TURBULENT COMBUSTION MODELLING OF FUELS USING LES/FDF WITH ISAT AND LS2T

by

Serdar Güryuva

Ph.D, Mechanical Engineering, Boğaziçi University, 2022

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Mechanical Engineering
Boğaziçi University

2022

ACKNOWLEDGEMENTS

This was a voyage I did not ever imagine ending.

Not because I never saw the end but because I enjoyed the struggle at every setback, the success of overcoming it, and reaching distances where no man has gone before.

I couldn't succeed without the support and trust of my thesis supervisor, Assoc. Prof. Hasan Bedir. Thank you for believing in me more than I believed in myself.

Special thanks to Prof. Kunt Atalik and Prof. A. Erhan Aksoylu for their valuable time, support, and advice.

I appreciate the support and understanding of my beloved wife Sevilay over these struggling years. I couldn't have done this without you.

My dear parents... Thank you for never giving up on me.

I acknowledge the support from Bogazici University under grant number 11A06-D1. Computing resources used in this work were provided by the Turkish National Center for High-Performance Computing (UYBHM) under grant number 1001202011. I appreciate Ford Otomotiv Sanayi A.Ş for the computational resources provided.

To my sweet little sunshine, Elif Ece.

ABSTRACT

TURBULENT COMBUSTION MODELLING OF FUELS USING LES/FDF WITH ISAT AND LS2T

Combustion simulations with high fidelity turbulence models and detailed chemistry may suffer from high computational power requirements due to the combined cost of time-scale dissipation and small integration steps. Such a limitation can be avoided by employing a hybrid reaction mechanism reduction method called local self-similarity tabulation (LS2T). LS2T directly solves several dominant species reactions and incorporates the effects of other species on dominant ones by data retrieval from pre-calculated tables. This study is based on the application of the LS2T method to high fidelity 3D combustion simulations with different fuels and combustion physics. The test cases that are selected for demonstration purposes are Sandia Flame-D, the premixed methane combustion, and Sandia Spray-A, the non-premixed n-dodecane combustion. The combustion simulations use large eddy simulation (LES) as turbulence solver and transported probability density function (TPDF) for species transport, to increase the accuracy of the simulation and avoid the use of any additional reaction model. This study is the first demonstrator of LS2T approach application to 3D combustion problem with Sandia Flame-D simulation and it is also the very first Spray-A simulation that is executed using LES and TPDF in a 3D resolved domain. The report consists of wide literature research regarding the LES combustion analyses of both test cases and chemistry reduction techniques applied, a detailed theory behind all the physics solution methods used, the computational methods implemented for the study, results, and discussions of the 0D and 3D simulations. The results show that by the use of the LS2T method, it is possible to have high accuracy and generate results similar to the detailed chemistry while maintaining an acceptable computational effort.

ÖZET

LES/FDF ve LS2T MODELLERİ KULLANILARAK TÜRBÜLANSLI YANMA MODELLEMESİ

Yüksek doğruluklu türbülans modellerine ve ayrıntılı kimyaya sahip yanma simülasyonları, zaman ölçeğindeki yayılım ve küçük integrasyon adımlarının toplam maliyeti nedeniyle ortaya çıkan yüksek hesaplama gücüne gereksinim duyar. Yerel öz-benzerlik tablosu (LS2T) adı verilen bir hibrit reaksiyon mekanizması indirgeme yöntemi kullanılarak hesaplama gücündeki artıştan kaçınılabilir. LS2T, birkaç baskın tür reaksiyonunu doğrudan çözerken ve diğer türlerin baskın olanlar üzerindeki etkilerini önceden hesaplanmış tablolardan veri alarak hesaplar. Bu çalışma, LS2T yönteminin farklı yakıtlar ve yanma fiziği ile yüksek doğrulukta 3B yanma simülasyonlarına uygulanmasına dayanmaktadır. Yöntemi sınamak amacıyla seçilen test koşulları, önceden karıştırılmış metan yanması olan Sandia Flame-D ve önceden karıştırılmamış n-dodekan yanması olan Sandia Spray-A'dır. Yanma simülasyonları, simülasyonun doğruluğunu artırmak ve herhangi bir ilave reaksiyon modelinin kullanılmasını önlemek için türbülans çözücü olarak büyük girdap benzeşim yöntemi (LES) ve türlerin taşınımı için taşınan olasılık yoğunluk işlevi (TPDF) kullanır. Bu çalışma, 3B yanma problemine LS2T yaklaşım uygulamasının Sandia Flame-D simülasyonu kullanılarak yapılmış ilk örneğidir. Aynı zamanda bu çalışma 3B çözümlenmiş bir alanda LES ve TPDF kullanılarak yürütülen ilk Spray-A simülasyonunu da içermektedir. Tez raporu, hem test senaryolarının hem de uygulanan kimya azaltma tekniklerinin LES yanma analizleriyle ilgili geniş literatür araştırmasını, kullanılan tüm fizik çözüm yöntemlerinin ardındaki ayrıntılı bir teoriyi, çalışma için uygulanan hesaplama yöntemlerini, 0B ve 3B analiz sonuçları ve ilgili tartışmalarını içerir. Sonuçlar, LS2T yönteminin kullanılmasıyla, kabul edilebilir bir hesaplama yükü ile yüksek doğruluk elde etmenin ve ayrıntılı kimyaya benzer sonuçlar üretmenin mümkün olduğunu göstermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

a_l	Droplet acceleration
B_0	Wave breakup model size constant
B_1	Wave breakup model time constant
CV_i	Linear interpolated value at each corner point
C_L	Levich constant
C_k	SGS kinetic energy transport model constant
C_{RT}	RT breakup model size constant
C_τ	RT breakup model time constant
C_ϕ	Mixing constant for direct quadrature method of moments
d_i	Distance of corner i for the inverse distance weighing
F	One-point, one-time joint filtered density function
$\mathbf{f^c}$	Computable functions by light species data
$\mathbf{f^{nc}}$	Non-computable functions that are dependent only on heavy species
G	High-pass spatial filter
$\mathbf{g_l^c}$	computable part of $\mathbf{g_l}$, function of only light species
$\mathbf{g_l^{nc}}$	Non-computed part of $\mathbf{g_l}$, function of both light and heavy species
$\mathbf{g_l}$	Function representing light species reaction rates and enthalpy change
$\widetilde{G}_{\alpha n}$	Favre averaged conditional mean composition of species α in n^{th} mode
h	Mixture specific enthalpy
h_α^h	Partial molar enthalpy of heavy species

h^l_α	Partial molar enthalpy of light species
IV	Interpolated value by inverse distance weighing
k	Turbulent kinetic energy
k_{sgs}	Subgrid scale kinetic energy
$\mathbf{k}$	Initial condition vector (p_0, T_0, ϕ)
N_S	Number of chemical species
N_h	Number of heavy species
N_l	Number of major(light) species
Oh	Ohnesorge number
p	Pressure
p_0	Initial pressure
$\mathbf{q}$	Heat flux vector
R	Gas constant
Re	Reynolds number
$S(\psi)$	Filtered reaction source term
S_{pen}	Spray penetration length
T_0	Initial temperature
t	Time
Ta	Taylor number
$\mathbf{u}$	Velocity vector
W	Molecular mass
WF_i	Weight factor of corner i for the inverse distance weighing
$\mathbf{Wi}$	Wiener process
w_n	Weight or height of each peak in probability
We	Weber number
Y_h	Mass fraction of heavy species

Y_l	Mass fraction of light species
Y_α	Mass fraction of species α
$\mathbf{y}_h$	State variable of heavy species concentrations and temperature
$\mathbf{y}_l$	State variable of light species concentrations and temperature
y_d	The dominant variable for LS2T, normalised temperature
z	The power term of corner i for the inverse distance weighing
α	Species index
Γ	Molecular diffusivity
Δ_f	Filter Size
δ	Dirac delta function
ϵ	Turbulent dissipation rate
Λ_{wave}	Wavelength of droplet breakup by wave model
μ	Dynamic viscosity
ρ	Density
ρ_g	Gas density
ρ_l	Liquid density
$\boldsymbol{\tau}$	Shear stress tensor
τ_b	Spray breakup time
ϕ	Composition space variables
ϕ_f	Initial equivalence ratio
$\widetilde{\phi}_{\alpha n}$	Conditional mean composition of species α in n^{th} mode
ψ	Composition space vector
Ω_m	Frequency of mixing within the subgrid
$\dot{\omega}_\alpha$	Chemical reaction rate of species α

$\dot{\omega}_{lh}$	Chemical source term by light and heavy species
$\dot{\omega}_{ll}$	Chemical source term by only light species
$''$	Fluctuating part of a density weighted quantity
$\bar{}$	Resolved part of a physical quantity
$\tilde{}$	Density weighted quantity
$'$	Filtered part of a physical quantity
$\varnothing_p$	Particle diameter

LIST OF ACRONYMS/ABBREVIATIONS

0D	Zero Dimensional
2D	Two Dimensional
3D	Three Dimensional
AFR	Air Fuel Ratio
CCS	Cartesian Coordinate System
CFD	Computational Fluid Dynamics
CMC	Conditional Moment Closure
CMC	Conditional Moment Closure
CPR	Constant Pressure Reactor
CSE	Conditional Source Term Estimation Model
CVR	Constant Volume Reactor
DQMM	Direct Quadrature Method Of Moments
EOS	Equation Of State
FDF	Filtered Density Function
FTT	Flow Trough Time
HRM	Heavily Reduced Chemistry
ICE	Internal Combustion Engine
IDV	Inverse Distance Weighting
IEM	Interaction By Exchange With The Mean
ILDM	Intrinsic Low-Dimensional Manifold
ISAT	In Situ Adaptive Tabulation
KH	Kelvin-Helmholtz
KHRT	Kelvin-Helmholtz And Rayleigh-Taylor
LES	Large Eddy Simulation
LFSS	Local Full Self-Similarity
LLNL	Lawrence Livermore National Laboratory
LPMC	Lagrangian Particle Monte Carlo
LPSS	Local Partial Self-Similarity

LPT	Lagrangian Particle Tracking
LS2T	Local Self-Similarity Tabulation
LSFM	Laminar Steady Flamelet Model
MD	Multi Dimensional
MEPDF	Multi-Environment Pdf
MMC	Multiple Mapping Conditioning
PDE	Partial Differential Equations
PDF	Probability Density Function
PISO	Pressure Implicit With Splitting Of Operator
PoI	Point Of Interest
RANS	Reynolds-Averaged Navier-Stokes
Re	Reynolds Number
RMS	Root Mean Square
RR	Reaction Rate
RT	Rayleigh-Taylor
S2FT	Self-Similar Flamelet Tabulation
SDE	Stochastic Differential Equation
SEMC	Stochastic Eulerian Monte Carlo Field Solution
SGS	Subgrid Scale
SMD	Sauter Mean Diameter
SOI	Start Of Injection
TAB	Taylor Analogy Breakup
TPDF	Transported Probability Density Function
UDF	User-Defined Function
UDM	User-Defined Memory Locations
UDS	User-Defined Scalar

1. INTRODUCTION

1.1. Turbulent Combustion with Large Eddy Simulation

Turbulence and chemistry resolution methods used in a turbulent reactive flow simulation need to be suitable to the physics of the problem to give accurate results, and they also need to provide an accurate solution in an acceptable computational time. The increased computational power provides an opportunity to use high-resolution models such as Large Eddy Simulation (LES) and Transported Probability Density Function (TPDF) [3] with stiff chemistry solutions. To reduce the turnaround time and improve computational efficiency for capacity computing [4] it is necessary to either divide the problem to smaller ones and reduce the size of the solution domain, or reduce the number of the equations to be solved. LES results in better approximations in turbulent reacting flows due to the resolution of scalar mixing at large scales and modelling of the small-scale residuals [5]. Nevertheless, the molecular mixing, which is enhanced by the formation of smaller eddies during eddy breakup process, requires better representation at unresolved small scales by additional modelling for the turbulence-chemistry interaction.

Turbulence-chemistry interaction models applied in simulations of reactive flows are classified as non-stochastic and stochastic models. The non-stochastic models, such as the flamelet models [5–9] and the conditional moment closure Conditional Moment Closure method [10, 11], relate species mass fractions and enthalpy with a scalar, like equivalence ratio. On the other hand, the stochastic models use an ensemble of transported scalars to represent chemical composition. Examples of stochastic methods are TPDF [3, 12–17] and its derivatives such as Probability Density Function (PDF) [18] interacting with Reynolds-averaged Navier-Stokes (RANS) turbulence models, Filtered Density Function (Filtered Density Function) with Sparse-Lagrangian Multiple Mapping Conditioning coupling [19] interacting with LES turbulence model. In the stochastic TPDF method [3, 12–17] proposed by Pope [20] for turbulent reactive flows, all of

the scalar quantities (composition variables) needed to describe the state of the reacting medium are represented by one-point, one-time joint PDF. The TPDF method has advantages, e.g. describing the scalars at subgrid scale (SGS) level and having a closed representation for non-linear chemical sources. However, due to the high number of scalars to be resolved with joint PDF, it is difficult to solve the transport equations with conventional discretization schemes. In order to overcome this difficulty, additional solution methods such as Lagrangian Particle Monte Carlo (Lagrangian Particle Monte Carlo) and Stochastic Eulerian Monte Carlo field solution (Stochastic Eulerian Monte Carlo field solution) methods have been developed. The stochastic Lagrangian Particle Monte Carlo methods [21] are computationally expensive since PDF is represented by particles. The particles advance according to stochastic partial differential equations (PDE) and mean values are calculated by weighted averaging. In contrast, in the Stochastic Eulerian Monte Carlo field solution method, the particles are replaced by Eulerian fields, making the method relatively more computationally efficient. In the Stochastic Eulerian Monte Carlo field solution method, particles are either advanced based on stochastic partial differential equations (PDE) [22, 23] or deterministic Eulerian field methods (multi-environment PDF (multi-environment PDF)). In the deterministic Eulerian field methods (multi-environment PDF), to reduce computational load, the composition PDF is represented by series of delta functions for each species and enthalpy and the probability and conditional mean equations are closed by a Direct Quadrature Method of Moments approach [24]. Since the number of equations to be solved increases with the number of species, the multi-environment PDF approach loses its computational speed advantage as the chemistry gets complicated but, the accuracy of the method increases accordingly.

The application of PDF methods with LES differs from the application with RANS methods by use of a filtered SGS PDF(Filtered Density Function). The filtered SGS PDF represents the one-point and one-time probability of a specific composition's existence within a filter volume [15]. Even though the molecular transport and chemical reactions occur at small and usually unresolved scales of LES, the quantitative advantages and resolution of turbulent energy cascade make LES advantageous for

calculation of turbulent reacting flows.

Use of an appropriate reaction mechanism plays an essential role in accurate calculation of heat release rate and species concentrations. The detailed mechanisms with hundreds of species and thousands of reactions would theoretically give the most accurate results. However, due to the combined cost of time-scales distribution over several orders of magnitude and small integration steps requirement for the solution of stiff equations modeling chemical reactions [25], it is still not practical to solve complex reactive flows with several thousands reactions. The computational cost for the chemistry solver can be reduced either by skeletal reduction techniques [26] and time-scale analysis [27] or by retrieval of pre-calculated data from flamelet-based tabulated chemistry based on key reaction parameters such as equivalence ratio, temperature and pressure [28].

The mechanism reduction is conducted with a sole or combined use of skeletal mechanism reduction and time-scale analysis techniques. The most common skeletal mechanism reduction techniques are sensitivity analysis [29], reaction flow analysis [30], reaction rate analysis [31], direct path flux analysis [32], atomic flux analysis [33] and directed graph methods [34]. The quasi-steady state assumption and the partial equilibrium methods [27] are the conventional time-scale analysis techniques developed for a specific purpose such as prediction of ignition delay at low-temperature conditions [35] or soot formation process [36] of n-dodecane which is used as diesel surrogate.

The flamelet based tabulation approaches such as diffusion flamelet tabulation methods [28, 37], premixed flamelet tabulation methods [38, 39], multi-regime flamelet methods [40–42] and spray flamelet methods [43,44] are developed for a specific combustion regime and applicable to many combustion problems [45–49] with good accuracy but with complete storage and retrieval computational overhead. On the other hand, instead of using flamelet based tabulation or reduced mechanisms during the entire computation, the change in the reaction state vector can be pre-calculated within the composition space and stored in tables for the generation of the chemistry representa-

tion as required. As an example, piece-wise implementation of solution mapping [50] method that uses quadratic polynomial fits, and in-situ adaptive tabulation method [51] that depends on linear approximations of the chemistry using the data retrieved from the tables fall in this category. There is also another model called the intrinsic low-dimensional manifold (ILDM) method that relies on the idea of reaction paths in the high-dimensional state space lie in low-dimensional manifolds as long as the pressure, total enthalpy and atomic element composition are fixed [52]. ILDM is called intrinsic since the only information needed to build this approximation is the Arrhenius-type elementary reaction mechanisms except for the degree of reduction. However, ILDM struggles to handle fuels with the increasing number of C atoms and does not work with very lean or rich mixtures or at low temperatures [53].

An alternative hybrid mechanism reduction method called local self similarity tabulation (local self-similarity tabulation (LS2T)) combines the method of solving primary species (light species) only reactions while retrieving the necessary reaction data of the rest of the species (heavy species) from a fore calculated database [25]. The tabulation is based on the idea that local full self-similarity of reaction thermo-chemical quantities when plotted against normalized temperature (T_n) for the same initial pressure (p_0), temperature (T_0) and equivalence ratio (ϕ_0), as shown by Harstad and Bellan [53] by examining the Lawrence Livermore National Laboratory (LLNL) databases [54]. Reactions partial self-similarity makes it possible to limit the number of species to the quasi-steady light species (with C < 3) and unsteady light species; keep reaction mechanisms belonging only light species and solve them while retrieving heavy species (species other than the light ones) contribution data from table constructed for partial initial condition (p_0,T_0,ϕ_0) space covered to include the effects of heavy species.

A heavily reduced chemistry (HRM) such as LS2T is supposed to give accurate results in turbulent combustion cases due to being free of assumptions to develop the reduced reaction mechanisms [55]. Also, even if laminar regions due to slower reaction and lower heat release rate values exist during combustion, the homogenising effect of the turbulence would ease application of HRM [56]. For testing such reduced mecha-

nism and coupling with the flow, 1D and 2D turbulent flame tests have been performed in the past with methane with on the run HRM [57]. Also, 1D laminar flame testing has been done for n-heptane with pre-calculated HRM [56], and excellent accuracy for OH, CO, CO_2 species have been reported, but two essential recommendations have been given. First, to increase the accuracy of predicting the mass fraction of light species, the transport equations should include all the necessary models (such as diffusion flux terms) to reinstate the effect of heavy species. Second, the assumption of the unity Lewis number may result in incorrect species mass fraction and temperature relationship, so it is not recommended. For the flames, where the Lewis number differs from unity, the heavy species influence on diffusion should be considered by pre-calculating and tabulating the heavy species mass flux data [56]. On the other hand, the assumption of the unity Lewis number is valid for the turbulent flame D since turbulent convection outweighs molecular diffusion [58].

Even though Bellan demonstrated the 1D laminar flame application of LS2T method [56], the application of the method to 3D turbulent combustion problems and methods 3D accuracy and efficiency assessment have not been shown yet.

1.2. Sandia Flame-D Experiment for Partially Premixed Combustion

In literature the Sandia piloted partially mixed CH_4 flames are employed to investigate the interaction between turbulence and chemistry. The flame series, namely flames D, E, and F, are cases with three successively increasing Reynolds numbers, resulting in different extinction characteristics. The extinction in flame D is very rare, flame E shows a significant amount of extinction, and flame F is very close to blowing off [59]. Sandia flame D is partially premixed flame operating at a jet Reynolds number, Re = 22,400, composed of 75% dry air (21% O_2, 79% N_2) and 25% CH_4 by volume. Sandia flame D is open to atmospheric pressure, providing a shorter flame length and better accuracy for scalar data measurement than non-premixed CH_4 flame. Also, the high mixing rate of the flame results in burning with a single reaction zone near the stoichiometric mixture fraction [59]. The dilution of fuel by 75 vol % of air minimizes

the formation of polycyclic aromatic hydrocarbons and soot avoiding the interference with the experimental measurements [8].

Table 1.1. Sandia Flame-D inlet boundary conditions.

Property	**Main Jet**	**Pilot**	**Co-Flow**
Inner Diameter [mm]	7.2	7.7	18.9
Outer Diameter [mm]	0	18.2	
Velocity [m/s]	49.6±2	11.4±0.5	0.9±0.05
Temperature [K]	294	1880±50	291
Y_{CH_4}	0.156		
Y_{O_2}	0.197	0.056	0.233
Y_{CO_2}		0.11	
Y_{H_2O}		0.092	
Y_{N_2}	0.647	0.742	0.767
Y_{OH}		0.0022	
ϕ	3.159	0	0

The partially premixed mixture of flame D is injected from a nozzle of 7.2 mm of diameter (d) with a velocity of 49.6 m/s (±2 m/s) to react with a lean ($\phi = 0.77$) pilot flame. The pilot is assumed to be completely burned, composed of N_2, O_2, H_2O, CO_2 and OH with mass fractions of 0.734, 0.056, 0.092, 0.11, 0.0022 respectively. The pilot flame is fed through an 18.2 mm outer and 7.7 mm inner diameter hole with a velocity of 11.4 m/s (±0.5 m/s). The Pilot is surrounded by a co-flow at 0.9 m/s (±0.05 m/s) [59]. The inlet temperatures for jet, pilot, co-flow and wall temperature for the solution domain are 294 K, 1880 K, 291 K, 953 K [60]. The summary of the inlet boundary conditions can be seen in Table 1.1.

The Sandia Flame D series have been employed in computational fluid dynamics (CFD) simulations by many researchers using LES as turbulence model coupled with wide range of finite rate chemistry models grouped under TPDF [13,14,16,17,19,61–67], Conditional Moment Closure (Conditional Moment Closure) [10,68] and, reduced-order manifold models grouped under steady laminar flamelet [5, 9, 69–72], the Lagrangian

flamelet [73] and the flamelet/progress-variable [6, 74] and others [8, 26, 48, 75–79]. The Table 1.2 lists the studies mainly using LES for turbulence resolution and a derivative of TPDF method, sorted by year. Other publications are also listed in a separate Table 1.3 for information since they were not in focus but glanced in case.

1.3. Sandia Spray-A Experiment for Non-Premixed Combustion

The Sandia closed vessel experiments are composed of spray flow and non-premixed combustion test cases of n-dodecane, n-heptane and other fuels conducted in a constant volume optically accessible cubic combustion chamber. The combustion vessel is developed to represent initial temperature value range between 450 and 1300K with zero to 21% oxygen mole fraction, pressure reaching up to 350 bar resulting in densities of 1 to 60 kg/m^3. The Spray-A test condition has been developed to mimic the combustion of high-pressure n-dodecane spray injections at the low-temperature combustion environment of engines with moderate EGR [80] and minimum NOx emission [81].

In the Sandia Spray-A experiments, preconditioning of test chamber is provided by spark controlled combustion of premixed C_2H_2 (or C_2H_4) gas with sufficient oxygen based on the condition to be modelled. The mixing is homogenized by the use of fans for the environment to reach the target temperature and pressure values enabling an ambient for auto-ignition of spray. In this way, direct-injection engine representative injectors can establish a quasi-stationary lifted turbulent spray flame as long as the spray can be supplied [82, 83]. The injector used in the Spray-A experiment is a single axial-holed injector developed by Bosch for experimental purposes and the fuel used is a single component fuel n-dodecane to provide all the necessary specifications of physical and chemical properties of the fuel [81].

Table 1.2. Sandia Flame-D studies with LES and TDPF methods.

Author	**Year**	**Turb.**	**Chemistry and Interaction Models**	**Reaction Mechanism**
Mustata et al.	2006	LES	Reduced Chemical mechanism with eulerian probability density function	
Raman and Pitsch	2007	LES	Reduced Chemical mechanism with Filtered Density Function	
James et al.	2007	LES	Reduced Chemical mechanism with Filtered Density Function	
Chen	2007	LES	Laminar steady flamelet model (LSFM) with eulerian Filtered Density Function	
Jones and Prasad	2010	LES	Eulerian joint velocity scalar transported PDF	15 R., 19 S. derived from the GRI 3.0
Ge et al	2011	LES	FDF with Sparse-Lagrangian MMC	
Ge, Cleary, and Klimenko	2013	LES	FDF with Sparse-Lagrangian MMC	GRI-3.0: 34 S., 219 R. (NOx excluded)
Jangi et al.	2015	RANS	Lagrangian-Eulerian PDF	16-SP: 16S., 41R.,DRM22: 22S., 104R., GRI 2.11
Zhao	2017	LES	MEPDF	A 19-species mechanism reduced from GRI-Mech 3.0
Sundaram et al.	2016	LES	MMC and sparse-Lagrangian LES	
Kim et al.	2018	LES	MEPDF approach	
Duan et al.	2020	LES	Eulerian Stochastic Field (ESF) model combined with the Flamelet Generated Manifolds (FGM)	GRI 3.0

Table 1.3. Other Sandia Flame-D studies.

Author	**Year**	**Turb.**	**Chemistry and Interaction Models**
Pitsch and Steiner	2000	LES	Lagrangian Flamelet Model with presumed-PDF
H. Steiner and W. K. Bushe	2001	LES	LSFM with Conditional source term estimation model (CSE)
Pitsch	2002	LES	Unsteady flamelet equation model with presumed-PDF
Kempf et. al.	2005	LES	LSFM with presumed-PDF
Sheikih et al.	2005	LES	LSFM with Filtered Density Function
Clayton and Jones	2008	LES	LSFM with presumed-PDF
Ferraris and Wen	2008	LES	LSFM with CSE
Ihme and Pitsch	2008	LES	LSFM with presumed-PDF
Vreman et. al.	2008	LES	Flamelet progress with presumed-PDF
Kemenov and Pope	2011	LES	LSFM based on the quadratic spline app.
Garmory and Mastorakos	2011	LES	Conditional Moment Closure
Hiremath et. al.	2013	LES	Rate-Controlled Constrained Equilibrium
Lysenko et. al.	2014	RANS LES	Perfectly stirred reactor, EDC β-PDF
Elbahloul and Rigopoulos	2015	LES	Rate-Controlled Constrained Equilibrium
Jaravel et al.	2018	LES	LES with Analytically Reduced Chemistries
Stanković	2018	LES	Conditional Moment Closure
Pant et al	2019	LES	LSFM with presumed-PDF to produce a flamelet table
Yang et al.	2019	LES	Finite-rate chemistry FRC-LES and Flamelet/progress-variable (FPV)-LES
Miles and Echekki	2020	LES	FDF tabulation-based kernel density estimation from ODT simulations

The operating conditions for the test case of Spray-A used in the study are given in Table 1.4. The available experimental data for non-reacting Spray-A are liquid and vapor penetration lengths, radial profiles of mean mixture fraction, and mixture-fraction contours and for reacting Spray-A are ignition delay, flame lift-off length, flame propagation images and soot volume in radial direction [84,85]. Since the experimental data is reproduced in different facilities, the common definitions of the global quantities are listed below [85]:

Liquid penetration length: The maximum distance from the nozzle to the farthest axial position with 0.1% liquid volume fraction, averaged over a volume of 1 mm in diameter and 1 mm in axial length.

Vapor penetration length: The maximum distance to the injection location of 0.1% vapor mass fraction.

Ignition delay: The moment when the Favre-averaged OH mass fraction reaches 2% of the maximum in the domain or the moment when the maximum rate of temperature increase in the domain occurs.

Flame Lift-Off Length: The location where the Favre-averaged OH mass fraction reaches 2% of the maximum in the domain.

Break-Up Length: The distance from the injection location until which the droplets reach a stable diameter and no further breakup occurs.

The initial conditions of the combustion chamber for the test conditions of interest are defined as 6 MPa, 900 K filled only with N_2, CO_2, H_2O gases for non-reacting spray flow [81] and also given in Table 1.4.

The Spray-A model in this study employs Eulerian field methods (multi-environment PDF) with Direct Quadrature Method of Moments approach [24] for the

turbulence-chemistry coupling and the solution of the transport Filtered Density Function equations. One of the earliest applications of the filtered TPDF method to LES spray simulations is done by W.P. Jones to an auto ignition of non-premixed gaseous n-heptane because of the TPDF methods ability to consider the effects of slow intermediate and fast reactions on auto ignition [15]. By involvement of the temperature fluctuations in the CFD analysis, it is claimed that inclusion of temperature fluctuations reduces the progress rate of combustion and delays the formation of auto-ignition kernels [86]. The addition of stochastic breakup model [87] for spray atomization modeling with LES provided application of LES-Eulerian PDF (multi-environment PDF) approach to spray combustion simulations [17].

Table 1.4. Spray-A operating conditions.

Ambient gas temperature	900 K
Ambient gas pressure	6.0 MPa
Ambient gas density	22.8 kg/m3
Ambient gas oxygen (by volume)	15% 0_2 (reacting); 0% 0_2 (non-reacting)
Ambient gas molar composition for non-reacting	$O_2 = 0.00$; $N_2 = 89.71$; $CO_2 = 6.52$; $H_2O = 3.77$
Ambient gas molar composition for reacting	$O_2 = 15.00$; $N_2 = 75.15$; $CO_2 = 6.22$; $H_2O = 3.62$
Ambient gas velocity	Near-quiescent, less than 1 m/s
Common rail fuel injector	Bosch solenoid-activated, generation 2.2
Fuel injector diameter	0.0894 mm
Nozzle K factor	$K = (d_{inlet} - d_{outlet})/10$ use $\mu m = 1.5$
Discharge coefficient	Cd = 0.86, using 10 MPa pressure drop and diesel fuel
Spray full included angle	0° (single axial hole)
Fuel injection pressure	150 MPa
Fuel	n-dodecane
Fuel temperature at nozzle	363 K (90°C)
Injection duration	1.5 ms
Injection mass	About 3.5 mg

As the experimental data becomes available at Sandia NL for Spray-A test case [81], it has been used by many researchers for validation of both inert spray flow and combustion of dodecane. The CFD work for Spray-A literature can be divided by classification of the turbulence-chemistry interaction models used. The ones that are classified as non-stochastic interaction models use well stirred reactor [2, 88–90], perfectly stirred reactor [91], multiple representative interactive flamelet [91], presumed β-PDF [92–96], flamelet generated manifold [92, 94, 95, 97, 98], Conditional Moment Closure [96, 99], multiple representative interactive flamelets [100], tabulated flamelet model [101], multi-zone combustion model [102], approximated diffusion flamelet and unsteady flamelet progress variable [103]. The ones that are classified as stochastic methods use TPDF [90,104–106] and sparse-Lagrangian multiple mapping conditioning model with Filtered Density Function [107]. It is interesting to note that three of the TPDF based simulation are done at Ansys Fluent with 2D solution domains using unsteady RANS turbulence model [104–106] and one done at Star-CD [90]. Also, the only study that employs LES and TPDF for Spray-A simulates spray as gas jet in OpenFOAM [107].

Until today, there is no publication that reports use of LES and TPDF with Lagrangian spray representation for simulation of Spray-A test condition. The list and details of the Spray-A related publications can be seen in Table 1.5.

Table 1.5. Sandia Spray-A Simulation Publications

Year	**Author**	**Title**	**CFD Tool**	**Turb. Model**	**Turb. and Chem. Int. Model**	**Reaction Mech**	**Mesh and Time St.**	**Spray Model**
2011	Som et al.	Three Dimensional Simulations of Diesel Sprays Using n-Dodecane as a Surrogate	Converge	URANS	WSR	Som-S103		KH-RT
2012	Ayyapureddi et al.	Application of the FGM Method to Spray A Conditions of the ECN Database	STAR-CD	URANS	Presumed PDF/FGM	Krithika-S253 Lu-S124		Various
2013	Bhattacharjee and Haworth	Simulations of Transient n-Heptane and n-Dodecane Spray Flames Under Engine-Relevant Conditions Using a Transported PDF Method	STAR-CD	URANS	Composition PDF w. Lagrangian Particle vs. WSR	Som-S103	0.3mm, 2.0E-7s	Blob with Reitz Diwakar Break-Up

Table 1.5. Sandia Spray-A Simulation Publications (cont.)

Year	**Author**	**Title**	**CFD Tool**	**Turb. Model**	**Turb. and Chem. Int. Model**	**Reaction Mech**	**Mesh and Time St.**	**Spray Model**
2013	Wehrfritz et. al.	Large Eddy Simulation Of High-Velocity Fuel Sprays Studying Mesh Resolution And Breakup Model Effects For Spray A	Open FOAM	LES	No-Combustion		0.0625mm, 2.0E−8s	Rosin-Ramler Dist. KH-RT and ETAB
2014	Gong et. al.	Large Eddy Simulation of n-Dodecane Spray Combustion in a High Pressure Combustion Vessel	Open FOAM	LES	WSR and CCM	Som-S103	0.25mm, 5.0E-8s	Huh-Gosman and WAVE
2014	D'Errico et. al.	Comparison of Well-Mixed and Multiple Representative Interactive Flamelet Approaches for Diesel Spray Combustion	Open FOAM	URANS	PSR vs mRIF		2D	Huh–Gosman

Table 1.5. Sandia Spray-A Simulation Publications (cont.)

Year	**Author**	**Title**	**CFD Tool**	**Turb. Model**	**Turb. and Chem. Int. Model**	**Reaction Mech**	**Mesh and Time St.**	**Spray Model**
2014	Luo et. al.	Development and Validation of an n-Dodecane Skeletal Mechanism for Spray Combustion Applications	Converge	URANS	WSR	Luo-S105		KH-RT
2015	Pei et. al.	Large Eddy Simulation of a Reacting Spray Flame with Multiple Realizations Under Compression Ignition Engine Conditions	Converge	LES	δ function combustion model	Som-S103	0.0625mm	KH-RT and No-Time Counter Collision
2015	Pei et. al.	Modelling n-Dodecane Spray and Combustion with the Transported Probability Density Function Method	Fluent v14.5	URANS	Composition PDF	Pei-88	2D	No breakup just evaporation

Table 1.5. Sandia Spray-A Simulation Publications (cont.)

Year	**Author**	**Title**	**CFD Tool**	**Turb. Model**	**Turb. and Chem. Int. Model**	**Reaction Mech**	**Mesh and Time St.**	**Spray Model**
2015	Abianeh et. al.	Turbulent Spray Combustion Simulations Based on a New Skeletal Mechanism for n-Dodecane	Converge	LES		Multiple, Abianeh-S85	0.03125 mm, 0.0625 mm av.	
2015	Chishty et. al.	A Numerical Study of 'Spray-A' with Multiple-Injections Using the Transported PDF method	Fluent v14.5	URANS	Composition PDF	Tianfeng Lu-53S	2D	No breakup just evaporation
2015	Farrace et. al.	Analysis of Averaging Methods for Large Eddy Simulations of Diesel Sprays	STAR-CD	LES			0.125mm, 2.0E-7s	Blob and KH-RT
2016	Abianeh et. al.	Determination of Modeled Luminosity-Based and Pressure-Based Ignition Delay Times of Turbulent Spray Combustion	Converge	LES	Multi-Zone Combustion	Multiple, Abianeh-S85	0.03125 mm, 0.0625 mm av.	Blob, KH-RT

Table 1.5. Sandia Spray-A Simulation Publications (cont.)

Year	**Author**	**Title**	**CFD Tool**	**Turb. Model**	**Turb. and Chem. Int. Model**	**Reaction Mech**	**Mesh and Time St.**	**Spray Model**
2016	Blomberg et. al.	Modeling Split Injections of ECN "Spray A" Using a CMC Combustion Model with RANS and LES	STAR-CD	LES	CMC	Yao-S54	0.125 mm	
2016	Pei et. al.	An Analysis of the Structure of an n-Dodecane Spray Flame Using TPDF Modelling	Fluent v14.5	URANS	Composition PDF	Pei-88	2D	
2016	Skeen et. al.	A Progress Review on Soot Experiments and Modeling in ECN	Review					
2016	Wehrfritz	Large Eddy Simulation of n-Dodecane Spray Flames Using Flamelet Generated Manifolds	Open FOAM	LES	FGM	Narayan-255 and Ranzi130	0.0625mm 4E-8s	Rosin-Ramler Dist. KH-RT and ETAB

Table 1.5. Sandia Spray-A Simulation Publications (cont.)

Year	**Author**	**Title**	**CFD Tool**	**Turb. Model**	**Turb. and Chem. Int. Model**	**Reaction Mech**	**Mesh and Time St.**	**Spray Model**
2017	Pandurangi et. al.	Onset and Progression of Soot in High-Pressure n-Dodecane Sprays Under Diesel Engine Conditions	STAR-CD	URANS	β PDF and CMC	Luo-S106	2D	
2017	Davidovic et. al.	LES of n-Dodecane Spray Combustion Using a Multiple Representative Interactive Flamelets Model	CIAO	LES	Multiple Repr. Interactive Flamelets	57S, 217R		Nozzle flow with DNS mapped to Lagrangian.
2017	Desantes et. al.	Experimental Validation and Analysis of Seven Different Chemical Kinetic Mechanisms for n-Dodecane Using a Rapid Compression-Expansion Machine	Chem-kin					

Table 1.5. Sandia Spray-A Simulation Publications (cont.)

Year	**Author**	**Title**	**CFD Tool**	**Turb. Model**	**Turb. and Chem. Int. Model**	**Reaction Mech**	**Mesh and Time St.**	**Spray Model**
2017	Salehi et. al.	Sparse-Lagrangian MMC Simulations of an n-Dodecane Jet at Engine-Relevant Conditions	Open FOAM	LES	TPDF and MMC	Luo-S106		Gas Jet
2017	Kundu et. al.	Importance of Turbulence-Chemistry Interactions at Low Temperature Engine Conditions	Converge	LES	Tabulated Flamelet Model	SOM-S103	0.0625mm	KH-RT, Dynamic drag model, Frossling evaporation
2018	Kahila et. al.	Large-Eddy Simulation on the Influence of Injection Pressure in Reacting Spray-A	Open FOAM	LES	FGM	Ranzi-130	0.0625mm, 4E-8s	Rosin-Ramler with SMD 6mic, KH-RT

Table 1.5. Sandia Spray-A Simulation Publications (cont.)

Year	**Author**	**Title**	**CFD Tool**	**Turb. Model**	**Turb. and Chem. Int. Model**	**Reaction Mech**	**Mesh and Time St.**	**Spray Model**
2019	Zhang et al.	Effects of Turbulence-Chemistry Interactions on Auto-Ignition and Flame Structure for n-Dodecane Spray Combustion	Open FOAM	URANS	FGM w. β-PDF	Narayan-S255	0.25mm, 5E-7s	Blob and KH-RT
2019	Owoyele	Application of Deep Artificial Neural Networks to Multi-Dimensional Flamelet Libraries and Spray Flames	Con-verge	LES	The grouped multi-target ANN	Som-S103	0.0625mm	
2019	Payri et al.	Influence of the n-Dodecane Chemical Mechanism on the CFD Modelling of the Diesel-Like ECN Spray A Flame Structure At Different Ambient Conditions	Open FOAM	URANS	Approximated Diffusion Flamelet and Unsteady Flamelet Progress Variable	Narayan-S255, Yao-S54, Cai-S57, Wang-S100	0.25mm	KH-RT

1.4. The Purpose and the Method of the Study

This study shows the extension of LS2T method for a 3D case, presents the application to the CFD code, explains the effect on flame characteristics at steady and transient flame conditions. Even though Bellan demonstrated the 1D laminar flame application of LS2T method [56], the application of the method to 3D turbulent combustion problems and methods 3D accuracy and efficiency assessment have not been shown yet. The study depends on methodology development for application of LS2T to a 3D combustion CFD model and assessment of methodology by using partially premixed combustion data of Sandia Flame-D experiment [59]. Although methane does not have the appropriate number of carbon atoms and number of reactions to demonstrate the power of the LS2T method, considering the details of the CFD physics, the computational load and necessity to run the same analysis for hundreds of times for testing, it is an efficient choice for implementing the method for 3D LES combustion analysis. In order to improve the quality of the work and test the ability of drastic reduction in the number of species that are operational, the method is modified for n-dodecane combustion and its application to the non-premixed spray combustion of Sandia Spray-A experiment [81].

In both test cases, for the turbulence and chemistry interaction solution, the stochastic composition TPDF [3, 12–17] method is coupled with LES. The composition TPDF is advantageous for describing the scalars at SGS level and having a closed representation for non-linear chemical sources. To overcome the difficulty of solving transport equation with conventional discretization schemes, two different additional solution methods have been employed for Sandia Flame-D and Spray-A due to the limitations in Ansys Fluent v19.2 solver [108]. The transport Filtered Density Function equations are solved with Lagrangian Particle Monte Carlo (Lagrangian Particle Monte Carlo) [21] in Flame-D. Spray-A model employs Eulerian field methods (multi-environment PDF) with Direct Quadrature Method of Moments approach [24] for solution of transport Filtered Density Function equations.

2. THEORY

The notations and formulations used in the publication are based on the review article of Haworth about PDF methods [109] and book of Garnier [110], but some of the variables notations have been changed for the sake of clarity and integrity. Since the transport in molecular level is significantly small compared to turbulent and momentum transport, molecular flux terms are omitted.

2.1. Conservation Equations

The conservation equations for a reacting system with N_S number of species are given in vector notation.

The mass conservation equation is written as follows

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{u} = 0, \tag{2.1}$$

where ρ is density, t is time, $\mathbf{u}$ is the velocity vector and the material derivative is

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla. \tag{2.2}$$

The species conservation equation is

$$\frac{\partial \rho_\alpha}{\partial} + \nabla \cdot (\rho_\alpha \mathbf{u} + \mathbf{j}_\alpha) = \dot{\omega}_\alpha, \tag{2.3}$$

where $\dot{\omega}_\alpha$ is the chemical reaction rate, $\mathbf{j}_\alpha$ is the mass flux and ρ_α is the density of species α. The species density is related to the mixture density through mass fraction Y_α as

$$\rho_\alpha = \rho Y_\alpha. \tag{2.4}$$

The momentum conservation equation is

$$\rho\frac{D\mathbf{u}}{Dt} = -\nabla\cdot\boldsymbol{\tau} - \nabla\cdot p\boldsymbol{\delta}, \tag{2.5}$$

where τ is the stress tensor, p is the pressure.

The energy conservation equation is

$$\rho\frac{Dh}{Dt} = -(\nabla\cdot\mathbf{q}) + \frac{Dp}{Dt} - (\boldsymbol{\tau} : \nabla\mathbf{u}), \tag{2.6}$$

where h is the mixture specific enthalpy which is the sum of formation and sensible entalphies, $\mathbf{q}$ is the heat flux vector. The energy equation can also be written in terms of temperature T as

$$\rho c_v\frac{DT}{Dt} = -(\nabla\cdot\mathbf{q}) - (\boldsymbol{\tau} : \nabla\mathbf{u}) - (p\boldsymbol{\delta} : \nabla\mathbf{u}) + \sum_{\alpha=1}^{N_s} c_{v\alpha}T(\nabla\cdot\mathbf{j}_\alpha - \dot{\omega}_\alpha), \tag{2.7}$$

where c_v is the mixture specific heat at constant volume and $c_{v\alpha}$ is the constant volume specific heat of species α.

The ideal gas equation of state is

$$p = \rho RT, \tag{2.8}$$

where R is the gas constant of the mixture.

2.2. Filtering and Density Weighting for LES

LES depends on the idea of separating a physical quantity Q in the flow domain by a high-pass special filter $G(|\mathbf{x}-\mathbf{y}|)$ (low pass in frequency domain) to solve large structures (larger than computational mesh size in this case) and model the effects of small structures. The filtered quantity defined for a filter width Δ as

$$\bar{Q}(\mathbf{x},t) = \int Q(\mathbf{y},t)G(|\mathbf{x}-\mathbf{y}|)d\mathbf{y}, \tag{2.9}$$

where consistency property shows $\int G(\mathbf{x})d\mathbf{x} = 1$. Any physical quantity $Q(\mathbf{x}, t)$ can be separated to its resolved $\bar{Q}(\mathbf{x}, t)$ and filtered(unresolved) $Q'(\mathbf{x}, t)$ part as

$$Q(\mathbf{x}, t) = \bar{Q}(\mathbf{x}, t) + Q'(\mathbf{x}, t). \tag{2.10}$$

For variable density flows density weighting is also used and formulated as

$$\widetilde{Q}(\mathbf{x}, t) = \overline{\rho Q}/\bar{\rho} = \frac{1}{\bar{\rho}} \int \rho(\mathbf{y}, t) Q(\mathbf{y}, t) G(|\mathbf{x} - \mathbf{y}|) d\mathbf{y} \tag{2.11}$$

allowing a physical quantity to be separated to its density averaged [111] $\widetilde{Q}(\mathbf{x}, t)$(Favre filtered) and fluctuating part $Q''(\mathbf{x}, t)$ as

$$Q(\mathbf{x}, t) = \widetilde{Q}(\mathbf{x}, t) + Q''(\mathbf{x}, t). \tag{2.12}$$

Some of the properties of both filtered and Favre averaged terms are summarized as below to be used for derivation of density weighted and filtered terms of LES:

- The filtered (or density weighted) value of unresolved perturbations is not zero: $\bar{Q}'' \neq 0$.
- Filtered and double filtered (or density weighted) values are not equal: $\bar{\bar{Q}} \neq \bar{Q}$.
- Convolution form of filtering satisfies linearity: $\overline{Q_1 + Q_2} = \bar{Q}_1 + \bar{Q}_2$.
- Filtered value of a derivative is equal to the derivative of filtered value, commutation with differentiation: $\overline{\frac{\partial Q}{\partial x_j}} = \frac{\partial \bar{Q}}{\partial x_j}$ and $\overline{\frac{\partial Q}{\partial t}} = \frac{\partial \bar{Q}}{\partial t}$.
- Filtered and density weighted value of a derivative is not equal to the derivative of filtered and density weighted value: $\widetilde{\frac{\partial Q}{\partial x_j}} \neq \frac{\partial \tilde{Q}}{\partial x_j}$ and $\widetilde{\frac{\partial Q}{\partial t}} \neq \frac{\partial \tilde{Q}}{\partial t}$.
- Change of variables: $\overline{\rho Q} = \bar{\rho}\widetilde{Q}$.

The filters used in LES are separated as implicit where the grid is assumed to be used as the LES low-pass filter or explicit where filters such as box or Gaussian in physical space or cut-off in spectral space is applied to the numerical grid [112]. Since implicit filter is using the full grid for resolution, and it is more advantageous for combustion analyses where the solution domain requires high resolution to catch the flame-front in LES.

2.3. Filtered Conservation Equations for LES

The filtered conservation equations are obtained by Favre filtering the mass, momentum, energy, and species mass conservation equations for a reacting system with N_s number of species ($\alpha = 1, .., N_s$). The filtered mass and momentum conservation equations are

$$\frac{\partial \overline{\rho}}{\partial t} + \nabla \cdot \overline{\rho}\, \widetilde{\mathbf{u}} = 0 \tag{2.13}$$

and

$$\frac{\partial \overline{\rho}\, \widetilde{\mathbf{u}}}{\partial t} + \widetilde{\mathbf{u}} \cdot \nabla(\overline{\rho}\, \widetilde{\mathbf{u}}) + \nabla \overline{p} - \mu \nabla^2 \widetilde{\mathbf{u}} = -\nabla \cdot \widetilde{\boldsymbol{\tau}}^s. \tag{2.14}$$

The unresolved SGS turbulent stress ($\widetilde{\boldsymbol{\tau}}^s$) is divided into its deviatoric ($\widetilde{\boldsymbol{\tau}}^s_d$) and isotropic ($\widetilde{\boldsymbol{\tau}}^s_i$) parts. The isotropic part is either added to the filtered pressure gradient term in the equation or neglected for low Mach number flows. The deviatoric part $\widetilde{\boldsymbol{\tau}}^s_d$ is modelled by employing the Boussinesq hypothesis [113] as

$$\widetilde{\boldsymbol{\tau}}^s_d = -\mu_{sgs} \left(2\overline{\boldsymbol{S}} - \frac{2}{3} \nabla \cdot \widetilde{\mathbf{u}} \right), \tag{2.15}$$

where $\overline{\boldsymbol{S}}$ is the rate of strain tensor for the resolved scale which is defined by

$$\overline{\boldsymbol{S}} = \frac{1}{2} \left(\nabla \widetilde{\mathbf{u}} + \nabla \widetilde{\mathbf{u}}^T \right). \tag{2.16}$$

The eddy viscosity is modelled using the dynamic SGS kinetic energy model which allows transport of sub-grid kinetic energy. In the dynamic SGS kinetic model, the μ_{sgs} is modelled as

$$\mu_{sgs} = C_k\, \overline{\rho}\, \widetilde{k}^{\frac{1}{2}}\, \Delta, \tag{2.17}$$

where the filter size Δ is the cube root of each cell volume, $V^{\frac{1}{3}}$, and C_k is a dynamically calculated model constant. The sub-grid kinetic energy, $\widetilde{k} \equiv \frac{1}{2}(\widetilde{u_i u_i} - \widetilde{u}_i \widetilde{u}_i)$, can be

obtained from the solution of its transport equation [114]

$$\frac{\partial \overline{\rho}\,\widetilde{k}}{\partial t} + \nabla \cdot (\overline{\rho}\,\widetilde{\mathbf{u}}\,\widetilde{k}) = -\widetilde{\boldsymbol{\tau}}^{s} \cdot \nabla \widetilde{\mathbf{u}}^{T} - C_{\epsilon}\,\overline{\rho}\,\frac{\widetilde{k}^{\frac{3}{2}}}{\Delta} + \nabla \cdot \left(\frac{\mu_{sgs}}{\sigma_k}\,\nabla \widetilde{k}\right), \tag{2.18}$$

where C_ϵ is a dynamically calculated model constant, σ_k is set as 1.0 [115].

The filtered species conservation equation is

$$\frac{\partial \overline{\rho}\,\widetilde{Y}_\alpha}{\partial t} + \nabla \cdot (\overline{\rho}\,\widetilde{\mathbf{u}}\,\widetilde{Y}_\alpha) - \widetilde{\dot{\omega}}_\alpha = -\nabla \cdot (\overline{\rho \mathbf{u} Y_\alpha} - \overline{\rho}\widetilde{\mathbf{u}}\widetilde{Y}_\alpha) - \nabla \cdot \tilde{\mathbf{j}}_\alpha, \tag{2.19}$$

where $\widetilde{Y}_\alpha$ is the Favre-filtered species mass fraction, $\widetilde{\dot{\omega}}_\alpha$ is the Favre-filtered chemical source term and $\tilde{\mathbf{j}}_\alpha$ is the filtered molecular mass diffusion term of species α. More will be discussed for the species conservation in Section 2.4.

The filtered energy conservation equation is

$$\begin{aligned} \frac{\partial \overline{\rho}\,\widetilde{c_v T}}{\partial t} + \nabla \cdot (\overline{\rho}\,\widetilde{\mathbf{u}}\,\widetilde{c_v T}) - \overline{p}\,\nabla \cdot \widetilde{\mathbf{u}} + \nabla \cdot \widetilde{\mathbf{q}} - \widetilde{\Phi} = \\ - \left[\nabla \cdot\, c_v \mathbf{Q} + \boldsymbol{\Pi}_{\mathbf{dil}} - \boldsymbol{\epsilon}_v + \nabla \cdot (\overline{\mathbf{q}} - \widetilde{\mathbf{q}})\right], \end{aligned} \tag{2.20}$$

where $\mathbf{Q}$ is the SGS temperature flux defined as

$$\mathbf{Q} \;=\; \overline{\rho}\widetilde{\mathbf{u}T} \;-\; \overline{\rho}\widetilde{\mathbf{u}}\widetilde{T}. \tag{2.21}$$

$\boldsymbol{\Pi}_{\mathbf{dil}}$ is the SGS pressure-dilatation which is defined as

$$\boldsymbol{\Pi}_{\mathbf{dil}} \;=\; \overline{p\,\nabla \cdot \mathbf{u}} \;-\; \overline{p}\,\nabla \cdot \widetilde{\mathbf{u}}. \tag{2.22}$$

The sum of SGS temperature flux and pressure-dilatation terms is modeled as

$$\nabla \cdot\, c_v \mathbf{Q} + \boldsymbol{\Pi}_{\mathbf{dil}} \;=\; -\nabla \left[\frac{c_p\,\overline{\rho}\,\mu_{sgs}}{Pr_{sgs}}\,\nabla \widetilde{T}\right], \tag{2.23}$$

where Pr_{sgs} is the SGS Prandtl number [116]. The SGS viscous dissipation ($\boldsymbol{\epsilon}_v$) is defined and modelled as

$$\boldsymbol{\epsilon}_v \;=\; \overline{\Phi} - \widetilde{\Phi}. \tag{2.24}$$

The equation can be re-written as

$$\boldsymbol{\epsilon_v} = \overline{\boldsymbol{\tau}^s \nabla \mathbf{u}} - \widetilde{\boldsymbol{\tau}^s} \nabla \widetilde{\mathbf{u}}, \tag{2.25}$$

since Φ refers to viscous dissipation.

The filtered heat flux term is defined as

$$\widetilde{\mathbf{q}} = \kappa \nabla \widetilde{T}, \tag{2.26}$$

where κ is the temperature-dependent thermal conductivity.

The filtered ideal gas equation of state is

$$\overline{p} = \overline{\rho}\, R \widetilde{T}. \tag{2.27}$$

2.4. Composition Transport Filtered Density Function

The turbulence and chemistry interaction which takes place at sub-filter levels of LES controls the rate of combustion and can be resolved by statistical approaches. A specified Filtered Density Function for the statistical description of fluctuations, enables the species reaction rate term $\widetilde{\dot{\omega}}_\alpha$ in equation (2.19) to be closed.

The filtered mass density function, F, is defined as

$$F(\boldsymbol{\psi}; \mathbf{x}, t) \equiv \int_{-\infty}^{+\infty} \rho(\mathbf{x}', t) \zeta[\boldsymbol{\psi}, \boldsymbol{\phi}(\mathbf{x}, t)] G(\mathbf{x}' - \mathbf{x}) d\mathbf{x}' = \langle \rho \zeta \rangle, \tag{2.28}$$

where ζ, the fine-grained density is defined as

$$\zeta[\boldsymbol{\psi}, \boldsymbol{\phi}(\mathbf{x}, t)] = \delta[\boldsymbol{\psi} - \boldsymbol{\phi}(\mathbf{x}, t)]. \tag{2.29}$$

$\boldsymbol{\phi}(\mathbf{x}, t)$ is the scalar array, δ is the delta function and $\boldsymbol{\psi}$ is the composition domain of the scalar array. F is the mass-weighted spatially filtered value of the fine-grained

density and due to integral property of the filtered mass density function it is defined as

$$\int_{-\infty}^{+\infty} F(\boldsymbol{\psi}; \mathbf{x}, t) d\boldsymbol{\psi} = \overline{\rho}. \tag{2.30}$$

The transport equation for F is obtained by multiplying the equation for the fine-grained density with the filter function and integrating it over the $\mathbf{x}'$ space. The resulting equation is

$$\frac{\partial F}{\partial t} + \nabla \cdot \widetilde{\mathbf{u}F}|\boldsymbol{\psi} = \frac{\partial}{\partial \boldsymbol{\psi}} \cdot \Big[-\frac{1}{\overline{\rho}} \nabla \cdot \widetilde{\Gamma_{\boldsymbol{\phi}} \nabla \boldsymbol{\phi}}|\boldsymbol{\psi} F \Big] - \frac{\partial}{\partial \boldsymbol{\psi}} \cdot \Big[\mathbf{S}(\boldsymbol{\psi}) F \Big], \tag{2.31}$$

where Γ_ϕ is the molecular diffusivity, $\widetilde{\ }|\boldsymbol{\psi}$ represents the mass-filtered conditional mean of a variable, and $\boldsymbol{\phi}$ represents composition space variables. The convection term is decomposed as

$$\widetilde{\mathbf{u}F}|\boldsymbol{\psi} = \widetilde{\mathbf{u}} F + [\widetilde{\mathbf{u}F}|\boldsymbol{\psi} - \widetilde{\mathbf{u}} F]. \tag{2.32}$$

Although the formulation closes the filtered reaction source term $\mathbf{S}(\boldsymbol{\psi})$, the turbulent scalar flux term (the second term on the right-hand side of equation (2.32)) and the conditional molecular mixing term requires additional models. The turbulent SGS scalar flux term is modelled by gradient-diffusion assumption as

$$\widetilde{\mathbf{u}F}|\boldsymbol{\psi} - \widetilde{\mathbf{u}} F = -\Gamma_{\phi\, sgs} \nabla (F/\overline{\rho}), \tag{2.33}$$

where $\Gamma_{\phi\, sgs}$ is the sub-grid transport coefficient. $\Gamma_{\phi\, sgs}$ is calculated from

$$\Gamma_{\phi\, sgs} = \mu_{sgs} / Sc_{\phi\, sgs}. \tag{2.34}$$

The sub-grid Schmidt number $Sc_{\phi\, sgs}$ is assumed to be constant and equal to Pr_{sgs} [21].

The conditional molecular mixing term (the first term on the right-hand side of equation (2.32)) is closed by a micro-mixing model called interaction by exchange with

the mean model which is defined as

$$\frac{\partial}{\partial \boldsymbol{\psi}} \cdot \Big[-\frac{1}{\overline{\rho}} \nabla \cdot \widetilde{\Gamma_\phi \nabla \boldsymbol{\phi}} | \boldsymbol{\psi} F \Big] = \nabla \cdot \Gamma_\phi \nabla (F/\overline{\rho}) + \frac{\partial}{\partial \boldsymbol{\psi}} \cdot \Big[\Omega_m (\widetilde{\boldsymbol{\phi}} - \boldsymbol{\psi}) F \Big], \tag{2.35}$$

where Ω_m is the frequency of mixing within the sub-grid. Its value is calculated from the following model

$$\Omega_m = C_\Omega (\Gamma_\phi + \Gamma_{\phi\, sgs}) / (\overline{\rho} \Delta_f^2), \tag{2.36}$$

where C_Ω is a model constant and Δ_f is the SGS filter size.

The final form of the Filtered Density Function transport equation becomes

$$\frac{\partial F}{\partial t} + \nabla \cdot \widetilde{\mathbf{u}} F = \nabla \cdot (\Gamma_\phi + \Gamma_{\phi\, sgs}) \nabla (F/\overline{\rho}) + \frac{\partial}{\partial \boldsymbol{\psi}} \cdot \Big[\Omega_m (\widetilde{\boldsymbol{\phi}} - \boldsymbol{\psi}) F \Big] + \frac{\partial}{\partial \boldsymbol{\psi}} \cdot \Big[\mathbf{S}(\boldsymbol{\psi}) F) \Big]. \tag{2.37}$$

The high dimensional Filtered Density Function transport equation can be efficiently solved by either Lagrangian (Lagrangian Particle Monte Carlo) or Eulerian (multi-environment PDF) solution methods. Use of the Lagrangian Particle Monte Carlo method [21] would require high particle density (10-15 per cell) for the Monte Carlo scheme solution, resulting in a linear increase of computational load with the resolution of the domain for LES [117] and with the number of species to be solved. The Eulerian field method (multi-environment PDF) on the other hand, where the composition Filtered Density Function is represented by delta functions for each species and enthalpy, is relatively more efficient since it solves a finite number of moments instead of resolving all moments of the Filtered Density Function. The Lagrangian Particle Monte Carlo method is employed in this study and the molecular diffusivity term Γ_ϕ is solved by a particle mixing model called Euclidean Minimum Spanning Trees [118].

2.4.1. Lagrangian Particle Monte Carlo Method

The Monte Carlo particles are employed in the solution of the Filtered Density Function transport equation. The particles move in physical space by convection and diffusion and have varying compositions due to mixing and reactions. The joint com-

position Filtered Density Function is represented by a series of Monte Carlo particles each with a set of composition space variable $\boldsymbol{\phi}$ and position vector $\mathbf{X}$. The spatial transport of particles is represented by an analogy to the stochastic differential equation and derived by comparison to the Fokker-Planck equation as [21]

$$\frac{\mathrm{d}\mathbf{X}}{\mathrm{d}t} = \left(\widetilde{\mathbf{u}} + \frac{1}{\overline{\rho}}\frac{\partial(\Gamma_\phi + \Gamma_{\phi\, sgs})}{\partial \mathbf{x}}\right) + \left(\sqrt{\frac{2(\Gamma_\phi + \Gamma_{\phi\, sgs})}{\overline{\rho}}}\right)\frac{\mathrm{d}\mathbf{W}_i}{\mathrm{d}t}, \tag{2.38}$$

where $\mathbf{W}_i$ denotes the standard isotropic Wiener process. It should be noted that Wiener process is a stochastic process and used for the solution of a deterministic special transport equation.

The composition space variables sub-grid mixing and reactions terms are implemented as

$$\frac{\mathrm{d}\phi_\alpha}{\mathrm{d}t} = -\Omega_m\left(\phi_\alpha - \tilde{\phi}_\alpha\right) + S_\alpha(\phi). \tag{2.39}$$

In accordance with the principle of equivalent systems [20], the direct integration of equation (2.39) and the direct integration of the Filtered Density Function equation result in the same statistics. The numerical integration process of the Lagrangian equations for each Δt time advancement is composed of two steps [119]. First, a prediction at time Δt/2 for the solution of spatial transport equation (2.38) is made based on the explicit Euler method. At the updated spatial coordinates, the mixing and source term in equation (2.39) is integrated. At the second step, the spatial transport equation (2.38) is integrated for half time step and added to previous spatial displacement.

2.4.2. Multi Environment Eulerian Solution Method

The joint composition Filtered Density Function for N_S+1 scalar (N_S for the number of species and 1 for enthalpy) is represented by series of delta functions$(1, 2...n...N)$ that are characterized by their location in composition space ($\widetilde{\phi}_{\alpha n}$) and weight of each peak (w_n) in probability [117]. The presumed joint composition Filtered Density Func-

tion is written as

$$F_\phi(\boldsymbol{\psi};\mathbf{x},t) = \sum_{n=1}^{N} \omega_n(\mathbf{x},t) \prod_{\alpha=1}^{N_S} \delta(\psi_\alpha - \widetilde{\phi}_{\alpha n}(\mathbf{x},t)), \tag{2.40}$$

where $\boldsymbol{\phi}$ is the composition space variable and δ is the delta function.

The Eulerian Filtered Density Function transport equation is obtained by substituting equation (2.40) to equation (2.37). The unknown weights w_n and locations $\widetilde{\phi}_{\alpha n}$ are determined by

$$\frac{\partial \overline{\rho}\,\omega_n}{\partial t} + \frac{\partial \overline{\rho}\,\widetilde{\mathbf{u}}\,\omega_n}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{x}}\left[(\Gamma_\phi + \Gamma_{\phi\,sgs})\frac{\partial\,\omega_n}{\partial \mathbf{x}}\right] + \overline{\rho} a_n \tag{2.41}$$

and

$$\frac{\partial \overline{\rho}\,\widetilde{G}_{\alpha n}}{\partial t} + \frac{\partial \overline{\rho}\,\widetilde{\mathbf{u}}\,\widetilde{G}_{\alpha n}}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{x}}\left[(\Gamma_\phi + \Gamma_{\phi\,sgs})\frac{\partial\,\widetilde{G}_{\alpha n}}{\partial \mathbf{x}}\right] + \overline{\rho} b_{\alpha n} \tag{2.42}$$

where $\widetilde{G}_{\alpha n} = w_n \widetilde{\phi}_{\alpha n}$ is the species Favre averaged conditional mean composition of n^{th} mode. The a_n and $b_{\alpha n}$ terms include the reaction, mixing by interaction by exchange with the mean (interaction by exchange with the mean) and correction terms.

The mean composition $\widetilde{\phi}_\alpha$ and its variance $\widetilde{\phi}''_\alpha$ are calculated as

$$\widetilde{\phi}_\alpha = \sum_{n=1}^{N} w_n S_\alpha(\widetilde{\phi}_{\alpha n}) \tag{2.43}$$

and

$$\widetilde{\phi''_\alpha} = \sum_{n=1}^{N} S_\alpha(\widetilde{\phi}_{\alpha n})\widetilde{G}_{\alpha n} - \widetilde{\phi}_\alpha. \tag{2.44}$$

For two-mode ($N = 2$) interaction by exchange with the mean-Direct Quadrature Method of Moments (Direct Quadrature Method of Moments) terms are

$$a_1 = a_2 = 0, \tag{2.45}$$

$$b_{\alpha 1} = \frac{1}{\widetilde{\phi}_{\alpha 1} - \widetilde{\phi}_{\alpha 2}} \sum_{n=1}^{2} w_n \Gamma_{\phi\, sgs} (\nabla \widetilde{\phi}_{\alpha n})^2 + \Omega_m (\omega_1 \widetilde{G}_{\alpha 2} - \omega_2 \widetilde{G}_{\alpha 1}) + \omega_1 S_\alpha (\widetilde{\phi}_{\alpha 1}) \qquad (2.46)$$

and

$$b_{\alpha 2} = \frac{-1}{\widetilde{\phi}_{\alpha 1} - \widetilde{\phi}_{\alpha 2}} \sum_{n=1}^{2} w_n \Gamma_{\phi\, sgs} (\nabla \widetilde{\phi}_{\alpha n})^2 - \Omega_m (\omega_1 \widetilde{G}_{\alpha 2} - \omega_1 \widetilde{G}_{\alpha 1}) + \omega_2 S_\alpha (\widetilde{\phi}_{\alpha 2}). \qquad (2.47)$$

The first term is the correction term to ensure realizability and boundedness, the second term is the mixing by interaction by exchange with the mean term, and the last term is the net reaction term S_α of species α [24, 117].

2.5. Local Self-Similarity Tabulation Method

LS2T is a chemistry reduction method and a combination of local partial self-similarity, local full self-similarity and a simple tabulation scheme to retrieve the necessary missing information to calculate the chemical kinetics accurately. The LS2T method is applicable to auto-ignition and oxidation of complex fuel species (such as n-dodecane, n-heptane, etc.) at constant pressure (constant mass) [120] or constant volume [25] reactors. The idea of the method is focusing on approximately 20 dominant intermediate or final species and calculating the reactions that are composed only of major light species (with C < 3) O, CH, CH_2, CH_3, HO, HCO, HO_2, HC_2, C_2H_3 and unsteady light species H_2O_4, CO_2, O_2, H, H_2, CO, CH_4, H_2O_2, C_2H_2, C_2H_4, CH_2O, and modelling the effect of the remaining species (heavy species) using the partial or full self-similarity characteristics [25]. Most of the major light species are common to all hydrocarbon combustion [121].

The initial value problem of the vector $\mathbf{y}_l$ representing light species concentrations and temperature for the N_m dimensional reduced model is

$$\frac{d\mathbf{y}_l}{dt} = \mathbf{g_l}(\mathbf{y}_l, \mathbf{y}_h) \quad , \quad \mathbf{y}_{l,0} = \mathbf{y}_{l,0}(\mathbf{k}). \qquad (2.48)$$

$\mathbf{k}$ is the initial condition vector $\mathbf{k}(p_0, T_0, \phi_f)$ representing initial pressure p_0, temperature T_0 and the equivalence ratio ϕ_f is defined as

$$\phi_f = \frac{Y_{F,u}/Y_{O_2,u}}{(Y_{F,u}/Y_{O_2,u})_{st}}. \tag{2.49}$$

The $\mathbf{g_l}$ function can be separated into $\mathbf{g_l^c}$ which is calculated only from the light species and $\mathbf{g_l^{nc}}$ which depends on both the light and the heavy species. $\mathbf{g_l^{nc}}$ is a function of $\mathbf{f^c}$, computable from the light species and $\mathbf{f^{nc}}$, which depends on both the heavy and the light species as below

$$\mathbf{g_l}(\mathbf{y}_l, \mathbf{y}_h) = \mathbf{g_l^c}(\mathbf{y}_l) + \mathbf{g_l^{nc}}(\mathbf{f^c}(\mathbf{y}_l), \mathbf{f^{nc}}(\mathbf{y}_l, \mathbf{y}_h)). \tag{2.50}$$

If the $\mathbf{f^c}$ is plotted with respect to a normalised dominant variable $y_{d,n}$ and the normalised $\mathbf{f^{nc}}$ curve is plotted with respect to $y_{d,n}$, the curves will be coincident for the same normalised initial conditions used. This implies that $\mathbf{f^c}$ is locally partially self-similar and the separation of the non-computed term to $\mathbf{f^c}$ and $\mathbf{f^{nc}}$ enables modelling due to local full self-similarity [25]. The resulting characteristic functions would take the form

$$\mathbf{g_l}(\mathbf{y}_l, \mathbf{y}_h) = \mathbf{g_l^c}(\mathbf{y}_l) + \mathbf{g_l^{nc}}(\mathbf{f^c}(\mathbf{y}_l), \mathbf{f^{nc}}(y_{d,n}, \mathbf{k})). \tag{2.51}$$

2.5.1. Governing Equations for Light Species

In the LS2T method, the transport and conservation equations are solved for the light species only, and the effect of the heavy species on reaction rate and energy terms are included by data retrieval from a pre-calculated database. The resulting filtered chemical reaction rate term for the light species $\dot{\omega}_l$ is

$$\dot{\omega}_l = \dot{\omega}_{ll} + \dot{\omega}_{lh}, \tag{2.52}$$

where $\dot{\omega}_{ll} = \dot{\omega}_{ll}(\mathbf{y}_l, T, p)$ is computable and $\dot{\omega}_{lh} = \dot{\omega}_{lh}(\mathbf{y}_l, \mathbf{y}_h, T, p)$ is retrievable. Since the chemical source term for the heavy species $\dot{\omega}_h$ is not zero (but not calculated),

in order to maintain the mass conservation, a surrogate heavy species is introduced in addition to the light species. The surrogate species has a certain mass but zero enthalpy and the mass of the surrogate species is calculated from the rate of change of mass fractions of light species.

Similarly, the heat generation term is decomposed into two parts as

$$-\sum_{\alpha=1}^{N_s} h_\alpha \dot{m}_\alpha = -\sum_{\alpha=1}^{N_l} h_\alpha^l \dot{m}_\alpha^l - \sum_{\alpha=1}^{N_h} h_\alpha^h \dot{m}_\alpha^h, \tag{2.53}$$

where h_α^l and h_α^h represent the partial specific enthalpy for the light and the heavy species, respectively. The light species term is computed and the heavy species term is retrieved from tables.

The constant pressure specific heat can be decomposed as

$$c_p = c_{pl} + c_{ph}, \tag{2.54}$$

where c_{pl} is computed as part of the solution and c_{ph} is to be retrieved from tables. We can collect the terms as per distinction made to clarify the classification that $\dot{\omega}_{ll} \in \mathbf{g_l^c}(\mathbf{y}_l)$, c_{pl} and $\sum_{\alpha=1}^{N_l} h_\alpha^l \dot{m}_\alpha^l \in \mathbf{f^c}(\mathbf{y}_l)$, and $\dot{m}_{lh}$,c_{ph} and $\sum_{\alpha=1}^{N_h} h_\alpha^h \dot{m}_\alpha^h \in \mathbf{f^{nc}}(\mathbf{y}_l, \mathbf{y}_h)$.

2.5.2. Modelling Approach for non-Computed Terms

The non-computed terms in LS2T are obtained from a table constructed by using 0D reactor models with detailed mechanisms. The data are generated at an orthogonal (T_0, p_0, ϕ_0) grid so that any data Point of interest (Point of interest) is encapsulated by at least eight corner points. At each grid point, the reactions with detailed kinetic mechanisms are integrated with non-uniform time steps to provide sufficient temperature resolution until the reactor reaches steady-state conditions. During the integration, the reaction and the heat release rates of all the reactions are calculated and saved. The reaction and the heat release rates for the reactions with light species are identified and saved separately. The contribution of heavy species is calculated by

subtracting light species data from all species data and tabulated separately at each time step. Unlike Kourdis and Bellan [120], the reaction data is not stored at uniform time steps but instead at uniform temperature differences to reduce the size of the data and achieve adequate resolution. The resulting data size depends on the initial conditions and maximum temperature reached. At the end of the data generation, the maximum and minimum value of all generated data are stored to create the database constructed from normalised values. Any normalised variable is defined as

$$\eta_n = \frac{\eta - \eta_{min}}{\eta_{max} - \eta_{min}}, \tag{2.55}$$

where η can be replaced by the temperature, species concentration, reaction rate etc. The normalised data are stored in equally spaced divisions as a function of normalised temperature values (T_n) including the maximum, and the minimum values from each grid point. Each division is labelled by T, p, ϕ values for ease of access. It should be noted that the maximum temperature is not always the final temperature as in Figure 2.1 due to endothermic reactions activated by the pyrolysis, resulting in a multi-value problem [25] with respect to T_n. The multi-value problem such as OH mass fraction, as seen in Figure 2.2, is solved by separating the reaction progress to two sections by the maximum temperature point where the first section is exothermic ($dT/dt > 0$) and the second section is endothermic ($dT/dt < 0$). The tabular data set used from the exothermic section is called the first and the data set from the endothermic section is called the second set of data. The switch from the first data set to the second data set during the retrieval process is done under two conditions: when dT/dt changes direction or temperature passes the approximated T_{max} value.

During the integration of reduced chemistry, when heavy species data are required at any location of interest (T_0, p_0, ϕ_0), normalised data at eight surrounding nodes are identified and retrieved from the database. If the normalized values lay outside, the corner points end values are taken as

$$y_{d,n} = \begin{cases} 0 & \text{if } y_d < y_{d,min}(\mathbf{k}) \\ 1 & \text{if } y_d > y_{d,max}(\mathbf{k}) \end{cases}. \tag{2.56}$$

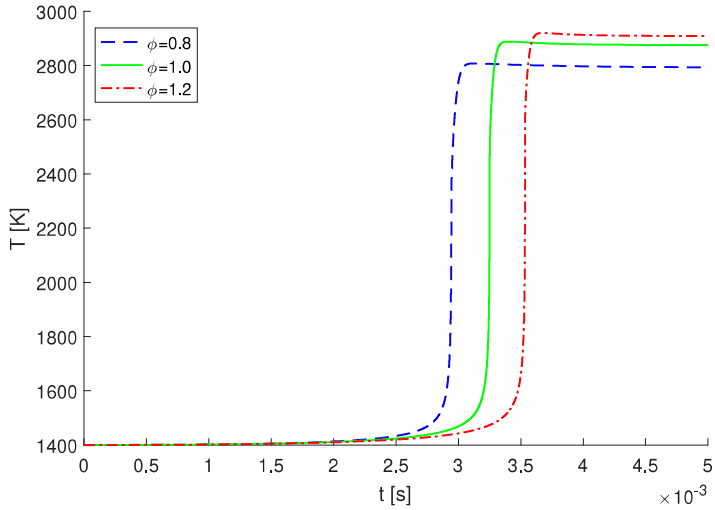

Figure 2.1. Temperature of constant volume CH_4 oxidation reaction for different equivalence ratios calculated using Cantera.

The extrema values of the eight surrounding nodes may be different from one another. The necessary extrema values to un-normalize interpolated heavy species data are approximated by interpolation using inverse distance weighting. The distance of each corner is calculated as

$$d_i = \sqrt{\left(\frac{T_0 - T_i}{T_{\max} - T_{\min}}\right) + \left(\frac{p_0 - p_i}{p_{\max} - p_{\min}}\right) + \left(\frac{\phi_0 - \phi_i}{\phi_{\max} - \phi_{\min}}\right)}, \tag{2.57}$$

where i is an index number from 1 to 8 representing the corners. The weight of each corner point is calculated by

$$WF_i = \frac{1}{{d_i}^z}, \tag{2.58}$$

where z is the power term that increases the influence of proximity and the interpolated value, IV, is obtained by inverse distance weighting from the corner values CV_i as

$$IV = \frac{\sum_{i=1}^{8} (CV_i \cdot WF_i)}{\sum_{i=1}^{8} WF_i}. \tag{2.59}$$

The inverse distance weighting approach enables the use of unstructured tabulated data and gives the same results with 3D-linear interpolation when uniformly spaced structured data are used.

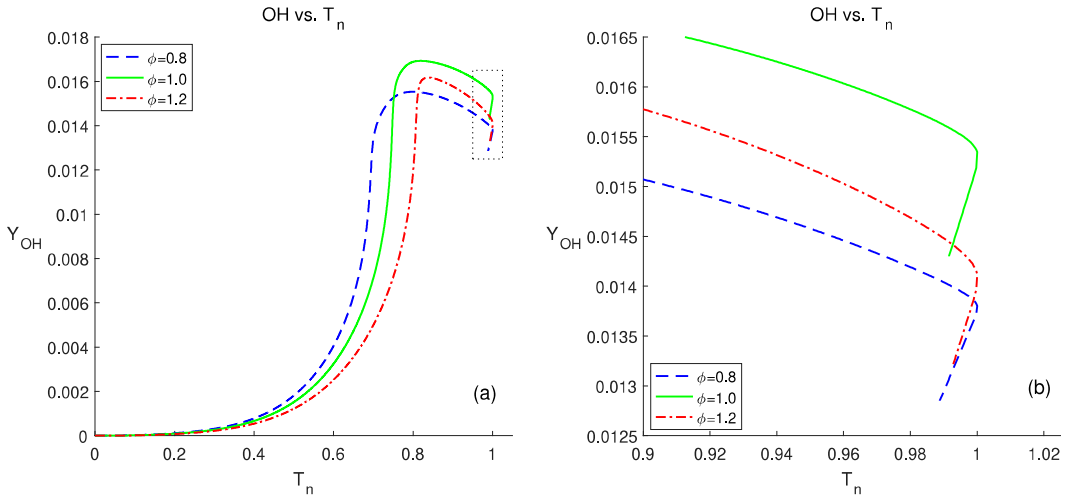

Figure 2.2. Species mass fractions vs normalised temperature T_n. Plot b is an enlargement of Plot a.

2.6. LES Computational Domain Resolution

The resolution of LES data is determined from the sub-grid scale filter used (mesh size in this case). The most common way for LES resolution evaluation are using the ratio of resolved turbulent kinetic energy to the total energy [122] or computing energy spectra and evaluating the distribution. However, Davidson showed that minimum 80% resolution for kinetic energy suggested by Pope or computing energy spectra to investigate if it exhibits a -5/3 ratio are not sufficient measures. Instead he recommended using a two-point correlation method. This method is based on the ratio of the integral length scale to the cell size [123] which indicates the number of cells necessary for resolution of largest scales. Davidson claimed that for a coarse LES analysis, the minimum number of cells required is 4, but this requirement is case dependent and even 8 cells may be insufficient in some cases for an adequate resolution of the largest scales [123]. In order to assess the LES resolution, the turbulent energy length scale L_{RANS} and Taylor microscale λ_T estimations from RANS model are used in this study. Turbulent energy length scale, which characterizes the size of the energy carrying large eddies and give a true indication of integral length scale, is obtained by dimensional analysis [124] as

$$L_{RANS} = \frac{k^{1.5}}{\epsilon}. \tag{2.60}$$

Turbulent Reynolds number based on L_{RANS} is

$$Re_L = \frac{k^2}{\epsilon\nu}. \tag{2.61}$$

Taylor micro-scale(λ_T), lies at the end of the inertial sub range and can be used as a measure for LES resolution. For high Reynolds number (Re) flows, ranging from 2000 to 10000, λ_T is estimated as

$$\lambda_T = \left(10\frac{k\nu}{\epsilon}\right)^{0.5}. \tag{2.62}$$

In the literature, to avoid over-resolved domain and an excessive computational load, it is advised to use the maximum of either Taylor scale or one-tenth of integral length scale as the filter scale for LES analysis. With the use of this filter means that large eddies are resolved at most by 10 cells [124].

2.7. Spray Theory

2.7.1. Eulerian-Lagrangian Modeling Approach

The multi-phase modeling of spray flows can be simulated by either Eulerian-Lagrangian coupled Lagrangian Particle Tracking (Lagrangian Particle Tracking) or Eulerian-Eulerian coupled methods [125]. Since the Eulerian-Eulerian approach solves the continuum of both gas and liquid phases, it is better suited for near nozzle flow solution of dense sprays. On the other hand, the Lagrangian Particle Tracking method treats the liquid phase as discrete particles making it more suitable for dilute spray regions [126] where liquid core is short and atomization with gas phase entrainment is fast [127].

The Eulerian-Lagrangian or Discrete Phase modelling approach is based on the idea of grouping droplets with similar properties as discrete particles (parcels) and

tracking them along the solution domain. The parcel trajectories can be computed at each or specified intervals individually along with the continuous phase calculation but the discrete phase exchanges mass, momentum and energy with the continuous phase at each time-step. The interaction with the continuous phase is achieved by additional formulations for drag, breakup, collision, evaporation and boiling [125]. The limitation of this approach is the assumption of neglecting particle-particle interactions. This limitation requires discrete phase to occupy a relatively low volume fraction in the cell, generally less than 10% [128], which makes it suitable for dilute spray as mentioned. This requirement determines the lower limit for grid resolution for LES turbulence model used with Lagrangian Particle Tracking method to maintain convergence.

2.7.2. Particle Motion

The particle trajectory is calculated from the integration of force per unit mass balance on the particle written in a Lagrangian reference frame [128] as

$$m_p \frac{d\mathbf{u}_p}{dt} = F_{Drag} + F_{Bouyancy} + F_{Other}. \tag{2.63}$$

The drag force is calculated [129] by

$$F_{Drag} = m_p \frac{18\mu}{\rho_p \varnothing_p{}^2} \frac{C_d Re_p}{24} (\mathbf{u}_p - \mathbf{u}), \tag{2.64}$$

where Re_p can be estimated as $\rho_p \varnothing_p |\mathbf{u}_p - \mathbf{u}|/\mu$. The drag coefficient C_d value can be computed by using several different models. Generally, most of the drag models assumes spherical droplet shape and calculates the drag coefficient based on the Re_p. However, for high Weber number flows, droplet will distort and change its shape as a disk, which would have higher drag coefficient than a spherical shaped droplet. Dynamic drag model takes into account droplet distortion and varies the drag coefficient linearly that of a sphere and 1.54 corresponding the disk [130] as

$$C_d = C_{d,Sphere}(1 + 2.632y), \tag{2.65}$$

where

$$C_{d,Sphere} = \begin{cases} 0.424 & Re_p > 1000 \\ \frac{24}{Re_p}\left(1 + \frac{1}{6}Re_p^{2/3}\right) & Re_p <= 1000 \end{cases}. \tag{2.66}$$

y is the droplet distortion calculated from solution of second order differential equation which is obtained from Taylor analogy breakup (TAB) breakup model but is suitable for any breakup model [131]. The shape of the droplet depends on the calculated y value, 0 for sphere and 1 for a disk.

2.7.3. Heat and Mass Transfer

The type of the droplet simulated determines the heat and mass transfer rules to be executed. A combusting droplet heats or cools, evaporates, boils or reacts at the surface.

When the temperature of the particle T_p is below vaporization temperature T_{vap} or when the volatile portion of the particle is consumed, the heat balance equation applied is

$$m_p c_p \frac{dT_p}{dt} = hA_p(T - T_p) + E_{Radiation}, \tag{2.67}$$

where c_p is the specific heat of the particle, A_p is the surface area of the particle, and T is the continuous phase temperature. Since radiation energy transfer is omitted in this work the details of $E_{Radiation}$ is not given.

The convective heat transfer coefficient h is calculated as [132]

$$Nu = \frac{h\varnothing_p}{k} = 2.0 + 0.6{Re_p}^{1/2}Pr^{1/3}, \tag{2.68}$$

where k is the thermal conductivity of continuous phase and Pr is the Prandtl number of the continuous phase. If the droplet temperature T_p is higher than T_{vap} but lower than the boiling temperature T_bp than both the mass (m_p) and T_p of the particle

changes due to evaporation. When the mass transfer is controlled by both convection and diffusion the mass of the droplet changes according to [133, 134]

$$\frac{dm_p}{dt} = k_c A_p \rho \ln(1 + B_m). \tag{2.69}$$

k_c is the mass transfer coefficient calculated by [132]

$$k_c = \frac{D_{i,m}}{\varnothing_p}(2.0 + 0.6 {Re_p}^{1/2} Sc^{1/3}), \tag{2.70}$$

where $D_{i,m}$ is the diffusion coefficient of vapor in continuous phase and Sc is the Schmidt number. B_m is the Spalding mass number calculated by

$$B_m = \frac{Y_{i,s} - Y_i}{1 - Y_{i,s}}, \tag{2.71}$$

where $Y_{i,s}$ is the vapor mass fraction at the surface of the droplet and Y_i is the mass fraction in the continuous phase.

At the end, the energy equation is modified to include the effect of mass transfer by evaporation as

$$m_p c_p \frac{dT_p}{dt} = hA_p(T - T_p) + E_{Radiation} - \frac{dm_p}{dt} h_{fg}, \tag{2.72}$$

where h_{fg} denotes the latent heat of evaporation. In addition, when evaporation rate is controlled by convection and diffusion, the model for the convective heat transfer h is updated as [134]

$$Nu = \frac{h\varnothing_p}{k} = \frac{1 + B_T}{B_T} 2.0 + 0.6 {Re_p}^{1/2} Pr^{1/3}. \tag{2.73}$$

B_T is the Spalding heat transfer number calculated as

$$B_T = \frac{c_{pv}(T - T_p)}{h_{fg} - \dot{q}_p/\dot{m}_p}. \tag{2.74}$$

$\dot{q}_p$ is the heat transfer to the droplet, $\dot{m}_p$ is the particle evaporation rate and c_{pv} is the specific heat of the droplet vapor.

When the temperature of particle (T_p) passes the boiling temperature of the particle T_{bp}, a boiling rate equation is applied [135]

$$\frac{d\varnothing_p}{dt} = \frac{4k}{\rho_p c_{gas} \varnothing_p} \left[1 + 0.23\sqrt{Re_p} \ln\left(1 + \frac{c_{gas}(T - T_p)}{h_{fg}} \right) \right], \tag{2.75}$$

where c_{gas} is the continuous phase specific heat capacity.

2.7.4. Secondary Breakup

Spray modelling involves several physical processes starting at the nozzle exit where liquid jet breaks up into liquid ligaments which further breakup into smaller droplets or collide to form larger ones. During these processes, droplets would shed due to drag force and evaporate or boil eventually as the temperature of the droplets increase [136]. Different breakup regimes of the fuel jet are possible due to interaction of nozzle cavitation effects, liquid inertia, surface tension, aerodynamic forces on the jet and turbulence. There are four different jet breakup regimes that have been defined in the literature [136] as shown in Figure 2.3. At Rayleigh breakup regime, droplet diameters are larger than the jet diameter due to collisions and breakup occurs many nozzle diameters downstream of the nozzle. At first wind-induced regime droplets are in the order of jet diameter but breakup occurs many nozzle diameters downstream of the nozzle. At second wind-induced regime, droplet sizes are smaller than the jet diameter and breakup starts at some distance downstream of nozzle. At atomization regime, the droplet sizes are much smaller than the jet diameter due to start of breakup at nozzle exit.

Based on the definitions, Spray-A shows the characteristics of second wind-induced regime. For simulation of a spray showing second wind induced regime characteristics, it is necessary to initiate the injection with a droplet size that is close to the nozzle diameter (blob injection), keeping the liquid core, and applying the necessary secondary droplet model with tuned parameters to reach a Sauter mean diameter (SMD) which conforms to droplet and vapour dispersion.

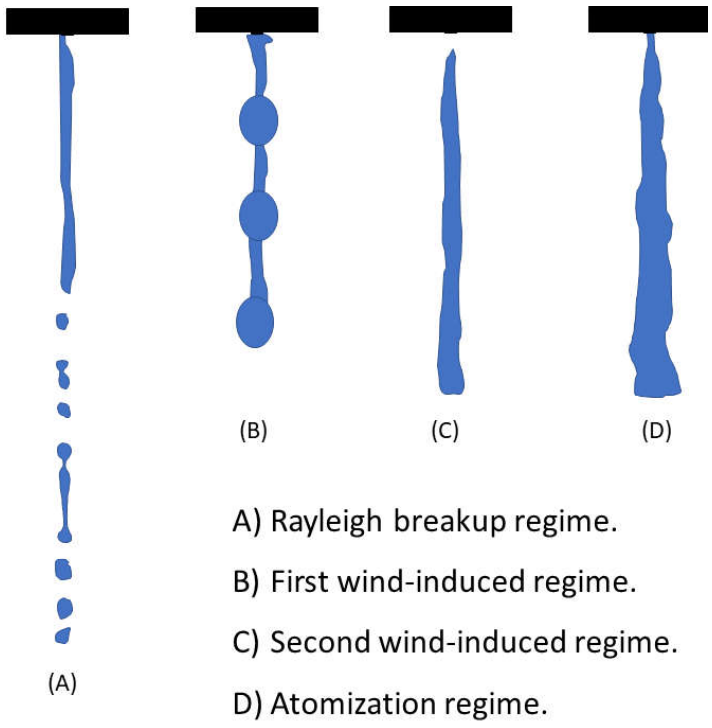

Figure 2.3. Jet breakup regimes.

In 1987, Reitz was the first who developed the "blob" injection model, by injecting liquid parcels that have the characteristic size equal to the nozzle diameter [1]. Although the injected droplets breakup due to interaction with the surrounding gas, there exists a core region near nozzle exit which contains large discrete liquid particles as in Figure 2.4. Kelvin-Helmholtz and Rayleigh-Taylor (KHRT) droplet breakup model assumes that the breakup time and resulting droplet size depend on the competition between fastest growing simultaneous KHRT instabilities [137].

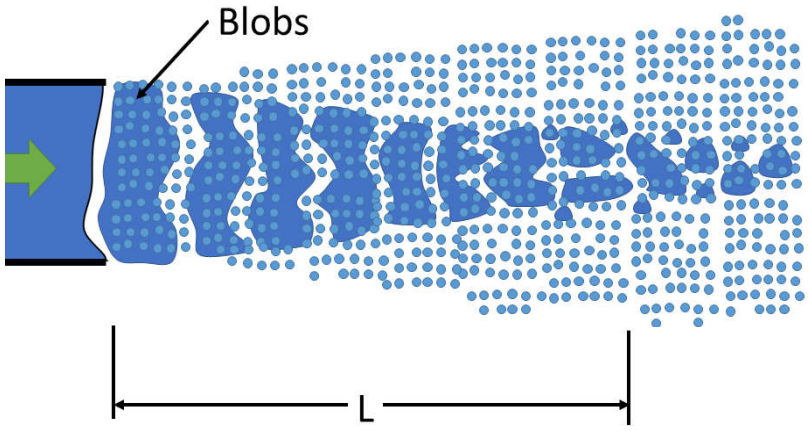

Figure 2.4. "Blob" injection breakup model [1].

The Kelvin–Helmholtz instability arises from the velocity shear in a single continuous fluid, or from the velocity difference over the interface of two fluids and shows itself as vertically and laterally progressing small scale motions [138]. The Kelvin–Helmholtz instability is modelled by wave spray breakup derived from jet stability analysis where Reitz and Bracco developed an equation for dispersion relation of waves [139]. Reitz numerically solved the equation and come up with curve fits for maximum growth rate, Ω_{wave}, and corresponding wave length, Λ_{wave} [1]. The effects of Ohnesorge, Taylor, Reynolds and Weber numbers on wave breakup [140] where σ, r_0, and ρ represent surface tension, initial droplet radius and density are shown respectively by equations below:

$$\Lambda_{wave} = 9.02 r_0 \frac{\left(1 + 0.45 Oh^{0.5}\right)\left(1 + 0.4 Ta^{0.7}\right)}{\left(1 + 0.865 {We_g}^{1.67}\right)^{0.6}}, \tag{2.76}$$

$$\Omega_{wave} = \frac{\left(0.34 + 0.385 {We_g}^{1.5}\right)}{\left(1 + Oh\right)\left(1 + 1.4 Ta^{0.6}\right)} \sqrt{\left(\frac{\sigma}{\rho_1 {r_0}^3}\right)}, \tag{2.77}$$

$$Oh = \sqrt{We_l}/Re_l, \tag{2.78}$$

$$Ta = Oh\sqrt{We_g}, \tag{2.79}$$

$$We_l = \rho_l U^2 a/\sigma, \tag{2.80}$$

$$We_g = \rho_g U^2 a/\sigma. \tag{2.81}$$

As shown in Figure 2.5, wave model assumes that the radius of newly formed droplets are proportional to wavelength of fastest growing or most probable unstable surface wave on parent droplet, which can be shown as

$$r = B_0 \Lambda_{wave}, \tag{2.82}$$

where B_0 is model constant set as 0.61 by Reitz. The rate of change of droplet radius is given as

$$\frac{dr}{dt} = -\frac{(r_0 - r)}{\tau_{wave}}, r \leq r_0, \tag{2.83}$$

where τ_{wave} is breakup time calculated as

$$\tau_{wave} = \frac{3.788 B_1 r_0}{\Lambda_{wave}\Omega_{wave}}. \tag{2.84}$$

The equation (2.83) is valid for cases where newly formed droplet size are smaller than the parent droplet. The effect of B_1 coefficient depends on the nozzle characteristics such as length over diameter ratio, cavitation and turbulence inside the injector. Although the suggested values for breakup time constant B_1 is 20 [1],there exists use ranges between 1.73 to 60 [140].

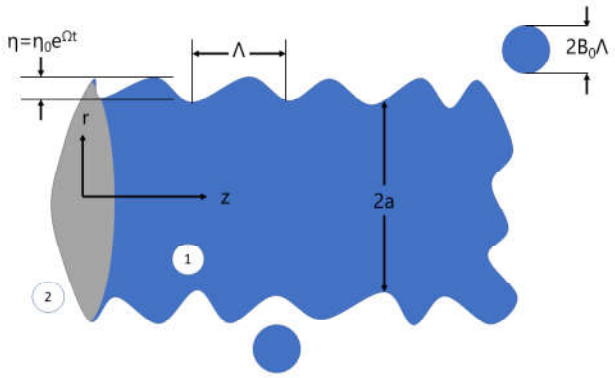

Figure 2.5. Surface waves and breakup of blob.

The Rayleigh-Taylor (RT) instability arises from the acceleration normal to the interface of two fluids. The acceleration a_l is found by division of the drag force by the droplet mass. The wavelength and frequency of fastest growing RT waves are calculated as [140]

$$\Lambda_{RT} = 2\pi\sqrt{\frac{3\sigma}{a_l\rho_l}} \tag{2.85}$$

and

$$\Omega_{RT} = \sqrt{\frac{2a_l}{3}}\left(\frac{a_l\rho_l}{3\sigma}\right)^{1/4}. \tag{2.86}$$

The droplet breakup for the RT waves occur when the wavelength is smaller than the droplet diameter and the growth of the waves takes longer than the RT breakup time τ_{RT} which is calculated by

$$\tau_{RT} = \frac{C_\tau}{\Omega_{RT}}, \tag{2.87}$$

where C_τ is the RT breakup time constant with default value of 0.5. The radius of the child droplet by RT breakup is calculated as

$$r_c = \frac{C_{RT}\Lambda_{RT}}{2}, \tag{2.88}$$

where C_{RT} is RT breakup radius constant with default value of 0.1.

The KHRT droplet breakup model is based on the assumption of having liquid core near the nozzle exit. Child droplets split from this core to smaller ones due strong shear at the interface (Kelvin-Helmholtz (KH) instability) and are shattered due to sudden acceleration as they enter the free-stream (RT instability). The KHRT model allows these acceleration and aerodynamic effects to compete to be the fastest growing instability. The liquid core length shown in Figure 2.4 is obtained from the Levich theory [141] as

$$L_{core} = C_L \varnothing_{n0} \sqrt{\frac{\rho_l}{\rho_g}}. \tag{2.89}$$

The KHRT breakup coefficients B_0, B_1, C_τ, C_{RT} and C_L values determine the breakup characteristic of the spray flow and should be set based on the nozzle design and the spray flow properties.

Based on the spray flow correlation of Hiroyasu and Arai [142], the breakup time τ_b and the spray penetration length S_{pen} are defined as

$$\tau_b = 28.7 \, \frac{\varnothing_{ne}\sqrt{\rho_l/\rho_g}}{\sqrt{\Delta P/\rho_l}} \tag{2.90}$$

and

$$S_{pen} = 0.39 \sqrt{2\frac{\Delta P}{\rho_l}}\, t, \tag{2.91}$$

where $\varnothing_{ne}$ is the effective nozzle diameter equal to the nozzle diameter $\varnothing_{n0}$ multiplied by the contraction coefficient C_a and ΔP is the pressure difference at the inlet at exit of the nozzle. Liquid core length L_{core} is equal to the spray penetration length at time(t) equal to τ_b and can be written as below by combination of equations (2.90) and (2.91) as

$$L_{core} = 11.193\ \sqrt{2}\ \sqrt{C_a}\ \varnothing_{n0}\sqrt{\frac{\rho_l}{\rho_g}}. \tag{2.92}$$

By equating the equation (2.89) and equation (2.92), Levich constant C_L can be found as

$$C_L = 11.193\ \sqrt{2}\ \sqrt{C_a}. \tag{2.93}$$

The contraction coefficient C_a can be found by experimental measurements. It should be noted that there are other definitions of the breakup time τ_b and the spray penetration length S_{pen} [143] but those definitions result in Levich constant to be dependent on the spray angle and would be under-predicted for high Weber number cases.

According to equations (2.83) and (2.84), the B_1 coefficient controls the rate of change of the radius of the parent droplet. The B_1 should be increased with the increasing injection Weber number in order to slow down the droplet breakup. Increased B_1 values also reduce the momentum exchange between gas and Lagrangian phases, avoiding divergence and resulting in increased spray penetration. Although the value of the B_1 coefficient varies between 1.73 to 60 [137,140], there is also a correlation given between B_1 and C_L values assuming inviscid flow and infinity gas Weber number [137] as $C_L = 0.5\ B_1$. This correlation gives the highest possible value for the B_1 coefficient for high Weber number flows and higher ratios are also reported in the literature [144].

The B_0 value which controls the size of the child droplets in KH breakup influences the rate of change of the parent droplets radius. B_0 value can increase from 0.61 [137] to 2 for high Weber number flows [144].

C_τ coefficient controls the rate of RT breakup where increasing it decelerates RT breakup. C_{RT} parameter determines the size of the broke-up droplets as of equation (2.88). Increasing C_{RT} value prevents smaller droplets to breakup. When C_{RT} is sufficiently high, breakup does not occur beyond liquid core. The usual values of C_τ and C_{RT} coefficients are 1 and 0.1 respectively [137] even though higher values (8 and 1) are also seen in the literature for high Weber number spray flows [144]. C_{RT} value of 0.1 might be too small for high Weber number cases and may cause divergence problem. For low Weber number cases it would result in very small child droplets. For both C_τ and C_{RT}, a range of 0.5 to 1 may produce good results [108].

2.7.5. Two-Way Coupling

Two way coupling means both continuous phase and discrete phases affect each other. This is achieved by alternately solving continuous and discrete phases until the solution converges for both.

The momentum change of a particle when it passes trough a control volume is computed as

$$\Delta Momentum = \sum \left(\frac{F_{Drag} + F_{Other}}{m_p} \right) \dot{m}_p \Delta t, \tag{2.94}$$

where $\dot{m}_p$ is the mass flow rate of particles in the control volume and Δt is the time step. This momentum change is included as a momentum source to the continuous phase.

The heat transfer from continuous phase to a particle when particle passes through a control volume is computed as

$$Q = \frac{\dot{m}_0}{m_0}\left[-m_{out}\int_{T_{ref}}^{T_{out}} C dT + m_{in}\int_{T_{ref}}^{T_{in}} C dT\right] \\ \frac{\dot{m}_0}{m_0}\left[(m_{in} - m_{out})(-H_{latent,ref} + H_{pyrolysis})\right], \tag{2.95}$$

where H_{latent} denotes the specific latent heat, $H_{pyrolysis}$ denotes the specific pyrolysis heat, C denotes heat capacity of the particle, subscript $_{ref}$ denotes the reference conditions and subscript $_0$ denotes initial conditions.

The mass transfer from discrete phase to the continuous phase when particle passes trough a control volume is computed as

$$m_{transfer} = \frac{\dot{m}_{p,0}}{m_{p,0}}\Delta m_p. \tag{2.96}$$

The mass transfer is included as a species source to the continuous phase.

3. COMPUTATIONAL MODELLING

3.1. LS2T Computational Method

The LS2T is a hybrid mechanism reduction method that solves the light species reactions only but retrieves the necessary reaction data of the rest of the species (heavy species) from a fore calculated database [25]. The method requires direct integration of chemical reactions for light species (C $<$ 3: O, CH, CH_2, CH_3, HO, HCO, HC_2, C_2H_3, CO_2, O_2, H, H_2, CO, CH_4, H_2O_2, C_2H_2, C_2H_4, CH_2O) for CH_4 oxidation and retrieving missing heavy species information from tables constructed for partial initial conditions (p_0,T_0,ϕ_0). However, based on the reaction mechanism, some of the light species reactions rate that are dependent on specific heavy species mass fractions such as C_2H_6 and AR and some species that are a direct contributor to emissions such as N_2, H_2O, NO, HCN, OH and NO_2 are also included into the light species group. This addition increase the number of light species and number of reactions to be solved.

The LS2T method can be divided into three computational parts, first is the tabular data generation, second is the data retrieval from table and calculation of heavy species contribution to light species reaction rates (lh terms) and third is the implementation of heavy species terms to the external chemistry solvers.

3.1.1. The Tabular Data Generation and Input File

The pre-processing and tabular data generation of LS2T method is done using the 0D thermo-chemical kinetic solver of Cantera [145] over Matlab [146] interface. Cantera is an open-source set of tools that can be used for solution of chemical kinetics, thermodynamics, and transport processes. Although the base language of Cantera is C++, it can be used from Python and Matlab, or in applications written in C/C++ and Fortran 90 [145].

When using Cantera, an appropriate reactor type should be chosen according to the 3D combustion problem at hand to provide accurate information from the 0D reactor. If the reaction domain is closed and the pressure change in the solution domain is as high as in internal combustion engines, during chemical reaction, there will be no flow going in or out since the chemistry time scales are comparably lower than transient CFD model time steps. In this case, each cell of the 3D CFD solution domain can be treated as a constant volume reactor for each time step. If the reaction domain is open such as a gas burner and the pressure is almost constant through the domain, the solution domain can be treated as a constant pressure reactor, and the data generated using such a reactor model should be applied.

The heavy species information for tabular data generation is obtained by subtracting the values (chemical source terms and enthalpy change) calculated by reactions that contain only the light species from the values calculated by the complete set of reactions. Retrieving the data at the initial conditions, where heavy species terms do not exist at all (such as inlets), may work for 0D or 1D reactor conditions. However, 3D problems require data to be retrieved at intermediate steps since the initial condition of each solution cell may correspond to a different state due to mixing and molecular transport. As a solution, to provide detailed and uniformly spaced data that can be used for a wide range of working conditions, the data retrieved from a 0D reactor is divided into multiple cases separated by temperature intervals of 25 K and classified as micro reactions with initial pressure, temperature and ϕ calculated at the midpoint.

Determination of initial conditions for the 3D reacting flow problem depends on the reaction type. If the reaction solved is a premixed-combustion problem the initial equivalence ratio is constant and the initial temperature and pressure is a function of ignition which can be determined easily. However, if the problem is partially-premixed or non-premixed combustion problem such as Sandia Flame the initial condition for the reactions is scattered which requires an additional assessment. The initial condition space for non-premixed combustion can be determined by solving the reacting flow with a global or HRM and review the domain. In Sandia Flame D combustion

case, the pressure in the flow domain is almost constant so only the ranges for the initial equivalence ratio and temperature should be determined. The minimum initial temperature is limited by ignition temperature (900 K for methane) for the fuel and maximum temperature depends on the flow problem, in Sandia Flame D case it can be given as the pilot temperature which enables ignition (1800 K). The equivalence ratio is highly dependent on mixing and can have values higher than the inlet conditions due to lack of oxygen in the domain. So, the rich limit for the Sandia Flame D problem is calculated by using the mixture fractions and flow rates at the inlet of the reactor with the assumption that the initial equivalence ratio of a cell can not be higher than this limit since both fuel and oxygen are consumed proportionally.

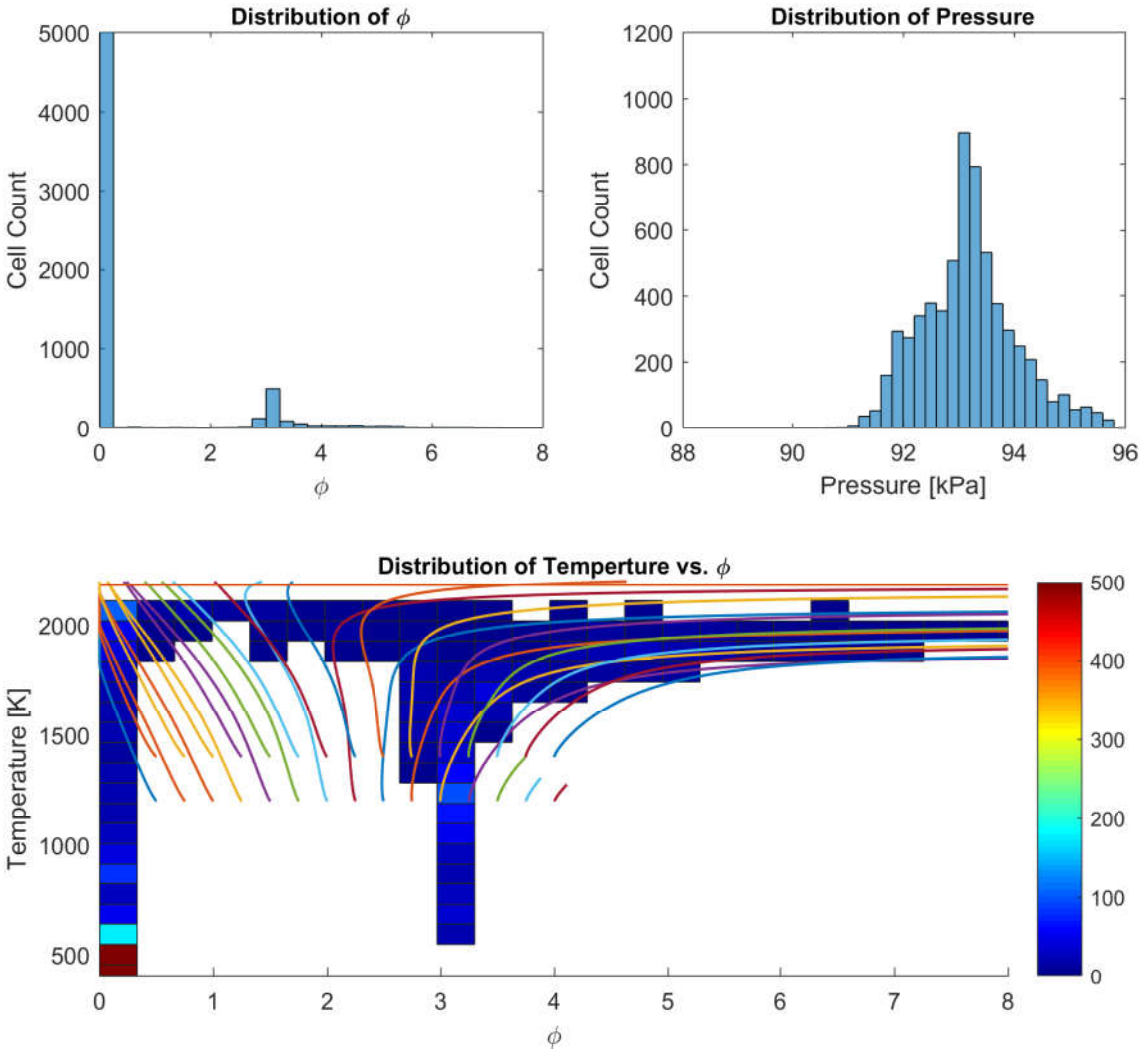

Figure 3.1. Histogram plot for temperature, pressure and ϕ distribution at inlet cylinder with 25mm radius and 250mm length.

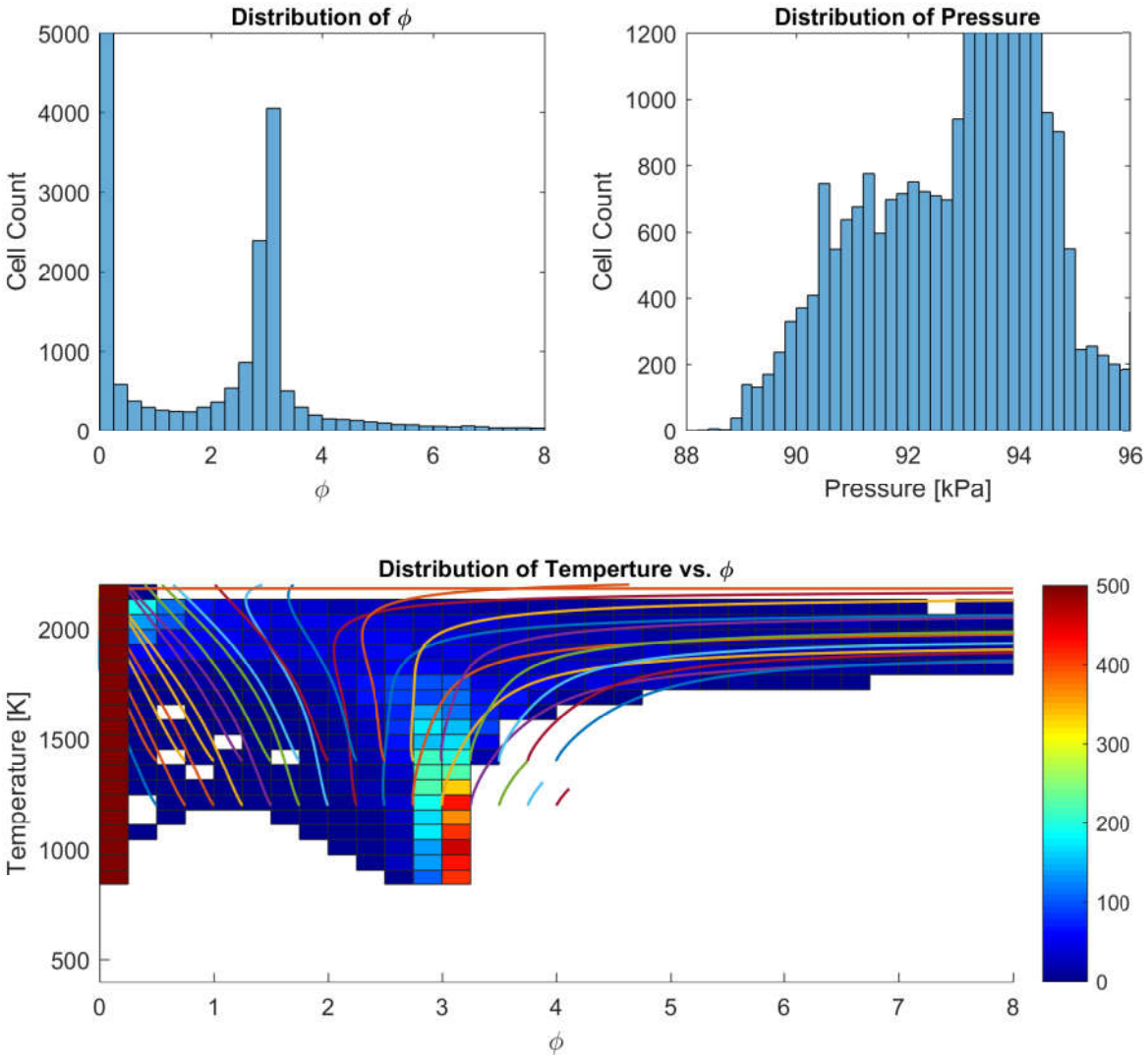

Figure 3.2. Histogram plot for temperature, pressure and ϕ distribution for cells that contain OH molecule.

At the predefined set of initial conditions, by using Matlab and modified Cantera scripts (explained in Section 3.2) the data generation and tabulation process is done in six steps:

(i) The Main Executor and Input Definitions:

GENERATE_DATA_v210613_R6_forFluent.m

This main Matlab file is for the execution of several sub-functions listed below with a set of given inputs. The first set of inputs are for the definition of fuel species, reaction mechanism, and definition of light species chosen. The second

set of inputs are for the determination of initial conditions for data generation. The third set of inputs are the data generation-specific settings. Those setting were determined during the study and can be treated as best practice settings. While the main file generates an initial condition space based on the values given and executes the below functions one by one but each function is executed in parallel by the built-in functionality of Matlab, reducing the data generation time significantly.

(ii) Initialization of Reaction Mechanism:

reactor_0_initializeChemReduction_v2.m

This initialization script first creates a reduced mechanism with the light species and any necessary additional species for reaction rate calculation. It determines the reactant, product, and net stoichiometric reaction coefficients and calculates the stoichiometric Air Fuel Ratio (AFR). It saves all the values as a **_init_data.mat* file to be used at each reaction calculation stage. This function avoids re-calculation of the same data for each reaction which may take up to a minute to calculate based on the size of the reaction.

(iii) Reaction Calculation at Predefined Initial Conditions:

reactor_1_RunReaction_adaptiveDt2_v7.m

This script integrates in time the governing equations for a constant pressure or constant volume reactor, starting from the initial condition. The integration terminates when the temperature reaches a steady value. To reduce the computational time, an adaptive time stepping method is implemented. This method increases or decreases the time step used based on the temperature difference between two time steps then redefines entire time steps based on a target temperature resolution of 0.4 K or target equivalence ratio resolution of 0.0005. During the reaction progress, at each time step, the reaction rate of light species from the reactions that contain only light species and the contribution of the reactions that include heavy species are calculated separately and saved.

(iv) Generation of Energy Terms from Calculated Reaction Data:

reactor_2_GenerateEnergyTerms_v8.m

Related Cantera function does not output the change in enthalpy values of the

species during the reactions. The change in enthalpy and total energy due to changes of enthalpy and species mass fractions are calculated at each time step and saved as the energy contribution of the reactions that contain heavy species.

(v) Data Population:

reactor_3_PopulateTables6.m

The storage and retrieval from the table depends on the idea of having structured data with a certain number of items at each set location. For that reason, each reaction data (light and heavy reaction rates, energy terms, temperature, species mass fraction etc.) is sampled to 25 points and saved separately by this script.

(vi) Normalization and Resizing:

reactor_4_Normalize_and_ResizeData_v5.m

The LS2T method is based on the idea of utilizing local self-similarity of the reactions when normalized. The method proposes normalization of reaction rate, and energy terms by reactions maximum and minimum values and tabulating them based on the normalized temperature values. This script normalizes the necessary data (reaction rate contribution ω_{lh} for each light species, and energy contribution by the heavy species) and saves them.

(vii) Table Creation:

reactor_5_CreateSingleInoutFile17.m

The generated normalized data should be saved in a predefined structure to find the Point of interest and retrieve data accordingly. This script collects all the data and arranges them in a predefined manner. Since reactions may show a multi-valued behaviour based on temperature, this script saves two separate files for positive and negative temperature gradient directions.

The normalized data tables of LS2T method includes the heavy species contribution to light species reaction rates and missing data for overall energy continuity. The first two rows of the table contain data for processing the normalized values. Starting from the third row each species additional reaction rate and heavy species energy data are listed. Each set point is saved one after another, starting at different columns. The output files are saved as binary files to reduce the file size. The format of the table can

be seen in Table 3.1.

3.1.2. LS2T Method Execution: Computing Source Terms

The execution of the LS2T method is based on the idea of directly solving reaction chemistry for only light species and adding heavy species contribution to the light species reactions and overall energy balance. The additional reaction rate and energy release are included as source terms. To calculate the heavy species source terms at a specific temperature, pressure, and equivalence ratio ϕ, necessary data are found from the tables and processed afterward. The retrieval of data from the table requires a search algorithm that finds the closest surrounding reaction conditions for the Point of interest (Fig. 3.3). As the surrounding conditions are found, a linear interpolation algorithm calculates the reaction data corresponding to Point of interest and an inverse distance weighting (inverse distance weighting) algorithm determines the weighted values from a set of scattered corner points. Unlike the trilinear interpolation method used by Kourdis and Bellan [25], the inverse distance weighting provides the freedom to use data in an unstructured grid and enables utilization of closest eight surrounding corner points data for a 3D problem.

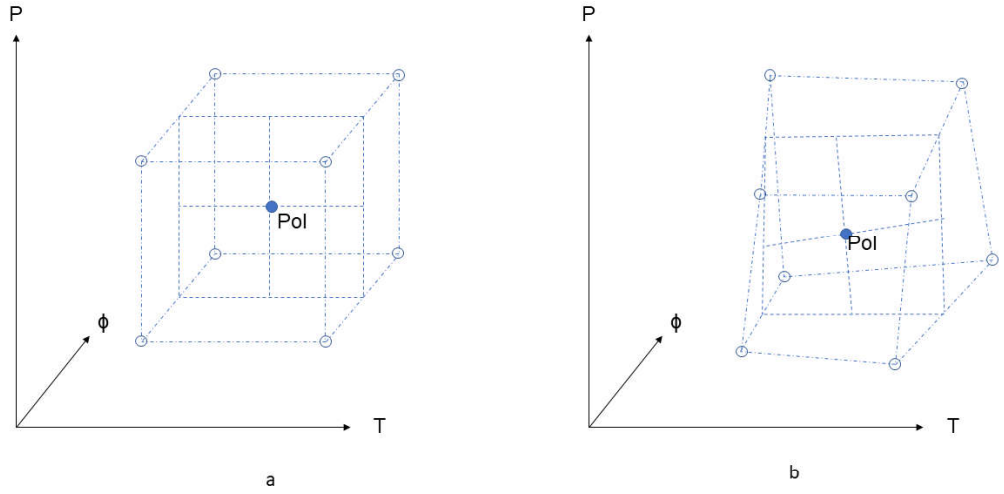

Figure 3.3. Data structure, Original Structured(a) vs. New Unstructured (b).

Table 3.1. The tabular input file format.

	1	**2**	**3**	**4**	**5**	**6**	...		**4+nPhi**	**5+nPhi**	**6+nPhi**	**7+nPhi**	**8+nPhi**	**9+nPhi**	**10+nPhi**
1	Number of File Rows	Number of File Columns	Number of Species (nS)	Number of Phi Sections (nPhi)	Phi Sections Column Locations				Phi increment	Number of Files	Max Phi	Global Maximum Tempera-ture	Global Minimum Temperature	-10000	-10000
2	File Number (Ex: 1)	Pressure	Temperature	Phi	First(1) or Second(2) Part of MVF	Number of columns for this point (nC)	...(nC)	File Number (Ex: 2)	Pressure	Temperature	Phi	First(1) or Second(2) Part of MVF	Number of columns for this point (nC)	...(2nC)	
3	Normalized Temperature					Local Max. of Temperature	Local Min. of Temperature	Normalized Temperature					Local Max. of Temperature	Local Min. of Temperature	...
4	Normalized Change in Entalphy					Local Max. of Entalphy Chan.	Local Min. of Entalphy Chan.	Normalized Change in Entalphy					Local Max. of Entalphy Chan.	Local Min. of Entalphy Chan.	
5	Normalized Change in Specific Heat					Local Max. of Sp. Heat Change	Local Min. of Sp. Heat Change	Normalized Change in Specific Heat					Local Max. of Sp. Heat Change	Local Min. of Sp. Heat Change	
6	Rate of change of fuel molar concenter.by Heavy omega_lh(1)					Local Max. of omega_lh(1)	Local Min. of omega_lh(1)	Rate of change of fuel molar concenter.by Heavy omega_lh(1)					Local Max. of omega_lh(1)	Local Min. of omega_lh(1)	
7	Rate of change of oxygen molar concenter.by Heavyomega_lh(2)					Local Max. of omega_lh(2)	Local Min. of omega_lh(2)	Rate of change of oxygen molar concenter.by Heavyomega_lh(2)					Local Max. of omega_lh(2)	Local Min. of omega_lh(2)	
...	...					...	...	...					...	...	
nS+5	Rate of change of oxygen molar concenter. by Heavy omega_lh(nS)					Local Max. of omega_lh(nS)	Local Min. of omega_lh(nS)	Rate of change of oxygen molar concenter. by Heavyomega_lh(nS)					Local Max. of omega_lh(nS)	Local Min. of omega_lh(nS)	
nS+6	Normalized Time (Unused)					Local Max. of Time	Local Min. of Time	Normalized Time (Unused)					Local Max. of Time	Local Min. of Time	

The main script for the LS2T method is developed using Matlab. However, when the Matlab functions are converted to C++ executable by use of Matlab's built-in converter, the execution of the converted file is as slow as Matlab itself and too complex to debug. In the end, all Matlab scripts have been converted to a standalone C++ script file (*KourdisCodes.cpp*) that can be included in any third-party combustion algorithm as long as it is C++ compatible. Sub-functions of this script are explained below:

(i) Read Input File

ReadFileandPassData(Location, DoubleDataVector)

The size of the input file is reduced by saving the input file in binary format. The binary file is read by this function and converted to a double array based on the number of row and column values supplied at the first two inputs of the binary file. The read double array data is passed to a double vector and saved in the memory for frequent use.

(ii) Search and Find Algorithm

Vector Int Find_SurroundingDataSeries_RandomFiles2X_5(...)

When the temperature, pressure, and equivalence ratio ϕ (Point of interest) are supplied to the LS2T executor, this function searches for the closest surrounding 8 corner data points to use for calculation of heavy species contribution terms. In order to reduce the search duration, the sections to search the data are eliminated based on the ϕ value. At the selected data sections the Point of interest is compared with the set point of each data and a distance and location at which side the data belongs to are calculated. At each corner location, the closest data point are chosen and exported to be used for calculations. The distance-based data search is necessary since the data are not equally distanced from the Point of interest and are unstructured as can be seen in Figure 3.3 and the search algorithm is the main difference from the method used by Kourdis and Bellan [25].

(iii) Inverse Distance Weighing Algorithm

Bool InterpolatebyInverseDistanceWeigthed37_2_wlhn_re(...)

The data retrieved from the table does not belong to the Point of interest but is

close enough so it can be used for calculation at Point of interest. The supplied data are tabulated as a function of normalized temperature. The values for non-computable heavy species reaction rate (ω_{lh} and energy terms) at Point of interest are estimated at the closest corners by linear interpolation of retrieved data based on the normalized temperature. In addition, the maximum and minimum values needed to unnormalize non-computable terms are also retrieved by this function. Since the retrieved data at the corners are not equally spaced from Point of interest the weight of the corner points are not equal. To consider the distance from Point of interest, the inverse-distance-weighing algorithm, (explained at section 2.5.2) has been used. The inverse distance weighting data are unnormalized by calculated maximum and minimum values and exported as source terms to be used in reaction calculations in the external program.

Table 3.2. New matlab functions.

Matlab Function	**Reactor Net Key**	**Purpose**
CanteraReadHeavyandSaveMemory	10	Reads the Cantera input file and tabular data file.
advancewithHeavy	11	Advances time with chemistry integration using LS2T method.

3.2. Implementation of LS2T method to Cantera for Matlab

The Cantera is open-sourced set of tools for the solution of problems involving chemical kinetics, thermodynamics, and transport processes [145]. It is written in C++ language but can be compiled to work in C++, Python, or Matlab environments. For this study, Cantera v2.2.1 has been used over the Matlab interface. The compiled functions of Cantera are called over Matlab interface using gateway Matlab functions that are included in the compilation library. The newly added functions are listed in Table 3.2. The new keys added in the *reactornetmethods.m* requires no modification in the Matlab file but requires new switches to be added to the counterpart C++ function *reactornetmethods.cpp* of Cantera for new option numbers 10 and 11.

3.2.1. Reading Input Files and Saving to Memory

The inputs necessary for execution of LS2T scripts are read from *Cantera_Inputs_Heavy.txt*. Based on the input parameters, the tabular data files are read and imported data are saved to the memory. The inputs given in*Cantera_Inputs_Heavy.txt* are listed in Table 3.3.

The tenth switch in *reactornetmethods.cpp* class calls the *reactornet_ReadHeavyArray* function in *ctreactor.cpp* of Cantera. The *reactornet_ReadHeavyArray* executes a function called *ReadFileandPassDataG* in *global.cpp* to trigger the read function of LS2T scripts. *ReadFileandPassDataG* function is a global function that can be executed at any point in Cantera and can pass data to variables defined in *application.h* where all variables and arrays are globally defined and be accessed anywhere within the Cantera.

3.2.2. Reaction Rate Calculation in Cantera with LS2T

The eleventh switch in the *reactornetmethods.cpp* class calls a new function *reactornet_advance_Heavy* in *ctreactor.cpp* of Cantera. The *reactornet_advance_Heavy* executes another function called *RunningHeavy* in *global.cpp* to invoke Cantera that it is running with the LS2T method. Cantera checks if it is satisfying the requirements for execution of LS2T methods and makes necessary adjustments accordingly in the reactor models.

The Ideal Gas Reactor(Cantera reactor id 2), the Ideal Gas constant pressure reactor (CPR) (6), and the Ideal Gas constant volume reactor (CVR) (5) have been modified to work with the LS2T. All the necessary reaction rate calculations and energy release at the end is processed in *evalEqs* function of each reactor (*IdealGasConstPressureReactor::evalEqs* for CPR).

Table 3.3. LS2T input text file.

Line Num.	**Example Inputs**	**Explanation**
1	AllData1.bin	Input file 1 for multi value reactions based on temperature
2	AllData2.bin	Input file 2 for multi value reactions based on temperature
3	0.25067	Stoichiometric fuel and oxygen ratio for phi calculation
4	0	Verbose file "Cantera_Verbose.txt". 1: Writes what is written to screen, 2 some functions entrance and time, 3 LS2T code executions, 4 LS2T interpolated values.
5	4	Weighting factor for IDV used as phi drops 0.9*Phi_first
6	-1	Phi limit for application of WF =1 until Phi_first*limit reached. If set below zero ignored.
7	30	The temperature search range to find the nearest temperature data in this range.
8	1000	1st Branch Temperature gradient limit by light species to decide if the reactions has ended. Always positive.
9	50000	2nd Branch Temperature gradient limit by light species to decide if the reactions has ended. Always positive, smaller the value, later the reaction ends.
10	1	Extrapolate pressure values out of limits. (Fluent)
11	5	End limit fuel air ER(phi) limit for reactions (Fluent)
12	1.67	Upper flammability limit fuel air equivalence ratio
13	0.5	Lower flammability limit fuel air equivalence ratio
14	580	Minimum ignition temperature limit (Fluent)
15	1.00E-09	Minimum Fuel Mass Fraction (Fluent) or Phi(Cantera) Limit for Reaction To End
16	1.00E-04	Minimum Fuel Mass Fraction Limit (Fluent) for Reaction To Begin
17	1	Heavy Energy multiplier (Fluent)

The first step of execution with LS2T is the retrieval of necessary data at Point of interest by calling *GetHeavyRR* inside the *evalEqs*, *GetHeavyRR* fucntion of *Reactor.cpp*. In the *GetHeavyRR* function, necessary variables are loaded and fed to the *InterpolatebyInverseDistanceWeigthed37_2_wlhn_re* of the LS2T method and resulting data is saved to the memory to be used at the inner integration steps. The heavy species reaction rate ω_{lh} is added to each light species reaction rate ω_{ll} in the *evalEqs* while calculating the surrogate species mass fraction change for mass continuity of heavy ones and adding the energy release by heavy species reactions. Overall process has been summed up in Figure 3.4.

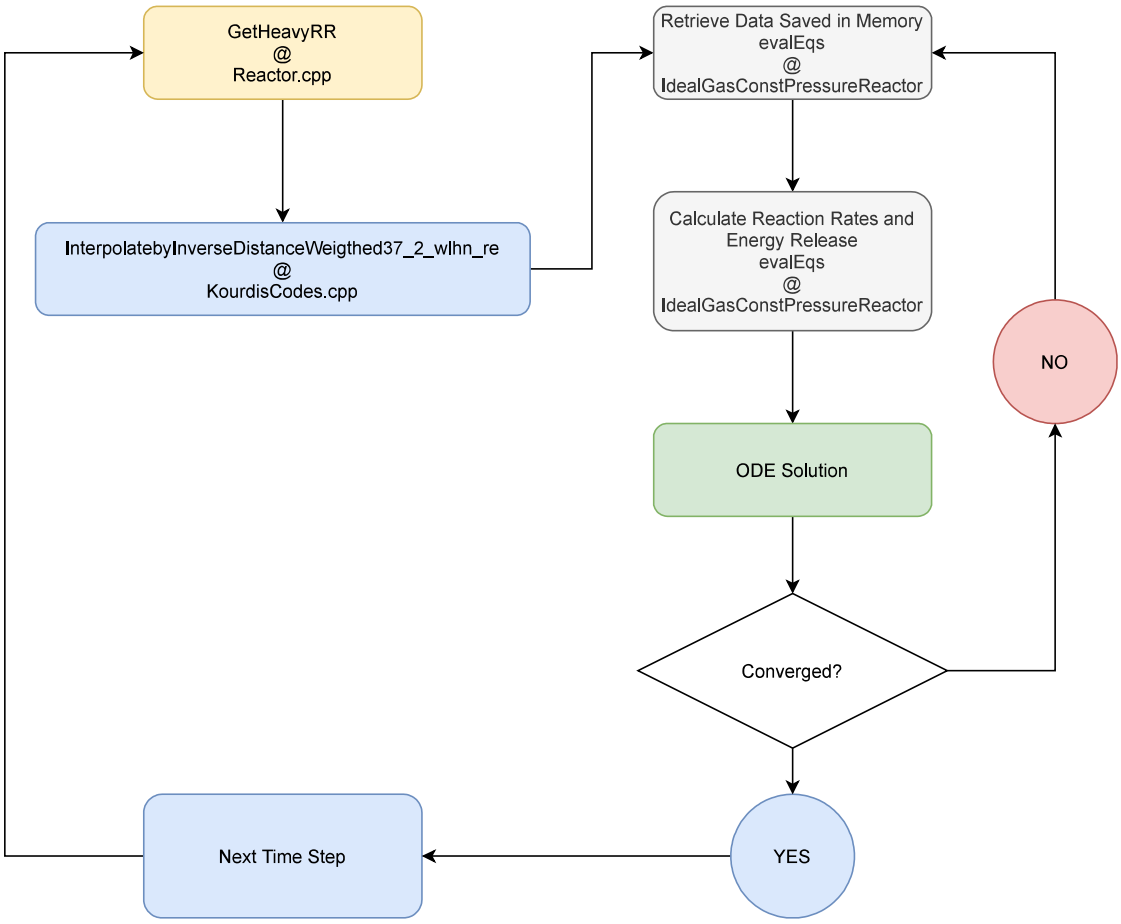

Figure 3.4. Cantera execution pseudo-code.

3.2.3. Start and End of Reaction Logic

Cantera 0D reactors do not contain any mixing or flow exchange, so the reaction starts at the first time step and there is no need for the additional start of reaction consideration. However, the end of the reaction should be determined not to redundantly extrapolate the heavy species effect on light ones. The end of reaction in Cantera is determined from two limits given in the user input file, the temperature gradient change, and the minimum ϕ. When the temperature rises over a limiting temperature (1400 K in this case), if the temperature change is smaller than a limiting value or the ϕ reduces below a certain limiting value, the function inside Cantera decides the reactions are finished.

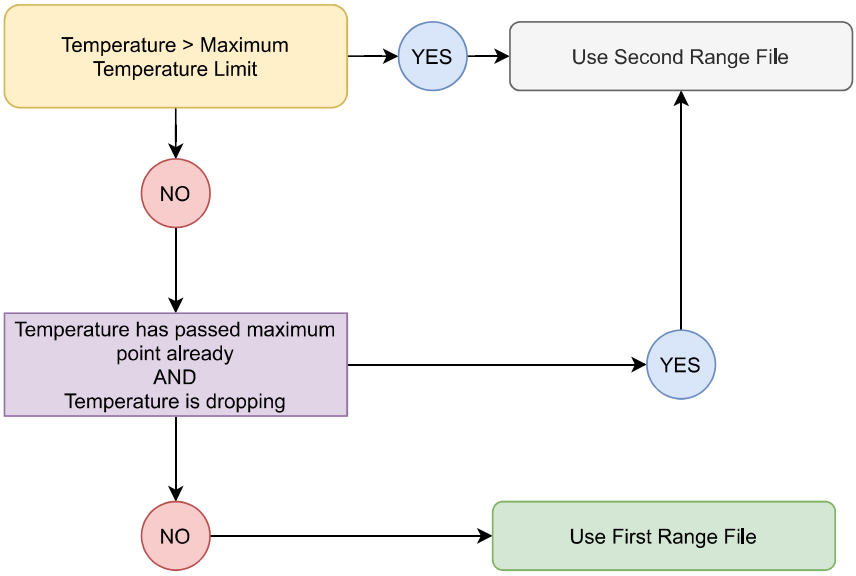

Figure 3.5. The determination logic to change the data set.

3.2.4. Use of Second Range of Reactions Logic

The multi-valued reaction rate data based on the normalized temperature is divided at the pivot points where the temperature gradient changes its direction. The pivot point can be the maximum temperature point where the reactions dominant are pyrolisis reactions and the temperature is dropping accordingly or the minimum temperature point due to evaporation or heat transfer. Based on the temperature gradient

direction, the reaction data is divided into two sets. The determination logic used by Kourdis and Bellan [25] to change the data set from first to the second range is presented in Figure 3.5. Kourdis and Bellan identified that for n-dodecane when the mixture is rich, the reaction shows an alternating behaviour based on temperature after reaching the maximum temperature [25]. However, this logic is not sufficiently covering all possible multi-valued reactions since it depends on the fact that pivot point is always the maximum temperature.

Based on the reaction mechanism and physics of the reacting domain, the temperature gradient can change the direction several times as can be seen in Section 4.2.1. In this study, a more general logic that can be applied to any reaction case is developed. As shown in Figure 3.6, instead of using only the maximum temperature as the pivot point, the program monitors the temperature in the domain and changes the input data file as the direction of the temperature gradient changes.

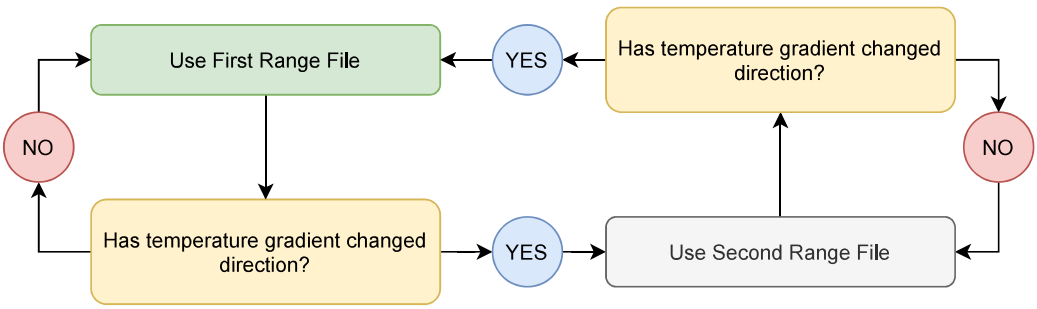

Figure 3.6. The determination logic to change the data set used in this study.

3.2.5. Compiling LS2T Scripts with Cantera for Matlab Interface

Cantera comes with pre-set software construction tool (Scons) which compiles and installs Cantera to multiple interfaces at once. To compile Cantera from the command prompt, Scons v2.4.11 (for Win64 is used) and Python v2.7.11.amd64 should be installed first. By updating the options listed in the configuration file *cantera.conf*, and using suitable test commands, Cantera can be built, installed, or uninstalled without any additional user control [145].

3.3. Implementation of LS2T Method to a 3D CFD Analysis Tool

The 3D CFD analysis tool used in this study is Ansys Fluent v19.2 [128]. However, the compiled LS2T application is platform and CFD tool independent as long as it can be interfaced with the CFD software like using user defined subroutines in Fluent. The user defined script for Fluent is given in Appendix C.

3.3.1. Reaction Rate Calculation by User Defined Functions

When stiff chemistry is preferred for the calculation of the reaction rate terms, the contribution of heavy species reaction rate to light species reactions should be included before the integration of reactions. Since the time step for calculation of heavy species terms and 3D CFD solver is different, the values obtained as source terms calculated at different time step sizes would result in incorrect calculations and even divergence of the solution. The modification of reaction rate calculation by additional terms requires a set of user-defined functions that calculate the Arrhenius reaction rates used by the PDF transport model to be written. The reaction rate calculation functions that are composed of the below elements have been provided by Ansys Fluent Support and used with minor modifications.

- *static void calc_thermo_props(double T, double ln_T, double *yi,double *cpi, double *hi, double *si)*: Calculates the thermodynamic properties of each species at Point of interest.
- *static void fill_thermo_props(Material *m, Material *sp, int i)*: Assigns calculated properties to variables.
- *static void fill_cache_variables(void)*: Assigns necessary values to global variables.
- *double CalculateRRforSpeciesTransport(Thread *t, cell_t c, double *press, double *temp, double *yi, double *wdot)*: Calculates the Arrhenius reaction rates based on the reaction mechanism in the model.

For the implementation of reaction rate subroutine, Ansys Fluent has predefined DEFINE_NET_REACTION_RATE function which provides the species mass fractions, temperature, and pressure term in the cell it is executed and retrieves the rate of change of molar concentration $\dot{\omega}$ at the end [147]. In this function, the script checks if the cell is eligible for reaction to occur or not, calculates the reaction rates for the light species and includes heavy species contribution to light ones. To maintain mass continuity and reduce the execution time for LS2T method, the LS2T functions are called once at the end of each time step (by predefined DEFINE_EXECUTE_AT_END function) and the outputs are saved to memory for retrieval as needed. The LS2T outputs are not updated at the inner iterations. The additional energy values are included by defining energy source terms in predefined DEFINE_SOURCE function by retrieving energy terms from LS2T outputs.

3.3.2. Reading Input Files and Saving to Memory

In the CFD tool the input files should be read for once at the beginning of the calculations and the data should be saved to memory. The DEFINE_ON_DEMAND (*Read_CanteraInputsHeavy*) function which creates functions that can be called any time within Fluent, is preferred to be used for this purpose. A DEFINE_ON_DEMAND function enables re-reading necessary inputs if changed, without the necessity to reset the memory at any time required. The DEFINE_ON_DEMAND function is called at the beginning of calculations and the tabular data and other parameters are passed to global variables. Two dynamic arrays are kept in the memory and retrieved as needed during Fluent execution. The ϕ, pressure, and temperature range of imported tabular data is cross-checked at each cell by *CheckIf_IsTableEncapsulating* function and used if the Point of interest fits into the range of tabular data.

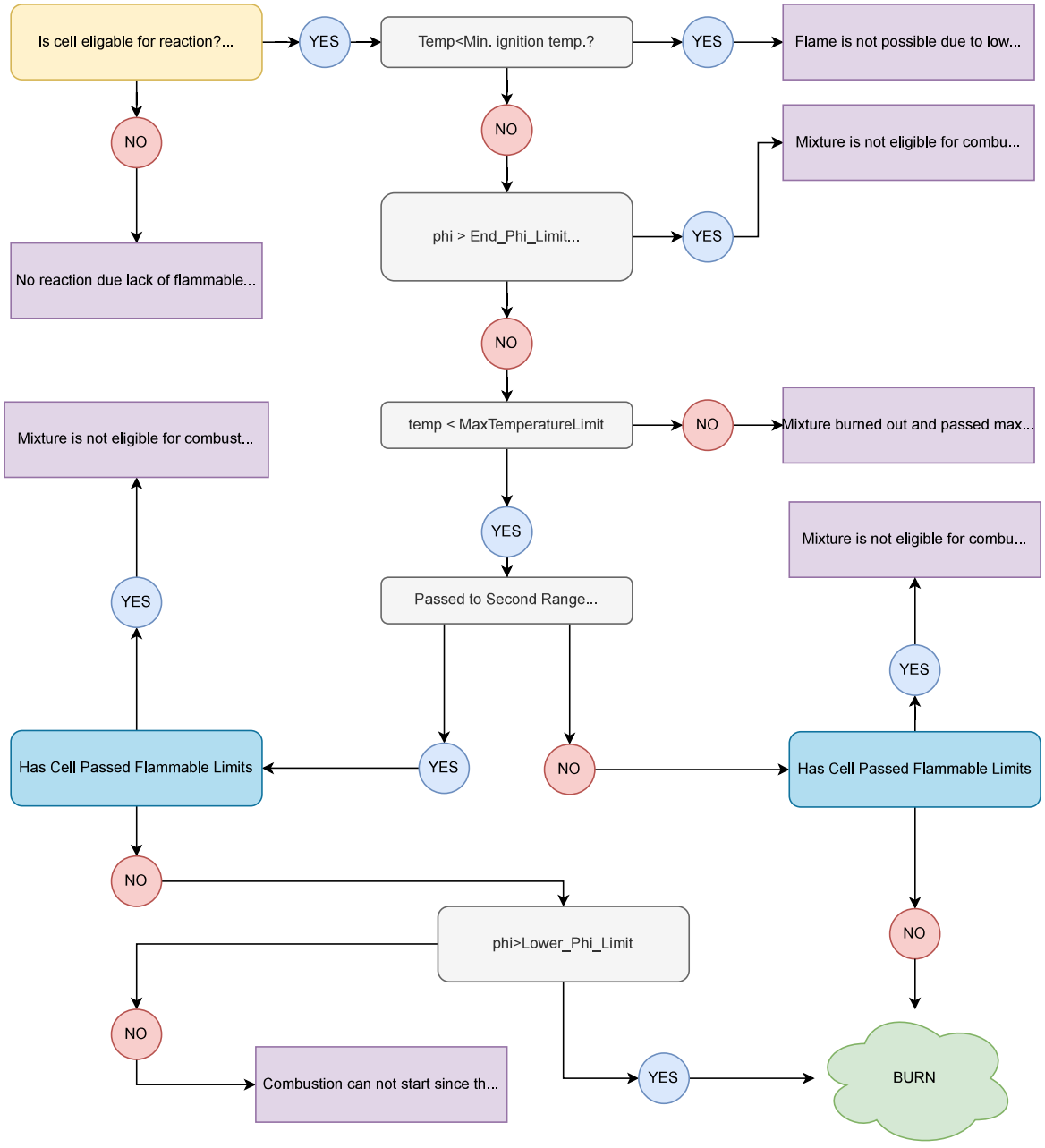

Figure 3.7. Start and end of reaction pseudo-code.

3.3.3. Start and End of Reaction Logic

To avoid unnecessary execution of the LS2T algorithm at a location where the mixture is not suitable for combustion initiation or where the fuel is burned out completely at the end, a decision logic has been developed. This decision logic is included with a user-defined scalar (user-defined scalar) to understand the start and end of the reaction and keep track of combustion eligibility along with the solution domain. The

user-defined scalar value inside the solution domain is initially set as zero. For premixed combustion cases, it is set as one at the inlet where the fuel is supplied. In the case of non-premixed spray combustion, the user-defined scalar value is initiated as one where the fuel mass fraction is over the flammable limit. The user-defined scalar is transported with the flow without diffusion and used as an indicator for the combustion eligible mixture content. The pseudo-code for the determination of mixtures' eligibility for combustion is given in Figure 3.7. The end of reaction determination requires consideration of more parameters compared to Cantera because of the mixing and the existence of fuel-rich cells due to combustion.

3.3.4. Compiling LS2T Scripts with Fluent User Defined Subroutine

The LS2T script is written in C++ but Fluent is not C++ compatible and uses C script instead. However, C++ function compiled can be called within C script by use of "extern" keyword. The library should be compiled over the command prompt and the pre-compiled user object should be pointed in the make-file. The compilation process in Windows 10 and Linux is slightly different. The details can be found in Appendix D.

3.3.5. LS2T Execution in Fluent

The LS2T method is implemented to Fluent solver by C programming language based and dynamically loaded user-defined function (user-defined function) [128]. For user-defined function implementation, the net reaction rate solver has been re-written in C with additional functions to include the LS2T tabular data, and data storage and retrieval subroutines with several other functions have been added. The functions that are necessary for the LS2T method, such as nearest (T_0, p_0, ϕ_0) point search, data retrieval algorithm, and inverse distance weighting algorithm are written in C++ and compiled as a separate program to make LS2T method CFD code independent.

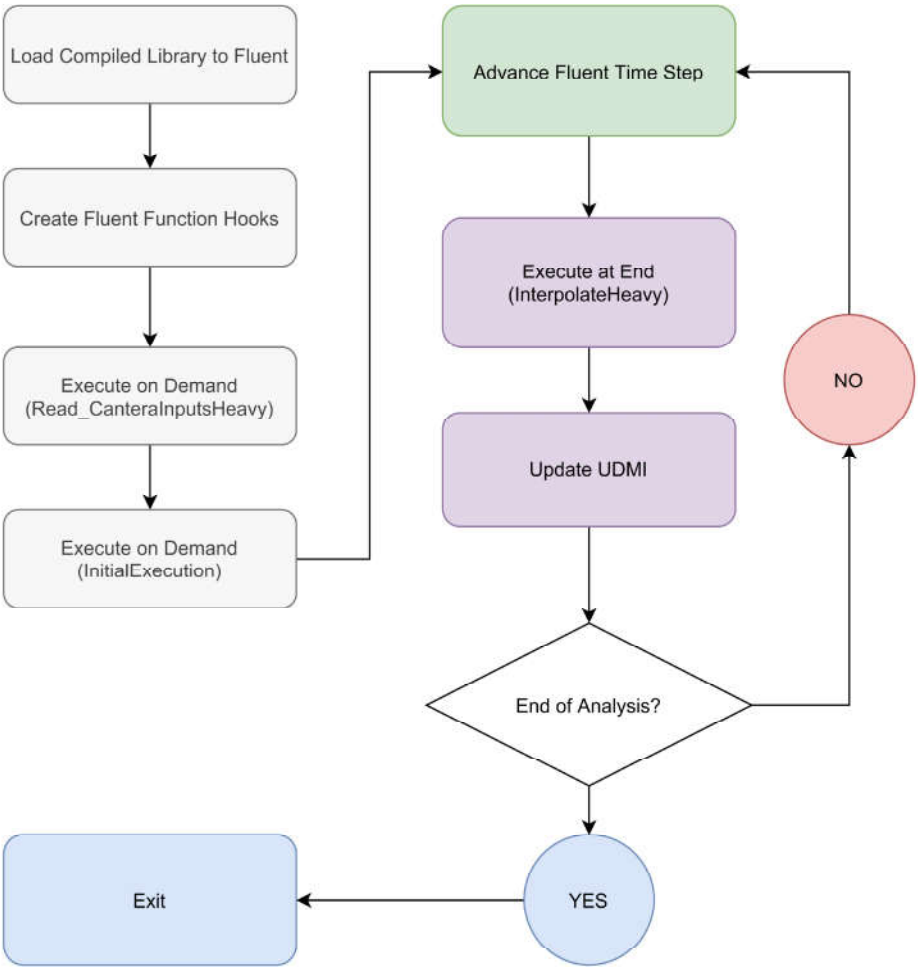

Figure 3.8. Fluent execution pseudo-code.

The first step of using the LS2T outputs is loading the tabular data and executing the script by calling *InitialExecution* on-demand function first. The on-demand function can be executed manually in the Fluent interface or can be automated by including the execution command in the Fluent Scheme or Journal file. As the tabular data loaded, the LS2T methods heavy species contribution calculation scripts are executed by *InterpolateHeavy* user-defined function as another on-demand function. At each solution cell, the outputs of LS2T script are saved to user-defined memory locations (user-defined memory locations) as additional reaction and energy terms to be included while integrating the light species reaction rates by *def_net_RR_wHeavy* user-defined function defined in DEFINE_NET_REACTION_RATE subroutine. After integration of species reaction rate terms, and including the energy source from heavy species, the species fractions are updated. When all the necessary continuum equations are solved,

the LS2T script is executed automatically by including the *InterpolateHeavy* function into the EXECUTE_AT_END subroutine of Fluent. The outputs of LS2T script are then replaced at user-defined memory locations to be used by *def_net_RR_wHeavy* user-defined function defined in DEFINE_NET_REACTION_RATE subroutine. The pseudo-code of the overall process can be seen in Figure 3.8.

3.4. 3D Combustion CFD Modelling

The success of a 3D combustion CFD simulation in duplicating the progress and results relies on the resolution on transport physics, its coupling with chemistry solution and level of detail in the reaction mechanism. The reduced reaction mechanism should be representative to the problem of interest. Nevertheless, the computational model should include the effects of removed species and reactions to maintain the accuracy of detailed chemistry with a reduced reaction mechanism and isolate the transport and chemistry coupling problem. In the case of using LS2T method for chemistry reduction, it is recommended to use a high-fidelity turbulence solution such as LES as in the current work to overcome the lack of heavy species diffusion influence on light species by increasing the fidelity of light species transport calculations [56].

3.4.1. 3D CFD Model for Sandia Flame-D in Ansys Fluent

The Sandia Flame-D is a partially premixed combustion flame of methane and air. The CFD solver employed in this work is Ansys Fluent R19.2 [128]. The turbulence model is selected as LES with kinetic energy transport sub-grid scale model [114] and the species transport model is selected as composition PDF transport with Lagrangian solution [148] where ten particles are used for each cell. The in situ adaptive tabulation algorithm has been applied [51] to reduce the computational time for stiff chemistry calculations. Pressure implicit with splitting of operator [149] has been chosen as the scheme for pressure-velocity coupling since it is recommended for transient solutions due to its additional correction to momentum balance equations [128]. Second or higher-order discretisation schemes are selected where applicable. Bounded second-

order implicit formulation with a fixed time step of 5E-6 s is used for the transient solution with iterative time advancement scheme [128].

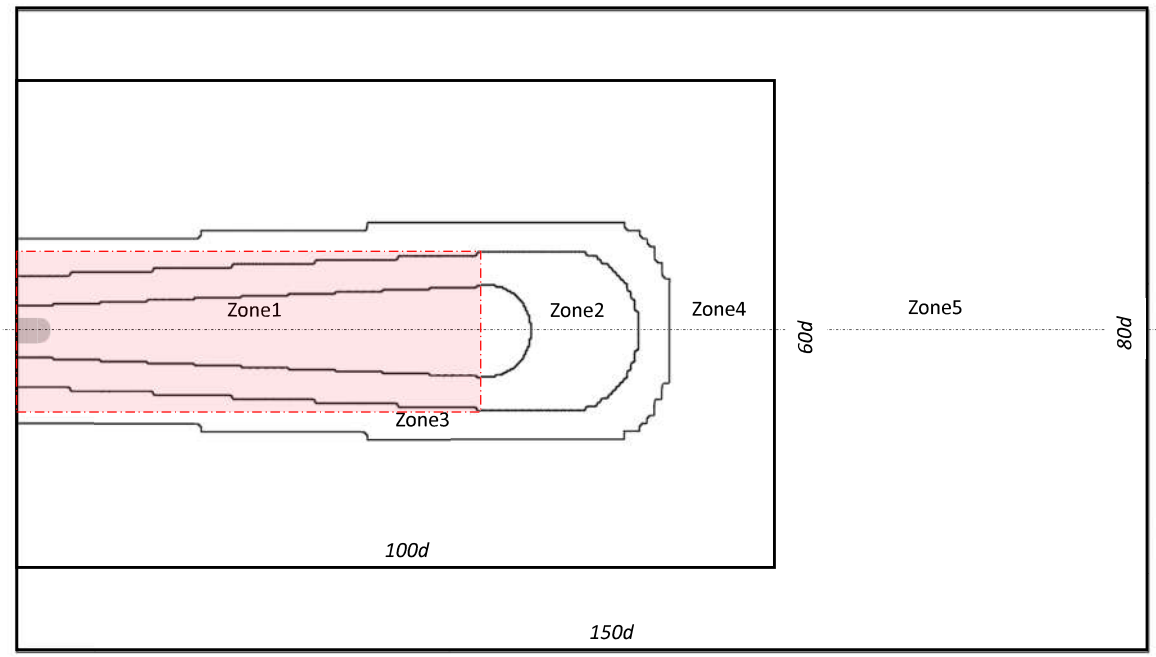

Figure 3.9. Solution domain and the zones separated by the cell size. The red shaded area is the section that is used in plots and figures.

The solution domain shown in Figure 3.9 is created based on the study of Vreman et.al. [6]. The solution domain is extended to 150d in axial and 40d in radial directions with the intention to minimize the effect of walls and outlet boundary condition on the solution. The total number of cells in the solution domain is controlled by introducing axisymmetric mesh refinement zones. The jet and pilot nozzles are covered by a finer zone (Zone 0) to increase the resolution for mixing and velocity gradient. Since the experimental data used for correlation is gathered up to 60d in axial and 15d in radial directions, a second (Zone 1) zone is used to provide adequate refinement within these borders. The rest of the zones are defined to provide a smooth increase in cell size and limit the total number of cells in the domain. Zone 1 is composed of trimmed hexahedral cells with uniform size of 0.9 mm (d/8) equal to minimum cell size used by Vreman et.al. [6]. The size and resolution of all zones are given in Table 3.4.

The boundary conditions are set based on the experimental data where bulk velocities are applied to the inlets, and constant pressure is used for the outlet. The disadvantage of wall effect on the solution domain is eliminated by applying slip (zero-shear) boundary conditions at outer walls and zero-gradient back flow for temperature and species fractions at the outlet.

Table 3.4. Solution domain mesh refinement zones.

Zone	**Radial Dimension [d]**	**Axial Dimension [d]**	**Cell Size [mm]**
0	2	3	0.45
1	R1:3, R2:6	60	0.9
2	R1:6, R2:10	80	1.8
3	R1:7, R2:11	85	3.6
4	Uniform		7.2
5	Extrusion in Normal Direction		7.2

3.4.2. Reaction Mechanism for Methane-Air Combustion

The reaction mechanism used in this study for CH_4 combustion is GRI 3.0 [150] which contains 53 species and 325 reactions and it contains some additional species and reactions for NO formation and re-burn chemistry. The reduced mechanism in LS2T approach contains 26 species (including NO) and 105 reactions. It is quite clear that CH_4 is not a good choice to show the power and benefits of using LS2T approach but due to limited computational power, it is chosen as the development case. In addition, CH_4 chemistry is a subset of heavy carbon species with more carbon atoms and includes most of the common reactions of light species. Based on these facts, it seems necessary to explain the critical elementary reactions of CH_4 combustion in detail since the change in species concentrations along with the CH_4 reaction domain depends on it.

At fuel-rich conditions as in Sandia Flame-D, most of the CH_4 are undergoing radical recombination reactions producing larger hydrocarbons with two or more carbon atoms. The rest of the CH_4 is converted to CH_3(methyl) and hydrogen compounds

by thermal decomposition, oxidation, and hydrogen abstraction reactions. Methyl radical oxidation and recombination reactions and later thermal decomposition of several other radicals produce intermediate CH_2O (formaldehyde) species. The formaldehyde reacts with other radical species producing CO, H_2O, H_2 and OH species rapidly. The CO oxidation reactions producing CO_2 are very slow and also gets inhibited by the presence of small amounts of hydrocarbons due to lack of OH in the environment since most of the CO_2 is created not by oxidation but by the reaction of CO with OH. At high temperature and low pressure (less than 20 atm) combustion of hydrogen and hydrocarbons, H atom is oxidized to produce OH and O atom at chain branching reaction. However, at low temperature (less than 1000 K) and high-pressure H atom oxidation produces HO_2 (hydroperoxyl) at the chain-terminating reaction. The competition between these two reactions results in non-linear dependence of burning velocity and laminar flame speed on pressure independent of the fuel used [121].

3.4.3. 3D CFD Model for Spray-A Combustion in Ansys Fluent

The Spray-A experiment is the non-premixed combustion of dodecane sprayed into air at engine working conditions. The CFD solver employed in this work is Ansys Fluent R19.2 [128], where the turbulence model selected as LES with kinetic energy transport sub-grid scale model [114] and the species transport model selected as composition PDF transport solved with Eulerian field method (multi-environment PDF) [24]. The main reason of selecting multi-environment PDF method is incompatibility of Lagrangian droplet solver with Lagrangian TPDF solver in Fluent. The in situ adaptive tabulation (ISAT) algorithm has been applied [51] to reduce the computational time for stiff chemistry calculations. Pressure Implicit with Splitting of Operator (PISO) [149] has been chosen as the scheme for pressure-velocity coupling since it is recommended for transient solutions due to its additional correction to momentum balance equations [128]. Second or higher-order discretization schemes are selected where applicable and bounded second-order implicit formulation with a fixed time step of 5E-5 s is used for the transient solution.

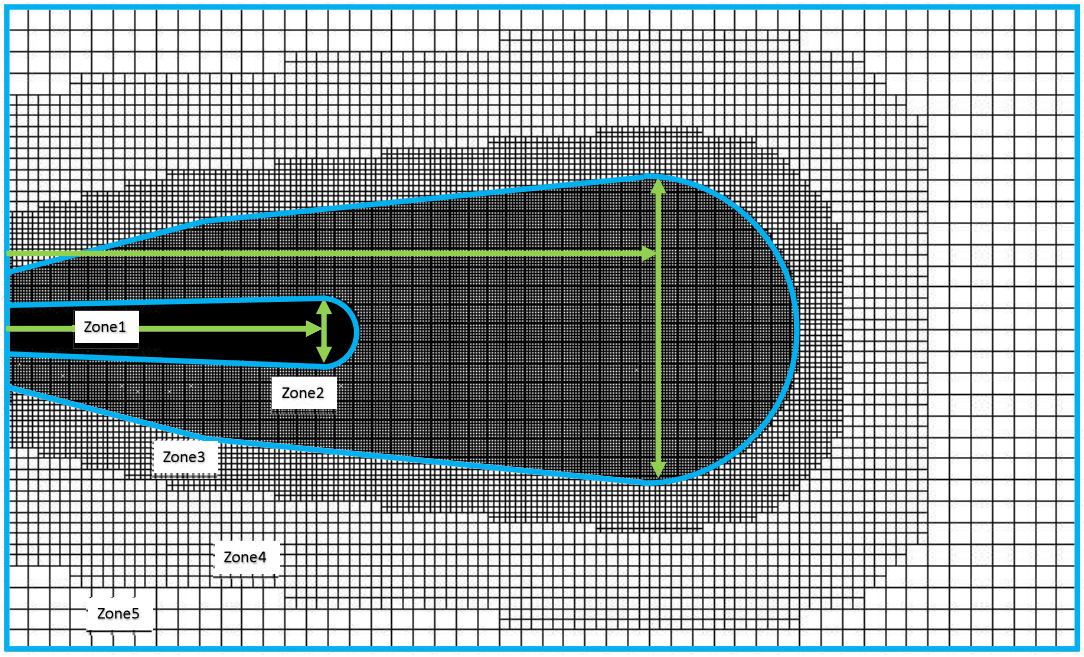

Figure 3.10. Full solution domain and the zones separated by the cell size.

The spray is modeled as Lagrangian particles that are coupled with the Eulerian flow field [125]. The spray at the exit of the nozzle is modeled as a blob (no primary breakup) [1] that is subjected to a secondary breakup due to interaction with the Eulerian field. The secondary breakup model used in this study is the KH-RT model [137] where coefficients are determined both analytically and experimentally. The drag law that is applied to the particles is the dynamic-drag law [130]. In this study, no-collision is applied to the droplet particles. Particles continuity is solved at each time step and coupled with the Eulerian field accordingly.

The spray injection is modeled as a single droplet injected with a constant diameter of 89.4 μm which is the same diameter as the nozzle for blob injection representation. The injection profile for the mass flow rate is retrieved from the virtual injection rate generation tool of CMT-Motores Térmicos [151].

The solution domain of spray simulations is rather tricky to be defined due to conflicting requirements of Lagrangian-Eulerian coupling and LES resolution. Lagrangian-Eulerian coupling requires dispersed phase to occupy a small volume to neglect the particle to particle interactions [152]. As per this requirement, it is advised that the

volume of the droplets that remain in a certain cell should not occupy more than 10% of cell volume. Since the volume of droplets can not be controlled, the cell should be large enough for particles to occupy less than 10% of cell volume. On the other hand, as mentioned in Section 2.6, the resolution of the LES data is determined by the cell size used as the sub-grid scale filter. The cell size can not be larger than the maximum of either Taylor scale or one-tenth of integral length scale for resolution of large eddies by at least ten cells [124].

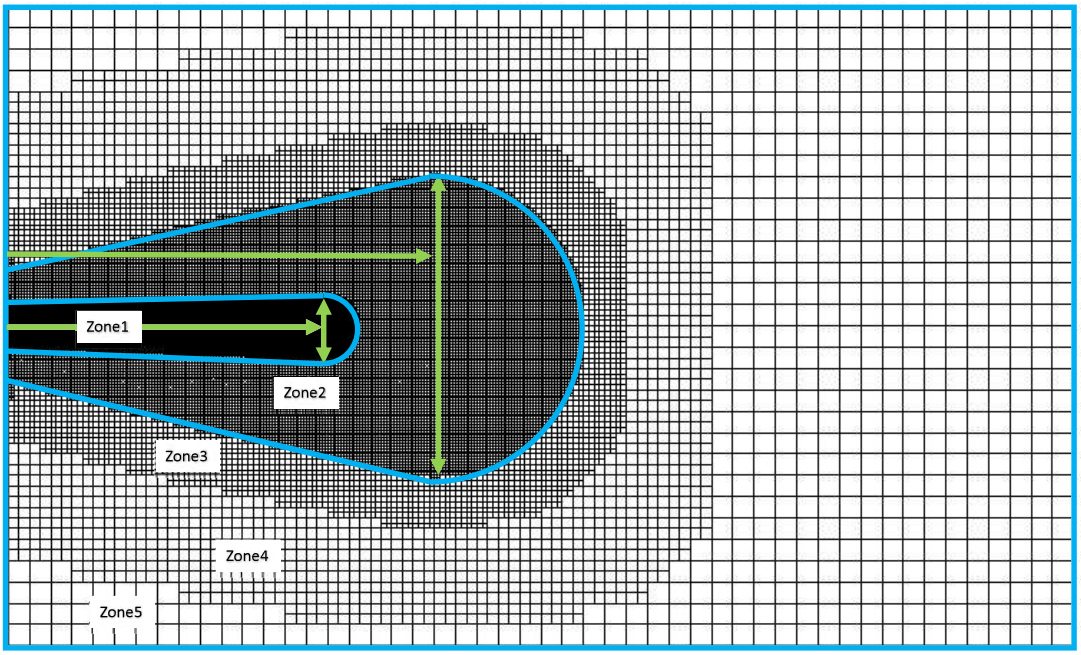

Figure 3.11. Test solution domain and the zones separated by the cell size.

Table 3.5. Refinement zones of Spray-A computational mesh.

Mesh Refinement Zone	Axial Direction for Full [mm]	Axial Direction for Test [mm]	Radial Direction [mm]	Cell Size [mm]
1	0-30	0-30	0-3	0.125
2	0-60	0-40	0-14	0.250
3	0-70	0-50	14-19	0.500
4	0-86	0-66	19-28	1.000
5	0-108	0-100	28-30	2.000

Considering these facts, the refinement of the solution domain is done based on the work of Farrace et.al. [153]. The solution domain shown in Figure 3.10 is a cylindrical domain that extends to 108 mm in axial and 30 mm in radial directions. It is composed of Cartesian cut cell mesh with anisotropic cell size summarized in Table

3.5. The smallest grid size is used around the spray to capture the velocity gradients between the spray and the Eulerian field as accurately as possible, since it affects the droplet drag, breakup behavior, and spray penetration eventually [153]. The grid refinement is reduced gradually towards the walls as the gradients reduce. Due to the limited resources, all the model development studies for mesh size and droplet breakup coefficients, etc. have been conducted using a test domain shown in Figure 3.11. The zone sizes of the test domain (also shown in Table 3.5) is reduced to provide enough data from analyses conducted until 0.5 ms of spray injection. The test domain contains 1.9 million cells while the full domain contains 2.7 million.

3.4.4. Reaction Mechanism for Dodecane Combustion

Dodecane is a heavy carbon species with detailed chemistry composed of more than 500 species and more than 2000 reactions. Using detailed chemistry in the Spray-A CFD problem is impractical since the computational sources are limited. As a result, a HRM of dodecane is used as the baseline in this test case to understand the opportunity of application of LS2T method. For spray combustion modeled using Lagrangian representation, Ansys Fluent can only use the Eulerian TPDF solver. In Fluent, the number of species for the Eulerian TPDF solver is limited to a maximum of 50. The most recent dodecane reduced mechanism of Yao et. al. [35] consists of 54 species and 269 reactions and was specifically developed for spray-A simulation with low ($<$ 900 K) and high-temperature combustion regimes. The low-temperature scheme has been implemented by involving 4 species and 18 reactions for low-T chemistry of n-decane [154]. By removing 4 species and 18 reactions from the Yao reaction mechanism, a mechanism that is compatible with the Fluent solver and suitable for combustion at temperatures over 900 K has been obtained. The other reaction mechanisms available in the literature are listed in Table 3.6.

Table 3.6. Dodecane reaction mechanism.

Designation	**Year**	**Author**	**Number of Species**	**Number of Reactions**	**Ref.**
Krithika-S253	2010	Narayanaswamy et.al.	253	1473	[155]
Som-S103	2011	Som et.al.	103	370	[88]
Sarathy-S151	2011	Sarathy et.al.	151		[156]
Lu-S124	2011	Lu et.al.	124	476	[157]
Luo-S106	2014	Luo et.al.	106	420	[2]
Narayan-S255	2014	Narayanaswamy et.al.	255	1509	[158]
Ranzi-130	2014	Ranzi	130	2395	[159]
Wang-S100	2014	Wang et.al.	100	432	[36]
Yao-S54	2015	Yao et.al.	54	269	[160]
Abianeh-S85	2015	Abianeh	85	266	[161]
Frassoldati-S96	2015	Frassoldati	96	993	[162]
Pei-S2885	2015	Pei et.al.	2885	11754	[163]
Pei-S163	2015	Pei et.al.	163	887	[163]
Cai-S57	2017	Davidovich et.al.	57	217	[100]

4. RESULTS AND DISCUSSION

This study focuses on applying LS2T method to 3D combustion problems which are solved using a high-resolution LES turbulence model. The method is first demonstrated at 0D reactors initialized according to the 3D combustion problem. 0D reactor analyses show the ability and accuracy of the method and make sure that the generated heavy species tables are suitable for the 3D combustion problem. Then it comes to the application of the LS2T method to the 3D combustion problem. The 3D combustion studies require analyses with a base HRM mechanism and must show the accuracy of the base 3D CFD simulation model before going on with the LS2T applied simulation. The final results are demonstrated by comparison of the base 3D CFD problem with the LS2T applied one.

4.1. Partially Premixed Methane Combustion and Sandia Flame-D

4.1.1. Assessment of LS2T Method in 0D Constant Pressure Reactor

The assessment of LS2T method for methane is done at Cantera based on the initial conditions given according to the working conditions of Sandia Flame-D. Figure 4.1 illustrates the comparison of GRI 3.0 methane oxidation mechanism [150] and LS2T method in a constant pressure 0D reactor of Cantera for different equivalence ratios. The reactor model and the initial temperature and pressure conditions (1200 K and 94 kPa, respectively), have been chosen to represent the Sandia flame D experiment. The equivalence ratio values 1.0 and 1.5 are chosen to show that the model is accurate for both stoichiometric and fuel-rich cases. For the 0D example, the grid used to retrieve data is equidistantly spaced from initial (T_0, p_0, ϕ_0) points of each case by ΔT 10 K, Δp 1 kPa and $\Delta\phi$ 0.05. As demonstrated in Figure 4.1, for the stoichiometric case, the final temperature is very close to the maximum temperature that can be reached based on initial conditions. For the fuel-rich case, the temperature gets reduced after peak point due to endothermic reactions activated by pyrolysis [120], resulting

in multi-valued heavy species reaction rate values. To select the proper data in the 0D reactor, the temperature of the reactor is compared with the maximum allowable temperature calculated using the retrieved tabular data surrounding the initial reaction condition (T_0, p_0, ϕ_0). Compared to estimated maximum temperature value, if the reactor temperature overshoots or the time derivative of temperature becomes negative, the model uses the secondary data range where endothermic pyrolysis reactions are dominant.

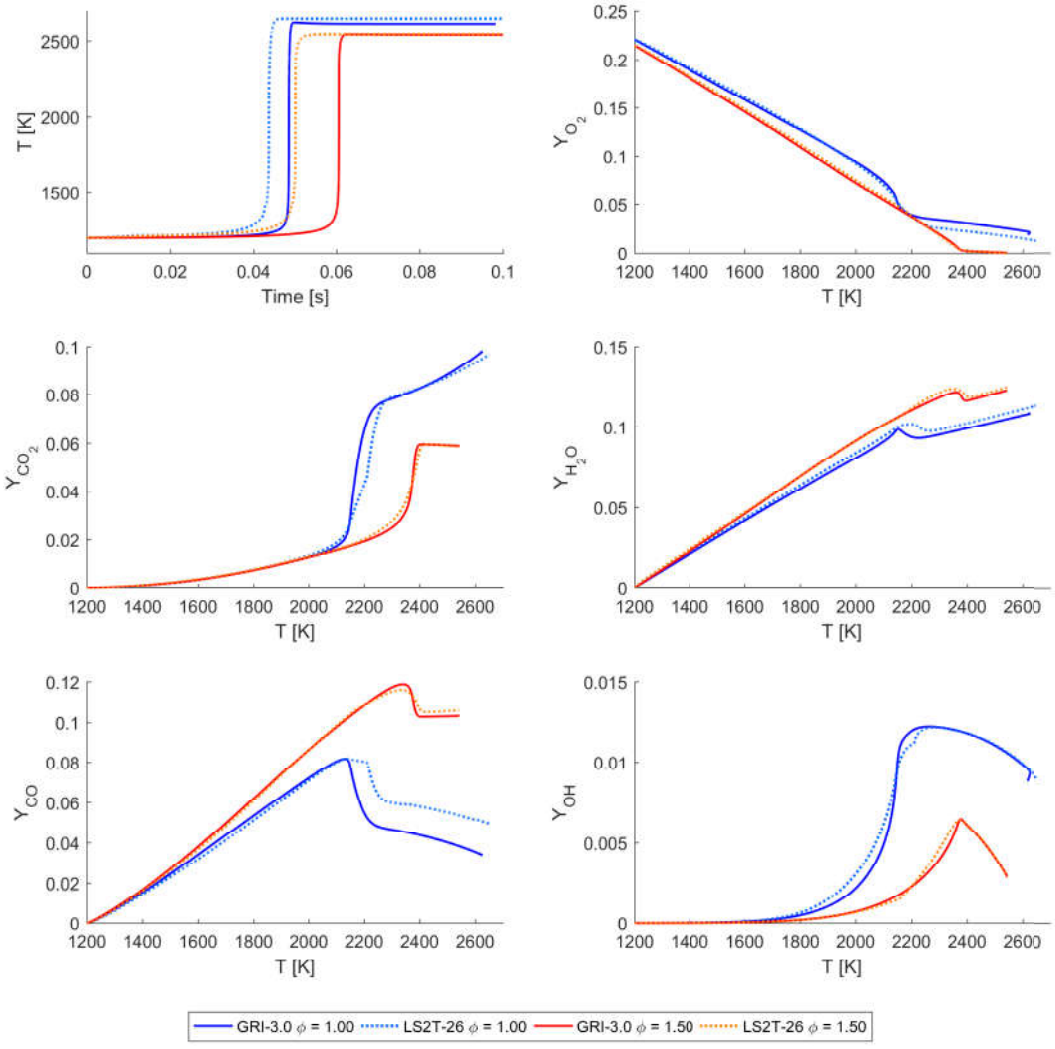

Figure 4.1. Temperature vs time and several major species progress vs temperature in constant pressure reactor for initial temperature 1200 K, pressure 94000 Pa and ϕ 1.0 and 1.5. The comparison is made between the reduced kinetics model LS2T-26 and GRI-3.0 Kinetics Model.

The temperature gradients and the heat release rate values of the LS2T model with 26 species are well matching with the GRI 3.0 mechanism for different equivalence ratio values, but the ignition delays calculated are comparably lower as shown in Figure 4.1. Similar behaviour of underestimating the ignition delay by LS2T method is also evident in the literature [56] for n-heptane, even though the essential chain-branching and chain-terminating reactions are directly solved.

The methane ignition is quite different from other hydrocarbons since its primary radical CH_3 (methyl) is difficult to oxidize. All the oxidation and thermal decomposition reactions of CH_3 are directly solved and several recombination reactions producing larger hydrocarbons with two or more carbon atoms such as ethene, ethylene and acetylene are included by LS2T method. Production of higher hydrocarbons results in chain branching and increase of ignition delay duration of methane. Moreover, these larger hydrocarbon producing reactions get more dominant as the mixture gets fuel richer [121]. During the early stages of oxidation, the modelled heavy species reaction rates ($\dot{\omega}_{lh}$) are underestimated compared to the light species reactions rates ($\dot{\omega}_{ll}$) and this underestimation effect increases as the mixture gets fuel richer. The tabular data are generated and grouped at constant temperature ranges to reduce the data search and retrieval duration and stored as a function of normalised temperature. However, the results indicate that converting time-resolved data to temperature resolved one results in loss of accuracy when the temperature gradients are very low and affect the ignition which is very sensitive to temperature. This situation is explained by departure from excellent self-similarity and having a very high sensitivity to temperature for a very short period. In the literature it is reported that the results were insensitive to the refinement of the tabulation grid used and the difference in ignition time was as high as 30% for iso-octane [164].

Considering all the plots in Figure 4.1, it can be concluded that the rate of change of species mass fractions plotted against the temperature are in a good agreement with detailed GRI 3.0 mechanism results. At $\phi = 1$, the distinct change in mass fraction gradients of O_2, CO_2, CO, OH after 2100 K are well captured. Similarly, at $\phi > 1$

the point where O_2 is completely consumed calculated accurately. In addition to the change of production rates after specific temperature values based on initial ϕ, the reduction in CO_2 production in favour of CO production as ϕ increases are modelled accurately. For the $\phi = 1.5$ case, the reduction in H_2O, CO_2 and OH concentrations after maximum temperature points are also very well captured, indicating how accurate the tabulation and data retrieval methods are [120].

4.1.2. Assessment of LS2T Method by 3D LES Analysis of Flame D

The effect of reduced chemistry by LS2T method on the results and its accuracy can only be shown by first indicating how accurate the LES turbulence and multi-environment PDF species transport models are when detailed GRI 3.0 chemistry is used and then compare the results of reduced chemistry by LS2T with detailed chemistry solution. Accordingly, two CFD models prepared with only difference in the number of species and reactions. For the sake of clarity, over the graphs, the model with detailed GRI 3.0 mechanism is denoted as GRI3.0 and the model with reduced chemistry and LS2T method is denoted as LS2T. The Favre averaged values of the experimental data [59] are used for correlation with root mean square (RMS) values printed over each data point as error bars to indicate the measurement uncertainty of the experiment. Several researchers also mentioned that because of the measurement uncertainty at rich zones of the flame, the experimental data shows under-predicted RMS mixture fraction and broad scatter of temperature [5, 64]

The application of LS2T method to the 3D CFD problem is demonstrated in Figure 4.2. The range of applicability of the tabular data generated for LS2T is evaluated by checking each cells T, p, ϕ values and comparing with the range of data in tables. As shown in Figure 4.2a by red color, almost all the flame is encapsulated by the table, except the downstream of the inner flame due to lack of oxygen and very high ϕ. To apply the heavy species effect in the reactive area of the domain only, the application of the LS2T is initiated within the flammability limits of fuel and reactions finalized by lack of oxygen and fuel in the cells. Figure 4.2b shows the reaction zone where the

LS2T is active and if it is using first or second set of multi-valued data considering the temperature [25]. Due to the strong turbulence-chemistry interactions the reaction zone is thinner at just downstream of the pilot and it gets thicker upstream [61]. Figure 4.2c expresses the heavy species contribution($\dot{\omega}_{lh}$) to reaction rate of light species OH ($\dot{\omega}_{ll}$) and presents the resulting OH species instantaneous mass fraction at the end of fourth flow trough time (FTT).

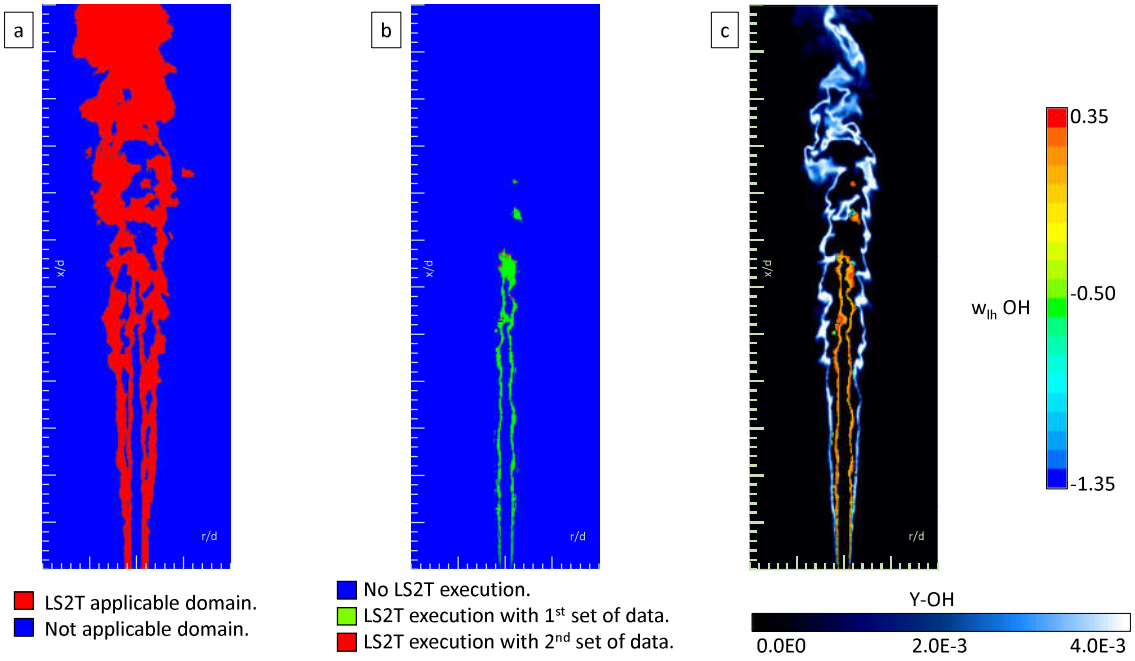

Figure 4.2. LS2T execution information at 0.060s: a-Domain where the tabular data exist based on the T_0, p_0, ϕ_0, b-Domain where LS2T is executed, c-Sample heavy species reaction rate contribution to OH printed over instantaneous OH mass fraction distribution.

The Favre average flow data for 3 FTT in Figure 4.3 shows adequate resemblance with experimental images [59] for both GRI3.0 and LS2T analysis. Both models are able to simulate the jet and pilot flow interactions and laminar characteristic of the flame until 10 x/d without any local extinction (Figure 4.4) as in the experiment. Both models can predict the flame length and distribution as can be inferred from average OH distribution, but CO concentration is higher at LS2T models upstream. The distribution of average flame temperature around 40x/d is well predicted for both but the maximum temperature reached by LS2T model is higher than GRI3.0 model

as in Figure 4.3. The slight difference between the radial dissipation of the species is probably due to earlier reaction and increased advection accordingly.

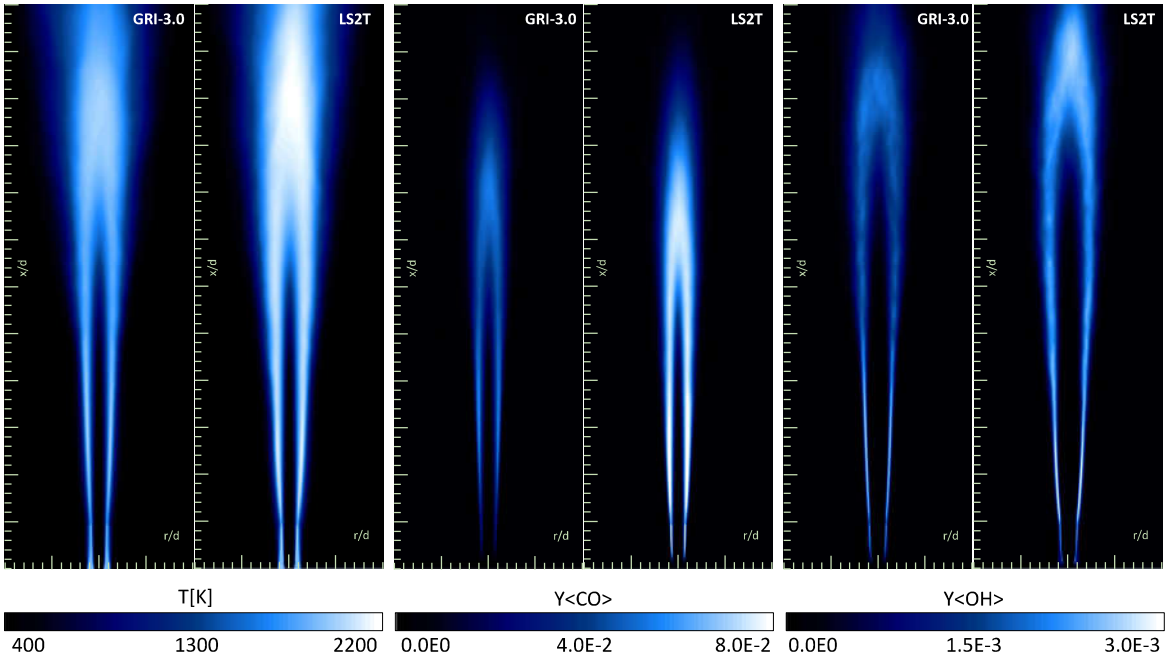

Figure 4.3. Favre averaged temperature and composition of CO and OH for the Sandia Flame D for 3FTT.

The instantaneous temperature distribution of GRI3.0 model agrees well with the literature [7,67,76] where the flame evolution is slightly different for LS2T model as can be seen in Figure 4.4. In the LS2T model, the flame evolution is faster and early flame initiation results in early temperature increase along the flame compared to GRI3.0 model. Shorter ignition delay caused by LS2T method initiates early consumption of hydrocarbon and higher CO concentration until 15 x/d. However, the balance between CO and OH species is captured fairly good that as the concentration of CO decreases the OH concentration increases which indicates that the LS2T contribution to species maintains the balance between them.

Figure 4.5 shows the axial distribution Favre averaged mixture fraction, temperature and species composition values averaged for 3 FTT. The mixture fraction and temperature is predicted within the RMS error limits for both GRI3.0 and LS2T models, but the temperature is slightly overshooting for both compared to the experimental

data after the temperature peak point at 50 x/d. Temperature of LS2T model is about 50 K higher than GRI3.0 model along the axis due to early ignition and the maximum temperature values are also different due to LS2T models higher reaction rates. In addition to high temperature, lower O_2 concentration after 50 x/d and higher product concentrations also indicate the mixing is not well captured beyond at about 50 x/d.

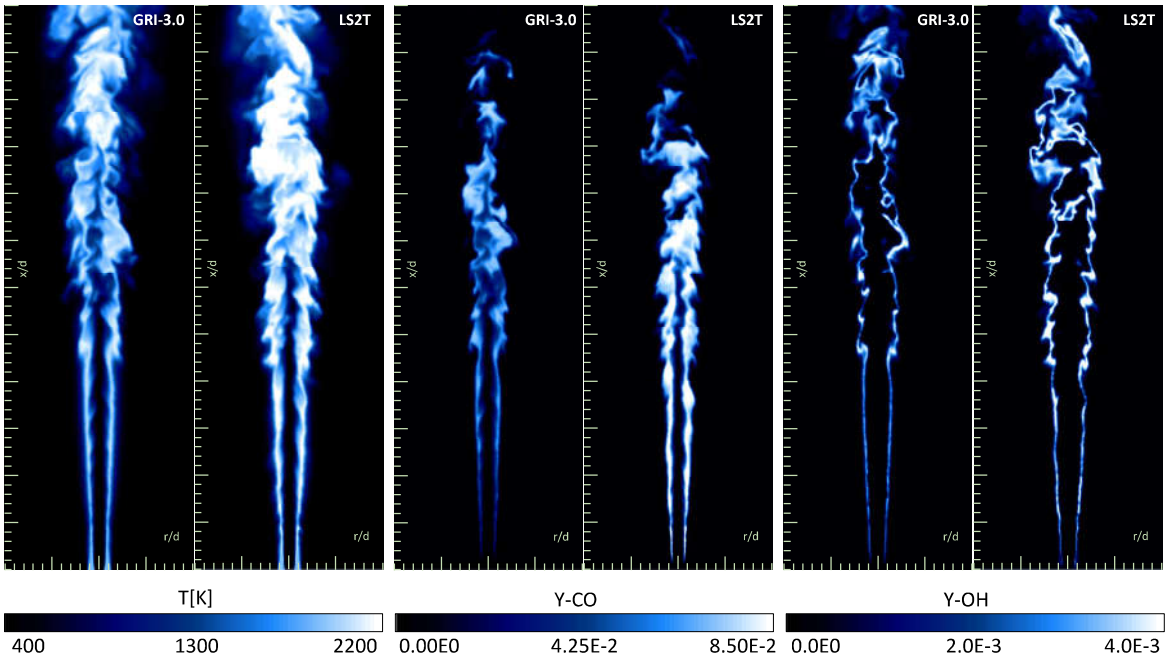

Figure 4.4. Instantaneous temperature and composition of CO and OH at the end of 0.060s for the Sandia Flame D.

Both models CH_4 and O_2 consumption and final H_2O and CO_2 productions along the axis are well aligned, and the production and destruction of CO, OH are also captured well. At about 40 x/d, where the OH concentration increases the concentration of CO decreases due to reacting with OH for CO_2 production. When the NO concentration is considered, the results are not that well for both models. The NO concentration at GRI3.0 model is matching well until the maximum temperature point at 40 x/d, but due to mixing, the NO concentration continues to increase while it is getting diluted in the experiment. The LS2T model could not resemble the NO distribution because the NO production is higher at the downstream of LS2T model and it is getting diluted faster than GRI3.0 due to longer mixing length.

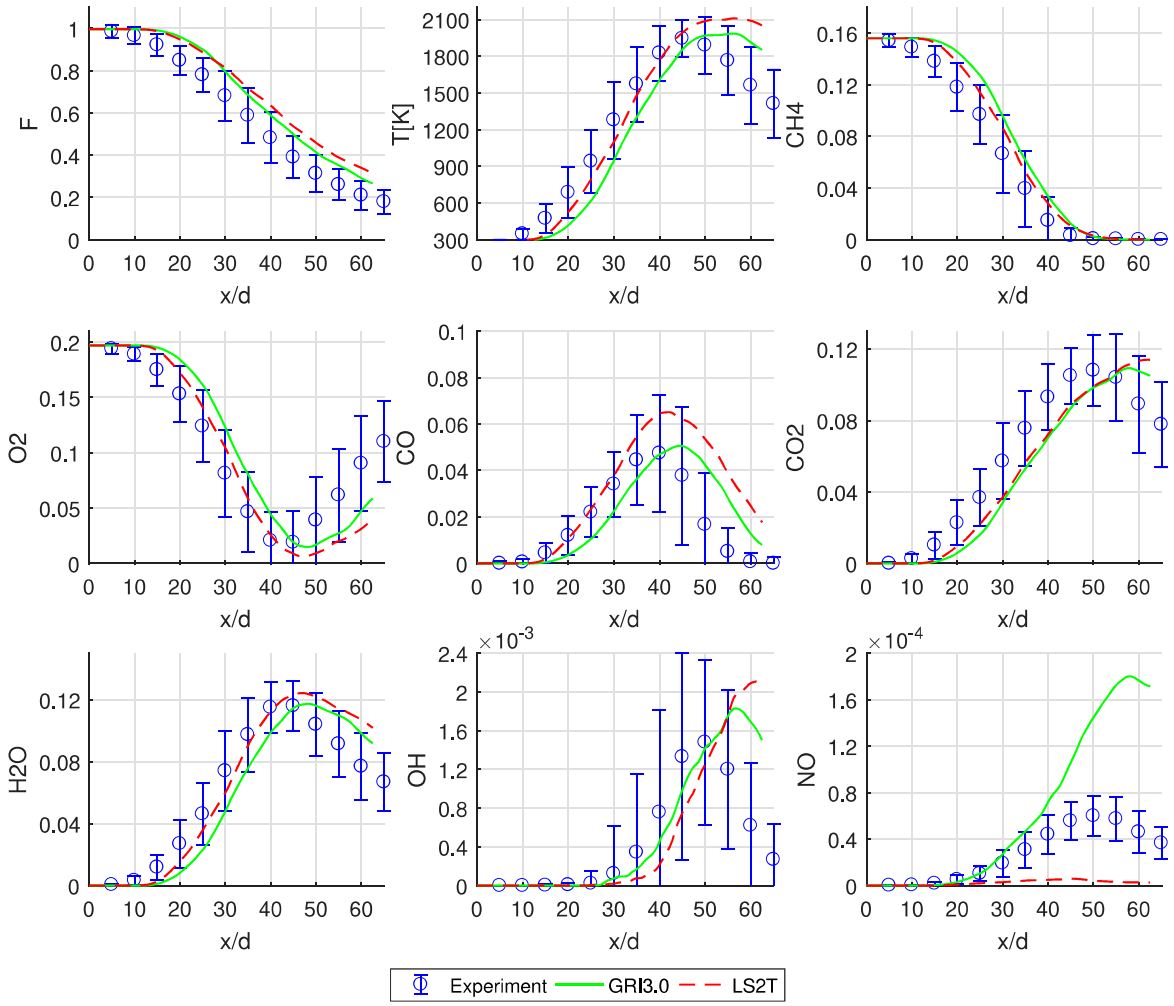

Figure 4.5. Favre averaged mixture fraction, temperature and species composition profiles along the axis for the Sandia Flame D.

The mixture fraction and temperature change along radial direction at certain axial locations can be seen in Figure 4.6. The mixture fraction is well captured at 15, 30 and 45 x/d locations quite well with GRI3.0 model and acceptably good with LS2T model. The temperature distribution is within the RMS limits for GRI 3.0 showing slightly higher temperature at mid radii and lower temperature values at the outer radii. As mentioned before due to higher reaction rates and early ignition at downstream of the flow, the temperature increases faster than the experiment with LS2T model as can be seen at 15 x/d, but the temperature difference gets smaller at x/d 30 and 45 respectively due to mixing.

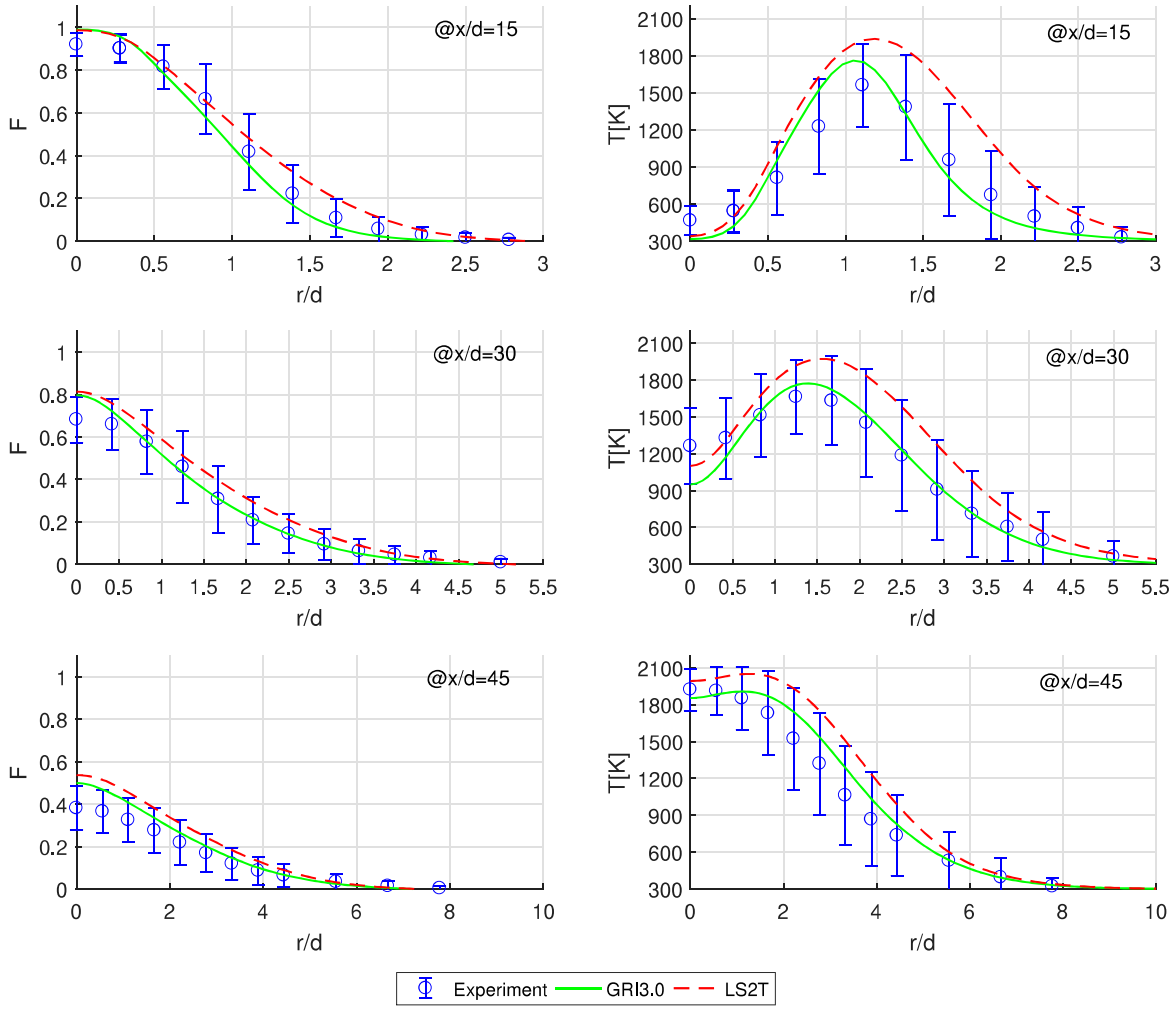

Figure 4.6. Favre averaged mixture fraction and temperature along the radial direction at axial distance of x/d 15, 30 and 45 for the Sandia Flame D.

Figure 4.7 and 4.8 shows that GRI 3.0 model also successful capturing the transport and radial distribution and of CH_4, O_2, H_2O, CO, CO_2 and OH species all within RMS fluctuation range compared with the experimental data. However, the higher downstream reaction rates results in the lower O_2 concentrations and higher product species (H_2O, CO, CO_2 and OH) concentrations at 15 x/d but the difference is getting smaller when the distributions at x/d 30 and 45 investigated. The CH_4 distribution is almost same for two models along the radii as shown in 4.7.

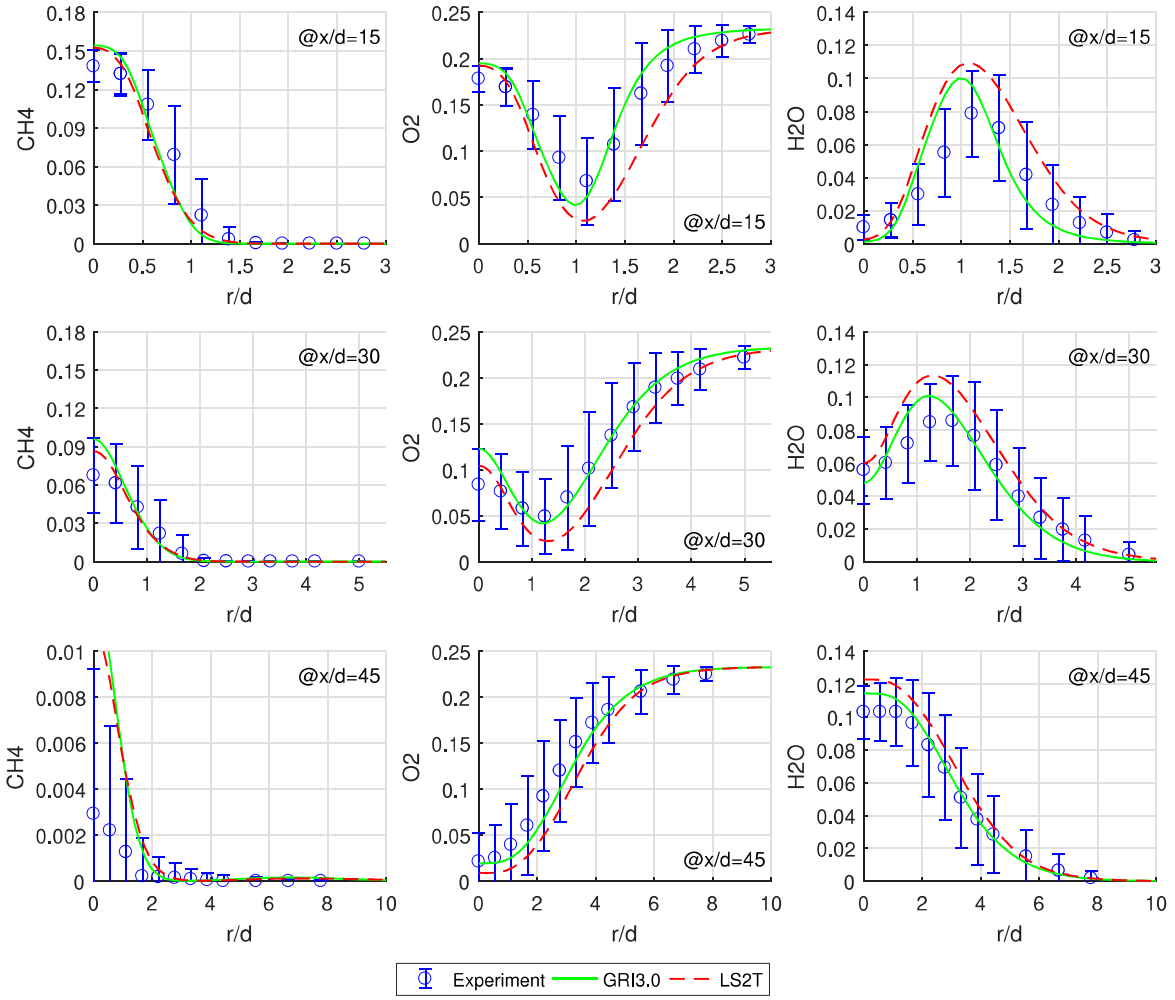

Figure 4.7. Favre averaged composition of CH_4, O_2 and CO_2 along the radial direction at axial distance of x/d 15, 30 and 45 for the Sandia Flame D.

Figure 4.8 shows the species CO,CO_2 and OH that are in close relation with each other chemically. The species reaction kinetics is accurately captured. Since most of the CO_2 is created not by oxidation but by the reaction of CO with OH [121], the increase in CO_2 concentration with the presence of CO and OH and reduction as their concentrations get reduced can be clearly seen for each axial location. The concentrations of CO_2 and OH are well within RMS limits with the difference from GRI3.0 mechanism gets reduced moving to upstream but the higher concentration at the outer radii arising from higher species diffusion in radial direction. The most distinct difference between the results can be seen in the radial distribution of CO at 15 x/d which is mainly due to higher reaction rates at the downstream of the flow.

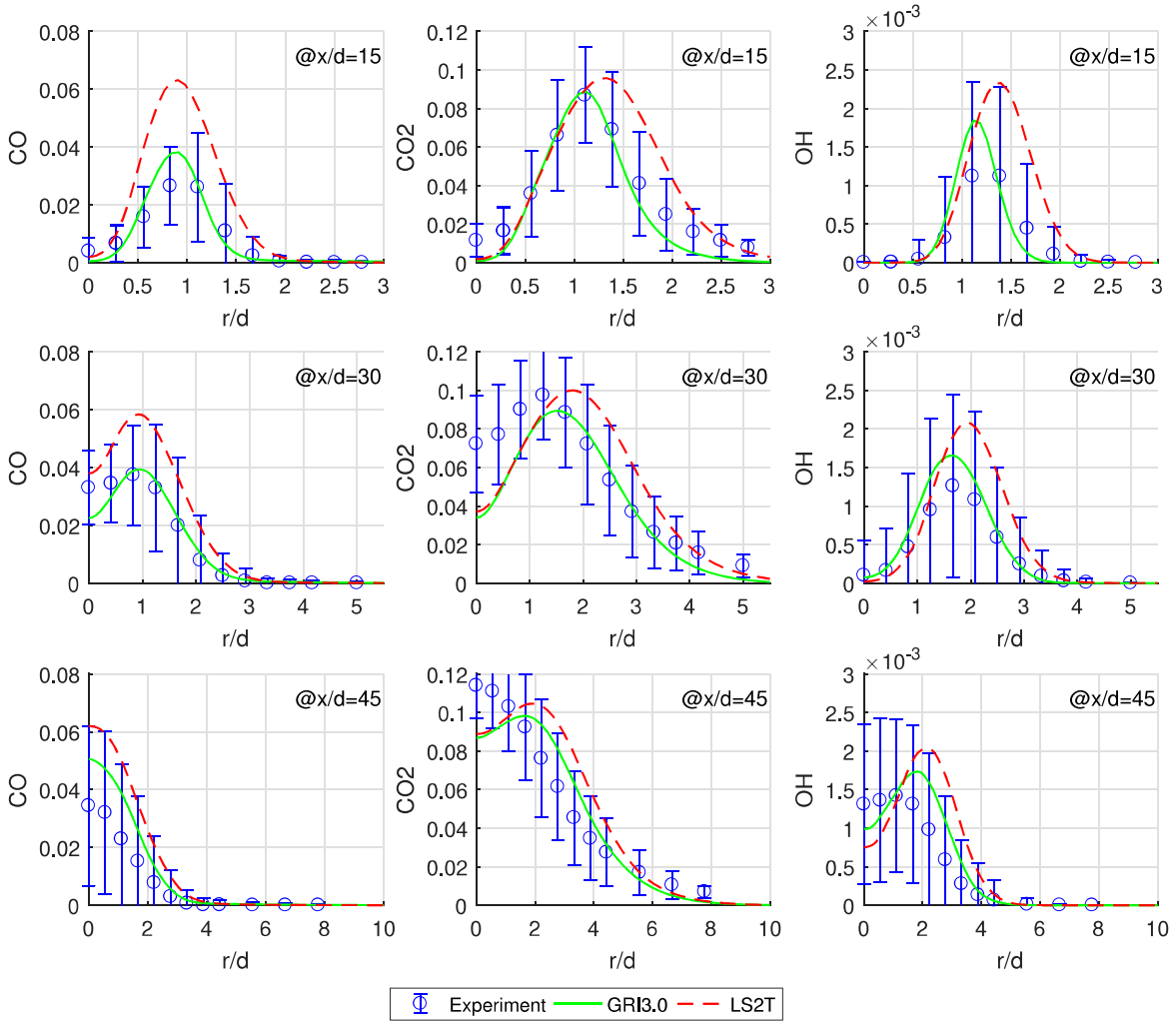

Figure 4.8. Favre averaged composition of CO, OH and H_2O along the radial direction at axial distance of x/d 15, 30 and 45 for the Sandia Flame D.

4.1.3. Computational Efficiency

The main purpose of applying LS2T method to 3D LES combustion simulation is to reduce the computational time for stiff chemistry integration but maintain a reasonable accuracy in the results. Figure 4.9 shows the computational time spent for solution at each time step of an already developed combustion flow of cases with GRI3.0 and LS2T approach. The GRI3.0 reaction mechanism of methane is composed of 53 species and 325 reactions and LS2T method with 27 species and 103 reactions. For LS2T the stiff chemistry integration is completed in about one third of the cpu time of GRI3.0, but a similar cpu time is spent for calculations specific to LS2T. The

values reported are in CPU hours to avoid the confusion by number of CPUs used. The total computational duration for GRI3.0 and LS2T are 2.5 and 1.8 CPU-h respectively.

Figure 4.9 is a representation of how efficient the LS2T method is for reaction mechanism reduction. The total computational duration reduced mainly by the time spent for the integration of 325 reactions and by the reduction in number of transport equations to be solved. However, LS2T model requires time to be spent for tabular data search, interpolation of Point of interest, inverse distance weighting, and rest of the arithmetic operation and the duration of LS2T execution is taking longer than reduced chemistry integration. The tabular data search is the most time-consuming stage of LS2T but as can be seen from the results the time spent is considerably low.

	GRI3.0: 53 Species, 325 Reactions	LS2T: 26 Species, 105 Reactions
Continuity Solution	0.4	0.2
LS2T	0.0	1.0
Stiff Chemistry Integration	1.9	0.6

Figure 4.9. Computational duration for single time-step.

4.2. Non-Premixed Dodecane Combustion and Spray-A

4.2.1. Assessment of LS2T Method in 0-D Constant Pressure Reactor for Dodecane Combustion

Dodecane has several reaction mechanisms that can be used for the assessment of the LS2T method. Since the number of species that can be used in Fluent is limited to 50 [128], the reaction mechanism of Yao et. al. [35] (Yao-Mechanism) is chosen to be used in the 3D combustion problem after excluding the low temperature reaction terms to reduce number of species to 50. The power of LS2T approach lies in the ability to reduce the number of species and reactions significantly but the reaction

mechanism of Yao et.al. [35] is not suitable for demonstrating the heavy reduction ability of LS2T. The reaction mechanism that would benefit most from the LS2T method is the one developed by Pei et.al. [163] which consists of 2885 species and 11754 reactions. However, even at a 0D combustor, this mechanism is not practical for development purposes because of the high computational effort it needs to create the necessary input tables for the application of LS2T.

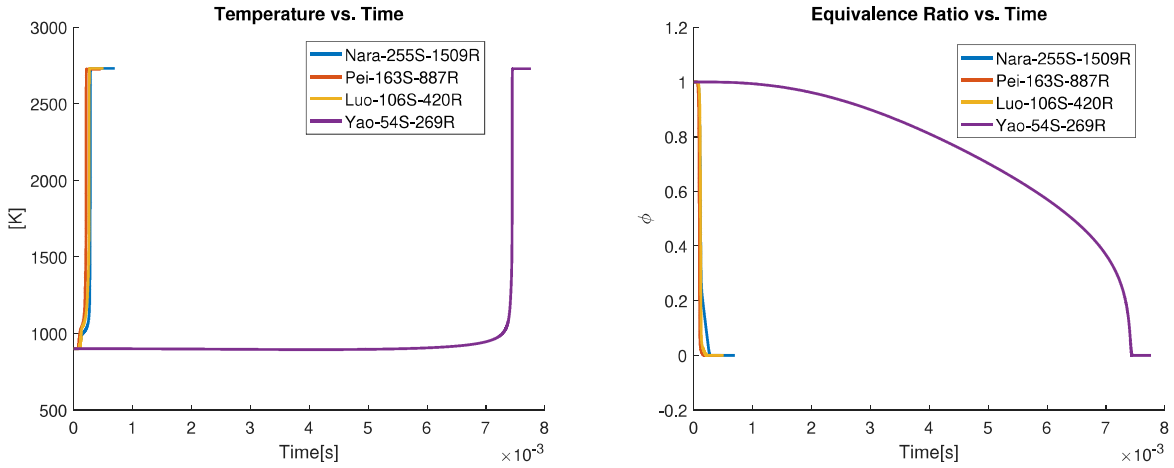

Figure 4.10. Temperature and equivalence ratio progress of several dodecane reaction mechanisms in a constant pressure reactor for the initial temperature of 900 K, pressure of 6 MPa and ϕ of 1.0.

The temperature and equivalence ratio change of available reaction mechanisms for dodecane combustion are plotted in Figure 4.10. The temperature plot in Figure 4.10 shows a significant delay in dodecane combustion with Yao-Mechanism indicating a lower heat release rate due to slower reactions. As can be seen from the equivalence ratio change plot on the right, the consumption of fuel is significantly slower than the other three reactions. In addition to that, when left plot in Figure 4.11 is investigated, the mechanisms of Luo et.al. [2], Pei et.al. [163] and Narayanaswamy et.al. [158] shows a stepped temperature increase behaviour, where at about 1000 K, the temperature gradient decreases since the importance of some reactions above 1000 K reduces significantly and high temperature very short time scaled reactions becomes dominant [121]. The mechanism of Narayanaswamy et.al. also shows an abrupt change in gradient of equivalence ratio in the right plot of Figure 4.11.

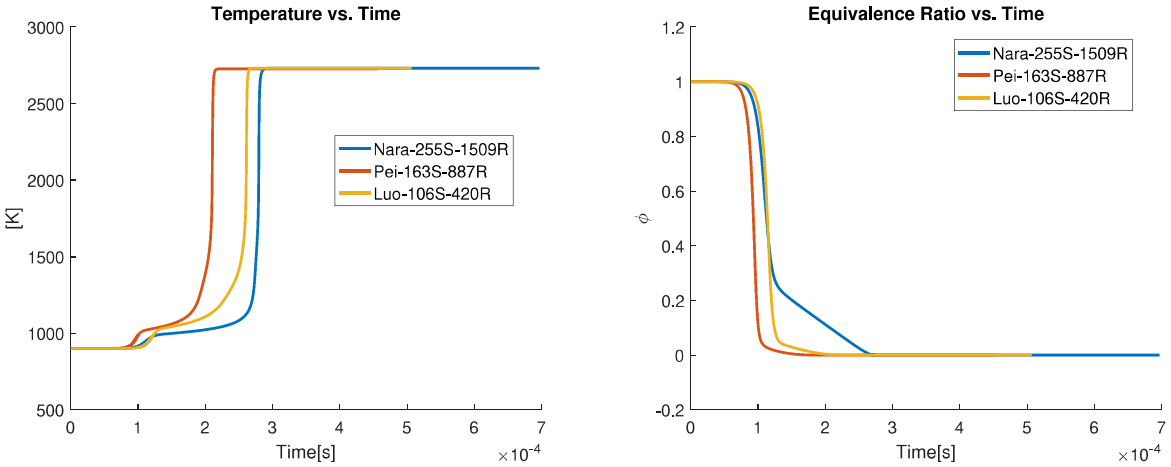

Figure 4.11. Temperature and equivalence ratio progress of several dodecane reaction mechanisms (except Yao-mechanism) in a constant pressure reactor for the initial temperature of 900 K, pressure of 6 MPa and ϕ of 1.0.

When the equivalence ratios are plotted against temperature, the difference in reaction paths of all four reactions is seen clearly in Figure 4.12. The low-temperature terms from the Yao-Mechanism are removed based on the information that those equations are not significant at temperatures above 900K [35]. The remaining high temperature reactions of Yao-Mechanism result in a negative temperature gradient and most of the fuel is consumed during this negative gradient region. This also creates an unusual multi-valued reaction rate problem with respect to temperature and is incorrect because dodecane combustion at this initial condition is a single-valued problem. The difference in this behavior arises from the pressure of the Yao-mechanism is tested. The system pressure divides the high and low-temperature regimes [165]. When the low-temperature reactions were determined by Yao et.al. [35] and the limit temperature was found as 900 K the system pressure used for testing was 20 bars but the Spray-A test is conducted at 60 bars. Based on the behavior of other reaction mechanisms as seen in Figure 4.11, the temperature that decides low and high-temperature reactions is about 1000 K at 60 bars. No effort has been spent to investigate the 50 species mechanism and 54 species mechanism of Yao et.al. [35] in detail (by running both mechanisms at 20 and 50 bars and comparing the results) because the original 54 species mechanism can not be used in Fluent [128] and it is certainly not necessary to be used for LS2T assessment. Since Yao-Mechanism has been used in the 3D combustion analysis, we

should see an increase in the ignition delay.

The reaction mechanisms of Luo et.al. [2], Pei et.al. [163] and Narayanaswamy et.al. [158] results in similar reaction progress as in Figure 4.12. Luo-Mechanism requires the least computational effort among all other reaction mechanisms (except Yao-Mechanism) is most suitable for method development purposes, so the mechanism of Luo et.al. [2] (Luo-Mechanism) which contains 106 species and 420 reactions, is chosen to be used for assessment of LS2T approach with dodecane.

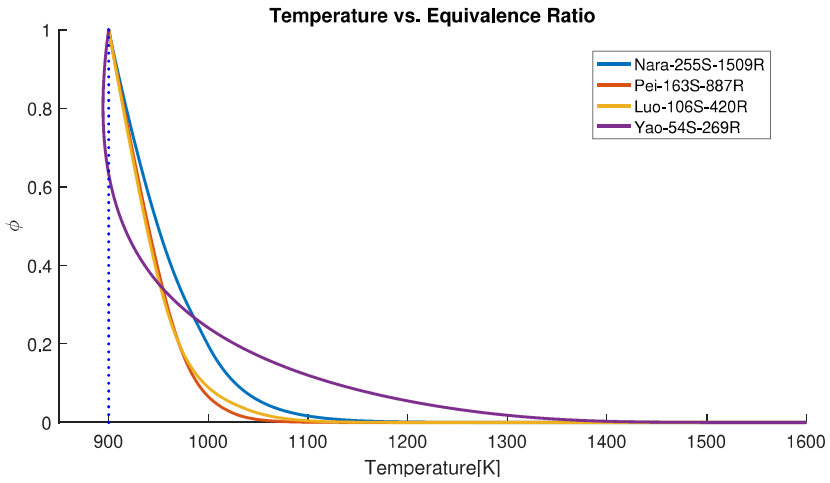

Figure 4.12. Equivalence ratio vs. temperature of several dodecane reaction mechanisms in a constant pressure reactor for the initial temperature of 900 K, pressure of 6 MPa and ϕ of 1.0.

The application of the LS2T method to dodecane chemistry is more complex and difficult compared to methane chemistry. Since the light species selected are almost half of the reacting species, the contribution of heavy species on light species reaction rates and total heat release is much lower in methane combustion than the dodecane combustion. Figure 4.13 shows the reaction rate of n-dodecane fuel and oxygen molecule plotted by normalized temperature values. ω_{ll} refers to the reaction rate calculated from reactions that contain only light species and ω_{lh} refers to the reaction rate contribution of reactions that contain heavy species also. As seen on the left plot, the ω_{ll} value of dodecane is zero, meaning that, all the reactions that contain dodecane molecule also contains a heavy species. This is correct because the first stage

of heavy carbon oxidation is fragmentation of fuel molecule to intermediate(heavy) species sequentially to be converted to final products [121]. So, the reaction rate value of dodecane is retrieved from tables of LS2T method and included as ω_{lh} as shown in Figure 4.13. This is not the case for all light species also as can be seen for the Oxygen molecule at the right plot of Figure 4.13. For oxygen, the reaction rate value calculated from reactions with light species ω_{ll} is significantly higher than ω_{lh} values. Based on the reaction mechanism, the dominance of a species ω_{ll} and ω_{lh} differs.

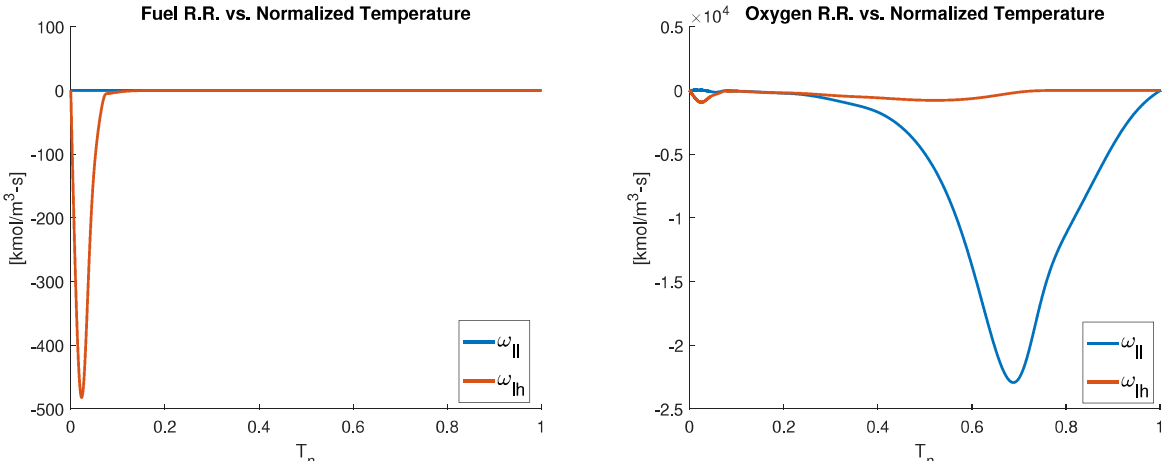

Figure 4.13. Reaction rate vs. normalized temperature of fuel and oxygen molecule for reaction mechanism of Luo et.al. [2] in a constant pressure reactor for the initial temperature of 900 K, pressure of 6 MPa and ϕ of 1.0.

The assessment of LS2T method for dodecane is done at Cantera based on the initial conditions given according to the working conditions of Sandia Spray-A. For the 0D example, the grid used to retrieve data is equidistantly spaced from initial (T_0, p_0, ϕ_0) points of each case by ΔT 25K, Δp 3kPa and $\Delta\phi$ 0.05.

Figure 4.14 illustrates the temperature comparison of Luo-Mechanism [2] and LS2T method in a constant pressure 0D reactor of Cantera for initial temperature of 900 K, pressure of 6 MPa and ϕ of 1.0. For the stoichiometric case, both the ignition delay and the final temperature is well matching with the maximum temperature that can be reached based on initial conditions. While running the analysis it is seen that after about 1040 K, the LS2T script was unable to find surrounding data points due to ϕ value at current point of interest, and as a result started to work with the nearest

points. The deviation in equivalence ratio arises from deviation in Oxygen mass fraction as it can clearly be seen from mass fraction of Oxygen in Figure 4.15.

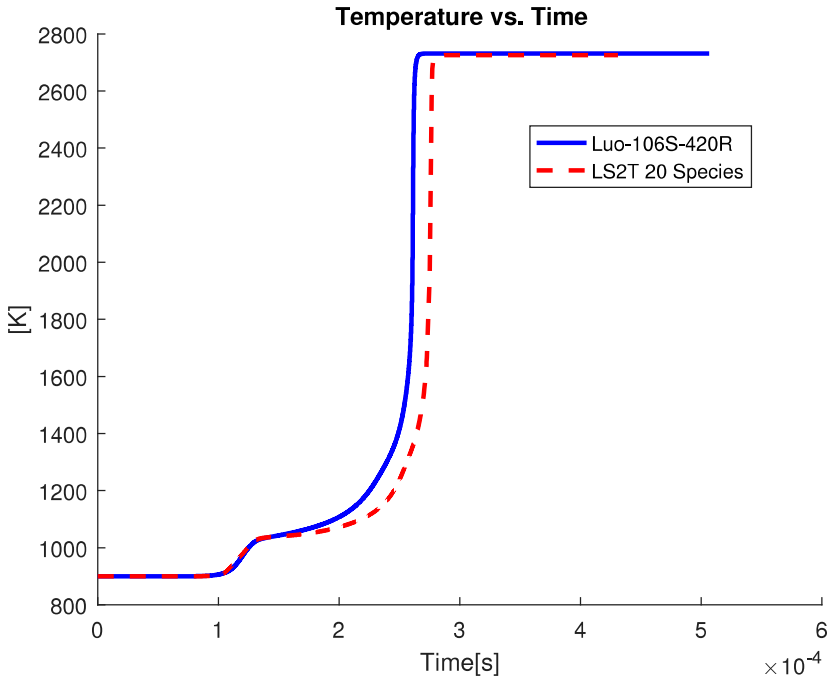

Figure 4.14. Temperature vs time in constant pressure reactor for initial temperature 900 K, pressure 6 MPa and ϕ 1.0. The comparison is made between the reduced kinetics model LS2T-20 and Luo et.al. [2] reaction mechanism.

The species mass fractions could not be caught as good as the results reported by Kourdis and Bellan [25]. The main reason for the difference is the nearest point interpolation of the LS2T method due to missing points in the tabular data. The reaction paths show some difference concerning the temperature but final mass fractions of the species calculated by the LS2T are matching well for CO_2, CO. The progress of O_2, H_2O and OH by LS2T method shows similarity to Luo-mechanism. The results of 0D dodecane combustion can be further improved by generating new tabular data or improving the search and find algorithm.

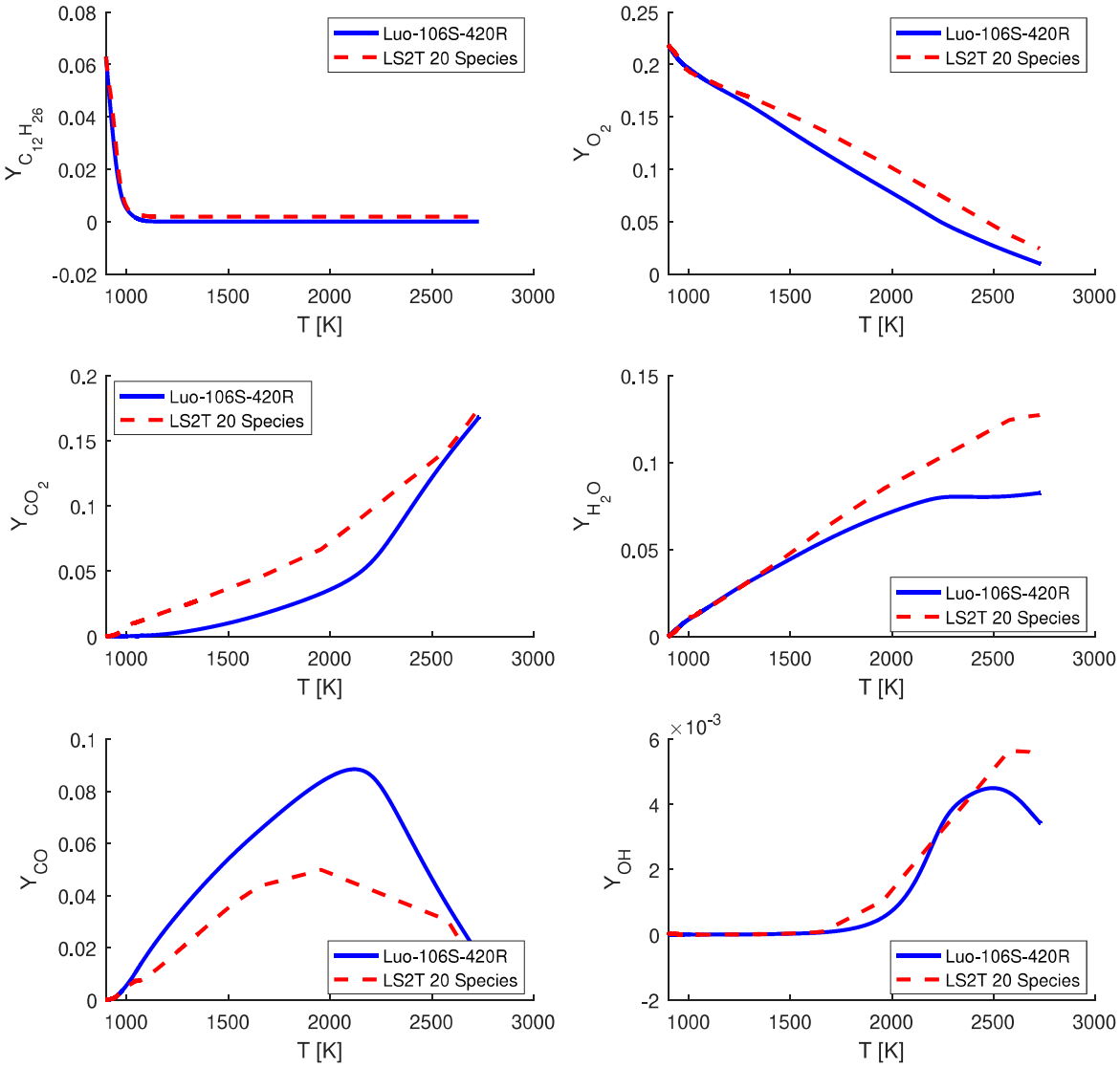

Figure 4.15. Several major species progress vs temperature in constant pressure reactor for initial temperature 900 K, pressure 6 MPa and ϕ 1.0. The comparison is made between the reduced kinetics model LS2T-20 and Luo et.al. [2] reaction mechanism.

4.2.2. Spray-A at Inert Environment

Spray flow experiment of dodecane at the inert environment is used to determine the most proper CFD model settings for spray flow without including the effects of combustion to provide the accurate prediction of air-fuel distribution. The spray penetration, dodecane vapor penetration, mass fraction, and velocity data at a certain location and time are used for validation of the simulation. For statistical independence

of the instantaneous LES data, at least 6 realizations are needed if each realizations average is calculated using 45° azimuthal sectors [153]. Due to limited computational resources, only 4 realizations of Spray-A simulation are used for post-processing.

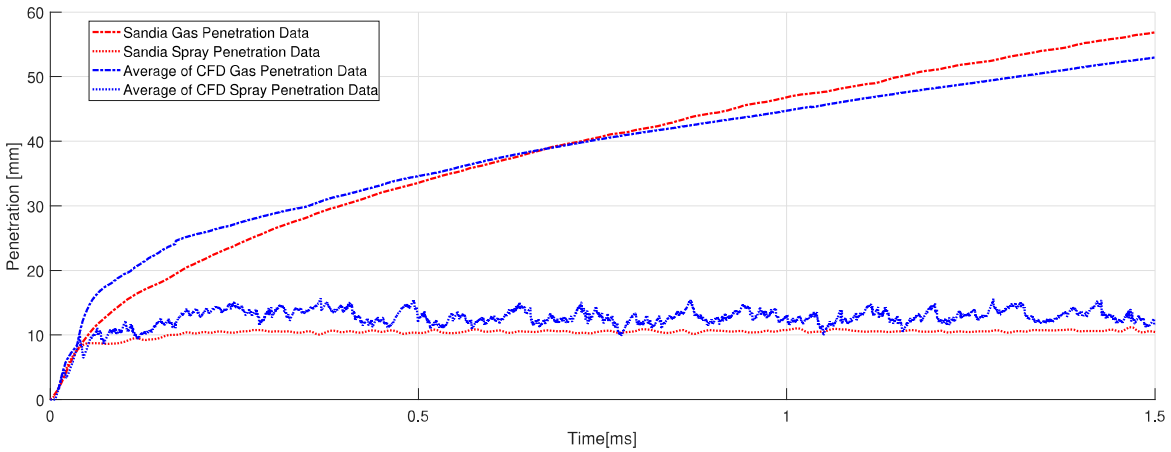

Figure 4.16. Spray and gas penetration of Spray-A at inert environment until 1.5ms after start of injection.

The liquid-phase measurements are done by either Mie-scattering and diffused back-illumination imaging techniques [166]. In the CFD analysis the spray penetration is defined as the axial distance that encompasses 95% of the injected parcel mass at the instant of time. The jet penetration is measured from the vapor boundary identified by shadowgraph or schlieren imaging [143]. The vapor penetration in the CFD analysis is defined as the maximum axial distance of the iso-surface that is created from mixture fraction (or fuel mass fraction for inert case) of 0.01. Figure 4.16 shows the average of spray and vapor penetration values for 6 realizations for 1.5ms of injection. Both the spray penetration and vapor penetration of simulation are very well matching with the experimental data. The shortness of the vapor penetration at the end is also evident in Figure 4.17 which shows the boundary of the iso-surface data compared with the experimental boundary.

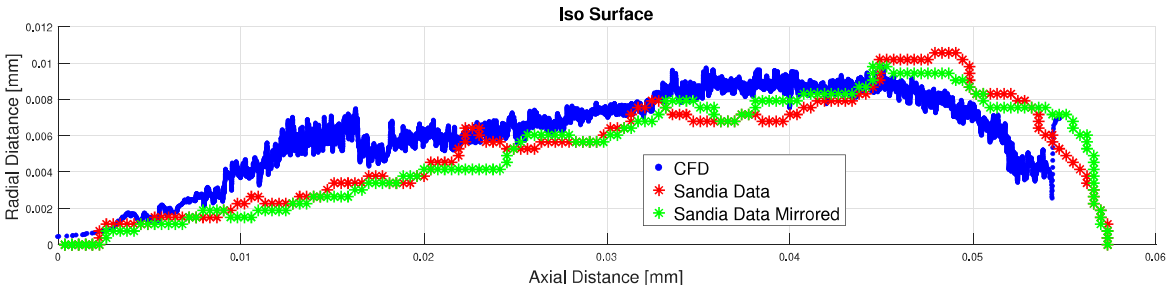

Figure 4.17. Iso-surface of n-dodecane at 1.5ms after start of injection.

The shortness of the vapor penetration in simulation is also evident in n-dodecane mass fraction distribution in the radial direction at 10mm, 25mm, and 45mm away from the spray injector as can be seen in Figure 4.18. At 25mmi the fuel vapor distribution is very close to the experimental data but at 45mm the mass fraction of n-dodecane is almost half of the experimental value. It is not possible to comment about the data at 10mm since no experimental data exists there. Figure 4.19 shows the mass fraction of n-dodecane in central axis. At an axial distance between 20mm and 50mm, the mass fraction of the dodecane in simulation is well matching with the experiment.

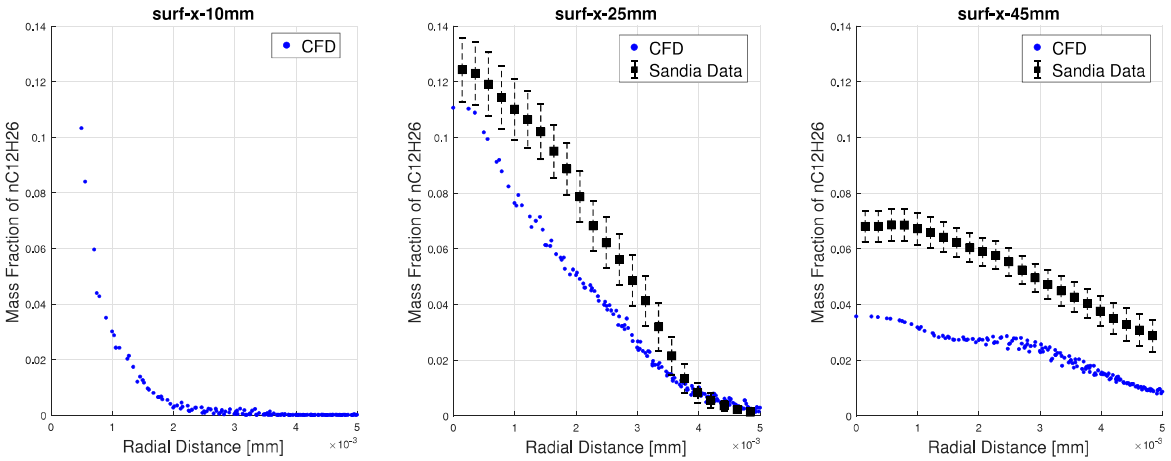

Figure 4.18. Mass fraction of n-dodecane along the radial direction at axial distances of 10mm, 25mm and 45mm at 1.5ms after start of injection.

It is clear that the dodecane vapor in the simulation is captured adequately well in the experiment. The vapor motion in the simulation arises from the shear forces between the Lagrangian spray and Eulerian gas regions and the momentum transfer is maintained by the Lagrangian-Eulerian coupling methods. The slight difference between test and simulation can be caused by several reasons like incorrect boundary

conditions setting, insufficient mesh refinement, improper KH-RT coefficients, or poor Lagrangian-Eulerian Coupling.

The initial and boundary conditions have been checked several times to make sure that the setting are representing the Spray-A test conditions. A mesh independence study has been conducted by refining the mesh suggested by Farrance et.al. [153] and no significant difference has been seen in the spray and vapor penetrations between different simulations as can be seen in Appendix E. The KH-RT coefficients are calculated based on the theory given in Section 2.7.4 and listed in Table 4.1. The effect of CL and B1 values on spray and gas penetration are evaluated with the test domain for 0.5ms and the final coefficients are defined accordingly. The results of the KH-RT coefficients study are given in Appendix F.

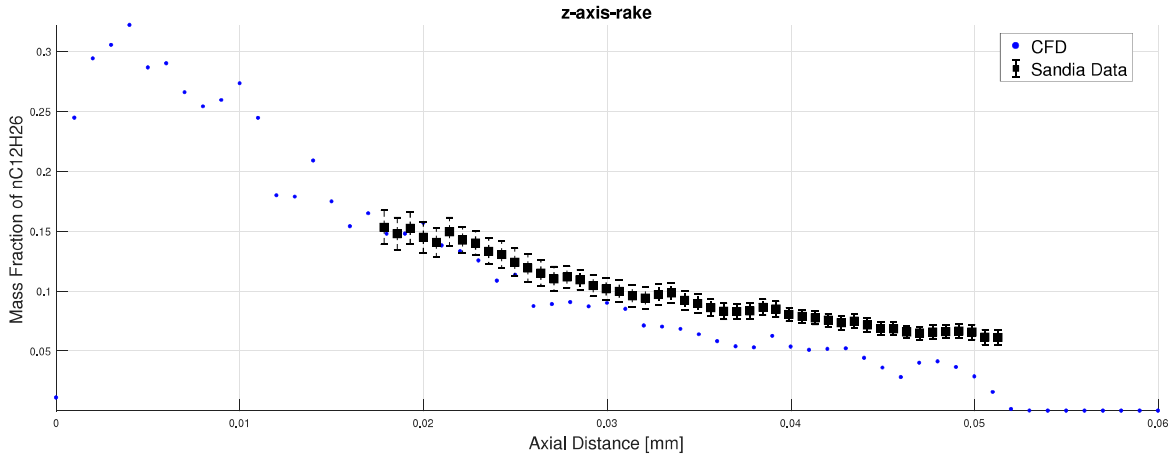

Figure 4.19. Mass fraction of n-dodecane along the central axis at 1.5ms after start of injection.

Figure 4.20 shows that the axial velocity in radial direction at axial distances of 10mm, 25mm and 45mm. The velocity at 10mm in axial direction is higher than the value reported in the experiment but it is getting close to the values of the experiment along the center line as we moved further away from the injector. This implies that the momentum transfer from Lagrangian to Eulerian field is not as good as is supposed to be. The poor coupling problem can be solved by increasing the number of particles injected to the domain at each time step, reduce coupling steps during inner iteration to increase data transfer sequence or reduce flow time step since particles are injected

at each time step.

Table 4.1. The KH-RT breakup coefficients for the Spray-A simulation.

For Spray-A (High We Flow)		
Coefficient	**Ansys Fluent Standard**	**Values Applied**
B0	0.61	0.61
CL/B1	8.61	0.52
B1	1.73	30
CL	14.9	7
Ctau	0.5	1
Crt	0.1	0.1

Figure 4.21 shows the mass fraction contour of n-dodecane and the particles diameter distribution in the domain at 1.5ms after the start of injection. The n-dodecane vapor penetration and axial distribution are adequately captured enough to proceed with the dodecane combustion simulations for demonstration of LS2T approach.

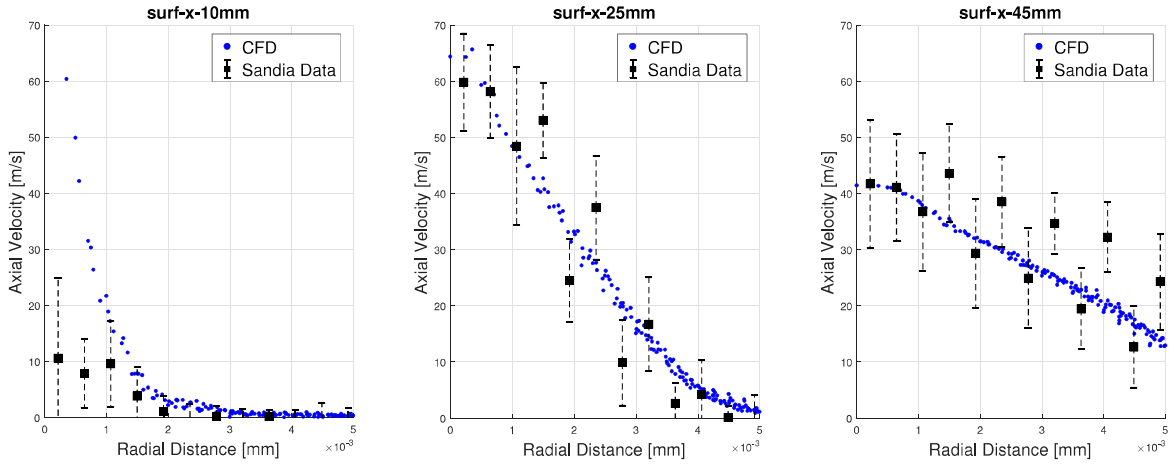

Figure 4.20. Axial velocity values along the radial direction at axial distances of 10mm, 25mm and 45mm at 1.5ms after start of injection.

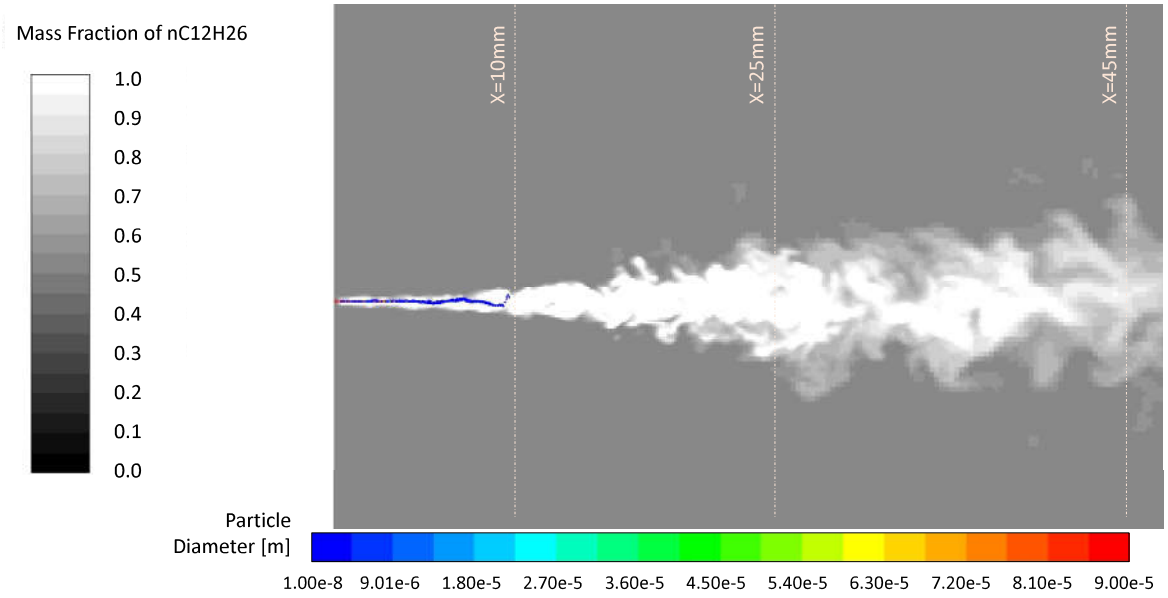

Figure 4.21. Mass fraction and spray contour of n-dodecane at center plane, 1.5ms after start of injection.

4.2.3. Spray-A Combustion Analysis

The Spray-A combustion experiments are conducted at an ambient condition provided by controlled premixed combustion of acetylene (C_2H_2), hydrogen (H_2), oxygen (O_2) and nitrogen (N_2). The test case used for combustion simulations consists of 15.0% O_2, 75.15% N_2, 3.62% H_2O 6.23% CO_2 initially.

The combustion experiments are post-processed using the start of ignition, flame lift-off, mixture fraction, temperature, OH and several other species mass fraction data. The ignition has two definitions and both are used in experiments and CFD. It is defined as the moment when the Favre-averaged OH mass fraction reaches 2% of the maximum in the domain or the moment of highest temperature increase rate in the domain. The flame lift-off is defined as the location where the Favre-averaged OH mass fraction reaches 2% of the maximum in the domain.

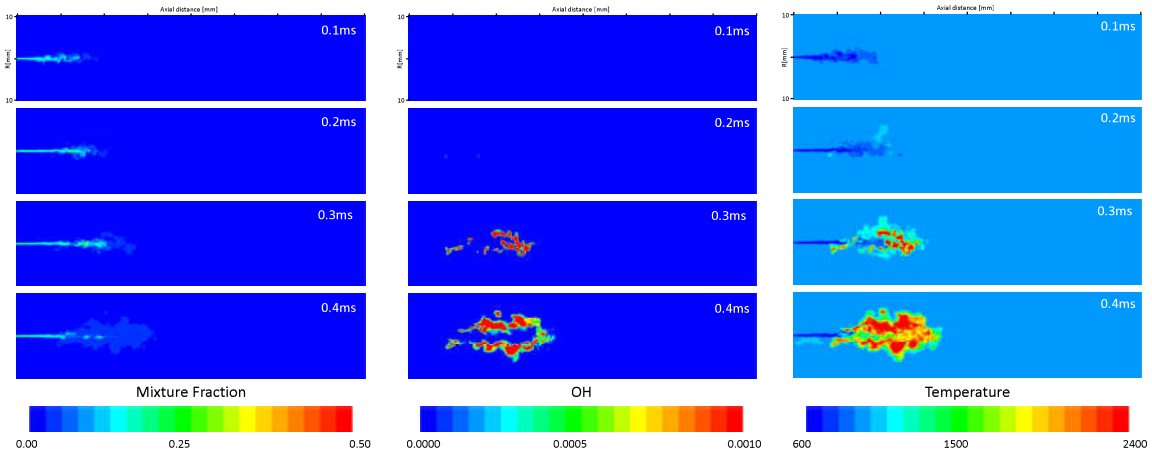

Figure 4.22. Temperature, mixture fraction and OH mass fraction contours at center plane.

The mixture fraction(F) is defined as

$$F = \frac{\phi}{S_p + \phi}, \tag{4.1}$$

where S_p is the stoichiometry parameter calculated by

$$S_p = \frac{n_O Y_{F,F}}{Y_{O,O}}. \tag{4.2}$$

n_O is the ratio of the mass of oxygen required to oxidize unit mass of fuel stoichiometrically, $Y_{F,F}$ is the mass fraction of fuel in the fuel stream, and $Y_{O,O}$ is the mass fraction of oxidizer in oxidizer stream. At the test condition, the values of n_O, $Y_{F,F}$ and $Y_{O,O}$ are 3.482, 1 and 0.164 and stoichiometric mixture fraction F_{st} is calculated as 0.045. Figure 4.22 shows the progress of the Spray-A combustion process over mixture fraction, temperature, and OH mass fraction contours until 0.4ms. The start of ignition in the experiment is about 470μs while the flame lift-off length is 16mm. In the analysis, the start of ignition is calculated based on the rapid temperature increase and iso-surface of OH mass fraction with a value of 2% of the maximum in the domain. As seen in Figure 4.23 the rapid temperature increase in the simulation is at around 220μs and the location of maximum temperature increase lies around 8mm away from the injector. In the simulation, the ignition starts earlier than the experiment resulting in a shorter flame lift-off length. These results contradict the results reported by Luo et.al. [2]. In

the simulations of Luo et.al., the ignition is delayed compared with the experiment and the flame lift-off is higher than the experiment eventually [2]. The early ignition is

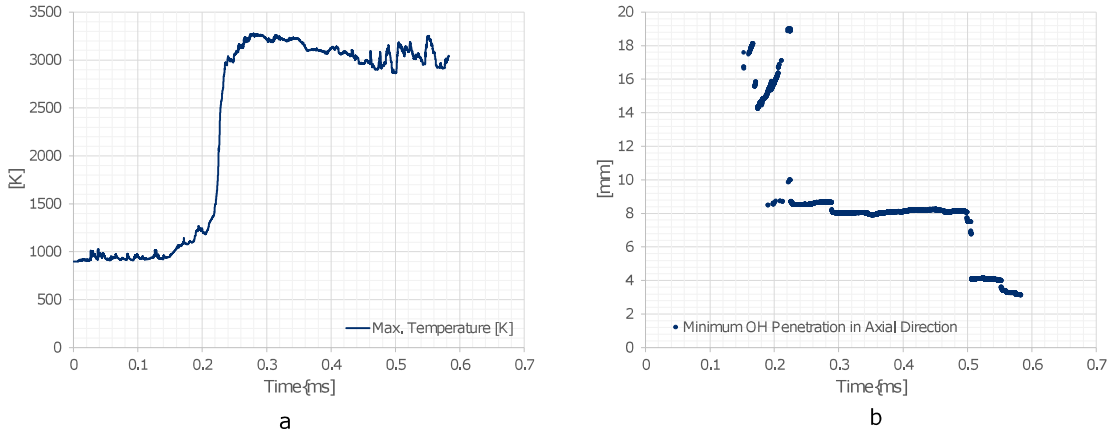

Figure 4.23. Maximum temperature in the domain.

caused by the shorter vapor penetration length of spray at an inert environment due to poor momentum coupling between Lagrangian and Eulerian phases. As a result, the mixing condition required for auto-ignition of dodecane is created at a location shorter and at a time earlier. Spray-A simulation has the improvement opportunity but this is not necessary, because the scope of the work is the application of LS2T methods to 3D CFD problem and this simulation is adequate enough for this purpose.

5. CONCLUSION

This thesis is the first demonstrator of the LS2T method applied to a 3D combustion problem using Sandia Flame-D experiment and it is also the very first simulation of Spray-A experiment that is executed using LES and TPDF in a 3D domain.

The combustion problem preferred for the development of LS2T method and application to 3D simulations is the partially premixed Sandia Flame-D of methane. The detailed GRI 3.0 reaction mechanism of methane consists of 53 species and 325 reactions [150]. Methane chemistry is not a good demonstrator for the heavy reduction ability of LS2T method but is adequate enough to develop the necessary methods and show the possibility of applying LS2T to a 3D combustion problem.

At the beginning of the study, necessary user-defined subroutines have been developed in order to implement the LS2T method to a 0D thermo-chemical kinetic solver, to generate the necessary heavy species tables, to operate the LS2T method, and compare the reaction progress and transient species concentrations with detailed reaction mechanism. Compared with previous studies [25, 120], a reasonable accuracy have been obtained from 0D constant pressure reactor of Cantera for both stoichiometric and rich cases. For the stoichiometric case, the final temperature calculated is very close to the maximum temperature that can be reached based on initial conditions. For the fuel-rich case, the temperature decreases after peak point due to endothermic reactions activated by pyrolysis [120], resulting in a multi-valued behavior. The rate of change of species mass fractions with respect to the temperature are well agreed with detailed GRI 3.0 mechanism results. The only major difference found is in the ignition delays. The calculated ignition delays of LS2T model with 26 species are comparably lower than the GRI 3.0 but such behavior is aligned with the results in the literature [56] for other fuels. As the development of the methodology was completed and its functionality was proven in 0D, the study has moved to application in 3D.

The application of the LS2T method to a 3D combustion problem by using the high-resolution LES turbulence model with TPDF does not require the use of additional turbulence-chemistry coupling models and incorporated the chemical reaction rates directly to the transport problem. The use of a high-resolution model allows the direct application of LS2T method with stiff chemistry solution. It also facilitates the comparison of the model performance with detailed chemistry without any additional modification. The LS2T subroutines are called at each time-step by the 3D CFD code and light species reaction rates are modified with the heavy species data retrieved.

Overall, the LS2T model has successfully replicated the detailed chemistry of methane combustion. The model predicts the species concentration rates and distribution within the domain of almost all light species within the RMS error range except for CO. The temperature field distribution is acceptable. The only flaw of 3D methane combustion arises from the early ignition of the mixture and higher reaction rate calculated with the LS2T model. Since such pitch in temperature has not to be seen in 0D combustion, the reason should not be the LS2T method related but due to several numerical errors that can be made during the implementation of the method to 3D combustion problem.

The application of LS2T to 3D methane combustion simulation proves LS2T as an accurate reaction mechanism reduction method that would reduce the computational effort significantly while maintaining high accuracy for most of the important species that have a significant role in combustion chemistry and emission predictions. However, the application of the LS2T method to the detailed CH_4 mechanism decreases the number of species by a factor of two. Considering the LS2T methods ability to reduce the number of species in heavy hydrocarbons reactions by almost 99%, an assessment with n-dodecane ($C_{12}H_{26}$) would be more beneficial for demonstration purposes.

At the beginning of the testing and development stage of $C_{12}H_{26}$ chemistry, several different reaction mechanisms were evaluated. It is seen that the reaction mechanism of Yao et. al. [160], without reactions defined for low-temperature application, is selected

to be used in Fluent due to the number of species limitations in TPDF solver [128]. But the 0D reactor assessment shows that the mechanism of Yao et. al. is not suitable for assessment of LS2T in the 0D reactor due to different reaction rate results than the rest of the mechanisms tested. Instead of the mechanism of Yao et. al. [160], the mechanism of Luo et.al. [2] has been used for testing in 0D. The reduced chemistry of $C_{12}H_{26}$ found to be more sensitive to the accuracy of LS2T approach since the contribution of heavy species on light species reaction rates are much higher than CH_4 case due to fragmentation of fuel molecule to intermediate (heavy) species sequentially to be converted to final products [121]. The 0D reactor results of $C_{12}H_{26}$ with LS2T is found to be good enough for demonstration purposes with some room for improvement. The ignition delay and temperature progress are found to be well-matched with the base reaction mechanism. If a wider tabular data range were used, the results would have been better.

The 3D combustion of $C_{12}H_{26}$ is done using the Spray-A test case. Spray-A has actually brought additional complexity to the combustion problem that requires a very accurate Lagrangian representation of spray in the continuous domain. This is achieved by assessing different grid refinements and different secondary breakup coefficients to find the most proper settings for the inert Spray-A experiment. At the end of the analysis, the spray penetration, vapor penetration, velocity, and $C_{12}H_{26}$ mass fractions have been used for assessment. It is seen that due to inadequate momentum transfer between Lagrangian and Eulerian fields, the vapor penetration is found to be shorter than the experiment and the $C_{12}H_{26}$ mass fraction is less spread in both radial and axial directions compared with the experiment. The inert spray simulation is used as the baseline for reacting Spray-A at ambient with 15% O_2. The reacting Spray-A case is post-processed using the ignition delay and flame lift-off data. Due to shorter vapor penetration, the mixture ignites earlier than the experiment and results in about 25% shorter flame lift-off.

The results of the thesis establish evidence that by the use of the LS2T method, it is possible to implement LS2T approach to a 3D combustion problem to maintain

high accuracy and generate results similar to the detailed chemistry with affordable computational effort.

REFERENCES

1. Reitz, R. D. and R. Diwakar, "Structure of High-Pressure Fuel Sprays", *SAE Transactions*, Vol. 96, pp. 492–509, 1987.

2. Luo, Z., S. Som, S. Sarathy, M. Plomer, W. Pitz, D. Longman and T. Lu, "Development and Validation of an n-Dodecane Skeletal Mechanism for Spray Combustion Applications", *Combustion Theory and Modelling*, Vol. 18, 2014.

3. Xu, J. and S. B. Pope, "PDF Calculations of Turbulent Nonpremixed Flames with Local Extinction", *Combustion and Flame*, Vol. 123, No. 3, pp. 281–307, 2000.

4. Council, N. R., *The Potential Impact of High-End Capability Computing on Four Illustrative Fields of Science and Engineering*, The National Academies Press, Washington, DC, 2008.

5. Ihme, M. and H. Pitsch, "Prediction of Extinction and Reignition in Non-Premixed Turbulent Flames Using a Flamelet/Progress Variable Model. 2. Application in LES of Sandia Flames D and E", *Combustion and Flame*, Vol. 155, No. 1-2, pp. 90–107, 2008.

6. Vreman, A. W., B. A. Albrecht, J. A. van Oijen, L. P. H. de Goey and R. J. M. Bastiaans, "Premixed And Nonpremixed Generated Manifolds in Large-Eddy Simulation of Sandia Flame D And F", *Combustion and Flame*, Vol. 153, No. 3, pp. 394–416, 2008.

7. Di Renzo, M., A. Coclite, M. D. De Tullio, P. De Palma and G. Pascazio, "LES of the Sandia Flame D Using an FPV Combustion Model", *Energy Procedia*, Vol. 82, pp. 402–409, 2015.

8. Pitsch, H., "Improved Pollutant Predictions in Large-Eddy Simulations of Turbu-

lent Non-Premixed Combustion by Considering Scalar Dissipation Rate Fluctuations", *Proceedings of the Combustion Institute*, Vol. 29, No. 2, pp. 1971–1978, 2002.

9. Kemenov, K. A. and S. B. Pope, "Molecular Diffusion Effects in LES of a Piloted Methane-Air Flame", *Combustion and Flame*, Vol. 158, No. 2, pp. 240–254, 2011.

10. Garmory, A. and E. Mastorakos, "Capturing Localised Extinction in Sandia Flame F with LES-CMC", *Proceedings of the Combustion Institute*, Vol. 33, No. 1, pp. 1673–1680, 2011.

11. Garmory, A. and E. Mastorakos, "Numerical Simulation of Oxy-Fuel Jet Flames Using Unstructured LES-CMC", *Proceedings of the Combustion Institute*, Vol. 35, No. 2, pp. 1207–1214, 2015.

12. Sheikhi, M. R. H., T. G. Drozda, P. Givi and S. B. Pope, "Velocity-Scalar Filtered Density Function for Large Eddy Simulation of Turbulent Flows", *Physics of Fluids*, Vol. 15, No. 8, pp. 2321–2337, 2003.

13. Mustata, R., L. Valiño, C. Jiménez, W. Jones and S. Bondi, "A Probability Density Function Eulerian Monte Carlo Field Method for Large Eddy Simulations: Application to A Turbulent Piloted Methane/Air Diffusion Flame (Sandia D)", *Combustion and Flame*, Vol. 145, pp. 88–104, 2006.

14. Raman, V. and H. Pitsch, "A Consistent LES/Filtered-Density Function Formulation for the Simulation of Turbulent Flames with Detailed Chemistry", *Proceedings of the Combustion Institute*, Vol. 31, pp. 1711–1719, 2007.

15. Jones, W. P. and S. Navarro-Martinez, "Large Eddy Simulation of Autoignition with a Subgrid Probability Density Function Method", *Combustion and Flame*, Vol. 150, No. 3, pp. 170–187, 2007.

16. Jaishree, J. and D. C. Haworth, "Comparisons of Lagrangian and Eulerian PDF

Methods in Simulations of Non-Premixed Turbulent Jet Flames with Moderate-to-Strong Turbulence-Chemistry Interactions", *Combustion Theory and Modelling*, Vol. 16, No. 3, pp. 435–463, 2012.

17. Jones, W. P. and V. N. Prasad, "Large Eddy Simulation of the Sandia Flame Series (D-F) Using the Eulerian Stochastic Field Method", *Combustion and Flame*, Vol. 157, No. 9, pp. 1621–1636, 2010.

18. Ferraro, F., Y. Ge and M. Pfitzner, "A Consistent Hybrid LES-RANS PDF Method for Non-Premixed Flames", *Physics Procedia*, Vol. 66, pp. 317–320, 2015.

19. Yi Peng Ge, A., M.J.Cleary, "Sparse-Lagrangian FDF Simulations of Sandia Flame E with Density Coupling", *Proceedings of the Combustion Institute*, Vol. 33, No. 1, pp. 1401–1409, 2011.

20. Pope, S. B., "PDF Methods for Turbulent Reacting Flows", *Progress in Energy and Combustion Science*, Vol. 11, No. 2, pp. 119–192, 1985.

21. Jaberi, F. A., P. J. Colucci, S. James, P. Givi and S. B. Pope, "Filtered Mass Density Function for Large-Eddy Simulation of Turbulent Reacting Flows", *Journal of Fluid Mechanics*, Vol. 401, p. 85–121, 1999.

22. Sabel'nikov, V. and O. Soulard, "Stochastic Partial Differential Equations as a Tool for Solving PDF Equations", *Progress in Turbulence*, pp. 107–110, Springer, 2005.

23. Valiño, L., "A Field Monte Carlo Formulation for Calculating the Probability Density Function of a Single Scalar in a Turbulent Flow", *Flow, Turbulence and Combustion*, Vol. 60, No. 2, pp. 157–172, 1998.

24. Fox, R. O., *Computational Models for Turbulent Reacting Flows*, Cambridge University Press, Cambridge, 2003.

25. Kourdis, P. D. and J. Bellan, "Heavy-Alkane Oxidation Kinetic-Mechanism Reduction Using Dominant Dynamic Variables, Self Similarity And Chemistry Tabulation", *Combustion and Flame*, Vol. 161, No. 5, pp. 1196–1223, 2014.

26. Hiremath, V. and S. B. Pope, "A Study of the Rate-Controlled Constrained-Equilibrium Dimension Reduction Method And Its Different Implementations", *Combustion Theory and Modelling*, Vol. 17, No. 2, pp. 260–293, 2013.

27. Turns, S. R., *An Introduction to Combustion: Concepts and Applications*, McGraw-Hill, 2012.

28. Peters, N., "Laminar Diffusion Flamelet Models in Non-Premixed Turbulent Combustion", *Progress in Energy and Combustion Science*, Vol. 10, No. 3, pp. 319–339, 1984.

29. Okino, M. S. and M. L. Mavrovouniotis, "Simplification of Mathematical Models of Chemical Reaction Systems", *Chemical Reviews*, Vol. 98, No. 2, pp. 391–408, 1998.

30. Warnatz, J., U. Maas and R. Dibble, *Combustion: Physical and Chemical Fundamentals, Modeling and Simulation, Experiments, Pollutant Formation*, Springer Berlin Heidelberg, 2006.

31. Xi, J. and B. J. Zhong, "Reduced Kinetic Mechanism of n-Heptane Oxidation in Modeling Polycyclic Aromatic Hydrocarbon Formation in Diesel Combustion", *Chemical Engineering & Technology*, Vol. 29, No. 12, pp. 1461–1468, 2006.

32. Sun, W., Z. Chen, X. Gou and Y. Ju, "A Path Flux Analysis Method for the Reduction of Detailed Chemical Kinetic Mechanisms", *Combustion and Flame*, Vol. 157, No. 7, pp. 1298–1307, 2010.

33. Luche, J., M. Reuillon, J. Boettner and M. Cathonnet, "Reduction of Large Detailed Kinetic Mechanisms: Application to Kerosene/Air Combustion", *Com-*

bustion Science and Technology, Vol. 176, pp. 1935–1963, 2004.

34. Niemeyer, K. E. and C.-J. Sung, "On the Importance of Graph Search Algorithms for Drgep-Based Mechanism Reduction Methods", *Combustion and Flame*, Vol. 158, No. 8, pp. 1439–1443, 2011.

35. Yao, T., Y. Pei, B.-J. Zhong, S. Som, T. Lu and K. H. Luo, "A Compact Skeletal Mechanism for n-Dodecane with Optimized Semi-Global Low-Temperature Chemistry for Diesel Engine Simulations", *Fuel*, Vol. 191, pp. 339 – 349, 2017.

36. Wang, H., Y. Ra, M. Jia and R. D. Reitz, "Development of a Reduced n-Dodecane-Pah Mechanism and Its Application for n-Dodecane Soot Predictions", *Fuel*, Vol. 136, No. Supplement C, pp. 25–36, 2014.

37. Pierce, C. D. and P. Moin, "Progress-Variable Approach for Large-Eddy Simulation of Non-Premixed Turbulent Combustion", *Journal of Fluid Mechanics*, Vol. 504, p. 73–97, 2004.

38. Gicquel, O., N. Darabiha and D. Thévenin, "Liminar Premixed Hydrogen/Air Counterflow Flame Simulations Using Flame Prolongation of ILDM with Differential Diffusion", *Proceedings of the Combustion Institute*, Vol. 28, No. 2, pp. 1901–1908, 2000.

39. van Oijen, J., F. Lammers and L. de Goey, "Modeling of Complex Premixed Burner Systems by Using Flamelet-Generated Manifolds", *Combustion and Flame*, Vol. 127, No. 3, pp. 2124–2134, 2001.

40. Knudsen, E. and H. Pitsch, "Capabilities and Limitations of Multi-Regime Flamelet Combustion Models", *Combustion and Flame*, Vol. 159, No. 1, pp. 242–264, 2012.

41. Nguyen, P.-D., L. Vervisch, V. Subramanian and P. Domingo, "Multidimensional Flamelet-Generated Manifolds for Partially Premixed Combustion", *Combustion*

and Flame, Vol. 157, No. 1, pp. 43–61, 2010.

42. van Oijen, J. and L. de Goey, "A Numerical Study of Confined Triple Flames Using a Flamelet-Generated Manifold", *Combustion Theory and Modelling*, Vol. 8, No. 1, pp. 141–163, 2004.

43. Hollmann, C. and E. Gutheil, "Diffusion Flames Based on a Laminar Spray Flame Library", *Combustion Science and Technology*, Vol. 135, No. 1-6, pp. 175–192, 1998.

44. Franzelli, B., B. Fiorina and N. Darabiha, "A Tabulated Chemistry Method for Spray Combustion", *Proceedings of the Combustion Institute*, Vol. 34, No. 1, pp. 1659–1666, 2013.

45. Salehi, M. M., W. K. Bushe, N. Shahbazian and C. P. Groth, "Modified Laminar Flamelet Presumed Probability Density Function for LES of Premixed Turbulent Combustion", *Proceedings of the Combustion Institute*, Vol. 34, No. 1, pp. 1203 – 1211, 2013.

46. Kim, W.-W. and S. Menon, "Numerical Modeling of Turbulent Premixed Flames in the Thin-Reaction-Zones Regime", *Combustion Science and Technology*, Vol. 160, No. 1, pp. 119–150, 2000.

47. Galpin, J., A. Naudin, L. Vervisch, C. Angelberger, O. Colin and P. Domingo, "Large-Eddy Simulation of a Fuel-Lean Premixed Turbulent Swirl-Burner", *Combustion and Flame*, Vol. 155, No. 1, pp. 247 – 266, 2008.

48. Ferraris, S. A. and J. X. Wen, "LES of the Sandia Flame D Using Laminar Flamelet Decomposition for Conditional Source-Term Estimation", *Flow, Turbulence and Combustion*, Vol. 81, pp. 609–639, 2008.

49. Bushe, W. and H. Steiner, "Laminar Flamelet Decomposition for Conditional Source-Term Estimation", *Physics of Fluids*, Vol. 15, pp. 1564–1575, 2003.

50. Bell, J. B., N. J. Brown, M. S. Day, M. Frenklach, J. F. Grcar, R. M. Propp and S. R. Tonse, "Scaling and Efficiency of Prism in Adaptive Simulations of Turbulent Premixed Flames", *Proceedings of the Combustion Institute*, Vol. 28, No. 1, pp. 107 – 113, 2000.

51. Pope, S., "Computationally Efficient Implementation of Combustion Chemistry Using in Situ Adaptive Tabulation", *Combustion Theory and Modelling*, Vol. 1, No. 1, pp. 41–63, 1997.

52. Maas, U. and S. Pope, "Simplifying Chemical Kinetics: Intrinsic Low-Dimensional Manifolds in Composition Space", *Combustion and Flame*, Vol. 88, No. 3, pp. 239 – 264, 1992.

53. Harstad, K. and J. Bellan, "A Model of Reduced Kinetics for Alkane Oxidation Using Constituents and Species: Proof of Concept for N-Heptane", *Combustion and Flame*, Vol. 157, No. 8, pp. 1594–1609, 2010.

54. Lawrence Livermore National Laboratory, "Mechanisms", `https://combustion.llnl.gov/mechanisms`, accessed in December 2021.

55. Kourdis, P. D. and J. Bellan, "Highly Reduced Species Mechanisms for iso-Cetane Using the Local Self-Similarity Tabulation Method", *International Journal of Chemical Kinetics*, Vol. 48, No. 11, pp. 739–752, 2016.

56. Bellan, J., "From Elementary Kinetics in Perfectly Stirred Reactors to Reduced Kinetics Utilizable in Turbulent Reactive Flow Simulations for Combustion Devices", *Combustion and Flame*, Vol. 184, No. Supplement C, pp. 286–296, 2017.

57. Ribert, G., L. Vervisch, P. Domingo and Y.-S. Niu, "Hybrid Transported-Tabulated Strategy to Downsize Detailed Chemistry for Numerical Simulation of Premixed Flames", *Flow, Turbulence and Combustion*, Vol. 92, No. 1, pp. 175–200, 2014.

58. Barlow, R., J. Frank, A. Karpetis and J.-Y. Chen, "Piloted Methane/Air Jet Flames: Transport Effects and Aspects of Scalar Structure", *Combustion and Flame*, Vol. 143, No. 4, pp. 433 – 449, 2005.

59. Robert Barlow, J. F., *Piloted CH4/Air Flames C, D, E, and F*, Tech. rep., Sandia National Laboratories, 2007.

60. Popov, P. P. and S. B. Pope, "Large Eddy Simulation/Probability Density Function Simulations of Bluff Body Stabilized Flames", *Combustion and Flame*, Vol. 161, No. 12, pp. 3100–3133, 2014.

61. James, S., J. Zhu and M. Anand, "Large Eddy Simulations of Turbulent Flames Using the Filtered Density Function Model", *Proceedings of the Combustion Institute*, Vol. 31, No. 2, pp. 1737–1745, 2007.

62. Chen, J.-Y., "A Eulerian PDF Scheme for LES of Nonpremixed Turbulent Combustion with Second-Order Accurate Mixture Fraction", *Combustion Theory and Modelling*, Vol. 11, No. 5, pp. 675–695, 2007.

63. Ge, Y., M. Cleary and A. Klimenko, "A Comparative Study of Sandia Flame Series (D–F) Using Sparse-Lagrangian MMC Modelling", *Proceedings of the Combustion Institute*, Vol. 34, No. 1, pp. 1325–1332, 2013.

64. Zhao, W., "Large-Eddy Simulation of Piloted Diffusion Flames Using Multi-Environment Probability Density Function Models", *Proceedings of the Combustion Institute*, Vol. 36, No. 2, pp. 1705–1712, 2017.

65. Sundaram, B., A. Y. Klimenko, M. J. Cleary and Y. Ge, "A Direct Approach to Generalised Multiple Mapping Conditioning for Selected Turbulent Diffusion Flame Cases", *Combustion Theory and Modelling*, Vol. 20, No. 4, pp. 735–764, 2016.

66. Kim, N., K. Jung and Y. Kim, "Large Eddy Simulation of Piloted Turbulent

Jet Flames Using the Multi-Environment PDF Model Based On the Flamelet Generated Manifold", *Journal of Mechanical Science and Technology*, Vol. 32, pp. 2399–2406, 2018.

67. Duan, Y., Z. Xia, L. Ma, Z. Luo, X. Huang and X. Deng, "LES of the Sandia Flame Series D-F Using the Eulerian Stochastic Field Method Coupled with Tabulated Chemistry", *Chinese Journal of Aeronautics*, Vol. 33, No. 1, pp. 116–133, 2020.

68. Stanković, I., "Modelling of Non-Premixed Turbulent Combustion with Conditional Moment Closure (CMC)", *The European Physical Journal E*, Vol. 41, No. 150, 2018.

69. Kempf, A., F. Flemming and J. Janicka, "Investigation of Length Scales, Scalar Dissipation, and Flame Orientation in a Piloted Diffusion Flame by LES", *Proceedings of the Combustion Institute*, Vol. 30, No. 1, pp. 557–565, 2005.

70. Sheikhi, M., T. Drozda, P. Givi, F. Jaberi and S. Pope, "Large Eddy Simulation of a Turbulent Non-Premixed Piloted Methane Jet Flame (Sandia Flame D)", *Proceedings of the Combustion Institute*, Vol. 30, pp. 549–556, 2005.

71. Clayton, D. J. and W. P. Jones, "Large Eddy Simulation of a Methane–Air Diffusion Flame", *Flow, Turbulence and Combustion*, Vol. 81, p. 497–521, 2008.

72. Pant, T., C. Han and H. Wang, "Examination of Errors of Table Integration in Flamelet/Progress Variable Modeling of a Turbulent Non-Premixed Jet Flame", *Applied Mathematical Modelling*, Vol. 72, pp. 369–384, 2019.

73. Pitsch, H. and H. Steiner, "Large-Eddy Simulation of a Turbulent Piloted Methane/Air Diffusion Flame (Sandia Flame D)", *Physics of Fluids*, Vol. 12, No. 10, pp. 2541–2554, 2000.

74. Yang, S., X. Wang, H. Huo, W. Sun and V. Yang, "An Efficient Finite-Rate Chem-

istry Model for a Preconditioned Compressible Flow Solver and Its Comparison with the Flamelet/Progress-Variable Model", *Combustion and Flame*, Vol. 210, pp. 172–182, 2019.

75. Steiner, H. and W. K. Bushe, "Large Eddy Simulation of a Turbulent Reacting Jet with Conditional Source-Term Estimation", *Physics of Fluids*, Vol. 13, No. 3, pp. 754–769, 2001.

76. Lysenko, D. A., I. S. Ertesvåg and K. E. Rian, "Numerical Simulations of the Sandia Flame D Using the Eddy Dissipation Concept", *Flow, Turbulence and Combustion*, Vol. 93, No. 4, p. 665–687, 2014.

77. Elbahloul, S. and S. Rigopoulos, "Rate-Controlled Constrained Equilibrium (RCCE) simulations of turbulent partially premixed flames (Sandia D/E/F) and comparison with detailed chemistry", *Combustion and Flame*, Vol. 162, No. 5, pp. 2256–2271, 2015.

78. Jaravel, T., E. Riber, B. Cuenot and P. Pepiot, "Prediction of Flame Structure and Pollutant Formation of Sandia Flame D Using Large Eddy Simulation with Direct Integration of Chemical Kinetics", *Combustion and Flame*, Vol. 188, pp. 180–198, 2018.

79. Miles, J. S. and T. Echekki, "A One-Dimensional Turbulence-Based Closure Model for Combustion LES", *Combustion Science and Technology*, Vol. 192, No. 1, pp. 78–111, 2020.

80. Bruneaux, G., L.-M. Malbec, L. Hermant, C. Christiansen, J. Schramm, L. M. Pickett and C. L. Genzale, "Comparison of Diesel Spray Combustion in Different High-Temperature, High-Pressure Facilities", *SAE International Journal of Engines*, Vol. 3, No. 2, pp. 156–181, 2010.

81. Pickett, L. M., C. L. Genzale, G. Bruneaux, L.-M. Malbec, L. Hermant, C. Chris-

tiansen and J. Schramm, "Comparison of Diesel Spray Combustion in Different High-Temperature, High-Pressure Facilities", *SAE International Journal of Engines*, Vol. 3, No. 2, pp. 156–181, 2010.

82. Siebers, D. L. and B. Higgins, "Flame Lift-Off on Direct-Injection Diesel Sprays Under Quiescent Conditions", *SAE Technical Paper*, 2001-01-0530, SAE International, 2001.

83. Higgins, B. and D. L. Siebers, "Measurement of the Flame Lift-Off Location on DI Diesel Sprays Using OH Chemiluminescence", *SAE Technical Paper*, 2001-01-0918, SAE International, 2001.

84. Meijer, M., B. Somers, J. Johnson, J. Naber, S.-Y. Lee, L. M. C. Malbec, G. Bruneaux, L. M. Pickett, M. Bardi, R. Payri and T. Bazyn, "Engine Combustion Network (ECN): Characterization and Comparison of Boundary Conditions for Different Combustion Vessels", *Atomization and Sprays*, Vol. 22, No. 9, pp. 777–806, 2012.

85. Bardi, M., R. Payri, L. M. C. Malbec, G. Bruneaux, L. M. Pickett, J. Manin, T. Bazyn and C. L. Genzale, "Engine Combustion Network: Comparison of Spray Development, Vaporization, and Combustion in Different Combustion Vessels", *Atomization and Sprays*, Vol. 22, No. 10, pp. 807–842, 2012.

86. Jones, W. and S. Navarro-Martinez, "Numerical Study of n-Heptane Auto-Ignition Using LES-PDF Methods", *Flow, Turbulence and Combustion*, Vol. 83, pp. 407–423, 2009.

87. Apte, S., M. Gorokhovski and P. Moin, "LES of Atomizing Spray with Stochastic Modeling of Secondary Breakup", *International Journal of Multiphase Flow*, Vol. 29, No. 9, pp. 1503–1522, 2003.

88. Som, S., D. Longman, Z. Luo, M. Plomer and T. Lu, "Three Dimensional Simula-

tions of Diesel Sprays Using n-Dodecane as a Surrogate", *Fall Technical Meeting of the Eastern States Section of the Combustion Institute*, Vol. 12, 2011.

89. Gong, C., M. Jangi and X.-S. Bai, "Large Eddy Simulation of n-Dodecane Spray Combustion in a High Pressure Combustion Vessel", *Applied Energy*, Vol. 136, p. 373–381, 2014.

90. Bhattacharjee, S. and D. Haworth, "Simulations of Transient n-Heptane and n-Dodecane Spray Flames Under Cross Engine-Relevant Conditions Using a Transported PDF Method", *Combustion and Flame*, Vol. 160, pp. 2083–2102, 2013.

91. D'Errico, G., T. Lucchini, F. Contino, M. Jangi and X.-S. Bai, "Comparison of Well-Mixed and Multiple Representative Interactive Flamelet Approaches for Diesel Spray Combustion Modelling", *Combustion Theory and Modelling*, Vol. 18, No. 1, pp. 65–88, 2014.

92. Ayyapureddi, S., U. Eguz, C. Bekdemir, L. Somers and L. Goey, de, "Application of the FGM method to Spray A conditions of the ECN database", *ICLASS 2012 :12th Triennial International Conference on Liquid Atomization and Spray Systems, September 2-6, 2012, Heidelberg, Germany*, pp. 1–8, ILASS Europe, 2012.

93. Pei, Y., S. Som, E. Pomraning, P. Senecal, S. Skeen, J. Manin and L. Pickett, "Large Eddy Simulation of a Reacting Spray Flame with Multiple Realizations Under Compression Ignition Engine Conditions", *Combustion and Flame*, Vol. 162, 2015.

94. Zhang, Y., H. Wang, A. Both, L. Ma and M. Yao, "Effects of Turbulence-Chemistry Interactions on Auto-Ignition and Flame Structure for n-Dodecane Spray Combustion", *Combustion Theory and Modelling*, Vol. 23, No. 5, pp. 907–934, 2019.

95. Kahila, H., A. Wehrfritz, O. Kaario, M. Ghaderi Masouleh, N. Maes, L. Somers and V. Vuorinen, "Large-Eddy Simulation on the Influence of Injection Pressure in Reacting Spray A", *Combustion and Flame*, Vol. 191, pp. 142–159, 2018.

96. Pandurangi, S. S., M. Bolla, Y. M. Wright, K. Boulouchos, S. a Skeen, J. Manin and L. M. Pickett, "Onset and Progression of Soot in High-Pressure n-Dodecane Sprays Under Diesel Engine Conditions", *International Journal of Engine Research*, Vol. 18, No. 5-6, pp. 436–452, 2017.

97. Wehrfritz, A., O. Kaario, V. Vuorinen and L. Somers, "Large Eddy Simulation of n-Dodecane Spray Flames Using Flamelet Generated Manifolds", *Combustion and Flame*, Vol. 167, pp. 113–131, 2016.

98. Maghbouli, A., B. Akkurt, T. Lucchini, G. D'Errico, N. G. Deen and B. Somers, "Modelling Compression Ignition Engines by Incorporation of the Flamelet Generated Manifolds Combustion Closure", *Combustion Theory and Modelling*, Vol. 23, No. 3, pp. 414–438, 2019.

99. Blomberg, C., L. Zeugin, S. Pandurangi, M. Bolla, K. Boulouchos and Y. Wright, "Modeling Split Injections of ECN "Spray A" Using a Conditional Moment Closure Combustion Model with RANS and LES", *SAE International Journal of Engines*, Vol. 9, pp. 2107–2119, 2016.

100. Davidovic, M., T. Falkenstein, M. Bode, L. Cai, S. Kang, J. Hinrichs and H. Pitsch, "LES of n-Dodecane Spray Combustion Using a Multiple Representative Interactive Flamelets Model", *Oil & Gas Science and Technology*, Vol. 72, p. 29, 2017.

101. Kundu, P., M. M. Ameen and S. Som, "Importance of Turbulence-Chemistry Interactions at Low Temperature Engine Conditions", *Combustion and Flame*, Vol. 183, pp. 283–298, 2017.

102. Samimi Abianeh, O., N. Curtis and C.-J. Sung, "Determination of Modeled Luminosity-Based and Pressure-Based Ignition Delay Times of Turbulent Spray Combustion", *International Journal of Heat and Mass Transfer*, Vol. 103, pp. 1297–1312, 2016.

103. Payri, F., J. Garcia-Oliver, R. Novella and E. Pérez-Sánchez, "Influence of the n-Dodecane Chemical Mechanism on the CFD Modelling of the Diesel-Like ECN Spray A Flame Structure at Different Ambient Conditions", *Combustion and Flame*, Vol. 208, pp. 198–218, 2019.

104. Pei, Y., E. Hawkes, S. Kook, G. Goldin and T. Lu, "Modelling n-Dodecane Spray and Combustion with the Transported Probability Density Function Method", *Combustion and Flame*, Vol. 162, pp. 2006–2019, 2015.

105. Chishty, M., M. Bolla, Y. Pei and S. Kook, "A Numerical Study of 'Spray A'with Multiple-Injections Using the Transported PDF Method", *10th Asia-Pacific Conference on Combustion, July 19-22, 2015, Beijing, China*, 124, 2015.

106. Pei, Y., E. R. Hawkes, M. Bolla, S. Kook, G. M. Goldin, Y. Yang, S. B. Pope and S. Som, "An Analysis of the Structure of an n-Dodecane Spray Flame Using TPDF Modelling", *Combustion and Flame*, Vol. 168, pp. 420–435, 2016.

107. Salehi, F., M. Cleary, A. Masri, Y. Ge and A. Klimenko, "Sparse-Lagrangian MMC Simulations of an n-Dodecane Jet at Engine-Relevant Conditions", *Proceedings of the Combustion Institute*, Vol. 36, 2016.

108. Team, A. S., *Kelvin-Helmholtz/Rayleigh-Taylor Model, a Brief Introduction and Discussion on Choosing Model Parameters*, Tech. rep., Ansys, 2009.

109. Haworth, D. C., "Progress in Probability Density Function Methods for Turbulent Reacting Flows", *Progress in Energy and Combustion Science*, Vol. 36, No. 2, pp. 168–259, 2010.

110. Garnier, E., N. Adams and P. Sagaut, *Large Eddy Simulation for Compressible Flows*, Springer Netherlands, 2009.

111. Favre, a., "Equations Statistiques Des Gaz Turbulents", *Comptes rendus de l'Académie des Sciences*, Vol. 246, 1958.

112. Denaro, F. M., "What Does Finite Volume-Based Implicit Filtering Really Resolve in Large-Eddy Simulations?", *Journal of Computational Physics*, Vol. 230, No. 10, pp. 3849–3883, 2011.

113. Ferziger, J. H. and M. Perić, *Computational Methods for Fluid Dynamics*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

114. Ghosal, S., T. S. Lund, P. Moin and K. Akselvoll, "A Dynamic Localization Model for Large-Eddy Simulation of Turbulent Flows", *Journal of Fluid Mechanics*, Vol. 286, pp. 229–255, 1995.

115. Kim, W. W. and S. Menon, *Application of the Localized Dynamic Subgrid-Scale Model to Turbulent Wall-Bounded Flows*, Tech. rep., American Institute of Aeronautics and Astronautics, Reno NV, 1997.

116. Vreman, A., *Direct and Large-Eddy Simulation of the Compressible Turbulent Mixing Layer*, Ph.D. Thesis, University of Twente, Netherlands, 1995.

117. Raman, V., H. Pitsch and R. O. Fox, "Eulerian Transported Probability Density Function Sub-Filter Model for Large-Eddy Simulations of Turbulent Combustion", *Combustion Theory and Modelling*, Vol. 10, No. 3, pp. 439–458, 2006.

118. Subramaniam, S. and S. Pope, "A Mixing Model for Turbulent Reactive Flows Based on Euclidean Minimum Spanning Trees", *Combustion and Flame*, Vol. 115, pp. 487–514, 1998.

119. Wang, H., P. P. Popov and S. B. Pope, "Weak Second-Order Splitting Schemes

for Lagrangian Monte Carlo Particle Methods for the Composition PDF/FDF Transport Equations", *Journal of Computational Physics*, Vol. 229, pp. 1852–1878, 2010.

120. Kourdis, P. D. and J. Bellan, "High-Pressure Reduced-Kinetics Mechanism for n-Hexadecane Autoignition and Oxidation at Constant Pressure", *Combustion and Flame*, Vol. 162, No. 3, pp. 571–579, 2015.

121. Westbrook, C. and F. Dryer, "Chemical Kinetic Modeling of Hydrocarbon Combustion", *Progress in Energy and Combustion Science*, Vol. 10, pp. 1–57, 1984.

122. Pope, S. B., "Ten Questions Concerning the Large-Eddy Simulation of Turbulent Flows", *New Journal of Physics*, Vol. 6, 2004.

123. Davidson, L., "Large Eddy Simulations: How to Evaluate Resolution", *International Journal of Heat and Fluid Flow*, Vol. 30, No. 5, pp. 1016–1025, 2009.

124. Gaitonde, U., D. Laurence and A. Revell, *Quality Criteria for Large Eddy Simulation*, Tech. rep., University of Manchester, 2008.

125. Stiesch, G., *Modeling Engine Spray and Combustion Processes*, Springer Science & Business Media, 2003.

126. Wehrfritz, A., V. Vuorinen, O. Kaario and M. Larmi, "Large Eddy Simulation of High-Velocity Fuel Sprays: Studying Mesh Resolution and Breakup Model Effects for Spray A", *Atomization and Sprays*, Vol. 23, pp. 419–442, 2013.

127. Siebers, D., "Liquid-Phase Fuel Penetration in Diesel Sprays", *SAE Technical Papers*, 1998.

128. Ansys, *Fluent R19.2 Theory Guide*, ANSYS Incorporated, 2018.

129. Gosman, A. D. and E. Loannides, "Aspects of Computer Simulation of Liquid-

Fueled Combustors", *Journal of Energy*, Vol. 7, No. 6, pp. 482–490, 1983.

130. Liu, A. B., D. Mather and R. D. Reitz, "Modeling the Effects of Drop Drag and Breakup on Fuel Sprays", *SAE Technical Paper*, 930072, SAE International, 1993.

131. O'Rourke, P. J. and A. A. Amsden, "The Tab Method for Numerical Calculation of Spray Droplet Breakup", *SAE Technical Paper*, 872089, SAE International, 1987.

132. Ranz, W. E., "Evaporation from drops : Part II", *Chemical Engineering Progress*, Vol. 48, pp. 173–180, 1952.

133. Miller, R., K. Harstad and J. Bellan, "Evaluation of Equilibrium and Non-Equilibrium Evaporation Models for Many-Droplet Gas-Liquid Flow Simulations", *International Journal of Multiphase Flow*, Vol. 24, No. 6, pp. 1025–1055, 1998.

134. Sazhin, S. S., "Advanced Models of Fuel Droplet Heating and Evaporation", *Progress in Energy and Combustion Science*, Vol. 32, No. 2, pp. 162–214, 2006.

135. Kuo, K. K., *Principles of Combustion*, Elsevier Science Publishing Company Inc., New York, NY, 1986.

136. Xin, J., L. Ricart and R. D. Reitz, "Computer Modeling of Diesel Spray Atomization and Combustion", *Combustion Science and Technology*, Vol. 137, No. 1-6, pp. 171–194, 1998.

137. Beale, J. C. and R. D. Reitz, "Modeling Spray Atomization with the Kelvin-Helmholtz/Rayleigh-Taylor Hybrid Model", *Atomization and Sprays*, Vol. 9, No. 6, pp. 623–650, 1999.

138. Friedlander, S. and A. Lipton-Lifschitz, "Chapter 8 - Localized Instabilities in Fluids", S. Friedlander and D. Serre (Editors), *Handbook of Mathematical Fluid*

Dynamics, Vol. 2, pp. 289–354, North-Holland, 2003.

139. Reitz, R. D., "Mechanisms of Breakup of Round Liquid Jets", *Encyclopedia of Fluid Mechanics*, Vol. 10, 1986.

140. Patterson, M. A. and R. D. Reitz, "Modeling the Effects of Fuel Spray Characteristics on Diesel Engine Combustion and Emission", *SAE Technical Paper*, 980131, SAE International, 1998.

141. Levich, V. G., *Physicochemical Hydrodynamics*, Prentice-Hall, Englewood Cliffs, N.J., 1967.

142. Hiroyasu, H. and M. Arai, "Structures of Fuel Sprays in Diesel Engines", *SAE Technical Paper*, SAE International, 1990.

143. Naber, J. D. and D. L. Siebers, "Effects of Gas Density and Vaporization on Penetration and Dispersion of Diesel Sprays", *International Congress & Exposition*, SAE International, 1996.

144. Abani, N., A. Munnannur and R. D. Reitz, "Reduction of Numerical Parameter Dependencies in Diesel Spray Models", *Journal of Engineering for Gas Turbines and Power*, Vol. 130, No. 3, 2008.

145. Goodwin, D. G., H. K. Moffat and R. L. Speth, "Cantera: an Object-oriented Software Toolkit for Chemical Kinetics, Thermodynamics, and Transport Processes", `https://www.cantera.org`, 2017, version 2.3.0.

146. MATLAB, *version 9.1.0.441655 (R2016b)*, The MathWorks Incorporated, Natick, Massachusetts, 2016.

147. Ansys, *ANSYS Fluent R19.2 Customization Manual*, ANSYS Incorporated, 2018.

148. Muradoglu, M., P. Jenny, S. Pope and D. A. Caughey, "A Consistent Hybrid

Finite-Volume/Particle Method for the PDF Equations of Turbulent Reactive Flows", *Journal of Computational Physics*, Vol. 154, No. 2, pp. 342–371, 1999.

149. Issa, R., "Solution of the Implicitly Discretised Fluid Flow Equations by Operator-Splitting", *Journal of Computational Physics*, Vol. 62, No. 1, pp. 40–65, 1986.

150. Gardiner, W., V. Lissianski, Z. Qin, G. Smith, D. Golden, M. Frenklach, B. Eiteneer, M. Goldenberg, N. Moriarty, C. Bowman *et al.*, "The GRI-Mechtm Model for Natural Gas Combustion and No Formation and Removal Chemistry", *5th International Conference on Combustion Technologies for a Clean Environment, July 12-15, 1999, Lisbon, Portugal*, 1999.

151. CMT-Motores Térmicos, "CMT Engine Combustion Network", `https://www.cmt.upv.es/ECN03.aspx`, accessed in December 2021.

152. Subramaniam, S., "Lagrangian–Eulerian Methods for Multiphase Flows", *Progress in Energy and Combustion Science*, Vol. 39, No. 2, pp. 215–245, 2013.

153. Farrace, D., R. Panier, M. Schmitt, K. Boulouchos and Y. M. Wright, "Analysis of Averaging Methods for Large Eddy Simulations of Diesel Sprays", *SAE International Journal of Fuels and Lubricants*, Vol. 8, No. 3, pp. 568–580, 2015.

154. Bikas, G. and N. Peters, "Kinetic Modelling of n-Decane Combustion and Autoignition: Modeling Combustion of n-Decane", *Combustion and Flame*, Vol. 126, No. 1, pp. 1456 – 1475, 2001.

155. Narayanaswamy, K., G. Blanquart and H. Pitsch, "A Consistent Chemical Mechanism for Oxidation of Substituted Aromatic Species", *Combustion and Flame*, Vol. 157, No. 10, pp. 1879–1898, 2010.

156. Sarathy, S., C. Westbrook, M. Mehl, W. Pitz, C. Togbe, P. Dagaut, H. Wang, M. Oehlschlaeger, U. Niemann, K. Seshadri, P. Veloo, C. Ji, F. Egolfopoulos and T. Lu, "Comprehensive Chemical Kinetic Modeling of the Oxidation of 2-

Methylalkanes from C7 to C20", *Combustion and Flame*, Vol. 158, No. 12, pp. 2338 – 2357, 2011.

157. Lu, T., M. Plomer, Z. Luo, S. Sarathy, W. Pitz, S. Som and D. Longman, *Directed relation graph with expert knowledge for skeletal mechanism reduction*, Tech. rep., Lawrence Livermore National Laboratories, Livermore, CA (United States), 2011.

158. Narayanaswamy, K., P. Pepiot and H. Pitsch, "A Chemical Mechanism for Low to High Temperature Oxidation of n-Dodecane As a Component of Transportation Fuel Surrogates", *Combustion and Flame*, Vol. 161, No. 4, pp. 866 – 884, 2014.

159. Ranzi, E., A. Frassoldati, A. Stagni, M. Pelucchi, A. Cuoci and T. Faravelli, "Reduced Kinetic Schemes of Complex Reaction Systems: Fossil and Biomass-Derived Transportation Fuels", *International Journal of Chemical Kinetics*, Vol. 46, No. 9, pp. 512–542, 2014.

160. Yao, T., Y. Pei, B.-J. Zhong, S. Som and T. Lu, "A Hybrid Mechanism for n-Dodecane Combustion with Optimized Low-Temperature Chemistry", *9th US National Combustion Meeting*, Vol. 5, 2015.

161. Abianeh, O. S., M. A. Oehlschlaeger and C.-J. Sung, "Turbulent Spray Combustion Simulations Based on a New Skeletal Mechanism for n-Dodecane", *ILASS Americas 27th Annual Conference on Liquid Atomization and Spray Systems, Raleigh, NC*, 2015.

162. Frassoldati, A., G. D'Errico, T. Lucchini, A. Stagni, A. Cuoci, T. Faravelli, A. Onorati and E. Ranzi, "Reduced Kinetic Mechanisms of Diesel Fuel Surrogate for Engine CFD Simulations", *Combustion and Flame*, Vol. 162, No. 10, pp. 3991–4007, 2015.

163. Pei, Y., M. Mehl, W. Liu, T. Lu, W. J. Pitz and S. Som, "A Multicomponent Blend as a Diesel Fuel Surrogate for Compression Ignition Engine Applications",

Journal of Engineering for Gas Turbines and Power, Vol. 137, No. 11, 2015.

164. Kourdis, P. D. and J. Bellan, "Twenty-Species and 15-Species Chemical Kinetic Mechanisms for Cyclohexane Using the Local Self-Similarity Tabulation Method", *International Journal of Chemical Kinetics*, Vol. 52, No. 8, pp. 526–547, 2020.

165. Westbrook, C. K., W. J. Pitz, O. Herbinet, H. J. Curran and E. J. Silke, "A Detailed Chemical Kinetic Reaction Mechanism for n-Alkane Hydrocarbons from n-Octane to n-Hexadecane", *Combustion and Flame*, Vol. 156, pp. 181–199, 2009.

166. Pickett, L. M., C. L. Genzale and J. Manin, "Uncertainty Quantification for Liquid Penetration of Evaporating Sprays at Diesel-like Conditions", *Atomization and Sprays*, Vol. 25, No. 5, pp. 425–452, 2015.

APPENDIX A: LS2T Script in C++

The LS2T script named as KourdisCodes.cpp is included as below.

```
// Windows comnpiler:
/*#include "stdafx.h" //TBD
#include "KourdisCodes.h" //TBD
#include <string>
#include <vector>
#include <numeric>
#include <iostream> //TBD
#include "GlobalVar.h"
#include <Windows.h>

using namespace std;
//typedef unsigned char BYTE;
bool debugTime = true;
double WeightingPower = 3;
*/

#include "KourdisCodes.h"
#include <string>
#include <vector>
#include <numeric>
#include <cstring>
#include <fstream>
#include <cmath>
//#include "cantera/base/global.h"
//@170919: Additional energy term for ideal gas constant volume
   reactor
//@171008: Removed unnecesarry functions
//@171227: Updated as per new file format
//@181112: Tn_vect to stay within limits of 0-1

//using namespace Cantera;
using namespace std;
```

```
32 typedef unsigned char BYTE;
33 //double PhiWF_Limit = 0.9;
34
35
36 double avg(vector<double> &v, int beg, int endi)
37 {
38   double return_value = accumulate(v.begin() + beg, v.begin() + endi +
       1, 0.0);
39   return (return_value / (endi - beg + 1));
40 }
41
42
43 vector<vector<double>> SortDualVecBynthColumnAccending(vector<vector<
     double>> &VectToSort, int Col){
44
45   vector<vector<double>> TempVect = VectToSort;
46   vector<vector<double>> Outputector(TempVect.size());
47   int MinIndex = 0;
48   int k = 0;
49
50   for (int i = 0; i < (int)VectToSort.size(); i++){
51     MinIndex = 0;
52     for (int j = 0; j < (int)TempVect.size(); j++){
53       if (TempVect[MinIndex][Col - 1] > TempVect[j][Col - 1]){
54         MinIndex = j;
55       }
56     }
57     Outputector[i] = TempVect[MinIndex];
58     TempVect.erase(TempVect.begin() + MinIndex);
59   }
60
61   return Outputector;
62 }
63
64 double interpolateData4(vector<double> &x, vector<double> &y, double
     newX, int &Loca){
65   //v4 is for single value only
```

```
66  vector<double> OutputVector;
67  int NumberOfDataRows = (int)x.size();
68  double newy = 0;
69
70  int LocationofClosestData = Loca;
71
72  //int LocationofClosestData = 0;
73  int PreviousDataLocation = 100000000;
74  double Smallest_DistanceOf_X_tobeCalculated;
75  int FirstDataRow, LastDataRow;
76  double x_jump, old_value, old_distance, new_value, new_distance,
     Old_DistanceOf_X_tobeCalculated, DistanceOf_X_tobeCalculated;
77  double grad1, grad2_3, x_m5, y_m5, x_m10, y_m10;
78
79
80  Smallest_DistanceOf_X_tobeCalculated = 100000000000;
81  // Finds the location of closest data points.
82  if (LocationofClosestData == -1){
83    for (int j = 0; j < NumberOfDataRows; j++){
84      DistanceOf_X_tobeCalculated = newX - x[j];
85      if (abs(DistanceOf_X_tobeCalculated) < abs(
     Smallest_DistanceOf_X_tobeCalculated)){
86        Smallest_DistanceOf_X_tobeCalculated =
     DistanceOf_X_tobeCalculated;
87        LocationofClosestData = j;
88      }
89      if (DistanceOf_X_tobeCalculated*
     Smallest_DistanceOf_X_tobeCalculated < 0){
90        break;
91      }
92    }
93    Loca = LocationofClosestData;
94  }
95
96
97  //Compares closest data points with the ones previously calculated
98  if (LocationofClosestData == 0 && newX < x[0]){
```

```
99      // if gradient is too high for first few numbers it would give
        over
100     // estimated or under estimated values.A smoothing is done.
101     grad1 = (y[1] - y[0]) / (x[1] - x[0]);
102     grad2_3 = (y[2] - y[1]) / (x[2] - x[1]);
103     if (abs(grad1 / grad2_3) > 2 || abs(grad1 / grad2_3) < 0.5){
104       x_m5 = avg(x, 0, 4);
105       y_m5 = avg(y, 0, 4);
106       x_m10 = avg(x, 5, 9);
107       y_m10 = avg(y, 5, 9);
108       newy = (y_m10 - y_m5) / (x_m10 - x_m5)*(newX - x_m5) + y_m5;
109       //Cantera::writelogFile("Inf0-1: Extrapolation with Smoothing,
        data begining\n");
110     }
111     else{
112       newy = grad1*(newX - x[0]) + y[0];
113       //Cantera::writelogFile("Inf0-2: Extrapolation with Smoothing\n
        ");
114     }
115   }
116   else if (LocationofClosestData == NumberOfDataRows - 1 && newX> x[
        NumberOfDataRows - 1]){
117     newy = (y[NumberOfDataRows - 1] - y[NumberOfDataRows - 2]) / (x[
        NumberOfDataRows - 1] - x[NumberOfDataRows - 2])*(newX - x[
        NumberOfDataRows - 2]) + y[NumberOfDataRows - 2];
118     //Cantera::writelogFile("Inf1-1: Extrapolation, data at the end\n
        ");
119   }
120   else if (newX> x[LocationofClosestData]){
121     newy = (y[LocationofClosestData] * (x[LocationofClosestData + 1] -
        newX) + y[LocationofClosestData + 1] * (newX - x[
        LocationofClosestData])) / (x[LocationofClosestData + 1] - x[
        LocationofClosestData]);
122     //Cantera::writelogFile("Inf2-1: Interpolation, data after closest
        point\n");
123   }
124   else if (newX< x[LocationofClosestData]){
```

```
125     newy = (y[LocationofClosestData - 1] * (x[LocationofClosestData] -
        newX) + y[LocationofClosestData] * (newX - x[
      LocationofClosestData - 1])) / (x[LocationofClosestData] - x[
      LocationofClosestData - 1]);
126     //Cantera::writelogFile("Inf3-1: Interpolation, data after closest
        point\n");
127   }
128   else if (newX == x[LocationofClosestData]){
129     newy = y[LocationofClosestData];
130     //Cantera::writelogFile("Inf4-1: Exact Value, data at closest
      point\n");
131   }
132   else {
133     //Cantera::writelogFile("Error\n");
134   }
135
136   if (newy > 1){
137     newy = 1;
138   }
139   else if (newy < 0){
140     newy = 0;
141   }else if (!(newy >= 0 && newy <= 1)){
142     newy = 0;
143   }
144
145   return newy;
146 }
147
148 vector<double> GetRangeofDoubleVector(vector<double> DatatoWork, int
      Startt, int Endd){
149
150
151
152
153
154   vector<double>::const_iterator first = DatatoWork.begin() + Startt;
155   vector<double>::const_iterator last = DatatoWork.begin() + Endd + 1;
```

```
156   vector<double> Output(first, last);
157
158
159
160
161
162   /*
163   vector<double> Output(Endd - Startt + 1);
164   int j = 0;
165   for (int i = Startt; i <= Endd; i++){
166     Output[j] = DatatoWork[i];
167     j++;
168   }
169
170
171
172   */
173   return Output;
174 }
175
176 bool InterpolatebyInverseDistanceWeigthed36_wlhn_re(double Pi, double
      Ti, double Phi_i, bool MaxReached, vector<double> &output_args,
      double* AdditionalConstants,vector<vector<double>> DataRead){
177
178   /*Cantera::writelogFile("#### Point to Interpolate p,T,Phi: ");
179   Cantera::writelogFile(Cantera::FloatToStr(Pi));
180   Cantera::writelogFile(" ");
181   Cantera::writelogFile(Cantera::FloatToStr(Ti));
182   Cantera::writelogFile(" ");
183   Cantera::writelogFile(Cantera::FloatToStr(Phi_i));
184   Cantera::writelogFile("\n");*/
185
186   WriteOut("Running: InterpolatebyInverseDistanceWeigthed36_wlhn_re\n"
      ,2);
187
188   double WeightingPower = AdditionalConstants[0];//default 3
189   double PhiFirst = AdditionalConstants[1];
```

```
190  double PhiWF_Limit = AdditionalConstants[2];
191  double TemperatureSearchRange = AdditionalConstants[3];
192
193  if(Phi_i>PhiWF_Limit*PhiFirst && PhiWF_Limit>0){
194    WeightingPower = 1;
195  }
196
197
198  time_t timer1, timer2;
199  double seconds;
200
201
202  /*
203  if(MaxReached){
204    //DataRead = Cantera::GetHeavyData(2);
205    DataRead = Cantera::GetHeavyData(2);
206    //Cantera::writelog("Using Second File\n");
207  }else{
208    DataRead = Cantera::GetHeavyData(1);
209  }
210  */
211
212  int NnumberofSpecies = (int)DataRead[0][2];
213  int NoPsiSections = (int)DataRead[0][3];
214  //bool ToBeURFed = false;
215
216  int NumberofRanges = 4 + NoPsiSections - 1;
217  //PhiSections.resize(NoPsiSections);
218
219  vector<double> TempData = DataRead[0];
220  vector<double>::const_iterator first = TempData.begin() + 4;
221  vector<double>::const_iterator last = TempData.begin() +
     NumberofRanges + 1;
222  vector<int> PhiSections(first, last);
223
224  int NUmberOfFiles = (int)DataRead[0][4 + NoPsiSections];
225  int NumberofDataPoints = (int)DataRead[0][1] / NUmberOfFiles;
```

```
226  double MaxPhi = DataRead[0][5 + NoPsiSections];
227
228  vector<double> output_args_expected(NnumberofSpecies + 5);
229  vector<double> interpolated_output_args(NnumberofSpecies + 5);
230  output_args.resize(NnumberofSpecies + 5);
231
232  double Phi_max = DataRead[0][NoPsiSections + 5];
233  double Tmax = DataRead[0][NoPsiSections + 6];
234  double Tmin = DataRead[0][NoPsiSections + 7];
235  double Pmax = DataRead[0][NoPsiSections + 10];
236  double Pmin = DataRead[0][NoPsiSections + 11];
237
238  //Cantera::writelogFile("### Phimax Tmax Tmin Pmax Pmin:" );
239  //Cantera::writelogFile(Cantera::FloatToStr(Phi_max));
240  //Cantera::writelogFile(" ");
241  //Cantera::writelogFile(Cantera::FloatToStr(Tmax));
242  //Cantera::writelogFile(" ");
243  //Cantera::writelogFile(Cantera::FloatToStr(Tmin));
244  //Cantera::writelogFile(" ");
245  //Cantera::writelogFile(Cantera::FloatToStr(Pmax));
246  //Cantera::writelogFile(" ");
247  //Cantera::writelogFile(Cantera::FloatToStr(Pmin));
248  //Cantera::writelogFile("\n");
249
250  double Gradi;
251
252  double Pnorm = (Pmax - Pmin);
253  double Tnorm = (Tmax - Tmin);
254  double Phinorm = Phi_max;
255  double Tn_vect;
256  //time(&timer1);
257
258  vector<int> FoundFiles = Find_SurroundingDataSeries_RandomFiles20(Pi
      , Ti, Phi_i, Pnorm, Tnorm, Phinorm, MaxReached, DataRead,
      TemperatureSearchRange);//v3 uses 8 corners, v4 finds nearest
      points.
259
```

```
260
261 int NumberOfFoundFiles = FoundFiles.size();
262
263 vector<vector<double>> ValOppose(NumberOfFoundFiles, vector<double>(
     NnumberofSpecies + 3)); // Defaults to zero initial value
264 vector<vector<double>> max_ValOppose(NumberOfFoundFiles, vector<
     double>(NnumberofSpecies + 4));
265 vector<vector<double>> min_ValOppose(NumberOfFoundFiles, vector<
     double>(NnumberofSpecies + 4));
266 vector<double> sum_max_ValOppose(NnumberofSpecies + 4);
267 vector<double> sum_min_ValOppose(NnumberofSpecies + 4);
268 vector<double> final_max_ValOppose(NnumberofSpecies + 4);
269 vector<double> final_min_ValOppose(NnumberofSpecies + 4);
270
271 vector<double> WeigthF1(NumberOfFoundFiles);
272 vector<double> distance(NumberOfFoundFiles);
273 vector<double> distance_phi(NumberOfFoundFiles);
274
275
276
277
278
279 if (NumberOfFoundFiles > 0){
280
281   double WeightTot = 0;
282   vector<double> SummAll_wlh(NnumberofSpecies);
283   //Interpolate Time
284   double SummAll_tim = 0;
285   double SummAll_ener = 0;
286   double SummAll_cv = 0;
287
288   // First loop to calculate extrema points
289
290   for (int i = 0; i < NumberOfFoundFiles; i++){
291
292     int locate = -1;
293
```

```
294       int DataPoint = FoundFiles[i];
295
296       double k[] = { DataRead[1][1 + NumberofDataPoints*(DataPoint)],
      DataRead[1][2 + NumberofDataPoints*(DataPoint)], DataRead[1][3 +
      NumberofDataPoints*(DataPoint)], DataRead[1][4 +
      NumberofDataPoints*(DataPoint)] };
297
298
299       TempData = DataRead[2];
300       double max_temp = TempData[NumberofDataPoints*(DataPoint + 1) -
      2];
301       double min_temp = TempData[NumberofDataPoints*(DataPoint + 1) -
      1];
302       max_ValOppose[i][NnumberofSpecies + 3] = max_temp;
303       min_ValOppose[i][NnumberofSpecies + 3] = min_temp;
304
305       TempData = DataRead[3];
306       double max_eh_dot_DYhDt = TempData[NumberofDataPoints*(DataPoint
       + 1) - 2];
307       double min_eh_dot_DYhDt = TempData[NumberofDataPoints*(DataPoint
       + 1) - 1];
308       max_ValOppose[i][NnumberofSpecies + 1] = max_eh_dot_DYhDt;;
309       min_ValOppose[i][NnumberofSpecies + 1] = min_eh_dot_DYhDt;
310
311       TempData = DataRead[4];
312       double max_cvh_dot_Yh = TempData[NumberofDataPoints*(DataPoint +
       1) - 2];
313       double min_cvh_dot_Yh = TempData[NumberofDataPoints*(DataPoint +
       1) - 1];
314       max_ValOppose[i][NnumberofSpecies + 2] = max_cvh_dot_Yh;
315       min_ValOppose[i][NnumberofSpecies + 2] = min_cvh_dot_Yh;
316
317       double Boun_T = k[1];
318       double Boun_P = k[0];
319       double Boun_Phi = k[2];
320
321       distance[i] = sqrt(pow(((Pi - Boun_P) / Pnorm), 2)
```

```
322         + pow(((Ti - Boun_T) / Tnorm), 2)
323         + pow(((Phi_i - Boun_Phi) / Phinorm), 2));
324
325
326
327
328       if (pow(distance[i], 3) != 0){
329         WeigthF1[i] = 1 / pow(distance[i], WeightingPower);// Power
    value determines influence of closest points, increase of power
    increases influence
330       }
331       else{
332         WeigthF1[i] = 10000000000;
333       }
334
335       distance_phi[i] = abs((Phi_i - Boun_Phi) / Phinorm);
336
337
338       sum_min_ValOppose[NnumberofSpecies + 1] = min_ValOppose[i][
    NnumberofSpecies + 1] * WeigthF1[i] + sum_min_ValOppose[
    NnumberofSpecies + 1];
339       sum_min_ValOppose[NnumberofSpecies + 2] = min_ValOppose[i][
    NnumberofSpecies + 2] * WeigthF1[i] + sum_min_ValOppose[
    NnumberofSpecies + 2];
340       sum_min_ValOppose[NnumberofSpecies + 3] = min_ValOppose[i][
    NnumberofSpecies + 3] * WeigthF1[i] + sum_min_ValOppose[
    NnumberofSpecies + 3];
341
342       sum_max_ValOppose[NnumberofSpecies + 1] = max_ValOppose[i][
    NnumberofSpecies + 1] * WeigthF1[i] + sum_max_ValOppose[
    NnumberofSpecies + 1];
343       sum_max_ValOppose[NnumberofSpecies + 2] = max_ValOppose[i][
    NnumberofSpecies + 2] * WeigthF1[i] + sum_max_ValOppose[
    NnumberofSpecies + 2];
344       sum_max_ValOppose[NnumberofSpecies + 3] = max_ValOppose[i][
    NnumberofSpecies + 3] * WeigthF1[i] + sum_max_ValOppose[
    NnumberofSpecies + 3];
```

```
345
346      vector<double> max_w_lh(NnumberofSpecies);
347      vector<double> min_w_lh(NnumberofSpecies);
348      //calculate for(int all heavy
349      for (int SpeciesNum = 0; SpeciesNum < NnumberofSpecies;
SpeciesNum++){
350        TempData = DataRead[5 + SpeciesNum];
351        max_w_lh[SpeciesNum] = TempData[NumberofDataPoints*(DataPoint
+ 1) - 2];
352        min_w_lh[SpeciesNum] = TempData[NumberofDataPoints*(DataPoint
+ 1) - 1];
353
354        max_ValOppose[i][SpeciesNum] = max_w_lh[SpeciesNum];
355        min_ValOppose[i][SpeciesNum] = min_w_lh[SpeciesNum];
356
357        sum_max_ValOppose[SpeciesNum] = max_ValOppose[i][SpeciesNum] *
 WeigthF1[i] + sum_max_ValOppose[SpeciesNum];
358        sum_min_ValOppose[SpeciesNum] = min_ValOppose[i][SpeciesNum] *
 WeigthF1[i] + sum_min_ValOppose[SpeciesNum];
359      }
360
361
362      WeightTot = WeightTot + WeigthF1[i];
363    }
364
365
366
367    for (int SpeciesNum = 0; SpeciesNum < NnumberofSpecies+4;
SpeciesNum++){
368      final_max_ValOppose[SpeciesNum] = sum_max_ValOppose[SpeciesNum]
/ WeightTot;
369      final_min_ValOppose[SpeciesNum] = sum_min_ValOppose[SpeciesNum]
/ WeightTot;
370    }
371
372
373
```

```
374     for (int i = 0; i < NumberOfFoundFiles; i++){
375
376       int locate = -1;
377       int DataPoint = FoundFiles[i];
378
379       double k[] = { DataRead[1][1 + NumberofDataPoints*(DataPoint)],
    DataRead[1][2 + NumberofDataPoints*(DataPoint)], DataRead[1][3 +
    NumberofDataPoints*(DataPoint)], DataRead[1][4 +
    NumberofDataPoints*(DataPoint)] };
380
381
382       //time(&timer1);
383
384       TempData = DataRead[2];
385       first = TempData.begin() + NumberofDataPoints*(DataPoint);
386       last = TempData.begin() + NumberofDataPoints*(DataPoint + 1);
387       vector<double> temp_n_re(first, last-2);
388
389
390       TempData = DataRead[3];
391       first = TempData.begin() + NumberofDataPoints*(DataPoint);
392       last = TempData.begin() + NumberofDataPoints*(DataPoint + 1);
393       vector<double> eh_dot_DYhDt_n_re(first, last-2);
394
395
396       TempData = DataRead[4];
397       first = TempData.begin() + NumberofDataPoints*(DataPoint);
398       last = TempData.begin() + NumberofDataPoints*(DataPoint + 1);
399       vector<double> cvh_dot_Yh_n_re(first, last-2);
400
401       double Boun_T = k[1];
402       double Boun_P = k[0];
403       double Boun_Phi = k[2];
404
405
406
407       Tn_vect = (Ti - final_min_ValOppose[NnumberofSpecies + 3]) / (
```

```
final_max_ValOppose[NnumberofSpecies + 3] - final_min_ValOppose[
NnumberofSpecies + 3]);
408
409       if(Tn_vect>1){
410         Tn_vect=1;
411       }else if(Tn_vect<0){
412         Tn_vect=0;
413       }
414
415       vector<double> w_lh_n_re_Spec(NumberofDataPoints);
416
417       //calculate for(int all heavy
418       for (int SpeciesNum = 0; SpeciesNum < NnumberofSpecies;
SpeciesNum++){
419         TempData = DataRead[5 + SpeciesNum];
420         first = TempData.begin() + NumberofDataPoints*(DataPoint);
421         last = TempData.begin() + NumberofDataPoints*(DataPoint + 1);
422         vector<double> w_lh_n_re_Spec(first, last-2);
423
424         if (SpeciesNum == 0){
425           //cout << "SGcomment..w_lh_n_re 1st DataPoint: " <<
w_lh_n_re_Spec[0] << "\n";
426           //cout << "SGcomment..w_lh_n_re Last DataPoint: " <<
w_lh_n_re_Spec[NumberofDataPoints-1] << "\n";
427         }
428         ValOppose[i][SpeciesNum] = interpolateData4(temp_n_re,
w_lh_n_re_Spec, Tn_vect, locate);
429
430         SummAll_wlh[SpeciesNum] = ValOppose[i][SpeciesNum] * WeigthF1[
i] + SummAll_wlh[SpeciesNum];
431       }
432
433
434
435
436       ValOppose[i][NnumberofSpecies + 1] = interpolateData4(temp_n_re,
 eh_dot_DYhDt_n_re, Tn_vect, locate);
```

```
437
438       ValOppose[i][NnumberofSpecies + 2] = interpolateData4(temp_n_re,
     cvh_dot_Yh_n_re, Tn_vect, locate);
439
440       SummAll_ener = ValOppose[i][NnumberofSpecies + 1] * WeigthF1[i]
    + SummAll_ener;
441       SummAll_cv = ValOppose[i][NnumberofSpecies + 2] * WeigthF1[i] +
    SummAll_cv;
442
443     }
444
445     interpolated_output_args = SummAll_wlh;//wlh1_n_interpolated
    WITHOUT DIVISION BY WEIGHT. IT IS DEVIDED IN THE NEXT FOR LOOP
446
447     interpolated_output_args.push_back(SummAll_tim / WeightTot);//
    tim_interpolated;
448     interpolated_output_args.push_back(SummAll_ener / WeightTot);//
    EnergyTerm_interpolated;
449     interpolated_output_args.push_back(SummAll_cv / WeightTot); //
    Cvh_dot_Yh_interpolated
450
451
452
453     for (int i = 0; i < NnumberofSpecies; i++){
454
455       output_args[i] = interpolated_output_args[i] / WeightTot * (
    final_max_ValOppose[i] - final_min_ValOppose[i]) +
    final_min_ValOppose[i];
456     }
457
458
459     output_args[NnumberofSpecies] = interpolated_output_args[
    NnumberofSpecies]; //Time
460     output_args[NnumberofSpecies + 1] = interpolated_output_args[
    NnumberofSpecies + 1] * (final_max_ValOppose[NnumberofSpecies + 1]
     - final_min_ValOppose[NnumberofSpecies + 1]) +
    final_min_ValOppose[NnumberofSpecies + 1];
```

```
461
462
463     output_args[NnumberofSpecies + 2] = interpolated_output_args[
        NnumberofSpecies + 2] * (final_max_ValOppose[NnumberofSpecies + 2]
        - final_min_ValOppose[NnumberofSpecies + 2]) +
        final_min_ValOppose[NnumberofSpecies + 2];
464     output_args[NnumberofSpecies + 3] = Tn_vect;
465     output_args[NnumberofSpecies + 4] = 1.0;
466     output_args[NnumberofSpecies + 5] = final_max_ValOppose[
        NnumberofSpecies + 3]; // MAximum local temperature
467
468     return true;
469
470   }else{
471     return false;
472     //Cantera::writelog("ERROR: No Surrounding Data Point Found.\n");
473   }
474
475
476
477   //Cantera::setIfCalled(true);
478 }
479
480 extern "C" int InterpolatebyInverseDistanceWeigthed37_2_wlhn_re(double
        Pi, double Ti, double Phi_i, int MaxReached, double*output_args,
        double* AdditionalConstants, double** DataRead, double*
        additional_outputs){
481
482   // For Fluent
483   WriteOut("Running: InterpolatebyInverseDistanceWeigthed37_2_wlhn_re\
        n",2);
484
485   double WeightingPower = AdditionalConstants[0];//default 3
486   double PhiFirst = AdditionalConstants[1];
487   double PhiWF_Limit = AdditionalConstants[2];
488   double TemperatureSearchRange = AdditionalConstants[3];
489
```

```
490   if(Phi_i>PhiWF_Limit*PhiFirst && PhiWF_Limit>0){
491     WeightingPower = 1;
492   }
493   //v17_2 works with array in includes additional energy term for
       ideal gas constant volume, this term is added as "mcvdTdt -=
       m_wdot_lh[NumofSpecies];"
494   time_t timer1, timer2;
495   double seconds;
496
497   /*
498   double** DataRead;
499
500   if (MaxReached){
501     //DataRead = Cantera::GetHeavyData(2);
502     DataRead = Cantera::GetHeavyArray(2);
503   //  WriteOut("Using Second File\n",2);
504   }
505   else{
506     DataRead = Cantera::GetHeavyArray(1);
507   }
508 */
509
510   //WriteOut("37_2 Point of Interest Pi, Ti,Phi: " + Cantera::
       FloatToStr(Pi) + " " + Cantera::FloatToStr(Ti) + " " + Cantera::
       FloatToStr(Phi_i)+"\n",2);
511
512
513   int NnumberofSpecies = (int)DataRead[0][2];
514   int NoPsiSections = (int)DataRead[0][3];
515   //bool ToBeURFed = false;
516
517
518
519   int NumberofRanges = 4 + NoPsiSections - 1;
520   //PhiSections.resize(NoPsiSections);
521
522   double* TempData = DataRead[0];
```

```
523  vector<int> PhiSections(&TempData[4], &TempData[4] + NoPsiSections);
524
525  int NUmberOfFiles = (int)DataRead[0][4 + NoPsiSections];
526  int NumberofDataPoints = (int)DataRead[0][1] / NUmberOfFiles;
527  double MaxPhi = DataRead[0][5 + NoPsiSections];
528
529  vector<double> output_args_expected(NnumberofSpecies + 5);
530  vector<double> interpolated_output_args(NnumberofSpecies + 5);
531  //output_args.resize(NnumberofSpecies + 5);
532
533  double Phi_max = DataRead[0][NoPsiSections + 5];
534  double Tmax = DataRead[0][NoPsiSections + 6];
535  double Tmin = DataRead[0][NoPsiSections + 7];
536  double Pmax = DataRead[0][NoPsiSections + 10];
537  double Pmin = DataRead[0][NoPsiSections + 11];
538
539  //double Tn_vect;//Tn_Actual = (Ti - Tmin) / (Tmax - Tmin);
540  double Gradi;
541
542  double Pnorm = (Pmax - Pmin);
543  double Tnorm = (Tmax - Tmin);
544  double Phinorm = Phi_max;
545  double Tn_vect;
546  //time(&timer1);
547
548
549  int NumberOfFoundFiles = 0;
550  vector<int> FoundFiles;
551
552
553  if (MaxReached){
554    WriteOut("Executing: Find_SurroundingDataSeries_RandomFiles20_5\n"
     ,2);
555    FoundFiles = Find_SurroundingDataSeries_RandomFiles20_5(Pi, Ti,
     Phi_i, Pnorm, Tnorm, Phinorm, MaxReached, DataRead,
     TemperatureSearchRange);
556  }
```

```
557   else{
558     WriteOut("Executing: Find_SurroundingDataSeries_RandomFiles21_4\n"
      ,2);
559     FoundFiles = Find_SurroundingDataSeries_RandomFiles21_4(Pi, Ti,
      Phi_i, Pnorm, Tnorm, Phinorm, MaxReached, DataRead,
      TemperatureSearchRange);//v3 uses 8 corners, v4 finds nearest
      points.
560
561   }
562
563   // Old format works with v20_2 and below, newer versions would not
      read the old files.
564   NumberOfFoundFiles = FoundFiles.size();
565   if (NumberOfFoundFiles < 8){ // If found files by corner is less
      than 4, then find the closest points.
566     FoundFiles = Find_SurroundingDataSeries_RandomFiles20_5(Pi, Ti,
      Phi_i, Pnorm, Tnorm, Phinorm, MaxReached, DataRead,
      TemperatureSearchRange);//v3 uses 8 corners, v4 finds nearest
      points.
567     NumberOfFoundFiles = FoundFiles.size();
568     //Cantera::writelog("Warning: Couldn't find at least 8 corner
      points. Working with nearest 8 points instead.\n");
569     //WriteOut("Number of corners found: " + Cantera::IntToStr(
      NumberOfFoundFiles) +"\n",2);
570   }
571   else{
572     //WriteOut("Number of corners found: " + Cantera::IntToStr(
      NumberOfFoundFiles) +"\n",2);
573   }
574
575
576   vector<vector<double>> ValOppose(NumberOfFoundFiles, vector<double>(
      NnumberofSpecies + 3)); // Defaults to zero initial value
577   vector<vector<double>> max_ValOppose(NumberOfFoundFiles, vector<
      double>(NnumberofSpecies + 4));
578   vector<vector<double>> min_ValOppose(NumberOfFoundFiles, vector<
      double>(NnumberofSpecies + 4));
```

```
579  vector<double> sum_max_ValOppose(NnumberofSpecies + 4);
580  vector<double> sum_min_ValOppose(NnumberofSpecies + 4);
581  vector<double> final_max_ValOppose(NnumberofSpecies + 4);
582  vector<double> final_min_ValOppose(NnumberofSpecies + 4);
583
584  vector<double> WeigthF1(NumberOfFoundFiles);
585  vector<double> distance(NumberOfFoundFiles);
586  vector<double> distance_phi(NumberOfFoundFiles);
587
588  if (NumberOfFoundFiles > 0){
589
590    double WeightTot = 0;
591    vector<double> SummAll_wlh(NnumberofSpecies);
592    //Interpolate Time
593    double SummAll_eh_mole_dot_w_lh = 0;
594    double SummAll_ener = 0;
595    double SummAll_cv = 0;
596
597
598
599    // First loop to calculate extrema points
600    for (int i = 0; i < NumberOfFoundFiles; i++){
601
602
603      int locate = -1;
604      int DataPoint = FoundFiles[i];
605      double k[] = { DataRead[1][1 + NumberofDataPoints*(DataPoint)],
     DataRead[1][2 + NumberofDataPoints*(DataPoint)], DataRead[1][3 +
     NumberofDataPoints*(DataPoint)], DataRead[1][4 +
     NumberofDataPoints*(DataPoint)] };
606
607      TempData = DataRead[2];
608      double max_temp = TempData[NumberofDataPoints*(DataPoint + 1) -
     2];
609      double min_temp = TempData[NumberofDataPoints*(DataPoint + 1) -
     1];
610      max_ValOppose[i][NnumberofSpecies + 3] = max_temp;
```

```
611       min_ValOppose[i][NnumberofSpecies + 3] = min_temp;
612
613       TempData = DataRead[3];
614       double max_eh_dot_DYhDt = TempData[NumberofDataPoints*(DataPoint
      + 1) - 2];
615       double min_eh_dot_DYhDt = TempData[NumberofDataPoints*(DataPoint
      + 1) - 1];
616       max_ValOppose[i][NnumberofSpecies + 1] = max_eh_dot_DYhDt;;
617       min_ValOppose[i][NnumberofSpecies + 1] = min_eh_dot_DYhDt;
618
619       TempData = DataRead[4];
620       double max_cvh_dot_Yh = TempData[NumberofDataPoints*(DataPoint +
      1) - 2];
621       double min_cvh_dot_Yh = TempData[NumberofDataPoints*(DataPoint +
      1) - 1];
622       max_ValOppose[i][NnumberofSpecies + 2] = max_cvh_dot_Yh;
623       min_ValOppose[i][NnumberofSpecies + 2] = min_cvh_dot_Yh;
624
625       TempData = DataRead[5];
626       double max_eh_mole_dot_w_lh = TempData[NumberofDataPoints*(
    DataPoint + 1) - 2];
627       double min_eh_mole_dot_w_lh = TempData[NumberofDataPoints*(
    DataPoint + 1) - 1];
628       max_ValOppose[i][NnumberofSpecies + 0] = max_eh_mole_dot_w_lh;
629       min_ValOppose[i][NnumberofSpecies + 0] = min_eh_mole_dot_w_lh;
630
631
632       double Boun_T = k[1];
633       double Boun_P = k[0];
634       double Boun_Phi = k[2];
635
636       //WriteOut("Working on File # " + Cantera::IntToStr(i) +"\n",2);
637       //WriteOut("  Found File DataPoint : " + Cantera::IntToStr(
    DataPoint) +"\n",2);
638       //WriteOut("  Point Found P,T,Phi: " + Cantera::FloatToStr(
    Boun_P) + " " + Cantera::FloatToStr(Boun_T) + " " + Cantera::
    FloatToStr(Boun_Phi) +"\n",2);
```

```
639
640       distance[i] = sqrt(pow(((Pi - Boun_P) / Pnorm), 2)
641         + pow(((Ti - Boun_T) / Tnorm), 2)
642         + pow(((Phi_i - Boun_Phi) / Phinorm), 2));
643
644       if (pow(distance[i], 3) != 0){
645         WeigthF1[i] = 1 / pow(distance[i], WeightingPower);// Power
    value determines influence of closest points, increase of power
    increases influence
646       }
647       else{
648         WeigthF1[i] = 10000000000;
649       }
650
651       //WriteOut("  WF " + Cantera::FloatToStr(WeigthF1[i]) +"\n",2);
652
653       distance_phi[i] = abs((Phi_i - Boun_Phi) / Phinorm);
654
655       sum_min_ValOppose[NnumberofSpecies + 0] = min_ValOppose[i][
    NnumberofSpecies + 0] * WeigthF1[i] + sum_min_ValOppose[
    NnumberofSpecies + 0];
656       sum_min_ValOppose[NnumberofSpecies + 1] = min_ValOppose[i][
    NnumberofSpecies + 1] * WeigthF1[i] + sum_min_ValOppose[
    NnumberofSpecies + 1];
657       sum_min_ValOppose[NnumberofSpecies + 2] = min_ValOppose[i][
    NnumberofSpecies + 2] * WeigthF1[i] + sum_min_ValOppose[
    NnumberofSpecies + 2];
658       sum_min_ValOppose[NnumberofSpecies + 3] = min_ValOppose[i][
    NnumberofSpecies + 3] * WeigthF1[i] + sum_min_ValOppose[
    NnumberofSpecies + 3];
659
660       sum_max_ValOppose[NnumberofSpecies + 0] = max_ValOppose[i][
    NnumberofSpecies + 0] * WeigthF1[i] + sum_max_ValOppose[
    NnumberofSpecies + 0];
661       sum_max_ValOppose[NnumberofSpecies + 1] = max_ValOppose[i][
    NnumberofSpecies + 1] * WeigthF1[i] + sum_max_ValOppose[
    NnumberofSpecies + 1];
```

```
662      sum_max_ValOppose[NnumberofSpecies + 2] = max_ValOppose[i][
    NnumberofSpecies + 2] * WeigthF1[i] + sum_max_ValOppose[
    NnumberofSpecies + 2];
663      sum_max_ValOppose[NnumberofSpecies + 3] = max_ValOppose[i][
    NnumberofSpecies + 3] * WeigthF1[i] + sum_max_ValOppose[
    NnumberofSpecies + 3];
664
665      vector<double> max_w_lh(NnumberofSpecies);
666      vector<double> min_w_lh(NnumberofSpecies);
667      //calculate for(int all heavy
668      for (int SpeciesNum = 0; SpeciesNum < NnumberofSpecies;
    SpeciesNum++){
669        TempData = DataRead[5 + SpeciesNum];
670        max_w_lh[SpeciesNum] = TempData[NumberofDataPoints*(DataPoint
    + 1) - 2];
671        min_w_lh[SpeciesNum] = TempData[NumberofDataPoints*(DataPoint
    + 1) - 1];
672
673        max_ValOppose[i][SpeciesNum] = max_w_lh[SpeciesNum];
674        min_ValOppose[i][SpeciesNum] = min_w_lh[SpeciesNum];
675
676        sum_max_ValOppose[SpeciesNum] = max_ValOppose[i][SpeciesNum] *
     WeigthF1[i] + sum_max_ValOppose[SpeciesNum];
677        sum_min_ValOppose[SpeciesNum] = min_ValOppose[i][SpeciesNum] *
     WeigthF1[i] + sum_min_ValOppose[SpeciesNum];
678      }
679
680
681      WeightTot = WeightTot + WeigthF1[i];
682    }
683
684    for (int SpeciesNum = 0; SpeciesNum < NnumberofSpecies + 4;
    SpeciesNum++){
685      final_max_ValOppose[SpeciesNum] = sum_max_ValOppose[SpeciesNum]
    / WeightTot;
686      final_min_ValOppose[SpeciesNum] = sum_min_ValOppose[SpeciesNum]
    / WeightTot;
```

```
687     }
688 
689     for (int i = 0; i < NumberOfFoundFiles; i++){
690 
691 
692        int locate = -1;
693        int DataPoint = FoundFiles[i];
694        double k[] = { DataRead[1][1 + NumberofDataPoints*(DataPoint)],
    DataRead[1][2 + NumberofDataPoints*(DataPoint)], DataRead[1][3 +
    NumberofDataPoints*(DataPoint)], DataRead[1][4 +
    NumberofDataPoints*(DataPoint)] };
695        TempData = DataRead[2];
696 
697        vector<double> temp_n_re(&TempData[NumberofDataPoints*(DataPoint
    )], &TempData[NumberofDataPoints*(DataPoint)+NumberofDataPoints
    -2]);
698 
699        TempData = DataRead[3];
700        vector<double> eh_dot_DYhDt_n_re(&TempData[NumberofDataPoints*(
    DataPoint)], &TempData[NumberofDataPoints*(DataPoint)+
    NumberofDataPoints-2]);
701 
702        TempData = DataRead[4];
703        vector<double> cvh_dot_Yh_n_re(&TempData[NumberofDataPoints*(
    DataPoint)], &TempData[NumberofDataPoints*(DataPoint)+
    NumberofDataPoints-2]);
704 
705        TempData = DataRead[5 + NnumberofSpecies];
706        vector<double> eh_mole_dot_w_lh(&TempData[NumberofDataPoints*(
    DataPoint)], &TempData[NumberofDataPoints*(DataPoint)+
    NumberofDataPoints-2]);
707 
708        double Boun_T = k[1];
709        double Boun_P = k[0];
710        double Boun_Phi = k[2];
711 
712        //Cantera::set_MaxTemperatureLimit(final_max_ValOppose[
```

```
   NnumberofSpecies + 3]);
713      additional_outputs[0] = final_max_ValOppose[NnumberofSpecies +
   3];
714
715      Tn_vect = (Ti - final_min_ValOppose[NnumberofSpecies + 3]) / (
   final_max_ValOppose[NnumberofSpecies + 3] - final_min_ValOppose[
   NnumberofSpecies + 3]);
716
717      if(Tn_vect>1){
718        Tn_vect=1;
719      }else if(Tn_vect<0){
720        Tn_vect=0;
721      }
722
723      /*
724      WriteOut("Working on File # " + Cantera::IntToStr(i) +"\n",2);
725      WriteOut("  Found File DataPoint : " + Cantera::IntToStr(
   DataPoint) +"\n",2);
726      WriteOut("  Point Found P,T,Phi: " + Cantera::FloatToStr(Boun_P)
    + " " + Cantera::FloatToStr(Boun_T) + " " + Cantera::FloatToStr(
   Boun_Phi) +"\n",2);
727 */
728      /*      distance[i] = sqrt(pow(((Pi - Boun_P) / Pnorm), 2)
729      + pow(((Ti - Boun_T) / Tnorm), 2)
730      + pow(((Phi_i - Boun_Phi) / Phinorm), 2));
731
732      if (pow(distance[i], 3) != 0){
733      WeigthF1[i] = 1 / pow(distance[i], WeightingPower);// Power
   value determines influence of closest points, increase of power
   increases influence
734      }
735      else{
736      WeigthF1[i] = 10000000000;
737      }
738
739      WriteOut("  WF "+Cantera::FloatToStr(WeigthF1[i]));
740
```

```
741       distance_phi[i] = abs((Phi_i - Boun_Phi) / Phinorm);
742
743       vector<double> OutputVect;
744       double Tn_vect = Tn_Actual; */
745
746       //vector<double> w_lh_n_re_Spec(NumberofDataPoints);
747
748       //calculate for(int all heavy
749
750       for (int SpeciesNum = 0; SpeciesNum < NnumberofSpecies;
    SpeciesNum++){
751         TempData = DataRead[5 + SpeciesNum];
752         vector<double> w_lh_n_re_Spec(&TempData[NumberofDataPoints*(
    DataPoint)], &TempData[NumberofDataPoints*(DataPoint)+
    NumberofDataPoints-2]);
753         ValOppose[i][SpeciesNum] = interpolateData4(temp_n_re,
    w_lh_n_re_Spec, Tn_vect, locate);
754         SummAll_wlh[SpeciesNum] = ValOppose[i][SpeciesNum] * WeigthF1[
    i] + SummAll_wlh[SpeciesNum];
755
756       //  WriteOut("  Interpolated wlh of Species #" + Cantera::
    IntToStr(SpeciesNum) + " : " + Cantera::FloatToStr(ValOppose[i][
    SpeciesNum]) +"\n",3);
757       }
758
759
760       ValOppose[i][NnumberofSpecies] = interpolateData4(temp_n_re,
    eh_mole_dot_w_lh, Tn_vect, locate);
761       ValOppose[i][NnumberofSpecies + 1] = interpolateData4(temp_n_re,
     eh_dot_DYhDt_n_re, Tn_vect, locate);
762       ValOppose[i][NnumberofSpecies + 2] = interpolateData4(temp_n_re,
     cvh_dot_Yh_n_re, Tn_vect, locate);
763
764       //WriteOut("  Interpolated eh_mole_dot_w_lh :" + Cantera::
    FloatToStr(ValOppose[i][NnumberofSpecies]) +"\n",3);
765       //WriteOut("  Interpolated eh_dot_DYhDt_n_re :" + Cantera::
    FloatToStr(ValOppose[i][NnumberofSpecies + 1]) +"\n",3);
```

```
766
767     SummAll_eh_mole_dot_w_lh = ValOppose[i][NnumberofSpecies] *
    WeigthF1[i] + SummAll_eh_mole_dot_w_lh;
768     SummAll_ener = ValOppose[i][NnumberofSpecies + 1] * WeigthF1[i]
    + SummAll_ener;
769     SummAll_cv = ValOppose[i][NnumberofSpecies + 2] * WeigthF1[i] +
    SummAll_cv;
770     //WeightTot = WeightTot + WeigthF1[i];
771    }
772
773    interpolated_output_args = SummAll_wlh;//wlh1_n_interpolated
    WITHOUT DIVISION BY WEIGHT. IT IS DEVIDED IN THE NEXT FOR LOOP
774
775    interpolated_output_args.push_back(SummAll_eh_mole_dot_w_lh /
    WeightTot);// tim_interpolated;
776    interpolated_output_args.push_back(SummAll_ener / WeightTot);//
    EnergyTerm_interpolated;
777    interpolated_output_args.push_back(SummAll_cv / WeightTot); //
    Cvh_dot_Yh_interpolated
778
779
780    for (int i = 0; i < NnumberofSpecies; i++){
781
782     output_args[i] = interpolated_output_args[i] / WeightTot * (
    final_max_ValOppose[i] - final_min_ValOppose[i]) +
    final_min_ValOppose[i];
783    }
784
785
786
787    output_args[NnumberofSpecies] = interpolated_output_args[
    NnumberofSpecies]; //Time
788    output_args[NnumberofSpecies + 1] = interpolated_output_args[
    NnumberofSpecies + 1] * (final_max_ValOppose[NnumberofSpecies + 1]
     - final_min_ValOppose[NnumberofSpecies + 1]) +
    final_min_ValOppose[NnumberofSpecies + 1];
789
```

```
790
791     output_args[NnumberofSpecies + 2] = interpolated_output_args[
        NnumberofSpecies + 2] * (final_max_ValOppose[NnumberofSpecies + 2]
         - final_min_ValOppose[NnumberofSpecies + 2]) +
        final_min_ValOppose[NnumberofSpecies + 2];
792     output_args[NnumberofSpecies + 3] = Tn_vect;
793     output_args[NnumberofSpecies + 4] = 1.0;
794     output_args[NnumberofSpecies + 5] = final_max_ValOppose[
        NnumberofSpecies + 3]; // MAximum local temperature
795
796         /*
797     output_args
798     0 -> NnumberofSpecies    : wlh term for all light species,
        unnormalized
799     NnumberofSpecies + 0   : Energy term for constant pressure reactor
        SummAll_eh_mole_dot_w_lh
800     NnumberofSpecies + 1   : Energy term for constant volume or
        entalphy solving reactor SummAll_ener
801     NnumberofSpecies + 2   : Specific heat term, Cp or Cv dependng on
        the heavy input file.
802     NnumberofSpecies + 3   : Tn_vect, normalized temperature based on
        interpolated maximum and minimum temperature limits
803     NnumberofSpecies + 4   : 1.0
804     NnumberofSpecies + 5   : Maximum temperature limit locally
805     */
806
807
808     //Kineticsmethods.ccp has : plhs[0] = mxCreateNumericMatrix(nsp
        +5,1,mxDOUBLE_CLASS,mxREAL);
809     return true;
810   }
811   else{
812     return false;
813     //Cantera::writelog("ERROR: No Surrounding Data Point Found.\n");
814   }
815
816
```

```
817   //Cantera::setIfCalled(true);
818   //WriteOut("Kourdis Interpolation completed succesfully!\n",2);
819 }
820
821
822 vector<int> Find_SurroundingDataSeries_RandomFiles20(double Pi, double
        Ti, double Phi_i, double Pnorm, double Tnorm, double Phinorm,
       bool MaxReached, vector<vector<double>> &DataRead, double
       RangeOfTemperatureToSearch){
823
824   /*
825
826   IMPORTANT:
827   Arrays Start from 0
828   CHECK If number of found files could be smaller than 8?
829   */
830
831   // Find all data files in folder
832   // Modified shepherd with R = 0.2 and pow 2
833   // Finds 16 closest points and sorts yb phi and takes 1st 8 points.
834   // v10 searches form all data in the memory
835   // v14 No prsistant but gets the alldata as an input
836   // v15 Modified for C, Exports only Data Locations in the array(//
       19 faster, overall increases speed by 5 // )
837   // v18 removed some for loops to speed up
838   // v19 works with row\colum changed file
839   //double RangeOfTemperatureToSearch = 20; //0.2 means + -20 // of
       target point
840
841   //double RangeOfTemperatureToSearch = Cantera::
       getTemperatureSearchRange(); //default 20
842
843   int NunOfPoints = 16; // should be even num.
844
845   // Decode DataRead:  these to be assigned to global variables after
       data is read.
846   int NoPsiSections = (int)DataRead[0][3];
```

```
847  int NumberofRanges = NoPsiSections + 3;
848
849  vector<double> TempData = DataRead[0];
850  vector<double>::const_iterator first = TempData.begin() + 4;
851  vector<double>::const_iterator last = TempData.begin() +
      NumberofRanges + 1;
852  //vector<double> Output(first, last);
853  vector<int> PhiSections(first, last);
854
855  int NUmberOfFiles = (int)DataRead[0][4 + NoPsiSections];
856  int NumOfRows = (int)DataRead[0][1] / NUmberOfFiles;
857  double MaxPhi = DataRead[0][5 + NoPsiSections];
858
859  vector<vector<double>> NEarestDataPoints(NunOfPoints, vector<double
      >(5, 1E15)); // Defaults to zero initial value
860  vector<vector<double>> NearestDataToExport(NunOfPoints, vector<
      double>(5, 1E15)); // Defaults to zero initial value
861  vector<vector<double>> distanceall(NUmberOfFiles, vector<double>(5))
      ; // Defaults to zero initial value
862
863
864
865  double PointOfInterest[] = { Pi, Ti, Phi_i };
866
867  double increment = MaxPhi / (NoPsiSections + 1);
868
869  int RangetoWork = (int)floor(Phi_i / increment);
870
871  int BoundMin, BoundMax;
872
873
874  if (RangetoWork == 0 || RangetoWork == 1){
875    BoundMin = 0;
876    BoundMax = PhiSections[RangetoWork + 1];
877  }
878  else if (RangetoWork == NoPsiSections || RangetoWork ==
      NoPsiSections - 1){
```

```
879     BoundMin = PhiSections[RangetoWork - 2];
880     BoundMax = NUmberOfFiles - 1;
881   }
882   else if (RangetoWork == NoPsiSections + 1){
883     BoundMin = PhiSections[RangetoWork - 3];
884     BoundMax = NUmberOfFiles - 1;
885   }
886   else{
887     BoundMin = PhiSections[RangetoWork - 2];
888     BoundMax = PhiSections[RangetoWork + 1];
889   }
890
891   double minDistance = 1000000000;
892   double maxDistance = 1000000000;
893   int counter = 0;
894   vector<double> TobeAdded = { 0, 0, 0, 0, 0 };
895
896   for (int i = BoundMin; i <= BoundMax; i++){
897
898     double k[] = { DataRead[1][1 + NumOfRows*(i)], DataRead[1][2 +
      NumOfRows*(i)], DataRead[1][3 + NumOfRows*(i)], DataRead[1][4 +
      NumOfRows*(i)] };
899
900     int MultiValRange = (int)k[3];
901
902     double Tdiff = abs((PointOfInterest[1] - k[1]));
903     double distance, distance1, distance2, distance3;
904
905
906
907
908
909     if (Tdiff <= RangeOfTemperatureToSearch){
910
911       distance1 = abs((PointOfInterest[0] - k[0]) / Pnorm); //
     PointOfInterest(1);
912       distance2 = abs((PointOfInterest[1] - k[1]) / Tnorm); //
```

```
  PointOfInterest(2);
913       distance3 = abs((PointOfInterest[2] - k[2]) / Phinorm); // /
  PointOfInterest(3);
914       distance = sqrt(pow(distance1, 2) + pow(distance2, 2) + pow(
  distance3, 2));
915
916       distanceall[i][0] = i;
917       distanceall[i][1] = distance;
918       distanceall[i][2] = distance1;
919       distanceall[i][3] = distance2;
920       distanceall[i][4] = distance3;
921
922
923       for (int j = 0; j < NunOfPoints; j++){
924
925         if (distance < NEarestDataPoints[j][1]){
926           TobeAdded = { (double)i, distance, distance1, distance2,
  distance3 };
927           NEarestDataPoints.insert(NEarestDataPoints.begin() + j,
  TobeAdded);
928           NEarestDataPoints.erase(NEarestDataPoints.begin() +
  NunOfPoints);
929           break;
930         }
931
932       }
933
934     }
935
936   }
937
938   double Mean_distance3 = 0.0;
939   for (int i = 0; i < NunOfPoints; i++)
940   {
941     Mean_distance3 += NEarestDataPoints[i][4];
942   }
943   Mean_distance3 = Mean_distance3 / NunOfPoints;
```

```
944
945   vector<vector<double>> NEarestDataPoints_SortedBYPhiDistance;
946
947   NEarestDataPoints_SortedBYPhiDistance =
       SortDualVecBynthColumnAccending(NEarestDataPoints, 5);
948
949   if (Phi_i < 1E-5){
950     NearestDataToExport = NEarestDataPoints_SortedBYPhiDistance;
951   }
952   else{
953     NearestDataToExport = NEarestDataPoints;
954   }
955
956
957   vector<int> FileNames(8);
958
959
960   for (int j = 0; j < (NunOfPoints / 2); j++){
961
962     if (NearestDataToExport[j][0] == 1E15){
963       break;
964     }
965     else{
966       FileNames[j] = (int)NearestDataToExport[j][0];
967     }
968   }
969
970
971   return FileNames;
972
973 }
974
975
976 vector<int> Find_SurroundingDataSeries_RandomFiles20_5(double Pi,
      double Ti, double Phi_i, double Pnorm, double Tnorm, double
      Phinorm, bool MaxReached, double** DataRead, double
      RangeOfTemperatureToSearch){
```

```
977
978  /*
979
980  IMPORTANT:
981  Arrays Start from 0
982  CHECK If number of found files could be smaller than 8?
983  */
984
985  // Find all data files in folder
986  // Modified shepherd with R = 0.2 and pow 2
987  // Finds 16 closest points and sorts yb phi and takes 1st 8 points.
988  // v10 searches form all data in the memory
989  // v14 No prsistant but gets the alldata as an input
990  // v15 Modified for C, Exports only Data Locations in the array(//
      19 faster, overall increases speed by 5 // )
991  // v18 removed some for loops to speed up
992  // v19 works with row\colum changed file
993
994  // v20 corrected
995  // v20_3 is for arrays again with corrections.
996  // v20_4 is for second brach, distance calculation based on T and P
      only, since Phi is too small to be copndisdered.
997
998  //double RangeOfTemperatureToSearch = 20; //0.2 means + -20 // of
      target point
999
1000 //double RangeOfTemperatureToSearch = Cantera::
      getTemperatureSearchRange(); //default 20
1001
1002 int NunOfPoints = 16; // should be even num.
1003
1004 // Decode DataRead:  these to be assigned to global variables after
      data is read.
1005 int NoPsiSections = (int)DataRead[0][3];
1006
1007 double* TempData = DataRead[0];
1008
```

```
1009
1010
1011
1012  vector<int> PhiSections(&TempData[4], &TempData[4] + NoPsiSections);
1013
1014
1015  int NUmberOfFiles = (int)DataRead[0][4 + NoPsiSections];
1016  PhiSections[NoPsiSections - 1] = NUmberOfFiles;
1017  int NumOfRows = (int)DataRead[0][1] / NUmberOfFiles;
1018  double MaxPhi = DataRead[0][5 + NoPsiSections];
1019
1020
1021
1022
1023
1024  double increment = DataRead[0][3 + NoPsiSections]; //MaxPhi / (
       NoPsiSections + 1);
1025  //double increment = MaxPhi / (NoPsiSections + 1);
1026
1027  vector<vector<double>> NEarestDataPoints(NunOfPoints, vector<double
       >(5, 1E15)); // Defaults to zero initial value
1028  vector<vector<double>> NearestDataToExport(NunOfPoints, vector<
       double>(5, 1E15)); // Defaults to zero initial value
1029  vector<vector<double>> distanceall(NUmberOfFiles, vector<double>(5))
       ; // Defaults to zero initial value
1030
1031  double PointOfInterest[] = { Pi, Ti, Phi_i };
1032
1033
1034
1035  int RangetoWork = (int)floor(Phi_i / increment);
1036
1037  int BoundMin, BoundMax;
1038
1039  if(MaxReached){
1040    RangeOfTemperatureToSearch = RangeOfTemperatureToSearch/2;
1041    BoundMin = 0;
```

```
1042      BoundMax = NUmberOfFiles - 1;
1043    }else{
1044
1045      if (RangetoWork == 0 || RangetoWork == 1){
1046        BoundMin = 0;
1047        BoundMax = PhiSections[RangetoWork + 1];
1048      }
1049      else if (RangetoWork == NoPsiSections - 1 || RangetoWork ==
          NoPsiSections - 2){
1050        BoundMin = PhiSections[RangetoWork - 2];
1051        BoundMax = NUmberOfFiles - 1;
1052      }
1053      else{
1054        BoundMin = PhiSections[RangetoWork - 2];
1055        BoundMax = PhiSections[RangetoWork + 1];
1056      }
1057    }
1058
1059
1060    double minDistance = 1000000000;
1061    double maxDistance = 1000000000;
1062    int counter = 0;
1063    vector<double> TobeAdded = { 0, 0, 0, 0, 0 };
1064
1065    //MaxReached = false;
1066
1067    for (int i = BoundMin; i <= BoundMax; i++){
1068
1069      double k[] = { DataRead[1][1 + NumOfRows*(i)], DataRead[1][2 +
          NumOfRows*(i)], DataRead[1][3 + NumOfRows*(i)], DataRead[1][4 +
          NumOfRows*(i)] };
1070
1071      int MultiValRange = (int)k[3];
1072
1073      double Tdiff = abs((PointOfInterest[1] - k[1]));
1074      double distance, distance1, distance2, distance3;
1075
```

```
1076
1077    if (Tdiff <= RangeOfTemperatureToSearch){
1078
1079       distance1 = abs((PointOfInterest[0] - k[0]) / Pnorm); //
    PointOfInterest(1);
1080       distance2 = abs((PointOfInterest[1] - k[1]) / Tnorm); //
    PointOfInterest(2);
1081       distance3 = abs((PointOfInterest[2] - k[2]) / Phinorm); // /
    PointOfInterest(3);
1082       /* if (MaxReached) {
1083         distance = sqrt(pow(distance1, 2) + pow(distance2, 2));
1084       }
1085       else{ */
1086         distance = sqrt(pow(distance1, 2) + pow(distance2, 2) + pow(
    distance3, 2));
1087       //}
1088       distanceall[i][0] = i;
1089       distanceall[i][1] = distance;
1090       distanceall[i][2] = distance1;
1091       distanceall[i][3] = distance2;
1092       distanceall[i][4] = distance3;
1093
1094
1095
1096       for (int j = 0; j < NunOfPoints; j++){
1097
1098         if (distance < NEarestDataPoints[j][1]){
1099           TobeAdded = { (double)i, distance, distance1, distance2,
    distance3 };
1100           NEarestDataPoints.insert(NEarestDataPoints.begin() + j,
    TobeAdded);
1101           NEarestDataPoints.erase(NEarestDataPoints.begin() +
    NunOfPoints);
1102           break;
1103         }
1104
1105
```

```
1106      }
1107
1108
1109    }
1110
1111  }
1112
1113  double Mean_distance3 = 0.0;
1114  for (int i = 0; i < NunOfPoints; i++)
1115  {
1116    Mean_distance3 += NEarestDataPoints[i][4];
1117  }
1118  Mean_distance3 = Mean_distance3 / NunOfPoints;
1119
1120  vector<vector<double>> NEarestDataPoints_SortedBYPhiDistance;
1121
1122  if (MaxReached) {
1123    NEarestDataPoints_SortedBYPhiDistance =
     SortDualVecBynthColumnAccending(NEarestDataPoints, 5);
1124    NearestDataToExport = NEarestDataPoints_SortedBYPhiDistance;
1125  }
1126  else{
1127    NEarestDataPoints_SortedBYPhiDistance =
     SortDualVecBynthColumnAccending(NEarestDataPoints, 5);
1128
1129    if (Phi_i < 1E-5){
1130      NearestDataToExport = NEarestDataPoints_SortedBYPhiDistance;
1131    }
1132    else{
1133      NearestDataToExport = NEarestDataPoints;
1134    }
1135  }
1136
1137  vector<int> FileNames(8);
1138
1139
1140  for (int j = 0; j < (NunOfPoints / 2); j++){
```

```
1141
1142      if (NearestDataToExport[j][0] == 1E15){
1143        break;
1144      }
1145      else{
1146        FileNames[j] = (int)NearestDataToExport[j][0];
1147      }
1148    }
1149
1150
1151    return FileNames;
1152
1153  }
1154
1155
1156  vector<int> Find_SurroundingDataSeries_RandomFiles21_4(double Pi,
       double Ti, double Phi_i, double Pnorm, double Tnorm, double
       Phinorm, bool MaxReached, double** DataRead, double
       RangeOfTemperatureToSearch){
1157
1158    /*
1159
1160    IMPORTANT:
1161    Arrays Start from 0
1162    CHECK If number of found files could be smaller than 8?
1163    */
1164
1165    // Find all data files in folder
1166    // Modified shepherd with R = 0.2 and pow 2
1167    // Finds 16 closest points and sorts yb phi and takes 1st 8 points.
1168    // v10 searches form all data in the memory
1169    // v14 No prsistant but gets the alldata as an input
1170    // v15 Modified for C, Exports only Data Locations in the array(//
       19 faster, overall increases speed by 5 // )
1171    // v18 removed some for loops to speed up
1172    // v19 works with row\colum changed file
1173
```

```
1174  // v20 corrected
1175  // v21 Finds corner points
1176  // v21_3 is for arrays again with corrections.
1177  // v21_4 is for second brach, distance calculation based on T and P
       only, since Phi is too small to be copndisdered.
1178
1179
1180
1181  //double RangeOfTemperatureToSearch = 50; //0.2 means + -20 // of
       target point
1182
1183  //double RangeOfTemperatureToSearch = Cantera::
       getTemperatureSearchRange(); //default 20
1184
1185  int NunOfPoints = 8; // should be even num.
1186
1187  // Decode DataRead:  these to be assigned to global variables after
       data is read.
1188  int NoPsiSections = (int)DataRead[0][3];
1189
1190
1191
1192
1193  double* TempData = DataRead[0];
1194
1195  vector<int> PhiSections(&TempData[4], &TempData[4] + NoPsiSections);
1196
1197
1198  int NUmberOfFiles = (int)DataRead[0][4 + NoPsiSections];
1199  PhiSections[NoPsiSections - 1] = NUmberOfFiles;
1200  int NumOfRows = (int)DataRead[0][1] / NUmberOfFiles;
1201  double MaxPhi = DataRead[0][5 + NoPsiSections];
1202
1203
1204  double increment = DataRead[0][3 + NoPsiSections]; //MaxPhi / (
       NoPsiSections + 1);
1205  //increment = MaxPhi / (NoPsiSections + 1);
```

```
1206
1207
1208  vector<vector<double>> NEarestDataPoints(NunOfPoints, vector<double
       >(5, 1E15)); // Defaults to zero initial value
1209  vector<vector<double>> NearestDataToExport(NunOfPoints, vector<
       double>(5, 1E15)); // Defaults to zero initial value
1210  vector<vector<double>> distanceall(NUmberOfFiles, vector<double>(5))
       ; // Defaults to zero initial value
1211
1212
1213
1214  double PointOfInterest[] = { Pi, Ti, Phi_i };
1215
1216  //WriteOut("21_4 Point of Interest Pi, Ti,Phi: " + Cantera::
       FloatToStr(Pi) + " " + Cantera::FloatToStr(Ti) + " "+Cantera::
       FloatToStr(Phi_i)+"\n",2);
1217
1218  //WriteOut("NormalizationVaues Pi, Ti,Phi: " + Cantera::FloatToStr(
       Pnorm) + " " + Cantera::FloatToStr(Tnorm) + " "+Cantera::
       FloatToStr(Phinorm)+"\n",2);
1219
1220  int RangetoWork = (int)floor(Phi_i / increment);
1221
1222  //WriteOut("Phi RangetoWork: " + Cantera::IntToStr(RangetoWork)+"\n
       ",2);
1223
1224  int BoundMin, BoundMax;
1225
1226
1227  if (RangetoWork == 0 || RangetoWork == 1){
1228    BoundMin = 0;
1229    BoundMax = PhiSections[RangetoWork + 1];
1230  }
1231  else if (RangetoWork == NoPsiSections - 1 || RangetoWork ==
       NoPsiSections - 2){
1232    BoundMin = PhiSections[RangetoWork - 2];
1233    BoundMax = NUmberOfFiles - 1;
```

```
1234  }
1235  else{
1236    BoundMin = PhiSections[RangetoWork - 2];
1237    BoundMax = PhiSections[RangetoWork + 1];
1238  }
1239
1240  //  WriteOut("BoundMin & BoundMax: " + Cantera::IntToStr(BoundMin) +
        " " + Cantera::IntToStr(BoundMax)+"\n" ,2 );
1241
1242  double minDistance = 1000000000;
1243  double maxDistance = 1000000000;
1244  int counter = 0;
1245  vector<double> TobeAdded = { 0, 0, 0, 0, 0 };
1246
1247  MaxReached=false;
1248
1249  for (int i = BoundMin; i <= BoundMax; i++){
1250
1251    double k[] = { DataRead[1][1 + NumOfRows*(i)], DataRead[1][2 +
     NumOfRows*(i)], DataRead[1][3 + NumOfRows*(i)], DataRead[1][4 +
     NumOfRows*(i)] };
1252
1253    int MultiValRange = (int)k[3];
1254
1255    double Tdiff = abs((PointOfInterest[1] - k[1]));
1256    double distance, distance1, distance2, distance3;
1257
1258    if (Tdiff <= RangeOfTemperatureToSearch){
1259
1260      if (MaxReached){
1261        // If max temperature is reached only consider T and P
     conditions
1262
1263        distance1 = ((PointOfInterest[0] - k[0]) / Pnorm); //
     PointOfInterest(1);
1264        distance2 = ((PointOfInterest[1] - k[1]) / Tnorm); //
     PointOfInterest(2);
```

```
1265        distance3 = ((PointOfInterest[2] - k[2]) / Phinorm); // /
    PointOfInterest(3);
1266
1267        distance = sqrt(pow(distance1, 2) + pow(distance2, 2));
1268
1269        distanceall[i][0] = i;
1270        distanceall[i][1] = distance;
1271        distanceall[i][2] = distance1;
1272        distanceall[i][3] = distance2;
1273        distanceall[i][4] = distance3;
1274
1275        int j = 0;
1276
1277        if (distance1*distance2 > 0){ // Group bounds based on their
    side
1278          if (distance1 > 0 && distance2 > 0 ){// % + +
1279            j = 0;
1280            if (distance < NEarestDataPoints[j][1]){
1281              TobeAdded = { (double)i, distance, distance1, distance2,
     distance3 };
1282              NEarestDataPoints[j] = TobeAdded;
1283            }
1284          }
1285          else{// % - -
1286            j = 1;
1287            if (distance < NEarestDataPoints[j][1]){//
1288              TobeAdded = { (double)i, distance, distance1, distance2,
     distance3 };
1289              NEarestDataPoints[j] = TobeAdded;
1290            }
1291          }
1292        }
1293        else{
1294          if (distance1 > 0 && distance2 < 0 ){// % + -
1295            j = 2;
1296            if (distance < NEarestDataPoints[j][1]){
1297              TobeAdded = { (double)i, distance, distance1, distance2,
```

```
 distance3 };
1298            NEarestDataPoints[j] = TobeAdded;
1299          }
1300        }
1301        else {// % - +
1302          j = 3;
1303          if (distance < NEarestDataPoints[j][1]){//
1304            TobeAdded = { (double)i, distance, distance1, distance2,
 distance3 };
1305            NEarestDataPoints[j] = TobeAdded;
1306          }
1307        }
1308      }
1309
1310
1311    }
1312    else{
1313
1314      distance1 = ((PointOfInterest[0] - k[0]) / Pnorm); //
PointOfInterest(1);
1315      distance2 = ((PointOfInterest[1] - k[1]) / Tnorm); //
PointOfInterest(2);
1316      distance3 = ((PointOfInterest[2] - k[2]) / Phinorm); // /
PointOfInterest(3);
1317      distance = sqrt(pow(distance1, 2) + pow(distance2, 2) + pow(
distance3, 2));
1318
1319      distanceall[i][0] = i;
1320      distanceall[i][1] = distance;
1321      distanceall[i][2] = distance1;
1322      distanceall[i][3] = distance2;
1323      distanceall[i][4] = distance3;
1324
1325      int j = 0;
1326
1327      if (distance1*distance2*distance3 > 0){ // Group bounds based
on their side
```

```
1328          if (distance1 > 0 && distance2 > 0 && distance3 > 0){// % +
    + +
1329            j = 0;
1330            if (distance < NEarestDataPoints[j][1]){
1331              TobeAdded = { (double)i, distance, distance1, distance2,
     distance3 };
1332              NEarestDataPoints[j] = TobeAdded;
1333            }
1334          }
1335          else if (distance1 < 0 && distance2>0 && distance3 < 0){// %
    - + -
1336            j = 1;
1337            if (distance < NEarestDataPoints[j][1]){
1338              TobeAdded = { (double)i, distance, distance1, distance2,
     distance3 };
1339              NEarestDataPoints[j] = TobeAdded;
1340            }
1341          }
1342          else if (distance1 > 0 && distance2 < 0 && distance3 < 0){//
    % + - -
1343            j = 2;
1344            if (distance < NEarestDataPoints[j][1]){
1345              TobeAdded = { (double)i, distance, distance1, distance2,
     distance3 };
1346              NEarestDataPoints[j] = TobeAdded;
1347            }
1348          }
1349          else{// % - - +
1350            j = 3;
1351            if (distance < NEarestDataPoints[j][1]){//
1352              TobeAdded = { (double)i, distance, distance1, distance2,
     distance3 };
1353              NEarestDataPoints[j] = TobeAdded;
1354            }
1355          }
1356        }
1357        else{
```

```
1358          if (distance1 < 0 && distance2 < 0 && distance3 < 0){// % -
    - -
1359            j = 4;
1360            if (distance < NEarestDataPoints[j][1]){
1361              TobeAdded = { (double)i, distance, distance1, distance2,
     distance3 };
1362              NEarestDataPoints[j] = TobeAdded;
1363            }
1364          }
1365          else if (distance1 > 0 && distance2 < 0 && distance3>0){// %
    + - +
1366            j = 5;
1367            if (distance < NEarestDataPoints[j][1]){
1368              TobeAdded = { (double)i, distance, distance1, distance2,
     distance3 };
1369              NEarestDataPoints[j] = TobeAdded;
1370            }
1371          }
1372          else if (distance1 < 0 && distance2>0 && distance3 > 0){// %
    - + +
1373            j = 6;
1374            if (distance < NEarestDataPoints[j][1]){
1375              TobeAdded = { (double)i, distance, distance1, distance2,
     distance3 };
1376              NEarestDataPoints[j] = TobeAdded;
1377            }
1378          }
1379          else {// % + + -
1380            j = 7;
1381            if (distance < NEarestDataPoints[j][1]){//
1382              TobeAdded = { (double)i, distance, distance1, distance2,
     distance3 };
1383              NEarestDataPoints[j] = TobeAdded;
1384            }
1385          }
1386        }
1387      }
```

```
1388      }
1389    }
1390
1391
1392    NearestDataToExport = NEarestDataPoints;
1393
1394    vector<int> FileNames(8);
1395
1396    int k = 0;
1397
1398    for (int j = 0; j < 8; j++){
1399      if (NearestDataToExport[j][0] != 1E15){
1400        FileNames[k] = (int)NearestDataToExport[j][0];
1401        k++;
1402      }
1403    }
1404    if (k<8) FileNames.resize(k);
1405
1406    return FileNames;
1407
1408 }
1409
1410 bool ReadFileandPassData(string Location, vector<vector<double>> &
       DataVector){
1411    bool HeavyFileReadSuccesfully = false;
1412    // If no string is given as inout
1413    if (Location.empty()){
1414      Location = "AllData.bin";
1415    }
1416
1417
1418
1419    ifstream pFile(Location, ios::in | ios::binary);
1420    string line;
1421    int row, col;
1422    double rowd;
1423    row = 0;
```

```
1424
1425  if (pFile.is_open())
1426  {
1427
1428
1429    // Get the size of the file in bytes
1430    size_t size = 0;
1431    pFile.seekg(0, ios::end);
1432    size = pFile.tellg();
1433    pFile.seekg(0, ios::beg);
1434
1435    // Allocate space in the buffer for the whole file
1436    char* oData = new char[size];
1437    // REed Neccesarry uint8 binary Data
1438    pFile.read(oData, size);
1439    pFile.close();
1440
1441    // binary to int first
1442    memcpy(&rowd, &oData[0], sizeof(double));
1443    row = (int)rowd;
1444    col = (int)size / 8 / row;
1445
1446    DataVector.resize(row);
1447    for (int i = 0; i < row; ++i)
1448      DataVector[i].resize(col);
1449
1450    for (int j = 0; j < col; ++j){
1451      for (int i = 0; i < row; ++i){
1452        memcpy(&rowd, &oData[sizeof(double) * (i + row*j)], sizeof(
     double));
1453        DataVector[i][j] = (double)rowd;
1454      }
1455    }
1456
1457    HeavyFileReadSuccesfully = true;
1458    delete[] oData; //The file content in the memory removed.
1459
```

```
1460   }
1461   //else
1462 
1463     int NnumberofSpecies = (int)DataVector[0][2];
1464     int NumberofSectionbyPhi = (int)DataVector[0][3];
1465     int n_Reaction = (int)DataVector[0][NumberofSectionbyPhi+15+
       NnumberofSpecies*2+2];
1466 //     Cantera::set_n_reaction(n_Reaction);
1467 
1468   return HeavyFileReadSuccesfully;
1469 
1470 }
1471 
1472 extern "C" double**  ReadFileandPassDataArray(char *Location){
1473 
1474   // If no string is given as inout
1475   /*if (Location.empty()){
1476     Location = "AllDataTemp.bin";
1477   }*/
1478 
1479   //cout << "DataLocation: " << Location << "\n";
1480 
1481   BYTE *fileBuf;          // Pointer to our buffered data
1482 
1483   ifstream pFile(Location, ios::in | ios::binary);
1484   string line;
1485   int row, col;
1486   double rowd, cold;
1487   row = 0;
1488   double** DataVector;//= new double*[];
1489 
1490   if (pFile.is_open())
1491   {
1492 
1493     //cout << "File Opened" << endl;
1494     // Get the size of the file in bytes
1495     size_t size = 0;
```

```
1496    pFile.seekg(0, ios::end);
1497    size = pFile.tellg();
1498    pFile.seekg(0, ios::beg);
1499
1500    // Allocate space in the buffer for the whole file
1501    char* oData = new char[size];
1502    // REed Neccesarry uint8 binary Data
1503    pFile.read(oData, size);
1504    pFile.close();
1505
1506    // binary to int first
1507    memcpy(&rowd, &oData[0], sizeof(double));
1508    row = (int)rowd;
1509    //row = 10; //RemoveThisat actual case
1510
1511    col =size / 8 / row;
1512
1513      DataVector = new double*[row];
1514      for (int i = 0; i < row; ++i)
1515        DataVector[i] = new double[col];
1516
1517
1518      for (int j = 0; j < col; ++j){
1519        for (int i = 0; i < row; ++i){
1520          memcpy(&rowd, &oData[sizeof(double) * (i + row*j)], sizeof(
     double));
1521          DataVector[i][j] = (double)rowd;
1522        }
1523      }
1524
1525      delete[] oData;
1526
1527
1528  }
1529  //else
1530
1531
```

```
1532     int NnumberofSpecies = (int)DataVector[0][2];
1533     int NumberofSectionbyPhi = (int)DataVector[0][3];
1534     int n_Reaction = (int)DataVector[0][NumberofSectionbyPhi+15+
       NnumberofSpecies*2+2];
1535     //Cantera::set_n_reaction(n_Reaction);
1536 
1537   return DataVector;
1538 }
1539 
1540 void WriteOut(char* DatatoWrite, int Level){
1541   if(Level==1){
1542     printf(DatatoWrite);}
1543   //Cantera::writelogFile(DatatoWrite,Level);
1544 }
1545 
1546 
```

APPENDIX B: Cantera Compilation File

```
python_package = 'none'
matlab_toolbox = 'y'
matlab_path = 'C:\\Program Files\\MATLAB\\R2016b'
f90_interface = 'n'
use_sundials = 'n'
extra_lib_dirs = 'C:\\Program Files\\MATLAB\\R2016b\\extern\\lib\\win32'
```

APPENDIX C: Fluent User Defined Subroutine in C

The C script that is compiled and used.

```
/*
DEFINE_NET_REACTION_RATE : homogeneous net mass reaction rate for all species

There are nine arguments to DEFINE_NET_REACTION_RATE: name, c, t, particle, pressure,
temp, yi, rr, and jac. You supply name, the name of the UDF. The variables c, t, particle,
pressure, temp, yi, rr, and jac are passed by the ANSYS Fluent solver to your UDF and have SI
units. The outputs of the function are the array of net molar reaction rates, rr (with units kmol/m3-s
for volumetric reactions and kmol/m2-s for the surface reactions

Note that during the course of the ODE solution, the species mass fractions can exceed realizable
bounds. For optimal ODE performance, the species mass fractions should not be clipped, but derived
quantities, such as concentrations which are raised to non-integer powers, must be bounded. Also, if
density is required, for instance to calculate concentrations, it should be calculated from the temperature
and species passed into the UDF. Finally, double precision should be used for all local variables.

 ...
 050: Overall corrections and reading the phi limits and autoignition temperature from input file.
 051: Some corrections
 052: UDS intoruduced for checking eligibility for reaction.
 053: IsReacting_v2 applied to all terms instead of heavy only.
 054: Minimum phi limit for reactions removed since it is over the phi
```

```
    values used at second range. Instead CH4 fraction limit
    introduced.
 055: Ignition Phi limit introduced.
 058: Temperature limit and mass fraction for reactions to proceed
    updated.
 059: Introduced minimum fuel limit for reaction to start.
 061: Minimum and maximum fuel mass fraction limits
 062: Correction in isreacting term and limits. Added energy
    multiplier term.
 063: Changed maximum temp limit calculation
 064: Introduced species upper flamability for reaction to start.
 065: Corrected incorrect is reacting term calculation during
    initialization.
 067: All reactions ended as heavy reactions end.
 068: Removed unnecessarry script
 069: Minor corrections\clean-up.

 */
#include "udf.h"
 /*#include <cstdio>
 #include <stdarg.h>
 #include "KourdisCodes.h" */

#define EXP_LIM(x) exp( MAX( MIN( x, 60. ), -60. ) )
#define NumOfSpecies 26 /* Actual number including dmy */
#define MAXT_Mult 1.0001


static double *cnu = NULL, *mw = NULL, *therm_poly = NULL, *
    therm_poly_pl = NULL;
static int n_spe_isat = 0;
static const int therm_stride = 17;
static int therm_stride_pl = 0;
static int poly_type[MAX_PDF_SPECIES] = { 0 };
static int max_species = 0;
double **DataRead;
double **DataRead1;
```

```
53 double **DataRead2;
54 double* TBDGlobbal;
55 double System_MaxP, System_MinP;
56 int RunforOnce = 1;
57 int MaterialPropertyMethod = -10000;
58 int RunningHeavy = 0;
59 float OxygenFuelRatio_Stoi = 0;
60 int DebugOption = 0;
61 float IDV_WeigthingFactor = 0;
62 float PhiLimit = 0;
63 float IDV_TemperatureSearchRange = 0;
64 float LigthEnergyLimit1 = 0;
65 float LigthEnergyLimit2 = 0;
66 int MaxReached = 0;
67 int HeavyFileReadSuccesfully = FALSE;
68 int IsReacting = FALSE;
69 int IsTableEncapsulating = FALSE;
70 int ExtrapolateSystemPressure = FALSE;
71 float Upper_Phi_Limit, End_Phi_Limit, Lower_Phi_Limit,
    isreacting_temp_auto, EnenrgyMult;
72 double Table_MinPhi, Table_MaxPhi, Table_MinTemp, Table_MinP,
    Table_MaxP;
73 float MinimumFuelLimitforReactinToEnd,
    MinimumFuelLimitforReactinToBegin;
74
75
76 /* ------------ USER INPUTS */
77 const char *BaseFolder = "";/*"D:\\Dropbox\\MATLAB\\";*/
78 /* These below limits are based on tabular data generated so it should
     be added to the data and retrieved as neccesarry. especially max
    values  */
79 double MaxPhi, MaxT, MinT, Pmax, Pmin, PhiMax;
80
81 extern double** ReadFileandPassDataArray(char *Location);
82 extern int InterpolatebyInverseDistanceWeigthed37_2_wlhn_re(double Pi,
     double Ti, double Phi_i, int MaxReached, double* output_args,
    double* AdditionalConstants, double** DataRead, double*
```

```
    additional_outputs); /* //prev 16_2 */
83
84  #define EVALUATE_CP_POLY(j,T) (therm_poly[j+3] + T*(therm_poly[j+4] +
    \
85          T*(therm_poly[j+5] + T*(therm_poly[j+6] + T*therm_poly[j+7])))
    )
86
87  #define EVALUATE_H_POLY(j,T) (therm_poly[j+8] + T*(therm_poly[j+3] + \
88          T*(therm_poly[j+4]/2. + T*(therm_poly[j+5]/3. + T*(therm_poly[
    j+6]/4. + \
89          T*therm_poly[j+7]/5.)))))
90
91  #define EVALUATE_S_POLY(j,T,ln_T) (therm_poly[j+9] + therm_poly[j+3]*
    ln_T + \
92          T*(therm_poly[j+4] + T*(therm_poly[j+5]/2. + T*(therm_poly[j
    +6]/3. + \
93          T*therm_poly[j+7]/4.))))
94
95  static void fill_thermo_props(Material *m, Material *sp, int i)
96  {
97    int j, k;
98    double x = 0.;
99    Polynomial *p = MATERIAL_POLYNOMIAL(m, PROP_Cp);
100   double rtemp = MAX(1., (double)MATERIAL_PROP(sp, PROP_reference_temp
      ));
101   MaterialPropertyMethod = MATERIAL_PROP_METHOD(m, PROP_Cp);
102
103   if (((MATERIAL_PROP_METHOD(m, PROP_Cp) == PROP_METHOD_POLYNOMIAL) &&
104     (p->type == PIECEWISE_POLYNOMIAL) && (PPY_NRANGES(p) == 2) &&
105     (PPY_NCOEFFS(p, 0) == 5) && (PPY_NCOEFFS(p, 1) == 5))) {
106     poly_type[i] = 1;
107     j = therm_stride*i;
108     therm_poly[j] = (double)PPY_XMIN(p, 0);
109     therm_poly[j + 1] = (double)PPY_XMAX(p, 0);
110     therm_poly[j + 2] = (double)PPY_XMAX(p, 1);
111     for (k = 0; k < PPY_NCOEFFS(p, 0); k++)
112       therm_poly[j + 3 + k] = (double)PPY_COEFF(p, 0, k);
```

```
113     for (k = 0; k < PPY_NCOEFFS(p, 1); k++)
114       therm_poly[j + 10 + k] = (double)PPY_COEFF(p, 1, k);
115
116     /* calculate cp coeff 0 for upper range by enforcing continuity */
117     x = therm_poly[j + 1];
118     therm_poly[j + 10] = EVALUATE_CP_POLY(j, x) -
119       x * (therm_poly[j + 11] +
120         x * (therm_poly[j + 12] +
121           x * (therm_poly[j + 13] +
122             x * therm_poly[j + 14])));
123
124     /* calculate enthalpy coeff 0 for lower range */
125     x = rtemp;
126     therm_poly[j + 8] = (double)MATERIAL_PROP(sp, PROP_hform) -
127       x * (therm_poly[j + 3] +
128         x * (therm_poly[j + 4] / 2. +
129           x * (therm_poly[j + 5] / 3. +
130             x * (therm_poly[j + 6] / 4. +
131               x * therm_poly[j + 7] / 5.))));
132
133     /* calculate enthalpy coeff 0 for upper range by enforcing
    continuity */
134     x = therm_poly[j + 1];
135     therm_poly[j + 15] = EVALUATE_H_POLY(j, x) -
136       x * (therm_poly[j + 10] +
137         x * (therm_poly[j + 11] / 2. +
138           x * (therm_poly[j + 12] / 3. +
139             x * (therm_poly[j + 13] / 4. +
140               x * therm_poly[j + 14] / 5.))));
141
142     /* calculate entropy coeff 0 for lower range */
143     x = rtemp;
144     therm_poly[j + 9] = (double)MATERIAL_PROP(sp, PROP_sform) -
145       (therm_poly[j + 3] * log(x) +
146         x * (therm_poly[j + 4] +
147           x * (therm_poly[j + 5] / 2. +
148             x * (therm_poly[j + 6] / 3. +
```

```
149               x * therm_poly[j + 7] / 4.))));
150
151     /* calculate entropy coeff 0 for upper range by enforcing
    continuity */
152     x = therm_poly[j + 1];
153     therm_poly[j + 16] = EVALUATE_S_POLY(j, x, log(x)) -
154       (therm_poly[j + 10] * log(x) +
155         x * (therm_poly[j + 11] +
156           x * (therm_poly[j + 12] / 2. +
157             x * (therm_poly[j + 13] / 3. +
158               x * therm_poly[j + 14] / 4.))));
159   }
160   else if (MATERIAL_PROP_METHOD(m, PROP_Cp) == PROP_METHOD_CONSTANT) {
161     j = therm_stride * i;
162     poly_type[i] = 2;
163     therm_poly[j] = (double)PROPERTY_CONSTANT(MATERIAL_PROPERTY(m),
    PROP_Cp);
164     therm_poly[j + 1] = (double)MATERIAL_PROP(sp, PROP_hform) -
165       rtemp * therm_poly[j];
166     therm_poly[j + 2] = (double)MATERIAL_PROP(sp, PROP_sform) -
167       therm_poly[j] * log(rtemp);
168   }
169   else if ((MATERIAL_PROP_METHOD(m, PROP_Cp) == PROP_METHOD_POLYNOMIAL
    ) &&
170     (p->type == POLYNOMIAL)) {
171     int nc = PY_NCOEFFS(p);
172     double y;
173     j = therm_stride*i;
174     if (nc > therm_stride - 3)
175       Error("Polynomial for %s limited to a maximum of %i co-
    efficients\n",
176         MIXTURE_SPECIE_NAME(m, i), therm_stride - 3);
177     else if (nc < 2)
178       Error("Polynomial for %s must have more than 1 co-efficient\n",
179         MIXTURE_SPECIE_NAME(m, i));
180     poly_type[i] = -nc;
181     for (k = 0; k < nc; k++)
```

```
182       therm_poly[j + k] = (double)PY_COEFF(p, k);
183     x = therm_poly[j + nc - 1] / (double)(nc);
184     y = therm_poly[j + nc - 1] / (double)(nc - 1);
185     for (k = nc - 2; k >= 0; --k) {
186       x = therm_poly[j + k] / (double)(k + 1) + x*rtemp;
187       if (k > 0)
188         y = therm_poly[j + k] / (double)(k)+y * rtemp;
189     }
190     x *= rtemp;
191     y *= rtemp;
192     y += therm_poly[j] * log(rtemp);
193     therm_poly[j + nc] = (double)MATERIAL_PROP(sp, PROP_hform) - x;
194     therm_poly[j + nc + 1] = (double)MATERIAL_PROP(sp, PROP_sform) - y
    ;
195   }
196   else if ((MATERIAL_PROP_METHOD(m, PROP_Cp) == PROP_METHOD_POLYNOMIAL
    ) &&
197     (p->type == PIECEWISE_LINEAR)) {
198     int j0, j1, j2, j3, nc = PL_NRANGES(p);
199     double y = 0.;
200     poly_type[i] = 3;
201     j0 = therm_stride_pl * i + 1;
202     j1 = j0 + nc;
203     j2 = j1 + nc;
204     j3 = j2 + nc;
205     therm_poly_pl[j0 - 1] = (double)nc;
206     for (k = 0; k < nc; k++) {
207       therm_poly_pl[j0 + k] = (double)PL_XMIN(p, k); /* T  */
208       therm_poly_pl[j1 + k] = (double)PL_YMIN(p, k); /* cp */
209     }
210     therm_poly_pl[j2] = therm_poly_pl[j1] * therm_poly_pl[j0];
211     therm_poly_pl[j3] = therm_poly_pl[j1] * log(therm_poly_pl[j0]);
212     for (k = 1; k < nc; k++) {
213       double T0 = therm_poly_pl[j0 + k - 1];
214       double T1 = therm_poly_pl[j0 + k];
215       double cp0 = therm_poly_pl[j1 + k - 1];
216       double cp1 = therm_poly_pl[j1 + k];
```

```
217       double cxi = (cp1 - cp0) / (T1 - T0);
218       therm_poly_pl[j2 + k] = therm_poly_pl[j2 + k - 1] +
219         0.5 * (cp0 + cp1)*(T1 - T0); /* h  */
220
221       therm_poly_pl[j3 + k] = therm_poly_pl[j3 + k - 1] +
222         (cp0 - cxi * T0) * log(T1 / T0) + cxi * (T1 - T0); /* s */
223     }
224     if (rtemp <= therm_poly_pl[j0]) {
225       x = therm_poly_pl[j1] * rtemp;
226       y = therm_poly_pl[j1] * log(rtemp);
227     }
228     else if (rtemp >= therm_poly_pl[j0 + nc - 1]) {
229       x = therm_poly_pl[j2 + nc - 1] + therm_poly_pl[j1 + nc - 1] * (
    rtemp - therm_poly_pl[j0 + nc - 1]);
230       y = therm_poly_pl[j3 + nc - 1] + therm_poly_pl[j1 + nc - 1] *
    log(rtemp / therm_poly_pl[j0 + nc - 1]);
231     }
232     else
233       for (k = 1; k < nc; k++)
234         if (rtemp <= therm_poly_pl[j0 + k]) {
235           double T0 = therm_poly_pl[j0 + k - 1];
236           double T1 = therm_poly_pl[j0 + k];
237           double cp0 = therm_poly_pl[j1 + k - 1];
238           double cp1 = therm_poly_pl[j1 + k];
239           double xi = (rtemp - T0) / (T1 - T0);
240           double cxi = (cp1 - cp0) / (T1 - T0);
241           double cpi = cp0 * (1. - xi) + cp1*xi;
242           x = therm_poly_pl[j2 + k - 1] + 0.5 * (cpi + cp0) * (rtemp -
    T0);
243           y = therm_poly_pl[j3 + k - 1] + (cp0 - cxi * T0) * log(rtemp
    / T0) + cxi * (rtemp - T0);
244           break;
245         }
246
247     for (k = 0; k < nc; k++) {
248       therm_poly_pl[j2 + k] += (double)MATERIAL_PROP(sp, PROP_hform) -
    x;
```

```
249       therm_poly_pl[j3 + k] += (double)MATERIAL_PROP(sp, PROP_sform) -
      y;
250       }
251     }
252     else
253       poly_type[i] = 0;
254 }
255
256 static void fill_cache_variables(void)
257 {
258   Material *m = mixture_material(Get_Domain(1)), *sp;
259   int i, j, k;
260
261   n_spe_isat = m->n_components;
262
263   if (NULLP(mw)) {
264     mw = (double *)(malloc(n_spe_isat * sizeof(double)));
265     mixture_species_loop(m, sp, i)
266       mw[i] = (double)MATERIAL_PROP(sp, PROP_mwi);
267   }
268
269   if (NULLP(therm_poly)) {
270     therm_poly = (double *)(malloc(n_spe_isat * therm_stride * sizeof(
    double)));
271
272     for (i = 0; i < n_spe_isat * therm_stride; i++)
273       therm_poly[i] = 0.;
274
275     if (MATERIAL_PROP_METHOD(m, PROP_Cp) == CP_MIXTURE ||
    MATERIAL_PROP_METHOD(m, PROP_Cp) == PROP_METHOD_UDF) {
276
277       mixture_species_loop(m, sp, i) {
278         Polynomial *p = MATERIAL_POLYNOMIAL(sp, PROP_Cp);
279         if ((MATERIAL_PROP_METHOD(sp, PROP_Cp) ==
    PROP_METHOD_POLYNOMIAL) &&
280           (p->type == PIECEWISE_LINEAR))
281           therm_stride_pl = MAX(therm_stride_pl, PL_NRANGES(p));
```

```
282       }
283
284       if (therm_stride_pl > 0) {
285         therm_stride_pl = therm_stride_pl * 4 + 1;
286         therm_poly_pl = (double *)(malloc(n_spe_isat * therm_stride_pl
    * sizeof(double)));
287       }
288
289       mixture_species_loop(m, sp, i) {
290         fill_thermo_props(sp, sp, i);
291       }
292     }
293     else if (MATERIAL_PROP_METHOD(m, PROP_Cp) == PROP_METHOD_CONSTANT
    ||
294       MATERIAL_PROP_METHOD(m, PROP_Cp) == PROP_METHOD_POLYNOMIAL) {
295       Polynomial *p = MATERIAL_POLYNOMIAL(m, PROP_Cp);
296       if ((MATERIAL_PROP_METHOD(m, PROP_Cp) == PROP_METHOD_POLYNOMIAL)
    &&
297           (p->type == PIECEWISE_LINEAR))
298         therm_stride_pl = MAX(therm_stride_pl, PL_NRANGES(p));
299
300       if (therm_stride_pl > 0) {
301         therm_stride_pl = therm_stride_pl * 4 + 1;
302         therm_poly_pl = (double *)(malloc(n_spe_isat * therm_stride_pl
    * sizeof(double)));
303       }
304
305       mixture_species_loop(m, sp, i) {
306         fill_thermo_props(m, sp, i);
307       }
308     }
309   }
310
311   if (NULLP(cnu)) {
312     Reaction * const reaction_list = m->reaction_list, *r;
313
314     j = 0;
```

```
315     max_species = 0;
316
317     loop(r, reaction_list) {
318       max_species = MAX(max_species, r->n_reactants);
319       max_species = MAX(max_species, r->n_products);
320       j++;
321     }
322
323     cnu = (double *)(malloc((2 * max_species + 1) * MAX(j, 1) * sizeof
    (double)));
324
325     j = 0;
326
327     loop(r, reaction_list) {
328       int n_r = r->n_reactants;
329       int n_p = r->n_products;
330       real *stoich_reactant = r->stoich_reactant;
331       real *stoich_product = r->stoich_product;
332       int ji = j * (2 * max_species + 1);
333
334       for (i = 0; i < max_species; ++i)
335         cnu[i + ji] = cnu[i + max_species + ji] = 0.;
336
337       for (i = 0; i < n_r; ++i)
338         cnu[i + ji] = -(double)stoich_reactant[i];
339
340       for (i = 0; i < n_p; ++i)
341         cnu[i + max_species + ji] = (double)stoich_product[i];
342
343       cnu[max_species * 2 + ji] = 0.;
344       for (i = 0; i < max_species; ++i)
345         cnu[max_species * 2 + ji] += (cnu[i + ji] + cnu[i +
   max_species + ji]);
346
347       j++;
348     }
349   }
```

```
350 }
351
352 static void calc_thermo_props(double T, double ln_T, double *yi,
      double *cpi, double *hi, double *si)
353 {
354   /* note this function returns ideal gas values at 1 atm for all
      density models */
355   Material * const m = mixture_material(Get_Domain(1)), *sp;
356   int i, j, k;
357   double Tmin, Tmax, Tmid, sumY = 0;
358   if (MATERIAL_PROP_METHOD(m, PROP_Cp) == CP_MIXTURE ||
359     MATERIAL_PROP_METHOD(m, PROP_Cp) == PROP_METHOD_CONSTANT ||
360     MATERIAL_PROP_METHOD(m, PROP_Cp) == PROP_METHOD_POLYNOMIAL ||
361     MATERIAL_PROP_METHOD(m, PROP_Cp) == PROP_METHOD_UDF) {
362     cpi[n_spe_isat] = 0.;
363     cpi[n_spe_isat + 1] = 0.;
364
365
366     spe_loop(i, n_spe_isat) {
367       if (poly_type[i] == 1) {
368         /* 2 piece-wise polynomials (NASA) */
369         j = therm_stride*i;
370         Tmin = therm_poly[j];
371         Tmid = therm_poly[j + 1];
372         Tmax = therm_poly[j + 2];
373         if (T > Tmid)
374           j += 7;
375         if (T < Tmin) {
376           cpi[i] = EVALUATE_CP_POLY(j, Tmin);
377           hi[i] = EVALUATE_H_POLY(j, Tmin) + cpi[i] * (T - Tmin);
378           si[i] = EVALUATE_S_POLY(j, Tmin, log(Tmin)) + cpi[i] * log(T
      / Tmin);
379         }
380         else if (T > Tmax) {
381           cpi[i] = EVALUATE_CP_POLY(j, Tmax);
382           hi[i] = EVALUATE_H_POLY(j, Tmax) + cpi[i] * (T - Tmax);
383           si[i] = EVALUATE_S_POLY(j, Tmax, log(Tmax)) + cpi[i] * log(T
```

```
 / Tmax);
384       }
385       else {
386         cpi[i] = EVALUATE_CP_POLY(j, T);
387         hi[i] = EVALUATE_H_POLY(j, T);
388         si[i] = EVALUATE_S_POLY(j, T, ln_T);
389       }
390     }
391     else if (poly_type[i] == 2) {
392       /* constant */
393       j = therm_stride*i;
394       cpi[i] = therm_poly[j];
395       hi[i] = therm_poly[j + 1] + T * therm_poly[j];
396       si[i] = therm_poly[j + 2] + ln_T * therm_poly[j];
397     }
398     else if (poly_type[i] == 3) {
399       /* piecewise linear */
400       int j0, j1, j2, j3, nc;
401       j0 = therm_stride_pl * i + 1;
402       nc = (int)(therm_poly_pl[j0 - 1] + 0.5);
403       j1 = j0 + nc;
404       j2 = j1 + nc;
405       j3 = j2 + nc;
406       if (T <= therm_poly_pl[j0]) {
407         cpi[i] = therm_poly_pl[j1];
408         hi[i] = therm_poly_pl[j2] + (T - therm_poly_pl[j0]) * cpi[i
];
409         si[i] = therm_poly_pl[j3] + (log(T / therm_poly_pl[j0])) *
cpi[i];
410       }
411       else if (T >= therm_poly_pl[j0 + nc - 1]) {
412         cpi[i] = therm_poly_pl[j1 + nc - 1];
413         hi[i] = therm_poly_pl[j2 + nc - 1] + (T - therm_poly_pl[j0 +
 nc - 1]) * cpi[i];
414         si[i] = therm_poly_pl[j3 + nc - 1] + (log(T / therm_poly_pl[
j0 + nc - 1])) * cpi[i];
415       }
```

```
416         else
417           for (k = 1; k < nc; k++)
418             if (T <= therm_poly_pl[j0 + k]) {
419               double T0 = therm_poly_pl[j0 + k - 1];
420               double T1 = therm_poly_pl[j0 + k];
421               double cp0 = therm_poly_pl[j1 + k - 1];
422               double cp1 = therm_poly_pl[j1 + k];
423               double xi = (T - T0) / (T1 - T0);
424               double cxi = (cp1 - cp0) / (T1 - T0);
425               cpi[i] = cp0 * (1. - xi) + cp1*xi;
426               hi[i] = therm_poly_pl[j2 + k - 1] + 0.5 * (cpi[i] + cp0)
    * (T - T0);
427               si[i] = therm_poly_pl[j3 + k - 1] + (cp0 - cxi * T0) *
  log(T / T0) + cxi * (T - T0);
428               break;
429             }
430       }
431       else if (poly_type[i] < 0) {
432         /* polynomial */
433         int nc = -poly_type[i];
434         j = therm_stride*i;
435         cpi[i] = therm_poly[j + nc - 1];
436         hi[i] = therm_poly[j + nc - 1] / (double)(nc);
437         si[i] = therm_poly[j + nc - 1] / (double)(nc - 1);
438         for (k = nc - 2; k >= 0; --k) {
439           cpi[i] = therm_poly[j + k] + cpi[i] * T;
440           hi[i] = therm_poly[j + k] / (double)(k + 1) + hi[i] * T;
441           if (k > 0)
442             si[i] = therm_poly[j + k] / (double)(k)+si[i] * T;
443         }
444         hi[i] *= T;
445         si[i] *= T;
446         hi[i] += therm_poly[j + nc];
447         si[i] += therm_poly[j + nc + 1] + therm_poly[j] * ln_T;
448       }
449       else {
450         real yi_s[MAX_PDF_SPECIES], hi_s[MAX_PDF_SPECIES], tr;
```

```
451         int rp_energy_hold = rp_energy;
452         spe_loop(k, n_spe_isat)
453           yi_s[k] = (real)yi[k];
454         tr = (real)T;
455         rp_energy = TRUE;
456         (void)Enthalpy(m, tr, 0., 0., yi_s, hi_s);
457         rp_energy = rp_energy_hold;
458         sp = MIXTURE_SPECIE(m, i);
459         {
460           real s0 = MATERIAL_PROP(sp, PROP_sform);
461           real Tref = MATERIAL_PROP(sp, PROP_reference_temp);
462           hi[i] = (double)hi_s[i];
463           si[i] = (double)(Specific_Heat_by_T_Integral(sp, Tref, tr,
 0., yi_s) + s0);
464           cpi[i] = (double)Specific_Heat(sp, tr, 0., 0., yi_s);
465         }
466       }
467       cpi[n_spe_isat] += cpi[i] * yi[i];
468       if (i < n_spe_isat - 1) {
469         sumY += yi[i];
470         cpi[n_spe_isat + 1] += cpi[i] * yi[i];
471       }
472
473
474
475     }
476   }
477   else if (MATERIAL_PROP_METHOD(m, PROP_Cp) ==
 PROP_METHOD_REAL_GAS_MIXTURE) {
478     Error("no real gas");
479   }
480   else {
481     real yi_s[MAX_PDF_SPECIES], hi_s[MAX_PDF_SPECIES], tr;
482     int rp_energy_hold = rp_energy;
483     spe_loop(k, n_spe_isat)
484       yi_s[k] = (real)yi[k];
485     tr = (real)T;
```

```
486     rp_energy = TRUE;
487     (void)Enthalpy(m, tr, 0., 0., yi_s, hi_s);
488     rp_energy = rp_energy_hold;
489     cpi[n_spe_isat] = 0.;
490     cpi[n_spe_isat + 1] = 0.;
491
492     mixture_species_loop(m, sp, i) {
493       real s0 = MATERIAL_PROP(sp, PROP_sform);
494       real Tref = MATERIAL_PROP(sp, PROP_reference_temp);
495       hi[i] = (double)hi_s[i];
496       si[i] = (double)(Specific_Heat_by_T_Integral(sp, Tref, tr, 0.,
    yi_s) + s0);
497       cpi[i] = (double)Specific_Heat(sp, tr, 0., 0., yi_s);
498       cpi[n_spe_isat] += cpi[i] * yi[i];
499       if (i < n_spe_isat - 1) {
500         sumY += yi[i];
501         cpi[n_spe_isat + 1] += cpi[i] * yi[i];
502       }
503     }
504   }
505   cpi[n_spe_isat + 1] = cpi[n_spe_isat + 1] / sumY;
506 }
507
508
509 double CalculateRRforSpeciesTransport(Thread *t, cell_t c, double *
    press, double *temp, double *yi, double *wdot)
510 {
511   int i, j, check1;
512   double w_mean = 0., rho, ln_T = log(*temp), Cmin, tsum = 0.;
513   double Ru = UNIVERSAL_GAS_CONSTANT, RTi = 1. / (Ru * (*temp)), Rgas
     = 0.;
514   double ci[MAX_PDF_SPECIES], z[MAX_PDF_SPECIES];
515   double hi[MAX_PDF_SPECIES], cpi[MAX_PDF_SPECIES + 2], si[
    MAX_PDF_SPECIES];
516   Material *m = mixture_material(Get_Domain(1));
517   Reaction *reaction_list, *r;
518
```

```
519  reaction_list = m->reaction_list;
520
521  if (NULLP(cnu))
522    fill_cache_variables();
523
524  Rgas = 0.;
525
526
527  spe_loop(i, n_spe_isat) {
528    z[i] = yi[i] / mw[i];
529    Rgas += z[i];
530  }
531
532  Rgas *= Ru;
533  rho = *press / (Rgas * (*temp));
534
535
536
537  spe_loop(i, n_spe_isat)
538    ci[i] = rho * z[i];
539
540  calc_thermo_props(*temp, ln_T, yi, cpi, hi, si);
541
542  spe_loop(i, n_spe_isat) {
543    hi[i] *= mw[i]; /* //is this correct? hi should be J/kmol-K, this
     term is effective only on backward reactions... */
544    si[i] *= mw[i];
545  }
546
547  j = 0;
548  check1 = 0;
549
550  loop(r, reaction_list) {
551    double sum_H = 0., sum_S = 0., sum_nu, Kf, Kb = 0., cm, rf, rb =
     0., rr;
552    real *exp_reactant = r->exp_reactant;
553    real *exp_product = r->exp_product;
```

```
554     int n_r = r->n_reactants;
555     int n_p = r->n_products;
556     int *reactant = r->reactant;
557     int *product = r->product;
558     int backward_reaction = r->backward_reaction;
559     int ji = j * (2 * max_species + 1), ni;
560
561     /*
562     if(check1<5) {
563       C_UDMI(c, t, 9+check1) = r->Cmin;
564       check1++;
565     }*/
566
567     Kf = EXP_LIM(r->logA + r->b * ln_T - r->E * RTi);
568
569     if (backward_reaction) {
570       const double patm = 101325.;
571       double hold, eqk;
572       for (i = 0; i < n_r; i++) {
573         ni = reactant[i];
574         sum_H += cnu[i + ji] * hi[ni];
575         sum_S += cnu[i + ji] * si[ni];
576       }
577       for (i = 0; i < n_p; i++) {
578         ni = product[i];
579         sum_H += cnu[i + max_species + ji] * hi[ni];
580         sum_S += cnu[i + max_species + ji] * si[ni];
581       }
582       sum_nu = cnu[max_species * 2 + ji];
583       eqk = EXP_LIM(sum_S / Ru - sum_H * RTi);
584       hold = patm*RTi;
585       if (sum_nu == 0.) {
586       }
587       else if (ABS(sum_nu + 1.) < SMALL)
588         eqk /= hold;
589       else if (ABS(sum_nu - 1.) < SMALL)
590         eqk *= hold;
```

```
591       else if (ABS(sum_nu - 2.) < SMALL)
592         eqk *= SQR(hold);
593       else if (ABS(sum_nu + 2.) < SMALL)
594         eqk /= SQR(hold);
595       else
596         eqk *= pow(hold, sum_nu);
597       Kb = Kf / MAX(eqk, SMALL);
598     }
599
600     /* third body efficiencies */
601     cm = 1.;
602     if (r->use_third_body_efficiencies) {
603       cm = 0.;
604       for (i = 0; i < n_r; i++)
605         if ((ni = reactant[i]) < n_spe_isat)
606           cm += ci[ni] * (double)r->eff_reactant[i];
607       for (i = 0; i < n_p; i++)
608         if ((ni = product[i]) < n_spe_isat)
609           cm += ci[ni] * (double)r->eff_product[i];
610       for (i = 0; i < r->n_others; i++)
611         if ((ni = r->other[i]) < n_spe_isat)
612           cm += ci[ni] * (double)r->eff_other[i];
613     }
614
615     if (r->press_react) {
616       double klow, pr, pcor;
617       klow = EXP_LIM((double)r->press_params[0] +
618         (double)r->press_params[1] * ln_T -
619         (double)r->press_params[2] * RTi);
620       if (r->bath_gas == 0) {
621         pr = klow / Kf*cm;
622         cm = 1.;
623       }
624       else
625         pr = klow / Kf * ci[r->bath_gas];
626       pcor = pr / (1. + pr);
627       if (r->press_type == 2) {
```

```
628         double fcent, apr, af, log_fcent;
629         double alp = (double)r->press_params[3];
630         fcent = (1. - alp) * EXP_LIM(-(*temp) / (double)r->
    press_params[4]) +
631           alp * EXP_LIM(-(*temp) / (double)r->press_params[5]);
632         if (ABS(r->press_params[6] + 1.) > SMALL)
633           fcent += EXP_LIM(-(double)r->press_params[6] / (*temp));
634         log_fcent = log10(MAX(fcent, 1.e-100));
635         apr = log10(MAX(pr, 1.e-100)) - 0.4 - 0.67 * log_fcent;
636         af = apr / (0.75 - 1.27 * log_fcent - 0.14 * apr);
637         af = log_fcent / (1. + SQR(af));
638         pcor *= pow(10., af);
639       }
640       else if (r->press_type == 3) {
641         double ax, apr, af;
642         apr = log10(MAX(pr, 1.e-100));
643         ax = 1. / (1. + SQR(apr));
644         af = (double)r->press_params[6] *
645           pow((double)r->press_params[3] * EXP_LIM(-(double)r->
    press_params[4] / (*temp)) +
646             EXP_LIM(-(*temp) / (double)r->press_params[5]), ax) *
647           pow((*temp), (double)r->press_params[7]);
648         pcor *= af;
649       }
650       Kf *= pcor;
651       Kb *= pcor;
652     }
653
654     rf = Kf*cm;
655     for (i = 0; i < n_r; i++) {
656       ni = reactant[i];
657       if ((double)exp_reactant[i] < 0.) {
658         double exp_r = (double)exp_reactant[i];
659         ci[ni] = MAX(SMALL_S, ci[ni]);
660         rf *= pow(ci[ni], exp_r);
661         if (ci[ni] < r->Cmin) {
662           double fac = r->Cmin / ci[ni];
```

```
663           fac = (2. - exp_r) * pow(fac, (exp_r - 1.)) +
664             (exp_r - 1.) * pow(fac, (exp_r - 2.));
665           rf *= fac;
666         }
667       }
668       else {
669         rf *= pow(MAX(ci[ni], 0.), (double)exp_reactant[i]);
670       }
671     }
672
673     if (backward_reaction) {
674       rb = Kb*cm;
675       for (i = 0; i < n_p; i++) {
676         ni = product[i];
677         if (cnu[i + max_species + ji] < 0.) {
678           double exp_p = cnu[i + max_species + ji];
679           ci[ni] = MAX(SMALL_S, ci[ni]);
680           rb *= pow(ci[ni], exp_p);
681           if (ci[ni] < r->Cmin) {
682             /* Cmin 1E-6 */
683             double fac = r->Cmin / ci[ni];
684             fac = (2. - exp_p) * pow(fac, (exp_p - 1.)) +
685               (exp_p - 1.) * pow(fac, (exp_p - 2.));
686             rb *= fac;
687           }
688         }
689         else {
690           rb *= pow(MAX(ci[ni], 0.), cnu[i + max_species + ji]);
691         }
692       }
693     }
694
695     else {
696       for (i = 0; i < n_p; i++)
697         if ((ni = product[i]) < n_spe_isat) {
698           double exp_p = (double)exp_product[i];
699           if (exp_p < 0.) {
```

```
700             ci[ni] = MAX(SMALL_S, ci[ni]);
701             rf *= pow(ci[ni], exp_p);
702             if (ci[ni] < r->Cmin) {
703               double fac = r->Cmin / ci[ni];
704               fac = (2. - exp_p) * pow(fac, (exp_p - 1.)) +
705                 (exp_p - 1.) * pow(fac, (exp_p - 2.));
706               rf *= fac;
707             }
708           }
709           else
710             rf *= pow(MAX(ci[ni], 0.), exp_p);
711         }
712     }
713
714     rr = rf - rb;
715
716     /* //This below term is supposed to be molar reaction rate kmol/m
    ^3-s */
717     for (i = 0; i < n_r; i++)
718       wdot[reactant[i]] += cnu[i + ji] * rr;
719     for (i = 0; i < n_p; i++)
720       wdot[product[i]] += cnu[i + max_species + ji] * rr;
721
722     j++;
723   }
724
725   return rho;
726 }
727
728 DEFINE_NET_REACTION_RATE(def_net_RR_wHeavy, c, t, particle, press,
    temp, yi, wdot, jac)
729 {
730
731 #if RP_NODE
732
733   int i;
734   double rho;
```

```
735  /* //int NumOfSpecies1; */
736
737  rho = CalculateRRforSpeciesTransport(t, c, press, temp, yi, wdot);
738
739  C_UDMI(c, t, NumOfSpecies + 15) = rho;
740
741
742
743  double Sum_dYdt_All = 0;
744  int HasCellReactedBefore = C_UDMI(c, t, NumOfSpecies + 12);
745  spe_loop(i, n_spe_isat) {
746    if (i < n_spe_isat - 1) { /* // n_spe_isat =18 but loops from 0 to
     17 */
747
748      if (HasCellReactedBefore > 0) { /* This is to check if the cell
   condition is suitable to pass activation energy*/
749        wdot[i] += C_UDMI(c, t, i);
750      }
751      else {
752        C_UDMI(c, t, NumOfSpecies - 1) = 0;
753        /*C_UDMI(c,t,NumOfSpecies+12) = -4;*/
754        wdot[i] = 0;
755      }
756
757      Sum_dYdt_All += wdot[i] * mw[i] / C_R(c, t);
758    }
759    else {
760      if (Sum_dYdt_All > 0) {
761        if (yi[i] >= 0) {
762          wdot[i] = -1 * Sum_dYdt_All*C_R(c, t) / mw[i];
763        }
764        else {
765          wdot[i] = 0;
766        }
767      }
768      else {
769        wdot[i] = -1 * Sum_dYdt_All*C_R(c, t) / mw[i];
```

```
770       }
771       C_UDMI(c, t, NumOfSpecies + 5) = wdot[i];
772     }
773   }
774
775   RunningHeavy = 1;
776
777
778   C_UDMI(c, t, NumOfSpecies + 6) = 0;
779
780 #endif
781 }
782
783 DEFINE_NET_REACTION_RATE(def_net_RR, c, t, particle, press, temp, yi,
      wdot, jac)
784 {
785   int i;
786   double rho = CalculateRRforSpeciesTransport(t, c, press, temp, yi,
      wdot);
787
788 }
789
790 int CheckIf_IsMaxReached(Thread *t, cell_t c, int MaxRe, double Y0) /*
      // UPDATE */
791 {
792   double PrevT = C_UDMI(c, t, NumOfSpecies + 8);
793   double PrevFuel = C_UDMI(c, t, NumOfSpecies + 14);
794
795   if (PrevT == 0) PrevT = 10000;
796   if (PrevFuel == 0) PrevFuel = 50;
797
798   double deltaFuel = Y0 - PrevFuel;
799
800   /* // Depends of Chemistry, This is for CH4
801   //Temperature */
802   double temp = C_T(c, t);
803   int UseFile = 1;
```

```
804   double MaxTemperatureLimit = 10000;
805   double deltaT = temp - PrevT; /*Checking the delta temperature is
       not a good option since it is mixing*/
806
807   if (C_UDMI(c, t, NumOfSpecies + 4) > 0) {
808     MaxTemperatureLimit = C_UDMI(c, t, NumOfSpecies + 4);
809     if (temp > MaxTemperatureLimit) {
810       UseFile = 2;
811     }
812     else {
813       if (MaxRe == 2 & deltaT <= 0) {/*if(deltaFuel<=0 && deltaT<=0){
       */
814         UseFile = 2;
815       }
816       else {
817         UseFile = 1;
818       }
819     }
820   }
821   else {
822     UseFile = 2;
823   }
824
825   return UseFile;
826
827 }
828
829 int CheckIf_IsReacting_v2(Thread *t, cell_t c, int MaxRe)
830 {
831   double PrevT = C_UDMI(c, t, NumOfSpecies + 8);
832   if (PrevT == 0) PrevT = 10000;
833
834   /* // Depends of Chemistry, This is for CH4
835   //Temperature */
836   double temp = C_T(c, t);
837
838
```

```
839  double Gradi = (temp - PrevT) / CURRENT_TIMESTEP;
840
841
842  double MaxTemperatureLimit = 10000;
843  if (C_UDMI(c, t, NumOfSpecies + 4) > 0) {
844    MaxTemperatureLimit = C_UDMI(c, t, NumOfSpecies + 4);
845  }
846  double phi = -1;
847  if (C_YI(c, t, 1) > 0) {
848    phi = C_YI(c, t, 0) / C_YI(c, t, 1) / OxygenFuelRatio_Stoi;
849  }
850  double FuelMassFraction = C_YI(c, t, 0);
851
852  double CellIsEligableForReaction = C_UDSI(c, t, 0);
853
854  int HasCellReactedBefore = C_UDMI(c, t, NumOfSpecies + 12);
855
856
857  /* C_UDSI Below are conditions for reaction to end due combustion*/
858  if (CellIsEligableForReaction > 0) {
859    if (temp < (isreacting_temp_auto + 273.15)) {
860      return -4; /*flame dies due mixing or lack of oxygen etc.*/
861    }
862    else {
863      if (phi > End_Phi_Limit || FuelMassFraction <
     MinimumFuelLimitforReactinToEnd) {
864        C_UDSI(c, t, 0) = 0; /*Cell is no longer eligable*/
865        return -5;
866      }
867      else {
868        if (temp < MaxTemperatureLimit*MAXT_Mult) {
869          /*autoignition temperature: the minimum temperature required
      to ignite a gas or vapor in air without a spark or flame being
     present*/
870          if (MaxRe == 2) {
871            if (HasCellReactedBefore > 0 && abs(Gradi) <
     LigthEnergyLimit2) {
```

```
872             C_UDSI(c, t, 0) = 0; /*Cell is no longer eligable*/
873             return -1; /* // Cantera::setReactionOngoing(); it does
    not end the reaction until it gives signal for 10 times */
874           }if (HasCellReactedBefore > 0 && Gradi > 0) {
875             C_UDSI(c, t, 0) = 0; /*Cell is no longer eligable*/
876             return -3; /* End reaction if temperature is rising at
    second range data*/
877           }
878           else {
879             if (phi > Lower_Phi_Limit && FuelMassFraction >
    MinimumFuelLimitforReactinToBegin) {
880               return 1;
881             }
882             else {
883               return -7; /*due to flamability limits it can no start
     ignition*/
884             }
885           }
886         }
887         else {
888           if ((HasCellReactedBefore > 0 && abs(Gradi) <
    LigthEnergyLimit1)) {
889             C_UDSI(c, t, 0) = 0; /*Cell is no longer eligable*/
890             return -2;
891           }
892           else {
893             if (phi > Lower_Phi_Limit && phi < Upper_Phi_Limit) {
894               return 1;
895             }
896             else {
897               return -8; /*due to flamability limits it can no start
     ignition*/
898             }
899
900           }
901         }
902       }
```

```
903         else {
904           C_UDSI(c, t, 0) = 0; /*Cell is no longer eligable*/
905           return -6; /* temperature passed maximum limit*/
906         }
907       }
908     }
909   }
910   else {
911     return 0;
912   }
913   /* // check by max phi limit given (and minimum temperature in the
       table for ignition but it is cheked inside table encapsulating
       control)...
914   // delta temperature change based on given limits and to be executed
       if larger than certain temperature up until that it is always
       reacting
915     */
916
917 }
918
919 int CheckIf_IsTableEncapsulating(Thread *t, cell_t c)
920 {
921   double press = C_P(c, t) + op_pres;
922   double temp = C_T(c, t);
923
924   if (C_YI(c, t, 1) > 0) {
925     double phi = C_YI(c, t, 0) / C_YI(c, t, 1) / OxygenFuelRatio_Stoi;
926     if (ExtrapolateSystemPressure) {
927       if (phi<Table_MaxPhi && temp>Table_MinTemp) {
928         return 1;
929       }
930       else {
931         return 0;
932       }
933     }
934     else {
935       if (phi<Table_MaxPhi && temp>Table_MinTemp && press > Table_MinP
```

```
    && press < Table_MaxP) {
936         return 1;
937       }
938       else {
939         return 0;
940       }
941     }
942   }
943   else {
944     return 0;
945   }
946 }
947
948 void interpolateHeavyRates(Thread *t, cell_t c)
949 {
950
951   IsTableEncapsulating = CheckIf_IsTableEncapsulating(t, c);
952
953
954   double Outtt[NumOfSpecies + 5];
955   double addOuttt[1];
956   double phi, press, temp;
957   double AdConstants[4];
958   int i = 0;
959   int returnval = 0;
960   /* //Pressure */
961
962   press = C_P(c, t) + op_pres;
963   if (ExtrapolateSystemPressure) {
964     if (press<Table_MinP || press>Table_MaxP) {
965       press = Table_MinP + (press - System_MinP) / (System_MaxP -
     System_MinP)*(Table_MaxP - Table_MinP);
966     }
967   }
968   /*
969     //Phi
970     //Message("Check pres: %f\n", press); */
```

```
971  if (C_YI(c, t, 1) > 0) {
972    phi = C_YI(c, t, 0) / C_YI(c, t, 1) / OxygenFuelRatio_Stoi;
973  }
974  else {
975    phi = 1000;
976    IsTableEncapsulating = 0;
977  }
978
979  MaxReached = CheckIf_IsMaxReached(t, c, C_UDMI(c, t, NumOfSpecies +
      7), C_YI(c, t, 0));
980  IsReacting = CheckIf_IsReacting_v2(t, c, MaxReached);
981
982  temp = C_T(c, t);
983
984  AdConstants[0] = IDV_WeigthingFactor;/* //default 3 */
985  AdConstants[2] = PhiLimit;
986  AdConstants[3] = IDV_TemperatureSearchRange;
987  AdConstants[1] = phi;
988
989
990  C_UDMI(c, t, NumOfSpecies + 6) = 0;
991
992  if (IsReacting > 0 && HeavyFileReadSuccesfully &&
      IsTableEncapsulating)
993  {
994
995    if (MaxReached == 2) {
996      DataRead = DataRead2;
997      /*      //Bool_SecondDataFileUsed = 1; */
998      C_UDMI(c, t, NumOfSpecies + 7) = 2;
999    }
1000   else {
1001     DataRead = DataRead1;
1002     C_UDMI(c, t, NumOfSpecies + 7) = 1;
1003   }
1004
1005
```

```
1006      returnval = InterpolatebyInverseDistanceWeigthed37_2_wlhn_re(press
          , temp, phi, MaxReached, Outtt, AdConstants, DataRead1, addOuttt);
1007
1008
1009      for (i = 0; i < NumOfSpecies + 5; i++) {
1010        C_UDMI(c, t, i) = Outtt[i];
1011      }
1012      C_UDMI(c, t, NumOfSpecies + 6) = 1; /* //OK interpolation */
1013
1014
1015
1016    }
1017    else {
1018
1019      for (i = 0; i < NumOfSpecies + 5; i++) {
1020        C_UDMI(c, t, i) = 0;
1021      }
1022
1023      C_UDMI(c, t, NumOfSpecies + 7) = 0;
1024      C_UDMI(c, t, NumOfSpecies + 6) = 0;
1025
1026    }
1027
1028    C_UDMI(c, t, NumOfSpecies + 8) = temp;
1029    C_UDMI(c, t, NumOfSpecies + 9) = press;
1030    C_UDMI(c, t, NumOfSpecies + 10) = phi;
1031    C_UDMI(c, t, NumOfSpecies + 11) = MaxReached;
1032    C_UDMI(c, t, NumOfSpecies + 12) = IsReacting;
1033    C_UDMI(c, t, NumOfSpecies + 13) = IsTableEncapsulating;
1034    C_UDMI(c, t, NumOfSpecies + 14) = C_YI(c, t, 0);
1035
1036
1037    if (HeavyFileReadSuccesfully == 0) Message("### ERROR: Heavy input
          file could not be read.\n");
1038
1039  }
1040
```

```
1041
1042 void interpolateHeavyRatesFirstTime(Thread *t, cell_t c)
1043 {
1044
1045   IsTableEncapsulating = C_UDMI(c, t, NumOfSpecies + 13);
1046
1047
1048   double Outtt[NumOfSpecies + 5];
1049   double addOuttt[1];
1050   double phi, press, temp;
1051   double AdConstants[4];
1052   int i = 0;
1053   int returnval = 0;
1054   /* //Pressure */
1055
1056   press = C_P(c, t) + op_pres;
1057   if (ExtrapolateSystemPressure) {
1058     if (press<Table_MinP || press>Table_MaxP) {
1059       press = Table_MinP + (press - System_MinP) / (System_MaxP -
      System_MinP)*(Table_MaxP - Table_MinP);
1060     }
1061   }
1062
1063   if (C_YI(c, t, 1) > 0) {
1064     phi = C_YI(c, t, 0) / C_YI(c, t, 1) / OxygenFuelRatio_Stoi;
1065   }
1066   else {
1067     phi = 1000;
1068     IsTableEncapsulating = 0;
1069   }
1070
1071   MaxReached = C_UDMI(c, t, NumOfSpecies + 11);
1072   IsReacting = C_UDMI(c, t, NumOfSpecies + 12);
1073
1074   temp = C_T(c, t);
1075
1076   AdConstants[0] = IDV_WeigthingFactor;/* //default 3 */
```

```
1077  AdConstants[2] = PhiLimit;
1078  AdConstants[3] = IDV_TemperatureSearchRange;
1079  AdConstants[1] = phi;
1080
1081  C_UDMI(c, t, NumOfSpecies + 6) = 0;
1082
1083  if (IsReacting > 0 && HeavyFileReadSuccesfully &&
       IsTableEncapsulating)
1084  {
1085
1086    if (MaxReached == 2) {
1087      DataRead = DataRead2;
1088      /*      //Bool_SecondDataFileUsed = 1; */
1089      C_UDMI(c, t, NumOfSpecies + 7) = 2;
1090    }
1091    else {
1092      DataRead = DataRead1;
1093      C_UDMI(c, t, NumOfSpecies + 7) = 1;
1094    }
1095
1096    returnval = InterpolatebyInverseDistanceWeigthed37_2_wlhn_re(press
     , temp, phi, MaxReached, Outtt, AdConstants, DataRead1, addOuttt);
1097
1098    for (i = 0; i < NumOfSpecies + 5; i++) {
1099      C_UDMI(c, t, i) = Outtt[i];
1100    }
1101    C_UDMI(c, t, NumOfSpecies + 6) = 1; /* //OK interpolation */
1102
1103  }
1104  else {
1105
1106    for (i = 0; i < NumOfSpecies + 5; i++) {
1107      C_UDMI(c, t, i) = 0;
1108    }
1109    /*  //Bool_zerocells = 1; */
1110    C_UDMI(c, t, NumOfSpecies + 7) = 0;
1111    C_UDMI(c, t, NumOfSpecies + 6) = 0;
```

```
1112
1113    }
1114
1115    C_UDMI(c, t, NumOfSpecies + 8) = temp;
1116    C_UDMI(c, t, NumOfSpecies + 9) = press;
1117    C_UDMI(c, t, NumOfSpecies + 10) = phi;
1118    C_UDMI(c, t, NumOfSpecies + 11) = MaxReached;
1119    C_UDMI(c, t, NumOfSpecies + 12) = IsReacting;
1120    C_UDMI(c, t, NumOfSpecies + 13) = IsTableEncapsulating;
1121    C_UDMI(c, t, NumOfSpecies + 14) = C_YI(c, t, 0);
1122
1123
1124    if (HeavyFileReadSuccesfully == 0) Message("### ERROR: Heavy input
        file could not be read.\n");
1125 }
1126
1127
1128 void interpolateandSavetoMemeory(int initialize)
1129 {
1130
1131
1132
1133    Domain *d;
1134    Thread *t;
1135    /* Integrate dissipation. */
1136    real sum_diss = 0.;
1137    cell_t c;
1138    double MaxP = -1E30;
1139    double MinP = 1E30;
1140
1141    d = Get_Domain(1); /* mixture domain if multiphase */
1142
1143    if (ExtrapolateSystemPressure) {
1144
1145      thread_loop_c(t, d)
1146      {
1147        if (FLUID_THREAD_P(t))
```

```
1148      {
1149        begin_c_loop(c, t)
1150        {
1151
1152          double press = C_P(c, t) + op_pres;
1153          MaxP = MAX(press, MaxP);
1154          MinP = MIN(press, MinP);
1155
1156        }end_c_loop(c, t)
1157      }
1158    }
1159
1160 #if RP_NODE
1161    System_MaxP = PRF_GRHIGH1(MaxP);
1162    System_MinP = PRF_GRLOW1(MinP);
1163 #endif
1164  }
1165
1166
1167
1168  thread_loop_c(t, d)
1169  {
1170    if (FLUID_THREAD_P(t))
1171    {
1172      begin_c_loop(c, t)
1173      {
1174        if (initialize > 0) {
1175          interpolateHeavyRatesFirstTime(t, c);
1176        }
1177        else {
1178          interpolateHeavyRates(t, c);
1179        }
1180
1181        /*C_UDMI(c,t,NumOfSpecies+6) = 1;*/
1182
1183      }end_c_loop(c, t)
1184    }
```

```
1185   }
1186
1187
1188 }
1189
1190 DEFINE_ON_DEMAND(InitialExecution)
1191 {
1192 #if RP_NODE
1193   interpolateandSavetoMemeory(1);
1194   Message("## Executed in Node.\n");
1195 #endif
1196
1197 }
1198
1199 DEFINE_ON_DEMAND(Read_CanteraInputsHeavy)
1200 {
1201   Message("## Reading Cantera_Inputs_Heavy.txt File.\n");
1202
1203
1204
1205   /*    // If ArrayKey =1 it is by vector, if 2 it if by Array */
1206
1207   char *Location1 = "";
1208   char *Location2 = "";
1209   char *File1 = "AllData1.bin";
1210   char *File2 = "AllData2.bin";
1211   char *InputFile = "Cantera_Inputs_HeavyFluent.txt";
1212
1213   char * buffer1 = (char *)malloc(1 + strlen(BaseFolder) + strlen(
       File1));
1214   char * buffer2 = (char *)malloc(1 + strlen(BaseFolder) + strlen(
       File2));
1215   char * buffer_in = (char *)malloc(1 + strlen(BaseFolder) + strlen(
       InputFile));
1216
1217   FILE *pFile1, *pFile2, *iFile;
1218
```

```
1219  const char *line;
1220  char str1[200];
1221
1222
1223
1224  strcpy(buffer1, BaseFolder);
1225  strcat(buffer1, File1);
1226
1227
1228  strcpy(buffer2, BaseFolder);
1229  strcat(buffer2, File2);
1230
1231
1232  strcpy(buffer_in, BaseFolder);
1233  strcat(buffer_in, InputFile);
1234
1235
1236
1237  pFile1 = fopen(buffer1, "r");
1238  pFile2 = fopen(buffer2, "r");
1239  iFile = fopen(buffer_in, "r");
1240
1241  if (pFile1 != NULL  && pFile2 != NULL)
1242  {
1243
1244
1245    Message("Cantera_Inputs_Heavy.txt does not exists. Default files
      opened and default values will be used.\n");
1246    fclose(pFile1);
1247    fclose(pFile2);
1248    Location1 = (char *)malloc(1 + strlen(buffer1));
1249    strcpy(Location1, buffer1);
1250    Location2 = (char *)malloc(1 + strlen(buffer2));
1251    strcpy(Location2, buffer2);
1252
1253
1254  }
```

```
1255  else if (iFile != NULL) {
1256
1257    Message("Input File Opened.\n");
1258
1259
1260    fscanf(iFile, "%s", str1);
1261    File1 = (char *)malloc(1 + strlen(str1));
1262    strcpy(File1, str1);
1263
1264    fscanf(iFile, "%s", str1);
1265    File2 = (char *)malloc(1 + strlen(str1));
1266    strcpy(File2, str1);
1267
1268
1269    fscanf(iFile, "%f", &OxygenFuelRatio_Stoi);
1270
1271    fscanf(iFile, "%d", &DebugOption);
1272
1273    fscanf(iFile, "%f", &IDV_WeigthingFactor);
1274
1275    fscanf(iFile, "%f", &PhiLimit);
1276
1277    fscanf(iFile, "%f", &IDV_TemperatureSearchRange);
1278
1279    fscanf(iFile, "%f", &LigthEnergyLimit1);
1280
1281    fscanf(iFile, "%f", &LigthEnergyLimit2);
1282
1283    fscanf(iFile, "%d", &ExtrapolateSystemPressure);
1284
1285    fscanf(iFile, "%f", &End_Phi_Limit);
1286
1287    fscanf(iFile, "%f", &Upper_Phi_Limit);
1288
1289    fscanf(iFile, "%f", &Lower_Phi_Limit);
1290
1291    fscanf(iFile, "%f", &isreacting_temp_auto);
```

```
1292
1293     fscanf(iFile, "%f", &MinimumFuelLimitforReactinToEnd);
1294
1295     fscanf(iFile, "%f", &MinimumFuelLimitforReactinToBegin);
1296
1297     fscanf(iFile, "%f", &EnenrgyMult);
1298
1299
1300
1301     buffer1 = (char *)malloc(1 + strlen(BaseFolder) + strlen(File1));
1302     strcpy(buffer1, BaseFolder);
1303     strcat(buffer1, File1);
1304
1305     buffer2 = (char *)malloc(1 + strlen(BaseFolder) + strlen(File2));
1306     strcpy(buffer2, BaseFolder);
1307     strcat(buffer2, File2);
1308
1309
1310     pFile1 = fopen(buffer1, "r");
1311     pFile2 = fopen(buffer2, "r");
1312
1313
1314     Message("## VALUES READ FROM INPUT FILE ######\n");
1315     Message("## Stoichiometric Oxygen-Fuel Ratio: %f\n",
    OxygenFuelRatio_Stoi);
1316     Message("## Weight factor for inverse distance weigthing: %f\n",
    IDV_WeigthingFactor);
1317     Message("## Phi limit for application of WF=1 : %f\n", PhiLimit);
1318     Message("## Range of temperature to use in interpolation: %f\n",
    IDV_TemperatureSearchRange);
1319     Message("## Energy release limit by ligth species, branch1: %f\n",
     LigthEnergyLimit1);
1320     Message("## Energy release limit by ligth species, branch2: %f\n",
     LigthEnergyLimit2);
1321     Message("## End F-A ER limit for reactions %f\n", End_Phi_Limit);
1322     Message("## Upper flamability F-A ER limit %f\n", Upper_Phi_Limit)
    ;
```

```
1323     Message("## Lower flamability F-A ER limit %f\n", Lower_Phi_Limit)
    ;
1324     Message("## Ignition temperature limit %f\n", isreacting_temp_auto
    );
1325     Message("## MinimumFuelMassFractionLimitforReactinToEnd %.3e\n",
    MinimumFuelLimitforReactinToEnd);
1326     Message("## MinimumFuelMassFractionLimitforReactinToBegin %.3e\n",
     MinimumFuelLimitforReactinToBegin);
1327     Message("## Energy term multiplier %f\n", EnenrgyMult);
1328     if (ExtrapolateSystemPressure) {
1329       Message("## System Pressure Extrapolation option is TRUE. Input
    files pressure values are to be adjusted.\n");
1330     }
1331     else {
1332       Message("## System Pressure Extrapolation option is FALSE.\n");
1333     }
1334     Message("################################################\n");
1335
1336
1337
1338     if (pFile1 != NULL && pFile2 != NULL)
1339     {
1340
1341       fclose(pFile1);
1342       fclose(pFile2);
1343       Message("Files Exists.\n");
1344
1345       Location1 = (char *)malloc(1 + strlen(buffer1));
1346       strcpy(Location1, buffer1);
1347       Location2 = (char *)malloc(1 + strlen(buffer2));
1348       strcpy(Location2, buffer2);
1349
1350     }
1351
1352     fclose(iFile);
1353
1354   }
```

```
1355  else {
1356    Message("Couldn't open any input files.\n\n");
1357  }
1358
1359
1360
1361  DataRead1 = ReadFileandPassDataArray(Location1);
1362  DataRead2 = ReadFileandPassDataArray(Location2);
1363
1364  Material *mm = mixture_material(Get_Domain(1)), *sp;
1365  int NumOfSpeciesinCase = mm->n_components;
1366
1367  int NnumberofSpecies = (int)DataRead1[0][2];
1368  int NumberofSectionbyPhi = (int)DataRead1[0][3];
1369  int n_Reaction = (int)DataRead1[0][NumberofSectionbyPhi + 15 +
       NnumberofSpecies * 2 + 2];
1370  /*
1371    // OK until here   */
1372
1373  Message("## NnumberofSpecies in Data File: %d\n", NnumberofSpecies);
1374  Message("## NnumberofSpecies in Case: %d\n", NumOfSpeciesinCase);
1375  Message("## NnumberofSpecies in Script: %d\n", NumOfSpecies);
1376  Message("## NumberofSectionbyPhi: %d\n", NumberofSectionbyPhi);
1377  Message("## n_Reaction: %d\n", n_Reaction);
1378  Message("################################################\n");
1379
1380
1381  if (DataRead1[0][0] > 0) {
1382    Message("## File read succesfully.\n");
1383    HeavyFileReadSuccesfully = 1;
1384  }
1385
1386  Message("HeavyFileReadSuccesfully: %d\n", HeavyFileReadSuccesfully);
1387
1388  Message("OK\n");
1389
1390  Table_MaxPhi = DataRead1[0][NumberofSectionbyPhi + 5];
```

```
1391   Table_MinTemp = DataRead1[0][NumberofSectionbyPhi + 7];
1392   Table_MinP = DataRead1[0][NumberofSectionbyPhi + 11];
1393   Table_MaxP = DataRead1[0][NumberofSectionbyPhi + 10];
1394
1395   Message("## Table_MaxPhi : %f\n", Table_MaxPhi);
1396   Message("## Table_MinTemp : %f\n", Table_MinTemp);
1397   Message("## Table_MinP : %f\n", Table_MinP);
1398   Message("## Table_MaxP : %f\n", Table_MaxP);
1399
1400
1401
1402
1403
1404
1405 }
1406
1407 DEFINE_ON_DEMAND(RenameUDMs)
1408 {
1409
1410   Reserve_User_Memory_Vars(50);
1411   Set_User_Memory_Name(0, "w_lh_FUEL");
1412   Set_User_Memory_Name(1, "w_lh_O2");
1413   Set_User_Memory_Name(2, "w_lh_ N2");
1414   Set_User_Memory_Name(3, "w_lh_H2");
1415   Set_User_Memory_Name(4, "w_lh_H2O");
1416   Set_User_Memory_Name(5, "w_lh_CO");
1417   Set_User_Memory_Name(6, "w_lh_CO2");
1418   Set_User_Memory_Name(7, "w_lh_CH3");
1419   Set_User_Memory_Name(8, "w_lh_CH2O");
1420   Set_User_Memory_Name(9, "w_lh_ C2H2");
1421   Set_User_Memory_Name(10, "w_lh_C2H4");
1422   Set_User_Memory_Name(11, "w_lh_ H2O2");
1423   Set_User_Memory_Name(12, "w_lh_NO");
1424   Set_User_Memory_Name(13, "w_lh_HCN");
1425   Set_User_Memory_Name(14, "w_lh_ H");
1426   Set_User_Memory_Name(15, "w_lh_OH");
1427   Set_User_Memory_Name(16, "w_lh_O");
```

```
1428   Set_User_Memory_Name(17, "w_lh_CH");
1429   Set_User_Memory_Name(18, "w_lh_CH2");
1430   Set_User_Memory_Name(19, "w_lh_HCO");
1431   Set_User_Memory_Name(20, "w_lh_HO2");
1432   Set_User_Memory_Name(21, "w_lh_C2H3");
1433   Set_User_Memory_Name(22, "w_lh_NO2");
1434   Set_User_Memory_Name(23, "w_lh_C2H6");
1435   Set_User_Memory_Name(24, "w_lh_AR");
1436   Set_User_Memory_Name(NumOfSpecies - 1, "eh_mole_dot_w_lh"); /*Heavy
       EnergyTerm ConstantP*/
1437   Set_User_Memory_Name(NumOfSpecies, "SummAll_ener");/*Heavy
       EnergyTerm ConstantV*/
1438   Set_User_Memory_Name(NumOfSpecies + 1, "Specific Heat Term");
1439   Set_User_Memory_Name(NumOfSpecies + 2, "Normalized Temperature");
1440   Set_User_Memory_Name(NumOfSpecies + 3, "Check if Executed in Kourdis
        Code");
1441   Set_User_Memory_Name(NumOfSpecies + 4, "Maximum Temperature Limit");
1442   /* Until here it is read from Kourdis Code*/
1443   Set_User_Memory_Name(NumOfSpecies + 5, "w_lh_DMY");
1444   Set_User_Memory_Name(NumOfSpecies + 6, "Inerpolation  key 0\1");
1445   Set_User_Memory_Name(NumOfSpecies + 7, "File to Use 1 or 2");
1446   Set_User_Memory_Name(NumOfSpecies + 8, "Temperature Int");
1447   Set_User_Memory_Name(NumOfSpecies + 9, "Pressure Int");
1448   Set_User_Memory_Name(NumOfSpecies + 10, "Phi Int");
1449   Set_User_Memory_Name(NumOfSpecies + 11, "MaxReached");
1450   Set_User_Memory_Name(NumOfSpecies + 12, "IsReacting");
1451   Set_User_Memory_Name(NumOfSpecies + 13, "IsTableEncapsulating");
1452   Set_User_Memory_Name(NumOfSpecies + 14, "Previous Fuel Fraction");
1453   Set_User_Memory_Name(NumOfSpecies + 15, "Density");
1454
1455
1456
1457 }
1458
1459 DEFINE_SOURCE(Heavy_Energy, c, t, dS, eqn)
1460 {
1461   double source = C_UDMI(c, t, NumOfSpecies - 1)*C_UDMI(c, t,
```

```
    NumOfSpecies + 15)*EnenrgyMult; /*Unit suuplied by code is J/m^3/s
    , W/kg:, unit expected by fluent:  W/m^3, so if multiplied by
    density it will be...
1462   C_UDMI(c, t, NumOfSpecies+15) , density
1463   in cnatera : dUh = m_wdot_lh[NumofSpecies + 1]*m_mass;  ydot[2] is
    in watts and eh_dot_Dydt is [W/kg]so no time term is neccesarry */
1464   dS[eqn] = 0;
1465   return source;
1466 }
1467
1468 DEFINE_EXECUTE_AT_END(InterpolateHeavy)
1469 {
1470 #if RP_NODE
1471   Message("## Initial execution:\n### Interpolating Heavy Terms.\n");
1472   interpolateandSavetoMemeory(0);
1473   Message("### Interpolated Heavy Terms.\n");
1474 #endif
1475
1476 }
1477
1478 DEFINE_SPECIFIC_HEAT(cp_wo_dmy, T, Tref, h, yi)
1479 {
1480   real cp = 0.;
1481
1482   if (NULLP(cnu))
1483     fill_cache_variables();
1484
1485   double sumY = 0;
1486
1487   double rho, ln_T = log(T);
1488   double hi[MAX_PDF_SPECIES], cpi[MAX_PDF_SPECIES + 2], si[
    MAX_PDF_SPECIES];
1489
1490   int i = 0;
1491
1492   calc_thermo_props(T, ln_T, yi, cpi, hi, si);
1493   /*cpi last term is the mass weigthed average of cp's*/
```

```
1494
1495
1496    cp = cpi[n_spe_isat + 1];
1497
1498    *h = cp*(T - Tref);
1499    return cp;
1500 }
1501
1502
1503 #undef EXP_LIM
1504 #undef EVALUATE_CP_POLY
1505 #undef EVALUATE_H_POLY
1506 #undef EVALUATE_S_POLY
```

APPENDIX D: Compilation of Fluent User Defined Subroutine

In Linux, for compilation of user defined subroutines, the necessary lines in *user.udf* and *makefile.udf* should be updated and following set of commands similar to below ones should be executed.

```
CompileFolder="/s/sguryuva/Development_190228/CompileKourdisLib5-Linux3D"
RunFolder="/s/sguryuva/Development_190228/M010_S26_R044_3D_LESwScript"
envirH="3ddp_host"
envirN="3ddp_node"
envirP="3ddp"
code="default-net-rate-v4_067_forspeciestransport.c"
cd $CompileFolder
rm KourdisCodes.o
g++ -std=c++11 -fPIC KourdisCodes.cpp -c
rm -rf $RunFolder/libudf
mkdir $RunFolder/libudf
mkdir $RunFolder/libudf/src
mkdir $RunFolder/libudf/lnamd64
mkdir $RunFolder/libudf/lnamd64/$envirH
mkdir $RunFolder/libudf/lnamd64/$envirN
mkdir $RunFolder/libudf/lnamd64/$envirP
cp $CompileFolder/makefile.udf2 $RunFolder/libudf/makefile
cp $CompileFolder/$code $RunFolder/libudf/src/
cp $CompileFolder/udf.h $RunFolder/libudf/src/
cp $CompileFolder/makefile.udf $RunFolder/libudf/src/makefile
cp $CompileFolder/KourdisCodes.cpp $RunFolder/libudf/
cp $CompileFolder/KourdisCodes.o $RunFolder/libudf/lnamd64/$envirH/
cp $CompileFolder/KourdisCodes.o $RunFolder/libudf/lnamd64/$envirN/
cp $CompileFolder/KourdisCodes.o $RunFolder/libudf/lnamd64/$envirP/
cp $CompileFolder/user.udf $RunFolder/libudf/lnamd64/$envirH/user.udf
cp $CompileFolder/user.udf $RunFolder/libudf/lnamd64/$envirN/user.udf
cp $CompileFolder/user.udf $RunFolder/libudf/lnamd64/$envirP/user.udf
cd $RunFolder/libudf/
```

```
make "FLUENT_ARCH=lnamd64"
cd $RunFolder
chmod -R 777 libudf
```

In Windows 10, for compilation of user defined subroutines, the necessary lines in *user_nt_h.udf, user_nt_n.udf, user_nt_s.udf* and *makefile* should be updated and following set of commands similar to below ones should be executed.

```
SetLocal EnableDelayedExpansion
SET CompileFolder=D:\GoogleDrive\Combustion\CompileKourdisLib6-Windows-19p2
SET RunFolder=D:\GoogleDrive\Combustion\M012_S26_R049_3D_LESwScript
SET envirH=3ddp_host
SET envirN=3ddp_node
SET envirS=3ddp
SET code=default-net-rate-v4_068_forspeciestransport.c
D:
cd %CompileFolder%
DEL KourdisCodes.obj
cl /c /Za /FAs KourdisCodes.cpp
dumpbin /symbols KourdisCodes.obj
rmdir %RunFolder%\libudf /s /q
mkdir %RunFolder%\libudf
mkdir %RunFolder%\libudf\src
mkdir %RunFolder%\libudf\win64
mkdir %RunFolder%\libudf\win64\%envirH%
mkdir %RunFolder%\libudf\win64\%envirN%
mkdir %RunFolder%\libudf\win64\%envirS%
copy %CompileFolder%\KourdisCodes.obj %RunFolder%\libudf\win64\%envirH%\
copy %CompileFolder%\KourdisCodes.obj %RunFolder%\libudf\win64\%envirN%\
copy %CompileFolder%\KourdisCodes.obj %RunFolder%\libudf\win64\%envirS%\
copy %CompileFolder%\user_nt_h.udf %RunFolder%\libudf\win64\%envirH%\user_nt.udf
copy %CompileFolder%\user_nt_n.udf %RunFolder%\libudf\win64\%envirN%\user_nt.udf
copy %CompileFolder%\user_nt_s.udf %RunFolder%\libudf\win64\%envirS%\user_nt.udf
```

```
copy %CompileFolder%\makefile %RunFolder%\libudf\win64\%envirH%\
copy %CompileFolder%\makefile %RunFolder%\libudf\win64\%envirN%\
copy %CompileFolder%\makefile %RunFolder%\libudf\win64\%envirS%\
copy %CompileFolder%\%code% %RunFolder%\libudf\src\
copy %CompileFolder%\udf.h %RunFolder%\libudf\src\
D:
cd %RunFolder%\libudf\win64\%envirH%\
nmake
cd %RunFolder%\libudf\win64\%envirN%\
nmake
cd %RunFolder%\libudf\win64\%envirS%\
nmake
cd %RunFolder%
"C:\Program Files\ANSYS Inc\v192\fluent\ntbin\win64\fluent.exe" -r19.2.0
```

APPENDIX E: Sandia Spray-A Mesh Independence Study

The level of grid refinement is tested by an independence study conducted with a finer mesh domain. As seen in Figure E.1 the base size used for setting the minimum cell size has not been changed due to the Lagrangian solution requirements [152] but the size of the minimum sized domain increased.

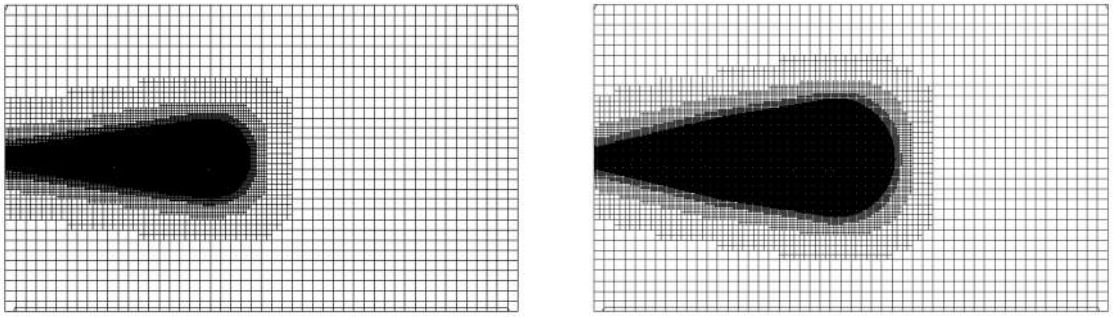

Figure E.1. The gird on the center-plane for two different mesh refinements.

Figure E.2 shows that there is no difference in spray and vapor penetrations based on the analyses ran for 0.5ms. As long as the finest mesh region covers the vapor domain, it is valid to increase the cell size gradually.

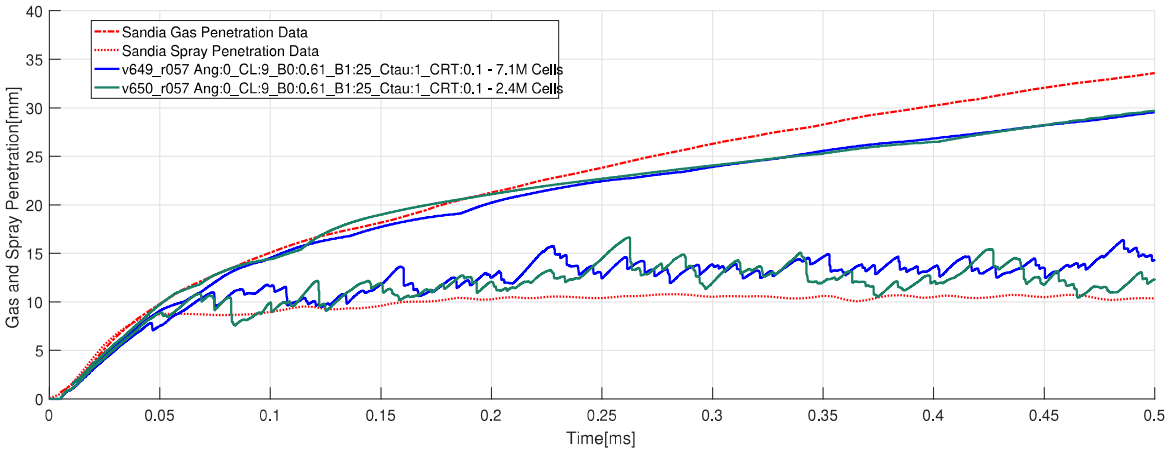

Figure E.2. Spray and gas penetration for 0.5ms.

APPENDIX F: Sandia Spray-A KH-RT Coefficients Assessment

Figure F.1 shows that when B1 is equal to 25, as the CL value changes the variation in spray and vapor penetration lies in the range of statistical variation of different realizations.

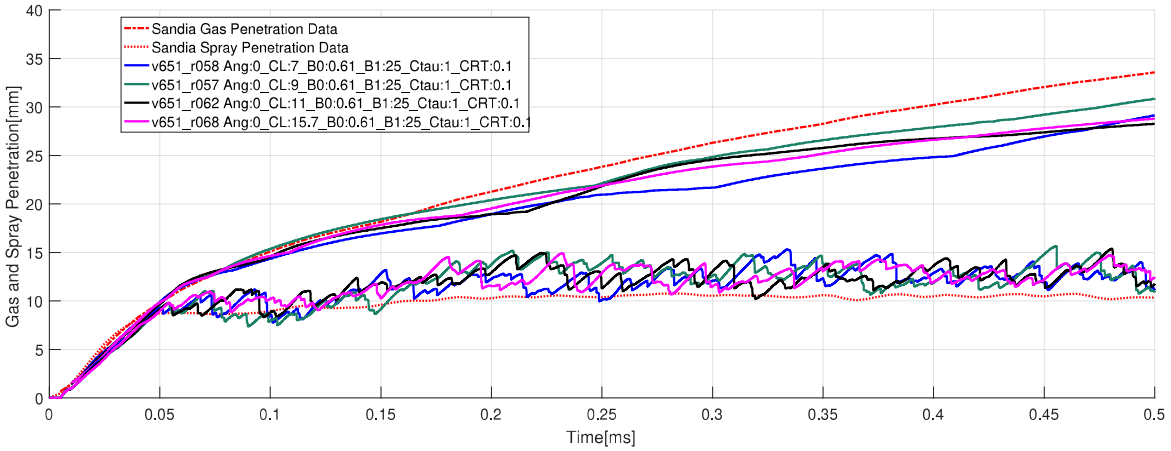

Figure F.1. Spray and gas penetration for 0.5ms for different CL values with B1 equals to 25.

Figure F.2 shows that when CL value is set as 7, the increase in B1 value up to 35 increases spray penetration but B1 of 45 does not cause a measurable change. Based on the vapor penetration values, it is not possible to make a valuable comment that since the change in vapor penetration lies within the statistical variance.

For LES simulations assessment of KH-RT parameters should be done using statistically averaged data which requires averaging of at least 24 realizations for each setting [153] and requires a significant amount of computational source.

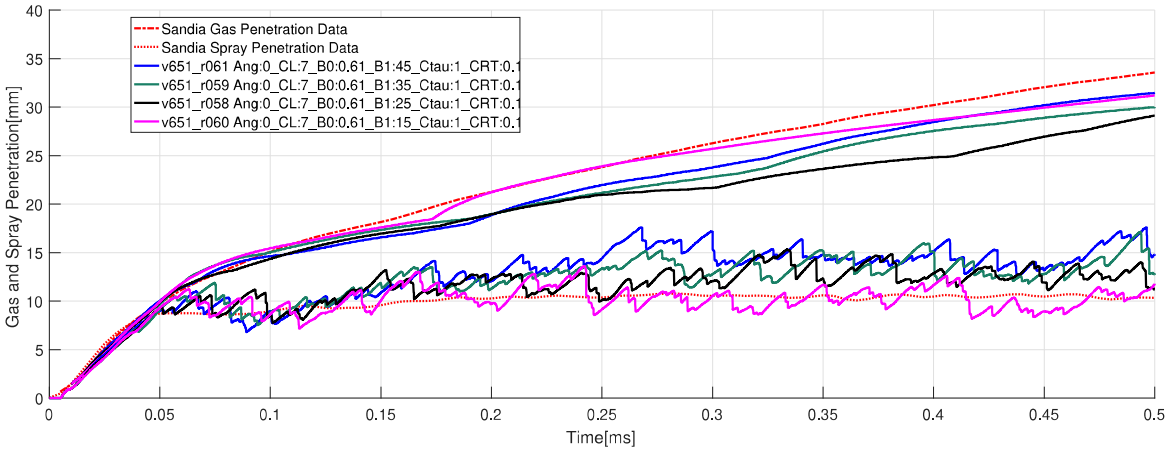

Figure F.2. Spray and gas penetration for 0.5ms for different B1 values with CL equals to 7.

APPENDIX G: About Figures Used in the Thesis

The images that emerged within the scope of this thesis work and whose copyrights were transferred to the publishing house were used in the thesis book in accordance with the publishing policy valid for the reuse of the text and graphics produced by the author on the website of the publisher.