CHALMERS
UNIVERSITY OF TECHNOLOGY

Numerical simulations of highly perturbed lean hydrogen-air flames

A detailed investigation on strained lean hydrogen combustion characteristics

Master's thesis in Automotive Engineering (MPAUT)

MANOJKUMAR CAUVERI RAMAKRISHNA

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's Thesis 2025

Numerical simulations of highly perturbed lean hydrogen-air flames

A detailed investigation on strained lean hydrogen combustion characteristics

Manojkumar Cauveri Ramakrishna

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Chalmers University of Technology
Gothenburg, Sweden 2025

Numerical simulations of highly perturbed lean hydrogen-air flames
A detailed investigation on strained lean hydrogen combustion characteristics

Supervisor and examiner: Mr. Andrei Lipatnikov

Chalmers University of Technology
SE 412 96, Gothenburg

Abstract

Premixed or partially premixed fuel configuration with turbulent combustion is widely used in all combustion technologies and applications available in the industry. Though there are many interesting alternatives exist to combustion technologies (e.g., battery technologies), combustion technologies still become relevant as of today because of the existing infrastructure built around the combustion applications and the ease of application. Many popular companies (e.g., Volvo Car, Siemens, etc) and research enthusiasts in Sweden and worldwide are developing interesting strategies to mitigate the global warming, which is one of the downsides of combustion technologies. Among various renewable fuels, hydrogen becomes one of the important carbon free fuels of interest. Also compared to the other fuels, hydrogen fuel has very good combustion characteristics (high laminar flame speed, wide flammability limits, low ignition energy, etc.) and can be produced using various technologies and renewable methods. It is important for researchers to develop efficient combustion models to make the application of hydrogen fuel more practical and adaptable in various applications. This can be done only by using the data obtained by experimental results and Computational Fluid Dynamics (CFD) tools effectively. However, there is not enough research available which can accurately predict the burning rate and combustion characteristics of turbulent hydrogen – air mixture. Hydrogen has a significant high burning rate which is usually controlled by local mixture composition changes due to higher molecular diffusivities of hydrogen air mixture. The most popular approach is based on the hypothesis that the entire turbulent flame regime is controlled by local flamelets which are highly perturbed. This thesis work mainly covers the study of such highly perturbed local flamelets. Eventually, the data can be further used to design or develop hydrogen-air turbulent combustion applications.

CHEMKIN-PRO is the main tool which is used in this project to simulate combustion characteristics of hydrogen – air mixture. CHEMKIN PRO gives flexibility to the user to setup specific boundary conditions based on user needs to obtain a desired output. This project can be divided into two parts, the first part – simulation of laminar hydrogen air flames and the second part – simulation of strained laminar flames. In laminar flame simulations, the main goal is to study the effect of different input configurations like inlet temperature, pressure, diffusion and transport model effects, gird resolution and curvature and different combustion mechanisms on laminar flame speed S_l for different equivalence ratios (φ) ranging from 0.5 to 2.9. It is observed that the flame speed increases rapidly up to $\varphi = 1.5$ and then reduces sharply. In the second part, strained laminar flame simulations, the goal is to study the strain rates at which the maximum consumption velocity S_C^{max} and the extinction has attained for lean hydrogen air flames ranging from $\varphi = 0.36\ to\ 0.80$, using Oppdif module in CHEMKIN PRO. For strained flames, consumption velocity S_c becomes more relevant as it represents the global burning rate of the mixture. Later, the effects of different mechanisms, inlet temperature, different equivalence ratios, pressure and flame thickness are studied and presented at the end of this thesis. A significant amount of time is also spent on extracting and visualizing the relevant data in MATLAB from the output obtained by CHEMKIN PRO.

Acknowledgement

Sincere thanks to Mr. Andrei Lipatnikov, Research Professor, Mechanics and Maritime Sciences for the continuous guidance and support throughout this project. Also, for the detailed knowledge transfer on this project in each step, which has a direct contribution on the output of this project.

This project would not have been possible without Mr. Andrei’s support and guidance. Thank you!

Manojkumar Cauveri Ramakrishna, Gothenburg, 2025

List of acronyms

Symbol	Description	Unit
A	Helmholtz free energy	J/mol
C_p	Molar heat capacity at constant pressure	J/mol
c_k	Molar concertation	mol/m^3
D	Mass diffusivity	m^2/s
Da	Damköhler number	-
E_a	Activation energy	J/mol
FAR	Fuel to Air ratio	-
G	Gibbs free energy	J/mol
H	Molar enthalpy	J/mol
L_m	Markstein length	m
Le	Lewis number	-
Ma	Markstein Number	-
P	Pressure	Pa
P_{atm}	Atmospheric pressure	Pa
Pr	Prandtl Number	-
R	Universal gas constant	$J/mol\ K$
Re	Reynolds number	-
S	Entropy	$J/mol\ K$
S_c	Consumption velocity	m/s
S_c^{max}	Peak consumption velocity	m/s
S_L	Laminal flame speed	m/s
T_b	Temperature of burnt gases	K
T_u	Temperature of unburnt gases	K
U	Internal energy	J/mol
U_T	Turbulent burning velocity	m/s
W	Molecular weight	kg/mol
X_k	Molar fraction	-
Y_k	Mass Fraction	-
a	Strain rate	s^{-1}
k_f, k_r, k_i	Forward, backward and equilibrium reaction constant	-
m_a	Mass of Air	kg
m_f	Mass of Fuel	kg

Greek symbols

Symbol	Description	Unit
α	Heat release parameter	-
β	Zel'dovich number	-
δ_L	Laminar flame thickness	m
Φ	Equivalence ratio	-
κ	Thermal diffusivity	m^2/s
λ	Thermal conductivity	W/mK
ν	Viscosity	$Pa.s$
ρ	Density	kg/m^3
τ_c	Timescale of chemical reaction	s
τ_f	Timescale of the mass transport phenomena	s
τ_d	Chemical timescale to the diffusion time constant	s
ν'_{ki}, ν''_{ki}	Stoichiometric coefficients	-
$\mathcal{K}$	Flame stretch factor	s^{-1}
$\dot{\omega}$	Rate of production	$mol/m^3 s$

List of figures

List of tables

Table of contents

1. Introduction

In recent years, there has been a wide focus in all industrial sectors on how to mitigate the carbon emissions but keeping current systems and eco system built around the fossil fuels. Due to the increased rate of global warming, abnormal increase in prices of conventional hydrocarbon fuels, geo political situation and supply chain disruptions, the interest on alternative, renewable, carbon free fuels has become the interest of choice of many industries [1]. For example, in line with the Paris agreement (COP21), International Maritime Organization has been put a strategy for significant reductions in carbon emissions, -40% by 2030 and -70% by 2050 compared to the emissions as of 2008 for the emissions happening because of the global sea shipments [2].

Though the combustion properties of hydrogen fuel are very different compared to the fossil fuels, hydrogen is one of the key promising fuels of future which can be a very good alternative to fossil fuels. Hydrogen is abundantly available and can be even produced using renewable energy resources, if required. As the technology on sustainable hydrogen generation progresses, the usage of hydrogen fuel in combustion systems can contribute to a significant greenhouse gas emissions reduction, which is one of the biggest problems as of today [1]. Though, majority of the current hydrogen production still depend on sources like natural gas, coal, crude oil and electrolysis, there have been successful innovations towards hydrogen generation using sustainable methods. Due to its wide range of applications ranging from fuel cells to combustion technology, the usage of hydrogen becomes even more interesting [1].

The concept of hydrogen combustion technology (key focus of this project) is an important combustion carbon free technology compared to the current fossil fuels which contributes to around 17% of the world's greenhouse gases [3]. The fossil fuels are responsible for many harmful combustion bi products such as carbon monoxide (CO), nitrous oxide emissions (NO_X) and particulate matter (PM) [3]. These emissions have a direct impact on the quality of the air, have a severe impact on atmosphere and the ozone layer which plays a significant role in shielding harmful rays entering the Earth's atmosphere [3]. Also, the other point to note is that there is already a developed ecosystem for combustion systems make use of hydrocarbon fuels. Economically, it can be a huge impact to make a complete transition into a new technology [3]. For example, the automotive sector, which has a dominance of hydrocarbon-based combustion systems, is now in the transition phase to electric powertrains. This requires a huge shift in the infrastructure [3]. With a minimum shift required in the infrastructure yet keeping the majority of existing ecosystem built around the existing fossil fuels, hydrogen can be used as an easy replacements to fossil fuels [3].

2. Objectives and Research scope

In the world of combustion science, hydrogen fuel has its own significance because of its distinctive characteristics compared to normal hydrocarbon fuels. The hydrogen typically has wide range of flammability limits and higher diffusivity compared to hydrocarbon fuels. This allows the user to operate under very rich to lean conditions. Another important property is the bi products of combustion. Though, hydrogen systems tend to produce NO_X emissions at higher operating temperatures, there are almost zero greenhouse gas emissions with a higher burning velocity and low ignition energy compared to conventional fuels [4].

Although, the ultimate interest of researchers is to predict the behaviour (flame speed or burning velocity) of turbulent flames and though, several theories, models exit to predict the premixed turbulent combustion, there is still enough work that needs to be carried out to predict the exact behaviour of premixed turbulent combustion systems because of its complexity and highly unpredictable behaviour [4]. It is widely accepted that turbulent flames consist of many local zones of highly perturbed laminar flame structures, which are stretched by turbulence but locally resemble laminar flames [4].

Many theories make use of only flame displacement speed or the rate at with the flame front moves within the unburnt mixture to calculate the turbulent flame speed. These theories completely ignore the complexities of turbulent flame mixtures and the rate at with the fuel is consumed, which is called as the consumption velocity S_c [4]. But, the studies reviewed by Lipatnikov and Chomiak [4] indicates that the use of peak consumption velocities S_c^{max}, see equation 2.1, to calculate the turbulent flame speed can better capture the real-time characteristics like local variations in enthalpy and mixture composition due to differences in molecular transport processed for different species, extinction and propagation phenomena compared to the models which only makes use of only displacement speed [5]. Also, S_c^{max} acts as one of the key inputs for Reynolds-Averaged Navier-Stokes (RANS) or Large Eddy Simulations (LES) in correcting the reaction source terms in turbulent flame-let libraries and allow models to account for preferential diffusion effects, which play an important role in lean hydrogen combustion [5].

$$U_t = \mathrm{A}u' Da^{\frac{1}{4}} \left(\frac{S_c^{max}}{S_l}\right)^{1/2} \tag{2.1}$$

$U_t - Turbulent\ burning\ velocity\ in\ m/s$

$A - Emperical\ constant$

$D_a - \ Damköhlers\ number$

$S_l - Laminar\ flame\ speed\ in\ m/s$

$S_c^{max} - Peak\ consumption\ velocity\ in\ m/s$

$u' - rms\ turbulent\ velocity\ in\ m/s$

One of the main objectives of this project is to calculate S_c^{max} of premixed lean hydrogen highly perturbed combustion flames using CHEMKIN PRO [6] simulation tool.

At first, using the prebuilt flame speed calculator, for different lean conditions, ranging from $\varphi = 0.36\ to\ 3$, laminar flame speed is calculated for different combustion mechanisms. In this part of the project, the combustion mechanisms such as Dublin [7], Princeton [8] and Konnov [9] are used to compare flame speeds yielded by those mechanisms. Later using mechanism Milano [10], the effect of pressure, inlet temperature and the effect of grid resolutions in CHEMKIN PRO is illustrated.

The next part of this project is mainly focused on calculation of peak consumption velocities S_c^{max} of premixed lean hydrogen combustion flames $\varphi < 0.8$. To do so, OPPDIF figure 3.2.3.1 simulation model which is readily available in the CHEMKIN PRO default models is selected. The strain rate is varied until the extinction point is reached and S_c^{max} is calculated manually using the output data obtained after OPPDIF simulations. Here again, the effect of different mechanisms on S_c^{max} is illustrated, followed by pressure and temperature effects on S_c^{max}.

3. Literature review

3.1 Characterizing combustion:

3.1.1 Non-dimensional quantities:

The non-dimensional quantities are generally used in fluid dynamics to characterize the fluid flows. The relevant non-dimensional quantities used to describe the combustible flows are detailed in this section. When a fuel undergoes combustion in the presence of air the stoichiometric ratio of mass of fuel (m_f) to mass of air (m_a) is called the stoichiometric Fuel-Air Ratio (FAR_{st}). However, when the reaction is carried out with Fuel-Air Ratio (FAR) other than the stoichiometric ratio, then equivalence ratio (Φ) of the mixture can be defined as the ratio of FAR to FAR_{st}. This is shown in equation 3.1. Equivalence ratio is used as a measure of the quantity of fuel and oxidizer available in the mixture and is detrimental to the rate of the reaction and the flame speeds. If $\Phi = 1$ the ratio is stoichiometric, if $\Phi > 1$ the mixture is a rich mixture and for $\Phi < 1$ the mixture can be called as a lean mixture.

$$\Phi = \frac{FAR}{FAR_{st}} \tag{3.1}$$

Lewis number (L_e) is defined as the ratio of thermal diffusivity (κ) of the mixture to the mass diffusivity (D) of the deficient reactant as shown in equation 3.2. It has been shown that the local flame speed and the surface curvature of the flame has a strong correlation with the Lewis number and the turbulent flame speed increases with a decrease in its value [11].

$$Le = \frac{\kappa}{D} \tag{3.2}$$

Prandtl number (Pr) is defined as the ratio of kinematic viscosity of the fluid (ν) to the thermal diffusivity and is given by equation 3.3 A very small value of this quantity (i.e. $Pr \ll 1$) implies that the thermal diffusivity is dominant in the flow whereas, a very large value (i.e. $Pr \gg 1$) indicates that momentum diffusivity is the dominant characteristic. This is mainly common in liquid metals and the viscous liquids but can be applicable also for gases at high temperatures [12, 13].

$$Pr = \frac{\nu}{\kappa} \tag{3.3}$$

Markstein number (Ma) is defined as the ratio of Markstein length (L_M) to the characteristic laminar flame thickness (δ_L). The Markstein number is used to describe the relation between the local burning rate with the local flame strain rate and local flame front curvature [14]. Equation 3.4 shows the mathematical expression for this quantity. Markstein length L_M

characterizes change in the flame speed with a small change in the strain rate represented by equation 3.5

$$Ma = \frac{L_M}{\delta_L} \tag{3.4}$$

$$L_M = \left.\frac{dS_L}{dK}\right|_{K\to 0} \tag{3.5}$$

Zel'dovich number (β) is a non-dimensional quantity that appears as an exponent in the Arrhenius equation. This number gives a measure of the activation energy for the chemical reaction [14] and is given by equation 3.6; where, E_a is the activation energy for the chemical reaction, R is the universal gas constant, T_b and T_u are the burnt and unburnt gas temperatures. The term $(T_b - T_u)/T_b$ is called the heat release parameter, α.

$$\beta = \frac{E_a}{\mathrm{R}T_b}\frac{(T_b - T_u)}{T_b} = \frac{E_a\alpha}{RT_b} \tag{3.6}$$

For very large values of activation energy (i.e. $\beta \to \infty$) and non-unity value of Lewis number, the asymptotic value of the Markstein number of the unburnt gas (Ma_u) can be given by equation 3.7 , where, Le_u is the Lewis number of the unburnt gas, α the heat release parameter and β the Zel'dovich number [15].

$$Ma_u = \frac{1}{\alpha}\ln\frac{1}{1-\alpha} + \beta\frac{Le_u - 1}{2}\cdot\frac{1-\alpha}{\alpha}\int_0^{\alpha/(1-\alpha)}\frac{\ln(1+x)}{x}dx \tag{3.7}$$

The other commonly used number is Damköhler number Da equation 3.8 which is the ratio of flow time scale τ_f to combustion time scale τ_c. If $Da \gg 1$, the reaction is fast compared to the flow and the reaction is slow if $Da \ll 1$ [14, 16].

$$Da = \frac{\tau_f}{\tau_c} \tag{3.8}$$

The Reynolds number (Re) is defined as the ratio of inertial forces to the viscous forces in the flow and is generally used to determine the flow regime, i.e., laminar, or turbulent [14, 17]. The mathematical expression for this number is shown in equation 3.9 , where u is flow velocity, L_r is characteristic length and v is the kinematic viscosity.

$$Re = \frac{uL_r}{v} \tag{3.9}$$

3.1.2 Basic thermodynamics:

In this section the fundamentals of thermodynamics are discussed in the context of an ideal multi-component gas undergoing a chemical reaction. At the outset, the molar heat capacity at constant pressure (C_p) for an ideal gas is assumed to be purely a function of the temperature of a gas (T). The expression for specific heat at constant pressure as shown in equation 3.10 , where R is the universal gas constant [14].

$$C_p = R\sum_{n=}^{N} a_n T^{n-1} \tag{3.10}$$

The molar enthalpy of the gas is then given by equation 3.11 , which on integration becomes equation 3.12

$$H = \int_0^T C_P\, dT \tag{3.11}$$

$$H = R\sum_{n=1}^{N} \frac{a_n T^n}{n} \tag{3.12}$$

The entropy (S) of the system is subsequently given by equation 3.13 , where S_o is the standard molar entropy at temperature T_o

$$\begin{aligned} S &= S_o + \int_{T_0}^{T} \frac{C_P}{T}\, dT \\ &= S_o + R\left[a_1 \ln T + \sum_{n=2}^{N} \frac{a_n T(n-1)}{n-1}\right]\Bigg|_{T_0}^{T} \end{aligned} \tag{3.13}$$

The internal energy (U) of the system is given by equation 3.14 and the Gibb's free energy (G) by equation 3.15 . Finally, the Helmholtz free energy (A) can be obtained as shown in equation 3.16.

$$U = H - RT \tag{3.14}$$

$$U = H - TS \tag{3.15}$$

$$A = U - TS \tag{3.16}$$

Since the process of combustion involves several gases with different molar heat capacities, it is necessary to have a mixture averaged molar specific heat. This is obtained from equation 3.17, where X_k is the mole fraction of the $k-th$ species and K is the total number of species. The respective specific heat capacity of the $k-th$ component is obtained by dividing the corresponding molar heat capacity (C_{pk}) by its molecular weight (W_k) (i.e., $c_{pk} = C_{pk}/W_k$).

$$\bar{C}_p = \sum_{k=1}^{K} C_{pk} X_k \tag{3.17}$$

While defining the entropy, Gibb's free energy and Helmholtz free energies of the mixture the corresponding pressures and mixing entropies need to be considered, which then renders different equation for entropy as shown in equation 3.18 for the $k-th$ component. The term P_{atm} refers to the atmospheric pressure and the term, S_k^o is the standard molar entropy of $k-th$ component. Equation 3.19 gives the equation for the mixture averaged entropy, equation 3.20 , the equation of mixture averaged Gibb's free energy ($\bar{G}$) and equation 3.21, the equation of mixture averaged Helmholtz free energy ($\bar{A}$) [14, 16].

$$S_k = S_k^o - R \ln X_k - R \ln\left(\frac{P}{P_{atm}}\right) \tag{3.18}$$

$$\bar{S} = \sum_{k=1}^{K} \left[S_k^o - R \ln X_k - R \ln\left(\frac{P}{P_{atm}}\right)\right] X_k \tag{3.19}$$

$$\bar{G} = \sum_{k=1}^{K} \left[H_k - T_K \left(S_k^o - R \ln X_k - R \ln\left(\frac{P}{P_{atm}}\right)\right)\right] X_k \tag{3.20}$$

$$\bar{A} = \sum_{k=1}^{K} \left[U_k - T_K \left(S_k^o - R \ln X_k - R \ln\left(\frac{P}{P_{atm}}\right)\right)\right] X_k \tag{3.21}$$

3.1.3 Equation of state:

For an ideal gas with K components, the relation between the system pressure (P), density (ρ) and temperature (T) can be obtained from the ideal gas equation shown in equation 3.22. The term W_k is the molecular weight of the $k-th$ component and c_k is the molar concentration (i.e., number of moles per unit volume) [14, 16].

$$\begin{aligned} P &= RT \sum_{k=1}^{K} [c_k] \\ \rho &= \sum_{k=1}^{K} [c_k] W_k \end{aligned} \tag{3.22}$$

The mean molecular weight ($\overline{W}$) of the mixture can be written as shown in equation 3.23, where X_k is the mole fraction, Y_k the mass fraction, c_k the molar concentration and W_k the molecular weight.

$$\bar{W} = \sum_{k=1}^{K} X_k W_k = \frac{1}{\sum_{k=1}^{K} \frac{Y_k}{W_K}} = \frac{\sum_{k=1}^{K} c_k W_k}{\sum_{k=1}^{K} c_k} \tag{3.23}$$

3.1.4 Reaction rates and stoichiometry:

The burning velocity or the flame speed in premixed laminar flames is significantly affected by the reaction rates [14]. This section provides a brief discussion on the equations governing the reaction rates. equation 3.24 shows a system of I reversible chemical reactions with the i_{th} reaction's stoichiometric coefficients of reactants and products given by ν'_{ki} and ν''_{ki} respectively. The reactants (and products) themselves are given by R_k.

$$\sum_{k=1}^{K} \nu'_{ki} R_k \rightleftharpoons \sum_{k=1}^{K} \nu''_{ki} R_k \tag{3.24}$$

The rate of production ($\dot{\omega}_k$) of k^{th} species can be given by equation 3.25 , where q_i is the progress rate variable for the i^{th} reaction given by equation 3.26. The reaction constant for the i^{th} forward (k_{fi}) reaction is obtained from the Arrhenius equation given by equation 3.27. The term β_i in this equation is the temperature exponent (not the same as Zel'dovich number), the term A_i is the scaling pre-exponential factor, R, the universal gas constant and E_i the activation energy for the reaction [14, 16].

$$\omega_k = \sum_{\iota=1}^{\dot{I}} \nu_{k\iota} q_\iota \tag{3.25}$$

$$\forall\, k \in [1, K], \qquad where\ \nu_{ki} = \nu''_{ki} - \nu'_{ki}$$

$$q_i = k_{fi} \prod_{k=1}^{K} (c_k)^{\nu'_{ki}} - k_{ri} \prod_{k=1}^{K} (c_k)^{\nu''_{ki}} \tag{3.26}$$

$$k_{fi} = A_i T^{\beta i} exp\left(\frac{-E_i}{RT}\right) \tag{3.27}$$

While the forward rate constant is obtained from the Arrhenius equation, the backward rate constant can be calculated from the relation between the forward (k_{fi}), backward (k_{ri}) and equilibrium constant (k_i), shown in equation 3.28 .

$$k_{ri} = \frac{k_{fi}}{k_i} \tag{3.28}$$

The equilibrium constant is further calculated with the help of the change in the values of $\Delta S_i^o/R$ and $\Delta H_i^o/RT$ as shown in equation 3.29 .

$$k_{ci} = \exp\left(\frac{\Delta S_i^o}{R} - \frac{\Delta H_i^o}{RT}\right)\left(\frac{P_{atm}}{RT}\right)^{\sum_{k=1}^{K} v_{ki}}$$

where:

$$\frac{\Delta S_i^o}{R} = \sum_{k=1}^{K} v_{ki}\frac{S_k^o}{R} \qquad (3.29)$$

and

$$\frac{\Delta H_i^o}{RT} = \sum_{k=1}^{K} v_{ki}\frac{H_k^o}{RT}$$

3.1.5 Fundamentals of fluid dynamics:

The general equations of a fluid in the state of motion are obtained from conservation of mass, momentum, and energy. However, if the fluid under consideration is undergoing a chemical reaction, the conservation of molecular species needs to be included in the set of governing equations [17].

The equation of mass conservation for a flow with a variable density in three dimensions (Cartesian coordinate system) is given by equation 3.30. The equation is written in tensor notations where a repeated index indicates summation and u_i (where $i = 1,2,3$) is the component of flow velocity in *x*, *y* and *z* directions respectively.

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_i}{\partial x_i} = 0 \qquad (3.30)$$

The equation for momentum conservation for a compressible fluid follows from the Newtons second law and is given by equation 3.31. The term Y_k represents the mass fraction of k^{th} species and f_k represents the component of body force on the k^{th} species. The term μ' is the bulk viscosity (which is zero for monoatomic gases and non-zero for polyatomic gases). However, the value of μ' has little significance for combustion and can be assumed as zero [17]. The term μ is the dynamic viscosity and the term δ_{ij} is the Kronecker delta.

$$\frac{\partial \rho u_i}{\partial t} + u_j\frac{\partial \rho u_i}{\partial x_j} = \rho\left(\sum_{k=1}^{K} Y_k f_k\right)_i - \frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left[\left(\mu' - \frac{2}{3}\mu\right)\frac{\partial u_k}{\partial x_k}\delta_{ij} + \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right]$$

$$\delta_{ij} = \begin{cases} 1, if\ i = j \\ 0, if\ i \neq j \end{cases} \qquad (3.31)$$

The equation for energy conservation for a compressible fluid is quite complex and involves the heat conduction ($\vec{q}_{cond}$) which follows the Fourier's law, in addition to inter-diffusion ($\vec{q}_{id}$) and Dufour effect ($\vec{q}_{du}$)). Inter-diffusion occurs in a multicomponent system when the average velocity (u_k) of the component k is different from the mass-averaged velocity of the system

(U_k). The Dufour effect is the reciprocal effect of the thermal diffusion (or Soret effect). The Soret effect states that the temperature gradient (∇T) drives the diffusion velocities, whereas the Dufour effect is the heat flux that is generated by the concentration gradient (∇X_k) [17]. Thus, the net heat flux into the control volume is given by equation 3.32. Where λ is the thermal conductivity, α_k is the thermal diffusivity for the k^{th} species, D_{kl} is the binary diffusivity of the $k-l$ species system, W_k is the molecular weight of the k^{th} species and R is the universal gas constant. Although the Dufour effect is small enough to be neglected [17], it is included here for the completeness of the equation.

$$\vec{q}_{net} = \vec{q}_{cond} + \vec{q}_{id} + \vec{q}_{du}$$

$$= -\lambda \nabla T + \rho \sum_{k=1}^{K} h_k Y_k u_k + RT \sum_{k=1}^{K} \sum_{l=1}^{K} \left(\frac{X_l \alpha_k}{W_k D_{kl}} \right) (u_k - u_l) \tag{3.32}$$

The overall equation for the energy conservation including the mechanical work done, influx of kinetic and internal energy by convection, heat addition due to heat flux and heat added from an external source ($\dot{Q}$), is given by equation 3.33. Note that, the bulk viscosity (μ') is assumed to be zero, the dot (.) represents the dot product, the term D/Dt represents the material derivative given by$\left(D/Dt = (\partial/\partial t) + u_j\left(\partial/\partial x_j\right)\right)$and the term h represents enthalpy of the system given by $h = \int_0^T C_p dT$.

$$\rho \frac{D_{ht}}{Dt} - \frac{\partial p}{\partial t} = \frac{\partial \left(u_i \tau_{ji}\right)}{\partial x_j} + \dot{Q} - \nabla . \vec{q}_{net} + \rho \sum_{k=1}^{K} Y_k f_k . (u_k + U_k)$$

where:

$$h_t = h + \frac{u_i u_i}{2}$$

$$\tau_{ji} = -\frac{2}{3} \mu \frac{\partial u_k}{\partial x_k} \delta_{ji} + \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \tag{3.33}$$

To simplify the equations sometimes the Fick's law of diffusion may be used. The mathematical expression is given by equation 3.34, where k^{th} species mass flux component along x_i is given by J_{ki}, diffusivity by D_k and concentration X, Y by ϕ_k. However, this simplification does not yield very accurate results as the thermal diffusion plays an important role in the transport of species [17].

$$J_{ki} = -D_k \frac{\partial \phi_k}{\partial x_i} \tag{3.34}$$

3.2 Flame theory:

The process of combustion is essentially a chemical reaction consisting of several intermediate reactions and their respective components. The visible part of the combustion reaction is referred to as the flame. In general, the flames can be classified into several different ways. Depending on the type of flow – laminar or turbulent flames. Depending on the type of air fuel mixture supplied – premixed or diffusion flames. Depending on the composition of the air fuel mixture – lean or rich flames. Depending on how the flame propagates inside the control volume – deflagration or detonation flames. Depending on the type of flame stabilization geometry – flat, spherical or counterflow flames [14, 17].

As illustrated in many of the popular book references [14, 17, 18], a flame consists of two main zones; the preheat zone and the reaction zone. The preheat zone is where the unburnt gases propagate and get thermally excited from the energy released in the reaction zone. The reaction zone is where the process of combustion occurs. The process of combustion reaction is highly sensitive to temperature [17, 18]. Based on the condition of mixing of the fuel and the oxidizer, the type of flame generated can be classified into two categories: premixed flames and non-premixed flames. The premixed flames are generated because of combustion of a homogeneous mixture of fuel and oxidizer whereas the non-premixed flames are the flame generated when fuel and oxidant are separated [18]. In this study, since the focus is on the premixed mixtures and their combustion, the rest of the content in this thesis is solely relevant to premixed flames.

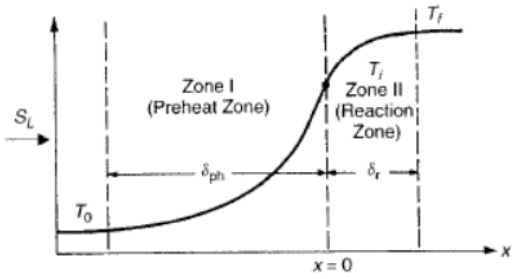

Figure 3.2.1.1 Temperature vs. flame length showing preheat zone and reaction zone [17].

As described above, depending on the type of flame structure, the flames can be classified into laminar and turbulent flames. The first part of this study section 5.1 is mainly focusing on the behaviour of laminar flames, where the flow upstream of the flame is irrotational. The turbulent flames, which are completely opposite to laminar flames are assumed to have several stretched/strained laminar flame pockets within the large turbulent flame domain according to many theories. The second part of this study section 5.2 is mainly focusing on the behaviour of such strained/stretched laminar flames. Reynolds number helps to determine if the flow is laminar, turbulent, or transitional. The flow is laminar if the Reynolds number is too low, and it is high for turbulent fluid flow domains. There exists a critical Reynolds number for certain boundary conditions at which the flow is transitional from laminar to turbulent flow domain [18].

3.2.1 Flame stretch:

The fuel entering the combustion chamber is non-uniform due to viscosity in the flow. Hence the flow field is curving and straining causing changes in the frontal area of the flame. The hydrodynamic stretching and preferential diffusion have considerable effects on the instability at the flame front [19, 20]. The flame stretch factor ($\mathcal{K}$) can be defined as the fractional rate of change of the flame surface area (A_f) and is given by equation 3.50 .

$$\mathcal{K} = \frac{1}{A_f}\frac{dA_f}{dt} \tag{3.50}$$

The flame stretch acts to decrease the flame thickness and influences the flame speed and structure as it is coupled with mass and heat diffusion. The relationship between the flame stretch, strain rate, volumetric expansion or dilation and the flame curvature is given by equation 3.51. The term R is the local equivalent radius of curvature of the flame and $n_i, n_j (i, j \in \{1,2,3\})$ are the *x*, *y* and *z* components of unit vector normal to the flame surface. The first term on the right-hand side of the equation is the term associated with the strain rate tensor, the second term is associated with dilation and the third with flame curvature [14, 16, 17].

$$\mathcal{K} = -n_i n_j \frac{\partial u_i}{\partial x_j} + \nabla.\vec{u} + \frac{S_L}{R} \tag{3.51}$$

3.2.2 Flammability limits:

The stability of the flame depends on several factors, but mainly the velocity of the fuel-air mixture through the nozzle and the equivalence ratio (ϕ). The equivalence ratio of the air-fuel mixture plays a vital role in determining whether the flame can be sustained or not. For very low FAR, the content of fuel is too little to sustain combustion and for very high FAR the oxidizer is too little. Hence there exists limits on ϕ beyond which combustion doesn't occur. For Hydrogen-air mixtures, this limit is given as [4%, 29.3%] by volume [17].

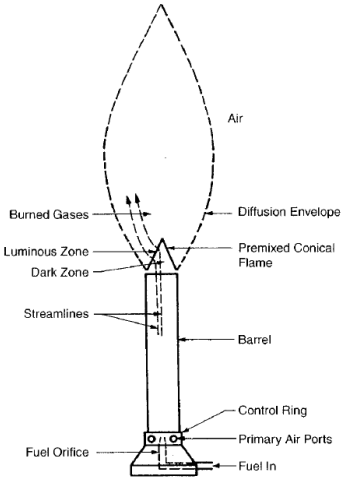

Figure 3.2.2.1 A schematic of the laminar flame in a Bunsen burner [17]

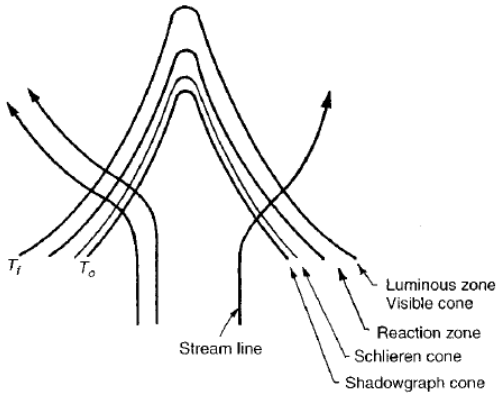

Figure 3.2.2.2 A close-up view of the mouth of the burner with representation of zones [17].

The supply velocity also has an impact on the flame stability. When the supply velocity of the fuel-air mixtures is lower than flame speed (i.e., $u_{supply} < S_L$), the flame propagates through the nozzle causing flashback. When the supply velocity is much higher (i.e., $u_{supply} > S_L$), then flame blows out. The velocity at which the flame blows out is called the extinction velocity. The flame extinction is affected by Lewis number of the deficient reactant (Le_d). If Le_d is larger than a critical value, then the flame stretch causes the extinction, whereas, if the Le_d is lower than the critical value then the extinction is caused due to incomplete combustion and flame stretch [21, 22].

3.2.3 Strained laminar flames

In this part, the behaviour and the key characteristics of the strained laminar flames have been illustrated in the context of hydrogen-air premixed flames. Hydrogen-air premixed strained laminar flames allow us to study a behaviour of hydrogen air flame configuration under the controlled aerodynamic stretch. Often, to study the combustion characteristic of such flames it is assumed that the flame exists at the stagnation plane at the centre with two opposing flow setups with two opposed flow nozzles injecting a premixed hydrogen air mixture in a controlled manner. This configuration introduces a symmetric strain field that deforms the flame and allows detailed observation of flame response to varying strain rates [16].

Equation 3.52 explains the effect of the strain rate a , is a key controlling parameter that affects the flame characteristics, defined by the inlet velocities and the nozzle separation distance L_n where and $U_1$and U_2 are the velocities of the opposing streams. As the strain rate increases, the flame is compressed toward the stagnation plane, forming a flame structure, with peak adiabatic temperature, and a reaction zone with certain thickness and characteristics which mainly depend on inlet hydrogen air fuel configuration [23].

$$a = \frac{U_1 + U_2}{L_n} \tag{3.52}$$

Hydrogen can be considered as one of the carbon free fuels which exhibits a high laminar flame speed S_L, with its low ignition energy and the wide flammability limits. Hydrogen-air mixtures can sustain combustion over a broad range of equivalence ratios and its excellent diffusion properties makes it one of the most special fuels. The Lewis number Le for hydrogen is significantly less than one, indicating that mass diffusivity exceeds thermal diffusivity. The high diffusion ability of hydrogen contributes to a significant reaction zone and higher flame stability even under lean conditions [24].

The consumption velocity S_c, which quantifies the rate at which the unburnt gases are consumed due to the flame reactions, is commonly defined as $S_c = \frac{\dot{m}''}{\rho_u}$: where $\dot{m}''$ is the mass flux rate of reactants and ρ_u is the density of the unburned mixture. In planar unstrained flames, the S_c is equal to the laminar flame speed S_L. However, in strained flames, S_c can vary significantly due to transport and other thermodynamic effects due to increase in strain rate [4]. A more detailed definition based on the heat release integral is illustrated in Equation 3.53.

$$S_c = \frac{\int_{-\infty}^{\infty} \dot{q}\, dx}{\rho_u (h_{sens,\infty} - h_{sens,-\infty})} \tag{3.53}$$

where,

$S_c \rightarrow Consumption\ velocity\ in\ m/s$

$\dot{q} \rightarrow Volumetric\ heat\ release\ rate\ in\ W/m^3\ or\ J/m^3/s$

$dx \rightarrow Distance\ in\ m$

$\rho_u \rightarrow Unburnt\ gas\ density\ in\ Kg/m^3$

$h_{sens} \rightarrow Sensible\ enthalpy\ J/kg$

This expression calculates S_c from the integrated heat release rate $\dot{q}$, normalized by the enthalpy change h_{sens} and unburned gas density ρ_u. It directly links the flame's thermal output to its reactive propagation capability. In case of strained flames, the parameter consumption velocity S_c becomes more relevant than flame speed as the consumption velocity captures the global burning behaviour which is more relevant for non-uniform stretched flames unlike laminar flames where the flame is planar and flame speed is the speed of the flame normal to itself [4].

As strain increases, the local transport properties increase, the steeper the gradients in temperature and the species concentration will become, further increasing the local rate of reaction, ultimately reaching its peak value. Beyond this point, the increase in strain leads to the supressed flame and chemical activity, leading to an extinction. The behaviour of S_c versus

strain rate is key to assessing flame behaviour in the systems make use of hydrogen as the primary fuel [25].

A widely used configuration to study this behaviour is the axisymmetric opposed-flow burner**.** The setup is like asymmetric opposed flow burner configuration as shown in figure 3.2.3.1 except where the flame is stabilized, which is slightly above or below the mid plane due to varying nozzle velocities. In axisymmetric setup, identical nozzles supply premixed hydrogen-air mixtures from opposite ends toward a central stagnation plane. The flame is stabilized at this plane where the momentum from both jets' balances. The domain is modelled as axisymmetric, with the axial direction representing flow and the radial direction capturing symmetry. This configuration is ideal for resolving the strain rate effects on flame structure, peak consumption velocity, and extinction [6], which can be used to predict the behaviour of extremely turbulent hydrogen air mixture, which is the main part of this thesis work.

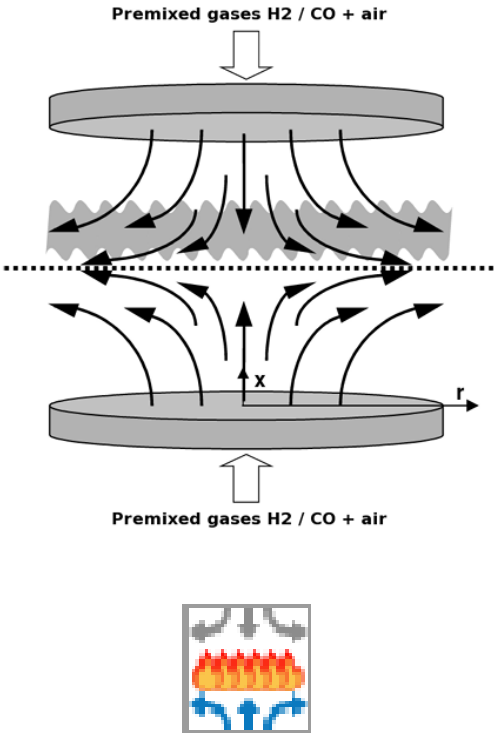

Figure 3.2.3.1: Asymmetric opposed flow burner illustration [6].

Extinction is considered as one of the key phenomena is strained flames, the extinction occurs when the flame cannot sustain the increased strain rate furthermore, the strain rate dominates the existence of flame. The extinction strain rate marks the threshold beyond which the flame quenches due to insufficient residence time for chemical reactions. Mathematically, this corresponds to a sharp drop in T_{max} the maximum adiabatic temperature, S_c consumption velocity and the fuel consumption rate $\dot{\omega}_{H_2}$, given by equation 3.54 where $\nu_{H_2,r}$ is the stoichiometric coefficient of hydrogen in reaction r, and $\dot{\omega}_r$ is the rate of reaction r [26].

$$\dot{\omega}_{H_2} = -\sum_r \nu_{H_2,r}.\dot{\omega}_r \tag{3.54}$$

In real-time simulations, extinction can be defined as a point where the temperature drops sharply below a defined value and the solver fails to converge due to the dominant flame losses. A useful parameter to assess this balance is the Damköhler number Da (equation 3.8). Extinction is typically observed when $Da \ll 1$, indicating that the flow strain dominates the chemical reaction rates [27]. Detailed flame simulations using tools like CHEMKIN-PRO or Cantera solve species, momentum, and energy conservation equations. The energy balance in the flame zone is represented as equation 3.55, where J_k is the diffusive flux of species k, h_k is the specific enthalpy, and $\dot{q}$ is the volumetric heat release rate [6].

$$\rho C_p u \frac{dT}{dx} = -\sum_k h_k \frac{dJ_k}{dx} + \dot{q} \tag{3.55}$$

3.3 Reaction kinetics:

The global reaction of the combustion of hydrogen in the presence of air can be written as equation 3.56. The stochiometric value of the number of moles of air (z_{st}) required for the complete combustion of a moles of H_2 can be determined as $z_{\mathrm{st}} = \mathrm{a}/2$. This value of z_{st} is used in the calculation of FAR_{st} [18].

$$\mathrm{aH_2} + \mathcal{Z}(0.21\mathrm{O_2} + 0.79\mathrm{N_2}) \rightarrow \mathrm{aH_2O} + \mathrm{cN_2} \tag{3.56}$$

3.3.1 Reaction pathways

The above reaction is only a global representation of the overall reaction. The actual reaction pathways are different and highly temperature dependent. The four fundamental types of reactions that define the reaction pathways are: chain initiation, chain branching, chain termination or recombination and chain propagation [18]. These concepts are explained below in the context of hydrogen combustion.

3.3.1.1 Chain initiation:

At the start of the reaction the initial molecules of H_2 and O_2 do not directly react with each other as the probability of breaking the $\mathrm{H-H}$ and $\mathrm{O-O}$ bonds by direct collision are very low. The thermally excited molecules of H_2 and O_2 have a higher probability of collision with other thermally excited molecules (M) present in the flame. This collision results in transient termolecular collision complexes that give rise to free radicals [18]. While there are several free radicals of different elements formed, the main reactions relevant to the combustion process of hydrogen are mentioned in equation 3.57 .

$$\begin{aligned} H_2 + M &\rightarrow H + H + M \\ O_2 + M &\rightarrow O + O + M \end{aligned} \quad (3.57)$$

3.3.1.2 Chain branching:

The branching reactions describes the reaction of these free radicals with the other molecules producing transient combination of different molecules while generating other free radicals [18]. This in the context of hydrogen and air mixtures are shown in equation 3.58 . The growth rate of the radicals is exponential and can be given by 2^N, where N is the number of collisions.

$$\begin{aligned} H_2 + O &\rightarrow OH + H \\ O_2 + H &\rightarrow H + OH \end{aligned} \quad (3.58)$$

3.3.1.3 Chain termination or recombination:

When the concentration of these free radicals is high, they react with themselves forming more stable species [18]. This is shown in equation 3.59

$$\begin{aligned} H + O_2 + M &\rightarrow HO_2 + M \\ O + H + M &\rightarrow OH + M \\ H + OH + M &\rightarrow H_2O + M \end{aligned} \quad (3.59)$$

3.3.1.4 Chain propagation:

The propagation reactions are those where the net number of radicals in the reaction remains unchanged. In other words, the more stable radicals formed in the previous steps react with the thermally excited molecules producing the product and a free radical as shown in equation 3.60. This reaction is responsible for the highest production of the H_2O species [18].

$$H_2 + OH \rightarrow H_2O + H \quad (3.60)$$

4. Methods

Further to different models which predict flame characteristics, the modern-day approach makes use of sophisticated modelling methods to predict the flame characteristics and their spatial derivatives. In this thesis work, CHEMKIN PRO is the only simulation tool used to determine the required parameters. CHEMKIN PRO has a built-in code which is designed based on consideration of all the governing equations and the numerical computation methods, which can accurately predict all the required parameters to solve the flame related problems. There are several standard models which allows the user to select a particular type of problem one wants to solve and depending on the type of mechanism, transport properties and other boundary conditions, CHEMKIN PRO can by itself to generate a required output. Overall, CHEMKIN PRO code provides a solid framework for solving one dimensional, steady state, constant pressure flame problems with its detailed transport and chemical properties [6]. Below are the major governing equations which CHEMKIN PRO embeds in its code to solve the premixed laminar flame problems. If $\dot{m}$ is the mass flow rate flowing through a cross sectional area A, with density ρ and λ is the thermal conductivity of the mixture with $\dot{\omega}_k \; and \; Y_k$ being reaction rate and mass fraction of the species.

Continuity equation:

$$\dot{m} = \rho u A \tag{4.01}$$

Energy equation – including conduction, species diffusion and reaction enthalpy terms:

$$\dot{m}\frac{dT}{dx} - \frac{1}{c_p}\frac{d}{dx}\left(\lambda\frac{dT}{dx}\right) + \frac{A}{c_p}\sum_{k=1}^{N}\rho Y_k V_k c_{pk}\frac{dT}{dx} + \frac{A}{c_p}\sum_{k=1}^{N}\dot{\omega}_k h_k M_{wk} = 0 \tag{4.02}$$

Species conservation equation for each species Y_k:

$$\dot{m}\frac{dY_k}{dx} + \frac{d}{dx}(\rho A Y_k V_k) - A\dot{\omega}_k M_{wk} = 0 \tag{4.03}$$

Equation of state:

$$\rho = \frac{p\bar{M}}{R_u T} \tag{4.04}$$

Using the equations above CHEMKIN PRO applies a finite difference numerical approximations to solve the above differential equations with an initial guess for temperature profiles, species mole/mass fractions and spatial resolution [17].

The spatial temperature gradient is approximated as

$$\dot{m}\frac{dT}{dx} \approx \dot{m}\left[\frac{h_{j-1}}{h_j\left(h_j + h_{j-1}\right)}T_{j+1} + \frac{h_j - h_{j-1}}{h_j h_{j-1}}T_j - \frac{h_j}{h_{j-1}\left(h_j + h_{j-1}\right)}T_{j-1}\right] \quad (4.05)$$

The thermal conduction term:

$$\frac{d}{dx}\left(\lambda A\frac{dT}{dx}\right) \approx \frac{2}{x_{j+1} - x_{j-1}}\left[\lambda A_{j+1/2}\left(\frac{T_{j+1} - T_j}{x_{j+1} - x_j}\right) - \lambda A_{j-1/2}\left(\frac{T_j - T_{j-1}}{x_j - x_{j-1}}\right)\right] \quad (4.06)$$

The diffusion velocity gradient term:

$$\frac{d}{dx}\left(\rho A Y_k V_k\right) \approx \frac{\left(\rho A Y_k V_k\right)_{j+1/2} - \left(\rho A Y_k V_k\right)_{j-1/2}}{x_{j+1/2} - x_{j-1/2}} \quad (4.07)$$

There are mainly 2 steps in CHEMKIN PRO - pre-processing where the flame model for a specific type of problem, chemical kinetics, thermodynamic properties are loaded along with the required boundary conditions. The next step is post processing where the CHEMKIN code solver solves the problem using the complex numerical methods assuming the temperature dependent properties using polynomial coefficient to generate the required flame profiles like temperature, species concentration, density, velocity, etc [6, 17].

Specific to this project, the thesis is mainly divided into 2 major parts 1. Premixed laminar flame simulations and 2. Strained laminar flame simulations, focusing mainly on lean hydrogen air flames and different reaction mechanisms.

The first part, steady state 1D laminar flame simulations is easy to calculate using CHEMKIN PRO. There are already pre-existing models which illustrates different flame related problems, the one used in this project is 'Flame Speed Calculator' which comes in handy to calculate the laminar flame speed and related scalar profiles, which is the main goal of this part. After choosing the required flame model according to the problem, the next step is pre-processing where the user needs to define the thermodynamic properties, transport properties and chemical kinetics. Here, there are several sets of pre-loaded mechanisms already available online, but in this project to compare the effects, there are mainly 4 mechanisms selected which are Konnov [9], Princeton [8], Dublin [7] and Milano [10]. Once the problem is setup, there are several options which give the user to carryout different studies ranging from different temperatures, pressures, and species profiles along with the specific parameter effects. In the first part, a detailed study of flame speed focusing on lean hydrogen flames ranging from 0.36 to 2.8 equivalence ratio is made. Subsequently, the effects of pressure, temperature, diffusion and transport properties, temperature and grid configurations are studied.

The second part, strained laminar flame simulations is a major part of this thesis work which makes use of a OPPDIF flame model available readily in CHEMKIN library which consists of 2 opposed flow burners separated by a fixed distance where the flame is stabilized on a stagnation plane between 2 opposed flow nozzles as illustrated in the figure 3.2.3.1. The flame is assumed to be radially axis symmetric which allows to reduce the complex 2D-3D flow to

1D flow where the fluid properties are the function of axial distance only. Later using the CHEMKIN code solver, this opposed flow configuration is solved to determine species, temperature and velocity profiles [6]. To begin, the input velocity which is usually higher than the laminar flame speed is set keeping the nozzle separation distance constant. The OPPDIF module predicts the subsequent velocity and time step, until the flame extinction is achieved. The next important work which is covered under this study is post processing of results. Though the user can extract and visualize majority of the output data directly through CHEMKIN visualizer, CHEMKIN also allows the user to extract the raw data in different forms which can be further processed to determine the required output parameters. In this thesis, after the simulations, the data from CHEMKIN is saved in .csv and. ckcsv files and then processed in MATLAB to plot the effect of strain rate on consumption velocity calculated using equation 3.53. From the large set of data, a code is written to first extract the required columns and the parameters. Later those columns are processed to obtain a specific output using a separate code (presented at the end in appendix part of the report). The goal is to identify the strain rate associated with the maximum consumption velocity which is normally just before the extinction point which is later presented in the results and discussion section, mainly focusing premixed lean hydrogen air flames. Later the effect of pressure, inlet temperature, and different equivalence ratios are also discussed.

5. Results and discussion

5.1 Premixed laminar flame simulations

5.1.1 Comparison of effect of equivalence ratio on flame speed

Fig 5.1.1 shows the comparison of flame speed with change in equivalence ratio ranging from 0.5 to 2.8. Here, the effects are studied using 3 mechanisms – Konnov, Princeton and Dublin.

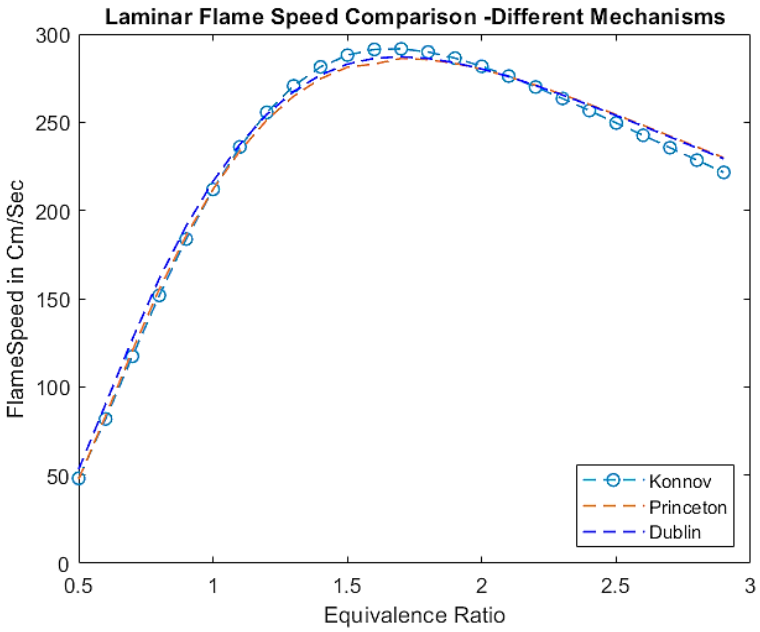

Figure 5.1.1. Effect of equivalence ratio on flame speed.

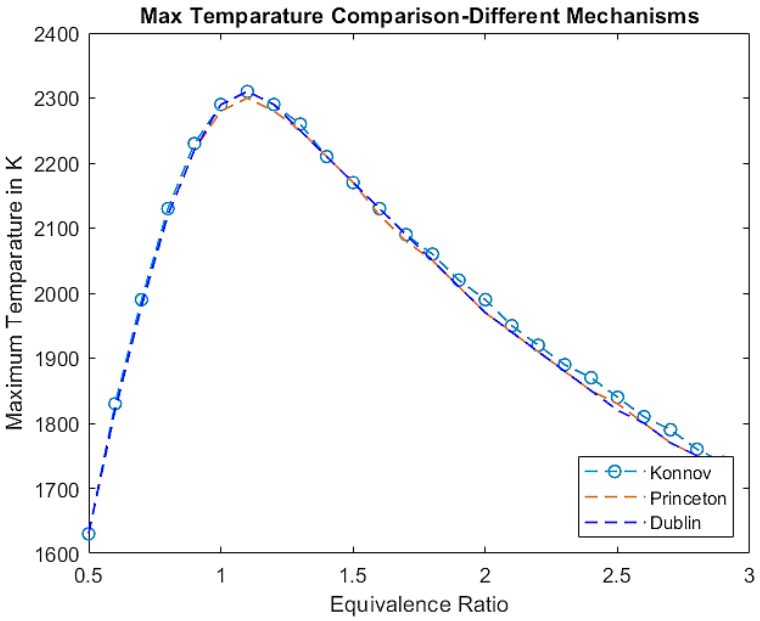

Figure 5.1.2. Effect of equivalence ratio on adiabatic flame temperature.

Here, the standard flame speed simulator is used to calculate the effect of equivalence ratio on flame speed, adiabatic flame temperature and net heat production. To achieve this, CHEMKIN PRO allows the user to setup a parameter study and here different equivalence ratios are entered from 0.5 to 2.8 with a step of 0.05. Upon successful completion of the simulation, CHEMKIN returns the output parameters for each equivalence ratio step, which is further summarized and presented as above using MATLAB. The similar process is repeated for different mechanisms.

From figure 5.1.1, it is visible that, there is almost less to no impact due to the different mechanisms on the flame speed. Initially for the lean conditions from 0.50 to <1.5 equivalence ratio, the laminar flame speed increases as the mixture becomes richer and reaches maximum close to equivalence ratio from 1.5 to <2.0 and then further decreases as the equivalence ratio further increases. But the system attains a maximum adiabatic flame temperature figure 5.1.2 slightly after the stoichiometric condition because this is where the intensity of combustion is much stronger due to the perfect balance in fuel and oxidizer ratio [17]. Under lean conditions, the adiabatic flame temperature and heat release rate are dominated by the excess air, less fuel and higher thermal diffusivity. But as the equivalence ratio further increases, the flame starts to quench due to excessive fuel, oxygen limited conditions and incomplete combustion [17, 18]. The similar trend is also observed when we plot the impact of equivalence ratio on the net heat release rate figure 5.1.3, but here the differences in the mechanisms especially Konnov is quite dominant compared to other 2. This is mainly due to the difference in reaction pathways among those mechanisms and the different transport properties embedded in each mechanisms [16].

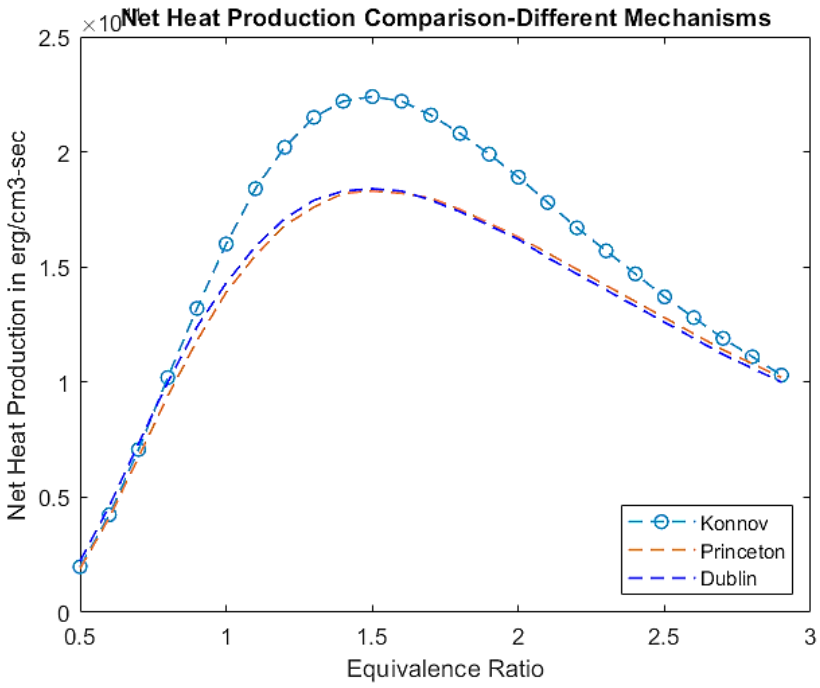

Figure 5.1.3. Effect of equivalence ratio on net heat production.

On the other hand, one can notice very similar trends between figure 5.1.1 and figure 5.1.3. The max flame speed and the net heat release rate is between $\varphi = 1.0$ to Phi $\varphi = 1.6$ whereas, the maximum temperature occurs at just after the stoichiometric condition. This is mainly due

to the relation between net heat release rate, reaction rate of products and the flame speed. The diffusion takes place faster due to steep gradients which results in max heat release and flame speed up to $\varphi \leq 1.6$, whereas the maximum temperature is mainly linked with the thermodynamic properties and independent of chemical reaction. The maximum flame temperature occurs at the equivalence ratio close to stoichiometric condition and that's where the complete combustion happens, and hence higher flame temperature is observed. There is an ideal balance between the fuel and air mixture and there is no heat loss due to the lack of other species to absorb the energy unlike the rich mixtures [16, 17]. As the mixture becomes richer at $\varphi => 1.6$, normally the reaction rate should be higher due to the higher concentration of fuel mixture. But the reaction rate is relatively low due to lower temperature and this effect is more dominating [14, 17].

Also, another interesting observation is that the comparison of flame speed of hydrogen vs. conventional hydrocarbon fuel. Figure 5.1.4 shows the effect of equivalence ratio on flame speed for n-heptane/air mixtures as presented in [28]. Here, the maximum flame speed is very close to the stoichiometric conditions. Whereas in case of hydrogen, as you can see in the figure 5.1.1 , the maximum flame speed is observed at equivalence ratio of 1.5 and is significantly larger. This is mainly due to the unique characteristics of hydrogen fuel compared to hydrocarbon fuels. In case of hydrocarbon fuels, at higher equivalence ratios, the excess fuel slows down the radical formation and the problem of incomplete combustion becomes more dominant with heavier fuel intermediates and lower heat release rate [28]. In case of hydrogen, primarily due to its higher diffusivity property D, the flame speed S_l directly increases as $S_l \propto \sqrt{D}$ [24]. And at fuel rich conditions, hydrogen tends to diffuse more allowing rapid radical formation resulting in higher fuel penetration to the reaction front. This mechanism allows the flame to be sustained even at higher equivalence ratios, unlike hydrocarbon fuels, where fuel rich conditions slow down the overall reaction rate and the flame speed [24].

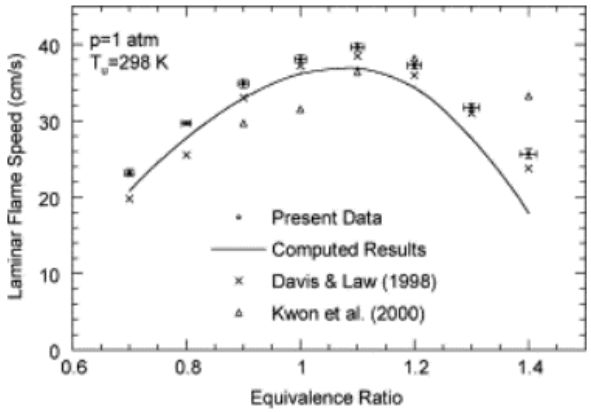

Figure 5.1.4. Effect of equivalence ratio on flame speed for n-heptane/air mixtures [28].

5.1.2 Analysis of combustion products associated at maximum adiabatic flame temperature at $\varphi = 1.1$

This section illustrates the concertation of the main combustion products before, during combustion and after combustion, at the point of equivalence ratio ($\varphi = 1.1$) where the adiabatic flame temperature is maximum, see figure 5.1.2. At this point, all the concentrations

of different radicals and products can be directly obtained by the CHEMKIN output files. These concentrations are plotted along the axial length of the flame, the special domain representing unburnt gases at one and the end products on the other side, which is kept 2 cm.

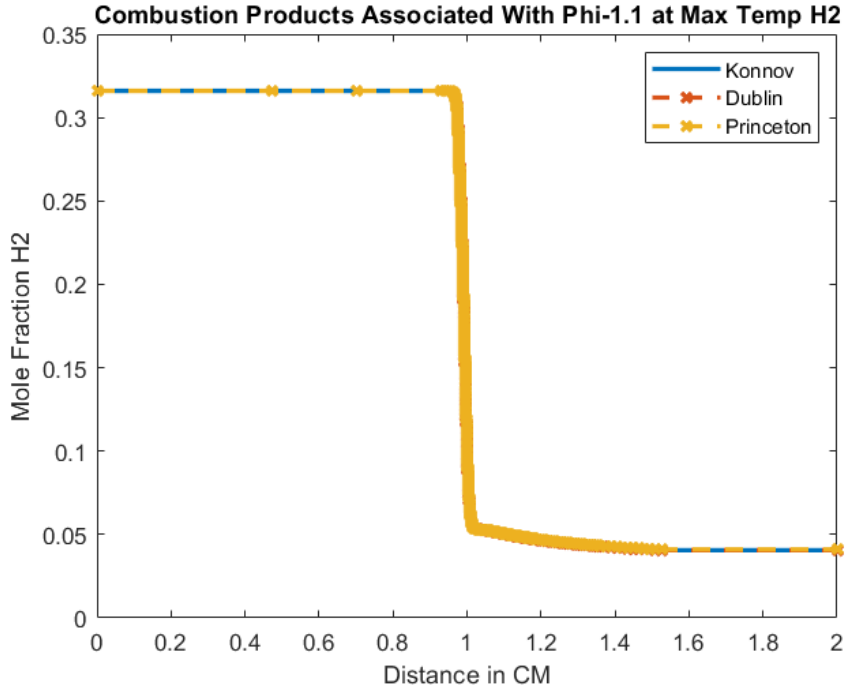

Figure 5.1.5a H2 concertation at $\varphi = 1.1$

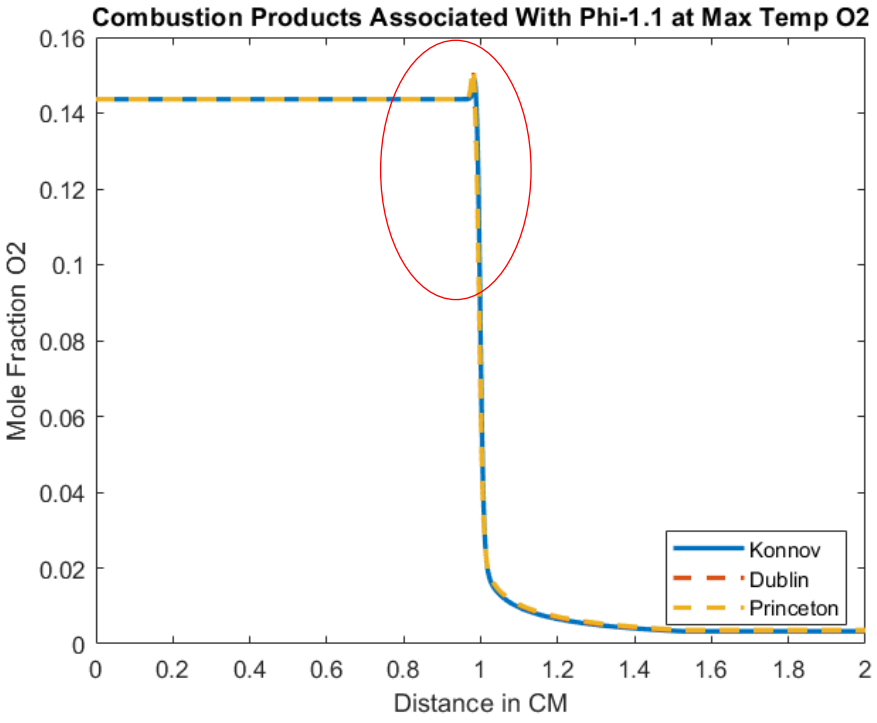

Figure 5.1.5b O2 concertation at $\varphi = 1.1$

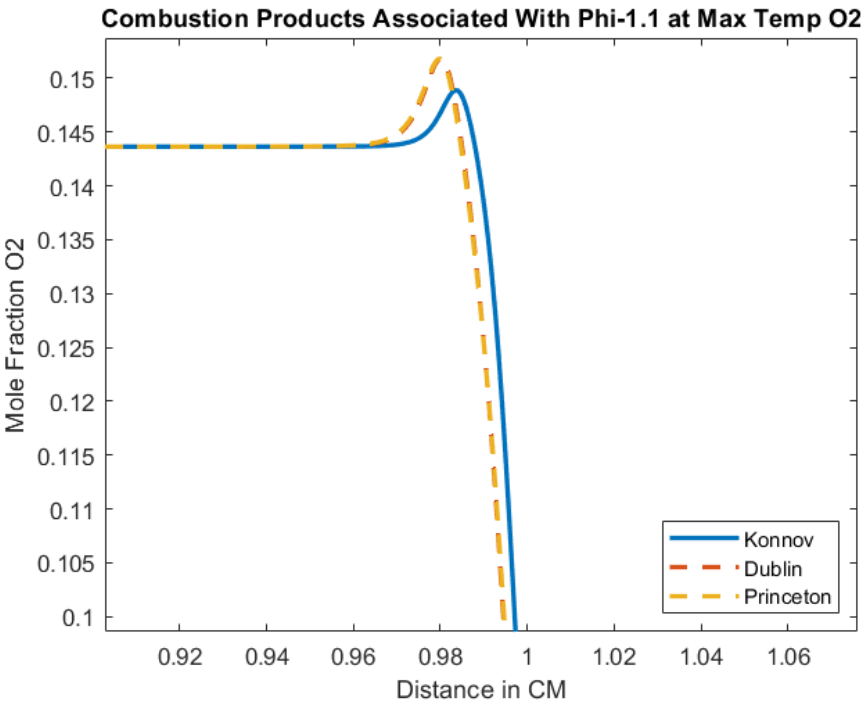

Figure 5.1.5b O2 concertation at $\varphi = 1.1$ (zoomed in)

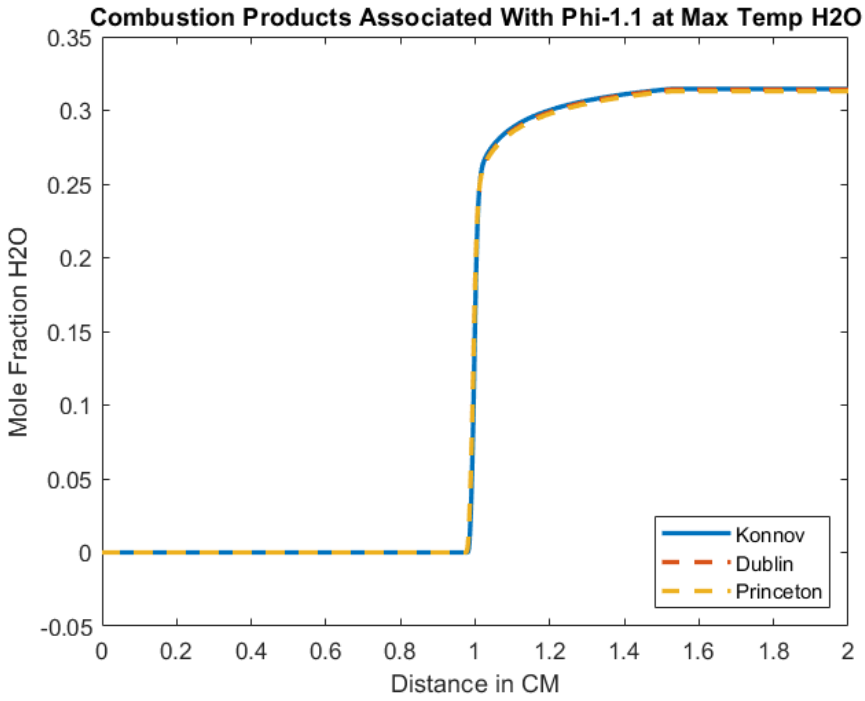

Figure 5.1.6a H2O concertation at $\varphi = 1.1$

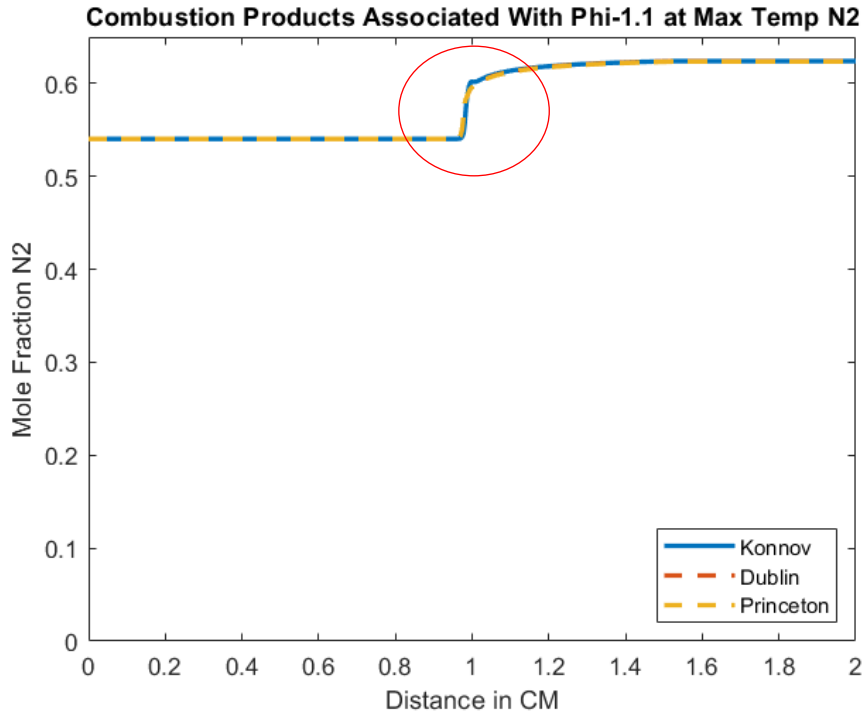

Figure 5.1.6b N2 concertation at $\varphi = 1.1$

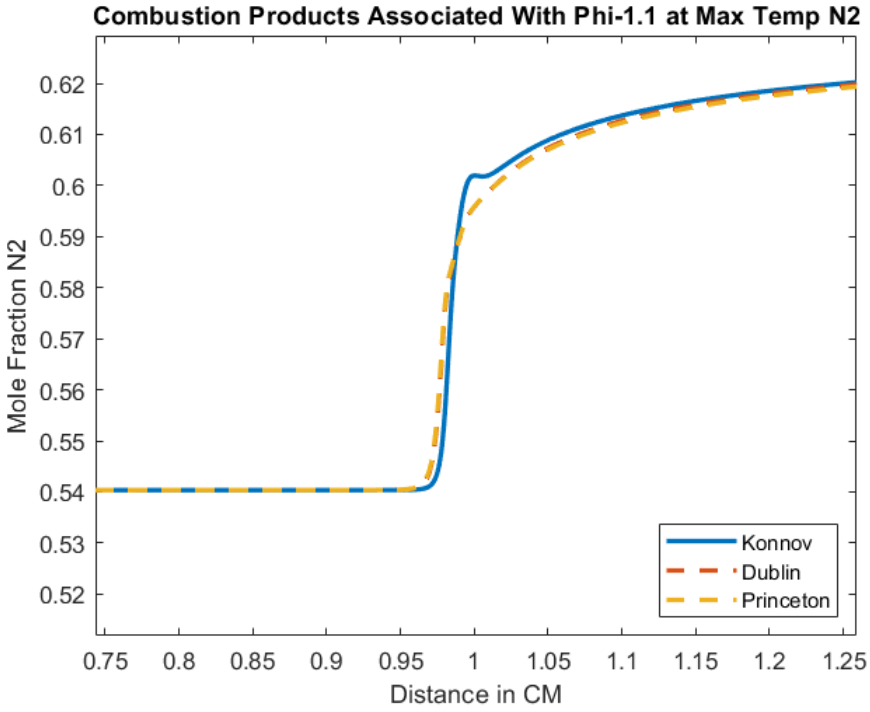

Figure 5.1.6b N2 concertation at $\varphi = 1.1$ (zoomed in)

Figure 5.1.5a and figure 5.1.5b illustrates the concentration of H_2 and O_2 species and figure 5.1.6a and 5.1.6b shows H_2O and N_2 concentrations respectively.

The O_2 mole fraction completely reaches close to zero which shows the signs of the complete combustion and it acts as a limiting reactant which controls the overall combustion [17]. The H_2 mole fraction, drops sharply as soon as it enters the reaction zone but since the mixture is rich $\varphi = 1.1$ and there is no sufficient oxygen to aid the complete combustion, there is still some unburnt hydrogen left behind [17]. The mole fraction of H_2O increases steeply which shows the formation of end products due to the combustion. An increase in the mole fraction of N_2 is very small and is mainly due to decrease in the number of moles of other reactants and it's a relative increase. Overall, N_2 remains totally inert and doesn't really participate in the actual combustion [17]. And all 3 mechanisms show very little to almost no deviation.

5.1.3 Comparison of thermal diffusion and multi component transport effects on flame speed

The graphs below illustrate the effects of thermal diffusion and the multicomponent transport on flame speed, see figure 5.1.7. Adiabatic flame temperature, see figure 5.1.8 and net heat production, see figure 5.1.9. Here, the legend O – indicates that the effect is applied and X – indicates that the effect is not applied.

When thermal diffusion is applied, the fuel tends to diffuse away from the reaction front causing slower reactions and reducing the overall net heat release rate and the flame temperature. But this is important to capture the exact flame speed in fuel rich and lean conditions where diffusion effects are play an important role. If thermal diffusion is not applied, there is a chance of over prediction of the flame speed as no thermal diffusion keeps the radicals to be concentrated at the flame front, which may result in higher flame speeds [16, 17].

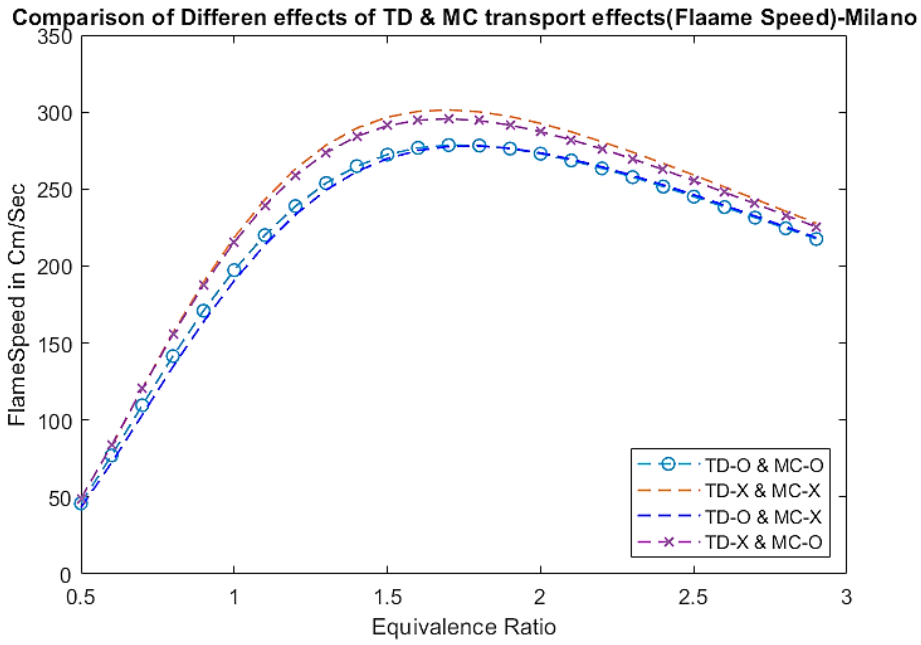

Figure 5.1.7. Flame speed vs. equivalence ratio (effect of TD and MC transport)

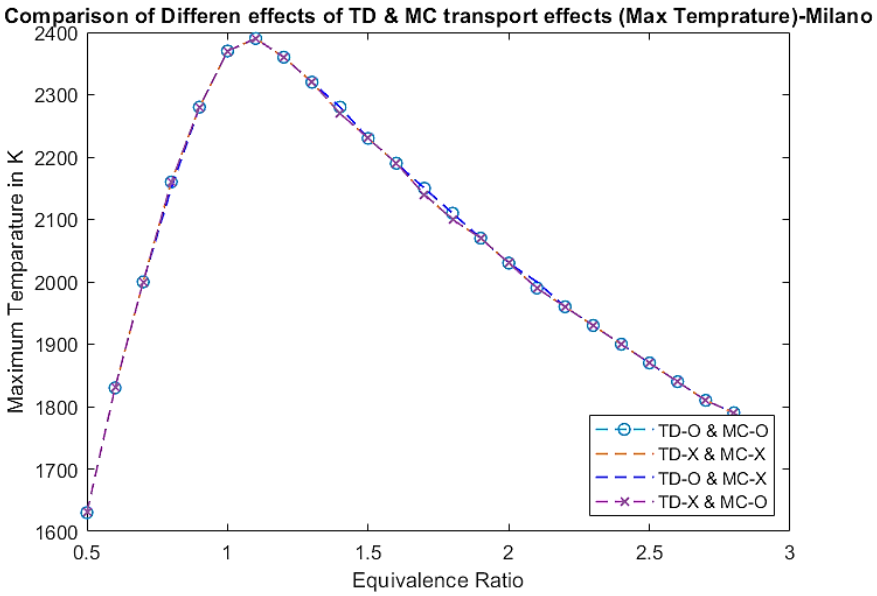

Figure 5.1.8. Adiabatic flame temperature vs. equivalence ratio (effect of TD and MC transport)

Normally, each species diffuse at its own rate and in order to predict the accurate flame speed it's important to capture these cross-species diffusion effects and the species gradients at rich and lean conditions [16, 17]. When multicomponent transport model is not applied, the system doesn't take into consideration of species cross diffusion effects, leading to the overprediction of flame speed. Applying multi component model and thermal diffusion effects in CHEMKIN PRO helps the user to predict accurate flame speed representing the actual conditions reducing under or over estimation of results.

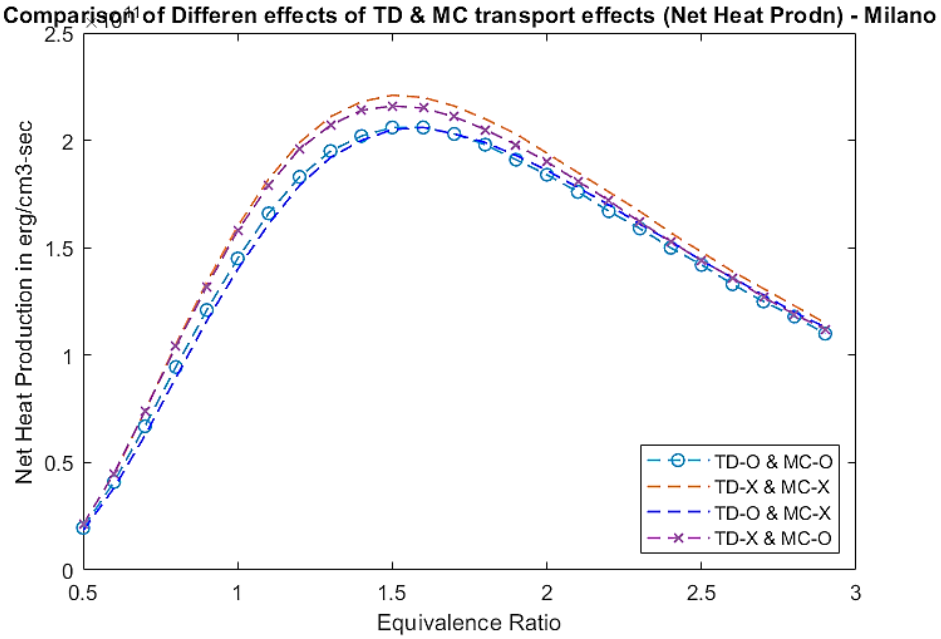

Figure 5.1.9. Net heat production vs. equivalence ratio (effects of TD and MC transport)

TD-X and MC-X represents the condition where the thermal diffusion and multicomponent effects are not taken into consideration, means higher reaction rates due to no diffusion and oversimplifies the system with no cross-diffusion effects taken into consideration too. Due to this, figure 5.1.7 - clearly shows that the flame speed is high in this condition compared to TD-O and MC-O where the flame speed is low compared to other settings, which is in line with the hypothesis described above.

5.1.4 Comparison of effect of pressure on flame speed

Figure 5.2.0 and figure 5.2.1 illustrates the effect of pressure on flame speed and net heat production at $P = 1$ atm,$P = 3$ atm & $P = 5$ atm. As it is evident from the simulation results below, pressure has a significant effect on flame speed. To simulate these conditions, as described in the earlier sections, a parameter setup is set for pressure in CHEMKIN PRO and the outputs are visualized using MATLAB.

The effect of pressure on flame speed and net heat production mainly depends on reaction order and the transport properties [16, 17]. At higher pressures, the chain recombination reactions (third order) are more dominant compared to chain branching reactions (second order), which are directly responsible for producing key radicals like O, H & OH. These radicals can sustain at lower pressures leading to a higher laminar flame speed at lower pressures. At higher pressures, the flame speed decreases due to the dominant recombination of radicals & slow radical build-up [16, 17].

In contrast, even though the flame speed decreases, the net heat release rate per unit volume increases with increase in pressure. As the pressure increases, density and the reactant concentration increase. The local concentration supports faster reaction rates with a shorter chemical time, but overall molecular transport reduces. This results in a thinner flame, leading to an increase in the net heat release rate per unit volume [16, 17]. Hence in the figure below 5.2.0 & 5.2.1, even though the flame speed decreases, the net heat release increases due to increase in pressure.

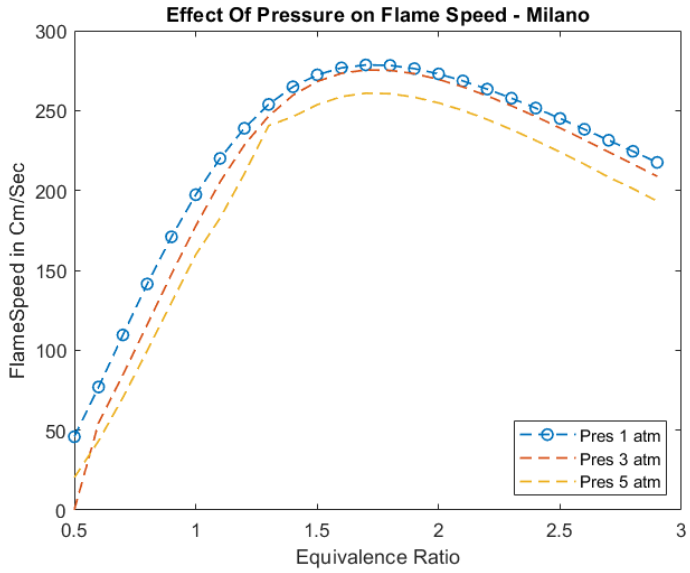

Figure 5.2.0. Flame speed vs. equivalence ratio (effect of pressure)

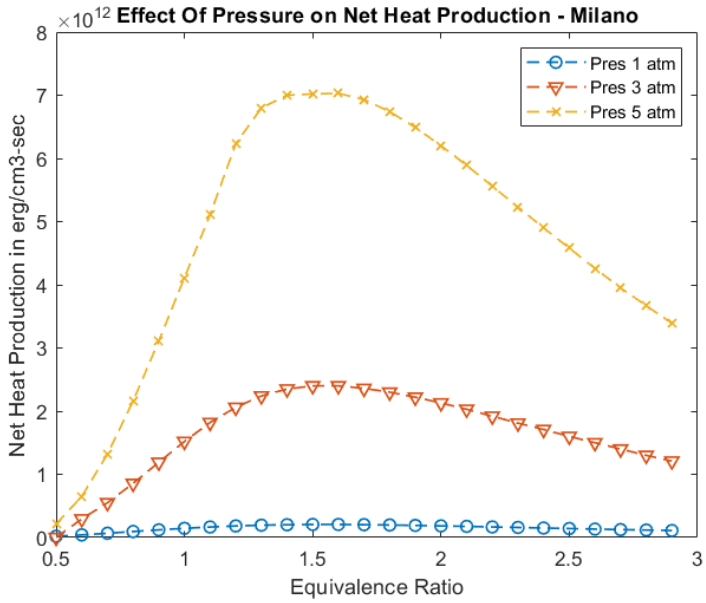

Figure 5.2.1. Net heat production vs. equivalence ratio (effect of pressure)

5.1.5 Comparison of effects of grid resolution on flame speed

Figure 5.2.2 and figure 5.2.3 illustrates the effect of grid resolution, i.e., gradient and curvature of the flame structure on flame speed and net maximum temperature. This is mainly the step size which needs to be use in numerical computational method used in CHEMKIN PRO to predict the flame characteristics. The finer the grid size is, the more accurately it is possible to predict the required parameters efficiently representing the real estimations. GRAD represents the spatial gradient, smaller the GRAD finer the spatial resolution. Finer the spatial resolution, it is possible to predict the flame speed accurately with sharper and steeper temperature and species gradient. If the grid size is larger, it is possible that the system can underestimate flame speed due to smooth GRAD, it could be possible that the system may not be able to capture the peak flame speed values [16, 24].

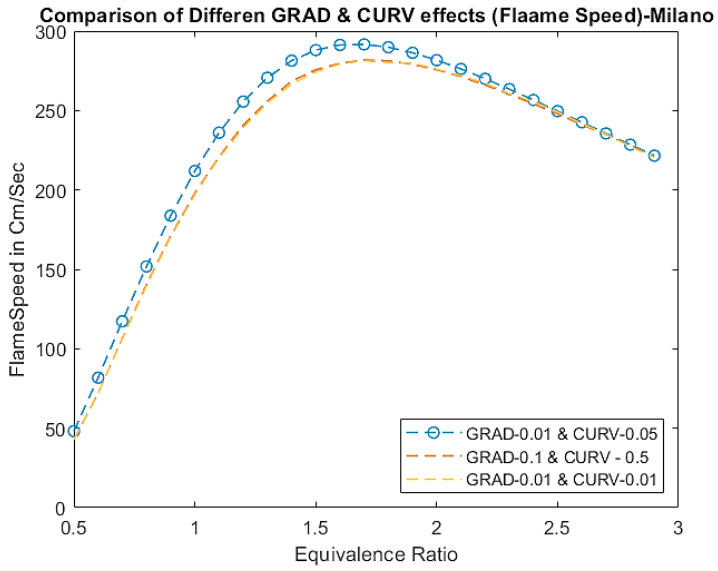

Figure 5.2.2. Flame speed vs. equivalence ratio (effect of grid resolution)

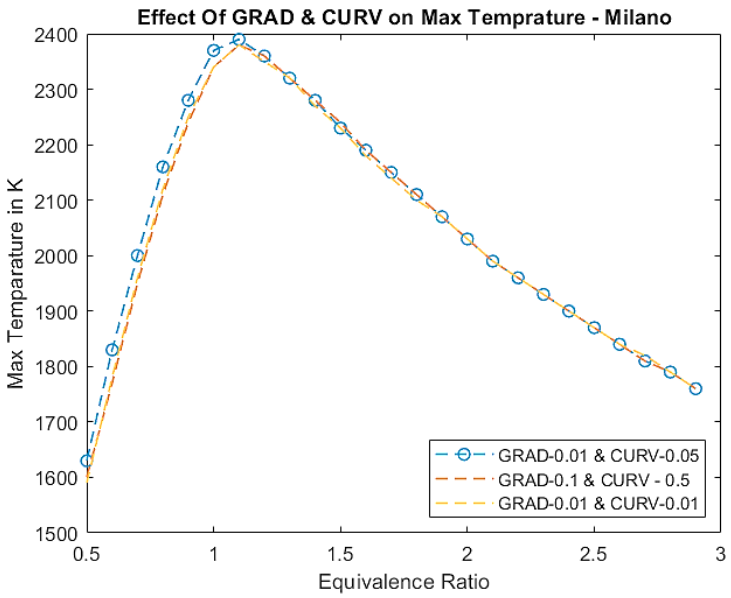

Figure 5.2.3. Flame speed vs. equivalence ratio (effect of grid resolution)

CURV defines the curvature input for a flame. Higher the curvature, more the flame stretch is. This results in reduced local radical concentration and slows the overall chain branching. Hence, the flame speed reduces and vice versa [16, 24]. This is also clearly visible in the figure above, GRAD -0.1 and CURV-0.5 has the lowest prediction in flame speeds compared to other finer grid resolutions.

5.2 Premixed strained laminar flame simulations

Majority of the time in this project was spent on this part of the simulations where the behaviour of the premixed lean hydrogen flames is studied in strained conditions. To do so, Oppdif module in CHEMKIN PRO is selected, which is already readily available in CHEMKIN PRO libraries. This setup exactly represents figure 3.2.3.1, where there are 2 opposed flow burners separated by a fixed distance which is assumed to be radially symmetric. When the premixed hydrogen air flame is supplied through the nozzles of 2 opposing flow burners, the flame is formed exactly at the stagnation plane which is towards the centre of the distance of 2 opposed flow burners.

This setup is used to simulate the effect of stain rate on flame behaviour. In case of strained flames, consumption velocity is more relevant than flame speed as in 1D unstretched laminar flames. The flame is strained and geometrically, consumption velocity represents the real and global burning rate. To illustrate the effects below, after selecting the Oppdif module in CHEMKIN, the input parameters are varied to obtain a specific output parameter. This is repeated for different mechanisms and equivalence ratios. In case of laminar flame speed estimation, user could directly select the equivalence ratio of the mixture. Whereas in case of strained laminar flames using Oppdif module, the equivalence ratio is adjusted by adjusting H_2 and O_2 mole or mass fractions. Accordingly, the simulation is carried for different equivalence ratios. In most of the simulations, GRAD is used as 0.01 and CURV is used as 0.05 with a maximum grid point of at least 100000 and the nozzle separation distance ranging from 0.9 cm to 1.4 cm.

Another important step in strained laminar flames is the post processing of results. CHEMKIN PRO doesn't readily calculate consumption velocity S_c, this must be calculated separately in MATLAB using the output obtained from CHEMKIN PRO. After the simulation is successful, CHEMKIN allows the user to save the data of spatial derivatives of hydrogen like temperature, species concentration, density, velocity, net heat release rate, specific enthalpy etc. using many output data forms, mainly .csv and .ckcsv formats. Using these files, the relevant parameters are extracted, sorted, and used in calculations of consumption velocity S_c using equation 3.53. The detailed MATLAB codes used to illustrate these calculations are attached in the appendix at the end of the report.

5.2.1 Effect of different mechanisms on consumption velocity S_c for $\varphi = 0.50$ and 0.80.

Figure 5.2.4 illustrates the change in consumption velocity and figure 5.2.5 the change in maximum adiabatic temperature with change in strain rate from a completely unstrained condition to the point at which the maximum consumption velocity is achieved and the flame is completely extinguished. For $\varphi = 0.5$, the Oppdif module allows the user to define the number of iteration steps and the conditions for flame quenching. The CHEMKIN solver automatically predicts the subsequent temperature step at which the solution exists until the flame quenching conditions are achieved, which is the point of extinction. The point of extinction is the point at which the flame temperature starts to drop sharply, and the combustion doesn't sustain anymore.

Figure 5.2.4 shows that, an increase in the strain rate increases the consumption velocity. The opposing strong flows aid the combustion by forcing the reactants towards the flame front up to a certain point and at the point of extinction the flame suddenly quenches and resulting in the sharp drop of consumption velocity. The maximum strain rate at which the extinction occurs, and the maximum consumption velocity is attained is called as extinction strain rate. At higher strain rates, the flame becomes thinner and there is a incomplete combustion due to increased heat losses and quenching of radicals [17, 24].

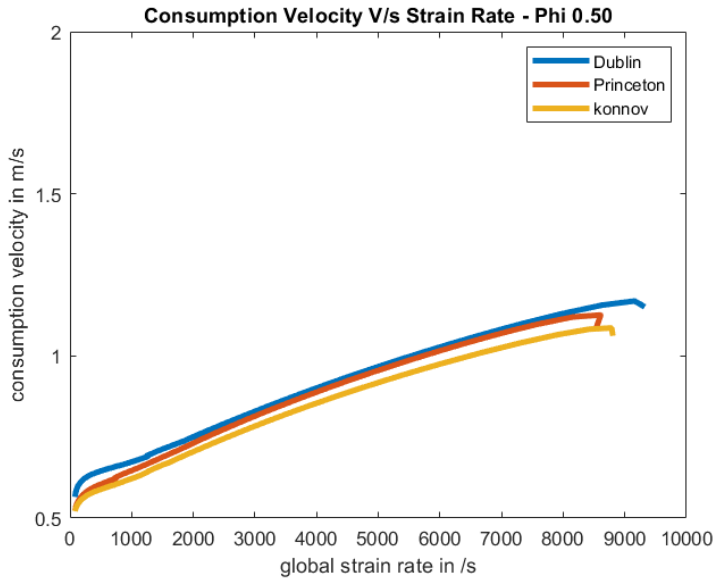

Figure 5.2.4. Consumption velocity and max temperature vs. strain rate $\varphi = 0.50$

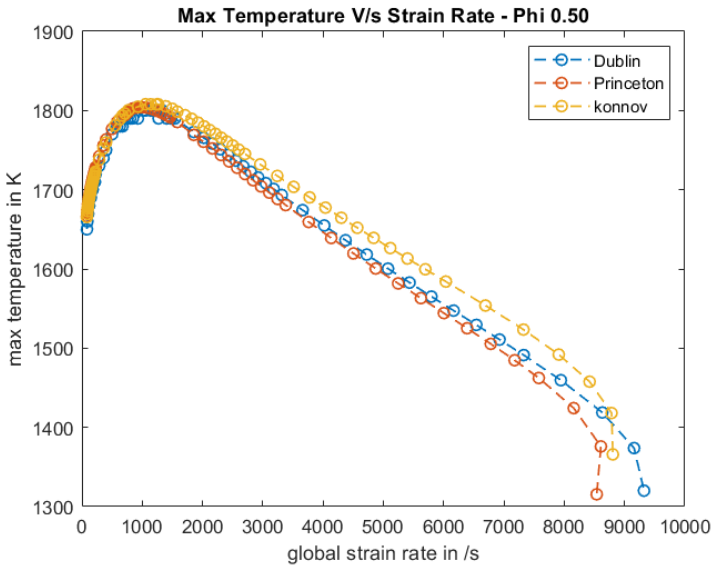

Figure 5.2.5. Max temperature v/s strain rate $\varphi = 0.50$

In figure 5.2.5, hence, the maximum flame temperature tends to increase until strain rate $1000s^{-1}$, the strain aids the complete combustion resulting in higher flame temperatures and drops sharply beyond this. After certain moderate strain rate, the quenching effects starts to grow, thinning the flame, by reducing the radical concentration which brings down the flame temperature [17]. At the extinction strain rate, the flame cannot further sustain and flame completely quenches. It is also evident from the comparison of mechanisms that Dublin mechanism can capture the consumption velocity even at the higher strain rates compared to Princeton and Konnov mechanisms, which tend to be strain sensitive compared to Dublin.

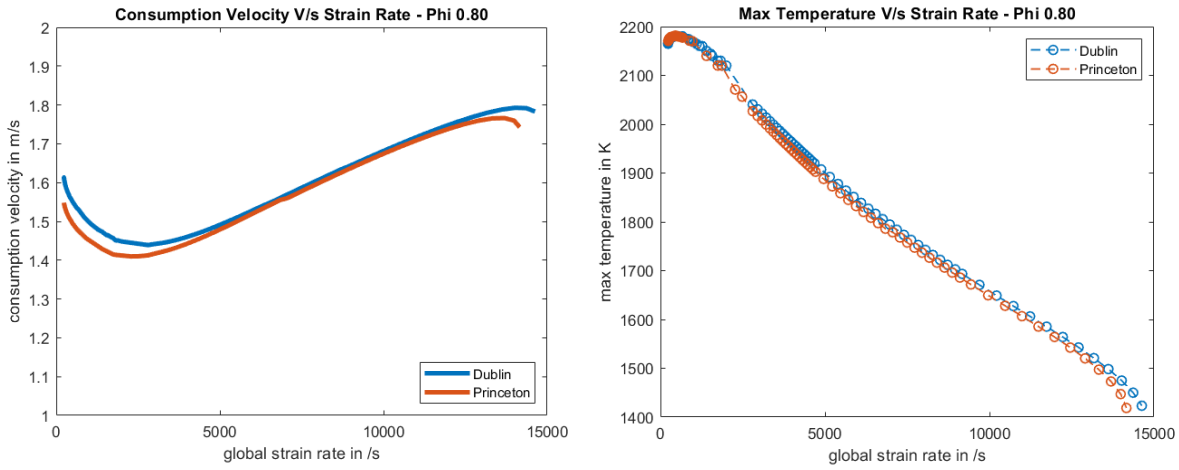

Figure 5.2.6. Consumption velocity and max temperature vs. strain rate

Figure 5.2.6 illustrates the example of consumption velocity and adiabatic temperature vs. strain rate for $\varphi = 0.80$, comparing Dublin and Princeton mechanisms. As the mixture becomes richer, the flame can withstand higher strain rates which is also visible in the figure above.

5.2.2 Comparison of properties at maximum temperature point and at maximum consumption velocity point. $\varphi = 0.50$- Dublin mechanism.

In this section, a comparison of different properties like net heat release rate, temperature, local equivalence ratio, progressive variable C_f, radical and bi products concentration are plotted against the spatial coordinates at the point where there is a maximum consumption velocity. These results are compared with counterpart profiles obtained at the point where the system attains the maximum adiabatic flame temperature for Dublin mechanism and for $\varphi = 0.50$. CHEMKIN PRO automatically saves the result of each iteration while calculating the point of extinction from unstretched condition.

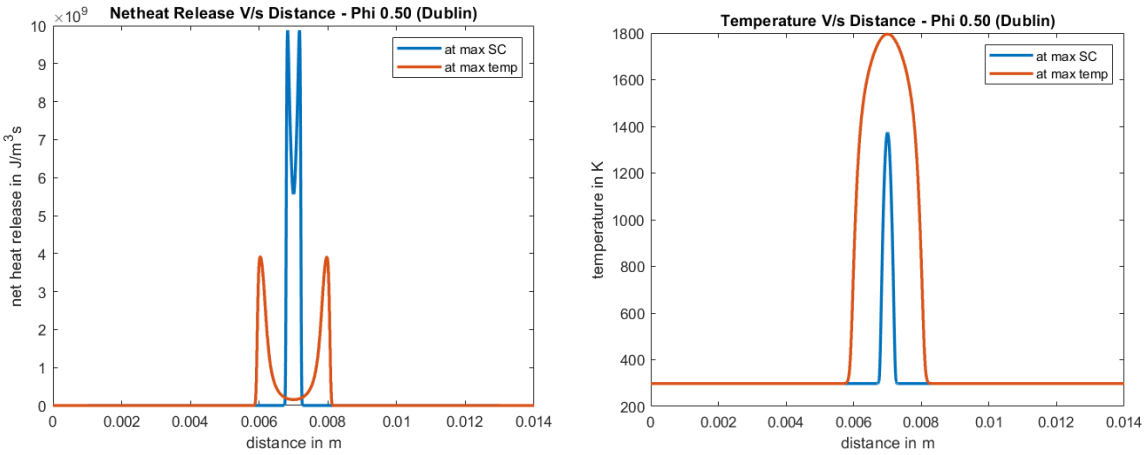

Figure 5.2.7. Net heat release and temperature vs. distance ($T_{max}\ vs. S_c^{max}$)

Here from the figure 5.2.4, the data of the points at which the maximum strain rate/consumption velocity and maximum temperature is extracted using the MATLAB codes and the following graphs are plotted to understand the behaviour of the lean hydrogen flame.

Figure 5.2.7 illustrates the difference in net heat release rate and temperature vs. the spatial distance. At maximum temperature, the strain rates are very moderate and there is a stable and more distributed combustion resulting in higher peak temperatures compared to point of maximum consumption velocity. There, due to the high strain rate, the reactions are highly localized and have shorter window time, higher thermal diffusion, increased reactant concentration and the effects of extinction are dominant compared to the sustained combustion. Here, the temperature is not maximum, but this causes the maximum heat release rate to reach sudden peak due to the faster localized reactions at high strains. On the other hand, at the point of maximum temperature, the heat release is broader shows the distributed heat flow under low strain [16, 17].

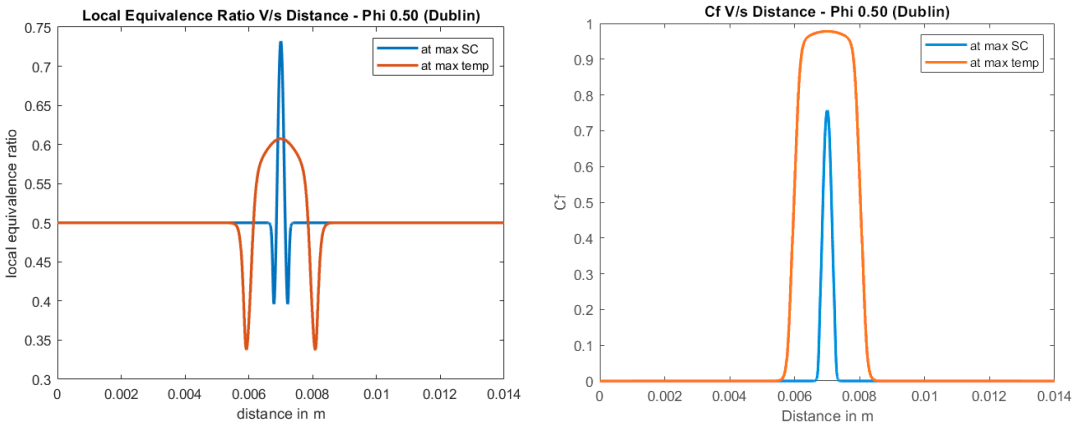

Figure 5.2.8. Local equivalence ratio and temperature vs. distance ($T_{max}\ vs. S_c^{max}$)

Figure 5.2.8 represents the local equivalence ratio and progress variable C_f vs. distance. Progress variable C_f can be calculated using the equation $C_f = 1 - \frac{X_{H_2}}{(X_{H_2})_{unb}}$, where $(X_{H_2})_{unb}$ is unburnt hydrogen mass fraction. Progress variable indicates the extent up to which the complete combustion has happened. At maximum temperature, the C_f is close to 1 compared to the point of maximum consumption velocity, this indicates the complete combustion at maximum temperature. At high strain rates, the combustion is incomplete, i.e., $C_f < 1$. This leads to an immediate sharp peak for a shorter period creating the sudden sharp peak as shown above. Local equivalence ratio is the exact equivalence ratio of the combustion mixture during the flame formation for the mixture with the global equivalence ratio of 0.50. This is varied mainly due to the preferential diffusion. The mixture undergoes an exchange of lean and fuel rich spots causing the differences in the local equivalence ratio. As soon as the reaction begins, as it is evident, the stretch in the initial phase of combustion induces the fuel lean conditions and it suddenly changes [16, 17].

Figure 5.2.9, figure 5.3.0 and figure 5.3.1 represents the concentration of the radicals H, O, OH and the intermediate species HO_2 , H_2O_2 formed during the combustion.

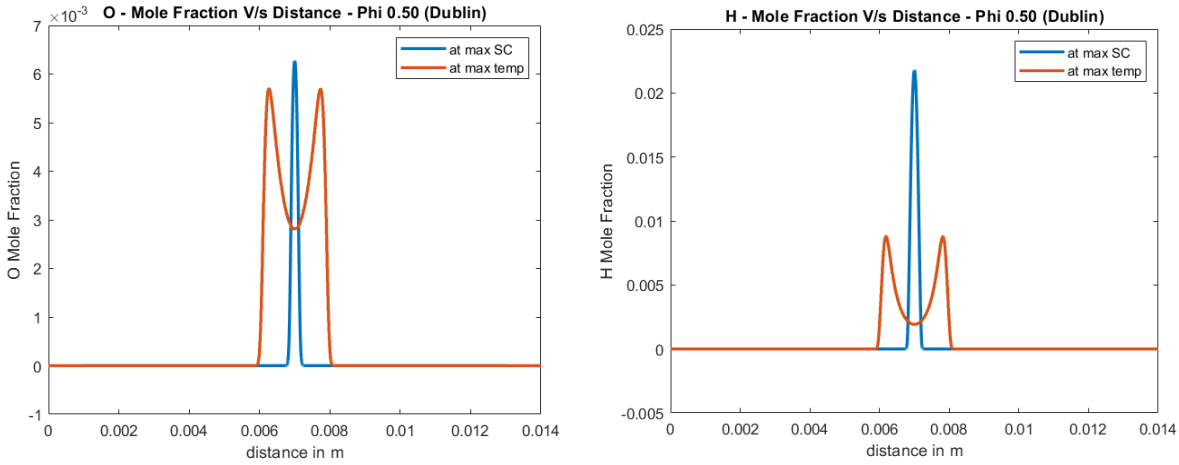

Figure 5.2.9. O and H mole fraction vs. distance ($T_{max}\ vs.\ S_c^{max}$)

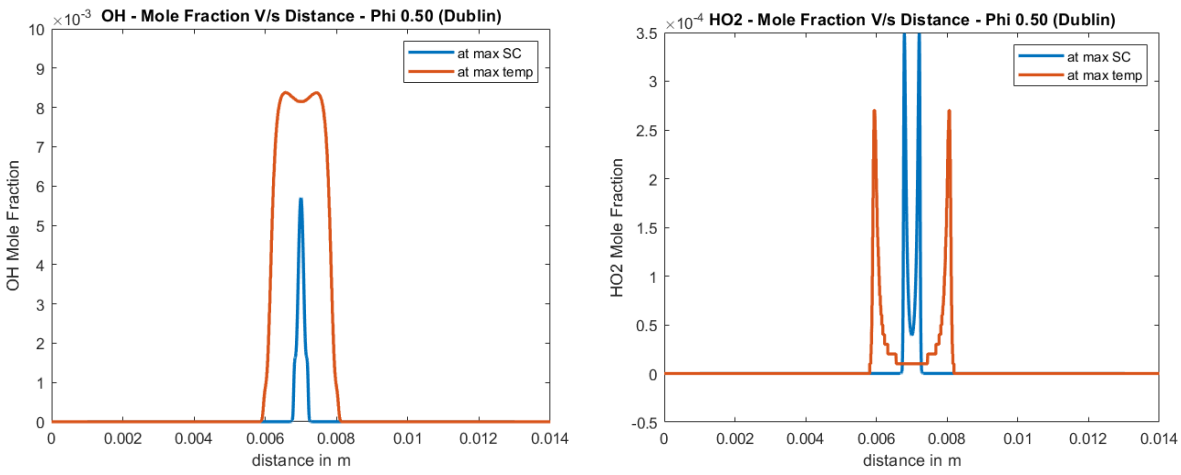

Figure 5.3.0. OH and HO_2 mole fraction vs. distance ($T_{max}\ vs. S_c^{max}$)

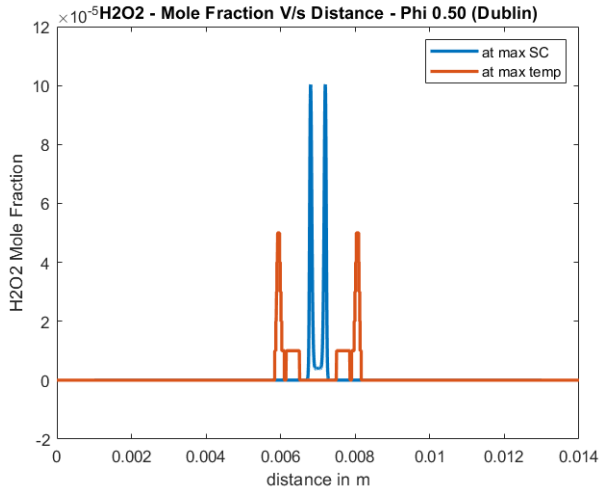

Figure 5.3.1. H_2O_2 mole fraction vs. distance ($T_{max}\ vs. S_c^{max}$)

At higher strain rates, chain branching is suppressed resulting in the reduced level of OH radicals. The intermediate species - HO_2, H_2O_2 build-up and the H radical peaks strongly because of limited consumption. At maximum temperature, the effect is quite opposite. Increased chain branching means that OH, HO_2 and H_2O_2 are consumed quickly. Due to the faster conversion, the H radical is kept lower [17, 24]. This is clearly visible in the graphs presented above.

5.2.3 Effect of pressure on consumption velocity. $\varphi = 0.50$- Dublin mechanism.

The figures 5.3.2a and 5.3.3 illustrate the effect of consumption velocity and maximum temperature due to different pressure conditions 1atm, 2atm and 3atm. As the pressure increases, the consumption velocity decreases due to reduced chain branching reactions and lower molecular transport. From the figure 5.3.2a, it looks like the flame can sustain higher absolute strain rate due to increase in the density because of higher pressure. But, when the absolute strains are normalized using diffusive – chemical time scale, $\tau_c = \frac{D}{S_l^2}$, the curves for 1atm, 2atm and 3atm, fall on one to each other, see figure 5.3.2b. This shows that the extinction

is still governed by the balance between the transport and reaction chemistry, not directly on pressure. [4, 17, 24].

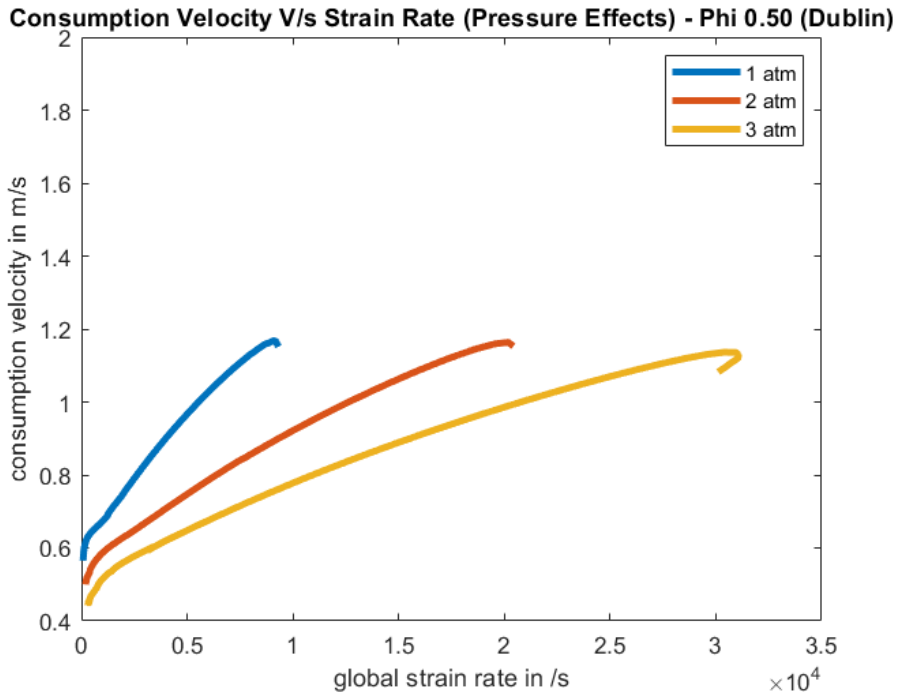

Figure 5.3.2a. Consumption velocity vs. strain rate (effect of pressure)

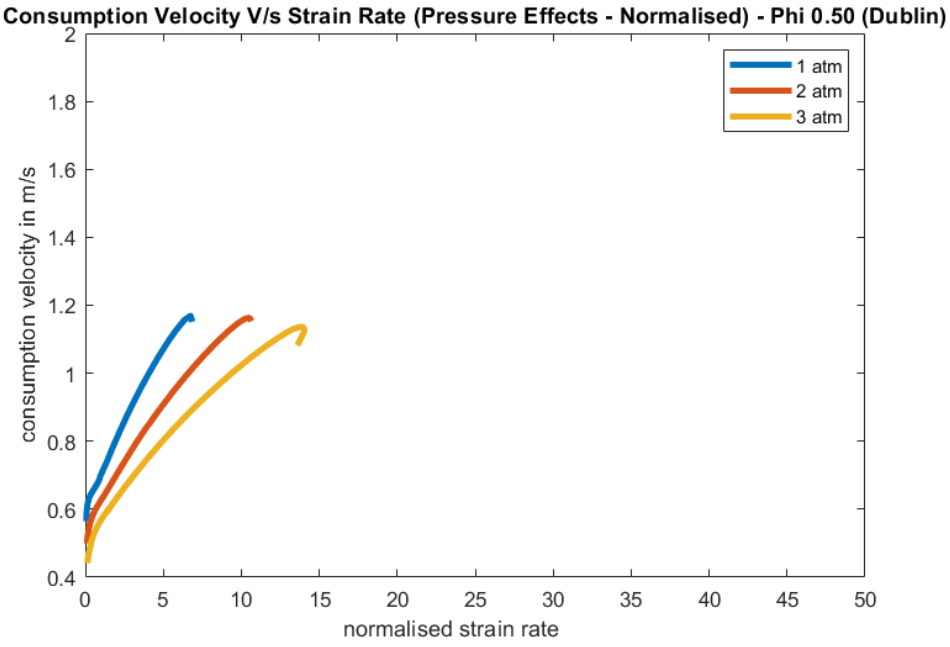

Figure 5.3.2b. Consumption velocity vs. normalised strain rate (effect of pressure)

In figure 5.3.3, as the strain rate increases the transport of species increases leading to the increased combustion rate leading to a higher combustion temperature. Later, the effect of strain rate dominates the flame stability leading to the incomplete combustion which results in decrease of temperature as the strain increases [16, 17].

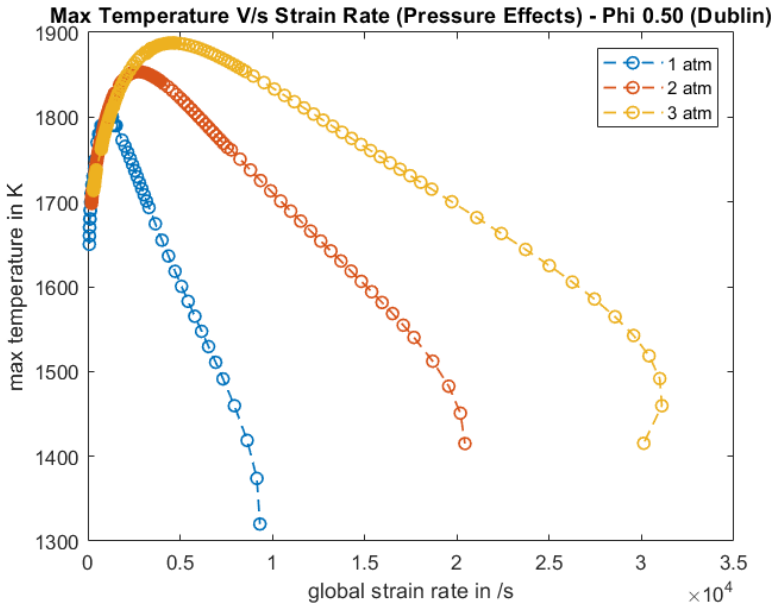

Figure 5.3.3. Temperature vs. strain rate (effect of pressure)

5.2.4 Effect of inlet temperature on consumption velocity. $\varphi = 0.50$- Dublin mechanism.

Figure 5.3.4 shows the change in consumption velocity with change in the inlet temperature. As it is evident from the figure 5.3.4 below, an increase in the inlet temperature increases the consumption velocity. As the temperature increases, as per the Arrhenius equation, the rates of many reactions also increase increasing the consumption velocity. Due to increased temperature, the radicals are already preheated. As a result, less energy is required for sustaining the combustion. This results in increasing consumption rate even at the lower strain rates. As the flame stretch increases, the rection zone becomes thinner leading to extinction at higher strain rates [16, 17].

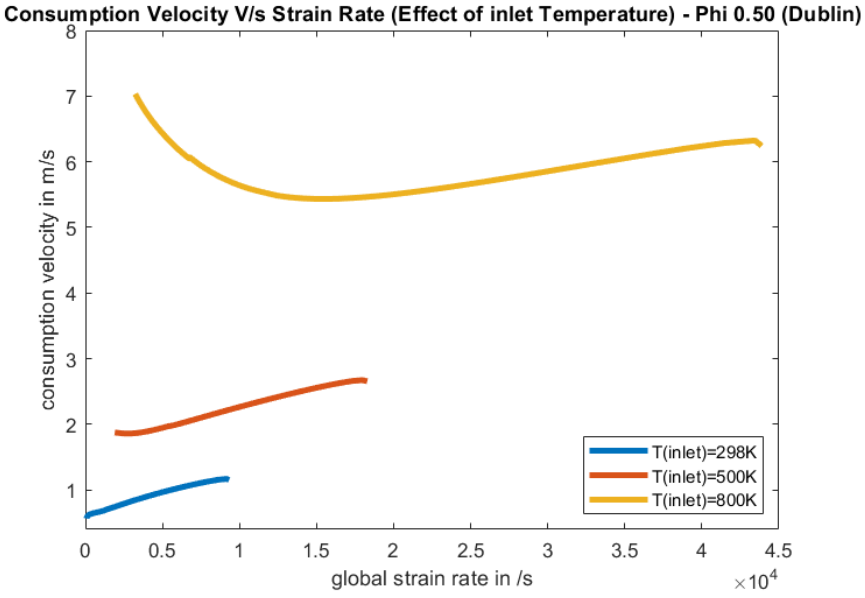

Figure 5.3.4. Consumption velocity vs. strain rate (effect of inlet temperature)

Another interesting observation is the difference in the trend between 298K, 500K and 800K. The trend for 800K is very different compared to the other inlet temperatures. This is a unique characteristic of hydrogen fuel at that temperature. At 800K, the species are already close to the auto ignition temperature (auto ignition temperature of $H_2 = 770K - 850K$). The reaction zone is already thinner and highly reactive leading to the maximum consumption velocity at low strain rates. As the strain rate slightly increases further, the consumption velocity tends to decrease initially until 15000 s^{-1}.. But later, the flame stretch effect balances the flame stability allowing an increased transport due to diffusion of species, leading to an increasing consumption velocity trend [16, 17, 24].

5.2.5 Effect of equivalence ratio on consumption velocity - Dublin mechanism.

Figure 5.3.5 illustrates the effect of the equivalence ratio on consumption velocity vs. the strain rate. As explained in equation 3.1, as the equivalence ratio increases, the fuel becomes richer and richer, i.e., the amount of fuel particles increases. As the amount of fuel increases in the air-fuel mixture, the consumption velocity also increases with the increase in the equivalence ratio [17]. The same is visible in figure 5.3.5. As the equivalence ratio increases, due to fuel rich condition, the flame can withstand the higher strain rates before extinction with as high as close to 15000 $/s$ for equivalence ratio $\varphi = 0.80$ compared to equivalence ratio $\varphi = 0.36$, which extinguishes at close to 5000 $/s$ strain rate.

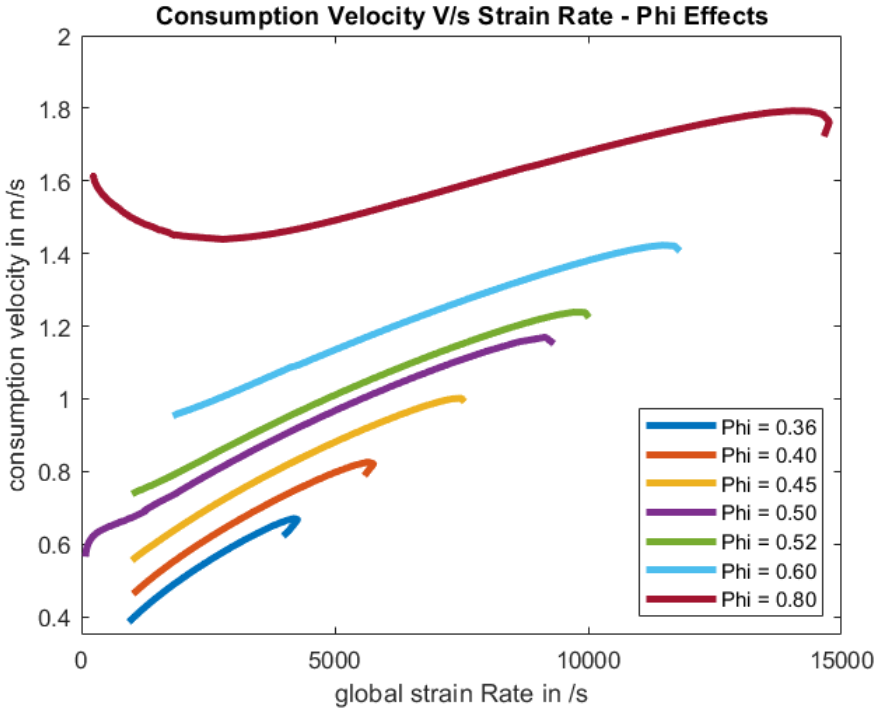

Figure 5.3.5. Consumption velocity vs. strain rate (effect of equivalence ratio)

Though the peak consumption velocity increases with increase in the equivalence ratio, another important observation is the trend for $\varphi = 0.80$, which is slightly different at low strain rates. Here, the consumption velocity decreases as the stain rate increases. This is mainly due to the

preferential diffusion and radical quenching which happens for the richer mixtures at low moderate stain rates. At higher strain rates, the flame reaches the stabilization and the consumption velocity gradually increases until it reaches the extinction [16, 17].

5.2.6 Summary – Results obtained

5.2.6.1. Unstrained/ laminar flames

Unstrained Flame Result Summary								
Phi	Pressure	Inlet Temperature	Laminar flame speed			Max adiabatic flame temperature		
			Princeton	Konnov	Dublin	Princeton	Konnov	Dublin
	atm	K	S_l in cm/s			T_{max} in K		
0.50	1	298	47.65	48.13	53.22	1630	1630	1630
0.60	1	298	83.46	81.80	90.12	1820	1830	1820
0.70	1	298	120.59	117.31	127.12	1980	1990	1980
0.80	1	298	155.33	151.85	161.35	2120	2130	2120
0.90	1	298	185.87	183.78	191.17	2220	2230	2220
1.00	1	298	212.01	211.94	216.51	2280	2290	2290
1.10	1	298	233.72	236.03	237.56	2300	2310	2310
1.20	1	298	251.26	255.60	254.49	2280	2290	2290
1.30	1	298	264.80	270.64	267.49	2250	2260	2250
1.40	1	298	274.69	281.37	276.87	2210	2210	2210
1.50	1	298	281.24	288.04	282.97	2170	2170	2170
1.60	1	298	283.00	291.28	286.26	2120	2130	2130
1.70	1	298	286.16	291.63	287.17	2080	2090	2090
1.80	1	298	285.47	289.82	286.17	2050	2060	2050
1.90	1	298	283.31	286.35	283.74	2010	2020	2010
2	1	298	280.02	281.69	280.19	1970	1990	1970
2.1	1	298	275.84	276.18	275.79	1940	1950	1940
2.2	1	298	271.03	269.98	270.84	1910	1920	1910
2.3	1	298	265.74	263.47	265.41	1880	1890	1880
2.4	1	298	260.13	256.61	259.68	1850	1870	1850
2.5	1	298	254.29	249.61	253.80	1830	1840	1820
2.6	1	298	248.28	242.60	247.76	1800	1810	1800
2.7	1	298	242.17	235.56	241.62	1770	1790	1770
2.8	1	298	236.03	228.54	235.43	1750	1760	1750
2.9	1	298	229.90	221.59	229.29	1730	1740	1730

Table 5.2.6.1 – Summary of results unstrained/laminar flames

5.2.6.2. Strained laminar flames

Strained Flame Result Summary - Dublin Mechanism							
Phi	Pressure	Inlet Temperature	Laminar flame speed	Max Consumption velocity	Max strain rate	Ratio	Flame thickness at S_c^{max} $\partial_t = \frac{T_{adb} - T_{unb}}{\left\|\frac{\partial t}{\partial x}\right\|_{max}}$
	atm	K	S_l in m/s	S_c^{max} in m/s	s^{-1}	S_c^{max}/S_l	δ_t in m
0.36	1	298	0.1360	0.6702	4192	4.928	1.7960E-04
0.40	1	298	0.2285	0.8256	5653	3.613	1.6510E-04
0.45	1	298	0.3700	1.001	7495	2.705	1.5330E-04
0.50	1	298	0.5324	1.1694	9157	2.196	1.4790E-04
	2	298	0.4140	1.1636	20183	2.811	6.7050E-05
	3	298	0.3410	1.1365	30405	3.333	4.3840E-05
	5	298	0.2470	-	-	-	-
	1	500	-	2.674	17955	-	1.9060E-04
0.52	1	298	0.6028	1.2384	9712	2.054	1.4750E-04
0.60	1	298	0.9012	1.42274	11473	1.579	1.4400E-04
0.80	1	298	1.6100	1.7923	14017	1.113	1.4670E-04

Table 5.2.6.2 – Summary of results strained laminar flames

6. Conclusions

As we have already discussed in the earlier sections, hydrogen becomes one of the common solutions which can immediately be used as the fuel in existing combustion systems with minor modifications. Hydrogen can also be produced using green methods, which makes hydrogen even more interesting as a fuel [1].

It is evident from the studies that though hydrogen combustion is still prone to NO_X emissions at higher temperatures compared to hydrocarbon fuels, hydrogen becomes a carbon free fuel, which has almost no carbon emissions as a bi product. This is an important advantage compared to other existing carbon-based fuels. Another major advantage of hydrogen is the flammability limits of hydrogen, hydrogen can handle a very wide operating range from very lean to very rich operating conditions. Due to high combustibility, it takes smaller energy to ignite the fuel-air mixture even at lean conditions. And running on lean conditions can further contribute for lesser NO_X formation as the combustion temperatures are smaller compared to near-stoichiometric mixtures [17].

Another important observation is the flame speed of hydrogen itself, which is another important combustion characteristics. Hydrogen has the highest flame velocity compared to fossil fuels which makes the combustion faster. For example, in case of combustion engines, the engine can operate under higher rpm as the combustion itself is not limited by the flame speed like for hydrocarbon fuels [29]. Hydrogen has high diffusivity compared to hydrocarbon fuels which in turn supports faster and complete combustion due to uniform mixing of air fuel mixture due to high diffusivity property of hydrogen [17]. When it comes to the quenching properties, hydrogen has shorter quenching distance compared to hydrocarbon fuel and hydrogen's lower density compared to fossil fuels make its application interesting and practical [17].

It is evident from the simulations above that the flame speed and the consumption velocity is sensitive to flame stretch, boundary conditions and the composition of the air fuel mixture itself. The consumption velocity increases with increase in the pressure, inlet temperature and the fuel air mixture composition. Richer the mixture the larger consumption velocity. At these conditions, the flame can also withstand higher strain rates before extinction. Also, through simulations, it can be observed that the resolution of grid points and the flame curvature also has direct effects on the calculation, hence it is also important for the user to predict the right approximations for right type of flame problem based on the outcome required.

This project successfully computes the peak consumption velocities at different mixture compositions – mainly focusing lean mixtures and different boundary conditions, using CHEMKIN PRO simulation tool as presented in section 5.2.5. Further, the data can be directly used to adopt the model developed by Lipatnikov and Chomiak [4] for predicting turbulent burning velocity. The model and computed data can be used for CFD research into engines fuelled with lean hydrogen mixtures.

7. References

[1] I. Vinoth Kanna and P. Paturu, "A study of hydrogen as an alternative fuel," *International Journal of Ambient Energy*, vol. 41, no. 12, pp. 1433-1436, 2020.

[2] S. Atilhan, S. Park, M. M. El-Halwagi, M. Atilhan, M. Moore, and R. B. Nielsen, "Green hydrogen as an alternative fuel for the shipping industry," *Current Opinion in Chemical Engineering*, vol. 31, p. 100668, 2021.

[3] A. Onorati *et al.*, "The role of hydrogen for future internal combustion engines," vol. 23, ed: Sage Publications Sage UK: London, England, 2022, pp. 529-540.

[4] A. Lipatnikov and J. Chomiak, "Molecular transport effects on turbulent flame propagation and structure," *Progress in energy and combustion science*, vol. 31, no. 1, pp. 1-73, 2005.

[5] S. Verma, F. Monnier, and A. N. Lipatnikov, "Validation of leading point concept in RANS simulations of highly turbulent lean syngas-air flames with well-pronounced diffusional-thermal effects," *International Journal of Hydrogen Energy*, vol. 46, no. 13, pp. 9222-9233, 2021.

[6] R. Design, "CHEMKIN-PRO 15131 User's Manual," Reaction Design, San Diego, CA, 2013.

[7] W. K. Metcalfe, S. Burke, and H. J. Curran, "AramcoMech 1.3 C3," Combustion Chemistry Centre, National University of Ireland, Galway, August 9, 2025 2013. [Online]. Available: http://c3.nuigalway.ie/

[8] M. P. Burke, M. Chaos, Y. Ju, F. L. Dryer, and S. J. Klippenstein, "H_2 Kinetic Mechanism (Version 6-10-2011)," Princeton University, August 6, 2025 2011. [Online]. Available: https://combustion.princeton.edu

[9] A. Konnov, "A. Konnov's detailed reaction mechanism — h/O_3 excited 2018," Unpublished, distributed for combustion modeling, August 9, 2025 2018.

[10] E. Ranzi *et al.*, "Hierarchical and comparative kinetic modeling of laminar flame speeds of hydrocarbon and oxygenated fuels," *Progress in Energy and Combustion Science*, vol. 38, no. 4, pp. 468-501, August 9, 2025 2012, doi: 10.1016/j.pecs.2012.03.004.

[11] C. Rutland and A. Trouvé, "Direct simulations of premixed turbulent flames with nonunity Lewis numbers," *Combustion and Flame*, vol. 94, no. 1-2, pp. 41-57, 1993.

[12] F. M. White and J. Majdalani, *Viscous fluid flow*. McGraw-Hill New York, 2006.

[13] H. Schlichting, "and K. Gersten: Boundary-Layer Theory, 9th," ed: Springer Berlin Heidelberg: Imprint: Springer, Berlin, 2017.

[14] S. R. Turns, "Introduction to combustion," vol. 287: McGraw-Hill Companies New York, NY, USA, 1996, ch. Laminar Premixed Flames, pp. 258-307.

[15] D. Dunn-Rankin, *Lean combustion: technology and control*. Academic Press, 2011.

[16] T. Poinsot and D. Veynante, *Theoretical and numerical combustion*. RT Edwards, Inc., 2005.

[17] K. K. Kuo, "Principles of combustion," 1986, ch. Premixed Laminar Flames, pp. 437-538.

[18] S. McAllister, J.-Y. Chen, and A. C. Fernandez-Pello, *Fundamentals of combustion processes*. Springer, 2011.

[19] B. Karlovitz, D. Denniston Jr, D. Knapschaefer, and F. Wells, "Studies on Turbulent flames: A. Flame Propagation Across velocity gradients B. turbulence Measurement in flames," in *Symposium (international) on combustion*, 1953, vol. 4, no. 1: Elsevier, pp. 613-620.

[20] G. Markstein, "Theory of flame propagation," in *AGARDograph*, vol. 75: Elsevier, 1964, pp. 5-14.

[21] V. Giovangigli and M. Smooke, "Extinction of strained premixed laminar flames with complex chemistry," *Combustion science and technology,* vol. 53, no. 1, pp. 23-49, 1987.

[22] P. A. Libby, A. Liñ´ n, and F. A. Williams, "Strained premixed laminar flames with nonunity Lewis numbers," *Combustion Science and Technology,* vol. 34, no. 1-6, pp. 257-293, 1983.

[23] A. Buekens, "Combustion: physical and chemical fundamentals, modeling and simulation, experiments, pollutant formation," *International Journal of Environment & Pollution,* vol. 17, no. 3, pp. 291-291, 2002.

[24] K. Chung, "Law. Combustion Physics," in *New York*, 2006, ch. Laminar Premixed Flames, pp. 243-301.

[25] V. Giovangigli, "Multicomponent flow modeling," *Science China Mathematics,* vol. 55, pp. 285-308, 2012.

[26] J. C. Keck and D. Gillespie, "Rate-controlled partial-equilibrium method for treating reacting gas mixtures," *Combustion and Flame,* vol. 17, no. 2, pp. 237-241, 1971.

[27] G. D. Roy, S. M. Frolov, A. A. Borisov, and D. W. Netzer, "Pulse detonation propulsion: challenges, current status, and future perspective," *Progress in Energy and Combustion Science,* vol. 30, no. 6, pp. 545-672, 2004.

[28] Y. Huang, C. Sung, and J. A. Eng, "Laminar flame speeds of primary reference fuels and reformer gas mixtures," *Combustion and Flame,* vol. 139, no. 3, pp. 239-251, 2004.

[29] B. Shadidi, G. Najafi, and T. Yusaf, "A review of hydrogen as a fuel in internal combustion engines," *Energies,* vol. 14, no. 19, p. 6209, 2021.

8. Appendix

8.1 MATLAB Codes:

8.1.1 Unstrained/laminar flames:

```
clc
clearvars

%% Unstarined Flame Simulations
% Extract .csv data from chemkin simulations
unstrainedkonnov = readmatrix('resultskonnoov.csv');
unstrainedmilanoo = readmatrix('resultsmilano.csv');
unstraineddublin = readmatrix('resultsdublin.csv');
unstrainedprinceton = readmatrix('resultsprinceton.csv');

% Convert table to data
% 4 column flame speed, 5 column max temp in k, 6 column net heat prodn (erg/cm3-sec)
% No Solution Exist for Princeton Phi 1.6
equi_ratio = linspace(0.5,2.9,25);

fl_speed_konnov = unstrainedkonnov(:,4);
fl_speed_milano = unstrainedmilanoo(:,4);
fl_speed_dublin = unstraineddublin(:,4);
fl_speed_princeton = unstrainedprinceton(:,4);

% compare fl_speed for different mechanisms - Grad=0.01, Curv =0.05, Grid = 1000

figure()
plot(equi_ratio,fl_speed_konnov,'--o', 'linewidth',1)
hold on
plot(equi_ratio,fl_speed_princeton,'--', 'linewidth',1)
plot(equi_ratio,fl_speed_dublin,'b--', 'linewidth',1)
hold off
legend('Konnov','Princeton','Dublin','Location','southeast')
title('Laminar Flame Speed Comparison -Different Mechanisms')
xlabel('Equivalence Ratio')
ylabel('FlameSpeed in Cm/Sec');

% Compare max temp

maxtemp_konnov = unstrainedkonnov(:,5);
maxtemp_milano = unstrainedmilanoo(:,5);
maxtemp_dublin = unstraineddublin(:,5);
maxtemp_princeton = unstrainedprinceton(:,5);

figure()
plot(equi_ratio,maxtemp_konnov,'--o', 'linewidth',1)
hold on
plot(equi_ratio,maxtemp_princeton,'--', 'linewidth',1)
plot(equi_ratio,maxtemp_dublin,'b--', 'linewidth',1)
hold off
legend('Konnov','Princeton','Dublin','Location','southeast')
title('Max Temparature Comparison-Different Mechanisms')
xlabel('Equivalence Ratio')
ylabel('Maximum Temparature in K');

% Compare net heat
figure()

netheat_konnov = unstrainedkonnov(:,6);
netheat_milano = unstrainedmilanoo(:,6);
netheat_dublin = unstraineddublin(:,6);
```

```
netheat_princeton = unstrainedprinceton(:,6);

plot(equi_ratio,netheat_konnov,'--o', 'linewidth',1)
hold on
plot(equi_ratio,netheat_princeton,'--', 'linewidth',1)
plot(equi_ratio,netheat_dublin,'b--', 'linewidth',1)
hold off
legend('Konnov','Princeton','Dublin','Location','southeast')
title('Net Heat Production Comparison-Different Mechanisms')
xlabel('Equivalence Ratio')
ylabel('Net Heat Production in erg/cm3-sec');

%% Comparison of different effects - Example Milano.
% 4 column flame speed, 5 column max temp in k, 6 column net heat prodn (erg/cm3-sec)
% Flame speed comparison
% With both TD & MC On
figure()
plot(equi_ratio,fl_speed_milano,'--o', 'linewidth',1)
hold on

% With both TD & MC Off - Solution exist only from Phi = 0.6
milano_mctd_off = readmatrix('milano_mc_td_both_off.csv');
fl_speed_milano_mctd_off = milano_mctd_off(:,4);
plot(equi_ratio(1,2:end),fl_speed_milano_mctd_off,'--', 'linewidth',1)

% Only TD On & MC Off
milano_td_on_mc_off = readmatrix('milano_mc_off_td_on.csv');
fl_speed_milano_td_on_mc_off = milano_td_on_mc_off(:,4);
plot(equi_ratio,fl_speed_milano_td_on_mc_off,'b--', 'linewidth',1)

% Only TD Off & MC On
milano_td_off_mc_on = readmatrix("milano_mc_on_td_off.csv");
fl_speed_milano_td_off_mc_on = milano_td_off_mc_on(:,4);
plot(equi_ratio,fl_speed_milano_td_off_mc_on,'x--', 'linewidth',1)
hold off
legend('TD-O & MC-O','TD-X & MC-X','TD-O & MC-X', 'TD-X & MC-O','Location','southeast')
title('Comparison of Differen effects of TD & MC transport effects(Flaame Speed)-Milano')
xlabel('Equivalence Ratio')
ylabel('FlameSpeed in Cm/Sec');

% Temprature Comparison
% With both TD & MC On
figure()
plot(equi_ratio,maxtemp_milano,'--o', 'linewidth',1)
hold on

% With both TD & MC Off - Solution exist only from Phi = 0.6
maxtemp_milano_mctd_off = milano_mctd_off(:,5);
plot(equi_ratio(1,2:end),maxtemp_milano_mctd_off,'--', 'linewidth',1)

% Only TD On & MC Off
maxtemp_milano_td_on_mc_off = milano_td_on_mc_off(:,5);
plot(equi_ratio,maxtemp_milano_td_on_mc_off,'b--', 'linewidth',1)

% Only TD Off & MC On
maxtemp_milano_td_off_mc_on = milano_td_off_mc_on(:,5);
plot(equi_ratio,maxtemp_milano_td_off_mc_on,'x--', 'linewidth',1)
hold off
legend('TD-O & MC-O','TD-X & MC-X','TD-O & MC-X', 'TD-X & MC-O','Location','southeast')
title('Comparison of Differen effects of TD & MC transport effects (Max Temprature)-
Milano')
xlabel('Equivalence Ratio')
ylabel('Maximum Temparature in K');

% Net Heat Prodn Comparison
figure()
```

```
plot(equi_ratio,netheat_milano,'--o', 'linewidth',1)
hold on

% With both TD & MC Off - Solution exist only from Phi = 0.6
netheat_milano_mctd_off = milano_mctd_off(:,6);
plot(equi_ratio(1,2:end),netheat_milano_mctd_off,'--', 'linewidth',1)

% Only TD On & MC Off
netheat_milano_td_on_mc_off = milano_td_on_mc_off(:,6);
plot(equi_ratio,netheat_milano_td_on_mc_off,'b--', 'linewidth',1)

% Only TD Off & MC On
netheat_milano_td_off_mc_on = milano_td_off_mc_on(:,6);
plot(equi_ratio,netheat_milano_td_off_mc_on,'x--', 'linewidth',1)
hold off
legend('TD-O & MC-O','TD-X & MC-X','TD-O & MC-X', 'TD-X & MC-O','Location','southeast')
title('Comparison of Differen effects of TD & MC transport effects (Net Heat Prodn) -
Milano')
xlabel('Equivalence Ratio')
ylabel('Net Heat Production in erg/cm3-sec');

%% Comparison of Pressure effects - Example Milano.
% 4 column flame speed, 5 column max temp in k, 6 column net heat prodn (erg/cm3-sec)
% MC TD ON, Grid 1000, Grad 0.01 Curv 0.05
% Flame Speed Comparison
% P = 1atm
figure()
plot(equi_ratio,fl_speed_milano,'--o', 'linewidth',1)
hold on

% P = 3atm
milano_pres_3atm= readmatrix('milano_pres_3atm.csv');
fl_speed_milano_pres_3atm = milano_pres_3atm(:,4);
plot(equi_ratio,fl_speed_milano_pres_3atm,'--', 'linewidth',1)

% P = 5atm - No data exist for 1.1,1.2,1.3,1.4 - Imaginary values Values
milano_pres_5atm= readmatrix('milano_pres_5atm.csv');
fl_speed_milano_pres_5atm = milano_pres_5atm(:,4);
plot(equi_ratio,fl_speed_milano_pres_5atm,'--', 'linewidth',1)
hold off
legend('Pres 1ATM','Pres 3ATM','Pres 5ATM','Location','southeast')
title('Effect Of Pressure on Flame Speed - Milano')
xlabel('Equivalence Ratio')
ylabel('FlameSpeed in Cm/Sec');

% Max Temprature Comparison
% P = 1atm
figure()
plot(equi_ratio,maxtemp_milano,'--o', 'linewidth',1)
hold on

% P = 3atm
maxtemp_milano_pres_3atm = milano_pres_3atm(:,5);
plot(equi_ratio,maxtemp_milano_pres_3atm,'--V', 'linewidth',1)

% P = 5atm - No data exist for 1.1,1.2,1.3,1.4 - Imaginary values Values
maxtemp_milano_pres_5atm = milano_pres_5atm(:,5);
plot(equi_ratio,maxtemp_milano_pres_5atm,'--x', 'linewidth',1)
hold off
legend('Pres 1ATM','Pres 3ATM','Pres 5ATM','Location','southeast')
title('Effect Of Pressure on Max Temprature - Milano')
xlabel('Equivalence Ratio')
ylabel('Max Temparature in K');

% Net Heat Comparison
% P = 1atm
```

```
figure()
plot(equi_ratio,netheat_milano,'--o', 'linewidth',1)
hold on

% P = 3atm
netheat_milano_pres_3atm = milano_pres_3atm(:,6);
plot(equi_ratio,netheat_milano_pres_3atm,'--V', 'linewidth',1)

% P = 5atm - No data exist for 1.1,1.2,1.3,1.4 - Imaginary values Values
netheat_milano_pres_5atm = milano_pres_5atm(:,6);
plot(equi_ratio,netheat_milano_pres_5atm,'--x', 'linewidth',1)
hold off
legend('Pres 1ATM','Pres 3ATM','Pres 5ATM','Location','southeast')
title('Effect Of Pressure on Net Heat Production - Milano')
xlabel('Equivalence Ratio')
ylabel('Net Heat Production in erg/cm3-sec');

%% Grad & Curv Effects
% 4 column flame speed, 5 column max temp in k, 6 column net heat prodn (erg/cm3-sec)
% Flame Speed
% Grad 0.01 & CURV = 0.05, Grid 1000
figure()
plot(equi_ratio,fl_speed_konnov,'--o', 'linewidth',1)
hold on

% Grad 0.1 & CURV = 0.5 , Grid 1000
milano_grad01_curv05= readmatrix('milano_grad0.1_curv0.5.csv');
fl_speed_milano_grad01_curv05 = milano_grad01_curv05(:,4);
plot(equi_ratio,fl_speed_milano_grad01_curv05,'--', 'linewidth',1)

% Grad 0.01 & CURV = 0.01 , Grid 1000_no solution for 1.8 Phi
milano_grad001_curv001= readmatrix('milano_grad0.01_curv0.01.csv');
fl_speed_milano_grad001_curv001 = milano_grad001_curv001(:,4);
plot(equi_ratio,fl_speed_milano_grad001_curv001,'--', 'linewidth',1)
hold off
legend('GRAD-0.01 & CURV-0.05','GRAD-0.1 & CURV - 0.5','GRAD-0.01 & CURV-0.01',
'Location','southeast')
title('Comparison of Differen GRAD & CURV effects (Flaame Speed)-Milano')
xlabel('Equivalence Ratio')
ylabel('FlameSpeed in Cm/Sec');

% Max Temperature
% Grad 0.01 & CURV = 0.05, Grid 1000
figure()
plot(equi_ratio,maxtemp_milano,'--o', 'linewidth',1)
hold on

% Grad 0.1 & CURV = 0.5 , Grid 1000
maxtemp_milano_grad01_curv05 = milano_grad01_curv05(:,5);
plot(equi_ratio,maxtemp_milano_grad01_curv05,'--', 'linewidth',1)

% Grad 0.01 & CURV = 0.01 , Grid 1000_no solution for 1.8 Phi
maxtemp_milano_grad001_curv001 = milano_grad001_curv001(:,5);
plot(equi_ratio,maxtemp_milano_grad001_curv001,'--', 'linewidth',1)
hold off
legend('GRAD-0.01 & CURV-0.05','GRAD-0.1 & CURV - 0.5','GRAD-0.01 & CURV-
0.01','Location','southeast')
title('Effect Of GRAD & CURV on Max Temprature - Milano')
xlabel('Equivalence Ratio')
ylabel('Max Temparature in K');

% Net Heat Production
% Grad 0.01 & CURV = 0.05, Grid 1000
figure()
plot(equi_ratio,netheat_milano,'--o', 'linewidth',1)
hold on
```

```
% Grad 0.1 & CURV = 0.5 , Grid 1000
netheat_milano_grad01_curv05 = milano_grad01_curv05(:,6);
plot(equi_ratio,netheat_milano_grad01_curv05,'--', 'linewidth',1)

% Grad 0.01 & CURV = 0.01 , Grid 1000_no solution for 1.8 Phi
netheat_milano_grad001_curv001 = milano_grad001_curv001(:,6);
plot(equi_ratio,netheat_milano_grad001_curv001,'--', 'linewidth',1)
hold off
legend('GRAD-0.01 & CURV-0.05','GRAD-0.1 & CURV - 0.5','GRAD-0.01 & CURV-
0.01','Location','southeast')
title('Effect Of GRAD & CURV on Net Heat Production - Milano')
xlabel('Equivalence Ratio')
ylabel('Net Heat Production in erg/cm3-sec');

%% Comparison of Euivalent Combustion products at equivalent combustion productst
associated with Highest T At Phi 1.1
% Plot Distance against H2, O2, N2, H20

% Konnov
% Column1 = Distance, Column 10 = Mole Fraction H2,
% Column 12 = Mole Fraction O2, Column 13 = Mole Fraction H2O,Column 20 = Mole Fraction N2
products_konnov = readmatrix('productscomparison_konnov_1.1.csv');

%Extract Data
products_konnov_dist = products_konnov (:,1);
products_konnov_h2 = products_konnov (:,10);
products_konnov_O2 = products_konnov (:,12);
products_konnov_H2O = products_konnov (:,13);
products_konnov_N2 = products_konnov (:,20);

% Dublin
%Column1=Distance, Column 9 =H2, Column 10 = O2, Column 11 = H2O,
% Column 17= N2
products_dublin = readmatrix('productscomparison_dublin_1.1.csv');

%Extract Data
products_dublin_dist = products_dublin (:,1);
products_dublin_h2 = products_dublin (:,9);
products_dublin_O2 = products_dublin (:,10);
products_dublin_H2O = products_dublin (:,11);
products_dublin_N2 = products_dublin (:,17);

% Princeton
%Column1=Distance, Column 10 =H2, Column 12 = O2, Column 14 = H2O,
% Column 17= N2
products_princeton = readmatrix('productscomparison_princeton_1.1.csv');

%Extract Data
products_princeton_dist = products_princeton (:,1);
products_princeton_h2 = products_princeton (:,10);
products_princeton_O2 = products_princeton (:,12);
products_princeton_H2O = products_princeton (:,14);
products_princeton_N2 = products_princeton (:,17);

% H2 Mole Fractions
figure()
plot(products_konnov_dist,products_konnov_h2,'-', 'linewidth',2)
hold on
plot(products_dublin_dist,products_dublin_h2,'--x', 'linewidth',2)
plot(products_princeton_dist,products_princeton_h2,'--x', 'linewidth',2)
hold off
legend('Konnov','Dublin','Princeton','Location','northeast')
title('Combustion Products Associated With Phi-1.1 at Max Temp H2')
xlabel('Distance in CM')
ylabel('Mole Fraction H2');
```

```
% O2 Mole Fractions
figure()
plot(products_konnov_dist,products_konnov_O2,'-', 'linewidth',2)
hold on
plot(products_dublin_dist,products_dublin_O2,'--', 'linewidth',2)
plot(products_princeton_dist,products_princeton_O2,'--', 'linewidth',2)
hold off
legend('Konnov','Dublin','Princeton','Location','southeast')
title('Combustion Products Associated With Phi-1.1 at Max Temp O2')
xlabel('Distance in CM')
ylabel('Mole Fraction O2');

% H2O Mole Fractions
figure()
plot(products_konnov_dist,products_konnov_H2O,'-', 'linewidth',2)
hold on
plot(products_dublin_dist,products_dublin_H2O,'--', 'linewidth',2)
plot(products_princeton_dist,products_princeton_H2O,'--', 'linewidth',2)
hold off
legend('Konnov','Dublin','Princeton','Location','southeast')
title('Combustion Products Associated With Phi-1.1 at Max Temp H2O')
xlabel('Distance in CM')
ylabel('Mole Fraction H2O');

% N2 Mole Fractions
figure()
plot(products_konnov_dist,products_konnov_N2,'-', 'linewidth',2)
hold on
plot(products_dublin_dist,products_dublin_N2,'--', 'linewidth',2)
plot(products_princeton_dist,products_princeton_N2,'--', 'linewidth',2)
hold off
legend('Konnov','Dublin','Princeton','Location','southeast')
title('Combustion Products Associated With Phi-1.1 at Max Temp N2')
xlabel('Distance in CM')
ylabel('Mole Fraction N2');

%% End Of Unstrained Flames
```

8.1.2 Strained laminar flames:

```
close all
clc
clearvars

%% Princeton 05
% general code for particular cases
% 0.5 Phi - Opposed flame analysis _ princeton_ default Grad 0.01, Curv - 0.05
% Grid - 100000

 % Load the CSV files
 princeton_default_netheat_05 = readmatrix('princeton_default_netheat_05.csv'); % Replace
with the actual filename
 princeton_default_distance_05 = readmatrix('princeton_default_distance_05.csv'); % Replace
with the actual filename

 % Determine the number of columns
 princeton_default_numColumns_integrate_05 = min(size(princeton_default_netheat_05, 2),
size(princeton_default_distance_05, 2));

 % Initialize the results matrix
 princeton_default_integration_results_05 = zeros(1,
princeton_default_numColumns_integrate_05);

 % Loop through each column
```

```
 for i = 1:princeton_default_numColumns_integrate_05
      % Extract current column for netheat and distance
     princeton_default_netheat_col_05 = princeton_default_netheat_05(:, i);
     princeton_default_distance_col_05 = princeton_default_distance_05(:, i);

     % Remove NaN values and ensure both arrays align
     princeton_default_valid_idx_05 = ~isnan(princeton_default_netheat_col_05) &
~isnan(princeton_default_distance_col_05);
     princeton_default_netheat_col_05 =
princeton_default_netheat_col_05(princeton_default_valid_idx_05);
     princeton_default_distance_col_05 =
princeton_default_distance_col_05(princeton_default_valid_idx_05);

     % Perform integration only if there are sufficient data points
     if length(princeton_default_distance_col_05) > 1
         % Sort distance and netheat by distance values for correct integration
         [princeton_default_distance_col_05, princeton_default_sortIdx_05] =
sort(princeton_default_distance_col_05);
         princeton_default_netheat_col_05 =
princeton_default_netheat_col_05(princeton_default_sortIdx_05);

         % Perform numerical integration using the trapezoidal rule
         princeton_default_integration_results_05(i) =
trapz(princeton_default_distance_col_05, princeton_default_netheat_col_05);
     else
         % Not enough data to integrate
         princeton_default_integration_results_05(i) = NaN;
     end
 end

 % Load the CSV file
 princeton_default_density_05 = readmatrix('princeton_default_density_05.csv'); % Replace
with the actual filename

 % Extract the first row of each column
 princeton_default_first_row_values_density_05 = princeton_default_density_05(1, :);

 % Load the CSV file
 princeton_default_sensibleenthalpy_05 =
readmatrix('princeton_default_sensibleenthalpy_05.csv'); % Replace with the actual filename

 % Get the number of columns
 princeton_default_numColumns_sense_05 = size(princeton_default_sensibleenthalpy_05, 2);

 % Initialize matrices to store first and last valid values
 princeton_default_sensible_enthalpy_first_05 = zeros(1,
princeton_default_numColumns_sense_05);
 princeton_default_sensible_enthalpy_last_05 = zeros(1,
princeton_default_numColumns_sense_05);

 % Loop through each column
 for i = 1:princeton_default_numColumns_sense_05
     % Extract the column
     princeton_default_column_data_05 = princeton_default_sensibleenthalpy_05(:, i);

     % Remove NaN values to get valid data
     princeton_default_valid_data_05 =
princeton_default_column_data_05(~isnan(princeton_default_column_data_05));

     % Get the first and last values from the valid data
     if ~isempty(princeton_default_valid_data_05)
         princeton_default_sensible_enthalpy_first_05(i) =
princeton_default_valid_data_05(1); % First value
         princeton_default_sensible_enthalpy_last_05(i) =
max(princeton_default_valid_data_05(:,end)); % Last value
     else
```

```
        % If the column is entirely NaN, assign NaN
        princeton_default_sensible_enthalpy_first_05(i) = NaN;
        princeton_default_sensible_enthalpy_last_05(i) = NaN;
    end
end


princeton_default_xyz_05 = princeton_default_sensible_enthalpy_last_05-
princeton_default_sensible_enthalpy_first_05;

princeton_default_den_05 =
princeton_default_first_row_values_density_05.*princeton_default_xyz_05;

princeton_default_SC_05 =
princeton_default_integration_results_05./(2.*princeton_default_den_05);

% Load the strain rate data from the CSV file
princeton_default_axialvel_05 = readmatrix('princeton_default_axialvelocity_05.csv');

% Remove rows with NaN values
princeton_default_first_row_values_axialvel_05 = princeton_default_axialvel_05(1, :);
princeton_default_max_dist_05 = max(princeton_default_distance_05, [], 1, 'omitnan');

% Find the maximum value in each column
% dublin_default_max_strainrate_05 = max(dublin_default_strainrate_05, [], 1); % Row
vector with max values for each column
% L = 0.014;
princeton_default_strainrate_05 =
2*princeton_default_first_row_values_axialvel_05./princeton_default_max_dist_05;


 % Load the strain rate data from the CSV file
princeton_default_maxtemp_05 = readmatrix('princeton_default_temperature_05.csv');

% Remove rows with NaN values
princeton_default_max_temp_05 = max(princeton_default_maxtemp_05, [], 1, 'omitnan');

%% Dublin -0.5
% 0.5 Phi - Opposed flame analysis _ dublin_ default Grad 0.01, Curv - 0.05
% Grid - 100000

 % Load the CSV files
dublin_default_netheat_05 = readmatrix('dublin_default_netheat_05.csv'); % Replace with
the actual filename
dublin_default_distance_05 = readmatrix('dublin_default_distance_05.csv'); % Replace with
the actual filename

% Determine the number of columns
dublin_default_numColumns_integrate_05 = min(size(dublin_default_netheat_05, 2),
size(dublin_default_distance_05, 2));

% Initialize the results matrix
dublin_default_integration_results_05 = zeros(1, dublin_default_numColumns_integrate_05);

% Loop through each column
for i = 1:dublin_default_numColumns_integrate_05
     % Extract current column for netheat and distance
    dublin_default_netheat_col_05 = dublin_default_netheat_05(:, i);
    dublin_default_distance_col_05 = dublin_default_distance_05(:, i);

    % Remove NaN values and ensure both arrays align
    dublin_default_valid_idx_05 = ~isnan(dublin_default_netheat_col_05) &
~isnan(dublin_default_distance_col_05);
    dublin_default_netheat_col_05 =
dublin_default_netheat_col_05(dublin_default_valid_idx_05);
```

```
    dublin_default_distance_col_05 =
dublin_default_distance_col_05(dublin_default_valid_idx_05);

    % Perform integration only if there are sufficient data points
    if length(dublin_default_distance_col_05) > 1
        % Sort distance and netheat by distance values for correct integration
        [dublin_default_distance_col_05, dublin_default_sortIdx_05] =
sort(dublin_default_distance_col_05);
        dublin_default_netheat_col_05 =
dublin_default_netheat_col_05(dublin_default_sortIdx_05);

        % Perform numerical integration using the trapezoidal rule
        dublin_default_integration_results_05(i) = trapz(dublin_default_distance_col_05,
dublin_default_netheat_col_05);
    else
        % Not enough data to integrate
        dublin_default_integration_results_05(i) = NaN;
    end
end


% Load the CSV file
dublin_default_density_05 = readmatrix('dublin_default_density_05.csv'); % Replace with
the actual filename

% Extract the first row of each column
dublin_default_first_row_values_density_05 = dublin_default_density_05(1, :);

% Load the CSV file
dublin_default_sensibleenthalpy_05 = readmatrix('dublin_default_sensibleenthalpy_05.csv');
% Replace with the actual filename

% Get the number of columns
dublin_default_numColumns_sense_05 = size(dublin_default_sensibleenthalpy_05, 2);

% Initialize matrices to store first and last valid values
dublin_default_sensible_enthalpy_first_05 = zeros(1, dublin_default_numColumns_sense_05);
dublin_default_sensible_enthalpy_last_05 = zeros(1, dublin_default_numColumns_sense_05);

% Loop through each column
for i = 1:dublin_default_numColumns_sense_05
    % Extract the column
    dublin_default_column_data_05 = dublin_default_sensibleenthalpy_05(:, i);

    % Remove NaN values to get valid data
    dublin_default_valid_data_05 =
dublin_default_column_data_05(~isnan(dublin_default_column_data_05));

    % Get the first and last values from the valid data
    if ~isempty(dublin_default_valid_data_05)
        dublin_default_sensible_enthalpy_first_05(i) = dublin_default_valid_data_05(1); %
First value
        dublin_default_sensible_enthalpy_last_05(i) =
max(dublin_default_valid_data_05(:,end)); % Last value
    else
        % If the column is entirely NaN, assign NaN
        dublin_default_sensible_enthalpy_first_05(i) = NaN;
        dublin_default_sensible_enthalpy_last_05(  i) = NaN;
    end
end

dublin_default_xyz_05 = dublin_default_sensible_enthalpy_last_05-
dublin_default_sensible_enthalpy_first_05;

dublin_default_den_05 = dublin_default_first_row_values_density_05.*dublin_default_xyz_05;
```

```
 dublin_default_SC_05 = dublin_default_integration_results_05./(2.*dublin_default_den_05);

 % Load the strain rate data from the CSV file
 dublin_default_axialvel_05 = readmatrix('dublin_default_axialvel_05.csv');

 % Remove rows with NaN values
 dublin_default_first_row_values_axialvel_05 = dublin_default_axialvel_05(1, :);
 dublin_default_max_dist_05 = max(dublin_default_distance_05, [], 1, 'omitnan');

 % Find the maximum value in each column
 % dublin_default_max_strainrate_05 = max(dublin_default_strainrate_05, [], 1); % Row
vector with max values for each column
 % L = 0.014;
 dublin_default_strainrate_05 =
2*dublin_default_first_row_values_axialvel_05./dublin_default_max_dist_05;


  % Load the strain rate data from the CSV file
 dublin_default_maxtemp_05 = readmatrix('dublin_default_temperature_05.csv');

 % Remove rows with NaN values
 dublin_default_max_temp_05 = max(dublin_default_maxtemp_05, [], 1, 'omitnan');

%% Konnov 05

  % Load the CSV files
 konnov_default_netheat_05 = readmatrix('konnov_default_netheat_05.csv'); % Replace with
the actual filename
 konnov_default_distance_05 = readmatrix('konnov_default_distance_05.csv'); % Replace with
the actual filename

 % Determine the number of columns
 konnov_default_numColumns_integrate_05 = min(size(konnov_default_netheat_05, 2),
size(konnov_default_distance_05, 2));

 % Initialize the results matrix
 konnov_default_integration_results_05 = zeros(1, konnov_default_numColumns_integrate_05);

 % Loop through each column
 for i = 1:konnov_default_numColumns_integrate_05
      % Extract current column for netheat and distance
     konnov_default_netheat_col_05 = konnov_default_netheat_05(:, i);
     konnov_default_distance_col_05 = konnov_default_distance_05(:, i);

     % Remove NaN values and ensure both arrays align
     konnov_default_valid_idx_05 = ~isnan(konnov_default_netheat_col_05) &
~isnan(konnov_default_distance_col_05);
     konnov_default_netheat_col_05 =
konnov_default_netheat_col_05(konnov_default_valid_idx_05);
     konnov_default_distance_col_05 =
konnov_default_distance_col_05(konnov_default_valid_idx_05);

     % Perform integration only if there are sufficient data points
     if length(konnov_default_distance_col_05) > 1
         % Sort distance and netheat by distance values for correct integration
         [konnov_default_distance_col_05, konnov_default_sortIdx_05] =
sort(konnov_default_distance_col_05);
         konnov_default_netheat_col_05 =
konnov_default_netheat_col_05(konnov_default_sortIdx_05);

         % Perform numerical integration using the trapezoidal rule
         konnov_default_integration_results_05(i) = trapz(konnov_default_distance_col_05,
konnov_default_netheat_col_05);
     else
         % Not enough data to integrate
         konnov_default_integration_results_05(i) = NaN;
```

```
    end
end


% Load the CSV file
konnov_default_density_05 = readmatrix('konnov_default_density_05.csv'); % Replace with
the actual filename

% Extract the first row of each column
konnov_default_first_row_values_density_05 = konnov_default_density_05(1, :);

% Load the CSV file
konnov_default_sensibleenthalpy_05 = readmatrix('konnov_default_sensibleenthalpy_05.csv');
% Replace with the actual filename

% Get the number of columns
konnov_default_numColumns_sense_05 = size(konnov_default_sensibleenthalpy_05, 2);

% Initialize matrices to store first and last valid values
konnov_default_sensible_enthalpy_first_05 = zeros(1, konnov_default_numColumns_sense_05);
konnov_default_sensible_enthalpy_last_05 = zeros(1, konnov_default_numColumns_sense_05);

% Loop through each column
for i = 1:konnov_default_numColumns_sense_05
    % Extract the column
    konnov_default_column_data_05 = konnov_default_sensibleenthalpy_05(:, i);

    % Remove NaN values to get valid data
    konnov_default_valid_data_05 =
konnov_default_column_data_05(~isnan(konnov_default_column_data_05));

    % Get the first and last values from the valid data
    if ~isempty(konnov_default_valid_data_05)
        konnov_default_sensible_enthalpy_first_05(i) = konnov_default_valid_data_05(1); %
First value
        konnov_default_sensible_enthalpy_last_05(i) =
max(konnov_default_valid_data_05(:,end)); % Last value
    else
        % If the column is entirely NaN, assign NaN
        konnov_default_sensible_enthalpy_first_05(i) = NaN;
        konnov_default_sensible_enthalpy_last_05(i) = NaN;
    end
end

konnov_default_xyz_05 = konnov_default_sensible_enthalpy_last_05-
konnov_default_sensible_enthalpy_first_05;

konnov_default_den_05 = konnov_default_first_row_values_density_05.*konnov_default_xyz_05;

konnov_default_SC_05 = konnov_default_integration_results_05./(2.*konnov_default_den_05);

% Load the strain rate data from the CSV file
konnov_default_axialvel_05 = readmatrix('konnov_default_axialvel_05.csv');

% Remove rows with NaN values
konnov_default_first_row_values_axialvel_05 = konnov_default_axialvel_05(1, :);
konnov_default_max_dist_05 = max(konnov_default_distance_05, [], 1, 'omitnan');

% Find the maximum value in each column
% L = 0.014;
konnov_default_strainrate_05 =
2*konnov_default_first_row_values_axialvel_05./konnov_default_max_dist_05;


 % Load the strain rate data from the CSV file
konnov_default_maxtemp_05 = readmatrix('konnov_default_temperature_05.csv');
```

```
 % Remove rows with NaN values
 konnov_default_max_temp_05 = max(konnov_default_maxtemp_05, [], 1, 'omitnan');

 % Find the maximum value in each column
figure()
plot (dublin_default_strainrate_05(1:84),dublin_default_SC_05(1:84),'-', 'linewidth',3)
hold on
plot (princeton_default_strainrate_05(1:91),princeton_default_SC_05(1:91),'-',
'linewidth',3)
plot (konnov_default_strainrate_05(1:74),konnov_default_SC_05(1:74),'-', 'linewidth',3)
ylim([0.5 2])
legend('Dublin','Princeton','konnov','Location','northeast')
title('Consumption Velocity V/s Strain Rate - Phi 0.50')
xlabel('global strain rate in /s')
ylabel('consumption velocity in m/s');
hold off

figure()
plot (dublin_default_strainrate_05(1:84),dublin_default_max_temp_05(1:84),'--o',
'linewidth',1)
hold on
plot (princeton_default_strainrate_05(1:91),princeton_default_max_temp_05(1:91),'--o',
'linewidth',1)
plot (konnov_default_strainrate_05(1:74),konnov_default_max_temp_05(1:74),'--o',
'linewidth',1)
legend('Dublin','Princeton','konnov','Location','northeast')
title('Max Temperature V/s Strain Rate - Phi 0.50')
xlabel('global strain rate in /s')
ylabel('max temperature in K');
hold off
%% other plots

% Find the maximum value of y and its index
[max_dublin_default_max_temp_05, maxIndex_1] = max(dublin_default_max_temp_05);

% To find velocity at max temp to search parameters
corresponding_dublin_default_strainrate_05 = dublin_default_strainrate_05(maxIndex_1);
corresponding_dublin_default_SC_05 = dublin_default_SC_05(maxIndex_1);
corresponding_dublin_default_first_row_values_axialvel_05 =
dublin_default_first_row_values_axialvel_05(maxIndex_1);

% To find velocity at  to searconsumption velocityc parameters
 [max_dublin_default_SC_05, maxIndex_2] = max(dublin_default_SC_05);
 max_dublin_default_strainrate_05 = dublin_default_strainrate_05(maxIndex_2);
 max_dublin_default_first_row_values_axialvel_05 =
dublin_default_first_row_values_axialvel_05(maxIndex_2);

% Display the results
disp(['corresponding_dublin_default_first_row_values_axialvel_05(at max temp) ',
num2str(corresponding_dublin_default_first_row_values_axialvel_05)]);
disp(['max_dublin_default_first_row_values_axialvel_05(at max SC) ',
num2str(max_dublin_default_first_row_values_axialvel_05)]);

% Plotting Graphs
dublin_default_analysis_atmax_sc_05 = readmatrix('dublin_default_analysis_atmaxsc_05.csv');
dublin_default_analysis_atmax_temp_05 =
readmatrix('dublin_default_analysis_atmaxtemp_05.csv');
Xoffset = 0.001;
% Net Heat
figure()
plot
(dublin_default_analysis_atmax_sc_05(:,1)+Xoffset,dublin_default_analysis_atmax_sc_05(:,2),
'-', 'linewidth',2)
hold on
```

```
plot (dublin_default_analysis_atmax_temp_05(:,1),dublin_default_analysis_atmax_temp_05(:,
2),'-', 'linewidth',2)
legend('at max SC','at max temp','Location','northeast')
title('Netheat Release V/s Distance - Phi 0.50 (Dublin)')
xlabel('distance in m')
ylabel('net heat release in J/m^3s');
hold off

%Temperature
figure()
plot
(dublin_default_analysis_atmax_sc_05(:,1)+Xoffset,dublin_default_analysis_atmax_sc_05(:,8),
'-', 'linewidth',2)
hold on
plot (dublin_default_analysis_atmax_temp_05(:,1),dublin_default_analysis_atmax_temp_05(:,
7),'-', 'linewidth',2)
legend('at max SC','at max temp','Location','northeast')
title('Temperature V/s Distance - Phi 0.50 (Dublin)')
xlabel('distance in m')
ylabel('temperature in K');
hold off

% (e)-(i) mass (or mole) fractions of H, O, OH, HO2, and H2O2
% O mass fraction
figure()
plot
(dublin_default_analysis_atmax_sc_05(:,1)+Xoffset,dublin_default_analysis_atmax_sc_05(:,15)
,'-', 'linewidth',2)
hold on
plot (dublin_default_analysis_atmax_temp_05(:,1),dublin_default_analysis_atmax_temp_05(:,
14),'-', 'linewidth',2)
legend('at max SC','at max temp','Location','northeast')
title('O - Mole Fraction V/s Distance - Phi 0.50 (Dublin)')
xlabel('distance in m')
ylabel('O Mole Fraction');
hold off

% H mass fraction
figure()
plot
(dublin_default_analysis_atmax_sc_05(:,1)+Xoffset,dublin_default_analysis_atmax_sc_05(:,16)
,'-', 'linewidth',2)
hold on
plot (dublin_default_analysis_atmax_temp_05(:,1),dublin_default_analysis_atmax_temp_05(:,
15),'-', 'linewidth',2)
legend('at max SC','at max temp','Location','northeast')
title('H - Mole Fraction V/s Distance - Phi 0.50 (Dublin)')
xlabel('distance in m')
ylabel('H Mole Fraction');
hold off

% OH mass fraction
figure()
plot
(dublin_default_analysis_atmax_sc_05(:,1)+Xoffset,dublin_default_analysis_atmax_sc_05(:,17)
,'-', 'linewidth',2)
hold on
plot (dublin_default_analysis_atmax_temp_05(:,1),dublin_default_analysis_atmax_temp_05(:,
16),'-', 'linewidth',2)
legend('at max SC','at max temp','Location','northeast')
title('OH - Mole Fraction V/s Distance - Phi 0.50 (Dublin)')
xlabel('distance in m')
ylabel('OH Mole Fraction');
hold off
```

```
% HO2 mass fraction
figure()
plot
(dublin_default_analysis_atmax_sc_05(:,1)+Xoffset,dublin_default_analysis_atmax_sc_05(:,18)
,'-', 'linewidth',2)
hold on
plot (dublin_default_analysis_atmax_temp_05(:,1),dublin_default_analysis_atmax_temp_05(:,
17),'-', 'linewidth',2)
legend('at max SC','at max temp','Location','northeast')
title('HO2 - Mole Fraction V/s Distance - Phi 0.50 (Dublin)')
xlabel('distance in m')
ylabel('HO2 Mole Fraction');
hold off

%H2O2 mass fraction
figure()
plot
(dublin_default_analysis_atmax_sc_05(:,1)+Xoffset,dublin_default_analysis_atmax_sc_05(:,19)
,'-', 'linewidth',2)
hold on
plot (dublin_default_analysis_atmax_temp_05(:,1),dublin_default_analysis_atmax_temp_05(:,
18),'-', 'linewidth',2)
legend('at max SC','at max temp','Location','northeast')
title('H2O2 - Mole Fraction V/s Distance - Phi 0.50 (Dublin)')
xlabel('distance in m')
ylabel('H2O2 Mole Fraction');
hold off

% Local Equivalence Ratio
dublin_default_xHtotal_atmaxsc_05 = 2*dublin_default_analysis_atmax_sc_05(:,12) +
2*dublin_default_analysis_atmax_sc_05(:,14) + 1*dublin_default_analysis_atmax_sc_05(:,17)+
1*dublin_default_analysis_atmax_sc_05(:,16)+ 2*dublin_default_analysis_atmax_sc_05(:,19)+
1*dublin_default_analysis_atmax_sc_05(:,18);
dublin_default_xOtotal_atmaxsc_05 = 2*dublin_default_analysis_atmax_sc_05(:,13) +
1*dublin_default_analysis_atmax_sc_05(:,14) + 1*dublin_default_analysis_atmax_sc_05(:,17) +
1*dublin_default_analysis_atmax_sc_05(:,15) + 2*dublin_default_analysis_atmax_sc_05(:,19)+
2*dublin_default_analysis_atmax_sc_05(:,18);

dublin_default_localphi_atmaxsc_05 =
dublin_default_xHtotal_atmaxsc_05./(2*dublin_default_xOtotal_atmaxsc_05);

dublin_default_xHtotal_atmaxtemp_05 = 2*dublin_default_analysis_atmax_temp_05(:,11) +
2*dublin_default_analysis_atmax_temp_05(:,13) +
1*dublin_default_analysis_atmax_temp_05(:,16)+
1*dublin_default_analysis_atmax_temp_05(:,15)+
2*dublin_default_analysis_atmax_temp_05(:,18)+
1*dublin_default_analysis_atmax_temp_05(:,17);
dublin_default_xOtotal_atmaxtemp_05 = 2*dublin_default_analysis_atmax_temp_05(:,12) +
1*dublin_default_analysis_atmax_temp_05(:,13) +
1*dublin_default_analysis_atmax_temp_05(:,16) +
1*dublin_default_analysis_atmax_temp_05(:,14) +
2*dublin_default_analysis_atmax_temp_05(:,18)+
2*dublin_default_analysis_atmax_temp_05(:,17);

dublin_default_localphi_atmaxtemp_05 =
dublin_default_xHtotal_atmaxtemp_05./(2*dublin_default_xOtotal_atmaxtemp_05);

figure()
plot
(dublin_default_analysis_atmax_sc_05(:,1)+Xoffset,dublin_default_localphi_atmaxsc_05,'-',
'linewidth',2)
hold on
plot (dublin_default_analysis_atmax_temp_05(:,1),dublin_default_localphi_atmaxtemp_05,'-',
'linewidth',2)
legend('at max SC','at max temp','Location','northeast')
title('Local Equivalence Ratio V/s Distance - Phi 0.50 (Dublin)')
```

```
xlabel('distance in m')
ylabel('local equivalence ratio');
hold off

% to calculate cF=1-YH2/YH2,u
dublin_default_cf_atmaxsc_05 = 1-(dublin_default_analysis_atmax_sc_05(:,12)./0.1736);
dublin_default_cf_atmaxtemp_05 = 1-(dublin_default_analysis_atmax_temp_05(:,11)./0.1736);
figure()
plot (dublin_default_analysis_atmax_sc_05(:,1)+Xoffset,dublin_default_cf_atmaxsc_05,'-',
'linewidth',2)
hold on
plot (dublin_default_analysis_atmax_temp_05(:,1),dublin_default_cf_atmaxtemp_05,'-',
'linewidth',2)
legend('at max SC','at max temp','Location','northeast')
title('Cf V/s Distance - Phi 0.50 (Dublin)')
xlabel('distance in m')
ylabel('Cf');
hold off

% heat release rate as a function of cF
figure()
plot (dublin_default_cf_atmaxsc_05+Xoffset, dublin_default_analysis_atmax_sc_05(:,2),'-',
'linewidth',2)
hold on
plot (dublin_default_cf_atmaxtemp_05,dublin_default_analysis_atmax_temp_05(:,2),'-',
'linewidth',2)
legend('at max SC','at max temp','Location','northeast')
title('Net Heat V/s Cf - Phi 0.50 (Dublin)')
xlabel('Cf')
ylabel('Net Heat in J/m^3s');
hold off

% Temperature as a function of cF
figure()
plot (dublin_default_cf_atmaxsc_05+Xoffset, dublin_default_analysis_atmax_sc_05(:,8),'-',
'linewidth',2)
hold on
plot (dublin_default_cf_atmaxtemp_05,dublin_default_analysis_atmax_temp_05(:,7),'-',
'linewidth',2)
legend('at max SC','at max temp','Location','northeast')
title('Temperature V/s Cf - Phi 0.50 (Dublin)')
xlabel('Cf')
ylabel('temperature in K');
hold off

%% princeton 08
%Phi 08

  % Load the CSV files
 princeton_default_netheat_08 = readmatrix('princeton_default_netheat_08.csv'); % Replace
with the actual filename
 princeton_default_distance_08 = readmatrix('princeton_default_distance_08.csv'); % Replace
with the actual filename

 % Determine the number of columns
 princeton_default_numColumns_integrate_08 = min(size(princeton_default_netheat_08, 2),
size(princeton_default_distance_08, 2));

 % Initialize the results matrix
 princeton_default_integration_results_08 = zeros(1,
princeton_default_numColumns_integrate_08);

 % Loop through each column
 for i = 1:princeton_default_numColumns_integrate_08
      % Extract current column for netheat and distance
     princeton_default_netheat_col_08 = princeton_default_netheat_08(:, i);
```

```
    princeton_default_distance_col_08 = princeton_default_distance_08(:, i);

    % Remove NaN values and ensure both arrays align
    princeton_default_valid_idx_08 = ~isnan(princeton_default_netheat_col_08) &
~isnan(princeton_default_distance_col_08);
    princeton_default_netheat_col_08 =
princeton_default_netheat_col_08(princeton_default_valid_idx_08);
    princeton_default_distance_col_08 =
princeton_default_distance_col_08(princeton_default_valid_idx_08);

    % Perform integration only if there are sufficient data points
    if length(princeton_default_distance_col_08) > 1
        % Sort distance and netheat by distance values for correct integration
        [princeton_default_distance_col_08, princeton_default_sortIdx_08] =
sort(princeton_default_distance_col_08);
        princeton_default_netheat_col_08 =
princeton_default_netheat_col_08(princeton_default_sortIdx_08);

        % Perform numerical integration using the trapezoidal rule
        princeton_default_integration_results_08(i) =
trapz(princeton_default_distance_col_08, princeton_default_netheat_col_08);
    else
        % Not enough data to integrate
        princeton_default_integration_results_08(i) = NaN;
    end
end

% Load the CSV file
princeton_default_density_08 = readmatrix('princeton_default_density_08.csv'); % Replace
with the actual filename

% Extract the first row of each column
princeton_default_first_row_values_density_08 = princeton_default_density_08(1, :);


% Load the CSV file
princeton_default_sensibleenthalpy_08 =
readmatrix('princeton_default_sensibleenthalpy_08.csv'); % Replace with the actual filename

% Get the number of columns
princeton_default_numColumns_sense_08 = size(princeton_default_sensibleenthalpy_08, 2);

% Initialize matrices to store first and last valid values
princeton_default_sensible_enthalpy_first_08 = zeros(1,
princeton_default_numColumns_sense_08);
princeton_default_sensible_enthalpy_last_08 = zeros(1,
princeton_default_numColumns_sense_08);

% Loop through each column
for i = 1:princeton_default_numColumns_sense_08
    % Extract the column
    princeton_default_column_data_08 = princeton_default_sensibleenthalpy_08(:, i);

    % Remove NaN values to get valid data
    princeton_default_valid_data_08 =
princeton_default_column_data_08(~isnan(princeton_default_column_data_08));

    % Get the first and last values from the valid data
    if ~isempty(princeton_default_valid_data_08)
        princeton_default_sensible_enthalpy_first_08(i) =
princeton_default_valid_data_08(1); % First value
        princeton_default_sensible_enthalpy_last_08(i) =
max(princeton_default_valid_data_08(:,end)); % Last value
    else
        % If the column is entirely NaN, assign NaN
        princeton_default_sensible_enthalpy_first_08(i) = NaN;
```

```
        princeton_default_sensible_enthalpy_last_08(i) = NaN;
    end
end

princeton_default_xyz_08 = princeton_default_sensible_enthalpy_last_08-
princeton_default_sensible_enthalpy_first_08;

princeton_default_den_08 =
princeton_default_first_row_values_density_08.*princeton_default_xyz_08;

princeton_default_SC_08 =
princeton_default_integration_results_08./(2.*princeton_default_den_08);

% Load the strain rate data from the CSV file
princeton_default_axialvel_08 = readmatrix('princeton_default_axialvel_08.csv');

% Remove rows with NaN values
princeton_default_first_row_values_axialvel_08 = princeton_default_axialvel_08(1, :);
princeton_default_max_dist_08 = max(princeton_default_distance_08, [], 1, 'omitnan');

% Find the maximum value in each column
% L = 0.014;
princeton_default_strainrate_08 =
2*princeton_default_first_row_values_axialvel_08./princeton_default_max_dist_08;


 % Load the strain rate data from the CSV file
princeton_default_maxtemp_08 = readmatrix('princeton_default_temperature_08.csv');

% Remove rows with NaN values
princeton_default_max_temp_08 = max(princeton_default_maxtemp_08, [], 1, 'omitnan');

%% Dublin08
% Dublin 08

% Load the CSV files
dublin_default_netheat_08 = readmatrix('dublin_default_netheat_08.csv'); % Replace with
the actual filename
dublin_default_distance_08 = readmatrix('dublin_default_distance_08.csv'); % Replace with
the actual filename

% Determine the number of columns
dublin_default_numColumns_integrate_08 = min(size(dublin_default_netheat_08, 2),
size(dublin_default_distance_08, 2));

% Initialize the results matrix
dublin_default_integration_results_08 = zeros(1, dublin_default_numColumns_integrate_08);

% Loop through each column
for i = 1:dublin_default_numColumns_integrate_08
     % Extract current column for netheat and distance
    dublin_default_netheat_col_08 = dublin_default_netheat_08(:, i);
    dublin_default_distance_col_08 = dublin_default_distance_08(:, i);

    % Remove NaN values and ensure both arrays align
    dublin_default_valid_idx_08 = ~isnan(dublin_default_netheat_col_08) &
~isnan(dublin_default_distance_col_08);
    dublin_default_netheat_col_08 =
dublin_default_netheat_col_08(dublin_default_valid_idx_08);
    dublin_default_distance_col_08 =
dublin_default_distance_col_08(dublin_default_valid_idx_08);

    % Perform integration only if there are sufficient data points
    if length(dublin_default_distance_col_08) > 1
        % Sort distance and netheat by distance values for correct integration
```

```
        [dublin_default_distance_col_08, dublin_default_sortIdx_08] =
sort(dublin_default_distance_col_08);
        dublin_default_netheat_col_08 =
dublin_default_netheat_col_08(dublin_default_sortIdx_08);

        % Perform numerical integration using the trapezoidal rule
        dublin_default_integration_results_08(i) = trapz(dublin_default_distance_col_08,
dublin_default_netheat_col_08);
    else
        % Not enough data to integrate
        dublin_default_integration_results_08(i) = NaN;
    end
end

% Load the CSV file
dublin_default_density_08 = readmatrix('dublin_default_density_08.csv'); % Replace with
the actual filename

% Extract the first row of each column
dublin_default_first_row_values_density_08 = dublin_default_density_08(1, :);

% Load the CSV file
dublin_default_sensibleenthalpy_08 = readmatrix('dublin_default_sensibleenthalpy_08.csv');
% Replace with the actual filename

% Get the number of columns
dublin_default_numColumns_sense_08 = size(dublin_default_sensibleenthalpy_08, 2);

% Initialize matrices to store first and last valid values
dublin_default_sensible_enthalpy_first_08 = zeros(1, dublin_default_numColumns_sense_08);
dublin_default_sensible_enthalpy_last_08 = zeros(1, dublin_default_numColumns_sense_08);

% Loop through each column
for i = 1:dublin_default_numColumns_sense_08
    % Extract the column
    dublin_default_column_data_08 = dublin_default_sensibleenthalpy_08(:, i);

    % Remove NaN values to get valid data
    dublin_default_valid_data_08 =
dublin_default_column_data_08(~isnan(dublin_default_column_data_08));

    % Get the first and last values from the valid data
    if ~isempty(dublin_default_valid_data_08)
        dublin_default_sensible_enthalpy_first_08(i) = dublin_default_valid_data_08(1); %
First value
        dublin_default_sensible_enthalpy_last_08(i) =
max(dublin_default_valid_data_08(:,end)); % Last value
    else
        % If the column is entirely NaN, assign NaN
        dublin_default_sensible_enthalpy_first_08(i) = NaN;
        dublin_default_sensible_enthalpy_last_08(  i) = NaN;
    end
end

dublin_default_xyz_08 = dublin_default_sensible_enthalpy_last_08-
dublin_default_sensible_enthalpy_first_08;

dublin_default_den_08 = dublin_default_first_row_values_density_08.*dublin_default_xyz_08;

dublin_default_SC_08 = dublin_default_integration_results_08./(2.*dublin_default_den_08);

% Load the strain rate data from the CSV file
dublin_default_axialvel_08 = readmatrix('dublin_default_axialvel_08.csv');
```

```
% Remove rows with NaN values
dublin_default_first_row_values_axialvel_08 = dublin_default_axialvel_08(1, :);
dublin_default_max_dist_08 = max(dublin_default_distance_08, [], 1, 'omitnan');

% Find the maximum value in each column
% L = 0.014;
dublin_default_strainrate_08 =
2*dublin_default_first_row_values_axialvel_08./dublin_default_max_dist_08;


 % Load the strain rate data from the CSV file
dublin_default_maxtemp_08 = readmatrix('dublin_default_temperature_08.csv');

% Remove rows with NaN values
dublin_default_max_temp_08 = max(dublin_default_maxtemp_08, [], 1, 'omitnan');

% Find the maximum value in each column
figure()
plot (dublin_default_strainrate_08(1:100),dublin_default_SC_08(1:100),'-', 'linewidth',3)
hold on
plot (princeton_default_strainrate_08(1:90),princeton_default_SC_08(1:90),'-',
'linewidth',3)
ylim([1 2])
legend('Dublin', 'Princeton','Location','southeast')
title('Consumption Velocity V/s Strain Rate - Phi 0.80')
xlabel('global strain rate in /s')
ylabel('consumption velocity in m/s');
hold off

figure()
plot (dublin_default_strainrate_08(1:100),dublin_default_max_temp_08(1:100),'--o',
'linewidth',1)
hold on
plot (princeton_default_strainrate_08(1:90),princeton_default_max_temp_08(1:90),'--o',
'linewidth',1)
legend('Dublin','Princeton','Location','northeast')
title('Max Temperature V/s Strain Rate - Phi 0.80')
xlabel('global strain rate in /s')
ylabel('max temperature in K');
hold off
%% Konnov08

%% Pressure Effect
% 2 ATM

 % Load the CSV files
dublin_default_netheat_05_2atm = readmatrix('dublin_default_netheat_05_2atm.csv'); %
Replace with the actual filename
dublin_default_distance_05_2atm = readmatrix('dublin_default_distance_05_2atm.csv'); %
Replace with the actual filename

% Determine the number of columns
dublin_default_numColumns_integrate_05_2atm = min(size(dublin_default_netheat_05_2atm, 2),
size(dublin_default_distance_05_2atm, 2));

% Initialize the results matrix
dublin_default_integration_results_05_2atm = zeros(1,
dublin_default_numColumns_integrate_05_2atm);

% Loop through each column
for i = 1:dublin_default_numColumns_integrate_05_2atm
     % Extract current column for netheat and distance
    dublin_default_netheat_col_05_2atm = dublin_default_netheat_05_2atm(:, i);
    dublin_default_distance_col_05_2atm = dublin_default_distance_05_2atm(:, i);

    % Remove NaN values and ensure both arrays align
```

```
    dublin_default_valid_idx_05_2atm = ~isnan(dublin_default_netheat_col_05_2atm) &
~isnan(dublin_default_distance_col_05_2atm);
    dublin_default_netheat_col_05_2atm =
dublin_default_netheat_col_05_2atm(dublin_default_valid_idx_05_2atm);
    dublin_default_distance_col_05_2atm =
dublin_default_distance_col_05_2atm(dublin_default_valid_idx_05_2atm);

    % Perform integration only if there are sufficient data points
    if length(dublin_default_distance_col_05_2atm) > 1
        % Sort distance and netheat by distance values for correct integration
        [dublin_default_distance_col_05_2atm, dublin_default_sortIdx_05_2atm] =
sort(dublin_default_distance_col_05_2atm);
        dublin_default_netheat_col_05_2atm =
dublin_default_netheat_col_05_2atm(dublin_default_sortIdx_05_2atm);

        % Perform numerical integration using the trapezoidal rule
        dublin_default_integration_results_05_2atm(i) =
trapz(dublin_default_distance_col_05_2atm, dublin_default_netheat_col_05_2atm);
    else
        % Not enough data to integrate
        dublin_default_integration_results_05_2atm(i) = NaN;
    end
end


% Load the CSV file
dublin_default_density_05_2atm = readmatrix('dublin_default_density_05_2atm.csv'); %
Replace with the actual filename

% Extract the first row of each column
dublin_default_first_row_values_density_05_2atm = dublin_default_density_05_2atm(1, :);

% Load the CSV file
dublin_default_sensibleenthalpy_05_2atm =
readmatrix('dublin_default_sensibleenthalpy_05_2atm.csv'); % Replace with the actual
filename

% Get the number of columns
dublin_default_numColumns_sense_05_2atm = size(dublin_default_sensibleenthalpy_05_2atm,
2);

% Initialize matrices to store first and last valid values
dublin_default_sensible_enthalpy_first_05_2atm = zeros(1,
dublin_default_numColumns_sense_05_2atm);
dublin_default_sensible_enthalpy_last_05_2atm = zeros(1,
dublin_default_numColumns_sense_05_2atm);

% Loop through each column
for i = 1:dublin_default_numColumns_sense_05_2atm
    % Extract the column
    dublin_default_column_data_05_2atm = dublin_default_sensibleenthalpy_05_2atm(:, i);

    % Remove NaN values to get valid data
    dublin_default_valid_data_05_2atm =
dublin_default_column_data_05_2atm(~isnan(dublin_default_column_data_05_2atm));

    % Get the first and last values from the valid data
    if ~isempty(dublin_default_valid_data_05_2atm)
        dublin_default_sensible_enthalpy_first_05_2atm(i) =
dublin_default_valid_data_05_2atm(1); % First value
        dublin_default_sensible_enthalpy_last_05_2atm(i) =
max(dublin_default_valid_data_05_2atm(:,end)); % Last value
    else
        % If the column is entirely NaN, assign NaN
        dublin_default_sensible_enthalpy_first_05_2atm(i) = NaN;
        dublin_default_sensible_enthalpy_last_05_2atm(  i) = NaN;
```

```
    end
end

dublin_default_xyz_05_2atm = dublin_default_sensible_enthalpy_last_05_2atm-
dublin_default_sensible_enthalpy_first_05_2atm;

dublin_default_den_05_2atm =
dublin_default_first_row_values_density_05_2atm.*dublin_default_xyz_05_2atm;

dublin_default_SC_05_2atm =
dublin_default_integration_results_05_2atm./(2.*dublin_default_den_05_2atm);

% Load the strain rate data from the CSV file
dublin_default_axialvel_05_2atm = readmatrix('dublin_default_axialvel_05_2atm.csv');

% Remove rows with NaN values
dublin_default_first_row_values_axialvel_05_2atm = dublin_default_axialvel_05_2atm(1, :);
dublin_default_max_dist_05_2atm = max(dublin_default_distance_05_2atm, [], 1, 'omitnan');

% Find the maximum value in each column
% L = 0.014;
dublin_default_strainrate_05_2atm =
2*dublin_default_first_row_values_axialvel_05_2atm./dublin_default_max_dist_05_2atm;


 % Load the strain rate data from the CSV file
dublin_default_maxtemp_05_2atm = readmatrix('dublin_default_temperature_05_2atm.csv');

% Remove rows with NaN values
dublin_default_max_temp_05_2atm = max(dublin_default_maxtemp_05_2atm, [], 1, 'omitnan');

%% 3ATM

 % Load the CSV files
dublin_default_netheat_05_3atm = readmatrix('dublin_default_netheat_05_3atm.csv'); %
Replace with the actual filename
dublin_default_distance_05_3atm = readmatrix('dublin_default_distance_05_3atm.csv'); %
Replace with the actual filename

% Determine the number of columns
dublin_default_numColumns_integrate_05_3atm = min(size(dublin_default_netheat_05_3atm, 2),
size(dublin_default_distance_05_2atm, 2));

% Initialize the results matrix
dublin_default_integration_results_05_3atm = zeros(1,
dublin_default_numColumns_integrate_05_3atm);

% Loop through each column
for i = 1:dublin_default_numColumns_integrate_05_3atm
     % Extract current column for netheat and distance
    dublin_default_netheat_col_05_3atm = dublin_default_netheat_05_3atm(:, i);
    dublin_default_distance_col_05_3atm = dublin_default_distance_05_3atm(:, i);

    % Remove NaN values and ensure both arrays align
    dublin_default_valid_idx_05_3atm = ~isnan(dublin_default_netheat_col_05_3atm) &
~isnan(dublin_default_distance_col_05_3atm);
    dublin_default_netheat_col_05_3atm =
dublin_default_netheat_col_05_3atm(dublin_default_valid_idx_05_3atm);
    dublin_default_distance_col_05_3atm =
dublin_default_distance_col_05_3atm(dublin_default_valid_idx_05_3atm);

    % Perform integration only if there are sufficient data points
    if length(dublin_default_distance_col_05_3atm) > 1
        % Sort distance and netheat by distance values for correct integration
        [dublin_default_distance_col_05_3atm, dublin_default_sortIdx_05_3atm] =
sort(dublin_default_distance_col_05_3atm);
```

```
        dublin_default_netheat_col_05_3atm =
dublin_default_netheat_col_05_3atm(dublin_default_sortIdx_05_3atm);

        % Perform numerical integration using the trapezoidal rule
        dublin_default_integration_results_05_3atm(i) =
trapz(dublin_default_distance_col_05_3atm, dublin_default_netheat_col_05_3atm);
    else
        % Not enough data to integrate
        dublin_default_integration_results_05_3atm(i) = NaN;
    end
end

% Load the CSV file
dublin_default_density_05_3atm = readmatrix('dublin_default_density_05_3atm.csv'); %
Replace with the actual filename

% Extract the first row of each column
dublin_default_first_row_values_density_05_3atm = dublin_default_density_05_3atm(1, :);

% Load the CSV file
dublin_default_sensibleenthalpy_05_3atm =
readmatrix('dublin_default_sensibleenthalpy_05_3atm.csv'); % Replace with the actual
filename

% Get the number of columns
dublin_default_numColumns_sense_05_3atm = size(dublin_default_sensibleenthalpy_05_3atm,
2);

% Initialize matrices to store first and last valid values
dublin_default_sensible_enthalpy_first_05_3atm = zeros(1,
dublin_default_numColumns_sense_05_3atm);
dublin_default_sensible_enthalpy_last_05_3atm = zeros(1,
dublin_default_numColumns_sense_05_3atm);

% Loop through each column
for i = 1:dublin_default_numColumns_sense_05_3atm
    % Extract the column
    dublin_default_column_data_05_3atm = dublin_default_sensibleenthalpy_05_3atm(:, i);

    % Remove NaN values to get valid data
    dublin_default_valid_data_05_3atm =
dublin_default_column_data_05_3atm(~isnan(dublin_default_column_data_05_3atm));

    % Get the first and last values from the valid data
    if ~isempty(dublin_default_valid_data_05_3atm)
        dublin_default_sensible_enthalpy_first_05_3atm(i) =
dublin_default_valid_data_05_3atm(1); % First value
        dublin_default_sensible_enthalpy_last_05_3atm(i) =
max(dublin_default_valid_data_05_3atm(:,end)); % Last value
    else
        % If the column is entirely NaN, assign NaN
        dublin_default_sensible_enthalpy_first_05_3atm(i) = NaN;
        dublin_default_sensible_enthalpy_last_05_3atm(  i) = NaN;
    end
end

dublin_default_xyz_05_3atm = dublin_default_sensible_enthalpy_last_05_3atm-
dublin_default_sensible_enthalpy_first_05_3atm;

dublin_default_den_05_3atm =
dublin_default_first_row_values_density_05_3atm.*dublin_default_xyz_05_3atm;

dublin_default_SC_05_3atm =
dublin_default_integration_results_05_3atm./(2.*dublin_default_den_05_3atm);

% Load the strain rate data from the CSV file
```

```
% dublin_default_strainrate_05 = readmatrix('dublin_default_normalstrain_05.csv');
dublin_default_axialvel_05_3atm = readmatrix('dublin_default_axialvel_05_3atm.csv');

% Remove rows with NaN values
dublin_default_first_row_values_axialvel_05_3atm = dublin_default_axialvel_05_3atm(1, :);
dublin_default_max_dist_05_3atm = max(dublin_default_distance_05_3atm, [], 1, 'omitnan');

% Find the maximum value in each column
% L = 0.014;
dublin_default_strainrate_05_3atm =
2*dublin_default_first_row_values_axialvel_05_3atm./dublin_default_max_dist_05_3atm;


 % Load the strain rate data from the CSV file
dublin_default_maxtemp_05_3atm = readmatrix('dublin_default_temperature_05_3atm.csv');

% Remove rows with NaN values
dublin_default_max_temp_05_3atm = max(dublin_default_maxtemp_05_3atm, [], 1, 'omitnan');

figure()
plot (dublin_default_strainrate_05(1:84),dublin_default_SC_05(1:84),'-', 'linewidth',3)
hold on
plot (dublin_default_strainrate_05_2atm(1:170),dublin_default_SC_05_2atm(1:170),'-',
'linewidth',3)
plot (dublin_default_strainrate_05_3atm(1:164),dublin_default_SC_05_3atm(1:164),'-',
'linewidth',3)
ylim([0.40 2])
legend('1ATM', '2ATM','3ATM','Location','northeast')
title('Consumption Velocity V/s Strain Rate (Pressure Effects) - Phi 0.50 (Dublin)')
xlabel('global strain rate in /s')
ylabel('consumption velocity in m/s');
hold off
%
figure()
plot (dublin_default_strainrate_05(1:84),dublin_default_max_temp_05(1:84),'--o',
'linewidth',1)
hold on
plot (dublin_default_strainrate_05_2atm(1:170),dublin_default_max_temp_05_2atm(1:170),'--
o', 'linewidth',1)
plot (dublin_default_strainrate_05_3atm(1:164),dublin_default_max_temp_05_3atm(1:164),'--
o', 'linewidth',1)
legend('1ATM', '2ATM','3ATM','Location','northeast')
title('Max Temperature V/s Strain Rate (Pressure Effects) - Phi 0.50 (Dublin)')
xlabel('global strain rate in /s')
ylabel('max temperature in K');
hold off

%% temperature effects

%t500
 % Load the CSV files
dublin_default_netheat_05_t500 = readmatrix('dublin_default_netheat_05_t500.csv'); %
Replace with the actual filename
dublin_default_distance_05_t500 = readmatrix('dublin_default_distance_05_t500.csv'); %
Replace with the actual filename

% Determine the number of columns
dublin_default_numColumns_integrate_05_t500 = min(size(dublin_default_netheat_05_t500, 2),
size(dublin_default_distance_05_t500, 2));

% Initialize the results matrix
dublin_default_integration_results_05_t500 = zeros(1,
dublin_default_numColumns_integrate_05_t500);

% Loop through each column
for i = 1:dublin_default_numColumns_integrate_05_t500
```

```
      % Extract current column for netheat and distance
     dublin_default_netheat_col_05_t500 = dublin_default_netheat_05_t500(:, i);
     dublin_default_distance_col_05_t500 = dublin_default_distance_05_t500(:, i);

     % Remove NaN values and ensure both arrays align
     dublin_default_valid_idx_05_t500 = ~isnan(dublin_default_netheat_col_05_t500) &
~isnan(dublin_default_distance_col_05_t500);
     dublin_default_netheat_col_05_t500 =
dublin_default_netheat_col_05_t500(dublin_default_valid_idx_05_t500);
     dublin_default_distance_col_05_t500 =
dublin_default_distance_col_05_t500(dublin_default_valid_idx_05_t500);

     % Perform integration only if there are sufficient data points
     if length(dublin_default_distance_col_05_t500) > 1
         % Sort distance and netheat by distance values for correct integration
         [dublin_default_distance_col_05_t500, dublin_default_sortIdx_05_t500] =
sort(dublin_default_distance_col_05_t500);
         dublin_default_netheat_col_05_t500 =
dublin_default_netheat_col_05_t500(dublin_default_sortIdx_05_t500);

         % Perform numerical integration using the trapezoidal rule
         dublin_default_integration_results_05_t500(i) =
trapz(dublin_default_distance_col_05_t500, dublin_default_netheat_col_05_t500);
     else
         % Not enough data to integrate
         dublin_default_integration_results_05_t500(i) = NaN;
     end
 end

 % Load the CSV file
 dublin_default_density_05_t500 = readmatrix('dublin_default_density_05_t500.csv'); %
Replace with the actual filename

 % Extract the first row of each column
 dublin_default_first_row_values_density_05_t500 = dublin_default_density_05_t500(1, :);

 % Load the CSV file
 dublin_default_sensibleenthalpy_05_t500 =
readmatrix('dublin_default_sensibleenthalpy_05_t500.csv'); % Replace with the actual
filename

 % Get the number of columns
 dublin_default_numColumns_sense_05_t500 = size(dublin_default_sensibleenthalpy_05_t500,
2);

 % Initialize matrices to store first and last valid values
 dublin_default_sensible_enthalpy_first_05_t500 = zeros(1,
dublin_default_numColumns_sense_05_t500);
 dublin_default_sensible_enthalpy_last_05_t500 = zeros(1,
dublin_default_numColumns_sense_05_t500);

 % Loop through each column
 for i = 1:dublin_default_numColumns_sense_05_t500
     % Extract the column
     dublin_default_column_data_05_t500 = dublin_default_sensibleenthalpy_05_t500(:, i);

     % Remove NaN values to get valid data
     dublin_default_valid_data_05_t500 =
dublin_default_column_data_05_t500(~isnan(dublin_default_column_data_05_t500));

     % Get the first and last values from the valid data
     if ~isempty(dublin_default_valid_data_05_t500)
         dublin_default_sensible_enthalpy_first_05_t500(i) =
dublin_default_valid_data_05_t500(1); % First value
         dublin_default_sensible_enthalpy_last_05_t500(i) =
max(dublin_default_valid_data_05_t500(:,end)); % Last value
```

```
    else
        % If the column is entirely NaN, assign NaN
        dublin_default_sensible_enthalpy_first_05_t500(i) = NaN;
        dublin_default_sensible_enthalpy_last_05_t500(  i) = NaN;
    end
end

dublin_default_xyz_05_t500 = dublin_default_sensible_enthalpy_last_05_t500-
dublin_default_sensible_enthalpy_first_05_t500;

dublin_default_den_05_t500 =
dublin_default_first_row_values_density_05_t500.*dublin_default_xyz_05_t500;

dublin_default_SC_05_t500 =
dublin_default_integration_results_05_t500./(2.*dublin_default_den_05_t500);

% Load the strain rate data from the CSV file
% dublin_default_strainrate_05 = readmatrix('dublin_default_normalstrain_05.csv');
dublin_default_axialvel_05_t500 = readmatrix('dublin_default_axialvel_05_t500.csv');

% Remove rows with NaN values
dublin_default_first_row_values_axialvel_05_t500 = dublin_default_axialvel_05_t500(1, :);
dublin_default_max_dist_05_t500 = max(dublin_default_distance_05_t500, [], 1, 'omitnan');

% Find the maximum value in each column
% L = 0.014;
dublin_default_strainrate_05_t500 =
2*dublin_default_first_row_values_axialvel_05_t500./dublin_default_max_dist_05_t500;


 % Load the strain rate data from the CSV file
dublin_default_maxtemp_05_t500 = readmatrix('dublin_default_temperature_05_t500.csv');

% Remove rows with NaN values
dublin_default_max_temp_05_t500 = max(dublin_default_maxtemp_05_t500, [], 1, 'omitnan');

%% T=800K

 %t800
 % Load the CSV files
dublin_default_netheat_05_t800 = readmatrix('dublin_default_netheat_05_t800.csv'); %
Replace with the actual filename
dublin_default_distance_05_t800 = readmatrix('dublin_default_distance_05_t800.csv'); %
Replace with the actual filename

% Determine the number of columns
dublin_default_numColumns_integrate_05_t800 = min(size(dublin_default_netheat_05_t800, 2),
size(dublin_default_distance_05_t800, 2));

% Initialize the results matrix
dublin_default_integration_results_05_t800 = zeros(1,
dublin_default_numColumns_integrate_05_t800);

% Loop through each column
for i = 1:dublin_default_numColumns_integrate_05_t800
     % Extract current column for netheat and distance
    dublin_default_netheat_col_05_t800 = dublin_default_netheat_05_t800(:, i);
    dublin_default_distance_col_05_t800 = dublin_default_distance_05_t800(:, i);

    % Remove NaN values and ensure both arrays align
    dublin_default_valid_idx_05_t800 = ~isnan(dublin_default_netheat_col_05_t800) &
~isnan(dublin_default_distance_col_05_t800);
    dublin_default_netheat_col_05_t800 =
dublin_default_netheat_col_05_t800(dublin_default_valid_idx_05_t800);
    dublin_default_distance_col_05_t800 =
dublin_default_distance_col_05_t800(dublin_default_valid_idx_05_t800);
```

```
    % Perform integration only if there are sufficient data points
    if length(dublin_default_distance_col_05_t800) > 1
        % Sort distance and netheat by distance values for correct integration
        [dublin_default_distance_col_05_t800, dublin_default_sortIdx_05_t800] =
sort(dublin_default_distance_col_05_t800);
        dublin_default_netheat_col_05_t800 =
dublin_default_netheat_col_05_t800(dublin_default_sortIdx_05_t800);

        % Perform numerical integration using the trapezoidal rule
        dublin_default_integration_results_05_t800(i) =
trapz(dublin_default_distance_col_05_t800, dublin_default_netheat_col_05_t800);
    else
        % Not enough data to integrate
        dublin_default_integration_results_05_t800(i) = NaN;
    end
end

% Load the CSV file
dublin_default_density_05_t800 = readmatrix('dublin_default_density_05_t800.csv'); %
Replace with the actual filename

% Extract the first row of each column
dublin_default_first_row_values_density_05_t800 = dublin_default_density_05_t800(1, :);

% Load the CSV file
dublin_default_sensibleenthalpy_05_t800 =
readmatrix('dublin_default_sensibleenthalpy_05_t800.csv'); % Replace with the actual
filename

% Get the number of columns
dublin_default_numColumns_sense_05_t800 = size(dublin_default_sensibleenthalpy_05_t800,
2);

% Initialize matrices to store first and last valid values
dublin_default_sensible_enthalpy_first_05_t800 = zeros(1,
dublin_default_numColumns_sense_05_t800);
dublin_default_sensible_enthalpy_last_05_t800 = zeros(1,
dublin_default_numColumns_sense_05_t800);

% Loop through each column
for i = 1:dublin_default_numColumns_sense_05_t800
    % Extract the column
    dublin_default_column_data_05_t800 = dublin_default_sensibleenthalpy_05_t800(:, i);

    % Remove NaN values to get valid data
    dublin_default_valid_data_05_t800 =
dublin_default_column_data_05_t800(~isnan(dublin_default_column_data_05_t800));

    % Get the first and last values from the valid data
    if ~isempty(dublin_default_valid_data_05_t800)
        dublin_default_sensible_enthalpy_first_05_t800(i) =
dublin_default_valid_data_05_t800(1); % First value
        dublin_default_sensible_enthalpy_last_05_t800(i) =
max(dublin_default_valid_data_05_t800(:,end)); % Last value
    else
        % If the column is entirely NaN, assign NaN
        dublin_default_sensible_enthalpy_first_05_t800(i) = NaN;
        dublin_default_sensible_enthalpy_last_05_t800(  i) = NaN;
    end
end


dublin_default_xyz_05_t800 = dublin_default_sensible_enthalpy_last_05_t800-
dublin_default_sensible_enthalpy_first_05_t800;
```

```
 dublin_default_den_05_t800 =
dublin_default_first_row_values_density_05_t800.*dublin_default_xyz_05_t800;

 dublin_default_SC_05_t800 =
dublin_default_integration_results_05_t800./(2.*dublin_default_den_05_t800);

 % Load the strain rate data from the CSV file
 % dublin_default_strainrate_05 = readmatrix('dublin_default_normalstrain_05.csv');
 dublin_default_axialvel_05_t800 = readmatrix('dublin_default_axialvel_05_t800.csv');

 % Remove rows with NaN values
 dublin_default_first_row_values_axialvel_05_t800 = dublin_default_axialvel_05_t800(1, :);
 dublin_default_max_dist_05_t800 = max(dublin_default_distance_05_t800, [], 1, 'omitnan');

 % Find the maximum value in each column
 % L = 0.014;
 dublin_default_strainrate_05_t800 =
2*dublin_default_first_row_values_axialvel_05_t800./dublin_default_max_dist_05_t800;


  % Load the strain rate data from the CSV file
 dublin_default_maxtemp_05_t800 = readmatrix('dublin_default_temperature_05_t800.csv');

 % Remove rows with NaN values
 dublin_default_max_temp_05_t800 = max(dublin_default_maxtemp_05_t800, [], 1, 'omitnan');

 figure()
 plot (dublin_default_strainrate_05(1:84),dublin_default_SC_05(1:84),'-', 'linewidth',3)
 hold on
 plot (dublin_default_strainrate_05_t500(1:58),dublin_default_SC_05_t500(1:58),'-',
'linewidth',3)
 plot (dublin_default_strainrate_05_t800(1:68),dublin_default_SC_05_t800(1:68),'-',
'linewidth',3)
 ylim([0.40 8])
 legend('T(inlet)=298K', 'T(inlet)=500K','T(inlet)=800K','Location','southeast')
 title('Consumption Velocity V/s Strain Rate (Effect of inlet Temperature) - Phi 0.50
(Dublin)')
 xlabel('global strain rate in /s')
 ylabel('consumption velocity in m/s');
 hold off
%
 figure()
 plot (dublin_default_strainrate_05(1:84),dublin_default_max_temp_05(1:84),'--o',
'linewidth',1)
 hold on
 plot (dublin_default_strainrate_05_t500(1:58),dublin_default_max_temp_05_t500(1:58),'--o',
'linewidth',1)
 plot (dublin_default_strainrate_05_t800(1:end),dublin_default_max_temp_05_t800(1:end),'--
o', 'linewidth',1)
 legend('T(inlet)=298K', 'T(inlet)=500K','T(inlet)=800K','Location','northeast')
 title('Max Temperature V/s Strain Rate (Effect of inlet temperature) - Phi 0.50 (Dublin)')
 xlabel('global strain rate in /s')
 ylabel('max temperature in K');
 hold off

 %% Phi Effects

 %% Phi 036

  % Load the CSV files
 dublin_default_netheat_036 = readmatrix('dublin_default_netheat_036.csv'); % Replace with
the actual filename
 dublin_default_distance_036 = readmatrix('dublin_default_distance_036.csv'); % Replace
with the actual filename
```

```
 % Determine the number of columns
 dublin_default_numColumns_integrate_036 = min(size(dublin_default_netheat_036, 2),
size(dublin_default_distance_036, 2));

 % Initialize the results matrix
 dublin_default_integration_results_036 = zeros(1,
dublin_default_numColumns_integrate_036);

 % Loop through each column
 for i = 1:dublin_default_numColumns_integrate_036
      % Extract current column for netheat and distance
     dublin_default_netheat_col_036 = dublin_default_netheat_036(:, i);
     dublin_default_distance_col_036 = dublin_default_distance_036(:, i);

     % Remove NaN values and ensure both arrays align
     dublin_default_valid_idx_036 = ~isnan(dublin_default_netheat_col_036) &
~isnan(dublin_default_distance_col_036);
     dublin_default_netheat_col_036 =
dublin_default_netheat_col_036(dublin_default_valid_idx_036);
     dublin_default_distance_col_036 =
dublin_default_distance_col_036(dublin_default_valid_idx_036);

     % Perform integration only if there are sufficient data points
     if length(dublin_default_distance_col_036) > 1
         % Sort distance and netheat by distance values for correct integration
         [dublin_default_distance_col_036, dublin_default_sortIdx_036] =
sort(dublin_default_distance_col_036);
         dublin_default_netheat_col_036 =
dublin_default_netheat_col_036(dublin_default_sortIdx_036);

         % Perform numerical integration using the trapezoidal rule
         dublin_default_integration_results_036(i) = trapz(dublin_default_distance_col_036,
dublin_default_netheat_col_036);
     else
         % Not enough data to integrate
         dublin_default_integration_results_036(i) = NaN;
     end
 end

 % Load the CSV file
 dublin_default_density_036 = readmatrix('dublin_default_density_036.csv'); % Replace with
the actual filename

 % Extract the first row of each column
 dublin_default_first_row_values_density_036 = dublin_default_density_036(1, :);

 % Load the CSV file
 dublin_default_sensibleenthalpy_036 =
readmatrix('dublin_default_sensibleenthalpy_036.csv'); % Replace with the actual filename

 % Get the number of columns
 dublin_default_numColumns_sense_036 = size(dublin_default_sensibleenthalpy_036, 2);

 % Initialize matrices to store first and last valid values
 dublin_default_sensible_enthalpy_first_036 = zeros(1,
dublin_default_numColumns_sense_036);
 dublin_default_sensible_enthalpy_last_036 = zeros(1, dublin_default_numColumns_sense_036);

 % Loop through each column
 for i = 1:dublin_default_numColumns_sense_036
     % Extract the column
     dublin_default_column_data_036 = dublin_default_sensibleenthalpy_036(:, i);

     % Remove NaN values to get valid data
     dublin_default_valid_data_036 =
dublin_default_column_data_036(~isnan(dublin_default_column_data_036));
```

```
     % Get the first and last values from the valid data
     if ~isempty(dublin_default_valid_data_036)
         dublin_default_sensible_enthalpy_first_036(i) = dublin_default_valid_data_036(1);
% First value
         dublin_default_sensible_enthalpy_last_036(i) =
max(dublin_default_valid_data_036(:,end)); % Last value
     else
         % If the column is entirely NaN, assign NaN
         dublin_default_sensible_enthalpy_first_036(i) = NaN;
         dublin_default_sensible_enthalpy_last_036(i) = NaN;
     end
 end

 dublin_default_xyz_036 = dublin_default_sensible_enthalpy_last_036-
dublin_default_sensible_enthalpy_first_036;

 dublin_default_den_036 =
dublin_default_first_row_values_density_036.*dublin_default_xyz_036;

 dublin_default_SC_036 =
dublin_default_integration_results_036./(2.*dublin_default_den_036);

 % Load the strain rate data from the CSV file
 dublin_default_axialvel_036 = readmatrix('dublin_default_axialvel_036.csv');

 % Remove rows with NaN values
 dublin_default_first_row_values_axialvel_036 = dublin_default_axialvel_036(1, :);
 dublin_default_max_dist_036 = max(dublin_default_distance_036, [], 1, 'omitnan');

 % Find the maximum value in each column
 % L = 0.014;
 dublin_default_strainrate_036 =
2*dublin_default_first_row_values_axialvel_036./dublin_default_max_dist_036;


  % Load the strain rate data from the CSV file
 dublin_default_maxtemp_036 = readmatrix('dublin_default_temperature_036.csv');

 % Remove rows with NaN values
dublin_default_max_temp_036 = max(dublin_default_maxtemp_036, [], 1, 'omitnan');

%% 0.40
 % Load the CSV files
 dublin_default_netheat_040 = readmatrix('dublin_default_netheat_040.csv'); % Replace with
the actual filename
 dublin_default_distance_040 = readmatrix('dublin_default_distance_040.csv'); % Replace
with the actual filename

 % Determine the number of columns
 dublin_default_numColumns_integrate_040 = min(size(dublin_default_netheat_040, 2),
size(dublin_default_distance_040, 2));

 % Initialize the results matrix
 dublin_default_integration_results_040 = zeros(1,
dublin_default_numColumns_integrate_040);

 % Loop through each column
 for i = 1:dublin_default_numColumns_integrate_040
      % Extract current column for netheat and distance
     dublin_default_netheat_col_040 = dublin_default_netheat_040(:, i);
     dublin_default_distance_col_040 = dublin_default_distance_040(:, i);

     % Remove NaN values and ensure both arrays align
     dublin_default_valid_idx_040 = ~isnan(dublin_default_netheat_col_040) &
~isnan(dublin_default_distance_col_040);
```

```
    dublin_default_netheat_col_040 =
dublin_default_netheat_col_040(dublin_default_valid_idx_040);
    dublin_default_distance_col_040 =
dublin_default_distance_col_040(dublin_default_valid_idx_040);

    % Perform integration only if there are sufficient data points
    if length(dublin_default_distance_col_040) > 1
        % Sort distance and netheat by distance values for correct integration
        [dublin_default_distance_col_040, dublin_default_sortIdx_040] =
sort(dublin_default_distance_col_040);
        dublin_default_netheat_col_040 =
dublin_default_netheat_col_040(dublin_default_sortIdx_040);

        % Perform numerical integration using the trapezoidal rule
        dublin_default_integration_results_040(i) = trapz(dublin_default_distance_col_040,
dublin_default_netheat_col_040);
    else
        % Not enough data to integrate
        dublin_default_integration_results_040(i) = NaN;
    end
end

% Load the CSV file
dublin_default_density_040 = readmatrix('dublin_default_density_040.csv'); % Replace with
the actual filename

% Extract the first row of each column
dublin_default_first_row_values_density_040 = dublin_default_density_040(1, :);

% Load the CSV file
dublin_default_sensibleenthalpy_040 =
readmatrix('dublin_default_sensibleenthalpy_040.csv'); % Replace with the actual filename

% Get the number of columns
dublin_default_numColumns_sense_040 = size(dublin_default_sensibleenthalpy_040, 2);

% Initialize matrices to store first and last valid values
dublin_default_sensible_enthalpy_first_040 = zeros(1,
dublin_default_numColumns_sense_040);
dublin_default_sensible_enthalpy_last_040 = zeros(1, dublin_default_numColumns_sense_040);

% Loop through each column
for i = 1:dublin_default_numColumns_sense_040
    % Extract the column
    dublin_default_column_data_040 = dublin_default_sensibleenthalpy_040(:, i);

    % Remove NaN values to get valid data
    dublin_default_valid_data_040 =
dublin_default_column_data_040(~isnan(dublin_default_column_data_040));

    % Get the first and last values from the valid data
    if ~isempty(dublin_default_valid_data_05)
        dublin_default_sensible_enthalpy_first_040(i) = dublin_default_valid_data_040(1);
% First value
        dublin_default_sensible_enthalpy_last_040(i) =
max(dublin_default_valid_data_040(:,end)); % Last value
    else
        % If the column is entirely NaN, assign NaN
        dublin_default_sensible_enthalpy_first_040(i) = NaN;
        dublin_default_sensible_enthalpy_last_040(i) = NaN;
    end
end

dublin_default_xyz_040 = dublin_default_sensible_enthalpy_last_040-
dublin_default_sensible_enthalpy_first_040;
```

```
 dublin_default_den_040 =
dublin_default_first_row_values_density_040.*dublin_default_xyz_040;

 dublin_default_SC_040 =
dublin_default_integration_results_040./(2.*dublin_default_den_040);

 % Load the strain rate data from the CSV file
 dublin_default_axialvel_040 = readmatrix('dublin_default_axialvel_040.csv');

 % Remove rows with NaN values
 dublin_default_first_row_values_axialvel_040 = dublin_default_axialvel_040(1, :);
 dublin_default_max_dist_040 = max(dublin_default_distance_040, [], 1, 'omitnan');

 % Find the maximum value in each column
 % L = 0.014;
 dublin_default_strainrate_040 =
2*dublin_default_first_row_values_axialvel_040./dublin_default_max_dist_040;


  % Load the strain rate data from the CSV file
 dublin_default_maxtemp_040 = readmatrix('dublin_default_temperature_040.csv');

 % Remove rows with NaN values
 dublin_default_max_temp_040 = max(dublin_default_maxtemp_040, [], 1, 'omitnan');

 %% 0.52

  % Load the CSV files
 dublin_default_netheat_052 = readmatrix('dublin_default_netheat_052.csv'); % Replace with
the actual filename
 dublin_default_distance_052 = readmatrix('dublin_default_distance_052.csv'); % Replace
with the actual filename

 % Determine the number of columns
 dublin_default_numColumns_integrate_052 = min(size(dublin_default_netheat_052, 2),
size(dublin_default_distance_052, 2));

 % Initialize the results matrix
 dublin_default_integration_results_052 = zeros(1,
dublin_default_numColumns_integrate_052);

 % Loop through each column
 for i = 1:dublin_default_numColumns_integrate_052
      % Extract current column for netheat and distance
     dublin_default_netheat_col_052 = dublin_default_netheat_052(:, i);
     dublin_default_distance_col_052 = dublin_default_distance_052(:, i);

     % Remove NaN values and ensure both arrays align
     dublin_default_valid_idx_052 = ~isnan(dublin_default_netheat_col_052) &
~isnan(dublin_default_distance_col_052);
     dublin_default_netheat_col_052 =
dublin_default_netheat_col_052(dublin_default_valid_idx_052);
     dublin_default_distance_col_052 =
dublin_default_distance_col_052(dublin_default_valid_idx_052);

     % Perform integration only if there are sufficient data points
     if length(dublin_default_distance_col_052) > 1
         % Sort distance and netheat by distance values for correct integration
         [dublin_default_distance_col_052, dublin_default_sortIdx_052] =
sort(dublin_default_distance_col_052);
         dublin_default_netheat_col_052 =
dublin_default_netheat_col_052(dublin_default_sortIdx_052);

         % Perform numerical integration using the trapezoidal rule
         dublin_default_integration_results_052(i) = trapz(dublin_default_distance_col_052,
dublin_default_netheat_col_052);
```

```
    else
        % Not enough data to integrate
        dublin_default_integration_results_052(i) = NaN;
    end
end

% Load the CSV file
dublin_default_density_052 = readmatrix('dublin_default_density_052.csv'); % Replace with
the actual filename

% Extract the first row of each column
dublin_default_first_row_values_density_052 = dublin_default_density_052(1, :);

% Load the CSV file
dublin_default_sensibleenthalpy_052 =
readmatrix('dublin_default_sensibleenthalpy_052.csv'); % Replace with the actual filename

% Get the number of columns
dublin_default_numColumns_sense_052 = size(dublin_default_sensibleenthalpy_052, 2);

% Initialize matrices to store first and last valid values
dublin_default_sensible_enthalpy_first_052 = zeros(1,
dublin_default_numColumns_sense_052);
dublin_default_sensible_enthalpy_last_052 = zeros(1, dublin_default_numColumns_sense_052);

% Loop through each column
for i = 1:dublin_default_numColumns_sense_052
    % Extract the column
    dublin_default_column_data_052 = dublin_default_sensibleenthalpy_052(:, i);

    % Remove NaN values to get valid data
    dublin_default_valid_data_052 =
dublin_default_column_data_052(~isnan(dublin_default_column_data_052));

    % Get the first and last values from the valid data
    if ~isempty(dublin_default_valid_data_052)
        dublin_default_sensible_enthalpy_first_052(i) = dublin_default_valid_data_052(1);
% First value
        dublin_default_sensible_enthalpy_last_052(i) =
max(dublin_default_valid_data_052(:,end)); % Last value
    else
        % If the column is entirely NaN, assign NaN
        dublin_default_sensible_enthalpy_first_052(i) = NaN;
        dublin_default_sensible_enthalpy_last_052(i) = NaN;
    end
end

dublin_default_xyz_052 = dublin_default_sensible_enthalpy_last_052-
dublin_default_sensible_enthalpy_first_052;

dublin_default_den_052 =
dublin_default_first_row_values_density_052.*dublin_default_xyz_052;

dublin_default_SC_052 =
dublin_default_integration_results_052./(2.*dublin_default_den_052);

% Load the strain rate data from the CSV file
dublin_default_axialvel_052 = readmatrix('dublin_default_axialvel_052.csv');

% Remove rows with NaN values
dublin_default_first_row_values_axialvel_052 = dublin_default_axialvel_052(1, :);
dublin_default_max_dist_052 = max(dublin_default_distance_052, [], 1, 'omitnan');
```

```
% Find the maximum value in each column
 % L = 0.014;
 dublin_default_strainrate_052 =
2*dublin_default_first_row_values_axialvel_052./dublin_default_max_dist_052;


  % Load the strain rate data from the CSV file
 dublin_default_maxtemp_052 = readmatrix('dublin_default_temperature_052.csv');

 % Remove rows with NaN values
 dublin_default_max_temp_052 = max(dublin_default_maxtemp_052, [], 1, 'omitnan');
 %% 0.45
% Load the CSV files
 dublin_default_netheat_045 = readmatrix('dublin_default_netheat_045.csv'); % Replace with
the actual filename
 dublin_default_distance_045 = readmatrix('dublin_default_distance_045.csv'); % Replace
with the actual filename

 % Determine the number of columns
 dublin_default_numColumns_integrate_045 = min(size(dublin_default_netheat_045, 2),
size(dublin_default_distance_045, 2));

 % Initialize the results matrix
 dublin_default_integration_results_045 = zeros(1,
dublin_default_numColumns_integrate_045);

 % Loop through each column
 for i = 1:dublin_default_numColumns_integrate_045
      % Extract current column for netheat and distance
     dublin_default_netheat_col_045 = dublin_default_netheat_045(:, i);
     dublin_default_distance_col_045 = dublin_default_distance_045(:, i);

     % Remove NaN values and ensure both arrays align
     dublin_default_valid_idx_045 = ~isnan(dublin_default_netheat_col_045) &
~isnan(dublin_default_distance_col_045);
     dublin_default_netheat_col_045 =
dublin_default_netheat_col_045(dublin_default_valid_idx_045);
     dublin_default_distance_col_045 =
dublin_default_distance_col_045(dublin_default_valid_idx_045);

     % Perform integration only if there are sufficient data points
     if length(dublin_default_distance_col_045) > 1
         % Sort distance and netheat by distance values for correct integration
         [dublin_default_distance_col_045, dublin_default_sortIdx_045] =
sort(dublin_default_distance_col_045);
         dublin_default_netheat_col_045 =
dublin_default_netheat_col_045(dublin_default_sortIdx_045);

         % Perform numerical integration using the trapezoidal rule
         dublin_default_integration_results_045(i) = trapz(dublin_default_distance_col_045,
dublin_default_netheat_col_045);
     else
         % Not enough data to integrate
         dublin_default_integration_results_045(i) = NaN;
     end
 end

 % Load the CSV file
 dublin_default_density_045 = readmatrix('dublin_default_density_045.csv'); % Replace with
the actual filename

 % Extract the first row of each column
 dublin_default_first_row_values_density_045 = dublin_default_density_045(1, :);
```

```
 % Load the CSV file
 dublin_default_sensibleenthalpy_045 =
readmatrix('dublin_default_sensibleenthalpy_045.csv'); % Replace with the actual filename

 % Get the number of columns
 dublin_default_numColumns_sense_045 = size(dublin_default_sensibleenthalpy_045, 2);

 % Initialize matrices to store first and last valid values
 dublin_default_sensible_enthalpy_first_045 = zeros(1,
dublin_default_numColumns_sense_045);
 dublin_default_sensible_enthalpy_last_045 = zeros(1, dublin_default_numColumns_sense_045);

 % Loop through each column
 for i = 1:dublin_default_numColumns_sense_045
     % Extract the column
     dublin_default_column_data_045 = dublin_default_sensibleenthalpy_045(:, i);

     % Remove NaN values to get valid data
     dublin_default_valid_data_045 =
dublin_default_column_data_045(~isnan(dublin_default_column_data_045));

     % Get the first and last values from the valid data
     if ~isempty(dublin_default_valid_data_045)
         dublin_default_sensible_enthalpy_first_045(i) = dublin_default_valid_data_045(1);
% First value
         dublin_default_sensible_enthalpy_last_045(i) =
max(dublin_default_valid_data_045(:,end)); % Last value
     else
         % If the column is entirely NaN, assign NaN
         dublin_default_sensible_enthalpy_first_045(i) = NaN;
         dublin_default_sensible_enthalpy_last_045(i) = NaN;
     end
 end

 dublin_default_xyz_045 = dublin_default_sensible_enthalpy_last_045-
dublin_default_sensible_enthalpy_first_045;

 dublin_default_den_045 =
dublin_default_first_row_values_density_045.*dublin_default_xyz_045;

 dublin_default_SC_045 =
dublin_default_integration_results_045./(2.*dublin_default_den_045);

 % Load the strain rate data from the CSV file
 dublin_default_axialvel_045 = readmatrix('dublin_default_axialvel_045.csv');

 % Remove rows with NaN values
 dublin_default_first_row_values_axialvel_045 = dublin_default_axialvel_045(1, :);
 dublin_default_max_dist_045 = max(dublin_default_distance_045, [], 1, 'omitnan');

 % Find the maximum value in each column
 % L = 0.014;
 dublin_default_strainrate_045 =
2*dublin_default_first_row_values_axialvel_045./dublin_default_max_dist_045;


  % Load the strain rate data from the CSV file
 dublin_default_maxtemp_045 = readmatrix('dublin_default_temperature_045.csv');

 % Remove rows with NaN values
 dublin_default_max_temp_045 = max(dublin_default_maxtemp_045, [], 1, 'omitnan');
```

```
%% 0.60

 % Load the CSV files
 dublin_default_netheat_060 = readmatrix('dublin_default_netheat_060.csv'); % Replace with
the actual filename
 dublin_default_distance_060 = readmatrix('dublin_default_distance_060.csv'); % Replace
with the actual filename

 % Determine the number of columns
 dublin_default_numColumns_integrate_060 = min(size(dublin_default_netheat_060, 2),
size(dublin_default_distance_060, 2));

 % Initialize the results matrix
 dublin_default_integration_results_060 = zeros(1,
dublin_default_numColumns_integrate_060);

 % Loop through each column
 for i = 1:dublin_default_numColumns_integrate_060
      % Extract current column for netheat and distance
     dublin_default_netheat_col_060 = dublin_default_netheat_060(:, i);
     dublin_default_distance_col_060 = dublin_default_distance_060(:, i);

     % Remove NaN values and ensure both arrays align
     dublin_default_valid_idx_060 = ~isnan(dublin_default_netheat_col_060) &
~isnan(dublin_default_distance_col_060);
     dublin_default_netheat_col_060 =
dublin_default_netheat_col_060(dublin_default_valid_idx_060);
     dublin_default_distance_col_060 =
dublin_default_distance_col_060(dublin_default_valid_idx_060);

     % Perform integration only if there are sufficient data points
     if length(dublin_default_distance_col_060) > 1
         % Sort distance and netheat by distance values for correct integration
         [dublin_default_distance_col_060, dublin_default_sortIdx_060] =
sort(dublin_default_distance_col_060);
         dublin_default_netheat_col_060 =
dublin_default_netheat_col_060(dublin_default_sortIdx_060);

         % Perform numerical integration using the trapezoidal rule
         dublin_default_integration_results_060(i) = trapz(dublin_default_distance_col_060,
dublin_default_netheat_col_060);
     else
         % Not enough data to integrate
         dublin_default_integration_results_060(i) = NaN;
     end
 end

 % Load the CSV file
 dublin_default_density_060 = readmatrix('dublin_default_density_060.csv'); % Replace with
the actual filename

 % Extract the first row of each column
 dublin_default_first_row_values_density_060 = dublin_default_density_060(1, :);

 % Load the CSV file
 dublin_default_sensibleenthalpy_060 =
readmatrix('dublin_default_sensibleenthalpy_060.csv'); % Replace with the actual filename

 % Get the number of columns
 dublin_default_numColumns_sense_060 = size(dublin_default_sensibleenthalpy_060, 2);

 % Initialize matrices to store first and last valid values
 dublin_default_sensible_enthalpy_first_060 = zeros(1,
dublin_default_numColumns_sense_060);
 dublin_default_sensible_enthalpy_last_060 = zeros(1, dublin_default_numColumns_sense_060);
```

```
 % Loop through each column
 for i = 1:dublin_default_numColumns_sense_060
     % Extract the column
     dublin_default_column_data_060 = dublin_default_sensibleenthalpy_060(:, i);

     % Remove NaN values to get valid data
     dublin_default_valid_data_060 =
dublin_default_column_data_060(~isnan(dublin_default_column_data_060));

     % Get the first and last values from the valid data
     if ~isempty(dublin_default_valid_data_060)
         dublin_default_sensible_enthalpy_first_060(i) = dublin_default_valid_data_060(1);
% First value
         dublin_default_sensible_enthalpy_last_060(i) =
max(dublin_default_valid_data_060(:,end)); % Last value
     else
         % If the column is entirely NaN, assign NaN
         dublin_default_sensible_enthalpy_first_060(i) = NaN;
         dublin_default_sensible_enthalpy_last_060(i) = NaN;
     end
 end

 dublin_default_xyz_060 = dublin_default_sensible_enthalpy_last_060-
dublin_default_sensible_enthalpy_first_060;

 dublin_default_den_060 =
dublin_default_first_row_values_density_060.*dublin_default_xyz_060;

 dublin_default_SC_060 =
dublin_default_integration_results_060./(2.*dublin_default_den_060);

 % Load the strain rate data from the CSV file
 dublin_default_axialvel_060 = readmatrix('dublin_default_axialvel_060.csv');

 % Remove rows with NaN values
 dublin_default_first_row_values_axialvel_060 = dublin_default_axialvel_060(1, :);
 dublin_default_max_dist_060 = max(dublin_default_distance_060, [], 1, 'omitnan');

 % Find the maximum value in each column
 % L = 0.014;
 dublin_default_strainrate_060 =
2*dublin_default_first_row_values_axialvel_060./dublin_default_max_dist_060;


  % Load the strain rate data from the CSV file
 dublin_default_maxtemp_060 = readmatrix('dublin_default_temperature_060.csv');

 % Remove rows with NaN values
dublin_default_max_temp_060 = max(dublin_default_maxtemp_060, [], 1, 'omitnan');

figure()
plot (dublin_default_strainrate_05(1:84),dublin_default_SC_05(1:84),'-', 'linewidth',3)
hold on
plot (dublin_default_strainrate_036(1:end),dublin_default_SC_036(1:end),'-', 'linewidth',3)
plot (dublin_default_strainrate_040(1:36),dublin_default_SC_040(1:36),'-', 'linewidth',3)
plot (dublin_default_strainrate_045(1:48),dublin_default_SC_045(1:48),'-', 'linewidth',3)
plot (dublin_default_strainrate_052(1:66),dublin_default_SC_052(1:66),'-', 'linewidth',3)
plot (dublin_default_strainrate_060(1:54),dublin_default_SC_060(1:54),'-', 'linewidth',3)
plot (dublin_default_strainrate_08(1:102),dublin_default_SC_08(1:102),'-', 'linewidth',3)
ylim([0.35 2])
legend('Phi = 0.50','Phi = 0.36','Phi = 0.40','Phi = 0.45','Phi = 0.52','Phi = 0.60','Phi =
0.80','Location','southeast')
title('Consumption Velocity V/s Strain Rate - Phi Effects')
xlabel('global strain Rate in /s')
ylabel('consumption velocity in m/s');
hold off
```

```
figure()
plot (dublin_default_strainrate_05(1:84),dublin_default_max_temp_05(1:84),'--o',
'linewidth',1)
hold on
plot (dublin_default_strainrate_036(1:end),dublin_default_max_temp_036(1:end),'--o',
'linewidth',1)
plot (dublin_default_strainrate_040(1:36),dublin_default_max_temp_040(1:36),'--o',
'linewidth',1)
plot (dublin_default_strainrate_045(1:48),dublin_default_max_temp_045(1:48),'--o',
'linewidth',1)
plot (dublin_default_strainrate_052(1:66),dublin_default_max_temp_052(1:66),'--o',
'linewidth',1)
plot (dublin_default_strainrate_060(1:54),dublin_default_max_temp_060(1:54),'--o',
'linewidth',1)
plot (dublin_default_strainrate_08(1:102),dublin_default_max_temp_08(1:102),'--o',
'linewidth',1)
legend('Phi = 0.50','Phi = 0.36','Phi = 0.40','Phi = 0.45','Phi = 0.52','Phi = 0.60','Phi =
0.80','Location','northeast')
title('Max Temperature V/s Strain Rate - Phi Effects')
xlabel('global strain rate in /s')
ylabel('max temperature in K');
hold off
```