristic algorithms, e.g. simulated annealing and genetic algorithms [39] are rapidly growing.

Compared to gradient-based methods, simulated annealing is a stronger method to find the optimal solution when the objective function has multiple local minima [89], but it is computationally more expensive since it needs many more analysis evaluations to converge.

Genetic algorithms are a set of algorithms that are derived from Darwin's theory of natural selection. The framework of genetic algorithms as search and optimization tools is described in details in Ref. [90]. Contrary to gradient-based methods, genetic algorithms search from a population of points, not a single point at a time, and they use objective function information, not derivatives. In addition, they are written based on a probabilistic approach,

unlike the gradient-based methods which are deterministic [90]. This method is especially suitable when the objective function is multi-modal, i.e. large numbers of local minima, where gradient-based methods are unlikely able to find an optimal point satisfactorily.

As noted above, gradient-based and metaheuristic methods operate completely differently, having their own advantages and disadvantages. In the current study, it is not clear which of the two approaches is more efficient. Implementation of a metaheuristic method, e.g. genetic algorithm, to extend the current case studies, in order to compare the capabilities, and suitability of each of the methods in typical combustion problems, would be a useful exercise.

5.2.5 Studying Other Design Variables and Objective Functions

Another way to enhance the performance of the combustion chambers described in this thesis is to investigate the effect of several other design variables. New design parameters that could be studied are air/fuel ratio, confinement ratio (the combustor diameter divided by burner throat diameter), inlet air temperature, number of injector holes, wall temperature distribution, amount and the location of the secondary inlet air, fuel staging, fuel flow rate, burner quarl shape (by mapping a B-spline to the boundary of the quarl), and heat flux input to the combustor. Each of these variables could change the aerodynamics of the flow, coupling between temperature and turbulent kinetics, and/or Damkohler number of the flow, consequently affecting most of the prescribed objective functions. Accordingly, all of these parameters have the potential to bring added improvements to the combustors; however in order to avoid the enlargement in dimension of the design space, global sensitivity analysis should be performed as explained in Section 5.2.3.

In addition to the design variables, several other objectives could be investigated for enhancing the performance of the combustors. Since, in the recent decades, emissions reduction in combustion-related industrial equipment is one of the major priorities to mitigate global warming, and environmental pollution, objectives such as minimizing CO and NO_x in combustion chambers has become of great importance. A comprehensive table of detailed chemical reactions along with powerful solvers for modeling chemical kinetics will lead to a reasonable CO estimation; however current NO_x models still have some discrepancies with respect to measurements. In addition, using complex chemical mechanisms greatly increases the computational time, requiring faster CPUs or parallel processors. Other objectives that could be studied are uniform radiant heat load on the combustor walls, uniform temperature distribution at some point in the furnace, percent of complete combustion, overall variation of temperature or heat flux. In a certain optimization problem, each of these objectives could be studied separately (single-objective) or could accompany other objectives (multi-objective) as defined in Section 5.2.2. Studying multi-objective optimization is of greater value to the industrial combustion community than single-objective optimization as the multi-objective approach satisfies several goals at one time in the practical design of the combustion equipment.

5.2.6 Parallel Processing in the Simulation of the GenTex Process Heater

In the optimization procedure of the BERL furnace, since the flow field for each objective function evaluation was initialized with the numerical results of the previously converged point, parallel processing doesn't decrease the calculation time appreciably. However as the GenTex process heater was modelled by an unsteady CFD simulation that is required to solve the problem for each point from the initial time, parallel processing could significantly

reduce the computation time. The response surface methodology is amenable to parallelization, as each objective function evaluation used to construct the response surface is carried out independently. Accordingly, each of the function evaluations could be tasked to a separate processor to speed computational, however OpenFOAM has the ability of parallel processing on its own. By reducing the calculation time more complex problems with more details could be solved, e.g. multi-objective optimization with detailed chemical mechanisms in complex geometries.

References

[1] R. H. Myers, D. C. Montgomery and C. M. Anderson-Cook, Response Surface Methodology, 3rd ed., Wiley, 2009.

[2] OpenFOAM, "the Open Source CFD Toolbox," 2008. [Online]. Available: http://www.openfoam.com/. [Accessed Version 1.5].

[3] A. Sayre, N. Lallemant, J. Dugue and R. Weber, "Scaling Characteristics of Aerodynamics and Low NOx Properties of Industrial Natural Gas Burners. The Scaling 400 StudyPart IV: The 300 kW BERL Test Results," International Flame Research Foundation, 1994.

[4] K. J. Daun, S. Hajitaheri and J. L. Wright, "Numerical Simulation of the GenTex 50M Heater," 2011.

[5] OpenFOAM, "User Guide," 2008. [Online]. Available: http://foam.sourceforge.net/docs/Guides-a4/UserGuide.pdf.

[6] C. Pianese and G. Rizzo, "Interactive Optimization of Internal Combustion Engine Tests by means of Sequential Experimental Design," in *ASME ESDA 96*, Montpellier, France, Jul. 1996.

[7] K. Nakakita, T. Kondoh, K. Ohsawa, T. Takahashi and S. Watanabe, "Optimization of Pilot Injection Pattern and its Effect on Diesel Combustion with High Pressure Injection," *JSME International Journal*, pp. 966-973, 1994.

[8] J. P. Bingue, A. V. Saveliev and L. A. Kennedy, "Optimization of Hydrogen Production by Filtration Combustion of Methane by Oxygen Enrichment and Depletion," *International Journal of Hydrogen Energy*, vol. 29(13), pp. 1365-1370, 2004.

[9] F. Hasselriis, "Optimization of Combustion Conditions to Minimize Dioxin Emissions," *Waste Management & Research*, vol. 5, p. 311–326, 1987.

[10] G. J. Hesselmann, "Optimization of Combustion by Fuel Testing in a NOx Reduction Test Facility," *Fuel*, vol. 76(13), pp. 1269-1275, 1997.

[11] J. Nocedal and S. J. Wright, Numerical Optimization, 2nd ed., New York: Springer, 2006.

[12] M. Sinoda, R. Tanaka and N. Arai, "Optimization of Heat Transfer Performances of a Heat-Recirculating Ceramic Burner during Methane/Air and Low-Calorific-Fuel/Air

Combustion," *Energy Conversion and Management,* vol. 43, pp. 1479-1491, 2002.

[13] D. Buche, P. Stoll, R. Dornberger and P. Koumoutsakos, "Multi-objective Evolutionary Algorithm for the Optimization of Noisy Combustion Processes," *IEEE Transactions on Systems, Management, and Cybernetics,* vol. 32(4), Nov. 2002.

[14] C. O. Pascherelt, B. Schuermans and D. Büche, "Combustion Process Optimization Using Evolutionary Algorithm," in *Proceedings of ASME Turbo Expo*, Jun. 2003.

[15] N. Hansen, A. S. P. Niederberger, L. Guzzella and P. Koumoutsakos, "A Method for Handling Uncertainty in Evolutionary Optimization with an Application to Feedback Control of Combustion," in *IEEE Transactions on Evolutionary Computation*, Feb. 2009.

[16] J. Z. Chu, S. S. Shieh, S. S. Jang, C. I. Chien, H. P. Wan and H. H. Ko, "Constrained Optimization of Combustion in a Simulated Coal-Fired Boiler using Artificial Neural Network Model and Information Analysis," *Fuel,* vol. 82, pp. 693-703, 2003.

[17] D. Büche, P. Stoll and P. Koumoutsakos, "An Evolutionary Algorithm for Multi-objective Optimization of Combustion Processes," *Center for Turbulence Research Annual Research Briefs,* pp. 231-239, 2001.

[18] J. I. Madsen, W. Shyvy and R. T. Haftka, "Response Surface Techniques for Diffuser Shape Optimization," *AIAA Journal,* vol. 38(9), pp. 1512-1518, 2000.

[19] S. Y. Han and J. S. Maeng, "Shape Optimization of Cut-off in a Multi-Blade Fan/Scroll System Using Neural Network," *International Journal of Heat and Mass Transfer,* vol. 46(15), pp. 2833-2839, 2003.

[20] C. H. Cheng and M. H. Chang, "Shape Design for a Cylinder with Uniform Temperature Distribution on the Outer Surface by Inverse Heat Transfer Method," *International Journal of Heat and Mass Transfer,* vol. 46(1), pp. 101-111, 2003.

[21] G. Fabbri, "Effect of Viscous Dissipation on the Optimization of the Heat Transfer in Internally Finned Tubes," *International Journal of Heat and Mass Transfer,* Vols. 47(14-16), pp. 3003-3015, 2004.

[22] G. Katsaros, F. Campos, D. Kyriazis and T. Varvarigou, "CFD Automatic Optimization using OpenFOAM in Grid Environments," in *OpenFOAM Intl. Conf.*, London, U.K., Nov. 2007.

[23] D. Thévenin and G. Janiga, Optimization and Computational Fluid Dynamics, Berlin: Springer, 2008.

[24] B. M. Aizenbud and Y. B. Band, "Optimization of a Model Internal Combustion Engine,"

Journal of Applied Physics, vol. 53(3), pp. 1277-1282, 1982.

[25] P. J. Smith, W. A. Sowa and P. O. Hedman, "Furnace Design Using Comprehensive Combustion Models," *Combustion and Flame*, vol. 79, pp. 111-121, 1990.

[26] C. D. Correa and P. J. Smith, "Optimization of Ehtylene Furnace Operations," in *1998 AiChe Annual General Meeting*, Miami Beach, 1998.

[27] D. Thévenin, K. Zähringer and G. Janiga, "Automatic Optimization of Two-Dimensional Burners," in *Proceedings of the European Combustion Meeting ECM05*, Louvain-la-Neuve, Belgium, 2005.

[28] G. Janiga and D. Thévenin, "Reducing the CO Emissions in a Laminar Burner using Different Numerical Optimization Methods," *Journal of Power and Energy*, vol. 221(5), p. 647–655, 2007.

[29] L. A. Catalano, A. Dadone, D. Manodoro and A. Saponaro, "Efficient Design Optimization of Duct-Burners for Combined-Cycle and Cogenerative Plants," *Engineering Optimization 38*, pp. 801-820, 2006.

[30] O. S. Motsamai, j. A. Visser and R. M. Morris, "Multi-Disciplinary Design Optimization of a Combustor," *Engineering Optimization* , vol. 40, no. 2, pp. 137-156, 2008.

[31] J. A. Snyman and A. M. Hay, "The Dynamic-Q Optimization Method: an Alternative to SQP?," *International Journal of Computers and Mathematics with Applications*, vol. 44, pp. 1589-1598, 2002.

[32] V. Yakhot, S. Orszag, S. Thangam, T. Gatski and C. Speziale, "Development of Turbulence Models for Shear Flows by a Double Expansion Technique," *Physics of Fluids A*, vol. 4, no. 7, pp. 1510-1520, 1992.

[33] J. Chomiak and A. Karlsson, "Flame Liftoff in Diesel Sprays," in *Twenty-Sixth Symposium (International) on Combustion, The Combustion Institute, Pittsburg*, Naples, Italy, 1996.

[34] V. I. Golovitchev, N. Nordin, R. Jarnicki and J. Chomiak, "3-D Diesel Spray Simulations Using a New Detailed Chemistry Turbulent Combustion Model," in *CEC/SAE Spring Fuels & Lubricants Meeting & Exposition*, Paris, June 19-22, 2000.

[35] C. K. Westbrook and F. L. Dryer, "Simplified Reaction Mechanisms for the Oxidation of Hydrocarbon Fuels in Flames," *Combustion Science and Technology*, vol. 27, pp. 31-43, 1981.

[36] R. Siegel and J. R. Howell, Thermal Radiation Heat Transfer, 4 ed., New York: Taylor & Francis Inc., 2002.

[37] N. W. Bressloff, J. B. Moss and P. Rubini, "The Differential Total Absorptivity Solution to the Radiative Transfer Equation for Mixtures of Combustion Gases and Soot," *Numerical Heat Transfer, Part B : Fundamentals,* vol. 31, no. 1, pp. 43-60, 1997.

[38] M. F. Modest, Radiative Heat Transfer, New York: Academic Press, 2003, pp. 465-492.

[39] C. E. J. Baukal, V. Y. Gershtein and X. Li, Computational Fluid Dynamics in Industrial Combustion, USA: CRC Press, 2001.

[40] M. S. Day and J. B. Bell, "Numerical Simulation of Laminar Reacting Flows with Complex Chemistry," *Combust. Theory Modelling,* vol. 4, no. 4, pp. 535-556, 2000.

[41] "ODE System Solvers," [Online]. Available: http://www.openfoam.com/features/ODE-solvers.php.

[42] F. J. H. and Peric M., Computational Methods for Fluid Dynamics, third ed., Springer-Verlag, 2001.

[43] D. Veynante and L. Vervisch, "Turbulent Combustion Modeling," *Progress in Energy and Combustion Science,* pp. 193-266, 2002.

[44] Y. Zhang and A. L. Boehman, "Autoignition of Binary Fuel Blends of n-Heptane and C7 Esters in a Motored Engine," *Combustion and Flame,* vol. 159, no. 4, pp. 1619-1630, 2011.

[45] W. P. Jones and S. Navarro-Martinez, "Numerical Study of n-Heptane Auto-ignition Using LES-PDF Methods," *Flow, Turbulence and Combustion,* vol. 83, no. 3, pp. 407-423, 2009.

[46] S. R. Turns, An Introduction to Combustion: Concepts and Applications, New York: McGraw-Hill, 1996.

[47] K. N. C. Bray, "Turbulent Transport in Flames," *Proceedings: Mathematical and Physical Sciences,* vol. 451, pp. 231-256, 1995.

[48] R. W. Bilger, "Turbulent Diffusion Flames," *Annual Review of Fluid Mechanics,* vol. 21, pp. 101-135, 1989.

[49] J. A. J. Karlsson, Modeling Auto-Ignition, Flame Propagation and Combustion in Non-stationary Turbulent Sprays, Goteborg: Chalmers University of Technology, 1995.

[50] K. J. Daun, "Heat Transfer Analysis of the GenTex 70M Process Heater," Prepared for: John Person, Tangent Engineering Design Services, 2009.

[51] C. E. J. Baukal, Heat Transfer in Industrial Combustion, Boca Raton FL: CRC Press, 2000, p. 106.

[52] M. Maesawa, Y. Tanaka, Y. Ogisu and Y. Tsukamoto, "Radiation from the Tuminous Flames of Liquid Fuel Jets in a Combustion Chamber," in *Twelfth Symposium (International) on Combustion*, Pittsburgh, PA, 1969.

[53] A. Stambuleanu, Flame Combustion Processes in Industry, Tunbridge Wells, UK: Abacus Press, 1976, p. 311.

[54] J. M. Beér and C. R. Howarth, "Radiation from Flames in Furnaces," in *Twelfth Symposium (International) on Combustion*, Pittsburgh, PA, 1969.

[55] D. B. Spalding, Combustion and Mass Transfer, Elsevier, 1978.

[56] C. Crowe, M. Sommerfeld and Y. Tsuji, Multiphase Flows with Droplets and Particles, CRC Press LLC, 1998.

[57] G. B. Macpherson, N. Nordin and H. G. Weller, "Particle Tracking in Unstructured, Arbitrary Polyhedral Meshes for Use in CFD and Molecular Dynamics," *Communications in Numerical Methods in Engineering*, 2008.

[58] F. M. White, Fluid Mechanics, McGraw Hill, 2010.

[59] S. W. Park, H. J. Kim and C. S. Lee, "Investigation of Atomization Characteristics and Prediction Accuracy of Hybrid Models for High-Speed Diesel Fuel Sprays," *SAE Paper 2003-01-1045*, 2003.

[60] R. Rotondi, G. Bella, C. Grimaldi and L. Postrioti, "Atomization of High-Pressure Diesel Spray: Experimental Validation of a New Breakup Model," *SAE paper 2001-01-1070*, 2001.

[61] J. Lee and S. Goto, "Comparison of Spray Characteristics in Butane and Diesel Fuels by Numerical Analysis," *SAE Technical Paper 2000-01-2941*, 2000.

[62] D. Fuster, G. Agbaglah, C. Josserand, S. Popinet and S. Zaleski, "Numerical Simulation of Droplets, Bubbles and Waves: State of the Art," *Fluid Dynamics Research*, vol. 41, 2009.

[63] R. Reitz, "Modeling Atomization Processes in High-Pressure Vaporizing Sprays," *Atomization and Spray Technology*, pp. 309-337, 1987.

[64] N. Nordin, "Complex Chemistry Modeling of Diesel Spray," Chalmers University of Technology, Goteborg., 2001.

[65] R. Patil, "Numerical Simulation of Quenching Process in Coal Gasification," Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen-Nürnberg, 2009.

[66] C. Mordasini, "Practical Numerical Training UKNum," [Online]. Available:

http://www.mpia-hd.mpg.de/~mordasini/UKNUM/Lecture_07.pdf. [Accessed 14 4 2012].

[67] A. P. Horsman, "Design Optimization of a Porous Radiant Burner," University of Waterloo, Waterloo, 2010.

[68] "NIST/SEMATECH e-Handbook of Statistical Methods," 2010. [Online]. Available: http://www.itl.nist.gov/div898/handbook/.

[69] P. E. Gill, W. Murray and M. H. Wright, Practical Optimization, London: Academic Press, 1986.

[70] A. P. Horsman and K. J. Daun, "Design Optimization of a Porous Radiant Burner," in *Combustion Institute Canadian Section*, Ottawa, Canada, May 9-12, 2010.

[71] K. C. Kaufman and W. A. Fiveland, "Validation of Industrial Gas Burner Models using In-furnace Laboratory Measurements," in *Proceedings of the AFRC Fall International Symposium*, Monterey, CA, Oct. 15-18, 1995.

[72] J. P. Jessee and W. A. Fiveland, "A Non-Orthogonal Combustion Model for Natural Gas Flames," in *Proceedings of the Third International Conference on Combustion Technologies for a clean Environment*, Lisbon, Portugal, July 3-6, 1995.

[73] A. Saario, A. Rebola, P. Coelho, M. Costa and A. Oksanen, "Heavy Fuel Oil Combustion in a Cylindrical Laboratory Furnace: Measurements and Modeling," *Fuel*, pp. 359-369, 2005.

[74] J. Abbott, "As-Built Nozzle/Atomizer Review and Updates to 200M Prototype Nozzle Design," Tangent Design Engineering Ltd., Calgary, AB, Dec 24th 2010.

[75] M. Kerker, Scattering of Light and Other Electromagnetic Radiation, New York: Academic Press, 1969.

[76] "Sympatec GmbH System-Partikel-Technik," [Online]. Available: http://www.sympatec.com.

[77] D. Kirpalani, S. Hashemi and F. Toll, "Spray Measurements for ThermoGen Heater 50M Y-Type Atomizer," ICPET/ITPCE, National Research of Canada (NRC/CNRC), Ottawa, May 2011.

[78] Tangent Design Engineering Ltd, "50M Heater Solidworks Edrawing," Tangent Design Engineering Ltd., Calgary, AB, 2010.

[79] S. S. Yoon, "Droplet Distributions at the Liquid Core of a Turbulent Spray," *Physics of Fluids*, vol. 17, no. 3, pp. 035103-035103-24, 2005.

[80] N. H. Afgan and J. M. Beer, Heat Transfer in Flames, Washington, D.C.: John Wiley & Sons, 1974.

[81] R. Weber, A. A. F. Peters, P. P. Breithaupt and B. M. Visser, "Mathematical Modeling of Swirling Flames of Pulverized Coal: What can Combustion Engineers Expect from Modeling?," *Journal of Fluid Engineering,* pp. 289-297, 1995.

[82] M. Modest, "The Full-Spectrum Correlated-k Distribution for Thermal Radiation from Molecular Gas-Particulate Mixtures," *Journal of heat transfer,* vol. 124, no. 1, pp. 30-38, 2002.

[83] T. Goel, R. Haftka, W. Shyy and N. Queipo, "Ensemble of Surrogates," *Structural Multidisciplinary Optimization,* vol. 33, pp. 199-216, 2007.

[84] W. Shyy, N. V. Queipo, R. T. Haftka, T. Goel, R. Vaidyanathan and P. K. Tucker, "Surrogate-Based Analysis and Optimization," *Progress in Aerospace Sciences,* vol. 41, pp. 1-28, 2005.

[85] Y. C. Cho, W. Du, G. Amit, W. Shyy, A. M. Sastry and C. C. Tseng, "Surrogate-Based Modeling and Dimension-Reduction Techniques for Thermo-Fluid & Energy Systems," in *Invited talk in ASME/JSME 8th Thermal Engineering Joint Conference*, Honolulu, Hawaii, Mar. 2011.

[86] Y. C. Cho, W. Du, G. Amit, W. Shyy, A. M. Sastry and C. C. Tseng, "Surrogate-based Modeling and Multi-Objective Optimization Techniques for Thermo-Fluid & Energy Systems," in *Second International Conference on Computational Methods for Thermal Problems*, Dalian, China, Sept. 5-7, 2011.

[87] T. Goel, D. Dorney, R. Haftka and W. Shyy, "Improving the Hydrodynamic Performance of Diffuser Vanes via Shape Optimization," *Computers & Fluids,* vol. 37, pp. 705-723, 2008.

[88] I. Sobol, "Sensitivity Estimates for non-Linear Mathematical Models," *Mathematical Modelling and Computational Experiments,* vol. 1, pp. 407-414, 1993.

[89] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi., "Optimization by Simulated Annealing," *Science, New Series,* vol. 220, pp. 671-680, 1983.

[90] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Boston, MA, USA: Addison Wesley, 1989.

[91] G. Smith, D. Golden, M. Frenklach, N. Moriarty, B. Eiteneer, M. Goldenberg, C. Bowman, R. Hanson, S. Song, W. Gardiner, V. Lissianski and Z. Qin, "GRI-Mech Version 3.0 Thermodynamics," 7/30/99.

Appendix A: CHEMKIN file

The chemistry solver in OpenFOAM is able to model all elementary or global reactions. The chemistry reader is written in the Chemkin file format. The mechanism used for the BERL furnace is a two-step reaction which deals with only CH_4, O_2, H_2O, CO, CO_2 species. The Chemkin file allows building reaction schemes and thermophysical properties for species which are not contained within the OpenFOAM's typical libraries, without any need to adapting the original solver; OpenFOAM includes a library that defines the thermophysical and transport properties for all the common species, so therefore only the reactions need to be defined by the user.

The Chemkin file comprises the elements that comprise the reactants and products, (H, N, etc.), the reactant and product species, the chemical reaction, and the Arrhenius equation constants (see Eq. (2.17)). The Arrhenius equation is derived for laminar reactions, while the current problem involves with turbulent combustion. In this case, the reaction rates are governed by the slower phenomenon, whether it is chemistry or turbulent mixing as explained in Section 2.2.2. The Chemkin file used for the case is as follows:

```
!<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>!
!  2-step reaction mechanisms                                                  !
!<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>!
ELEMENTS
H   O    C   N
END
SPECIE
O2     H2O    CO     CO2    N2     CH4
END
REACTIONS JOULES/MOLE
 2CH4 + 3O2 => 2CO + 4H2O  5.012E11      0      2e5
       FORD   / CH4  0.7 /
       FORD   / O2   0.8 /

 2CO + O2 <=> 2CO2         2.239E12      0      1.7e5
       FORD   / CO   1.0 /
       FORD   / O2   0.25 /
END
```

Thermophysical data are provided to the Chemkin linker within a separate file. These properties are derived based on the NASA data-bases for enthalpy, specific heat, and entropy for each chemical species [91]. The thermophysical properties file is typically named "therm.dat" which generally contains the polynomial coefficients representing thermodynamic properties of a selection of molecules. A typical “therm.dat” in OpenFOAM is shown here;

```
THERMO
   300.000  1000.000  5000.000
! GRI-Mech Version 3.0 Thermodynamics released 7/30/99
! NASA Polynomial format for CHEMKIN-II
! see README file for disclaimer
O                 L 1/90O   1               G   200.000  3500.000  1000.000    1
 2.56942078E+00-8.59741137E-05 4.19484589E-08-1.00177799E-11 1.22833691E-15    2
 2.92175791E+04 4.78433864E+00 3.16826710E+00-3.27931884E-03 6.64306396E-06    3
-6.12806624E-09 2.11265971E-12 2.91222592E+04 2.05193346E+00                   4
O2                TPIS89O   2               G   200.000  3500.000  1000.000    1
 3.28253784E+00 1.48308754E-03-7.57966669E-07 2.09470555E-10-2.16717794E-14    2
-1.08845772E+03 5.45323129E+00 3.78245636E+00-2.99673416E-03 9.84730201E-06    3
-9.68129509E-09 3.24372837E-12-1.06394356E+03 3.65767573E+00                   4
H                 L 7/88H   1               G   200.000  3500.000  1000.000    1
 2.50000001E+00-2.30842973E-11 1.61561948E-14-4.73515235E-18 4.98197357E-22    2
 2.54736599E+04-4.46682914E-01 2.50000000E+00 7.05332819E-13-1.99591964E-15    3
 2.30081632E-18-9.27732332E-22 2.54736599E+04-4.46682853E-01                   4
H2                TPIS78H   2               G   200.000  3500.000  1000.000    1
 3.33727920E+00-4.94024731E-05 4.99456778E-07-1.79566394E-10 2.00255376E-14    2
-9.50158922E+02-3.20502331E+00 2.34433112E+00 7.98052075E-03-1.94781510E-05    3
 2.01572094E-08-7.37611761E-12-9.17935173E+02 6.83010238E-01                   4
OH                RUS 78O   1H   1          G   200.000  3500.000  1000.000    1
 3.09288767E+00 5.48429716E-04 1.26505228E-07-8.79461556E-11 1.17412376E-14    2
 3.85865700E+03 4.47669610E+00 3.99201543E+00-2.40131752E-03 4.61793841E-06    3
-3.88113333E-09 1.36411470E-12 3.61508056E+03-1.03925458E-01                   4
H2O               L 8/89H   2O   1          G   200.000  3500.000  1000.000    1
 3.03399249E+00 2.17691804E-03-1.64072518E-07-9.70419870E-11 1.68200992E-14    2
-3.00042971E+04 4.96677010E+00 4.19864056E+00-2.03643410E-03 6.52040211E-06    3
-5.48797062E-09 1.77197817E-12-3.02937267E+04-8.49032208E-01                   4
HO2               L 5/89H   1O   2          G   200.000  3500.000  1000.000    1
 4.01721090E+00 2.23982013E-03-6.33658150E-07 1.14246370E-10-1.07908535E-14    2
 1.11856713E+02 3.78510215E+00 4.30179801E+00-4.74912051E-03 2.11582891E-05    3
-2.42763894E-08 9.29225124E-12 2.94808040E+02 3.71666245E+00                   4
H2O2              L 7/88H   2O   2          G   200.000  3500.000  1000.000    1
 4.16500285E+00 4.90831694E-03-1.90139225E-06 3.71185986E-10-2.87908305E-14    2
-1.78617877E+04 2.91615662E+00 4.27611269E+00-5.42822417E-04 1.67335701E-05    3
-2.15770813E-08 8.62454363E-12-1.77025821E+04 3.43505074E+00                   4
C                 L11/88C   1               G   200.000  3500.000  1000.000    1
 2.49266888E+00 4.79889284E-05-7.24335020E-08 3.74291029E-11-4.87277893E-15    2
 8.54512953E+04 4.80150373E+00 2.55423955E+00-3.21537724E-04 7.33792245E-07    3
-7.32234889E-10 2.66521446E-13 8.54438832E+04 4.53130848E+00                   4
CH                TPIS79C   1H   1          G   200.000  3500.000  1000.000    1
 2.87846473E+00 9.70913681E-04 1.44445655E-07-1.30687849E-10 1.76079383E-14    2
 7.10124364E+04 5.48497999E+00 3.48981665E+00 3.23835541E-04-1.68899065E-06    3
 3.16217327E-09-1.40609067E-12 7.07972934E+04 2.08401108E+00                   4
CH2               L S/93C   1H   2          G   200.000  3500.000  1000.000    1
 2.87410113E+00 3.65639292E-03-1.40894597E-06 2.60179549E-10-1.87727567E-14    2
 4.62636040E+04 6.17119324E+00 3.76267867E+00 9.68872143E-04 2.79489841E-06    3
-3.85091153E-09 1.68741719E-12 4.60040401E+04 1.56253185E+00                   4
CH2(S)            L S/93C   1H   2          G   200.000  3500.000  1000.000    1
 2.29203842E+00 4.65588637E-03-2.01191947E-06 4.17906000E-10-3.39716365E-14    2
 5.09259997E+04 8.62650169E+00 4.19860411E+00-2.36661419E-03 8.23296220E-06    3
-6.68815981E-09 1.94314737E-12 5.04968163E+04-7.69118967E-01                   4
CH3               L11/89C   1H   3          G   200.000  3500.000  1000.000    1
 2.28571772E+00 7.23990037E-03-2.98714348E-06 5.95684644E-10-4.67154394E-14    2
```

```
 1.67755843E+04 8.48007179E+00 3.67359040E+00 2.01095175E-03 5.73021856E-06    3
-6.87117425E-09 2.54385734E-12 1.64449988E+04 1.60456433E+00                   4
CH4               L 8/88C   1H   4          G   200.000  3500.000  1000.000    1
 7.48514950E-02 1.33909467E-02-5.73285809E-06 1.22292535E-09-1.01815230E-13    2
-9.46834459E+03 1.84373180E+01 5.14987613E+00-1.36709788E-02 4.91800599E-05    3
-4.84743026E-08 1.66693956E-11-1.02466476E+04-4.64130376E+00                   4
CO                TPIS79C   1O   1          G   200.000  3500.000  1000.000    1
 2.71518561E+00 2.06252743E-03-9.98825771E-07 2.30053008E-10-2.03647716E-14    2
-1.41518724E+04 7.81868772E+00 3.57953347E+00-6.10353680E-04 1.01681433E-06    3
 9.07005884E-10-9.04424499E-13-1.43440860E+04 3.50840928E+00                   4
CO2               L 7/88C   1O   2          G   200.000  3500.000  1000.000    1
 3.85746029E+00 4.41437026E-03-2.21481404E-06 5.23490188E-10-4.72084164E-14    2
-4.87591660E+04 2.27163806E+00 2.35677352E+00 8.98459677E-03-7.12356269E-06    3
 2.45919022E-09-1.43699548E-13-4.83719697E+04 9.90105222E+00                   4
HCO               L12/89H   1C   1O   1     G   200.000  3500.000  1000.000    1
 2.77217438E+00 4.95695526E-03-2.48445613E-06 5.89161778E-10-5.33508711E-14    2
 4.01191815E+03 9.79834492E+00 4.22118584E+00-3.24392532E-03 1.37799446E-05    3
-1.33144093E-08 4.33768865E-12 3.83956496E+03 3.39437243E+00                   4
CH2O              L 8/88H   2C   1O   1     G   200.000  3500.000  1000.000    1
 1.76069008E+00 9.20000082E-03-4.42258813E-06 1.00641212E-09-8.83855640E-14    2
-1.39958323E+04 1.36563230E+01 4.79372315E+00-9.90833369E-03 3.73220008E-05    3
-3.79285261E-08 1.31772652E-11-1.43089567E+04 6.02812900E-01                   4
CH2OH             GUNL93C   1H   3O   1     G   200.000  3500.000  1000.000    1
 3.69266569E+00 8.64576797E-03-3.75101120E-06 7.87234636E-10-6.48554201E-14    2
-3.24250627E+03 5.81043215E+00 3.86388918E+00 5.59672304E-03 5.93271791E-06    3
-1.04532012E-08 4.36967278E-12-3.19391367E+03 5.47302243E+00                   4
CH3O              121686C   1H   3O   1     G   300.00   3000.00   1000.000    1
 0.03770799E+02 0.07871497E-01-0.02656384E-04 0.03944431E-08-0.02112616E-12    2
 0.12783252E+03 0.02929575E+02 0.02106204E+02 0.07216595E-01 0.05338472E-04    3
-0.07377636E-07 0.02075610E-10 0.09786011E+04 0.13152177E+02                   4
CH3OH             L 8/88C   1H   4O   1     G   200.000  3500.000  1000.000    1
 1.78970791E+00 1.40938292E-02-6.36500835E-06 1.38171085E-09-1.17060220E-13    2
-2.53748747E+04 1.45023623E+01 5.71539582E+00-1.52309129E-02 6.52441155E-05    3
-7.10806889E-08 2.61352698E-11-2.56427656E+04-1.50409823E+00                   4
C2H               L 1/91C   2H   1          G   200.000  3500.000  1000.000    1
 3.16780652E+00 4.75221902E-03-1.83787077E-06 3.04190252E-10-1.77232770E-14    2
 6.71210650E+04 6.63589475E+00 2.88965733E+00 1.34099611E-02-2.84769501E-05    3
 2.94791045E-08-1.09331511E-11 6.68393932E+04 6.22296438E+00                   4
C2H2              L 1/91C   2H   2          G   200.000  3500.000  1000.000    1
 4.14756964E+00 5.96166664E-03-2.37294852E-06 4.67412171E-10-3.61235213E-14    2
 2.59359992E+04-1.23028121E+00 8.08681094E-01 2.33615629E-02-3.55171815E-05    3
 2.80152437E-08-8.50072974E-12 2.64289807E+04 1.39397051E+01                   4
C2H3              L 2/92C   2H   3          G   200.000  3500.000  1000.000    1
 3.01672400E+00 1.03302292E-02-4.68082349E-06 1.01763288E-09-8.62607041E-14    2
 3.46128739E+04 7.78732378E+00 3.21246645E+00 1.51479162E-03 2.59209412E-05    3
-3.57657847E-08 1.47150873E-11 3.48598468E+04 8.51054025E+00                   4
C2H4              L 1/91C   2H   4          G   200.000  3500.000  1000.000    1
 2.03611116E+00 1.46454151E-02-6.71077915E-06 1.47222923E-09-1.25706061E-13    2
 4.93988614E+03 1.03053693E+01 3.95920148E+00-7.57052247E-03 5.70990292E-05    3
-6.91588753E-08 2.69884373E-11 5.08977593E+03 4.09733096E+00                   4
C2H5              L12/92C   2H   5          G   200.000  3500.000  1000.000    1
 1.95465642E+00 1.73972722E-02-7.98206668E-06 1.75217689E-09-1.49641576E-13    2
 1.28575200E+04 1.34624343E+01 4.30646568E+00-4.18658892E-03 4.97142807E-05    3
-5.99126606E-08 2.30509004E-11 1.28416265E+04 4.70720924E+00                   4
C2H6              L 8/88C   2H   6          G   200.000  3500.000  1000.000    1
 1.07188150E+00 2.16852677E-02-1.00256067E-05 2.21412001E-09-1.90002890E-13    2
-1.14263932E+04 1.51156107E+01 4.29142492E+00-5.50154270E-03 5.99438288E-05    3
-7.08466285E-08 2.68685771E-11-1.15222055E+04 2.66682316E+00                   4
CH2CO             L 5/90C   2H   2O   1     G   200.000  3500.000  1000.000    1
 4.51129732E+00 9.00359745E-03-4.16939635E-06 9.23345882E-10-7.94838201E-14    2
-7.55105311E+03 6.32247205E-01 2.13583630E+00 1.81188721E-02-1.73947474E-05    3
 9.34397568E-09-2.01457615E-12-7.04291804E+03 1.22156480E+01                   4
HCCO              SRIC91H   1C   2O   1     G   300.00   4000.00   1000.000    1
 0.56282058E+01 0.40853401E-02-0.15934547E-05 0.28626052E-09-0.19407832E-13    2
 0.19327215E+05-0.39302595E+01 0.22517214E+01 0.17655021E-01-0.23729101E-04    3
 0.17275759E-07-0.50664811E-11 0.20059449E+05 0.12490417E+02                   4
HCCOH              SRI91C   2O   1H   2     G   300.000  5000.000  1000.000    1
 0.59238291E+01 0.67923600E-02-0.25658564E-05 0.44987841E-09-0.29940101E-13    2
 0.72646260E+04-0.76017742E+01 0.12423733E+01 0.31072201E-01-0.50866864E-04    3
 0.43137131E-07-0.14014594E-10 0.80316143E+04 0.13874319E+02                   4
H2CN               41687H   2C   1N   1     G   300.00   4000.000  1000.000    1
```

```
 0.52097030E+01 0.29692911E-02-0.28555891E-06-0.16355500E-09 0.30432589E-13    2
 0.27677109E+05-0.44444780E+01 0.28516610E+01 0.56952331E-02 0.10711400E-05    3
-0.16226120E-08-0.23511081E-12 0.28637820E+05 0.89927511E+01                   4
HCN               GRI/98H   1C   1N   1     G   200.000  6000.000  1000.000    1
 0.38022392E+01 0.31464228E-02-0.10632185E-05 0.16619757E-09-0.97997570E-14    2
 0.14407292E+05 0.15754601E+01 0.22589886E+01 0.10051170E-01-0.13351763E-04    3
 0.10092349E-07-0.30089028E-11 0.14712633E+05 0.89164419E+01                   4
HNO               And93 H   1N   1O   1     G   200.000  6000.000  1000.000    1
 0.29792509E+01 0.34944059E-02-0.78549778E-06 0.57479594E-10-0.19335916E-15    2
 0.11750582E+05 0.86063728E+01 0.45334916E+01-0.56696171E-02 0.18473207E-04    3
-0.17137094E-07 0.55454573E-11 0.11548297E+05 0.17498417E+01                   4
N                 L 6/88N   1             G   200.000  6000.000  1000.000    1
 0.24159429E+01 0.17489065E-03-0.11902369E-06 0.30226245E-10-0.20360982E-14    2
 0.56133773E+05 0.46496096E+01 0.25000000E+01 0.00000000E+00 0.00000000E+00    3
 0.00000000E+00 0.00000000E+00 0.56104637E+05 0.41939087E+01                   4
NNH               T07/93N   2H   1          G   200.000  6000.000  1000.000    1
 0.37667544E+01 0.28915082E-02-0.10416620E-05 0.16842594E-09-0.10091896E-13    2
 0.28650697E+05 0.44705067E+01 0.43446927E+01-0.48497072E-02 0.20059459E-04    3
-0.21726464E-07 0.79469539E-11 0.28791973E+05 0.29779410E+01                   4
N2O               L 7/88N   2O   1          G   200.000  6000.000  1000.000    1
 0.48230729E+01 0.26270251E-02-0.95850874E-06 0.16000712E-09-0.97752303E-14    2
 0.80734048E+04-0.22017207E+01 0.22571502E+01 0.11304728E-01-0.13671319E-04    3
 0.96819806E-08-0.29307182E-11 0.87417744E+04 0.10757992E+02                   4
NH                And94 N   1H   1          G   200.000  6000.000  1000.000    1
 0.27836928E+01 0.13298430E-02-0.42478047E-06 0.78348501E-10-0.55044470E-14    2
 0.42120848E+05 0.57407799E+01 0.34929085E+01 0.31179198E-03-0.14890484E-05    3
 0.24816442E-08-0.10356967E-11 0.41880629E+05 0.18483278E+01                   4
NH2               And89 N   1H   2          G   200.000  6000.000  1000.000    1
 0.28347421E+01 0.32073082E-02-0.93390804E-06 0.13702953E-09-0.79206144E-14    2
 0.22171957E+05 0.65204163E+01 0.42040029E+01-0.21061385E-02 0.71068348E-05    3
-0.56115197E-08 0.16440717E-11 0.21885910E+05-0.14184248E+00                   4
NH3               J 6/77N   1H   3          G   200.000  6000.000  1000.000    1
 0.26344521E+01 0.56662560E-02-0.17278676E-05 0.23867161E-09-0.12578786E-13    2
-0.65446958E+04 0.65662928E+01 0.42860274E+01-0.46605230E-02 0.21718513E-04    3
-0.22808887E-07 0.82638046E-11-0.67417285E+04-0.62537277E+00                   4
NO                RUS 78N   1O   1          G   200.000  6000.000  1000.000    1
 0.32606056E+01 0.11911043E-02-0.42917048E-06 0.69457669E-10-0.40336099E-14    2
 0.99209746E+04 0.63693027E+01 0.42184763E+01-0.46389760E-02 0.11041022E-04    3
-0.93361354E-08 0.28035770E-11 0.98446230E+04 0.22808464E+01                   4
NO2               L 7/88N   1O   2          G   200.000  6000.000  1000.000    1
 0.48847542E+01 0.21723956E-02-0.82806906E-06 0.15747510E-09-0.10510895E-13    2
 0.23164983E+04-0.11741695E+00 0.39440312E+01-0.15854290E-02 0.16657812E-04    3
-0.20475426E-07 0.78350564E-11 0.28966179E+04 0.63119917E+01                   4
HCNO              BDEA94H   1N   1C   1O   1G   300.000  5000.000  1382.000    1
 6.59860456E+00 3.02778626E-03-1.07704346E-06 1.71666528E-10-1.01439391E-14    2
 1.79661339E+04-1.03306599E+01 2.64727989E+00 1.27505342E-02-1.04794236E-05    3
 4.41432836E-09-7.57521466E-13 1.92990252E+04 1.07332972E+01                   4
HOCN              BDEA94H   1N   1C   1O   1G   300.000  5000.000  1368.000    1
 5.89784885E+00 3.16789393E-03-1.11801064E-06 1.77243144E-10-1.04339177E-14    2
-3.70653331E+03-6.18167825E+00 3.78604952E+00 6.88667922E-03-3.21487864E-06    3
 5.17195767E-10 1.19360788E-14-2.82698400E+03 5.63292162E+00                   4
HNCO              BDEA94H   1N   1C   1O   1G   300.000  5000.000  1478.000    1
 6.22395134E+00 3.17864004E-03-1.09378755E-06 1.70735163E-10-9.95021955E-15    2
-1.66599344E+04-8.38224741E+00 3.63096317E+00 7.30282357E-03-2.28050003E-06    3
-6.61271298E-10 3.62235752E-13-1.55873636E+04 6.19457727E+00                   4
NCO               EA 93 N   1C   1O   1     G   200.000  6000.000  1000.000    1
 0.51521845E+01 0.23051761E-02-0.88033153E-06 0.14789098E-09-0.90977996E-14    2
 0.14004123E+05-0.25442660E+01 0.28269308E+01 0.88051688E-02-0.83866134E-05    3
 0.48016964E-08-0.13313595E-11 0.14682477E+05 0.95504646E+01                   4
CN                HBH92 C   1N   1          G   200.000  6000.000  1000.000    1
 0.37459805E+01 0.43450775E-04 0.29705984E-06-0.68651806E-10 0.44134173E-14    2
 0.51536188E+05 0.27867601E+01 0.36129351E+01-0.95551327E-03 0.21442977E-05    3
-0.31516323E-09-0.46430356E-12 0.51708340E+05 0.39804995E+01                   4
HCNN              SRI/94C   1N   2H   1     G   300.000  5000.000  1000.000    1
 0.58946362E+01 0.39895959E-02-0.15982380E-05 0.29249395E-09-0.20094686E-13    2
 0.53452941E+05-0.51030502E+01 0.25243194E+01 0.15960619E-01-0.18816354E-04    3
 0.12125540E-07-0.32357378E-11 0.54261984E+05 0.11675870E+02                   4
N2                121286N   2             G   300.000  5000.000  1000.000    1
 0.02926640E+02 0.14879768E-02-0.05684760E-05 0.10097038E-09-0.06753351E-13    2
-0.09227977E+04 0.05980528E+02 0.03298677E+02 0.14082404E-02-0.03963222E-04    3
 0.05641515E-07-0.02444854E-10-0.10208999E+04 0.03950372E+02                   4
```

```
AR                120186AR  1              G   300.000  5000.000  1000.000    1
 0.02500000E+02 0.00000000E+00 0.00000000E+00 0.00000000E+00 0.00000000E+00    2
-0.07453750E+04 0.04366000E+02 0.02500000E+02 0.00000000E+00 0.00000000E+00    3
 0.00000000E+00 0.00000000E+00-0.07453750E+04 0.04366000E+02                   4
C3H8              L 4/85C   3H   8          G   300.000  5000.000  1000.000    1
 0.75341368E+01 0.18872239E-01-0.62718491E-05 0.91475649E-09-0.47838069E-13    2
-0.16467516E+05-0.17892349E+02 0.93355381E+00 0.26424579E-01 0.61059727E-05    3
-0.21977499E-07 0.95149253E-11-0.13958520E+05 0.19201691E+02                   4
C3H7              L 9/84C   3H   7          G   300.000  5000.000  1000.000    1
 0.77026987E+01 0.16044203E-01-0.52833220E-05 0.76298590E-09-0.39392284E-13    2
 0.82984336E+04-0.15480180E+02 0.10515518E+01 0.25991980E-01 0.23800540E-05    3
-0.19609569E-07 0.93732470E-11 0.10631863E+05 0.21122559E+02                   4
CH3CHO            L 8/88C   2H   4O   1     G   200.000  6000.000  1000.000    1
 0.54041108E+01 0.11723059E-01-0.42263137E-05 0.68372451E-09-0.40984863E-13    2
-0.22593122E+05-0.34807917E+01 0.47294595E+01-0.31932858E-02 0.47534921E-04    3
-0.57458611E-07 0.21931112E-10-0.21572878E+05 0.41030159E+01                   4
CH2CHO            SAND86O   1H   3C   2     G   300.000  5000.000  1000.000    1
 0.05975670E+02 0.08130591E-01-0.02743624E-04 0.04070304E-08-0.02176017E-12    2
 0.04903218E+04-0.05045251E+02 0.03409062E+02 0.10738574E-01 0.01891492E-04    3
-0.07158583E-07 0.02867385E-10 0.15214766E+04 0.09558290E+02                   4
END
```

Appendix B: RSM Computer Code

This appendix provides the RSM optimization code used in the current study. The code is written in C++ under Linux and it contains algorithms, libraries, and functions used in the RSM technique. Also, it displays how RSM connects to OpenFOAM to call the necessary CFD solvers, libraries, or dictionaries.

```
#include <iostream>
#include <vector>
#include <fstream>
#include <time.h>
#include <stdio.h>     //Sina
#include <cmath>       //Sina
#include <math.h>      //Sina
#include <stdlib.h>    //Sina
#include <unistd.h>    //Sina
#include <iomanip>     //Sina
#include <string>      //Sina
#include <omp.h>       //Sina
#include <cstring>     //Sina
#include <string.h>    //Sina

using std::cout;       //Sina
using std::endl;       //Sina
//#include <fstream.h> //Sina
//#include <vectorop.h>       //Sina
//#include <string>    //Sina
//#include <cstdlib>   //Sina
//#include <Cantera.h>
//#include <onedim.h>
//#include <IdealGasMix.h>
//#include <equilibrium.h>
//#include <transport.h>

using namespace std;
//using namespace Cantera;

double Combust(vector<double> vars);
int RSM();
int RSM2D();
double GRGM(vector<double> &x0, double f0, double alpha, double lcon, double rcon, int dim);
double norm(vector<double> x);
double inprod(vector<double>x, vector<double>y);
vector<double> Mv(double **M,vector<double> v,double row, double col);
double **MM(double **M1, double **M2, unsigned int a, double b, unsigned int c);// Sina "double
a" and "double c" --> "unsigned int a" and "unsigned int c"
vector<double> LUSolve(double **M,vector<double> v);
double sum(vector<double> x);
double newfun(vector<double> x, vector<double> b, double &g, double &H);
double newfun2D(vector<double> x, vector<double> b, vector<double> &grad, double **&Hess);
double fun2(double x, int maxIter);
double fun2D(double x,double y, int maxIter);
double **MT(double **M, unsigned int n, unsigned int p); // Sina "double n" and "double p" -->
"unsigned int n" and "unsigned int p"
double max(double x, double y);
double min(double x, double y);
int sign(double x);
double f;
double fnew;
int CFDiter(int no, int maxIter);     //Sina
double readUnoSwirl ();       //Sina
vector<double> g;
vector<double> gnew;
```

```
vector<double> xnew;
int IDvariable;               //Sina
int maxIter;   //Sina

int main()
{
        //Setup Timer
        time_t start;
        time_t end;
        double duration;
        time (&start);
/*
//********Sina******** (it had been better defined as a seperate function)
//Checking the number of the private processors
  int argc;
  char *argv[256];
  int id, nthreads;
  #pragma omp parallel private(id)
  {
    id = omp_get_thread_num();
    printf("Hello Sina from thread %d\n", id);
    #pragma omp barrier
    if ( id == 0 ) {
      nthreads = omp_get_num_threads();
    printf("There are %d threads\n\n",nthreads);
    }
  }
//********************
*/
        //int dude=RSM();      //*****
        int dude=RSM2D();      //*****

        //End Timer
        time (&end);
        duration = difftime (end,start)/3600.0;
        cout<< "duration = " << duration <<" hours"<< endl ;

        //system("PAUSE");     Sina
        return(0);
}

int RSM()
{
        vector<double> x0(1);
        vector<double> xprev(1);

        float fi;      //Sina
        cout << endl;
        printf ("Enter the initial point in the model region: ");
        scanf ("%f",&fi);      // Sina Maximum Point*****
        x0[0]=fi;
        cout<<"x0[0]:  " <<fi<<endl;  //Sina
        cout << endl;

        printf ("Choose the Design Variable: ");
        cout << endl;
        printf ("1 = Quarl Angle ");
        cout << endl;
        printf ("2 = Swirl Number ");
        cout << endl;
        scanf ("%d",&IDvariable);
        cout << endl;

        //x0[0]=20;
        xprev[0]=x0[0];
        //Selecting starting points
        vector<double> points(5);     //*****
        points[0]=0.36;        //Sina  //*****
        points[1]=0.46;
        points[2]=0.56;
        points[3]=0.66;
```

```
        points[4]=0.76;
        double alpha=(points[4]-points[0])/4.0;      //*****
        //Initialize quantities
        double length=points.size();
        vector<double> f(length);
        double diff1=10000;
        double diff2=10000;
        int minloc=0;  //*****
        int maxloc=4;  //*****
        double lcon=0.1;  //Left Constraint   *****
        double rcon=0.8;   //Right Constraint *****
        int count=0;        //Added to control number of shrinks
        double error;
        int no = 0;    //Number of Optimization iteration
        int maxIter;
        int maxCFD();

        maxIter = maxCFD();

        f[0]=fun2(points[0], maxIter); //objective function evaluation for 1-D case       *****
        cout<<endl<<"Point = " <<points[0]<<"     "<<"Function = " <<f[0]<<endl<<endl;

        system("cp -r constant iterations/");// Sina
        system("cp -r system/controlDict iterations/system");// Sina
        system("cp -r system/fvSchemes iterations/system");// Sina
        system("cp -r system/fvSolution iterations/system");// Sina
        system("cp -r 0 iterations/");// Sina

        //Loop until the model and function have same value or the model points solution is the
same
        while (/*(diff1>0.0001) & */(diff2>0.02)) // ***** I believe diff1 is not necessary Sina,
for angle diff2 could be "1"
        {
                for (int i=1;i<length;i++)    //i should be equal to 1  Sina
                {
                        f[i]=fun2(points[i], maxIter);
                        cout<<endl<<endl<<points[i]<<"   "<<f[i]<<endl<<endl;
                }

                //Perform Least Squares fit
                //double n=length;     //Sina
                unsigned int n = length;      //Sina
                int k=1; //This is equal to the number of variables; *****
                //double p=2*k+1;  //Number of regressor variable.  Need to change for higher
order.  Sina    *****
                unsigned int p=2*k+1;  //Sina

                double **X;
                X=new double* [n];
                for (int i=0;i<n;i++)
                {
                        *(X+i)=new double[p];
                }

                for (int i=0;i<n;i++)
                {
                        for (int c=0;c<k;c++)
                        {
                                X[i][c]=1;
                                X[i][c+1]=points[i];
                                X[i][c+k+1]=pow(points[i],2);
                        }
                }
                double **Xt;
                Xt=new double* [p];
                for (int i=0;i<p;i++)
                {
                        *(Xt+i)=new double[n];
                }
                Xt=MT(X,n,p);
```

```
double **A;
A=new double* [p];
for (int i=0;i<p;i++)
{
	*(A+i)=new double[p];
}
A=MM(Xt,X,p,n,p);
vector<double> c(p);
c=Mv(Xt,f,p,n);

vector<double> b(p);
b=LUSolve(A,c);
//Perform Newtons Method
double g;
double H;
double fmin=newfun(x0,b,g,H);
if (H<0)
{
	double g1;
	double H1;
	vector<double> p(1);
	p[0]=points[minloc];
	double b1=newfun(p,b,g1,H1);
	double g2;
	double H2;
	p[0]=points[maxloc];
	double b2=newfun(p,b,g2,H2);
	if (b1<=b2)
	{
		fmin=b1;
		x0[0]=points[minloc];
	}
	else
	{
		fmin=b2;
		x0[0]=points[maxloc];
	}
}
else
{
	double d=-g/H;
	double temp;  //Added to stop solver from going beyond model range.
	temp=x0[0]+d;
	if (temp<points[minloc])
	{
		x0[0]=points[minloc];
	}
	else if (temp>points[maxloc])
	{
		x0[0]=points[maxloc];
	}
	else
	{
		x0[0]=x0[0]+d;
		if (count<3)
		{
			alpha=alpha/2;//Shrinks if convex and inside box
			count=count+1;
		}
		else
		{
			xprev[0]=x0[0];
		}
	}
	fmin=newfun(x0,b,g,H);
}
//cout<<"Next Step:  "<<x0[0]<<endl<<endl;   //Sina*****
double fnew=fun2(x0[0], maxIter);

int ignore=0;
if (fnew>f[0])
```

```
		{
			x0[0]=points[0];
			fnew=f[0];
			xprev[0]=x0[0];
			ignore=1;
		}


		//Error Stuff
		if (ignore==0)
		{
			vector<double> xm(p);
			xm[0]=1;
			xm[1]=x0[0];
			xm[2]=pow(x0[0],2);
			vector<double> yhat(n);
			yhat=Mv(X,b,n,p);
			vector<double> fsurf(n);
			for (int i=0;i<n;i++)
			{
				vector<double> node(1);
				node[0]=points[0];
				fsurf[i]=pow((f[i]-yhat[i]),2)/(n-p);
			}
			double temp=sum(fsurf);
			double s=sqrt(temp);
			vector<double> error1(p);
			error1=LUSolve(A,xm);
			error=inprod(xm,error1);
			error=s*2.919986*sqrt(error);  //The number is for 90% confidence from
students t
		}

		cout<<endl<<endl<<"Next Point: "<<x0[0]<<"  Objective Function:
"<<fnew<<endl<<endl;

//--------------Sina  Copying CFD results due to the last iteration into Opt. iteration folder---
-------------

		char optIt [50], cfdIt [50], cfdIt2 [50], copyCons [50] ;
		char copySys1 [50], copySys2 [50], copySys3 [50], copyZero [50];
		int iter,ndTime;
		no = no+1;

		sprintf(optIt,"mkdir -p iterations/0%d", no);
		system (optIt);
		iter = CFDiter(no, maxIter);

		sprintf(cfdIt,"cp -r %d iterations/0%d/", iter, no);
		system (cfdIt);
		sprintf(cfdIt2,"cp -r %d iterations/0%d/", iter, no+1);
		system (cfdIt2);
		sprintf(copyCons,"cp -r constant iterations/0%d/", no);
		system (copyCons);
		sprintf(copySys1,"cp -r system/controlDict iterations/0%d/system/", no);
		system (copySys1);
		sprintf(copySys2,"cp -r system/fvSchemes iterations/0%d/system/", no);
		system (copySys2);
		sprintf(copySys3,"cp -r system/fvSolution iterations/0%d/system/", no);
		system (copySys3);
		sprintf(copyZero,"cp -r 0 iterations/0%d", no);
		system (copyZero);
//-----------------------------------------------------------------------------------------------
-------------


		//Update loop ending parameters
		diff1=sqrt(pow((fnew-fmin)/fnew,2));
		vector<double> xtemp(1);
		xtemp[0]=x0[0]-xprev[0];
		xprev[0]=x0[0];
```

```
diff2=norm(xtemp);
cout<<"Difference between the last two O. F.:  " <<diff1<<endl;
cout<<"Difference between the last two points:   "<<diff2<<endl;

f[0]=fnew;
//Update points
points[0]=x0[0];
if (x0[0]<(lcon+2*alpha))
{
        if (x0[0]<=(lcon+alpha))
        {
                if (x0[0]<=lcon)
                {
                        if (count<3)
                        {
                                alpha=alpha/2;//Shrink if on min edge
                                count=count+1;
                        }
                        points[0]=lcon;
                        points[1]=points[0]+alpha;
                        points[2]=points[1]+alpha;
                        points[3]=points[2]+alpha;
                        points[4]=points[3]+alpha;
                        minloc=0;
                        maxloc=4;
                }
                else
                {
                        points[1]=lcon;
                        points[2]=points[0]+alpha;
                        points[3]=points[2]+alpha;
                        points[4]=points[3]+alpha;
                        minloc=1;
                        maxloc=4;
                }
        }
        else
        {
                points[1]=lcon;
                points[2]=points[0]-alpha;
                points[3]=points[0]+alpha;
                points[4]=points[3]+alpha;
                minloc=1;
                maxloc=4;
        }
}
else if (x0[0]>(rcon-2*alpha))
{
        if (x0[0]>=(rcon-alpha))
        {
                if (x0[0]>=rcon)
                {
                        if (count<3)
                        {
                                alpha=alpha/2;//Shrink if on max edge
                                count=count+1;
                        }
                        points[0]=rcon;
                        points[1]=points[0]-alpha;
                        points[2]=points[1]-alpha;
                        points[3]=points[2]-alpha;
                        points[4]=points[3]-alpha;
                        minloc=4;
                        maxloc=0;
                }
                else
                {
                        points[1]=rcon;
                        points[2]=points[0]-alpha;
                        points[3]=points[2]-alpha;
                        points[4]=points[3]-alpha;
```

```
                                        minloc=4;
                                        maxloc=1;
                                }
                        }
                        else
                        {
                                points[1]=rcon;
                                points[2]=points[0]+alpha;
                                points[3]=points[0]-alpha;
                                points[4]=points[3]-alpha;
                                minloc=4;
                                maxloc=1;
                        }
                }
                else
                {
                        points[1]=points[0]-alpha;
                        points[2]=points[1]-alpha;
                        points[3]=points[0]+alpha;
                        points[4]=points[3]+alpha;
                        minloc=2;
                        maxloc=4;
                }
        }
        cout<<"x*="<<x0[0]<<" with fmax="<<f[0]<<" +/-"<<error<<endl;
        return(0);
}


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


int RSM2D()
{
        vector<double> x0(2);
        vector<double> xprev(2);

        float fi0;      //Sina
        float fi1;      //Sina
        cout << endl;
        printf ("Enter the initial point for Quarl Angle: ");
        scanf ("%f",&fi0);      // Sina *****

        cout << endl;
        printf ("Enter the initial point for Swirl Number: ");
        scanf ("%f",&fi1);      // Sina *****
        cout << endl;

        x0[0]=fi0;      //Sina
        x0[1]=fi1;      //Sina

        //x0[0]=0.0;    //*****Sina 1.445
        //x0[1]=0.0;    //*****Sina 0.88
        xprev[0]=x0[0];
        xprev[1]=x0[1];         //Sina ***** "-" --> "="
                cout<<"xprev[0]:  " <<xprev[0]<<endl; //Sina
                cout<<"xprev[1]:  " <<xprev[1]<<endl; //Sina
        //Selecting starting points
        double **points;
        points=new double* [9];
        for (int i=0;i<9;i++)
        {
                *(points+i)=new double[2];
        }
        points[0][0]=5.0607;    //*****Sina 1.37
        points[1][0]=5.0607;    //*****Sina 1.37
```

```
        points[2][0]=5.0607;   //*****Sina 1.37
        points[3][0]=10.0607;  //*****Sina 1.445
        points[4][0]=10.0607;  //*****Sina 1.445
        points[5][0]=10.0607;  //*****Sina 1.445
        points[6][0]=15.0607;  //*****Sina 1.52
        points[7][0]=15.0607;  //*****Sina 1.52
        points[8][0]=15.0607;  //*****Sina 1.52

        points[0][1]=0.5;      //*****Sina .87
        points[1][1]=0.7;      //*****Sina .88
        points[2][1]=0.8;      //*****Sina .89
        points[3][1]=0.5;      //*****Sina .87
        points[4][1]=0.7;      //*****Sina .88
        points[5][1]=0.8;      //*****Sina .89
        points[6][1]=0.5;      //*****Sina .87
        points[7][1]=0.7;      //*****Sina .88
        points[8][1]=0.8;      //*****Sina .89

        double alpha1=sqrt(pow((points[3][0]-points[0][0]),2));     //*****Sina 0.075  ?
        double alpha2=sqrt(pow((points[1][1]-points[0][1]),2));     //*****Sina 0.01   ?

        //Initialize quantities
        double length=9;
        vector<double> f(length);
        double diff1=10000;
        double diff2=10000;
        double diff2x=10000;
        double diff2y=10000;
        double lcon=  2;       //*****Sina 0.69
        double rcon= 30;       //*****Sina 1.52
        double bcon=0.1;       //*****Sina 0.865
        double tcon=0.8;       //*****Sina 0.95
        double error;
        int flagl=0;
        int flagr=0;
        int flagb=0;
        int flagt=0;
        int count1=0;
        int count2=0;
        int nshrinks=4;
        int noshrinkh=0;
        int noshrinkv=0;
        int start=0;

        int no = 0;    //Number of Optimization iteration    Sina
        int maxIter;   //Sina
        int maxCFD();  //Sina

        maxIter = maxCFD();    //Sina

        cout<< "approaching for the start point, (x,y) = " << points[0][0] << " , " <<
points[0][1] << endl;  //Sina
        f[0]=fun2D(points[0][0],points[0][1], maxIter);

        cout<<endl<<"Point = " <<points[0][0]<<" , "<<points[0][1]<<"     "<<"Function = "
<<f[0]<<endl<<endl;
        system("cp -r constant iterations/");// Sina
        system("cp -r system/controlDict iterations/system");// Sina
        system("cp -r system/fvSchemes iterations/system");// Sina
        system("cp -r system/fvSolution iterations/system");// Sina
        system("cp -r 0 iterations/");// Sina

        //Loop until the model points solution is the same
        while (/*(diff1>0.0001) & (diff2>0.001)*/ (diff2x>1) || (diff2y>0.02))     //*****Sina
        {
                for (int i=1;i<length;i++)
                {
                        cout<< "approaching the point; (x,y) = " << points[i][0] << " , " <<
points[i][1] << endl;  //Sina
                        f[i]=fun2D(points[i][0],points[i][1], maxIter);
```

```
                        cout<<endl<<"Point = "<<points[i][0]<<" , "<<points[i][1]<<"
"<<"Function = "<<f[i]<<endl<<endl;
                }
                //Perform Least Squares fit
                //double n=length;     //Sina
                unsigned int n = length;
                int k=2;
                //double p=2*k+2;      //Sina ???
                unsigned int p=2*k+2;  //Sina

                double **X;
                X=new double* [n];
                for (int i=0;i<n;i++)
                {
                        *(X+i)=new double[p];
                }

                for (int i=0;i<n;i++)
                {
                        X[i][0]=1;
                        for (int c=0;c<k;c++)
                        {
                                X[i][c+1]=points[i][c];
                                X[i][c+k+1]=pow(points[i][c],2);
                        }
                        for (int c=0;c<k-1;c++)
                        {
                                for (int j=c;j<k;j++)
                                {
                                        X[i][c+2*k+1]=points[i][c]*points[i][j];
                                }
                        }
                }

                double **Xt;
                Xt=new double* [p];
                for (int i=0;i<p;i++)
                {
                        *(Xt+i)=new double[n];
                }
                Xt=MT(X,n,p);

                double **A;
                A=new double* [p];
                for (int i=0;i<p;i++)
                {
                        *(A+i)=new double[p];
                }
                A=MM(Xt,X,p,n,p);
                vector<double> c(p);
                c=Mv(Xt,f,p,n);

                vector<double> b(p);
                b=LUSolve(A,c);
                //Perform Newtons Method
                vector<double> grad(2);
                double **Hess;
                Hess=new double* [2];
                for (int i=0;i<2;i++)
                {
                        *(Hess+i)=new double[2];
                }
                double fmin=newfun2D(x0,b,grad,Hess);
                vector<double> d(2);
                vector<double> gneg(2);
                gneg[0]=-grad[0];
                gneg[1]=-grad[1];
                double **Hes;
                Hes=new double* [2];
                for (int i=0;i<2;i++)
                {
```

```
        *(Hes+i)=new double[2];
}
Hes[0][0]=Hess[0][0];
Hes[0][1]=Hess[0][1];
Hes[1][0]=Hess[1][0];
Hes[1][1]=Hess[1][1];
d=LUSolve(Hes,gneg);

//New code added to deal with indefinite Hessians.  Gaurentees descent.
double LS=inprod(gneg,d);
if (LS<0)
{
        vector<double> dprime(2);
        dprime[0]=sqrt(pow(gneg[0],2));
        dprime[1]=sqrt(pow(gneg[1],2));
        double alpha1prime=min(alpha1,min(rcon-x0[0],x0[0]-lcon));
        double alpha2prime=min(alpha2,min(tcon-x0[1],x0[1]-bcon));

        if ((alpha1prime==rcon-x0[0])&(sign(gneg[0])==-1))
        {
                alpha1prime=alpha1;
        }
        else if ((alpha1prime==x0[0]-lcon)&(sign(gneg[0])==1))
        {
                alpha1prime=alpha1;
        }
        if ((alpha2prime==tcon-x0[1])&(sign(gneg[1])==-1))
        {
                alpha2prime=alpha2;
        }
        else if ((alpha2prime==x0[1]-bcon)&(sign(gneg[1])==1))
        {
                alpha2prime=alpha2;
        }

        vector<double> dnew(2);
        double gamma1=atan(alpha2prime/alpha1prime);
        double gamma2=atan(dprime[1]/dprime[0]);
        if (gamma2>=gamma1)
        {
                dnew[0]=dprime[0]*alpha2prime/dprime[1];
                dnew[1]=alpha2prime;
                d[0]=sign(gneg[0])*dnew[0];
                d[1]=sign(gneg[1])*dnew[1];
        }
        else
        {
                dnew[0]=alpha1prime;
                dnew[1]=dprime[1]*alpha1prime/dprime[0];
                d[0]=sign(gneg[0])*dnew[0];
                d[1]=sign(gneg[1])*dnew[1];
        }
}
else
{
        vector<double> dprime(2);
        dprime[0]=sqrt(pow(d[0],2));
        dprime[1]=sqrt(pow(d[1],2));
        double alpha1prime=min(alpha1,min(rcon-x0[0],x0[0]-lcon));
        double alpha2prime=min(alpha2,min(tcon-x0[1],x0[1]-bcon));

        if ((alpha1prime==rcon-x0[0])&(sign(d[0])==-1))
        {
                alpha1prime=alpha1;
        }
        else if ((alpha1prime==x0[0]-lcon)&(sign(d[0])==1))
        {
                alpha1prime=alpha1;
        }
        if ((alpha2prime==tcon-x0[1])&(sign(d[1])==-1))
        {
```

```
                alpha2prime=alpha2;
        }
        else if ((alpha2prime==x0[1]-bcon)&(sign(d[1])==1))
        {
                alpha2prime=alpha2;
        }

        vector<double> dnew(2);
        if ((dprime[0]>alpha1prime)|(dprime[1]>alpha2prime))
        {
                double gamma1=atan(alpha2prime/alpha1prime);
                double gamma2=atan(dprime[1]/dprime[0]);
                if (gamma2>=gamma1)
                {
                        dnew[0]=dprime[0]*alpha2prime/dprime[1];
                        dnew[1]=alpha2prime;
                        d[0]=sign(d[0])*dnew[0];
                        d[1]=sign(d[1])*dnew[1];
                }
                else
                {
                        dnew[0]=alpha1prime;
                        dnew[1]=dprime[1]*alpha1prime/dprime[0];
                        d[0]=sign(d[0])*dnew[0];
                        d[1]=sign(d[1])*dnew[1];
                }
        }
        else
        {
                if (count1<nshrinks)
                {
                        if (count2<nshrinks)
                        {
                                alpha1=alpha1/2;
                                alpha2=alpha2/2;
                                count1=count1+1;
                                count2=count2+1;
                        }
                        else
                        {
                                alpha1=alpha1/2;
                                count1=count1+1;
                        }
                }
                else if (count2<nshrinks)
                {
                        alpha2/alpha2/2;
                        count2=count2+1;
                }
                else
                {
                        xprev[0]=x0[0];
                        xprev[1]=x0[1];
                }
        }
}
//Update point
x0[0]=x0[0]+d[0];
x0[1]=x0[1]+d[1];
fmin=newfun2D(x0,b,grad,Hess);
double fnew=fun2D(x0[0],x0[1], maxIter);
cout<<endl<<endl<<x0[0]<<"   "<<x0[1]<<"   "<<fnew<<endl<<endl;

//If new point is worse than previous stop
int ignore=0;
if (fnew>f[0])
{
        if (start==1)
        {
                x0[0]=points[0][0];
                x0[1]=points[0][1];
```

```
                                fnew=f[0];
                                flagt=0;
                                flagb=0;
                                flagl=0;
                                flagr=0;
                                xprev[0]=x0[0];
                                xprev[1]=x0[1];
                                diff2=0;
                                diff2x=0;       //Sina
                                diff2y=0;       //Sina
                                ignore=1;
                        }
                        else
                        {
                                start=1;
                        }
                }

                //Error Calc
                if (ignore==0)
                {
                        vector<double> xm(p);
                        xm[0]=1;
                        xm[1]=x0[0];
                        xm[2]=x0[1];
                        xm[3]=pow(x0[0],2);
                        xm[4]=pow(x0[1],2);
                        xm[5]=x0[0]*x0[1];
                        vector<double> yhat(n);
                        yhat=Mv(X,b,n,p);
                        vector<double> fsurf(n);
                        for (int i=0;i<n;i++)
                        {
                                fsurf[i]=pow((f[i]-yhat[i]),2)/(n-p);
                        }
                        double temp=sum(fsurf);
                        double s=sqrt(temp);
                        vector<double> error1(p);
                        error1=LUSolve(A,xm);
                        error=inprod(xm,error1);
                        error=s*2.919986*sqrt(error);  //The number is for 90% confidence from
students t
                }

                cout<<endl<<endl<<"Next Point: "<<x0[0]<<" , "<<x0[1]<<"  Objective Function:
"<<fnew<<endl<<endl;

//--------------Sina  Copying CFD results due to the last iteration into Opt. iteration folder---
-------------

                char optIt [50], cfdIt [50], cfdIt2 [50], copyCons [50] ;
                char copySys1 [50], copySys2 [50], copySys3 [50], copyZero [50];
                int iter,ndTime;
                no = no+1;

                sprintf(optIt,"mkdir -p iterations/0%d", no);
                system (optIt);
                iter = CFDiter(no, maxIter);

                sprintf(cfdIt,"cp -r %d iterations/0%d/", iter, no);
                system (cfdIt);
                sprintf(cfdIt2,"cp -r %d iterations/0%d/", iter, no+1);
                system (cfdIt2);
                sprintf(copyCons,"cp -r constant iterations/0%d/", no);
                system (copyCons);
                sprintf(copySys1,"cp -r system/controlDict iterations/0%d/system/", no);
                system (copySys1);
                sprintf(copySys2,"cp -r system/fvSchemes iterations/0%d/system/", no);
                system (copySys2);
                sprintf(copySys3,"cp -r system/fvSolution iterations/0%d/system/", no);
                system (copySys3);
```

```
            sprintf(copyZero,"cp -r 0 iterations/0%d", no);
            system (copyZero);
//-------------------------------------------------------------------------------------------------
-------------


            //Update loop ending parameters
            diff1=sqrt(pow((fnew-fmin)/fnew,2));
            vector<double> xtemp(2);
            xtemp[0]=x0[0]-xprev[0];
            xtemp[1]=x0[1]-xprev[1];
            cout<<"x0[0]:  " <<x0[0]<<endl;              //Sina
            cout<<"x0[1]:  " <<x0[1]<<endl;              //Sina
            cout<<"xprev[0]:  " <<xprev[0]<<endl; //Sina
            cout<<"xprev[1]:  " <<xprev[1]<<endl; //Sina
            xprev[0]=x0[0];
            xprev[1]=x0[1];
            diff2=norm(xtemp);
            diff2x = sqrt(pow(xtemp[0],2)); //Sina
            diff2y = sqrt(pow(xtemp[1],2)); //Sina


            cout<<"Difference between the last two O. F.:  " <<diff1<<endl;
            cout<<"Difference between the last two points of X:   "<<diff2x<<endl;
            cout<<"Difference between the last two points of Y:   "<<diff2y<<endl;
            cout<<"diff2 (norm of dX and dY:   "<<diff2<<endl;

            //Check if GRGM needs to be used
            if (diff2<0.0001)
            {
                    if ((flagl==1)|(flagr==1))
                    {
                            fnew=GRGM(x0,fnew,alpha2,bcon,tcon,1);
                            if (count2<nshrinks)
                            {
                                    count2=count2+1;
                                    alpha2=alpha2/2;
                                    if (noshrinkh==0)
                                    {
                                            alpha1=alpha1*2;
                                            noshrinkh=1;
                                    }
                            }
                            else
                            {
                                    xprev[0]=x0[0];
                                    xprev[1]=x0[1];
                            }
                    }
                    else if ((flagb==1)|(flagt==1))
                    {
                            fnew=GRGM(x0,fnew,alpha1,lcon,rcon,0);
                            if (count1<nshrinks)
                            {
                                    count1=count1+1;
                                    alpha1=alpha1/2;
                                    if (noshrinkv==0)
                                    {
                                            alpha2=alpha2*2;
                                            noshrinkv=1;
                                    }
                            }
                            else
                            {
                                    xprev[0]=x0[0];
                                    xprev[1]=x0[1];
                            }
                    }
                    xtemp[0]=x0[0]-xprev[0];
                    xtemp[1]=x0[1]-xprev[1];
                    xprev[0]=x0[0];
```

```
        xprev[1]=x0[1];
        diff2=norm(xtemp);
        cout<<diff1<<"   "<<diff2<<endl;
        //system("PAUSE");
}

f[0]=fnew;
//Update points
points[0][0]=x0[0];
points[0][1]=x0[1];

if (x0[0]<(lcon+alpha1))
{
        if (x0[0]<=lcon)
        {
                if (count1<nshrinks)
                {
                        alpha1=alpha1/2;//Shrink if on left edge.
                        count1=count1+1;
                        flagl=1;
                }
                else
                {
                        flagl=0;
                }
                points[0][0]=lcon;
                points[1][0]=lcon;
                points[2][0]=lcon;
                points[3][0]=points[0][0]+alpha1;
                points[4][0]=points[0][0]+alpha1;
                points[5][0]=points[0][0]+alpha1;
                points[6][0]=points[3][0]+alpha1;
                points[7][0]=points[3][0]+alpha1;
                points[8][0]=points[3][0]+alpha1;
                flagr=0;
                flagb=0;
                flagt=0;
        }
        else
        {
                points[1][0]=points[0][0];
                points[2][0]=points[0][0];
                points[3][0]=lcon;
                points[4][0]=lcon;
                points[5][0]=lcon;
                points[6][0]=points[0][0]+alpha1;
                points[7][0]=points[0][0]+alpha1;
                points[8][0]=points[0][0]+alpha1;
                flagl=0;
                flagr=0;
                flagb=0;
                flagt=0;
                noshrinkh=0;
        }
}
else if (x0[0]>(rcon-alpha1))
{
        if (x0[0]>=rcon)
        {
                if (count1<nshrinks)
                {
                        alpha1=alpha1/2;//Shrink if on right edge.
                        count1=count1+1;
                        flagr=1;
                }
                else
                {
                        flagr=0;
                }
                points[0][0]=rcon;
                points[1][0]=rcon;
```

```
                points[2][0]=rcon;
                points[3][0]=points[0][0]-alpha1;
                points[4][0]=points[0][0]-alpha1;
                points[5][0]=points[0][0]-alpha1;
                points[6][0]=points[3][0]-alpha1;
                points[7][0]=points[3][0]-alpha1;
                points[8][0]=points[3][0]-alpha1;
                flagl=0;
                flagb=0;
                flagt=0;
        }
        else
        {
                points[1][0]=points[0][0];
                points[2][0]=points[0][0];
                points[3][0]=rcon;
                points[4][0]=rcon;
                points[5][0]=rcon;
                points[6][0]=points[0][0]-alpha1;
                points[7][0]=points[0][0]-alpha1;
                points[8][0]=points[0][0]-alpha1;
                flagl=0;
                flagr=0;
                flagb=0;
                flagt=0;
                noshrinkh=0;
        }
}
else
{
        points[1][0]=points[0][0];
        points[2][0]=points[0][0];
        points[3][0]=points[0][0]+alpha1;
        points[4][0]=points[0][0]+alpha1;
        points[5][0]=points[0][0]+alpha1;
        points[6][0]=points[0][0]-alpha1;
        points[7][0]=points[0][0]-alpha1;
        points[8][0]=points[0][0]-alpha1;
        flagl=0;
        flagr=0;
        flagb=0;
        flagt=0;
        noshrinkh=0;
}

if (x0[1]<(bcon+alpha2))
{
        if (x0[1]<=bcon)
        {
                if (count2<nshrinks)
                {
                        alpha2=alpha2/2;//Shrink if on bottom edge.
                        count2=count2+1;
                        flagb=1;
                }
                else
                {
                        flagb=0;
                }
                points[0][1]=bcon;
                points[1][1]=points[0][1]+alpha2;
                points[2][1]=points[1][1]+alpha2;
                points[3][1]=bcon;
                points[4][1]=points[0][1]+alpha2;
                points[5][1]=points[1][1]+alpha2;
                points[6][1]=bcon;
                points[7][1]=points[0][1]+alpha2;
                points[8][1]=points[1][1]+alpha2;
                flagl=0;
                flagr=0;
                flagt=0;
```

```
        }
        else
        {
                points[1][1]=points[0][1]+alpha2;
                points[2][1]=bcon;
                points[3][1]=points[0][1];
                points[4][1]=points[0][1]+alpha2;
                points[5][1]=bcon;
                points[6][1]=points[0][1];
                points[7][1]=points[0][1]+alpha2;
                points[8][1]=bcon;
                flagl=0;
                flagr=0;
                flagb=0;
                flagt=0;
                noshrinkv=0;
        }
}
else if (x0[1]>(tcon-alpha2))
{
        if (x0[1]>=tcon)
        {
                if (count2<nshrinks)
                {
                        alpha2=alpha2/2;//Shrink if on top edge.
                        count2=count2+1;
                        flagt=1;
                }
                else
                {
                        flagt=0;
                }
                points[0][1]=tcon;
                points[1][1]=points[0][1]-alpha2;
                points[2][1]=points[1][1]-alpha2;
                points[3][1]=tcon;
                points[4][1]=points[0][1]-alpha2;
                points[5][1]=points[1][1]-alpha2;
                points[6][1]=tcon;
                points[7][1]=points[0][1]-alpha2;
                points[8][1]=points[1][1]-alpha2;
                flagl=0;
                flagr=0;
                flagb=0;
        }
        else
        {
                points[1][1]=points[0][1]-alpha2;
                points[2][1]=tcon;
                points[3][1]=points[0][1];
                points[4][1]=points[0][1]-alpha2;
                points[5][1]=tcon;
                points[6][1]=points[0][1];
                points[7][1]=points[0][1]-alpha2;
                points[8][1]=tcon;
                flagl=0;
                flagr=0;
                flagb=0;
                flagt=0;
                noshrinkv=0;
        }
}
else
{
        points[1][1]=points[0][1]+alpha2;
        points[2][1]=points[0][1]-alpha2;
        points[3][1]=points[0][1];
        points[4][1]=points[0][1]+alpha2;
        points[5][1]=points[0][1]-alpha2;
        points[6][1]=points[0][1];
        points[7][1]=points[0][1]+alpha2;
```

```
                        points[8][1]=points[0][1]-alpha2;
                        flagl=0;
                        flagr=0;
                        flagb=0;
                        flagt=0;
                        noshrinkv=0;
                }
        }
        cout<<"x*=("<<x0[0]<<","<<x0[1]<<") with fmax="<<f[0]<<"+/-"<<error<<endl;
        return(0);
}

double GRGM(vector<double> &x0, double f0, double alpha, double lcon, double rcon, int dim)
{
        //Selecting starting points
        alpha=alpha/2;
        int stop=0;
        double minloc;
        double maxloc;
        int adim;
        if (dim==1)
        {
                adim=0;
        }
        else
        {
                adim=1;
        }
        vector<double> points(5);
        points[0]=x0[dim];

        //checks which piece of the x vector we are altering and then chooses points
        if (x0[dim]<(lcon+2*alpha))
        {
                if (x0[dim]<=(lcon+alpha))
                {
                        if (x0[dim]<=lcon)
                        {
                                stop=1;
                        }
                        else
                        {
                                points[1]=lcon;
                                points[2]=points[0]+alpha;
                                points[3]=points[2]+alpha;
                                points[4]=points[3]+alpha;
                                minloc=1;
                                maxloc=4;
                        }
                }
                else
                {
                        points[1]=lcon;
                        points[2]=points[0]-alpha;
                        points[3]=points[0]+alpha;
                        points[4]=points[3]+alpha;
                        minloc=1;
                        maxloc=4;
                }
        }
        else if (x0[dim]>(rcon-2*alpha))
        {
                if (x0[dim]>=(rcon-alpha))
                {
                        if (x0[dim]>=rcon)
                        {
                                stop=1;
                        }
                        else
                        {
                                points[1]=rcon;
```

```
                        points[2]=points[0]-alpha;
                        points[3]=points[2]-alpha;
                        points[4]=points[3]-alpha;
                        minloc=4;
                        maxloc=1;
                }
        }
        else
        {
                points[1]=rcon;
                points[2]=points[0]+alpha;
                points[3]=points[0]-alpha;
                points[4]=points[3]-alpha;
                minloc=4;
                maxloc=1;
        }
}
else
{
        points[1]=points[0]-alpha;
        points[2]=points[1]-alpha;
        points[3]=points[0]+alpha;
        points[4]=points[3]+alpha;
        minloc=2;
        maxloc=4;
}

//Initialize quantities
double length=points.size();
double fnew;
vector<double> f(length);
f[0]=f0;
//Loop until the model and function have same value or the model points solution is the
same
while (stop==0)
{
        for (int i=1;i<length;i++)
        {
                if (adim==0)
                {
                        f[i]=fun2D(x0[adim],points[i], maxIter);
                }
                else
                {
                        f[i]=fun2D(points[i],x0[adim], maxIter);
                }
        }
        //Perform Least Squares fit
        //double n=length;     //Sina
        unsigned int n=length; //Sina
        int k=1; //This is equal to the number of variables;
        //double p=2*k+1;  //Number of regressor variable.  Need to change for higher
order.
        unsigned int p=2*k+1;

        double **X;
        X=new double* [n];
        for (int i=0;i<n;i++)
        {
                *(X+i)=new double[p];
        }

        for (int i=0;i<n;i++)
        {
                //Need to add here for higher dimensions
                for (int c=0;c<k;c++)
                {
                        X[i][c]=1;
                        X[i][c+1]=points[i];
                        X[i][c+k+1]=pow(points[i],2);
                }
```

```
}
double **Xt;
Xt=new double* [p];
for (int i=0;i<p;i++)
{
        *(Xt+i)=new double[n];
}
Xt=MT(X,n,p);

double **A;
A=new double* [p];
for (int i=0;i<p;i++)
{
        *(A+i)=new double[p];
}
A=MM(Xt,X,p,n,p);
vector<double> c(p);
c=Mv(Xt,f,p,n);

vector<double> b(p);
b=LUSolve(A,c);
//Perform Newtons Method
double g;
double H;
double fmin=newfun(x0,b,g,H);
if (H<0)
{
        double g1;
        double H1;
        vector<double> p(1);
        p[0]=points[minloc];
        double b1=newfun(p,b,g1,H1);
        double g2;
        double H2;
        p[0]=points[maxloc];
        double b2=newfun(p,b,g2,H2);
        if (b1<=b2)
        {
                fmin=b1;
                x0[dim]=points[minloc];
        }
        else
        {
                fmin=b2;
                x0[dim]=points[maxloc];
        }
}
else
{
        double d=-g/H;
        double temp;  //Added to stop solver from going beyond model range.
        temp=x0[dim]+d;
        if (temp<points[minloc])
        {
                x0[dim]=points[minloc];
        }
        else if (temp>points[maxloc])
        {
                x0[dim]=points[maxloc];
        }
        else
        {
                x0[dim]=x0[dim]+d;
                stop=1;
        }
}
if (adim==0)
{
        fnew=fun2D(x0[adim],x0[dim], maxIter);
}
else
```

```
{
        fnew=fun2D(x0[dim],x0[adim], maxIter);
}
cout<<endl<<endl<<x0[dim]<<"   "<<fnew<<endl<<endl;
//system("PAUSE");
if (fnew>f[0])
{
        x0[dim]=points[0];
        fnew=f[0];
        stop=1;
}

f[0]=fnew;
//Update points
points[0]=x0[dim];
//Added consraints back in and generalized April 14, 2010
if (x0[dim]<(lcon+2*alpha))
{
        if (x0[dim]<=(lcon+alpha))
        {
                if (x0[dim]<=lcon)
                {
                        stop=1;
                }
                else
                {
                        points[1]=lcon;
                        points[2]=points[0]+alpha;
                        points[3]=points[2]+alpha;
                        points[4]=points[3]+alpha;
                        minloc=1;
                        maxloc=4;
                }
        }
        else
        {
                points[1]=lcon;
                points[2]=points[0]-alpha;
                points[3]=points[0]+alpha;
                points[4]=points[3]+alpha;
                minloc=1;
                maxloc=4;
        }
}
else if (x0[dim]>(rcon-2*alpha))
{
        if (x0[dim]>=(rcon-alpha))
        {
                if (x0[dim]>=rcon)
                {
                        stop=1;
                }
                else
                {
                        points[1]=rcon;
                        points[2]=points[0]-alpha;
                        points[3]=points[2]-alpha;
                        points[4]=points[3]-alpha;
                        minloc=4;
                        maxloc=1;
                }
        }
        else
        {
                points[1]=rcon;
                points[2]=points[0]+alpha;
                points[3]=points[0]-alpha;
                points[4]=points[3]-alpha;
                minloc=4;
                maxloc=1;
        }
```

```
                }
                else
                {
                        points[1]=points[0]-alpha;
                        points[2]=points[1]-alpha;
                        points[3]=points[0]+alpha;
                        points[4]=points[3]+alpha;
                        minloc=2;
                        maxloc=4;
                }
        }
        return(fnew);
}
double norm(vector<double> x)
//finds norms of vectors
{
        double length=x.size();
        double sum=0;
        for (int i=0;i<length;i++)
        {
                sum+=x[i]*x[i];
        }
        double norm=sqrt(sum);
        return(norm);
}

double inprod(vector<double>x, vector<double>y)
//inner product of vectors
{
        double length=x.size();
        double ans=0;
        for(int i=0;i<length;i++)
        {
                ans+=x[i]*y[i];
        }
        return(ans);
}

vector<double> Mv(double **M,vector<double> v,double row, double col)
//matrix times a vector
{
        vector<double> ans(row,0);
        for(int i=0;i<row;i++)
        {
                for(int j=0;j<col;j++)
                {
                        ans[i]=ans[i]+M[i][j]*v[j];
                }
        }
        return(ans);
}

double **MM(double **M1, double **M2, unsigned int a, double b, unsigned int c)// Sina "double a"
and "double c" --> "unsigned int a" and "unsigned int c"
//matrix times matrix
{
        //unsigned int a;      //Sina
        double **ans;
        ans=new double* [a];
        for(int i=0;i<a;i++)
        {
                *(ans+i)=new double[c];
        }
        for(int i=0;i<a;i++)
        {
                for(int j=0;j<c;j++)
                {
                        ans[i][j]=0;
                }
        }
        for(int k=0;k<a;k++)
```

```
	{
		for(int i=0;i<c;i++)
		{
			for(int j=0;j<b;j++)
			{
				ans[k][i]=ans[k][i]+M1[k][j]*M2[j][i];
			}
		}
	}
	return(ans);
}

vector<double> LUSolve(double **M,vector<double> v)
//LU decomposion solver
{
	int length=v.size();
	for(int i=0;i<length-1;i++)
	{
		for(int j=i+1;j<length;j++)
		{
			double m=M[j][i]/M[i][i];
			M[j][i]=0;
			for(int k=i+1;k<length;k++)
			{
				M[j][k]=M[j][k]-m*M[i][k];
			}
			M[j][i]=m;
		}
	}
	for(int i=1;i<length;i++)
	{
		for (int j=0;j<i;j++)
		{
			v[i]=v[i]-M[i][j]*v[j];
		}
	}
	v[length-1]=v[length-1]/M[length-1][length-1];
	for (int i=0;i<length-1;i++)
	{
		for (int j=length-1-i;j<length;j++)
		{
			v[length-2-i]=v[length-2-i]-M[length-2-i][j]*v[j];
		}
		v[length-2-i]=v[length-2-i]/M[length-2-i][length-2-i];
	}
	return(v);
}
double sum(vector<double> x)
//add vector components
{
	double add=0;
	double length=x.size();
	for (int i=0;i<length;i++)
	{
		add+=x[i];
	}
	return (add);
}

//----------Sina-------------
	int maxCFD()
	{
	int maxIter;
	printf ("Enter the max. CFD iterations per solution: ");
	scanf ("%d",&maxIter);
	cout << endl;
	return(maxIter);
	}
//---------------------------
```

```
//-------Sina  Reading the startTime, endTime, and writeInterval for the next simulation in
"controlDict" file ---
double readStartT ()
{
   FILE * stFile;

   char itstring1 [300];
   char itstring2 [9];
   char itstring4[] = "startTime       ";
   int startTime;

  stFile = fopen ("system/controlDict","r+");
  if (stFile==NULL) perror ("Error opening controlDict file");
  else
  {
    while (!feof(stFile)) {
        fgets (itstring1 , 17 , stFile);

        if (strcmp (itstring1,itstring4) != 0)       {
                fgets (itstring1 , 17 , stFile);

                if (strcmp (itstring1,itstring4) != 0)       { fgets (itstring1 , 300 , stFile);}
                else {
                fgets (itstring2 , 7 , stFile);
                startTime = atoi (itstring2);
                cout <<" startTime before next run  = " <<startTime<<endl;
                }
        }
        ////fgets (itstring2 , 300 , itFile); Don't use this line it ruins everything!
      }
  }
fclose (stFile);
return(startTime);
}
//-----------------

//-------Sina   Reading the last CFD iteration from "log" file ------

int CFDiter(int no, int maxIter)
{
  FILE * itFile;

   char itstring1 [300];
   char itstring2 [8];
   char itstring3[] = "Time = ";
   int iter;

  itFile = fopen ("log","rb");
  if (itFile==NULL) perror ("Error opening log file");
  else
  {
    while (!feof(itFile)) {
        fgets (itstring1 , 8 , itFile);

        if (strcmp (itstring1,itstring3) != 0)       {
                fgets (itstring1 , 8 , itFile);

                if (strcmp (itstring1,itstring3) != 0)       { fgets (itstring1 , 300 , itFile);}
                else {
                //cout << itstring1 << endl;
                //cout << itstring2 << endl;
                fgets (itstring2 , 8 , itFile);
                }

        }
        ////fgets (itstring2 , 300 , itFile); Don't use this line it ruins everything!
      }
                iter = atoi (itstring2);
                cout << "The Last Iteration = " << iter << endl;
                //getchar();
    fclose (itFile);
```

```
  }
  return (iter);
}

//-------Sina  Writing the startTime, endTime, and writeInterval for the next simulation in
"controlDict" file ---
double updateInitial (int iter, int maxIter)
{
   FILE * itFile;
   int ndTime;

   char itstring1 [300];
   char itstring2 [8];
   char itstring3[] = "endTime         ";
   char itstring4[] = "startTime       ";
   char itstring5[] = "writeInterval   ";

        //int mm = 8;  //Max. iteration per solution
  ndTime = iter + maxIter; //Sina *****
  cout <<" maxIter = " <<maxIter;
  //getchar();

  itFile = fopen ("system/controlDict","r+");
  if (itFile==NULL) perror ("Error opening controlDict file");
  else
  {
    while (!feof(itFile)) {
        fgets (itstring1 , 17 , itFile);

        if (strcmp (itstring1,itstring4) != 0)       {
                fgets (itstring1 , 17 , itFile);

                if (strcmp (itstring1,itstring4) != 0)       { fgets (itstring1 , 300 , itFile);}
                else {
                //cout << itstring1 << endl;
                //cout << itstring2 << endl;
                sprintf (itstring2, "%d", iter);
                cout <<" startTime = " <<iter;
                //char itstring2 []="13";
                fputs (itstring2 , itFile);

                }

        }
        ////fgets (itstring2 , 300 , itFile); Don't use this line it ruins everything!
      }

    fclose (itFile);
  }

  itFile = fopen ("system/controlDict","r+");
  if (itFile==NULL) perror ("Error opening controlDict file");
  else
  {
    while (!feof(itFile)) {
        fgets (itstring1 , 17 , itFile);

        if (strcmp (itstring1,itstring3) != 0)       {
                fgets (itstring1 , 17 , itFile);

                if (strcmp (itstring1,itstring3) != 0)       { fgets (itstring1 , 300 , itFile);}
                else {
                //cout << itstring1 << endl;
                //cout << itstring2 << endl;

                sprintf (itstring2, "%d", ndTime);
                cout <<" endTime = " <<ndTime<<endl;
                //char itstring2 []="13";
                fputs (itstring2 , itFile);

                }
```

```
        }
        ////fgets (itstring2 , 300 , itFile); Don't use this line it ruins everything!
       }


    fclose (itFile);
  }

  itFile = fopen ("system/controlDict","r+");
  if (itFile==NULL) perror ("Error opening controlDict file");
  else
  {
    while (!feof(itFile)) {
        fgets (itstring1 , 17 , itFile);

        if (strcmp (itstring1,itstring5) != 0)       {
                fgets (itstring1 , 17 , itFile);

                if (strcmp (itstring1,itstring5) != 0)       { fgets (itstring1 , 300 , itFile);}
                else {
                //cout << itstring1 << endl;
                //cout << itstring2 << endl;
                sprintf (itstring2, "%d", ndTime);
                cout <<" writeInterval = " <<ndTime<<endl;
                //char itstring2 []="13";
                fputs (itstring2 , itFile);

                }

        }
        ////fgets (itstring2 , 300 , itFile); Don't use this line it ruins everything!
       }

    fclose (itFile);
  }
//getchar();

/*
                FILE * cntDict;
                cntDict = fopen ( "system/controlDict" , "r+" );
                if (cntDict==NULL) perror ("Error opening controlDict file.\n ");
                int ij;
                char buff1[300];
                char buff2[]=";";
                char buff3[50];
                char buff4[50];

                for (ij=1; ij<=28; ij++) {
                        fgets (buff1 , 300 , cntDict);
                }
                fgets (buff1 , 17 , cntDict);
                sprintf (buff3, "%d", iter);
                fputs ( buff3 , cntDict );
                //fputs ( buff2 , cntDict );

                fgets (buff1 , 300 , cntDict);
                fgets (buff1 , 300 , cntDict);
                fgets (buff1 , 300 , cntDict);

                int maxIter = 5;
                ndTime = iter + maxIter; //Sina *****
                sprintf (buff3, "%d", ndTime);
                cout <<" endTime = " <<ndTime;

                fgets (buff1 , 17 , cntDict);
                fputs (buff3 , cntDict );
                //fputs ( buff2 , cntDict );
                getchar();

                fclose(cntDict);
```

```
*/

return (0);
}
//---------------------------------
//-------Sina  reading Ux (inlet air axial velocity) from "UnoSwirl" file---
double readUnoSwirl ()
{
   FILE * UFile;
   double Ux;
   int i;
   char itstring1 [300];
   char itstring2 [9];

  UFile = fopen ("0/noSwirl/U","r+");
  if (UFile==NULL) perror ("Error opening U file from noSwirl directory");
  else
  {
    for (i=1; i<=26; i++) {
        fgets (itstring1 , 300 , UFile);
    }
        fgets (itstring1 , 34 , UFile);
        fgets (itstring2 , 7 , UFile);
        Ux = atof (itstring2);
    fclose (UFile);
  }
return(Ux);
}
//-------------------------

//-------Sina  Writing the new angular veolicty of inlet air on "swirlAndRotationProperties"
file---
double writeOmega (double omega)
{
   FILE * omegaFile;
   int i;
   char itstring1 [300];
   char itstring2 [14];

  omegaFile = fopen ("constant/swirlAndRotationProperties","r+");
  if (omegaFile==NULL) perror ("Error opening swirlAndRotationProperties file");
  else
  {
    for (i=1; i<=20; i++) {
        fgets (itstring1 , 300 , omegaFile);
    }
        fgets (itstring1 , 44 , omegaFile);

        sprintf (itstring2, "%7.3f", omega);
        fputs (itstring2 , omegaFile);
        //getchar();
    fclose (omegaFile);
  }
return(0);
}
//-------------------------

//----------
double newfun(vector<double> x, vector<double> b, double &g, double &H)
//1-D model function
{
       double val=b[0]+b[1]*x[0]+b[2]*pow(x[0],2);
       g=b[1]+2*b[2]*x[0];
       H=2*b[2];
       return(val);
}
double newfun2D(vector<double> x, vector<double> b, vector<double> &grad, double **&Hess)
//2-D model function
{
       double val=b[0]+b[1]*x[0]+b[2]*x[1]+b[3]*pow(x[0],2)+b[4]*pow(x[1],2)+b[5]*x[0]*x[1];
       grad[0]=b[1]+2*b[3]*x[0]+b[5]*x[1];
```

```
        grad[1]=b[2]+2*b[4]*x[1]+b[5]*x[0];
        Hess[0][0]=2*b[3];
        Hess[0][1]=b[5];
        Hess[1][0]=b[5];
        Hess[1][1]=2*b[4];
        return(val);
}

//--------INPUT for 1D Optimization-------- *****
double fun2 (double x, int maxIter)
//combustion function evaluation for 1-D case
{
        //x: Quarl angle or swirl number, eff= ?
cout << "IDvariable =" << IDvariable <<endl;
//----------SINA------------------

// Changing the vertices locations based on the given Quarl angle
      const int linesize = 256;
      char bufferY[linesize], bufferYp[linesize], bufferZ[linesize], bufferZp[linesize];
      FILE* outfile;
        char cc;
        int nn = 0;
        double y, yPositive, z, zPositive, pi, xrad, x1=39.44; //*****
      //ifstream infile("constant/polyMesh/blockMeshDict", ios::in);
        //ofstream outfile ("1000/CO2", ios::out);

//------ x = quarl angle------------
if (IDvariable==1) {
        outfile = fopen ( "constant/polyMesh/blockMeshDict" , "r+" );
        pi=4*atan(1);
        xrad=x*pi/180;
   y = (x1*tan(xrad)+57.86)*cos(2.5*pi/180);
   yPositive = y;
   z = (-1)*(x1*tan(xrad)+57.86)*sin(2.5*pi/180);
   zPositive = (-1)*z;
  sprintf (bufferY, "%6.2f", y);
  sprintf (bufferYp,"%6.2f", yPositive);
  sprintf (bufferZ, "%6.2f", z);
  sprintf (bufferZp,"%5.2f", zPositive);


  if (outfile==NULL) perror ("Error opening blockMeshDict file ");
  if (outfile!=NULL)
  {
    do {
      cc = fgetc (outfile);

      if (cc == '/') {
               cc = fgetc (outfile);
               if (cc == '/') {
                       cc = fgetc (outfile);
                        if (cc == '1') {
                               cc = fgetc (outfile);
                               if (cc == '4') {
                                       cc = fgetc (outfile);
        nn++;
        fseek ( outfile , 12 , SEEK_CUR );
        fputs ( bufferY , outfile );  //fputs(const char*, FILE*)
        //fseek ( outfile , 1 , SEEK_CUR );
        fputs ( bufferZ , outfile );
        }}}}
    } while (cc != EOF && nn != 1 ); //*****

    do {
      cc = fgetc (outfile);
      if (cc == '/') {
               cc = fgetc (outfile);
               if (cc == '/') {
                       cc = fgetc (outfile);
                        if (cc == '4') {
                               cc = fgetc (outfile);
```

```
                                if (cc == '6') {
                                        cc = fgetc (outfile);
        nn++;
        fseek ( outfile , 12 , SEEK_CUR );
        fputs ( bufferYp , outfile );  //fputs(const char*, FILE*)
        //fseek ( outfile , 1 , SEEK_CUR );
        fputs ( bufferZp , outfile );

        }}}}
    } while (cc != EOF && nn != 2 ); //*****

  }

        cout << "The number of the changed points in blockMeshDict are: " << nn << endl;
        fclose (outfile);


        system("/home/administrator/OpenFOAM/OpenFOAM-
1.5/applications/bin/linuxGccDPOpt/blockMesh >logBlockMesh");
}
//---------------------------------


//--------- x = Swirl Number-------
//read ux, omega;
double s1, s2, Utheta, Ux, omega;
char copy [50];
int startTime;

if (IDvariable==2) {
s1 = x;
Ux = readUnoSwirl ();
cout << "Axial velocity of Air = " << Ux << endl;
//getchar();

s2 = (sqrt(1+4*0.795*pow(s1,2))-1)/(2*0.795*s1);
Utheta = (1.5/1.0406)*Ux*s2;
omega = Utheta/(20.97)*482.069;
cout << "Angular velocity = " << omega << endl;
writeOmega (omega);
//getchar();

startTime = readStartT ();

//system("cp -r 0/noSwirl/U 0/U");
sprintf(copy,"cp -r 0/noSwirl/U %d/U", startTime);
system (copy);
system("addSwirlAndRotation >log2");
}

system("/home/administrator/OpenFOAM/administrator-
1.5/applications/bin/linuxGccDPOpt/alternateSteadyReactingFoam >log");
//---------------------------------

        //create a folder for each optimization iteration iteration 1, iteration 2, and so on.


//------Reading the latest CFD iteration from "log" file -------
        int iter, no;
        iter = CFDiter(no, maxIter);
        cout << "iter =" << iter << endl;
//-----------------------------------------

//--------------Sina  updating the startTime, endTime, and writeInterval for the next simulation
in "controlDict" file ---
        //double updateInitial (int iter, int maxIter);
        updateInitial (iter, maxIter);
//-------------------O.F.= Max CO2 concentration--------------

        char unzip [50];
        char buff [50];
```

```
        char myLine[16];
        int i;
        fstream eghra;
        char * pEnd;
        double d1, d2=0.0, averageCO2;
        int noPoints;
        double eff;

        if (iter!=0) {
        sprintf(unzip,"gunzip -c %d/CO2.gz > %d/CO2", iter, iter);
        //s = printf ("%s",iteration);
        system (unzip);
        }
        sprintf(buff,"%d/CO2", iter);

        eghra.open(buff);
        if(eghra.fail()) {
                cout<<"Could not open CO2.\n";
        }

      for (i=1; i<=20; i++) {
                eghra.getline(myLine,256,'\n'); //  get ( char* s, streamsize n, char delim );
        }
        eghra.getline(myLine,256,'\n');
        noPoints = atof ( myLine );
        //cout << "noPoints =" << noPoints<< endl;

        eghra.getline(myLine,256,'\n');
      for (i=1; i<=noPoints; i++) {

                eghra.getline(myLine,256,'\n'); //  get ( char* s, streamsize n, char delim );

                //if(eghra) cout << myLine << endl;

//  d1 = strtod (myLine,&pEnd);
//  d2 = strtod (pEnd,NULL);
        d1 = atof ( myLine );
        d2 = d1 + d2;
        }

        averageCO2 = d2/(double (noPoints)) ;
        cout << endl << "Average of CO2 Concentrtion = " << averageCO2 << endl << endl;
        eghra.close();
        //getchar();

        eff=(1-averageCO2);
//-----------------------------------------------------------------

//-------------------O.F.= Lowest Flame Temperature----------------
        sprintf(unzip,"gunzip -c %d/T.gz >%d/T", iter, iter);
        system(unzip);
        fstream readT;
        char myLine2[16];
        int j;

        sprintf(buff,"%d/T", iter);
        readT.open(buff);

        //cout << "iter =" << iter << endl;
        //getchar();

        if(readT.fail()) {
                cout<<"Could not open T.\n";
        }

      for (j=1; j<=22; j++) {
                readT.getline(myLine2,256,'\n'); //  get ( char* s, streamsize n, char delim );
        }
        double T1, T2=0.0;
      for (j=1; j<=noPoints; j++) {
```

```
                readT.getline(myLine2,256,'\n'); //  get ( char* s, streamsize n, char delim );

//  d1 = strtod (myLine2,&pEnd);
//  d2 = strtod (pEnd,NULL);
        T1 = atof ( myLine2 );
        //cout << "T1 = " << T1 << endl << endl;
        if (T2<=T1)    {T2=T1;}

        }
        cout << "Lowest Max Flame Temperature = " << T2 << endl << endl;

        readT.close();

//-------------------------------------------------------------

        eff = (1-averageCO2); //*****
        //eff = T2;
        return (eff);  //Sina  Objective Function
}

//----------------------------------------

/*---Sina-----
double fun2(double x)
//combustion function evaluation for 1-D case
{
        vector<double> vars(1);
        vars[0]=x/1000;          //Divide by 1000 for pore diameter
        double eff=-Combust(vars); //Added negative for maximization.
        return(eff);
}
*/
//------------

double fun2D(double Vx, double Vy, int maxIter)
//combustion function evaluation for 2-D case
{
        //double eff=100*pow((y-pow(x,2)),2)+pow((1-x),2);  //Rosenbrock    Sina

/*//-----Sina-----
        vector<double> vars(2);       //vars = variables
        vars[0]=x/1000;
        vars[1]=y;
        double eff=-Combust(vars);    //Sina
//--------------
*/

        //Vx: Quarl angle and Vy:swirl number, eff= ?
//----------SINA-----------------

// Changing the vertices locations based on the given Quarl angle
      const int linesize = 256;
      char bufferY[linesize], bufferYp[linesize], bufferZ[linesize], bufferZp[linesize];
      FILE* outfile;
        char cc;
        int nn = 0;
        double y, yPositive, z, zPositive, pi, xrad, x1=39.44; //*****
      //ifstream infile("constant/polyMesh/blockMeshDict", ios::in);
        //ofstream outfile ("1000/CO2", ios::out);

//------ Vx = quarl angle------------
        outfile = fopen ( "constant/polyMesh/blockMeshDict" , "r+" );
        pi=4*atan(1);
        xrad=Vx*pi/180;


   y = (x1*tan(xrad)+57.86)*cos(2.5*pi/180);
   yPositive = y;
   z = (-1)*(x1*tan(xrad)+57.86)*sin(2.5*pi/180);
   zPositive = (-1)*z;
```

```
  sprintf (bufferY, "%6.2f", y);
  sprintf (bufferYp,"%6.2f", yPositive);
  sprintf (bufferZ, "%6.2f", z);
  sprintf (bufferZp,"%5.2f", zPositive);


  if (outfile==NULL) perror ("Error opening blockMeshDict file ");
  if (outfile!=NULL)
  {
    do {
      cc = fgetc (outfile);

      if (cc == '/') {
               cc = fgetc (outfile);
               if (cc == '/') {
                       cc = fgetc (outfile);
                        if (cc == '1') {
                               cc = fgetc (outfile);
                               if (cc == '4') {
                                       cc = fgetc (outfile);
        nn++;
        fseek ( outfile , 12 , SEEK_CUR );
        fputs ( bufferY , outfile );  //fputs(const char*, FILE*)
        //fseek ( outfile , 1 , SEEK_CUR );
        fputs ( bufferZ , outfile );
        }}}}
    } while (cc != EOF && nn != 1 ); //*****

    do {
      cc = fgetc (outfile);
      if (cc == '/') {
               cc = fgetc (outfile);
               if (cc == '/') {
                       cc = fgetc (outfile);
                        if (cc == '4') {
                               cc = fgetc (outfile);
                               if (cc == '6') {
                                       cc = fgetc (outfile);
        nn++;
        fseek ( outfile , 12 , SEEK_CUR );
        fputs ( bufferYp , outfile );  //fputs(const char*, FILE*)
        //fseek ( outfile , 1 , SEEK_CUR );
        fputs ( bufferZp , outfile );

        }}}}
    } while (cc != EOF && nn != 2 ); //*****

  }

        cout << "The number of the changed points in blockMeshDict are: " << nn << endl;
        fclose (outfile);


        system("/home/administrator/OpenFOAM/OpenFOAM-
1.5/applications/bin/linuxGccDPOpt/blockMesh >logBlockMesh");
//---------------------------------


//--------- Vy = Swirl Number-------
//read ux, omega;
double s1, s2, Utheta, Ux, omega;
char copy [50];
int startTime;

s1 = Vy;
Ux = readUnoSwirl ();
cout << "Axial velocity of Air = " << Ux << endl;
//getchar();

s2 = (sqrt(1+4*0.795*pow(s1,2))-1)/(2*0.795*s1);
Utheta = (1.5/1.0406)*Ux*s2;
```

```
omega = Utheta/(20.97)*482.069;
cout << "Angular velocity = " << omega << endl;
writeOmega (omega);
//getchar();

startTime = readStartT ();

//system("cp -r 0/noSwirl/U 0/U");
sprintf(copy,"cp -r 0/noSwirl/U %d/U", startTime);
system (copy);
system("addSwirlAndRotation >log2");

system("/home/administrator/OpenFOAM/administrator-
1.5/applications/bin/linuxGccDPOpt/alternateSteadyReactingFoam >log");
//---------------------------------


//------Reading the latest CFD iteration from "log" file -------
       int iter, no;
       iter = CFDiter(no, maxIter);
       cout << "iter =" << iter << endl;
//-----------------------------------------

//--------------Sina  updating the startTime, endTime, and writeInterval for the next simulation
in "controlDict" file ---
       //double updateInitial (int iter, int maxIter);
       updateInitial (iter, maxIter);
//-------------------O.F.= Max CO2 concentration--------------

       char unzip [50];
       char buff [50];
       char myLine[16];
       int i;
       fstream eghra;
       char * pEnd;
       double d1, d2=0.0, averageCO2;
       int noPoints;
       double eff;

       if (iter!=0) {
       sprintf(unzip,"gunzip -c %d/CO2.gz > %d/CO2", iter, iter);
       //s = printf ("%s",iteration);
       system (unzip);
       }
       sprintf(buff,"%d/CO2", iter);

       eghra.open(buff);
       if(eghra.fail()) {
              cout<<"Could not open CO2.\n";
       }

      for (i=1; i<=20; i++) {
              eghra.getline(myLine,256,'\n'); //  get ( char* s, streamsize n, char delim );
       }
       eghra.getline(myLine,256,'\n');
       noPoints = atof ( myLine );
       //cout << "noPoints =" << noPoints<< endl;

       eghra.getline(myLine,256,'\n');
      for (i=1; i<=noPoints; i++) {

              eghra.getline(myLine,256,'\n'); //  get ( char* s, streamsize n, char delim );

              //if(eghra) cout << myLine << endl;

//  d1 = strtod (myLine,&pEnd);
//  d2 = strtod (pEnd,NULL);
       d1 = atof ( myLine );
       d2 = d1 + d2;
       }
```

```
        averageCO2 = d2/(double (noPoints)) ;
        cout << endl << "Average of CO2 Concentrtion = " << averageCO2 << endl << endl;
        eghra.close();
        //getchar();

        eff=(1-averageCO2);
//-----------------------------------------------------------------

//--------------------O.F.= Lowest Flame Temperature----------------
        sprintf(unzip,"gunzip -c %d/T.gz >%d/T", iter, iter);
        system(unzip);
        fstream readT;
        char myLine2[16];
        int j;

        sprintf(buff,"%d/T", iter);
        readT.open(buff);

        //cout << "iter =" << iter << endl;
        //getchar();

        if(readT.fail()) {
                cout<<"Could not open T.\n";
        }

      for (j=1; j<=22; j++) {
                readT.getline(myLine2,256,'\n'); //  get ( char* s, streamsize n, char delim );
        }
        double T1, T2=0.0;
      for (j=1; j<=noPoints; j++) {

                readT.getline(myLine2,256,'\n'); //  get ( char* s, streamsize n, char delim );

//  d1 = strtod (myLine2,&pEnd);
//  d2 = strtod (pEnd,NULL);
        T1 = atof ( myLine2 );
        //cout << "T1 = " << T1 << endl << endl;
        if (T2<=T1)    {T2=T1;}

        }
        cout << "Lowest Max Flame Temperature = " << T2 << endl << endl;

        readT.close();

//-----------------------------------------------------------------

        eff = (1-averageCO2); //*****
        //eff = averageCO2; //Sina
        //eff = T2;
        return (eff);  //Sina  Objective Function

}

double **MT(double **M, unsigned int n, unsigned int p) // Sina "double n" and "double p" -->
"unsigned int n" and "unsigned int p"
//matrix times its transpose
{
        double **ans;
        ans=new double* [p];
        for(int i=0;i<p;i++)
        {
                *(ans+i)=new double[n];
        }
        for (int i=0;i<p;i++)
        {
                for (int j=0;j<n;j++)
                {
                        ans[i][j]=M[j][i];
                }
        }
        return(ans);
```

```
}
double max(double x, double y)
//max of two numbers
{
        double temp1=sqrt(pow(x,2));
        double temp2=sqrt(pow(y,2));
        double ans;
        if (temp1>=temp2)
        {
                ans=temp1;
        }
        else
        {
                ans=temp2;
        }
        return(ans);
}
double min(double x, double y)
//min of two numbers
{
        double temp1=sqrt(pow(x,2));
        double temp2=sqrt(pow(y,2));
        double ans;
        if (temp1<=temp2)
        {
                ans=temp1;
        }
        else
        {
                ans=temp2;
        }
        return(ans);
}
int sign(double x)
//sign of a number
{
        int ans;
        if (x>=0)
        {
                ans=1;
        }
        else
        {
                ans=-1;
        }
        return(ans);
}
```